
Amazon Neptune

User Guide

API Version 2020-12-18



Amazon Neptune: User Guide

Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Neptune?	1
Latest Updates	3
Getting started	22
What's a graph database?	22
Why use graphs?	23
Graph database applications	24
Online Neptune course	26
Query languages	26
Query examples	27
openCypher	29
Try out using graphs	29
Use notebooks	29
Digging deeper	30
Notebooks setup	30
Using Neptune workbench	30
Local hosting	33
Workbench magics	34
%seed	35
%load	35
%load_ids	35
%load_status	35
%cancel_load	35
%status	36
%gremlin-status	36
%opencypher-status, or %oc_status	36
%sparql-status	36
%graph_notebook_config	36
%graph_notebook_host	36
%graph_notebook_version	37
%graph_notebook_vis_options	37
%%graph_notebook_config	37
%%sparql	37
%%gremlin	38
%%opencypher, or %%oc	38
%%graph_notebook_vis_options	39
%neptune_ml	39
%%neptune_ml	42
Graph visualization	43
Gremlin visualization	43
SPARQL visualization	44
Visualization tutorials	45
Neptune Overview	46
Standards Compliance	47
Gremlin standards compliance	47
SPARQL Standards Compliance	56
Graph Data Model	61
Indexing Strategy	61
Gremlin data model	63
Lookup cache	64
Use cases for the lookup cache	64
Using the cache	65
Transaction Semantics	66
Isolation Levels	66
Neptune Isolation Levels	67

Transaction Examples	69
Exceptions and Retries	71
Clusters and Instances	73
The primary DB instance	73
Read-replica instances	73
Sizing instances	74
Monitoring instances	74
Storage, reliability and availability	75
Allocation	75
Storage billing	75
Storage best practices	76
Reliability and high availability	76
Endpoint Connections	78
Endpoint considerations	80
Custom endpoints	80
Create custom endpoints	81
View custom endpoints	81
Edit custom endpoints	82
Delete custom endpoints	82
Custom queryId	83
Using the HTTP Header	83
Using a SPARQL Query Hint	83
Using queryId to Check Status	83
Lab Mode	85
Using Lab Mode	85
The DFE engine	86
OSGP index	86
Transaction Semantics	86
Neptune DFE engine	87
Enabling/disabling the DFE	87
Queries executed by the DFE	88
DFE statistics	88
JDBC connectivity	95
Getting started	95
Using Tableau	96
Troubleshooting	98
Neptune Engine Updates	99
Upgrading	99
Testing before you upgrade	100
Engine Version Numbers	100
Manually Upgrading Using the Console	101
Automatic Upgrading Using the Console	101
Minor upgrade using the CLI	101
Major upgrade using the CLI	102
Automatic Updates Using the CLI	103
Security	104
Data Protection	104
VPC Access to DB Clusters	105
Encryption in Transit	108
Encryption at Rest	109
Identity and Access Management (IAM)	111
IAM Overview	114
Enabling IAM Authentication	117
IAM Policies	117
Tag-based access control	122
Neptune Service-Linked Roles	138
Temporary Credentials	141

Connecting and Signing	146
Logging and Monitoring	164
Compliance Validation	165
Resilience	165
Neptune setup	166
DNS in Your VPC	166
Creating a DB Cluster	166
Prerequisites	167
Create a Cluster	170
Accessing the Graph	171
Amazon EC2 Connection	171
Setting Up curl	172
Query Languages	172
IAM Authentication	173
Using Gremlin	173
Using RDF/SPARQL	176
Loading Data	177
Neptune Security	178
Monitoring Neptune	178
Troubleshooting and Best Practices	178
Migrating to Neptune	180
Migrating from TinkerPop	181
Migrating from RDF	182
Using AWS DMS to migrate	183
Migrating from Neo4j	184
Migrating from Blazegraph	185
Neptune compatibility	185
Provisioning infrastructure	185
Exporting data	186
Create an Amazon S3 bucket	188
Import the data	188
Loading data	190
Neptune Bulk Loader	190
IAM Role and Amazon S3 Access	191
Data Formats	194
Loading Example	204
Optimizing a bulk load	207
Loader Reference	208
Load data using DMS	225
GraphMappingConfig	226
Replicating to Neptune	229
Exporting data	233
neptune-export	234
Neptune-Export service	235
Install the service	235
Enable access to Neptune	237
Enable access to Neptune-Export	237
Run an export job	237
Monitor the job	238
Cancel a job	239
neptune-export utility	240
Prerequisites	240
Running neptune-export	241
Example commands	241
Exported files	243
Export parameters	244
command	245

outputS3Path	245
jobSize	245
params	246
additionalParams	246
params	247
Filtering examples	255
Troubleshooting	259
Querying	262
Gremlin	262
Installing the Gremlin console	263
HTTPS REST	266
Java	267
Python	274
.NET	276
Node.js	277
Query hints	278
Query status	284
Query cancellation	286
Gremlin sessions	286
Using the Gremlin API	287
Caching query results	288
Gremlin explain	292
Gremlin and DFE	340
SPARQL	342
RDF4J Console	342
RDF4J Workbench	344
Java	345
HTTP API	348
Query hints	357
Query status	365
Query cancellation	367
Graph store protocol	368
SPARQL explain	369
SPARQL SERVICE Extension	390
openCypher	392
Enabling openCypher	392
Overview of openCypher	392
Status endpoint	394
HTTPS endpoint	396
Using the Bolt protocol	399
Data model	402
Restrictions	403
Query queuing	404
Finding how many queries are in the queue	405
Query timeouts	405
Managing Neptune	406
Parameter groups	407
Editing a Parameter Group	407
Creating a Parameter Group	408
Parameters	409
neptune_enable_audit_log	409
neptune_lab_mode	409
neptune_query_timeout	409
neptune_streams	409
neptune_lookup_cache	410
neptune_autoscaling_config	410
neptune_ml_iam_role	410

neptune_ml_endpoint	410
neptune_query_timeout	410
neptune_result_cache	411
neptune_enforce_ssl	411
Launching a Cluster	412
Stopping and Starting a Cluster	416
Stopping and Starting Overview	416
Stopping a Cluster	416
Starting a DB Cluster	417
Fast reset API	419
Using IAM-Auth	421
The %db_reset magic	421
Common errors	422
Creating a Replica	423
Modifying a DB Cluster	424
Modify an Instance	424
Adding Neptune Replicas	426
Performance and Scaling	428
Storage Scaling	428
Instance Scaling;	428
Read Scaling	428
Auto-scaling	429
Enabling auto-scaling	430
Remove auto-scaling	432
Maintaining a DB Cluster	433
Maintenance Window	433
Neptune Updates	435
Manually Upgrading	435
Automatic Upgrading	435
Cloning a DB Cluster	437
Limitations	438
Copy-on-Write Protocol	438
Deleting a Source Database	440
Managing Instances	441
T3 Burstable Instances	442
Modifying an Instance	444
Renaming a Neptune DB Instance	447
Rebooting a DB Instance	448
Deleting a DB Instance	449
Neptune Streams	451
Using Streams	452
Enabling Streams	452
Disabling Streams	453
Calling the Streams API	453
Streams Response	454
Streams Exceptions	455
Streams Record Formats	456
PG_JSON	456
RDF-NQUADS	459
Streams Examples	459
AT_SEQUENCE_NUMBER Examples	459
AFTER_SEQUENCE_NUMBER Example	460
TRIM_HORIZON Example	461
LATEST Example	461
Compression Example	462
Neptune-to-Neptune Replication Setup	463
Choose an AWS CloudFormation template	463

Add stack details	464
Run the Template	466
Updating the stream poller	466
Streams for disaster recovery	467
Replication setup	467
Other considerations	470
Neptune full text search	471
Use of Lucene syntax	472
Full-text search setup	472
Set up Full-Text Search in Your Region	473
Add stack details	474
Run the Template	476
Updating the stream poller	476
Two approaches to enabling full text search	477
Custom OpenSearch fields	477
Neptune Full-Text Search Data Model	478
SPARQL Sample Document	478
Gremlin Sample Document	479
Full-Text Search Parameters	480
Non-string indexing	483
Updating an existing stack	484
Excluding fields	485
Datatype mappings	487
Datatype validation	488
Sample queries	492
Full-Text-Search Query Execution	494
Sample SPARQL full-text search queries	495
match Query	495
prefix	495
fuzzy	495
term	496
query_string	496
simple_query_string	496
sort by string field	497
sort by non-string field	497
sort by ID	497
sort by label	497
sort by doc_type	498
Lucene syntax	498
Sample Gremlin Full-Text Search Queries	498
Basic match	499
match	499
fuzzy	499
query_string fuzzy	499
query_string Regex	500
Hybrid Query	500
'+' and '-'	500
query_string, '+' and '-'	501
query_string, AND and OR	502
term	502
prefix	502
Lucene Syntax	502
Modern TinkerPop graph	503
Sort by string field	504
Sort by non-string field	504
Sort by ID field	504
Sort by label field	504

Sort by document_type field	504
Troubleshooting and metrics	505
Reads troubleshooting	505
Writes troubleshooting	505
ExpiredEventException errors	506
AWS Lambda functions	507
Gremlin WebSocket connections	507
Gremlin Lambda recommendations	508
Write-request recommendations	508
Read-request recommendations	509
Cold-start latency	509
Creating a Lambda Function	510
Lambda function examples	511
Java example	512
JavaScript example	515
Python example	518
Neptune machine learning	522
Neptune ML capabilities	522
Quick-start setup	523
Manual setup	526
IAM setup	527
DB cluster setup	532
Using Neptune ML	535
Starting workflow	535
Handling evolving data	537
Custom model workflow	538
Instance selection	539
For data processing	539
For model training and model transform	539
For an inference endpoint	539
Data export	540
Neptune-Export examples	540
params settings	541
additionalParams	541
targets	544
features	549
Examples	555
Data processing	564
Managing data processing	564
Updated processing	564
Feature encoding	566
Editing a training data file	570
Model training	577
Models and training	579
Customizing hyperparameters	581
Training best practices	589
Model transform	592
Incremental inference	592
Model transform for any job	592
Model artifacts	593
Artifacts for different tasks	593
Generating new artifacts	593
Custom models	595
Custom model overview	596
Custom model development	598
Inference endpoint	602
Managing inference endpoints	602

Inference queries	603
Gremlin inference queries	604
SPARQL inference queries	615
Neptune ML API	619
The dataprocessing command	620
The modeltraining command	624
The modeltransform command	629
The endpoints command	633
Exceptions	637
Limits	638
SageMaker limits	638
Monitoring Neptune	639
Instance Status	639
Sample output	641
Using CloudWatch	641
Using the Console	642
Using the AWS CLI	642
Using the CloudWatch API	642
Monitor instance performance	643
Neptune Metrics	644
Neptune Dimensions	650
Audit Logs with Neptune	651
Enable Audit Logs	651
Viewing Audit Logs	651
Audit Log Details	651
Neptune Logs in CloudWatch Logs	652
Publish Logs to CloudWatch Logs (Console)	652
Publish Logs to CloudWatch Logs (CLI)	653
Monitoring Log Events	653
Logging Neptune API Calls with AWS CloudTrail	654
Neptune Information in CloudTrail	654
Understanding Neptune Log File Entries	655
Event Notifications	656
Categories and messages	657
Subscribing to events	664
Manage subscriptions	665
Tagging Neptune Resources	665
Tagging Overview	666
Tagging in the Console	667
Tagging With the CLI	668
Tagging With the API	668
Working with ARNs	669
Backing Up and Restoring	673
Backup and Restore Overview	674
Fault Tolerance	674
Backups	674
Backup metrics	675
Restoring Data	676
Backup Window	676
Creating a Snapshot	678
Using the Console	678
Restoring from a Snapshot	679
Important restore considerations	679
Restoring	680
Copying a Snapshot	681
Limitations	681
Snapshot Copy Retention	681

Encryption	682
Cross-Region Snapshot Copying	682
Copying a Snapshot on the Console	682
Copying a Snapshot with the AWS CLI	683
Sharing a Snapshot	686
Encrypted Snapshots	686
Sharing	688
Deleting a Snapshot	690
Using the Console	690
Using the AWS CLI	690
Using the Neptune API	690
Best Practices	691
Basic Operational Guidelines	692
Security	693
If you have many predicates	693
Avoid long-running transactions	693
Using Metrics	693
Tuning Queries	694
Load Balancing	694
Use a Temporary Instance	694
Resizing an instance	695
Avoid different instance sizes	695
Task Interrupted Error	695
Gremlin (General)	696
GLV execution differences	697
Multithreaded Writes	697
Pruning Records	698
datetime()	698
Native Date and Time	698
Gremlin (Java client)	699
Use latest client version	699
Re-use the client object	700
Separate Clients for Reading and Writing	700
Multiple replica endpoints	700
Close the client when finished	700
New connection after failover	701
Set maxInProcessPerConnection = maxSimultaneousUsagePerConnection	701
Send queries as bytecode	701
Completely consume query results	702
Bulk add vertices and edges	702
Disable JVM DNS caching	703
Per-query timeouts	703
Use Graphbinary Serialization	703
Work around a bug in earlier versions	704
Troubleshoot exceptions	704
SPARQL	705
Query All Named Graphs	705
Specify a Named Graph to Load	705
FILTER vs. VALUES	705
Neptune Limits	707
Regions	707
China regions	707
Instance Sizes	707
Per Account	708
VPC Required	708
SSL Required	708
Availability Zones and Subnet Groups	708

HTTP Request Payload	708
Gremlin	709
SPARQL UPDATE LOAD	709
Authentication and Access	709
WebSockets Limits	709
Properties and Labels	710
Bulk-loading	710
Neptune Errors	711
Engine Error Codes	711
Error Format	711
Query Errors	711
IAM Errors	713
API Errors	715
Loader Errors	716
Engine Releases	718
Release: 1.1.0.0 (2021-11-19)	718
New Features	719
Improvements	719
Defects Fixed	720
Supported Query-Language Versions	721
Upgrade Paths	721
Upgrading	721
Release: 1.0.5.1 (2021-10-01)	722
Patch Releases	722
New Features	722
Improvements	722
Defects Fixed	723
Supported Query-Language Versions	723
Upgrade Paths	723
Upgrading	723
Release: 1.0.5.1.R2 (2021-10-26)	724
Release: 1.0.5.0 (2021-07-27)	725
Patch Releases	725
New Features	725
Improvements	725
Defects Fixed	726
Supported Query-Language Versions	726
Upgrade Paths	726
Upgrading	727
Release: 1.0.5.0.R3 (2021-09-15)	727
Release: 1.0.5.0.R2 (2021-08-16)	728
Release: 1.0.4.2 (2021-06-01)	729
Release: 1.0.4.2.R5 (2021-08-16)	730
Release: 1.0.4.2.R4 (2021-07-23)	730
Release: 1.0.4.2.R3 (2021-06-28)	731
Release: 1.0.4.2.R2 (2021-06-01)	732
Release: 1.0.4.2.R1 (2021-05-27)	734
Release: 1.0.4.1 (2020-12-08)	734
Patch Releases	734
New Features	734
Improvements	734
Defects Fixed	735
Supported Query-Language Versions	735
Upgrade Paths	735
Upgrading	735
Release: 1.0.4.1.R1.1 (2021-03-22)	736
Release: 1.0.4.1.R2 (2021-02-24)	737

Release: 1.0.4.0 (2020-10-12)	739
Patch Releases	739
New Features	739
Improvements	740
Defects Fixed	740
Supported Query-Language Versions	741
Upgrade Paths	741
Upgrading	741
Release: 1.0.4.0.R2 (2021-02-24)	741
Release: 1.0.3.0 (2020-08-03)	743
Patch Releases	743
New Features	743
Improvements	743
Defects Fixed	743
Supported Query-Language Versions	744
Upgrade Paths	744
Upgrading	744
Release: 1.0.3.0.R3 (2021-02-19)	745
Release: 1.0.3.0.R2 (2020-10-12)	746
Release: 1.0.2.2 (2020-03-09)	748
Patch Releases	748
Improvements	748
Defects Fixed	748
Supported Query-Language Versions	749
Upgrade Paths	749
Upgrading	749
Release: 1.0.2.2.R6 (2021-02-19)	749
Release: 1.0.2.2.R5 (2020-10-12)	751
Release: 1.0.2.2.R4 (2020-07-23)	752
Release: 1.0.2.2.R3 (2020-07-22)	753
Release: 1.0.2.2.R2 (2020-04-02)	754
Neptune engine release 1.0.2.1 (2019-11-22)	755
Patch Releases	755
New Features	755
Improvements	755
Defects Fixed	756
Supported Query-Language Versions	756
Upgrade Paths	756
Upgrading	756
Release: 1.0.2.1.R6 (2020-04-22)	757
Release: 1.0.2.1.R5 (2020-04-22)	758
Release: 1.0.2.1.R4 (2019-12-20)	758
Release: 1.0.2.1.R3 (2019-12-12)	759
Release: 1.0.2.1.R2 (2019-11-25)	760
Release: 1.0.2.0 (2019-11-08)	761
IMPORTANT: THIS VERSION IS NOW DEPRECATED	761
Patch Releases	762
New Features	762
Supported Query-Language Versions	762
Upgrade Paths	762
Upgrading	762
Release: 1.0.2.0.R3 (2020-05-05)	763
Release: 1.0.2.0.R2 (2019-11-21)	764
Release: 1.0.1.2 (2020-06-10)	765
IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED	765
Improvements	766
Defects Fixed	766

Supported Query-Language Versions	766
Release: 1.0.1.1 (2020-06-26)	766
IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED	766
Defects Fixed	766
Supported Query-Language Versions	766
Release: 1.0.1.0 (2019-07-02)	767
IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED	767
Release 1.0.1.0.200502.0 (2019-10-31)	767
Release 1.0.1.0.200463.0 (2019-10-15)	767
Release 1.0.1.0.200457.0 (2019-09-19)	768
Release 1.0.1.0.200369.0 (2019-08-13)	769
Release 1.0.1.0.200366.0 (2019-07-26)	770
Release 1.0.1.0.200348.0 (2019-07-02)	771
Earlier Releases	771
Upgrading	779
Testing before you upgrade	780
Engine Version Numbers	780
Manually Upgrading Using the Console	781
Automatic Upgrading Using the Console	781
Minor upgrade using the CLI	781
Major upgrade using the CLI	782
Automatic Updates Using the CLI	783
Neptune Management API	784
Clusters	788
DBCluster	788
DBClusterMember	792
DBClusterRole	792
ClusterPendingModifiedValues	792
CreateDBCluster	793
DeleteDBCluster	799
ModifyDBCluster	804
StartDBCluster	810
StopDBCluster	813
AddRoleToDBCluster	817
RemoveRoleFromDBCluster	818
FailoverDBCluster	818
PromoteReadReplicaDBCluster	822
DescribeDBClusters	826
Instances	827
DBInstance	828
DBInstanceStatusInfo	831
OrderableDBInstanceOption	831
ValidDBInstanceModificationsMessage	833
CreateDBInstance	833
DeleteDBInstance	841
ModifyDBInstance	846
RebootDBInstance	855
DescribeDBInstances	859
DescribeOrderableDBInstanceOptions	860
DescribeValidDBInstanceModifications	861
Parameters	862
Parameter	863
DBParameterGroup	863
DBClusterParameterGroup	864
DBParameterGroupStatus	864
CopyDBParameterGroup	865
CopyDBClusterParameterGroup	866

CreateDBParameterGroup	867
CreateDBClusterParameterGroup	869
DeleteDBParameterGroup	870
DeleteDBClusterParameterGroup	871
ModifyDBParameterGroup	871
ModifyDBClusterParameterGroup	872
ResetDBParameterGroup	873
ResetDBClusterParameterGroup	874
DescribeDBParameters	875
DescribeDBParameterGroups	876
DescribeDBClusterParameters	877
DescribeDBClusterParameterGroups	878
DescribeEngineDefaultParameters	879
DescribeEngineDefaultClusterParameters	880
Subnets	881
Subnet	881
DBSubnetGroup	882
CreateDBSubnetGroup	882
DeleteDBSubnetGroup	884
ModifyDBSubnetGroup	884
DescribeDBSubnetGroups	885
Snapshots	886
DBClusterSnapshot	887
DBClusterSnapshotAttribute	888
DBClusterSnapshotAttributesResult	889
CreateDBClusterSnapshot	889
DeleteDBClusterSnapshot	891
CopyDBClusterSnapshot	893
ModifyDBClusterSnapshotAttribute	896
RestoreDBClusterFromSnapshot	898
RestoreDBClusterToPointInTime	904
DescribeDBClusterSnapshots	910
DescribeDBClusterSnapshotAttributes	912
Events	912
Event	913
EventCategoriesMap	913
EventSubscription	914
CreateEventSubscription	915
DeleteEventSubscription	917
ModifyEventSubscription	918
DescribeEventSubscriptions	920
AddSourceIdentifierToSubscription	921
RemoveSourceIdentifierFromSubscription	922
DescribeEvents	923
DescribeEventCategories	925
Other	925
CloudwatchLogsExportConfiguration	926
DBEngineVersion	926
EngineDefaults	927
PendingMaintenanceAction	927
PendingModifiedValues	928
PendingCloudwatchLogsExports	929
ResourcePendingMaintenanceActions	929
UpgradeTarget	929
Tag	930
AddTagsToResource	930
ListTagsForResource	931

RemoveTagsFromResource	931
ApplyPendingMaintenanceAction	932
DescribePendingMaintenanceActions	933
DescribeDBEngineVersions	934
Datatypes	935
AvailabilityZone	935
DBSecurityGroupMembership	935
DomainMembership	936
DoubleRange	936
Endpoint	936
Filter	937
Range	937
Timezone	937
VpcSecurityGroupMembership	937
API Faults	938
AuthorizationAlreadyExistsFault	939
AuthorizationNotFoundFault	940
AuthorizationQuotaExceededFault	940
CertificateNotFoundFault	940
DBClusterAlreadyExistsFault	940
DBClusterNotFoundFault	940
DBClusterParameterGroupNotFoundFault	941
DBClusterQuotaExceededFault	941
DBClusterRoleAlreadyExistsFault	941
DBClusterRoleNotFoundFault	941
DBClusterRoleQuotaExceededFault	942
DBClusterSnapshotAlreadyExistsFault	942
DBClusterSnapshotNotFoundFault	942
DBInstanceAlreadyExistsFault	942
DBInstanceNotFoundFault	942
DBLogFileNotFoundFault	943
DBParameterGroupAlreadyExistsFault	943
DBParameterGroupNotFoundFault	943
DBParameterGroupQuotaExceededFault	943
DBSecurityGroupAlreadyExistsFault	944
DBSecurityGroupNotFoundFault	944
DBSecurityGroupNotSupportedFault	944
DBSecurityGroupQuotaExceededFault	944
DBSnapshotAlreadyExistsFault	944
DBSnapshotNotFoundFault	945
DBSubnetGroupAlreadyExistsFault	945
DBSubnetGroupDoesNotCoverEnoughAZs	945
DBSubnetGroupNotAllowedFault	945
DBSubnetGroupNotFoundFault	946
DBSubnetGroupQuotaExceededFault	946
DBSubnetQuotaExceededFault	946
DBUpgradeDependencyFailureFault	946
DomainNotFoundFault	946
EventSubscriptionQuotaExceededFault	947
InstanceQuotaExceededFault	947
InsufficientDBClusterCapacityFault	947
InsufficientDBInstanceCapacityFault	947
InsufficientStorageClusterCapacityFault	948
InvalidDBClusterSnapshotStateFault	948
InvalidDBClusterStateFault	948
InvalidDBInstanceStateFault	948
InvalidDBParameterGroupStateFault	948

InvalidDBSecurityGroupStateFault	949
InvalidDBSnapshotStateFault	949
InvalidDBSubnetGroupFault	949
InvalidDBSubnetGroupStateFault	949
InvalidDBSubnetStateFault	950
InvalidEventSubscriptionStateFault	950
InvalidOptionGroupStateFault	950
InvalidRestoreFault	950
InvalidSubnet	950
InvalidVPCNetworkStateFault	951
KMSKeyNotAccessibleFault	951
OptionGroupNotFoundFault	951
PointInTimeRestoreNotEnabledFault	951
ProvisionedIopsNotAvailableInAZFault	952
ResourceNotFoundFault	952
SNSInvalidTopicFault	952
SNSNoAuthorizationFault	952
SNSTopicArnNotFoundFault	952
SharedSnapshotQuotaExceededFault	953
SnapshotQuotaExceededFault	953
SourceNotFoundFault	953
StorageQuotaExceededFault	953
StorageTypeNotSupportedFault	954
SubnetAlreadyInUse	954
SubscriptionAlreadyExistFault	954
SubscriptionCategoryNotFoundFault	954
SubscriptionNotFoundFault	954

What Is Amazon Neptune?

Amazon Neptune is a fast, reliable, fully managed graph database service that makes it easy to build and run applications that work with highly connected datasets. The core of Neptune is a purpose-built, high-performance graph database engine. This engine is optimized for storing billions of relationships and querying the graph with milliseconds latency. Neptune supports the popular graph query languages Apache TinkerPop Gremlin and W3C's SPARQL, enabling you to build queries that efficiently navigate highly connected datasets. Neptune powers graph use cases such as recommendation engines, fraud detection, knowledge graphs, drug discovery, and network security.

Neptune is highly available, with read replicas, point-in-time recovery, continuous backup to Amazon S3, and replication across Availability Zones. Neptune provides data security features, with support for encryption at rest and in transit. Neptune is fully managed, so you no longer need to worry about database management tasks like hardware provisioning, software patching, setup, configuration, or backups.

To learn more about using Amazon Neptune, we recommend that you start with the following sections:

- [Getting started with graph databases \(p. 22\)](#)
- [Overview of Amazon Neptune Features \(p. 46\)](#)

If you're new to graphs, or are not yet ready to invest in a full Neptune production environment, visit our [Getting started \(p. 22\)](#) topic to find out how to use Neptune Jupyter notebooks for learning and developing without incurring costs.

Also, before you begin designing a database, we recommend that you consult the GitHub repository [AWS Reference Architectures for Using Graph Databases](#), where you can inform your choices about graph data models and query languages, and browse examples of reference deployment architectures.

Key Service Components

- *Primary DB instance* – Supports read and write operations, and performs all of the data modifications to the cluster volume. Each Neptune DB cluster has one primary DB instance that is responsible for writing (that is, loading or modifying) graph database contents.
- *Neptune replica* – Connects to the same storage volume as the primary DB instance and supports only read operations. Each Neptune DB cluster can have up to 15 Neptune Replicas in addition to the primary DB instance. This provides high availability by locating Neptune Replicas in separate Availability Zones and distribution load from reading clients.
- *Cluster volume* – Neptune data is stored in the cluster volume, which is designed for reliability and high availability. A cluster volume consists of copies of the data across multiple Availability Zones in a single AWS Region. Because your data is automatically replicated across Availability Zones, it is highly durable, and there is little possibility of data loss.

Supports Open Graph APIs

Amazon Neptune supports open graph APIs for both Gremlin and SPARQL. It provides high performance for both of these graph models and their query languages. You can choose the Property Graph (PG) model and its open source query language, or the Apache TinkerPop [Gremlin graph traversal language](#). Or, you can use the W3C standard Resource Description Framework (RDF) model and its standard [SPARQL Query Language](#).

Highly Secure

Neptune provides multiple levels of security for your database. Security features include network isolation using [Amazon VPC](#), and encryption at rest using keys that you create and control through [AWS Key Management Service \(AWS KMS\)](#). On an encrypted Neptune instance, data in the underlying storage is encrypted, as are the automated backups, snapshots, and replicas in the same cluster.

Fully Managed

With Amazon Neptune, you don't have to worry about database management tasks like hardware provisioning, software patching, setup, configuration, or backups.

You can use Neptune to create sophisticated, interactive graph applications that can query billions of relationships in milliseconds. SQL queries for highly connected data are complex and hard to tune for performance. With Neptune, you can use the popular graph query languages TinkerPop Gremlin and SPARQL to execute powerful queries that are easy to write and perform well on connected data. This capability significantly reduces code complexity so that you can quickly create applications that process relationships.

Neptune is designed to offer greater than 99.99 percent availability. It increases database performance and availability by tightly integrating the database engine with an SSD-backed virtualized storage layer that is built for database workloads. Neptune storage is fault-tolerant and self-healing. Disk failures are repaired in the background without loss of database availability. Neptune automatically detects database crashes and restarts without the need for crash recovery or rebuilding the database cache. If the entire instance fails, Neptune automatically fails over to one of up to 15 read replicas.

Changes and Updates to Amazon Neptune

The following table describes important changes to Amazon Neptune.

update-history-change	update-history-description	update-history-date
Engine version 1.1.0.0 (p. 718)	As of 2021-11-19, engine version 1.1.0.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.1.0.0 .	November 19, 2021
Engine version 1.0.5.1.R2 (p. 724)	As of 2021-10-26, engine version 1.0.5.1.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.5.1.R2 .	October 26, 2021
Engine version 1.0.5.1 (p. 722)	As of 2021-10-01, engine version 1.0.5.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.5.1 .	October 1, 2021
Blog post about why developers like TinkerPop (p. 3)	See Why developers like Apache TinkerPop, an open source framework for graph computing , by Brad Bebee, Kelvin Lawrence, and Stephen Mallette.	September 27, 2021
Engine version 1.0.5.0.R3 (p. 727)	As of 2021-09-15, engine version 1.0.5.0.R3 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.5.0.R3 .	September 15, 2021

Blog post about building a data discovery solution with Amundsen and Neptune (p. 3)	See Building a data discovery solution with Amundsen and Amazon Neptune , by Peter Hanssens and Don Simpson.	September 8, 2021
Neptune has updated the stream-poller to support non-string full-text search queries (p. 3)	Included in this release are many improvements to full-text search, including support for indexing of property values that are not strings. See Non-string OpenSearch indexing in Amazon Neptune .	August 23, 2021
Engine version 1.0.5.0.R2 (p. 728)	As of 2021-08-16, engine version 1.0.5.0.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.5.0.R2 .	August 16, 2021
Engine version 1.0.4.2.R5 (p. 730)	As of 2021-08-16, engine version 1.0.4.2.R5 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.2.R5 .	August 16, 2021
Blog post about Graph Store Protocol support in Neptune (p. 3)	See Introducing Graph Store Protocol support for Amazon Neptune , by Chris Smith.	August 2, 2021
Blog post about new features in Neptune ML (p. 3)	See Discover more insights in your graphs with new features from Amazon Neptune ML , by Soji Adeshina.	July 30, 2021
Blog post about getting predictions faster with Neptune ML (p. 3)	See Get predictions for evolving graph data faster with Amazon Neptune ML , by Soji Adeshina.	July 30, 2021
Blog post about easier and faster graph machine learning with Neptune ML (p. 3)	See Easier and faster graph machine learning with Amazon Neptune ML , by Soji Adeshina.	July 30, 2021
Blog post about Neptune openCypher support (p. 3)	See Announcing openCypher for Amazon Neptune: Building better graph applications with openCypher and Gremlin together , by Brad Bebee.	July 29, 2021

Engine version 1.0.5.0 (p. 725)	As of 2021-07-27, engine version 1.0.5.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.5.0 .	July 27, 2021
Engine version 1.0.4.2.R4 (p. 730)	As of 2021-07-23, engine version 1.0.4.2.R4 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.2.R4 .	July 23, 2021
Neptune launched in China (Beijing) (p. 3)	Amazon Neptune is now available in China (Beijing) (cn-north-1).	July 21, 2021
Engine version 1.0.4.2.R3 (p. 731)	As of 2021-06-28, engine version 1.0.4.2.R3 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.2.R3 .	June 28, 2021
Blog post about how Dream11 scaled their social network using Neptune (p. 3)	See Learn how Dream11, the World's largest fantasy sports platform, scale their social network with Amazon Neptune and Amazon ElastiCache .	June 25, 2021
Blog post about transforming data into knowledge with Neptune and PoolParty Semantic Suite (p. 3)	See Transform data into knowledge with PoolParty Semantic Suite and Amazon Neptune , by Ioanna Lytra and Albin Ahmeti.	June 16, 2021
Blog post about using Neptune to explore the UniProt knowledgebase (p. 3)	See Exploring the UniProt protein knowledgebase with AWS Open Data and Amazon Neptune , by Eric Greene, Rafa Xu, and Yuan Shi.	June 10, 2021
Blog post about using Neptune for data-driven risk analysis (p. 3)	See Field Notes: Data-Driven Risk Analysis with Amazon Neptune and Amazon OpenSearch Service , by Adriaan de Jonge and Rohit Satyanarayana.	June 10, 2021

Engine version 1.0.4.2.R2 (p. 732)	As of 2021-06-01, engine version 1.0.4.2.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.2.R2 .	June 1, 2021
Blog post about using Neptune to visualize your AWS infrastructure (p. 3)	See Visualize your AWS Infrastructure with Amazon Neptune and AWS Config , by Rohan Raizada and Amey Dhavle.	May 25, 2021
Blog post about configuration for using Data Lens with Neptune (p. 3)	See Configure AWS services to build a knowledge graph in Amazon Neptune using Data Lens , by Russell Waterson.	May 5, 2021
Blog post about building a knowledge graph in Neptune using Data Lens (p. 3)	See Build a knowledge graph in Amazon Neptune using Data Lens , by Russell Waterson.	May 5, 2021
Engine versions 1.0.1.0, 1.0.1.1, and 1.0.1.2 are now deprecated (p. 3)	Beginning now, no new DB instance will be created using any of these engine versions, or any patches related to them.	April 26, 2021
English translation of case study about Japan's Ministry of Economy, Trade and Industry using Neptune (p. 3)	See Japan's Ministry of Economy, Trade and Industry Powers gBizINFO Corporate Information Search Database with AWS .	March 31, 2021
Blog post about using Neptune with Amazon Comprehend and Lex (p. 3)	See Supercharge your knowledge graph using Amazon Neptune, Amazon Comprehend, and Amazon Lex , by Dave Bechberger.	March 31, 2021
Blog post about using Lambda functions with Neptune (p. 3)	See Use AWS Lambda functions with Amazon Neptune , by Ian Robinson.	March 26, 2021
Engine version 1.0.4.1.R1.1 (p. 736)	As of 2021-03-22, engine version 1.0.4.1.R1.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.1.R1.1 .	March 22, 2021

Engine version 1.0.4.1.R2.1 (p. 738)	As of 2021-03-11, engine version 1.0.4.1.R2.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.1.R2.1 .	March 11, 2021
Blog post about using Neptune's open source graph notebook for graph visualization (p. 3)	See Getting started with open source graph notebook for graph visualization , by Joy Wang, Ora Lassila, and Stephen Mallette.	March 10, 2021
Tutorial about integrating Neptune with the Amundsen data discovery and metadata engine (p. 3)	See How to use Amundsen with Amazon Neptune , by Andrew Ciabrone.	March 2, 2021
Engine version 1.0.4.1.R2 (p. 737)	As of 2021-02-24, engine version 1.0.4.1.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.1.R2 .	February 24, 2021
Engine version 1.0.4.0.R2 (p. 741)	As of 2021-02-24, engine version 1.0.4.0.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.0.R2 .	February 24, 2021
Engine version 1.0.3.0.R3 (p. 745)	As of 2021-02-19, engine version 1.0.3.0.R3 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.3.0.R3 .	February 19, 2021

Engine version 1.0.2.2.R6 (p. 749)	As of 2021-02-19, engine version 1.0.2.2.R6 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.2.2.R6 .	February 19, 2021
Blog post about building a knowledge graph using Amazon Comprehend events (p. 3)	See Building a knowledge graph in Amazon Neptune using Amazon Comprehend Events , by Brian O'Keefe, Graham Horwood, and Navtanay Sinha.	January 19, 2021
Blog post about enabling low code graph data apps (p. 3)	See Enabling low code graph data apps with Amazon Neptune and Graphistry , by Leo Meyerovich, Dave Bechberger, and Taylor Riggan.	January 18, 2021
Added notebook documentation for getting started with graph data. (p. 22)	Added a section integrating with the Neptune workbench that helps you get started creating graph data and developing graph applications without having to spin up a Neptune cluster until you're ready.	January 15, 2021
Blog post about resetting your Neptune graph data in seconds (p. 3)	See Resetting your graph data in Amazon Neptune in seconds , by Niraj Jetly and Navtanay Sinha.	December 17, 2020
Blog post about how Novartis AG uses SageMaker and Neptune with BERT (p. 3)	See Novartis AG uses Amazon SageMaker and Amazon Neptune to build and enrich a knowledge graph using BERT , by Othmane Hamzaoui, Fatema Alkhanaizi, and Viktor Malesevic.	December 14, 2020
Blog post about building a knowledge graph with topic networks (p. 3)	See Building a knowledge graph with topic networks in Amazon Neptune , by Edward Brown, Head of AI Projects, Eduardo Piairo, Architect, Marcia Oliveira, Lead Data Scientist, and Jack Hampson, CEO at Deeper Insights.	December 14, 2020

Engine version 1.0.4.1 (p. 734)	As of 2020-12-08, engine version 1.0.4.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.1 .	December 8, 2020
Blog post about getting started with Neptune ML (p. 3)	See How to get started with Neptune ML , by George Karypis, Dave Bechberger, and Karthik Bharathy.	December 8, 2020
Neptune now has a fast reset API (p. 3)	Using the fast-reset API, you can quickly and easily delete all the data in a DB cluster. See Fast reset API .	December 4, 2020
Blog post about building a biological knowledge graph at Pendulum (p. 3)	See Building a biological knowledge graph at Pendulum using Amazon Neptune , by Connor Skennerton.	November 26, 2020
Blog post about TinkerPop 3.4.8's new features in Neptune (p. 3)	See Exploring Apache TinkerPop 3.4.8's new features in Amazon Neptune , by Stephen Mallette.	November 18, 2020
Blog post about using the Amazon Kendra search service with Neptune (p. 3)	See Incorporating your enterprise knowledge graph into Amazon Kendra , by Yazdan Shirvany, Mohit Mehta, and Dipto Chakravarty.	November 17, 2020
Event notifications now available (p. 3)	Neptune now supports event notifications that you can use to more easily monitor DB clusters. See Using Neptune Event Notification , by.	October 29, 2020
Customs endpoints now available (p. 3)	Neptune now supports custom endpoints for greater control connecting to DB instances. See Connecting to Amazon Neptune Endpoints .	October 29, 2020
Blog post about using AWS Database Migration Service (DMS) to populate your Neptune graph (p. 3)	See Populating your graph in Amazon Neptune from a relational database using AWS Database Migration Service (DMS) – Part 4: Putting it all together , by Chris Smith.	October 22, 2020

<p>Blog post about using AWS Database Migration Service (DMS) to populate your Neptune graph (p. 3)</p>	<p>See Populating your graph in Amazon Neptune from a relational database using AWS Database Migration Service (DMS) – Part 3: Designing the RDF Model, by Chris Smith.</p>	<p>October 22, 2020</p>
<p>Blog post about using AWS Database Migration Service (DMS) to populate your Neptune graph (p. 3)</p>	<p>See Populating your graph in Amazon Neptune from a relational database using AWS Database Migration Service (DMS) – Part 2: Designing the property graph model, by Chris Smith.</p>	<p>October 22, 2020</p>
<p>Blog post about using AWS Database Migration Service (DMS) to populate your Neptune graph (p. 3)</p>	<p>See Populating your graph in Amazon Neptune from a relational database using AWS Database Migration Service (DMS) – Part 1: Setting the stage, by Chris Smith.</p>	<p>October 22, 2020</p>
<p>Engine version 1.0.4.0 (p. 739)</p>	<p>As of 2020-10-12, engine version 1.0.4.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.4.0.</p>	<p>October 12, 2020</p>
<p>Engine version 1.0.3.0.R2 (p. 746)</p>	<p>As of 2020-10-12, engine version 1.0.3.0.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.3.0.R2.</p>	<p>October 12, 2020</p>
<p>Engine version 1.0.2.2.R5 (p. 751)</p>	<p>As of 2020-10-12, engine version 1.0.2.2.R5 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.2.2.R5.</p>	<p>October 12, 2020</p>
<p>Blog post about configuring your VPC for SPARQL federated queries (p. 3)</p>	<p>See Configure Amazon VPC for SPARQL 1.1 Federated Query with Amazon Neptune, by Charles Ivie.</p>	<p>October 12, 2020</p>

Blog post about writing a SPARQL cascading delete (p. 3)	See Write a cascading delete in SPARQL , by Ora Lassila.	October 5, 2020
Blog post about graphing AWS resources using Neptune (p. 3)	See Graph your AWS resources with Amazon Neptune , by Dave Bechberger.	September 28, 2020
Blog post about building MedDRA terminology mapping for pharmacovigilance and adverse event reporting using Neptune (p. 3)	See Building Amazon Neptune based MedDRA terminology mapping for pharmacovigilance and adverse event reporting , by Vaijayanti Joshi, Deven Atnoor, Ph.D, and Sudhanshu Malhotra.	September 24, 2020
Blog post about building a knowledge graph out of a data warehouse using Neptune, to complement commercial intelligence (p. 3)	See Complement Commercial Intelligence by Building a Knowledge Graph out of a Data Warehouse with Amazon Neptune , by Shahria Hossain and Mikael Graindorge.	September 23, 2020
Blog post about load balancing using the Neptune Gremlin client (p. 3)	See Load balance graph queries using the Amazon Neptune Gremlin Client , by Ian Robinson.	September 16, 2020
Blog post about digital personalization using an identity graph at Cox Automotive (p. 3)	See Cox Automotive scales digital personalization using an identity graph powered by Amazon Neptune , by Carlos Rendon and Niraj Jetly.	September 16, 2020
Blog post about collaborative filtering on Yelp data (p. 3)	See Using collaborative filtering on Yelp data to build a recommendation system in Amazon Neptune , by Chad Tindel.	September 8, 2020
Blog post about query-result visualization in Amazon Neptune (p. 3)	See Visualize query results using the Amazon Neptune workbench , by Kelvin Lawrence.	September 2, 2020
Neptune released graph visualization (p. 3)	Amazon Neptune now provides extensive graph visualization capabilities in Jupyter notebooks in the Neptune workbench, along with a number of new features that make notebooks easier to use. See Graph visualization .	August 12, 2020
Neptune launched in South America (São Paulo) (p. 3)	Amazon Neptune is now available in South America (São Paulo) (sa-east-1).	August 6, 2020
Neptune launched in Asia Pacific (Hong Kong) (p. 3)	Amazon Neptune is now available in Asia Pacific (Hong Kong) (ap-east-1).	August 6, 2020

Engine version 1.0.3.0 (p. 743)	As of 2020-08-03, engine version 1.0.3.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.3.0 .	August 3, 2020
Engine version 1.0.2.2.R4 (p. 752)	As of 2020-07-23, engine version 1.0.2.2.R4 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.2.2.R4 .	July 23, 2020
Blog post about Zerobase's automated contact tracing using Amazon Neptune (p. 3)	See Zerobase creates private, secure, and automated contact tracing using Amazon Neptune , by David Harris and Aron Szanto.	July 13, 2020
Neptune launched in US West (N. California) (p. 3)	Amazon Neptune is now available in US West (N. California) (<code>us-west-1</code>).	July 9, 2020
Amazon Neptune supports tag-based access control (p. 3)	You can now use AWS tags in IAM policies to control access to your Neptune database. See Tag-based access control in Amazon Neptune .	July 7, 2020
A Java stream poller is now available (p. 3)	Amazon Neptune now supports a Java version of the lambda stream poller for Neptune streams as well as the Python one. See Add details about the Neptune streams consumer stack you are creating .	July 6, 2020
Blog post about the AWS COVID-19 knowledge graph (p. 3)	See Building and querying the AWS COVID-19 knowledge graph , by Ninad Kulkarni, Colby Wise, George Price, and Miguel Romero.	July 1, 2020
Engine version 1.0.1.1 (p. 766)	As of 2020-06-26, engine version 1.0.1.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.1.1 .	June 26, 2020

Blog post about migrating from Blazegraph to Amazon Neptune (p. 3)	See Moving to the cloud: Migrating Blazegraph to Amazon Neptune , by Dave Bechberger.	June 25, 2020
Blog post about changing data capture from Neo4j to Amazon Neptune (p. 3)	See Change data capture from Neo4j to Amazon Neptune using Amazon Managed Streaming for Apache Kafka , by Sanjeet Sahay.	June 22, 2020
Blog post about how Waves is using Amazon Neptune (p. 3)	See How Waves runs user queries and recommendations at scale with Amazon Neptune , by Pavel Vasilyev.	June 16, 2020
Engine version 1.0.1.2 (p. 765)	As of 2020-06-10, engine version 1.0.1.2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.1.2 .	June 10, 2020
Blog post about building a customer knowledge repository (p. 3)	See Building a customer 360 knowledge repository with Amazon Neptune and Amazon Redshift , by Ram Bhandarkar.	June 9, 2020
Blog post about how Gunosy is using Amazon Neptune (p. 3)	See How Gunosy built a comment feature in News Pass using Amazon Neptune by Yosuke Uchiyama.	June 8, 2020
Blog post about the AWS COVID-19 knowledge graph (p. 3)	See Building and querying the AWS COVID-19 knowledge graph by Ninad Kulkarni, Colby Wise, George Price, and Miguel Romero.	June 2, 2020
Blog post about exploring COVID-19 research using Amazon Neptune (p. 3)	See Exploring scientific research on COVID-19 with Amazon Neptune, Amazon Comprehend Medical, and the Tom Sawyer Graph Database Browser by George Price, Colby Wise, Miguel Romero, and Ninad Kulkarni.	June 2, 2020
You can now load data into Neptune using AWS DMS (p. 3)	See Using AWS Database Migration Service to load data into Amazon Neptune from a different data store .	June 1, 2020

Engine version 1.0.2.0 is being deprecated (p. 761)	Amazon Neptune engine version 1.0.2.0 is now deprecated. Clusters running on this engine version will be upgraded to version 1.0.2.1 automatically during the first maintenance window following June 1, 2020.	May 19, 2020
Blog post about building a customer identity graph using Neptune (p. 3)	See Building a customer identity graph with Amazon Neptune by Rajesh Wunnava and Taylor Riggan.	May 12, 2020
Engine version 1.0.2.0.R3 (p. 763)	As of 2020-05-05, engine version 1.0.2.0.R3 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.2.0.R3 .	May 5, 2020
Engine version 1.0.2.1.R6 (p. 757)	As of 2020-04-22, engine version 1.0.2.1.R6 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.2.1.R6 .	April 22, 2020
Blog post about migrating data from Neo4j to Neptune (p. 3)	See Migrating a Neo4j graph database to Amazon Neptune with a fully automated utility by Sanjeet Sahay.	April 13, 2020
Blog post about lowering the cost of building graph applications with Neptune (p. 3)	See Lower the cost of building graph apps by up to 76% with Amazon Neptune T3 instances by Karthik Bharathy and Brad Bebee.	April 9, 2020
Neptune offers a T3 burstable instance class (p. 3)	You can now create an Amazon Neptune T3 burstable instance for cost-effective development and testing purposes. See Neptune T3 Burstable Instance Class .	April 8, 2020

Engine version 1.0.2.2.R2 (p. 754)	<p>As of 2020-04-02, engine version 1.0.2.2.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.2.2.R2.</p>	April 2, 2020
Blog post about graphing investment dependency at EDGAR (p. 3)	<p>See Graphing investment dependency with Amazon Neptune by Lawrence Verdi.</p>	March 17, 2020
Neptune launched in Europe (Paris) (p. 3)	<p>Amazon Neptune is now available in Europe (Paris) (eu-west-3).</p>	March 11, 2020
Engine version 1.0.2.2 (p. 748)	<p>As of 2020-03-09, engine version 1.0.2.2 is being generally deployed. Please note that it takes several days for a new release to become available in every region. For more information about this engine version, see Neptune Engine Release 1.0.2.2.</p>	March 9, 2020
Stopping and Restarting a DB Cluster (p. 3)	<p>You can now stop a DB Cluster for 7 days using the Neptune console, and later restart it when you need it again. While your DB cluster is stopped, you are charged only for cluster storage, manual snapshots, and automated backup storage, but not for any DB instance hours. See Stopping and Starting an Amazon Neptune DB Cluster.</p>	February 19, 2020
Video about a social graph at Nike (p. 3)	<p>Listen in as Todd Escalona of AWS talks with Marc Wangenheim, Senior Engineering Manager at Nike, about how the company powers a number of applications via a social graph built on Amazon Neptune. See Nike: A Social Graph at Scale with Amazon Neptune.</p>	February 11, 2020

Neptune clusters can now be configured to require SSL connections (p. 3)	<p>In regions that still support HTTP connections, SSL is now turned on by default in all new parameter groups. There are no changes to existing parameter groups, but you can force clients to use SSL by changing the <code>neptune_enforce_ssl</code> parameter to 1. See Encryption in Transit: Connecting to Neptune Using SSL/HTTPS for information about how to enable HTTP connections for a cluster in a region that still supports them. See Parameters That You Can Use to Configure Amazon Neptune for a description of cluster and instance parameters.</p>	February 10, 2020
You can now specify engine version and deletion protection in Neptune's CloudFormation template (p. 3)	<p>Amazon Neptune has updated its CloudFormation template to include an <code>AWS::Neptune::DBCluster.EngineVersion</code> parameter that lets you specify a particular engine version for your new DB cluster, and an <code>AWS::Neptune::DBCluster.DeletionProtection</code> parameter that lets you turn on deletion protection for it.</p>	February 9, 2020
Deletion protection (p. 3)	<p>Amazon Neptune has delivered deletion protection for DB clusters and instances. As long as deletion protection is enabled on a DB cluster or instance, you cannot delete it. See You cannot delete a DB Instance if Deletion Protection is enabled.</p>	January 20, 2020
Neptune launched in China (Ningxia) (p. 3)	<p>Amazon Neptune is now available in China (Ningxia) (cn-northwest-1).</p>	January 15, 2020
Engine version 1.0.2.1.R4 (p. 758)	<p>Patch R4 for engine version 1.0.2.1 is generally available. For more information, see Neptune Engine Release 1.0.2.1.R4.</p>	December 20, 2019
Engine version 1.0.2.1.R3 (p. 759)	<p>Patch R3 for engine version 1.0.2.1 is generally available. For more information, see Neptune Engine Release 1.0.2.1.R3.</p>	December 12, 2019
Blog post about using Neptune to analyze social media feeds (p. 3)	<p>See Analyzing social media feeds using Amazon Neptune.</p>	November 27, 2019

Engine version 1.0.2.1.R2 (p. 760)	Patch R2 for engine version 1.0.2.1 is generally available. For more information, see Neptune Engine Release 1.0.2.1.R2 .	November 25, 2019
Engine version 1.0.2.1.R1 (p. 755)	Amazon Neptune engine version 1.0.2.1.R1 is generally available. For more information, see Neptune Engine Release 1.0.2.1 .	November 22, 2019
Engine version 1.0.2.0.R2 (p. 764)	Patch R2 for engine version 1.0.2.0 is generally available. For more information, see Neptune Engine Release 1.0.2.0.R2 .	November 21, 2019
Blog post about Neptune sessions and workshops at re:Invent 2019 (p. 3)	See Your guide to Amazon Neptune sessions, workshops, and chalk talks at AWS re:Invent 2019 .	November 20, 2019
Engine version 1.0.2.0.R1 (p. 761)	Amazon Neptune engine version 1.0.2.0.R1 is generally available. For more information, see Neptune Engine Release 1.0.2.0 .	November 8, 2019
Blog post about capturing graph changes using Neptune Streams (p. 3)	See Capture Graph Changes using Neptune Streams .	November 6, 2019
Engine version 1.0.1.0.200502.0 (p. 767)	Amazon Neptune engine version 1.0.1.0.200502.0 is generally available. For more information, see Update 1.0.1.0.200502.0 .	October 31, 2019
Neptune launched in Middle East (Bahrain) (p. 3)	Amazon Neptune is now available in Middle East (Bahrain) (me-south-1).	October 30, 2019
Neptune launched in Canada (Central) (p. 3)	Amazon Neptune is now available in Canada (Central) (ca-central-1).	October 30, 2019
Blog post about Neptune's new SPARQL Streams feature and SPARQL federated query support (p. 3)	See Amazon Neptune releases Streams, SPARQL federated query for graphs and more .	October 17, 2019
Engine version 1.0.1.0.200463.0 (p. 767)	Amazon Neptune engine version 1.0.1.0.200463.0 is generally available. For more information, see Update 1.0.1.0.200463.0 .	October 15, 2019
Engine version 1.0.1.0.200457.0 (p. 768)	Amazon Neptune engine version 1.0.1.0.200457.0 is generally available. For more information, see Update 1.0.1.0.200457.0 .	September 19, 2019

Blog post about Neptune's new SPARQL explain feature (p. 3)	See Using SPARQL explain to understand query execution in Amazon Neptune .	September 17, 2019
Blog post about Neptune support for TinkerPop 3.4 (p. 3)	See Amazon Neptune now supports TinkerPop 3.4 features .	September 6, 2019
Blog post about using Neptune with PyTorch on Amazon SageMaker (p. 3)	See A personalized 'shop-by-style' experience using PyTorch on Amazon SageMaker and Amazon Neptune .	August 22, 2019
Blog post about using Neptune with AWS AppSync and Amazon ElastiCache (p. 3)	See Integrating alternative data sources with AWS AppSync: Amazon Neptune and Amazon ElastiCache .	August 22, 2019
Neptune launched in AWS GovCloud (US-East) (p. 3)	Amazon Neptune is now available in AWS GovCloud (US-East) (us-gov-east-1).	August 21, 2019
Neptune launched in AWS GovCloud (US-West) (p. 3)	Amazon Neptune is now available in AWS GovCloud (US-West) (us-gov-west-1).	August 14, 2019
Engine version 1.0.1.0.200369.0 (p. 769)	Amazon Neptune engine version 1.0.1.0.200369.0 is generally available. For more information, see Update 1.0.1.0.200369.0 .	August 13, 2019
Engine version 1.0.1.0.200366.0 (p. 770)	Amazon Neptune engine version 1.0.1.0.200366.0 is generally available. For more information, see Update 1.0.1.0.200366.0 .	July 26, 2019
Blog post about using Neptune with PyTorch on Amazon SageMaker (p. 3)	See A personalized 'shop-by-style' experience using PyTorch on Amazon SageMaker and Amazon Neptune .	July 3, 2019
Engine version 1.0.1.0.200348.0 (p. 771)	Amazon Neptune engine version 1.0.1.0.200348.0 is generally available. For more information, see Update 1.0.1.0.200348.0 .	July 2, 2019
Neptune launched in Europe (Stockholm) (p. 3)	Amazon Neptune is now available in Europe (Stockholm) (eu-north-1).	June 27, 2019
Neptune can now publish audit logs to CloudWatch Logs (p. 3)	For more information, see Publishing Neptune Logs to Amazon CloudWatch Logs .	June 18, 2019
Engine version 1.0.1.0.200310.0 (p. 771)	Amazon Neptune engine version 1.0.1.0.200310.0 is generally available. For more information, see Update 1.0.1.0.200310.0 .	June 12, 2019

Blog post about LifeOmic's JupiterOne (p. 3)	See How LifeOmic's JupiterOne simplifies security and compliance operations with Amazon Neptune .	May 2, 2019
Neptune launched in Asia Pacific (Seoul) (p. 3)	Amazon Neptune is now available in Asia Pacific (Seoul) (ap-northeast-2).	May 1, 2019
Engine version 1.0.1.0.200296.0 (p. 772)	Amazon Neptune engine version 1.0.1.0.200296.0 is generally available. For more information, see Update 1.0.1.0.200296.0 .	May 1, 2019
Neptune launched in Asia Pacific (Mumbai) (p. 3)	Amazon Neptune is now available in Asia Pacific (Mumbai) (ap-south-1).	March 6, 2019
Blog post about Gremlin query hints (p. 3)	See Introducing Gremlin query hints for Amazon Neptune .	February 26, 2019
Neptune launched in Asia Pacific (Tokyo) (p. 3)	Amazon Neptune is now available in Asia Pacific (Tokyo) (ap-northeast-1).	January 23, 2019
AWS CloudFormation template for creating an AWS Lambda function to access Neptune (p. 166)	Updated the getting started section and added an AWS CloudFormation template to create a Lambda function to use with Neptune. For more information, see Getting Started with Neptune .	January 23, 2019
Engine version 1.0.1.0.200267.0 (p. 773)	Amazon Neptune engine version 1.0.1.0.200267.0 is generally available. For more information, see Update 1.0.1.0.200267.0 .	January 21, 2019
Neptune launched in Asia Pacific (Sydney) (p. 3)	Amazon Neptune is now available in Asia Pacific (Sydney) (ap-southeast-2).	January 9, 2019
Blog post about using Metaphactory (p. 3)	See Exploring Knowledge Graphs on Amazon Neptune Using Metaphactory .	January 9, 2019
Neptune launched in Asia Pacific (Singapore) (p. 3)	Amazon Neptune is now available in Asia Pacific (Singapore) (ap-southeast-1).	December 13, 2018
Engine version 1.0.1.0.200264.0 (p. 774)	Amazon Neptune engine version 1.0.1.0.200264.0 is generally available. For more information, see Update 1.0.1.0.200264.0 .	November 19, 2018
Amazon Neptune SSL Support (p. 108)	Neptune now supports SSL connections.	November 19, 2018

Consolidated Error topics (p. 711)	All error message and code information are now in a single topic.	November 15, 2018
Updated Getting Started Topic (p. 166)	Updated Getting Started topic with additional links and reorganized documentation.	November 14, 2018
Engine version 1.0.1.0.200258.0 (p. 775)	Amazon Neptune engine version 1.0.1.0.200258.0 is generally available. For more information, see Update 1.0.1.0.200258.0 .	November 8, 2018
Neptune launched in Europe (Frankfurt) (p. 3)	Amazon Neptune is now available in Europe (Frankfurt) (eu-central-1).	November 7, 2018
Blog post #1 in a series (p. 3)	See Let Me Graph That For You – Part 1 – Air Routes .	November 7, 2018
Blog post about using Amazon SageMaker Jupyter Notebooks (p. 3)	See Analyze Amazon Neptune Graphs using Amazon SageMaker Jupyter Notebooks .	November 1, 2018
Engine version 1.0.1.0.200255.0 (p. 775)	Amazon Neptune engine version 1.0.1.0.200255.0 is generally available. For more information, see Update 1.0.1.0.200255.0 .	October 29, 2018
Neptune launched in Europe (London) (p. 3)	Amazon Neptune is now available in Europe (London) (eu-west-2).	October 3, 2018
Engine version 1.0.1.0.200237.0 (p. 776)	Amazon Neptune engine version 1.0.1.0.200237.0 is generally available. For more information, see Update 1.0.1.0.200237.0 .	September 6, 2018
Engine version 1.0.1.0.200236.0 (p. 777)	Amazon Neptune engine version 1.0.1.0.200236.0 is generally available. For more information, see Update 1.0.1.0.200236.0 .	July 24, 2018
Engine version 1.0.1.0.200233.0 (p. 778)	Amazon Neptune engine version 1.0.1.0.200233.0 is generally available. For more information, see Update 1.0.1.0.200233.0 .	June 22, 2018
New Neptune Quick Start (p. 166)	Updated quick start with AWS CloudFormation and the Gremlin Console tutorial. For more information, see Amazon Neptune Quick Start Using AWS CloudFormation .	June 19, 2018
Amazon Neptune initial release (p. 3)	This is the initial release of the Neptune User Guide. See also the release blog post, Amazon Neptune Generally Available .	May 30, 2018

[Introductory Neptune Blog Post \(p. 3\)](#)

See [Amazon Neptune – A Fully Managed Graph Database Service](#).

November 29, 2017

Getting started with graph databases

Amazon Neptune is a fully managed graph database service that scales to handle billions of relationships and lets you query them with milliseconds latency, at a low cost for that kind of capacity.

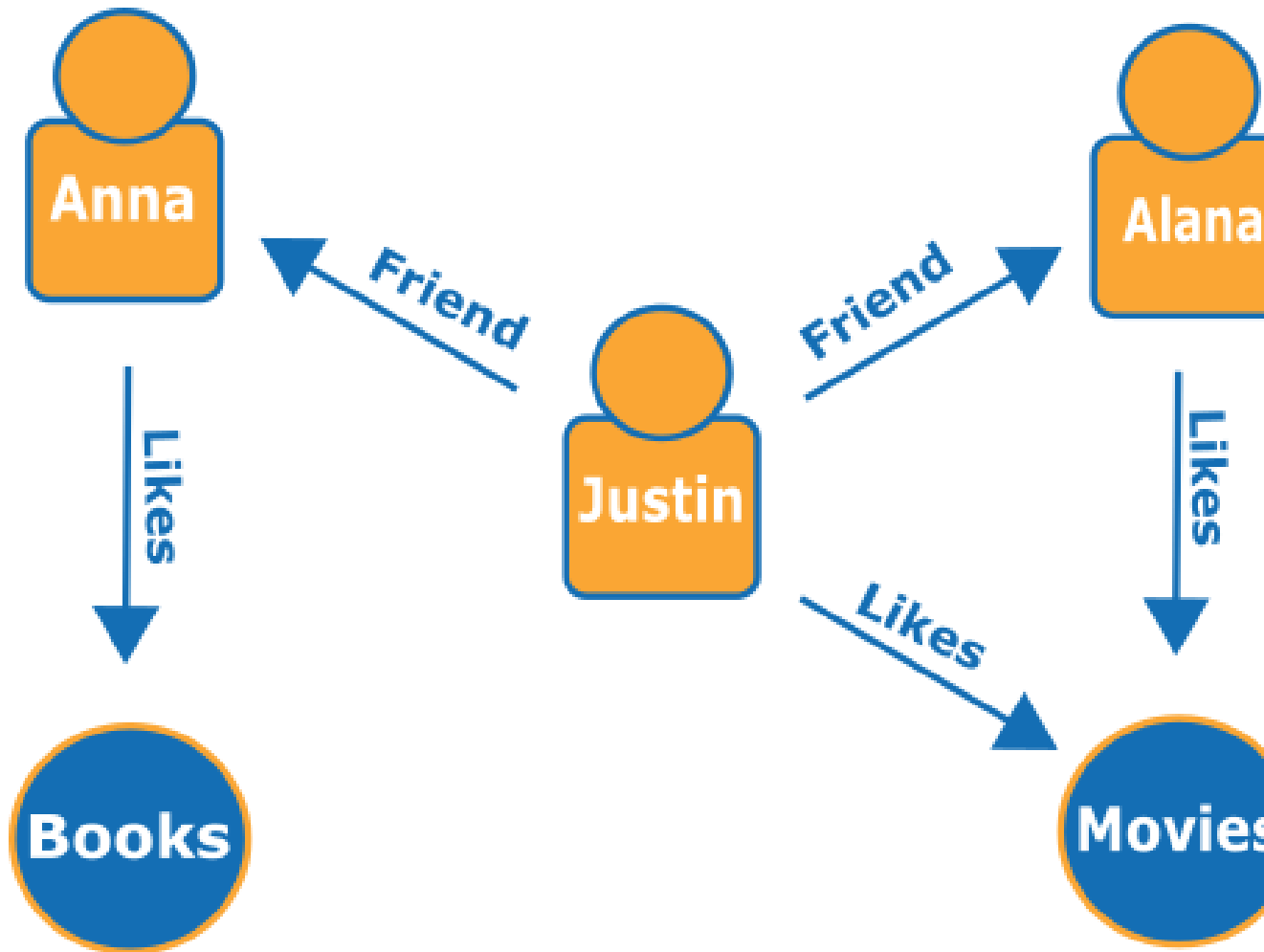
If you know about graphs already, jump ahead to [Try out using graphs \(p. 29\)](#). Or, if you want to go ahead and create a Neptune database, see [Using an AWS CloudFormation Stack to Create a Neptune DB Cluster \(p. 170\)](#).

What exactly *is* a graph database?

Graph databases are optimized to store and query the *relationships* between data items.

They store data items themselves as *vertices* of the graph, and the relationships between them as *edges*. Each edge has a type, and is directed from one vertex (the start) to another (the end). Relationships can be called *predicates* as well as edges, and vertices are also sometimes referred to as *nodes*. In so-called property graphs, both vertices and edges can have additional *properties* associated with them too.

Here is a small graph representing friends and hobbies in a social network:



The edges are shown as named arrows, and the vertices represent specific people and hobbies that they connect.

A simple traversal of this graph can tell you what Justin's friends like.

Why use a graph database?

Whenever connections or relationships between entities are at the core of the data that you're trying to model, a graph database is your natural choice.

For one thing, it's easy to model data interconnections as a graph, and then write complex queries that extract real-world information from the graph.

Building an equivalent application using a relational database requires you to create many tables with multiple foreign keys and then write nested SQL queries and complex joins. Not only does that approach quickly become unwieldy from a coding perspective, its performance degrades quickly as the amount of data increases.

By contrast, a graph database like Neptune can query relationships between billions of vertices without bogging down.

What can you do with a graph database?

Graphs can represent the interrelationships of real-world entities in many ways, in terms of actions, ownership, parentage, purchase choices, personal connections, family ties, and so on.

Here are some of the most common areas where graph databases are used:

- **Knowledge graphs** – Knowledge graphs let you organize and query all kinds of connected information to answer general questions. Using a knowledge graph, you can add topical information to product catalogs, and model diverse information such as is contained in [Wikidata](#).

To learn more about how knowledge graphs work and where they are being used, see [Knowledge Graphs on AWS](#).

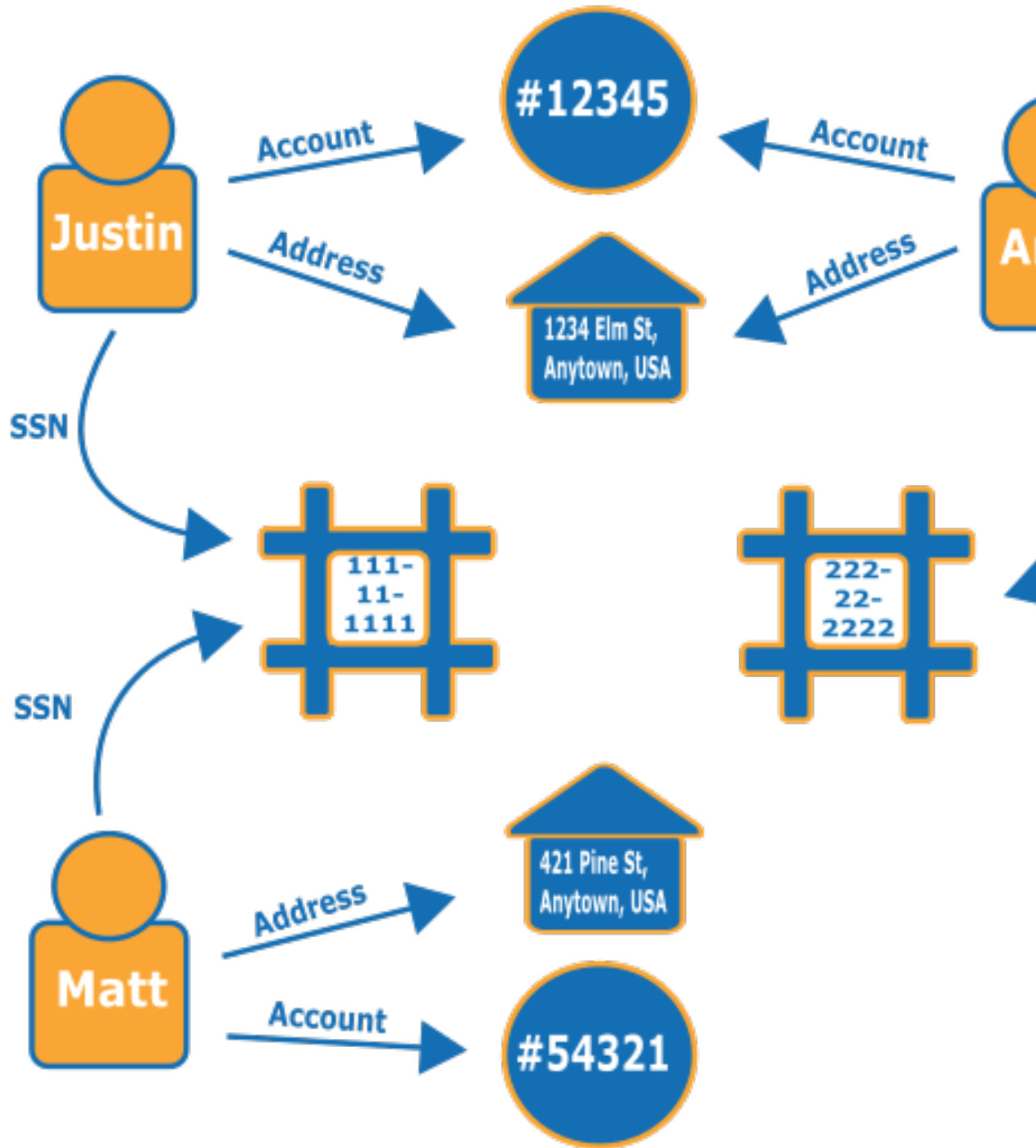
- **Identity graphs** – In a graph database, you can store relationships between information categories such as customer interests, friends, and purchase history, and then query that data to make recommendations which are personalized and relevant.

For example, you can use a graph database to make product recommendations to a user based on which products are purchased by others who follow the same sport and have a similar purchase history. Or, you can identify people who have a friend in common but don't yet know each other, and make a friendship recommendation.

Graphs of this kind are known as identity graphs, and are widely used for personalizing interactions with users. To find out more, see [Identity Graphs on AWS](#). To get started building your own identity graph, you can begin with the [Identity Graph Using Amazon Neptune](#) sample.

- **Fraud graphs** – This is a common use for graph databases. They can help you track credit card purchases and purchase locations to detect uncharacteristic use, or to detect a purchaser is trying to use the same email address and credit card as was used in a known fraud case. They can let you check for multiple people associated with a personal email address, or multiple people in different physical locations who share the same IP address.

Consider the following graph. It shows the relationship of three people and their identity-related information. Each person has an address, a bank account, and a social security number. However, we can see that Matt and Justin share the same social security number, which is irregular and indicates possible fraud by one of them. A query to a fraud graph can reveal connections of this kind so that they can be reviewed.



To learn out more about fraud graphs and where they are being used, see [Fraud Graphs on AWS](#).

- **Social networking** – One of the first and most common areas where graph databases are used is in social networking applications.

For example, suppose that you want to build a social feed into a web site. You can easily use a graph database on the back end to deliver results to users that reflect the latest updates from their families, their friends, from people whose updates they "like," and from people who live close to them.

- **Driving directions** – A graph can help find the best route from a starting point to a destination, given current traffic and typical traffic patterns.
- **Logistics** – Graphs can help identify the most efficient way to use available shipping and distribution resources to meet customer requirements.
- **Diagnostics** – Graphs can represent complex diagnostic trees that can be queried to identify the source of observed problems and failures.
- **Scientific research** – With a graph database, you build can applications that store and navigate scientific data and even sensitive medical information using encryption at rest. For example, you can store models of disease and gene interactions. You can search for graph patterns within protein pathways to find other genes that might be associated with a disease. You can model chemical compounds as a graph and query for patterns in molecular structures. You can correlate patient data from medical records in different systems. You can topically organize research publications to find relevant information quickly.
- **Regulatory rules** – You can store complex regulatory requirements as graphs, and query them to detect situations where they might apply to your day-to-day business operations.
- **Network topology and events** – A graph database can help you manage and protect an IT network. When you store the network topology as a graph, you can also store and process many different kinds of events on the network. You can answer questions such as how many hosts are running a given application. You can query for patterns that might show that a given host has been compromised by a malicious program, and query for connection data that can help trace the program to the original host that downloaded it.

Take an online course in using Amazon Neptune

If you like learning with videos, AWS offers an online course in the [AWS Online Tech Talks](#) to help you get going:

[Getting Started with Amazon Neptune](#)

The course consists of 7 videos that walk you through setting up and using Amazon Neptune.

Graph query languages

The main graph query languages are:

- **Gremlin**, defined by [Apache TinkerPop](#) for creating and querying property graphs.

A query in Gremlin is a traversal made up of discrete steps, each of which follows an edge to a node.

See [Accessing a Neptune graph with Gremlin \(p. 262\)](#) to learn about using Gremlin in Neptune, and [Gremlin standards compliance in Amazon Neptune \(p. 47\)](#) to find specific details about the Neptune implementation of Gremlin.

- **SPARQL** is a language supported by the [World Wide Web Consortium](#) for querying [RDF](#) data.

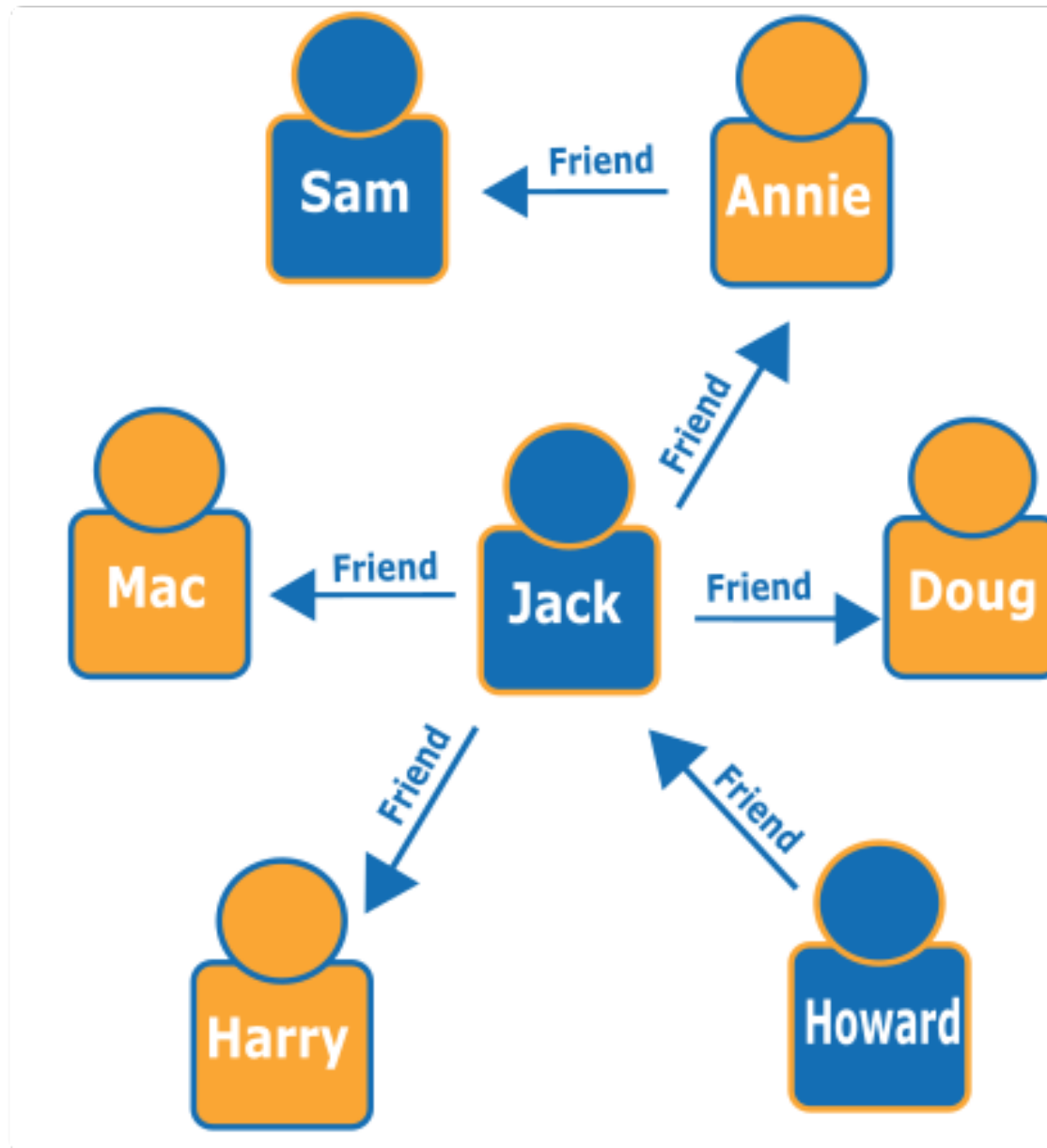
SPARQL is a declarative query language based on graph pattern-matching.

See [Accessing the Neptune graph with SPARQL \(p. 342\)](#) to learn about using SPARQL in Neptune, and [SPARQL Standards Compliance in Amazon Neptune \(p. 56\)](#) to find specific details about the Neptune implementation of SPARQL.

- [openCypher \(p. 392\)](#) – openCypher is a declarative query language for property graphs that was originally developed by Neo4j, then open-sourced in 2015, and contributed to the [openCypher](#) project under an Apache 2 open-source license. See the [Cypher Query Language Reference \(Version 9\)](#) for the language specification, as well as the [Cypher Style Guide](#) for additional information.

Examples of a Gremlin and a SPARQL query

Given the following graph of people (nodes) and their relationships (edges), you can find out who the "friends of friends" of a particular person are— for example, the friends of Howard's friends.



Looking at the graph, you can see that Howard has one friend, Jack, and Jack has four friends: Annie, Harry, Doug, and Mac. This is a simple example with a simple graph, but these types of queries can scale in complexity, dataset size, and result size.

Here is a Gremlin traversal query that returns the names of the friends of Howard's friends:

```
g.V().has('name', 'Howard').out('friend').out('friend').values('name')
```

Here is a SPARQL query that returns the names of the friends of Howard's friends:

```
prefix : <#>

select ?names where {
  ?howard :name "Howard" .
  ?howard :friend/:friend/:name ?names .
}
```

Note

Each part of any Resource Description Framework (RDF) triple has a URI associated with it. In the example above, the URI prefix is intentionally short.

Another up-and-coming query language

A newer language is also worth mentioning, namely [Cypher](#), which was initially developed by [Neo4j](#) and then released to the [openCypher project](#) as an open standard.

The openCypher language is currently defined by the [Cypher Query Language Reference \(Version 9\)](#).

It is providing the basis for the [GQL \(Graph Query Language\)](#) project, which is in early stages of development and aims to create an ISO standard language.

How to try out a graph database for the first time

If you're ready to create a new Neptune database right away, jump to [Using an AWS CloudFormation Stack to Create a Neptune DB Cluster \(p. 170\)](#).

If you'd like an introduction to graph technology and Neptune with guidance and tutorials, try out Neptune notebooks. [Notebooks setup \(p. 30\)](#) tells you how to set them up.

For example, the Neptune workbench lets you create a fully managed environment for the Neptune Jupyter notebooks and for any new ones that you create (see [Using Neptune workbench \(p. 30\)](#)).

Use interactive Neptune graph notebooks to get going quickly and test your queries

Neptune maintains an open-source [graph notebook](#) project in GitHub that provides Jupyter notebooks with tutorials that show you how to query graphs in simple and complex ways using sample data sets.

For information about Jupyter notebooks, see the [Jupyter documentation](#).

Whether you're new to graphs and want to experiment and learn a graph query language, or you're experienced and want to test and refine your queries, the interactive coding environment that Neptune notebooks provide are a great platform and a huge time-saver.

The [Neptune workbench \(p. 30\)](#) is a managed platform for using these notebooks that you can easily set up in the [Neptune console](#) when you create a Neptune database. The Neptune workbench uses Amazon SageMaker to host the notebooks, and automatically connects to the [graph notebook](#) project for you.

You can also install Jupyter locally and run the notebooks from your laptop, connected either to Neptune or to a local instance of one of the open-source graph database. In the latter case you can experiment with graph technology as much as you want before you spend a penny, and then move smoothly to the managed production environment that Neptune offers.

[Notebooks setup \(p. 30\)](#) shows you how to get set up with Neptune notebooks in whatever way you want.

Notebooks in the graph-notebook project

- Notebooks that help you get started with simple queries are located in the [01-Getting-Started](#) folder.
- Notebooks in the [02-Visualization](#) folder provide much more detailed examples of how to load and query sample data and visualize the results.
- If you're interested in working with [Amazon Neptune ML for machine learning on graphs \(p. 522\)](#), the [04-Machine-Learning](#) folder contains notebooks that walk you through the steps it takes to train different kinds of models using data exported from a Neptune database, and query them.

Digging deeper into graph reference architecture

As you think about what problems a graph database could solve for you, and how to approach them, one of the best places to start is the [Neptune graph reference architectures GitHub project](#).

There you can find detailed descriptions of graph workload types, and three sections to help you design an effective graph database:

- [Data Models and Query Languages](#) – This section walks you through the differences between Gremlin and SPARQL and how to choose between them.
- [Graph Data Modeling](#) – This is a thorough discussion of how to make graph data modeling decisions, including detailed walkthroughs of property graph modeling using Gremlin and RDF modeling using SPARQL.
- [Converting Other Data Models to a Graph Model](#) – Here you can find out how to go about translating a relational data model into a graph model.

There are also three sections that walk you through specific steps for using Neptune:

- [Connecting to Amazon Neptune from Clients Outside the Neptune VPC](#) – This section shows you several options for connecting to Neptune from outside the VPC where your DB cluster is located.
- [Accessing Amazon Neptune from AWS Lambda Functions](#) – Here you'll find out how to connect reliably to Neptune from Lambda functions.
- [Writing to Amazon Neptune from an Amazon Kinesis Data Stream](#) – This section can help you handle high write throughput scenarios with Neptune.

Getting set up with Neptune Jupyter notebooks

The notebooks in the [graph notebook](#) project are all open-source, located in the `/src/graph_notebook/notebooks` folder of the graph-notebook GitHub repository.

You can use them to walk through setting up, configuring, populating and querying graphs using different query languages, different data sets, and even different databases on the back end.

Using the Neptune workbench to host Neptune notebooks

An easy way to get started with Amazon Neptune is to use the Neptune workbench. The workbench lets you work with your Neptune DB cluster using Jupyter notebooks hosted by Amazon SageMaker, including the ones that Neptune provides in the [graph notebook](#) project.

Neptune offers t3 instance types that you can get started with for only \$0.10/Hr (see [the Neptune pricing page](#)).

You are billed for workbench resources through Amazon SageMaker, separately from your Neptune billing.

You can use the Neptune console to set up the Neptune workbench in Amazon SageMaker and load the Neptune notebooks, or create a new Jupyter notebook of your own:

To create a Jupyter notebook using the Neptune workbench

1. Make sure that the security group attached in the VPC where Neptune is running has a rule that allows inbound connections from SageMaker.
2. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
3. In the navigation pane on the left, choose **Notebooks**.
4. Choose **Create notebook**.
5. In the **Cluster** list, choose your Neptune DB cluster. If you don't yet have a DB cluster, choose **Create cluster** to create one.
6. Give your notebook a name, and optionally a description.
7. Unless you already created an AWS Identity and Access Management (IAM) role for your notebooks, choose **Create an IAM role**, and enter an IAM role name.
8. Choose **Create notebook**. The creation process may take 10 to 15 minutes before everything is ready.
9. After your notebook is created, select it and then choose **Open notebook**.

The console can create an AWS Identity and Access Management (IAM) role for your notebooks, or you can create one yourself. The policy for this role should include the following:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::aws-neptune-notebook",
        "arn:aws:s3:::aws-neptune-notebook/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "neptune-db:connect",
      "Resource": [
        "your-cluster-arn/*"
      ]
    }
  ]
}
```

Also, the role should establish the following trust relationship:

```
{
  "Version": "2012-10-17",
```



```
"Statement": [  
  {  
    "Effect": "Allow",  
    "Principal": {  
      "Service": "sagemaker.amazonaws.com"  
    },  
    "Action": "sts:AssumeRole"  
  }  
]  
}
```

Again, getting everything ready to go can take 10 to 15 minutes.

Using Python to connect a generic SageMaker notebook to Neptune

Connecting a notebook to Neptune is easy if you have installed the Neptune magics, but it is also possible to connect a SageMaker notebook to Neptune using Python, even if you are not using a Neptune notebook.

Steps to take to connect to Neptune in a SageMaker notebook cell

1. Install the Gremlin Python client:

```
!pip install gremlinpython
```

Neptune notebooks install the Gremlin Python client for you, so this step is only necessary if you're using a plain SageMaker notebook.

2. Write code such as the following to connect and issue a Gremlin query:

```
from gremlin_python import statics  
from gremlin_python.structure.graph import Graph  
from gremlin_python.process.graph_traversal import __  
from gremlin_python.process.strategies import *  
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection  
from gremlin_python.driver.aihttp.transport import AihttpTransport  
from gremlin_python.process.traversal import *  
import os  
  
port = 8182  
server = '(your server endpoint)'  
  
endpoint = f'wss://{server}:{port}/gremlin'  
  
graph=Graph()  
  
connection = DriverRemoteConnection(endpoint, 'g',  
                                   transport_factory=lambda:AihttpTransport(call_from_event_loop=True))  
  
g = graph.traversal().withRemote(connection)  
  
results = (g.V().hasLabel('airport').  
           sample(10).  
           order().by('code').  
           local(__.values('code', 'city').fold()).  
           toList())  
  
# Print the results in a tabular form with a row index  
for i,c in enumerate(results,1):  
    print("%3d %4s %s" % (i,c[0],c[1]))
```

```
connection.close()
```

Note

If you happen to be using a version of the Gremlin Python client that is older than 3.5.0, this line:

```
connection = DriverRemoteConnection(endpoint, 'g',  
    transport_factory=lambda:AiohttpTransport(call_from_event_loop=True))
```

Would just be:

```
connection = DriverRemoteConnection(endpoint, 'g')
```

Setting up graph notebooks on your local machine

The graph-notebook project has instructions for setting up Neptune notebooks on your local machine:

- [Prerequisites](#)
- [Jupyter installation](#)
- [Connecting to a graph database.](#)

You can connect your local notebooks either to a Neptune DB cluster, or to a local or remote instance of an open-source graph database.

Using Neptune notebooks with Neptune clusters

If you are connecting to a Neptune cluster on the back end, you may want to run the notebooks in Amazon SageMaker. Connecting to Neptune from SageMaker can be more convenient than from a local installation of the notebooks, and it will let you work more easily with [Neptune ML \(p. 522\)](#).

For instructions about how to set up notebooks in SageMaker, see [Launching graph-notebook using Amazon SageMaker](#).

For instructions about how to set up and configure Neptune itself, see [Setting up Neptune \(p. 166\)](#).

You can also connect a local installation of the Neptune notebooks to a Neptune DB cluster. This can be somewhat more complicated because Amazon Neptune DB clusters can only be created in an Amazon Virtual Private Cloud (VPC), which is by design isolated from the outside world. There are a number ways to connect into a VPC from the outside it. One is to use a load balancer. Another is to use VPC peering (see the [Amazon Virtual Private Cloud Peering Guide](#)).

The most convenient way for most people, however, is to connect to set up an Amazon EC2 proxy server within the VPC and then use [SSH tunnelling](#) (also called port forwarding), to connect to it. You can find instructions about how to set up at [Connecting graph notebook locally to Amazon Neptune](#) in the `additional-databases/neptune` folder of the [graph-notebook](#) GitHub project.

Using Neptune notebooks with open-source graph databases

To get started with graph technology at no cost, you can also use Neptune notebooks with various open-source databases on the back end. Examples are the TinkerPop [Gremlin server](#), and the [Blazegraph](#) database.

To use Gremlin Server as your back-end database, follow the instructions at:

- The [Connecting graph-notebook to a Gremlin Server GitHub folder](#).
- The [graph-notebook Gremlin configuration GitHub folder](#).

To use a local instance of [Blazegraph](#) as your back-end database, follow these instructions:

- [Blazegraph quick-start instructions](#)
- The [graph-notebook Blazegraph configuration GitHub folder](#).

Using Neptune workbench magics in your notebooks

The Neptune workbench provides a number of so-called *magic* commands in the notebooks that save a great deal of time and effort. They fall into two categories: *line magics* and *cell magics*.

Line magics are commands preceded by a single percent sign (%). They only take line input, not input from the rest of the cell body. Neptune workbench provides the following line magics:

- [%seed \(p. 35\)](#)
- [%load \(p. 35\)](#)
- [%load_ids \(p. 35\)](#)
- [%load_status \(p. 35\)](#)
- [%cancel_load \(p. 35\)](#)
- [%status \(p. 36\)](#)
- [%gremlin-status \(p. 36\)](#)
- [%opencypher-status, or %oc_status \(p. 36\)](#)
- [%sparql-status \(p. 36\)](#)
- [%graph_notebook_config \(p. 36\)](#)
- [%graph_notebook_host \(p. 36\)](#)
- [%graph_notebook_version \(p. 37\)](#)
- [%graph_notebook_vis_options \(p. 37\)](#)

Cell magics are preceded by two percent signs (%%) rather than one, and use the cell content as input, although they can also take line content as input. Neptune workbench provides the following cell magics:

- [%%sparql \(p. 37\)](#)
- [%%gremlin \(p. 38\)](#)
- [%%opencypher, or %%oc \(p. 38\)](#)
- [%%graph_notebook_config \(p. 37\)](#)
- [%%graph_notebook_vis_options \(p. 39\)](#)

There are also two magics, a line magic and a cell magic, for working with [Neptune machine learning \(p. 522\)](#):

- [%neptune_ml \(p. 39\)](#)

- [%%neptune_ml](#) (p. 42)

Note

When working with Neptune magics, you can generally get help text using a `--help` or `-h` parameter. With a cell magic, the body cannot be empty, so when getting help, put filler text, even a single character, in the body. For example:

```
%%gremlin --help  
x
```

The %seed line magic

The %seed line magic is a convenient way to add data to your Neptune endpoint that you can use to explore and experiment with Gremlin, openCypher, or SPARQL queries. It provides a form where you can select the data model you want to explore (property-graph or RDF) and then choose from among a number of different sample data sets that Neptune provides.

The %load line magic

The %load line magic generates a form that you can use to submit a bulk load request to Neptune (see [Neptune Loader Command](#) (p. 209)). The source must be an Amazon S3 path in the same region as the Neptune cluster.

The %load_ids line magic

The %load_ids line magic retrieves the load IDs that have been submitted to the notebook's host endpoint (see [Neptune Loader Get-Status request parameters](#) (p. 217)). The request takes this form:

```
GET https://your-neptune-endpoint:port/loader
```

The %load_status line magic

The %load_status line magic retrieves the load status of a particular load job that has been submitted to the notebook's host endpoint, specified by the line input (see [Neptune Loader Get-Status request parameters](#) (p. 217)). The request takes this form:

```
GET https://your-neptune-endpoint:port/loader?loadId=loadId
```

The line magic looks like this:

```
%load_status load id
```

The %cancel_load line magic

The %cancel_load line magic cancels a particular load job (see [Neptune Loader Cancel Job](#) (p. 224)). The request takes this form:

```
DELETE https://your-neptune-endpoint:port/loader?loadId=loadId
```

The line magic looks like this:

```
%cancel_load load id
```

The %status line magic

Retrieves [status information \(p. 639\)](#) from the notebook's host endpoint ([%graph_notebook_config \(p. 36\)](#) shows the host endpoint).

The %gremlin-status line magic

Retrieves [Gremlin query status information \(p. 284\)](#).

The %opencypher-status line magic (also %oc_status)

Retrieves query status for an opencypher query. This line magic takes the following optional arguments:

- **--queryId** or **-q** – Specifies the ID of a specific running query for which to show the status.
- **--cancel_query** or **-c** – Cancels a running query. Does not take a value.
- **--silent** or **-s** – If **--silent** is set to `true` when cancelling a query, the running query is cancelled with an HTTP response code of 200. Otherwise, the HTTP response code would be 500.
- **--store-to** – Specifies the name of a variable in which to store the query results.

The %sparql-status line magic

Retrieves [SPARQL query status information \(p. 365\)](#).

The %graph_notebook_config line magic

This line magic displays a JSON object containing the configuration that the notebook is using to communicate with Neptune. The configuration includes:

- **host**: The endpoint to which to connect and issue commands.
- **port**: The port used when issuing commands to Neptune. The default is 8182.
- **auth_mode**: The mode of authentication to use when issuing commands to Neptune. Must be either `DEFAULT` or `IAM`.
- **load_from_s3_arn**: Specifies an Amazon S3 ARN for the `%load` magic to use. If this value is empty, the ARN must be specified in the `%load` command.
- **ssl**: A Boolean value indicating whether or not to connect to Neptune using TLS. The default value is `true`.
- **aws_region**: The region where this notebook is deployed. This information is used for IAM authentication and for `%load` requests.

You can change the configuration by copying the `%graph_notebook_config` output into a new cell and make changes to it there. Then if you run the [%%graph_notebook_config \(p. 37\)](#) cell magic on the new cell, the configuration will be changed accordingly.

The %graph_notebook_host line magic

Sets the line input as the notebook's host.

The %graph_notebook_version line magic

The %graph_notebook_version line magic returns the Neptune workbench notebook release number. For example, graph visualization was introduced in version 1.27.

The %graph_notebook_vis_options line magic

The %graph_notebook_vis_options line magic displays the current visualization settings that the notebook is using. These options are explained in the [vis.js](#) documentation.

You can modify these settings by copying the output into a new cell, making the changes you want, and then running the %%graph_notebook_vis_options cell magic on the cell.

To restore the visualization settings to their default values, you can run the %graph_notebook_vis_options line magic with a `reset` parameter. This resets all the visualization settings:

```
%graph_notebook_vis_options reset
```

The %%graph_notebook_config cell magic

The %%graph_notebook_config cell magic uses a JSON object containing configuration information to modify the settings that the notebook is using to communicate with Neptune, if possible. The configuration takes the same form returned by the [%graph_notebook_config](#) (p. 36) line magic.

For example:

```
%%graph_notebook_config
{
  "host": "my-new-cluster-endpoint.amazon.com",
  "port": 8182,
  "auth_mode": "DEFAULT",
  "load_from_s3_arn": "",
  "ssl": true,
  "aws_region": "us-east-1"
}
```

The %%sparql cell magic

The %%sparql cell magic issues a SPARQL query to the Neptune endpoint. It accepts the following optional line input:

- **-h or --help** – Returns help text about these parameters.
- **--path** – Prefixes a path to the SPARQL endpoint. For example, if you specify `--path "abc/def"` then the endpoint called would be `host:port/abc/def`.
- **--expand-all** – This is a query visualization hint that tells the visualizer to include all `?s ?p ?o` results in the graph diagram regardless of binding type.

By default, a SPARQL visualization only includes triple patterns where the `o?` is a `uri` or a `bnode` (blank node). All other `?o` binding types such as literal strings or integers are treated as properties of the `?s` node that can be viewed using the **Details** pane in the **Graph** tab.

Use the `--expand-all` query hint when you may want to include such literal values as vertices in the visualization instead.

Don't combine this visualization hint with explain parameters, because explain queries are not visualized.

- **--explain-type** – Used to specify the explain mode to use (one of: `dynamic`, `static`, or `details`).
- **--explain-format** – Used to specify the response format for an explain query (one of `text/csv` or `text/html`).
- **--store-to** – Used to specify a variable to which to store the query results.

Example of an explain query:

```
%%sparql explain
SELECT * WHERE {?s ?p ?o} LIMIT 10
```

Example of a visualization query with an `--expand-all` visualization hint parameter (see [SPARQL visualization \(p. 44\)](#)):

```
%%sparql --expand-all
SELECT * WHERE {?s ?p ?o} LIMIT 10
```

The %%gremlin cell magic

The %%gremlin cell magic issues a Gremlin query to the Neptune endpoint using WebSocket. It accepts an optional line input to toggle into [Gremlin explain \(p. 292\)](#) /> mode or [Gremlin profile API \(p. 303\)](#), and a separate optional visualization hint input to modify visualization output behavior (see [Gremlin visualization \(p. 43\)](#)).

Example of an explain query:

```
%%gremlin explain
g.V().limit(10)
```

Example of a profile query:

```
%%gremlin profile
g.V().limit(10)
```

Example of a visualization query with a visualization query hint:

```
%%gremlin -p v,outv
g.V().out().limit(10)
```

The %%opencypher cell magic (also %%oc)

The %%opencypher cell magic (which also has the abbreviated %%oc form), issues an openCypher query to the Neptune endpoint. It accepts the following optional line input arguments:

- **mode** – The query mode: either `query` or `bolt`. The default value if you don't supply this argument is `query`.

- **--group-by** or **-g** – Specifies the property used to group nodes. For example, `code`, `-id`. The default value if you don't supply this argument is `~labels`.
- **--ignore-groups** – If present, all grouping options are ignored.
- **--display-property** or **-d** – Specifies the property whose value should be displayed for each vertex. The default value if you don't supply this argument is `~labels`.
- **--edge-display-property** or **-de** – Specifies the property whose value should be displayed for each edge. The default value if you don't supply this argument is `~labels`.
- **--label-max-length** or **-l** – Specifies the maximum number of characters of a vertex label to display. The default value if you don't supply this argument is 10.
- **--store-to** or **-s** – Specifies the name of a variable in which to store the query results.

The %%graph_notebook_vis_options cell magic

The `%%graph_notebook_vis_options` cell magic lets you set visualization options for the notebook. You can copy the settings returned by the `%graph-notebook-vis-options` line magic into a new cell, make changes to them, and use the `%%graph_notebook_vis_options` cell magic to set the new values.

These options are explained in the [vis.js](#) documentation.

To restore the visualization settings to their default values, you can run the `%graph_notebook_vis_options` line magic with a `reset` parameter. This resets all the visualization settings:

```
%graph_notebook_vis_options reset
```

The %neptune_ml line magic

You can use the `%neptune_ml` line magic to initiate and manage various Neptune ML operations.

Note

You can also initiate and manage some Neptune ML operations using the [%neptune_ml \(p. 42\)](#) cell magic.

- **%neptune_ml export start** – Starts a new export job.

Parameters

- **--export-url** *exporter-endpoint* – (optional) The Amazon API Gateway endpoint where the exporter can be called.
- **--export-iam** – (optional) Flag indicating that requests to the export url must be signed using SigV4.
- **--export-no-ssl** – (optional) Flag indicating that SSL should not be used when connecting to the exporter.
- **--wait** – (optional) Flag indicating that the operation should wait until the export has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between export status checks (Default: 60).
- **--wait-timeout** *timeout-seconds* – (optional) Sets the time, in seconds, to wait for the export job to complete before returning the most recent status (Default: 3,600).
- **--store-to** *location-to-store-result* – (optional) The variable in which to store the export result. If `--wait` is specified, the final status will be stored there.
- **%neptune_ml export status** – Retrieves the status of an export job.

Parameters

- **--job-id** *export job ID* – The ID of the export job for which to retrieve status.
- **--export-url** *exporter-endpoint* – (optional) The Amazon API Gateway endpoint where the exporter can be called.
- **--export-iam** – (optional) Flag indicating that requests to the export url must be signed using SigV4.
- **--export-no-ssl** – (optional) Flag indicating that SSL should not be used when connecting to the exporter.
- **--wait** – (optional) Flag indicating that the operation should wait until the export has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between export status checks (Default: 60).
- **--wait-timeout** *timeout-seconds* – (optional) Sets the time, in seconds, to wait for the export job to complete before returning the most recent status (Default: 3,600).
- **--store-to** *location-to-store-result* – (optional) The variable in which to store the export result. If **--wait** is specified, the final status will be stored there.
- **%neptune_ml dataprocessing start** – Starts the Neptune ML dataprocessing step.

Parameters

- **--job-id** *ID for this job* – (optional) ID to assign to this job.
- **--s3-input-uri** *S3 URI* – (optional) The S3 URI at which to find the input for this dataprocessing job.
- **--config-file-name** *file name* – (optional) Name of the configuration file for this dataprocessing job.
- **--store-to** *location-to-store-result* – (optional) The variable in which to store the dataprocessing result.
- **--wait** – (optional) Flag indicating that the operation should wait until the dataprocessing has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between dataprocessing status checks (Default: 60).
- **--wait-timeout** *timeout-seconds* – (optional) Sets the time, in seconds, to wait for the dataprocessing job to complete before returning the most recent status (Default: 3,600).
- **%neptune_ml dataprocessing status** – Retrieves the status of a dataprocessing job.

Parameters

- **--job-id** *ID of the job* – ID of the job for which to retrieve the status.
- **--store-to** *instance type* – (optional) The variable in which to store the model-training result.
- **--wait** – (optional) Flag indicating that the operation should wait until the model-training has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between model-training status checks (Default: 60).
- **--wait-timeout** *timeout-seconds* – (optional) Sets the time, in seconds, to wait for the dataprocessing job to complete before returning the most recent status (Default: 3,600).
- **%neptune_ml training start** – Starts the Neptune ML model-training process.

Parameters

- **--job-id** *ID for this job* – (optional) ID to assign to this job.

- **--data-processing-id** *dataprocessing job ID* – (optional) ID of the dataprocessing job that created the artifacts to use for training.
- **--s3-output-uri** *S3 URI* – (optional) The S3 URI at which to store the output from this model-training job.
- **--instance-type** *S3 URI* – (optional) The instance size to use for this model-training job.
- **--store-to** *location-to-store-result* – (optional) The variable in which to store the model-training result.
- **--wait** – (optional) Flag indicating that the operation should wait until the model-training has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between model-training status checks (Default: 60).
- **--wait-timeout** *timeout-seconds* – (optional) Sets the time, in seconds, to wait for the model-training job to complete before returning the most recent status (Default: 3,600).
- **%neptune_ml training status** – Retrieves the status of a Neptune ML model-training job.

Parameters

- **--job-id** *ID of the job* – ID of the job for which to retrieve the status.
- **--store-to** *instance type* – (optional) The variable in which to store the status result.
- **--wait** – (optional) Flag indicating that the operation should wait until the model-training has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between model-training status checks (Default: 60).
- **--wait-timeout** *timeout-seconds* – (optional) Sets the time, in seconds, to wait for the dataprocessing job to complete before returning the most recent status (Default: 3,600).
- **%neptune_ml endpoint create** – Creates a query endpoint for a Neptune ML model.

Parameters

- **--job-id** *ID for this job* – (optional) ID to assign to this job.
- **--model-job-id** *model-training job ID* – (optional) ID of the model-training job for which to create a query endpoint.
- **--instance-type** *S3 URI* – (optional) The instance size to use for the query endpoint.
- **--store-to** *location-to-store-result* – (optional) The variable in which to store the result of the endpoint creation.
- **--wait** – (optional) Flag indicating that the operation should wait until the endpoint creation has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between status checks (Default: 60).
- **--wait-timeout** *timeout-seconds* – (optional) Sets the time, in seconds, to wait for the endpoint creation job to complete before returning the most recent status (Default: 3,600).
- **%neptune_ml endpoint status** – Retrieves the status of a Neptune ML query endpoint.

Parameters

- **--job-id** *endpoint creation ID* – (optional) ID of an endpoint creation job for which to report status.
- **--store-to** *location-to-store-result* – (optional) The variable in which to store the status result.
- **--wait** – (optional) Flag indicating that the operation should wait until the endpoint creation has completed.
- **--wait-interval** *interval-to-wait* – (optional) Sets the time, in seconds, between status checks (Default: 60).

- `--wait-timeout timeout-seconds` – (optional) Sets the time, in seconds, to wait for the endpoint creation job to complete before returning the most recent status (Default: 3,600).

The %%neptune_ml cell magic

The %%neptune_ml cell magic ignores line inputs such as `--job-id` or `--export-url`. Instead, it lets you provide those inputs and others within within the cell body.

You can also save such inputs in another cell, assigned to a Jupyter variable, and then inject them into the cell body using that variable. That way, you can use such inputs over and over without having to re-enter them all every time.

This only works if the injecting variable is the only content of the cell. You cannot use multiple variables in one cell, or a combination of text and a variable.

For example, the %%neptune_ml `export start` cell magic can consume a JSON document in the cell body that contains all the parameters described in [Parameters used to control the Neptune export process](#) (p. 244).

In the [Neptune-ML-01-Introduction-to-Node-Classification-Gremlin](#) notebook, under **Configuring Features** in the **Export the data and model configuration** section, you can see how the following cell holds export parameters in a document assigned to a Jupyter variable named `export-params`:

```
export_params = {
  "command": "export-pg",
  "params": {
    "endpoint": neptune_ml.get_host(),
    "profile": "neptune_ml",
    "useIamAuth": neptune_ml.get_iam(),
    "cloneCluster": False
  },
  "outputS3Path": f'{s3_bucket_uri}/neptune-export',
  "additionalParams": {
    "neptune_ml": {
      "targets": [
        {
          "node": "movie",
          "property": "genre"
        }
      ],
      "features": [
        {
          "node": "movie",
          "property": "title",
          "type": "word2vec"
        },
        {
          "node": "user",
          "property": "age",
          "type": "bucket_numerical",
          "range" : [1, 100],
          "num_buckets": 10
        }
      ]
    }
  },
  "jobSize": "medium"}
```

When you run this cell, Jupyter saves the parameters document under that name. Then, you can use `#{export_params}` to inject the JSON document into the body of a %%neptune_ml `export start` cell, like this:

```
%%neptune_ml export start --export-url {neptune_ml.get_export_service_host()} --export-iam  
--wait --store-to export_results  
  
${export_params}
```

Available forms of the %%neptune_ml cell magic

The %%neptune_ml cell magic can be used in the following forms:

- %%neptune_ml **export start** – Starts a Neptune ML export process.
- %%neptune_ml **dataprocessing start** – Starts a Neptune ML dataprocessing job.
- %%neptune_ml **training start** – Starts a Neptune ML model-training job.
- %%neptune_ml **endpoint create** – Creates a Neptune ML query endpoint for a model.

Graph visualization in the Neptune workbench

In many cases the Neptune workbench can create a visual diagram of your query results as well as returning them in tabular form. The graph visualization is available in the **Graph** tab in the query results whenever visualization is possible.

Note

To get access to recently added functionality and fixes in notebooks that you are already using, first stop and then re-start your notebook instance.

Visualizing Gremlin query results

Neptune workbench creates a visualization of the query results for any Gremlin query that returns a path. To see the visualization, select the **Graph** tab to the right of the **Console** tab under the query after you run it.

You can use query visualization hints to control how the visualizer diagrams query output. These hints follow the %%gremlin cell magic and are preceded by the --path-pattern (or its short form, -p) parameter name:

```
%%gremlin -p comma-separated hints
```

You can also use the --group-by (or -g) flag to specify a property of the vertices to group them by. This allows specifying a color or icon for different groups of vertices.

The names of the hints reflect the Gremlin steps commonly used when traversing between vertices, and they behave accordingly. Multiple hints can be used in combination, separated by commas, without any spaces between them. The hints used should match the corresponding Gremlin steps in the query being visualized. Here is an example:

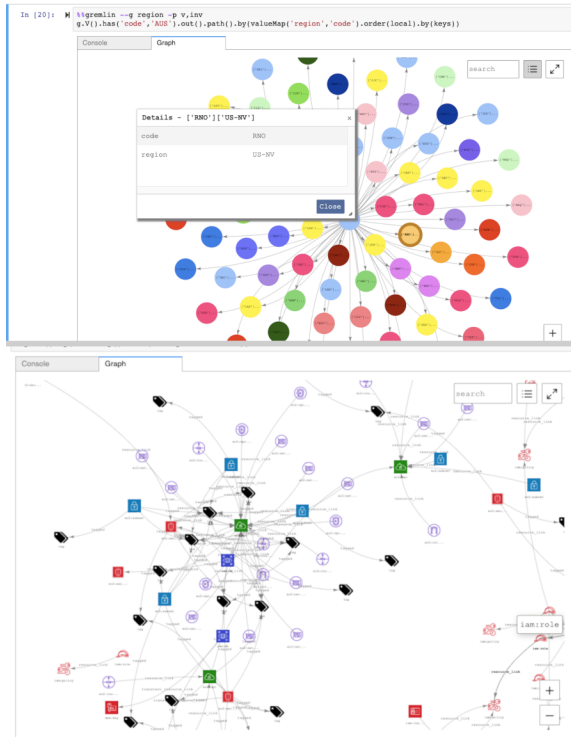
```
%%gremlin -p v,outv,inv  
g.V().hasLabel('airport').outE().inV().path().by('code').by('dist').limit(5)
```

Available visualization hints are as follows:

```
v  
inv  
outv  
e
```

```
ine  
oute
```

Here are some examples of graph visualizations using groups:



Visualizing SPARQL query results

Neptune workbench creates a visualization of the query results for any SPARQL query that takes either one of these forms:

- `SELECT ?subject ?predicate ?object`
- `SELECT ?s ?p ?o`

To see the visualization, select the **Graph** tab to the right of the **Table** tab under the query after you run it.

By default, a SPARQL visualization only includes triple patterns where the `o?` is a `uri` or a `bnode` (blank node). All other `?o` binding types such as literal strings or integers are treated as properties of the `?s` node that can be viewed using the **Details** pane in the **Graph** tab.

In many cases, however, you may want to include such literal values as vertices in the visualization. To do that, use the `--expand-all` query hint after the `%%sparql` cell magic:

```
%%sparql --expand-all
```

This tells the visualizer to include all `?s ?p ?o` results in the graph diagram regardless of binding type.

You can see this hint used throughout the `Air-Routes-SPARQL.ipynb` notebook and you can experiment by running the queries with and without the hint to see what difference it makes in the visualization.

Accessing visualization tutorial notebooks in the Neptune workbench

The two visualization tutorial notebooks that come with the Neptune workbench provide a wealth of examples in Gremlin and in SPARQL of how to query graph data effectively and visualize the results.

Navigate to the Visualization notebooks

1. In the navigation pane on the left, choose the **Open Notebook** button to the right.
2. Once the Neptune workbench opens, running Jupyter, you will see a **Neptune** folder at the top level. Choose it to open the folder.
3. At the next level is a folder named **02-Visualization**. Open this folder. Inside are several notebooks that walk you through different ways to query your graph data, in Gremlin and in SPARQL, and how to visualize the query results:
 - [Air-Routes-Gremlin](#)
 - [Air-Routes-SPARQL](#)
 - [Workbench Visualization blog](#)
 - [EPL-Gremlin](#)
 - [EPL-SPARQL](#)

Select a notebook to experiment with the queries it contains.

Overview of Amazon Neptune Features

This section provides an overview of Neptune features, including:

- [Neptune compliance with query-language standards \(p. 47\)](#).
- [Neptune's graph data model \(p. 61\)](#).
- [An explanation of the new transaction semantics that Neptune has implemented \(p. 66\)](#).
- [An introduction to Neptune clusters and instances \(p. 73\)](#).
- [Neptune's storage, reliability and availability \(p. 75\)](#).
- [An explanation of Neptune endpoints \(p. 78\)](#).
- [How Neptune's custom query IDs let you check query status \(p. 83\)](#).
- [Using Neptune's *lab mode* to enable experimental features \(p. 85\)](#).
- [A description of Neptune's experimental DFE engine \(p. 87\)](#).
- [A list of Neptune engine releases and how to update your engine \(p. 718\)](#).

Note

This section does not cover using the query language that you can use to access the data in a Neptune graph.

For information about how to connect to a running Neptune DB cluster with Gremlin, see [Accessing a Neptune graph with Gremlin \(p. 262\)](#).

For information about how to connect to a running Neptune DB cluster with openCypher, see [Accessing the Neptune Graph with openCypher \(p. 392\)](#).

For information about how to connect to a running Neptune DB cluster with SPARQL, see [Accessing the Neptune graph with SPARQL \(p. 342\)](#).

Topics

- [Notes on Amazon Neptune Standards Compliance \(p. 47\)](#)
- [Neptune Graph Data Model \(p. 61\)](#)
- [The Neptune lookup cache can accelerate read queries \(p. 64\)](#)
- [Transaction Semantics in Neptune \(p. 66\)](#)
- [Amazon Neptune DB Clusters and Instances \(p. 73\)](#)
- [Amazon Neptune storage, reliability and availability \(p. 75\)](#)
- [Connecting to Amazon Neptune Endpoints \(p. 78\)](#)
- [Inject a Custom ID Into a Neptune Gremlin or SPARQL Query \(p. 83\)](#)
- [Neptune Lab Mode \(p. 85\)](#)
- [The Amazon Neptune alternative query engine \(DFE\) \(p. 87\)](#)
- [Amazon Neptune JDBC connectivity \(p. 95\)](#)
- [Amazon Neptune Engine Updates \(p. 99\)](#)

Notes on Amazon Neptune Standards Compliance

Amazon Neptune complies with applicable standards in implementing the Gremlin and SPARQL graph query languages in most cases.

These sections describe the standards as well as those areas where Neptune extends or diverges from them.

Topics

- [Gremlin standards compliance in Amazon Neptune \(p. 47\)](#)
- [SPARQL Standards Compliance in Amazon Neptune \(p. 56\)](#)

Gremlin standards compliance in Amazon Neptune

The following sections provide an overview of the Neptune implementation of Gremlin and how it differs from the Apache TinkerPop implementation.

Neptune implements some Gremlin steps natively in its engine, and uses the Apache TinkerPop Gremlin implementation to process others (see [Native Gremlin step support in Amazon Neptune \(p. 336\)](#)). The following sections provide an overview of cases where Neptune's implementation differs from the TinkerPop implementation.

Topics

- [Applicable Standards for Gremlin \(p. 47\)](#)
- [Neptune Gremlin Implementation Differences \(p. 47\)](#)

Applicable Standards for Gremlin

- The Gremlin language is defined by [Apache TinkerPop Documentation](#) and the Apache TinkerPop implementation of Gremlin rather than by a formal specification.
- For numeric formats, Gremlin follows the IEEE 754 standard ([IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic](#). For more information, also see the [Wikipedia IEEE 754 page](#)).

Neptune Gremlin Implementation Differences

There are a few important differences between the Amazon Neptune implementation of Gremlin and the implementation defined by Apache TinkerPop.

Note

For some concrete examples of these implementation differences shown in Gremlin Console and Amazon Neptune, see the [the section called "Using Gremlin" \(p. 173\)](#) section of the Quick Start.

The following is a list of implementation differences:

Topics

- [Pre-Bound Variables \(p. 48\)](#)
- [TinkerPop Enumerations \(p. 48\)](#)
- [Java Code \(p. 49\)](#)
- [Date and Time \(p. 49\)](#)
- [Script Execution \(p. 49\)](#)

- [Sessions \(p. 50\)](#)
- [Transactions \(p. 50\)](#)
- [Vertex and Edge IDs \(p. 50\)](#)
- [User-Supplied IDs \(p. 50\)](#)
- [Vertex Property IDs \(p. 51\)](#)
- [Cardinality of Vertex Properties \(p. 51\)](#)
- [Updating a Vertex Property \(p. 51\)](#)
- [Labels \(p. 51\)](#)
- [Variables \(p. 51\)](#)
- [Escape Characters \(p. 51\)](#)
- [Groovy Limitations \(p. 51\)](#)
- [Serialization \(p. 51\)](#)
- [Lambda Steps \(p. 52\)](#)
- [Unsupported Gremlin Methods \(p. 52\)](#)
- [Unsupported Gremlin Steps \(p. 52\)](#)
- [Other Features \(p. 52\)](#)
- [Gremlin Graph Supported Features \(p. 53\)](#)

Pre-Bound Variables

The traversal object `g` is Pre-bound. The `graph` object is not supported.

TinkerPop Enumerations

Neptune does not support fully qualified class names for enumeration values. For example, you must use `single` and not `org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.single` in your Groovy request.

The enumeration type is determined by parameter type.

The following table shows the allowed enumeration values and the related TinkerPop fully qualified name.

Allowed Values	Class
<code>id, key, label, value</code>	<code>org.apache.tinkerpop.gremlin.structure.T</code>
<code>T.id, T.key, T.label, T.value</code>	<code>org.apache.tinkerpop.gremlin.structure.T</code>
<code>set, single</code>	<code>org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality</code>
<code>asc, desc, shuffle</code>	<code>org.apache.tinkerpop.gremlin.process.traversal.Order</code>
<code>Order.asc, Order.desc, Order.shuffle</code>	<code>org.apache.tinkerpop.gremlin.process.traversal.Order</code>
<code>global, local</code>	<code>org.apache.tinkerpop.gremlin.process.traversal.Scope</code>
<code>Scope.global, Scope.local</code>	<code>org.apache.tinkerpop.gremlin.process.traversal.Scope</code>
<code>all, first, last, mixed</code>	<code>org.apache.tinkerpop.gremlin.process.traversal.Pop</code>
<code>normSack</code>	<code>org.apache.tinkerpop.gremlin.process.traversal.SackFunctions.Ba</code>

addAll, and, assign, div, max, min, minus, mult, or, sum, sumLong	org.apache.tinkerpop.gremlin.process.traversal.Operator
keys, values	org.apache.tinkerpop.gremlin.structure.Column
BOTH, IN, OUT	org.apache.tinkerpop.gremlin.structure.Direction
any, none	org.apache.tinkerpop.gremlin.process.traversal.step.TraversalOpt

Note

The `incr` and `decr` enum options on `Order` have been deprecated by TinkerPop in favor of `asc` and `desc` respectively. Please use the new options, because `incr` and `decr` will be removed in future versions.

Java Code

Neptune does not support calls to methods defined by arbitrary Java or Java library calls other than supported Gremlin APIs. For example, `java.lang.*`, `Date()`, and `g.V().tryNext().orElseGet()` are not allowed.

Date and Time

Neptune provides the `datetime` method for specifying dates and times for queries sent in the Gremlin **Groovy** variant. This includes the Gremlin Console, text strings using the HTTP REST API, and any other serialization that uses Groovy. A date and time value must be specified with the `datetime()` function.

Important

This **only** applies to methods where you send the Gremlin query as a **text string**. If you are using a Gremlin Language Variant (GLV), you must use the native date classes and functions for the language. For more information, see the Best Practices section, [the section called "Native Date and Time" \(p. 698\)](#).

The `datetime()` function takes a string value with an ISO-8601 compliant datetime up to millisecond precision. For example, `datetime('2018-01-01T00:00:00')`.

Some examples of ISO-8601 datetime formats are as follows:

- YYYY-MM-DD
- YYYY-MM-DDTHH:mm
- YYYY-MM-DDTHH:mm:SS
- YYYY-MM-DDTHH:mm:SS.ssss
- YYYY-MM-DDTHH:mm:SSZ

Important

If you enter a datetime value in your graph that has an explicit time zone specified, and later query that value, the value is returned in Coordinated Universal Time (UTC) without any time-zone information.

The reason for this is that Neptune transforms and stores all dates and times as UTC so that they can be compared efficiently. This is important for performance, but results in the loss of information about the originating time zone.

Script Execution

All queries must begin with `g`, the traversal object.

In String query submissions, multiple traversals can be issued separated by a semicolon (;) or a newline character (\n). To be executed, every statement other than the last must end with an `.iterate()`

step. Only the final traversal data is returned. Note that this does not apply to GLV ByteCode query submissions.

Sessions

Sessions in Neptune are limited to only 10 minutes in duration. See [Gremlin sessions \(p. 286\)](#) and the [TinkerPop Session Reference](#) for more information.

Transactions

Neptune opens a new transaction at the beginning of each Gremlin traversal and closes the transaction upon the successful completion of the traversal. The transaction is rolled back when there is an error.

Multiple statements separated by a semicolon (;) or a newline character (\n) are included in a single transaction. Every statement other than the last must end with a `next()` step to be executed. Only the final traversal data is returned.

Manual transaction logic using `tx.commit()` and `tx.rollback()` is not supported.

Vertex and Edge IDs

Neptune Gremlin Vertex and Edge IDs must be of type `String`. If you don't supply an ID when you add a vertex or an edge, a UUID is generated and converted to a string; for example, "48af8178-50ce-971a-fc41-8c9a954cea62".

Note

This means that user-supplied IDs are supported, but they are optional in normal usage. However, the Neptune `Load` command requires that all IDs be specified using the `~id` field in the Neptune CSV format.

User-Supplied IDs

User-supplied IDs are allowed in Neptune Gremlin with the following stipulations.

- Supplied IDs are optional.
- Only vertexes and edges are supported.
- Only type `String` is supported.

To create a new vertex with a custom ID, use the `property` step with the `id` keyword:
`g.addV().property(id, 'customid')`.

Note

Do not put quotation marks around the `id` keyword.

All vertex IDs must be unique, and all edge IDs must be unique. However, Neptune does allow a vertex and an edge to have the same ID.

If you try to create a new vertex using the `g.addV()` and a vertex with that ID already exists, the operation fails. The exception to this is if you specify a new label for the vertex, the operation succeeds but adds the new label and any additional properties specified to the existing vertex. Nothing is overwritten. A new vertex is not created. The vertex ID does not change and remains unique.

For example, the following Gremlin Console commands succeed:

```
gremlin> g.addV('label1').property(id, 'customid')
gremlin> g.addV('label2').property(id, 'customid')
gremlin> g.V('customid').label()
==>label1::label2
```

Vertex Property IDs

Vertex property IDs are generated automatically and can show up as positive or negative numbers when queried.

Cardinality of Vertex Properties

Neptune supports set cardinality and single cardinality. If it isn't specified, set cardinality is selected. This means that if you set a property value, it adds a new value to the property, but only if it doesn't already appear in the set of values. This is the Gremlin enumeration value of [Set](#).

`List` is not supported. For more information about property cardinality, see the [Vertex](#) topic in the Gremlin JavaDoc.

Updating a Vertex Property

To update a property value without adding an additional value to the set of values, specify `single` cardinality in the property step.

```
g.V('exampleid01').property(single, 'age', 25)
```

This removes all existing values for the property.

Labels

Neptune supports multiple labels for a vertex. When you create a label, you can specify multiple labels by separating them with `::`. For example, `g.addV("Label1::Label2::Label3")` adds a vertex with three different labels. The `hasLabel` step matches this vertex with any of those three labels: `hasLabel("Label1")`, `hasLabel("Label2")`, and `hasLabel("Label3")`.

Important

The `::` delimiter is reserved for this use only. You cannot specify multiple labels in the `hasLabel` step. For example, `hasLabel("Label1::Label2")` does not match anything.

Variables

Neptune does not support Gremlin variables and does not support the `bindings` property.

Escape Characters

Neptune resolves all escape characters as described in the [Escaping Special Characters](#) section of the Apache Groovy language documentation.

Groovy Limitations

Neptune doesn't support Groovy commands that don't start with `g`. This includes `math` (for example, `1+1`), `system` calls (for example, `System.nanoTime()`), and `variable` definitions (for example, `1+1`).

Important

Neptune does not support fully qualified class names. For example, you must use `single` and not `org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.single` in your Groovy request.

Serialization

Neptune supports the following serializations based on the requested MIME type.

MIME type	Serialization	Configuration
application/vnd.gremlin-v1.0+gryo	GryoMessageSerializerV1d0	ioRegistries: [org.apache.tinkerpop.gremlin.tinker
application/vnd.gremlin-v1.0+gryo-stringd	GryoMessageSerializerV1d0	serializeResultToString: true}
application/vnd.gremlin-v3.0+gryo	GryoMessageSerializerV3d0	ioRegistries: [org.apache.tinkerpop.gremlin.tinker
application/vnd.gremlin-v3.0+gryo-stringd	GryoMessageSerializerV3d0	serializeResultToString: true
application/vnd.gremlin-v1.0+json	GraphSONMessageSerializerV1d0	ioRegistries: [org.apache.tinkerpop.gremlin.tinker
application/vnd.gremlin-v2.0+json	GraphSONMessageSerializerV2d0 (only works with WebSockets)	ioRegistries: [org.apache.tinkerpop.gremlin.tinker
application/vnd.gremlin-v3.0+json	GraphSONMessageSerializerV3d0	ioRegistries: [org.apache.tinkerpop.gremlin.tinker
application/json	GraphSONMessageSerializerV3d0	ioRegistries: [org.apache.tinkerpop.gremlin.tinker
application/vnd.graphbinary-v1.0	GraphBinaryMessageSerializerV1	

Lambda Steps

Neptune does not support Lambda Steps.

Unsupported Gremlin Methods

Neptune does not support the following Gremlin methods:

- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.program(org.apac`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.sideEffect(java.`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.from(org.apache.`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.to(org.apache.ti`

For example, the following traversal is not allowed:

```
g.V().addE('something').from(g.V().next()).to(g.V().next()).
```

Unsupported Gremlin Steps

Neptune does not support the following Gremlin steps:

- The Gremlin [io\(\) Step](#) is not supported in Neptune. For example, the query `g.io("graph.xml").read().iterate()` would not work with Neptune.

Other Features

The Neptune implementation of Gremlin does not expose the graph object. The following section describes the supported and unsupported graph features.

Gremlin Graph Supported Features

The following is a set of features as implemented by the Neptune Gremlin graph. These features are the same as would be returned by the `graph.features()` command.

Graph Feature	Enabled
Transactions	true
ThreadedTransactions	false
Computer	false
Persistence	true
ConcurrentAccess	true

Variable Feature	Enabled
Variables	false
SerializableValues	false
UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	false
ByteValues	false
DoubleValues	false
FloatValues	false
IntegerValues	false
LongValues	false
MapValues	false
MixedListValues	false
StringValues	false
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

Vertex Feature	Enabled
----------------	---------

MetaProperties	false
DuplicateMultiProperties	false
AddVertices	true
RemoveVertices	true
MultiProperties	true
UserSuppliedIds	true
AddProperty	true
RemoveProperty	true
NumericIds	false
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false

Vertex Property Feature	Enabled
UserSuppliedIds	false
AddProperty	true
RemoveProperty	true
NumericIds	true
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false
Properties	true
SerializableValues	false
UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	true
ByteValues	true
DoubleValues	true

FloatValues	true
IntegerValues	true
LongValues	true
MapValues	false
MixedListValues	false
StringValues	true
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

Edge Feature	Enabled
AddEdges	true
RemoveEdges	true
UserSuppliedIds	true
AddProperty	true
RemoveProperty	true
NumericIds	false
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false

Edge Property Feature	Enabled
Properties	true
SerializableValues	false
UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	true
ByteValues	true

DoubleValues	true
FloatValues	true
IntegerValues	true
LongValues	true
MapValues	false
MixedListValues	false
StringValues	true
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

SPARQL Standards Compliance in Amazon Neptune

Amazon Neptune complies with the following standards in implementing the SPARQL graph query language.

Topics

- [Applicable Standards for SPARQL \(p. 56\)](#)
- [Neptune SPARQL Implementation Specifics \(p. 56\)](#)

Applicable Standards for SPARQL

- SPARQL is defined by the W3C [SPARQL 1.1 Query Language](#) recommendation of March 21, 2013.
- The SPARQL Update protocol and query language are defined by the W3C [SPARQL 1.1 Update](#) specification.
- For numeric formats, SPARQL follows the [W3C XML Schema Definition Language \(XSD\) 1.1 Part 2: Datatypes](#) specification, which is consistent with the IEEE 754 specification ([IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic](#)). For more information, see also the [Wikipedia IEEE 754 page](#). However, features that were introduced after the [IEEE 754-1985](#) version are not included in the specification.

Neptune SPARQL Implementation Specifics

The following sections provide specific details of how SPARQL is implemented in Neptune.

Topics

- [Default Namespace Prefixes in Neptune SPARQL \(p. 57\)](#)
- [SPARQL Default Graph and Named Graphs \(p. 57\)](#)
- [SPARQL XPath Constructor Functions Supported by Neptune \(p. 57\)](#)
- [xsd:dateTime Values in Neptune \(p. 58\)](#)
- [Neptune Handling of Special Floating Point Values \(p. 58\)](#)
- [Neptune Limitation of Arbitrary-Length Values \(p. 58\)](#)
- [Neptune Extends Equals Comparison in SPARQL \(p. 59\)](#)
- [Handling of Out-of-Range Literals in Neptune SPARQL \(p. 59\)](#)

Default Namespace Prefixes in Neptune SPARQL

Neptune defines the following prefixes by default for use in SPARQL queries. For more information, see [Prefixed Names](#) in the SPARQL specification.

- `rdf` – `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- `rdfs` – `http://www.w3.org/2000/01/rdf-schema#`
- `owl` – `http://www.w3.org/2002/07/owl#`
- `xsd` – `http://www.w3.org/2001/XMLSchema#`

SPARQL Default Graph and Named Graphs

Amazon Neptune associates every triple with a named graph. The default graph is defined as the union of all named graphs.

Default Graph for Queries

If you submit a SPARQL query without explicitly specifying a graph via the `GRAPH` keyword or constructs such as `FROM NAMED`, Neptune always considers all triples in your DB instance. For example, the following query returns all triples from a Neptune SPARQL endpoint:

```
SELECT * WHERE { ?s ?p ?o }
```

Triples that appear in more than one graph are returned only once.

For information about the default graph specification, see the [RDF Dataset](#) section of the SPARQL 1.1 Query Language specification.

Specifying the Named Graph for Loading, Inserts, or Updates

If you don't specify a named graph when loading, inserting, or updating triples, Neptune uses the fallback named graph defined by the URI `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`.

When you issue a Neptune `Load` request using a triple-based format, you can specify the named graph to use for all triples by using the `parserConfiguration: namedGraphUri` parameter. For information about the `Load` command syntax, see [the section called "Loader Command" \(p. 209\)](#).

Important

If you don't use this parameter, and you don't specify a named graph, the fallback URI is used:
`http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`.

This fallback named graph is also used if you load triples via `SPARQL UPDATE` without explicitly providing a named graph target.

You can use the quads-based format N-Quads to specify a named graph for each triple in the database.

Note

Using N-Quads allows you to leave the named graph blank. In this case, `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` is used. You can override the default named graph for N-Quads using the `namedGraphUri` parser configuration option.

SPARQL XPath Constructor Functions Supported by Neptune

The SPARQL standard allows SPARQL engines to support an extensible set of XPath constructor functions. Neptune currently supports the following constructor functions, where the `xsd` prefix is defined as `http://www.w3.org/2001/XMLSchema#`:

- `xsd:boolean`
- `xsd:integer`
- `xsd:double`
- `xsd:float`
- `xsd:decimal`
- `xsd:long`
- `xsd:unsignedLong`

xsd:dateTime Values in Neptune

For performance reasons, Neptune always stores date/time values as Coordinated Universal Time (UTC). This makes direct comparisons very efficient.

This also means that if you enter a `dateTime` value that specifies a particular time zone, Neptune translates the value to UTC and discards that time-zone information. Then, when you retrieve the `dateTime` value later, it is expressed in UTC, not the time of the original time zone, and you can no longer tell what that original time zone was.

Neptune Handling of Special Floating Point Values

Neptune handles special floating-point values in SPARQL as follows.

SPARQL NaN Handling in Neptune

In Neptune, SPARQL can accept a value of `NaN` in a query. No distinction is made between signaling and quiet `NaN` values. Neptune treats all `NaN` values as quiet.

Semantically, no comparison of a `NaN` is possible, because nothing is greater than, less than, or equal to a `NaN`. This means that a value of `NaN` on one side of a comparison in theory never matches *anything* on the other side.

However, the [XSD specification](#) does treat two `xsd:double` or `xsd:float` `NaN` values as equal. Neptune follows this for the `IN` filter, for the equal operator in filter expressions, and for exact match semantics (having a `NaN` in the object position of a triple pattern).

SPARQL Infinite Value Handling in Neptune

In Neptune, SPARQL can accept a value of `INF` or `-INF` in a query. `INF` compares as greater than any other numeric value, and `-INF` compares as less than any other numeric value.

Two `INF` values with matching signs compare as equal to each other regardless of their type (for example, a float `-INF` compares as equal to a double `-INF`).

Of course, no comparison with a `NaN` is possible because nothing is greater than, less than, or equal to a `NaN`.

SPARQL Negative Zero Handling in Neptune

Neptune normalizes a negative zero value to an unsigned zero. You can use negative zero values in a query, but they aren't recorded as such in the database, and they compare as equal to unsigned zeros.

Neptune Limitation of Arbitrary-Length Values

Neptune limits the storage size of XSD integer, floating point, and decimal values in SPARQL to 64 bits. As a result, larger values might be arbitrarily truncated, with unexpected results.

Neptune Extends Equals Comparison in SPARQL

The SPARQL standard defines a ternary logic for value expressions, where a value expression can either evaluate to `true`, `false`, or `error`. The default semantics for term equality as defined in the [SPARQL 1.1 specification](#)), which applies to `=` and `!=` comparisons in `FILTER` conditions, produces an `error` when comparing data types that are not explicitly comparable in the [operators table](#) in the specification.

This behavior can lead to unintuitive results, as in the following example.

Data:

```
<http://example.com/Server/1> <http://example.com/ip> "127.0.0.1"^^<http://example.com/datatype/IPAddress>
```

Query 1:

```
SELECT * WHERE {
  <http://example.com/Server/1> <http://example.com/ip> ?o .
  FILTER(?o = "127.0.0.2"^^<http://example.com/datatype/IPAddress>)
}
```

Query 2:

```
SELECT * WHERE {
  <http://example.com/Server/1> <http://example.com/ip> ?o .
  FILTER(?o != "127.0.0.2"^^<http://example.com/datatype/IPAddress>)
}
```

With the default SPARQL semantics that Neptune used before release 1.0.2.1, both queries would return the empty result. The reason is that `?o = "127.0.0.2"^^<http://example.com/IPAddress>` when evaluated for `?o := "127.0.0.1"^^<http://example.com/IPAddress>` produces an `error` rather than `false` because there are no explicit comparison rules specified for the custom data type `<http://example.com/IPAddress>`. As a result, the negated version in the second query also produces an `error`. In both queries, the `error` causes the candidate solution to be filtered out.

Starting with release 1.0.2.1, Neptune has extended the SPARQL inequality operator in accord with the specification. See the [SPARQL 1.1 section on operator extensibility](#), which allows engines to define additional rules on how to compare across user-defined and non-comparable built-in data types.

Using this option, Neptune now treats a comparison of any two data types that is not explicitly defined in the operator-mapping table as evaluating to `true` if the literal values and data types are syntactically equal, and `false` otherwise. An `error` is not produced in any case.

Using these new semantics, the second query would return `"127.0.0.1"^^<http://example.com/IPAddress>` instead of an empty result.

Handling of Out-of-Range Literals in Neptune SPARQL

XSD semantics define each numeric type with its value space, except for `integer` and `decimal`. These definitions limit each type to a range of values. For example, the range of an `xsd:byte` range is from -128 to +127, inclusive. Any value outside of this range is considered invalid.

If you try to assign a literal value outside of the value space of a type (for example, if you try to set an `xsd:byte` to a literal value of 999), Neptune accepts the out-of-range value as-is, without rounding or truncating it. But it doesn't persist it as a numeric value because the given type can't represent it.

That is, Neptune accepts `"999"^^xsd:byte` even though it is a value outside of the defined `xsd:byte` value range. However, after the value is persisted in the database, it can only be used in exact match

semantics, in an object position of a triple pattern. No range filter can be executed on it because out-of-range literals are not treated as numeric values.

The SPARQL 1.1 specification defines [range operators](#) in the form *numeric-operator-numeric*, *string-operator-string*, *literal-operator-literal*, and so forth. Neptune can't execute a range comparison operator anything like *invalid-literal-operator-numeric-value*.

Neptune Graph Data Model

The basic unit of Amazon Neptune graph data is a four-position (quad) element, which is similar to a Resource Description Framework (RDF) quad. The following are the four positions of a Neptune quad:

- `subject` (S)
- `predicate` (P)
- `object` (O)
- `graph` (G)

Each quad is a statement that makes an assertion about one or more resources. A statement can assert the existence of a relationship between two resources, or it can attach a property (key-value pair) to a resource. You can think of the quad predicate value generally as the verb of the statement. It describes the type of relationship or property that's being defined. The object is the target of the relationship, or the value of the property. The following are examples:

- A relationship between two vertices can be represented by storing the source vertex identifier in the S position, the target vertex identifier in the O position, and the edge label in the P position.
- A property can be represented by storing the element identifier in the S position, the property key in the P position, and the property value in the O position.

The graph position G is used differently in the different stacks. For RDF data in Neptune, the G position contains a [named graph identifier](#). For property graphs in Gremlin, it is used to store the edge ID value in the case of an edge. In all other cases, it defaults to a fixed value.

User-facing values in a quad statement are usually stored separately in a dictionary index, where the statement indexes reference them using an 8-byte long term identifier. The exception to this is numeric values, including date and datetime values (represented as milliseconds from the epoch). These can be stored inline directly in the statement indexes.

A set of quad statements with shared resource identifiers creates a graph.

How Statements Are Indexed in Neptune

When you query a graph of quads, for each quad position, you can either specify a value constraint, or not. The query returns all the quads that match the value constraints that you specified.

Neptune uses indexes to resolve queries. In the 2005 paper, *Optimized Index Structures for Querying RDF from the Web*, Andreas Harth and Stefan Decker observed that there are 16 (2^4) possible access patterns for the four quad positions. You can query all 16 patterns efficiently without having to scan and filter by using six quad statement indexes. Each quad statement index uses a key that is composed of the four position values concatenated in a different order.

Access Pattern	Index key order
1. ???? (No constraints; returns every quad)	SPOG
2. SPOG (Every position is constrained)	SPOG
3. SPO? (S, P, and O are constrained; G is not)	SPOG
4. SP?? (S and P are constrained; O and G are not)	SPOG
5. S??? (S is constrained; P, O, and G are not)	SPOG
6. ?POG (P, O, and G are constrained; S is not)	POGS
7. ?PO? (P and O are constrained; S and G are not)	POGS
8. ?P?? (P is constrained; S, O, and G are not)	POGS

9.	?P?G (P and G are constrained; S and O are not)	GPSO
10.	SP?G (S, P, and G are constrained; O is not)	GPSO
11.	S??G (S and G are constrained; P and O are not)	GPSO
12.	???G (G is constrained; S, P, and O are not)	GPSO
13.	S?OG (S, O, and G are constrained; P is not)	OGSP
14.	??OG (O and G are constrained; S and P are not)	OGSP
15.	??O? (O is constrained; S, P, and G are not)	OGSP
16.	S?O? (S and O are constrained; P and G are not)	OSGP

Neptune creates and maintains only three out of those six indexes by default:

- **SPOG** - Uses a key composed of Subject + Predicate + Object + Graph.
- **POGS** - Uses a key composed of Predicate + Object + Graph + Subject.
- **GPSO** - Uses a key composed of Graph + Predicate + Subject + Object.

These three indexes handle many of the most common access patterns. Maintaining only three full statement indexes instead of six greatly reduces the resources that you need to support rapid access without scanning and filtering. For example, the **SPOG** index allows efficient lookup whenever a prefix of the positions, such as the vertex or vertex and property identifier, is bound. The **POGS** index allows efficient access when only the edge or property label stored in **P** position is bound.

The low-level API for finding statements takes a statement pattern in which some positions are known and the rest are left for discovery by index search. By composing the known positions into a key prefix according to the index key order for one of the statement indexes, Neptune performs a range scan to retrieve all the statements matching the known positions.

However, one of the statement indexes that Neptune does *not* create by default is a reverse traversal **OSGP** index, which can gather predicates across objects and subjects. Instead, Neptune by default tracks distinct predicates in a separate index that it uses to do a union scan of {all **P** x **POGS**}. When you are working with Gremlin, a predicate corresponds to a property or an edge label.

If the number of distinct predicates in a graph becomes large, the default Neptune access strategy can become inefficient. In Gremlin, for example, an `in()` step where no edge labels are given, or any step that uses `in()` internally such as `both()` or `drop()`, may become quite inefficient.

Enabling OSGP Index Creation Using Lab Mode

If your data model creates a large number of distinct predicates, you may experience reduced performance and higher operational costs that can be dramatically improved by using Lab Mode to enable the **OSPG index** (p. 86) in addition to the three indexes that Neptune maintains by default.

Note

This feature is available starting in [Neptune engine release 1.0.1.0.200463.0](#) (p. 767).

Enabling the **OSPG** index can have a few down-sides:

- The insert rate may slow by up to 23%.
- Storage increases by up to 20%.
- Read queries that touch all indexes equally (which is quite rare) may have increased latencies.

In general, however, it is worth enabling the **OSGP** index for DB Clusters with a large number of distinct predicates. Object-based searches become highly efficient (for example, finding all incoming edges to a vertex, or all subjects connected to a given object), and as a result dropping vertices becomes much more efficient too.

Important

You can only enable the OSGP index in an empty DB cluster, before you load any data into it.

Gremlin statements in the Neptune data model

Gremlin property-graph data is expressed in the SPOG model using three classes of statements, namely:

- [Vertex Label Statements \(p. 294\)](#)
- [Edge Statements \(p. 294\)](#)
- [Property Statements \(p. 294\)](#)

For an explanation of how these are used in Gremlin queries, see [Understanding how Gremlin queries work in Neptune \(p. 293\)](#).

The Neptune lookup cache can accelerate read queries

Amazon Neptune implements a lookup cache that uses the R5d instance's NVMe-based SSD to improve read performance for queries with frequent, repetitive lookups of property values or RDF literals. The lookup cache temporarily stores these values in the NVMe SSD volume where they can be accessed rapidly.

This feature is available starting with [Amazon Neptune Engine Version 1.0.4.2.R2 \(2021-06-01\)](#) (p. 732).

Read queries that return the properties of a large number of vertices and edges, or many RDF triples, can have a high latency if the property values or literals need to be retrieved from cluster storage volumes rather than memory. Examples include long-running read queries that return a large number of full names from an identity graph, or of IP addresses from a fraud-detection graph. As the number of property values or RDF literals returned by your query increases, available memory decreases and your query execution can significantly degrade.

Use cases for the Neptune lookup cache

The lookup cache only helps when your read queries are returning the properties of a very large number of vertices and edges, or of RDF triples.

To optimize query performance, Amazon Neptune uses the R5d instance type to create a large cache for such property values or literals. Retrieving them from the cache is then much faster than retrieving them from cluster storage volumes.

As a rule of thumb, it's only worthwhile to enable the lookup cache if all three of the following conditions are met:

- You have been observing increased latency in read queries.
- You're also observing a drop in the `BufferCacheHitRatio` [CloudWatch metric](#) (p. 644) when running read queries (see [Monitoring Neptune Using Amazon CloudWatch](#) (p. 641)).
- Your read queries are spending a lot of time in materializing return values prior to rendering the results (see the [Gremlin-profile](#) example below for a way to determine how many property values are being materialized for a query).

Note

This feature is helpful *only* in the specific scenario described above. For example, the lookup cache doesn't help aggregation queries at all. Unless you are running queries that would benefit from the lookup cache, there is no reason to use an R5d instance type instead of an equivalent and less expensive R5 instance type.

If you're using Gremlin, you can assess the materialization costs of a query with the [Gremlin profile API](#) (p. 303). Under "Index Operations", it shows the number of terms materialized during execution:

```
Index Operations
Query execution:
  # of statement index ops: 3
  # of unique statement index ops: 3
  Duplication ratio: 1.0
  # of terms materialized: 5273
Serialization:
  # of statement index ops: 200
  # of unique statement index ops: 140
```

```
Duplication ratio: 1.43  
# of terms materialized: 32693
```

The number of non-numerical terms that are materialized is directly proportional to the number of term look-ups that Neptune has to perform.

Using the lookup cache

The lookup cache is only available on an `R5d` instance type, where it is automatically enabled by default. Neptune `R5d` instances have the same specifications as `R5` instances, plus up to 1.8 TB of local NVMe-based SSD storage. Lookup caches are instance-specific, and workloads that benefit can be directed specifically to `R5d` instances in a Neptune cluster, while other workloads can be directed to `R5` or other instance types.

To use the lookup cache on a Neptune instance, simply upgrade that instance to the `R5d` instance type. When you do, Neptune automatically sets the `neptune_lookup_cache` (p. 410) DB cluster parameter to `'enabled'`, and creates the lookup cache on that particular instance. You can then use the [Instance Status](#) (p. 639) API to confirm that the cache has been enabled.

Similarly, to disable the lookup cache on a given instance, scale the instance down from an `R5d` instance type to an equivalent `R5` instance type.

When an `R5d` instance is launched, the lookup cache is enabled and in cold-start mode, meaning that it is empty. Neptune first checks in the lookup cache for property values or RDF literals while processing queries, and adds them if they are not yet present. This gradually warms up the cache.

When you direct the read queries that require property-value or RDF-literal lookups to an `R5d reader` instance, read performance degrades slightly while its cache is warming up. When the cache is warmed up, however, read performance speeds up significantly and you may also see a drop in I/O costs related to lookups hitting the cache rather than cluster storage. Memory utilization also improves.

If your *writer* instance is an `R5d`, it warms up its lookup cache automatically on every write operation. This approach does increase latency for write queries slightly, but warms up the lookup cache more efficiently. Then if you direct the read queries that require property-value or RDF-literal lookups to the writer instance, you start getting improved read performance immediately, since the values have already been cached there.

Also, if you are running the bulk loader on an `R5d` writer instance, you may notice that its performance is slightly degraded because of the cache.

Because the lookup cache is specific to each node, host replacement resets the cache to a cold start.

You can temporarily disable the lookup cache on all instances in your DB cluster by setting the `neptune_lookup_cache` (p. 410) DB cluster parameter to `'disabled'`. In general, however, it makes more sense to disable the cache on specific instances by scaling them down from `R5d` to `R5` instance types.

Transaction Semantics in Neptune

Amazon Neptune is designed to support highly concurrent online transactional processing (OLTP) workloads over data graphs. The [W3C SPARQL Query Language for RDF](#) specification and the [Apache TinkerPop Gremlin Graph Traversal Language](#) documentation do not define transaction semantics for concurrent query processing. Because ACID support and well-defined transaction guarantees can be very important, Neptune has introduced formalized semantics to help you avoid data anomalies.

This section defines these semantics and illustrates how they apply to various common use cases in Neptune.

Topics

- [Definition of Isolation Levels \(p. 66\)](#)
- [Transaction Isolation Levels in Neptune \(p. 67\)](#)
- [Examples of Neptune Transaction Semantics \(p. 69\)](#)
- [Exception Handling and Retries \(p. 71\)](#)

Definition of Isolation Levels

The "I" in ACID stands for *isolation*. The degree of isolation of a transaction determines how much or little other concurrent transactions can affect the data that it operates on.

The [SQL:1992 Standard](#) created a vocabulary for describing isolation levels. It defines three types of interactions (that it calls *phenomena*) that can occur between two concurrent transactions, Tx1 and Tx2:

- **Dirty read** – This occurs when Tx1 modifies an item, and then Tx2 reads that item before Tx1 has committed the change. Then, if Tx1 never succeeds in committing the change, or rolls it back, Tx2 has read a value that never made it into the database.
- **Non-repeatable read** – This happens when Tx1 reads an item, then Tx2 modifies or deletes that item and commits the change, and then Tx1 tries to reread the item. Tx1 now reads a different value than before, or finds that the item no longer exists.
- **Phantom read** – This happens when Tx1 reads a set of items that satisfy a search criterion, and then Tx2 adds a new item that satisfies the search criterion, and then Tx1 repeats the search. Tx1 now obtains a different set of items than it did before.

Each of these three types of interaction can cause inconsistencies in the resulting data in a database.

The SQL:1992 standard defined four isolation levels that have different guarantees in terms of the three types of interaction and the inconsistencies that they can produce. At all four levels, a transaction can be guaranteed to execute completely or not at all:

- **READ UNCOMMITTED** – Allows all three kinds of interaction (that is, dirty reads, non-repeatable reads, and phantom reads).
- **READ COMMITTED** – Dirty reads are not possible, but nonrepeatable and phantom reads are.
- **REPEATABLE READ** – Neither dirty reads nor nonrepeatable reads are possible, but phantom reads still are.
- **SERIALIZABLE** – None of the three types of interaction phenomena can occur.

Multiversion concurrency control (MVCC) allows one other kind of isolation, namely *SNAPSHOT* isolation. This guarantees that a transaction operates on a snapshot of data as it exists when the transaction begins, and that no other transaction can change that snapshot.

Transaction Isolation Levels in Neptune

Amazon Neptune implements different transaction isolation levels for read-only queries and for mutation queries. SPARQL and Gremlin queries are classified as read-only or mutation based on the following criteria:

- In SPARQL, there is a clear distinction between read queries (`SELECT`, `ASK`, `CONSTRUCT`, and `DESCRIBE` as defined in the [SPARQL 1.1 Query Language](#) specification), and mutation queries (`INSERT` and `DELETE` as defined in the [SPARQL 1.1 Update](#) specification).

Note that Neptune treats multiple mutation queries submitted together (for example, in a `POST` message, separated by semicolons) as a single transaction. They are guaranteed either to succeed or fail as an atomic unit, and in the case of failure, partial changes are rolled back.

- However, in Gremlin, Neptune classifies a query as a read-only query or a mutation query based on whether it contains any query-path steps such as `addE()`, `addV()`, or `drop()` that manipulates data. If the query contains any such path step, it is classified and executed as a mutation query.

It is also possible to use standing sessions in Gremlin. For more information, see [Gremlin sessions \(p. 286\)](#). In these sessions, all queries, including read-only queries, are executed with locking reads as if they were mutation queries.

Topics

- [Read-Only Query Isolation \(p. 67\)](#)
- [Mutation Query Isolation \(p. 68\)](#)
- [Conflict Resolution Using Lock-Wait Timeouts \(p. 68\)](#)

Read-Only Query Isolation

Neptune evaluates read-only queries under snapshot isolation semantics. This means that a read-only query logically operates on a consistent snapshot of the database taken when query evaluation begins. Neptune can then guarantee that none of the following phenomena will happen:

- **Dirty reads** – Read-only queries in Neptune will never see uncommitted data from a concurrent transaction.
- **Non-repeatable reads** – A read-only transaction that reads the same data more than once will always get back the same values.
- **Phantom reads** – A read-only transaction will never read data that was added after the transaction began.

Because snapshot isolation is achieved using multiversion concurrency control (MVCC), read-only queries have no need to lock data and therefore do not block mutation queries.

Read replicas only accept read-only queries, so all queries against read replicas execute under `SNAPSHOT` isolation semantics.

The only additional consideration when querying a read replica is that there can be a small replication lag between the writer and read replicas. This means that an update made on the writer might take a short time to be propagated to the read replica you are reading from. The actual replication time depends on the write-load against the primary instance. Neptune architecture supports low-latency replication and the replication lag is instrumented in an Amazon CloudWatch metric.

Still, because of the `SNAPSHOT` isolation level, read queries always see a consistent state of the database, even if it is not the most recent one.

In cases where you require a strong guarantee that a query observes the result of a previous update, send the query to the writer endpoint itself rather than to a read replica.

Mutation Query Isolation

Reads made as part of mutation queries are executed under `READ COMMITTED` transaction isolation, which rules out the possibility of dirty reads. Going beyond the usual guarantees provided for `READ COMMITTED` transaction isolation, Neptune provides the strong guarantee that neither `NON-REPEATABLE` nor `PHANTOM` reads can happen.

These strong guarantees are achieved by locking records and ranges of records when reading data. This prevents concurrent transactions from making insertions or deletions in index ranges after they have been read, thus guaranteeing repeatable reads.

Note

However, a concurrent mutation transaction `Tx2` could begin after the start of mutation transaction `Tx1`, and could commit a change before `Tx1` had locked data to read it. In that case, `Tx1` would see `Tx2`'s change just as if `Tx2` had completed before `Tx1` started. Because this only applies to committed changes, a `dirty read` could never occur.

To understand the locking mechanism that Neptune uses for mutation queries, it helps first to understand the details of the Neptune [Graph Data Model \(p. 61\)](#) and [Indexing Strategy \(p. 61\)](#). Neptune manages data using three indexes, namely `SPOG`, `POGS`, and `GPSO`.

To achieve repeatable reads for the `READ COMMITTED` transaction level, Neptune takes range locks in the index that is being used. For example, if a mutation query reads all properties and outgoing edges of a vertex named `person1`, the node would lock the entire range defined by the prefix `S=person1` in the `SPOG` index before reading the data.

The same mechanism applies when using other indexes. For example, when a mutation transaction looks up all the source-target vertex pairs for a given edge label using the `POGS` index, the range for the edge label in the `P` position would be locked. Any concurrent transaction, regardless of whether it was a read-only or mutation query, could still perform reads within the locked range. However, any mutation involving insertion or deletion of new records in the locked prefix range would require an exclusive lock and would be prevented.

In other words, when a range of the index has been read by a mutation transaction, there is a strong guarantee that this range will not be modified by any concurrent transactions until the end of the reading transaction. This guarantees that no `non-repeatable reads` will occur.

Conflict Resolution Using Lock-Wait Timeouts

If a second transaction tries to modify a record in a range that a first transaction has locked, Neptune detects the conflict immediately and blocks the second transaction.

If no dependency deadlock is detected, Neptune automatically applies a lock-wait timeout mechanism, in which the blocked transaction waits for up to 60 seconds for the transaction that holds the lock to finish and release the lock.

- If the lock-wait timeout expires before the lock is released, the blocked transaction is rolled back.
- If the lock is released within the lock-wait timeout, the second transaction is unblocked and can finish successfully without needing to retry.

However, if Neptune detects a dependency deadlock between the two transactions, automatic reconciliation of the conflict is not possible. In this case, Neptune immediately cancels and rolls back the second transaction without initiating a lock-wait timeout.

Examples of Neptune Transaction Semantics

The following examples illustrate different use cases for transaction semantics in Amazon Neptune.

Topics

- [Example 1 – Inserting a Property Only If It Does Not Exist \(p. 69\)](#)
- [Example 2 – Asserting That a Property Value Is Globally Unique \(p. 69\)](#)
- [Example 3 – Changing a Property If Another Property Has a Specified Value \(p. 70\)](#)
- [Example 4 – Replacing an Existing Property \(p. 70\)](#)
- [Example 5 – Avoiding Dangling Properties or Edges \(p. 71\)](#)

Example 1 – Inserting a Property Only If It Does Not Exist

Suppose that you want to ensure that a property is set only once. For example, suppose that multiple queries are trying to assign a person a credit score concurrently. You only want one instance of the property to be inserted, and the other queries to fail because the property has already been set.

```
# GREMLIN:
g.V('person1').hasLabel('Person').coalesce(has('creditScore'), property('creditScore', 'AAA+'))

# SPARQL:
INSERT { :person1 :creditScore "AAA+" .}
WHERE { :person1 rdf:type :Person .
        FILTER NOT EXISTS { :person1 :creditScore ?o .}
```

The Gremlin `property()` step inserts a property with the given key and value. The `coalesce()` step executes the first argument in the first step, and if it fails, then it executes the second step:

Before inserting the value for the `creditScore` property for a given `person1` vertex, a transaction must try to read the possibly non-existent `creditScore` value for `person1`. This attempted read locks the SP range for `S=person1` and `P=creditScore` in the SPOG index where the `creditScore` value either exists or will be written.

Taking this range lock prevents any concurrent transaction from inserting a `creditScore` value concurrently. When there are multiple parallel transactions, at most one of them can update the value at a time. This rules out the anomaly of more than one `creditScore` property being created.

Example 2 – Asserting That a Property Value Is Globally Unique

Suppose that you want to insert a person with a Social Security number as a primary key. You would want your mutation query to guarantee that, at a global level, no one else in the database has that same Social Security number:

```
# GREMLIN:
g.V().has('ssn', 123456789).fold()
.coalesce(__.unfold(),
          __.addV('Person').property('name', 'John Doe').property('ssn', 123456789))

# SPARQL:
INSERT { :person1 rdf:type :Person .
        :person1 :name "John Doe" .
        :person1 :ssn 123456789 .}
WHERE { FILTER NOT EXISTS { ?person :ssn 123456789 } }
```

This example is similar to the previous one. The main difference is that the range lock is taken on the POGS index rather than the SPOG index.

The transaction executing the query must read the pattern, `?person :ssn 123456789`, in which the P and O positions are bound. The range lock is taken on the POGS index for `P=ssn` and `O=123456789`.

- If the pattern does exist, no action is taken.
- If it does not exist, the lock prevents any concurrent transaction from inserting that Social Security number also

Example 3 – Changing a Property If Another Property Has a Specified Value

Suppose that various events in a game move a person from level one to level two, and assign them a new `level2Score` property set to zero. You need to be sure that multiple concurrent instances of such a transaction could not create multiple instances of the level-two score property. The queries in Gremlin and SPARQL might look like the following.

```
# GREMLIN:
g.V('person1').hasLabel('Person').has('level', 1)
  .property('level2Score', 0)
  .property(Cardinality.single, 'level', 2)

# SPARQL:
DELETE { :person1 :level 1 .}
INSERT { :person1 :level2Score 0 .
        :person1 :level 2 .}
WHERE { :person1 rdf:type :Person .
        :person1 :level 1 .}
```

In Gremlin, when `Cardinality.single` is specified, the `property()` step either adds a new property or replaces an existing property value with the new value that is specified.

Any update to a property value, such as increasing the `level` from 1 to 2, is implemented as a deletion of the current record and insertion of a new record with the new property value. In this case, the record with level number 1 is deleted and a record with level number 2 is reinserted.

For the transaction to be able to add `level2Score` and update the `level` from 1 to 2, it must first validate that the `level` value is currently equal to 1. In doing so, it takes a range lock on the SPO prefix for `S=person1`, `P=level`, and `O=1` in the SPOG index. This lock prevents concurrent transactions from deleting the version 1 triple, and as a result, no conflicting concurrent updates can happen.

Example 4 – Replacing an Existing Property

Certain events might update a person's credit score to a new value (here BBB). But you want to be sure that concurrent events of that type can't create multiple credit score properties for a person.

```
# GREMLIN:
g.V('person1').hasLabel('Person')
  .sideEffect(properties('creditScore').drop())
  .property('creditScore', 'BBB')

# SPARQL:
DELETE { :person1 :creditScore ?o .}
INSERT { :person1 :creditScore "BBB" .}
WHERE { :person1 rdf:type :Person .
        :person1 :creditScore ?o .}
```

This case is similar to example 2, except that instead of locking the `SPO` prefix, Neptune locks the `SP` prefix with `S=person1` and `P=creditScore` only. This prevents concurrent transactions from inserting or deleting any triples with the `creditScore` property for the `person1` subject.

Example 5 – Avoiding Dangling Properties or Edges

The update on an entity should not leave a dangling element, that is, a property or edge associated to an entity that is not typed. This is only an issue in SPARQL, because Gremlin has built-in constraints to prevent dangling elements.

```
# SPARQL:
tx1: INSERT { :person1 :age 23 } WHERE { :person1 rdf:type :Person }
tx2: DELETE { :person1 ?p ?o }
```

The `INSERT` query must read and lock the `SPO` prefix with `S=person1`, `P=rdf:type`, and `O=Person` in the `SPOG` index. The lock prevents the `DELETE` query from succeeding in parallel.

In the race between the `DELETE` query trying to delete the `:person1 rdf:type :Person` record and the `INSERT` query reading the record and creating a range lock on its `SPO` in the `SPOG` index, the following outcomes are possible:

- If the `INSERT` query commits before the `DELETE` query reads and deletes all records for `:person1`, `:person1` is removed entirely from the database, including the newly inserted record.
- If the `DELETE` query commits before the `INSERT` query tries to read the `:person1 rdf:type :Person` record, the read observes the committed change. That is, it does not find any `:person1 rdf:type :Person` record and hence becomes a no-op.
- If the `INSERT` query reads before the `DELETE` query does, the `:person1 rdf:type :Person` triple is locked and the `DELETE` query is blocked until the `INSERT` query commits, as in the first case previously.
- If the `DELETE` reads before the `INSERT` query, and the `INSERT` query tries to read and take a lock on the `SPO` prefix for the record, a conflict is detected. This is because the triple has been marked for removal, and the `INSERT` then fails.

In all these different possible sequences of events, no dangling edge is created.

Exception Handling and Retries

When transactions are canceled because of unresolvable conflicts or lock-wait timeouts, Amazon Neptune responds with a `ConcurrentModificationException`. For more information, see [Engine Error Codes \(p. 711\)](#). As a best practice, clients should always catch and handle these exceptions.

In many cases, when the number of `ConcurrentModificationException` instances is low, an exponential backoff-based retry mechanism works well as a way to handle them. In such a retry approach, the maximum number of retries and waiting time generally depends on the maximum size and duration of the transactions.

However, if your application has highly concurrent update workloads, and you observe a large number of `ConcurrentModificationException` events, you might be able to modify your application to reduce the number of conflicting concurrent modifications.

For example, consider an application that makes frequent updates to a set of vertices and uses multiple concurrent threads for these updates to optimize the write throughput. If each thread continuously executes queries that update one or more node properties, concurrent updates of the same node can produce `ConcurrentModificationExceptions`. This in turn can degrade write performance.

You can greatly reduce the likelihood of such collisions if you can serialize updates that are likely to conflict with each other. For example, if you can ensure that all update queries for a given node are

made on the same thread (maybe using a hash-based assignment), you can be sure that they will be executed one after another rather than concurrently. Although it is still possible that a range lock taken on a neighboring node can cause a `ConcurrentModificationException`, you eliminate concurrent updates to the same node.

Amazon Neptune DB Clusters and Instances

An Amazon Neptune *DB cluster* manages access to your data through queries. A cluster consists of:

- One *primary DB instance*.
- Up to 15 *read-replica DB instances*.

All the instances in a cluster share the same [underlying managed storage volume \(p. 75\)](#), which is designed for reliability and high availability.

You connect to the DB instances in your DB cluster through [Neptune endpoints \(p. 78\)](#).

The primary DB instance in a Neptune DB cluster

The primary DB instance coordinates all write operations to the DB cluster's underlying storage volume. It also supports read operations.

There can only be one primary DB instance in a Neptune DB cluster. If the primary instance becomes unavailable, Neptune automatically fails over to one of the read-replica instances with a priority that you can specify.

Read-replica DB instances in a Neptune DB cluster

After you create the primary instance for a DB cluster, you can create up to 15 read-replica instances in your DB cluster to support read-only queries.

Neptune read-replica DB instances work well for scaling read capacity because they are fully dedicated to read operations on your cluster volume. All write operations are managed by the primary instance. Each read-replica DB instance has its own endpoint.

Because the cluster storage volume is shared among all instances in a cluster, all read-replica instances return the same data for query results with very little replication lag. This lag is usually much less than 100 milliseconds after the primary instance writes an update, although it can be somewhat longer when the volume of write operations is very large.

Having one or more read-replica instances available in different Availability Zones can increase availability, because read-replicas serve as failover targets for the primary instance. That is, if the primary instance fails, Neptune promotes a read-replica instance to become the primary instance. When this happens, there is a brief interruption while the promoted instance is rebooted, during which read and write requests made to the primary instance fail with an exception.

By contrast, if your DB cluster doesn't include any read-replica instances, your DB cluster remains unavailable when the primary instance fails until it has been re-created. Re-creating the primary instance takes considerably longer than promoting a read-replica.

To ensure high availability, we recommend that you create one or more read-replica instances that have the same DB instance class as the primary instance and are located in different Availability Zones than the primary instance. See [Fault Tolerance for a Neptune DB Cluster \(p. 674\)](#).

When you create read-replicas across Availability Zones, Neptune automatically provisions and maintains them synchronously. The primary DB instance is synchronously replicated across Availability Zones to the read-replicas to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups. Running a DB instance with high availability can enhance availability during planned system maintenance, and help protect your databases against failure and Availability Zone disruption.

Using the console, you can create a Multi-AZ deployment by simply specifying Multi-AZ when creating a DB cluster. If a DB cluster is in a single Availability Zone, you can make it a Multi-AZ DB cluster adding a Neptune replica in a different Availability Zone.

Note

You can't create an encrypted read-replica instance for an unencrypted Neptune DB cluster, or an unencrypted read-replica instance for an encrypted Neptune DB cluster.

For details on how to create a Neptune read-replica DB instance, see [Creating a Neptune Replica Using the Console \(p. 423\)](#).

Sizing DB instances in a Neptune DB cluster

Size the instances in your Neptune DB cluster based on your CPU and memory requirements. The number of vCPUs on an instance determines the number of query threads that handle incoming queries. The amount of memory on an instance determines the size of the buffer cache, used for storing copies of data pages fetched from the underlying storage volume.

Each Neptune DB instance has a number of query threads equal to 2 x number of vCPUs on that instance. An `r4.xlarge`, for example, with 2 vCPUs, has 4 query threads, and can therefore process 4 requests concurrently. An `r5.4xlarge`, with 16 vCPUs, has 32 query threads, and can therefore process 32 queries concurrently.

Additional queries that arrive while all query threads are occupied are put into a server-side queue, and processed in a FIFO manner as query threads become available. This server-side queue can hold approximately 8000 pending requests. Once it's full, Neptune respond to additional requests with a `ThrottlingException`. You can monitor the number of pending requests with the `MainRequestQueuePendingRequests` CloudWatch metric, or by using the [Gremlin query status endpoint \(p. 284\)](#) with the `includeWaiting` parameter.

Query execution time from a client perspective includes of any time spent in the queue, in addition to the time taken to actually execute the query.

A sustained concurrent write load that utilizes all the query threads on the primary DB instance ideally shows 90% or more CPU utilization, which indicates that all the query threads on the server are actively engaged in doing useful work. However, actual CPU utilization is often somewhat lower, even under a sustained concurrent write load. This is usually because query threads are waiting on I/O operations to the underlying storage volume to complete. Neptune uses quorum writes that make six copies of your data across three Availability Zones, and four out of those six storage nodes must acknowledge a write for it to be considered durable. While a query thread waits for this quorum from the storage volume, it is stalled, which reduces CPU utilization.

If you have a serial write load where you are performing one write after another and waiting for the first to complete before beginning the next, you can expect the CPU utilization to be lower still. The exact amount will be a function of the number of vCPUs and query threads (the more query threads, the less overall CPU per query), with some reduction caused by waiting for I/O.

Monitoring DB instance performance in Neptune

You can use CloudWatch metrics in Neptune to monitor the performance of your DB instances and keep track of query latency as observed by the client. See [Using CloudWatch to monitor DB instance performance in Neptune \(p. 643\)](#).

Amazon Neptune storage, reliability and availability

Amazon Neptune uses a distributed and shared storage architecture that scales automatically as your database storage needs grow.

Neptune data is stored in a cluster volume, which is a single, virtual volume that uses Non-Volatile Memory Express (NVMe) SSD-based drives. The cluster volume consists of a collection of logical blocks known as segments. Each of these segments is allocated 10 gigabytes (GB) of storage. The data in each segment is replicated into six copies, which are then allocated across three availability zones (AZs) in the AWS Region where the DB cluster resides.

When a Neptune DB cluster is created, it is allocated a single segment of 10 GB. As the volume of data increases and exceeds the currently allocated storage, Neptune automatically expands the cluster volume by adding new segments. A Neptune cluster volume can grow to a maximum size of 64 tebibytes (TiB).

The DB cluster volume contains all your user data, indices and dictionaries (described in the [Neptune Graph Data Model \(p. 61\)](#) section, as well as internal metadata such as internal transaction logs. All this graph data, including indices and internal logs, cannot exceed the maximum size of the cluster volume, which is 64 TiB.

Neptune storage allocation

Even though a Neptune cluster volume can grow to 64 TiB, you are only charged for the space actually allocated. The total space allocated is determined by the storage *high water mark*, which is the maximum amount allocated to the cluster volume at any time during its existence.

This means that even if user data is removed from a cluster volume, such as by a drop query like `g.V().drop()`, the total allocated space remains the same. Neptune does automatically optimize the unused allocated space for reuse in the future.

In addition to user data, two additional types of content consume internal storage space, namely dictionary data and internal transaction logs. Although dictionary data is stored with graph data, it persists indefinitely, even when the graph data it supports has been deleted, which means that entries can be re-used if data is re-introduced. Internal log data is stored in a separate internal storage space that has its own high internal log expires, the storage it occupied can be re-used for other logs, but not water mark. When an for graph data. The amount of internal space that has been allocated for logs is included in the total space reported by the `VolumeBytesUsed` [CloudWatch metric \(p. 641\)](#).

Check [Storage best practices \(p. 76\)](#) for ways to keep allocated storage to a minimum and to re-use space.

Neptune storage billing

Storage costs are billed based on the storage *high water mark*, as described above. Although your data is replicated into six copies, you are only billed for one copy of the data.

You can determine what the current storage high water mark of your DB cluster is by monitoring the `VolumeBytesUsed` CloudWatch metric (see [Monitoring Neptune Using Amazon CloudWatch \(p. 641\)](#)).

Other factors that can affect your Neptune storage costs include database snapshots and backup, which are billed separately as backup storage and are based on the Neptune storage costs (see [CloudWatch metrics that are useful for managing Neptune backup storage \(p. 675\)](#)).

If you create a [clone \(p. 437\)](#) of your database, however, the clone points to the same cluster volume that your DB cluster itself uses, so there is no additional storage charge for the original data. Subsequent changes to the clone use the [copy-on-write protocol \(p. 438\)](#), and do result in additional storage costs.

For more Neptune pricing information, see [Amazon Neptune Pricing](#).

Neptune storage best practices

Because certain types of data consume permanent storage in Neptune, use these best practices to avoid large spikes in storage growth:

- When designing your graph data model, avoid as much as possible using property keys and user-facing values that are temporary in nature.
- If you plan on making changes to your data model, do not load data onto an existing DB cluster using the new model until you have cleared the data in that DB cluster using the [fast reset API \(p. 419\)](#). The best thing is often to load data that uses a new model onto a new DB cluster.
- Transactions that operate on large amounts of data generate correspondingly large internal logs, which can permanently increase the high water mark of the internal log space. For example, a single transaction that deletes all the data in your DB cluster could generate a huge internal log that would require allocating a great deal of internal storage and thus permanently reduce space available for graph data.

To avoid this, split large transactions into smaller ones and allow time between them so that the associated internal logs have a chance to expire and release their internal storage for re-use by subsequent logs.

- For monitoring the growth of your Neptune cluster volume, you can set a CloudWatch alarm on the `VolumeBytesUsed` CloudWatch metric. This can be particularly helpful if your data is reaching the maximum size of the cluster volume. For more information, see [Using Amazon CloudWatch alarms](#).

The only way to shrink the storage space used by your DB cluster when you have a large amount of unused allocated space is to export all the data in your graph and then reload it into a new DB cluster. See [Neptune's data export service and utility \(p. 540\)](#) for an easy way to export data from a DB cluster, and [Neptune's bulk loader \(p. 190\)](#) for an easy way to import data back into Neptune.

Note

Creating and restoring a [snapshot \(p. 679\)](#) does not reduce the amount of storage allocated for your DB cluster, because a snapshot retains the original image of the cluster's underlying storage. If a substantial amount of your allocated storage is not being used, the only way to shrink the amount of allocated storage is to export your graph data and reload it into a new DB cluster.

Neptune storage reliability and high availability

Amazon Neptune is designed to be reliable, durable, and fault tolerant.

The fact that six copies of your Neptune data are maintained across three availability zones (AZs) ensures that storage of the data is highly durable, with very low likelihood of data loss. The data is replicated automatically across the availability zones regardless of whether there are DB instances in them, and the amount of replication is independent of the number of DB instances in your cluster.

This means that you can add a read-replica quickly, because Neptune doesn't make a new copy of the graph data. Instead, the read-replica connects to the cluster volume that already contains your data. Similarly, removing a read-replica doesn't remove any of the underlying data.

You can delete the cluster volume and its data only after deleting all of its DB instances.

Neptune also automatically detects failures in the segments that make up the cluster volume. When a copy of the data in a segment is corrupted, Neptune immediately repairs that segment, using other copies of the data within the same segment to ensure that the repaired data is current. As a result, Neptune avoids data loss and reduces the need to perform point-in-time restore to recover from a disk failure.

Connecting to Amazon Neptune Endpoints

Amazon Neptune uses a cluster of DB instances rather than a single instance. Each Neptune connection is handled by a specific DB instance. When you connect to a Neptune cluster, the host name and port that you specify point to an intermediate handler called an *endpoint*. An endpoint is a URL that contains a host address and a port. Neptune endpoints use encrypted Transport Layer Security/Secure Sockets Layer (TLS/SSL) connections.

Neptune uses the endpoint mechanism to abstract these connections so that you don't have to hardcode the hostnames, or write your own logic for rerouting connections when some DB instances are unavailable.

Using endpoints, you can map each connection to the appropriate instance or group of instances, depending on your use case. Custom endpoints let you connect to subsets of DB instances. The following endpoints are available in a Neptune DB cluster:

Cluster endpoint

A cluster endpoint is an endpoint for a Neptune DB cluster that connects to the current primary DB instance for that DB cluster. Each Neptune DB cluster has a cluster endpoint and one primary DB instance.

The cluster endpoint provides failover support for read/write connections to the DB cluster. Use the cluster endpoint for all write operations on the DB cluster, including inserts, updates, deletes, and data definition language (DDL) changes. You can also use the cluster endpoint for read operations, such as queries.

If the current primary DB instance of a DB cluster fails, Neptune automatically fails over to a new primary DB instance. During a failover, the DB cluster continues to serve connection requests to the cluster endpoint from the new primary DB instance, with minimal interruption of service.

The following example illustrates a cluster endpoint for a Neptune DB cluster.

```
mydbcluster.cluster-123456789012.us-east-1.neptune.amazonaws.com:8182
```

Reader endpoint

A reader endpoint is an endpoint for a Neptune DB cluster that connects to one of the available Neptune replicas for that DB cluster. Each Neptune DB cluster has a reader endpoint. If there is more than one Neptune replica, the reader endpoint directs each connection request to one of the Neptune replicas.

The reader endpoint provides round-robin routing for read-only connections to the DB cluster. Use the reader endpoint for read operations, such as queries.

You can't use the reader endpoint for write operations unless you have a single-instance cluster (a cluster with no read-replicas). In that case and that case only, the reader can be used for write operations as well as read operations.

The reader endpoint round-robin routing works by changing the host that the DNS entry points to. Each time you resolve the DNS, you get a different IP, and connections are opened against those IPs. After a connection is established, all the requests for that connection are sent to the same host. The client must create a new connection and resolve the DNS record again to get a connection to potentially different read replica.

Note

WebSockets connections are often kept alive for long periods. To get different read replicas, do the following:

- Ensure that your client resolves the DNS entry each time it connects.

- Close the connection and reconnect.

Various client software might resolve DNS in different ways. For example, if your client resolves DNS and then uses the IP for every connection, it directs all requests to a single host.

DNS caching for clients or proxies resolves the DNS name to the same endpoint from the cache. This is a problem for both round robin routing and failover scenarios.

Note

Disable any DNS caching settings to force DNS resolution each time.

The DB cluster distributes connection requests to the reader endpoint among available Neptune replicas. If the DB cluster contains only a primary DB instance, the reader endpoint serves connection requests from the primary DB instance. If a Neptune replica is created for that DB cluster, the reader endpoint continues to serve connection requests to the reader endpoint from the new Neptune replica, with minimal interruption in service.

The following example illustrates a reader endpoint for a Neptune DB cluster.

```
mydbcluster.cluster-ro-123456789012.us-east-1.neptune.amazonaws.com:8182
```

Instance endpoint

An instance endpoint is an endpoint for a DB instance in a Neptune DB cluster that connects to that specific DB instance. Each DB instance in a DB cluster, regardless of instance type, has its own unique instance endpoint. So, there is one instance endpoint for the current primary DB instance of the DB cluster. There is also one instance endpoint for each of the Neptune replicas in the DB cluster.

The instance endpoint provides direct control over connections to the DB cluster, for scenarios where using the cluster endpoint or reader endpoint might not be appropriate. For example, your client application might require fine-grained load balancing based on workload type. In this case, you can configure multiple clients to connect to different Neptune replicas in a DB cluster to distribute read workloads.

The following example illustrates an instance endpoint for a DB instance in a Neptune DB cluster.

```
mydbinstance.123456789012.us-east-1.neptune.amazonaws.com:8182
```

Custom endpoint

A *custom endpoint* for a Neptune cluster represents a set of DB instances that you choose. When you connect to the endpoint, Neptune chooses one of the instances in the group to handle the connection. You define which instances this endpoint refers to, and you decide what purpose the endpoint serves.

A Neptune DB cluster has no custom endpoints until you create one, and you can create up to five custom endpoints for each provisioned Neptune cluster.

The custom endpoint provides load-balanced database connections based on criteria other than the read-only or read/write capability of the DB instances. Because the connection can go to any DB instance associated with the endpoint, make sure that all the instances within that group share the same performance and memory capacity characteristics. When you use custom endpoints, you typically don't use the reader endpoint for that cluster.

This feature is intended for advanced users with specialized kinds of workloads where it isn't practical to keep all the Neptune Replicas in the cluster identical. With custom endpoints, you can adjust the capacity of the DB instances used with each connection.

For example, if you define several custom endpoints that connect to groups of instances with different instance classes, you can then direct users with different performance needs to the endpoints that best suit their use cases.

The following example illustrates a custom endpoint for a DB instance in a Neptune DB cluster:


```
myendpoint.cluster-custom-123456789012.us-east-1.neptune.amazonaws.com:8182
```

Neptune endpoint considerations

Consider the following when working with Neptune endpoints:

- Before using an instance endpoint to connect to a specific DB instance in a DB cluster, consider using the cluster endpoint or reader endpoint for the DB cluster instead.

The cluster endpoint and reader endpoint provide support for high-availability scenarios. If the primary DB instance of a DB cluster fails, Neptune automatically fails over to a new primary DB instance. It does so by either promoting an existing Neptune replica to a new primary DB instance or creating a new primary DB instance. If a failover occurs, you can use the cluster endpoint to reconnect to the newly promoted or created primary DB instance, or use the reader endpoint to reconnect to one of the other Neptune replicas in the DB cluster.

If you don't take this approach, you can still make sure that you're connecting to the right DB instance in the DB cluster for the intended operation. To do so, you can manually or programmatically discover the resulting set of available DB instances in the DB cluster and confirm their instance types after failover, before using the instance endpoint of a specific DB instance.

For more information about failovers, see [Fault Tolerance for a Neptune DB Cluster \(p. 674\)](#).

- The reader endpoint only directs connections to available Neptune replicas in a Neptune DB cluster. It does not direct specific queries.

Important

Neptune does not load balance.

If you want to load balance queries to distribute the read workload for a DB cluster, you must manage that in your application. You must use instance endpoints to connect directly to Neptune replicas to balance the load.

- The reader endpoint round-robin routing works by changing the host that the DNS entry points to. The client must create a new connection and resolve the DNS record again to get a connection to a potentially new read replica.
- During a failover, the reader endpoint might direct connections to the new primary DB instance of a DB cluster for a short time, when a Neptune replica is promoted to the new primary DB instance.

Custom endpoints

When you add a DB instance to a custom endpoint or remove it from a custom endpoint, any existing connections to that DB instance remain active.

You can define a list of DB instances to include in a custom endpoint (the *static* list), or one to exclude from the custom endpoint (the *exclusion* list). You can use the inclusion/exclusion mechanism to subdivide the DB instances into groups and make sure that the custom endpoints covers all the DB instances in the cluster. Each custom endpoint can contain only one of these list types.

In the AWS Management Console, the choice is represented by the check box **Attach future instances added to this cluster**. When you keep check box clear, the custom endpoint uses a static list containing

only the DB instances specified in the dialog. When you check the check box, the custom endpoint uses an exclusion list. In that case, the custom endpoint represents all DB instances in the cluster (including any that you add in the future) except the ones left unselected in the dialog.

Neptune doesn't change the DB instances specified in the static or exclusion lists when DB instances change roles between primary instance and Neptune Replica because of failover or promotion.

You can associate a DB instance with more than one custom endpoint. For example, suppose you add a new DB instance to a cluster. In that case the DB instance is added to all custom endpoints for which it is eligible. The static or exclusion list defined for it determines which DB instance can be added to it.

If an endpoint includes a static list of DB instances, newly added Neptune Replicas aren't added to it. Conversely, if the endpoint has an exclusion list, newly added Neptune Replicas are added to it provided that they aren't named in the exclusion list.

If a Neptune Replica becomes unavailable, it remains associated with its custom endpoints. This is true whether it is unhealthy, stopped, rebooting, or unavailable for another reason. However, as long as it remains unavailable you can't connect to it through any endpoint.

Because newly created Neptune clusters have no custom endpoints, you must create and manage them yourself. This is also true for Neptune clusters restored from snapshots, because custom endpoints are not included in the snapshot. You have create them again after restoring, and choose new endpoint names if the restored cluster is in the same region as the original one.

Creating a custom endpoint

Manage custom endpoints using the Neptune console. Do this by navigating to the details page for your Neptune cluster and use the controls in the **Custom Endpoints** section.

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Navigate to the cluster detail page.
3. Choose the `Create custom endpoint` action in the **Endpoints** section.
4. Choose a name for the custom endpoint that is unique for your user ID and region. The name must be 63 characters or less in length and take the following form:

endpointName.cluster-custom-customerDnsIdentifier.dnsSuffix

Because custom endpoint names don't include the name of your cluster, you don't have to change those names if you rename a cluster. However, you can't reuse the same custom endpoint name for more than one cluster in the same region. Give each custom endpoint a name that is unique across the clusters owned by your user ID within a particular region.

5. To choose a list of DB instances that remains the same even as the cluster expands, keep the check box **Attach future instances added to this cluster** clear. When that check box is checked, the custom endpoint dynamically adds any new instances as that are added to the cluster.

Viewing custom endpoints

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Navigate to the cluster detail page of your DB cluster.
3. The **Endpoints** section only contains information about custom endpoints (details about the built-in endpoints are listed in the main **Details** section). To see details for a specific custom endpoint, select its name to bring up the detail page for that endpoint.

Editing a custom endpoint

You can edit the properties of a custom endpoint to change which DB instances are associated with it. You can also switch between a static list and an exclusion list.

You can't connect to or use a custom endpoint while the changes from an edit action are in progress. It might take some minutes after you make a change before the endpoint status returns to **Available** and you can connect again.

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Navigate to the cluster detail page.
3. In the **Endpoints** section, choose the name of the custom endpoint you want to edit.
4. In the detail page for that endpoint, choose the **Edit** action.

Deleting a custom endpoint

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Navigate to the cluster detail page.
3. In the **Endpoints** section, choose the name of the custom endpoint you want to delete.
4. In the detail page for that endpoint, choose the **Delete** action.

Inject a Custom ID Into a Neptune Gremlin or SPARQL Query

By default, Neptune assigns a unique `queryId` value to every query. You can use this ID to get information about a running query (see [Gremlin query status API \(p. 284\)](#) or [SPARQL query status API \(p. 365\)](#)), or cancel it (see [Gremlin query cancellation \(p. 286\)](#) or [SPARQL query cancellation \(p. 367\)](#)).

Neptune also lets you specify your own `queryId` value for a Gremlin or SPARQL query, either in the HTTP header, or for a SPARQL query by using the `queryId` query hint. Assigning your own `queryId` makes it easy to keep track of a query so as to get status or cancel it.

Note

This feature is available starting with [Release 1.0.1.0.200463.0 \(2019-10-15\) \(p. 767\)](#).

Injecting a Custom `queryId` Value Using the HTTP Header

For both Gremlin and SPARQL, the HTTP header can be used to inject your own `queryId` value into a query.

Gremlin Example

```
curl -XPOST https://your-neptune-endpoint:port \  
-d "{\"gremlin\": \  
  \"g.V().limit(1).count()\" , \  
  \"queryId\": \"4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47\" }"
```

SPARQL Example

```
curl https://your-neptune-endpoint:port/sparql \  
-d "query=SELECT * WHERE { ?s ?p ?o } " \  
--data-urlencode \  
"queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

Injecting a Custom `queryId` Value Using a SPARQL Query Hint

Here is an example of how you would use the SPARQL `queryId` query hint to inject a custom `queryId` value into a SPARQL query:

```
curl https://your-neptune-endpoint:port/sparql \  
-d "PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#> \  
  SELECT * WHERE { hint:Query hint:queryId \"4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47\" \  
  {?s ?p ?o}}"
```

Using the `queryId` Value to Check Query Status

Gremlin Example

```
curl https://your-neptune-endpoint:port/gremlin/status \  
-d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

```
-d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

SPARQL Example

```
curl https://your-neptune-endpoint:port/sparql/status \  
-d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

Neptune Lab Mode

You can use Amazon Neptune *lab mode* to enable new features that are in the current Neptune engine release, but that aren't yet ready for production use and aren't enabled by default. This lets you try out these features in your development and test environments.

Note

This feature is available starting with [Release 1.0.1.0.200463.0 \(2019-10-15\) \(p. 767\)](#).

Using Neptune Lab Mode

Use the `neptune_lab_mode` DB cluster parameter (p. 409) to enable or disable features. You do this by including `(feature name)=enabled` or `(feature name)=disabled` in the value of the `neptune_lab_mode` parameter in the DB Cluster Parameter group.

For example, in this engine release you might set the `neptune_lab_mode` parameter to `Streams=disabled, ReadWriteConflictDetection=enabled`.

For information about how to edit the DB cluster parameter group for your database, see [Editing a Parameter Group \(p. 407\)](#). Note that you cannot edit the default DB cluster parameter group; if you are using the default group, you must create a new DB cluster parameter group before you can set the `neptune_lab_mode` parameter.

Important

At present, if you supply the wrong lab-mode parameters or your request fails for another reason, you may not be notified of the failure. You should always verify that a lab-mode change request has succeeded by calling the [status API \(p. 639\)](#) as shown below:

```
curl -G https://your-neptune-endpoint:port/status
```

The status results include lab-mode information which will show whether or not the changes you requested were made:

```
{
  "status": "healthy",
  "startTime": "Thu Jun 17 17:11:09 UTC 2021",
  "dbEngineVersion": "development",
  "role": "writer",
  "gremlin": {
    "version": "tinkerpop-3.4.10"
  },
  "sparql": {
    "version": "sparql-1.1"
  },
  "labMode": {
    "ObjectIndex": "disabled",
    "DFEQueryEngine": "viaQueryHint",
    "ReadWriteConflictDetection": "enabled"
  },
  "features": {
    "LookupCache": {
      "status": "Available"
    },
    "IAMAuthentication": "disabled",
    "Streams": "disabled",
    "AuditLog": "disabled"
  },
  "settings": {
    "clusterQueryTimeoutInMs": "120000"
  }
}
```

```
}
```

The following features are currently accessed using lab mode:

The Neptune DFE alternative query engine

Note

The DFE engine was first available in [Neptune engine release 1.0.3.0 \(p. 743\)](#), and starting in [Neptune engine release 1.0.5.0 \(p. 725\)](#), it became enabled by default for use with query hints, because `DFEQueryEngine=viaQueryHint` became the default lab-mode setting.

You can fully enable the new experimental [Neptune DFE engine \(p. 87\)](#) by setting `DFEQueryEngine=enabled` in the `neptune_lab_mode` DB cluster parameter, or you can disable it by setting `DFEQueryEngine=disabled`. The default setting is `DFEQueryEngine=viaQueryHint`, which means that the DFE engine is enabled, but will only be used for queries with the `useDFE` query hint present and set to `true`.

Note

When the DFE is enabled, [DFE data statistics \(p. 88\)](#) and openCypher support are also enabled.

The OSGP index

Neptune can now maintain a fourth index, namely the OSGP index, which is useful for data sets having a large number of predicates (see [Enabling an OSGP Index \(p. 62\)](#)).

Note

This feature is available starting in [Neptune engine release 1.0.2.1 \(p. 755\)](#).

You can enable an OSGP index in a new, empty Neptune DB cluster by setting `ObjectIndex=enabled` in the `neptune_lab_mode` DB cluster parameter. An OSGP index can only be enabled in a new, empty DB cluster.

By default, the OSGP index is disabled.

Warning

If you disable an enabled OSGP index by setting `ObjectIndex=disabled` after data has been added to the cluster, data will enter an inconsistent state. Rather than disable the index, move the data to a new DB cluster in which the OSGP index has not been enabled.

Formalized Transaction Semantics

Neptune has updated the formal semantics for concurrent transactions (see [Transaction Semantics in Neptune \(p. 66\)](#)).

Use `ReadWriteConflictDetection` as the name in the `neptune_lab_mode` parameter that enables or disables formalized transaction semantics.

By default, formalized transaction semantics are already enabled. If you want to revert to the earlier behavior, include `ReadWriteConflictDetection=disabled` in the value set for the DB Cluster `neptune_lab_mode` parameter.

The Amazon Neptune alternative query engine (DFE)

Amazon Neptune has a new, alternative query engine known as the DFE that uses DB instance resources such as CPU cores, memory, and I/O more efficiently than the current engine. It is currently available as a lab-mode feature, for development purposes only.

Note

The DFE engine was first available in [Neptune engine release 1.0.3.0 \(p. 743\)](#), and starting in [Neptune engine release 1.0.5.0 \(p. 725\)](#), it became enabled by default, but only for use with query hints and for openCypher support, because `DFEQueryEngine=viaQueryHint` became the default lab-mode setting. Support for openCypher in Neptune depends on the DFE engine being enabled.

The new DFE runs both SPARQL and Gremlin queries, and supports a wide variety of plan types, including left-deep, bushy, and hybrid ones. Plan operators can invoke both compute operations, which run on a reserved set of compute cores, and I/O operations, each of which runs on its own thread in an I/O thread pool.

The DFE uses pre-generated statistics about your Neptune graph data to make informed decisions about how to structure queries. See [DFE statistics \(p. 88\)](#) for information about how these statistics are generated and how you can manage them.

The choice of plan type and the number of compute threads used is made automatically based on pre-generated statistics and on the resources that are available in the Neptune head node. The order of results is not predetermined for plans that have internal compute parallelism.

Important

The experimental DFE engine is currently not tuned for use in a `t3.medium` instance type. As long as the engine is still in lab mode, please only enable it in larger instance sizes.

Enabling and disabling the Neptune DFE

You can enable or disable the DFE at any time in [Neptune Lab Mode \(p. 85\)](#) by setting `DFEQueryEngine` to `enabled`, `disabled`, or `viaQueryHint` in the `neptune_lab_mode` parameter in the DB cluster parameter group.

- Setting `DFEQueryEngine` to `enabled` enables the new query engine as well as the older engine, and causes the new engine to be used wherever possible, unless the `useDFE` query hint is present and set to `false`.
- Setting `DFEQueryEngine` to `disabled` enables only the older engine.
- Setting `DFEQueryEngine` to `viaQueryHint` (the default) enables both the new query engine and the older engine, but the newer engine is only used when the `useDFE` query hint is present and set to `true`.

You can also disable DFE for a specific query with the `useDFE` query hint (see [Gremlin useDFE query hint \(p. 282\)](#) for Gremlin, and [The useDFE SPARQL query hint \(p. 365\)](#) for SPARQL). This query hint lets you prevent the DFE from executing that specific query.

Important

The DFE is currently experimental. It is intended for use in development, and is **not** recommended for production use.

You can confirm whether or not the DFE is enabled using an [Instance Status \(p. 639\)](#) call, like this:


```
curl -G https://your-neptune-endpoint:port/status
```

The status response then specifies whether the DFE is enabled or not:

```
{
  "status": "healthy",
  "startTime": "Tue Nov 05 22:49:06 UTC 2019",
  "dbEngineVersion": "development",
  "role": "writer",
  "gremlin": {"version": "tinkerpop-3.4.1"},
  "sparql": {"version": "sparql-1.1"},
  "labMode": {
    "Streams": "disabled",
    "ReadWriteConflictDetection": "enabled",
    "DFEQueryEngine": "enabled"
  },
  "rollingBackTrxCount": "5",
  "rollingBackTrxEarliestStartTime": "Fri Jan 10 01:26:21 UTC 2020"
}
```

The Gremlin `explain` and `profile` results tell you whether a query is being executed by the DFE. See [Information contained in a Gremlin explain report \(p. 297\)](#) for `explain` and [DFE profile reports \(p. 307\)](#) for `profile`.

Similarly, SPARQL `explain` tells you whether a SPARQL query is being executed by the DFE. See [Example of SPARQL explain output when the DFE is enabled \(p. 378\)](#) and [DFENode operator \(p. 385\)](#) for more details.

Query constructs supported by the Neptune DFE

Currently, the Neptune DFE supports a subset of SPARQL and Gremlin query constructs.

For SPARQL, this is the subset of conjunctive [basic graph patterns](#).

For Gremlin, it is the generally subset of queries that contain a chain of traversals which do not contain some of the more complex steps.

You can find out whether one of your queries is being executed in whole or in part by the DFE as follows:

- In Gremlin, `explain` and `profile` results tell you what parts of a query are being executed by the DFE, if any. See [Information contained in a Gremlin explain report \(p. 297\)](#) for `explain` and [DFE profile reports \(p. 307\)](#) for `profile`. Also, see [Tuning Gremlin queries using explain and profile \(p. 322\)](#).

Details about Neptune engine support for individual Gremlin steps are documented in [Gremlin step support \(p. 336\)](#).

- Similarly, SPARQL `explain` tells you whether a SPARQL query is being executed by the DFE. See [Example of SPARQL explain output when the DFE is enabled \(p. 378\)](#) and [DFENode operator \(p. 385\)](#) for more details.

Managing statistics for the Neptune DFE to use

Note

The DFE engine was first available in [Neptune engine release 1.0.3.0 \(p. 743\)](#), and starting in [Neptune engine release 1.0.5.0 \(p. 725\)](#), it became enabled by default for use with query hints, because `DFEQueryEngine=viaQueryHint` became the default lab-mode setting. This means that statistics computation takes place by default unless you explicitly disable the DFE engine.

The experimental DFE query engine uses information about the data in your Neptune graph to make effective trade-offs when planning query execution. This information takes the form of statistics that include so-called characteristic sets and predicate statistics that can guide query planning.

By default, these DFE statistics are auto-generated when you enable DFE.

Currently, as long as auto-generation remains enabled, the DFE statistics are re-generated whenever either more than 10% of data in your graph has changed or when the latest statistics are more than 10 days old, but these triggers may change in the future.

You can manage the generation of DFE statistics through one of the following endpoints:

- `https://your-neptune-host:port/sparql/statistics` (for SPARQL).
- `https://your-neptune-host:port/propertygraph/statistics` (for Gremlin and openCypher), or aliased as `https://your-neptune-host:port/pg/statistics`.

Note

As of [engine release 1.1.0.0 \(p. 718\)](#), the Gremlin-specific statistics endpoint (`https://your-neptune-host:port/gremlin/statistics`) is being deprecated. It is still supported for backward compatibility but may be removed in future releases.

In the examples below, `$_STATISTICS_ENDPOINT` stands for either of these endpoint URLs.

Note

If a DFE statistics endpoint is on a reader instance, the only requests that it can process are [status requests \(p. 89\)](#). Other requests will fail with a `ReadOnlyViolationException`.

Size limits for DFE statistic generation

Currently, DFE statistics generation halts if either of the following size limits is reached:

- The number of characteristic sets generated may not exceed 50,000.
- The number of predicate statistics generated may not exceed one million.

These limits may change.

Current status of DFE statistics

You can check the current status of DFE statistics using the following `curl` request:

```
curl -G "$STATISTICS_ENDPOINT"
```

The response to a status request contains the following fields:

- `status` – the HTTP return code of the request. If the request succeeded, the code is 200. See [Common errors \(p. 94\)](#) for a list of common errors.
- `payload`:
 - `autoCompute` – (Boolean) Indicates whether or not automatic statistics generation is enabled.
 - `active` – (Boolean) Indicates whether or not DFE statistics generation is enabled at all.
 - `statisticsId` – Reports the ID of the current statistics generation run. A value of `-1` indicates that no statistics have been generated.
 - `date` – The UTC time at which DFE statistics have most recently been generated.
 - `note` – A note about problems in the case where statistics are invalid.
 - `summary` – Contains information about the characteristic sets generated in the statistics. These are generally not directly actionable:

- `signatureCount` – The total number of signatures across all characteristic sets.
- `instanceCount` – The total number of characteristic-set instances.
- `predicateCount` – The total number of unique predicates.

The response to a status request when no statistics have been generated looks like this:

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : false,
    "statisticsId" : -1
  }
}
```

If DFE statistics are available, the response looks like this:

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : true,
    "statisticsId" : 1588893232718,
    "date" : "2020-05-07T23:13Z",
    "summary" : {
      "signatureCount" : 5,
      "instanceCount" : 1000,
      "predicateCount" : 20
    }
  }
}
```

If the generation of DFE statistics failed, for example because it exceeded the [statistics size limitation \(p. 89\)](#), the response looks like this:

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : false,
    "statisticsId" : 1588713528304,
    "date" : "2020-05-05T21:18Z",
    "note" : "Limit reached: Statistics are not available"
  }
}
```

Disabling automatic generation of DFE statistics

By default, auto-generation of DFE statistics is enabled when you enable DFE.

You can disable auto-generation as follows:

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "disableAutoCompute" }'
```

If the request is successful, the HTTP response code is 200 and the response is:

```
{
```

```
"status" : "200 OK"
}
```

You can confirm that automatic generation is disabled by issuing a [status request \(p. 89\)](#) and checking that the `autoCompute` field in the response is set to `false`.

Disabling auto-generation of statistics does not terminate a statistics computation that is in progress.

If you make a request to disable auto-generation to a reader instance rather than the writer instance of your DB cluster, the request fails with an HTTP return code of 400 and output like the following:

```
{
  "detailedMessage" : "Writes are not permitted on a read replica instance",
  "code" : "ReadOnlyViolationException",
  "requestId" : "8eb8d3e5-0996-4a1b-616a-74e0ec32d5f7"
}
```

See [Common errors \(p. 94\)](#) for a list of other common errors.

Re-enabling automatic generation of DFE statistics

By default, auto-generation of DFE statistics is already enabled when you enable DFE. If you disable auto-generation, you can re-enable it later as follows:

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "enableAutoCompute" }'
```

If the request is successful, the HTTP response code is 200 and the response is:

```
{
  "status" : "200 OK"
}
```

You can confirm that automatic generation is enabled by issuing a [status request \(p. 89\)](#) and checking that the `autoCompute` field in the response is set to `true`.

Manually triggering the generation of DFE statistics

You can initiate DFE statistics generation manually as follows:

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "refresh" }'
```

If the request succeeds, the output is as follows, with an HTTP return code of 200:

```
{
  "status" : "200 OK",
  "payload" : {
    "statisticsId" : 1588893232718
  }
}
```

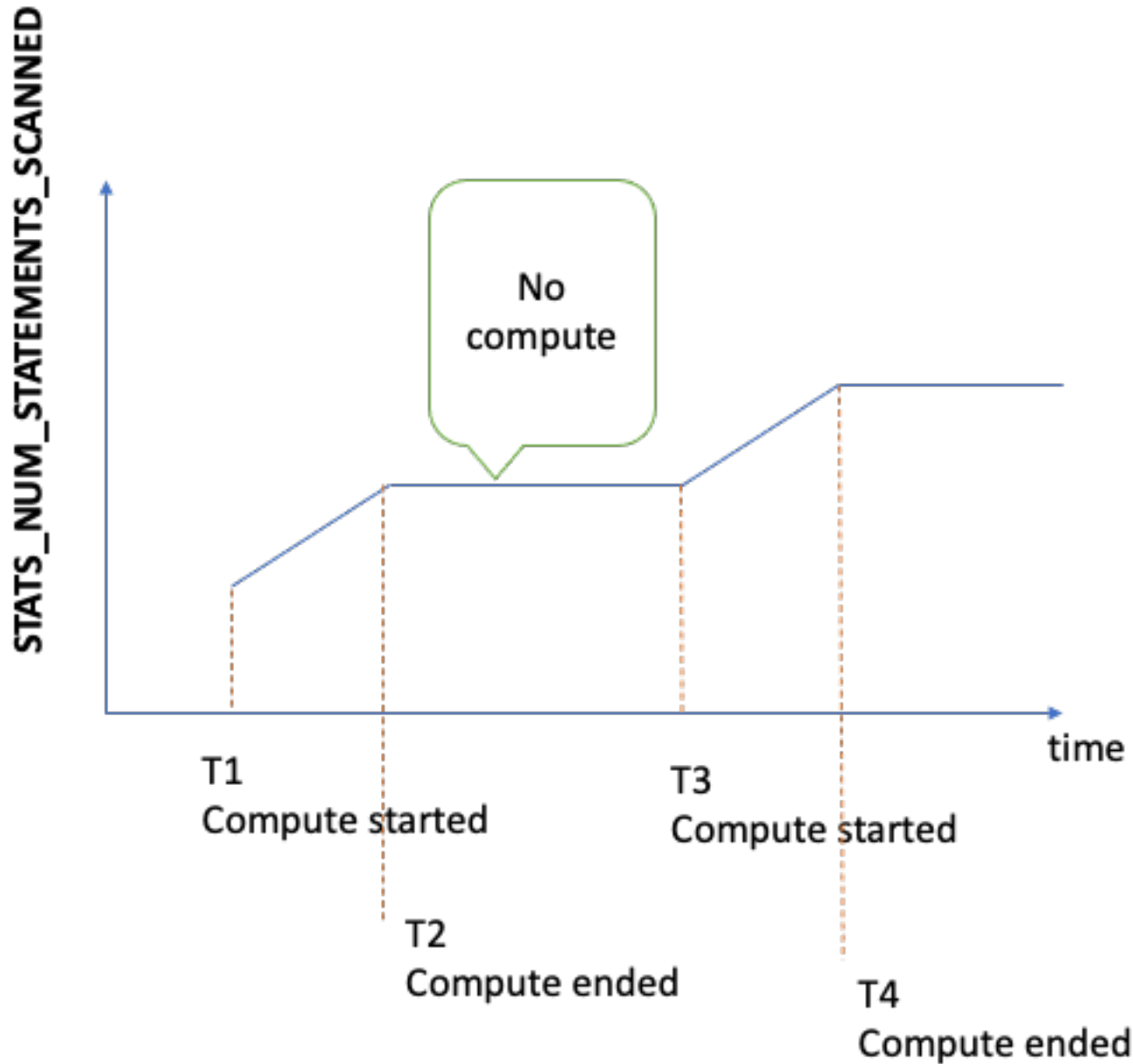
The `statisticsId` in the output is the ID of the statistics generation run that is currently occurring. If a run was already in process at the time of the request, the request returns the ID of that run rather than initiating a new one. Only one statistics generation run can occur at a time.

If a fail-over happens while DFE statistics are being generated, the new writer node will pick up the last processed checkpoint and resume the statistics run from there.

Using the StatsNumStatementsScanned CloudWatch metric to monitor statistics computation

The StatsNumStatementsScanned CloudWatch metric returns the total number of statements scanned for statistics computation since the server started. It is updated at each statistics computation slice.

Every time statistics computation is triggered, this number increases, and when no computation is happening, it remains constant. Looking at a plot of StatsNumStatementsScanned values over time therefore gives you a pretty clear picture of when statistics computation was happening and how fast:



When computation is happening, the slope of the graph shows you how fast (the steeper the slope, the faster statistics are being computed).

If the graph is simply a flat line at 0, the statistics feature has been enabled, but no statistics have been computed at all. If the statistics feature has been disabled, or if you're using an engine version that does not support statistics computation, the `StatsNumStatementsScanned` does not exist.

As mentioned earlier, you can disable statistics computation using the statistics API, but leaving it off can result in statistics not being up to date, which in turn can result in poor query plan generation for the DFE engine.

See [Monitoring Neptune Using Amazon CloudWatch \(p. 641\)](#) for information about how to use CloudWatch.

Using AWS Identity and Access Management (IAM) authentication with DFE statistics endpoints

You can access DFE statistics endpoints securely with IAM authentication by using `awscli` or any other tool that works with HTTP and IAM. After passing in your access key and secret key, you can then make DFE statistics requests that use IAM authentication.

For example, a status request:

```
awscli --service neptune-db "$STATISTICS_ENDPOINT"
```

Or, for example, you can create a JSON file named `request.json` that contains:

```
{ "mode" : "refresh" }
```

You can then manually initiate statistics generation like this:

```
awscli --service neptune-db "$STATISTICS_ENDPOINT" -X POST -d @request.json
```

Deleting DFE statistics

You can delete all statistics in the database by making an HTTP DELETE request to the statistics endpoint:

```
curl -X "DELETE" "$STATISTICS_ENDPOINT"
```

Valid HTTP return codes are:

- 200 – the delete was successful.

In this case, a typical response would look like:

```
{
  "status" : "200 OK",
  "payload" : {
    "active" : false,
    "statisticsId" : -1
  }
}
```

- 204 – there were no statistics to delete.

In this case, the response is blank (no response).

If you send a delete request to a statistics endpoint on a reader node, a `ReadOnlyViolationException` is thrown.

Common error codes for DFE statistics request

The following is a list of common errors that can occur when you make a request to a statistics endpoint:

- `AccessDeniedException` – *Return code: 400. Message: Missing Authentication Token.*
- `BadRequestException` (for Gremlin and openCypher) – *Return code: 400. Message: Bad route: /propertygraph/statistics.*
- `BadRequestException` (for SPARQL) – *Return code: 400. Message: Bad route: /sparql/statistics.*
- `InvalidParameterException` – *Return code: 400. Message: Statistics command parameter 'mode' has unsupported value 'the invalid value'.*
- `MissingParameterException` – *Return code: 400. Message: Content-type header not specified..*
- `ReadOnlyViolationException` – *Return code: 400. Message: Writes are not permitted on a read replica instance.*

For example, if you make a request when the DFE and statistics are not enabled, you would get a response like the following:

```
{
  "code" : "BadRequestException",
  "requestId" : "b2b8f8ee-18f1-e164-49ea-836381a3e174",
  "detailedMessage" : "Bad route: /sparql/statistics"
}
```

Amazon Neptune JDBC connectivity

Amazon Neptune has released an [open-source JDBC driver](#) that supports openCypher, Gremlin, SQL-Gremlin, and SPARQL queries. JDBC connectivity makes it easy to connect to Neptune with business intelligence (BI) tools such as Tableau. There is no additional cost to using the JDBC driver with Neptune — you still pay only for the Neptune resources that are consumed.

The driver is compatible with JDBC 4.2, and requires at least Java 8. See the [JDBC API documentation](#) for information about how to use a JDBC driver.

The GitHub project, where you can file issues and open feature requests, contains detailed documentation for the driver:

JDBC Driver for Amazon Neptune

- [Using SQL with the JDBC driver](#)
- [Using Gremlin with the JDBC Driver](#)
- [Using openCypher with the JDBC Driver](#)
- [Using SPARQL with the JDBC Driver](#)

Getting started with the Neptune JDBC driver

To use the Neptune JDBC driver to connect to a Neptune instance, either the JDBC driver must be deployed on an Amazon EC2 instance in the same VPC as your Neptune DB cluster, or the instance must be available through an SSH tunnel or load balancer. An SSH tunnel can be set up in the driver internally, or it can be set up externally.

You can download the driver [here](#). The driver comes packaged as a single JAR file with a name like `neptune-jdbc-1.0.0-all.jar`. To use it, place the JAR file in the `classpath` of your application. Or, if your application uses Maven or Gradle, you can use the appropriate Maven or Gradle commands to install the driver from the JAR.

The driver needs a JDBC connection URL to connect with Neptune, in a form like this:

```
jdbc:neptune:(connection type);//(host);property=value;property=value;...;property=value
```

The sections for each query language in the GitHub project describe the properties that you can set in the JDBC connection URL for that query language.

If the JAR file is in your application's `classpath`, no other configuration is necessary. You can connect the driver using the JDBC `DriverManager` interface and a Neptune connection string. For example, if your Neptune DB cluster is accessible through the endpoint `neptune-example.com` on port 8182, you would be able to connect with openCypher like this:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

void example() {
    String url = "jdbc:neptune:opencypher://bolt://neptune-example:8182";

    Connection connection = DriverManager.getConnection(url);
    Statement statement = connection.createStatement();
}
```



```
connection.close();  
}
```

The documentation sections in the GitHub project for each query language describe how to construct the connection string when using that query language.

Using Tableau with the Neptune JDBC driver

To use Tableau with the Neptune JDBC driver, start by downloading and installing the most recent version of [Tableau Desktop](#). Download the JAR file for the Neptune JDBC driver, and also the Neptune Tableau connector file (a `.taco` file).

To connect to Tableau for Neptune on a Mac

1. Place the Neptune JDBC driver JAR file in the `/Users/(your user name)/Library/Tableau/Drivers` folder.
2. Place the Neptune Tableau connector `.taco` file in the `/Users/(your user name)/Documents/My Tableau Repository/Connectors` folder.
3. If you have IAM authentication enabled, set up the environment for it. Note that environment variables set in `.zprofile/`, `.zshenv/`, `.bash_profile`, and so forth, will not work. The environment variables must be set so that they can be loaded by a GUI application.

One way to set your credentials is by placing your access key and secret key in the `/Users/(your user name)/.aws/credentials` file.

An easy way to set the service region is to open a terminal and enter the following command, using your application's region (for example, `us-east-1`):

```
launchctl setenv SERVICE_REGION region name
```

There are other ways to set environment variables that persist after a restart, but whatever technique you use must set variables that are accessible to a GUI application.

4. To get environment variables to load into a GUI on the Mac, enter this command on a terminal:

```
/Applications/Tableau/Desktop/2021.1.app/Contents/MacOS/Tableau
```

To connect to Tableau for Neptune on a Windows machine

1. Place the Neptune JDBC driver JAR file in the `C:\Program Files\Tableau\Drivers` folder.
2. Place the Neptune Tableau connector `.taco` file in the `C:\Users\(your user name)\Documents\My Tableau Repository\Connectors` folder.
3. If you have IAM authentication enabled, set up the environment for it.

This can be as simple as setting user `ACCESS_KEY`, `SECRET_KEY`, and `SERVICE_REGION` environment variables.

With Tableau open, select **More** on the left side of the window. If the Tableau connector file is properly located, you can select **Amazon Neptune by AWS** in the list that appears:

Amazon Neptune

Neptune Endpoint

database-1.cluster-cdffsmv2nzf7.us-east-

Port

8182

Use IAM Authentication

Require SSL

For support, contact Amazon. 

You should not have to edit the port, or add any connection options. Enter your Neptune endpoint and set your IAM and SSL configuration (you must enable SSL if you are using IAM).

When you select **Sign In**, it may take more than 30 seconds to connect if your graph is large. Tableau is collecting vertex and edge tables and join vertices on edges, as well as creating visualizations.

Troubleshooting a JDBC driver connection

If the driver fails to connect to the server, use the `isValid` function of the `JDBC Connection` object to check whether the connection is valid. If the function returns `false`, meaning that the connection is invalid, check that the endpoint being connected to is correct and that you are in the VPC of your Neptune DB cluster or that you have a valid SSH tunnel to the cluster.

If you get a `No suitable driver found for (connection string)` response from the `DriverManager.getConnection` call, there is likely an issue at the beginning of your connection string. Make sure that your connection string starts like this:

```
jdbc:neptune:opencypher://...
```

To gather more information about the connection, you can add a `LogLevel` to your connection string like this:

```
jdbc:neptune:opencypher://(JDBC URL):(port);logLevel=trace
```

Alternatively, you can add `properties.put("logLevel", "trace")` in your input properties to log trace information.

Amazon Neptune Engine Updates

Amazon Neptune releases engine updates regularly. You can determine which engine release version you currently have installed using the [instance-status API \(p. 639\)](#). Engine version numbers are explained in [the section below \(p. 100\)](#).

Engine releases are listed at [Engine Releases for Amazon Neptune \(p. 718\)](#), and patches are listed at [Latest Updates \(p. 3\)](#).

Before November 2019, Neptune only supported one engine version at a time, and engine version numbers all took the form, `1.0.1.0.200<xxx>`, where `xxx` was the patch number. New engine versions were all released as patches to earlier versions.

Starting in November 2019, Neptune supports multiple versions, allowing customers better control over their upgrade paths. As a result, engine release numbering changed.

Upgrading Your Neptune Engine

Note

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you experience downtime ranging from 20 or 30 seconds to several minutes, after which you can resume using the DB cluster.

Patch releases (`.R2`, `.R3`, etc.) are always installed automatically during your next maintenance window. This is independent of how the `AutoMinorVersionUpgrade` parameter is set.

If you want, you can choose to have minor engine updates also installed automatically to your entire cluster. You do this by setting the `AutoMinorVersionUpgrade` parameter of your primary writer instance to `true`. An automatic minor engine update will then be installed on every instance in your cluster during the maintenance window once the new engine version has proven stable, 2 to 3 weeks after the release.

- You can update your engine manually using the AWS Management Console as explained in [Using the Console to Upgrade Your Engine to a More Recent Version \(p. 101\)](#).
- You can also [use the CLI to update your engine manually \(p. 101\)](#).
- You can use the console to set `AutoMinorVersionUpgrade` to `true` in your primary writer instance. The result is that your cluster will always be automatically upgraded to the latest minor version during a maintenance window after a stabilization window of 2 to 3 weeks following the latest release.
- You can also [use the CLI to set `AutoMinorVersionUpgrade` \(p. 103\)](#) to `true` in your primary writer instance. Setting it to `true` in a reader instance has no effect.

If you are manually upgrading using the AWS CLI, be sure to specify the engine version to which you want to upgrade. If you do not, your engine may be upgraded to a version that is not the most recent one or the one you expect.

You can always determine what engine version your DB cluster is running by using the [Instance Status \(p. 639\)](#) API. If you are using the [CreateDBCluster \(p. 793\)](#) to create a cluster, the API returns the engine version used to create the cluster in its response.

The timing of automatic updates depends on the AWS region and maintenance window settings for your DB cluster. You can view or change your maintenance window settings on the [Neptune console](#). For more information, see [Neptune Maintenance Window \(p. 433\)](#).

Major engine releases can only be installed manually.

Note

Before performing a major upgrade, Neptune always creates a manual snapshot of your DB cluster, which lets you revert to the state of your cluster before the upgrade if you need to. This snapshot remains available until you delete it after you finish all post-upgrade testing.

Testing a new version of the Neptune engine before you upgrade

When a new major Neptune engine version is released, always test your Neptune applications on it first, before upgrading. If you do not have `AutoMinorVersionUpgrade` set, you may also want to do that for minor engine versions that introduce new features or behavior that could affect your code.

The best way to test a new version without upgrading your production DB cluster is to clone your cluster so that the clone is running the new engine version. You can then run queries on the clone without affecting your production DB cluster.

Neptune Engine Version Numbering

Neptune version numbers have 3 parts. Take version number `1.0.2.0.R2` as an example:

1. The first part (the first two numbers, namely the `1.0` in `1.0.2.0.R2`) is the database major version number.

This part only changes when a major incompatible change occurs, such as a change in the way data is stored that requires data migration when upgrading. Upgrading to a new major version often requires downtime proportional to the size of the cluster being upgraded, and can take much longer than other upgrades.

2. The second part (the third and fourth numbers, namely the `2.0` in `1.0.2.0.R2`) is the minor version number of the engine release.

You can choose to upgrade your engine to new minor versions automatically, as described below, or choose to decide for each minor version change.

3. The third part (the last number, namely the `R2` in `1.0.2.0.R2`) is the patch number for the minor version of the engine.

Patches involve urgent changes such as bug fixes or security changes that are deployed between minor engine releases. The first release of an engine version is implicitly `R0`, and patches to it are numbered `R1`, `R2`, and so forth.

Patches are always automatically applied during system maintenance windows, as described below.

You can list the available engine releases using the AWS CLI as follows:

For Linux, OS X, or Unix:

```
aws neptune describe-db-engine-versions \  
  --region <your-region> \  
  --engine neptune
```

For Windows:

```
aws neptune describe-db-engine-versions ^  
  --region <your-region> ^  
  --engine neptune
```

Available engine releases include only those releases that have a version number higher than the current one and for which an upgrade path is defined.

Using the Console to Upgrade Your Engine to a More Recent Version

You can perform a minor version upgrade of a Neptune DB cluster using the AWS Management Console as follows:

To upgrade the engine version of a DB cluster using the console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Clusters**, and then choose the DB cluster that you want to modify.
3. Choose **Actions**, and then choose **Modify cluster**. The **Modify DB cluster** page appears.
4. For **DB engine version**, choose the new version.
5. Choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, choose **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify Cluster** to save your changes.

Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

Using the Console to Enable Automatic Engine Updates

If you set the `AutoMinorVersionUpgrade` field to `true` in the writer instance of a DB cluster, Neptune will automatically update the engine of the entire cluster to the latest minor version after a stabilization window of 2 to 3 weeks.

You can use the AWS Management Console to set this field:

To set the `AutoMinorVersionUpgrade` for a DB cluster

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the list of DB instances, choose the primary instance (the writer instance) of the DB cluster.
3. Choose **Instance actions**, and then choose **Modify**. The **Modify DB Instance** page appears.
4. Choose **Actions**, and then choose **Modify instance**. The **Modify DB instance** page appears.
5. Set the auto minor version update field.

See [Modifying an Instance \(p. 444\)](#) for more information.

Manually performing a minor upgrade of the Neptune DB Engine using the CLI

You can manually perform a minor upgrade of the engine version of a DB cluster as follows, using the AWS CLI and the [ModifyDBCluster \(p. 804\)](#) API:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version (new-engine-version) \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version <new-engine-version> ^  
  --apply-immediately
```

Be sure to include the engine version that you want to upgrade to. If you do not, your engine may be upgraded to a version that is not the latest one.

Manually performing a major upgrade of the Neptune DB Engine using the CLI

To perform a major upgrade of the engine version of a DB cluster manually using the AWS CLI and the [ModifyDBCluster \(p. 804\)](#) API, you need to provide more parameters than for a minor upgrade:

It is important to include the engine version that you want to upgrade to. If you do not, your engine may be upgraded to a version that is not the latest one.

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine neptune \  
  --engine-version (new-engine-version) \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier <your-neptune-cluster> ^  
  --engine neptune ^  
  --engine-version <new-engine-version> ^  
  --apply-immediately
```

Instead of `--apply-immediately`, you can specify `--no-apply-immediately`.

If your cluster uses a custom cluster parameter group, be sure to include this parameter to specify it:

```
--db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

Similarly, if any instances in the cluster use a custom DB parameter group, be sure to include this parameter to specify it:

```
---db-instance-parameter-group-name (name of the custom instance parameter group)
```

Using the CLI to Enable Automatic Neptune Engine Updates

You can use the AWS CLI to enable or disable automatic engine updates for a DBcluster by setting the `AutoMinorVersionUpgrade` parameter of its primary instance.

To enable automatic engine upgrading for your DB cluster:

For Linux, OS X, or Unix:

```
aws neptune modify-db-instance \  
  --db-instance-identifier <your-cluster-write-instance> \  
  --auto-minor-version-upgrade \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-instance ^ \  
  --db-instance-identifier <your-cluster-write-instance> ^ \  
  --auto-minor-version-upgrade ^ \  
  --apply-immediately
```

Similarly, you can also disable automatic engine updating:

For Linux, OS X, or Unix:

```
aws neptune modify-db-instance \  
  --db-instance-identifier <your-cluster-write-instance> \  
  --no-auto-minor-version-upgrade \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-instance ^ \  
  --db-instance-identifier <your-cluster-write-instance> ^ \  
  --no-auto-minor-version-upgrade ^ \  
  --apply-immediately
```


Security in Amazon Neptune

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon Neptune, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Neptune. The following topics show you how to configure Neptune to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Neptune resources.

Topics

- [Data Protection in Amazon Neptune \(p. 104\)](#)
- [Identity and Access Management in Amazon Neptune \(p. 111\)](#)
- [Logging and Monitoring Amazon Neptune Resources \(p. 164\)](#)
- [Compliance Validation for Amazon Neptune \(p. 165\)](#)
- [Resilience in Amazon Neptune \(p. 165\)](#)

Data Protection in Amazon Neptune

The AWS [shared responsibility model](#) applies to data protection in Amazon Neptune. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the [AWS Security Blog](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Neptune or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

You use AWS published API calls to manage Neptune through the network. Clients must support Transport Layer Security (TLS) 1.2 or later using strong cipher suites, as described in [Encryption in Transit \(p. 108\)](#). Most modern systems such as Java 7 and later support these modes.

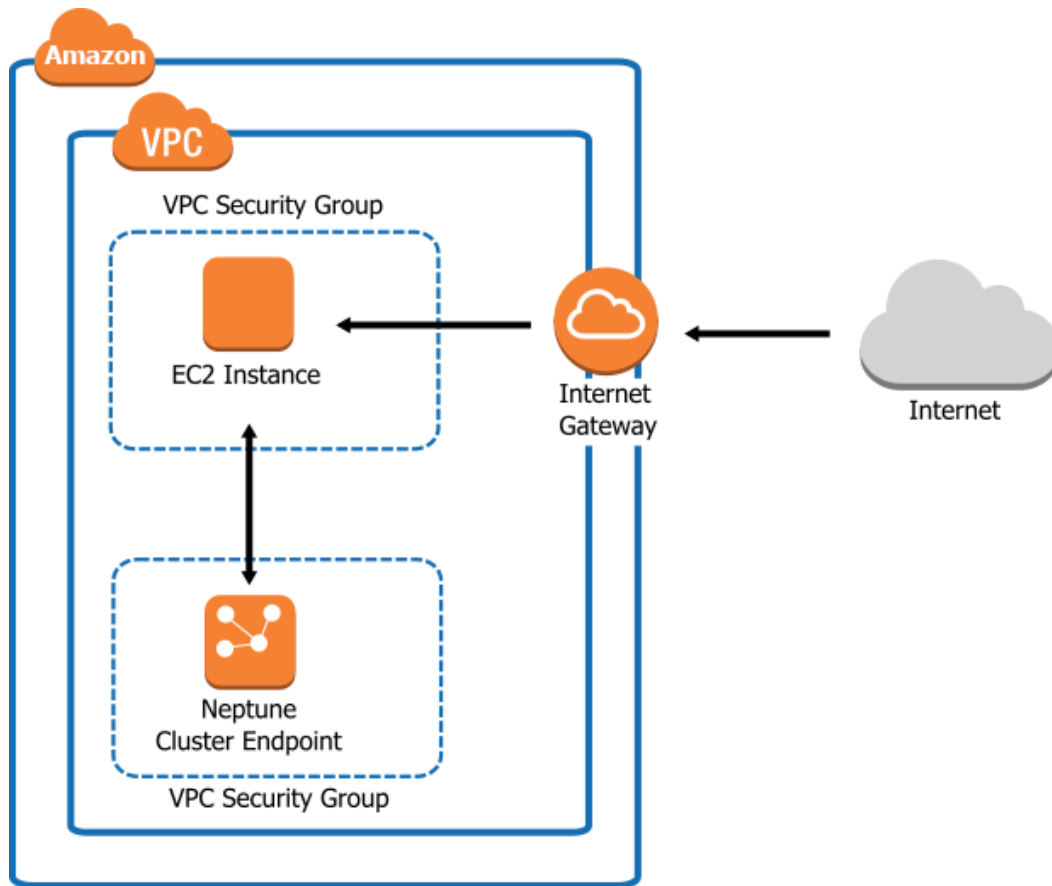
The following sections describe further how Neptune data is protected.

Topics

- [Accessing Neptune DB Clusters in an Amazon VPC \(p. 105\)](#)
- [Encryption in Transit: Connecting to Neptune Using SSL/HTTPS \(p. 108\)](#)
- [Encrypting Neptune Resources at Rest \(p. 109\)](#)

Accessing Neptune DB Clusters in an Amazon VPC

An Amazon Neptune DB cluster can *only* be created in an Amazon Virtual Private Cloud (Amazon VPC). Its endpoints are only accessible within that VPC, usually from an Amazon Elastic Compute Cloud (Amazon EC2) instance running in that VPC.



You need to create a VPC security group to allow the Amazon EC2 instance to connect to the Neptune DB instance from within the VPC. Access from the internet is allowed only to the EC2 instance, and access to the graph database is allowed only to the EC2 instance.

Note

You can communicate with a Neptune DB cluster in a VPC through an Amazon EC2 instance that is not in that VPC using Amazon EC2 [ClassicLink](#).

Setting Up a Neptune VPC

If you created your AWS account after December 4, 2013, you already have a default virtual private cloud (VPC) in each AWS Region. If you aren't sure whether you have a default VPC, see [Detecting Whether You Have a Default VPC](#) in the *Amazon VPC User Guide*.

There are many possible ways to configure a VPC or multiple VPCs. For more information about default VPCs, see [Default VPC and Default Subnets](#). For information about creating your own VPCs, see the [Amazon VPC User Guide](#).

An Amazon Neptune DB cluster can only be created in an Amazon VPC that has at least two subnets in at least two Availability Zones. By distributing your cluster instances across at least two Availability Zones, Neptune helps ensure that there are instances available in your DB cluster in the unlikely event of an Availability Zone failure. The cluster volume for your Neptune DB cluster always spans three Availability Zones to provide durable storage with less possibility of data loss.

If you're using the Amazon Neptune console to create your Neptune DB cluster, you can have Neptune automatically create a VPC for you. Or, you can use an existing VPC or create a new VPC for your

Neptune DB cluster. Your VPC must have at least two subnets in order for you to use it with an Amazon Neptune DB cluster.

If you have a default VPC, you can create a VPC security group to allow an Amazon EC2 instance to connect to the Neptune DB instance from within that VPC. Access from the internet is allowed only to the EC2 instance. The EC2 instance is allowed access to the graph database.

If you don't have a default VPC, and you haven't created one, you can have Neptune automatically create a VPC for you when you create a Neptune DB cluster using the console. Neptune can also create a VPC security group and a DB subnet group for you. Otherwise, you must do the following:

- Create a VPC with at least two subnets in at least two Availability Zones.
- Specify a VPC security group that authorizes connections to your Neptune DB cluster. You can do this in the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
- Specify a Neptune DB subnet group with at least two subnets with each subnet in a different Availability Zone. You can create a DB subnet group in the Neptune console at <https://console.aws.amazon.com/neptune/home>.

You must create a DB subnet group using the console.

For information about setting up a security group for a VPC, see [Creating a Security Group to Provide Access to a Neptune DB Instance in a VPC \(p. 107\)](#).

Creating a Security Group to Provide Access to a Neptune DB Instance in a VPC

Your Neptune DB instance is launched in a virtual private cloud (VPC). Security groups provide access to the Neptune DB instance in the VPC. They act as a firewall for the associated Neptune DB instance, controlling both inbound and outbound traffic at the instance level. Neptune DB instances are created by default with a firewall and a default security group that prevents access to the Neptune DB instance. You must add rules to a security group so that you can connect to your DB instance.

The security group that you need to create is a *VPC security group*. Neptune DB instances in a VPC require that you add rules to a VPC security group to allow access to the instance.

The following procedure shows you how to add a custom TCP rule that specifies the port range and IP addresses that the Amazon EC2 instance uses to access the database. You can use the VPC security group assigned to the EC2 instance rather than the IP address.

To create a VPC security group for Neptune on the console

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the upper-right corner of the console, choose the AWS Region in which you want to create the VPC security group and the Neptune DB instance. In the list of Amazon VPC resources for that Region, it should show that you have at least one VPC and several subnets. If it does not, you don't have a default VPC in that Region.
3. In the navigation pane, choose **Security Groups**.
4. Choose **Create Security Group**.
5. In the **Create Security Group** window, enter the **Name tag**, **Group name**, and **Description** of your security group. Choose the **VPC** that you want to create your Neptune DB instance in. Choose **Yes, Create**.
6. The VPC security group that you created should still be selected. The details pane at the bottom of the console window displays the details for the security group, and tabs for working with inbound and outbound rules. Choose the **Inbound Rules**

7. On the **Inbound Rules** tab, choose **Edit**. In the **Type** list, choose **Custom TCP Rule**.
8. In the **PortRange** text box, enter **8182**, the default port value for a Neptune DB instance. Then enter the IP address range (CIDR value) from where you will access the instance. Or, choose a security group name in the **Source** text box.
9. If you need to add more IP addresses or different port ranges, choose **Add another rule**.
10. When you finish, choose **Save**.

Use the VPC security group that you just created as the security group for your DB instance when you create it.

If you use a default VPC, a default subnet group spanning all of the VPC's subnets is already created for you. When you use the **Launch a Neptune DB instance** wizard to create a DB instance, you can choose the default VPC and use **default** for the **DB Subnet Group**.

After you complete the setup requirements, you can use your settings and the security group that you created to launch a Neptune DB instance.

Encryption in Transit: Connecting to Neptune Using SSL/HTTPS

Beginning with engine version 1.0.4.0, Amazon Neptune only allows Secure Sockets Layer (SSL) connections through HTTPS to any instance or cluster endpoint.

Neptune requires TLS version 1.2, using the following strong cipher suites:

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

Even where HTTP connections are allowed in earlier engine versions, any DB cluster that uses a new DB cluster parameter group is required to use SSL by default. *To protect your data, we recommend that you always connect to Neptune endpoints through SSL, using HTTPS instead of HTTP.*

Neptune automatically provides SSL certificates for your Neptune DB instances. You don't need to request any certificates. The certificates are provided when you create a new instance.

Neptune assigns a single wildcard SSL certificate to the instances in your account for each AWS Region. The certificate provides entries for the cluster endpoints, cluster read-only endpoints, and instance endpoints.

Certificate Details

The following entries are included in the provided certificate:

- Cluster endpoint — *.cluster-*a1b2c3d4wxyz.region*.neptune.amazonaws.com
- Read-only endpoint — *.cluster-ro-*a1b2c3d4wxyz.region*.neptune.amazonaws.com
- Instance endpoints — *.*a1b2c3d4wxyz.region*.neptune.amazonaws.com

Only the entries listed here are supported.

Proxy Connections

The certificates support only the hostnames that are listed in the previous section.

If you are using a load balancer or a proxy server (such as HAProxy), you must use SSL termination and have your own SSL certificate on the proxy server.

SSL passthrough doesn't work because the provided SSL certificates don't match the proxy server hostname.

Root CA Certificates

The certificates for Neptune instances are normally validated using the local trust store of the operating system or SDK (such as the Java SDK).

If you need to provide a root certificate manually, you can download the [Amazon Root CA certificate](#) in PEM format from the [Amazon Trust Services Policy Repository](#).

More Information

For more information about connecting to Neptune endpoints with SSL, see [the section called "Installing the Gremlin console" \(p. 263\)](#) and [the section called "HTTP REST" \(p. 348\)](#).

Encrypting Neptune Resources at Rest

Neptune encrypted instances provide an additional layer of data protection by helping to secure your data from unauthorized access to the underlying storage. You can use Neptune encryption to increase data protection of your applications that are deployed in the cloud. You can also use it to fulfill compliance requirements for data-at-rest encryption.

To manage the keys used for encrypting and decrypting your Neptune resources, you use [AWS Key Management Service \(AWS KMS\)](#). AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Using AWS KMS, you can create encryption keys and define the policies that control how these keys can be used. AWS KMS supports AWS CloudTrail, so you can audit key usage to verify that keys are being used appropriately. You can use your AWS KMS keys in combination with Neptune and supported AWS services such as Amazon Simple Storage Service (Amazon S3), Amazon Elastic Block Store (Amazon EBS), and Amazon Redshift. For a list of services that support AWS KMS, see [How AWS Services Use AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

All logs, backups, and snapshots are encrypted for a Neptune encrypted instance.

Enabling Encryption for a Neptune DB Instance

To enable encryption for a new Neptune DB instance, choose **Yes** in the **Enable encryption** section on the Neptune console. For information about creating a Neptune DB instance, see [Creating a New Neptune DB Cluster Using AWS CloudFormation or Manually](#) (p. 166).

When you create an encrypted Neptune DB instance, you can also supply the AWS KMS key identifier for your encryption key. If you don't specify an AWS KMS key identifier, Neptune uses your default Amazon RDS encryption key (`aws/rds`) for your new Neptune DB instance. AWS KMS creates your default encryption key for Neptune for your AWS account. Your AWS account has a different default encryption key for each AWS Region.

After you create an encrypted Neptune DB instance, you can't change the encryption key for that instance. So, be sure to determine your encryption key requirements before you create your encrypted Neptune DB instance.

You can use the Amazon Resource Name (ARN) of a key from another account to encrypt a Neptune DB instance. If you create a Neptune DB instance with the same AWS account that owns the AWS KMS encryption key that's used to encrypt that new Neptune DB instance, the AWS KMS key ID that you pass can be the AWS KMS key alias instead of the key's ARN.

Important

If Neptune loses access to the encryption key for a Neptune DB instance—for example, when Neptune access to a key is revoked—the encrypted DB instance is placed into a terminal state and can only be restored from a backup. We strongly recommend that you always enable backups for encrypted NeptuneDB instances to guard against the loss of encrypted data in your databases.

Key permissions needed when enabling encryption

The IAM user or role creating an encrypted Neptune DB instance must have at least the following permissions for the KMS key:

- "kms:Encrypt"
- "kms:Decrypt"
- "kms:GenerateDataKey"
- "kms:ReEncryptTo"
- "kms:GenerateDataKeyWithoutPlaintext"
- "kms:CreateGrant"
- "kms:ReEncryptFrom"
- "kms:DescribeKey"

Here is an example of a key policy that includes the necessary permissions:

```
{
  "Version": "2012-10-17",
  "Id": "key-consolepolicy-3",
  "Statement": [
    {
      "Sid": "Enable Permissions for root principal",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow use of the key for Neptune",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/NeptuneFullAccess"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:ReEncryptTo",
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:CreateGrant",
        "kms:ReEncryptFrom",
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "rds.us-east-1.amazonaws.com"
        }
      }
    }
  ]
}
```

```
{
  "Sid": "Deny use of the key for non Neptune",
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::123456789012:role/NeptuneFullAccess"
  },
  "Action": [
    "kms:*"
  ],
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "kms:ViaService": "rds.us-east-1.amazonaws.com"
    }
  }
}
```

- The first statement in this policy is optional. It gives access to the user's root principal.
- The second statement provides access to all the required AWS KMS APIs for this role, scoped down to the RDS Service Principal.
- The third statement tightens the security more by enforcing that this key is not usable by this role for any other AWS service.

You could also scope `createGrant` permissions down further by adding:

```
"Condition": {
  "Bool": {
    "kms:GrantIsForAWSResource": true
  }
}
```

Limitations of Neptune Encryption

The following limitations exist for encrypting Neptune clusters:

- You cannot convert an unencrypted DB cluster to an encrypted one.

However, you can restore an unencrypted DB cluster snapshot to an encrypted DB cluster. To do this, specify a KMS encryption key when you restore from the unencrypted DB cluster snapshot.
- You cannot convert an unencrypted DB instance to an encrypted one. You can only enable encryption for a DB instance when you create it.
- Also, DB instances that are encrypted can't be modified to disable encryption.
- You can't have an encrypted Read Replica of an unencrypted DB instance, or an unencrypted Read Replica of an encrypted DB instance.
- Encrypted Read Replicas must be encrypted with the same key as the source DB instance.

Identity and Access Management in Amazon Neptune

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and

authorized (have permissions) to use Neptune resources. IAM is an AWS service that you can use with no additional charge.

You can use AWS Identity and Access Management (IAM) to authenticate to your Neptune DB instance or DB cluster. When IAM database authentication is enabled, each request must be signed using AWS Signature Version 4.

AWS Signature Version 4 adds authentication information to AWS requests. For security, all requests to Neptune DB clusters with IAM authentication enabled must be signed with an access key. This key consists of an access key ID and secret access key. The authentication is managed externally using IAM policies.

Neptune authenticates on connection, and for WebSockets connections it verifies the permissions periodically to ensure that the user still has access.

Note

- Revoking, deleting, or rotating of credentials associated with the IAM user is not recommended because it does not terminate any connections that are already open.
- There are limits on the number of concurrent WebSocket connections per database instance, and on how long a connection can remain open. For more information, see [WebSockets Limits \(p. 709\)](#).

Contents

- [Identity and Access Management Overview \(p. 114\)](#)
 - [IAM Use Depends on Your Role \(p. 115\)](#)
 - [Authenticating with Identities \(p. 115\)](#)
 - [AWS account root user \(p. 115\)](#)
 - [IAM Users and Groups \(p. 115\)](#)
 - [IAM Roles \(p. 116\)](#)
- [Enabling IAM Database Authentication in Neptune \(p. 117\)](#)
- [Managing Access Using IAM Policies \(p. 117\)](#)
 - [Identity-Based Policies \(p. 117\)](#)
 - [Access Control Lists \(ACLs\) \(p. 118\)](#)
 - [Permissions Required to Use the Amazon Neptune Console \(p. 118\)](#)
 - [AWS Managed \(Predefined\) Policies for Amazon Neptune \(p. 118\)](#)
 - [Creating and using your own IAM policies for access to Neptune \(p. 119\)](#)
 - [Constructing a Resource ARN for a Cluster \(p. 119\)](#)
 - [More IAM Custom Policy Examples \(p. 120\)](#)
 - [Attaching an IAM Policy to an IAM User \(p. 122\)](#)
 - [IAM Policy Limitations \(p. 122\)](#)
- [Tag-based access control in Amazon Neptune \(p. 122\)](#)
 - [Example 1: Grant permission for actions on a resource using a custom tag that can take multiple values \(p. 123\)](#)
 - [Example 2: Limit the set of tag keys and values that can be used to tag a resource \(p. 124\)](#)
 - [Example 3: Allow full access to Neptune resources based on aws:ResourceTag \(p. 124\)](#)
 - [Actions that you can use in Neptune policies \(p. 125\)](#)
 - [AddRoleToDBCluster \(p. 125\)](#)
 - [AddSourceIdentifierToSubscription \(p. 125\)](#)
 - [AddTagsToResource \(p. 126\)](#)
 - [ApplyPendingMaintenanceAction \(p. 126\)](#)

- [CopyDBClusterParameterGroup](#) (p. 126)
- [CopyDBClusterSnapshot](#) (p. 126)
- [CopyDBParameterGroup](#) (p. 126)
- [CreateDBCluster](#) (p. 126)
- [CreateDBClusterParameterGroup](#) (p. 127)
- [CreateDBClusterSnapshot](#) (p. 127)
- [CreateDBInstance](#) (p. 127)
- [CreateDBParameterGroup](#) (p. 128)
- [CreateDBSubnetGroup](#) (p. 128)
- [CreateEventSubscription](#) (p. 128)
- [DeleteDBCluster](#) (p. 128)
- [DeleteDBClusterParameterGroup](#) (p. 129)
- [DeleteDBClusterSnapshot](#) (p. 129)
- [DeleteDBInstance](#) (p. 129)
- [DeleteDBParameterGroup](#) (p. 129)
- [DeleteDBSubnetGroup](#) (p. 129)
- [DeleteEventSubscription](#) (p. 129)
- [DescribeDBClusterParameterGroups](#) (p. 129)
- [DescribeDBClusterParameters](#) (p. 130)
- [DescribeDBClusterSnapshotAttributes](#) (p. 130)
- [DescribeDBClusterSnapshots](#) (p. 130)
- [DescribeDBClusters](#) (p. 130)
- [DescribeDBEngineVersions](#) (p. 130)
- [DescribeDBInstances](#) (p. 130)
- [DescribeDBParameterGroups](#) (p. 130)
- [DescribeDBParameters](#) (p. 131)
- [DescribeDBSubnetGroups](#) (p. 131)
- [DescribeEventCategories](#) (p. 131)
- [DescribeEventSubscriptions](#) (p. 131)
- [DescribeEvents](#) (p. 131)
- [DescribeOrderableDBInstanceOptions](#) (p. 131)
- [DescribePendingMaintenanceActions](#) (p. 131)
- [DescribeValidDBInstanceModifications](#) (p. 132)
- [FailoverDBCluster](#) (p. 132)
- [ListTagsForResource](#) (p. 132)
- [ModifyDBCluster](#) (p. 132)
- [ModifyDBClusterParameterGroup](#) (p. 132)
- [ModifyDBClusterSnapshotAttribute](#) (p. 133)
- [ModifyDBInstance](#) (p. 133)
- [ModifyDBParameterGroup](#) (p. 133)
- [ModifyDBSubnetGroup](#) (p. 133)
- [ModifyEventSubscription](#) (p. 133)
- [RebootDBInstance](#) (p. 133)
- [RemoveRoleFromDBCluster](#) (p. 133)
- [RemoveSourceIdentifierFromSubscription](#) (p. 134)

- [RemoveTagsFromResource](#) (p. 134)
- [ResetDBClusterParameterGroup](#) (p. 134)
- [ResetDBParameterGroup](#) (p. 134)
- [RestoreDBClusterFromSnapshot](#) (p. 134)
- [RestoreDBClusterToPointInTime](#) (p. 135)
- [StartDBCluster](#) (p. 135)
- [StopDBCluster](#) (p. 135)
- [IAM Resource Types Supported by Amazon Neptune](#) (p. 135)
- [Condition Keys for Amazon Neptune](#) (p. 137)
- [Using Service-Linked Roles for Neptune](#) (p. 138)
 - [Service-Linked Role Permissions for Neptune](#) (p. 138)
 - [Creating a Service-Linked Role for Neptune](#) (p. 139)
 - [Editing a Service-Linked Role for Neptune](#) (p. 139)
 - [Deleting a Service-Linked Role for Neptune](#) (p. 140)
 - [Cleaning Up a Service-Linked Role Before Deleting](#) (p. 140)
 - [Deleting All of Your Instances](#) (p. 140)
 - [Deleting All of Your Clusters](#) (p. 141)
- [IAM Authentication Using Temporary Credentials](#) (p. 141)
 - [Getting Temporary Credentials Using the AWS CLI](#) (p. 142)
 - [Setting Up AWS Lambda for Neptune IAM Authentication](#) (p. 144)
 - [Setting Up Amazon EC2 for Neptune IAM Authentication](#) (p. 145)
- [Connecting and Signing with AWS Signature Version 4](#) (p. 146)
 - [Prerequisites on Amazon Linux EC2](#) (p. 147)
 - [Connecting to Neptune Using the Gremlin Console with Signature Version 4 Signing](#) (p. 148)
 - [Using a version of TinkerPop earlier than 3.4.11 to connect to Neptune with Sig4 signing](#) (p. 148)
 - [Using TinkerPop 3.4.11 or higher to connect to Neptune with Sig4 signing](#) (p. 151)
 - [Connecting to Neptune Using Java and Gremlin with Signature Version 4 Signing](#) (p. 151)
 - [Using a version of TinkerPop earlier than 3.4.11 to connect to Neptune with Sig4 signing](#) (p. 151)
 - [Using TinkerPop 3.4.11 or higher to connect to Neptune with Sig4 signing](#) (p. 154)
 - [Connecting to Neptune Using Java and SPARQL with Signature Version 4 Signing \(RDF4J and Jena\)](#) (p. 154)
 - [Connecting to Neptune Using SPARQL and Node.js with Signature Version 4 Signing](#) (p. 156)
 - [Example: Connecting to Neptune Using Python with Signature Version 4 Signing](#) (p. 157)

Identity and Access Management Overview

Topics

- [IAM Use Depends on Your Role](#) (p. 115)
- [Authenticating with Identities](#) (p. 115)

IAM Use Depends on Your Role

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Neptune.

Service user – If you use the Neptune service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Neptune features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator.

Service administrator – If you're in charge of Neptune resources at your company, you probably have full access to Neptune. It's your job to determine which Neptune features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Neptune, see [Enabling IAM Database Authentication in Neptune \(p. 117\)](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Neptune. To view example Neptune identity-based policies that you can use in IAM, see [Creating and using your own IAM policies for access to Neptune \(p. 119\)](#).

Authenticating with Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see [Signing in to the AWS Management Console as an IAM user or root user](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [AWS Management Console](#), use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 signing process](#) in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing access keys for IAM users](#) in the *IAM*

User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated users and roles](#) in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
 - **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see in the *Service Authorization Reference*.
 - **Service role** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
 - **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the

EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

Enabling IAM Database Authentication in Neptune

By default, IAM database authentication is disabled when you create an Amazon Neptune DB cluster. You can enable IAM database authentication (or disable it again) using the AWS Management Console.

To create a new Neptune DB cluster with IAM authentication by using the console, follow the instructions for creating a Neptune DB cluster in [Launching a Neptune DB Cluster Using the Console \(p. 412\)](#).

On the second page of the creation process, for **Enable IAM DB Authentication**, choose **Yes**.

To enable or disable IAM authentication for an existing DB instance or cluster

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Clusters**.
3. Choose the Neptune DB cluster that you want to modify, and choose **Cluster actions**. Then choose **Modify cluster**.
4. In the **Database options** section, for **IAM DB Authentication**, choose either **Enable IAM DB authorization** or **No** (to disable). Then choose **Continue**.
5. To apply the changes immediately, choose **Apply immediately**.
6. Choose **Modify cluster**.

Managing Access Using IAM Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform,

on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Permissions Required to Use the Amazon Neptune Console

For a user to work with the Amazon Neptune console, that user must have a minimum set of permissions. These permissions allow the user to describe the Neptune resources for their AWS account and to provide other related information, including Amazon EC2 security and network information.

If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy. To ensure that those users can still use the Neptune console, also attach the `NeptuneReadOnlyAccess` managed policy to the user, as described in [AWS Managed \(Predefined\) Policies for Amazon Neptune \(p. 118\)](#).

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the Amazon Neptune API.

AWS Managed (Predefined) Policies for Amazon Neptune

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. Managed policies grant necessary permissions for common use cases so you can avoid having to investigate what permissions are needed. For more information, see [AWS Managed Policies](#) in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are for using Amazon Neptune management APIs:

- **NeptuneReadOnlyAccess** – Grants read-only access to all Amazon Neptune resources for the root AWS account.
- **NeptuneFullAccess** – Grants full access to all Amazon Neptune resources for the root AWS account. This is recommended if you need full Neptune access from the AWS CLI or SDK, but not AWS Management Console access.
- **NeptuneConsoleFullAccess** – Grants full access to all Amazon Neptune resources for the root AWS account and includes additional permissions to simplify Neptune access from the console, including limited IAM and EC2 (VPC) permissions.

The above policies only provide a user access to the Neptune management APIs, not to the data stored in a Neptune database. You can also create custom IAM policies that allow users to access the required Amazon Neptune API actions and resources, and you can attach these custom policies to the IAM users or groups that require those permissions.

Neptune IAM roles and policies grant some access to Amazon RDS resources, because Neptune shares operational technology with Amazon RDS for certain management features. This includes management API permissions. To restrict access to a specific set of actions, see [Access Control Overview](#) in the *Amazon RDS User Guide*.

Creating and using your own IAM policies for access to Neptune

To allow an IAM user to connect to your Amazon Neptune DB cluster, you must create an IAM policy. After that, you attach the policy to an IAM user or role. For information about attaching a policy to a role, see [Adding and Removing IAM Policies](#) in the *IAM User Guide*.

Note

The IAM policy, IAM user, and Neptune DB cluster must be in the same account. Cross-account access is not supported.

The following example policy allows an IAM user to connect to Neptune DB cluster using IAM database authentication.

Important

The `neptune-db` Amazon Resource Name (ARN) for the IAM authorization `Resource` is not the same as the ARN assigned to the cluster on creation. You must construct the ARN as shown in [Constructing a Resource ARN for a Cluster \(p. 119\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:*"
      ],
      "Resource": [
        "arn:aws:neptune-db:us-east-1:123456789012:cluster-ABCD1234EFGH5678IJKL90MNOP/*"
      ]
    }
  ]
}
```

Important

The `neptune-db:` prefix and the `neptune-db:*` action are only for IAM database authentication. They are not valid in any other context.

The preceding example includes a resource ARN in a format that is particular to Neptune IAM authentication. To construct the ARN, see [Constructing a Resource ARN for a Cluster \(p. 119\)](#). The ARN for the IAM authorization `Resource` is not the same as the ARN assigned to the cluster on creation. Construct the ARN as shown.

Constructing a Resource ARN for a Cluster

The resource ARN for a cluster in an IAM policy is part of a statement that has these elements:

- **Effect**—To grant access to the DB cluster, specify `Allow`. If you don't explicitly allow access, access is denied by default.
- **Action**—To allow connection to the DB cluster, specify `neptune-db:*`.
- **Resource**—The ARN that identifies a the DB cluster, in the following format:

```
arn:aws:neptune-db:region:account-id:cluster-resource-id/*
```


The ARN contains the following parts:

- *region* is the AWS Region for the Amazon Neptune DB cluster. In the example policy, the AWS Region is `us-east-1`.
- *account-id* is the AWS account number for the DB cluster. In the example policy, the account number is `123456789012`.
- *cluster-resource-id* is a resource id for the DB cluster. In the example policy, the identifier is `cluster-ABCD1234EFGH5678IJKL90MNOP`.

Important

The `cluster-resource-id` is different from the cluster identifier.

To find a cluster resource ID in the AWS Management Console for Amazon Neptune, choose the DB cluster that you want. The **Resource ID** is shown in the **Configuration** section.

More IAM Custom Policy Examples

The following example policies require that you construct a *resource ARN*. The example resource ARNs can use wildcards to include multiple resources.

Important

- Changes to an IAM policy take up to 10 minutes to apply to the specified Neptune resources.
- IAM policies that are applied to a Neptune DB cluster apply to all instances in that cluster.

Granting Access to All Clusters

The following policy uses the "*" character to match all of the DB clusters for a particular AWS account and Region.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:*"
      ],
      "Resource": [
        "arn:aws:neptune-db:us-east-1:123456789012:*/*"
      ]
    }
  ]
}
```

The IAM user has access to everything on the Neptune DB cluster. Neptune does not support fine-grained access control.

The resource ARN in the "Resource" list in the policy is not the same as a cluster ARN. It must be constructed according to the resource ARN format. For more information, see [Constructing a Resource ARN for a Cluster \(p. 119\)](#).

Denying Access to a Specific Cluster

The following policy denies access to a DB cluster for a particular AWS account and AWS Region.

The default IAM action is to deny access to a DB cluster unless an `Allow Effect` is granted.

To ensure that access is blocked, you can use the *Deny Effect*. The explicit *Deny Effect* takes precedent over any *Allow Effect*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "neptune-db:*"
      ],
      "Resource": [
        "arn:aws:neptune-db:us-east-1:123456789012:cluster-ABCD1234EFGH5678IJKL90MNOP/*"
      ]
    }
  ]
}
```

The IAM user is denied access to everything on the Neptune DB cluster. Neptune does not support fine-grained access control.

The resource ARN in the "Resource" list in the policy is not the same as a cluster ARN. It must be constructed according to the resource ARN format. For more information, see [Constructing a Resource ARN for a Cluster \(p. 119\)](#).

Denying Access to All Clusters

The following policy denies access to all DB clusters for a particular AWS account and Region.

The default IAM action is to deny access to a DB cluster unless an *Allow Effect* is granted.

To ensure that access is blocked, you can use the *Deny Effect*. The explicit *Deny Effect* takes precedent over any *Allow Effect*.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "neptune-db:*"
      ],
      "Resource": [
        "arn:aws:neptune-db:us-east-1:123456789012:*/*"
      ]
    }
  ]
}
```

The IAM user is denied access to everything on the Neptune DB cluster. Neptune does not support fine-grained access control.

The resource ARN in the "Resource" list in the policy is not the same as a cluster ARN. It must be constructed according to the resource ARN format. For more information, see [Constructing a Resource ARN for a Cluster \(p. 119\)](#).

Next, you attach the IAM policy to a user. For more information, see [Attaching an IAM Policy to an IAM User \(p. 122\)](#).

Attaching an IAM Policy to an IAM User

After you create an IAM policy to allow database authentication, you attach the policy to an IAM user. For a tutorial on this topic, see [Create and Attach Your First Customer Managed Policy](#) in the *IAM User Guide*.

As you work through the tutorial, you can use one of the policy examples shown in this section as a starting point and tailor it to your needs. At the end of the tutorial, you have an IAM user with an attached policy that can use the `neptune-db:*` action.

Important

- Changes to an IAM policy take up to 10 minutes to apply to the specified Neptune resources.
- IAM policies applied to a Neptune DB cluster apply to all instances in that cluster.

IAM Policy Limitations

Changes to an IAM policy take up to 10 minutes to apply to the specified Neptune resources.

IAM policies that are applied to a Neptune DB cluster apply to all instances in that cluster.

Neptune does not support the following:

- [AWS global and IAM condition context keys](#).
- Fine-grained access control.
- Actions other than `neptune-db:*`. A user must have a policy granting the `neptune-db:*` action to access an instance.
- Cross-account access.

Tag-based access control in Amazon Neptune

Amazon Neptune supports specifying conditions in an IAM policy using custom tags, to control access to Neptune through the [Neptune Management API \(p. 784\)](#).

For example, if you add a tag named `environment` to your DB instances, with values such as `beta`, `staging`, and `production`, you can then create a policy that restricts access to the instances based on the value of that tag.

Important

Tag-based access control only works for [IAM resources supported by Neptune \(p. 135\)](#), accessed using the Neptune management API. It does not work for access to resources containing `neptune-db:` or `neptune-db:connect` through data-plane actions.

If you manage access to your Neptune resources using tagging, be sure to secure access to the tags. You can restrict access to the tags by creating policies for the `AddTagsToResource` and `RemoveTagsFromResource` actions.

For example, you could use the following policy to deny users the ability to add or remove tags for all resources. Then, you could create policies to allow specific users to add or remove tags.

```
{ "Version": "2012-10-17",
  "Statement": [
    { "Sid": "DenyTagUpdates",
      "Effect": "Deny",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource": "*"
    }
  ]
}
```

```
    ]
  }
```

Tags can be attached to a Neptune resource such as a DB instance, or can be passed in the request. You use the `Condition` element of an IAM policy to control access to the resource that those tags are attached to (see [Condition](#) in the *IAM User Guide*), and also [Condition Operators](#) and [Tags as Policy Variables](#).

Note

Custom tag identifiers, names, and values of this sort are case-sensitive.

The syntax for custom tag conditions is as follows:

```
"Condition": {"StringEquals": {"rds:tag-identifier/tag-name": ["tag-value"] } }
```

For example, the following `Condition` element applies to DB instances with a tag named `environment` and a tag value of `production`:

```
"Condition": {"StringEquals": {"rds:db-tag/environment": ["production"] } }
```

Note

Because Neptune shares some functionality with Amazon RDS, the actions and condition keys in Neptune policies use an `rds:` prefix by design.

The following table lists Amazon RDS tag identifiers that you can use in a `Condition` element:

RDS Tag Identifier	Applies To
<code>cluster-tag</code>	DB clusters
<code>cluster-pg-tag</code>	DB cluster parameter groups
<code>cluster-snapshot-tag</code>	DB cluster snapshots
<code>db-tag</code>	DB instances, including read replicas
<code>es-tag</code>	Event subscriptions
<code>pg-tag</code>	DB parameter groups
<code>snapshot-tag</code>	DB snapshots
<code>subgrp-tag</code>	DB subnet groups

For a list of all the condition keys supported by Neptune, see [Condition Keys for Amazon Neptune \(p. 137\)](#)

Example 1: Grant permission for actions on a resource using a custom tag that can take multiple values

The policy below allows use of the `ModifyDBInstance`, `CreateDBInstance` or `DeleteDBInstance` API on any DB instance that has the `env` tag set to either `dev` or `test`:

```
{ "Version": "2012-10-17",
  "Statement": [
```

```
{
  "Sid": "AllowDevTestAccess",
  "Effect": "Allow",
  "Action": [
    "rds:ModifyDBInstance",
    "rds>CreateDBInstance",
    "rds>DeleteDBInstance"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "rds:db-tag/env": [
        "dev",
        "test"
      ],
      "rds:DatabaseEngine": "neptune"
    }
  }
}
```

Example 2: Limit the set of tag keys and values that can be used to tag a resource

This policy uses a Condition key to allow a tag that has the key `env` and a value of `test`, `qa`, or `dev` to be added to a resource:

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagAccessForDevResources",
      "Effect": "Allow",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "rds:req-tag/env": [
            "test",
            "qa",
            "dev"
          ],
          "rds:DatabaseEngine": "neptune"
        }
      }
    }
  ]
}
```

Example 3: Allow full access to Neptune resources based on `aws:ResourceTag`

The following policy is similar to the first example above, but uses the `aws:ResourceTag` instead:

```
{ "Version": "2012-10-17",
  "Statement": [
    {
```

```
"Sid": "AllowFullAccessToDev",
"Effect": "Allow",
"Action": [
  "rds:*"
],
"Resource": "*",
"Condition": {
  "StringEquals": {
    "aws:ResourceTag/env": "dev",
    "rds:DatabaseEngine": "neptune"
  }
}
]
```

The service-specific actions, resources, and condition keys that you can use in Neptune IAM policies are listed in the following sections.

Topics

- [Actions that you can use in Neptune policies \(p. 125\)](#)
- [IAM Resource Types Supported by Amazon Neptune \(p. 135\)](#)
- [Condition Keys for Amazon Neptune \(p. 137\)](#)

Actions that you can use in Neptune policies

You can specify the following actions in the `Action` element of an IAM policy statement. Use policies to grant permissions to perform an operation in AWS. When you use an action in a policy, you usually allow or deny access to the API operation or CLI command with the same name. However, in some cases, a single action controls access to more than one operation. Alternatively, some operations require several different actions.

The **Resource Types** column indicates whether each action supports resource-level permissions. If there is no value for this column, you must specify all resources ("`*`") in the `Resource` element of your policy statement. If the column includes a resource type, then you can specify an ARN of that type in a statement with that action. Required resources are indicated in the table with an asterisk (*). If you specify a resource-level permission ARN in a statement using this action, then it must be of this type. Some actions support multiple resource types. If the resource type is optional (not indicated as required), then you can choose to use one but not the other.

For details about the fields listed here, see [action table](#) in the [IAM User Guide](#).

AddRoleToDBCluster

[AddRoleToDBCluster \(p. 817\)](#) associates an IAM role with a Neptune DB cluster.

Access level: `Write`.

Dependent actions: `iam:PassRole`.

Resource type: `cluster` (p. 136) (required).

AddSourceIdentifierToSubscription

[AddSourceIdentifierToSubscription \(p. 921\)](#) adds a source identifier to an existing Neptune event notification subscription.

Access level: `Write`.

Resource type: [es \(p. 136\)](#) (required).

AddTagsToResource

[AddTagsToResource \(p. 930\)](#) associates an IAM role with a Neptune DB cluster.

Access level: `write`.

Resource types:

- [db \(p. 136\)](#)
- [es \(p. 136\)](#)
- [pg \(p. 136\)](#)
- [cluster-snapshot \(p. 136\)](#)
- [subgrp \(p. 136\)](#)

Condition Keys:

- `aws:RequestTag/tag-key` ([p. 137](#))
- `aws:TagKeys` ([p. 137](#))

ApplyPendingMaintenanceAction

[ApplyPendingMaintenanceAction \(p. 932\)](#) applies a pending maintenance action to a resource.

Access level: `write`.

Resource type: [db \(p. 136\)](#) (required).

CopyDBClusterParameterGroup

[CopyDBClusterParameterGroup \(p. 866\)](#) copies the specified DB cluster parameter group.

Access level: `write`.

Resource type: [cluster-pg \(p. 136\)](#) (required).

CopyDBClusterSnapshot

[CopyDBClusterSnapshot \(p. 893\)](#) copies a snapshot of a DB cluster.

Access level: `write`.

Resource type: [cluster-snapshot \(p. 136\)](#) (required).

CopyDBParameterGroup

[CopyDBParameterGroup \(p. 865\)](#) copies the specified DB parameter group.

Access level: `write`.

Resource type: [pg \(p. 136\)](#) (required).

CreateDBCluster

[CreateDBCluster \(p. 793\)](#) creates a new Neptune DB cluster.

Access level: Tagging.

Dependent actions: iam:PassRole.

Resource types:

- [cluster \(p. 136\)](#) (required).
- [cluster-pg \(p. 136\)](#) (required).
- [subgrp \(p. 136\)](#) (required).

Condition Keys:

- [aws:RequestTag/tag-key \(p. 137\)](#)
- [aws:TagKeys \(p. 137\)](#)
- [neptune-rds_DatabaseEngine \(p. 137\)](#)
- [neptune-rds_StorageEncrypted \(p. 137\)](#)

CreateDBClusterParameterGroup

[CreateDBClusterParameterGroup \(p. 869\)](#) creates a new DB cluster parameter group.

Access level: Tagging.

Resource type: [cluster-pg \(p. 136\)](#) (required).

Condition Keys:

- [aws:RequestTag/tag-key \(p. 137\)](#)
- [aws:TagKeys \(p. 137\)](#)

CreateDBClusterSnapshot

[CreateDBClusterSnapshot \(p. 889\)](#) creates a snapshot of a DB cluster.

Access level: Tagging.

Resource types:

- [cluster \(p. 136\)](#) (required).
- [cluster-snapshot \(p. 136\)](#) (required).

Condition Keys:

- [aws:RequestTag/tag-key \(p. 137\)](#)
- [aws:TagKeys \(p. 137\)](#)

CreateDBInstance

[CreateDBInstance \(p. 833\)](#) creates a new DB instance.

Access level: Tagging.

Dependent actions: iam:PassRole.

Resource types:

- [db](#) (p. 136) (required).
- [pg](#) (p. 136) (required).
- [subgrp](#) (p. 136) (required).

Condition Keys:

- [aws:RequestTag/tag-key](#) (p. 137)
- [aws:TagKeys](#) (p. 137)

CreateDBParameterGroup

[CreateDBParameterGroup](#) (p. 867) creates a new DB parameter group.

Access level: Tagging.

Resource type: [pg](#) (p. 136) (required).

Condition Keys:

- [aws:RequestTag/tag-key](#) (p. 137)
- [aws:TagKeys](#) (p. 137)

CreateDBSubnetGroup

[CreateDBSubnetGroup](#) (p. 882) creates a new DB subnet group.

Access level: Tagging.

Resource type: [subgrp](#) (p. 136) (required).

Condition Keys:

- [aws:RequestTag/tag-key](#) (p. 137)
- [aws:TagKeys](#) (p. 137)

CreateEventSubscription

[CreateEventSubscription](#) (p. 915) creates a Neptune event notification subscription.

Access level: Tagging.

Resource type: [es](#) (p. 136) (required).

Condition Keys:

- [aws:RequestTag/tag-key](#) (p. 137)
- [aws:TagKeys](#) (p. 137)

DeleteDBCluster

[DeleteDBCluster](#) (p. 799) deletes an existing Neptune DB cluster.

Access level: Write.

Resource types:

- [cluster](#) (p. 136) (required).
- [cluster-snapshot](#) (p. 136) (required).

DeleteDBClusterParameterGroup

[DeleteDBClusterParameterGroup](#) (p. 871) deletes a specified DB cluster parameter group.

Access level: write.

Resource type: [cluster-pg](#) (p. 136) (required).

DeleteDBClusterSnapshot

[DeleteDBClusterSnapshot](#) (p. 891) deletes a DB cluster snapshot.

Access level: write.

Resource type: [cluster-snapshot](#) (p. 136) (required).

DeleteDBInstance

[DeleteDBInstance](#) (p. 841) deletes a specified DB instance.

Access level: write.

Resource type: [db](#) (p. 136) (required).

DeleteDBParameterGroup

[DeleteDBParameterGroup](#) (p. 870) deletes a specified DBParameterGroup.

Access level: write.

Resource type: [pg](#) (p. 136) (required).

DeleteDBSubnetGroup

[DeleteDBSubnetGroup](#) (p. 884) deletes a DB subnet group.

Access level: write.

Resource type: [subgrp](#) (p. 136) (required).

DeleteEventSubscription

[DeleteEventSubscription](#) (p. 917) deletes an event notification subscription.

Access level: write.

Resource type: [es](#) (p. 136) (required).

DescribeDBClusterParameterGroups

[DescribeDBClusterParameterGroups](#) (p. 878) returns a list of DBClusterParameterGroup descriptions.

Access level: List.

Resource type: [cluster-pg \(p. 136\)](#) (required).

DescribeDBClusterParameters

[DescribeDBClusterParameters \(p. 877\)](#) returns the detailed parameter list for a particular DB cluster parameter group.

Access level: List.

Resource type: [cluster-pg \(p. 136\)](#) (required).

DescribeDBClusterSnapshotAttributes

[DescribeDBClusterSnapshotAttributes \(p. 912\)](#) returns a list of DB cluster snapshot attribute names and values for a manual DB cluster snapshot.

Access level: List.

Resource type: [cluster-snapshot \(p. 136\)](#) (required).

DescribeDBClusterSnapshots

[DescribeDBClusterSnapshots \(p. 910\)](#) returns information about DB cluster snapshots.

Access level: Read.

DescribeDBClusters

[DescribeDBClusters \(p. 826\)](#) returns information about a provisioned Neptune DB cluster.

Access level: List.

Resource type: [cluster \(p. 136\)](#) (required).

DescribeDBEngineVersions

[DescribeDBEngineVersions \(p. 934\)](#) returns a list of the available DB engines.

Access level: List.

Resource type: [pg \(p. 136\)](#) (required).

DescribeDBInstances

[DescribeDBInstances \(p. 859\)](#) returns information about DB instances.

Access level: List.

Resource type: [es \(p. 136\)](#) (required).

DescribeDBParameterGroups

[DescribeDBParameterGroups \(p. 876\)](#) returns a list of DBParameterGroup descriptions.

Access level: List.

Resource type: [pg \(p. 136\)](#) (required).

DescribeDBParameters

[DescribeDBParameters \(p. 875\)](#) returns a detailed parameter list for a particular DB parameter group.

Access level: List.

Resource type: [pg \(p. 136\)](#) (required).

DescribeDBSubnetGroups

[DescribeDBSubnetGroups \(p. 885\)](#) returns a list of DBSubnetGroup descriptions.

Access level: List.

Resource type: [subgrp \(p. 136\)](#) (required).

DescribeEventCategories

[DescribeEventCategories \(p. 925\)](#) returns a list of categories for all event source types, or, if specified, for a specified source type.

Access level: List.

DescribeEventSubscriptions

[DescribeEventSubscriptions \(p. 920\)](#) lists all the subscription descriptions for a customer account.

Access level: List.

Resource type: [es \(p. 136\)](#) (required).

DescribeEvents

[DescribeEvents \(p. 923\)](#) returns events related to DB instances, DB security groups, and DB parameter groups for the past 14 days.

Access level: List.

Resource type: [es \(p. 136\)](#) (required).

DescribeOrderableDBInstanceOptions

[DescribeOrderableDBInstanceOptions \(p. 860\)](#) returns a list of orderable DB instance options for the specified engine.

Access level: List.

DescribePendingMaintenanceActions

[DescribePendingMaintenanceActions \(p. 933\)](#) returns a list of resources (for example, DB instances) that have at least one pending maintenance action.

Access level: List.

Resource type: [db \(p. 136\)](#) (required).

DescribeValidDBInstanceModifications

[DescribeValidDBInstanceModifications \(p. 861\)](#) lists available modifications you can make to your DB instance.

Access level: List.

Resource type: [db \(p. 136\)](#) (required).

FailoverDBCluster

[FailoverDBCluster \(p. 818\)](#) forces a failover for a DB cluster.

Access level: Write.

Resource type: [cluster \(p. 136\)](#) (required).

ListTagsForResource

[ListTagsForResource \(p. 931\)](#) lists all tags on a Neptune resource.

Access level: Read.

Resource types:

- [cluster-snapshot \(p. 136\)](#)
- [db \(p. 136\)](#)
- [es \(p. 136\)](#)
- [pg \(p. 136\)](#)
- [subgrp \(p. 136\)](#)

ModifyDBCluster

[ModifyDBCluster \(p. 804\)](#)

Modifies a setting for a Neptune DB cluster.

Access level: Write.

Dependent actions: `iam:PassRole`.

Resource types:

- [cluster \(p. 136\)](#) (required).
- [cluster-pg \(p. 136\)](#) (required).

ModifyDBClusterParameterGroup

[ModifyDBClusterParameterGroup \(p. 872\)](#) modifies the parameters of a DB cluster parameter group.

Access level: Write.

Resource type: [cluster-pg \(p. 136\)](#) (required).

ModifyDBClusterSnapshotAttribute

[ModifyDBClusterSnapshotAttribute](#) (p. 896) adds an attribute and values to, or removes an attribute and values from, a manual DB cluster snapshot.

Access level: write.

Resource type: [cluster-snapshot](#) (p. 136) (required).

ModifyDBInstance

[ModifyDBInstance](#) (p. 846) modifies settings for a DB instance.

Access level: write.

Dependent actions: iam:PassRole.

Resource types:

- [db](#) (p. 136) (required).
- [pg](#) (p. 136) (required).

ModifyDBParameterGroup

[ModifyDBParameterGroup](#) (p. 871) modifies the parameters of a DB parameter group.

Access level: write.

Resource type: [pg](#) (p. 136) (required).

ModifyDBSubnetGroup

[ModifyDBSubnetGroup](#) (p. 884) modifies an existing DB subnet group.

Access level: write.

Resource type: [subgrp](#) (p. 136) (required).

ModifyEventSubscription

[ModifyEventSubscription](#) (p. 918) modifies an existing Neptune event notification subscription.

Access level: write.

Resource type: [es](#) (p. 136) (required).

RebootDBInstance

[RebootDBInstance](#) (p. 855) restarts the database engine service for the instance.

Access level: write.

Resource type: [db](#) (p. 136) (required).

RemoveRoleFromDBCluster

[RemoveRoleFromDBCluster](#) (p. 818) disassociates an AWS Identity and Access Management (IAM) role from an Amazon Neptune DB cluster.

Access level: write.

Dependent actions: iam:PassRole.

Resource type: [cluster \(p. 136\)](#) (required).

RemoveSourceIdentifierFromSubscription

[RemoveSourceIdentifierFromSubscription \(p. 922\)](#) removes a source identifier from an existing Neptune event notification subscription.

Access level: write.

Resource type: [es \(p. 136\)](#) (required).

RemoveTagsFromResource

[RemoveTagsFromResource \(p. 931\)](#) removes metadata tags from a Neptune resource.

Access level: Tagging.

Resource types:

- [cluster-snapshot \(p. 136\)](#)
- [db \(p. 136\)](#)
- [es \(p. 136\)](#)
- [pg \(p. 136\)](#)
- [subgrp \(p. 136\)](#)

Condition Keys:

- [aws:RequestTag/tag-key \(p. 137\)](#)
- [aws:TagKeys \(p. 137\)](#)

ResetDBClusterParameterGroup

[ResetDBClusterParameterGroup \(p. 874\)](#) modifies the parameters of a DB cluster parameter group to the default value.

Access level: write.

Resource type: [cluster-pg \(p. 136\)](#) (required).

ResetDBParameterGroup

[ResetDBParameterGroup \(p. 873\)](#) modifies the parameters of a DB parameter group to the engine/system default value.

Access level: write.

Resource type: [pg \(p. 136\)](#) (required).

RestoreDBClusterFromSnapshot

[RestoreDBClusterFromSnapshot \(p. 898\)](#) creates a new DB cluster from a DB cluster snapshot.

Access level: write.

Dependent actions: `iam:PassRole`.

Resource types:

- [cluster \(p. 136\)](#) (required).
- [cluster-snapshot \(p. 136\)](#) (required).

Condition Keys:

- `aws:RequestTag/tag-key` ([p. 137](#))
- `aws:TagKeys` ([p. 137](#))

RestoreDBClusterToPointInTime

`RestoreDBClusterToPointInTime` ([p. 904](#)) restores a DB cluster to an arbitrary point in time.

Access level: `write`.

Dependent actions: `iam:PassRole`.

Resource types:

- [cluster \(p. 136\)](#) (required).
- [subgrp \(p. 136\)](#) (required).

Condition Keys:

- `aws:RequestTag/tag-key` ([p. 137](#))
- `aws:TagKeys` ([p. 137](#))

StartDBCluster

`StartDBCluster` ([p. 810](#)) starts the specified DB cluster.

Access level: `write`.

Resource type: [cluster \(p. 136\)](#) (required).

StopDBCluster

`StopDBCluster` ([p. 813](#)) stops the specified DB cluster.

Access level: `write`.

Resource type: [cluster \(p. 136\)](#) (required).

IAM Resource Types Supported by Amazon Neptune

Neptune supports the resource types in the following table for use in the `Resource` element of IAM permission policy statements. For more information about the `Resource` element, see [IAM JSON Policy Elements: Resource](#).

Each action in the [Actions table \(p. 125\)](#) identifies the resource types that can be specified with that action. A resource type also determines which condition keys you can include in a policy, as specified in the last column of the table below.

The `ARN` column in the table specifies the Amazon Resource Name (ARN) format that you must use to reference resources of this type. The portions that are preceded by a `$` must be replaced by the actual values for your scenario. For example, if you see `$user-name` in an ARN, you must replace that string with either the actual IAM user's name or a policy variable that contains an IAM user's name. For more information about ARNs, see [IAM ARNs](#).

The `Condition Keys` column specifies condition context keys that you can include in an IAM policy statement only when both this resource and a compatible supporting action are included in the statement.

Resource Types	ARN	Condition Keys
cluster (a DB cluster)	<code>arn:partition:rds:region:account-id:cluster:instance-name</code>	<code>aws:ResourceTag/tag-key</code> (p. 137) <code>rds:cluster-tag/tag-key</code> (p. 137)
cluster-pg (a DB cluster parameter group)	<code>arn:partition:rds:region:account-id:cluster-pg:neptune-DBClusterParameterGroupName</code>	<code>aws:ResourceTag/tag-key</code> (p. 137)
cluster-snapshot (a DB cluster snapshot)	<code>arn:partition:rds:region:account-id:cluster-snapshot:neptune-DBClusterSnapshotName</code>	<code>aws:ResourceTag/tag-key</code> (p. 137) <code>rds:cluster-snapshot-tag/tag-key</code> (p. 137)
db (a DB instance)	<code>arn:partition:rds:region:account-id:db:neptune-DbInstanceName</code>	<code>aws:ResourceTag/tag-key</code> (p. 137) <code>rds:DatabaseClass</code> (p. 137) <code>rds:DatabaseEngine</code> (p. 137) <code>rds:StorageEncrypted</code> (p. 137) <code>rds:db-tag/tag-key</code> (p. 137)
es (an event subscription)	<code>arn:partition:rds:region:account-id:es:neptune-CustSubscriptionId</code>	<code>aws:ResourceTag/tag-key</code> (p. 137) <code>rds:es-tag/tag-key</code> (p. 138)
pg (a DB parameter group)	<code>arn:partition:rds:region:account-id:pg:neptune-ParameterGroupName</code>	<code>aws:ResourceTag/tag-key</code> (p. 137) <code>rds:pg-tag/tag-key</code> (p. 138)
subgrp (a DB subnet group)	<code>arn:partition:rds:region:account-id:subgrp:neptune-DBSubnetGroupName</code>	<code>aws:ResourceTag/tag-key</code> (p. 137)

Resource Types	ARN	Condition Keys
		rds:subgrp-tag/tag-key (p. 138)

Condition Keys for Amazon Neptune

The `Condition` element in a policy lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM Policy Elements: Variables and Tags](#) in the *IAM User Guide*.

Neptune defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS Global Condition Context Keys](#) in the *IAM User Guide*.

Amazon Neptune supports the condition keys in the following table. The `Type` column specifies the data type of the condition key.

The data type in the table's `Type` column determines which condition operators you can use to compare values in the request with the values in the policy statement. If you use an operator that is not compatible with the data type, then the match always fails and the policy statement never applies.

Condition Keys	Description	Type
aws:RequestTag/tag-key	Filters actions based on the presence of tag key-value pairs in the request.	String
aws:ResourceTag/tag-key	Filters actions based on tag key-value pairs attached to the resource.	String
aws:TagKeys	Filters actions based on the presence of tag keys in the request.	String
<code>rds:DatabaseClass</code>	A type of DB instance class.	String
<code>rds:DatabaseEngine</code>	The database engine, namely neptune.	String
<code>rds:StorageEncrypted</code>	A value that specifies whether the DB instance storage should be encrypted. To enforce storage encryption, specify true.	Boolean
<code>rds:cluster-tag/tag-key</code>	A tag attached to a DB cluster.	String
<code>rds:cluster-snapshot-tag/tag-key</code>	A tag attached to a DB cluster snapshot.	String
<code>rds:db-tag/tag-key</code>	A tag attached to a DB instance.	String

Condition Keys	Description	Type
<code>rds:es-tag/tag-key</code>	A tag attached to an event subscription.	String
<code>rds:pg-tag/tag-key</code>	A tag attached to a DB parameter group.	String
<code>rds:subgrp-tag/tag-key</code>	A tag attached to a DB subnet group.	String

Using Service-Linked Roles for Neptune

Amazon Neptune uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Neptune. Service-linked roles are predefined by Neptune and include all the permissions that the service requires to call other AWS services on your behalf.

Important

For certain management features, Amazon Neptune uses operational technology that is shared with Amazon RDS. This includes the *service-linked role* and management API permissions.

A service-linked role makes using Neptune easier because you don't have to manually add the necessary permissions. Neptune defines the permissions of its service-linked roles, and unless defined otherwise, only Neptune can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This protects your Neptune resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#), and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Neptune

Neptune uses the `AWSServiceRoleForRDS` service-linked role to allow Neptune and Amazon RDS to call AWS services on behalf of your database instances. The `AWSServiceRoleForRDS` service-linked role trusts the `rds.amazonaws.com` service to assume the role.

The role permissions policy allows Neptune to complete the following actions on the specified resources:

- Actions on ec2:
 - `AssignPrivateIpAddresses`
 - `AuthorizeSecurityGroupIngress`
 - `CreateNetworkInterface`
 - `CreateSecurityGroup`
 - `DeleteNetworkInterface`
 - `DeleteSecurityGroup`
 - `DescribeAvailabilityZones`
 - `DescribeInternetGateways`
 - `DescribeSecurityGroups`
 - `DescribeSubnets`
 - `DescribeVpcAttribute`

- DescribeVpcs
- ModifyNetworkInterfaceAttribute
- RevokeSecurityGroupIngress
- UnassignPrivateIpAddresses
- Actions on sns:
 - ListTopic
 - Publish
- Actions on cloudwatch:
 - PutMetricData
 - GetMetricData
 - CreateLogStream
 - PullLogEvents
 - DescribeLogStreams
 - CreateLogGroup

Note

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. You might encounter the following error message: Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.

If you see this message, make sure that you have the following permissions enabled:

```
{
  "Action": "iam:CreateServiceLinkedRole",
  "Effect": "Allow",
  "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
  "Condition": {
    "StringLike": {
      "iam:AWSServiceName": "rds.amazonaws.com"
    }
  }
}
```

For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Creating a Service-Linked Role for Neptune

You don't need to manually create a service-linked role. When you create an instance or a cluster, Neptune creates the service-linked role for you.

Important

To learn more, see [A New Role Appeared in My IAM Account](#) in the *IAM User Guide*.

If you delete this service-linked role and then need to create it again, you can use the same process to re-create the role in your account. When you create an instance or a cluster, Neptune creates the service-linked role for you again.

Editing a Service-Linked Role for Neptune

Neptune does not allow you to edit the `AWSServiceRoleForRDS` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a Service-Linked Role for Neptune

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must delete all of your instances and clusters before you can delete the associated service-linked role.

Cleaning Up a Service-Linked Role Before Deleting

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**. Then choose the name (not the check box) of the `AWSServiceRoleForRDS` role.
3. On the **Summary** page for the selected role, choose the **Access Advisor** tab.
4. On the **Access Advisor** tab, review recent activity for the service-linked role.

Note

If you are unsure whether Neptune is using the `AWSServiceRoleForRDS` role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the Regions where the role is being used. If the role is being used, then you must wait for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

If you want to remove the `AWSServiceRoleForRDS` role, you must first delete *all* of your instances and clusters.

Deleting All of Your Instances

Use one of these procedures to delete each of your instances.

To delete an instance (console)

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Instances**.
3. In the **Instances** list, choose the instance that you want to delete.
4. Choose **Instance actions**, and then choose **Delete**.
5. If you are prompted for **Create final Snapshot?**, choose **Yes** or **No**.
6. If you chose **Yes** in the previous step, for **Final snapshot name** enter the name of your final snapshot.
7. Choose **Delete**.

To delete an instance (AWS CLI)

See [delete-db-instance](#) in the *AWS CLI Command Reference*.

To delete an instance (API)

See [DeleteDBInstance](#).

Deleting All of Your Clusters

Use one of the following procedures to delete a single cluster, and then repeat the procedure for each of your clusters.

To delete a cluster (console)

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the **Clusters** list, choose the cluster that you want to delete.
3. Choose **Cluster Actions**, and then choose **Delete**.
4. Choose **Delete**.

To delete a cluster (CLI)

See `delete-db-cluster` in the *AWS CLI Command Reference*.

To delete a cluster (API)

See `DeleteDBCluster`

You can use the IAM console, the IAM CLI, or the IAM API to delete the `AWSServiceRoleForRDS` service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

IAM Authentication Using Temporary Credentials

Amazon Neptune supports IAM authentication using temporary credentials.

You can use an assumed role to authenticate using an IAM authentication policy, like one of the example policies in the previous sections.

If you are using temporary credentials, you must specify `AWS_SESSION_TOKEN` in addition to `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `SERVICE_REGION`.

Note

The temporary credentials expire after a specified interval, *including the session token*. You must update your session token when you request new credentials. For more information, see [Using Temporary Security Credentials to Request Access to AWS Resources](#).

The following sections describe how to allow access and retrieve temporary credentials.

To authenticate using temporary credentials

1. Create an IAM role with permission to access a Neptune cluster. For information about creating this role, see [the section called "Custom IAM Policies" \(p. 119\)](#).
2. Add a trust relationship to the role that allows access to the credentials.

Retrieve the temporary credentials, including the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`.

3. Connect to the Neptune cluster and sign the requests using the temporary credentials. For more information about connecting and signing requests, see [the section called "Connecting and Signing" \(p. 146\)](#).

There are various methods for retrieving temporary credentials depending on the environment.

Topics

- [Getting Temporary Credentials Using the AWS CLI \(p. 142\)](#)
- [Setting Up AWS Lambda for Neptune IAM Authentication \(p. 144\)](#)
- [Setting Up Amazon EC2 for Neptune IAM Authentication \(p. 145\)](#)

Getting Temporary Credentials Using the AWS CLI

To get credentials using the AWS Command Line Interface (AWS CLI), first you need to add a trust relationship that grants permission to assume the role to the AWS user that will run the AWS CLI command.

Add the following trust relationship to the Neptune IAM authentication role. If you don't have a Neptune IAM authentication role, see [the section called "Custom IAM Policies" \(p. 119\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/test"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For information about adding the trust relationship to the role, see [Editing the Trust Relationship for an Existing Role](#) in the *AWS Directory Service Administration Guide*.

If the Neptune policy is not yet attached to a role, create a new role. Attach the Neptune IAM authentication policy, and then add the trust policy. For information about creating a new role, see [Creating a New Role](#).

Note

The following sections assume that you have the AWS CLI installed.

To run the AWS CLI manually

1. Enter the following command to request the credentials using the AWS CLI. Replace the role ARN, session name, and profile with your own values.

```
aws sts assume-role --role-arn arn:aws:iam::123456789012:role/NeptuneIAMAuthRole --
role-session-name test --profile testprofile
```

2. The following is example output from the command. The `Credentials` section contains the values that you need.

Note

Record the `Expiration` value because you need to get new credentials after this time.

```
{
  "AssumedRoleUser": {
    "AssumedRoleId": "ARO3XFRBF535PLBIFPI4:s3-access-example",
    "Arn": "arn:aws:sts::123456789012:assumed-role/xaccounts3access/s3-access-
example"
  },
  "Credentials": {
    "AccessKeyId": "ASIA...",
    "Expiration": "2018-01-01T12:00:00Z",
    "SecretAccessKey": "wJalrXU3Wh...",
    "SessionToken": "FQWU..."
  }
}
```

```
"Credentials": {
  "SecretAccessKey": "9drTJvcXLB89EXAMPLELB8923FB892xMFI",
  "SessionToken": "AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/
qwjzP2iEXAMPLEbw/m3hsj8VBTkPORGvr9jM5sgP+w9IZWznU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYLFVgAUj
+7Indz3LU0aTWk1WKIjHmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nym
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFIPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8Bri2/
IcrxSpnWEXAMPLEXSDFTAQAM6DL9zR0tXoybnLrZIwMLMi1Kcgo5OytwU=",
  "Expiration": "2016-03-15T00:05:07Z",
  "AccessKeyId": "ASIAJEXAMPLEXEG2JICEA"
}
```

3. Set the environment variables using the returned credentials.

```
export AWS_ACCESS_KEY_ID=ASIAJEXAMPLEXEG2JICEA
export AWS_SECRET_ACCESS_KEY=9drTJvcXLB89EXAMPLELB8923FB892xMFI
export AWS_SESSION_TOKEN=AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/
qwjzP2iEXAMPLEbw/m3hsj8VBTkPORGvr9jM5sgP+w9IZWznU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYLFVgAUj
+7Indz3LU0aTWk1WKIjHmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nym
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFIPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8Bri2/
IcrxSpnWEXAMPLEXSDFTAQAM6DL9zR0tXoybnLrZIwMLMi1Kcgo5OytwU=

export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-central-1
or
eu-north-1 or eu-west-1 or eu-west-2 or eu-west-3 or eu-central-1
or me-south-1 or
ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-
southeast-2 or ap-south-1 or
cn-north-1 or cn-northwest-1 or
us-gov-east-1 or us-gov-west-1
```

4. Connect using one of the following methods.
 - the section called “Gremlin Console” (p. 148)
 - the section called “Gremlin Java” (p. 151)
 - the section called “SPARQL Java (RDF4J and Jena)” (p. 154)
 - the section called “Python Example” (p. 157)

To use a script to get the credentials

1. Run the following command to install the `jq` command. The script uses this command to parse the output of the AWS CLI command.

```
sudo yum -y install jq
```

2. Create a file named `credentials.sh` in a text editor and add the following text. Replace the service Region, role ARN, session name, and profile with your own values.

```
#!/bin/bash

creds_json=$(aws sts assume-role --role-arn arn:aws:iam::123456789012:role/
NeptuneIAMAuthRole --role-session-name test --profile testprofile)

export AWS_ACCESS_KEY_ID=$(echo "$creds_json" | jq .Credentials.AccessKeyId | tr -d '"')
export AWS_SECRET_ACCESS_KEY=$(echo "$creds_json" | jq .Credentials.SecretAccessKey | tr
-d '"')
export AWS_SESSION_TOKEN=$(echo "$creds_json" | jq .Credentials.SessionToken | tr -d '"')
```



```
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-central-1  
or eu-north-1 or eu-west-1 or eu-west-2 or eu-west-3 or eu-central-1  
or me-south-1 or ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-  
southeast-2 or ap-south-1 or cn-north-1 or cn-northwest-1 or  
us-gov-east-1 or us-gov-west-1
```

3. Connect using one of the following methods.
 - the section called “Gremlin Console” (p. 148)
 - the section called “Gremlin Java” (p. 151)
 - the section called “SPARQL Java (RDF4J and Jena)” (p. 154)
 - the section called “Python Example” (p. 157)

Setting Up AWS Lambda for Neptune IAM Authentication

AWS Lambda includes credentials automatically each time the Lambda function is run.

First you add a trust relationship that grants permission to assume the role to the Lambda service.

Add the following trust relationship to the Neptune IAM authentication role. If you don't have a Neptune IAM authentication role, see [the section called “Custom IAM Policies” \(p. 119\)](#).

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "",  
      "Effect": "Allow",  
      "Principal": {  
        "Service": "lambda.amazonaws.com"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```

For information about adding the trust relationship to the role, see [Editing the Trust Relationship for an Existing Role](#) in the *AWS Directory Service Administration Guide*.

If the Neptune policy is not yet attached to a role, create a new role. Attach the Neptune IAM authentication policy, and then add the trust policy. For information about creating a new role, see [Creating a New Role](#) in the *AWS Directory Service Administration Guide*.

To access Neptune from Lambda

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Create a new Lambda function for Python version 3.6.
3. Assign the `AWSLambdaVPCAccessExecutionRole` role to the Lambda function. This is required to access Neptune resources, which are VPC only.
4. Assign the Neptune authentication IAM role to the Lambda function.

For more information, see [AWS Lambda Permissions](#) in the *AWS Lambda Developer Guide*.

5. Copy the IAM authentication Python sample into the Lambda function code.

For more information about the sample and the sample code, see [the section called "Python Example" \(p. 157\)](#).

Setting Up Amazon EC2 for Neptune IAM Authentication

Amazon EC2 allows you to use instance profiles to automatically provide credentials. For more information, see [Using Instance Profiles](#) in the *IAM User Guide*.

First you add a trust relationship that grants permission to assume the role to the Amazon EC2 service.

Add the following trust relationship to the Neptune IAM authentication role. If you don't have a Neptune IAM authentication role, see [the section called "Custom IAM Policies" \(p. 119\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For information about adding the trust relationship to the role, see [Editing the Trust Relationship for an Existing Role](#) in the *AWS Directory Service Administration Guide*.

If the Neptune policy is not yet attached to a role, create a new role. Attach the Neptune IAM authentication policy, and then add the trust policy. For information about creating a new role, see [Creating a New Role](#) in the *AWS Directory Service Administration Guide*.

To use a script to get the credentials

1. Run the following command to install the `jq` command. The script uses this command to parse the output of the `curl` command.

```
sudo yum -y install jq
```

2. Create a file named `credentials.sh` in a text editor and add the following text. Replace the service Region with your own value.

```
role_name=$( curl -s http://169.254.169.254/latest/meta-data/iam/security-
credentials/ )
creds_json=$(curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/
${role_name})

export AWS_ACCESS_KEY_ID=$(echo "$creds_json" | jq .AccessKeyId |tr -d "'")
export AWS_SECRET_ACCESS_KEY=$(echo "$creds_json" | jq .SecretAccessKey| tr -d "'")
export AWS_SESSION_TOKEN=$(echo "$creds_json" | jq .Token|tr -d "'")

export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-central-1
or
eu-north-1 or eu-west-1 or eu-west-2 or eu-west-3 or eu-central-1
or me-south-1 or
```

```
ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-  
southeast-2 or ap-south-1 or  
cn-north-1 or cn-northwest-1 or  
us-gov-east-1 or us-gov-west-1
```

3. Run the script in the bash shell using the `source` command:

```
source credentials.sh
```

Even better is to add the commands in this script to the `.bashrc` file on your EC2 instance so that they will be invoked automatically when you log in, making temporary credentials available to the Gremlin console.

4. Connect using one of the following methods.
 - the section called “Gremlin Console” (p. 148)
 - the section called “Gremlin Java” (p. 151)
 - the section called “SPARQL Java (RDF4J and Jena)” (p. 154)
 - the section called “Python Example” (p. 157)

Connecting and Signing with AWS Signature Version 4

Amazon Neptune resources that have IAM DB authentication enabled require all HTTP requests to be signed using AWS Signature Version 4. For general information about signing requests with AWS Signature Version 4, see [Signature Version 4 Signing Process](#).

AWS Signature Version 4 is the process to add authentication information to AWS requests. For security, most requests to AWS must be signed with an access key, which consists of an access key ID and secret access key.

Note

If you are using temporary credentials, they expire after a specified interval, *including the session token*.

You must update your session token when you request new credentials. For more information, see [Using Temporary Security Credentials to Request Access to AWS Resources](#).

Important

Accessing Neptune with IAM-based authentication requires that you create HTTP requests and sign the requests yourself.

How Signature Version 4 Works

1. You create a canonical request.
2. You use the canonical request and some other information to create a string-to-sign.
3. You use your AWS secret access key to derive a signing key, and then use that signing key and the string-to-sign to create a signature.
4. You add the resulting signature to the HTTP request in a header or as a query string parameter.

When Neptune receives the request, it performs the same steps that you did to calculate the signature. Neptune then compares the calculated signature to the one you sent with the request. If the signatures match, the request is processed. If the signatures don't match, the request is denied.

For general information about signing requests with AWS Signature Version 4, see [Signature Version 4 Signing Process](#) in the *AWS General Reference*.

The following sections contain examples that show how to send signed requests to the Gremlin and SPARQL endpoints of a Neptune DB instance with IAM authentication enabled.

Topics

- [Prerequisites on Amazon Linux EC2 \(p. 147\)](#)
- [Connecting to Neptune Using the Gremlin Console with Signature Version 4 Signing \(p. 148\)](#)
- [Connecting to Neptune Using Java and Gremlin with Signature Version 4 Signing \(p. 151\)](#)
- [Connecting to Neptune Using Java and SPARQL with Signature Version 4 Signing \(RDF4J and Jena\) \(p. 154\)](#)
- [Connecting to Neptune Using SPARQL and Node.js with Signature Version 4 Signing \(p. 156\)](#)
- [Example: Connecting to Neptune Using Python with Signature Version 4 Signing \(p. 157\)](#)

Prerequisites on Amazon Linux EC2

The following are instructions for installing Apache Maven and Java 8 on an Amazon EC2 instance. These are required for the Amazon Neptune Signature Version 4 authentication samples.

To Install Apache Maven and Java 8 on your EC2 instance

1. Connect to your Amazon EC2 instance with an SSH client.
2. Install Apache Maven on your EC2 instance. First, enter the following to add a repository with a Maven package.

```
sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
```

Enter the following to set the version number for the packages.

```
sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
```

Then you can use **yum** to install Maven.

```
sudo yum install -y apache-maven
```

3. The Gremlin libraries require Java 8. Enter the following to install Java 8 on your EC2 instance.

```
sudo yum install java-1.8.0-devel
```

4. Enter the following to set Java 8 as the default runtime on your EC2 instance.

```
sudo /usr/sbin/alternatives --config java
```

When prompted, enter the number for Java 8.

5. Enter the following to set Java 8 as the default compiler on your EC2 instance.

```
sudo /usr/sbin/alternatives --config javac
```

When prompted, enter the number for Java 8.

Connecting to Neptune Using the Gremlin Console with Signature Version 4 Signing

Using a version of TinkerPop earlier than 3.4.11 to connect to Neptune with Sig4 signing

This section shows how to connect to Amazon Neptune using the Gremlin Console with Signature Version 4 authentication when using a version of TinkerPop prior to 3.4.11. Starting with TinkerPop 3.4.11, which Neptune supported as of [engine release 1.0.5.0 \(p. 725\)](#), you can add a callback in the TinkerPop driver to intercept the WebSocket handshake, and then use the standard Sig4 library to do the signing. See [the example showing how to do that \(p. 151\)](#).

Note

There are limits on the number of concurrent WebSocket connections per database instance, and on how long a connection can remain open. For more information, see [WebSockets Limits \(p. 709\)](#).

Prerequisites

- Java 8 or higher.
- Apache Maven 3.3 or higher.

For information about installing these prerequisites on an EC2 instance running Amazon Linux, see [Prerequisites on Amazon Linux EC2 \(p. 147\)](#).

- IAM credentials to sign the requests. For more information, see [Using the Default Credential Provider Chain](#) in the *AWS SDK for Java Developer Guide*.

Note

If you are using temporary credentials, they expire after a specified interval, *including the session token*.

You must update your session token when you request new credentials. For more information, see [Using Temporary Security Credentials to Request Access to AWS Resources](#) in the *IAM User Guide*.

- Set the `SERVICE_REGION` variable to one of the following, indicating the Region of your Neptune DB instance:
 - US East (N. Virginia): `us-east-1`
 - US East (Ohio): `us-east-2`
 - US West (N. California): `us-west-1`
 - US West (Oregon): `us-west-2`
 - Canada (Central): `ca-central-1`
 - South America (São Paulo): `sa-east-1`
 - Europe (Stockholm): `eu-north-1`
 - Europe (Ireland): `eu-west-1`
 - Europe (London): `eu-west-2`
 - Europe (Paris): `eu-west-3`
 - Europe (Frankfurt): `eu-central-1`
 - Middle East (Bahrain): `me-south-1`
 - Asia Pacific (Hong Kong): `ap-east-1`
 - Asia Pacific (Tokyo): `ap-northeast-1`
 - Asia Pacific (Seoul): `ap-northeast-2`
 - Asia Pacific (Singapore): `ap-southeast-1`
 - Asia Pacific (Sydney): `ap-southeast-2`

- Asia Pacific (Mumbai): `ap-south-1`
- China (Beijing): `cn-north-1`
- China (Ningxia): `cn-northwest-1`
- AWS GovCloud (US-West): `us-gov-west-1`
- AWS GovCloud (US-East): `us-gov-east-1`

To connect to Neptune using the Gremlin Console with Signature Version 4 signing and a TinkerPop version earlier than 3.4.11

1. Download [Gremlin Console](#) (version 3.4.8 to 3.4.10) from the [Apache Tinkerpop3](#) website to your EC2 instance.
2. Unzip the Gremlin Console zip file.

```
unzip apache-tinkerpop-gremlin-console-3.4.10-bin.zip
```

3. Clone the sample repository from GitHub.

```
git clone https://github.com/aws/amazon-neptune-gremlin-java-sigv4.git
```

4. Change directory into `amazon-neptune-gremlin-java-sigv4`.

```
cd amazon-neptune-gremlin-java-sigv4
```

5. Get the version of the sigv4 project by checking out the branch appropriate to your TinkerPop version. That would be 2.1.1 for TinkerPop client versions earlier than 3.4.9:

```
git checkout $(git describe --tags `git checkout v2.1.1`)
```

For TinkerPop version 3.4.9 and above, get version 2.1.2:

```
git checkout $(git describe --tags `git checkout v2.1.2`)
```

6. Build, package, and get the dependencies of the project.

```
mvn package dependency:copy-dependencies
```

7. Copy the package jar and all dependency jars to the Gremlin Console library directory.

```
cp target/{*.jar,dependency/*.jar} ../apache-tinkerpop-gremlin-console-3.4.8/lib
```

8. Remove conflicting dependency jars in the Gremlin Console library directory.

```
rm ../apache-tinkerpop-gremlin-console-3.4.8/lib/netty-all-4.1.22.Final.jar
```

9. Change directories into the unzipped Gremlin Console directory.

```
cd ../apache-tinkerpop-gremlin-console-3.4.8
```

10. Download the CA certificate. Gremlin Console requires a certificate to verify the remote certificate.

```
wget https://www.amazontrust.com/repository/SFSRootCAG2.pem
```

11. In the `conf` subdirectory of the extracted directory, create a file named `neptune-remote.yaml` with the following text. Replace `your-neptune-endpoint` with the hostname or IP address of your Neptune DB instance. The square brackets (`[]`) are required.

Note

For information about finding the hostname of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

```
hosts: [your-neptune-endpoint]
port: 8182
connectionPool: {
  channelizer: org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer,
  enableSsl: true,
  trustCertChainFile: "SFSRootCAG2.pem"
}
serializer: { className:
  org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0, config:
  { serializeResultToString: true }}
```

12. **Important**
You must provide IAM credentials to sign the requests. Enter the following commands to set your credentials as environment variables, replacing the relevant items with your credentials.

```
export AWS_ACCESS_KEY_ID=access_key_id
export AWS_SECRET_ACCESS_KEY=secret_access_key
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                        eu-north-1 or eu-west-1 or eu-west-2 or eu-west-3 or eu-
central-1 or me-south-1 or
                        ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-
southeast-2 or ap-south-1 or
                        cn-north-1 or cn-northwest-1 or
                        us-gov-east-1 or us-gov-west-1
```

The Neptune Version 4 Signer uses the default credential provider chain. For additional methods of providing credentials, see [Using the Default Credential Provider Chain](#) in the *AWS SDK for Java Developer Guide*.

The `SERVICE_REGION` variable is required, even when using a credentials file.

13. In a terminal, navigate to the Gremlin Console directory (`apache-tinkerpop-gremlin-console-3.4.8`). Then enter the following command to run the Gremlin Console.

```
bin/gremlin.sh
```

You should see the following output:

```
  \, , /
   (o o)
-----o00o-(3)-o00o-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

You are now at the `gremlin>` prompt. You enter the remaining steps at this prompt.

14. At the `gremlin>` prompt, enter the following to connect to the Neptune DB instance.

```
:remote connect tinkerpop.server conf/neptune-remote.yaml
```

15. At the `gremlin>` prompt, enter the following to switch to remote mode. This sends all Gremlin queries to the remote connection.

```
:remote console
```

Using TinkerPop 3.4.11 or higher to connect to Neptune with Sig4 signing

Here is an example of how to connect to Neptune with Sig4 signing in the Gremlin console when using TinkerPop 3.4.11 or higher:

```
:install com.amazonaws amazon-neptune-sigv4-signer 2.4.0
:install com.amazonaws aws-java-sdk-core 1.12.9

// The lines above only need to be executed once to get the dependencies into the console

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain
import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer
System.setProperty("aws.accessKeyId", "your-access-key")
System.setProperty("aws.secretKey", "your-secret-key")
cluster = Cluster.build("your-cluster").
    enableSsl(true).keyCertChainFile("SFSRootCAG2.pem").
    handshakeInterceptor{ r ->
        def sigV4Signer = new NeptuneNettyHttpSigV4Signer("Amazon region", new
        DefaultAWSCredentialsProviderChain())
        sigV4Signer.signRequest(r)
        return r
    }.create()
client = cluster.connect()
client.submit("g.V().has('code', 'IAD')").all().get()
```

Connecting to Neptune Using Java and Gremlin with Signature Version 4 Signing

Using a version of TinkerPop earlier than 3.4.11 to connect to Neptune with Sig4 signing

This section shows how to connect to Amazon Neptune using the Gremlin Java API with Signature Version 4 authentication when using a version of TinkerPop earlier than 3.4.11.

Note that if you're using TinkerPop 3.4.11 or higher, supported by Neptune starting with [engine release 1.0.5.0 \(p. 725\)](#), you can use the TinkerPop driver directly to connect with Sig4 signing. See [the example showing how to do that \(p. 154\)](#).

Note

There are limits on the number of concurrent WebSocket connections per database instance, and on how long a connection can remain open. See [WebSockets Limits \(p. 709\)](#) for details.

Prerequisites

- Java 8 or higher.
- Apache Maven 3.3 or higher.

For information about installing these prerequisites on an Amazon EC2 instance running Amazon Linux, see [Prerequisites on Amazon Linux EC2 \(p. 147\)](#).

- IAM credentials to sign the requests. For more information, see [Using the Default Credential Provider Chain](#) in the *AWS SDK for Java Developer Guide*.

Note

If you are using temporary credentials, they expire after a specified interval, *including the session token*.

You must update your session token when you request new credentials. For more information, see [Using Temporary Security Credentials to Request Access to AWS Resources](#) in the *IAM User Guide*.

- Set the `SERVICE_REGION` variable to one of the following, indicating the Region of your Neptune DB instance:
 - US East (N. Virginia): `us-east-1`
 - US East (Ohio): `us-east-2`
 - US West (N. California): `us-west-1`
 - US West (Oregon): `us-west-2`
 - Canada (Central): `ca-central-1`
 - South America (São Paulo): `sa-east-1`
 - Europe (Stockholm): `eu-north-1`
 - Europe (Ireland): `eu-west-1`
 - Europe (London): `eu-west-2`
 - Europe (Paris): `eu-west-3`
 - Europe (Frankfurt): `eu-central-1`
 - Middle East (Bahrain): `me-south-1`
 - Asia Pacific (Hong Kong): `ap-east-1`
 - Asia Pacific (Tokyo): `ap-northeast-1`
 - Asia Pacific (Seoul): `ap-northeast-2`
 - Asia Pacific (Singapore): `ap-southeast-1`
 - Asia Pacific (Sydney): `ap-southeast-2`
 - Asia Pacific (Mumbai): `ap-south-1`
 - China (Beijing): `cn-north-1`
 - China (Ningxia): `cn-northwest-1`
 - AWS GovCloud (US-West): `us-gov-west-1`
 - AWS GovCloud (US-East): `us-gov-east-1`

To connect to Neptune using the Gremlin Java API with Signature Version 4 signing

1. Clone the sample repository from GitHub.

```
git clone https://github.com/aws/amazon-neptune-gremlin-java-sigv4.git
```

2. Change into the cloned directory.

```
cd amazon-neptune-gremlin-java-sigv4
```

3. Get the latest version of the project by checking out the branch with the latest tag.

```
git checkout $(git describe --tags `git rev-list --tags --max-count=1`)
```

4. Important

You must provide IAM credentials to sign the requests. Enter the following commands to set your credentials as environment variables, replacing the relevant items with your credentials.

```
export AWS_ACCESS_KEY_ID=access_key_id
export AWS_SECRET_ACCESS_KEY=secret_access_key
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-central-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-west-3 or eu-central-1 or me-south-1 or ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1 or cn-north-1 or cn-northwest-1 or us-gov-east-1 or us-gov-west-1
```

The Neptune Version 4 Signer uses the default credential provider chain. For additional methods of providing credentials, see [Using the Default Credential Provider Chain](#) in the *AWS SDK for Java Developer Guide*.

The `SERVICE_REGION` variable is required, even when using a credentials file.

5. Enter the following command to compile and run the example code.

Replace *your-neptune-endpoint* with the hostname or IP address of your Neptune DB instance. The default port is 8182.

Note

For information about finding the hostname of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

```
mvn compile exec:java \
-Dexec.mainClass="com.amazon.neptune.gremlin.driver.example.NeptuneGremlinSigV4Example" \
-Dexec.args="-e your-neptune-endpoint -p 8182"
```

6. To view the source code for the example, see the `NeptuneGremlinSigV4Example.java` file in the `src/main/java/com/amazon/neptune/gremlin/driver/example` directory.

To use the SigV4 signing driver in your own Java application, add the following to the `<dependencies>` section of your `pom.xml`.

```
<dependency>
  <groupId>org.apache.tinkerpop</groupId>
  <artifactId>gremlin-driver</artifactId>
  <version>3.4.8</version>
</dependency>
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk-core</artifactId>
  <version>1.11.307</version>
</dependency>
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-sigv4-signer</artifactId>
  <version>2.1.1</version>
</dependency>
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-gremlin-java-sigv4</artifactId>
  <version>2.1.1</version>
```

```
</dependency>
```

Using TinkerPop 3.4.11 or higher to connect to Neptune with Sig4 signing

Here is an example of how to connect to Neptune using the Gremlin Java API with Sig4 signing when using TinkerPop 3.4.11 or higher. First, define the dependencies:

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-sigv4-signer</artifactId>
  <version>2.4.0</version>
</dependency>
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk-core</artifactId>
  <version>1.12.9</version>
</dependency>
```

Then, use code like the following:

```
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer;

...

System.setProperty("aws.accessKeyId", "your-access-key");
System.setProperty("aws.secretKey", "your-secret-key");

...

cluster = Cluster.build(your-cluster).
    enableSsl(true).keyCertChainFile(pem-file).
    handshakeInterceptor( r -> {
        NeptuneNettyHttpSigV4Signer sigV4Signer = new
        NeptuneNettyHttpSigV4Signer(region, new DefaultAWSCredentialsProviderChain());
        sigV4Signer.signRequest(r);
        return r;
    }).create();
Client client = cluster.connect();
client.submit("g.V().has('code', 'IAD')").all().get();
```

Connecting to Neptune Using Java and SPARQL with Signature Version 4 Signing (RDF4J and Jena)

This section shows how to connect to Neptune using either RDF4J or Apache Jena with Signature Version 4 authentication.

Prerequisites

- Java 8 or higher.
- Apache Maven 3.3 or higher.

For information about installing these prerequisites on an EC2 instance running Amazon Linux, see [Prerequisites on Amazon Linux EC2 \(p. 147\)](#).

- IAM credentials to sign the requests. For more information, see [Using the Default Credential Provider Chain](#) in the *AWS SDK for Java Developer Guide*.

Note

If you are using temporary credentials, they expire after a specified interval, *including the session token*.

You must update your session token when you request new credentials. For more information, see [Using Temporary Security Credentials to Request Access to AWS Resources](#) in the *IAM User Guide*.

- Set the `SERVICE_REGION` variable to one of the following, indicating the Region of your Neptune DB instance:
 - US East (N. Virginia): `us-east-1`
 - US East (Ohio): `us-east-2`
 - US West (N. California): `us-west-1`
 - US West (Oregon): `us-west-2`
 - Canada (Central): `ca-central-1`
 - South America (São Paulo): `sa-east-1`
 - Europe (Stockholm): `eu-north-1`
 - Europe (Ireland): `eu-west-1`
 - Europe (London): `eu-west-2`
 - Europe (Paris): `eu-west-3`
 - Europe (Frankfurt): `eu-central-1`
 - Middle East (Bahrain): `me-south-1`
 - Asia Pacific (Hong Kong): `ap-east-1`
 - Asia Pacific (Tokyo): `ap-northeast-1`
 - Asia Pacific (Seoul): `ap-northeast-2`
 - Asia Pacific (Singapore): `ap-southeast-1`
 - Asia Pacific (Sydney): `ap-southeast-2`
 - Asia Pacific (Mumbai): `ap-south-1`
 - China (Beijing): `cn-north-1`
 - China (Ningxia): `cn-northwest-1`
 - AWS GovCloud (US-West): `us-gov-west-1`
 - AWS GovCloud (US-East): `us-gov-east-1`

To connect to Neptune using either RDF4J or Apache Jena with Signature Version 4 signing

1. Clone the sample repository from GitHub.

```
git clone https://github.com/aws/amazon-neptune-sparql-java-sigv4.git
```

2. Change into the cloned directory.

```
cd amazon-neptune-sparql-java-sigv4
```

3. Get the latest version of the project by checking out the branch with the latest tag.

```
git checkout $(git describe --tags `git rev-list --tags --max-count=1`)
```

4. Enter one of the following commands to compile and run the example code.

Replace *your-neptune-endpoint* with the hostname or IP address of your Neptune DB instance.
The default port is 8182.

Note

For information about finding the hostname of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

Eclipse RDF4J

Enter the following to run the RDF4J example.

```
mvn compile exec:java \  
-Dexec.mainClass="com.amazonaws.neptune.client.rdf4j.NeptuneRdf4JSigV4Example" \  
-Dexec.args="https://your-neptune-endpoint:port"
```

Apache Jena

Enter the following to run the Apache Jena example.

```
mvn compile exec:java \  
-Dexec.mainClass="com.amazonaws.neptune.client.jena.NeptuneJenaSigV4Example" \  
-Dexec.args="https://your-neptune-endpoint:port"
```

5. To view the source code for the example, see the examples in the `src/main/java/com/amazonaws/neptune/client/` directory.

To use the SigV4 signing driver in your own Java application, add the `amazon-neptune-sigv4-signer` Maven package to the `<dependencies>` section of your `pom.xml`. We recommend that you use the examples as a starting point.

Connecting to Neptune Using SPARQL and Node.js with Signature Version 4 Signing

Here is an example of how to connect to Neptune SPARQL using Node.js with Signature Version 4 authentication:

```
var AWS = require('aws-sdk');  
  
var region = 'us-west-2'; // e.g. us-west-1  
var neptune_endpoint = 'your-Neptune-cluster-endpoint'; // like: 'cluster-id.region.neptune.amazonaws.com'  
var query = `query=PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
PREFIX class: <http://aws.amazon.com/neptune/csv2rdf/class/>  
PREFIX resource: <http://aws.amazon.com/neptune/csv2rdf/resource/>  
PREFIX prop: <http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/>  
PREFIX objprop: <http://aws.amazon.com/neptune/csv2rdf/objectProperty/>  
  
SELECT ?movies ?title WHERE {  
  ?jel prop:name "James Earl Jones" .  
  ?movies ?p2 ?jel .  
  ?movies prop:title ?title  
} LIMIT 10`;  
  
runQuery(query);  
  
function runQuery(q) {  
  
  var endpoint = new AWS.Endpoint(neptune_endpoint);  
  endpoint.port = 8182;  
  var request = new AWS.HttpRequest(endpoint, region);  
  request.path += 'sparql';
```

```
request.body = encodeURI(query);
request.headers['Content-Type'] = 'application/x-www-form-urlencoded';
request.headers['host'] = neptune_endpoint;
request.method = 'POST';

var credentials = new AWS.CredentialProviderChain();
credentials.resolve((err, cred)=>{
    var signer = new AWS.Signers.V4(request, 'neptune-db');
    signer.addAuthorization(cred, new Date());
});

var client = new AWS.HttpClient();
client.handleRequest(request, null, function(response) {
    console.log(response.statusCode + ' ' + response.statusMessage);
    var responseBody = '';
    response.on('data', function (chunk) {
        responseBody += chunk;
    });
    response.on('end', function (chunk) {
        console.log('Response body: ' + responseBody);
    });
}, function(error) {
    console.log('Error: ' + error);
});
}
```

Example: Connecting to Neptune Using Python with Signature Version 4 Signing

This section shows an example program written in Python that illustrates how to work with Signature Version 4 for Amazon Neptune. This example is based on the examples in the [Signature Version 4 Signing Process](#) section in the *Amazon Web Services General Reference*.

To work with this example program, you need the following:

- Python 3.x installed on your computer, which you can get from the [Python site](#). These programs were tested using Python 3.6.
- The [Python requests library](#), which is used in the example script to make web requests. A convenient way to install Python packages is to use `pip`, which gets packages from the Python package index site. You can then install `requests` by running `pip install requests` at the command line.
- An access key (access key ID and secret access key) in environment variables named `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. As a best practice, we recommend that you do *not* embed credentials in code. For more information, see [Best Practices for Managing AWS Access Keys](#) in the *Amazon Web Services General Reference*.

The Region of your Neptune DB cluster in an environment variable named `SERVICE_REGION`.

If you are using temporary credentials, you must specify `AWS_SESSION_TOKEN` in addition to `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `SERVICE_REGION`.

Note

If you are using temporary credentials, they expire after a specified interval, *including the session token*.

You must update your session token when you request new credentials. For more information, see [Using Temporary Security Credentials to Request Access to AWS Resources](#).

The following example shows how to make signed requests to Neptune using Python. The request makes a either a GET or POST request. Authentication information is passed using the `Authorization` request header.

This example also works as an AWS Lambda function. For more information, see [the section called "Setting Up Lambda" \(p. 144\)](#).

To make signed requests to the Gremlin and SPARQL Neptune endpoints

1. Create a new file named `neptunesigv4.py`, and open it in a text editor.
2. Copy the following code and paste it into the `neptunesigv4.py` file.

```
# Amazon Neptune version 4 signing example (version v3)

# The following script requires python 3.6+
# (sudo yum install python36 python36-virtualenv python36-pip)
# => the reason is that we're using urllib.parse() to manually encode URL
# parameters: the problem here is that SIGV4 encoding requires whitespaces
# to be encoded as %20 rather than not or using '+', as done by previous/
# default versions of the library.

# See: https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
import sys, datetime, hashlib, hmac
import requests # pip3 install requests
import urllib
import os
import json
from botocore.auth import SigV4Auth
from botocore.awsrequest import AWSRequest
from botocore.credentials import ReadOnlyCredentials
from types import SimpleNamespace
from argparse import RawTextHelpFormatter
from argparse import ArgumentParser

# Configuration. https is required.
protocol = 'https'

# The following lines enable debugging at httplib level (requests->urllib3->http.client)
# You will see the REQUEST, including HEADERS and DATA, and RESPONSE with HEADERS but
# without DATA.
#
# The only thing missing will be the response.body which is not logged.
#
# import logging
# from http.client import HTTPConnection
# HTTPConnection.debuglevel = 1
# logging.basicConfig()
# logging.getLogger().setLevel(logging.DEBUG)
# requests_log = logging.getLogger("requests.packages.urllib3")
# requests_log.setLevel(logging.DEBUG)
# requests_log.propagate = True

# Read AWS access key from env. variables. Best practice is NOT
# to embed credentials in code.
access_key = os.getenv('AWS_ACCESS_KEY_ID', '')
secret_key = os.getenv('AWS_SECRET_ACCESS_KEY', '')
region = os.getenv('SERVICE_REGION', '')

# AWS_SESSION_TOKEN is optional environment variable. Specify a session token only if
# you are using temporary
# security credentials.
session_token = os.getenv('AWS_SESSION_TOKEN', '')

### Note same script can be used for AWS Lambda (runtime = python3.6).
## Steps to use this python script for AWS Lambda
```

```
# 1. AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY and AWS_SESSION_TOKEN and AWS_REGION
variables are already part of Lambda's Execution environment
# No need to set them up explicitly.
# 3. Create Lambda deployment package https://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html
# 4. Create a Lambda function in the same VPC and assign an IAM role with neptune
access

def lambda_handler(event, context):
    # sample_test_input = {
    #     "host": "END_POINT:8182",
    #     "method": "GET",
    #     "query_type": "gremlin",
    #     "query": "g.V().count()"
    # }

    # Lambda uses AWS_REGION instead of SERVICE_REGION
    global region
    region = os.getenv('AWS_REGION', '')

    host = event['host']
    method = event['method']
    query_type = event['query_type']
    query = event['query']

    return make_signed_request(host, method, query_type, query)

def validate_input(method, query_type):
    # Supporting GET and POST for now:
    if (method != 'GET' and method != 'POST'):
        print('First parameter must be "GET" or "POST", but is "' + method + '".')
        sys.exit()

    # SPARQL UPDATE requires POST
    if (method == 'GET' and query_type == 'sparqlupdate'):
        print('SPARQL UPDATE is not supported in GET mode. Please choose POST.')
        sys.exit()

def get_canonical_uri_and_payload(query_type, query, method):
    # Set the stack and payload depending on query_type.
    if (query_type == 'sparql'):
        canonical_uri = '/sparql/'
        payload = {'query': query}

    elif (query_type == 'sparqlupdate'):
        canonical_uri = '/sparql/'
        payload = {'update': query}

    elif (query_type == 'gremlin'):
        canonical_uri = '/gremlin/'
        payload = {'gremlin': query}
        if (method == 'POST'):
            payload = json.dumps(payload)

    elif (query_type == 'openCypher'):
        canonical_uri = '/openCypher/'
        payload = {'query': query}

    elif (query_type == "loader"):
        canonical_uri = "/loader/"
        payload = json.loads(query)

    elif (query_type == "status"):
        canonical_uri = "/status/"
        payload = {}
```



```
elif (query_type == "gremlin/status"):
    canonical_uri = "/gremlin/status/"
    payload = {}

elif (query_type == "openCypher/status"):
    canonical_uri = "/openCypher/status/"
    payload = {}

elif (query_type == "sparql/status"):
    canonical_uri = "/sparql/status/"
    payload = {}

else:
    print(
        'Third parameter should be from ["gremlin", "sparql", "sparqlupdate",
"loader", "status"] but is "' + query_type + '"')
    sys.exit()
    ## return output as tuple
    return canonical_uri, payload

def make_signed_request(host, method, query_type, query):
    service = 'neptune-db'
    endpoint = protocol + '://' + host

    print()
    print('+++++ USER INPUT +++++')
    print('host = ' + host)
    print('method = ' + method)
    print('query_type = ' + query_type)
    print('query = ' + query)

    # validate input
    validate_input(method, query_type)

    # get canonical_uri and payload
    canonical_uri, payload = get_canonical_uri_and_payload(query_type, query, method)
    print('payload = {}'.format(payload))

    # assign payload to data or params
    data = payload if method == 'POST' else None
    params = payload if method == 'GET' else None

    # create request URL
    request_url = endpoint + canonical_uri

    # create and sign request
    creds = SimpleNamespace(
        access_key=access_key, secret_key=secret_key, token=session_token,
region=region,
    )

    request = AWSRequest(method=method, url=request_url, data=data, params=params)
    SigV4Auth(creds, service, region).add_auth(request)

    r = None

    # ***** SEND THE REQUEST *****
    if (method == 'GET'):

        print('++++ BEGIN GET REQUEST +++++')
        print('Request URL = ' + request_url)
        r = requests.get(request_url, headers=request.headers, verify=False,
params=params)

    elif (method == 'POST'):
```

```

        print('\n+++++ BEGIN POST REQUEST +++++')
        print('Request URL = ' + request_url)
        r = requests.post(request_url, headers=request.headers, verify=False,
data=data)

    else:
        print('Request method is neither "GET" nor "POST", something is wrong here.')

    if r:
        print()
        print('+++++ RESPONSE +++++')
        print('Response code: %d\n' % r.status_code)
        response = r.text
        r.close()
        print(response)

    return response

help_msg = '''
export AWS_ACCESS_KEY_ID=[MY_ACCESS_KEY_ID]
export AWS_SECRET_ACCESS_KEY=[MY_SECRET_ACCESS_KEY]
export AWS_SESSION_TOKEN=[MY_AWS_SESSION_TOKEN]
export SERVICE_REGION=[us-east-1|us-east-2|us-west-2|eu-west-1]

python version >=3.6 is required.

Examples: For help
python3 program_name.py -h

Examples: Queries
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q status
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql/status
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql -d
"SELECT ?s WHERE { ?s ?p ?o }"
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q sparql -d
"SELECT ?s WHERE { ?s ?p ?o }"
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q sparqlupdate -d
"INSERT DATA { <https://s> <https://p> <https://o> }"
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin/status
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin -d
"g.V().count()"
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q gremlin -d
"g.V().count()"
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher/
status
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher -d
"MATCH (n1) RETURN n1 LIMIT 1;"
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q openCypher -d
"MATCH (n1) RETURN n1 LIMIT 1;"
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
'{"loadId": "68b28dcc-8e15-02b1-133d-9bd0557607e6"}'
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d '{}''
python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q loader -d
'{"source": "source", "format" : "csv", "failOnError": "fail_on_error", "iamRoleArn":
"iam_role_arn", "region": "region"}'

Environment variables must be defined as AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY
and SERVICE_REGION.
You should also set AWS_SESSION_TOKEN environment variable if you are using
temporary credentials (ex. IAM Role or EC2 Instance profile).

Current Limitations:
- Query mode "sparqlupdate" requires POST (as per the SPARQL 1.1 protocol)
'''

def exit_and_print_help():

```

```
print(help_msg)
exit()

def parse_input_and_query_neptune():

    parser = ArgumentParser(description=help_msg, formatter_class=RawTextHelpFormatter)
    group_host = parser.add_mutually_exclusive_group()
    group_host.add_argument("-ho", "--host", type=str)
    group_port = parser.add_mutually_exclusive_group()
    group_port.add_argument("-p", "--port", type=int, help="port ex. 8182,
default=8182", default=8182)
    group_action = parser.add_mutually_exclusive_group()
    group_action.add_argument("-a", "--action", type=str, help="http action, default =
GET", default="GET")
    group_endpoint = parser.add_mutually_exclusive_group()
    group_endpoint.add_argument("-q", "--query_type", type=str, help="query_type,
default = status ", default="status")
    group_data = parser.add_mutually_exclusive_group()
    group_data.add_argument("-d", "--data", type=str, help="data required for the http
action", default="")

    args = parser.parse_args()
    print("parsed input: ")
    print(args)

    # Read command line parameters
    host = args.host
    port = args.port
    method = args.action
    query_type = args.query_type
    query = args.data

    if (access_key == ''):
        print('!!! ERROR: Your AWS_ACCESS_KEY_ID environment variable is undefined.')
        exit_and_print_help()

    if (secret_key == ''):
        print('!!! ERROR: Your AWS_SECRET_ACCESS_KEY environment variable is
undefined.')
        exit_and_print_help()

    if (region == ''):
        print('!!! ERROR: Your SERVICE_REGION environment variable is undefined.')
        exit_and_print_help()

    if host is None:
        print('!!! ERROR: Neptune DNS is missing')
        exit_and_print_help()

    host = host + ":" + str(port)
    make_signed_request(host, method, query_type, query)

if __name__ == "__main__":
    print('here')
    parse_input_and_query_neptune()
```

3. In a terminal, navigate to the location of the `neptunesigv4.py` file.
4. Enter the following commands, replacing the access key, secret key, and Region with the correct values.

```
export AWS_ACCESS_KEY_ID=MY_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY=MY_SECRET_ACCESS_KEY
```

```
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-central-1  
or  
eu-north-1 or eu-west-1 or eu-west-2 or eu-west-3 or eu-central-1  
or me-south-1 or  
ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-  
southeast-2 or ap-south-1 or  
cn-north-1 or cn-northwest-1 or  
us-gov-east-1 or us-gov-west-1
```

If you are using temporary credentials, you must specify `AWS_SESSION_TOKEN` in addition to `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `SERVICE_REGION`.

```
export AWS_SESSION_TOKEN=MY_AWS_SESSION_TOKEN
```

Note

If you are using temporary credentials, they expire after a specified interval, *including the session token*.

You must update your session token when you request new credentials. For more information, see [Using Temporary Security Credentials to Request Access to AWS Resources](#).

5. Enter one of the following commands to send a signed request to the Neptune DB instance. These examples use Python version 3.6.

Endpoint Status

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -a GET -q status
```

Gremlin

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin -d  
"g.V().count()"
```

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q gremlin -d  
"g.V().count()"
```

Gremlin status

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin/status
```

SPARQL

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql -d  
"SELECT ?s WHERE { ?s ?p ?o }"
```

SPARQL UPDATE

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q sparql -d  
"SELECT ?s WHERE { ?s ?p ?o }" -p 8182 -a POST -q sparqlupdate -d "INSERT DATA  
{ <https://s> <https://p> <https://o> }"
```

SPARQL status

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql/status
```

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher -d  
"MATCH (n1) RETURN n1 LIMIT 1;"  
  
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q openCypher -d  
"MATCH (n1) RETURN n1 LIMIT 1;"
```

openCypher status

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher/status
```

Loader

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d  
'{"loadId": "68b28dcc-8e15-02b1-133d-9bd0557607e6"}'  
  
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d '{}'  
  
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q loader -d  
'{"source": "source", "format": "csv", "failOnError": "fail_on_error", "iamRoleArn":  
"iam_role_arn", "region": "region"}'
```

6. The syntax for running the Python script is as follows:

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p port -a GET/POST -q gremlin/  
sparql/sparqlupdate/loader/status -d "string0data"
```

SPARQL UPDATE requires POST.

Logging and Monitoring Amazon Neptune Resources

Amazon Neptune supports various methods for monitoring performance and usage:

- **Cluster status** – Check the health of a Neptune cluster's graph database engine. For more information, see [the section called "Instance Status" \(p. 639\)](#).
- **Amazon CloudWatch** – Neptune automatically sends metrics to CloudWatch and also supports CloudWatch Alarms. For more information, see [the section called "Using CloudWatch" \(p. 641\)](#).
- **Audit log files** – View, download, or watch database log files using the Neptune console. For more information, see [the section called "Audit Logs with Neptune" \(p. 651\)](#).
- **Publishing logs to Amazon CloudWatch Logs** – You can configure a Neptune DB cluster to publish audit log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, use CloudWatch to create alarms and view metrics, and use CloudWatch Logs to store your log records in highly durable storage. For more information, see [Neptune Logs in CloudWatch Logs \(p. 652\)](#).
- **AWS CloudTrail** – Neptune supports API logging using CloudTrail. For more information, see [the section called "Logging Neptune API Calls with AWS CloudTrail" \(p. 654\)](#).
- **Tagging** – Use tags to add metadata to your Neptune resources and track usage based on tags. For more information, see [the section called "Tagging Neptune Resources" \(p. 665\)](#).

Compliance Validation for Amazon Neptune

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs, such as SOC, PCI, FedRAMP, and HIPAA.

To learn whether Neptune or other AWS services are in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

Note

Not all services are compliant with HIPAA.

- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Evaluating Resources with Rules](#) in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon Neptune

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

An Amazon Neptune DB cluster can only be created in an Amazon VPC that has at least two subnets in at least two Availability Zones. By distributing your cluster instances across at least two Availability Zones, Neptune helps ensure that there are instances available in your DB cluster in the unlikely event of an Availability Zone failure. The cluster volume for your Neptune DB cluster always spans three Availability Zones to provide durable storage with less possibility of data loss.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

Setting up Neptune

Welcome to Amazon Neptune. This section helps you create a new Neptune DB cluster and find what you are looking for in the Neptune documentation.

Note

For AWS graph database reference architectures and reference deployment architectures, See [Amazon Neptune Resources](#). These resources can help inform your choices about graph data models and query languages, and accelerate your development process.

Topics

- [Be Sure DNS is Set Up in Your VPC \(p. 166\)](#)
- [Creating a New Neptune DB Cluster Using AWS CloudFormation or Manually \(p. 166\)](#)
- [Getting Started Accessing Your Neptune Graph \(p. 171\)](#)
- [Connecting to Neptune Using IAM Authentication \(p. 173\)](#)
- [Using Gremlin to Access the Graph \(p. 173\)](#)
- [Using RDF and SPARQL to Access the Graph \(p. 176\)](#)
- [Loading Data into Neptune \(p. 177\)](#)
- [Security in Amazon Neptune \(p. 178\)](#)
- [Monitoring Amazon Neptune \(p. 178\)](#)
- [Troubleshooting and Best Practices in Neptune \(p. 178\)](#)

Be Sure DNS is Set Up in Your VPC

Domain Name System (DNS) is a standard by which names used on the internet are resolved to their corresponding IP addresses. A DNS hostname uniquely names a computer and consists of a host name and a domain name. DNS servers resolve DNS hostnames to their corresponding IP addresses.

Check to make sure that DNS hostnames and DNS resolution are both enabled in your VPC. The VPC network attributes `enableDnsHostnames` and `enableDnsSupport` must be set to `true`. To view and modify these attributes, go to the VPC console at <https://console.aws.amazon.com/vpc/>.

For more information, see [Using DNS with your VPC](#).

Note

If you are using Route 53, confirm that your configuration does not override DNS network attributes in your VPC.

Creating a New Neptune DB Cluster Using AWS CloudFormation or Manually

This section shows you how to use an AWS CloudFormation template to set up Amazon Neptune and create required resources quickly, without having to do everything by hand. This is the easiest way to get started with Neptune. The AWS CloudFormation template performs much of the setup for you, including creating an Amazon Elastic Compute Cloud (Amazon EC2) instance.

Important

The AWS CloudFormation stack that is created by this template creates multiple resources, including resources in Neptune, Amazon EC2, Amazon Virtual Private Cloud (Amazon VPC), and AWS Identity and Access Management (IAM).

Some of these resources are not free-tier resources. For pricing information, see [Amazon Neptune pricing](#) and [Amazon EC2 pricing](#). To stop any charges, you can delete the stack when you are finished with it.

This AWS CloudFormation stack is intended as a basis for a tutorial about Neptune. We recommend that you use stricter AWS Identity and Access Management (IAM) policies and security for your production environment if you use this template. For information about securing resources, see [Security in Amazon Virtual Private Cloud](#) and [Security in Amazon EC2](#).

You can also use an AWS CloudFormation template to create a Lambda function to use with Neptune (see [Using AWS CloudFormation to Create a Lambda Function to Use in Neptune](#) (p. 510)).

For information about creating an Amazon Neptune cluster manually using the AWS Management Console, see [Launching a Neptune DB Cluster Using the Console](#) (p. 412).

For information about managing clusters and instances in Neptune, see [Managing Your Amazon Neptune Database](#) (p. 406).

For general information about Neptune, see [Overview of Amazon Neptune Features](#) (p. 46).

Prerequisites for Using AWS CloudFormation to Set Up Neptune

Before you create an Amazon Neptune cluster, you need to have the following:

- The required IAM permissions.
- A key pair.

IAM Permissions

The following permissions allow you to create resources for the AWS CloudFormation stack:

AWS Managed Policies

- `AWSCloudFormationReadOnlyAccess`
- `NeptuneFullAccess`

Additional IAM Permissions

In addition to the managed policies mentioned above, you will also need to add the following permissions in order to create and delete this CloudFormation stack:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "rds:CreateDBCluster",
        "rds:CreateDBInstance"
      ],
      "Resource": [
        "arn:aws:rds:*:*:*"
      ],
      "Condition": {
        "StringEquals": {
          "rds:DatabaseEngine": ["graphdb", "neptune"]
        }
      }
    }
  ]
}
```



```
}
},
{
  "Action": [
    "rds:AddRoleToDBCluster",
    "rds:AddSourceIdentifierToSubscription",
    "rds:AddTagsToResource",
    "rds:ApplyPendingMaintenanceAction",
    "rds:CopyDBClusterParameterGroup",
    "rds:CopyDBClusterSnapshot",
    "rds:CopyDBParameterGroup",
    "rds>CreateDBClusterParameterGroup",
    "rds>CreateDBClusterSnapshot",
    "rds>CreateDBParameterGroup",
    "rds>CreateDBSubnetGroup",
    "rds>CreateEventSubscription",
    "rds>DeleteDBCluster",
    "rds>DeleteDBClusterParameterGroup",
    "rds>DeleteDBClusterSnapshot",
    "rds>DeleteDBInstance",
    "rds>DeleteDBParameterGroup",
    "rds>DeleteDBSubnetGroup",
    "rds>DeleteEventSubscription",
    "rds:DescribeAccountAttributes",
    "rds:DescribeCertificates",
    "rds:DescribeDBClusterParameterGroups",
    "rds:DescribeDBClusterParameters",
    "rds:DescribeDBClusterSnapshotAttributes",
    "rds:DescribeDBClusterSnapshots",
    "rds:DescribeDBClusters",
    "rds:DescribeDBEngineVersions",
    "rds:DescribeDBInstances",
    "rds:DescribeDBLogFiles",
    "rds:DescribeDBParameterGroups",
    "rds:DescribeDBParameters",
    "rds:DescribeDBSecurityGroups",
    "rds:DescribeDBSubnetGroups",
    "rds:DescribeEngineDefaultClusterParameters",
    "rds:DescribeEngineDefaultParameters",
    "rds:DescribeEventCategories",
    "rds:DescribeEventSubscriptions",
    "rds:DescribeEvents",
    "rds:DescribeOptionGroups",
    "rds:DescribeOrderableDBInstanceOptions",
    "rds:DescribePendingMaintenanceActions",
    "rds:DescribeValidDBInstanceModifications",
    "rds:DownloadDBLogFilePortion",
    "rds:FailoverDBCluster",
    "rds:ListTagsForResource",
    "rds:ModifyDBCluster",
    "rds:ModifyDBClusterParameterGroup",
    "rds:ModifyDBClusterSnapshotAttribute",
    "rds:ModifyDBInstance",
    "rds:ModifyDBParameterGroup",
    "rds:ModifyDBSubnetGroup",
    "rds:ModifyEventSubscription",
    "rds:PromoteReadReplicaDBCluster",
    "rds:RebootDBInstance",
    "rds:RemoveRoleFromDBCluster",
    "rds:RemoveSourceIdentifierFromSubscription",
    "rds:RemoveTagsFromResource",
    "rds:ResetDBClusterParameterGroup",
    "rds:ResetDBParameterGroup",
    "rds:RestoreDBClusterFromSnapshot",
    "rds:RestoreDBClusterToPointInTime"
  ],
}
```

```

    "Effect": "Allow",
    "Resource": [
      "*"
    ]
  },
  {
    "Action": [
      "cloudwatch:GetMetricStatistics",
      "cloudwatch:ListMetrics",
      "ec2:DescribeAccountAttributes",
      "ec2:DescribeAvailabilityZones",
      "ec2:DescribeSecurityGroups",
      "ec2:DescribeSubnets",
      "ec2:DescribeVpcAttribute",
      "ec2:DescribeVpcs",
      "kms:ListAliases",
      "kms:ListKeyPolicies",
      "kms:ListKeys",
      "kms:ListRetirableGrants",
      "logs:DescribeLogStreams",
      "logs:GetLogEvents",
      "sns:ListSubscriptions",
      "sns:ListTopics",
      "sns:Publish"
    ],
    "Effect": "Allow",
    "Resource": [
      "*"
    ]
  },
  {
    "Action": "iam:PassRole",
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "iam:passedToService": "rds.amazonaws.com"
      }
    }
  },
  {
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
    "Condition": {
      "StringLike": {
        "iam:AWSServiceName": "rds.amazonaws.com"
      }
    }
  }
]
}

```

Note

The following permissions are only required to delete a stack: iam:DeleteRole, iam:RemoveRoleFromInstanceProfile, iam:DeleteRolePolicy, iam:DeleteInstanceProfile, and ec2:DeleteVpcEndpoints. Also note that ec2:*Vpc grants ec2:DeleteVpc permissions.

EC2 Key Pair

















You must have a key pair (and the PEM file) available in the Region that you create the AWS CloudFormation stack in. If you need to create a key pair, see either [Creating a Key Pair Using Amazon](#)

[EC2](#) in the Amazon EC2 User Guide for Linux Instances or [Creating a Key Pair Using Amazon EC2](#) in the Amazon EC2 User Guide for Windows Instances for instructions.

Using an AWS CloudFormation Stack to Create a Neptune DB Cluster

You can use an AWS CloudFormation template to set up a Neptune DB Cluster.

1. To launch the AWS CloudFormation stack on the AWS CloudFormation console, choose one of the **Launch Stack** buttons in the following table.

Region	View	View in Designer	Launch
US East (N. Virginia)	View	View in Designer	
US East (Ohio)	View	View in Designer	
US West (N. California)	View	View in Designer	
US West (Oregon)	View	View in Designer	
Canada (Central)	View	View in Designer	
South America (São Paulo)	View	View in Designer	
Europe (Stockholm)	View	View in Designer	
Europe (Ireland)	View	View in Designer	
Europe (London)	View	View in Designer	
Europe (Paris)	View	View in Designer	
Europe (Frankfurt)	View	View in Designer	
Middle East (Bahrain)	View	View in Designer	
Asia Pacific (Hong Kong)	View	View in Designer	
Asia Pacific (Tokyo)	View	View in Designer	
Asia Pacific (Seoul)	View	View in Designer	
Asia Pacific (Singapore)	View	View in Designer	

Region	View	View in Designer	Launch
Asia Pacific (Sydney)	View	View in Designer	
Asia Pacific (Mumbai)	View	View in Designer	
China (Beijing)	View	View in Designer	
China (Ningxia)	View	View in Designer	
AWS GovCloud (US-West)	View	View in Designer	
AWS GovCloud (US-East)	View	View in Designer	

2. On the **Select Template** page, choose **Next**.
3. On the **Specify Details** page, choose a key pair for the **EC2SSHKeyPairName**.

This key pair is required to access the EC2 instance. Ensure that you have the PEM file for the key pair that you choose.

4. Choose **Next**.
5. On the **Options** page, choose **Next**.
6. On the **Review** page, select the first check box to acknowledge that AWS CloudFormation will create IAM resources. Select the second check box to acknowledge `CAPABILITY_AUTO_EXPAND` for the new stack.

Note

`CAPABILITY_AUTO_EXPAND` explicitly acknowledges that macros will be expanded when creating the stack, without prior review. Users often create a change set from a processed template so that the changes made by macros can be reviewed before actually creating the stack. For more information, see the AWS CloudFormation [CreateStack](#) API.

Then choose **Create**.

Getting Started Accessing Your Neptune Graph

Once you have created a Neptune DB cluster, the next step is to set up a connection to it.

Connecting to a Neptune DB Cluster from Amazon Elastic Compute Cloud

After creating an instance in Amazon Elastic Compute Cloud (Amazon EC2), you can log into that instance using SSH and connect to a Amazon Neptune DB cluster. For information about connecting to an EC2 instance using SSH, see [Connect to Your Linux Instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

If you are using a Linux or macOS command line to connect to the EC2 instance, you can paste the SSH command from the **SSHAccess** item in the **Outputs** section of the AWS CloudFormation stack. You must have the PEM file in the current directory and the PEM file permissions must be set to 400 (`chmod 400 keypair.pem`).

In order for the Amazon EC2 instance to connect to your Neptune endpoint on, for example, port 8182, you will need to set up a security group to do that. If your Amazon EC2 instance is using a security group named, for example, `ec2-sg1`, you need to create another Amazon EC2 security group (let's say `db-sg1`) that has inbound rules for port 8182 and has `ec2-sg1` as its source. Then, add `db-sg1` to your Neptune cluster to allow the connection.

Setting Up curl to Communicate with Your Neptune Endpoint

As illustrated in many of the examples in this documentation, the `curl` command line tool is a handy option for communicating with your Neptune endpoint. For information about the tool, see the [curl man page](#), and the book *Everything curl*.

To connect using HTTPS (as we recommend and as Neptune requires in most Regions), `curl` needs access to appropriate certificates. To learn how to obtain these certificates and how to format them properly into a certificate authority (CA) certificate store that `curl` can use, see [SSL Certificate Verification](#) in the `curl` documentation.

You can then specify the location of this CA certificate store using the `CURL_CA_BUNDLE` environment variable. On Windows, `curl` automatically looks for it in a file named `curl-ca-bundle.crt`. It looks first in the same directory as `curl.exe` and then elsewhere on the path. For more information, see [SSL Certificate Verification](#).

As long as `curl` can locate the appropriate certificates, it handles HTTPS connections just like HTTP connections, without extra parameters. Examples in this documentation are based on that scenario.

Using a Query Language to Access Graph Data in Your Neptune DB Cluster

Once you are connected, you can use one of the two graph query languages supported by Neptune to access your new graph: Gremlin and SPARQL.

- **Gremlin** (See [Apache TinkerPop3](#)) — Gremlin is a graph traversal language. A query in Gremlin is a traversal made up of discrete steps. Each step follows an edge to a node. To learn about connecting to Neptune using Gremlin, see [Accessing a Neptune graph with Gremlin \(p. 262\)](#).

The Neptune implementation of Gremlin has some differences from other implementations, especially when you are using Gremlin-Groovy (Gremlin queries sent as serialized text). For more information, see [Gremlin standards compliance in Amazon Neptune \(p. 47\)](#).

For a walkthrough of basic Gremlin commands, see the [the section called "Using Gremlin" \(p. 173\)](#) section of the Amazon Neptune Quick Start.

- **SPARQL** (See [SPARQL 1.1](#)) — SPARQL is a declarative query language that is based on the graph pattern matching that is standardized by the World Wide Web Consortium (W3C) and described in the [SPARQL 1.1 Query Language](#) specification. To learn about connecting to Neptune using SPARQL, see [Accessing the Neptune graph with SPARQL \(p. 342\)](#).

Note

You can store Gremlin and SPARQL data on the same cluster. However, they are separated on the cluster, and any data that is loaded or stored with one query language cannot be queried by the other.

You might want to use IAM authentication to connect to Gremlin or SPARQL. For more information about this option, see [Connecting to Neptune Using IAM Authentication \(p. 173\)](#).

Connecting to Neptune Using IAM Authentication

For information about setting up AWS Identity and Access Management (IAM) authentication, see [the section called "Identity and Access Management \(IAM\)" \(p. 111\)](#).

For information about using temporary credentials to authenticate, including examples for the AWS CLI, AWS Lambda, and Amazon EC2, see [the section called "Temporary Credentials" \(p. 141\)](#).

The following links provide information about connecting to Neptune using IAM authentication.

Gremlin

- [the section called "Gremlin Console" \(p. 148\)](#)
- [the section called "Gremlin Java" \(p. 151\)](#)
- [the section called "Python Example" \(p. 157\)](#)

Note

This example applies to both Gremlin and SPARQL.

SPARQL

- [the section called "SPARQL Java \(RDF4J and Jena\)" \(p. 154\)](#)
- [the section called "Python Example" \(p. 157\)](#)

Note

This example applies to both Gremlin and SPARQL.

Using Gremlin to Access the Graph

You can use the Gremlin Console to experiment with TinkerPop graphs and queries in a REPL (read-eval-print loop) environment.

The following tutorial walks you through using the Gremlin console to add vertices, edges, properties, and more to a Neptune graph, highlights some differences in the Neptune-specific Gremlin implementation.

Note

This example assumes that you have completed the following:

- You have connected using SSH to an Amazon EC2 instance.
- You have created a Neptune cluster as detailed in [Creating a DB Cluster \(p. 166\)](#).
- You have installed the Gremlin console as described in [Installing the Gremlin console \(p. 263\)](#).

Using the Gremlin Console

1. Change directories into the folder where the Gremlin console files are unzipped.

```
cd apache-tinkerpop-gremlin-console-3.4.8
```

2. Enter the following command to run the Gremlin Console.

```
bin/gremlin.sh
```

You should see the following output:

```
      \, , , /
      (o o)
-----o00o-(3)-o00o-----
plugin activated: tinkerspop.server
plugin activated: tinkerspop.utilities
plugin activated: tinkerspop.tinkersgraph
gremlin>
```

You are now at the `gremlin>` prompt. You enter the remaining steps at this prompt.

3. At the `gremlin>` prompt, enter the following to connect to the Neptune DB instance.

```
:remote connect tinkerspop.server conf/neptune-remote.yaml
```

4. At the `gremlin>` prompt, enter the following to switch to remote mode. This sends all Gremlin queries to the remote connection.

```
:remote console
```

5. **Add vertex with label and property.**

```
g.addV('person').property('name', 'justin')
```

The vertex is assigned a `string` ID containing a GUID. All vertex IDs are strings in Neptune.

6. **Add a vertex with custom id.**

```
g.addV('person').property(id, '1').property('name', 'martin')
```

The `id` property is not quoted. It is a keyword for the ID of the vertex. The vertex ID here is a string with the number 1 in it.

Normal property names must be contained in quotation marks.

7. **Change property or add property if it doesn't exist.**

```
g.V('1').property(single, 'name', 'marko')
```

Here you are changing the name property for the vertex from the previous step. This removes all existing values from the name property.

If you didn't specify `single`, it instead appends the value to the name property if it hasn't done so already.

8. **Add property, but append property if property already has a value.**

```
g.V('1').property('age', 29)
```

Neptune uses set cardinality as the default action.

This command adds the age property with the value 29, but it does not replace any existing values.

If the age property already had a value, this command appends 29 to the property. For example, if the age property was 27, the new value would be [27, 29].

9. **Add multiple vertices.**

```
g.addV('person').property(id, '2').property('name', 'vadas').property('age', 27).iterate()
g.addV('software').property(id, '3').property('name', 'lop').property('lang', 'java').iterate()
g.addV('person').property(id, '4').property('name', 'josh').property('age', 32).iterate()
g.addV('software').property(id, '5').property('name', 'ripple').property('lang', 'java').iterate()
g.addV('person').property(id, '6').property('name', 'peter').property('age', 35)
```

You can send multiple statements at the same time to Neptune.

Statements can be separated by newline (`'\n'`), spaces (`' '`), semicolon (`';'`), or nothing (for example: `g.addV('person').iterate()g.V()` is valid).

Note

The Gremlin Console sends a separate command at every newline (`'\n'`), so they are each a separate transaction in that case. This example has all the commands on separate lines for readability. Remove the newline (`'\n'`) characters to send it as a single command via the Gremlin Console.

All statements other than the last statement must end in a terminating step, such as `.next()` or `.iterate()`, or they will not run. The Gremlin Console does not require these terminating steps. Use `.iterate` whenever you don't need the results to be serialized.

All statements that are sent together are included in a single transaction and succeed or fail together.

10. Add edges.

```
g.V('1').addE('knows').to(g.V('2')).property('weight', 0.5).iterate()
g.addE('knows').from(g.V('1')).to(g.V('4')).property('weight', 1.0)
```

Here are two different ways to add an edge.

11. Add the rest of the Modern graph.

```
g.V('1').addE('created').to(g.V('3')).property('weight', 0.4).iterate()
g.V('4').addE('created').to(g.V('5')).property('weight', 1.0).iterate()
g.V('4').addE('knows').to(g.V('3')).property('weight', 0.4).iterate()
g.V('6').addE('created').to(g.V('3')).property('weight', 0.2)
```

12. Delete a vertex.

```
g.V().has('name', 'justin').drop()
```

Removes the vertex with the name property equal to `justin`.

Important

Stop here, and you have the full Apache TinkerPop Modern graph. The examples in the [Traversal section](#) of the TinkerPop documentation use the Modern graph.

13. Run a traversal.

```
g.V().hasLabel('person')
```

Returns all `person` vertices.

14. Run a Traversal with values (`valueMap()`).


```
g.V().has('name', 'marko').out('knows').valueMap()
```

Returns key, value pairs for all vertices that marko “knows.”

15. Specify multiple labels.

```
g.addV("Label1::Label2::Label3")
```

Neptune supports multiple labels for a vertex. When you create a label, you can specify multiple labels by separating them with `::`.

This example adds a vertex with three different labels.

The `hasLabel` step matches this vertex with any of those three labels: `hasLabel("Label1")`, `hasLabel("Label2")`, and `hasLabel("Label3")`.

The `::` delimiter is reserved for this use only.

You cannot specify multiple labels in the `hasLabel` step. For example, `hasLabel("Label1::Label2")` does not match anything.

16. Specify Time/date.

```
g.V().property(single, 'lastUpdate', datetime('2018-01-01T00:00:00'))
```

Neptune does not support Java Date. Use the `datetime()` function instead. `datetime()` accepts an ISO8061-compliant `datetime` string.

It supports the following formats: `YYYY-MM-DD`, `YYYY-MM-DDTHH:mm`, `YYYY-MM-DDTHH:mm:ss`, and `YYYY-MM-DDTHH:mm:ssz`.

17. Delete vertices, properties, or edges.

```
g.V().hasLabel('person').properties('age').drop().iterate()
g.V('1').drop().iterate()
g.V().outE().hasLabel('created').drop()
```

Here are several drop examples.

Note

The `.next()` step does not work with `.drop()`. Use `.iterate()` instead.

18. When you are finished, enter the following to exit the Gremlin Console.

```
:exit
```

Note

Use a semicolon (;) or a newline character (\n) to separate each statement.

Each traversal preceding the final traversal must end in `iterate()` to be executed. Only the data from the final traversal is returned.

Using RDF and SPARQL to Access the Graph

SPARQL is a query language for the Resource Description Framework (RDF), which is a graph data format designed for the web. Amazon Neptune is compatible with SPARQL 1.1. This means that you can connect

to a Neptune DB instance and query the graph using the query language described in the [SPARQL 1.1 Query Language](#) specification.

A query in SPARQL consists of a `SELECT` clause to specify the variables to return and a `WHERE` clause to specify which data to match in the graph. If you are unfamiliar with SPARQL queries, see [Writing Simple Queries](#) in the [SPARQL 1.1 Query Language](#).

The HTTP endpoint for SPARQL queries to a Neptune DB instance is `https://your-neptune-endpoint:port/sparql`.

To connect to SPARQL

1. You can get the SPARQL endpoint for your Neptune cluster from the **SparqlEndpoint** item in the **Outputs** section of the AWS CloudFormation stack.
2. Enter the following to submit a SPARQL **UPDATE** using HTTP `POST` and the `curl` command.

```
curl -X POST --data-binary 'update=INSERT DATA { <https://test.com/s> <https://test.com/p> <https://test.com/o> . }' https://your-neptune-endpoint:port/sparql
```

The preceding example inserts the following triple into the SPARQL default graph: `<https://test.com/s> <https://test.com/p> <https://test.com/o>`

3. Enter the following to submit a SPARQL **QUERY** using HTTP `POST` and the `curl` command.

```
curl -X POST --data-binary 'query=select ?s ?p ?o where {?s ?p ?o} limit 10' https://your-neptune-endpoint:port/sparql
```

The preceding example returns up to 10 of the triples (subject-predicate-object) in the graph by using the `?s ?p ?o` query with a limit of 10. To query for something else, replace it with another SPARQL query.

Note

The default MIME type of a response is `application/sparql-results+json` for `SELECT` and `ASK` queries.

The default MIME type of a response is `application/n-quads` for `CONSTRUCT` and `DESCRIBE` queries.

For a list of all available MIME types, see [SPARQL HTTP API \(p. 348\)](#).

Loading Data into Neptune

Amazon Neptune provides a process for loading data from external files directly into a Neptune DB instance. You can use this process instead of executing a large number of `INSERT` statements, `addV` and `addE` steps, or other API calls.

The following are links to additional loading information.

- **Prerequisites for loading** – [the section called “IAM Role and Amazon S3 Access” \(p. 191\)](#)
- **Neptune CSV format** – [the section called “Gremlin data format” \(p. 195\)](#)
- **Supported RDF formats** – [the section called “RDF data formats” \(p. 203\)](#)
- **Loading example** – [the section called “Loading Example” \(p. 204\)](#)

Security in Amazon Neptune

There are multiple ways for you to secure your Amazon Neptune clusters.

IAM Permissions for Cluster Management

To control who can perform Neptune management actions on Neptune DB clusters and DB instances, you use AWS Identity and Access Management (IAM). When you connect to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions that are required to perform Neptune management operations. For more information, see [Creating and using your own IAM policies for access to Neptune](#) (p. 119).

If you are using an IAM account to access the Neptune console, you must first sign in to the AWS Management Console using your IAM account. Then open the Neptune console at <https://console.aws.amazon.com/neptune/home>.

VPC and VPC Security Groups

Neptune DB clusters must be created in an Amazon Virtual Private Cloud (Amazon VPC). To control which devices and EC2 instances can open connections to the endpoint and port of the DB instance for Neptune DB clusters in a VPC, you use a VPC security group. For more information about VPCs, see [Creating a Security Group to Provide Access to a Neptune DB Instance in a VPC](#) (p. 107).

IAM Authentication

You can use IAM database authentication for Neptune. With IAM database authentication, you authenticate to your Neptune DB cluster with an IAM user. For more information, see [Identity and Access Management in Amazon Neptune](#) (p. 111).

Encryption at Rest

You can use [AWS Key Management Service \(AWS KMS\)](#) to create encryption keys and then use those keys to encrypt Neptune cluster data at rest. For more information, see [Encrypting Neptune Resources at Rest](#) (p. 109).

Monitoring Amazon Neptune

Amazon Neptune supports the following monitoring methods.

- **Amazon CloudWatch** – Amazon Neptune automatically sends metrics to CloudWatch and also supports CloudWatch Alarms. For more information, see [the section called “Using CloudWatch”](#) (p. 641).
- **AWS CloudTrail** – Amazon Neptune supports API logging using CloudTrail. For more information, see [the section called “Logging Neptune API Calls with AWS CloudTrail”](#) (p. 654).
- **Tagging** – Use tags to add metadata to your Neptune resources and track usage based on tags. For more information, see [the section called “Tagging Neptune Resources”](#) (p. 665).
- **Audit log files** – View, download, or watch database log files using the Neptune console. For more information, see [the section called “Audit Logs with Neptune”](#) (p. 651).
- **Instance status** – Check the health of a Neptune instance's graph database engine, find out what version of the engine is installed, and obtain other engine status information using the [instance status API](#) (p. 639).

Troubleshooting and Best Practices in Neptune

The following links might be helpful for resolving issues with Amazon Neptune.

- **Best practices** – For solutions to common issues and performance suggestions, see [Best Practices \(p. 691\)](#).
- **Service errors** – For a list of errors for both management APIs and graph database connections, see [Neptune Errors \(p. 711\)](#).
- **Service limits** – For information about Neptune limits, see [Neptune Limits \(p. 707\)](#).
- **Engine releases** – For information about graph engine releases, including release notes, see [Engine Releases \(p. 718\)](#).
- **Support forums** – To join a discussion about Neptune, see the [Amazon Neptune Forum](#).
- **Pricing** – For information about the costs of using Amazon Neptune, see [Amazon Neptune Pricing](#).
- **AWS Support** – For help and guidance from the experts, see [AWS Support](#).

Migrating an existing graph to Amazon Neptune

There are a number of tools and techniques that can help you migrate existing graph data into Amazon Neptune from another data store.

A simple migration workflow involves the following steps:

- Export the data from its existing store into Amazon Simple Storage Service (Amazon S3).
- Clean and format it for import.
- Load it into a Neptune DB cluster using [Neptune Bulk Loader \(p. 190\)](#).
- Configure your Gremlin or SPARQL application to use the corresponding endpoint that Neptune provides.

Note

Your Neptune cluster must be running in a VPC that your application can access.

There are ways to simplify and automate some of these steps, depending on where the data is stored:

Topics

- [Migrating an existing graph from an Apache TinkerPop Gremlin server to Amazon Neptune \(p. 181\)](#)
- [Migrating an existing graph from an RDF triple store to Amazon Neptune \(p. 182\)](#)
- [Using AWS Database Migration Service \(AWS DMS\) to migrate from a relational or NoSQL database to Amazon Neptune \(p. 183\)](#)
- [Migrating from Neo4j to Amazon Neptune \(p. 184\)](#)
- [Migrating from Blazegraph to Amazon Neptune \(p. 185\)](#)

Migrating an existing graph from an Apache TinkerPop Gremlin server to Amazon Neptune

If you have graph data in an Apache TinkerPop Gremlin Server that you would like to migrate to Amazon Neptune, you would take the following steps:

1. Export the data from Gremlin server into Amazon Simple Storage Service (Amazon S3).
2. Convert the exported data to a [CSV format that the Neptune bulk loader can import \(p. 195\)](#).
3. Using [Neptune Bulk Loader \(p. 190\)](#), import the data into a Neptune DB cluster that you have prepared.
4. Modify your existing application to connect to Neptune's Gremlin endpoint, and make any changes necessary to conform with [Neptune Gremlin implementation differences \(p. 47\)](#).

Migrating an existing graph from an RDF triple store to Amazon Neptune

If you have graph data in an RDF/SPARQL to migrate to Amazon Neptune, you would take the following steps:

1. Export the data from your RDF triple store.
2. Convert the exported data to a [format that the Neptune bulk loader can import \(p. 203\)](#).
3. Store the data to be imported in Amazon Simple Storage Service (Amazon S3).
4. Using [Neptune Bulk Loader \(p. 190\)](#), import the data from Amazon S3 into a Neptune DB cluster that you have prepared.
5. Modify your existing application to connect to Neptune's SPARQL endpoint.

If you want to try out migrating property-graph CSV data into RDF, you can use the [Amazon Neptune CSV to RDF converter](#).

Using AWS Database Migration Service (AWS DMS) to migrate from a relational or NoSQL database to Amazon Neptune

AWS Database Migration Service (AWS DMS) is a cloud service that makes it easy to migrate relational databases, data warehouses, NoSQL databases, and other types of data stores. If you have graph data stored in one of the relational or NoSQL [databases that AWS DMS supports](#), AWS DMS can help you migrate to Neptune quickly and securely, without requiring downtime from your current database. See [Using AWS Database Migration Service to load data into Amazon Neptune from a different data store \(p. 225\)](#) for details.

The migration dataflow using AWS DMS is as follows:

- Create a AWS DMS table-mapping object. This JSON object specifies which tables should be read from your source database and in what order, and how their columns are named. It can also filter the rows being copied and provide simple value transformations such as converting to lower case or rounding.
- Create a Neptune `GraphMappingConfig` to specify how the data extracted from the source database should be loaded into Neptune.
 - For RDF data (queried using SPARQL), the `GraphMappingConfig` is written in the W3's standard [R2RML](#) mapping language.
 - For property graph data (queried using Gremlin), the `GraphMappingConfig` is a JSON object, as described in [GraphMappingConfig Layout for Property-Graph/Gremlin Data \(p. 227\)](#)
- Create an AWS DMS replication instance in the same VPC as your Neptune DB cluster, to perform the migration.
- Create an Amazon S3 bucket to be used as intermediary storage for staging the data being migrated.
- Run the AWS DMS migration task.

See [Using AWS Database Migration Service to load data into Amazon Neptune from a different data store \(p. 225\)](#) for the details, and also Chris Smith's four-piece blog post, "Populating your graph in Amazon Neptune from a relational database using AWS Database Migration Service (DMS):"

- [Part 1: Setting the stage](#)
- [Part 2: Designing the property graph model](#)
- [Part 3: Designing the RDF Model](#)
- [Part 4: Putting it all together](#)

Migrating from Neo4j to Amazon Neptune

Migrating from Neo4j – Neptune provides a [neo4j-to-neptune](#) command line utility and an [example AWS CDK application](#) to make migration easy.

See [Migrating a Neo4j graph database to Amazon Neptune with a fully automated utility](#) in the [AWS Database Blog](#) for a complete, step-by-step walkthrough.

After migrating, you may want to capture and replicate ongoing updates to the Neo4j database in Neptune as well. The blog post, [Change data capture from Neo4j to Amazon Neptune using Amazon Managed Streaming for Apache Kafka](#) shows you how to set that up.

Migrating from Blazegraph to Amazon Neptune

If you have a graph in the open-source [Blazegraph](#) RDF triplestore, you can migrate to your graph data to Amazon Neptune using the following steps:

- *Provision AWS infrastructure.* Begin by provisioning the required Neptune infrastructure using an AWS CloudFormation template (see [Creating a DB Cluster \(p. 166\)](#)).
- *Export data from Blazegraph.* There are two main methods for exporting data from Blazegraph, namely using SPARQL CONSTRUCT queries or using the Blazegraph Export utility.
- *Import the data into Neptune.* You can then load the exported data files into Neptune using the [Neptune workbench \(p. 30\)](#) and [Neptune Bulk Loader \(p. 190\)](#).

This approach is also generally applicable for migrating from other RDF triplestore databases.

Blazegraph to Neptune compatibility

Before migrating your graph data to Neptune, there are several significant differences between Blazegraph and Neptune that you should be aware of. These differences can require changes to queries, the application architecture, or both, or even make migration impractical:

- **Full-text search** – In Blazegraph, you can either use internal full-text search or external full-text search capabilities through an integration with Apache Solr. If you use either of these features, stay informed about the latest updates on the full-text search features that Neptune supports. See [Neptune full text search \(p. 471\)](#).
- **Query hints** – Both Blazegraph and Neptune extend SPARQL using the concept of query hints. During a migration, you need to migrate any query hints you use. For information about the latest query hints Neptune supports, see [SPARQL query hints \(p. 357\)](#).
- **Inference** – Blazegraph supports inference as a configurable option in triples mode, but not in quads mode. Neptune does not yet support inference.
- **Geospatial search** – Blazegraph supports the configuration of namespaces that enable geospatial support. This feature is not yet available in Neptune.
- **Multi-tenancy** – Blazegraph supports multi-tenancy within a single database. In Neptune, multi-tenancy is supported either by storing data in named graphs and using the USING NAMED clauses for SPARQL queries, or by creating a separate database cluster for each tenant.
- **Federation** – Neptune currently supports SPARQL 1.1 federation to locations accessible to the Neptune instance, such as within the private VPC, across VPCs, or to external internet endpoints. Depending on the specific setup and required federation endpoints, you may need some additional network configuration.
- **Blazegraph standards extensions** – Blazegraph includes multiple extensions to both the SPARQL and REST API standards, whereas Neptune is only compatible with the standards specifications themselves. This may require changes to your application, or make migration difficult.

Provisioning AWS infrastructure for Neptune

Although you can construct the required AWS infrastructure manually through the AWS Management Console or AWS CLI, it's often more convenient to use a CloudFormation template instead, as described below:

Provisioning Neptune using a CloudFormation template:

1. Navigate to [Using an AWS CloudFormation Stack to Create a Neptune DB Cluster \(p. 170\)](#).

2. Choose **Launch Stack** in your preferred region.
3. Set the required parameters (stack name and `EC2SSHPairName`). Also set the following optional parameters to ease the migration process:
 - Set `AttachBulkloadIAMRoleToNeptuneCluster` to true. This parameter allows for creating and attaching the appropriate IAM role to your cluster to allow for bulk loading data.
 - Set `NotebookInstanceType` to your preferred instance type. This parameter creates a Neptune workbook that you use to run the bulk load into Neptune and validate the migration.
4. Choose **Next**.
5. Set any other stack options you want.
6. Choose **Next**.
7. Review your options and select both check boxes to acknowledge that AWS CloudFormation may require additional capabilities.
8. Choose **Create stack**.

The stack creation process can take a few minutes.

Exporting data from Blazegraph

The next step is to export data out of Blazegraph in a [format that is compatible with the Neptune bulk loader \(p. 203\)](#).

Depending on how the data is stored in Blazegraph (triples or quads) and how many named graphs are in use, Blazegraph may require that you perform the export process multiple times and generate multiple data files:

- If the data is stored as triples, you need to run one export for each named graph.
- If the data is stored as quads, you may choose to either export data in N-Quads format or export each named graph in a triples format.

Below we assume that you export a single namespace as N-Quads, but you can repeat the process for additional namespaces or desired export formats.

If you need Blazegraph to be online and available during the migration, use SPARQL CONSTRUCT queries. This requires that you install, configure, and run a Blazegraph instance with an accessible SPARQL endpoint.

If you don't need Blazegraph to be online, use [the BlazeGraph Export utility](#). To do this you must download Blazegraph, and the data file and configuration files need to be accessible, but the server doesn't need to be running.

Exporting data from Blazegraph using SPARQL CONSTRUCT

SPARQL CONSTRUCT is a feature of SPARQL that returns an RDF graph matching the a specified query template. For this use case, you use it to export your data one namespace at a time, using a query like the following:

```
CONSTRUCT WHERE { hint:Query hint:analytic "true" . hint:Query hint:constructDistinctSPO "false" . ?s ?p ?o }
```

Although other RDF tools exist to export this data, the easiest way to run this query is by using the REST API endpoint provided by Blazegraph. The following script demonstrates how to use a Python (3.6+) script to export data as N-Quads:

```
import requests

# Configure the URL here: e.g. http://localhost:9999/sparql
url = "http://localhost:9999/sparql"
payload = {'query': 'CONSTRUCT WHERE { hint:Query hint:analytic "true" . hint:Query
  hint:constructDistinctSPO "false" . ?s ?p ?o }'}
# Set the export format to be n-quads
headers = {
'Accept': 'text/x-nquads'
}
# Run the http request
response = requests.request("POST", url, headers=headers, data = payload, files = [])
#open the file in write mode, write the results, and close the file handler
f = open("export.nq", "w")
f.write(response.text)
f.close()
```

If the data is stored as triples, you need to change the `Accept` header parameter to export data in an appropriate format (N-Triples, RDF/XML, or Turtle) using the values specified on the [Blazegraph GitHub repo](#).

Using the Blazegraph export utility to export data

Blazegraph contains a utility method to export data, namely the `ExportKB` class. `ExportKB` facilitates exporting data from Blazegraph, but unlike the previous method, requires that the server be offline while the export is running. This makes it the ideal method to use when you can take Blazegraph offline during migration, or the migration can occur from a backup of the data.

You run the utility from a Java command line on a machine that has Blazegraph installed but not running. The easiest way to run this command is to download the latest [blazegraph.jar](#) release located on GitHub. Running this command requires several parameters:

- **log4j.primary.configuration** – The location of the log4j properties file.
- **log4j.configuration** – The location of the log4j properties file.
- **output** – The output directory for the exported data. Files are located as a `tar.gz` in a subdirectory named as documented in the knowledge base.
- **format** – The desired output format followed by the location of the `RWStore.properties` file. If you're working with triples, you need to change the `-format` parameter to `N-Triples`, `Turtle`, or `RDF/XML`.

For example, if you have the Blazegraph journal file and properties files, export data as N-Quads using the following code:

```
java -cp blazegraph.jar \
  com.bigdata.rdf.sail.ExportKB \
  -outdir ~/temp/ \
  -format N-Quads \
  ./RWStore.properties
```

If the export is successful, you see output like this:

```
Exporting kb as N-Quads on /home/ec2-user/temp/kb
Effective output directory: /home/ec2-user/temp/kb
Writing /home/ec2-user/temp/kb/kb.properties
Writing /home/ec2-user/temp/kb/data.nq.gz
Done
```

Create an Amazon Simple Storage Service (Amazon S3) bucket and copy the exported data into it

Once you have exported your data from Blazegraph, create an Amazon Simple Storage Service (Amazon S3) bucket in the same Region as the target Neptune DB cluster for the Neptune bulk loader to use to import the data from.

For instructions on how to create an Amazon S3 bucket, see [How do I create an S3 Bucket?](#) in the [Amazon Simple Storage Service User Guide](#), and [Examples of creating a bucket](#) in the [Amazon Simple Storage Service User Guide](#).

For instructions about how to copy the data files you have exported into the new Amazon S3 bucket, see [Uploading an object to a bucket](#) in the [Amazon Simple Storage Service User Guide](#), or [Using high-level \(\\$3\) commands with the AWS CLI](#). You can also use Python code like the following to copy the files one by one:

```
import boto3

region = 'region name'
bucket_name = 'bucket name'
s3 = boto3.resource('s3')
s3.meta.client.upload_file('export.nq', bucket_name, 'export.nq')
```

Use the Neptune bulk loader to import the data into Neptune

After exporting your data from Blazegraph and copying it into an Amazon S3 bucket, you are ready to import the data into Neptune. Neptune has a bulk loader that loads data faster and with less overhead than performing load operations using SPARQL. The bulk loader process is started by a call to the loader endpoint API to load data stored in the identified S3 bucket into Neptune.

Although you could do this with a direct call to the loader REST endpoint, you must have access to the private VPC in which the target Neptune instance runs. You could set up a bastion host, SSH into that machine, and run the cURL command, but using [Neptune Workbench \(p. 30\)](#) is easier.

Neptune Workbench is a preconfigured Jupyter notebook running as an Amazon SageMaker notebook, with several Neptune-specific notebook magics installed. These magics simplify common Neptune operations such as checking the cluster status, running SPARQL and Gremlin traversals, and running a bulk loading operation.

To start the bulk load process use the `%load` magic, which provides an interface to run the [Neptune Loader Command \(p. 209\)](#):

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Select **aws-neptune-blazegraph-to-neptune**.
3. Choose **Open notebook**.
4. In the running instance of Jupyter, either select an existing notebook or create a new one using the Python 3 kernel.
5. In your notebook, open a cell, enter `%load`, and run the cell.
6. Set the parameters for the bulk loader:
 - a. For **Source**, enter the location of a source file to import: `s3://{bucket_name}/{file_name}`.

- b. For **Format**, choose the appropriate format, which in this example is `nquads`.
 - c. For **Load ARN**, enter the ARN for the `IAMBulkLoad` role (this information is located on the IAM console under **Roles**).
7. Choose **Submit**.

The result contains the status of the request. Bulk loads are often long-running processes, so the response doesn't mean that the load has completed, only that it has begun. This status information is updated periodically until it reports that the job is complete.

Note

This information is also available in the blog post, [Moving to the cloud: Migrating Blazegraph to Amazon Neptune](#).

Loading data into Amazon Neptune

There are several different ways to load graph data into Amazon Neptune:

- If you only need to load a relatively small amount of data, you can use queries such as SPARQL `INSERT` statements or Gremlin `addV` and `addE` steps.
- You can take advantage of [Neptune Bulk Loader \(p. 190\)](#) to ingest large amounts of data that reside in external files. The bulk loader command is faster and has less overhead than the query-language commands. It is optimized for large datasets, and supports both RDF (Resource Description Framework) data and Gremlin data.
- You can use AWS Database Migration Service (AWS DMS) to import data from other data stores (see [Using AWS Database Migration Service to load data into Amazon Neptune from a different data store \(p. 225\)](#), and [AWS Database Migration Service User Guide](#)).

Topics

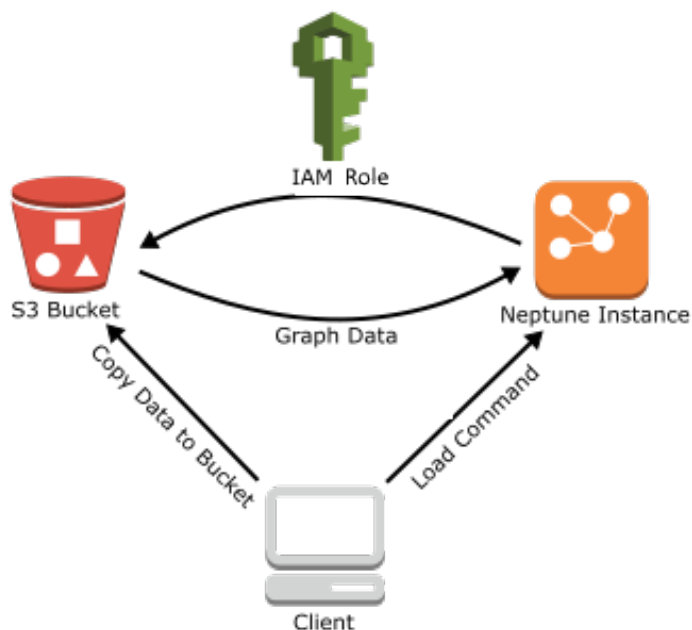
- [Using the Amazon Neptune Bulk Loader to Ingest Data \(p. 190\)](#)
- [Using AWS Database Migration Service to load data into Amazon Neptune from a different data store \(p. 225\)](#)

Using the Amazon Neptune Bulk Loader to Ingest Data

Amazon Neptune provides a `Loader` command for loading data from external files directly into a Neptune DB instance. You can use this command instead of executing a large number of `INSERT` statements, `addV` and `addE` steps, or other API calls.

The Neptune **Loader** command is faster, has less overhead, is optimized for large datasets, and supports both Gremlin data and the RDF (Resource Description Framework) data used by SPARQL.

The following diagram shows an overview of the load process:



Here are the steps of the loading process:

1. Copy the data files to an Amazon Simple Storage Service (Amazon S3) bucket.
2. Create an IAM role with Read and List access to the bucket.
3. Create an Amazon S3 VPC endpoint.
4. Start the Neptune loader by sending a request via HTTP to the Neptune DB instance.
5. The Neptune DB instance assumes the IAM role to load the data from the bucket.

Note

You can load encrypted data from Amazon S3 if it was encrypted using the Amazon S3 SSE-S3 mode. In that case, Neptune can impersonate your credentials and issue `s3:getObject` calls on your behalf. However, Neptune does not currently support loading data encrypted using the SSE-C mode.

The following sections provide instructions for preparing and loading data into Neptune.

Topics

- [Prerequisites: IAM Role and Amazon S3 Access \(p. 191\)](#)
- [Load Data Formats \(p. 194\)](#)
- [Example: Loading Data into a Neptune DB Instance \(p. 204\)](#)
- [Optimizing an Amazon Neptune bulk load \(p. 207\)](#)
- [Neptune Loader Reference \(p. 208\)](#)

Prerequisites: IAM Role and Amazon S3 Access

Loading data from an Amazon Simple Storage Service (Amazon S3) bucket requires an AWS Identity and Access Management (IAM) role that has access to the bucket. Amazon Neptune assumes this role to load the data.

Note

You can load encrypted data from Amazon S3 if it was encrypted using the Amazon S3 SSE-S3 mode. In that case, Neptune is able to impersonate your credentials and issue `s3:getObject` calls on your behalf.

You can also load encrypted data from Amazon S3 that was encrypted using the SSE-KMS mode, as long as your IAM role includes the necessary permissions to access AWS KMS. Without proper AWS KMS permissions, the bulk load operation fails and returns a `LOAD_FAILED` response.

Neptune does not currently support loading Amazon S3 data encrypted using the SSE-C mode.

The following sections show how to create an IAM policy and an IAM role, associate the two, and then attach the role to your Neptune cluster.

Topics

- [Creating an IAM Role to Allow Amazon Neptune to Access Amazon S3 Resources \(p. 192\)](#)
- [Adding the IAM Role to an Amazon Neptune Cluster \(p. 193\)](#)
- [Creating the Amazon S3 VPC Endpoint \(p. 194\)](#)

Note

These instructions require that you have access to the IAM console and permissions to manage IAM roles and policies. For more information, see [Permissions for Working in the AWS Management Console](#) in the *IAM User Guide*.

The Amazon Neptune console requires the user to have the following IAM permissions to attach the role to the Neptune cluster:

```
iam:GetAccountSummary on resource: *
iam:ListAccountAliases on resource: *
iam:PassRole on resource: * with iam:PassedToService restricted to
rds.amazonaws.com
```

Creating an IAM Role to Allow Amazon Neptune to Access Amazon S3 Resources

After creating an IAM policy to allow Neptune to access Amazon S3 resources, create an IAM role and attach the IAM policy to the new IAM role. Start with an Amazon S3 role and modify it to allow Amazon Neptune.

To create an IAM role to allow Amazon Neptune to access AWS services

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Under **AWS service**, choose **S3**.
5. Choose **Next: Permissions**.
6. Use the filter box to filter by the term **S3** and check the box next to **AmazonS3ReadOnlyAccess**.

Note

This policy grants `s3:Get*` and `s3:List*` permissions to all buckets. Later steps restrict access to the role using the trust policy.

The loader only requires `s3:Get*` and `s3:List*` permissions to the bucket you are loading from, so you can also restrict these permissions by the Amazon S3 resource.

If your S3 bucket is encrypted, you need to add `kms:Decrypt` permissions

7. Choose **Next: Review**.

8. Set **Role Name** to a name for your IAM role, for example: `NeptuneLoadFromS3`. You can also add an optional **Role Description** value, such as "Allows Neptune to access Amazon S3 resources on your behalf."
9. Choose **Create Role**.
10. In the navigation pane, choose **Roles**.
11. In the **Search** field, enter the name of the role you created, and choose the role when it appears in the list.
12. On the **Trust Relationships** tab, choose **Edit trust relationship**.
13. In the text field, paste the following trust policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

14. Choose **Update trust policy**.
15. Complete the steps in [Adding the IAM Role to an Amazon Neptune Cluster \(p. 193\)](#).

Adding the IAM Role to an Amazon Neptune Cluster

Use the console to add the IAM role to an Amazon Neptune cluster. This allows any Neptune DB instance in the cluster to assume the role and load from Amazon S3.

Note

The Amazon Neptune console requires the user to have the following IAM permissions to attach the role to the Neptune cluster:

```
iam:GetAccountSummary on resource: *
iam:ListAccountAliases on resource: *
iam:PassRole on resource: * with iam:PassedToService restricted to
  rds.amazonaws.com
```

To add an IAM role to an Amazon Neptune cluster

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Databases**.
3. Choose the radio button next to the cluster that you want to modify.
4. Under **Actions**, choose **Manage IAM roles**.
5. Choose the role you created in the previous section.
6. Choose **Done**.
7. Wait until the IAM role becomes accessible to the cluster before you use it.

Creating the Amazon S3 VPC Endpoint

The Neptune loader requires a VPC endpoint for Amazon S3.

To set up access for Amazon S3

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.
3. Choose **Create Endpoint**.
4. Choose the **Service Name** `com.amazonaws.region.s3`.

Note

If the Region here is incorrect, make sure that the console Region is correct.

5. Choose the VPC that contains your Neptune DB instance.
6. Select the check box next to the route tables that are associated with the subnets related to your cluster. If you only have one route table, you must select that box.
7. Choose **Create Endpoint**.

For information about creating the endpoint, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about the limitations of VPC endpoints, [VPC Endpoints for Amazon S3](#).

Next Steps

Now that you have granted access to the Amazon S3 bucket, you can prepare to load data. For information about supported formats, see [Load Data Formats](#) (p. 194).

Load Data Formats

The Amazon Neptune `Load` API currently requires specific formats for incoming data. The following formats are available, and are listed with their identifiers for the Neptune loader API in parentheses.

- CSV format (`csv`) for property graph / Gremlin
- CSV format (`csv`) for property graph / openCypher
- N-Triples (`ntriples`) format for RDF / SPARQL
- N-Quads (`nquads`) format for RDF / SPARQL
- RDF/XML (`rdxml`) format for RDF / SPARQL
- Turtle (`turtle`) format for RDF / SPARQL

Important

All files must be encoded in UTF-8 format. If a file is not in UTF format, Neptune tries to load it anyway as UTF-8 data.

If your data is not in a supported format, you must convert it before you load it into a Neptune DB instance.

A tool for converting GraphML to the Neptune CSV format is available in the [GraphML2CSV project](#) on [GitHub](#).

Compression Support

Neptune supports compression of *single* files in `gzip` or `bzip2` format. The file name must end in the `.gz` or `.bz2` extension and must contain a single text file encoded in UTF-8 format. Multiple files can

be loaded, but each one must be contained in a separate `.gz` or `.bz2` file (or uncompressed text file). Archive files (for example, `.tar`, `.tar.gz`, and `.tgz`) are not supported.

The following sections describe the formats in more detail.

Topics

- [Property graph data formats for the Neptune bulk loader \(p. 195\)](#)
- [RDF load data formats \(p. 203\)](#)

Property graph data formats for the Neptune bulk loader

There are two types of CSV property-graph load format:

Topics

- [Gremlin load data format \(p. 195\)](#)
- [Load format for openCypher data \(p. 200\)](#)

Gremlin load data format

To load Apache TinkerPop Gremlin data using the CSV format, you must specify the vertices and the edges in separate files.

The loader can load from multiple vertex files and multiple edge files in a single load job.

For each load command, the set of files to be loaded must be in the same folder in the Amazon S3 bucket, and you specify the folder name for the `source` parameter. The file names and file name extensions are not important.

The Amazon Neptune CSV format follows the RFC 4180 CSV specification. For more information, see [Common Format and MIME Type for CSV Files](#) on the Internet Engineering Task Force (IETF) website.

Note

All files must be encoded in UTF-8 format.

Each file has a comma-separated header row. The header row consists of both system column headers and property column headers.

System Column Headers

The required and allowed system column headers are different for vertex files and edge files.

Each system column can appear only once in a header.

All labels are case sensitive.

Vertex headers

- `~id` - **Required**

An ID for the vertex.

- `~label`

A label for the vertex. Multiple label values are allowed, separated by semicolons (`;`).

Edge headers

- **~id - Required**
An ID for the edge.
- **~from - Required**
The vertex ID of the *from* vertex.
- **~to - Required**
The vertex ID of the *to* vertex.
- **~label**
A label for the edge. Edges can only have a single label.

Property Column Headers

You can specify a column (:) for a property by using the following syntax. The type names are not case sensitive. Note, however, that if a colon appears within a property name, it must be escaped by preceding it with a backslash: \:

```
propertyname:type
```

Note

Spaces are not allowed in the column headers, so property names cannot include spaces.

You can specify a column for an array type by adding [] to the type:

```
propertyname:type[ ]
```

Note

Edge properties can only have a single value and will cause an error if an array type is specified or a second value is specified.

The following example shows the column header for a property named *age* of type *Int*.

```
age:Int
```

Every row in the file would be required to have an integer in that position or be left empty.

Arrays of strings are allowed, but strings in an array cannot include the semicolon (;) character unless it is escaped using a backslash (like this: \;).

Specifying the Cardinality of a Column

Starting in [Release 1.0.1.0.200366.0 \(2019-07-26\) \(p. 770\)](#), the column header can be used to specify *cardinality* for the property identified by the column. This allows the bulk loader to honor cardinality similarly to the way Gremlin queries do.

You specify the cardinality of a column like this:

```
propertyname:type(cardinality)
```

The *cardinality* value can be either *single* or *set*. The default is assumed to be *set*, meaning that the column can accept multiple values. In the case of edge files, cardinality is always single and specifying any other cardinality causes the loader to throw an exception.

If the cardinality is `single`, the loader throws an error if a previous value is already present when a value is loaded, or if multiple values are loaded. This behavior can be overridden so that an existing value is replaced when a new value is loaded by using the `updateSingleCardinalityProperties` flag. See [Loader Command \(p. 209\)](#).

It is possible to use a cardinality setting with an array type, although this is not generally necessary. Here are the possible combinations:

- `name:type` – the cardinality is `set`, and the content is single-valued.
- `name:type[]` – the cardinality is `set`, and the content is multi-valued.
- `name:type(single)` – the cardinality is `single`, and the content is single-valued.
- `name:type(set)` – the cardinality is `set`, which is the same as the default, and the content is single-valued.
- `name:type(set)[]` – the cardinality is `set`, and the content is multi-valued.
- `name:type(single)[]` – this is contradictory and causes an error to be thrown.

The following section lists all the available Gremlin data types.

Gremlin Data Types

This is a list of the allowed property types, with a description of each type.

Bool (or Boolean)

Indicates a Boolean field. Allowed values: `false`, `true`

Note

Any value other than `true` will be treated as `false`.

Whole Number Types

Values outside of the defined ranges result in an error.

Type	Range
Byte	-128 to 127
Short	-32768 to 32767
Int	-2 ³¹ to 2 ³¹ -1
Long	-2 ⁶³ to 2 ⁶³ -1

Decimal Number Types

Supports both decimal notation or scientific notation. Also allows symbols such as (+/-) Infinity or NaN. INF is not supported.

Type	Range
Float	32-bit IEEE 754 floating point
Double	64-bit IEEE 754 floating point

Float and double values that are too long are loaded and rounded to the nearest value for 24-bit (float) and 53-bit (double) precision. A midway value is rounded to 0 for the last remaining digit at the bit level.

String

Quotation marks are optional. Commas, newline, and carriage return characters are automatically escaped if they are included in a string surrounded by double quotation marks ("). *Example:* "Hello, World"

To include quotation marks in a quoted string, you can escape the quotation mark by using two in a row: *Example:* "Hello ""World"""

Arrays of strings are allowed, but strings in an array cannot include the semicolon (;) character unless it is escaped using a backslash (like this: \;).

If you want to surround strings in an array with quotation marks, you must surround the whole array with one set of quotation marks. *Example:* "String one; String 2; String 3"

Date

Java date in ISO-8601 format. Supports the following formats: yyyy-MM-dd, yyyy-MM-ddTHH:mm, yyyy-MM-ddTHH:mm:ss, yyyy-MM-ddTHH:mm:ssZ

Gremlin Row Format

Delimiters

Fields in a row are separated by a comma. Records are separated by a newline or a newline followed by a carriage return.

Blank Fields

Blank fields are allowed for non-required columns (such as user-defined properties). A blank field still requires a comma separator. The example in the next section has a blank field in each example vertex.

Vertex IDs

`~id` values must be unique for all vertices in every vertex file. Multiple vertex rows with identical `~id` values are applied to a single vertex in the graph.

Edge IDs

Additionally, `~id` values must be unique for all edges in every edge file. Multiple edge rows with identical `~id` values are applied to the single edge in the graph.

Labels

Labels are case sensitive.

String Values

Quotation marks are optional. Commas, newline, and carriage return characters are automatically escaped if they are included in a string surrounded by double quotation marks (").

CSV Format Specification

The Neptune CSV format follows the RFC 4180 CSV specification, including the following requirements.

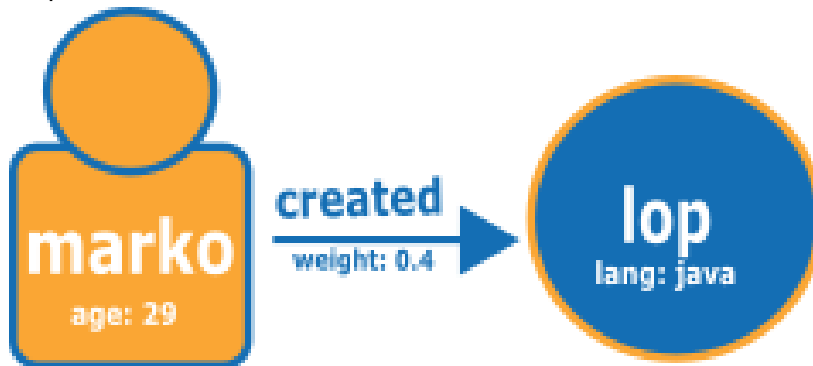
- Both Unix and Windows style line endings are supported (`\n` or `\r\n`).
- Any field can be quoted (using double quotation marks).

- Fields containing a line-break, double-quote, or commas must be quoted. (If they are not, load aborts immediately.)
- A double quotation mark character (") in a field must be represented by two (double) quotation mark characters. For example, a string Hello "World" must be present as "Hello ""World"" in the data.
- Surrounding spaces between delimiters are ignored. If a row is present as value1, value2, they are stored as "value1" and "value2".
- Any other escape characters are stored verbatim. For example, "data1\tdata2" is stored as "data1\tdata2". No further escaping is needed as long as these characters are enclosed within quotation marks.
- Blank fields are allowed. A blank field is considered an empty value.
- Multiple values for a field are specified with a semicolon (;) between values.

For more information, see [Common Format and MIME Type for CSV Files](#) on the Internet Engineering Task Force (IETF) website.

Gremlin Example

The following diagram shows an example of two vertices and an edge taken from the TinkerPop Modern Graph.



The following is the graph in Neptune CSV load format.

Vertex file:

```

~id, name:String, age:Int, lang:String, interests:String[], ~label
v1, "marko", 29, , "sailing;graphs", person
v2, "lop", , "java", , software
  
```

Tabular view of the vertex file:

~id	name:String	age:Int	lang:String	interests:String[]	~label
v1	"marko"	29		["sailing", "graphs"]	person
v2	"lop"		"java"	software	

Edge file:

```

~id, ~from, ~to, ~label, weight:Double
  
```



```
e1, v1, v2, created, 0.4
```

Tabular view of the edge file:

~id	~from	~to	~label	weight:Double
e1	v1	v2	created	0.4

Next Steps

Now that you know more about the loading formats, see [Example: Loading Data into a Neptune DB Instance \(p. 204\)](#).

Load format for openCypher data

To load openCypher data using the openCypher CSV format, you must specify nodes and relationships in separate files. The loader can load from multiple of these node files and relationship files in a single load job.

For each load command, the set of files to be loaded must have the same path prefix in an Amazon Simple Storage Service bucket. You specify that prefix in the source parameter. The actual file names and extensions are not important.

In Amazon Neptune, the openCypher CSV format conforms to the RFC 4180 CSV specification. For more information, see [Common Format and MIME Type for CSV Files \(https://tools.ietf.org/html/rfc4180\)](https://tools.ietf.org/html/rfc4180) on the Internet Engineering Task Force (IETF) website.

Note

These files MUST be encoded in UTF-8 format.

Each file has a comma-separated header row that contains both system column headers and property column headers.

System column headers in openCypher data loading files

A given system column can only appear once in each file. All system column header labels are case-sensitive.

The system column headers that are required and allowed are different for openCypher node load files and relationship load files:

System column headers in node files

- **:ID** – (Required) An ID for the node.

An optional ID space can be added to the node **:ID** column header like this: **:ID(*ID Space*)**. An example is **:ID(movies)**.

When loading relationships that connect the nodes in this file, use the same ID spaces in the relationship files' **:START_ID** and/or **:END_ID** columns.

The node **:ID** column can optionally be stored as a property in the form, ***property name*:ID**. An example is **name:ID**.

Node IDs should be unique across all node files in the current and previous loads. If an ID space is used, node IDs should be unique across all node files that use the same ID space in the current and previous loads.

- **:LABEL** – A label for the node.

Multiple label values are allowed, separated by semicolons (;).

System column headers in relationship files

- **:ID** – An ID for the relationship. This is required when `userProvidedEdgeIds` is `true` (the default), but invalid when `userProvidedEdgeIds` is `false`.

Relationship IDs should be unique across all relationship files in current and previous loads.

- **:START_ID** – (*Required*) The node ID of the node this relationship starts from.

Optionally, an ID space can be associated with the start ID column in the form `:START_ID(ID Space)`. The ID space assigned to the start node ID should match the ID space assigned to the node in its node file.

- **:END_ID** – (*Required*) The node ID of the node this relationship ends at.

Optionally, an ID space can be associated with the end ID column in the form `:END_ID(ID Space)`. The ID space assigned to the end node ID should match the ID space assigned to the node in its node file.

- **:TYPE** – A type for the relationship. Relationships can only have a single type.

Note

See [Loading openCypher data \(p. 214\)](#) for information about how duplicate node or relationship IDs are handled by the bulk load process.

Property column headers in openCypher data loading files

You can specify that a column holds the values for a particular property using a property column header in the following form:

```
propertyname:type
```

Spaces are not allowed in the column headers, and colons (:) are not allowed in property names. Here is an example of a column header for a property named `age` of type `Int`:

```
age: Int
```

The column with `age: Int` as a column header would then have to contain either an integer or an empty value in every row.

Data types in Neptune openCypher data loading files

- **Bool** or **Boolean** – A Boolean field. Allowed values are `true` and `false`.

Any value other than `true` is treated as `false`.

- **Byte** – A whole number in the range `-128` through `127`.
- **Short** – A whole number in the range `-32,768` through `32,767`.
- **Int** – A whole number in the range `-231` through `231 - 1`.
- **Long** – A whole number in the range `-263` through `263 - 1`.
- **Float** – A 32-bit IEEE 754 floating point number. Decimal notation and scientific notation are both supported. `Infinity`, `-Infinity`, and `NaN` are all recognized, but `INF` is not.

Values with too many digits to fit are rounded to the nearest value (a midway value is rounded to 0 for the last remaining digit at the bit level).

- **Double** – A 64-bit IEEE 754 floating point number. Decimal notation and scientific notation are both supported. `Infinity`, `-Infinity`, and `NaN` are all recognized, but `INF` is not.

Values with too many digits to fit are rounded to the nearest value (a midway value is rounded to 0 for the last remaining digit at the bit level).

- **String** – Quotation marks are optional. Comma, newline, and carriage return characters are automatically escaped if they are included in a string that is surrounded by double quotation marks (") like `"Hello, World"`.

You can include quotation marks in a quoted string by using two in a row, like `"Hello ""World"""`.

Arrays of strings are allowed, separated by semicolons (;). To include a literal semicolon in a string array, escape it with a backslash like this: `\;`. If you want each string in a string array to be surrounded by quotation marks, instead surround the entire array in quotation marks, like this: `"String one; String 2; String 3"`.

- **DateTime** – A Java date in one of the following ISO-8601 formats:
 - `yyyy-MM-dd`
 - `yyyy-MM-ddTHH:mm`
 - `yyyy-MM-ddTHH:mm:ss`
 - `yyyy-MM-ddTHH:mm:ssZ`

Auto-cast data types in Neptune openCypher data loading files

Auto-cast data types are provided to load data types not currently supported natively by Neptune. Data in such columns are stored as strings, verbatim with no verification against their intended formats. The following auto-cast data types are allowed:

- **Char** – A Char field. Stored as a string.
- **Date**, **LocalDate**, and **LocalDateTime**, – See [Neo4j Temporal Instant](#)s for a description of the `date`, `localdate`, and `localdatetime` types. The values are loaded verbatim as strings, without validation.
- **Duration** – See the [Neo4j Duration format](#). The values are loaded verbatim as strings, without validation.
- **Point** – A point field, for storing spatial data. See [Neo4j Point format](#). The values are loaded verbatim as strings, without validation.

Example of the openCypher load format

The following diagram taken from the TinkerPop Modern Graph shows an example of two nodes and a relationship:



The following is the graph in the normal Neptune openCypher load format.

Node file:

```
:ID, name:String, age:Int, lang:String, :LABEL
v1, "marko", 29, , person
v2, "lop", , "java", software
```

Relationship file:

```
:ID, :START_ID, :END_ID, :TYPE, weight:Double
e1, v1, v2, created, 0.4
```

Alternatively, you could use ID spaces and ID as a property, as follows:

First node file:

```
name:ID(person), age:Int, lang:String, :LABEL
"marko", 29, , person
```

Second node file:

```
name:ID(software), age:Int, lang:String, :LABEL
"lop", , "java", software
```

Relationship file:

```
:ID, :START_ID, :END_ID, :TYPE, weight:Double
e1, "marko", "lop", created, 0.4
```

RDF load data formats

To load Resource Description Framework (RDF) data, you can use one of the following standard formats as specified by the World Wide Web Consortium (W3C):

- N-Triples (`ntriples`) from the specification at <https://www.w3.org/TR/n-triples/>
- N-Quads (`nquads`) from the specification at <https://www.w3.org/TR/n-quads/>
- RDF/XML (`rdfxml`) from the specification at <https://www.w3.org/TR/rdf-syntax-grammar/>
- Turtle (`turtle`) from the specification at <https://www.w3.org/TR/turtle/>

Important

All files must be encoded in UTF-8 format.

For N-Quads and N-triples data that includes Unicode characters, `\uxxxxx` escape sequences are supported. However, Neptune does not support normalization. If a value is present that requires normalization, it will not match byte-to-byte during querying. For more information about normalization, see the [Normalization](#) page on Unicode.org.

Next Steps

Now that you know more about the loading formats, see [Example: Loading Data into a Neptune DB Instance \(p. 204\)](#).

Example: Loading Data into a Neptune DB Instance

This example shows how to load data into Amazon Neptune. Unless stated otherwise, you must follow these steps from an Amazon Elastic Compute Cloud (Amazon EC2) instance in the same Amazon Virtual Private Cloud (VPC) as your Neptune DB instance.

Prerequisites for the Data Loading Example

Before you begin, you must have the following:

- A Neptune DB instance.

For information about launching a Neptune DB instance, see [Creating a New Neptune DB Cluster Using AWS CloudFormation or Manually](#) (p. 166).

- An Amazon Simple Storage Service (Amazon S3) bucket to put the data files in.

You can use an existing bucket. If you don't have an S3 bucket, see [Create a Bucket](#) in the *Amazon S3 Getting Started Guide*.

- Graph data to load, in one of the formats supported by the Neptune loader:

If you are using Gremlin to query your graph, Neptune can load data in a comma-separated-values (CSV) format, as described in [Gremlin load data format](#) (p. 195).

If you are using SPARQL, Neptune can load data in a number of RDF formats, as described in [RDF load data formats](#) (p. 203).

- An IAM role for the Neptune DB instance to assume that has an IAM policy that allows access to the data files in the S3 bucket. The policy must grant Read and List permissions.

For information about creating a role that has access to Amazon S3 and then associating it with a Neptune cluster, see [Prerequisites: IAM Role and Amazon S3 Access](#) (p. 191).

Note

The Neptune Load API needs read access to the data files only. The IAM policy doesn't need to allow write access or access to the entire bucket.

- An Amazon S3 VPC endpoint. For more information, see the [Creating an Amazon S3 VPC Endpoint](#) (p. 204) section.

Creating an Amazon S3 VPC Endpoint

The Neptune loader requires a VPC endpoint for Amazon S3.

To set up access for Amazon S3

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the left navigation pane, choose **Endpoints**.
3. Choose **Create Endpoint**.
4. Choose the **Service Name** `com.amazonaws.region.s3`.

Note

If the Region here is incorrect, make sure that the console Region is correct.

5. Choose the VPC that contains your Neptune DB instance.
6. Select the check box next to the route tables that are associated with the subnets related to your cluster. If you only have one route table, you must select that box.

7. Choose **Create Endpoint**.

For information about creating the endpoint, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about the limitations of VPC endpoints, see [VPC Endpoints for Amazon S3](#).

To load data into a Neptune DB instance

1. Copy the data files to an Amazon S3 bucket. The S3 bucket must be in the same AWS Region as the cluster that loads the data.

You can use the following AWS CLI command to copy the files to the bucket.

Note

This command does not need to be run from the Amazon EC2 instance.

```
aws s3 cp data-file-name s3://bucket-name/object-key-name
```

Note

In Amazon S3, an **object key name** is the entire path of a file, including the file name.

Example: In the command `aws s3 cp datafile.txt s3://examplebucket/mydirectory/datafile.txt`, the object key name is `mydirectory/datafile.txt`.

Alternatively, you can use the AWS Management Console to upload files to the S3 bucket. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>, and choose a bucket. In the upper-left corner, choose **Upload** to upload files.

2. From a command line window, enter the following to run the Neptune loader. Replace the values for the endpoint, Amazon S3 path, format, and access keys.

The `format` parameter can be any of the following values: `csv` (Gremlin), `ntriples`, `nquads`, `turtle`, and `rdfxml` (RDF). For information about the other parameters, see [Neptune Loader Command \(p. 209\)](#).

For information about finding the hostname of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

The `Region` parameter must match the Region of the cluster and the S3 bucket.

Amazon Neptune is available in the following AWS Regions:

- US East (N. Virginia): `us-east-1`
- US East (Ohio): `us-east-2`
- US West (N. California): `us-west-1`
- US West (Oregon): `us-west-2`
- Canada (Central): `ca-central-1`
- South America (São Paulo): `sa-east-1`
- Europe (Stockholm): `eu-north-1`
- Europe (Ireland): `eu-west-1`
- Europe (London): `eu-west-2`
- Europe (Paris): `eu-west-3`
- Europe (Frankfurt): `eu-central-1`
- Middle East (Bahrain): `me-south-1`
- Asia Pacific (Hong Kong): `ap-east-1`
- Asia Pacific (Tokyo): `ap-northeast-1`
- Asia Pacific (Seoul): `ap-northeast-2`

- Asia Pacific (Singapore): `ap-southeast-1`
- Asia Pacific (Sydney): `ap-southeast-2`
- Asia Pacific (Mumbai): `ap-south-1`
- China (Beijing): `cn-north-1`
- China (Ningxia): `cn-northwest-1`
- AWS GovCloud (US-West): `us-gov-west-1`
- AWS GovCloud (US-East): `us-gov-east-1`

```
curl -X POST \  
  -H 'Content-Type: application/json' \  
  https://your-neptune-endpoint:port/loader -d '  
  {  
    "source" : "s3://bucket-name/object-key-name",  
    "format" : "format",  
    "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",  
    "region" : "region",  
    "failOnError" : "FALSE",  
    "parallelism" : "MEDIUM",  
    "updateSingleCardinalityProperties" : "FALSE",  
    "queueRequest" : "TRUE",  
    "dependencies" : [ "load_A_id", "load_B_id" ]  
  }'
```

For information about creating and associating an IAM role with a Neptune cluster, see [Prerequisites: IAM Role and Amazon S3 Access \(p. 191\)](#).

Note

See [Neptune Loader Request Parameters \(p. 210\)](#) for detailed information about load request parameters. In brief:
The source parameter accepts an Amazon S3 URI that points to either a single file or a folder. If you specify a folder, Neptune loads every data file in the folder. The folder can contain multiple vertex files and multiple edge files. The URI can be in any of the following formats.

- `s3://bucket_name/object-key-name`
- `https://s3.amazonaws.com/bucket_name/object-key-name`
- `https://s3-us-east-1.amazonaws.com/bucket_name/object-key-name`

The format parameter can be one of the following:

- CSV format (`csv`) for property graph / Gremlin
- N-Triples (`ntriples`) format for RDF / SPARQL
- N-Quads (`nquads`) format for RDF / SPARQL
- RDF/XML (`rdxml`) format for RDF / SPARQL
- Turtle (`turtle`) format for RDF / SPARQL

The optional `parallelism` parameter lets you restrict the number of threads used in the bulk load process. It can be set to `LOW`, `MEDIUM`, `HIGH`, or `OVERSUBSCRIBE`. When `updateSingleCardinalityProperties` is set to `"FALSE"`, the loader returns an error if more than one value is provided in a source file being loaded for an edge or single-cardinality vertex property. Setting `queueRequest` to `"TRUE"` causes the load request to be placed in a queue if there is already a load job running.

The `dependencies` parameter makes execution of the load request contingent on the successful completion of one or more load jobs that have already been placed in the queue.

3. The Neptune loader returns a `job id` that allows you to check the status or cancel the loading process; for example:

```
{
  "status" : "200 OK",
  "payload" : {
    "loadId" : "ef478d76-d9da-4d94-8ff1-08d9d4863aa5"
  }
}
```

4. Enter the following to get the status of the load with the `loadId` from **Step 3**:

```
curl -G 'https://your-neptune-endpoint:port/loader/ef478d76-
d9da-4d94-8ff1-08d9d4863aa5'
```

If the status of the load lists an error, you can request more detailed status and a list of the errors. For more information and examples, see [Neptune Loader Get-Status API \(p. 216\)](#).

5. (Optional) Cancel the Load job.

Enter the following to `Delete` the loader job with the `job id` from **Step 3**:

```
curl -X DELETE 'https://your-neptune-endpoint:port/loader/ef478d76-
d9da-4d94-8ff1-08d9d4863aa5'
```

The `DELETE` command returns the HTTP code `200 OK` upon successful cancellation.

The data from files from the load job that has finished loading is not rolled back. The data remains in the Neptune DB instance.

Optimizing an Amazon Neptune bulk load

Use the following strategies to keep the load time to a minimum for a Neptune bulk load:

- **Clean your data:**

- Be sure to convert your data into a [supported data format \(p. 194\)](#) before loading.
- Remove any duplicates or known errors.
- Reduce the number of unique predicates (such as properties of edges and vertices) as much as you can.

- **Optimize your files:**

- If you load large files such CSV files from an Amazon S3 bucket, the loader can break up the large files and manage concurrency for you.
- If you load multiple files from an Amazon S3 folder, the loader automatically loads vertex files first, then edge files afterwards.
- Breaking up very large files into a few smaller ones can let the loader process them in parallel.
- Compressing the files reduces transfer times. The loader supports `gzip` compression of source files.

- **Check your loader settings:**

- If you don't need to perform any other operations during the load, use the [OVERSUBSCRIBE parallelism \(p. 209\)](#) parameter. This parameter setting causes the bulk loader to use all available CPU resources when it runs. It generally takes 60%-70% of CPU capacity to keep the operation running as fast as possible.

Note

When `parallelism` is set to `OVERSUBSCRIBE` or `HIGH` (the default setting), there is the risk when loading openCypher data that threads may encounter a race condition and deadlock, resulting in a `LOAD_DATA_DEADLOCK` error. In this case, set `parallelism` to a lower setting and retry the load.

- If your load job will include multiple load requests, use the `queueRequest` parameter. Setting `queueRequest` to `TRUE` lets Neptune queue up your requests so you don't have to wait for one to finish before issuing another.
- If your load requests are being queued, you can set up levels of dependency using the `dependencies` parameter, so that the failure of one job causes dependent jobs to fail. This can prevent inconsistencies in the loaded data.
- If a load job is going to involve updating previously loaded values, be sure to set the `updateSingleCardinalityProperties` parameter to `TRUE`. If you don't, the loader will treat an attempt to update an existing single-cardinality value as an error. For Gremlin data, cardinality is also specified in property column headers (see [Property Column Headers \(p. 196\)](#)).
- You can use the `failOnError` parameter to determine whether bulk load operations should fail or continue when an error is encountered. Also, you can use the `mode` parameter to be sure that a load job resumes loading from the point where a previous job failed rather than reloading data that had already been loaded.
- **Scale up** – Set the writer instance of your DB cluster to the maximum size before bulk loading. Note that if you do this, you must either scale up any read-replica instances in the DB cluster as well, or remove them until you have finished loading the data.

When your bulk load is complete, be sure to scale the writer instance down again.

See [Request Parameters \(p. 210\)](#) for more details about setting loader request parameters.

Neptune Loader Reference

This section describes the `Loader` APIs for Amazon Neptune that are available from the HTTP endpoint of a Neptune DB instance.

Note

See [Neptune Loader Error and Feed Messages \(p. 716\)](#) for a list of the error and feed messages returned by the loader in case of errors.

Contents

- [Neptune Loader Command \(p. 209\)](#)
 - [Neptune Loader Request Syntax \(p. 209\)](#)
 - [Neptune Loader Request Parameters \(p. 210\)](#)
 - [Special considerations for loading openCypher data \(p. 214\)](#)
 - [Neptune Loader Response Syntax \(p. 215\)](#)
 - [Neptune Loader Errors \(p. 215\)](#)
 - [Neptune Loader Examples \(p. 216\)](#)
- [Neptune Loader Get-Status API \(p. 216\)](#)
 - [Neptune Loader Get-Status requests \(p. 217\)](#)
 - [Loader Get-Status request syntax \(p. 217\)](#)
 - [Neptune Loader Get-Status request parameters \(p. 217\)](#)
 - [Neptune Loader Get-Status Responses \(p. 218\)](#)
 - [Neptune Loader Get-Status Response JSON layout \(p. 218\)](#)
 - [Neptune Loader Get-Status overallStatus and failedFeeds response objects \(p. 219\)](#)

- [Neptune Loader Get-Status errors response object \(p. 220\)](#)
- [Neptune Loader Get-Status errorLogs response object \(p. 220\)](#)
- [Neptune Loader Get-Status Examples \(p. 221\)](#)
 - [Example request for load status \(p. 221\)](#)
 - [Example request for loadIds \(p. 222\)](#)
 - [Example request for detailed status \(p. 222\)](#)
- [Neptune Loader Get-Status errorLogs examples \(p. 223\)](#)
 - [Example detailed status response when errors occurred \(p. 223\)](#)
 - [Example of a Data prefetch task interrupted error \(p. 224\)](#)
- [Neptune Loader Cancel Job \(p. 224\)](#)
 - [Cancel Job request syntax \(p. 224\)](#)
 - [Cancel Job Request Parameters \(p. 225\)](#)
 - [Cancel Job Response Syntax \(p. 225\)](#)
 - [Cancel Job Errors \(p. 225\)](#)
 - [Cancel Job Error Messages \(p. 225\)](#)
 - [Cancel Job Examples \(p. 225\)](#)

Neptune Loader Command

Loads data from an Amazon S3 bucket into a Neptune DB instance.

To load data, you must send an HTTP POST request to the `https://your-neptune-endpoint:port/loader` endpoint. The parameters for the loader request can be sent in the POST body or as URL-encoded parameters.

Important

The MIME type must be `application/json`.

The S3 bucket must be in the same AWS Region as the cluster.

Note

You can load encrypted data from Amazon S3 if it was encrypted using the Amazon S3 SSE-S3 mode. In that case, Neptune is able to impersonate your credentials and issue `s3:getObject` calls on your behalf.

You can also load encrypted data from Amazon S3 that was encrypted using the SSE-KMS mode, as long as your IAM role includes the necessary permissions to access AWS KMS. Without proper AWS KMS permissions, the bulk load operation fails and returns a `LOAD_FAILED` response.

Neptune does not currently support loading Amazon S3 data encrypted using the SSE-C mode.

You don't have to wait for one load job to finish before you start another one. Neptune can queue up as many as 64 jobs requests at a time, provided that their `queueRequest` parameters are all set to `"TRUE"`. If you don't want a load job to be queued up, on the other hand, you can set its `queueRequest` parameter to `"FALSE"` (the default), so that the load job will fail if another one is already in progress.

You can use the `dependencies` parameter to queue up a job that must only be run after specified previous jobs in the queue have completed successfully. If you do that and any of those specified jobs fails, your job will not be run and its status will be set to `LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED`.

Neptune Loader Request Syntax

```
{
  "source" : "string",
  "format" : "string",
```

```
"iamRoleArn" : "string",
"mode": "NEW|RESUME|AUTO",
"region" : "us-east-1",
"failOnError" : "string",
"parallelism" : "string",
"parserConfiguration" : {
  "baseUri" : "http://base-uri-string",
  "namedGraphUri" : "http://named-graph-string"
},
"updateSingleCardinalityProperties" : "string",
"queueRequest" : "TRUE",
"dependencies" : [\"Load_A_id\", \"Load_B_id\"]
}
```

Neptune Loader Request Parameters

- **source** – An Amazon S3 URI.

The **SOURCE** parameter accepts an Amazon S3 URI that identifies a single file, multiple files, a folder, or multiple folders. Neptune loads every data file in any folder that is specified.

The URI can be in any of the following formats.

- `s3://bucket_name/object-key-name`
- `https://s3.amazonaws.com/bucket_name/object-key-name`
- `https://s3.us-east-1.amazonaws.com/bucket_name/object-key-name`

The `object-key-name` element of the URI is equivalent to the `prefix` parameter in an Amazon S3 [ListObjects](#) API call. It identifies all the objects in the specified Amazon S3 bucket whose names begin with that prefix. That can be a single file or folder, or multiple files and/or folders.

The specified folder or folders can contain multiple vertex files and multiple edge files.

- **format** – The format of the data. For more information about data formats for the Neptune Loader command, see [Using the Amazon Neptune Bulk Loader to Ingest Data \(p. 190\)](#).

Allowed values: `csv` for Gremlin data, `opencypher` for openCypher CSV data, or `ntriples`, `nquads`, `rdfxml`, or `turtle` for RDF data.

- **iamRoleArn** – The Amazon Resource Name (ARN) for an IAM role to be assumed by the Neptune DB instance for access to the S3 bucket. For information about creating a role that has access to Amazon S3 and then associating it with a Neptune cluster, see [Prerequisites: IAM Role and Amazon S3 Access \(p. 191\)](#).
- **region** – The region parameter must match the AWS Region of the cluster and the S3 bucket.

Amazon Neptune is available in the following Regions:

- US East (N. Virginia): `us-east-1`
- US East (Ohio): `us-east-2`
- US West (N. California): `us-west-1`
- US West (Oregon): `us-west-2`
- Canada (Central): `ca-central-1`
- South America (São Paulo): `sa-east-1`
- Europe (Stockholm): `eu-north-1`
- Europe (Ireland): `eu-west-1`
- Europe (London): `eu-west-2`
- Europe (Paris): `eu-west-3`
- Europe (Frankfurt): `eu-central-1`
- Middle East (Bahrain): `me-south-1`

- Asia Pacific (Hong Kong): `ap-east-1`
- Asia Pacific (Tokyo): `ap-northeast-1`
- Asia Pacific (Seoul): `ap-northeast-2`
- Asia Pacific (Singapore): `ap-southeast-1`
- Asia Pacific (Sydney): `ap-southeast-2`
- Asia Pacific (Mumbai): `ap-south-1`
- China (Beijing): `cn-north-1`
- China (Ningxia): `cn-northwest-1`
- AWS GovCloud (US-West): `us-gov-west-1`
- AWS GovCloud (US-East): `us-gov-east-1`
- **mode** – The load job mode.

Allowed values: RESUME, NEW, AUTO.

Default value: AUTO

- **RESUME** – In RESUME mode, the loader looks for a previous load from this source, and if it finds one, resumes that load job. If no previous load job is found, the loader stops.

The loader avoids reloading files that were successfully loaded in a previous job. It only tries to process failed files. If you dropped previously loaded data from your Neptune cluster, that data is not reloaded in this mode. If a previous load job loaded all files from the same source successfully, nothing is reloaded, and the loader returns success.

- **NEW** – In NEW mode, the loader creates a new load request regardless of any previous loads. You can use this mode to reload all the data from a source after dropping previously loaded data from your Neptune cluster, or to load new data available at the same source.
- **AUTO** – In AUTO mode, the loader looks for a previous load job from the same source, and if it finds one, resumes that job, just as in RESUME mode.

If the loader doesn't find a previous load job from the same source, it loads all data from the source, just as in NEW mode.

- **failOnError** – A flag to toggle a complete stop on an error.

Allowed values: "TRUE", "FALSE".

Default value: "TRUE".

When this parameter is set to "FALSE", the loader tries to load all the data in the location specified, skipping any entries with errors.

When this parameter is set to "TRUE", the loader stops as soon as it encounters an error. Data loaded up to that point persists.

- **parallelism** – This is an optional parameter that can be set to reduce the number of threads used by the bulk load process.

Allowed values:

- **LOW** – The number of threads used is the number of available vCPUs divided by 8.
- **MEDIUM** – The number of threads used is the number of available vCPUs divided by 2.
- **HIGH** – The number of threads used is the same as the number of available vCPUs.
- **OVERSUBSCRIBE** – The number of threads used is the number of available vCPUs multiplied by 2. If this value is used, the bulk loader takes up all available resources.

This does not mean, however, that the `OVERSUBSCRIBE` setting results in 100% CPU utilization. Because the load operation is I/O bound, the highest CPU utilization to expect is in the 60% to 70% range.

Default value: HIGH

The `parallelism` setting can sometimes result in a deadlock between threads when loading openCypher data. When this happens, Neptune returns the `LOAD_DATA_DEADLOCK` error. You can generally fix the issue by setting `parallelism` to a lower setting and retrying the load command.

- **parserConfiguration** – An optional object with additional parser configuration values. Each of the child parameters is also optional:

Name	Example Value	Description
<code>namedGraphUri</code>	<code>http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph</code>	The default graph for all RDF formats when no graph is specified (for non-quads formats and NQUAD entries with no graph). The default is <code>http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph</code>
<code>baseUri</code>	<code>http://aws.amazon.com/neptune/default</code>	The base URI for RDF/XML and Turtle formats. The default is <code>http://aws.amazon.com/neptune/default</code> .
<code>allowEmptyStrings</code>	<code>true</code>	Gremlin users need to be able to pass empty string values("") as node and edge properties when loading CSV data. If <code>allowEmptyStrings</code> is set to <code>false</code> (the default), such empty strings are treated as nulls and are not loaded. If <code>allowEmptyStrings</code> is set to <code>true</code> , the loader treats empty strings as valid property values and loads them accordingly.

For more information, see [SPARQL Default Graph and Named Graphs \(p. 57\)](#).

- **updateSingleCardinalityProperties** – This is an optional parameter that controls how the bulk loader treats a new value for single-cardinality vertex or edge properties. This is not supported for loading openCypher data (see [Loading openCypher data \(p. 214\)](#)).

Allowed values: "TRUE", "FALSE".

Default value: "FALSE".

By default, or when `updateSingleCardinalityProperties` is explicitly set to "FALSE", the loader treats a new value as an error, because it violates single cardinality.

When `updateSingleCardinalityProperties` is set to "TRUE", on the other hand, the bulk loader replaces the existing value with the new one. If multiple edge or single-cardinality vertex property values are provided in the source file(s) being loaded, the final value at the end of the bulk load could be any one of those new values. The loader only guarantees that the existing value has been replaced by one of the new ones.

- **queueRequest** – This is an optional flag parameter that indicates whether the load request can be queued up or not.

You don't have to wait for one load job to complete before issuing the next one, because Neptune can queue up as many as 64 jobs at a time, provided that their `queueRequest` parameters are all set to "TRUE".

If the `queueRequest` parameter is omitted or set to "FALSE", the load request will fail if another load job is already running.

Allowed values: "TRUE", "FALSE".

Default value: "FALSE".

- **dependencies** – This is an optional parameter that can make a queued load request contingent on the successful completion of one or more previous jobs in the queue.

Neptune can queue up as many as 64 load requests at a time, if their `queueRequest` parameters are set to "TRUE". The `dependencies` parameter lets you make execution of such a queued request dependent on the successful completion of one or more specified previous requests in the queue.

For example, if load `Job-A` and `Job-B` are independent of each other, but load `Job-C` needs `Job-A` and `Job-B` to be finished before it begins, proceed as follows:

1. Submit `load-job-A` and `load-job-B` one after another in any order, and save their load-ids.
2. Submit `load-job-C` with the load-ids of the two jobs in its `dependencies` field:

```
"dependencies" : [\"job_A_load_id\", \"job_B_load_id\"]
```

Because of the `dependencies` parameter, the bulk loader will not start `Job-C` until `Job-A` and `Job-B` have completed successfully. If either one of them fails, `Job-C` will not be executed, and its status will be set to `LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED`.

You can set up multiple levels of dependency in this way, so that the failure of one job will cause all requests that are directly or indirectly dependent on it to be cancelled.

- **userProvidedEdgeIds** – This parameter is required only when loading openCypher data that contains relationship IDs. It must be included and set to `True` when openCypher relationship IDs are explicitly provided in the load data (recommended).

When `userProvidedEdgeIds` is present and set to `True`, an `:ID` column must be present in every relationship file in the load.

When `userProvidedEdgeIds` is absent or set to `False`, relationship files in the load **must not** contain an `:ID` column. Instead, the Neptune loader automatically generates an ID for each relationship.

It's useful to provide relationship IDs explicitly so that the loader can resume loading after error in the CSV data have been fixed, without having to reload any relationships that have already been loaded. If relationship IDs have not been explicitly assigned, the loader cannot resume a failed load if any relationship file has had to be corrected, and must instead reload all the relationships.

- **accessKey** – **[deprecated]** An access key ID of an IAM role with access to the S3 bucket and data files.

The `iamRoleArn` parameter is recommended instead. For information about creating a role that has access to Amazon S3 and then associating it with a Neptune cluster, see [Prerequisites: IAM Role and Amazon S3 Access](#) (p. 191).

For more information, see [Access keys \(access key ID and secret access key\)](#).

- `secretKey` – **[deprecated]** The `iamRoleArn` parameter is recommended instead. For information about creating a role that has access to Amazon S3 and then associating it with a Neptune cluster, see [Prerequisites: IAM Role and Amazon S3 Access](#) (p. 191).

For more information, see [Access keys \(access key ID and secret access key\)](#).

Special considerations for loading openCypher data

- When loading openCypher data in CSV format, the format parameter must be set to `opencypher`.
- The `updateSingleCardinalityProperties` parameter is not supported for openCypher loads because all openCypher properties have single cardinality. The openCypher load format does not support arrays, and if an ID value appears more than once, it is treated as a duplicate or an insertion error (see below).
- The Neptune loader handles duplicates that it encounters in openCypher data as follows:
 - If the loader encounters multiple rows with the same node or relationship ID, only one of them is loaded, and which particular one is non-deterministic.
 - The loader never updates property values of an existing node or relationship in the database if it encounters load data having the ID of the existing node or relationship. Instead, when the loader encounters a row containing the ID of an existing node or relationship, it simply ignores that row.
- Although you don't have to assign IDs to relationships, it is usually a good idea (see the `userProvidedEdgeIds` parameter above). Without explicit relationship IDs, the loader must reload all relationships in case of an error in a relationship file, rather than resuming the load from where it failed.

Also, if the load data doesn't contain explicit relationship IDs, the loader has no way of detecting duplicate relationships.

Here is an example of an openCypher load command:

```
curl -X POST https://your-neptune-endpoint:port/loader \  
-H 'Content-Type: application/json' \  
-d '  
  {  
    "source" : "s3://bucket-name/object-key-name",  
    "format" : "opencypher",  
    "userProvidedEdgeIds" : "TRUE",  
    "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",  
    "region" : "region",  
    "failOnError" : "FALSE",  
    "parallelism" : "MEDIUM",  
  }'
```

The loader response is the same as normal. For example:

```
{  
  "status" : "200 OK",  
  "payload" : {  
    "loadId" : "guid_as_string"  
  }  
}
```

```
}
```

Neptune Loader Response Syntax

```
{  
  "status" : "200 OK",  
  "payload" : {  
    "loadId" : "guid_as_string"  
  }  
}
```

200 OK

Successfully started load job returns a 200 code.

Neptune Loader Errors

When an error occurs, a JSON object is returned in the BODY of the response. The message object contains a description of the error.

Error Categories

- **Error 400** – Syntax errors return an HTTP 400 bad request error. The message describes the error.
- **Error 500** – A valid request that cannot be processed returns an HTTP 500 internal server error. The message describes the error.

The following are possible error messages from the loader with a description of the error.

Loader Error Messages

- Couldn't find the AWS credential for `iam_role_arn` (HTTP 400)

The credentials were not found. Verify the supplied credentials against the IAM console or AWS CLI output.

- S3 bucket not found for source (HTTP 400)

The S3 bucket does not exist. Check the name of the bucket.

- The source `source-uri` does not exist/not reachable (HTTP 400)

No matching files were found in the S3 bucket.

- Unable to connect to S3 endpoint. Provided source = `source-uri` and region = `aws-region` (HTTP 500)

Unable to connect to Amazon S3. Region must match the cluster Region. Ensure that you have a VPC endpoint. For information about creating a VPC endpoint, see [Creating an Amazon S3 VPC Endpoint \(p. 204\)](#).

- Bucket is not in provided Region (`aws-region`) (HTTP 400)

The bucket must be in the same AWS Region as your Neptune DB instance.

- Unable to perform S3 list operation (HTTP 400)

The IAM user or role provided does not have `List` permissions on the bucket or the folder. Check the policy or the access control list (ACL) on the bucket.

- Start new load operation not permitted on a read replica instance (HTTP 405)

Loading is a write operation. Retry load on the read/write cluster endpoint.

- Failed to start load because of unknown error from S3 (HTTP 500)

Amazon S3 returned an unknown error. Contact [AWS Support](#).

- Invalid S3 access key (HTTP 400)

Access key is invalid. Check the provided credentials.

- Invalid S3 secret key (HTTP 400)

Secret key is invalid. Check the provided credentials.

- Max concurrent load limit breached (HTTP 400)

If a load request is submitted without "queueRequest" : "TRUE", and a load job is currently running, the request will fail with this error.

- Failed to start new load for the source "*source name*". Max load task queue size limit breached. Limit is 64 (HTTP 400)

Neptune supports queuing up as many as 64 loader jobs at a time. If an additional load request is submitted to the queue when it already contains 64 jobs, the request fails with this message.

Neptune Loader Examples

Example Request

The following is a request sent via HTTP POST using the curl command. It loads a file in the Neptune CSV format. For more information, see [Gremlin load data format \(p. 195\)](#).

```
curl -X POST \  
  -H 'Content-Type: application/json' \  
  https://your-neptune-endpoint:port/loader -d '  
  {  
    "source" : "s3://bucket-name/object-key-name",  
    "format" : "csv",  
    "iamRoleArn" : "ARN for the IAM role you are using",  
    "region" : "region",  
    "failOnError" : "FALSE",  
    "parallelism" : "MEDIUM",  
    "updateSingleCardinalityProperties" : "FALSE",  
    "queueRequest" : "FALSE"  
  }'
```

Example Response

```
{  
  "status" : "200 OK",  
  "payload" : {  
    "loadId" : "ef478d76-d9da-4d94-8ff1-08d9d4863aa5"  
  }  
}
```

Neptune Loader Get-Status API

Gets the status of a loader job.

To get load status, you must send an HTTP GET request to the `https://your-neptune-endpoint:port/loader` endpoint. To get the status for a particular load request, you must include the `loadId` as a URL parameter, or append the `loadId` to the URL path.

Neptune only keeps track of the most recent 1,024 bulk load jobs, and only stores the last 10,000 error details per job.

See [Neptune Loader Error and Feed Messages \(p. 716\)](#) for a list of the error and feed messages returned by the loader in case of errors.

Contents

- [Neptune Loader Get-Status requests \(p. 217\)](#)
 - [Loader Get-Status request syntax \(p. 217\)](#)
 - [Neptune Loader Get-Status request parameters \(p. 217\)](#)
- [Neptune Loader Get-Status Responses \(p. 218\)](#)
 - [Neptune Loader Get-Status Response JSON layout \(p. 218\)](#)
 - [Neptune Loader Get-Status overallStatus and failedFeeds response objects \(p. 219\)](#)
 - [Neptune Loader Get-Status errors response object \(p. 220\)](#)
 - [Neptune Loader Get-Status errorLogs response object \(p. 220\)](#)
- [Neptune Loader Get-Status Examples \(p. 221\)](#)
 - [Example request for load status \(p. 221\)](#)
 - [Example request for loadIds \(p. 222\)](#)
 - [Example request for detailed status \(p. 222\)](#)
- [Neptune Loader Get-Status errorLogs examples \(p. 223\)](#)
 - [Example detailed status response when errors occurred \(p. 223\)](#)
 - [Example of a Data prefetch task interrupted error \(p. 224\)](#)

Neptune Loader Get-Status requests

Loader Get-Status request syntax

```
GET https://your-neptune-endpoint:port/loader?loadId=loadId
```

```
GET https://your-neptune-endpoint:port/loader/loadId
```

```
GET https://your-neptune-endpoint:port/loader
```

Neptune Loader Get-Status request parameters

- **loadId** – The ID of the load job. If you do not specify a `loadId`, a list of load IDs is returned.
- **details** – Include details beyond overall status.

Allowed values: TRUE, FALSE.

Default value: FALSE.

- **errors** – Include the list of errors.

Allowed values: TRUE, FALSE.

Default value: FALSE.

The list of errors is paged. The `page` and `errorsPerPage` parameters allow you to page through all the errors.

- **page** – The error page number. Only valid with the `errors` parameter set to TRUE.

Allowed values: Positive integers.

Default value: 1.

- **errorsPerPage** – The number of errors per each page. Only valid with the `errors` parameter set to `TRUE`.

Allowed values: Positive integers.

Default value: 10.

- **limit** – The number of load ids to list. Only valid when requesting a list of load IDs by sending a `GET` request with no `loadId` specified.

Allowed values: Positive integers from 1 through 100.

Default value: 100.

- **includeQueuedLoads** – An optional parameter that can be used to exclude the load IDs of queued load requests when a list of load IDs is requested.

Note

This parameter is available starting in [Neptune engine release 1.0.3.0 \(p. 743\)](#).

By default, the load IDs of all load jobs with status `LOAD_IN_QUEUE` are included in such a list. They appear before the load IDs of other jobs, sorted by the time they were added to the queue from most recent to earliest.

Allowed values: `TRUE`, `FALSE`.

Default value: `TRUE`.

Neptune Loader Get-Status Responses

Neptune Loader Get-Status Response JSON layout

The general layout of a loader status response is as follows:

```
{
  "status" : "200 OK",
  "payload" : {
    "feedCount" : [
      {
        "LOAD_FAILED" : number
      }
    ],
    "overallStatus" : {
      "fullUri" : "s3://bucket/key",
      "runNumber" : number,
      "retryNumber" : number,
      "status" : "string",
      "totalTimeSpent" : number,
      "startTime" : number,
      "totalRecords" : number,
      "totalDuplicates" : number,
      "parsingErrors" : number,
      "datatypeMismatchErrors" : number,
      "insertErrors" : number,
    },
    "failedFeeds" : [
      {
        "fullUri" : "s3://bucket/key",
        "runNumber" : number,
      }
    ]
  }
}
```

```
        "retryNumber" : number,  
        "status" : "string",  
        "totalTimeSpent" : number,  
        "startTime" : number,  
        "totalRecords" : number,  
        "totalDuplicates" : number,  
        "parsingErrors" : number,  
        "datatypeMismatchErrors" : number,  
        "insertErrors" : number,  
    }  
],  
"errors" : {  
    "startIndex" : number,  
    "endIndex" : number,  
    "loadId" : "string",  
    "errorLogs" : [ ]  
}  
}
```

Neptune Loader Get-Status overallStatus and failedFeeds response objects

The possible responses returned for each failed feed, including the error descriptions, are the same as for the overallStatus object in a Get-Status response.

The following fields appear in the overallStatus object for all loads, and the failedFeeds object for each failed feed:

- **fullUri** – The URI of the file or files to be loaded.

Type: string

Format: s3://bucket/key.

- **runNumber** – The run number of this load or feed. This is incremented when the load is restarted.

Type: unsigned long.

- **retryNumber** – The retry number of this load or feed. This is incremented when the loader automatically retries a feed or load.

Type: unsigned long.

- **status** – The returned status of the load or feed. `LOAD_COMPLETED` indicates a successful load with no problems. For a list of other load-status messages, see [Neptune Loader Error and Feed Messages \(p. 716\)](#).

Type: string.

- **totalTimeSpent** – The time, in seconds, spent to parse and insert data for the load or feed. This does not include the time spent fetching the list of source files.

Type: unsigned long.

- **totalRecords** – Total records loaded or attempted to load.

Type: unsigned long.

- **totalDuplicates** – The number of duplicate records encountered.

Type: unsigned long.

For openCypher loads, a duplicate is counted when:

- The loader detects that a row in a node file has an ID without an ID space that is the same as another ID value without an ID space, either in another row or belonging to an existing node.

- The loader detects that a row in a node file has an ID with an ID space that is the same as another ID value with ID space, either in another row or belonging to an existing node.

See [Special considerations for loading openCypher data \(p. 214\)](#).

- **parsingErrors** – The number of parsing errors encountered.

Type: unsigned long.

- **datatypeMismatchErrors** – The number of records with a data type that did not match the given data.

Type: unsigned long.

- **insertErrors** – The number of records that could not be inserted due to errors.

Type: unsigned long.

Neptune Loader Get-Status `errors` response object

Errors fall into the following categories:

- **Error 400** – An invalid `loadId` returns an HTTP 400 bad request error. The message describes the error.
- **Error 500** – A valid request that cannot be processed returns an HTTP 500 internal server error. The message describes the error.

See [Neptune Loader Error and Feed Messages \(p. 716\)](#) for a list of the error and feed messages returned by the loader in case of errors.

When an error occurs, a JSON `errors` object is returned in the `BODY` of the response, with the following fields:

- **startIndex** – The index of the first included error.

Type: unsigned long.

- **endIndex** – The index of the last included error.

Type: unsigned long.

- **loadId** – The ID of the load. You can use this ID to print the errors for the load by setting the `errors` parameter to `TRUE`.

Type: string.

- **errorLogs** – A list of the errors.

Type: list.

Neptune Loader Get-Status `errorLogs` response object

The `errorLogs` object under `errors` in the loader Get-Status response contains an object describing each error using the following fields:

- **errorCode** – Identifies the nature of error.

It can take one of the following values:

- `PARSING_ERROR`
- `S3_ACCESS_DENIED_ERROR`
- `FROM_OR_TO_VERTEX_ARE_MISSING`

- `ID_ASSIGNED_TO_MULTIPLE_EDGES`
- `SINGLE_CARDINALITY_VIOLATION`
- `FILE_MODIFICATION_OR_DELETION_ERROR`
- `OUT_OF_MEMORY_ERROR`
- `INTERNAL_ERROR` (returned when the bulk loader cannot determine the type of the error).
- **errorMessage** – A message describing the error.

This can be a generic message associated with the error code or a specific message containing details, for example about a missing from/to vertex or about a parsing error.

- **fileName** – The name of the feed.
- **recordNum** – In the case of a parsing error, this is the record number in the file of the record that could not be parsed. It is set to zero if the record number is not applicable to the error, or if it could not be determined.

For example, the bulk loader would generate a parsing error if it encountered a faulty row such as the following in an RDF nquads file:

```
<http://base#subject> |http://base#predicate> <http://base#true> .
```

As you can see, the second `http` in the row above should be preceded by `<` rather than `|`. The resulting error object under `errorLogs` in a status response would look like this:

```
{
  "errorCode" : "PARSING_ERROR",
  "errorMessage" : "Expected '<', found: '|",
  "fileName" : "s3://bucket/key",
  "recordNum" : 12345
},
```

Neptune Loader Get-Status Examples

Example request for load status

The following is a request sent via HTTP GET using the `curl` command.

```
curl -X GET 'https://your-neptune-endpoint:port/loader/loadId (a UUID)'
```

Example Response

```
{
  "status" : "200 OK",
  "payload" : {
    "feedCount" : [
      {
        "LOAD_FAILED" : 1
      }
    ],
    "overallStatus" : {
      "datatypeMismatchErrors" : 0,
      "fullUri" : "s3://bucket/key",
      "insertErrors" : 0,
      "parsingErrors" : 5,
      "retryNumber" : 0,
      "runNumber" : 1,
      "status" : "LOAD_FAILED",

```

```
        "totalDuplicates" : 0,  
        "totalRecords" : 5,  
        "totalTimeSpent" : 3.0  
    }  
  }  
}
```

Example request for loadIds

The following is a request sent via HTTP GET using the `curl` command.

```
curl -X GET 'https://your-neptune-endpoint:port/loader?limit=3'
```

Example Response

```
{  
  "status" : "200 OK",  
  "payload" : {  
    "loadIds" : [  
      "a2c0ce44-a44b-4517-8cd4-1dc144a8e5b5",  
      "09683a01-6f37-4774-bb1b-5620d87f1931",  
      "58085eb8-ceb4-4029-a3dc-3840969826b9"  
    ]  
  }  
}
```

Example request for detailed status

The following is a request sent via HTTP GET using the `curl` command.

```
curl -X GET 'https://your-neptune-endpoint:port/loader/loadId (a UUID)?details=true'
```

Example Response

```
{  
  "status" : "200 OK",  
  "payload" : {  
    "failedFeeds" : [  
      {  
        "datatypeMismatchErrors" : 0,  
        "fullUri" : "s3://bucket/key",  
        "insertErrors" : 0,  
        "parsingErrors" : 5,  
        "retryNumber" : 0,  
        "runNumber" : 1,  
        "status" : "LOAD_FAILED",  
        "totalDuplicates" : 0,  
        "totalRecords" : 5,  
        "totalTimeSpent" : 3.0  
      }  
    ],  
    "feedCount" : [  
      {  
        "LOAD_FAILED" : 1  
      }  
    ],  
    "overallStatus" : {  
      "datatypeMismatchErrors" : 0,  
      "fullUri" : "s3://bucket/key",  
      "insertErrors" : 0,  
      "parsingErrors" : 5,  
      "retryNumber" : 0,  
      "runNumber" : 1,  
      "status" : "LOAD_FAILED",  
      "totalDuplicates" : 0,  
      "totalRecords" : 5,  
      "totalTimeSpent" : 3.0  
    }  
  }  
}
```

```
    "parsingErrors" : 5,  
    "retryNumber" : 0,  
    "runNumber" : 1,  
    "status" : "LOAD_FAILED",  
    "totalDuplicates" : 0,  
    "totalRecords" : 5,  
    "totalTimeSpent" : 3.0  
  }  
}  
}
```

Neptune Loader Get-Status errorLogs examples

Example detailed status response when errors occurred

This a request sent via HTTP GET using curl:

```
curl -X GET 'https://your-neptune-endpoint:port/loader/0a237328-afd5-4574-a0bc-c29ce5f54802?details=true&errors=true&page=1&errorsPerPage=3'
```

Example of a detailed response when errors have occurred

This is an example of the response that you might get from the query above, with an `errorLogs` object listing the load errors encountered:

```
{  
  "status" : "200 OK",  
  "payload" : {  
    "failedFeeds" : [  
      {  
        "datatypeMismatchErrors" : 0,  
        "fullUri" : "s3://bucket/key",  
        "insertErrors" : 0,  
        "parsingErrors" : 5,  
        "retryNumber" : 0,  
        "runNumber" : 1,  
        "status" : "LOAD_FAILED",  
        "totalDuplicates" : 0,  
        "totalRecords" : 5,  
        "totalTimeSpent" : 3.0  
      }  
    ],  
    "feedCount" : [  
      {  
        "LOAD_FAILED" : 1  
      }  
    ],  
    "overallStatus" : {  
      "datatypeMismatchErrors" : 0,  
      "fullUri" : "s3://bucket/key",  
      "insertErrors" : 0,  
      "parsingErrors" : 5,  
      "retryNumber" : 0,  
      "runNumber" : 1,  
      "status" : "LOAD_FAILED",  
      "totalDuplicates" : 0,  
      "totalRecords" : 5,  
      "totalTimeSpent" : 3.0  
    },  
    "errors" : {  
      "endIndex" : 3,  
      "errorLogs" : [  

```



```
{
  "errorCode" : "PARSING_ERROR",
  "errorMessage" : "Expected '<', found: |",
  "fileName" : "s3://bucket/key",
  "recordNum" : 1
},
{
  "errorCode" : "PARSING_ERROR",
  "errorMessage" : "Expected '<', found: |",
  "fileName" : "s3://bucket/key",
  "recordNum" : 2
},
{
  "errorCode" : "PARSING_ERROR",
  "errorMessage" : "Expected '<', found: |",
  "fileName" : "s3://bucket/key",
  "recordNum" : 3
}
],
"loadId" : "0a237328-afd5-4574-a0bc-c29ce5f54802",
"startIndex" : 1
}
}
```

Example of a Data prefetch task interrupted error

Occasionally when you get a `LOAD_FAILED` status and then request more detailed information, the error returned may be a `PARSING_ERROR` with a Data prefetch task interrupted message, like this:

```
"errorLogs" : [
  {
    "errorCode" : "PARSING_ERROR",
    "errorMessage" : "Data prefetch task interrupted: Data prefetch task for 11467
failed",
    "fileName" : "s3://some-source-bucket/some-source-file",
    "recordNum" : 0
  }
]
```

This error occurs when there was a temporary interruption in the data load process that was typically not caused by your request or your data. It can usually be resolved simply by running the bulk upload request again. If you are using default settings, namely `"mode": "AUTO"`, and `"failOnError": "TRUE"`, the loader skips the files that it already successfully loaded and resumes loading files it had not yet loaded when the interruption occurred.

Neptune Loader Cancel Job

Cancels a load job.

To cancel a job, you must send an HTTP `DELETE` request to the `https://your-neptune-endpoint:port/loader` endpoint. The `loadId` can be appended to the `/loader` URL path, or included as a variable in the URL.

Cancel Job request syntax

```
DELETE https://your-neptune-endpoint:port/loader?loadId=loadId
```

```
DELETE https://your-neptune-endpoint:port/loader/loadId
```

Cancel Job Request Parameters

loadId

The ID of the load job.

Cancel Job Response Syntax

```
no response body
```

200 OK

Successfully deleted load job returns a 200 code.

Cancel Job Errors

When an error occurs, a JSON object is returned in the `BODY` of the response. The `message` object contains a description of the error.

Error Categories

- **Error 400** – An invalid `loadId` returns an HTTP 400 bad request error. The message describes the error.
- **Error 500** – A valid request that cannot be processed returns an HTTP 500 internal server error. The message describes the error.

Cancel Job Error Messages

The following are possible error messages from the cancel API with a description of the error.

- The load with id = `load_id` does not exist or not active (HTTP 404) – The load was not found. Check the value of `id` parameter.
- Load cancellation is not permitted on a read replica instance. (HTTP 405) – Loading is a write operation. Retry load on the read/write cluster endpoint.

Cancel Job Examples

Example Request

The following is a request sent via HTTP `DELETE` using the `curl` command.

```
curl -X DELETE 'https://your-neptune-endpoint:port/loader/0a237328-afd5-4574-a0bc-c29ce5f54802'
```

Using AWS Database Migration Service to load data into Amazon Neptune from a different data store

AWS Database Migration Service (AWS DMS) can load data into Neptune from [supported source databases](#) quickly and securely. The source database remains fully operational during the migration, minimizing downtime for applications that rely on it.

You can find detailed information about AWS DMS in the [AWS Database Migration Service User Guide](#) and the [AWS Database Migration Service API Reference](#). In particular, you can find out how to set up a Neptune cluster as a target for migration in [Using Amazon Neptune as a Target for AWS Database Migration Service](#).

Here are some prerequisites for importing data into Neptune using AWS DMS:

- You will need to create a AWS DMS table mapping object to define how data should be extracted from the source database (see [Specifying table selection and transformations by table mapping using JSON](#) in the AWS DMS Userguide for details). This table mapping configuration object specifies which tables should be read and in what order, and how their columns are named. It can also filter the rows being copied and provide simple value transformations such as converting to lower case or rounding.
- You will need to create a Neptune `GraphMappingConfig` to specify how the data extracted from the source database should be loaded into Neptune. For RDF data (queried using SPARQL), the `GraphMappingConfig` is written in the W3's standard [R2RML](#) mapping language. For property graph data (queried using Gremlin), the `GraphMappingConfig` is a JSON object, described in [GraphMappingConfig Layout for Property-Graph/Gremlin Data \(p. 227\)](#).
- You must use AWS DMS to create a replication instance in the same VPC as your Neptune DB cluster, to mediate the transfer of data.
- You will also need an Amazon S3 bucket to be used as intermediate storage for staging the migration data.

Creating a Neptune GraphMappingConfig

The `GraphMappingConfig` that you create specifies how data extracted from a source data store should be loaded into a Neptune DB cluster. Its format differs depending on whether it is intended for loading RDF data or for loading property-graph data.

For RDF data, you can use the W3 [R2RML](#) language for mapping relational data to RDF.

If you are loading property-graph data to be queried using Gremlin, you create a JSON object for `GraphMappingConfig`.

GraphMappingConfig Layout for RDF/SPARQL Data

If you are loading RDF data to be queried using SPARQL, you write the `GraphMappingConfig` in [R2RML](#). R2RML is a standard W3 language for mapping relational data to RDF. Here is one example:

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ex: <http://example.com/ns#> .

<#TriplesMap1>
  rr:logicalTable [ rr:tableName "nodes" ];
  rr:subjectMap [
    rr:template "http://data.example.com/employee/{id}";
    rr:class ex:Employee;
  ];
  rr:predicateObjectMap [
    rr:predicate ex:name;
    rr:objectMap [ rr:column "label" ];
  ] .
```

Here is another example:

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ex: <http://example.com/#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<#TriplesMap2>
  rr:logicalTable [ rr:tableName "Student" ];
  rr:subjectMap [ rr:template "http://example.com/{ID}{Name}";
                 rr:class foaf:Person ];
  rr:predicateObjectMap [
    rr:predicate ex:id ;
    rr:objectMap [ rr:column "ID";
                  rr:datatype xsd:integer ]
  ];
  rr:predicateObjectMap [
    rr:predicate foaf:name ;
    rr:objectMap [ rr:column "Name" ]
  ] .
```

The W3 Recommendation at [R2RML: RDB to RDF Mapping Language](#) provides details of the language.

GraphMappingConfig Layout for Property-Graph/Gremlin Data

A comparable GraphMappingConfig for property-graph data is a JSON object that provides a mapping rule for each graph entity to be generated from the source data. The following template shows what each rule in this object looks like:

```
{
  "rules": [
    {
      "rule_id": "(an identifier for this rule)",
      "rule_name": "(a name for this rule)",
      "table_name": "(the name of the table or view being loaded)",
      "vertex_definitions": [
        {
          "vertex_id_template": "{col1}",
          "vertex_label": "(the vertex to create)",
          "vertex_definition_id": "(an identifier for this vertex)",
          "vertex_properties": [
            {
              "property_name": "(name of the property)",
              "property_value_template": "{col2} or text",
              "property_value_type": "(data type of the property)"
            }
          ]
        }
      ]
    },
    {
      "rule_id": "(an identifier for this rule)",
      "rule_name": "(a name for this rule)",
      "table_name": "(the name of the table or view being loaded)",
      "edge_definitions": [
        {
          "from_vertex": {
            "vertex_id_template": "{col1}",
            "vertex_definition_id": "(an identifier for the vertex referenced above)"
          },
          "to_vertex": {
            "vertex_id_template": "{col3}",
            "vertex_definition_id": "(an identifier for the vertex referenced above)"
          },
          "edge_id_template": {
            "label": "(the edge label to add)",
            "template": "{col1}_{col3}"
          }
        }
      ]
    }
  ]
}
```

```
    "edge_properties":[
      {
        "property_name": "(the property to add)",
        "property_value_template": "{col4} or text",
        "property_value_type": "(data type like String, int, double)"
      }
    ]
  }
]
}
```

Note that the presence of a vertex label implies that the vertex is being created here, whereas its absence implies that the vertex is created by a different source, and this definition is only adding vertex properties.

Here is a sample rule for an employee record:

```
{
  "rules": [
    {
      "rule_id": "1",
      "rule_name": "vertex_mapping_rule_from_nodes",
      "table_name": "nodes",
      "vertex_definitions": [
        {
          "vertex_id_template": "{emp_id}",
          "vertex_label": "employee",
          "vertex_definition_id": "1",
          "vertex_properties": [
            {
              "property_name": "name",
              "property_value_template": "{emp_name}",
              "property_value_type": "String"
            }
          ]
        }
      ]
    },
    {
      "rule_id": "2",
      "rule_name": "edge_mapping_rule_from_emp",
      "table_name": "nodes",
      "edge_definitions": [
        {
          "from_vertex": {
            "vertex_id_template": "{emp_id}",
            "vertex_definition_id": "1"
          },
          "to_vertex": {
            "vertex_id_template": "{mgr_id}",
            "vertex_definition_id": "1"
          },
          "edge_id_template": {
            "label": "reportsTo",
            "template": "{emp_id}_{mgr_id}"
          },
          "edge_properties":[
            {
              "property_name": "team",
              "property_value_template": "{team}",
              "property_value_type": "String"
            }
          ]
        }
      ]
    }
  ]
}
```

```
}  
  ]  
} ]  
} ]  
}
```

Creating an AWS DMS Replication Task With Neptune as the Target

Once you have created your table mapping and graph mapping configurations, use the following process to load data from the source store into Neptune. Consult the AWS DMS documentation for more details about the APIs in question.

Step 1: Create an AWS DMS Replication Instance

Create an AWS DMS replication instance in the VPC where your Neptune DB cluster is running (see [Working with an AWS DMS Replication Instance](#) and [CreateReplicationInstance](#) in the AWS DMS User Guide). You can use an AWS CLI command like the following to do that:

```
aws dms create-replication-instance \  
  --replication-instance-identifier (the replication instance identifier) \  
  --replication-instance-class (the size and capacity of the instance, like  
'dms.t2.medium') \  
  --allocated-storage (the number of gigabytes to allocate for the instance initially) \  
  --engine-version (the DMS engine version that the instance should use) \  
  --vpc-security-group-ids (the security group to be used with the instance)
```

Step 2. Create an AWS DMS Endpoint for the Source Database

The next step is to create an AWS DMS endpoint for your source data store. You can use the AWS DMS [CreateEndpoint](#) API in the AWS CLI like this:

```
aws dms create-endpoint \  
  --endpoint-identifier (source endpoint identifier) \  
  --endpoint-type source \  
  --engine-name (name of source database engine) \  
  --username (user name for database login) \  
  --password (password for login) \  
  --server-name (name of the server) \  
  --port (port number) \  
  --database-name (database name)
```

Step 3. Set Up an Amazon S3 Bucket for Neptune to Use for Staging Data

If you do not have an Amazon S3 bucket that you can use for staging data, create one as explained in [Creating a Bucket](#) in the Amazon S3 Getting-Started Guide, or [How Do I Create an S3 Bucket?](#) in the Console User Guide.

You will need to create an IAM policy granting `GetObject`, `PutObject`, `DeleteObject` and `ListObject` permissions to the bucket if you do not already have one:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": "s3:ListObject",  
      "Resource": "arn:aws:s3:::bucket-name/*",  
      "Effect": "Allow",  
      "Principal": "*" }  
    ]  
}
```

```
{
  "Sid": "ListObjectsInBucket",
  "Effect": "Allow",
  "Action": [
    "s3:ListBucket"
  ],
  "Resource": [
    "arn:aws:s3:::(bucket-name)"
  ]
},
{
  "Sid": "AllObjectActions",
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:PutObject",
    "s3:DeleteObject",
    "s3:ListObject"
  ],
  "Resource": [
    "arn:aws:s3:::(bucket-name)/*"
  ]
}
]
```

If your Neptune DB cluster has IAM authentication enabled, you will also need to include the following policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": "(the ARN of your Neptune DB cluster resource)"
    }
  ]
}
```

Create an IAM role as a trust document to attach the policy to:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Sid": "neptune",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
}
```

After attaching the policy to the role, attach the role to your Neptune DB cluster. This will allow AWS DMS to use the bucket for staging the data being loaded.

Step 4. Create an Amazon S3 Endpoint in the Neptune VPC

Now create a VPC Gateway endpoint for your intermediary Amazon S3 bucket, in the VPC where your Neptune cluster is located. You can use the AWS Management Console or the AWS CLI to do this, as described in [Creating a gateway endpoint](#).

Step 5. Create an AWS DMS Target Endpoint for Neptune

Create an AWS DMS endpoint for your target Neptune DB cluster. You can use the AWS DMS [CreateEndpoint](#) API with the `NeptuneSettings` parameter like this::

```
aws dms create-endpoint \
  --endpoint-identifier (target endpoint identifier) \
  --endpoint-type target \
  --engine-name neptune \
  --server-name (name of the server) \
  --port (port number) \
  --neptune-settings '{ \
    "ServiceAccessRoleArn": "(ARN of the service access role)", \
    "S3BucketName": "(name of S3 bucket to use for staging files when migrating)", \
    "S3BucketFolder": "(name of the folder to use in that S3 bucket)", \
    "ErrorRetryDuration": (number of milliseconds to wait between bulk-load retries), \
    "MaxRetryCount": (the maximum number of times to retry a failing bulk-load job), \
    "MaxFileSize": (maximum file size, in bytes, of the staging files written to S3), \
    "isIamAuthEnabled": (set to true if IAM authentication is enabled on the Neptune cluster) }'
```

The JSON object passed to the AWS DMS `CreateEndpoint` API in its `NeptuneSettings` parameter has the following fields:

- **ServiceAccessRoleArn** – *(required)* The ARN of an IAM role that permits fine-grained access to the S3 bucket used to stage migration of the data into Neptune. This Role should also have permissions to access your Neptune DB cluster if IAM authorization is enabled on it.
- **S3BucketName** – *(required)* For Full Load migration, the replication instance converts all RDS data into CSV, quad files and uploads them to this staging bucket in S3 and then bulk-loads them into Neptune.
- **S3BucketFolder** – *(required)* The folder to use in the S3 staging bucket.
- **ErrorRetryDuration** – *(optional)* The number of milliseconds to wait after a Neptune request fails before making a retry request. The default is 250.
- **MaxRetryCount** – *(optional)* The maximum number of retry requests AWS DMS should make after a retryable failure. The default is 5.
- **MaxFileSize** – *(optional)* The maximum size in bytes of each staging file saved to S3 during the migration. The default is 1,048,576 KB (1 GB).
- **isIAMAuthEnabled** – *(optional)* Set to `true` if IAM authentication is enabled on the Neptune DB cluster, or `false` if not. The default is `false`.

Step 6. Test Connections to the New Endpoints

You can test the connection to each of these new endpoints using the AWS DMS [url="https://docs.aws.amazon.com/dms/latest/APIReference/API_TestConnection.html">TestConnection](https://docs.aws.amazon.com/dms/latest/APIReference/API_TestConnection.html) API like this:


```
aws dms test-connection \  
  --replication-instance-arn (the ARN of the replication instance) \  
  --endpoint-arn (the ARN of the endpoint you are testing)
```

Step 7. Create an AWS DMS Replication Task

Once you have completed the previous steps successfully, create a replication task for migrating data from your source data store to Neptune, using the AWS DMS [CreateReplicationTask](#) API like this:

```
aws dms create-replication-task \  
  --replication-task-identifier (name for the replication task) \  
  --source-endpoint-arn (ARN of the source endpoint) \  
  --target-endpoint-arn (ARN of the target endpoint) \  
  --replication-instance-arn (ARN of the replication instance) \  
  --migration-type full-load \  
  --table-mappings (table-mapping JSON object or URI like 'file:///tmp/table-mappings,json') \  
  --task-data (a GraphMappingConfig object or URI like 'file:///tmp/graph-mapping-config.json')
```

The `TaskData` parameter provides the [GraphMappingConfig](#) (p. 226) that specifies how the data being copied should be stored in Neptune.

Step 8. Start the AWS DMS Replication Task

Now you can start the replication task:

```
aws dms start-replication-task \  
  --replication-task-arn (ARN of the replication task started in the previous step) \  
  --start-replication-task-type start-replication
```

Exporting data from a Neptune DB cluster

There are several good ways to export data from a Neptune DB cluster:

- For small amounts of data, simply use the results of a query or queries.
- For RDF data, the [Graph Store Protocol \(GSP\)](#) (p. 368) can make exporting easy. For example:

```
curl --request GET \  
'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//www.example.com/named/graph'
```

- There is also a powerful and flexible open-source tool for exporting Neptune data, namely [neptune-export](#). The following sections describe the features of this tool and how to use it.

Topics

- [Using neptune-export](#) (p. 234)
- [Using the Neptune-Export service to export Neptune data](#) (p. 235)
- [Using the neptune-export command-line tool to export data from Neptune](#) (p. 240)
- [Files exported by Neptune-Export and neptune-export](#) (p. 243)
- [Parameters used to control the Neptune export process](#) (p. 244)
- [Troubleshooting the Neptune export process](#) (p. 259)

Using `neptune-export`

You can use the open-source `neptune-export` tool in two different ways:

- **As the Neptune-Export service (p. 235).** When you export data from Neptune using the Neptune-Export service, you trigger and monitor export jobs through a REST API.
- **As the `neptune-export` Java command-line utility (p. 240).** To use this command-line tool to export Neptune data, you have to run it in an environment where your Neptune DB cluster is accessible.

Both the Neptune-Export service and the `neptune-export` command line tool publish data to Amazon Simple Storage Service (Amazon S3), encrypted using Amazon S3 server-side encryption (SSE-S3).

If you try to export data from a Neptune DB cluster whose data is changing while the export is happening, the consistency of the exported data is not guaranteed. That is, if your cluster is servicing write traffic while an export job is in progress, there may be inconsistencies in the exported data. This is true whether you export from the primary instance in the cluster or from one or more read replicas.

To guarantee that exported data is consistent, it is best to export from a [clone of your DB cluster \(p. 437\)](#). This both provides the export tool with a static version of your data and ensures that the export job doesn't slow down queries in your original DB cluster.

To make this easier, you can indicate that you want to clone the source DB cluster when you trigger an export job. If you do, the export process automatically creates the clone, uses it for the export, and then deletes it when the export is finished.

Using the Neptune-Export service to export Neptune data















You can use the following steps to export data from your Neptune DB cluster to Amazon S3 using the Neptune-Export service:







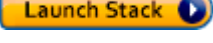
Install the Neptune-Export service

Use an AWS CloudFormation template to create the stack:

To install the Neptune-Export service

1. Launch the AWS CloudFormation stack on the AWS CloudFormation console by choosing one of the **Launch Stack** buttons in the following table:

Region	View	View in Designer	Launch
US East (N. Virginia)	View	View in Designer	
US East (Ohio)	View	View in Designer	
US West (N. California)	View	View in Designer	
US West (Oregon)	View	View in Designer	
Canada (Central)	View	View in Designer	
South America (São Paulo)	View	View in Designer	
Europe (Stockholm)	View	View in Designer	
Europe (Ireland)	View	View in Designer	
Europe (London)	View	View in Designer	
Europe (Paris)	View	View in Designer	
Europe (Frankfurt)	View	View in Designer	
Middle East (Bahrain)	View	View in Designer	
Asia Pacific (Hong Kong)	View	View in Designer	
Asia Pacific (Tokyo)	View	View in Designer	

Region	View	View in Designer	Launch
Asia Pacific (Seoul)	View	View in Designer	
Asia Pacific (Singapore)	View	View in Designer	
Asia Pacific (Sydney)	View	View in Designer	
Asia Pacific (Mumbai)	View	View in Designer	
China (Beijing)	View	View in Designer	
China (Ningxia)	View	View in Designer	
AWS GovCloud (US-West)	View	View in Designer	
AWS GovCloud (US-East)	View	View in Designer	

2. On the **Select Template** page, choose **Next**.
3. On the **Specify Details** page, the template, set the following parameters:
 - **VPC** – The easiest way to set up the Neptune-Export service is to install it in the same Amazon VPC as your Neptune database. If you want to install it in a separate VPC you can use [VPC peering](#) to establish connectivity between the Neptune DB cluster's VPC and the Neptune-Export service VPC.
 - **Subnet1** – The Neptune-Export service must be installed in a subnet in your VPC that allows outbound IPv4 HTTPS traffic from the subnet to the internet. This is so that the Neptune-Export service can call the [AWS Batch API](#) to create and run an export job.

If you created your Neptune cluster using the CloudFormation template on the [Creating a DB Cluster \(p. 166\)](#) page in the Neptune documentation, you can use the `PrivateSubnet1` and `PrivateSubnet2` outputs from that stack to populate this and the next parameter.

 - **Subnet2** – A second subnet in the VPC that allows outbound IPv4 HTTPS traffic from the subnet to the internet.
 - **EnableIAM** – Set this to `true` to secure the Neptune-Endpoint API using AWS Identity and Access Management (IAM). We recommend that you do so.

If you do enable IAM authentication, you must `sigv4` sign all HTTPS requests to the endpoint. You can use a tool such as [awscurl](#) to sign requests on your behalf.

 - **VPCOnly** – Setting this to `true` makes the export endpoint VPC-only, so that you can only access it from within the VPC where the Neptune-Export service is installed. This restricts the Neptune-Export API to being used only from within that VPC.

We recommend that you set `VPCOnly` to `true`.
4. Choose **Next**.
5. On the **Options** page, choose **Next**.
6. On the **Review** page, select the first check box to acknowledge that AWS CloudFormation will create IAM resources. Select the second check box to acknowledge `CAPABILITY_AUTO_EXPAND` for the new stack.

Note

`CAPABILITY_AUTO_EXPAND` explicitly acknowledges that macros will be expanded when creating the stack, without prior review. Users often create a change set from a processed template so that the changes made by macros can be reviewed before actually creating the stack. For more information, see the AWS CloudFormation [CreateStack](#) API.

Then choose **Create**.

Enable access to Neptune from Neptune-Export

After the Neptune-Export installation has completed, update your [Neptune VPC security group \(p. 107\)](#) to allow access from Neptune-Export. When the Neptune-Export AWS CloudFormation stack has been created, the **Outputs** tab includes a `NeptuneExportSecurityGroup` ID. Update your Neptune VPC security group to allow access from this Neptune-Export security group.

Enable access to the Neptune-Export endpoint from a VPC-based EC2 instance

If you make your Neptune-Export endpoint VPC-only, you can only access it from within the VPC in which the Neptune-Export service is installed. To allow connectivity from an Amazon EC2 instance in the VPC from which you can make Neptune-Export API calls, attach the `NeptuneExportSecurityGroup` created by the AWS CloudFormation stack to that Amazon EC2 instance.

Run a Neptune-Export job using the Neptune-Export API

The **Outputs** tab of the AWS CloudFormation stack also includes the `NeptuneExportApiUri`. Use this URI whenever you send a request to the Neptune-Export endpoint.

Run an export job

- Be sure that the user or role under which the export runs has been granted `execute-api:Invoke` permission.
- If you set the `EnableIAM` parameter to `true` in the AWS CloudFormation stack when you installed Neptune-Export, you need to `Sigv4` sign all requests to the Neptune-Export API. We recommend using [awscurl](#) to make requests to the API. All the examples here assume that IAM auth is enabled.
- If you set the `VPCOnly` parameter to `true` in the AWS CloudFormation stack when you installed Neptune-Export, you must call the Neptune-Export API from within the VPC, typically from an Amazon EC2 instance located in the VPC.

To start exporting data, send a request to the `NeptuneExportApiUri` endpoint with `command` and `outputS3Path` request parameters and an `endpoint` export parameter.

The following is an example of a request that exports property-graph data from Neptune and publishes it to Amazon S3:

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
```

```
"command": "export-pg",
"outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
"params": { "endpoint": "(your Neptune endpoint DNS name)" }
}' \
--region us-east-1
```

Similarly, here is an example of a request that exports RDF data from Neptune to Amazon S3:

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
    "command": "export-rdf",
    "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
    "params": { "endpoint": "(your Neptune endpoint DNS name)" }
  }' \
  --region us-east-1
```

If you omit the `command` request parameter, by default Neptune-Export attempts to export property-graph data from Neptune.

If the previous command ran successfully, the output would look like this:

```
{
  "jobName": "neptune-export-abc12345-1589808577790",
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f"
}
```

Monitor the export job you just started

To monitor a running job, append its `jobID` to your `NeptuneExportApiUri`, something like this:

```
curl \
  (your NeptuneExportApiUri)(the job ID) \
  --region eu-west-1
```

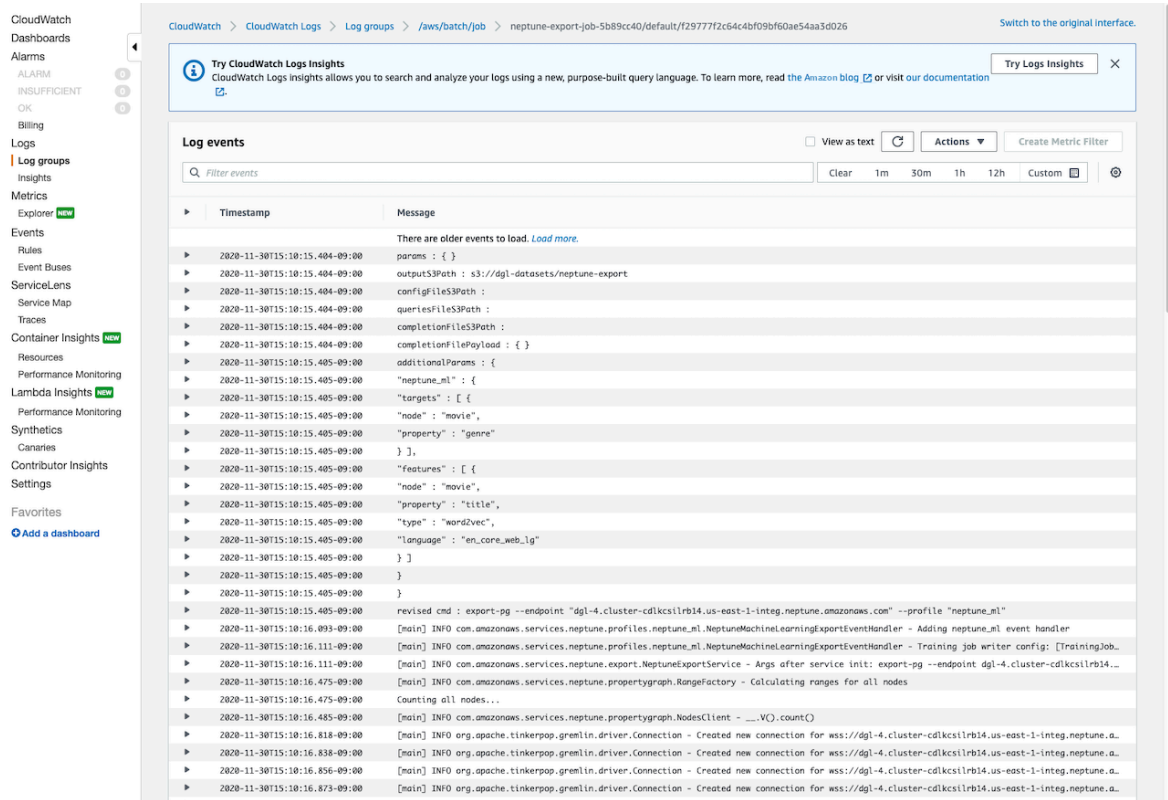
If the service had not yet started the export job, the response would look like this:

```
{
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f",
  "status": "pending"
}
```

When you repeat the command after the export job has started, the response would look something like this:

```
{
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f",
  "status": "running",
  "logs": "https://us-east-1.console.aws.amazon.com/cloudwatch/home?..."
}
```

If you open the logs in CloudWatch Logs using the URI provided by the status call, you can then monitor the progress of the export in detail:



Cancel a running export job

To cancel a running export job using the AWS Management Console

1. Open the AWS Batch console at <https://console.aws.amazon.com/batch/>.
2. Choose **Jobs**.
3. Locate the running job that you want to cancel, based on its jobID.
4. Select **Cancel job**.

To cancel a running export job using the Neptune export API:

Send an HTTP `DELETE` request to the `NeptuneExportApiUri` with the jobID appended, like this:

```
curl -X DELETE (your NeptuneExportApiUri)(the job ID) \  
--region eu-west-1
```


Using the `neptune-export` command-line tool to export data from Neptune

You can use the following steps to export data from your Neptune DB cluster to Amazon S3 using the `neptune-export` command-line utility:

Prerequisites for using the `neptune-export` command-line utility

Before you start

- **Have version 8 of the JDK** – You need version 8 of the [Java SE Development Kit \(JDK\)](#) installed.
- **Download the `neptune-export` utility** – Download and install the [neptune-export.jar](#) file.
- **Make sure `neptune-export` has access to your Neptune VPC** – Run `neptune-export` from a location where it can access the VPC where your Neptune DB cluster is located.

For example, you can run it on an Amazon EC2 instance within the Neptune VPC, or in a separate VPC that is peered with the Neptune VPC, or on a separate bastion host.

- **Make sure the VPC security groups grant access to `neptune-export`** – Check that the VPC security group(s) attached to the Neptune VPC allow access to your DB cluster from the IP address or security group associated with the `neptune-export` environment.
- **Set up the necessary IAM permissions** – If your database has AWS Identity and Access Management (IAM) database authentication enabled, make sure that the role under which `neptune-export` runs is associated with an IAM policy that allows connections to Neptune. For information about Neptune policies, see [IAM Policies \(p. 117\)](#).

If you want to use the `clusterId` export parameter in your query requests, the role under which `neptune-export` runs requires the following IAM permissions:

- `rds:DescribeDBClusters`
- `rds:DescribeDBInstances`
- `rds:ListTagsForResource`

If you want to export from a cloned cluster, the role under which `neptune-export` runs requires the following IAM permissions:

- `rds:AddTagsToResource`
- `rds:DescribeDBClusters`
- `rds:DescribeDBInstances`
- `rds:ListTagsForResource`
- `rds:DescribeDBClusterParameters`
- `rds:DescribeDBParameters`
- `rds:ModifyDBParameterGroup`
- `rds:ModifyDBClusterParameterGroup`
- `rds:RestoreDBClusterToPointInTime`
- `rds>DeleteDBInstance`
- `rds>DeleteDBClusterParameterGroup`
- `rds>DeleteDBParameterGroup`
- `rds>DeleteDBCluster`
- `rds>CreateDBInstance`

- `rds:CreateDBClusterParameterGroup`
- `rds:CreateDBParameterGroup`

To publish the exported data to Amazon S3, the role under which `neptune-export` runs requires the following IAM permissions for the Amazon S3 location(s):

- `s3:PutObject`
- `s3:PutObjectTagging`
- `s3:GetObject`
- **Set the `SERVICE_REGION` environment variable** – Set the `SERVICE_REGION` environment variable to identify the Region where your DB cluster is located (see [Connecting to Neptune \(p. 151\)](#) for a list of Region identifiers).

Running the `neptune-export` utility to initiate an export operation

Use the following command to run `neptune-export` from the command line and start an export operation:

```
java -jar neptune-export.jar nesvc \  
  -root-path (path to a local directory) \  
  -json (the JSON file that defines the export)
```

The command has two parameters:

Parameters for `neptune-export` when starting an export

- `--root-path` – Path to a local directory where export files are written before being published to Amazon S3.
- `--json` – A JSON object that defines the export.

Example commands using the `neptune-export` command line utility

To export property-graph data directly from your source DB cluster:

```
java -jar neptune-export.jar nesvc \  
  --root-path /home/ec2-user/neptune-export \  
  --json '{  
    "command": "export-pg",  
    "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",  
    "params": {  
      "endpoint" : "(your neptune DB cluster endpoint)"  
    }  
  }'
```

To export RDF data directly from your source DB cluster:

```
java -jar neptune-export.jar nesvc \  
  --root-path /home/ec2-user/neptune-export \  
  --json '{  
    "command": "export-rdf",
```

```
"outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
"params": {
  "endpoint" : "(your neptune DB cluster endpoint)"
}
}'
```

If you omit the command request parameter, the `neptune-export` utility reverts to export property-graph data from Neptune by default.

To export from a clone of your DB cluster:

```
java -jar neptune-export.jar nesvc \
--root-path /home/ec2-user/neptune-export \
--json '{
  "command": "export-pg",
  "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint" : "(your neptune DB cluster endpoint)",
    "cloneCluster" : true
  }
}'
```

To export from your DB cluster using IAM authentication:

```
java -jar neptune-export.jar nesvc \
--root-path /home/ec2-user/neptune-export \
--json '{
  "command": "export-pg",
  "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint" : "(your neptune DB cluster endpoint)"
    "useIamAuth" : true
  }
}'
```

Files exported by Neptune-Export and `neptune-export`

When an export is complete, the export files are published to the Amazon S3 location you have specified. All files published to Amazon S3 are encrypted using Amazon S3 server-side encryption (SSE-S3). The folders and files published to Amazon S3 vary depending on whether you are exporting property-graph or RDF data. If you open the Amazon Amazon S3 location where the files are published, you see the following content:

Locations of exported files in Amazon S3

- **nodes/** – This folder contains node data files in either a comma-separated value (CSV) or a JSON format.

In Neptune, nodes can have one or more labels. Nodes with different individual labels (or different combinations of multiple labels) are written to different files, meaning that no individual file contains data for nodes with different combinations of labels. If a node has multiple labels, these labels are sorted alphabetically before they are assigned to a file.

- **edges/** – This folder contains edge data files in either a comma-separated value (CSV) or a JSON format.

As with the nodes files, edge data is written to different files based on a combinations of their labels. For purposes of model-training, edge data is assigned to different files based on a combination of the edge's label plus the labels of the edge's start and end nodes.

- **statements/** – This folder contains **RDF** data files in Turtle, N-Quads, N-Triples, or JSON format.
- **config.json** – This file contains the *schema* of the graph as inferred by the export process.
- **lastEventId.json** – This file contains the `commitNum` and `opNum` of the last event on the database's Neptune streams. The export process only includes this file if you set the `includeLastEventId` export parameter to `true`, and the database from which you are exporting data has [Neptune streams \(p. 452\)](#) enabled.

Parameters used to control the Neptune export process

Whether you are using the Neptune-Export service or the `neptune-export` command line utility, the parameters you use to control the export are mostly the same. They contain a JSON object passed to the Neptune-Export endpoint or to `neptune-export` on the command line.

The object passed in to the export process has up to five top-level fields:

```
-d '{
  "command" : "(either export-pg or export-rdf)",
  "outputS3Path" : "s3:/(your Amazon S3 bucket)/(path to the folder for exported
data)",
  "jobsize" : "(for Neptune-Export service only)",
  "params" : { (a JSON object that contains export-process parameters) },
  "additionalParams": { (a JSON object that contains parameters for training
configuration) }
}'
```

Contents

- [The command parameter \(p. 245\)](#)
- [The outputS3Path parameter \(p. 245\)](#)
- [The jobSize parameter \(p. 245\)](#)
- [The params object \(p. 246\)](#)
- [The additionalParams object \(p. 246\)](#)
- [Export parameter fields in the params top-level JSON object \(p. 247\)](#)
 - [List of possible fields in the export parameters params object \(p. 247\)](#)
 - [List of fields common to all types of export \(p. 247\)](#)
 - [List of fields for property-graph exports \(p. 247\)](#)
 - [List of fields for RDF exports \(p. 247\)](#)
 - [Fields common to all types of export \(p. 247\)](#)
 - [cloneCluster field in params \(p. 247\)](#)
 - [cloneClusterInstanceType field in params \(p. 248\)](#)
 - [cloneClusterReplicaCount field in params \(p. 248\)](#)
 - [clusterId field in params \(p. 248\)](#)
 - [endpoint field in params \(p. 249\)](#)
 - [endpoints field in params \(p. 249\)](#)
 - [profile field in params \(p. 249\)](#)
 - [uselamAuth field in params \(p. 250\)](#)
 - [includeLastEventId field in params \(p. 250\)](#)
 - [Fields for property-graph export \(p. 250\)](#)
 - [concurrency field in params \(p. 250\)](#)
 - [edgeLabels field in params \(p. 250\)](#)
 - [filter field in params \(p. 251\)](#)
 - [filterConfigFile field in params \(p. 251\)](#)
 - [format field used for property-graph data in params \(p. 251\)](#)
 - [gremlinFilter field in params \(p. 252\)](#)
 - [gremlinNodeFilter field in params \(p. 252\)](#)

- [gremlinEdgeFilter](#) field in params (p. 252)
- [nodeLabels](#) field in params (p. 253)
- [scope](#) field in params (p. 253)
- [Fields for property-graph export](#) (p. 253)
 - [format](#) field used for RDF data in params (p. 253)
 - [rdfExportScope](#) field in params (p. 253)
 - [sparql](#) field in params (p. 254)
- [Examples of filtering what is exported](#) (p. 255)
 - [Filtering the export of property-graph data](#) (p. 255)
 - [Example of using scope to export only edges](#) (p. 255)
 - [Example of using nodeLabels and edgeLabels to export only nodes and edges having specific labels](#) (p. 255)
 - [Example of using filter to export only specified nodes, edges and properties](#) (p. 255)
 - [Example that uses gremlinFilter](#) (p. 256)
 - [Example that uses gremlinNodeFilter](#) (p. 256)
 - [Example that uses gremlinEdgeFilter](#) (p. 256)
 - [Combining filter, gremlinNodeFilter, nodeLabels, edgeLabels and scope](#) (p. 256)
 - [Filtering the export of RDF data](#) (p. 257)
 - [Using rdfExportScope and sparql to export specific edges](#) (p. 257)

The command parameter

The `command` top-level parameter determines whether to export property-graph data or RDF data. If you omit the `command` parameter, the export process defaults to exporting property-graph data.

- **export-pg** – Export property-graph data.
- **export-rdf** – Export RDF data.

The outputS3Path parameter

The `outputS3Path` top-level parameter is required, and must contain the URI of an Amazon S3 location to which the exported files can be published:

```
"outputS3Path" : "s3://(your Amazon S3 bucket)/(path to output folder)"
```

The value must begin with `s3://`, followed by a valid bucket name and optionally a folder path within the bucket.

The jobSize parameter

The `jobSize` top-level parameter is only used with the the Neptune-Export service, not with the `neptune-export` command line utility, and is optional. It lets you characterize the size of the export job you are starting, which helps determine the amount of compute resources devoted to the job and its maximum concurrency level.

```
"jobSize" : "(one of four size descriptors)"
```

The four valid size descriptors are:

- `small` – Maximum concurrency: 8. Suitable for storage volumes up to 10 GB.
- `medium` – Maximum concurrency: 32. Suitable for storage volumes up to 100 GB.
- `large` – Maximum concurrency: 64. Suitable for storage volumes over 100 GB but less than 1 TB.
- `xlarge` – Maximum concurrency: 96. Suitable for storage volumes over 1 TB.

By default, an export initiated on the Neptune-Export service runs as a `small` job.

The performance of an export depends not only on the `jobSize` setting, but also on the number of database instances that you're exporting from, the size of each instance, and the effective concurrency level of the job.

For property-graph exports, you can configure the number of database instances using the [cloneClusterReplicaCount](#) (p. 248) parameter, and you can configure the job's effective concurrency level using the [concurrency](#) (p. 250) parameter.

The `params` object

The `params` top-level parameter is a JSON object that contains parameters that you use to control the export process itself, as explained in [Export parameter fields in the `params` top-level JSON object](#) (p. 247). Some of the fields in the `params` object are specific to property-graph exports, some to RDF.

The `additionalParams` object

The `additionalParams` top-level parameter is a JSON object that contains parameters you can use to control actions that are applied to the data after it has been exported. At present, `additionalParams` is used only for exporting training data for [Neptune ML](#) (p. 541).

Export parameter fields in the `params` top-level JSON object

The Neptune export `params` JSON object allows you to control the export, including the type and format of the exported data.

List of possible fields in the export parameters `params` object

Listed below are all the possible top-level fields that can appear in a `params` object. Only a subset of these fields appear in any one object.

List of fields common to all types of export

- `cloneCluster` (p. 247)
- `cloneClusterInstanceType` (p. 248)
- `cloneClusterReplicaCount` (p. 248)
- `clusterId` (p. 248)
- `endpoint` (p. 249)
- `endpoints` (p. 249)
- `profile` (p. 249)
- `useIamAuth` (p. 250)
- `includeLastEventId` (p. 250)

List of fields for property-graph exports

- `concurrency` (p. 250)
- `edgeLabels` (p. 250)
- `filter` (p. 251)
- `filterConfigFile` (p. 251)
- `gremlinFilter` (p. 252)
- `gremlinNodeFilter` (p. 252)
- `gremlinEdgeFilter` (p. 252)
- `format` (p. 251)
- `nodeLabels` (p. 253)
- `scope` (p. 253)

List of fields for RDF exports

- `format` (p. 253)
- `rdfExportScope` (p. 253)
- `sparql` (p. 254)

Fields common to all types of export

`cloneCluster` field in `params`

(Optional). Default: `false`.

If the `cloneCluster` parameter is set to `true`, the export process uses a fast clone of your DB cluster:

```
"cloneCluster" : true
```

By default, the export process exports data from the DB cluster that you specify using the `endpoint`, `endpoints` or `clusterId` parameters. However, if your DB cluster is in use while the export is going on, and data is changing, the export process cannot guarantee the consistency of the data being exported.

To ensure that the exported data is consistent, use the `cloneCluster` parameter to export from a static clone of your DB cluster instead.

The cloned DB cluster is created in the same VPC as the source DB cluster and inherits the security group, subnet group and IAM database authentication settings of the source. When the export is complete, Neptune deletes the cloned DB cluster.

By default, a cloned DB cluster consists of a single instance of the same instance type as the primary instance in the source DB cluster. You can change the instance type used for the cloned DB cluster by specifying a different one using `cloneClusterInstanceType`.

Note

If you don't use the `cloneCluster` option, and are exporting directly from your main DB cluster, you might need to increase the timeout on the instances from which data is being exported. For large data sets, the timeout should be set to several hours.

`cloneClusterInstanceType` field in params

(Optional).

If the `cloneCluster` parameter is present and set to `true`, you can use the `cloneClusterInstanceType` parameter to specify the instance type used for the cloned DB cluster:

By default, a cloned DB cluster consists of a single instance of the same instance type as the primary instance in the source DB cluster.

```
"cloneClusterInstanceType" : "(for example, r5.12xlarge)"
```

`cloneClusterReplicaCount` field in params

(Optional).

If the `cloneCluster` parameter is present and set to `true`, you can use the `cloneClusterReplicaCount` parameter to specify the number of read-replica instances created in the cloned DB cluster:

```
"cloneClusterReplicaCount" : (for example, 3)
```

By default, a cloned DB cluster consists of a single primary instance. The `cloneClusterReplicaCount` parameter lets you specify how many additional read-replica instances should be created.

`clusterId` field in params

(Optional).

The `clusterId` parameter specifies the ID of a DB cluster to use:

```
"clusterId" : "(the ID of your DB cluster)"
```

If you use the `clusterId` parameter, the export process uses all available instances in that DB cluster to extract data.

Note

The `endpoint`, `endpoints`, and `clusterId` parameters are mutually exclusive. Use one and only one of them.

endpoint field in params

(Optional).

Use `endpoint` to specify an endpoint of a Neptune instance in your DB cluster that the export process can query to extract data (see [Endpoint Connections \(p. 78\)](#)). This is the DNS name only, and does not include the protocol or port:

```
"endpoint" : "(a DNS endpoint of your DB cluster)"
```

Use a cluster or instance endpoint, but not the main reader endpoint.

Note

The `endpoint`, `endpoints`, and `clusterId` parameters are mutually exclusive. Use one and only one of them.

endpoints field in params

(Optional).

Use `endpoints` to specify a JSON array of endpoints in your DB cluster that the export process can query to extract data (see [Endpoint Connections \(p. 78\)](#)). These are DNS names only, and do not include the protocol or port:

```
"endpoints": [  
  "(one endpoint in your DB cluster)",  
  "(another endpoint in your DB cluster)",  
  "(a third endpoint in your DB cluster)"  
]
```

If you have multiple instances in your cluster (a primary and one or more read replicas), you can improve export performance by using the `endpoints` parameter to distribute queries across a list of those endpoints.

Note

The `endpoint`, `endpoints`, and `clusterId` parameters are mutually exclusive. Use one and only one of them.

profile field in params

(Required to export training data for Neptune ML, unless the `neptune_ml` field is present in the `additionalParams` field).

The `profile` parameter provides sets of pre-configured parameters for specific workloads. At present, the export process only supports the `neptune_ml` profile

If you are exporting training data for Neptune ML, add the following parameter to the `params` object:

```
"profile" : "neptune_ml"
```

useIamAuth field in params

(Optional). Default: `false`.

If the database from which you are exporting data has [IAM authentication enabled \(p. 117\)](#), you must include the `useIamAuth` parameter set to `true`:

```
"useIamAuth" : true
```

includeLastEventId field in params

If you set `includeLastEventId` to `true`, and the database from which you are exporting data has [Neptune Streams \(p. 452\)](#) enabled, the export process writes a `lastEventId.json` file to your specified export location. This file contains the `commitNum` and `opNum` of the last event in the stream.

```
"includeLastEventId" : true
```

A cloned database created by the export process inherits the streams setting of its parent. If the parent has streams enabled, the clone will likewise have streams enabled. The contents of the stream on the clone will reflect the contents of the parent (including the same event IDs) at the point in time the clone was created.

Fields for property-graph export

concurrency field in params

(Optional). Default: `4`.

The `concurrency` parameter specifies the number of parallel queries that the export process should use:

```
"concurrency" : (for example, 24)
```

A good guideline is to set the concurrency level to twice the number of vCPUs on all the instances from which you are exporting data. An `r5.xlarge` instance, for example, has 4 vCPUs. If you are exporting from a cluster of 3 `r5.xlarge` instances, you can set the concurrency level to 24 (= 3 x 2 x 4).

If you are using the Neptune-Export service, the concurrency level is limited by the [jobSize \(p. 245\)](#) setting. A small job, for example, supports a concurrency level of 8. If you try to specify a concurrency level of 24 for a small job using the `concurrency` parameter, the effective level remains at 8.

If you export from a cloned cluster, the export process calculates an appropriate concurrency level based on the size of the cloned instances and the job size.

edgeLabels field in params

(Optional).

Use `edgeLabels` to export only those edges that have labels that you specify:

```
"edgeLabels" : ["(a label)", "(another label)"]
```

Each label in the JSON array must be a single, simple label.

The `scope` parameter takes precedence over the `edgeLabels` parameter, so if the `scope` value does not include edges, the `edgeLabels` parameter has no effect.

filter field in params

(Optional).

Use `filter` to specify that only nodes and/or edges with specific labels should be exported, and to filter the properties that are exported for each node or edge.

The general structure of a `filter` object, either inline or in a filter-configuration file, is as follows:

```
"filter" : {  
  "nodes": [ (array of node label and properties objects) ],  
  "edges": [ (array of edge definition an properties objects) ]  
}
```

- **nodes** – Contains a JSON array of nodes and node properties in the following form:

```
"nodes" : [  
  {  
    "label": "(node label)",  
    "properties": [ "(a property name)", "(another property name)", ( ... ) ]  
  }  
]
```

- **label** – The node's property-graph label or labels.

Takes a single value or, if the node has multiple labels, an array of values.

- **properties** – Contains an array of the names of the node's properties that you want to export.
- **edges** – Contains a JSON array of edge definitions in the following form:

```
"edges" : [  
  {  
    "label": "(edge label)",  
    "properties": [ "(a property name)", "(another property name)", ( ... ) ]  
  }  
]
```

- **label** – The edge's property graph label. Takes a single value.

- **properties** – Contains an array of the names of the edge's properties that you want to export.

filterConfigFile field in params

(Optional).

Use `filterConfigFile` to specify a JSON file that contains a filter configuration in the same form that the `filter` parameter takes:

```
"filterConfigFile" : "s3://(your Amazon S3 bucket)/neptune-export/(the name of the JSON file)"
```

See [filter \(p. 251\)](#) for the format of the `filterConfigFile` file.

format field used for property-graph data in params

(Optional). *Default: csv* (comma-separated values)

The `format` parameter specifies the output format of the exported property graph data:

```
"format" : (one of: csv, csvNoHeaders, json, neptuneStreamsJson)
```

- **csv** – Comma-separated value (CSV) formatted output, with column headings formatted according to the [Gremlin load data format \(p. 195\)](#).
- **csvNoHeaders** – CSV formatted data with no column headings.
- **json** – JSON formatted data.
- **neptuneStreamsJson** – JSON formatted data that uses the [GREMLIN_JSON change serialization format \(p. 456\)](#).

gremlinFilter field in params

(Optional).

The `gremlinFilter` parameter allows you to supply a Gremlin snippet, such as a `has()` step, that is used to filter both nodes and edges:

```
"gremlinFilter" : (a Gremlin snippet)
```

Field names and string values should be surrounded by escaped double quotes. For dates and times, you can use the [datetime \(p. 698\)](#) method.

The following example exports only those nodes and edges with a `date-created` property whose value is greater than 2021-10-10:

```
"gremlinFilter" : "has(\"created\", gt(datetime(\"2021-10-10\")))"
```

gremlinNodeFilter field in params

(Optional).

The `gremlinNodeFilter` parameter allows you to supply a Gremlin snippet, such as a `has()` step, that is used to filter nodes:

```
"gremlinNodeFilter" : (a Gremlin snippet)
```

Field names and string values should be surrounded by escaped double quotes. For dates and times, you can use the [datetime \(p. 698\)](#) method.

The following example exports only those nodes with a `deleted` Boolean property whose value is `true`:

```
"gremlinNodeFilter" : "has(\"deleted\", true)"
```

gremlinEdgeFilter field in params

(Optional).

The `gremlinEdgeFilter` parameter allows you to supply a Gremlin snippet, such as a `has()` step, that is used to filter edges:

```
"gremlinEdgeFilter" : (a Gremlin snippet)
```

Field names and string values should be surrounded by escaped double quotes. For dates and times, you can use the [datetime \(p. 698\)](#) method.

The following example exports only those edges with a `strength` numerical property whose value is 5:

```
"gremlinEdgeFilter" : "has(\"strength\", 5)"
```

nodeLabels field in params

(Optional).

Use `nodeLabels` to export only those nodes that have labels you specify:

```
"nodeLabels" : ["(a label)", "(another label)"]
```

Each label in the JSON array must be a single, simple label.

The `scope` parameter takes precedence over the `nodeLabels` parameter, so if the `scope` value does not include nodes, the `nodeLabels` parameter has no effect.

scope field in params

(Optional). Default: `all`.

The `scope` parameter specifies whether to export only nodes, or only edges, or both nodes and edges:

```
"scope" : (one of: nodes, edges, or all)
```

- `nodes` – Export nodes and their properties only.
- `edges` – Export edges and their properties only.
- `all` – Export both nodes and edges and their properties (the default).

Fields for property-graph export

format field used for RDF data in params

(Optional). Default: `turtle`

The `format` parameter specifies the output format of the exported RDF data:

```
"format" : (one of: turtle, nquads, ntriples, neptuneStreamsJson)
```

- `turtle` – Turtle formatted output.
- `nquads` – N-Quads formatted data with no column headings.
- `ntriples` – N-Triples formatted data.
- `neptuneStreamsJson` – JSON formatted data that uses the [SPARQL NQUADS change serialization format \(p. 456\)](#).

rdfExportScope field in params

(Optional). Default: `graph`.

The `rdfExportScope` parameter specifies the scope of the RDF export:

```
"rdfExportScope" : (one of: graph, edges, or query)
```

- `graph` – Export all RDF data.
- `edges` – Export only those triples that represent edges.
- `query` – Export data retrieved by a SPARQL query that is supplied using the `sparql` field.

sparql field in params

(Optional).

The `sparql` parameter allows you to specify a SPARQL query to retrieve the data to export:

```
"sparql" : (a SPARQL query)
```

If you supply a query using the `sparql` field, you must also set the `rdfExportScope` field to `query`.

Examples of filtering what is exported

Here are examples that illustrate ways to filter the data that is exported.

Filtering the export of property-graph data

Example of using `scope` to export only edges

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "scope": "edges"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Example of using `nodeLabels` and `edgeLabels` to export only nodes and edges having specific labels

The `nodeLabels` parameter in the following example specifies that only nodes having a `Person` label or a `Post` label should be exported. The `edgeLabels` parameter specifies that only edges with a `likes` label should be exported:

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "nodeLabels": ["Person", "Post"],
    "edgeLabels": ["likes"]
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Example of using `filter` to export only specified nodes, edges and properties

The `filter` object in this example exports `country` nodes with their `type`, `code` and `desc` properties, and also `route` edges with their `dist` property.

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "filter": {
      "nodes": [
        {
          "label": "country",
          "properties": [
            "type",
            "code",
            "desc"
          ]
        }
      ]
    },
    "edges": [
      {
        "label": "route",
        "properties": [

```



```
        "dist"
      ]
    }
  ]
},
"outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Example that uses `gremlinFilter`

This example uses `gremlinFilter` to export only those nodes and edges created after 2021-10-10 (that is, with a `created` property whose value is greater than 2021-10-10):

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinFilter" : "has(\"created\", gt(datetime(\"2021-10-10\")))"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Example that uses `gremlinNodeFilter`

This example uses `gremlinNodeFilter` to export only deleted nodes (nodes with a Boolean `deleted` property whose value is true):

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinNodeFilter" : "has(\"deleted\", true)"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Example that uses `gremlinEdgeFilter`

This example uses `gremlinEdgeFilter` to export only edges with a `strength` numerical property whose value is 5:

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinEdgeFilter" : "has(\"strength\", 5)"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Combining `filter`, `gremlinNodeFilter`, `nodeLabels`, `edgeLabels` and `scope`

The `filter` object in this example exports:

- `country` nodes with their `type`, `code` and `desc` properties
- `airport` nodes with their `code`, `icao` and `runways` properties

- route edges with their `dist` property

The `gremlinNodeFilter` parameter filters the nodes so that only nodes with a `code` property whose value begins with `A` are exported.

The `nodeLabels` and `edgeLabels` parameters further restrict the output so that only `airport` nodes and `route` edges are exported.

Finally, the `scope` parameter eliminates edges from the export, which leaves only the designated `airport` nodes in the output.

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "filter": {
      "nodes": [
        {
          "label": "airport",
          "properties": [
            "code",
            "icao",
            "runways"
          ]
        },
        {
          "label": "country",
          "properties": [
            "type",
            "code",
            "desc"
          ]
        }
      ],
      "edges": [
        {
          "label": "route",
          "properties": [
            "dist"
          ]
        }
      ]
    },
    "gremlinNodeFilter": "has(\"code\", startingWith(\"A\"))",
    "nodeLabels": [
      "airport"
    ],
    "edgeLabels": [
      "route"
    ],
    "scope": "nodes"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Filtering the export of RDF data

Using `rdfExportScope` and `sparql` to export specific edges

This example exports triples whose predicate is `<http://kelvinlawrence.net/air-routes/objectProperty/route>` and whose object is not a literal:

```
{
  "command": "export-rdf",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "rdfExportScope": "query",
    "sparql": "CONSTRUCT { ?s <http://kelvinlawrence.net/air-routes/objectProperty/route> ?
o } WHERE { ?s ?p ?o . FILTER(!isLiteral(?o)) }"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

Troubleshooting the Neptune export process

The Amazon Neptune export process uses [AWS Batch](#) to provision the compute and storage resources necessary to export your Neptune data. When an export is running, you can use the link in the `logs` field to access the CloudWatch logs for the export job.

However, the CloudWatch logs for the AWS Batch job that performs the export are only available when the AWS Batch job is running. If Neptune export reports that an export is in a pending state, there won't be a logs link through which you can access CloudWatch logs. If an export job remains in the `pending` state for more than a few minutes, there may be a problem provisioning the underlying AWS Batch resources.

When the export job leaves the pending state, you can check its status as follows:

To check the status of a AWS Batch job

1. Open the AWS Batch console at <https://console.aws.amazon.com/batch/>.
2. Select the `neptune-export` job queue.
3. Look for the job whose name matches the `jobName` returned by Neptune export when you started the export.

The screenshot shows the AWS Batch console interface. At the top, there's a section titled "Jobs (6)". Below this, there are two filter dropdowns: "Job queue" set to "neptune-export-queue-ef4c12d0" and "Status" set to "None". To the right of these filters is a toggle switch. Below the filters is a table with two columns: "Name" and "ID". The table contains one row with the following data:

Name	ID
neptune-export-ef4c12d0-1637310099957	9a4e45c6-5152-4b9a-ac5a-e28d34c6f099

If the job remains stuck in a `RUNNABLE` state, it may be because networking or security issues are preventing the container instance from joining the underlying Amazon Elastic Container Service (Amazon ECS) cluster. See the section about verifying network and security settings of the compute environment in [this support article](#).

Another thing you can check is for problems with auto-scaling:

To check the Amazon EC2 auto-scaling group for the AWS Batch compute environment

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Select the **Auto Scaling** group for the neptune-export compute environment.
3. Open the **Activity** tab and check the activity history for unsuccessful events.

EC2 > Auto Scaling groups > neptune-export-compute-environment

Details

Activity

Automatic scaling

Instance manag

Activity notifications (0)

🔍 *Filter notifications*



Send to

Activity history (12)

🔍 *Filter activity history*

Status



Description



Cause

Launching a new EC2 instance. Status Reason: We currently do not have sufficient c5.9xlarge

capacity in the Availability Zone you requested (eu-west-2b). Our system will be

Querying a Neptune Graph

Neptune supports the following graph query languages to access a graph:

- **Gremlin**, defined by [Apache TinkerPop](#) for creating and querying property graphs.

A query in Gremlin is a traversal made up of discrete steps, each of which follows an edge to a node.

See [Accessing a Neptune graph with Gremlin \(p. 262\)](#) to learn about using Gremlin in Neptune, and [Gremlin standards compliance in Amazon Neptune \(p. 47\)](#) to find specific details about the Neptune implementation of Gremlin.

- **SPARQL** is a declarative language based on graph pattern-matching, for querying [RDF](#) data. It is supported by the [World Wide Web Consortium](#).

See [Accessing the Neptune graph with SPARQL \(p. 342\)](#) to learn about using SPARQL in Neptune, and [SPARQL Standards Compliance in Amazon Neptune \(p. 56\)](#) to find specific details about the Neptune implementation of SPARQL.

- **openCypher (p. 392)** is a declarative query language for property graphs that was originally developed by Neo4j, then open-sourced in 2015, and contributed to the [openCypher](#) project under an Apache 2 open-source license. Its syntax is documented in the [Cypher Query Language Reference, Version 9](#).

Topics

- [Accessing a Neptune graph with Gremlin \(p. 262\)](#)
- [Accessing the Neptune graph with SPARQL \(p. 342\)](#)
- [Accessing the Neptune Graph with openCypher \(p. 392\)](#)
- [Query queuing in Amazon Neptune \(p. 404\)](#)

Accessing a Neptune graph with Gremlin

Amazon Neptune is compatible with Apache TinkerPop3 and Gremlin. This means that you can connect to a Neptune DB instance and use the Gremlin traversal language to query the graph (see [The Graph](#) in the Apache TinkerPop3 documentation). For differences in the Neptune implementation of Gremlin, see [Gremlin standards compliance \(p. 47\)](#).

Different Neptune engine versions support different Gremlin versions. Check the [engine release page \(p. 718\)](#) of the Neptune version you are running to determine which Gremlin release it supports.

A *traversal* in Gremlin is a series of chained steps. It starts at a vertex (or edge). It walks the graph by following the outgoing edges of each vertex and then the outgoing edges of those vertices. Each step is an operation in the traversal. For more information, see [The Traversal](#) in the TinkerPop3 documentation.

There are Gremlin language variants and support for Gremlin access in various programming languages. For more information, see [On Gremlin Language Variants](#) in the TinkerPop3 documentation.

This documentation describes how to access Neptune with the following variants and programming languages.

As discussed in [Encryption in Transit: Connecting to Neptune Using SSL/HTTPS \(p. 108\)](#), you can connect to Neptune using Transport Layer Security/Secure Sockets Layer (TLS/SSL) in all AWS Regions. And there

are some Regions where you can *only* connect using TLS/SSL. Regardless of your Region, it is a best practice to use HTTPS instead of HTTP in your connections going forward.

Gremlin-Groovy

The Gremlin Console and HTTP REST examples in this section use the Gremlin-Groovy variant. For more information about the Gremlin Console and Amazon Neptune, see the [the section called "Using Gremlin" \(p. 173\)](#) section of the Quick Start.

Gremlin-Java

The Java sample is written with the official TinkerPop3 Java implementation and uses the Gremlin-Java variant.

Gremlin-Python

The Python sample is written with the official TinkerPop3 Python implementation and uses the Gremlin-Python variant.

The following sections walk you through how to use the Gremlin Console, **REST** over HTTPS, and various programming languages to connect to a Neptune DB instance.

Before you begin, you must have the following:

- A Neptune DB instance. For information about creating a Neptune DB instance, see [Creating a New Neptune DB Cluster Using AWS CloudFormation or Manually \(p. 166\)](#).
- An Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

For more information about loading data into Neptune, including prerequisites, loading formats, and load parameters, see [Loading data into Amazon Neptune \(p. 190\)](#).

Topics

- [Set up the Gremlin console to connect to a Neptune DB instance \(p. 263\)](#)
- [Using the HTTPS REST endpoint to connect to a Neptune DB instance \(p. 266\)](#)
- [Java-based Gremlin clients to use with Amazon Neptune \(p. 267\)](#)
- [Using Python to connect to a Neptune DB instance \(p. 274\)](#)
- [Using .NET to connect to a Neptune DB instance \(p. 276\)](#)
- [Using Node.js to connect to a Neptune DB instance \(p. 277\)](#)
- [Gremlin query hints \(p. 278\)](#)
- [Gremlin query status API \(p. 284\)](#)
- [Gremlin query cancellation \(p. 286\)](#)
- [Support for Gremlin sessions \(p. 286\)](#)
- [Using the Gremlin API with Amazon Neptune \(p. 287\)](#)
- [Caching query results in Amazon Neptune Gremlin \(p. 288\)](#)
- [Analyzing Neptune query execution using Gremlin explain \(p. 292\)](#)
- [Using Gremlin with the Neptune DFE query engine \(p. 340\)](#)

Set up the Gremlin console to connect to a Neptune DB instance

The Gremlin Console allows you to experiment with TinkerPop graphs and queries in a REPL (read-eval-print loop) environment.

You can use the Gremlin Console to connect to a remote graph database. The following section walks you through installing and configuring the Gremlin Console to connect remotely to a Neptune DB instance. You must follow these instructions from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

Note

If you have [IAM authentication enabled \(p. 117\)](#) on your Neptune DB cluster, follow the instructions in [Connecting to Neptune Using the Gremlin Console with Signature Version 4 Signing \(p. 148\)](#) to install the Gremlin console rather than the instructions here.

To install the Gremlin Console and connect to Neptune

1. The Gremlin Console binaries require Java 8. Enter the following to install Java 8 on your EC2 instance.

```
sudo yum install java-1.8.0-devel
```

2. Enter the following to set Java 8 as the default runtime on your EC2 instance.

```
sudo /usr/sbin/alternatives --config java
```

When prompted, enter the number for Java 8.

3. Download the appropriate version of the Gremlin console from the Apache web site. You can check the [engine release page \(p. 718\)](#) for the Neptune engine version you are currently running to determine which Gremlin version it supports. For example, for version 3.4.8, you can download the [Gremlin console](#) from the [Apache Tinkerpop3](#) website onto your EC2 instance like this:

```
wget https://archive.apache.org/dist/tinkerpop/3.4.8/apache-tinkerpop-gremlin-console-3.4.8-bin.zip
```

4. Unzip the Gremlin Console zip file.

```
unzip apache-tinkerpop-gremlin-console-3.4.8-bin.zip
```

5. Change directories into the unzipped directory.

```
cd apache-tinkerpop-gremlin-console-3.4.8
```

6. Install the CA certificate. Gremlin Console requires a certificate to verify the remote certificate.

- a. Download the certificate:

```
wget https://www.amazontrust.com/repository/SFSRootCAG2.cer
```

- b. Create a directory for certificates:

```
mkdir /tmp/certs/
```

- c. Copy Java certificates into the new directory:

```
cp jre_path/lib/security/cacerts /tmp/certs/cacerts
```

- d. Add the Amazon certificate to the repository:

```
sudo keytool -importcert
```

```
-alias neptune-tests-ca \  
-keystore /tmp/certs/cacerts \  
-file /home/ec2-user/apache-tinkerpop-gremlin-console-3.4.8/  
SFSRootCAG2.cer \  
-noprompt \  
-storepass changeit
```

7. In the `conf` subdirectory of the extracted directory, create a file named `neptune-remote.yaml` with the following text. Replace `your-neptune-endpoint` with the hostname or IP address of your Neptune DB instance. The square brackets (`[]`) are required.

Note

For information about finding the hostname of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

```
hosts: [your-neptune-endpoint]  
port: 8182  
connectionPool: { enableSsl: true, trustStore: /tmp/certs/cacerts }  
serializer: { className:  
  org.apache.tinkerpop.gremlin.driver.ser.GryoMessageSerializerV3d0, config:  
  { serializeResultToString: true } }
```

8. In a terminal, navigate to the Gremlin Console directory (`apache-tinkerpop-gremlin-console-3.4.8`), and then enter the following command to run the Gremlin Console.

```
bin/gremlin.sh
```

You should see the following output:

```
  \,,,\n  (o o)\n-----o00o-(3)-o00o-----  
plugin activated: tinkerpop.server  
plugin activated: tinkerpop.utilities  
plugin activated: tinkerpop.tinkergraph  
gremlin>
```

You are now at the `gremlin>` prompt. You will enter the remaining steps at this prompt.

9. At the `gremlin>` prompt, enter the following to connect to the Neptune DB instance.

```
:remote connect tinkerpop.server conf/neptune-remote.yaml
```

10. At the `gremlin>` prompt, enter the following to switch to remote mode. This sends all Gremlin queries to the remote connection.

```
:remote console
```

11. Enter the following to send a query to the Gremlin Graph.

```
g.V().limit(1)
```

12. When you are finished, enter the following to exit the Gremlin Console.

```
:exit
```

Note

Use a semicolon (`;`) or a newline character (`\n`) to separate each statement.

Each traversal preceding the final traversal must end in `next()` to be executed. Only the data from the final traversal is returned.

For more information on the Neptune implementation of Gremlin, see [the section called "Gremlin standards compliance" \(p. 47\)](#).

Using the HTTPS REST endpoint to connect to a Neptune DB instance

Amazon Neptune provides an HTTPS endpoint for Gremlin queries. The REST interface is compatible with whatever Gremlin version your DB cluster is using (see the [engine release page \(p. 718\)](#) of the Neptune engine version you are running to determine which Gremlin release it supports).

Note

As discussed in [Encryption in Transit: Connecting to Neptune Using SSL/HTTPS \(p. 108\)](#), Neptune now requires that you connect using HTTPS instead of HTTP.

The following instructions walk you through connecting to the Gremlin endpoint using the `curl` command and HTTPS. You must follow these instructions from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

The HTTPS endpoint for Gremlin queries to a Neptune DB instance is `https://your-neptune-endpoint:port/gremlin`.

Note

For information about finding the hostname of your Neptune DB instance, see [Connecting to Amazon Neptune Endpoints \(p. 78\)](#).

To connect to Neptune using the HTTP REST endpoint

The following example uses `curl` to submit a Gremlin query through HTTP **POST**. The query is submitted in JSON format in the body of the post as the `gremlin` property.

```
curl -X POST -d '{"gremlin":"g.V().limit(1)}' https://your-neptune-endpoint:port/gremlin
```

This example returns the first vertex in the graph by using the `g.V().limit(1)` traversal. You can query for something else by replacing it with another Gremlin traversal.

Important

By default, the REST endpoint returns all results in a single JSON result set. If this result set is too large, an `OutOfMemoryError` exception can occur on the Neptune DB instance. You can avoid this by enabling chunked responses (results returned in a series of separate responses). See [Use optional HTTP trailing headers to enable multi-part Gremlin responses \(p. 266\)](#).

Although HTTP **POST** requests are recommended for sending Gremlin queries, it is also possible to use HTTP **GET** requests:

```
curl -G "https://your-neptune-endpoint:port?gremlin=g.V().count()"
```

Note

Neptune does not support the `bindings` property.

Use optional HTTP trailing headers to enable multi-part Gremlin responses

By default, the HTTP response to Gremlin queries is returned in a single JSON result set. In the case of a very large result set, this can cause an `OutOfMemoryError` exception on the DB instance.

However, you can enable *chunked* responses (responses that are returned in multiple separate parts). You do this by including a transfer-encoding (TE) trailers header (`te: trailers`) in your request. See [the MDN page about TE request headers](#) for more information about TE headers.

When a response is returned in multiple parts, it can be hard to diagnose a problem that occurs after the first part is received, since the first part arrives with an HTTP status code of 200 (OK). A subsequent failure usually results in a message body containing a corrupt response, at the end of which Neptune appends an error message.

To make detection and diagnosis of this kind of failure easier, Neptune also includes two new header fields within the trailing headers of every response chunk:

- `X-Neptune-Status` – contains the response code followed by a short name. For instance, in case of success the trailing header would be: `X-Neptune-Status: 200 OK`. In the case of failure, the response code would be one of the [Neptune engine error code \(p. 711\)](#), such as `X-Neptune-Status: 500 TimeLimitExceededException`.
- `X-Neptune-Detail` – is empty for successful requests. In the case of errors, it contains the JSON error message. Because only ASCII characters are allowed in HTTP header values, the JSON string is URL encoded.

Note

Neptune does not currently support gzip compression of chunked responses. If the client requests both chunked encoding and compression at the same time, Neptune skips the compression.

Java-based Gremlin clients to use with Amazon Neptune

You can use either of two open-source Java-based Gremlin clients with Amazon Neptune: the [Apache TinkerPop Java Gremlin client](#), or the [Gremlin client for Amazon Neptune](#).

Apache TinkerPop Java Gremlin client

If you can, always use the latest version of the [Apache TinkerPop Java Gremlin client](#) that your engine version supports. Newer versions contain numerous bug fixes which can improve the stability, performance and usability of the client.

The table below lists the earliest and latest versions of TinkerPop client supported by different Neptune engine versions:

Neptune Engine Version	Minimum TinkerPop Version	Maximum TinkerPop Version
1.1.0.0 (patches before 200366.0)	3.3.0	3.3.2
1.1.0.0 (patches from 200366.0 on)	3.4.0	3.4.1
1.0.1.1	3.3.0	3.3.2
1.0.1.2	3.4.0	3.4.1
1.0.1.3	3.4.0	3.4.1
1.0.2.0	3.4.0	3.4.1

Neptune Engine Version	Minimum TinkerPop Version	Maximum TinkerPop Version
1.0.2.1	3.4.0	3.4.1
1.0.2.2	3.4.0	3.4.3
1.0.3.0	3.4.0	3.4.3
1.0.3.0.R3	3.4.0	3.4.8
1.0.4.0	3.4.0	3.4.11
1.0.4.1	3.4.0	3.4.11
1.0.4.2	3.4.0	3.4.11
1.0.5.0	3.4.0	3.4.11
1.0.5.1	3.4.0	3.4.11

TinkerPop clients are usually backwards compatible within a series (3.3.x, for example, or 3.4.x). There are exceptional cases where backward compatibility has to be broken, so it's best to check the [TinkerPop upgrade recommendation](#) before upgrading to a new client version.

The client might not be able to use new steps or new features introduced in versions later than what the server supports, but you can expect existing queries and features to work unless the [upgrade recommendation](#) calls out a breaking change.

Note

Python users should avoid using TinkerPop version 3.4.9 because of a default timeout setting that requires direct configuration (see [TINKERPOP-2505](#)).

Gremlin Java client for Amazon Neptune

The Gremlin client for Amazon Neptune is an [open-source Java-based Gremlin client](#) that acts as a drop-in replacement for the standard TinkerPop Java client.

The Neptune Gremlin client is optimized for Neptune clusters. It lets you manage traffic distribution across multiple instances in a cluster, and adapts to changes in cluster topology when you add or remove a replica. You can even configure the client to distribute requests across a subset of instances in your cluster, based on role, instance type, availability zone (AZ), or tags associated with instances.

The [latest version of the Neptune Gremlin Java client](#) is available on Maven Central.

For more information about the Neptune Gremlin Java client, see [this blog post](#). For code samples and demos, check out the [client's GitHub project](#).

Using a Java client to connect to a Neptune DB instance

The following section walks you through the running of a complete Java sample that connects to a Neptune DB instance and performs a Gremlin traversal using the Apache TinkerPop Gremlin client.

These instructions must be followed from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

To connect to Neptune using Java

1. Install Apache Maven on your EC2 instance. First, enter the following to add a repository with a Maven package:

```
sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
```

Enter the following to set the version number for the packages:

```
sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
```

Then use **yum** to install Maven:

```
sudo yum install -y apache-maven
```

2. The Gremlin libraries require Java 8. Enter the following to install Java 8 on your EC2 instance:

```
sudo yum install java-1.8.0-devel
```

3. Enter the following to set Java 8 as the default runtime on your EC2 instance:

```
sudo /usr/sbin/alternatives --config java
```

When prompted, enter the number for Java 8 (2).

4. Enter the following to set Java 8 as the default compiler on your EC2 instance:

```
sudo /usr/sbin/alternatives --config javac
```

When prompted, enter the number for Java 8 (2).

5. Create a new directory named `gremlinjava`:

```
mkdir gremlinjava  
cd gremlinjava
```

6. In the `gremlinjava` directory, create a `pom.xml` file, and then open it in a text editor:

```
nano pom.xml
```

7. Copy the following into the `pom.xml` file and save it:

```
<project xmlns="https://maven.apache.org/POM/4.0.0"  
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://  
maven.apache.org/maven-v4_0_0.xsd">  
  <properties>  
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  </properties>  
  <modelVersion>4.0.0</modelVersion>  
  <groupId>com.amazonaws</groupId>  
  <artifactId>GremlinExample</artifactId>  
  <packaging>jar</packaging>  
  <version>1.0-SNAPSHOT</version>  
  <name>GremlinExample</name>  
  <url>https://maven.apache.org</url>  
  <dependencies>  
    <dependency>  
      <groupId>org.apache.tinkerpop</groupId>  
      <artifactId>gremlin-driver</artifactId>  
      <version>3.4.8</version>
```

```
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tinkerpop/gremlin-groovy -->
<dependency>
  <groupId>org.apache.tinkerpop</groupId>
  <artifactId>gremlin-groovy</artifactId>
  <version>3.4.8</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-jdk14</artifactId>
  <version>1.7.25</version>
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.0.2</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>1.3</version>
      <configuration>
        <executable>java</executable>
        <arguments>
          <argument>-classpath</argument>
          <classpath/>
          <argument>com.amazonaws.App</argument>
        </arguments>
        <mainClass>com.amazonaws.App</mainClass>
        <complianceLevel>1.8
        </complianceLevel>
        <killAfter>-1</killAfter>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>
```

Note

If you are modifying an existing Maven project, the required dependency is highlighted in the preceding code.

8. Download the CA certificate. The Gremlin Java SDK requires a certificate to verify the remote certificate.

```
wget https://www.amazontrust.com/repository/SFSRootCAG2.pem
```

9. Create subdirectories for the example source code (`src/main/java/com/amazonaws/`) by typing the following at the command line:

```
mkdir -p src/main/java/com/amazonaws/
```

10. In the `src/main/java/com/amazonaws/` directory, create a file named `App.java`, and then open it in a text editor.

```
nano src/main/java/com/amazonaws/App.java
```

11. Copy the following into the App.java file. Replace *your-neptune-endpoint* with the address of your Neptune DB instance. Do *not* include the `https://` prefix in the `addContactPoint` method.

Note

For information about finding the hostname of your Neptune DB instance, see [Connecting to Amazon Neptune Endpoints \(p. 78\)](#).

```
package com.amazonaws;
import org.apache.tinkerpop.gremlin.driver.Cluster;
import org.apache.tinkerpop.gremlin.driver.Client;
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal;
import static
    org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource.traversal;
import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
import org.apache.tinkerpop.gremlin.structure.T;

public class App
{
    public static void main( String[] args )
    {
        Cluster.Builder builder = Cluster.build();
        builder.addContactPoint("your-neptune-endpoint");
        builder.port(8182);
        builder.enableSsl(true);
        builder.keyCertChainFile("SFRootCAG2.pem");

        Cluster cluster = builder.create();

        GraphTraversalSource g =
        traversal().withRemote(DriverRemoteConnection.using(cluster));

        // Add a vertex.
        // Note that a Gremlin terminal step, e.g. iterate(), is required to make a
        request to the remote server.
        // The full list of Gremlin terminal steps is at https://tinkerpop.apache.org/
        docs/current/reference/#terminal-steps
        g.addV("Person").property("Name", "Justin").iterate();

        // Add a vertex with a user-supplied ID.
        g.addV("Custom Label").property(T.id, "CustomId1").property("name", "Custom id
        vertex 1").iterate();
        g.addV("Custom Label").property(T.id, "CustomId2").property("name", "Custom id
        vertex 2").iterate();

        g.addE("Edge Label").from(g.V("CustomId1")).to(g.V("CustomId2")).iterate();

        // This gets the vertices, only.
        GraphTraversal t = g.V().limit(3).elementMap();

        t.forEachRemaining(
            e -> System.out.println(t.toList())
        );

        cluster.close();
    }
}
```

12. Compile and run the sample using the following Maven command:


```
mvn compile exec:exec
```

The preceding example returns a map of the key and values of each property for the first two vertices in the graph by using the `g.V().limit(3).elementMap()` traversal. To query for something else, replace it with another Gremlin traversal with one of the appropriate ending methods.

Note

The final part of the Gremlin query, `.toList()`, is required to submit the traversal to the server for evaluation. If you don't include that method or another equivalent method, the query is not submitted to the Neptune DB instance.

You also must append an appropriate ending when you add a vertex or edge, such as when you use the `addV()` step.

The following methods submit the query to the Neptune DB instance:

- `toList()`
- `toSet()`
- `next()`
- `nextTraverser()`
- `iterate()`

Java example of connecting to a Neptune DB instance with reconnect logic

The following Java example demonstrates how to connect to the Gremlin client with reconnect logic to recover from an unexpected disconnect.

It has the following dependencies:

```
<dependency>
  <groupId>org.apache.tinkerpop</groupId>
  <artifactId>gremlin-driver</artifactId>
  <version>${gremlin.version}</version>
</dependency>

<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-sigv4-signer</artifactId>
  <version>${sig4.signer.version}</version>
</dependency>

<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-gremlin-java-sigv4</artifactId>
  <version>${sig4.signer.version}</version>
</dependency>

<dependency>
  <groupId>com.evanlennick</groupId>
  <artifactId>retry4j</artifactId>
  <version>0.15.0</version>
</dependency>
```

Here is the sample code:

```
public static void main(String args[]) {
```

```
boolean useIam = true;

// Create Gremlin cluster and traversal source
Cluster.Builder builder = Cluster.build()
    .addContactPoint(System.getenv("neptuneEndpoint"))
    .port(Integer.parseInt(System.getenv("neptunePort")))
    .enableSsl(true)
    .minConnectionPoolSize(1)
    .maxConnectionPoolSize(1)
    .serializer(Serializers.GRAPHBINARY_V1D0)
    .reconnectInterval(2000);

if (useIam) {
    builder = builder.channelizer(SigV4WebSocketChannelizer.class);
}

Cluster cluster = builder.create();

GraphTraversalSource g = AnonymousTraversalSource
    .traversal()
    .withRemote(DriverRemoteConnection.using(cluster));

// Configure retries
RetryConfig retryConfig = new RetryConfigBuilder()
    .retryOnCustomExceptionLogic(getRetryLogic())
    .withDelayBetweenTries(1000, ChronoUnit.MILLIS)
    .withMaxNumberOfTries(5)
    .withFixedBackoff()
    .build();

@SuppressWarnings("unchecked")
CallExecutor<Object> retryExecutor = new CallExecutorBuilder<Object>()
    .config(retryConfig)
    .build();

// Do lots of queries
for (int i = 0; i < 100; i++){
    String id = String.valueOf(i);

    @SuppressWarnings("unchecked")
    Callable<Object> query = () -> g.V(id)
        .fold()
        .coalesce(
            unfold(),
            addV("Person").property(T.id, id))
        .id().next();

    // Retry query
    // If there are connection failures, the Java Gremlin client will automatically
    // attempt to reconnect in the background, so all we have to do is wait and retry.
    Status<Object> status = retryExecutor.execute(query);

    System.out.println(status.getResult().toString());
}

cluster.close();
}

private static Function<Exception, Boolean> getRetryLogic() {

    return e -> {

        Class<? extends Exception> exceptionClass = e.getClass();

        StringWriter stringWriter = new StringWriter();
        String message = stringWriter.toString();
    }
}
```

```
if (RemoteConnectionException.class.isAssignableFrom(exceptionClass)){
    System.out.println("Retrying because RemoteConnectionException");
    return true;
}

// Check for connection issues
if (message.contains("Timed out while waiting for an available host") ||
    message.contains("Timed-out waiting for connection on Host") ||
    message.contains("Connection to server is no longer active") ||
    message.contains("Connection reset by peer") ||
    message.contains("SSL Engine closed already") ||
    message.contains("Pool is shutdown") ||
    message.contains("ExtendedClosedChannelException") ||
    message.contains("Broken pipe") ||
    message.contains(System.getenv("neptuneEndpoint")))
{
    System.out.println("Retrying because connection issue");
    return true;
};

// Concurrent writes can sometimes trigger a ConcurrentModificationException.
// In these circumstances you may want to backoff and retry.
if (message.contains("ConcurrentModificationException")) {
    System.out.println("Retrying because ConcurrentModificationException");
    return true;
}

// If the primary fails over to a new instance, existing connections to the old primary
will
// throw a ReadOnlyViolationException. You may want to back and retry.
if (message.contains("ReadOnlyViolationException")) {
    System.out.println("Retrying because ReadOnlyViolationException");
    return true;
}

System.out.println("Not a retrieable error");
return false;
};
}
```

Using Python to connect to a Neptune DB instance

The following section walks you through the running of a Python sample that connects to an Amazon Neptune DB instance and performs a Gremlin traversal.

You must follow these instructions from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

Before you begin, do the following:

- Download and install Python 3.6 or later from the [Python.org website](https://www.python.org/).
- Verify that you have **pip** installed. If you don't have **pip** or you're not sure, see [Do I need to install pip?](#) in the **pip** documentation.
- If your Python installation does not already have it, download futures as follows: `pip install futures`

To connect to Neptune using Python

1. Enter the following to install the `gremlinpython` package:

```
pip install gremlinpython --user
```

2. Create a file named `gremlinexample.py`, and then open it in a text editor.
3. Copy the following into the `gremlinexample.py` file. Replace `your-neptune-endpoint` with the address of your Neptune DB instance.

For information about finding the address of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

```
from __future__ import print_function # Python 2/3 compatibility

from gremlin_python import statics
from gremlin_python.structure.graph import Graph
from gremlin_python.process.graph_traversal import __
from gremlin_python.process.strategies import *
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection

graph = Graph()

remoteConn = DriverRemoteConnection('wss://your-neptune-endpoint:8182/gremlin', 'g')
g = graph.traversal().withRemote(remoteConn)

print(g.V().limit(2).toList())
remoteConn.close()
```

4. Enter the following command to run the sample:

```
python gremlinexample.py
```

The Gremlin query at the end of this example returns the vertices (`g.V().limit(2)`) in a list. This list is then printed with the standard Python `print` function.

Note

The final part of the Gremlin query, `toList()`, is required to submit the traversal to the server for evaluation. If you don't include that method or another equivalent method, the query is not submitted to the Neptune DB instance.

The following methods submit the query to the Neptune DB instance:

- `toList()`
- `toSet()`
- `next()`
- `nextTraverser()`
- `iterate()`

The preceding example returns the first two vertices in the graph by using the `g.V().limit(2).toList()` traversal. To query for something else, replace it with another Gremlin traversal with one of the appropriate ending methods.

Using .NET to connect to a Neptune DB instance

The following section contains a code example written in C# that connects to a Neptune DB instance and performs a Gremlin traversal.

Connections to Amazon Neptune must be from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance. This sample code was tested on an Amazon EC2 instance running Ubuntu.

Before you begin, do the following:

- Install .NET on the Amazon EC2 instance. To get instructions for installing .NET on multiple operating systems, including Windows, Linux, and macOS, see [Get Started with .NET](#).
- Install Gremlin.NET by running `dotnet add package gremlin.net` for your package. For more information, see [Gremlin.NET](#) in the TinkerPop documentation.

To connect to Neptune using Gremlin.NET

1. Create a new .NET project.

```
dotnet new console -o gremlinExample
```

2. Change directories into the new project directory.

```
cd gremlinExample
```

3. Copy the following into the `Program.cs` file. Replace *your-neptune-endpoint* with the address of your Neptune DB instance.

For information about finding the address of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Gremlin.Net;
using Gremlin.Net.Driver;
using Gremlin.Net.Driver.Remote;
using Gremlin.Net.Structure;
using static Gremlin.Net.Process.Traversal.AnonymousTraversalSource;
namespace gremlinExample
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                var endpoint = "your-neptune-endpoint";
                // This uses the default Neptune and Gremlin port, 8182
                var gremlinServer = new GremlinServer(endpoint, 8182, enableSsl: true );
                var gremlinClient = new GremlinClient(gremlinServer);
                var remoteConnection = new DriverRemoteConnection(gremlinClient, "g");
                var g = Traversal().WithRemote(remoteConnection);
                g.AddV("Person").Property("Name", "Justin").Iterate();
                g.AddV("Custom Label").Property("name", "Custom id vertex 1").Iterate();
                g.AddV("Custom Label").Property("name", "Custom id vertex 2").Iterate();
                var output = g.V().Limit<Vertex>(3).ToList();
                foreach(var item in output) {
```

```
        Console.WriteLine(item);
    }
}
catch (Exception e)
{
    Console.WriteLine("{0}", e);
}
}
}
```

4. Enter the following command to run the sample:

```
dotnet run
```

The Gremlin query at the end of this example returns the count of a single vertex for testing purposes. It is then printed to the console.

Note

The final part of the Gremlin query, `next()`, is required to submit the traversal to the server for evaluation. If you don't include that method or another equivalent method, the query is not submitted to the Neptune DB instance.

The following methods submit the query to the Neptune DB instance:

- `toList()`
- `toSet()`
- `next()`
- `nextTraverser()`
- `iterate()`

Use `next()` if you need the query results to be serialized and returned, or `iterate()` if you don't.

The preceding example returns a list by using the `g.V().Limit(3).ToList()` traversal. To query for something else, replace it with another Gremlin traversal with one of the appropriate ending methods.

Using Node.js to connect to a Neptune DB instance

The following section walks you through the running of a Node.js sample that connects to an Amazon Neptune DB instance and performs a Gremlin traversal.

You must follow these instructions from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

Before you begin, do the following:

- Verify that Node.js version 8.11 or higher is installed. If it is not, download and install Node.js from the [Nodejs.org website](https://nodejs.org).

To connect to Neptune using Node.js

1. Enter the following to install the `gremlin-javascript` package:

```
npm install gremlin
```

2. Create a file named `gremlinexample.js` and open it in a text editor.
3. Copy the following into the `gremlinexample.js` file. Replace `your-neptune-endpoint` with the address of your Neptune DB instance.

For information about finding the address of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

```
const gremlin = require('gremlin');
const DriverRemoteConnection = gremlin.driver.DriverRemoteConnection;
const Graph = gremlin.structure.Graph;

dc = new DriverRemoteConnection('wss://your-neptune-endpoint:8182/gremlin',{});

const graph = new Graph();
const g = graph.traversal().withRemote(dc);

g.V().limit(1).count().next().
  then(data => {
    console.log(data);
    dc.close();
  }).catch(error => {
    console.log('ERROR', error);
    dc.close();
  });
```

4. Enter the following command to run the sample:

```
node gremlinexample.js
```

The preceding example returns the count of a single vertex in the graph by using the `g.V().limit(1).count().next()` traversal. To query for something else, replace it with another Gremlin traversal with one of the appropriate ending methods.

Note

The final part of the Gremlin query, `next()`, is required to submit the traversal to the server for evaluation. If you don't include that method or another equivalent method, the query is not submitted to the Neptune DB instance.

The following methods submit the query to the Neptune DB instance:

- `toList()`
- `toSet()`
- `next()`
- `nextTraverser()`
- `iterate()`

Use `next()` if you need the query results to be serialized and returned, or `iterate()` if you don't.

Important

This is a standalone Node.js example. If you are planning to run code like this in an AWS Lambda function, see [Lambda function examples \(p. 511\)](#) for details about using JavaScript efficiently in a Neptune Lambda function.

Gremlin query hints

You can use query hints to specify optimization and evaluation strategies for a particular Gremlin query in Amazon Neptune.

Query hints are specified by adding a `withSideEffect` step to the query with the following syntax.

```
g.withSideEffect(hint, value)
```

- *hint* – Identifies the type of the hint to apply.
- *value* – Determines the behavior of the system aspect under consideration.

For example, the following shows how to include a `repeatMode` hint in a Gremlin traversal.

Note

All Gremlin query hints side effects are prefixed with `Neptune#`.

```
g.withSideEffect('Neptune#repeatMode',  
'DFS').V("3").repeat(out()).times(10).limit(1).path()
```

The preceding query instructs the Neptune engine to traverse the graph *Depth First* (DFS) rather than the default Neptune, *Breadth First* (BFS).

The following sections provide more information about the available query hints and their usage.

Topics

- [Gremlin repeatMode query hint \(p. 279\)](#)
- [Gremlin noReordering query hint \(p. 281\)](#)
- [Gremlin typePromotion query hint \(p. 281\)](#)
- [Gremlin useDFE query hint \(p. 282\)](#)
- [Gremlin query hints for using the results cache \(p. 282\)](#)

Gremlin repeatMode query hint

The Neptune `repeatMode` query hint specifies how the Neptune engine evaluates the `repeat()` step in a Gremlin traversal: breadth first, depth first, or chunked depth first.

The evaluation mode of the `repeat()` step is important when it is used to find or follow a path, rather than simply repeating a step a limited number of times.

Syntax

The `repeatMode` query hint is specified by adding a `withSideEffect` step to the query.

```
g.withSideEffect('Neptune#repeatMode', 'mode').gremlin-traversal
```

Note

All Gremlin query hints side effects are prefixed with `Neptune#`.

Available Modes

- BFS

Breadth-First Search

Default execution mode for the `repeat()` step. This gets all sibling nodes before going deeper along the path.

This version is memory-intensive and frontiers can get very large. There is a higher risk that the query will run out of memory and be cancelled by the Neptune engine. This most closely matches other Gremlin implementations.

- `DFS`

Depth-First Search

Follows each path to the maximum depth before moving on to the next solution.

This uses less memory. It may provide better performance in situations like finding a single path from a starting point out multiple hops.

- `CHUNKED_DFS`

Chunked Depth-First Search

A hybrid approach that explores the graph depth-first in chunks of 1,000 nodes, rather than 1 node (`DFS`) or all nodes (`BFS`).

The Neptune engine will get up to 1,000 nodes at each level before following the path deeper.

This is a balanced approach between speed and memory usage.

It is also useful if you want to use `BFS`, but the query is using too much memory.

Example

The following section describes the effect of the repeat mode on a Gremlin traversal.

In Neptune the default mode for the `repeat()` step is to perform a breadth-first (`BFS`) execution strategy for all traversals.

In most cases, the TinkerGraph implementation uses the same execution strategy, but in some cases it alters the execution of a traversal.

For example, the TinkerGraph implementation modifies the following query.

```
g.V("3").repeat(out()).times(10).limit(1).path()
```

The `repeat()` step in this traversal is "unrolled" into the following traversal, which results in a depth-first (`DFS`) strategy.

```
g.V(<id>).out().out().out().out().out().out().out().out().out().out().limit(1).path()
```

Important

The Neptune query engine does not do this automatically.

Breadth-first (`BFS`) is the default execution strategy, and is similar to TinkerGraph in most cases. However, there are certain cases where depth-first (`DFS`) strategies are preferable.

BFS (Default)

Breadth-first (`BFS`) is the default execution strategy for the `repeat()` operator.

```
g.V("3").repeat(out()).times(10).limit(1).path()
```

The Neptune engine fully explores the first nine-hop frontiers before finding a solution ten hops out. This is effective in many cases, such as a shortest-path query.

However, for the preceding example, the traversal would be much faster using the depth-first (DFS) mode for the `repeat()` operator.

DFS

The following query uses the depth-first (DFS) mode for the `repeat()` operator.

```
g.withSideEffect("Neptune#repeatMode", "DFS").V("3").repeat(out()).times(10).limit(1)
```

This follows each individual solution out to the maximum depth before exploring the next solution.

Gremlin `noReordering` query hint

When you submit a Gremlin traversal, the Neptune query engine investigates the structure of the traversal and reorders parts of the query, trying to minimize the amount of work required for evaluation and query response time. For example, a traversal with multiple constraints, such as multiple `has()` steps, is typically not evaluated in the given order. Instead it is reordered after the query is checked with static analysis.

The Neptune query engine tries to identify which constraint is more selective and runs that one first. This often results in better performance, but the order in which Neptune chooses to evaluate the query might not always be optimal.

If you know the exact characteristics of the data and want to manually dictate the order of the query execution, you can use the Neptune `noReordering` query hint to specify that the traversal be evaluated in the order given.

Syntax

The `noReordering` query hint is specified by adding a `withSideEffect` step to the query.

```
g.withSideEffect('Neptune#noReordering', true or false).gremlin-traversal
```

Note

All Gremlin query hints side effects are prefixed with `Neptune#`.

Available Values

- `true`
- `false`

Gremlin `typePromotion` query hint

When you submit a Gremlin traversal that filters on a numerical value or range, the Neptune query engine must normally use type promotion when it executes the query. This means that it has to examine values of every type that could hold the value you are filtering on.

For example, if you are filtering for values equal to 55, the engine must look for integers equal to 55, long integers equal to 55L, floats equal to 55.0, and so forth. Each type promotion requires an additional lookup on storage, which can cause an apparently simple query to take an unexpectedly long time to complete.

Let's say you are searching for all vertexes with a customer-age property greater than 5:

```
g.V().has('customerAge', gt(5))
```

To execute that traversal thoroughly, Neptune must expand the query to examine every numeric type that the value you are querying for could be promoted to. In this case, the `gt` filter has to be applied for any integer over 5, any long over 5L, any float over 5.0, and any double over 5.0. Because each of these type promotions requires an additional lookup on storage, you will see multiple filters per numeric filter when you run the [Gremlin profile API \(p. 303\)](#) for this query, and it will take significantly longer to complete than you might expect.

Often type promotion is unnecessary because you know in advance that you only need to find values of one specific type. When this is the case, you can speed up your queries dramatically by using the `typePromotion` query hint to turn off type promotion.

Syntax

The `typePromotion` query hint is specified by adding a `withSideEffect` step to the query.

```
g.withSideEffect('Neptune#typePromotion', true or false).gremlin-traversal
```

Note

All Gremlin query hints side effects are prefixed with `Neptune#`.

Available Values

- `true`
- `false`

To turn off type promotion for the query above, you would use:

```
g.withSideEffect('Neptune#typePromotion', false).V().has('customerAge', gt(5))
```

Gremlin useDFE query hint

Use this query hint to disable use of the DFE alternative query engine for executing the query. Because the default value of this hint is `true`, there is no need to include the hint unless you do not want the DFE to be used to execute the query.

Note

If you use this query hint when the DFE is not enabled in Lab Mode, an error will result.

Example of disabling the DFE for a query:

```
g.withSideEffect('Neptune#useDFE', false).V().out()
```

Gremlin query hints for using the results cache

The following query hints can be used when the [query results cache \(p. 288\)](#) is enabled.

Gremlin enableResultCache query hint

The `enableResultCache` query hint with a value of `true` causes query results to be returned from the cache if they have already been cached. If not, it returns new results and caches them until such time as they are cleared from the cache. For example:

```
g.with('Neptune#enableResultCache', true)
```

```
.V().has('genre', 'drama').in('likes')
```

Later, you can access the cached results by issuing exactly the same query again.

If the value of this query hint is `false`, or if it isn't present, query results are not cached. However, setting it to `false` does not clear existing cached results. To clear cached results, use the `invalidateResultCache` or `invalidateResultCachekey` hint.

Gremlin `enableResultCacheWithTTL` query hint

The `enableResultCacheWithTTL` query hint also returns cached results if there are any, without affecting the TTL of results already in the cache. If there are currently no cached results, the query returns new results and caches them for the time to live (TTL) specified by the `enableResultCacheWithTTL` query hint. That time to live is specified in seconds. For example, the following query specifies a time to live of sixty seconds:

```
g.with('Neptune#enableResultCacheWithTTL', 60)
.V().has('genre', 'drama').in('likes')
```

Before the 60-second time-to-live is over, you can use the same query (here, `g.V().has('genre', 'drama').in('likes')`) with either the `enableResultCache` or the `enableResultCacheWithTTL` query hint to access the cached results.

Note

The time to live specified with `enableResultCacheWithTTL` does not affect results that have already been cached.

- If results were previously cached using `enableResultCache`, the cache must first be explicitly cleared before `enableResultCacheWithTTL` generates new results and caches them for the TTL that it specifies.
- If results were previously cached using `enableResultCacheWithTTL`, that previous TTL must first expire before `enableResultCacheWithTTL` generates new results and caches them for the TTL that it specifies.

After the time to live has passed, the cached results for the query are cleared, and a subsequent instance of the same query then returns new results. If `enableResultCacheWithTTL` is attached to that subsequent query, the new results are cached with the TTL that it specifies.

Gremlin `invalidateResultCacheKey` query hint

The `invalidateResultCacheKey` query hint can take a `true` or `false` value. A `true` value causes cached results for the the query to which `invalidateResultCacheKey` is attached to be cleared. For example, the following example causes results cached for the query key `g.V().has('genre', 'drama').in('likes')` to be cleared:

```
g.with('Neptune#invalidateResultCacheKey', true)
.V().has('genre', 'drama').in('likes')
```

The example query above does not cause its new results to be cached. You can include `enableResultCache` (or `enableResultCacheWithTTL`) in the same query if you want to cache the new results after clearing the existing cached ones:

```
g.with('Neptune#enableResultCache', true)
.with('Neptune#invalidateResultCacheKey', true)
.V().has('genre', 'drama').in('likes')
```

Gremlin `invalidateResultCache` query hint

The `invalidateResultCache` query hint can take a `true` or `false` value. A `true` value causes all results in the results cache to be cleared. For example:

```
g.with('Neptune#invalidateResultCache', true)
  .V().has('genre', 'drama').in('likes')
```

The example query above does not cause its results to be cached. You can include `enableResultCache` (or `enableResultCacheWithTTL`) in the same query if you want to cache new results after completely clearing the existing cache:

```
g.with('Neptune#enableResultCache', true)
  .with('Neptune#invalidateResultCache', true)
  .V().has('genre', 'drama').in('likes')
```

Gremlin `numResultsCached` query hint

The `numResultsCached` query hint can only be used with queries that contain `iterate()`, and it specifies the maximum number of results to cache for the query to which it is attached. Note that the results cached when `numResultsCached` is present are not returned, only cached.

For example, the following query specifies that up to 100 of its results should be cached, but none of those cached results returned:

```
g.with('Neptune#enableResultCache', true)
  .with('Neptune#numResultsCached', 100)
  .V().has('genre', 'drama').in('likes').iterate()
```

You can then use a query like the following to retrieve a range of the cached results (here, the first ten):

```
g.with('Neptune#enableResultCache', true)
  .with('Neptune#numResultsCached', 100)
  .V().has('genre', 'drama').in('likes').range(0, 10)
```

Gremlin `noCacheExceptions` query hint

The `noCacheExceptions` query hint can take a `true` or `false` value. A `true` value causes any exceptions related to the results cache to be suppressed. For example:

```
g.with('Neptune#enableResultCache', true)
  .with('Neptune#noCacheExceptions', true)
  .V().has('genre', 'drama').in('likes')
```

In particular, this suppresses the `QueryLimitExceededException`, which is raised if the results of a query are too large to fit in the results cache.

Gremlin query status API

To get the status of Gremlin queries, use HTTP `GET` or `POST` to make a request to the `https://your-neptune-endpoint:port/gremlin/status` endpoint.

Gremlin query status request parameters

- **queryId** (*optional*) – The ID of a running Gremlin query. Only displays the status of the specified query.

- **includeWaiting** (*optional*) – Returns the status of all waiting queries.

Normally, only running queries are included in the response, but when the `includeWaiting` parameter is specified, the status of all waiting queries is also returned.

Gremlin query status response syntax

```
{
  "acceptedQueryCount": integer,
  "runningQueryCount": integer,
  "queries": [
    {
      "queryId": "guid",
      "queryEvalStats": {
        "waited": integer,
        "elapsed": integer,
        "cancelled": boolean
      },
      "queryString": "string"
    }
  ]
}
```

Gremlin query status response values

- **acceptedQueryCount** – The number of queries that have been accepted but not yet completed, including queries in the queue.
- **runningQueryCount** – The number of currently running Gremlin queries.
- **queries** – A list of the current Gremlin queries.
- **queryId** – A GUID id for the query. Neptune automatically assigns this ID value to each query, or you can also assign your own ID (see [Inject a Custom ID Into a Neptune Gremlin or SPARQL Query \(p. 83\)](#)).
- **queryEvalStats** – Statistics for this query.
- **subqueries** – The number of subqueries in this query.
- **elapsed** – The number of milliseconds the query has been running so far.
- **cancelled** – True indicates that the query was cancelled.
- **queryString** – The submitted query. This is truncated to 1024 characters if it is longer than that.
- **waited** – Indicates how long the query waited, in microseconds.

Gremlin query status example

The following is an example of the status command using `curl` and HTTP `GET`.

```
curl https://your-neptune-endpoint:port/gremlin/status
```

This output shows a single running query.

```
{
  "acceptedQueryCount": 9,
  "runningQueryCount": 1,
  "queries": [
    {
      "queryId": "fb34cd3e-f37c-4d12-9cf2-03bb741bf54f",
```

```
    "queryEvalStats":
      {
        "waited": 0,
        "elapsed": 23,
        "cancelled": false
      },
    "queryString": "g.V().addV()"
  }
]
```

Gremlin query cancellation

To get the status of Gremlin queries, use HTTP GET or POST to make a request to the `https://your-neptune-endpoint:port/gremlin/status` endpoint.

Gremlin query cancellation request parameters

- **cancelQuery** – Required for cancellation. This parameter has no corresponding value.
- **queryId** – The ID of the running Gremlin query to cancel.

Gremlin query cancellation example

The following is an example of the `curl` command to cancel a query.

```
curl https://your-neptune-endpoint:port/gremlin/status \
  --data-urlencode "cancelQuery" \
  --data-urlencode "queryId=fb34cd3e-f37c-4d12-9cf2-03bb741bf54f"
```

Successful cancellation returns HTTP 200 OK.

Support for Gremlin sessions

You can use Gremlin sessions with implicit transactions in Amazon Neptune. For information about Gremlin sessions, see [Considering Sessions](#) in the Apache TinkerPop documentation. The sections below describe how to use Gremlin sessions with Java.

Note

This feature is available starting in [Neptune engine release 1.0.1.0.200463.0 \(p. 767\)](#).

Important

Currently, the longest time Neptune can keep a session open is 10 minutes. If you don't close a session before that, the session times out and everything in it is rolled back.

Also, currently, you can only use string script submission with sessions, not bytecode.

Topics

- [Gremlin sessions on the Gremlin console \(p. 286\)](#)
- [Gremlin sessions in the Gremlin Language Variant \(p. 287\)](#)

Gremlin sessions on the Gremlin console

If you create a remote connection on the Gremlin Console without the `session` parameter, the remote connection is created in `sessionless` mode. In this mode, each request that is submitted to the server is

treated as a complete transaction in itself, and no state is saved between requests. If a request fails, only that request is rolled back.

If you create a remote connection that *does* use the `session` parameter, you create a session that lasts until you close the remote connection. Every session is identified by a unique UUID that the console generates and returns to you.

The following is an example of one console call that creates a session. After queries are submitted, another call closes the session and commits the queries.

```
gremlin> :remote connect tinkerspop.server conf/neptune-remote.yaml session
: : :
: : :
gremlin> :remote close
```

For more information and examples, see [Sessions](#) in the TinkerPop documentation.

All the queries that you run during a session form a single transaction that isn't committed until all the queries succeed and you close the remote connection. If a query fails, or if you don't close the connection within the maximum session lifetime that Neptune supports, the session transaction is not committed, and all the queries in it are rolled back.

Gremlin sessions in the Gremlin Language Variant

In the Gremlin language variant (GLV), you need to create a `SessionedClient` object to issue multiple queries in a single transaction, as in the following example.

```
Cluster cluster = Cluster.open();           // line 1
Client client = cluster.connect("sessionName"); // line 2
: : :
: : :
client.close( );
```

Line 2 in the preceding example creates the `SessionedClient` object according to the configuration options set for the cluster in question. The `sessionName` string that you pass to the connect method becomes the unique name of the session. To avoid collisions, use a UUID for the name.

The client starts a session transaction when it is initialized. All the queries that you run during the session form are committed only when you call `client.close()`. Again, if a single query fails, or if you don't close the connection within the maximum session lifetime that Neptune supports, the session transaction fails, and all the queries in it are rolled back.

Using the Gremlin API with Amazon Neptune

Note

Amazon Neptune does not support the `bindings` property.

Gremlin HTTPS requests all use a single endpoint: `https://your-neptune-endpoint:port/gremlin`. All Neptune connections must use HTTPS.

You can connect the Gremlin Console to a Neptune graph directly through WebSockets.

For more information about connecting to the Gremlin endpoint, see [Accessing a Neptune graph with Gremlin \(p. 262\)](#).

The Amazon Neptune implementation of Gremlin has specific details and differences that you need to consider. For more information, see [Gremlin standards compliance in Amazon Neptune \(p. 47\)](#).

For information about the Gremlin language and traversals, see [The Traversal](#) in the Apache TinkerPop documentation.

Caching query results in Amazon Neptune Gremlin

Starting in [engine release 1.0.5.1 \(p. 722\)](#), Amazon Neptune supports a results cache for Gremlin queries.

You can enable the query results cache and then use a query hint to cache the results of a Gremlin read-only query.

Any re-run of the query then retrieves the cached results with low latency and no I/O costs, as long as they are still in the cache. This works for queries submitted both on an HTTP endpoint and using Websockets, either as byte-code or in string form.

You can control how the Neptune query results cache behaves in several ways. For example:

- You can get cached results paginated, in blocks.
- You can specify the time-to-live (TTL) for specified queries.
- You can clear the cache for specified queries.
- You can clear the entire cache.
- You can set up to be notified if results exceed the cache size.

The cache is maintained using a least-recently-used (LRU) policy, meaning that once the space allotted to the cache is full, the least-recently-used results are removed to make room when new results are being cached.

Important

The query-results cache is not available on `t2.medium` or `t3.medium` instance types.

Enabling the query results cache in Neptune

To enable the query results cache in Neptune, use the console to set the `neptune_result_cache` DB instance parameter to `1` (enabled).

Once the results cache is enabled, Neptune sets aside a portion of current memory for caching query results. The larger the instance type you're using and the more memory is available, the more memory Neptune sets aside for the cache.

If the results cache memory fills up, Neptune automatically drops least-recently-used (LRU) cached results to make way for new ones.

You can check the current status of the results cache using the [Instance Status \(p. 639\)](#) command.

Using hints to cache query results

Once the query results cache is enabled, you use query hints to control query caching. All the examples below apply to the same query traversal, namely:

```
g.V().has('genre','drama').in('likes')
```

Using `enableResultCache`

With the query results cache enabled, you can cache the results of a Gremlin query using the `enableResultCache` query hint, as follows:

```
g.with('Neptune#enableResultCache', true)
.V().has('genre', 'drama').in('likes')
```

Neptune then returns the query results to you, and also caches them. Later, you can access the cached results by issuing exactly the same query again:

```
g.with('Neptune#enableResultCache', true)
.V().has('genre', 'drama').in('likes')
```

The cache key that identifies the cached results is the query string itself, namely:

```
g.V().has('genre', 'drama').in('likes')
```

Using `enableResultCacheWithTTL`

You can specify how long the query results should be cached for by using the `enableResultCacheWithTTL` query hint. For example, the following query specifies that the query results should expire after 120 seconds:

```
g.with('Neptune#enableResultCacheWithTTL', 120)
.V().has('genre', 'drama').in('likes')
```

Again, the cache key that identifies the cached results is the base query string:

```
g.V().has('genre', 'drama').in('likes')
```

And again, you can access the cached results using that query string with the `enableResultCache` query hint:

```
g.with('Neptune#enableResultCache', true)
.V().has('genre', 'drama').in('likes')
```

If 120 or more seconds have passed since the results were cached, that query will return new results, and cache them, without any time-to-live.

You can also access the cached results by issuing the same query again with the `enableResultCacheWithTTL` query hint. For example:

```
g.with('Neptune#enableResultCacheWithTTL', 140)
.V().has('genre', 'drama').in('likes')
```

Until 120 seconds have passed (that is, the TTL currently in effect), this new query using the `enableResultCacheWithTTL` query hint returns the cached results. After 120 seconds, it would return new results and cache them with a time-to-live of 140 seconds.

Note

If results for a query key are already cached, then the same query key with `enableResultCacheWithTTL` does not generate new results and has no effect on the time-to-live of the currently cached results.

- If results were previously cached using `enableResultCache`, the cache must first be cleared before `enableResultCacheWithTTL` generates new results and caches them for the TTL that it specifies.

- If results were previously cached using `enableResultCacheWithTTL`, that previous TTL must first expire before `enableResultCacheWithTTL` generates new results and caches them for the TTL that it specifies.

Using `invalidateResultCacheKey`

You can use the `invalidateResultCacheKey` query hint to clear cached results for one particular query. For example:

```
g.with('Neptune#invalidateResultCacheKey', true)
.V().has('genre', 'drama').in('likes')
```

That query clears the cache for the query key, `g.V().has('genre', 'drama').in('likes')`, and returns new results for that query.

You can also combine `invalidateResultCacheKey` with `enableResultCache` or `enableResultCacheWithTTL`. For example, the following query clears the current cached results, caches new results, and returns them:

```
g.with('Neptune#enableResultCache', true)
.with('Neptune#invalidateResultCacheKey', true)
.V().has('genre', 'drama').in('likes')
```

Using `invalidateResultCache`

You can use the `invalidateResultCache` query hint to clear all cached results in the query result cache. For example:

```
g.with('Neptune#invalidateResultCache', true)
.V().has('genre', 'drama').in('likes')
```

That query clears the entire result cache and returns new results for the query.

You can also combine `invalidateResultCache` with `enableResultCache` or `enableResultCacheWithTTL`. For example, the following query clears the entire results cache, caches new results for this query, and returns them:

```
g.with('Neptune#enableResultCache', true)
.with('Neptune#invalidateResultCache', true)
.V().has('genre', 'drama').in('likes')
```

Paginating cached query results

Suppose you have already cached a large number of results like this:

```
g.with('Neptune#enableResultCache', true)
.V().has('genre', 'drama').in('likes')
```

Now suppose you issue the following range query:

```
g.with('Neptune#enableResultCache', true)
.V().has('genre', 'drama').in('likes').range(0,10)
```

Neptune first looks for the full cache key, namely `g.V().has('genre', 'drama').in('likes').range(0,10)`. If that key doesn't exist,

Neptune next looks to see if there is a key for that query string without the range (namely `g.V().has('genre', 'drama').in('likes')`). When it finds that key, Neptune then fetches the first ten results from its cache, as the range specifies.

Note

If you use the `invalidateResultCacheKey` query hint with a query that has a range at the end, Neptune clears the cache for a query without the range if it doesn't find an exact match for the query with the range.

Using `numResultsCached` with `.iterate()`

Using the `numResultsCached` query hint, you can populate the results cache without returning all the results being cached, which can be useful when you prefer to paginate a large number of results.

The `numResultsCached` query hint only works with queries that end with `iterate()`.

For example, if you want to cache the first 50 results of the sample query:

```
g.with("Neptune#enableResultCache", true)
  .with("Neptune#numResultsCached", 50)
  .V().has('genre', 'drama').in('likes').iterate()
```

In this case the query key in the cache is: `g.with("Neptune#numResultsCached", 50).V().has('genre', 'drama').in('likes')`. You can now retrieve the first ten of the cached results with this query:

```
g.with("Neptune#enableResultCache", true)
  .with("Neptune#numResultsCached", 50)
  .V().has('genre', 'drama').in('likes').range(0, 10)
```

And, you can retrieve the next ten results from the query as follows:

```
g.with("Neptune#enableResultCache", true)
  .with("Neptune#numResultsCached", 50)
  .V().has('genre', 'drama').in('likes').range(10, 20)
```

Don't forget to include the `numResultsCached` hint! It is an essential part of the query key and must therefore be present in order to access the cached results.

Some things to keep in mind when using `numResultsCached`

- **The number you supply with `numResultsCached` is applied at the end of the query.** This means, for example, that the following query actually caches results in the range (1000, 1500):

```
g.with("Neptune#enableResultCache", true)
  .with("Neptune#numResultsCached", 500)
  .V().range(1000, 2000).iterate()
```

- **The number you supply with `numResultsCached` specifies the maximum number of results to cache.** This means, for example, that the following query actually caches results in the range (1000, 2000):

```
g.with("Neptune#enableResultCache", true)
  .with("Neptune#numResultsCached", 100000)
  .V().range(1000, 2000).iterate()
```

- **Results cached by queries that end with `.range().iterate()` have their own range.** For example, suppose you cache results using a query like this:

```
g.with("Neptune#enableResultCache", true)
  .with("Neptune#numResultsCached", 500)
  .V().range(1000, 2000).iterate()
```

To retrieve the first 100 results from the cache, you would write a query like this:

```
g.with("Neptune#enableResultCache", true)
  .with("Neptune#numResultsCached", 500)
  .V().range(1000, 2000).range(0, 100)
```

Those hundred results would be equivalent to results from the base query in the range (1000, 1100).

The query cache keys used to locate cached results

After the results of a query have been cached, subsequent queries with the same *query cache key* retrieve results from the cache rather than generating new ones. The query cache key of a query is evaluated as follows:

1. Query hints are ignored, except for `numResultsCached`.
2. A final `iterate()` step is ignored.
3. The rest of the query is ordered according to its byte-code representation.

The resulting string is matched against an index of the query results already in the cache to determine whether there is a cache hit for the query.

Exceptions related to the results cache

If the results of a query that you are trying to cache are too large to fit in the cache memory even after removing everything previously cached, Neptune raises a `QueryLimitExceededException` fault. No results are returned, and the exception generates the following error message:

```
The result size is larger than the allocated cache,
  please refer to results cache best practices for options to rerun the query.
```

You can suppress this message using the `noCacheExceptions` query hint, as follows:

```
g.with('Neptune#enableResultCache', true)
  .with('Neptune#noCacheExceptions', true)
  .V().has('genre', 'drama').in('likes')
```

Analyzing Neptune query execution using Gremlin explain

Amazon Neptune has added a Gremlin feature named *explain*. This feature is a self-service tool for understanding the execution approach taken by the Neptune engine. You invoke it by adding an `explain` parameter to an HTTP call that submits a Gremlin query.

The `explain` feature provides information about the logical structure of query execution plans. You can use this information to identify potential evaluation and execution bottlenecks and tune your query, as

explained in [Tuning Gremlin queries \(p. 322\)](#). You can also use [query hints \(p. 278\)](#) to improve query execution plans.

Note

This feature is available starting with [Release 1.0.1.0.200463.0 \(2019-10-15\) \(p. 767\)](#).

Topics

- [Understanding how Gremlin queries work in Neptune \(p. 293\)](#)
- [Using the Gremlin explain API in Neptune \(p. 297\)](#)
- [Gremlin profile API in Neptune \(p. 303\)](#)
- [Tuning Gremlin queries using explain and profile \(p. 322\)](#)
- [Native Gremlin step support in Amazon Neptune \(p. 336\)](#)

Understanding how Gremlin queries work in Neptune

To take full advantage of the Gremlin `explain` and `profile` reports in Amazon Neptune, it is helpful to understand some background information about Gremlin queries.

Topics

- [Gremlin statements in Neptune \(p. 293\)](#)
- [How Neptune processes Gremlin queries using statement indexes \(p. 295\)](#)
- [How Gremlin queries are processed in Neptune \(p. 296\)](#)

Gremlin statements in Neptune

Property graph data in Amazon Neptune is composed of four-position (quad) statements. Each of these statements represents an individual atomic unit of property graph data. For more information, see [Neptune Graph Data Model \(p. 61\)](#). Similar to the Resource Description Framework (RDF) data model, these four positions are as follows:

- `subject (S)`
- `predicate (P)`
- `object (O)`
- `graph (G)`

Each statement is an assertion about one or more resources. For example, a statement can assert the existence of a relationship between two resources, or it can attach a property (key-value pair) to some resource.

You can think of the predicate as the verb of the statement, describing the type of relationship or property. The object is the target of the relationship, or the value of the property. The graph position is optional and can be used in many different ways. For the Neptune property graph (PG) data, it is either unused (null graph) or it is used to represent the identifier for an edge. A set of statements with shared resource identifiers creates a graph.

There are three classes of statements in the Neptune property graph data model:

Topics

- [Gremlin Vertex Label Statements \(p. 294\)](#)
- [Gremlin Edge Statements \(p. 294\)](#)
- [Gremlin Property Statements \(p. 294\)](#)

Gremlin Vertex Label Statements

Vertex label statements in Neptune serve two purposes:

- They track the labels for a vertex.
- The presence of at least one of these statements is what implies the existence of a particular vertex in the graph.

The subject of these statements is a vertex identifier, and the object is a label, both of which are specified by the user. You use a special fixed predicate for these statements, displayed as `<-label>`, and a default graph identifier (the null graph), displayed as `<->`.

For example, consider the following `addV` traversal.

```
g.addV("Person").property(id, "v1")
```

This traversal results in the following statement being added to the graph.

```
StatementEvent[Added(<v1> <-label> <Person> <->) .]
```

Gremlin Edge Statements

A Gremlin edge statement is what implies the existence of an edge between two vertices in a graph in Neptune. The subject (S) of an edge statement is the source `from` vertex. The predicate (P) is a user-supplied edge label. The object (O) is the target `to` vertex. The graph (G) is a user-supplied edge identifier.

For example, consider the following `addE` traversal.

```
g.addE("knows").from(V("v1")).to(V("v2")).property(id, "e1")
```

The traversal results in the following statement being added to the graph.

```
StatementEvent[Added(<v1> <knows> <v2> <e1>) .]
```

Gremlin Property Statements

A Gremlin property statement in Neptune asserts an individual property value for a vertex or edge. The subject is a user-supplied vertex or edge identifier. The predicate is the property name (key), and the object is the individual property value. The graph (G) is again the default graph identifier, the null graph, displayed as `<->`.

Consider the following example.

```
g.V("v1").property("name", "John")
```

This statement results in the following.

```
StatementEvent[Added(<v1> <name> "John" <->) .]
```

Property statements differ from others in that their object is a primitive value (a `string`, `date`, `byte`, `short`, `int`, `long`, `float`, or `double`). Their object is not a resource identifier that could be used as the subject of another assertion.

For multi-properties, each individual property value in the set receives its own statement.

```
g.V("v1").property(set, "phone", "956-424-2563").property(set, "phone", "956-354-3692")
  (tel:9563543692)"
```

This results in the following.

```
StatementEvent[Added(<v1> <phone> "956-424-2563" <->) .]
StatementEvent[Added(<v1> <phone> "956-354-3692" <->) .]
```

How Neptune processes Gremlin queries using statement indexes

Statements are accessed in Amazon Neptune by way of three statement indexes, as detailed in [How Statements Are Indexed in Neptune \(p. 61\)](#). Neptune extracts a statement *pattern* from a Gremlin query in which some positions are known, and the rest are left for discovery by index search.

Neptune assumes that the size of the property graph schema is not large. This means that the number of distinct edge labels and property names is fairly low, resulting in a low total number of distinct predicates. Neptune tracks distinct predicates in a separate index. It uses this cache of predicates to do a union scan of { *all P x POGS* } rather than use an OSGP index. Avoiding the need for a reverse traversal OSGP index saves both storage space and load throughput.

The Neptune Gremlin Explain/Profile API lets you obtain the predicate count in your graph. You can then determine whether your application invalidates the Neptune assumption that your property graph schema is small.

The following examples help illustrate how Neptune uses indexes to process Gremlin queries.

Question: What are the labels of vertex v1?

```
Gremlin code:    g.V('v1').label()
Pattern:         (<v1>, <-label>, ?, ?)
Known positions: SP
Lookup positions: OG
Index:          SPOG
Key range:      <v1>:<-label>:*
```

Question: What are the 'knows' out-edges of vertex v1?

```
Gremlin code:    g.V('v1').out('knows')
Pattern:         (<v1>, <knows>, ?, ?)
Known positions: SP
Lookup positions: OG
Index:          SPOG
Key range:      <v1>:<knows>:*
```

Question: Which vertices have a Person vertex label?

```
Gremlin code:    g.V().hasLabel('Person')
Pattern:         (?, <-label>, <Person>, <->)
Known positions: POG
Lookup positions: S
Index:          POGS
Key range:      <-label>:<Person>:<->:*
```

Question: What are the from/to vertices of a given edge e1?


```
Gremlin code:    g.E('e1').bothV()
Pattern:         (?, ?, ?, <e1>)
Known positions: G
Lookup positions: SPO
Index:           GPSO
Key range:       <e1>:*
```

One statement index that Neptune does **not** have is a reverse traversal OSGP index. This index could be used to gather all incoming edges across all edge labels, as in the following example.

Question: What are the incoming adjacent vertices v1?

```
Gremlin code:    g.V('v1').in()
Pattern:         (?, ?, <v1>, ?)
Known positions: O
Lookup positions: SPG
Index:           OSGP // <-- Index does not exist
```

How Gremlin queries are processed in Neptune

In Amazon Neptune, more complex traversals can be represented by a series of patterns that create a relation based on the definition of named variables that can be shared across patterns to create joins. This is shown in the following example.

Question: What is the two-hop neighborhood of vertex v1?

```
Gremlin code:    g.V('v1').out('knows').out('knows').path()
Pattern:         (?1=<v1>, <knows>, ?2, ?) X Pattern(?2, <knows>, ?3, ?)
```

The pattern produces a three-column relation (?1, ?2, ?3) like this:

```
?1    ?2    ?3
=====
v1    v2    v3
v1    v2    v4
v1    v5    v6
```

By sharing the ?2 variable across the two patterns (at the O position in the first pattern and the S position of the second pattern), you create a join from the first hop neighbors to the second hop neighbors. Each Neptune solution has bindings for the three named variables, which can be used to re-create a [TinkerPop Traverser](#) (including path information).

The first step in Gremlin query processing is to parse the query into a TinkerPop [Traversal](#) object, composed of a series of TinkerPop [steps](#). These steps, which are part of the open-source [Apache TinkerPop project](#), are both the logical and physical operators that compose a Gremlin traversal in the reference implementation. They are both used to represent the model of the query. They are executable operators that can produce solutions according to the semantics of the operator that they represent. For example, `.V()` is both represented and executed by the TinkerPop [GraphStep](#).

Because these off-the-shelf TinkerPop steps are executable, such a TinkerPop Traversal can execute any Gremlin query and produce the correct answer. However, when executed against a large graph, TinkerPop steps can sometimes be very inefficient and slow. Instead of using them, Neptune tries to convert the traversal into a declarative form composed of groups of patterns, as described previously.

Neptune doesn't currently support all Gremlin operators (steps) in its native query engine. So it tries to collapse as many steps as possible down into a single `NeptuneGraphQueryStep`, which contains the declarative logical query plan for all the steps that have been converted. Ideally, all steps are converted.

But when a step is encountered that can't be converted, Neptune breaks out of native execution and defers all query execution from that point forward to the TinkerPop steps. It doesn't try to weave in and out of native execution.

After the steps are translated into a logical query plan, Neptune runs a series of query optimizers that rewrite the query plan based on static analysis and estimated cardinalities. These optimizers do things like reorder operators based on range counts, prune unnecessary or redundant operators, rearrange filters, push operators into different groups, and so on.

After an optimized query plan is produced, Neptune creates a pipeline of physical operators that do the work of executing the query. This includes reading data from the statement indices, performing joins of various types, filtering, ordering, and so on. The pipeline produces a solution stream that is then converted back into a stream of TinkerPop Traverser objects.

Serialization of query results

Amazon Neptune currently relies on the TinkerPop response message serializers to convert query results (TinkerPop Traversers) into the serialized data to be sent over the wire back to the client. These serialization formats tend to be quite verbose.

For example, to serialize the result of a vertex query such as `g.V().limit(1)`, the Neptune query engine must perform a single search to produce the query result. However, the GraphSON serializer would perform a large number of additional searches to package the vertex into the serialization format. It would have to perform one search to get the label, one to get the property keys, and one search per property key for the vertex to get all the values for each key.

Some of the serialization formats are more efficient, but all require additional searches. Additionally, the TinkerPop serializers don't try to avoid duplicated searches, often resulting in many searches being repeated unnecessarily.

This makes it very important to write your queries so that they ask specifically just for the information they need. For example, `g.V().limit(1).id()` would return just the vertex ID and eliminate all the additional serializer searches. The [Gremlin profile API in Neptune \(p. 303\)](#) allows you to see how many search calls are made during query execution and during serialization.

Using the Gremlin explain API in Neptune

The Amazon Neptune Gremlin explain API returns the query plan that would be executed if a specified query were run. Because the API doesn't actually run the query, the plan is returned almost instantaneously.

It differs from the TinkerPop `.explain()` step so as to be able to report information specific to the Neptune engine.

Information contained in a Gremlin explain report

An `explain` report contains the following information:

- The query string as requested.
- **The original traversal.** This is the TinkerPop Traversal object produced by parsing the query string into TinkerPop steps. It is equivalent to the original query produced by running `.explain()` on the query against the TinkerPop TinkerGraph.
- **The converted traversal.** This is the Neptune Traversal produced by converting the TinkerPop Traversal into the Neptune logical query plan representation. In many cases the entire TinkerPop traversal is converted into two Neptune steps: one that executes the entire query (`NeptuneGraphQueryStep`) and one that converts the Neptune query engine output back into TinkerPop Traversers (`NeptuneTraverserConverterStep`).

- **The optimized traversal.** This is the optimized version of the Neptune query plan after it has been run through a series of static work-reducing optimizers that rewrite the query based on static analysis and estimated cardinalities. These optimizers do things like reorder operators based on range counts, prune unnecessary or redundant operators, rearrange filters, push operators into different groups, and so on.
- **The predicate count.** Because of the Neptune indexing strategy described earlier, having a large number of different predicates can cause performance problems. This is especially true for queries that use reverse traversal operators with no edge label (`.in` or `.both`). If such operators are used and the predicate count is high enough, the `explain` report displays a warning message.
- **DFE information.** When the DFE alternative engine is enabled, the following traversal components may show up in the optimized traversal:
 - **DFEStep** – A Neptune optimized DFE step in the traversal that contains a child `DFENode`. `DFEStep` represents the part of the query plan that is executed in the DFE engine.
 - **DFENode** – Contains the intermediate representation as one or more child `DFEJoinGroupNodes`.
 - **DFEJoinGroupNode** – Represents a join of one or more `DFENode` or `DFEJoinGroupNode` elements.
 - **NeptuneInterleavingStep** – A Neptune optimized DFE step in the traversal that contains a child `DFEStep`.

Also contains a `stepInfo` element that contains information about the traversal, such as the frontier element, the path elements used, and so on. This information is used to process the child `DFEStep`.

An easy way to find out if your query is being evaluated by DFE is to check whether the `explain` output contains a `DFEStep`. Any part of the traversal that is not part of the `DFEStep` will not be executed by DFE and will be executed by the TinkerPop engine.

See [Example with DFE enabled \(p. 301\)](#) for a sample report.

Gremlin explain syntax

The syntax of the `explain` API is the same as that for the HTTP API for query, except that it uses `/gremlin/explain` as the endpoint instead of `/gremlin`, as in the following example.

```
curl -X POST https://your-neptune-endpoint:port/gremlin/explain -d  
'{"gremlin":"g.V().limit(1)"}'
```

The preceding query would produce the following output.

```
*****  
                Neptune Gremlin Explain  
*****  
  
Query String  
=====  
g.V().limit(1)  
  
Original Traversal  
=====  
[GraphStep(vertex,[]), RangeGlobalStep(0,1)]  
  
Converted Traversal  
=====  
Neptune steps:  
[  
  NeptuneGraphQueryStep(Vertex) {  
    JoinGroupNode {
```

```

        PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .]
    }, finishers=[limit(1)], annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
  },
  NeptuneTraverserConverterStep
]

Optimized Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
      PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .],
      {estimatedCardinality=INFINITY}
    }, finishers=[limit(1)], annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
  },
  NeptuneTraverserConverterStep
]

Predicates
=====
# of predicates: 18

```

Unconverted TinkerPop Steps

Ideally, all TinkerPop steps in a traversal have native Neptune operator coverage. When this isn't the case, Neptune falls back on TinkerPop step execution for gaps in its operator coverage. If a traversal uses a step for which Neptune does not yet have native coverage, the `explain` report displays a warning showing where the gap occurred.

When a step without a corresponding native Neptune operator is encountered, the entire traversal from that point forward is run using TinkerPop steps, even if subsequent steps do have native Neptune operators.

The exception to this is when Neptune full-text search is invoked. The `NeptuneSearchStep` implements steps without native equivalents as full-text search steps.

Example of `explain` output where all steps in a query have native equivalents

The following is an example `explain` report for a query where all steps have native equivalents:

```

*****
                Neptune Gremlin Explain
*****

Query String
=====
g.V().out()

Original Traversal
=====
[GraphStep(vertex, []), VertexStep(OUT,vertex)]

Converted Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
      PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .]
      PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
      PatternNode[(?3, <-label>, ?4, <->) . project ask .]
    }
  }
]

```

```

    }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
  },
  NeptuneTraverserConverterStep
]

Optimized Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
      PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .],
      {estimatedCardinality=INFINITY}
    }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
  },
  NeptuneTraverserConverterStep
]

Predicates
=====
# of predicates: 18

```

Example where some steps in a query do not have native equivalents

Neptune handles both `GraphStep` and `VertexStep` natively, but if you introduce a `FoldStep` and `UnfoldStep`, the resulting explain output is different:

```

*****
                Neptune Gremlin Explain
*****

Query String
=====
g.V().fold().unfold().out()

Original Traversal
=====
[GraphStep(vertex,[]), FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

Converted Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
      PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .]
    }, annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
  },
  NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

Optimized Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
      PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .],
      {estimatedCardinality=INFINITY}
    }, annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
  },
  NeptuneTraverserConverterStep,
  NeptuneMemoryTrackerStep
]

```

```
]
+ not converted into Neptune steps: [FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

WARNING: >> FoldStep << is not supported natively yet
```

In this case, the `FoldStep` breaks you out of native execution. But even the subsequent `VertexStep` is no longer handled natively because it appears downstream of the `Fold/Unfold` steps.

For performance and cost-savings, it's important that you try to formulate traversals so that the maximum amount of work possible is done natively inside the Neptune query engine, instead of by the TinkerPop step implementations.

Example of a query that uses Neptune full-text-search

The following query uses Neptune full-text search:

```
g.withSideEffect("Neptune#fts.endpoint", "some_endpoint")
  .V()
  .tail(100)
  .has("Neptune#fts mark*")
  -----
  .has("name", "Neptune#fts mark*")
  .has("Person", "name", "Neptune#fts mark*")
```

The `.has("name", "Neptune#fts mark*")` part limits the search to vertexes with name, while `.has("Person", "name", "Neptune#fts mark*")` limits the search to vertexes with name and the label `Person`. This results in the following traversal in the `explain` report:

```
Final Traversal
[NeptuneGraphQueryStep(Vertex) {
  JoinGroupNode {
    PatternNode[(?1, termid(1,URI), ?2, termid(0,URI)) . project distinct ?1 .],
    {estimatedCardinality=INFINITY}
  }, annotations={path=[Vertex(?1):GraphStep], maxVarId=4}
}, NeptuneTraverserConverterStep, NeptuneTailGlobalStep(10),
NeptuneTinkerpopTraverserConverterStep, NeptuneSearchStep {
  JoinGroupNode {
    SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
    {endpoint=some_endpoint}
  }
  JoinGroupNode {
    SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
    {endpoint=some_endpoint}
  }
}]
```

Example of using `explain` when the DFE is enabled

The following is an example of an `explain` report when the DFE alternative query engine is enabled:

```
*****
      Neptune Gremlin Explain
*****

Query String
=====

g.V().as("a").out().has("name", "josh").out().in().where(eq("a"))
```

Original Traversal

=====

```
[GraphStep(vertex,[])@[a], VertexStep(OUT,vertex), HasStep([name.eq(josh)]),  
VertexStep(OUT,vertex), VertexStep(IN,vertex), WherePredicateStep(eq(a))]
```

Converted Traversal

=====

Neptune steps:

```
[  
  DFESStep(Vertex) {  
    DFENode {  
      DFEJoinGroupNode[ children={  
        DFEPatternNode[ (?1, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, ?2,  
<http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> ) . project DISTINCT[?1]  
{rangeCountEstimate=unknown}],  
        DFEPatternNode[ (?1, ?3, ?4, ?5) . project ALL[?1, ?4] graphFilters!=(= <http://  
aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ), {rangeCountEstimate=unknown}]  
      }, {rangeCountEstimate=unknown}  
    ]  
  } [Vertex(?1):GraphStep@[a], Vertex(?4):VertexStep]  
  },  
  NeptuneTraverserConverterDFESStep  
]
```

+ not converted into Neptune steps: HasStep([name.eq(josh)]),

Neptune steps:

```
[  
  NeptuneInterleavingStep {  
    StepInfo[ joinVars=[?7, ?1], frontierElement=Vertex(?7):HasStep,  
pathElements={a=(last,Vertex(?1):GraphStep@[a])}, listPathElement={}, indexTime=0ms],  
    DFESStep(Vertex) {  
      DFENode {  
        DFEJoinGroupNode[ children={  
          DFEPatternNode[ (?7, ?8, ?9, ?10) . project ALL[?7, ?9]  
graphFilters!=(= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),  
{rangeCountEstimate=unknown}],  
          DFEPatternNode[ (?12, ?11, ?9, ?13) . project ALL[?9, ?12]  
graphFilters!=(= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),  
{rangeCountEstimate=unknown}]  
        }, {rangeCountEstimate=unknown}  
      ]  
    } [Vertex(?9):VertexStep, Vertex(?12):VertexStep]  
  }  
}
```

+ not converted into Neptune steps: WherePredicateStep(eq(a)),

Neptune steps:

```
[  
  DFECleanupStep  
]
```

Optimized Traversal

=====

Neptune steps:

```
[  
  DFESStep(Vertex) {  
    DFENode {  
      DFEJoinGroupNode[ children={  
        DFEPatternNode[ (?1, ?3, ?4, ?5) . project ALL[?1, ?4] graphFilters!=(=  
defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}]  
      }, {rangeCountEstimate=unknown}  
    ]  
  } [Vertex(?1):GraphStep@[a], Vertex(?4):VertexStep]  
  },  
  NeptuneTraverserConverterDFESStep  
]
```

```
]
+ not converted into Neptune steps: NeptuneHasStep([name.eq(josh)]),
Neptune steps:
[
  NeptuneMemoryTrackerStep,
  NeptuneInterleavingStep {
    StepInfo[joinVars=[?7, ?1], frontierElement=Vertex(?7):HasStep,
    pathElements={a=(last,Vertex(?1):GraphStep@[a]), listPathElement={}, indexTime=0ms],
    DFESTep(Vertex) {
      DFENode {
        DFEJoinGroupNode[ children={
          DFEPatternNode[ (?7, ?8, ?9, ?10) . project ALL[?7, ?9] graphFilters=(!=
defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}],
          DFEPatternNode[ (?12, ?11, ?9, ?13) . project ALL[?9, ?12] graphFilters=(!=
defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}]
        }, {rangeCountEstimate=unknown}
      ]
    } [Vertex(?9):VertexStep, Vertex(?12):VertexStep]
  }
]
+ not converted into Neptune steps: WherePredicateStep(eq(a)),
Neptune steps:
[
  DFECleanupStep
]

WARNING: >> [NeptuneHasStep([name.eq(josh)]), WherePredicateStep(eq(a))] << (or one of the
children for each step) is not supported natively yet

Predicates
=====
# of predicates: 8
```

See [Information in explain \(p. 297\)](#) for a description of the DFE-specific sections in the report.

Gremlin profile API in Neptune

The Neptune Gremlin profile API runs a specified Gremlin traversal, collects various metrics about the run, and produces a profile report as output.

Note

This feature is available starting with [Release 1.0.1.0.200463.0 \(2019-10-15\) \(p. 767\)](#).

It differs from the TinkerPop `.profile()` step so as to be able to report information specific to the Neptune engine.

The profile report includes the following information about the query plan:

- The physical operator pipeline
- The index operations for query execution and serialization
- The size of the result

The profile API uses an extended version of the HTTP API syntax for query, with `/gremlin/profile` as the endpoint instead of `/gremlin`.

Parameters specific to Neptune Gremlin profile

- **profile.results** – boolean, allowed values: `TRUE` and `FALSE`, default value: `TRUE`.

If true, the query results are gathered and displayed as part of the `profile` report. If false, only the result count is displayed.

- **profile.chop** – `int`, default value: 250.

If non-zero, causes the results string to be truncated at that number of characters. This does not keep all results from being captured. It simply limits the size of the string in the profile report. If set to zero, the string contains all the results.

- **profile.serializer** – `string`, default value: `<null>`.

If non-null, the gathered results are returned in a serialized response message in the format specified by this parameter. The number of index operations necessary to produce that response message is reported along with the size in bytes to be sent to the client.

Allowed values are `<null>` or any of the valid MIME type or TinkerPop driver "Serializers" enum values.

<code>"application/json"</code>	or	<code>"MIME_JSON"</code>
<code>"application/vnd.gremlin-v1.0+json"</code>	or	<code>"GRAPHSON_V1D0"</code>
<code>"application/vnd.gremlin-v2.0+json"</code>	or	<code>"GRAPHSON_V2D0"</code>
<code>"application/vnd.gremlin-v3.0+json"</code>	or	<code>"GRAPHSON_V3D0"</code>
<code>"application/vnd.gremlin-v1.0+gryo"</code>	or	<code>"GRYO_V1D0"</code>
<code>"application/vnd.gremlin-v3.0+gryo"</code>	or	<code>"GRYO_V3D0"</code>
<code>"application/vnd.gremlin-v1.0+gryo-lite"</code>	or	<code>"GRYO_LITE_V1D0"</code>
<code>"application/vnd.graphbinary-v1.0"</code>	or	<code>"GRAPHBINARY_V1D0"</code>

- **profile.indexOps** – `boolean`, allowed values: `TRUE` and `FALSE`, default value: `FALSE`.

If true, shows a detailed report of all index operations that took place during query execution and serialization. Warning: This report can be verbose.

Sample output of Neptune Gremlin profile

The following is a sample profile query.

```
curl -X POST https://your-neptune-endpoint:port/gremlin/profile \  
-d '{"gremlin":"g.V().hasLabel(\"airport\")  
    .has(\"code\", \"AUS\")  
    .emit()  
    .repeat(in().simplePath())  
    .times(2)  
    .limit(100)",  
"profile.serializer":"application/vnd.gremlin-v3.0+gryo"}'
```

This query generates the following profile report when executed on the air-routes sample graph from the blog post, [Let Me Graph That For You – Part 1 – Air Routes](#).

```
*****  
      Neptune Gremlin Profile  
*****  
  
Query String  
=====  
g.V().hasLabel("airport").has("code",  
"AUS").emit().repeat(in().simplePath()).times(2).limit(100)  
  
Original Traversal  
=====
```

```
[GraphStep(vertex,[]), HasStep([-label.eq(airport), code.eq(AUS)]), RepeatStep(emit(true),
[VertexStep(IN,vertex), PathFilterStep(simple), RepeatEndStep],until(loops(2))),
RangeGlobalStep(0,100)]
```

Optimized Traversal

=====

Neptune steps:

```
[
  NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
      PatternNode[ (?1, <code>, "AUS", ?) . project ?1 . ], {estimatedCardinality=1,
indexTime=84, hashJoin=true, joinTime=3, actualTotalOutput=1}
      PatternNode[ (?1, <~label>, ?2=<airport>, <->) . project ask . ],
{estimatedCardinality=3374, indexTime=29, hashJoin=true, joinTime=0, actualTotalOutput=61}
      RepeatNode {
        Repeat {
          PatternNode[ (?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .
SimplePathFilter(?1, ?3)) . ], {hashJoin=true, estimatedCardinality=50148, indexTime=0,
joinTime=3}
        }
        Emit {
          Filter(true)
        }
        LoopsCondition {
          LoopsFilter([?1, ?3],eq(2))
        }
      }, annotations={repeatMode=BFS, emitFirst=true, untilFirst=false, leftVar=?1,
rightVar=?3}
      }, finishers=[limit(100)], annotations={path=[Vertex(?1):GraphStep, Repeat[Vertex(?
3):VertexStep]], joinStats=true, optimizationTime=495, maxVarId=7, executionTime=323}
    },
    NeptuneTraverserConverterStep
  ]
```

Physical Pipeline

=====

NeptuneGraphQueryStep

```
|-- StartOp
|-- JoinGroupOp
  |-- SpoolerOp(100)
    |-- DynamicJoinOp(PatternNode[ (?1, <code>, "AUS", ?) . project ?1 . ],
{estimatedCardinality=1, indexTime=84, hashJoin=true})
      |-- SpoolerOp(100)
        |-- DynamicJoinOp(PatternNode[ (?1, <~label>, ?2=<airport>, <->) . project ask . ],
{estimatedCardinality=3374, indexTime=29, hashJoin=true})
          |-- RepeatOp
            |-- <upstream input> (Iteration 0) [visited=1, output=1 (until=0, emit=1),
next=1]
              |-- BindingSetQueue (Iteration 1) [visited=61, output=61 (until=0, emit=61),
next=61]
                |-- SpoolerOp(100)
                  |-- DynamicJoinOp(PatternNode[ (?3, ?5, ?1, ?6) . project ?
1,?3 . IsEdgeIdFilter(?6) . SimplePathFilter(?1, ?3)) . ], {hashJoin=true,
estimatedCardinality=50148, indexTime=0})
                    |-- BindingSetQueue (Iteration 2) [visited=38, output=38 (until=38, emit=0),
next=0]
                      |-- SpoolerOp(100)
                        |-- DynamicJoinOp(PatternNode[ (?3, ?5, ?1, ?6) . project ?
1,?3 . IsEdgeIdFilter(?6) . SimplePathFilter(?1, ?3)) . ], {hashJoin=true,
estimatedCardinality=50148, indexTime=0})
                          |-- LimitOp(100)
```

Runtime (ms)

=====

Query Execution: 392.686

Serialization: 2636.380

```

Traversal Metrics
=====
Step                                     Count  Traversers
Time (ms)    % Dur
-----
NeptuneGraphQueryStep(Vertex)           100     100
  314.162    82.78
NeptuneTraverserConverterStep           100     100
  65.333    17.22
                                     >TOTAL
  379.495    -

Repeat Metrics
=====
Iteration  Visited  Output  Until  Emit  Next
-----
          0         1       1       0       1       1
          1        61      61       0      61      61
          2        38      38      38       0       0
-----
                    100    100    38    62    62

Predicates
=====
# of predicates: 16

WARNING: reverse traversal with no edge label(s) - .in() / .both() may impact query
performance

Results
=====
Count: 100
Output: [v[3], v[3600], v[3614], v[4], v[5], v[6], v[7], v[8], v[9], v[10], v[11], v[12],
  v[47], v[49], v[136], v[13], v[15], v[16], v[17], v[18], v[389], v[20], v[21], v[22],
  v[23], v[24], v[25], v[26], v[27], v[28], v[416], v[29], v[30], v[430], v[31], v[9...
Response serializer: GRYO_V3D0
Response size (bytes): 23566

Index Operations
=====
Query execution:
  # of statement index ops: 3
  # of unique statement index ops: 3
  Duplication ratio: 1.0
  # of terms materialized: 0
Serialization:
  # of statement index ops: 200
  # of unique statement index ops: 140
  Duplication ratio: 1.43
  # of terms materialized: 393

```

In addition to the query plans returned by a call to Neptune `explain`, the `profile` results include runtime statistics around query execution. Each Join operation is tagged with the time it took to perform its join as well as the actual number of solutions that passed through it.

The `profile` output includes the time taken during the core query execution phase, as well as the serialization phase if the `profile.serializer` option was specified.

The breakdown of the index operations performed during each phase is also included at the bottom of the `profile` output.

Note that consecutive runs of the same query may show different results in terms of run-time and index operations because of caching.

For queries using the `repeat()` step, a breakdown of the frontier on each iteration is available if the `repeat()` step was pushed down as part of a `NeptuneGraphQueryStep`.

Differences in `profile` reports when DFE is enabled

When the Neptune DFE alternative query engine is enabled, `profile` output is somewhat different:

Optimized Traversal: This section is similar to the one in `explain` output, but contains additional information. This includes the type of DFE operators that were considered in planning, and the associated worst case and best case cost estimates.

Physical Pipeline: This section captures the operators that are used to execute the query. `DFESubQuery` elements abstract the physical plan that is used by DFE to execute the portion of the plan it is responsible for. The `DFESubQuery` elements are unfolded in the following section where DFE statistics are listed.

DFEQueryEngine Statistics: This section shows up only when at least part of the query is executed by DFE. It outlines various runtime statistics that are specific to DFE, and contains a detailed breakdown of the time spent in the various parts of the query execution, by `DFESubQuery`.

Nested subqueries in different `DFESubQuery` elements are flattened in this section, and unique identifiers are marked with a header that starts with `subQuery=`.

Traversal metrics: This section shows step-level traversal metrics, and when the DFE engine runs all or part of the query, displays metrics for `DFEStep` and/or `NeptuneInterleavingStep`. See [Tuning Gremlin queries using `explain` and `profile`](#) (p. 322).

Note

DFE is an experimental feature released under lab mode, so the exact format of the `profile` output is still subject to change.

Sample `profile` output when the Neptune Dataflow engine (DFE) is enabled

When the DFE engine is being used to run Gremlin queries, output of the [Gremlin `profile` API](#) (p. 303) is formatted as shown in this example:

```
*****
      Neptune Gremlin Profile
*****

Query String
=====

g.V().as("a").out().has("name", "josh").out().in().where(eq("a"))

Original Traversal
=====
[GraphStep(vertex,[ ])[a], VertexStep(OUT,vertex), HasStep([name.eq(josh)]),
VertexStep(OUT,vertex), VertexStep(IN,vertex), WherePredicateStep(eq(a))]

Optimized Traversal
=====
Neptune steps:
[
  DFEStep(Vertex) {
    DFENode {
      DFEJoinGroupNode[ children={
        DFENaturalJoinNode[ children={
          DFGNodeReference(node_0),
          DFEPatternNode[ (?1, ?3, ?4, ?5) . project ALL[?1, ?4] graphFilters=(!=
defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807},
```

```

OperatorInfoWithAlternative[
  rec=OperatorInfo[
    type=PatternJoin[LEFT_TO_RIGHT],
    costEstimates=OperatorCostEstimates[
costEstimate=OperatorCostEstimate[in=1.0000,out=30.0000,io=0.0001,comp=0.0000,mem=0],
worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=30.0000,io=0.0001,comp=0.0000,mem=0]]],
  alt=OperatorInfo[
    type=PatternJoin[BOTTOM_UP],
    costEstimates=OperatorCostEstimates[
costEstimate=OperatorCostEstimate[in=1.0000,out=30.0000,io=Infinity,comp=0.0000,mem=295147905179352830],
worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=30.0000,io=Infinity,comp=0.0000,mem=295147905179352830]
    ], {rangeCountEstimate=unknown}
  ], {rangeCountEstimate=unknown}
] [Vertex(?1):GraphStep@[a], Vertex(?4):VertexStep
] ,
NeptuneTraverserConverterDFEStep
]
+ not converted into Neptune steps: NeptuneHasStep([name.eq(josh)]),
Neptune steps:
[
  NeptuneMemoryTrackerStep,
  NeptuneInterleavingStep {
    StepInfo[joinVars=[?7, ?1], frontierElement=Vertex(?7):HasStep,
pathElements={a=(last,Vertex(?1):GraphStep@[a])}, listPathElement={}, indexTime=1ms],
    DFESTep(Vertex) {
      DFENode {
        DFEJoinGroupNode[ children={
          DFENaturalJoinNode[ children={
            DFGNodeReference(node_0),
            DFEPatternNode[ (?7, ?8, ?9, ?10) . project ALL[?7, ?9] graphFilters=(!=
defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807},
            OperatorInfoWithAlternative[
              rec=OperatorInfo[
                type=PatternJoin[LEFT_TO_RIGHT],
                costEstimates=OperatorCostEstimates[
costEstimate=OperatorCostEstimate[in=1.0000,out=2.5000,io=0.0000,comp=0.0000,mem=0],
worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=2.5000,io=0.0000,comp=0.0000,mem=0]]],
              alt=OperatorInfo[
                type=PatternJoin[BOTTOM_UP],
                costEstimates=OperatorCostEstimates[
costEstimate=OperatorCostEstimate[in=1.0000,out=2.5000,io=Infinity,comp=0.0000,mem=295147905179352830],
worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=2.5000,io=Infinity,comp=0.0000,mem=295147905179352830]
                ], {rangeCountEstimate=unknown}
              ],
            DFEPatternNode[ (?12, ?11, ?9, ?13) . project ALL[?9, ?12] graphFilters=(!=
defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807},
            OperatorInfoWithAlternative[
              rec=OperatorInfo[
                type=PatternJoin[LEFT_TO_RIGHT],
                costEstimates=OperatorCostEstimates[
costEstimate=OperatorCostEstimate[in=2.5000,out=3.1250,io=0.0000,comp=0.0000,mem=0],
worstCaseCostEstimate=OperatorCostEstimate[in=2.5000,out=3.1250,io=0.0000,comp=0.0000,mem=0]]],
              alt=OperatorInfo[
                type=PatternJoin[BOTTOM_UP],

```


Amazon Neptune User Guide
Gremlin explain

```
# # # # # # # # # # #
#####
# 2 # 3 # - # DFEProject # columns=[?1, ?4] # - # 6 #
6 # 1.00 # 0.08 # 1 # 1 # 6.00 # 6.00 #
# 1 # 0 # 0 # 0.08 # 0.01 # 2
# 2 # 2 # 0 # 0 # 0 #
# # # # # inSchema=[?1, ?4] # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 3 # 4 # - # DFEMergeChunks # inSchema=[?1, ?4] # - # 6 #
6 # 1.00 # 0.03 # 1 # 1 # 6.00 # 6.00 #
# 1 # 0 # 0 # 0.21 # 0.02 # 2
# 2 # 2 # 0 # 0 # 0 #
# # # # # outSchema=[?1, ?4] # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 4 # - # - # DFEDrain # inSchema=[?1, ?4] # - # 6 #
0 # 0.00 # 0.02 # 1 # 0 # 6.00 # 0.00 #
# 1 # 0 # 0 # 0.00 # 0.01 # 2
# 2 # 2 # 0 # 0.02 # 0 #
# # # # # outSchema=[?1, ?4] # # #
# # # # # # # # # #
# # # # # # # # # #
#####
subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/
graph#ae3b0492-328b-4f44-92d8-cdbelbd17699/graph_1
#####
# ID # Out #1 # Out #2 # Name # Arguments
# Mode # Units In # Units Out # Ratio # Time (ms) # Chunks In # Chunks Out # Units In
(per chunk) # Units Out (per chunk) # Invocations # OutWaitMs # Out Blocked # Rate [M/
s] # GCElapsedMs # blocksIncGC # blocksDecGC # progressCount # init [ms] # done [ms] #
finalize [ms] #
#####
# 0 # 1 # - # DFEPipelineScan # pattern=project ?1 ?4 (?1,?3,?4) # -
# 0 # 6 # 0.00 # 0.49 # 0 # 1 # 0.00
# 6.00 # 1 # 0 # 0 # 0 # 0 # 0.01 # 0.01
# 2 # 0 # 0 # 0 # 0 # 0 # 0 #
# # # # # constraints=[(?g != defaultGraph)] #
# # # # # # # # # #
# # # # # # # # # #
```


Amazon Neptune User Guide
Gremlin explain

```
# 0 # 1 # - # DFEsolutionInjection # outSchema=[?1, ?7] # - # 0 #
1 # 0.00 # 0.11 # 0 # 1 # 0.00 # 1.00
# 1 # 0 # 0 # 0.01 # 0.03 # 12
# 0 # 1 # 0 # 0 # 0 # 0 #
#####
# 1 # 2 # - # DFChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#448f04da-2d8c-475f-bb7f-10ac2fc469bd/graph_1 # - # 1
# 2 # 2.00 # 5.50 # 1 # 1 # 1.00 # 2.00
# 1 # 0 # 0 # 0.00 # 0.03 # 9
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 2 # 3 # - # DFChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#448f04da-2d8c-475f-bb7f-10ac2fc469bd/graph_2 # - # 2
# 4 # 2.00 # 3.61 # 1 # 1 # 2.00 # 4.00
# 1 # 0 # 0 # 0.00 # 0.04 # 10
# 9 # 3 # 0 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 3 # 4 # - # DFProject # columns=[?1, ?7, ?9, ?12] # - # 4 #
4 # 1.00 # 0.14 # 1 # 1 # 4.00 # 4.00
# 1 # 0 # 0 # 0.03 # 0.03 # 7
# 10 # 2 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
```

Amazon Neptune User Guide
Gremlin explain

```
#####
# 4 # 5 # - # DFEMergeChunks # inSchema=[?1, ?7, ?9, ?12] # - # 4 #
4 # 1.00 # 0.03 # 1 # 1 # 4.00 # 4.00
# 7 # 1 # 0 # 0 # 0 # 0.12 # 0.03 # 7
# # # # # # # #
# # # # # # # #
# # # # # # # #
#####
# 5 # - # - # DFEDrain # inSchema=[?1, ?7, ?9, ?12] # - # 4 #
0 # 0.00 # 0.04 # 1 # 0 # 4.00 # 0.00
# 7 # 1 # 0 # 0 # 0.02 # 0 # 7
# # # # # # # #
# # # # # # # #
# # # # # # # #
#####

subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#448f04da-2d8c-475f-
bb7f-10ac2fc469bd/graph_2

#####
# ID # Out #1 # Out #2 # Name # Arguments
# Mode # Units In # Units Out # Ratio # Time (ms) # Chunks In # Chunks Out # Units In
(per chunk) # Units Out (per chunk) # Invocations # OutWaitMs # Out Blocked # Rate [M/
s] # GCElapsedMs # blocksIncGC # blocksDecGC # progressCount # init [ms] # done [ms] #
finalize [ms] #
#####
# 0 # 1 # - # DFESolutionInjection # outSchema=[?1, ?100, ?7, ?9] # -
# 0 # 2 # 0.00 # 0.08 # 0 # 0 # 1 # 0.00
# 2.00 # 1 # 0 # 0 # 0 # 0.02 # 0.03
# 18 # 0 # 0 # 0 # 0 # 0 # 0 #
#####
# 1 # 2 # 3 # DFETee # inSchema=[?1, ?100, ?7, ?9] # -
# 2 # 4 # 2.00 # 0.05 # 1 # 2 # 2.00
# 2.00 # 1 # 0 # 0 # 0.08 # 0.05
# 18 # 9 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # #
# # # # # # # #
# # # # # # # #
#####
# 2 # 4 # - # DFEDHashIndexBuild # vars=[?9] # -
# 2 # 2 # 1.00 # 0.10 # 1 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0.02 # 0.04
# 9 # 9 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # #
#####
```

Amazon Neptune User Guide
Gremlin explain

```
# # # # # # # # # #
#####
# 3 # 11 # 12 # DFEsync # lhsSchema=[?1, ?100, ?7, ?9] # -
# 6 # 6 # 1.00 # 0.12 # 2 # 2 # 2 # 3.00
# 3.00 # 2 # 0 # 0 # 0 # 0.05 # 0.10
# 20 # 11 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 4 # 5 # 6 # DFETee # inSchema=[?1, ?100, ?7, ?9] # -
# 2 # 4 # 2.00 # 0.06 # 1 # 2 # 2.00
# 2.00 # 1 # 0 # 0 # 0 # 0.07 # 0.05
# 18 # 9 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 5 # 7 # - # DFEGroupByHashIndex # inSchema=[?1, ?100, ?7, ?9] # -
# 2 # 2 # 1.00 # 0.19 # 1 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0 # 0.01 # 0.02
# 1 # 9 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 6 # 8 # - # DFEProject # columns=[?9] # -
# 2 # 2 # 1.00 # 0.08 # 1 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0 # 0.03 # 0.02
# 1 # 9 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 7 # 8 # - # DFEDistinctColumn # column=?-1000 # -
# 2 # 2 # 1.00 # 0.10 # 1 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0 # 0.02 # 0.01
# 1 # 1 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
```

Amazon Neptune User Guide
Gremlin explain

```

# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 8 # 9 # - # DFESelectRows # lhsSchema=[?9] # -
# 4 # 2 # 0.50 # 0.12 # 2 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0 # 0 # 0.02 # 0.01
# 1 # 2 # 0 # 0 # 0 # 0 # 0 #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 9 # 10 # - # DFEDistinctColumn # column=?9 # -
# 2 # 2 # 1.00 # 0.39 # 1 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0 # 0 # 0.01 # 0.01
# 2 # # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 10 # 3 # - # DFEPipelineJoin # pattern=project ?12 ?9 (?12,?11,?9) # -
# 2 # 4 # 2.00 # 0.83 # 1 # 1 # 2.00
# 4.00 # 1 # 0 # 0 # 0 # 0 # 0.00 # 0.02
# 2 # # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 11 # 13 # - # DFEForwardValue # inSchema=[?1, ?100, ?7, ?9] # -
# 2 # 2 # 1.00 # 0.04 # 1 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0 # 0 # 0.05 # 0.04
# 9 # 9 # 0 # 0 # 0 # 0 # 0 #

```

Amazon Neptune User Guide
Gremlin explain

```

# # # # # outSchema=[?1, ?100, ?7, ?9] #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 12 # 13 # - # DFEForwardValue # inSchema=[?9, ?12] # -
# 4 # 4 # 1.00 # 0.03 # 1 # 1 # 4.00
# 4.00 # 1 # 0 # 0 # 0 # 0.13 # 0.02
# 2 # 2 # 0 # 0 # 0 # 0 # 0 #
# # # # # outSchema=[?9, ?12] #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 13 # 14 # - # DFEDHashIndexJoin # lhsSchema=[?1, ?100, ?7, ?9] # -
# 6 # 4 # 0.67 # 0.37 # 2 # 1 # 3.00
# 4.00 # 1 # 0 # 0 # 0 # 0.01 # 0.04
# 10 # 11 # 0 # 0 # 0 # 0 # 0 #
# # # # # rhsSchema=[?9, ?12] #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # # outSchema=[?1, ?100, ?7, ?9, ?12] #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 14 # - # - # DFEDrain # inSchema=[?1, ?100, ?7, ?9, ?12] # -
# 4 # 0 # 0.00 # 0.07 # 1 # 0 # 4.00
# 0.00 # 1 # 0 # 0 # 0 # 0.00 # 0.05
# 10 # 10 # 0 # 0 # 0 # 0.03 # 0 #
# # # # # outSchema=[?1, ?100, ?7, ?9, ?12] #
# # # # # # # # # # #
# # # # # # # # # # #
#####
subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#448f04da-2d8c-475f-
bb7f-10ac2fc469bd/graph_1
#####
# ID # Out #1 # Out #2 # Name # Arguments #
Mode # Units In # Units Out # Ratio # Time (ms) # Chunks In # Chunks Out # Units In
(per chunk) # Units Out (per chunk) # Invocations # OutWaitMs # Out Blocked # Rate [M/
s] # GCElapsedMs # blocksIncGC # blocksDecGC # progressCount # init [ms] # done [ms] #
finalize [ms] #
#####
# 0 # 1 # - # DFESolutionInjection # outSchema=[?1, ?7] # -
# 0 # 1 # 0.00 # 0.10 # 0 # 1 # 0.00
# 1.00 # 1 # 0 # 0 # 0 # 0 # 0.01 # 0.02
# 12 # 0 # 0 # 0 # 0 # 0 # 0 #
#
#####
# 1 # 2 # 3 # DFETee # inSchema=[?1, ?7] # -
# 1 # 2 # 2.00 # 0.04 # 1 # 2 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.05 # 0.03
# 12 # 6 # 0 # 0 # 0 # 0 # 0 #
#

```

Amazon Neptune User Guide
Gremlin explain

```
# # # # # outSchema=[?1, ?7] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 2 # 4 # - # DFEBindRelation # inputVars=[?7] # -
# 1 # 1 # 1.00 # 0.17 # 1 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.01 # 0.02
# 3 # 6 # 0 # 0 # 0 # 0
# # # # # outputVars=[?100] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # # inSchema=[?1, ?7] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # # outSchema=[?100] #
# # # # # # # # # #
#####
# 3 # 5 # - # DFEBindRelation # inputVars=[?7, ?1, ?100] # -
# 2 # 1 # 0.50 # 0.22 # 2 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.00 # 0.03
# 8 # 8 # 0 # 0 # 0 # 0
# # # # # outputVars=[?7, ?1, ?100] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # # inSchema=[?1, ?100, ?7] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # # outSchema=[?1, ?100, ?7] #
# # # # # # # # # #
# # # # # # # # # #
#####
# 4 # 3 # - # DFETermResolution # column=?100 #
value2id # 1 # 1 # 1.00 # 0.31 # 1 # 1 # 1.00
0.01 # 1.00 # 1 # 0 # 0 # 0 # 0.00 #
# 2 # 3 # 0 # 0 # 0 # 0
# # # # # inSchema=[?100] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # # outSchema=[?100] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
```

Amazon Neptune User Guide
Gremlin explain

```
#####
# 5 # 6 # - # DFEBindRelation # inputVars=[?7, ?1, ?100] # -
# 1 # 1 # 1.00 # 0.26 # 1 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.00 # 0.03
# 8 # 8 # 0 # 0 # 0 # 0
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 6 # 7 # 8 # DFETee # inSchema=[?1, ?100, ?7] # -
# 1 # 2 # 2.00 # 0.05 # 1 # 2 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.04 # 0.04
# 16 # 8 # 0 # 0 # 0 # 0
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 7 # 9 # - # DFEMergeChunks # inSchema=[?1, ?100, ?7] # -
# 1 # 1 # 1.00 # 0.05 # 1 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.02 # 0.05
# 8 # 8 # 0 # 0 # 0 # 0
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 8 # 17 # 18 # DFESync # lhsSchema=[?1, ?100, ?7] # -
# 3 # 3 # 1.00 # 0.10 # 2 # 2 # 1.50
# 1.50 # 2 # 0 # 0 # 0 # 0.03 # 0.08
# 18 # 10 # 0 # 0 # 0 # 0
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
```

Amazon Neptune User Guide
Gremlin explain

```
# # # # # # #
#####
# 9 # 10 # - # DFEDistinctColumn # vars=[?7] # -
# 1 # 1 # 1.00 # 0.09 # 1 # 1 # 1.00
# 1.00 # 1 # 1 # 0 # 0 # 0.01 # 0.03
# 8 # 8 # 0 # 0 # 0 # 0
# # # # # inSchema=[?1, ?100, ?7] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # outSchema=[?1, ?100, ?7] #
# # # # # # # # # #
# # # # # # # # # #
#####
# 10 # 11 # 12 # DFETee # inSchema=[?1, ?100, ?7] # -
# 1 # 2 # 2.00 # 0.05 # 1 # 2 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.04 # 0.04
# 16 # 8 # 0 # 0 # 0 # 0
# # # # # outSchema=[?1, ?100, ?7] #
# # # # # # # # # #
# # # # # # # # # #
#####
# 11 # 13 # - # DFEGroupByHashIndex # inSchema=[?1, ?100, ?7] # -
# 1 # 1 # 1.00 # 0.18 # 1 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.01 # 0.02
# 1 # 8 # 0 # 0 # 0 # 0
# # # # # outSchema=[?-1000] #
# # # # # # # # # #
# # # # # # # # # #
#####
# 12 # 14 # - # DFEProject # columns=[?7] # -
# 1 # 1 # 1.00 # 0.08 # 1 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.01 # 0.02
# 2 # 8 # 0 # 0 # 0 # 0
# # # # # inSchema=[?1, ?100, ?7] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # outSchema=[?7] #
# # # # # # # # # #
# # # # # # # # # #
#####
# 13 # 14 # - # DFEDistinctColumn # column=?-1000 # -
# 1 # 1 # 1.00 # 0.11 # 1 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0.01 # 0.01
```


Amazon Neptune User Guide
Gremlin explain

```
# # 1 # 1 # 0 # 0 # 0 # 0
# # # # # # ordered=false #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 14 # 15 # - # DFESelectRows # lhsSchema=[?7] # -
# 2 # 1 # 0.50 # 0.16 # 2 # 1 # 1.00 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0 # 0.01 # 0.02
# 2 # 3 # 0 # 0 # 0 # 0 # 0 # 0
# # # # # # # rhsSchema=[?-1000] #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 15 # 16 # - # DFEDistinctColumn # column=?7 # -
# 1 # 1 # 1.00 # 0.52 # 1 # 1 # 1.00 # 1.00
# 1.00 # 1 # 0 # 0 # 0 # 0 # 0.00 # 0.02
# 4 # 2 # 0 # 0 # 0 # 0 # 0 # 0
# # # # # # # ordered=false #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
# # # # # # # # # # #
#####
# 16 # 8 # - # DFEPipelineJoin # pattern=project ?7 ?9 (?7,?8,?9) # -
# 1 # 2 # 2.00 # 1.38 # 1 # 1 # 1.00 # 1.00
# 2.00 # 1 # 0 # 0 # 0 # 0 # 0.00 # 0.02
# 2 # 2 # 0 # 0 # 0 # 0 # 0 # 0
# # # # # # # constraints=[(?g != defaultGraph)] #
# # # # # # # # # # #
# # # # # # # # # # #
```

Amazon Neptune User Guide
Gremlin explain

```
# # # # #
# # # # # inSchema=[?7] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 17 # 19 # - # DFEMForwardValue # inSchema=[?1, ?100, ?7] # -
# 1 # 1 # 1.00 # 0.05 # 1 # 1 # 1.00
# 1.00 # 1 # 0 # 0 # 0.02 # 0.04
# 8 # 8 # 0 # 0 # 0 # 0
# # # # # outSchema=[?1, ?100, ?7] #
# # # # # # # # # #
# # # # # # # # # #
#####
# 18 # 19 # - # DFEMForwardValue # inSchema=[?7, ?9] # -
# 2 # 2 # 1.00 # 0.03 # 1 # 1 # 2.00
# 2.00 # 1 # 0 # 0 # 0.07 # 0.02
# 2 # 2 # 0 # 0 # 0 # 0
# # # # # outSchema=[?7, ?9] #
# # # # # # # # # #
# # # # # # # # # #
#####
# 19 # 20 # - # DFEMHashIndexJoin # lhsSchema=[?1, ?100, ?7] # -
# 3 # 2 # 0.67 # 0.34 # 2 # 1 # 1.50
# 2.00 # 1 # 0 # 0 # 0.01 # 0.04
# 9 # 10 # 0 # 0 # 0 # 0
# # # # # rhsSchema=[?7, ?9] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####
# 20 # - # - # DFEMDrain # inSchema=[?1, ?100, ?7, ?9] # -
# 2 # 0 # 0.00 # 0.07 # 1 # 0 # 2.00
# 0.00 # 1 # 0 # 0 # 0.00 # 0.04
# 9 # 9 # 0 # 0 # 0 # 0.03 # 0
#
```

```

# # # # # outSchema=[?1, ?100, ?7, ?9] #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
#####

Runtime (ms)
=====
Query Execution: 56.518

Traversal Metrics
=====
Step                                     Count  Traversers
Time (ms)   % Dur

-----
DFEStep(Vertex)                               6      6
  11.748    20.93
NeptuneTraverserConverterDFEStep             6      6
  0.318     0.57
NeptuneHasStep([name.eq(josh)])              1      1
  1.307     2.33
NeptuneInterleavingStep                      4      4
  42.410    75.54
WherePredicateStep(eq(a))                     1      1
  0.186     0.33
NoOpBarrierStep(2500)                         1      1
  0.173     0.31
                                     >TOTAL
    56.144     -

Predicates
=====
# of predicates: 8

Results
=====
Count: 1

Index Operations
=====
Query execution:
  # of statement index ops: 6
  # of unique statement index ops: 4
  Duplication ratio: 1.5
  # of terms materialized: 0

```

Note

Because the DFE engine is an experimental feature released in lab mode, the exact format of the profile output is subject to change.

Tuning Gremlin queries using explain and profile

You can often tune your Gremlin queries in Amazon Neptune to get better performance, using the information available to you in the reports you get from the Neptune [explain \(p. 297\)](#) and [profile \(p. 303\)](#) APIs. To do so, it helps to understand how Neptune processes Gremlin traversals.

Important

A change was made in TinkerPop version 3.4.11 that improves correctness of how queries are processed, but for the moment can sometimes seriously impact query performance.

For example, a query of this sort may run significantly slower:

```
g.V().hasLabel('airport').  
  order().  
    by(out().count(),desc).  
  limit(10).  
  out()
```

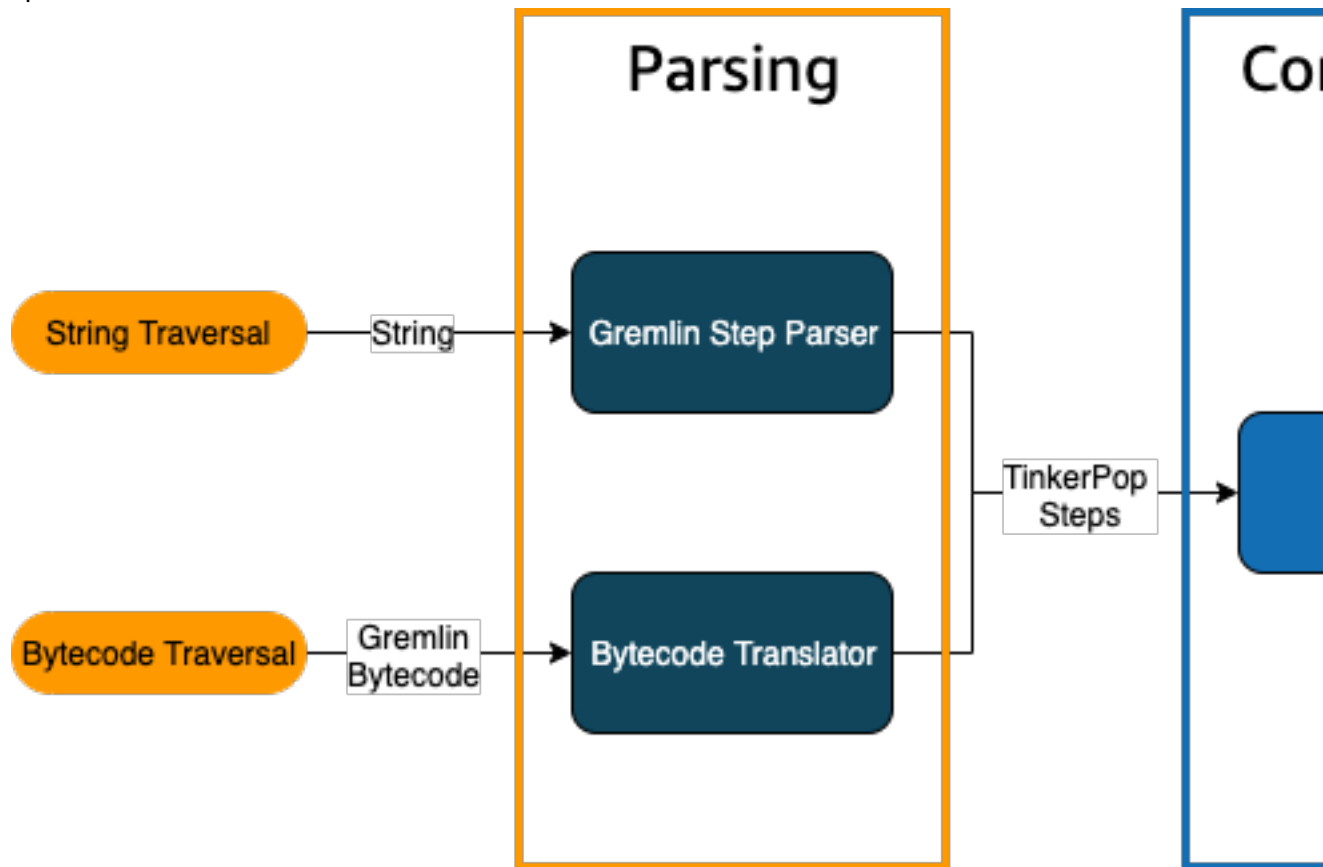
The vertices after the limit step are now fetched in a non-optimal way because of the TinkerPop 3.4.11 change. To avoid this, you can modify the query by adding the `barrier()` step at any point after the `order().by()`. For example:

```
g.V().hasLabel('airport').  
  order().  
    by(out().count(),desc).  
  limit(10).  
  barrier().  
  out()
```

TinkerPop 3.4.11 was enabled in Neptune [engine version 1.0.5.0](#) (p. 725).

Understanding Gremlin traversal processing in Neptune

When a Gremlin traversal is sent to Neptune, there are three main processes that transform the traversal into an underlying execution plan for the engine to execute. These are parsing, conversion, and optimization:



The traversal parsing process

The first step in processing a traversal is to parse it into a common language. In Neptune, that common language is the set of TinkerPop steps that are part of the [TinkerPop API](#). Each of these steps represents a unit of computation within the traversal.

You can send a Gremlin traversal to Neptune either as a string or as bytecode. The REST endpoint and the Java client driver `submit()` method send traversals as strings, as in this example:

```
client.submit("g.V()")
```

Applications and language drivers using [Gremlin language variants \(GLV\)](#) send traversals in bytecode.

The traversal conversion process

The second step in processing a traversal is to convert its TinkerPop steps into a set of converted and non-converted Neptune steps. Most steps in the Apache TinkerPop Gremlin query language are converted to Neptune-specific steps that are optimized to run on the underlying Neptune engine. When a TinkerPop step without a Neptune equivalent is encountered in a traversal, that step and all subsequent steps in the traversal are processed by the TinkerPop query engine.

For more information about what steps can be converted under what circumstances, see [Gremlin step support \(p. 336\)](#).

The traversal optimization process

The final step in traversal processing is to run the series of converted and non-converted steps through the optimizer, to try to determine the best execution plan. The output of this optimization is the execution plan that the Neptune engine processes.

Using the Neptune Gremlin explain API to tune queries

The Neptune explain API is not the same as the `explain()` step. It returns the final execution plan that the Neptune engine would process when executing the query. Because it does not perform any processing, it returns the same plan regardless of the parameters used, and its output contains no statistics about actual execution.

Consider the following simple traversal that finds all the airport vertices for Anchorage:

```
g.V().has('code', 'ANC')
```

There are two ways you can run this traversal through the Neptune `explain` API. The first way is to make a REST call to the `explain` endpoint, like this:

```
curl -X POST https://your-neptune-endpoint:port/gremlin/explain -d  
'{"gremlin": "g.V().has('code', 'ANC')"}'
```

The second way is to use the Neptune workbench's [%%gremlin \(p. 38\)](#) cell magic with the `explain` parameter. This passes the traversal contained in the cell body to the Neptune `explain` API and then displays the resulting output when you run the cell:

```
%%gremlin explain  
g.V().has('code', 'ANC')
```

In either case, the resulting `explain` API output describes Neptune's execution plan for the traversal. As you can see in the image below, the plan includes each of the 3 steps in the processing pipeline:

Explain

```
*****  
                        Neptune Gremlin Explain  
*****
```

Query String

```
=====
```

```
g.V().has('code','ANC')
```

Original Traversal

```
=====
```

```
[GraphStep(vertex,[]), HasStep([code.eq(ANC)])]
```

Converted Traversal

```
=====
```

Neptune steps:

```
[  
  NeptuneGraphQueryStep(Vertex) {  
    JoinGroupNode {  
      PatternNode[ (?1, <-label>, ?2, <->) . project dist  
      PatternNode[ (?1, <code>, "ANC", ?) . project ask  
    }, annotations={path=[Vertex(?1):GraphStep], maxVarId  
  },  
  NeptuneTraverserConverterStep  
]
```

Optimized Traversal

```
=====
```

Neptune steps:

```
[  
  NeptuneGraphQueryStep(Vertex) {  
    JoinGroupNode {  
      PatternNode[ (?1, <code>, "ANC", ?) . project ?1 .  
    }, annotations={path=[Vertex(?1):GraphStep], maxVarId  
  },  
  NeptuneTraverserConverterStep  
]
```

Predicates

```
=====
```

```
# of predicates: 22
```

Tuning a traversal by looking at steps that are not converted

One of the first things to look for in the Neptune `explain` API output is for Gremlin steps that are not converted to Neptune native steps. In a query plan, when a step is encountered that cannot be converted to a Neptune native step, it and all subsequent steps in the plan are processed by the Gremlin server.

In the example above, all steps in the traversal were converted. Let's examine `explain` API output for this traversal:

```
g.V().has('code','ANC').out().choose(hasLabel('airport'), values('code'), constant('Not an airport'))
```

As you can see in the image below, Neptune could not convert the `choose()` step:

Explain

```
*****  
                Neptune Gremlin Explain  
*****
```

Query String

```
*****
```

```
g.V().has('code','ANC').out().choose(hasLabel('airport'), val
```

Original Traversal

```
*****
```

```
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(OU
```

Converted Traversal

```
*****
```

Neptune steps:

```
[  
  NeptuneGraphQueryStep(Vertex) {  
    JoinGroupNode {  
      PatternNode[ (?1, <-label>, ?2, <->) . project dis  
      PatternNode[ (?1, <code>, "ANC", ?) . project ask  
      PatternNode[ (?1, ?5, ?3, ?6) . project ?1,?3 . Is  
      PatternNode[ (?3, <-label>, ?4, <->) . project ask  
    }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3  
  },  
  NeptuneTraverserConverterStep  
]
```

```
+ not converted into Neptune steps: [ChooseStep([HasStep([~la
```

Optimized Traversal

```
*****
```

Neptune steps:

```
[  
  NeptuneGraphQueryStep(Vertex) {  
    JoinGroupNode {  
      PatternNode[ (?1, <code>, "ANC", ?) . project ?1 .  
      PatternNode[ (?1, ?5, ?3, ?6) . project ?1,?3 . Is  
    }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3  
  },  
  NeptuneTraverserConverterStep  
]
```

```
+ not converted into Neptune steps: [ChooseStep([NeptuneHasSt
```

There are several things you could do to tune the performance of the traversal. The first would be to rewrite it in such a way as to eliminate the step that could not be converted. Another would be to move the step to the end of the traversal so that all other steps can be converted to native ones.

A query plan with steps that are not converted does not always need to be tuned. If the steps that cannot be converted are at the end of the traversal, and are related to how output is formatted rather than how the graph is traversed, they may have little effect on performance.

Another thing to look for when examining output from the Neptune explain API is steps that do not use indexes. The following traversal finds all airports with flights that land in Anchorage:

```
g.V().has('code','ANC').in().values('code')
```

Output from the explain API for this traversal is:

```
*****
                Neptune Gremlin Explain
*****

Query String
=====

g.V().has('code','ANC').in().values('code')

Original Traversal
=====
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(IN,vertex),
 PropertiesStep([code],value)]

Converted Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(PropertyValue) {
    JoinGroupNode {
      PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .]
      PatternNode[(?1, <code>, "ANC", ?) . project ask .]
      PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
      PatternNode[(?3, <-label>, ?4, <->) . project ask .]
      PatternNode[(?3, ?7, ?8, <->) . project ?3,?8 . ContainsFilter(?7 in
(<code>)) .]
    }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep, PropertyValue(?
8):PropertiesStep], maxVarId=9}
  },
  NeptuneTraverserConverterStep
]

Optimized Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(PropertyValue) {
    JoinGroupNode {
      PatternNode[(?1, <code>, "ANC", ?) . project ?1 .], {estimatedCardinality=1}
      PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .],
{estimatedCardinality=INFINITY}
      PatternNode[(?3, ?7=<code>, ?8, <->) . project ?3,?8 .],
{estimatedCardinality=7564}
    }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep, PropertyValue(?
8):PropertiesStep], maxVarId=9}
  },
  NeptuneTraverserConverterStep
]
```

```
]

Predicates
=====
# of predicates: 26

WARNING: reverse traversal with no edge label(s) - .in() / .both() may impact query
performance
```

The WARNING message at the bottom of the output occurs because the `in()` step in the traversal cannot be handled using one of the 3 indexes that Neptune maintains (see [How Statements Are Indexed in Neptune \(p. 61\)](#) and [Gremlin statements in Neptune \(p. 293\)](#)). Because the `in()` step contains no edge filter, it cannot be resolved using the SPOG, POGS or GPSO index. Instead, Neptune must perform a union scan to find the requested vertices, which is much less efficient.

There are two ways to tune the traversal in this situation. The first is to add one or more filtering criteria to the `in()` step so that an indexed lookup can be used to resolve the query. For the example above, this might be:

```
g.V().has('code','ANC').in('route').values('code')
```

Output from the Neptune `explain` API for the revised traversal no longer contains the WARNING message:

```
*****
                Neptune Gremlin Explain
*****

Query String
=====

g.V().has('code','ANC').in('route').values('code')

Original Traversal
=====
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(IN,[route],vertex),
 PropertiesStep([code],value)]

Converted Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(PropertyValue) {
    JoinGroupNode {
      PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .]
      PatternNode[(?1, <code>, "ANC", ?) . project ask .]
      PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .
ContainsFilter(?5 in (<route>)) .]
      PatternNode[(?3, <-label>, ?4, <->) . project ask .]
      PatternNode[(?3, ?7, ?8, <->) . project ?3,?8 . ContainsFilter(?7 in
(<code>)) .]
    }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep, PropertyValue(?
8):PropertiesStep], maxVarId=9}
  },
  NeptuneTraverserConverterStep
]

Optimized Traversal
=====
Neptune steps:
[
  NeptuneGraphQueryStep(PropertyValue) {
```

```
JoinGroupNode {
  PatternNode[(?1, <code>, "ANC", ?) . project ?1 .], {estimatedCardinality=1}
  PatternNode[(?3, ?5=<route>, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .],
{estimatedCardinality=32042}
  PatternNode[(?3, ?7=<code>, ?8, <->) . project ?3,?8 .],
{estimatedCardinality=7564}
  }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep, PropertyValue(?
8):PropertiesStep], maxVarId=9}
  },
  NeptuneTraverserConverterStep
]

Predicates
=====
# of predicates: 26
```

Another option if you are running many traversals of this kind is to run them in a Neptune DB cluster that has the optional OSGP index enabled (see [Enabling an OSGP Index \(p. 62\)](#)). Enabling an OSGP index has drawbacks:

- It must be enabled in a DB cluster before any data is loaded.
- Insertion rates for vertices and edges may slow by up to 23%.
- Storage usage will increase by around 20%.
- Read queries that scatter requests across all indexes may have increased latencies.

Having an OSGP index makes a lot of sense for a restricted set of query patterns, but unless you are running those frequently, it is usually preferable to try to ensure that the traversals you write can be resolved using the three primary indexes.

Using a large number of predicates

Neptune treats each edge label and each distinct vertex or edge property name in your graph as a predicate, and is designed by default to work with a relatively low number of distinct predicates. When you have more than a few thousand predicates in your graph data, performance can degrade.

Neptune explain output will warn you if this is the case:

```
Predicates
=====
# of predicates: 9549
WARNING: high predicate count (# of distinct property names and edge labels)
```

If it is not convenient to rework your data model to reduce the number of labels and properties, and therefore the number of predicates, the best way to tune traversals is to run them in a DB cluster that has the OSGP index enabled, as discussed above.

Using the Neptune Gremlin profile API to tune traversals

The Neptune profile API is quite different from the Gremlin profile() step. Like the explain API, its output includes the query plan that the Neptune engine uses when executing the traversal. In addition, the profile output includes actual execution statistics for the traversal, given how its parameters are set.

Again, take the simple traversal that finds all airport vertices for Anchorage:

```
g.V().has('code', 'ANC')
```

As with the `explain` API, you can invoke the `profile` API using a REST call:

```
curl -X POST https://your-neptune-endpoint:port/gremlin/profile -d  
'{"gremlin": "g.V().has('code', 'ANC')"}'
```

You use also the Neptune workbench's [%%gremlin \(p. 38\)](#) cell magic with the `profile` parameter. This passes the traversal contained in the cell body to the Neptune `profile` API and then displays the resulting output when you run the cell:

```
%%gremlin profile  
  
g.V().has('code', 'ANC')
```

In either case, the resulting `profile` API output contains both Neptune's execution plan for the traversal and statistics about the plan's execution, as you can see in this image:

Profile

```
*****  
                        Neptune Gremlin Profile  
*****
```

Query String

```
=====
```

```
g.V().has('code','ANC')
```

Original Traversal

```
=====
```

```
[GraphStep(vertex,[]), HasStep([code.eq(ANC)])]
```

Optimized Traversal

```
=====
```

Neptune steps:

```
[  
  NeptuneGraphQueryStep(Vertex) {  
    JoinGroupNode {  
      PatternNode[ (?1, <code>, "ANC", ?) . project ?1 .  
    }, annotations={path=[Vertex(?1):GraphStep], joinStats=  
  },  
  NeptuneTraverserConverterStep  
]
```

Physical Pipeline

```
=====
```

NeptuneGraphQueryStep

```
|-- StartOp
```

```
|-- JoinGroupOp
```

```
  |-- SpoolerOp(1000)
```

```
  |-- DynamicJoinOp(PatternNode[ (?1, <code>, "ANC", ?) . project ?1 .
```

Runtime (ms)

```
=====
```

Query Execution: 5.096

Traversal Metrics

```
=====
```

Step

```
-----
```

NeptuneGraphQueryStep(Vertex)

NeptuneTraverserConverterStep

In `profile` output, the execution plan section only contains the final execution plan for the traversal, not the intermediate steps. The pipeline section contains the physical pipeline operations that were performed as well as the actual time (in milliseconds) that traversal execution took. The runtime metric is extremely helpful in comparing the times that two different versions of a traversal take as you are optimizing them.

Note

The initial runtime of a traversal is generally longer than subsequent runtimes, because the first one causes the relevant data to be cached.

The third section of the `profile` output contains execution statistics and the results of the traversal. To see how this information can be useful in tuning a traversal, consider the following traversal, which finds every airport whose name begins with "Anchorage", and all the airports reachable in two hops from those airports, returning airport codes, flight routes, and distances:

```
%%gremlin profile
g.withSideEffect("Neptune#fts.endpoint", "{your-OpenSearch-endpoint-URL}").
  V().has("city", "Neptune#fts Anchorage~").
  repeat(outE('route').inV().simplePath()).times(2).
  project('Destination', 'Route').
  by('code').
  by(path().by('code').by('dist'))
```

Traversal metrics in Neptune profile API output

The first set of metrics that is available in all `profile` output is the traversal metrics. These are similar to the Gremlin `profile()` step metrics, with a few differences:

Traversal Metrics			
=====			
Step		Count	Traversers
Time (ms)	% Dur		

NeptuneGraphQueryStep(Vertex)		3856	3856
91.701	9.09		
NeptuneTraverserConverterStep		3856	3856
38.787	3.84		
ProjectStep([Destination, Route],[value(code), ...		3856	3856
878.786	87.07		
PathStep([value(code), value(dist)])		3856	3856
601.359			
		>TOTAL	-
1009.274	-		

The first column of the traversal-metrics table lists the steps executed by the traversal. The first two steps are generally the Neptune-specific steps, `NeptuneGraphQueryStep` and `NeptuneTraverserConverterStep`.

`NeptuneGraphQueryStep` represents the execution time for the entire portion of the traversal that could be converted and executed natively by the Neptune engine.

`NeptuneTraverserConverterStep` represents the process of converting the output of those converted steps into TinkerPop traversers which allow steps that could not be converted steps, if any, to be processed, or to return the results in a TinkerPop-compatible format.

In the example above, we have several non-converted steps, so we see that each of these TinkerPop steps (`ProjectStep`, `PathStep`) then appears as a row in the table.

The second column in the table, `Count`, reports the number of *represented* traversers that passed through the step, while the third column, `Traversers`, reports the number of traversers which passed through that step, as explained in the [TinkerPop profile step documentation](#).

In our example there are 3,856 vertices and 3,856 traversers returned by the `NeptuneGraphQueryStep`, and these numbers remain the same throughout the remaining processing because `ProjectStep` and `PathStep` are formatting the results, not filtering them.

Note

Unlike TinkerPop, the Neptune engine does not optimize performance by *bulking* in its `NeptuneGraphQueryStep` and `NeptuneTraverserConverterStep` steps. Bulking is the TinkerPop operation that combines traversers on the same vertex to reduce operational overhead, and that is what causes the `Count` and `Traversers` numbers to differ. Because bulking only occurs in steps that Neptune delegates to TinkerPop, and not in steps that Neptune handles natively, the `Count` and `Traverser` columns seldom differ.

The `Time` column reports the number of milliseconds that the step took, and the `% Dur` column reports what percent of the total processing time the step took. These are the metrics that tell you where to focus your tuning efforts by showing the steps that took the most time.

Index operation metrics in Neptune profile API output

Another set of metrics in the output of the Neptune profile API is the index operations:

```
Index Operations
=====
Query execution:
  # of statement index ops: 23191
  # of unique statement index ops: 5960
  Duplication ratio: 3.89
  # of terms materialized: 0
```

These report:

- The total number of index lookups.
- The number of unique index lookups performed.
- The ratio of total index lookups to unique ones. A lower ratio indicates less redundancy.
- The number of terms materialized from the term dictionary.

Repeat metrics in Neptune profile API output

If your traversal uses a `repeat()` step as in the example above, then a section containing repeat metrics appears in the profile output:

```
Repeat Metrics
=====
Iteration  Visited  Output  Until  Emit  Next
-----
          0         2       0       0       0       2
          1        53       0       0       0       53
          2       3856     3856     3856     0       0
-----
          3911     3856     3856     0       55
```

These report:

- The loop count for a row (the `Iteration` column).
- The number of elements visited by the loop (the `Visited` column).
- The number of elements output by the loop (the `Output` column).
- The last element output by the loop (the `Until` column).
- The number of elements emitted by the loop (the `Emit` column).
- The number of elements passed from the loop to the subsequent loop (the `Next` column).

These repeat metrics are very helpful in understanding the branching factor of your traversal, to get a feeling for how much work is being done by the database. You can use these numbers to diagnose performance problems, especially when the same traversal performs dramatically differently with different parameters.

Full-text search metrics in Neptune profile API output

When a traversal uses a [full-text search \(p. 471\)](#) lookup, as in the example above, then a section containing the full-text search (FTS) metrics appears in the profile output:

```
FTS Metrics
=====
SearchNode[(idVar=?1, query=Anchor~, field=city) . project ?1 .],
  {endpoint=your-OpenSearch-endpoint-URL, incomingSolutionsThreshold=1000,
  estimatedCardinality=INFINITY,
  remoteCallTimeSummary=[total=65, avg=32.500000, max=37, min=28],
  remoteCallTime=65, remoteCalls=2, joinTime=0, indexTime=0, remoteResults=2}

  2 result(s) produced from SearchNode above
```

This shows the query sent to the ElasticSearch (ES) cluster and reports several metrics about the interaction with ElasticSearch that can help you pinpoint performance problems relating to full-text search:

- Summary information about the calls into the ElasticSearch index:
 - The total number of milliseconds required by all remoteCalls to satisfy the query (`total`).
 - The average number of milliseconds spent in a remoteCall (`avg`).
 - The minimum number of milliseconds spent in a remoteCall (`min`).
 - The maximum number of milliseconds spent in a remoteCall (`max`).
- Total time consumed by remoteCalls to ElasticSearch (`remoteCallTime`).
- The number of remoteCalls made to ElasticSearch (`remoteCalls`).
- The number of milliseconds spent in joins of ElasticSearch results (`joinTime`).
- The number of milliseconds spent in index lookups (`indexTime`).
- The total number of results returned by ElasticSearch (`remoteResults`).

Native Gremlin step support in Amazon Neptune

The Amazon Neptune engine does not currently have full native support for all Gremlin steps, as explained in [Tuning Gremlin queries \(p. 322\)](#). Current support falls into four categories:

- [Gremlin steps that can always be converted to native Neptune engine operations \(p. 336\)](#)
- [Gremlin steps that can be converted to native Neptune engine operations in some cases \(p. 337\)](#)
- [Gremlin steps that are never converted to native Neptune engine operations \(p. 340\)](#)
- [Gremlin steps that are not supported in Neptune at all \(p. 340\)](#)

Gremlin steps that can always be converted to native Neptune engine operations

Many Gremlin steps can be converted to native Neptune engine operations as long as they meet the following conditions:

- They are not preceded in the query by a step that cannot be converted.
- Their parent step, if any, can be converted,

- All their child traversals, if any, can be converted.

The following Gremlin steps are always converted to native Neptune engine operations if they meet those conditions:

- `and()`
- `as()`
- `count()`
- `E()`
- `emit()`
- `explain()`
- `group()`
- `groupCount()`
- `has()`
- `identity()`
- `is()`
- `key()`
- `label()`
- `limit()`
- `local()`
- `loops()`
- `not()`
- `or()`
- `profile()`
- `properties()`
- `subgraph()`
- `until()`
- `V()`
- `value()`
- `valueMap()`
- `values()`

Gremlin steps that can be converted to native Neptune engine operations in some cases

Some Gremlin steps can be converted to native Neptune engine operations in some situations but not in others:

- `addE()` – The `addE()` step can generally be converted to a native Neptune engine operation, unless it is immediately followed by a `property()` step containing a traversal as a key.
- `addV()` – The `addV()` step can generally be converted to a native Neptune engine operation, unless it is immediately followed by a `property()` step containing a traversal as a key, or unless multiple labels are assigned.
- `aggregate()` – The `aggregate()` step can generally be converted to a native Neptune engine operation, unless the step is used in a child traversal or sub-traversal, or unless the value being stored is something other than a vertex, edge, id, label or property value.

In example below, `aggregate()` is not converted because it is being used in a child traversal:

```
g.V().has('code', 'ANC').as('a')
    .project('flights').by(select('a'))
    .outE().aggregate('x')
```

In this example, `aggregate()` is not converted because what is stored is the `min()` of a value:

```
g.V().has('code', 'ANC').outE().aggregate('x').by(values('dist')).min()
```

- **`barrier()`** – The `barrier()` step can generally be converted to a native Neptune engine operation, unless the step following it is not converted.
- **`cap()`** – The only case in which the `cap()` step is converted is when it is combined with the `unfold()` step to return an unfolded version of an aggregate of vertex, edge, id, or property values. In this example, `cap()` will be converted because it is followed by `.unfold()`:

```
g.V().has('airport', 'country', 'IE').aggregate('airport').limit(2)
    .cap('airport').unfold()
```

However, if you remove the `.unfold()`, `cap()` will not be converted:

```
g.V().has('airport', 'country', 'IE').aggregate('airport').limit(2)
    .cap('airport')
```

- **`coalesce()`** – The only case where the `coalesce()` step is converted is when it follows the [Upsert pattern](#) recommended on the [TinkerPop recipes page](#). Other `coalesce()` patterns are not allowed. Conversion is limited to the case where all child traversals can be converted, they all produce the same type as output (vertex, edge, id, value, key, or label), they all traverse to a new element, and they do not contain the `repeat()` step.
- **`constant()`** – The `constant()` step is currently only converted if it is used within a `sack().by()` part of a traversal to assign a constant value, like this:

```
g.V().has('code', 'ANC').sack(assign).by(constant(10)).out().limit(2)
```

- **`cyclicPath()`** – The `cyclicPath()` step can generally be converted to a native Neptune engine operation, unless the step is used with `by()`, `from()`, or `to()` modulators. In the following queries, for example, `cyclicPath()` is not converted:

```
g.V().has('code', 'ANC').as('a').out().out().cyclicPath().by('code')
g.V().has('code', 'ANC').as('a').out().out().cyclicPath().from('a')
g.V().has('code', 'ANC').as('a').out().out().cyclicPath().to('a')
```

- **`drop()`** – The `drop()` step can generally be converted to a native Neptune engine operation, unless the step is used inside a `sideEffect()` or `optional()` step.
- **`fold()`** – There are only two situations where the `fold()` step can be converted, namely when it is used in the [Upsert pattern](#) recommended on the [TinkerPop recipes page](#), and when it is used in a `group().by()` context like this:

```
g.V().has('code', 'ANC').out().group().by().by(values('code', 'city')).fold()
```

- **`id()`** – The `id()` step is converted unless it is used on a property, like this:

```
g.V().has('code', 'ANC').properties('code').id()
```

- **`order()`** – The `order()` step can generally be converted to a native Neptune engine operation, unless one of the following is true:

- The `order()` step is within a nested child traversal, like this:

```
g.V().has('code', 'ANC').where(V().out().order().by(id))
```

- Local ordering is being used, as for example with `order(local)`.
- A custom comparator is being used in the `by()` modulation to order by. An example is this use of `sack()`:

```
g.withSack(0).
  V().has('code', 'ANC').
    repeat(outE().sack(sum).by('dist').inV()).times(2).limit(10).
      order().by(sack())
```

- There are multiple orderings on the same element.
- **project()** – The `project()` step can generally be converted to a native Neptune engine operation, unless the number of `by()` statements following the `project()` does not match the number of labels specified, as here:

```
g.V().has('code', 'ANC').project('x', 'y').by(id)
```

- **range()** – The `range()` step is only converted when the lower end of the range in question is zero (for example, `range(0, 3)`).
- **repeat()** – The `repeat()` step can generally be converted to a native Neptune engine operation, unless it is nested within another `repeat()` step, like this:

```
g.V().has('code', 'ANC').repeat(out().repeat(out()).times(2)).times(2)
```

- **sack()** – The `sack()` step can generally be converted to a native Neptune engine operation, except in the following cases:
 - If a non-numeric sack operator is being used.
 - If a numeric sack operator other than `+`, `-`, `mult`, `div`, `min` and `max` is being used.
 - If `sack()` is used inside a `where()` step to filter based on a sack value, as here:

```
g.V().has('code', 'ANC').sack(assign).by(values('code')).where(sack().is('ANC'))
```

- **sum()** – The `sum()` step can generally be converted to a native Neptune engine operation, but not when used to calculate a global summation, like this:

```
g.V().has('code', 'ANC').outE('routes').values('dist').sum()
```

- **union()** – The `union()` step can be converted to a native Neptune engine operation except when used as a non-terminal step.
- **unfold()** – The `unfold()` step can only be converted to a native Neptune engine operation when it is used in the [Upsert pattern](#) recommended on the [TinkerPop recipes page](#), and when it is used together with `cap()` like this:

```
g.V().has('airport', 'country', 'IE').aggregate('airport').limit(2)
  .cap('airport').unfold()
```

- **where()** – The `where()` step can generally be converted to a native Neptune engine operation, except in the following cases:
 - When `by()` modulations are used, like this:

```
g.V().hasLabel('airport').as('v').
```

```
.where(gt('a')).by('runways')
```

- When comparison operators other than `eq`, `neg`, `within`, and `without` are used.
- When user-supplied aggregations are used.

Gremlin steps that are never converted to native Neptune engine operations

The following Gremlin steps are supported in Neptune but are never converted to native Neptune engine operations. Instead, they are executed by the Gremlin server.

- `choose()`
- `coin()`
- `inject()`
- `match()`
- `math()`
- `max()`
- `mean()`
- `min()`
- `option()`
- `optional()`
- `path()`
- `propertyMap()`
- `sample()`
- `skip()`
- `tail()`
- `timeLimit()`
- `tree()`

Gremlin steps that are not supported in Neptune at all

The following Gremlin steps are not supported at all in Neptune. In most cases this is because they require a `GraphComputer`, which Neptune does not currently support.

- `connectedComponent()`
- `io()`
- `shortestPath()`
- `withComputer()`
- `pageRank()`
- `peerPressure()`
- `program()`

Using Gremlin with the Neptune DFE query engine

If you fully enable the Neptune [alternative query engine \(p. 87\)](#) known as the DFE in [lab mode \(p. 85\)](#) (by setting the `neptune_lab_mode` DB cluster parameter to `DFEQueryEngine=enabled`), then Neptune translates read-only Gremlin queries/traversals into an intermediate logical representation and runs them on the DFE engine whenever possible.

However, the DFE does not yet support all Gremlin steps. When a step can't be run natively on the DFE, Neptune falls back on TinkerPop to run the step. The `explain` and `profile` reports include warnings when this happens.

Note

Beginning with [engine release 1.0.5.0 \(p. 725\)](#), the default DFE behavior for handling Gremlin steps without native support has changed. Where previously the DFE engine fell back on the Neptune Gremlin engine, now it falls back on the vanilla TinkerPop engine.

Gremlin steps that are natively supported by the DFE engine

- **GraphStep**
- **VertexStep**
- **EdgeVertexStep**
- **IdStep**
- **TraversalFilterStep**
- **PropertiesStep**
- **HasStep** filtering support for vertices and edges on properties and ids and labels, with the exception of text and `Without` predicates.
- **WherePredicateStep** with `Path`-scoped filters, but no `ByModulation`, `SideEffect` or `Map` look up support
- **DedupGlobalStep**, excepting `ByModulation`, `SideEffect`, and `Map` look-up support.

Query planning interleaving

When the translation process encounters a Gremlin step that does not have a corresponding native DFE operator, before falling back to using Tinkerpop it tries to find other intermediate query parts that can be run natively on the DFE engine. It does this by applying interleaving logic to the top level traversal. The result is that supported steps are used wherever possible.

Any such intermediate, non-prefix query translation is represented using `NeptuneInterleavingStep` in the `explain` and `profile` outputs.

For performance comparison, you might want to turn off interleaving in a query, while still using the DFE engine to run the prefix part. Or, you might want to use only the TinkerPop engine for non-prefix query execution. You can do this by using `disableInterleaving` query hint.

Just as the [useDFE \(p. 282\)](#) query hint with a value of `false` prevents a query from being run on the DFE at all, the `disableInterleaving` query hint with a value of `true` turns off DFE interleaving for translation of a query. For example:

```
g.with('Neptune#disableInterleaving', true)
  .V().has('genre', 'drama').in('likes')
```

Updated Gremlin `explain` and `profile` output

Gremlin [explain \(p. 292\)](#) provides details about the optimized traversal that Neptune uses to run a query. See the [sample DFE explain output \(p. 301\)](#) for an example of what `explain` output looks like when the DFE engine is enabled.

The [Gremlin profile API \(p. 303\)](#) runs a specified Gremlin traversal, collects various metrics about the run, and produces a profile report that contains details about the optimized query plan and the

runtime statistics of various operators. See [sample DFE profile output \(p. 307\)](#) for an example of what `profile` output looks like when the DFE engine is enabled.

Note

Because the DFE engine is an experimental feature released in lab mode, the exact format of the `explain` and `profile` output is subject to change.

Accessing the Neptune graph with SPARQL

SPARQL is a query language for the Resource Description Framework (RDF), which is a graph data format designed for the web. Amazon Neptune is compatible with SPARQL 1.1. This means that you can connect to a Neptune DB instance and query the graph using the query language described in the [SPARQL 1.1 Query Language](#) specification.

A query in SPARQL consists of a `SELECT` clause to specify the variables to return and a `WHERE` clause to specify which data to match in the graph. If you are unfamiliar with SPARQL queries, see [Writing Simple Queries](#) in the [SPARQL 1.1 Query Language](#).

Important

To load data, `SPARQL UPDATE INSERT` may work well for a small dataset, but if you need to load a substantial amount of data from a file, see [Using the Amazon Neptune Bulk Loader to Ingest Data \(p. 190\)](#).

For more information about the specifics of Neptune's SPARQL implementation, see [SPARQL Specifics \(p. 56\)](#).

Before you begin, you must have the following:

- A Neptune DB instance. For information about creating a Neptune DB instance, see [Creating a New Neptune DB Cluster Using AWS CloudFormation or Manually \(p. 166\)](#).
- An Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

Topics

- [Using the RDF4J console to connect to a Neptune DB instance \(p. 342\)](#)
- [Using RDF4J Workbench to connect to a Neptune DB instance \(p. 344\)](#)
- [Using Java to connect to a Neptune DB instance \(p. 345\)](#)
- [SPARQL HTTP API \(p. 348\)](#)
- [SPARQL query hints \(p. 357\)](#)
- [SPARQL query status API \(p. 365\)](#)
- [SPARQL query cancellation \(p. 367\)](#)
- [Using the SPARQL 1.1 Graph Store HTTP Protocol \(GSP\) in Amazon Neptune \(p. 368\)](#)
- [Analyzing Neptune query execution using SPARQL explain \(p. 369\)](#)
- [SPARQL federated queries in Neptune using the SERVICE extension \(p. 390\)](#)

Using the RDF4J console to connect to a Neptune DB instance

The RDF4J Console allows you to experiment with Resource Description Framework (RDF) graphs and queries in a REPL (read-eval-print loop) environment.

You can add a remote graph database as a repository and query it from the RDF4J Console. This section walks you through the configuration of the RDF4J Console to connect remotely to a Neptune DB instance.

To connect to Neptune using the RDF4J Console

1. Download the RDF4J SDK from the [Download page](#) on the RDF4J website.
2. Unzip the RDF4J SDK zip file.
3. In a terminal, navigate to the RDF4J SDK directory, and then enter the following command to run the RDF4J Console:

```
bin/console.sh
```

You should see output similar to the following:

```
14:11:51.126 [main] DEBUG o.e.r.c.platform.PlatformFactory - os.name = linux
14:11:51.130 [main] DEBUG o.e.r.c.platform.PlatformFactory - Detected Posix platform
Connected to default data directory
RDF4J Console 3.6.1

3.6.1
Type 'help' for help.
>
```

You are now at the > prompt. This is the general prompt for the RDF4J Console. You use this prompt for setting up repositories and other operations. A repository has its own prompt for running queries.

4. At the > prompt, enter the following to create a SPARQL repository for your Neptune DB instance:

```
create sparql
```

5. The RDF4J Console prompts you for values for the variables required to connect to the SPARQL endpoint.

```
Please specify values for the following variables:
```

Specify the following values:

Variable Name	Value
SPARQL query endpoint	<code>https://<i>your-neptune-endpoint</i>:<i>port</i>/sparql</code>
SPARQL update endpoint	<code>https://<i>your-neptune-endpoint</i>:<i>port</i>/sparql</code>
Local repository ID [endpoint@localhost]	neptune
Repository title [SPARQL endpoint repository @localhost]	Neptune DB instance

For information about finding the address of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

If the operation is successful, you see the following message:

```
Repository created
```

6. At the `>` prompt, enter the following to connect to the Neptune DB instance:

```
open neptune
```

If the operation is successful, you see the following message:

```
Opened repository 'neptune'
```

You are now at the `neptune>` prompt. At this prompt, you can run queries against the Neptune graph.

Note

Now that you have added the repository, the next time you run `bin/console.sh`, you can immediately run the `open neptune` command to connect to the Neptune DB instance.

7. At the `neptune>` prompt, enter the following to run a SPARQL query that returns up to 10 of the triples (subject-predicate-object) in the graph by using the `?s ?p ?o` query with a limit of 10. To query for something else, replace the text after the `sparql` command with another SPARQL query.

```
sparql select ?s ?p ?o where {?s ?p ?o} limit 10
```

Using RDF4J Workbench to connect to a Neptune DB instance

This section walks you through connecting to an Amazon Neptune DB instance using RDF4J Workbench and RDF4J Server. RDF4J Server is required because it acts as a proxy between the Neptune SPARQL HTTP REST endpoint and RDF4J Workbench.

RDF4J Workbench provides an easy interface for experimenting with a graph, including loading local files. For information, see the [Add section](#) in the RDF4J documentation.

Prerequisites

Before you begin, do the following:

- Install Java 1.8 or later.
- Install RDF4J Server and RDF4J Workbench. For information, see [Installing RDF4J Server and RDF4J Workbench](#).

To use RDF4J Workbench to connect to Neptune

1. In a web browser, navigate to the URL where the RDF4J Workbench web app is deployed. For example, if you are using Apache Tomcat, the URL is: https://ec2_hostname:8080/rdf4j-workbench/.
2. If you are asked to **Connect to RDF4J Server**, verify that **RDF4J Server** is installed, running, and that the server URL is correct. Then, proceed to the next step.

3. In the left pane, choose **New repository**.

In **New repository**:

- In the **Type** drop-down list, choose **SPARQL endpoint proxy**.
- For **ID**, type **neptune**.
- For **Title**, type **Neptune DB instance**.

Choose **Next**.

4. In **New repository**:

- For **SPARQL query endpoint URL**, type `https://your-neptune-endpoint:port/sparql`.
- For **SPARQL update endpoint URL**, type `https://your-neptune-endpoint:port/sparql`.

For information about finding the address of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

Choose **Create**.

5. The **neptune** repository now appears in the list of repositories. It might take a few minutes before you can use the new repository.
6. In the **Id** column of the table, choose the **neptune** link.
7. In the left pane, choose **Query**.

Note

If the menu items under **Explore** are disabled, you might need to reconnect to the RDF4J Server and choose the **neptune** repository again. You can do this by using the **[change]** links in the upper-right corner.

8. In the query field, type the following SPARQL query, and then choose **Execute**.

```
select ?s ?p ?o where {?s ?p ?o} limit 10
```

The preceding example returns up to 10 of the triples (subject-predicate-object) in the graph by using the `?s ?p ?o` query with a limit of 10.

Using Java to connect to a Neptune DB instance

This section walks you through the running of a complete Java sample that connects to an Amazon Neptune DB instance and performs a SPARQL query.

Follow these instructions from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

To connect to Neptune using Java

1. Install Apache Maven on your EC2 instance. First, enter the following to add a repository with a Maven package:

```
sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
```

Enter the following to set the version number for the packages:

```
sudo sed -i s/\$releasever/6/g /etc/yum.repos.d/epel-apache-maven.repo
```

Then you can use **yum** to install Maven:

```
sudo yum install -y apache-maven
```

2. This example was tested with Java 8 only. Enter the following to install Java 8 on your EC2 instance:

```
sudo yum install java-1.8.0-devel
```

3. Enter the following to set Java 8 as the default runtime on your EC2 instance:

```
sudo /usr/sbin/alternatives --config java
```

When prompted, enter the number for Java 8.

4. Enter the following to set Java 8 as the default compiler on your EC2 instance:

```
sudo /usr/sbin/alternatives --config javac
```

When prompted, enter the number for Java 8.

5. In a new directory, create a `pom.xml` file, and then open it in a text editor.
6. Copy the following into the `pom.xml` file and save it (you can usually adjust the version numbers to the latest stable version):

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://www.w3.org/2001/
XMLSchema-instance"
  xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws</groupId>
  <artifactId>RDFExample</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>RDFExample</name>
  <url>https://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.eclipse.rdf4j</groupId>
      <artifactId>rdf4j-runtime</artifactId>
      <version>3.6</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.2.1</version>
        <configuration>
          <mainClass>com.amazonaws.App</mainClass>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
```

```
<configuration>
  <source>1.8</source>
  <target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

Note

If you are modifying an existing Maven project, the required dependency is highlighted in the preceding code.

7. To create subdirectories for the example source code (`src/main/java/com/amazonaws/`), enter the following at the command line:

```
mkdir -p src/main/java/com/amazonaws/
```

8. In the `src/main/java/com/amazonaws/` directory, create a file named `App.java`, and then open it in a text editor.
9. Copy the following into the `App.java` file. Replace `your-neptune-endpoint` with the address of your Neptune DB instance.

Note

For information about finding the hostname of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

```
package com.amazonaws;

import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.http.HTTPRepository;
import org.eclipse.rdf4j.repository.sparql.SPARQLRepository;

import java.util.List;
import org.eclipse.rdf4j.RDF4JException;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.model.Value;

public class App
{
    public static void main( String[] args )
    {
        String sparqlEndpoint = "https://your-neptune-endpoint:port/sparql";
        Repository repo = new SPARQLRepository(sparqlEndpoint);
        repo.initialize();

        try (RepositoryConnection conn = repo.getConnection()) {
            String queryString = "SELECT ?s ?p ?o WHERE { ?s ?p ?o } limit 10";

            TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
                queryString);

            try (TupleQueryResult result = tupleQuery.evaluate()) {
                while (result.hasNext()) { // iterate over the result
                    BindingSet bindingSet = result.next();

                    Value s = bindingSet.getValue("s");
                    Value p = bindingSet.getValue("p");
                    Value o = bindingSet.getValue("o");
                }
            }
        }
    }
}
```

```
        System.out.print(s);  
        System.out.print("\t");  
        System.out.print(p);  
        System.out.print("\t");  
        System.out.println(o);  
    }  
} } }
```

10. Use the following Maven command to compile and run the sample:

```
mvn compile exec:java
```

The preceding example returns up to 10 of the triples (subject-predicate-object) in the graph by using the `?s ?p ?o` query with a limit of 10. To query for something else, replace the query with another SPARQL query.

The iteration of the results in the example prints the value of each variable returned. The `value` object is converted to a `String` and then printed. If you change the `SELECT` part of the query, you must modify the code.

SPARQL HTTP API

SPARQL HTTP requests are accepted at the following endpoint: `https://your-neptune-endpoint:port/sparql`

For more information about connecting to Amazon Neptune with SPARQL, see [Accessing the Neptune graph with SPARQL \(p. 342\)](#).

For more information about the SPARQL protocol and query language, see the [SPARQL 1.1 Protocol](#) and the [SPARQL 1.1 Query Language](#) specification.

The following topics provide information about SPARQL RDF serialization formats and how to use the SPARQL HTTP API with Neptune.

Contents

- [Using the HTTP REST endpoint to connect to a Neptune DB instance \(p. 348\)](#)
- [Optional HTTP trailing headers for multi-part SPARQL responses \(p. 349\)](#)
- [RDF media types used by SPARQL in Neptune \(p. 350\)](#)
 - [RDF serialization formats used by Neptune SPARQL \(p. 350\)](#)
 - [SPARQL result serialization formats used by Neptune SPARQL \(p. 350\)](#)
 - [Media-Types that Neptune can use to import RDF data \(p. 351\)](#)
 - [Media-Types that Neptune can use to export query results \(p. 351\)](#)
- [Using SPARQL UPDATE LOAD to import data into Neptune \(p. 352\)](#)
- [Using SPARQL UPDATE UNLOAD to delete data from Neptune \(p. 355\)](#)

Using the HTTP REST endpoint to connect to a Neptune DB instance

Amazon Neptune provides an HTTP endpoint for SPARQL queries. The REST interface is compatible with SPARQL version 1.1.

Important

[Release: 1.0.4.0 \(2020-10-12\) \(p. 739\)](#) made TLS 1.2 and HTTPS mandatory for all connections to Amazon Neptune. It is no longer possible to connect to Neptune using unsecured HTTP, or using HTTPS with a version of TLS earlier than 1.2.

The following instructions walk you through connecting to the SPARQL endpoint using the **curl** command, connecting through HTTPS, and using HTTP syntax. Follow these instructions from an Amazon EC2 instance in the same virtual private cloud (VPC) as your Neptune DB instance.

The HTTP endpoint for SPARQL queries to a Neptune DB instance is: `https://your-neptune-endpoint:port/sparql`.

Note

For information about finding the hostname of your Neptune DB instance, see the [Connecting to Amazon Neptune Endpoints \(p. 78\)](#) section.

QUERY Using HTTP POST

The following example uses **curl** to submit a SPARQL **QUERY** through HTTP **POST**.

```
curl -X POST --data-binary 'query=select ?s ?p ?o where {?s ?p ?o} limit 10' https://your-neptune-endpoint:port/sparql
```

The preceding example returns up to 10 of the triples (subject-predicate-object) in the graph by using the `?s ?p ?o` query with a limit of 10. To query for something else, replace it with another SPARQL query.

Note

The default MIME media type of a response is `application/sparql-results+json` for **SELECT** and **ASK** queries.

The default MIME type of a response is `application/n-quads` for **CONSTRUCT** and **DESCRIBE** queries.

For a list of the media types used by Neptune for serialization, see [RDF serialization formats used by Neptune SPARQL \(p. 350\)](#).

UPDATE Using HTTP POST

The following example uses **curl** to submit a SPARQL **UPDATE** through HTTP **POST**.

```
curl -X POST --data-binary 'update=INSERT DATA { <https://test.com/s> <https://test.com/p> <https://test.com/o> . }' https://your-neptune-endpoint:port/sparql
```

The preceding example inserts the following triple into the SPARQL default graph: `<https://test.com/s> <https://test.com/p> <https://test.com/o>`

Optional HTTP trailing headers for multi-part SPARQL responses

Note

This feature is available starting in [Neptune engine release 1.0.3.0 \(p. 743\)](#).

The HTTP response to SPARQL queries and updates is often returned in more than one part or chunk. It can be hard to diagnose a failure that occurs after a query or update begins sending these chunks, especially since the first one arrives with an HTTP status code of 200.

Unless you explicitly request trailing headers, Neptune only reports such a failure by appending an error message to the message body, which is usually corrupted.

To make detection and diagnosis of this kind of problem easier, you can include a transfer-encoding (TE) trailers header (`te: trailers`) in your request (see, for example, [the MDN page about TE request headers](#)). Doing this will cause Neptune to include two new header fields within the trailing headers of the response chunks:

- `X-Neptune-Status` – contains the response code followed by a short name. For instance, in case of success the trailing header would be: `X-Neptune-Status: 200 OK`. In the case of failure, the response code would be an [Neptune engine error code \(p. 711\)](#) such as `X-Neptune-Status: 500 TimeLimitExceededException`.
- `X-Neptune-Detail` – is empty for successful requests. In the case of errors, it contains the JSON error message. Because only ASCII characters are allowed in HTTP header values, the JSON string is URL encoded. The error message is also still appended to the response message body.

RDF media types used by SPARQL in Neptune

Resource Description Framework (RDF) data can be serialized in many different ways, most of which SPARQL can consume or output:

RDF serialization formats used by Neptune SPARQL

- **RDF/XML** – XML serialization of RDF, defined in [RDF 1.1 XML Syntax](#). Media type: `application/rdf+xml`. Typical file extension: `.rdf`.
- **N-Triples** – A line-based, plain-text format for encoding an RDF graph, defined in [RDF 1.1 N-Triples](#). Media type: `application/n-triples`, `text/turtle`, or `text/plain`. Typical file extension: `.nt`.
- **N-Quads** – A line-based, plain-text format for encoding an RDF graph, defined in [RDF 1.1 N-Quads](#). It is an extension of N-Triples. Media type: `application/n-quads`, or `text/x-nquads` when encoded with 7-bit US-ASCII. Typical file extension: `.nq`.
- **Turtle** – A textual syntax for RDF defined in [RDF 1.1 Turtle](#) that allows an RDF graph to be completely written in a compact and natural text form, with abbreviations for common usage patterns and datatypes. Turtle provides levels of compatibility with the N-Triples format as well as SPARQL's triple pattern syntax. Media type: `text/turtle`. Typical file extension: `.ttl`.
- **TriG** – A textual syntax for RDF defined in [RDF 1.1 TriG](#) that allows an RDF graph to be completely written in a compact and natural text form, with abbreviations for common usage patterns and datatypes. TriG is an extension of the Turtle format. Media type: `application/trig`. Typical file extension: `.trig`.
- **N3 (Notation3)** – An assertion and logic language defined in [Notation3 \(N3\): A readable RDF syntax](#). N3 extends the RDF data model by adding formulae (literals which are graphs themselves), variables, logical implication, and functional predicates, and provides a textual syntax alternative to RDF/XML. Media type: `text/n3`. Typical file extension: `.n3`.
- **JSON-LD** – A data serialization and messaging format defined in [JSON-LD 1.0](#). Media type: `application/ld+json`. Typical file extension: `.jsonld`.
- **TriX** – A serialization of RDF in XML, defined in [TriX: RDF Triples in XML](#). Media type: `application/trix`. Typical file extension: `.trix`.
- **SPARQL JSON Results** – A serialization of RDF using the [SPARQL 1.1 Query Results JSON Format](#). Media type: `application/sparql-results+json`. Typical file extension: `.srj`.
- **RDF4J Binary Format** – A binary format for encoding RDF data, documented in [RDF4J Binary RDF Format](#). Media type: `application/x-binary-rdf`.

SPARQL result serialization formats used by Neptune SPARQL

- **SPARQL XML Results** – An XML format for the variable binding and boolean results formats provided by the SPARQL query language, defined in [SPARQL Query Results XML Format \(Second Edition\)](#). Media type: `application/sparql-results+xml`. Typical file extension: `.srx`.

- **SPARQL CSV and TSV Results** – The use of comma-separated values and tab-separated values to express SPARQL query results from `SELECT` queries, defined in [SPARQL 1.1 Query Results CSV and TSV Formats](#). Media type: `text/csv` for comma-separated values, and `text/tab-separated-values` for tab-separated values. Typical file extensions: `.csv` for comma-separated values, and `.tsv` for tab-separated values.
- **Binary Results Table** – A binary format for encoding the output of SPARQL queries. Media type: `application/x-binary-rdf-results-table`.
- **SPARQL JSON Results** – A serialization of RDF using the [SPARQL 1.1 Query Results JSON Format](#). Media type: `application/sparql-results+json`.

Media-Types that Neptune can use to import RDF data

Media-types supported by the [Neptune bulk-loader \(p. 190\)](#)

- [N-Triples](#)
- [N-Quads](#)
- [RDF/XML](#)
- [Turtle](#)

Media-types that SPARQL UPDATE LOAD can import

- [N-Triples](#)
- [N-Quads](#)
- [RDF/XML](#)
- [Turtle](#)
- [TriG](#)
- [N3](#)
- [JSON-LD](#)

Media-Types that Neptune can use to export query results

To specify the output format for a SPARQL query response, send an "Accept: *media-type*" header with the query request. For example:

```
curl -H "Accept: application/nquads" ...
```

RDF media-types that SPARQL SELECT can output from Neptune

- [SPARQL JSON Results](#) (This is the default)
- [SPARQL XML Results](#)
- **Binary Results Table** (media type: `application/x-binary-rdf-results-table`)
- [Comma-Separated Values \(CSV\)](#)
- [Tab-Separated Values \(TSV\)](#)

RDF media-types that SPARQL ASK can output from Neptune

- [SPARQL JSON Results](#) (This is the default)
- [SPARQL XML Results](#)

- **Boolean** (media type: `text/boolean`, meaning "true" or "false")

RDF media-types that SPARQL CONSTRUCT can output from Neptune

- [N-Quads](#) (This is the default)
- [RDF/XML](#)
- [JSON-LD](#)
- [N-Triples](#)
- [Turtle](#)
- [N3](#)
- [TriX](#)
- [TriG](#)
- [SPARQL JSON Results](#)
- [RDF4J Binary RDF Format](#)

RDF media-types that SPARQL DESCRIBE can output from Neptune

- [N-Quads](#) (This is the default)
- [RDF/XML](#)
- [JSON-LD](#)
- [N-Triples](#)
- [Turtle](#)
- [N3](#)
- [TriX](#)
- [TriG](#)
- [SPARQL JSON Results](#)
- [RDF4J Binary RDF Format](#)

Using SPARQL UPDATE LOAD to import data into Neptune

The syntax of the SPARQL UPDATE LOAD command is specified in the [SPARQL 1.1 Update recommendation](#):

```
LOAD SILENT (URL of data to be loaded) INTO GRAPH (named graph into which to load the data)
```

- **SILENT** – (*Optional*) Causes the operation to return success even if there was an error during processing.

This can be useful when a single transaction contains multiple statements like "LOAD ... ; LOAD ... ; UNLOAD ... ; LOAD ... ;" and you want the transaction to complete even if some of the remote data could not be processed.

- *URL of data to be loaded* – (*Required*) Specifies a remote data file containing data to be loaded into a graph.

The remote file must have one of the following extensions:

- `.nt` for N-Triples.
- `.nq` for N-Quads.
- `.trig` for Trig.

- `.rdf` for RDF/XML.
- `.ttl` for Turtle.
- `.n3` for N3.
- `.jsonld` for JSON-LD.
- **INTO GRAPH**(*named graph into which to load the data*) – (Optional) Specifies the graph into which the data should be loaded.

Neptune associates every triple with a named graph. You can specify the default named graph using the fallback named-graph URI, `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`, like this:

```
INTO GRAPH <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>
```

Note

When you need to load a lot of data, we recommend that you use the Neptune bulk loader rather than `UPDATE LOAD`. For more information about the bulk loader, see [Using the Amazon Neptune Bulk Loader to Ingest Data \(p. 190\)](#).

You can use `SPARQL UPDATE LOAD` to load data directly from Amazon S3, or from files obtained from a self-hosted web server. The resources to be loaded must reside in the same region as the Neptune server, and the endpoint for the resources must be allowed in the VPC. For information about creating an Amazon S3 endpoint, see [Creating an Amazon S3 VPC Endpoint \(p. 204\)](#).

All `SPARQL UPDATE LOAD` URIs must start with `https://`. This includes Amazon S3 URLs.

In contrast to the Neptune bulk loader, a call to `SPARQL UPDATE LOAD` is fully transactional.

Loading files directly from Amazon S3 into Neptune using SPARQL UPDATE LOAD

Because Neptune does not allow you to pass an IAM role to Amazon S3 when using `SPARQL UPDATE LOAD`, either the Amazon S3 bucket in question must be public or you must use a [pre-signed Amazon S3 URL](#) in the `LOAD` query.

To generate a pre-signed URL for an Amazon S3 file, you can use an AWS CLI command like this:

```
aws s3 presign --expires-in (number of seconds) s3://(bucket name)/(path to file of data to load)
```

Then you can use the resulting pre-signed URL in your `LOAD` command:

```
curl https://(a Neptune endpoint URL):8182/sparql \  
  --data-urlencode 'update=load (pre-signed URL of the remote Amazon S3 file of data to be loaded) \  
  \  
  into graph (named graph)'
```

For more information, see [Authenticating Requests: Using Query Parameters](#). The [Boto3 documentation](#) shows how to use a Python script to generate a presigned URL.

Also, the content type of the files to be loaded must be set correctly.

1. Set the content type of files when you upload them into Amazon S3 by using the `-metadata` parameter, like this:

```
aws s3 cp test.nt s3://bucket-name/my-plain-text-input/test.nt --metadata Content-Type=text/plain
```

```
aws s3 cp test.rdf s3://bucket-name/my-rdf-input/test.rdf --metadata Content-Type=application/rdf+xml
```

2. Confirm that the media-type information is actually present. Run:

```
curl -v bucket-name/folder-name
```

The output of this command should show the media-type information that you set when uploading the files.

3. Then you can use the SPARQL UPDATE LOAD command to import these files into Neptune:

```
curl https://your-neptune-endpoint:port/sparql \  
-d "update=LOAD <https://s3.amazonaws.com/bucket-name/my-rdf-input/test.rdf>"
```

The steps above work only for a public Amazon S3 bucket, or for a bucket that you access using a [pre-signed Amazon S3 URL](#) in the LOAD query.

You can also set up a web proxy server to load from a private Amazon S3 bucket, as shown below:

Using a web server to load files into Neptune with SPARQL UPDATE LOAD

1. Install a web server on a machine running within the VPC that is hosting Neptune and the files to be loaded. For example, using Amazon Linux, you might install Apache as follows:

```
sudo yum install httpd mod_ssl  
sudo /usr/sbin/apachectl start
```

2. Define the MIME type(s) of the RDF file-content that you are going to load. SPARQL uses the Content-type header sent by the web server to determine the input format of the content, so you must define the relevant MIME types for the web Server.

For example, suppose you use the following file extensions to identify file formats:

- .nt for NTriples.
- .nq for NQuads.
- .trig for Trig.
- .rdf for RDF/XML.
- .ttl for Turtle.
- .n3 for N3.
- .jsonld for JSON-LD.

If you are using Apache 2 as the web server, you would edit the file `/etc/mime.types` and add the following types:

```
text/plain nt  
application/n-quads nq  
application/trig trig  
application/rdf+xml rdf  
application/x-turtle ttl  
text/rdf+n3 n3  
application/ld+json jsonld
```

3. Confirm that the MIME-type mapping works. Once you have your web server up and running and hosting RDF files in the format(s) of your choice, you can test the configuration by sending a request to the web server from your local host.

For instance, you might send a request such as this:

```
curl -v http://localhost:80/test.rdf
```

Then, in the detailed output from `curl`, you should see a line such as:

```
Content-Type: application/rdf+xml
```

This shows that the content-type mapping was defined successfully.

4. You are now ready to load data using the SPARQL UPDATE command:

```
curl https://your-neptune-endpoint:port/sparql \
  -d "update=LOAD <http://web_server_private_ip:80/test.rdf>"
```

Note

Using SPARQL UPDATE LOAD can trigger a timeout on the web server when the source file being loaded is large. Neptune processes the file data as it is streamed in, and for a big file that can take longer than the timeout configured on the server. This in turn may cause the server to close the connection, which can result in the following error message when Neptune encounters an unexpected EOF in the stream:

```
{
  "detailedMessage": "Invalid syntax in the specified file",
  "code": "InvalidParameterException"
}
```

If you receive this message and don't believe your source file contains invalid syntax, try increasing the timeout settings on the web server. You can also diagnose the problem by enabling debug logs on the server and looking for timeouts.

Using SPARQL UPDATE UNLOAD to delete data from Neptune

Neptune also provides a custom SPARQL operation, UNLOAD, for removing data that is specified in a remote source. UNLOAD can be regarded as a counterpart to the LOAD operation. Its syntax is:

Note

This feature is available starting in [Neptune engine release 1.0.4.1 \(p. 734\)](#).

```
UNLOAD SILENT (URL of the remote data to be unloaded) FROM GRAPH (named graph from which to remove the data)
```

- **SILENT** – (*Optional*) Causes the operation to return success even if there was an error when processing the data.

This can be useful when a single transaction contains multiple statements like "LOAD ...; LOAD ...; UNLOAD ...; LOAD ...;" and you want the transaction to complete even if some of the remote data could not be processed.

- *URL of the remote data to be unloaded* – (*Required*) Specifies a remote data file containing data to be unloaded from a graph.

The remote file must have one of the following extensions (these are the same formats that UPDATE-LOAD supports):

- `.nt` for NTriples.

- `.nq` for NQuads.
- `.trig` for Trig.
- `.rdf` for RDF/XML.
- `.ttl` for Turtle.
- `.n3` for N3.
- `.jsonld` for JSON-LD.

All the data that this file contains will be removed from your DB cluster by the UNLOAD operation.

Any Amazon S3 authentication must be included in the URL for the data to unload. You can pre-sign an Amazon S3 file and then use the resulting URL to access it securely. For example:

```
aws s3 presign --expires-in (number of seconds) s3://(bucket name)/(path to file of data to unload)
```

Then:

```
curl https://(a Neptune endpoint URL):8182/sparql \
  --data-urlencode 'update=unload (pre-signed URL of the remote Amazon S3 data to be unloaded) \
  from graph (named graph)'
```

For more information, see [Authenticating Requests: Using Query Parameters](#).

- **FROM GRAPH** *(named graph from which to remove the data)* – *(Optional)* Specifies the named graph from which the remote data should be unloaded.

Neptune associates every triple with a named graph. You can specify the default named graph using the fallback named-graph URI, `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`, like this:

```
FROM GRAPH <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>
```

In the same way that LOAD corresponds to INSERT DATA { *(inline data)* }, UNLOAD corresponds to DELETE DATA { *(inline data)* }. Like DELETE DATA, UNLOAD does not work on data that contains blank nodes.

For example, if a local web server serves a file named `data.nt` that contains the following 2 triples:

```
<http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/resource#b> .
<http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/resource#c> .
```

The following UNLOAD command would delete those two triples from the named graph, `<http://example.org/graph1>`:

```
UNLOAD <http://localhost:80/data.nt> FROM GRAPH <http://example.org/graph1>
```

This would have the same effect as using the following DELETE DATA command:

```
DELETE DATA {
  GRAPH <http://example.org/graph1> {
    <http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/resource#b> .
```

```
<http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/resource#c> .  
}  
}
```

Exceptions thrown by the UNLOAD command

- **InvalidParameterException** – There were blank nodes in the data. *HTTP status: 400 Bad Request.*
Message: Blank nodes are not allowed for UNLOAD
- **InvalidParameterException** – There was broken syntax in the data. *HTTP status: 400 Bad Request.*
Message: Invalid syntax in the specified file.
- **UnloadUrlAccessDeniedException** – Access was denied. *HTTP status: 400 Bad Request.*
Message: Update failure: Endpoint (*Neptune endpoint*) reported access denied error. Please verify access.
- **BadRequestException** – The remote data cannot be retrieved. *HTTP status: 400 Bad Request.*
Message: (depends on the HTTP response).

SPARQL query hints

You can use query hints to specify optimization and evaluation strategies for a particular SPARQL query in Amazon Neptune.

Query hints are expressed using additional triple patterns that are embedded in the SPARQL query with the following parts:

```
scope hint value
```

- *scope* – Determines the part of the query that the query hint applies to, such as a certain group in the query or the full query.
- *hint* – Identifies the type of the hint to apply.
- *value* – Determines the behavior of the system aspect under consideration.

The query hints and scopes are exposed as predefined terms in the Amazon Neptune namespace `http://aws.amazon.com/neptune/vocab/v01/QueryHints#`. The examples in this section include the namespace as a *hint* prefix that is defined and included in the query:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
```

For example, the following shows how to include a `joinOrder` hint in a `SELECT` query:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>  
SELECT ... {  
  hint:Query hint:joinOrder "Ordered" .
```

```
...  
}
```

The preceding query instructs the Neptune engine to evaluate joins in the *given* order and disables any automatic reordering.

Consider the following when using query hints:

- You can combine different query hints in a single query. For example, you can use the `bottomUp` query hint to annotate a subquery for bottom-up evaluation and a `joinOrder` query hint to fix the join order inside the subquery.
- You can use the same query hint multiple times, in different non-overlapping scopes.
- Query hints are hints. Although the query engine generally aims to consider given query hints, it might also ignore them.
- Query hints are semantics preserving. Adding a query hint does not change the output of the query (except for the potential result order when no ordering guarantees are given—that is, when the result order is not explicitly enforced by using `ORDER BY`).

The following sections provide more information about the available query hints and their usage in Neptune.

Topics

- [Scope of SPARQL query hints in Neptune \(p. 358\)](#)
- [The `joinOrder` SPARQL query hint \(p. 359\)](#)
- [The `evaluationStrategy` SPARQL query hint \(p. 361\)](#)
- [The `queryTimeout` SPARQL query hint \(p. 363\)](#)
- [The `rangeSafe` SPARQL query hint \(p. 364\)](#)
- [The `queryId` SPARQL Query Hint \(p. 365\)](#)
- [The `useDFE` SPARQL query hint \(p. 365\)](#)

Scope of SPARQL query hints in Neptune

The following table shows the available scopes, associated hints, and descriptions for SPARQL query hints in Amazon Neptune. The `hint` prefix in these entries represents the Neptune namespace for hints:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
```

Scope	Supported Hint	Description
<code>hint:Query</code>	joinOrder (p. 359)	The query hint applies to the whole query.
<code>hint:Query</code>	queryTimeout (p. 363)	The time-out value applies to the entire query.
<code>hint:Query</code>	rangeSafe (p. 364)	Type promotion is disabled for the entire query.
<code>hint:Query</code>	queryId (p. 365)	The query ID value applies to the entire query.
<code>hint:Query</code>	useDFE (p. 365)	Use of the DFE is disabled for the entire query.

Scope	Supported Hint	Description
<code>hint:Group</code>	joinOrder (p. 359)	The query hint applies to the top-level elements in the specified group, but not to nested elements (such as subqueries) or parent elements.
<code>hint:SubQuery</code>	evaluationStrategy (p. 361)	The hint is specified and applied to a nested SELECT subquery. The subquery is evaluated independently, without considering solutions computed before the subquery.

The joinOrder SPARQL query hint

When you submit a SPARQL query, the Amazon Neptune query engine investigates the structure of the query. It reorders parts of the query and tries to minimize the amount of work required for evaluation and query response time.

For example, a sequence of connected triple patterns is typically not evaluated in the given order. It is reordered using heuristics and statistics such as the selectivity of the individual patterns and how they are connected through shared variables. Additionally, if your query contains more complex patterns such as subqueries, FILTERs, or complex OPTIONAL or MINUS blocks, the Neptune query engine reorders them where possible, aiming for an efficient evaluation order.

For more complex queries, the order in which Neptune chooses to evaluate the query might not always be optimal. For instance, Neptune might miss instance data-specific characteristics (such as hitting power nodes in the graph) that emerge during query evaluation.

If you know the exact characteristics of the data and want to manually dictate the order of the query execution, use the Neptune `joinOrder` query hint to specify that the query be evaluated in the given order.

joinOrder SPARQL hint syntax

The `joinOrder` query hint is specified as a triple pattern included in a SPARQL query.

For clarity, the following syntax uses a `hint` prefix defined and included in the query to specify the Neptune query-hint namespace:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
scope hint:joinOrder "Ordered" .
```

Available Scopes

- `hint:Query`
- `hint:Group`

For more information about query hint scopes, see [Scope of SPARQL query hints in Neptune \(p. 358\)](#).

joinOrder SPARQL hint example

This section shows a query written with and without the `joinOrder` query hint and related optimizations.

For this example, assume that the dataset contains the following:

- A single person named `John` that `:likes` 1,000 persons, including `Jane`.
- A single person named `Jane` that `:likes` 10 persons, including `John`.

No Query Hint

The following SPARQL query extracts all the pairs of people named `John` and `Jane` who both like each other from a set of social networking data:

```
PREFIX : <https://example.com/>
SELECT ?john ?jane {
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

The Neptune query engine might evaluate the statements in a different order than written. For example, it might choose to evaluate in the following order:

1. Find all persons named `John`.
2. Find all persons connected to `John` by a `:likes` edge.
3. Filter this set by persons named `Jane`.
4. Filter this set by those connected to `John` by a `:likes` edge.

According to the dataset, evaluating in this order results in 1,000 entities being extracted in the second step. The third step narrows this down to the single node, `Jane`. The final step then determines that `Jane` also `:likes` the `John` node.

Query Hint

It would be favorable to start with the `Jane` node because she has only 10 outgoing `:likes` edges. This reduces the amount of work during the evaluation of the query by avoiding the extraction of the 1,000 entities during the second step.

The following example uses the **joinOrder** query hint to ensure that the `Jane` node and its outgoing edges are processed first by disabling all automatic join reordering for the query:

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?john ?jane {
  hint:Query hint:joinOrder "Ordered" .
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

An applicable real-world scenario might be a social network application in which persons in the network are classified as either influencers with many connections or normal users with few connections. In such a scenario, you could ensure that the normal user (`Jane`) is processed before the influencer (`John`) in a query like the preceding example.

Query Hint and Reorder

You can take this example one step further. If you know that the `:name` attribute is unique to a single node, you could speed up the query by reordering and using the `joinOrder` query hint. This step ensures that the unique nodes are extracted first.

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?john ?jane {
  hint:Query hint:joinOrder "Ordered" .
  ?person1 :name "Jane" .
  ?person2 :name "John" .
  ?person1 :likes ?person2 .
  ?person2 :likes ?person1 .
}
```

In this case, you can reduce the query to the following single actions in each step:

1. Find the single person node with `:name Jane`.
2. Find the single person node with `:name John`.
3. Check that the first node is connected to the second with a `:likes` edge.
4. Check that the second node is connected to the first with a `:likes` edge.

Important

If you choose the wrong order, the `joinOrder` query hint can lead to significant performance drops. For example, the preceding example would be inefficient if the `:name` attributes were not unique. If all 100 nodes were named `Jane` and all 1,000 nodes were named `John`, then the query would end up checking 1,000 * 100 (100,000) pairs for `:likes` edges.

The `evaluationStrategy` SPARQL query hint

The `evaluationStrategy` query hint tells the Amazon Neptune query engine that the fragment of the query annotated should be evaluated from the bottom up, as an independent unit. This means that no solutions from previous evaluation steps are used to compute the query fragment. The query fragment is evaluated as a standalone unit, and its produced solutions are joined with the remainder of the query after it is computed.

Using the `evaluationStrategy` query hint implies a blocking (non-pipelined) query plan, meaning that the solutions of the fragment annotated with the query hint are materialized and buffered in main memory. Using this query hint might significantly increase the amount of main memory needed to evaluate the query, especially if the annotated query fragment computes a large number of results.

`evaluationStrategy` SPARQL hint syntax

The `evaluationStrategy` query hint is specified as a triple pattern included in a SPARQL query.

For clarity, the following syntax uses a `hint` prefix defined and included in the query to specify the Neptune query-hint namespace:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
hint:SubQuery hint:evaluationStrategy "BottomUp" .
```

Available Scopes

- `hint:SubQuery`

Note

This query hint is supported only in nested subqueries.

For more information about query hint scopes, see [Scope of SPARQL query hints in Neptune \(p. 358\)](#).

evaluationStrategy SPARQL hint example

This section shows a query written with and without the `evaluationStrategy` query hint and related optimizations.

For this example, assume that the dataset has the following characteristics:

- It contains 1,000 edges labeled `:connectedTo`.
- Each `component` node is connected to an average of 100 other `component` nodes.
- The typical number of four-hop cyclical connections between nodes is around 100.

No Query Hint

The following SPARQL query extracts all `component` nodes that are cyclically connected to each other via four hops:

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  ?component1 :connectedTo ?component2 .
  ?component2 :connectedTo ?component3 .
  ?component3 :connectedTo ?component4 .
  ?component4 :connectedTo ?component1 .
}
```

The approach of the Neptune query engine is to evaluate this query using the following steps:

- Extract all 1,000 `connectedTo` edges in the graph.
- Expand by 100x (the number of outgoing `connectedTo` edges from `component2`).

Intermediate results: 100,000 nodes.

- Expand by 100x (the number of outgoing `connectedTo` edges from `component3`).

Intermediate results: 10,000,000 nodes.

- Scan the 10,000,000 nodes for the cycle close.

This results in a streaming query plan, which has a constant amount of main memory.

Query Hint and Subqueries

You might want to trade off main memory space for accelerated computation. By rewriting the query using an `evaluationStrategy` query hint, you can force the engine to compute a join between two smaller, materialized subsets.

```
PREFIX : <https://example.com/>
      PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  {
    SELECT * WHERE {
      hint:SubQuery hint:evaluationStrategy "BottomUp" .
      ?component1 :connectedTo ?component2 .
      ?component2 :connectedTo ?component3 .
    }
  }
}
```

```
{
  SELECT * WHERE {
    hint:SubQuery hint:evaluationStrategy "BottomUp" .
    ?component3 :connectedTo ?component4 .
    ?component4 :connectedTo ?component1 .
  }
}
```

Instead of evaluating the triple patterns in sequence while iteratively using results from the previous triple pattern as input for the upcoming patterns, the `evaluationStrategy` hint causes the two subqueries to be evaluated independently. Both subqueries produce 100,000 nodes for intermediate results, which are then joined together to form the final output.

In particular, when you run Neptune on the larger instance types, temporarily storing these two 100,000 subsets in main memory increases memory usage in return for significantly speeding up evaluation.

The `queryTimeout` SPARQL query hint

The `queryTimeout` query hint specifies a timeout that is shorter than the `neptune_query_timeout` value set in the DB parameters group.

If the query terminates as a result of this hint, a `TimeLimitExceededException` is thrown, with an `Operation terminated (deadline exceeded)` message.

`queryTimeout` SPARQL hint syntax

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ... WHERE {
  hint:Query hint:queryTimeout 10 .
  # OR
  hint:Query hint:queryTimeout "10" .
  # OR
  hint:Query hint:queryTimeout "10"^^xsd:integer .
  ...
}
```

The time-out value is expressed in milliseconds.

The time-out value must be smaller than the `neptune_query_timeout` value set in the DB parameters group. Otherwise, a `MalformedQueryException` exception is thrown with a `Malformed query: Query hint 'queryTimeout' must be less than neptune_query_timeout DB Parameter Group` message.

The `queryTimeout` query hint should be specified in the `WHERE` clause of the main query, or in the `WHERE` clause of one of the subqueries as shown in the example below.

It must be set only once across all the queries/subqueries and SPARQL Updates sections (such as `INSERT` and `DELETE`). Otherwise, a `MalformedQueryException` exception is thrown with a `Malformed query: Query hint 'queryTimeout' must be set only once` message.

Available Scopes

The `queryTimeout` hint can be applied both to SPARQL queries and updates.

- In a SPARQL query, it can appear in the `WHERE` clause of the main query or a subquery.
- In a SPARQL update, it can be set in the `INSERT`, `DELETE`, or `WHERE` clause. If there are multiple update clauses, it can only be set in one of them.

For more information about query hint scopes, see [Scope of SPARQL query hints in Neptune \(p. 358\)](#).

queryTimeout SPARQL hint example

Here is an example of using `hint:queryTimeout` in the main `WHERE` clause of an `UPDATE` query:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
INSERT {
  ?s ?p ?o
} WHERE {
  hint:Query hint:queryTimeout 100 .
  ?s ?p ?o .
}
```

Here, the `hint:queryTimeout` is in the `WHERE` clause of a subquery:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  ?s ?p ?o .
  {
    SELECT ?s WHERE {
      hint:Query hint:queryTimeout 100 .
      ?s ?p1 ?o1 .
    }
  }
}
```

The rangeSafe SPARQL query hint

Use this query hint to turn off type promotion for a SPARQL query.

When you submit a SPARQL query that includes a `FILTER` over a numerical value or range, the Neptune query engine must normally use type promotion when it executes the query. This means that it has to examine values of every type that could hold the value you are filtering on.

For example, if you are filtering for values equal to 55, the engine must look for integers equal to 55, long integers equal to 55L, floats equal to 55.0, and so forth. Each type promotion requires an additional lookup on storage, which can cause an apparently simple query to take an unexpectedly long time to complete.

Often type promotion is unnecessary because you know in advance that you only need to find values of one specific type. When this is the case, you can speed up your queries dramatically by using the `rangeSafe` query hint to turn off type promotion.

rangeSafe SPARQL hint syntax

The `rangeSafe` query hint takes a value of `true` to turn off type promotion. It also accepts a value of `false` (the default).

Example. The following example shows how to turn off type promotion when filtering for an integer value of `o` greater than 1:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  ?s ?p ?o .
  hint:Prior hint:rangeSafe 'true' .
  FILTER (?o > '1'^^<http://www.w3.org/2001/XMLSchema#int>)
```

The queryId SPARQL Query Hint

Use this query hint to assign your own queryId value to a SPARQL query.

Example:

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * WHERE {
  hint:Query hint:queryId "4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
  {?s ?p ?o}}
```

The value you assign must be unique across all queries in the Neptune DB.

The useDFE SPARQL query hint

Use this query hint to disable use of the DFE for executing the query. Because the default value of this hint is `true`, there is no need to include the hint unless you don't want the query to be executed by the DFE.

Note

If you use this query hint when the DFE is not enabled in Lab Mode, the hint is treated as a regular triple pattern.

Example:

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>

SELECT ?john ?jane
{
  hint:Query hint:useDFE false .
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

SPARQL query status API

To get the status of SPARQL queries, use HTTP `GET` or `POST` to make a request to the `https://your-neptune-endpoint:port/sparql/status` endpoint.

SPARQL query status request parameters

queryId (optional)

The ID of a running SPARQL query. Only displays the status of the specified query.

SPARQL query status response syntax

```
{
  "acceptedQueryCount": integer,
  "runningQueryCount": integer,
  "queries": [
    {
      "queryId": "guid",
      "queryEvalStats":
```

```
{
  "subqueries": integer,
  "elapsed": integer,
  "cancelled": boolean
},
"queryString": "string"
]
}
```

SPARQL query status response values

acceptedQueryCount

The number of queries accepted since the last restart of the Neptune engine.

runningQueryCount

The number of currently running SPARQL queries.

queries

A list of the current SPARQL queries.

queryId

A GUID id for the query. Neptune automatically assigns this ID value to each query, or you can also assign your own ID (see [Inject a Custom ID Into a Neptune Gremlin or SPARQL Query \(p. 83\)](#)).

queryEvalStats

Statistics for this query.

subqueries

Number of subqueries in this query.

elapsed

The number of milliseconds the query has been running so far.

cancelled

True indicates that the query was cancelled.

queryString

The submitted query.

SPARQL query status example

The following is an example of the status command using `curl` and HTTP `GET`.

```
curl https://your-neptune-endpoint:port/sparql/status
```

This output shows a single running query.

```
{
  "acceptedQueryCount": 9,
```

```
"runningQueryCount":1,
"queries": [
  {
    "queryId":"fb34cd3e-f37c-4d12-9cf2-03bb741bf54f",
    "queryEvalStats":
      {
        "subqueries": 0,
        "elapsed": 29256,
        "cancelled": false
      },
    "queryString": "SELECT ?s ?p ?o WHERE {?s ?p ?o}"
  }
]
```

SPARQL query cancellation

To get the status of SPARQL queries, use HTTP `GET` or `POST` to make a request to the `https://your-neptune-endpoint:port/sparql/status` endpoint.

SPARQL query cancellation request parameters

cancelQuery

(Required) Tells the status command to cancel a query. This parameter does not take a value.

queryId

(Required) The ID of the running SPARQL query to cancel.

silent

(Optional) If `silent=true` then the running query is cancelled and the HTTP response code is 200. If `silent` is not present or `silent=false`, the query is cancelled with an HTTP 500 status code.

SPARQL query cancellation examples

Example 1: Cancellation with `silent=false`

The following is an example of the status command using `curl` to cancel a query with the `silent` parameter set to `false`:

```
curl https://your-neptune-endpoint:port/sparql/status \
-d "cancelQuery" \
-d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47" \
-d "silent=false"
```

Unless the query has already started streaming results, the cancelled query would then return an HTTP 500 code with a response like this:

```
{
  "code": "CancelledByUserException",
  "requestId": "4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47",
  "detailedMessage": "Operation terminated (cancelled by user)"
}
```

If the query already returned an HTTP 200 code (OK) and has started streaming results before being cancelled, the timeout exception information is sent to the regular output stream.

Example 2: Cancellation with `silent=true`

The following is an example of the same status command as above except with the `silent` parameter now set to `true`:

```
curl https://your-neptune-endpoint:port/sparql/status \  
-d "cancelQuery" \  
-d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47" \  
-d "silent=true"
```

This command would return the same response as when `silent=false`, but the cancelled query would now return an HTTP 200 code with a response like this:

```
{  
  "head" : {  
    "vars" : [ "s", "p", "o" ]  
  },  
  "results" : {  
    "bindings" : [ ]  
  }  
}
```

Using the SPARQL 1.1 Graph Store HTTP Protocol (GSP) in Amazon Neptune

In the [SPARQL 1.1 Graph Store HTTP Protocol](#) recommendation, the W3C defined an HTTP protocol for managing RDF graphs. It defines operations for removing, creating, and replacing RDF graph content as well as for adding RDF statements to existing content.

The graph-store protocol (GSP) provides a convenient way to manipulate your entire graph without having to write complex SPARQL queries.

As of [Release: 1.0.5.0 \(2021-07-27\) \(p. 725\)](#), Neptune fully supports this protocol.

The endpoint for the graph-store protocol (GSP) is:

```
https://your-neptune-cluster:port/sparql/gsp/
```

To access the default graph with GSP, use:

```
https://your-neptune-cluster:port/sparql/gsp/?default
```

To access a named graph with GSP, use:

```
https://your-neptune-cluster:port/sparql/gsp/?graph=named-graph-URI
```

Special details of the Neptune GSP implementation

Neptune fully implements the [W3C recommendation](#) that defines GSP. However, there are a few situations that the specification doesn't cover.

One of these is the case where a `PUT` or `POST` request specifies one or more named graphs in the request body that differ from the graph specified by the request URL. This can only happen when the request

body RDF format supports named graphs, as, for example, using `Content-Type: application/n-quads` or `Content-Type: application/trig`.

In this situation, Neptune adds or updates all the named graphs present in the body, as well as the named graph specified in the URL.

For example, suppose that starting with an empty database, you send a `PUT` request to upsert votes into three graphs. One, named `urn:votes`, contains all votes from all election years. Two others, named `urn:votes:2005` and `urn:votes:2019`, contain votes from specific election years. The request and its payload look like this:

```
PUT "http://your-Neptune-cluster:port/sparql/gsp/?graph=urn:votes"
Host: example.com
Content-Type: application/n-quads

PAYLOAD:

<urn:JohnDoe> <urn:votedFor> <urn:Labour> <urn:votes:2005>
<urn:JohnDoe> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
<urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes:2005>
<urn:JaneSmith> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
```

After the request is executed, the data in the database looks like this:

```
<urn:JohnDoe> <urn:votedFor> <urn:Labour> <urn:votes:2005>
<urn:JohnDoe> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
<urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes:2005>
<urn:JaneSmith> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
<urn:JohnDoe> <urn:votedFor> <urn:Labour> <urn:votes>
<urn:JohnDoe> <urn:votedFor> <urn:Conservative> <urn:votes>
<urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes>
<urn:JaneSmith> <urn:votedFor> <urn:Conservative> <urn:votes>
```

Another ambiguous situation is where more than one graph is specified in the request URL itself, using any of `PUT`, `POST`, `GET` or `DELETE`. For example:

```
POST "http://your-Neptune-cluster:port/sparql/gsp/?
graph=urn:votes:2005&graph=urn:votes:2019"
```

Or:

```
GET "http://your-Neptune-cluster:port/sparql/gsp/?default&graph=urn:votes:2019"
```

In this situation, Neptune returns an HTTP 400 with a message indicating that only one graph can be specified in the request URL.

Analyzing Neptune query execution using SPARQL explain

Amazon Neptune has added a SPARQL feature named *explain*. This feature is a self-service tool for understanding the execution approach taken by the Neptune engine. You invoke it by adding an `explain` parameter to an HTTP call that submits a SPARQL query.

The `explain` feature provides information about the logical structure of query execution plans. You can use this information to identify potential evaluation and execution bottlenecks. You can then use [query hints](#) (p. 357) to improve your query execution plans.

Topics

- [How the SPARQL query engine works in Neptune \(p. 370\)](#)
- [How to use SPARQL explain to analyze Neptune query execution \(p. 371\)](#)
- [Examples of invoking SPARQL explain in Neptune \(p. 371\)](#)
- [Neptune SPARQL explain operators \(p. 384\)](#)
- [Limitations of SPARQL explain in Neptune \(p. 390\)](#)

How the SPARQL query engine works in Neptune

To use the information that the SPARQL `explain` feature provides, you need to understand some details about how the Amazon Neptune SPARQL query engine works.

The engine translates every SPARQL query into a pipeline of operators. Starting from the first operator, intermediate solutions known as *binding lists* flow through this operator pipeline. You can think of a binding list as a table in which the table headers are a subset of the variables used in the query. Each row in the table represents a result, up to the point of evaluation.

Let's assume that two namespace prefixes have been defined for our data:

```
@prefix ex: <http://example.com> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

The following would be an example of a simple binding list in this context:

```
?person      | ?firstName
-----
ex:JaneDoe   | "Jane"
ex:JohnDoe   | "John"
ex:RichardRoe | "Richard"
```

For each of three people, the list binds the `?person` variable to an identifier of the person, and the `?firstName` variable to the person's first name.

In the general case, variables can remain unbound, if, for example, there is an `OPTIONAL` selection of a variable in a query for which no value is present in the data.

The `PipelineJoin` operator is an example of a Neptune query engine operator present in the `explain` output. It takes as input an incoming binding set from the previous operator and joins it against a triple pattern, say `(?person, foaf:lastName, ?lastName)`. This operation uses the bindings for the `?person` variable in its input stream, substitutes them into the triple pattern, and looks up triples from the database.

When executed in the context of the incoming bindings from the previous table, `PipelineJoin` would evaluate three lookups, namely the following:

```
(ex:JaneDoe, foaf:lastName, ?lastName)
(ex:JohnDoe, foaf:lastName, ?lastName)
(ex:RichardRoe, foaf:lastName, ?lastName)
```

This approach is called *as-bound* evaluation. The solutions from this evaluation process are joined back against the incoming solutions, padding the detected `?lastName` in the incoming solutions. Assuming that you find a last name for all three persons, the operator would produce an outgoing binding list that would look something like this:

?person	?firstName	?lastName
ex:JaneDoe	"Jane"	"Doe"
ex:JohnDoe	"John"	"Doe"
ex:RichardRoe	"Richard"	"Roe"

This outgoing binding list then serves as input for the next operator in the pipeline. At the end, the output of the last operator in the pipeline defines the query result.

Operator pipelines are often linear, in the sense that every operator emits solutions for a single connected operator. However, in some cases, they can have more complex structures. For example, a `UNION` operator in a SPARQL query is mapped to a `Copy` operation. This operation duplicates the bindings and forwards the copies into two subplans, one for the left side and the other for the right side of the `UNION`.

For more information about operators, see [Neptune SPARQL explain operators \(p. 384\)](#).

How to use SPARQL explain to analyze Neptune query execution

The SPARQL `explain` feature is a self-service tool in Amazon Neptune that helps you understand the execution approach taken by the Neptune engine. To invoke `explain`, you pass a parameter to an HTTP or HTTPS request in the form `explain=mode`.

The mode value can be one of `static`, `dynamic`, or `details`:

- In *static* mode, `explain` prints only the static structure of the query plan.
- In *dynamic* mode, `explain` also includes dynamic aspects of the query plan. These aspects might include the number of intermediate bindings flowing through the operators, the ratio of incoming bindings to outgoing bindings, and the total time taken by operators.
- In *details* mode, `explain` prints the information shown in *dynamic* mode plus additional details such as the actual SPARQL query string and the estimated range count for the pattern underlying a join operator.

Neptune supports using `explain` with all three SPARQL query access protocols listed in the [W3C SPARQL 1.1 Protocol](#) specification, namely:

1. HTTP GET
2. HTTP POST using URL-encoded parameters
3. HTTP POST using text parameters

For information about the SPARQL query engine, see [How the SPARQL query engine works in Neptune \(p. 370\)](#).

For information about the kind of output produced by invoking SPARQL `explain`, see [Examples of invoking SPARQL explain in Neptune \(p. 371\)](#).

Examples of invoking SPARQL explain in Neptune

The examples in this section show the various kinds of output you can produce by invoking the SPARQL `explain` feature to analyze query execution in Amazon Neptune.

Topics

- [Understanding Explain Output \(p. 372\)](#)

- [Example of details mode output \(p. 374\)](#)
- [Example of static mode output \(p. 375\)](#)
- [Different ways of encoding parameters \(p. 376\)](#)
- [Other output types besides text/plain \(p. 376\)](#)
- [Example of SPARQL explain output when the DFE is enabled \(p. 378\)](#)

Understanding Explain Output

In this example, Jane Doe knows two people, namely John Doe and Richard Roe:

```
@prefix ex: <http://example.com> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

ex:JaneDoe foaf:knows ex:JohnDoe .
ex:JohnDoe foaf:firstName "John" .
ex:JohnDoe foaf:lastName "Doe" .
ex:JaneDoe foaf:knows ex:RichardRoe .
ex:RichardRoe foaf:firstName "Richard" .
ex:RichardRoe foaf:lastName "Roe" .
.
```

To determine the first names of all the people whom Jane Doe knows, you can write the following query:

```
curl http(s)://your_server:your_port/sparql \
  -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
    SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person foaf:firstName ?
firstName }" \
  -H "Accept: text/csv"
```

This simple query returns the following:

```
firstName
John
Richard
```

Next, change the curl command to invoke explain by adding `-d "explain=dynamic"` and using the default output type instead of `text/csv`:

```
curl http(s)://your_server:your_port/sparql \
  -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
    SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person foaf:firstName ?
firstName }" \
  -d "explain=dynamic"
```

The query now returns output in pretty-printed ASCII format (HTTP content type `text/plain`), which is the default output type:

```
#####
# ID # Out #1 # Out #2 # Name # Arguments
# Mode # Units In # Units Out # Ratio # Time (ms) #
#####
# 0 # 1 # - # SolutionInjection # solutions=[{}]
# - # 0 # 1 # 0.00 # 0 #
#####
```

```
# 1 # 2      # -      # PipelineJoin      # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person)      # -      # 1          # 2          # 2.00 # 1          #
#          #          #          #          #          #          #
#          #          #          #          #          #          #
#          #          #          #          #          #          #
#####
# 2 # 3      # -      # PipelineJoin      # pattern=distinct(?person, foaf:firstName, ?
firstName) # -      # 2          # 2          # 1.00 # 1          #
#          #          #          #          #          #          #
#          #          #          #          #          #          #
#          #          #          #          #          #          #
#####
# 3 # 4      # -      # Projection      # vars=[?firstName]
# retain # 2          # 2          # 1.00 # 0          #
#####
# 4 # -      # -      # TermResolution  # vars=[?firstName]
# id2value # 2          # 2          # 1.00 # 1          #
#####
```

For details about the operations in the Name column and their arguments, see [explain operators \(p. 384\)](#).

The following describes the output row by row:

1. The first step in the main query always uses the `SolutionInjection` operator to inject a solution. The solution is then expanded to the final result through the evaluation process.

In this case, it injects the so-called universal solution `{ }`. In the presence of `VALUES` clauses or a `BIND`, this step might also inject more complex variable bindings to start out with.

The `Units Out` column indicates that this single solution flows out of the operator. The `Out #1` column specifies the operator into which this operator feeds the result. In this example, all operators are connected to the operator that follows in the table.

2. The second step is a `PipelineJoin`. It receives as input the single universal (fully unconstrained) solution produced by the previous operator (`Units In := 1`). It joins it against the tuple pattern defined by its `pattern` argument. This corresponds to a simple lookup for the pattern. In this case, the triple pattern is defined as the following:

```
distinct( ex:JaneDoe, foaf:knows, ?person )
```

The `joinType := join` argument indicates that this is a normal join (other types include optional joins, existence check joins, and so on).

The `distinct := true` argument says that you extract only distinct matches from the database (no duplicates), and you bind the distinct matches to the variable `joinProjectionVars := ?person`, deduplicated.

The fact that the `Units Out` column value is 2 indicates that there are two solutions flowing out. Specifically, these are the bindings for the `?person` variable, reflecting the two people that the data shows that Jane Doe knows:

```
?person
-----
ex:JohnDoe
ex:RichardRoe
```

3. The two solutions from stage 2 flow as input (`Units In := 2`) into the second `PipelineJoin`. This operator joins the two previous solutions with the following triple pattern:

```
distinct(?person, foaf:firstName, ?firstName)
```

The `?person` variable is known to be bound either to `ex:JohnDoe` or to `ex:RichardRoe` by the operator's incoming solution. Given that, the `PipelineJoin` extracts the first names, John and Richard. The outgoing two solutions (Units Out := 2) are then as follows:

```
?person      | ?firstName
-----
ex:JohnDoe   | John
ex:RichardRoe | Richard
```

- The next projection operator takes as input the two solutions from stage 3 (Units In := 2) and projects onto the `?firstName` variable. This eliminates all other variable bindings in the mappings and passes on the two bindings (Units Out := 2):

```
?firstName
-----
John
Richard
```

- To improve performance, Neptune operates where possible on internal identifiers that it assigns to terms such as URLs and string literals, rather than on the strings themselves. The final operator, `TermResolution`, performs a mapping from these internal identifiers back to the corresponding term strings.

In regular (non-explain) query evaluation, the result computed by the last operator is then serialized into the requested serialization format and streamed to the client.

Example of details mode output

Note

SPARQL explain details mode is available starting in [Neptune engine release 1.0.2.1 \(p. 755\)](#).

Suppose that you run the same query as the previous in *details* mode instead of *dynamic* mode:

```
curl http(s)://your_server:your_port/sparql \
-d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
  SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person foaf:firstName ?
firstName }" \
-d "explain=details"
```

As this example shows, the output is the same with some additional details such as the query string at the top of the output, and the `patternEstimate` count for the `PipelineJoin` operator:

```
Query:
PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://www.example.com/>
SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person foaf:firstName ?
firstName }
```

```
#####
# ID # Out #1 # Out #2 # Name # Arguments
# Mode # Units In # Units Out # Ratio # Time (ms) #
#####
# 0 # 1 # - # - # SolutionInjection # solutions=[{}]
# # - # 0 # 1 # 0.00 # 0 #
#####
```

```
# 1 # 2 # - # PipelineJoin # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person) # - # 1 # 2 # 2.00 # 13 #
# # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
#####
# 2 # 3 # - # PipelineJoin # pattern=distinct(?person, foaf:firstName, ?
firstName) # - # 2 # 2 # 1.00 # 3 #
# # # # # # #
# # # # # # #
# # # # # # #
# # # # # # #
#####
# 3 # 4 # - # Projection # vars=[?firstName]
# retain # 2 # 2 # 1.00 # 1 #
#####
# 4 # - # - # TermResolution # vars=[?firstName]
# id2value # 2 # 2 # 1.00 # 7 #
#####
```

Example of static mode output

Suppose that you run the same query as the previous in *static* mode (the default) instead of *details* mode:

```
curl http(s)://your_server:your_port/sparql \
-d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person foaf:firstName ?
firstName }" \
-d "explain=static"
```

As this example shows, the output is the same, except that it omits the last three columns:

```
#####
# ID # Out #1 # Out #2 # Name # Arguments
# Mode #
#####
# 0 # 1 # - # SolutionInjection # solutions=[{}]
```

```
#####
# 1 # 2 # - # PipelineJoin # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person) # - #
# # # # # # #
# # # # # # #
# # # # # # #
#####
# 2 # 3 # - # PipelineJoin # pattern=distinct(?person, foaf:firstName, ?
firstName) # - #
# # # # # # #
# # # # # # #
# # # # # # #
#####
# 3 # 4 # - # Projection # vars=[?firstName]
# retain #
#####
```



```
# 4 # - # - # TermResolution # vars=[?firstName]
# id2value #
#####
```

Different ways of encoding parameters

The following example queries illustrate two different ways to encode parameters when invoking SPARQL explain.

Using URL encoding – This example uses URL encoding of parameters, and specifies *dynamic* output:

```
curl -XGET "http(s)://your_server:your_port/sparql?query=SELECT%20*%20WHERE%20%7B%20%3Fs%20%3Fp%20%3Fo%20%7D%20LIMIT%20%31&explain=dynamic"
```

Specifying the parameters directly – This is the same as the previous query except that it passes the parameters through POST directly:

```
curl http(s)://your_server:your_port/sparql \
-d "query=SELECT * WHERE { ?s ?p ?o } LIMIT 1" \
-d "explain=dynamic"
```

Other output types besides text/plain

The preceding examples use the default `text/plain` output type. Neptune can also format SPARQL explain output in two other MIME-type formats, namely `text/csv` and `text/html`. You invoke them by setting the HTTP Accept header, which you can do using the `-H` flag in `curl`, as follows:

```
-H "Accept: output type"
```

Here are some examples:

text/csv Output

This query calls for CSV MIME-type output by specifying `-H "Accept: text/csv"`:

```
curl http(s)://your_server:your_port/sparql \
-d "query=SELECT * WHERE { ?s ?p ?o } LIMIT 1" \
-d "explain=dynamic" \
-H "Accept: text/csv"
```

The CSV format, which is handy for importing into a spreadsheet or database, separates the fields in each explain row by semicolons (;), like this:

```
ID;Out #1;Out #2;Name;Arguments;Mode;Units In;Units Out;Ratio;Time (ms)
0;1;-;SolutionInjection;solutions=[{}];-;0;1;0.00;0
1;2;-;PipelineJoin;pattern=distinct(?s, ?p, ?o),joinType=join,joinProjectionVars=[?s, ?p, ?o];-;1;6;6.00;1
2;3;-;Projection;vars=[?s, ?p, ?o];retain;6;6;1.00;2
3;-;-;Slice;limit=1;-;1;1;1.00;1
```

text/html Output

If you specify `-H "Accept: text/html"`, then explain generates an HTML table:

```
<!DOCTYPE html>
```

```

<html>
  <body>
    <table border="1px">
      <thead>
        <tr>
          <th>ID</th>
          <th>Out #1</th>
          <th>Out #2</th>
          <th>Name</th>
          <th>Arguments</th>
          <th>Mode</th>
          <th>Units In</th>
          <th>Units Out</th>
          <th>Ratio</th>
          <th>Time (ms)</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>0</td>
          <td>1</td>
          <td>-</td>
          <td>SolutionInjection</td>
          <td>solutions=[{}]</td>
          <td>-</td>
          <td>0</td>
          <td>1</td>
          <td>0.00</td>
          <td>0</td>
        </tr>
        <tr>
          <td>1</td>
          <td>2</td>
          <td>-</td>
          <td>PipelineJoin</td>
          <td>pattern=distinct(?s, ?p, ?o)<br>
            joinType=join<br>
            joinProjectionVars=[?s, ?p, ?o]</td>
          <td>-</td>
          <td>1</td>
          <td>6</td>
          <td>6.00</td>
          <td>1</td>
        </tr>
        <tr>
          <td>2</td>
          <td>3</td>
          <td>-</td>
          <td>Projection</td>
          <td>vars=[?s, ?p, ?o]</td>
          <td>retain</td>
          <td>6</td>
          <td>6</td>
          <td>1.00</td>
          <td>2</td>
        </tr>
        <tr>
          <td>3</td>
          <td>-</td>
          <td>-</td>
          <td>Slice</td>
          <td>limit=1</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>

```

```
<td>--</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
<td>1</td>
</tr>
</tbody>
</table>
</body>
</html>
```

The HTML renders in a browser something like the following:

ID	Out #1	Out #2	Name	Arguments	Mode	Units In	Units Out	Ratio	Time (ms)
0	1	-	SolutionInjection	solutions={}	-	0	1	0.00	0
1	2	-	PipelineJoin	pattern=distinct(?s, ?p, ?o) joinType=join joinProjectionVars=[?s, ?p, ?o]	-	1	6	6.00	1
2	3	-	Projection	vars=[?s, ?p, ?o]	retain	6	6	1.00	2
3	-	-	Slice	limit=1	-	1	1	1.00	1

Example of SPARQL explain output when the DFE is enabled

The following is an example of SPARQL explain output when the Neptune DFE alternative query engine is enabled:

```
#####
# ID # Out #1 # Out #2 # Name # Arguments # Mode #
# Units In # Units Out # Ratio # Time (ms) #
#####
# 0 # 1 # - # SolutionInjection # solutions={} # - #
# 0 # 1 # 0.00 # 0 #
#####
# 1 # 2 # - # HashIndexBuild # solutionSet=solutionSet1 # - #
# 1 # 1 # 1.00 # 22 #
# # # # #
# # # # #
# # # # #
# # # # #
#####
# 2 # 3 # - # DFENode # DFE Stats= # -
# # 101 # 100 # 0.99 # 32 #
# # # # # # ==> DFE execution time (measured by
DFEQueryEngine) #
# # # # # #
# # # # # # accepted [micros]=127 #
# # # # # # ready [micros]=2 #
```



```
# # # # # # # # # #
# # # # # # # # # # rec=OperatorInfo[
# # # # # # # # # #
# # # # # # # # # # type=INCREMENTAL_HASH_JOIN,
# # # # # # # # # #
# # # # # # # # # # costEstimates=OperatorCostEstimates[
# # # # # # # # # #
# # # # # # # # # # costEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0003,comp=0.0000,mem=6400],
# # # # # # # # # #
# # # # # # # # # # worstCaseCostEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0003,comp=0.0000,mem=6400]],
# # # # # # # # # #
# # # # # # # # # # alt=OperatorInfo[
# # # # # # # # # #
# # # # # # # # # # type=INCREMENTAL_PIPELINE_JOIN,
# # # # # # # # # #
# # # # # # # # # # costEstimates=OperatorCostEstimates[
# # # # # # # # # #
# # # # # # # # # # costEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0010,comp=0.0000,mem=0],
# # # # # # # # # #
# # # # # # # # # # worstCaseCostEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0010,comp=0.0000,mem=0]]]]
# # # # # # # # # #
# # # # # # # # # # # },
# # # # # # # # # # # ]
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # # # ==> DFE configuration:
# # # # # # # # # #
# # # # # # # # # # # solutionChunkSize=5000
# # # # # # # # # #
# # # # # # # # # # # ouputQueueSize=20
# # # # # # # # # #
# # # # # # # # # #
```


Amazon Neptune User Guide
SPARQL explain

```

#
# # # # # # # #
# # # # # # # # # -> 10.99% of total time (excl. wait): merge (1
instances)
# # # # # # # #
# # # # # # # # # -> 41.17% of total time (excl. wait):
symmetricHashJoin (1 instances)
# # # # # # # #
# # # # # # # # # -> 0.19% of total time (excl. wait): drain (1
instances)
# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # # # nodeId | out0 | out1 | opName |
args | rowsIn | rowsOut | chunksIn | chunksOut
| elapsed* | outWait | outBlocked | ratio | rate* [M/s] | rate [M/s] | % #
# # # # # # # #
# # # # # # # # # ----- | ----- | ---- | ----- |
----- | ----- | ----- | ----- |
# # # # # # # #
# # # # # # # # # node_0 | node_2 | - | pipelineScan |
(?1, TERM[117442062], ?2, ?3) DISTINCT [?1, ?2] | 0 | 100 | 0 | 1
| 874 | 0 | 0 | Infinity | 0.1144 | 0.1144 | 23.83 #
# # # # # # # #
# # # # # # # # # node_1 | node_2 | - | pipelineScan |
(?1, TERM[150997262], ?4, ?5) DISTINCT [?1, ?4] | 0 | 100 | 0 | 1
| 874 | 0 | 0 | Infinity | 0.1144 | 0.1144 | 23.83 #
# # # # # # # #
# # # # # # # # # node_2 | node_4 | - | symmetricHashJoin |
| 200 | 100 | 2 | 2
1510 | 73 | 0 | 0.50 | 0.0662 | 0.0632 | 41.17 #
# # # # # # # #
# # # # # # # # # node_3 | - | - | drain |
| 100 | 0 | 1 | 0
7 | 0 | 0 | 0.00 | 0.0000 | 0.0000 | 0.19 #
# # # # # # # #
# # # # # # # # # node_4 | node_3 | - | merge |
| 100 | 100 | 2 | 1
403 | 0 | 0 | 1.00 | 0.2481 | 0.2481 | 10.99 #
# # # # # # # #
#####
# 3 # 4 # - # HashIndexJoin # solutionSet=solutionSet1
# # # # # # # # # - #
100 # 100 # 1.00 # 4 #
# # # # # # # # # # joinType=join
# # # # # # # #
# # # # # # # # #
#####
# 4 # 5 # - # Distinct # vars=[?s, ?o, ?o1]
# # # # # # # # # - #
100 # 100 # 1.00 # 9 #
#####
# 5 # 6 # - # Projection # vars=[?s, ?o, ?o1]
# # # # # # # # # # retain #
100 # 100 # 1.00 # 2 #
#####

```



```
# 6 # - # - # TermResolution # vars=[?s, ?o, ?o1]
# id2value #
100 # 100 # 1.00 # 11 #
#####
```

Neptune SPARQL explain operators

The following sections describe the operators and parameters for the SPARQL explain feature currently available in Amazon Neptune.

Important

The SPARQL explain feature is still being refined. The operators and parameters documented here might change in future versions.

Topics

- [Aggregation operator \(p. 384\)](#)
- [ConditionalRouting operator \(p. 385\)](#)
- [Copy operator \(p. 385\)](#)
- [DFENode operator \(p. 385\)](#)
- [Distinct operator \(p. 385\)](#)
- [Federation operator \(p. 386\)](#)
- [Filter operator \(p. 386\)](#)
- [HashIndexBuild operator \(p. 386\)](#)
- [HashIndexJoin operator \(p. 386\)](#)
- [MergeJoin operator \(p. 387\)](#)
- [NamedSubquery operator \(p. 387\)](#)
- [PipelineJoin operator \(p. 387\)](#)
- [PipelineCountJoin operator \(p. 388\)](#)
- [Projection operator \(p. 388\)](#)
- [PropertyPath operator \(p. 388\)](#)
- [TermResolution operator \(p. 389\)](#)
- [Slice operator \(p. 389\)](#)
- [SolutionInjection operator \(p. 389\)](#)
- [Sort operator \(p. 390\)](#)
- [VariableAlignment operator \(p. 390\)](#)

Aggregation operator

Performs one or more aggregations, implementing the semantics of SPARQL aggregation operators such as count, max, min, sum, and so on.

Aggregation comes with optional grouping using `groupBy` clauses, and optional `having` constraints.

Arguments

- `groupBy` – (*Optional*) Provides a `groupBy` clause that specifies the sequence of expressions according to which the incoming solutions are grouped.
- `aggregates` – (*Required*) Specifies an ordered list of aggregation expressions.

- `having` – (*Optional*) Adds constraints to filter on groups, as implied by the `having` clause in the SPARQL query.

ConditionalRouting operator

Routes incoming solutions based on a given condition. Solutions that satisfy the condition are routed to the operator ID referenced by `Out #1`, whereas solutions that do not are routed to the operator referenced by `Out #2`.

Arguments

- `condition` – (*Required*) The routing condition.

Copy operator

Delegates the solution stream as specified by the specified mode.

Modes

- `forward` – Forwards the solutions to the downstream operator identified by `Out #1`.
- `duplicate` – Duplicates the solutions and forwards them to each of the two operators identified by `Out #1` and `Out #2`.

Copy has no arguments.

DFENode operator

This operator is an abstraction of the plan that is run by the DFE alternative query engine. The detailed DFE plan is outlined in the arguments for this operator. The argument is currently overloaded to contain the detailed runtime statistics of the DFE plan. It contains the time spent in the various steps of query execution by DFE.

The logical optimized abstract syntax tree (AST) for the DFE query plan is printed with information about the operator types that were considered while planning and the associated best- and worst-case costs to run the operators. The AST consists of the following type of nodes at the moment:

- `DFEJoinGroupNode` – Represents a join of one or more `DFEPatternNodes`.
- `DFEPatternNode` – Encapsulates an underlying pattern using which matching tuples are projected out of the underlying database.

The sub-section, `Statistics & Operator histogram`, contains details about the execution time of the `DataflowOp` plan and the breakdown of CPU time used by each operator. Below this there is a table which prints detailed runtime statistics of the plan executed by DFE.

Note

Because the DFE is an experimental feature released in lab mode, the exact format of its `explain` output may change.

Distinct operator

Computes the distinct projection on a subset of the variables, eliminating duplicates. As a result, the number of solutions flowing in is larger than or equal to the number of solutions flowing out.

Arguments

- `vars` – (*Required*) The variables to which to apply the `Distinct` projection.

Federation operator

Passes a specified query to a specified remote SPARQL endpoint.

Arguments

- `endpoint` – *(Required)* The endpoint URL in the SPARQL `SERVICE` statement. This can be a constant string, or if the query endpoint is determined based on a variable within the same query, it can be the variable name.
- `query` – *(Required)* The reconstructed query string to be sent to the remote endpoint. The engine adds default prefixes to this query even when the client doesn't specify any.
- `silent` – *(Required)* A Boolean that indicates whether the `SILENT` keyword appeared after the keyword. `SILENT` tells the engine not to fail the whole query even if the remote `SERVICE` portion fails.

Filter operator

Filters the incoming solutions. Only those solutions that satisfy the filter condition are forwarded to the upstream operator, and all others are dropped.

Arguments

- `condition` – *(Required)* The filter condition.

HashIndexBuild operator

Takes a list of bindings and spools them into a hash index whose name is defined by the `solutionSet` argument. Typically, subsequent operators perform joins against this solution set, referring to it by that name.

Arguments

- `solutionSet` – *(Required)* The name of the hash index solution set.
- `sourceType` – *(Required)* The type of the source from which the bindings to store in the hash index are obtained:
 - `pipeline` – Spools the incoming solutions from the downstream operator in the operator pipeline into the hash index.
 - `binding set` – Spools the fixed binding set specified by the `sourceBindingSet` argument into the hash index.
- `sourceBindingSet` – *(Optional)* If the `sourceType` argument value is `binding set`, this argument specifies the static binding set to be spooled into the hash index.

HashIndexJoin operator

Joins the incoming solutions against the hash index solution set identified by the `solutionSet` argument.

Arguments

- `solutionSet` – *(Required)* Name of the solution set to join against. This must be a hash index that has been constructed in a prior step using the `HashIndexBuild` operator.
- `joinType` – *(Required)* The type of join to be performed:
 - `join` – A normal join, requiring an exact match between all shared variables.
 - `optional` – An optional join that uses the SPARQL `OPTIONAL` operator semantics.

- `minus` – A minus operation retains a mapping for which no join partner exists, using the SPARQL `MINUS` operator semantics.
- `existence check` – Checks whether there is a join partner or not, and binds the `existenceCheckResultVar` variable to the result of this check.
- `constraints` – (*Optional*) Additional join constraints that are considered during the join. Joins that do not satisfy these constraints are discarded.
- `existenceCheckResultVar` – (*Optional*) Only used for joins where `joinType` equals `existence check` (see the `joinType` argument earlier).

MergeJoin operator

A merge join over multiple solution sets, as identified by the `solutionSets` argument.

Arguments

- `solutionSets` – (*Required*) The solution sets to join together.

NamedSubquery operator

Triggers evaluation of the subquery identified by the `subQuery` argument and spools the result into the solution set specified by the `solutionSet` argument. The incoming solutions for the operator are forwarded to the subquery and then to the next operator.

Arguments

- `subQuery` – (*Required*) Name of the subquery to evaluate. The subquery is rendered explicitly in the output.
- `solutionSet` – (*Required*) The name of the solution set in which to store the subquery result.

PipelineJoin operator

Receives as input the output of the previous operator and joins it against the tuple pattern defined by the `pattern` argument.

Arguments

- `pattern` – (*Required*) The pattern, which takes the form of a subject-predicate-object, and optionally -graph tuple that underlies the join. If `distinct` is specified for the pattern, the join only extracts distinct solutions from projection variables specified by the `projectionVars` argument, rather than all matching solutions.
- `inlineFilters` – (*Optional*) A set of filters to be applied to the variables in the pattern. The pattern is evaluated in conjunction with these filters.
- `joinType` – (*Required*) The type of join to be performed:
 - `join` – A normal join, requiring an exact match between all shared variables.
 - `optional` – An optional join that uses the SPARQL `OPTIONAL` operator semantics.
 - `minus` – A minus operation retains a mapping for which no join partner exists, using the SPARQL `MINUS` operator semantics.
 - `existence check` – Checks whether there is a join partner or not, and binds the `existenceCheckResultVar` variable to the result of this check.
- `constraints` – (*Optional*) Additional join constraints that are considered during the join. Joins that do not satisfy these constraints are discarded.
- `projectionVars` – (*Optional*) The projection variables. Used in combination with `distinct := true` to enforce the extraction of distinct projections over a specified set of variables.

- `cutoffLimit` – *(Optional)* A cutoff limit for the number of join partners extracted. Although there is no limit by default, you can set this to 1 when performing joins to implement `FILTER (NOT) EXISTS` clauses, where it is sufficient to prove or disprove that there is a join partner.

PipelineCountJoin operator

Variant of the `PipelineJoin`. Instead of joining, it just counts the matching join partners and binds the count to the variable specified by the `countVar` argument.

Arguments

- `countVar` – *(Required)* The variable to which the count result, namely the number of join partners, should be bound.
- `pattern` – *(Required)* The pattern, which takes the form of a subject-predicate-object, and optionally -graph tuple that underlies the join. If `distinct` is specified for the pattern, the join only extracts distinct solutions from projection variables specified by the `projectionVars` argument, rather than all matching solutions.
- `inlineFilters` – *(Optional)* A set of filters to be applied to the variables in the pattern. The pattern is evaluated in conjunction with these filters.
- `joinType` – *(Required)* The type of join to be performed:
 - `join` – A normal join, requiring an exact match between all shared variables.
 - `optional` – An optional join that uses the SPARQL `OPTIONAL` operator semantics.
 - `minus` – A minus operation retains a mapping for which no join partner exists, using the SPARQL `MINUS` operator semantics.
 - `existence check` – Checks whether there is a join partner or not, and binds the `existenceCheckResultVar` variable to the result of this check.
- `constraints` – *(Optional)* Additional join constraints that are considered during the join. Joins that do not satisfy these constraints are discarded.
- `projectionVars` – *(Optional)* The projection variables. Used in combination with `distinct := true` to enforce the extraction of distinct projections over a specified set of variables.
- `cutoffLimit` – *(Optional)* A cutoff limit for the number of join partners extracted. Although there is no limit by default, you can set this to 1 when performing joins to implement `FILTER (NOT) EXISTS` clauses, where it is sufficient to prove or disprove that there is a join partner.

Projection operator

Projects over a subset of the variables. The number of solutions flowing in equals the number of solutions flowing out, but the shape of the solution differs, depending on the mode setting.

Modes

- `retain` – Retain in solutions only the variables that are specified by the `vars` argument.
- `drop` – Drop all the variables that are specified by the `vars` argument.

Arguments

- `vars` – *(Required)* The variables to retain or drop, depending on the mode setting.

PropertyPath operator

Enables recursive property paths such as `+` or `*`. Neptune implements a fixed-point iteration approach based on a template specified by the `iterationTemplate` argument. Known left-side or right-side

variables are bound in the template for every fixed-point iteration, until no more new solutions can be found.

Arguments

- `iterationTemplate` – (*Required*) Name of the subquery template used to implement the fixed-point iteration.
- `leftTerm` – (*Required*) The term (variable or constant) on the left side of the property path.
- `rightTerm` – (*Required*) The term (variable or constant) on the right side of the property path.
- `lowerBound` – (*Required*) The lower bound for fixed-point iteration (either 0 for * queries, or 1 for + queries).

TermResolution operator

Translates internal string identifier values back to their corresponding external strings, or translates external strings to internal string identifier values, depending on the mode.

Modes

- `value2id` – Maps terms such as literals and URIs to corresponding internal ID values (encoding to internal values).
- `id2value` – Maps internal ID values to the corresponding terms such as literals and URIs (decoding of internal values).

Arguments

- `vars` – (*Required*) Specifies the variables whose strings or internal string IDs should be mapped.

Slice operator

Implements a slice over the incoming solution stream, using the semantics of SPARQL's `LIMIT` and `OFFSET` clauses.

Arguments

- `limit` – (*Optional*) A limit on the solutions to be forwarded.
- `offset` – (*Optional*) The offset at which solutions are evaluated for forwarding.

SolutionInjection operator

Receives no input. Statically injects solutions into the query plan and records them in the `solutions` argument.

Query plans always begin with this static injection. If static solutions to inject can be derived from the query itself by combining various sources of static bindings (for example, from `VALUES` or `BIND` clauses), then the `SolutionInjection` operator injects these derived static solutions. In the simplest case, these reflect bindings that are implied by an outer `VALUES` clause.

If no static solutions can be derived from the query, `SolutionInjection` injects the empty, so-called universal solution, which is expanded and multiplied throughout the query-evaluation process.

Arguments

- `solutions` – (*Required*) The sequence of solutions injected by the operator.

Sort operator

Sorts the solution set using specified sort conditions.

Arguments

- `sortOrder` – *(Required)* An ordered list of variables, each containing an `ASC` (ascending) or `DESC` (descending) identifier, used sequentially to sort the solution set.

VariableAlignment operator

Inspects solutions one by one, performing alignment on each one over two variables: a specified `sourceVar` and a specified `targetVar`.

If `sourceVar` and `targetVar` in a solution have the same value, the variables are considered aligned and the solution is forwarded, with the redundant `sourceVar` projected out.

If the variables bind to different values, the solution is filtered out entirely.

Arguments

- `sourceVar` – *(Required)* The source variable, to be compared to the target variable. If alignment succeeds in a solution, meaning that the two variables have the same value, the source variable is projected out.
- `targetVar` – *(Required)* The target variable, with which the source variable is compared. Is retained even when alignment succeeds.

Limitations of SPARQL explain in Neptune

The release of the Neptune SPARQL `explain` feature has the following limitations.

Neptune Currently Supports Explain Only in SPARQL SELECT Queries

For information about the evaluation process for other query forms, such as `ASK`, `CONSTRUCT`, `DESCRIBE`, and `SPARQL UPDATE` queries, you can transform these queries into a `SELECT` query. Then use `explain` to inspect the corresponding `SELECT` query instead.

For example, to obtain `explain` information about an `ASK WHERE {...}` query, run the corresponding `SELECT WHERE {...} LIMIT 1` query with `explain`.

Similarly, for a `CONSTRUCT {...} WHERE {...}` query, drop the `CONSTRUCT {...}` part and run a `SELECT` query with `explain` on the second `WHERE {...}` clause. Evaluating the second `WHERE` clause generally reveals the main challenges of processing the `CONSTRUCT` query, because solutions flowing out of the second `WHERE` into the `CONSTRUCT` template generally only require straightforward substitution.

Explain Operators May Change in Future Releases

The SPARQL `explain` operators and their parameters may change in future releases.

Explain Output May Change in Future Releases

For example, column headers could change, and more columns might be added to the tables.

SPARQL federated queries in Neptune using the SERVICE extension

Amazon Neptune fully supports the SPARQL federated query extension that uses the `SERVICE` keyword. (For more information, see [SPARQL 1.1 Federated Query](#).)

Note

This feature is available starting in [Release 1.0.1.0.200463.0 \(2019-10-15\)](#) (p. 767).

The `SERVICE` keyword instructs the SPARQL query engine to execute a portion of the query against a remote SPARQL endpoint and compose the final query result. Only `READ` operations are possible. `WRITE` and `DELETE` operations are not supported. Neptune can only run federated queries against SPARQL endpoints that are accessible within its virtual private cloud (VPC). However, you can also use a reverse proxy in the VPC to make an external data source accessible within the VPC.

Note

When SPARQL `SERVICE` is used to federate a query to two or more Neptune clusters in the same VPC, the security groups must be configured to allow all those Neptune clusters to talk to each another.

Important

SPARQL 1.1 Federation makes service requests on your behalf when passing queries and parameters to external SPARQL endpoints. It is your responsibility to verify that the external SPARQL endpoints satisfy your application's data handling and security requirements.

Example of a Neptune federated query

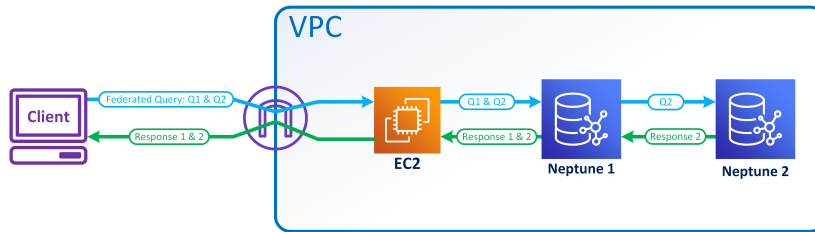
The following simple example shows how SPARQL federated queries work.

Suppose that a customer sends the following query to *Neptune-1* at `http://neptune-1:8182/sparql`.

```
SELECT * WHERE {
  ?person rdf:type foaf:Person .
  SERVICE <http://neptune-2:8182/sparql> {
    ?person foaf:knows ?friend .
  }
}
```

1. *Neptune-1* evaluates the first query pattern (Q-1) which is `?person rdf:type foaf:Person`, uses the results to resolve `?person` in Q-2 (`?person foaf:knows ?friend`), and forwards the resulting pattern to *Neptune-2* at `http://neptune-2:8182/sparql`.
2. *Neptune-2* evaluates Q-2 and sends the results back to *Neptune-1*.
3. *Neptune-1* joins the solutions for both patterns and sends the results back to the customer.

This flow is shown in the following diagram.



Note

"By default, the optimizer determines at what point in query execution that the `SERVICE` instruction is executed. You can override this placement using the `joinOrder` (p. 359) query hint.

Access control for federated queries in Neptune

Neptune uses AWS Identity and Access Management (IAM) for authentication and authorization. Access control for a federated query can involve more than one Neptune DB instance. These instances might

have different requirements for access control. In certain circumstances, this can limit your ability to make a federated query.

Consider the simple example presented in the previous section. *Neptune-1* calls *Neptune-2* with the same credentials it was called with.

- If *Neptune-1* requires IAM authentication and authorization, but *Neptune-2* does not, all you need is appropriate IAM permissions for *Neptune-1* to make the federated query.
- If *Neptune-1* and *Neptune-2* both require IAM authentication and authorization, you need to attach IAM permissions for both databases to make the federated query.
- However, in the case where *Neptune-1* is not IAM-enabled but *Neptune-2* is, you can't make a federated query. The reason is that *Neptune-1* can't retrieve your IAM credentials and pass them on to *Neptune-2* to authorize the second part of the query.

Accessing the Neptune Graph with openCypher

Warning

This lab-mode release of the Neptune openCypher graph engine is intended for evaluation purposes only. **Do not use it in production.**

openCypher is a declarative query language for property graphs that was originally developed by Neo4j, then open-sourced in 2015, and contributed to the [openCypher](#) project under an Apache 2 open-source license. Its syntax is documented in the [Cypher Query Language Reference, Version 9](#).

Amazon Neptune released experimental support for openCypher in [engine release 1.0.5.0 \(p. 725\)](#), for you to try out. The initial release is not a complete implementation of openCypher, as is documented in [Neptune openCypher restrictions \(p. 403\)](#), and the features, APIs and functionality that are implemented here may change before the production release. This initial release is intended for evaluation purposes only, and is not intended for use in production.

Important

We do want to hear back from you about anything that isn't working in this release, and about what features and functionality you'd like to see prioritized in future releases.

Please email us your thoughts at <neptune-opencypher-feedback@amazon.com>.

Enabling openCypher support in Amazon Neptune

Support for openCypher is automatically enabled in Neptune starting with [engine release 1.0.5.0 \(p. 725\)](#).

However, openCypher support depends on the [Neptune DFE engine \(p. 87\)](#), which is currently only available in [lab mode \(p. 85\)](#). The default `DFEQueryEngine` setting in the `neptune_lab_mode` DB cluster parameter is now `DFEQueryEngine=viaQueryHint`, which means that the engine is enabled but only used for queries that have the `useDFE` query hint present and set to `true`. If you disable the DFE engine by setting `DFEQueryEngine=disabled`, you will not be able to use openCypher.

Overview of the openCypher query language in Amazon Neptune

The Cypher query language was originally created by [Neo4j](#) and then open-sourced in 2015, and contributed to the [openCypher project](#) under an Apache 2 open-source license. See the [Cypher Query Language Reference, Version 9](#) for the language specification.

This Neptune release gives you the option to build graph applications using openCypher, currently one of the most popular query languages for developers working with graph databases. Developers,

business analysts, and data scientists like openCypher's SQL-inspired syntax because it provides a familiar structure to compose queries for graph applications.

Gremlin vs. openCypher: similarities and differences

Gremlin and openCypher are both property-graph query languages, and they are complementary in many ways.

Gremlin was designed to appeal to programmers and fit seamlessly into code. As a result, Gremlin is imperative by design, whereas openCypher's declarative syntax may feel more familiar for people with SQL or SPARQL experience. Gremlin might seem more natural to a data scientist using Python in a Jupyter notebook, whereas openCypher might seem more intuitive to a business user with some SQL background.

The nice thing is that **you don't have to choose** between Gremlin and openCypher in Neptune. Queries in either language can operate on the same graph regardless of which of the two language was used to enter that data. You may find it more convenient to use Gremlin for some things and openCypher for others, depending on what you're doing.

Gremlin uses an imperative syntax that lets you control how you move through your graph in a series of steps, each of which takes in a stream of data, performs some action on it (using a filter, map, and so forth), and then outputs the results to the next step. A Gremlin query commonly takes the form, `g.V()`, followed by additional steps.

In openCypher, you use a declarative syntax, inspired by SQL, that specifies a pattern of nodes and relationships to find in your graph using a motif syntax (like `()-[]->()`). An openCypher query often starts with a `MATCH` clause, followed by other clauses such as `WHERE`, `WITH`, and `RETURN`.

Getting started using openCypher

You can query property-graph data in Neptune using openCypher regardless of how it was loaded, but you can't use openCypher to query data loaded as RDF.

The [Neptune bulk loader \(p. 190\)](#) accepts property-graph data in a [CSV format for Gremlin \(p. 195\)](#), and in a [CSV format for openCypher \(p. 200\)](#). Also, of course, you can add property data to your graph using Gremlin and/or openCypher queries.

There are many online tutorials available for learning the Cypher query language. Here, a few quick examples of openCypher queries may help you get an idea of the language, but by far the best and easiest way to get started using openCypher to query your Neptune graph is by using the openCypher notebooks in the [Neptune workbench \(p. 29\)](#). The workbench is open-source, and is hosted on GitHub at <https://github.com/aws-samples/amazon-neptune-samples>.

You'll find the openCypher notebooks in the GitHub [Neptune graph-notebook repository](#). In particular, check out the [Air-routes visualization](#), and [English Premier Teams](#) notebooks for openCypher.

Data processed by openCypher takes the form of an unordered series of key/value maps. The main way to refine, manipulate, and augment these maps is to use clauses that perform tasks such as pattern matching, insertion, update, and deletion on the key/value pairs.

There are several clauses in openCypher for finding data patterns in the graph, of which `MATCH` is the most common. `MATCH` lets you specify the pattern of nodes, relationships, and filters that you want to look for in your graph. For example:

- **Get all nodes**

```
MATCH (n) RETURN n
```

- **Find connected nodes**

```
MATCH (n)-[r]->(d) RETURN n, r, d
```

- **Find a path**

```
MATCH p=(n)-[r]->(d) RETURN p
```

- **Get all nodes with a label**

```
MATCH (n:airport) RETURN n
```

Note that the first query above returns every single node in your graph, and the next two return every node that has a relationship— this is not generally recommended! In almost all cases, you want to narrow down the data being returned, which you can do by specifying node or relationship labels and properties, as in the fourth example.

You can find a handy cheat-sheet for openCypher syntax in the Neptune [github sample repository](#).

Important

Special feedback link: If you encounter any problems using openCypher in this Neptune lab-mode release or have suggestions, please email us at <neptune-opencypher-feedback@amazon.com>. We're here to help.

Neptune openCypher status servlet and status endpoint

The openCypher status endpoint provides access to information about queries that are currently running on the server or waiting to run. It also lets you cancel those queries. The endpoint is:

```
https://(the server):(the port number)/openCypher/status
```

You can use the HTTP `GET` and `POST` methods to get current status from the server, or to cancel a query. You can also use the `DELETE` method to cancel a running or waiting query.

Parameters for status requests

Status query parameters

- **includeWaiting** (`true` or `false`) – When set to `true` and other parameters are not present, causes status information for waiting queries to be returned as well as for running queries.
- **cancelQuery** – Used only with `GET` and `POST` methods, to indicate that this is a cancellation request. The `DELETE` method does not need this parameter.

The value of the `cancelQuery` parameter is not used, but when `cancelQuery` is present, the `queryId` parameter is required, to identify which query to cancel.

- **queryId** – Contains the ID of a specific query.

When used with the `GET` or `POST` method and the `cancelQuery` parameter is not present, `queryId` causes status information to be returned for the specific query it identifies. If the `cancelQuery` parameter is present, then the specific query that `queryId` identifies is canceled.

When used with the `DELETE` method, `queryId` always indicates a specific query to be canceled.

- **silent** – Only used when canceling a query. If set to `true`, causes the cancellation to happen silently.

Examples of status requests and responses

- Request for the status of all queries, including those waiting:

```
curl https://server:port/openCypher/status \  
--data-urlencode "includeWaiting=true"
```

Response:

```
{  
  "acceptedQueryCount" : 0,  
  "runningQueryCount" : 0,  
  "queries" : [ ]  
}
```

- Request for the status of running queries, not including those waiting::

```
curl https://server:port/openCypher/status
```

Response:

```
{  
  "acceptedQueryCount" : 0,  
  "runningQueryCount" : 0,  
  "queries" : [ ]  
}
```

- Request for the status of a single query:

```
curl https://server:port/openCypher/status \  
--data-urlencode "queryId=eadc6eea-698b-4a2f-8554-5270ab17ebee"
```

Response:

```
{  
  "queryId" : "eadc6eea-698b-4a2f-8554-5270ab17ebee",  
  "queryString" : "MATCH (n1)-[:knows]->(n2), (n2)-[:knows]->(n3), (n3)-[:knows]->(n4),  
(n4)-[:knows]->(n5), (n5)-[:knows]->(n6), (n6)-[:knows]->(n7), (n7)-[:knows]->(n8),  
(n8)-[:knows]->(n9), (n9)-[:knows]->(n10) RETURN COUNT(n1);",  
  "queryEvalStats" : {  
    "waited" : 0,  
    "elapsed" : 23463,  
    "cancelled" : false  
  }  
}
```

- Requests to cancel a query

1. Using POST:

```
curl -X POST https://server:port/openCypher/status \  
--data-urlencode "cancelQuery" \  
--data-urlencode "queryId=f43ce17b-db01-4d37-a074-c76d1c26d7a9"
```

Response:

```
{
```

```
"status" : "200 OK",  
"payload" : true  
}
```

2. Using GET:

```
curl -X GET https://server:port/openCypher/status \  
--data-urlencode "cancelQuery" \  
--data-urlencode "queryId=588af350-cfde-4222-bee6-b9cedc87180d"
```

Response:

```
{  
  "status" : "200 OK",  
  "payload" : true  
}
```

3. Using DELETE:

```
curl -X DELETE \  
-s "https://server:port/openCypher/status?queryId=b9a516d1-d25c-4301-bb80-10b2743ecf0e"
```

Response:

```
{  
  "status" : "200 OK",  
  "payload" : true  
}
```

Important

Special feedback link: If you encounter any problems using openCypher in this Neptune lab-mode release or have suggestions, please email us at <neptune-opencypher-feedback@amazon.com>. We're here to help.

The Amazon Neptune openCypher HTTPS endpoint

Topics

- [openCypher read and write queries on the HTTPS endpoint \(p. 396\)](#)
- [The default openCypher JSON results format \(p. 397\)](#)

openCypher read and write queries on the HTTPS endpoint

The openCypher HTTPS endpoint supports read and update queries using both the `GET` and the `POST` method. The `DELETE` and `PUT` methods are not supported.

The syntax is:

```
HTTPS://(the server):(the port number)/openCypher
```

Here are sample read queries, one that uses `POST` and one that uses `GET`:

1. Using POST:

```
curl HTTPS://server:port/openCypher \  
-d "query=MATCH (n1) RETURN n1;"
```

2. Using GET (the query string is URL-encoded):

```
curl -X GET \  
"HTTPS://server:port/openCypher?query=MATCH%20(n1)%20RETURN%20n1"
```

Here are sample write/update queries, one that uses POST and one that uses GET:

1. Using POST:

```
curl HTTPS://server:port/openCypher \  
-d "query=CREATE (n:Person { age: 25 })"
```

2. Using GET (the query string is URL-encoded):

```
curl -X GET \  
"HTTPS://server:port/openCypher?query=CREATE%20(n%3APerson%20%7B%20age%3A%2025%20%7D)"
```

The default openCypher JSON results format

The following JSON format is returned by default, or by setting the request header explicitly to `Accept: application/json`. This format is designed to be easily parsed into objects using native-language features of most libraries.

The JSON document that is returned contains one field, `results`, which contains the query return values. The examples below show the JSON formatting for common values.

Value response example:

```
{  
  "results": [  
    {  
      "count(a)": 121  
    }  
  ]  
}
```

Node response example:

```
{  
  "results": [  
    {  
      "a": {  
        "~id": "22",  
        "~entityType": "node",  
        "~labels": [  
          "airport"  
        ],  
        "~properties": {  
          "desc": "Seattle-Tacoma",  
          "lon": -122.30899810791,  
          "runways": 3,  
          "type": "airport",  
          "country": "US",  
          "region": "US-WA",  
          "lat": 47.4490013122559,  
        }  
      }  
    }  
  ]  
}
```

```
    "elev": 432,  
    "city": "Seattle",  
    "icao": "KSEA",  
    "code": "SEA",  
    "longest": 11901  
  }  
}  
]  
}
```

Relationship response example:

```
{  
  "results": [  
    {  
      "r": {  
        "~id": "7389",  
        "~entityType": "relationship",  
        "~start": "22",  
        "~end": "151",  
        "~type": "route",  
        "~properties": {  
          "dist": 956  
        }  
      }  
    }  
  ]  
}
```

Path response example:

```
{  
  "results": [  
    {  
      "p": [  
        {  
          "~id": "22",  
          "~entityType": "node",  
          "~labels": [  
            "airport"  
          ],  
          "~properties": {  
            "desc": "Seattle-Tacoma",  
            "lon": -122.30899810791,  
            "runways": 3,  
            "type": "airport",  
            "country": "US",  
            "region": "US-WA",  
            "lat": 47.4490013122559,  
            "elev": 432,  
            "city": "Seattle",  
            "icao": "KSEA",  
            "code": "SEA",  
            "longest": 11901  
          }  
        },  
        {  
          "~id": "7389",  
          "~entityType": "relationship",  
          "~start": "22",  
          "~end": "151",  
          "~type": "route",  
          "~properties": {  
            "dist": 956  
          }  
        }  
      ]  
    }  
  ]  
}
```

```
    "-properties": {
      "dist": 956
    }
  },
  {
    "-id": "151",
    "-entityType": "node",
    "-labels": [
      "airport"
    ],
    "-properties": {
      "desc": "Ontario International Airport",
      "lon": -117.600997924805,
      "runways": 2,
      "type": "airport",
      "country": "US",
      "region": "US-CA",
      "lat": 34.0559997558594,
      "elev": 944,
      "city": "Ontario",
      "icao": "KONT",
      "code": "ONT",
      "longest": 12198
    }
  }
]
}
```

Important

Special feedback link: If you encounter any problems using openCypher in this Neptune lab-mode release or have suggestions, please email us at <neptune-opencypher-feedback@amazon.com>. We're here to help.

Using the Bolt protocol to make openCypher queries to Neptune

Bolt is a statement-oriented client/server protocol initially developed by Neo4j and licensed under the Creative Commons 3.0 [Attribution-ShareAlike](#) license. It is client-driven, meaning that the client always initiates message exchanges.

To connect to Neptune using Neo4j's Bolt drivers, simply replace the URL and Port number with your cluster endpoints using the `bolt+s` URI scheme. If you have a single Neptune instance running, use the `read_write` endpoint. If multiple instances are running, then two drivers are recommended, one for the writer and another for all the read replicas. If you have only the default two endpoints, a `read_write` and a `read_only` driver are sufficient, but if you have custom endpoints as well, consider creating a driver instance for each one.

Neptune supports the latest Neo4j drivers for Bolt message specification versions: [4.0.3](#), [4.1.4](#), [4.2.7](#) and [4.3.3](#).

For examples of openCypher queries in various languages that use the Bolt drivers, see the Neo4j [Drivers & Language Guides](#) documentation.

Using Bolt with Java to connect to Neptune

You can download a driver for whatever version you want to use from the Maven [MVN repository](#), or can add this dependency to your project:


```
<dependency>
  <groupId>org.neo4j.driver</groupId>
  <artifactId>neo4j-java-driver</artifactId>
  <version>4.3.3</version>
</dependency>
```

Then, to connect to Neptune in Java using one of these Bolt drivers, create a driver instance for the primary/writer instance in your cluster using code like the following:

```
import org.neo4j.driver.Driver;
import org.neo4j.driver.GraphDatabase;

Driver driver = GraphDatabase.driver("bolt+s://(your cluster endpoint URL):8182");
```

If you have one or more reader replicas, you can similarly create a driver instance for them using code like this:

```
Driver read_only_driver =
    GraphDatabase.driver("bolt+s://(your cluster endpoint URL):8182");

Driver read_only_driver_with_timeout =
    GraphDatabase.driver("bolt+s://(your cluster endpoint URL):8182",
        Config.builder().withConnectionTimeout(30, TimeUnit.SECONDS).build());
```

If you have custom endpoints, it may also be worthwhile to create a driver instance for each one.

A Python openCypher query example using Bolt

Here is how to make an openCypher query in Python using Bolt:

```
python -m pip install neo4j
```

```
from neo4j import GraphDatabase
```

```
from neo4j import GraphDatabase
uri = "bolt+s://(your cluster endpoint URL):8182"
driver = GraphDatabase.driver(uri, auth=("", ""))
```

Or, with encryption:

```
from neo4j import GraphDatabase
uri = "bolt+s://(your cluster endpoint URL):8182"
driver = GraphDatabase.driver(uri, auth=("", "", encrypted=True))
```

Note that the auth parameters are ignored. You can then use your driver instance like this:

```
def create_friend_of(tx, name, friend):
    tx.run(f"MATCH (a:Person) WHERE a.name = '" + friend + "'"
           "CREATE (a)-[:KNOWS]->(:Person {name: $friend})",
           name=name, friend=friend)

with driver.session() as session:
    session.write_transaction(create_friend_of, "Alice", "Bob")

with driver.session() as session:
    session.write_transaction(create_friend_of, "Alice", "Carl")
```

```
driver.close()
```

A .NET openCypher query example using Bolt

Here is how to make an openCypher query in .NET using Bolt:

```
Install-Package Neo4j.Driver-4.3.0
```

```
Using Neo4j.Driver-4.3.0;

var driver = GraphDatabase.Driver("bolt+s://(your cluster endpoint URL):8182",
    AuthTokens.Basic("", ""));

public class HelloWorldExample : IDisposable
{
    private bool _disposed = false;

    ~HelloWorldExample() => Dispose(false);

    public void PrintGreeting(string message)
    {
        using (var session = driver.Session())
        {
            var greeting = session.WriteTransaction(tx =>
            {
                var result = tx.Run("CREATE (a:Greeting) " +
                    "SET a.message = '" + message + "'" +
                    "RETURN a.message + ', from node ' + id(a)",
                    new {message});
                return result.Single()[0].As<string>();
            });
            Console.WriteLine(greeting);
        }
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
        if (_disposed)
            return;

        _disposed = true;
    }

    public static void Main()
    {
        using (var greeter = new HelloWorldExample( ))
        {
            greeter.PrintGreeting("hello, world");
        }
    }
}
```

Bolt connection behavior in Neptune

Here are some things to keep in mind about Neptune Bolt connections:

- Because Bolt connections are created at the TCP layer, you can't use an [Application Load Balancer](#) in front of them, as you can with an HTTP endpoint.
- The port that Neptune uses for Bolt connections is **8182**.
- Based on the Bolt preamble passed to it, the Neptune server selects the highest appropriate Bolt version (1, 2, 3, or 4.2).
- The maximum number of connections to the Neptune server that a client can have open at any point in time is 1,000.
- If the client doesn't close a connection after a query, that connection can be used to execute the next query.
- However, if a connection is idle for 20 minutes, the server closes it automatically.
- When IAM authentication is enabled, a Bolt connection is always disconnected a few minutes more than 10 days after it was established if it hasn't already closed for some other reason.
- If the client sends a query for execution over a connection without having consumed the results of a previous query, the new query is discarded. To discard the previous results instead, the client must send a reset message over the connection.
- Only one transaction at a time can be created on a given connection.
- If an exception occurs during a transaction, the Neptune server rolls back the transaction and closes the connection. In this case, the driver creates a new connection for the next query.
- Be aware that sessions are not thread-safe. Multiple parallel operations must use multiple separate sessions.

Important

Special feedback link: If you encounter any problems using openCypher in this Neptune lab-mode release or have suggestions, please email us at <neptune-opencypher-feedback@amazon.com>. We're here to help.

openCypher data model

The Neptune openCypher engine builds on the same property-graph model as Gremlin. In particular:

- Every node has one or more labels. If you insert a node without labels, a default label named `vertex` is attached. If you try to delete all of a node's labels, an error is thrown.
- A relationship is an entity that has exactly one relationship type and that forms a unidirectional connection between two nodes (that is, *from* one of the nodes *to* the other).
- Both nodes and relationships can have properties, but don't have to. Neptune supports nodes and relationships with zero properties.
- Neptune does not support metaproperties, which are not included in the openCypher specification either.
- Properties in your graph can be multi-valued if they were created using Gremlin. That is a node or relationship can have the same property multiple times, each time with a different property value. Neptune has extended openCypher semantics to handle multi-valued properties gracefully.

Supported data types are documented in [openCypher data format \(p. 200\)](#). However, we do not recommend inserting `Date` or `Array` property values into an openCypher graph at present. Although it is possible to insert `Date` data using the Neptune bulk loader, you can't execute a query that has `Date` either as a filter or in the output. Similarly, you can insert an array property value using the bulk loader, the current Neptune openCypher release treats it as a set of multi-valued properties instead of as a single list value.

Below is the list of data types supported in this release:

- `Bool`

- `Byte`
- `Short`
- `Int`
- `Long`
- `Float` (Includes plus and minus Infinity and NaN, but not INF)
- `Double` (Includes plus and minus Infinity and NaN, but not INF)
- `String`

Important

Special feedback link: If you encounter any problems using openCypher in this Neptune lab-mode release or have suggestions, please email us at [<neptune-opencypher-feedback@amazon.com>](mailto:neptune-opencypher-feedback@amazon.com). We're here to help.

Neptune openCypher restrictions

The Amazon Neptune lab mode release of openCypher comes with a few limitations. There are parts of the [openCypher V9 specification](#) that are not yet supported. Future releases are expected to address many of those limitations.

Functions in Neptune openCypher that are currently not implemented or only partially implemented

Almost all functions specified in the [Cypher Query Language Reference, Version 9](#) are supported. Below is the list of functions which are not yet supported or are only partially supported:

Functions that are not yet implemented at all in Neptune openCypher

- **Trigonometric functions** – These are `acos`, `asin`, `atan`, `atan2`, `cos`, `cot`, `degrees`, `pi`, `radians`, `sin`, and `tan`.
- **Aggregating functions** – These are `percentileCont`, and `percentileDisc`.
- **Id values** – When using the Neo4j Bolt drivers, Neptune Id values are not returned. The reason is that those drivers only allow LONG values for Ids, and Neptune uses strings values for Ids.
- **List accessors** – These are `head` and `last`.
- **Date-time functions** – The `timestamp` function is not supported, and when using the Neo4j Bolt drivers, `Datetime` property values are not returned; an error is thrown instead.
- **Math operators** – `modulo` and `pow`.

Functions that are only partially supported in Neptune openCypher

- **size** – This is an overloaded method that operates over lists, pattern expressions, and strings. Currently, it only works for lists and strings.

Other constructs in Neptune openCypher that are currently not implemented or only partially implemented

Unimplemented or only partially implemented features

- **Clauses that are not yet implemented**
 - `CALL` (for calling a procedure).

- MANDATORY MATCH.
- UNION
- UNION ALL
- **Clauses that are partially implemented**
 - WITH – LIMIT and SKIP aren't supported in this release.
- **Expressions that are not currently implemented**
 - Parameters for parameterized queries.
 - Accessing dynamically computed property keys using [] with a node, relationship, or map.
 - Property access for maps.
- **Pattern expressions** – These currently cannot be projected using a PROJECTION clause, but can be used in the WHERE part of clauses for additional filtering.
- **Paths** – Paths are not yet supported as a structural type. As a result, dependent constructs such as shortestPath and allShortestPath are not supported either.
- **Variable-length pattern matching** – is only supported to a maximum depth of 10. Also, only outgoing or incoming relationships are supported, not bidirectional relationships.
- **Inequality expressions** – GreaterThan, GreaterThanOrEqualTo, LessThan, and LessThanOrEqualTo are not supported when either of the arguments is a Path.

Other limitations in Neptune openCypher language semantics

- Neptune represents node and relationship IDs as strings rather than integers. The ID equals the ID supplied via the data loader. If there is a namespace for the column, the namespace plus the ID. Consequently, the id function returns a string instead of an integer.
- The INTEGER datatype is limited to 64 bits. When converting larger floating point or string values to an integer using the TOINTEGER function, negative values are truncated to LLONG_MIN and positive values are truncated to LLONG_MAX.

For example:

```
RETURN TOINTEGER(2^100)
> 9223372036854775807

RETURN TOINTEGER(-1 * 2^100)
> -9223372036854775808
```

Differences between Neptune openCypher and Cypher

- Neptune openCypher removes whitespace as defined by Unicode in the trim(), ltrim() and rtrim() functions.
- In Neptune openCypher, toString(double) does not automatically switch to E notation for large values of the double.
- Although openCypher CREATE queries do not create multi-valued properties, they can exist in data created using Gremlin. If Neptune openCypher encounters a multi-value property, one of the values is arbitrarily chosen, creating a non-deterministic result.

Query queuing in Amazon Neptune

When developing and tuning graph applications, it can be helpful to know the implications of how queries are being queued by the database. In Amazon Neptune, query queuing occurs as follows:

- The maximum number of queries that can be queued up per instance, regardless of the instance size, is 8,192. Any queries over that number are rejected and fail with a `ThrottlingException`.
- The maximum number of queries that can be executing at one time is determined by the number of worker threads assigned, which is generally set to twice the number of virtual CPU cores (vCPUs) that are available.
- Query latency includes the time a query spends in the queue as well as network round-tripping and the time it actually takes to execute.

Determining how many queries are in your queue at a given moment

The `MainRequestQueuePendingRequests` CloudWatch metric records the the number of requests waiting in the input queue at a five-minutes granularity (see [Neptune CloudWatch Metrics \(p. 644\)](#)).

For Gremlin, you can obtain a current count of queries in the queue using the `acceptedQueryCount` value returned by the [Gremlin query status API \(p. 284\)](#). Note, however, that the `acceptedQueryCount` value returned by the [SPARQL query status API \(p. 365\)](#) includes all queries accepted since the server was started, including completed queries.

How query queuing can affect timeouts

As noted above, query latency includes the time a query spends in the queue as well as the time it takes to execute.

Because a query's timeout period is generally measured starting from when it enters the queue, a slow-moving queue can make many queries time out as soon as they are dequeued. This is obviously undesirable, so it is good to avoid queuing up a large number of queries unless they can be executed rapidly.

Managing Your Amazon Neptune Database

This section shows how to manage and maintain your Neptune DB cluster using the AWS Management Console and the AWS CLI.

Neptune operates on clusters of database servers that are connected in a replication topology. Thus, managing Neptune often involves deploying changes to multiple servers and making sure that all Neptune replicas are keeping up with the primary server.

Because Neptune transparently scales the underlying storage as your data grows, managing Neptune requires relatively little management of disk storage. Likewise, because Neptune automatically performs continuous backups, a Neptune cluster does not require extensive planning or downtime for performing backups.

Topics

- [Amazon Neptune parameter groups \(p. 407\)](#)
- [Amazon Neptune parameters \(p. 409\)](#)
- [Launching a Neptune DB Cluster Using the Console \(p. 412\)](#)
- [Stopping and Starting an Amazon Neptune DB Cluster \(p. 416\)](#)
- [Empty an Amazon Neptune DB cluster using the fast reset API \(p. 419\)](#)
- [Creating a Neptune Replica Using the Console \(p. 423\)](#)
- [Modifying a Neptune DB Cluster Using the Console \(p. 424\)](#)
- [Adding Neptune Replicas to a DB Cluster \(p. 426\)](#)
- [Performance and Scaling in Amazon Neptune \(p. 428\)](#)
- [Auto-scaling the number of replicas in an Amazon Neptune DB cluster \(p. 429\)](#)
- [Maintaining an Amazon Neptune DB Cluster \(p. 433\)](#)
- [Amazon Neptune Updates \(p. 435\)](#)
- [Database Cloning in Neptune \(p. 437\)](#)
- [Managing Amazon Neptune Instances \(p. 441\)](#)

Amazon Neptune parameter groups

You manage your database configuration in Amazon Neptune by using parameters in a parameter group. Parameter groups act as a *container* for engine configuration values that are applied to one or more DB instances.

There are two types of parameter group, namely DB cluster parameter groups and DB parameter groups:

- *DB parameter groups* apply at the instance level and generally are associated with settings for the Neptune graph engine, such as the `neptune_query_timeout` parameter.
- *DB cluster parameter groups* apply to every instance in the cluster and generally have broader settings. Every Neptune cluster is associated with a DB cluster parameter group. Every DB instance within that cluster inherits the engine configuration values contained in the DB cluster parameter group.

Any configuration values that you modify in the DB cluster parameter group override default values in the DB parameter group. If you edit the corresponding values in the DB parameter group, those values override the settings in the DB cluster parameter group.

A default DB parameter group is used if you create a DB instance without specifying a custom DB parameter group. You can't modify the parameter settings of a default DB parameter group. You must create your own DB parameter group to change parameter settings from their default value. Not all DB engine parameters can be changed in a custom DB parameter group.

Here are some important points you should know about working with parameters in a DB parameter group:

- When you change a static parameter and save the instance DB parameter group, the parameter change takes effect after you manually reboot the DB instance.
- When you change a static parameter and save the DB cluster parameter group, the parameter change takes effect after you manually reboot every DB instance in the cluster.
- Improperly setting parameters in a DB parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying database parameters, and back up your data before modifying a DB parameter group. Try out your parameter group setting changes on a test DB instance before applying those changes to a production DB instance.

Editing a DB Cluster Parameter Group or DB Parameter Group

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Choose **Parameter groups** in the navigation pane.
3. Choose the **Name** link for the DB parameter group that you want to edit.

(Optional) Choose **Create parameter group** to create a new cluster parameter group and create the new group. Then choose the **Name** of the new parameter group.

Important

This step is *required* if you only have the default DB cluster parameter group because the default DB cluster parameter group can't be modified.

4. Choose **Edit parameters**.
5. Set the value for the parameters that you want to change.

6. Choose **Save changes**.
7. Reboot every Neptune DB instance in the Neptune cluster.

Creating a DB Cluster Parameter Group or DB Parameter Group

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Choose **Parameter groups** in the left navigation pane.
3. Choose **Create DB parameter group**.

The **Create DB parameter group** page appears.

4. In the **Type** list, choose **DB Parameter Group** or **DB Cluster Parameter Group**.
5. In the **Group name** box, type the name of the new DB parameter group.
6. In the **Description** box, type a description for the new DB parameter group.
7. Choose **Create**.

Amazon Neptune parameters

You manage your database configuration in Amazon Neptune by using parameters in [parameter groups](#) (p. 407). The following parameters are available for configuring your Neptune database:

Contents

- [neptune_enable_audit_log](#) (cluster-level parameter) (p. 409)
- [neptune_lab_mode](#) (cluster-level parameter) (p. 409)
- [neptune_query_timeout](#) (cluster-level parameter) (p. 409)
- [neptune_streams](#) (cluster-level parameter) (p. 409)
- [neptune_lookup_cache](#) (cluster-level parameter) (p. 410)
- [neptune_autoscaling_config](#) (cluster-level parameter) (p. 410)
- [neptune_ml_iam_role](#) (cluster-level parameter) (p. 410)
- [neptune_ml_endpoint](#) (cluster-level parameter) (p. 410)
- [neptune_query_timeout](#) (instance-level parameter) (p. 410)
- [neptune_result_cache](#) (instance-level parameter) (p. 411)
- [neptune_enforce_ssl](#) (DEPRECATED cluster-level parameter) (p. 411)

neptune_enable_audit_log (cluster-level parameter)

This parameter toggles audit logging for Neptune.

Allowed values are 0 (disabled) and 1 (enabled). The default value is 0.

neptune_lab_mode (cluster-level parameter)

When set, this parameter enables specific experimental features of Neptune. See [Neptune Lab Mode](#) (p. 85) for the experimental features currently available.

To enable or disable an experimental feature, include *(feature name)=enabled* or *(feature name)=disabled* in this parameter. You can enable or disable multiple features by separating them with commas, like this:

```
(feature #1 name)=enabled, (feature #2 name)=enabled
```

Lab mode features are typically disabled by default. An exception is the `DFEQueryEngine` feature, which became enabled by default for use with query hints (`DFEQueryEngine=viaQueryHint`) starting in [Neptune engine release 1.0.5.0](#) (p. 725).

neptune_query_timeout (cluster-level parameter)

Specifies a specific timeout duration for graph queries, in milliseconds.

Allowed values range from 10 to 2,147,483,647 ($2^{31} - 1$). The default value is 120,000 (2 minutes).

neptune_streams (cluster-level parameter)

Enables or disables [Neptune Streams](#) (p. 451).

Allowed values are 0 (disabled, which is the default), and 1 (enabled).

neptune_lookup_cache (cluster-level parameter)

Disables or re-enables the [Neptune lookup cache](#). (p. 64) on R5d instances.

Allowed values are `enabled` and `disabled`. The default value is `disabled`, but whenever an R5d instance is created in the DB cluster, the `neptune_lookup_cache` parameter is automatically set to `enabled` and a lookup cache is created on that instance.

neptune_autoscaling_config (cluster-level parameter)

Sets configuration parameters for the read-replica instances that [Neptune auto-scaling](#) (p. 429) creates and manages.

Using a JSON string that you set as the value of the `neptune_autoscaling_config` parameter, you can specify:

- The instance type that Neptune auto-scaling uses for all the new read-replica instances that it creates.
- The maintenance windows assigned to those read-replicas.
- Tags to be associated with all the new read-replicas.

The JSON string has a structure like this:

```
"{
  \"tags\": [
    { \"key\" : \"reader tag-0 key\", \"value\" : \"reader tag-0 value\" },
    { \"key\" : \"reader tag-1 key\", \"value\" : \"reader tag-1 value\" },
  ],
  \"maintenanceWindow\" : \"wed:12:03-wed:12:33\",
  \"dbInstanceClass\" : \"db.r5.xlarge\"
}"
```

Note that the quotation marks within the string must all be escaped with a backslash character (\).

Any of the three configuration settings not specified in the `neptune_autoscaling_config` parameter are copied from the configuration of the DB cluster's primary writer instance.

neptune_ml_iam_role (cluster-level parameter)

Specifies the IAM role ARN used in Neptune ML. The value can be any valid IAM role ARN.

You can specify the default IAM role ARN for machine learning on graphs.

neptune_ml_endpoint (cluster-level parameter)

Specifies the endpoint used for Neptune ML. The value can be any valid [SageMaker endpoint name](#).

You can specify the default SageMaker endpoint for machine learning on graphs.

neptune_query_timeout (instance-level parameter)

This DB instance parameter specifies a timeout duration for graph queries, in milliseconds, for one instance.

Allowed values range from 10 to 2,147,483,647 ($2^{31} - 1$). The default value is 120,000 (2 minutes).

neptune_result_cache (instance-level parameter)

neptune_result_cache – This DB instance parameter enables or disables [Caching query results](#) (p. 288).

Allowed values are 0, (disabled, which is the default), and 1 (enabled).

neptune_enforce_ssl (DEPRECATED cluster-level parameter)

(Deprecated) There used to be regions that permitted HTTP connections to Neptune, and this parameter was used to force all connections to use HTTPS when it was set to 1. This parameter is no longer relevant, however, since Neptune now only accepts HTTPS connections in all regions.

Launching a Neptune DB Cluster Using the Console

The following procedure describes how to use the AWS Management Console to launch an Amazon Neptune DB cluster.

Before you can access the Neptune console and create a Neptune cluster, you must have an IAM user with `NeptuneFullAccess` permissions. For information about adding these permissions, see [AWS Managed \(Predefined\) Policies for Amazon Neptune \(p. 118\)](#).

The `NeptuneFullAccess` IAM policy does not grant permissions for a virtual private cloud (VPC), VPC endpoints, or Amazon Simple Storage Service (Amazon S3). You must add these permissions to your IAM user or role to operate on other services. For example, if you choose to have the Neptune console create a VPC, the IAM user or role must have VPC permissions.

Note

Neptune requires permission to create a service-linked role the first time you create any Neptune resources. Add the following `iam:CreateServiceLinkedRole` permissions to the same user or role that you give `NeptuneFullAccess`.

```
{
  "Action": "iam:CreateServiceLinkedRole",
  "Effect": "Allow",
  "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
  "Condition": {
    "StringLike": {
      "iam:AWSServiceName": "rds.amazonaws.com"
    }
  }
}
```

For more information, see [Using Service-Linked Roles for Neptune \(p. 138\)](#).

To launch a Neptune DB cluster using the console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. Navigate to the **Databases** page.
3. Choose **Create database**.
4. On the **Specify DB details** page, under **Instance specifications** you can select a particular DB engine version for your new DB cluster. Unless you have a reason to use an older engine version, you can just leave the default value (the most recent version).
5. Under **Purpose**, choose either **Production** or **Development and Testing**.
6. If you selected **Production**, choose from among the available fixed-performance DB instance classes listed. The DB instance class you choose determines the processing and memory capacity of the primary write instance of your new DB cluster. For a current listing of DB instance classes that Neptune supports in different regions, see [the Neptune pricing page](#). In your region these may include:
 - `db.r5.large`
 - `db.r5.xlarge`
 - `db.r5.2xlarge`
 - `db.r5.4xlarge`
 - `db.r5.8xlarge`
 - `db.r5.12xlarge`
 - `db.r4.large`

- `db.r4.xlarge`
- `db.r4.2xlarge`
- `db.r4.4xlarge`
- `db.r4.8xlarge`

If you selected **Development and Testing**, you can select a T3 burstable instance class (for example, `db.t3.medium`). A burstable instance class is the most cost-effective for development, where you generally don't need high capacity all the time. See [T3 Burstable Instances \(p. 442\)](#).

- By default, Neptune will locate read-replica instances you create for a DB cluster in different availability zones (AZs) to improve availability. Unless you prefer not to do that, leave the **Enable high availability (Multi-AZ)** setting at the default (enabled).
- Under **Settings**, enter a name for the primary write instance in your DB cluster. This identifier is used in the endpoint address of the instance, and must meet the following criteria:
 - It must contain from 1 to 63 alphanumeric characters or hyphens.
 - Its first character must be a letter.
 - It cannot end with a hyphen or contain two consecutive hyphens.
 - It must be unique across all DB instances in your AWS account in a given AWS Region.
- Choose **Next**. On the **Configure advanced settings** page, you can customize additional settings for your Neptune DB cluster. The following table shows the advanced settings for a DB cluster.

For this option...	Do this
Virtual Private Cloud (VPC)	Choose the VPC that will host the DB cluster. Choose Create a new VPC to have Neptune create a VPC for you. You need to create an Amazon EC2 instance in this same VPC to access the Neptune instance. For more information, see Accessing Neptune DB Clusters in an Amazon VPC (p. 105) .
Subnet group	Choose the Neptune DB subnet group to use for the DB cluster. If your VPC does not have any subnet groups, Neptune creates a DB subnet group for you. For more information, see Accessing Neptune DB Clusters in an Amazon VPC (p. 105) .
Availability Zone	Specify a particular availability zone, or choose No preference to have Neptune choose one for you.
VPC security groups	Choose one or more VPC security groups to secure network access to the DB cluster. Choose Create a new VPC security group to have Neptune create a VPC security group for you. For more information, see Creating a Security Group to Provide Access to a Neptune DB Instance in a VPC (p. 107) .
DB cluster identifier	The identifier for your DB cluster. If you don't specify this value, Neptune creates one based on the DB instance identifier.
Port	The port for all HTTP and WebSockets connections. Neptune DB clusters use 8182 as the default.
DB parameter group	Select a parameter group. Neptune has a default parameter group you can use, or you can create your own

For this option...	Do this
	parameter group. For more information about parameter groups, see Parameters (p. 409) .
IAM DB authentication	Choose Enable IAM DB authentication to manage access and authentication using AWS Identity and Access Management (IAM). Important This requires that you sign all requests with AWS Signature Version 4 signing. For more information, see Identity and Access Management in Amazon Neptune (p. 111) .
Encryption	Choose Enable encryption if you want to enable encryption-at-rest for this DB cluster. For more information, see Encrypting Neptune Resources at Rest (p. 109) .
Failover priority	Choose a priority tier for failover. If there is contention within a tier, the replica that is the same size as the primary instance is selected.
Backup retention period	Choose the length of time, from 1 to 35 days, that Neptune will retain backup copies of the database. You can use backup copies for point-in-time restores (PITR) of your database down to the second.
Log exports	Choose the log types to publish to Amazon CloudWatch logs.
Auto minor version upgrade	Choose Enable auto minor version upgrade if you want to enable your Neptune DB cluster to receive minor Neptune DB Engine version upgrades automatically when they become available. The Auto minor version upgrade option applies only to upgrades to Neptune minor engine versions for your Amazon Neptune DB cluster, not to regular patches that are applied automatically to maintain system stability.
Maintenance window	If you want, you can choose a specific weekly time range during which system maintenance can occur.
Deletion protection	Deletion protection blocks your DB cluster from being deleted. Unless you uncheck Enable deletion protection , it is enabled by default.

- Choose **Create database** to launch your Neptune DB instance, and then choose **Close** to close the wizard.

On the Amazon Neptune console, the new DB cluster appears in the list of Databases. The DB cluster has a status of **Creating** until it is created and ready for use. When the state changes to **Available**, you can connect to the primary instance for your DB cluster. Depending on the DB instance class and store allocated, it can take several minutes for the new instances to be available.

To view the newly created cluster, choose the **Databases** view in the Neptune console.

Note

If you delete all Neptune DB instances in a DB cluster using the AWS Management Console, the DB cluster is deleted automatically. If you are using the AWS CLI or SDK, you must delete the DB cluster manually after you delete the last instance.

Note the **Cluster endpoint** value. You need this to connect to your Neptune DB cluster.

Stopping and Starting an Amazon Neptune DB Cluster

Stopping and starting Amazon Neptune clusters helps you manage costs for development and test environments. You can temporarily stop all the DB instances in your cluster, instead of setting up and tearing down all the DB instances each time that you use the cluster.

Topics

- [Overview of Stopping and Starting a Neptune DB Cluster \(p. 416\)](#)
- [Stopping a Neptune DB Cluster \(p. 416\)](#)
- [Starting a Stopped Neptune DB Cluster \(p. 417\)](#)

Overview of Stopping and Starting a Neptune DB Cluster

During periods where you don't need a Neptune cluster, you can stop all instances in that cluster at once. You can start the cluster again anytime you need to use it. Starting and stopping simplifies the setup and teardown processes for clusters used for development, testing, or similar activities that don't require continuous availability. You can accomplish this in the AWS Management Console with a single action, regardless of how many instances there are in the cluster.

While your DB cluster is stopped, you are charged only for cluster storage, manual snapshots, and automated backup storage within your specified retention window. You aren't charged for any DB instance hours.

After seven days, Neptune automatically starts your DB cluster again to make sure that it doesn't fall behind any required maintenance updates.

To minimize charges for a lightly loaded Neptune cluster, you can stop the cluster instead of deleting all its read replicas. For clusters with more than one or two instances, frequently deleting and recreating the DB instances is only practical using the AWS CLI or Neptune API, and deletions can also be difficult to perform in the right order. For example, you must delete all read replicas before deleting the primary instance to avoid activating the failover mechanism.

Don't use starting and stopping if you need to keep your DB cluster running but you want to reduce capacity. If your cluster is too costly or not very busy, you can delete one or more DB instances or change your DB instances to use a smaller instance class, but you can't stop an individual DB instance.

Stopping a Neptune DB Cluster

When you won't be using it for awhile, you can stop a running Neptune DB cluster, and then start it again when you need it. While the cluster is stopped you are charged for cluster storage, manual snapshots, and automated backup storage within your specified retention window, but not for DB instance hours.

The stop operation stops all the cluster's read replica instances before stopping the primary instance, to avoid activating the failover mechanism.

Stopping a DB Cluster Using the AWS Management Console

To use the AWS Management Console to stop a Neptune cluster

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.

2. In the navigation pane, choose **Databases**, and then choose a cluster. You can perform the stop operation from this page, or navigate to the details page for the DB cluster that you want to stop.
3. In **Actions**, choose **Stop**.

Stopping a DB Cluster Using the AWS CLI

To stop a DB instance by using the AWS CLI, call the [stop-db-cluster \(p. 813\)](#) command, using the `--db-cluster-identifier` parameter to identify the DB cluster you want to stop.

Example

```
aws neptune stop-db-cluster --db-cluster-identifier mydbcluster
```

Stopping a DB Cluster Using the Neptune Management API

To stop a DB instance by using the Neptune Management API, call the [StopDBCluster \(p. 813\)](#) API and use the `DBClusterIdentifier` parameter to identify the DB cluster you want to stop.

What Can Happen While a DB Cluster Is Stopped

- You **can** restore it from a snapshot (see [Restoring from a DB Cluster Snapshot \(p. 679\)](#)).
- You **can't** modify the configuration of the DB cluster or any of its DB instances.
- You **can't** add or remove DB instances from the cluster.
- You **can't** delete the cluster if it still has any associated DB instances.
- In general, you must re-start a stopped DB cluster to perform most administrative actions.
- Neptune applies any scheduled maintenance to your stopped cluster as soon as it is started again. Remember that after seven days, Neptune automatically re-starts a stopped cluster so that it doesn't fall too far behind in maintenance status.
- Neptune does not perform any automated backups of a stopped DB cluster, because the underlying data cannot change while the cluster is stopped.
- Neptune does not extend the backup retention period for the DB cluster while it is stopped.

Starting a Stopped Neptune DB Cluster

You can only start a Neptune DB cluster that is in the stopped state. When you start the cluster, all its DB instances become available again. The cluster retains its configuration settings, such as endpoints, parameter groups and VPC security groups.

Starting a Stopped DB Cluster Using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Databases**, and then choose a cluster. You can perform the start operation from this page, or navigate to the details page for that DB cluster and start from there.
3. In **Actions**, choose **Start**.

Starting a Stopped DB Cluster Using the AWS CLI

To start a stopped DB cluster using the AWS CLI, call the [start-db-cluster \(p. 810\)](#) command using the `--db-cluster-identifier` parameter to specify the stopped DB cluster that you want to start. Provide either the cluster name that you chose when creating the DB cluster, or use a DB instance name that you chose with `-cluster` appended to the end of it.

Example

```
aws neptune start-db-cluster --db-cluster-identifier mydbcluster
```

Starting a Stopped DB Cluster Using the Neptune Management API

To start a Neptune DB cluster by using the Neptune management API, call the [StartDBCluster \(p. 810\)](#) API using the `DBCluster` parameter to specify the stopped DB cluster that you want to start. Provide either the cluster name that you chose when creating the DB cluster, or use a DB instance name that you chose, with `-cluster` appended to the end of it.

Empty an Amazon Neptune DB cluster using the fast reset API

The Neptune fast reset REST API lets you reset a Neptune graph quickly and easily, removing all of its data. You can also do this within a Neptune notebook using the [%db_reset \(p. 421\)](#) line magic.

Note

This feature is available starting in [Neptune engine release 1.0.4.0 \(p. 739\)](#).

- In most cases, a fast reset operation completes within couple of minutes. The duration can vary somewhat depending on the load on the cluster when the operation is initiated.
- A fast reset operation does not result in additional I/Os.
- Storage volume size does not shrink after a fast reset. Instead, the storage is reused as new data is inserted. This means that the volume sizes of snapshots taken before and after a fast reset operation will be the same. Volume sizes of restored clusters using the snapshots created before and after a fast reset operation will also be the same
- As part of the reset operation, all instances in the DB cluster are restarted.

Note

Under rare conditions, these server restarts may also result in failover of the cluster.

Important

Using fast reset may break the integration of your Neptune DB cluster with other services. For example:

- Fast reset deletes all stream data from your database and completely resets streams. This means that your stream consumers may no longer work without new configuration.
- Fast reset removes all metadata about SageMaker resources being used by Neptune ML, including jobs and endpoints. They continue to exist in SageMaker, and you can continue to use existing SageMaker endpoints for Neptune ML inference queries, but the Neptune ML management APIs no longer work with them.
- Integrations such as the full-text-search integration with Elasticsearch are also wiped out by fast reset, and must be re-established manually before they can be used again.

To delete all data from a Neptune DB cluster using the API

1. First, you generate a token that you can then use to perform the database reset. This step is intended to help prevent anyone from accidentally resetting a database.

You do this by sending an HTTP `POST` request to the `/system` endpoint on the writer instance of your DB cluster to specify the `initiateDatabaseReset` action.

The `curl` command using the JSON content-type would be:

```
curl -X POST \  
-H 'Content-Type: application/json' \  
  https://your_writer_instance_endpoint:8182/system \  
-d '{"action" : "initiateDatabaseReset" }'
```

Or, using the `x-www-form-urlencoded` content type:

```
curl -X POST \  
-H 'Content-Type: application/x-www-form-urlencoded' \  
  https://your_writer_instance_endpoint:8182/system \  
-d 'action=initiateDatabaseReset'
```

```
https://your_writer_instance_endpoint:8182/system \
-d 'action=initiateDatabaseReset '
```

The `initiateDatabaseReset` request returns the reset token in its JSON response, like this:

```
{
  "status" : "200 OK",
  "payload" : {
    "token" : "new_token_guid"
  }
}
```

The token remains valid for one hour (60 minutes) after it is issued.

If you send the request to a reader instance or to the status endpoint, Neptune will throw a `ReadOnlyViolationException`.

If you send multiple `initiateDatabaseReset` requests, only the latest token generated will be valid for the second step, where you actually perform the reset.

If the server restarts right after your `initiateDatabaseReset` request, the generated token becomes invalid, and you need to send a new request to get a new token.

2. Next, you send a `performDatabaseReset` request with the token that you got back from `initiateDatabaseReset` to the `/system` endpoint on the writer instance of your DB cluster. This deletes all data from your DB cluster.

The `curl` command using the JSON content-type is:

```
curl -X POST \
-H 'Content-Type: application/json' \
  https://your_writer_instance_endpoint:8182/system \
-d '{
  "action" : "performDatabaseReset",
  "token" : "token_guid"
}'
```

Or, using the `x-www-form-urlencoded` content type:

```
curl -X POST \
-H 'Content-Type: application/x-www-form-urlencoded' \
  https://your_writer_instance_endpoint:8182/system \
-d 'action=performDatabaseReset&token=token_guid'
```

The request returns a JSON response. If the request is accepted, the response is:

```
{
  "status" : "200 OK"
}
```

If the token you sent doesn't match the one that was issued, the response looks like this:

```
{
  "code" : "InvalidParameterException",
  "requestId": "token_guid",
  "detailedMessage" : "System command parameter 'token' : 'token_guid' does not match database reset token"
}
```

If the request is accepted and the reset begins, the server restarts and deletes the data. You cannot send any other requests to the DB cluster while it is resetting.

Using the fast reset API with IAM-Auth

If you have IAM-Auth enabled on your DB cluster, you can use `awscli` to send fast reset commands that are authenticated using IAM-Auth:

Using `awscli` to send fast-reset requests with IAM-Auth

1. Set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables correctly (and also `AWS_SECURITY_TOKEN` if you are using temporary credential).
2. An `initiateDatabaseReset` request looks like this:

```
awscli -X POST --service neptune-db "$SYSTEM_ENDPOINT" \  
-H 'Content-Type: application/json' --region us-west-2 \  
-d '{ "action" : "initiateDatabaseReset" }'
```

3. A `performDatabaseReset` request looks like this:

```
awscli -X POST --service neptune-db "$SYSTEM_ENDPOINT" \  
-H 'Content-Type: application/json' --region us-west-2 \  
-d '{ "action" : "performDatabaseReset" }'
```

Using the Neptune workbench `%db_reset` line magic to reset a DB cluster

The Neptune workbench supports a `%db_reset` line magic that lets you perform a fast database reset in a Neptune notebook.

If you invoke the magic without any parameters, you see a screen asking if you want to delete all the data in your cluster, with a checkbox asking you to acknowledge that the cluster data will no longer be available after you delete it. At that point, you can choose to go ahead and delete the data, or cancel the operation.

A more dangerous option is to invoke `%db_reset` with the `--yes` or `-y` option, which causes the deletion to be performed with no further prompting.

You can also perform the reset in two steps, just as with the REST API:

```
%db_reset --generate-token
```

The response is:

```
{  
  "status" : "200 OK",  
  "payload" : {  
    token : "new_token_guid"  
  }  
}
```

Then do:

```
%db_reset --token "new_token_guid"
```

The response is:

```
{
  "status" : "200 OK"
}
```

Common error codes for fast reset operations

Neptune error code	HTTP status	Message	Example
InvalidParameterException	400	System command parameter ' <i>action</i> ' has unsupported value ' <i>xxx</i> '	Invalid parameter
InvalidParameterException	400	Too many values supplied for: <i>action</i>	A fast reset request with more than one action sent with header 'Content-type:application/x-www-form-urlencoded'
InvalidParameterException	400	Duplicate field 'action'	A fast reset request with more than one action sent with header 'Content-Type: application/json'
MethodNotAllowedException	400	Bad route: / <i>bad_endpoint</i>	Request sent to an incorrect endpoint
MissingParameterException	400	Missing required parameters: [action]	A fast reset request doesn't contain the required 'action' parameter
ReadOnlyViolationException	400	Writes are not permitted on a read replica instance	A fast reset request was sent to a reader or status endpoint
AccessDeniedException	403	Missing Authentication Token	A fast reset request was sent without correct signatures to a DB endpoint with IAM-Auth enabled
ServerShutdownException	500	Database reset is in progress. Please retry the query after the cluster is available.	When fast reset begins, existing and incoming Gremlin/Sparql queries fail.

Creating a Neptune Replica Using the Console

After creating the primary instance for your Neptune DB cluster, you can add additional Neptune replicas by using the **Create read replica** page on the Amazon Neptune console.

To create a Neptune replica using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**.
3. Select the check box to the left of the primary instance for your Neptune DB cluster.
4. Choose **Instance Actions**, and then choose **Create read replica**.
5. On the **Create replica DB instance** page, specify options for your Neptune replica. The following table shows settings for a Neptune read replica.

For This Option...	Do This
Availability zone	Specify an Availability Zone. Choose a different zone than the primary DB instance. The list includes only those Availability Zones that are mapped by the DB subnet group for the DB cluster.
Encryption	Enable or disable encryption.
DB instance class	Choose a DB instance class that defines the processing and memory requirements for the Neptune replica. For a current listing of the DB instance classes that Neptune offers in different regions, see the Neptune pricing page .
Read replica source	Choose the identifier of the primary instance to create a Neptune replica for.
DB instance identifier	Enter a name for the instance that is unique for your account in the Region that you selected. You might choose to add some intelligence to the name, such as including the Availability Zone selected, for example <code>neptune-us-east-1c</code> .
DB parameter group	The parameter group for this instance.
Auto Minor Version Upgrade	<p>Choose Yes if you want to enable your Neptune replica to receive minor Neptune DB engine version upgrades automatically when they become available.</p> <p>The Auto Minor Version Upgrade option applies only to minor upgrades. It does not apply to engine maintenance patches, which are always applied automatically to maintain system stability.</p>

6. Choose **Create read replica** to create the Neptune replica instance.

Modifying a Neptune DB Cluster Using the Console

When you modify a DB instance using the AWS Management Console, you can choose to apply the changes right away by selecting **Apply Immediately**. If you choose to apply changes immediately, your new changes and any changes in the pending modifications queue are applied at once.

If you don't choose to apply changes immediately, the changes are put into the pending modifications queue. During the next maintenance window, any pending changes in the queue are applied.

Important

If any pending modifications require downtime, choosing to apply changes immediately can cause unexpected downtime for the DB instance in question. There is no downtime for the other DB instances in the DB cluster.

Note

When you modify a DB cluster in Neptune, the **Apply Immediately** setting only affects changes to the **DB cluster identifier**, **IAM DB authentication**. All other modifications are applied immediately, regardless of the value of the **Apply Immediately** setting.

To modify a DB cluster using the console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Clusters**, and then choose the DB cluster that you want to modify.
3. Choose **Actions**, and then choose **Modify cluster**. The **Modify DB cluster** page appears.
4. Change any of the settings that you want.

Note

On the console, some instance level changes only apply to the current DB instance, whereas others apply to the entire DB cluster. To change a setting that modifies the entire DB cluster at the instance level on the console, follow the instructions in [Modifying a DB Instance in a DB Cluster \(p. 424\)](#).

5. When all the changes are as you want them, choose **Continue** and check the summary.
6. To apply the changes immediately, select **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify cluster** to save your changes.

To edit your changes, choose **Back**, or to cancel your changes, choose **Cancel**.

Modifying a DB Instance in a DB Cluster

To modify a DB Instance in a DB cluster using the console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**, and then choose the DB instance that you want to modify.
3. Choose **Instance actions**, and then choose **Modify**. The **Modify DB Instance** page appears.
4. Change any of the settings that you want.

Note

Some settings apply to the entire DB cluster and must be changed at the cluster level. To change those settings, follow the instructions in [Modifying a Neptune DB Cluster Using the Console \(p. 424\)](#).

In the AWS Management Console, some instance-level changes apply only to the current DB instance, whereas others apply to the entire DB cluster.

5. When all the changes are as you want them, choose **Continue** and check the summary.
6. To apply the changes immediately, select **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

To edit your changes, choose **Back**, or to cancel your changes, choose **Cancel**.

Adding Neptune Replicas to a DB Cluster

In Neptune DB clusters, there is one primary DB instance and up to 15 Neptune replicas. The primary DB instance supports read and write operations, and performs all of the data modifications to the cluster volume. Neptune replicas connect to the same storage volume as the primary DB instance and support only read operations. Neptune replicas can offload read workloads from the primary DB instance.

We recommend that you distribute the primary instance and Neptune replicas in your DB cluster over multiple Availability Zones to improve the availability of your DB cluster.

For more information about Neptune replicas, see [Creating a Neptune Replica Using the Console \(p. 423\)](#).

To add a Neptune replica to a DB cluster

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**, and then choose a DB instance in the DB cluster. The selected DB instance can be the primary DB instance for the DB cluster or a Neptune replica.
3. Choose **Instance actions**, and then choose **Create Neptune replica**.

The **Create Neptune replica** page appears.

4. On the **Create Neptune replica** page, specify options for your Neptune replica. The following table shows settings for a Neptune replica.

For This Option	Do This
Availability Zone	Determine if you want to specify a particular Availability Zone. The list includes only those Availability Zones that are mapped by the DB subnet group you specified earlier.
Encryption	Select Enable encryption to enable encryption at rest for this Neptune replica. For more information, see Encrypting Neptune Resources at Rest (p. 109) .
DB instance class	Choose a DB instance class that defines the processing and memory requirements for the Neptune replica.
Neptune replica source	Select the identifier of the primary instance to create a Neptune replica for.
DB instance identifier	Enter a name for the instance that is unique for your account in the AWS Region that you selected. You might choose to add some intelligence to the name, such as including the AWS Region and DB engine that you selected (for example gremlin-read-instance1).
Priority	Choose a failover priority for the instance. If you don't choose a value, the default is tier-1 . This priority determines the order in which Neptune replicas are promoted when recovering from a primary instance failure. For more information, see Fault Tolerance for a Neptune DB Cluster (p. 674) .
Database port	The port for a Neptune replica is the same as the port for the DB cluster. The default is 8182.

For This Option	Do This
DB parameter group	Choose a parameter group. Neptune has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see Amazon Neptune parameters (p. 409) .
Auto minor version upgrade	Select Enable auto minor version upgrade if you want to enable your Neptune DB cluster to receive minor engine version upgrades automatically when they become available. The <i>Auto minor version upgrade</i> option only applies to upgrades to minor engine versions for your Amazon Neptune DB cluster. It doesn't apply to regular patches applied to maintain system stability.

5. Choose **Create Neptune replica** to create the Neptune replica.

To remove a Neptune replica from a DB cluster, follow the instructions in [Deleting a DB Instance in Amazon Neptune \(p. 449\)](#) to delete the Neptune replica DB instance.

Performance and Scaling in Amazon Neptune

Neptune DB clusters and instances scale at three different levels:

- [Storage Scaling \(p. 428\)](#)
- [Instance Scaling; \(p. 428\)](#)
- [Read Scaling \(p. 428\)](#)

Storage Scaling in Neptune

Neptune storage automatically scales with the data in your cluster volume. As your data grows, your cluster volume storage grows up to 64 TiB.

The size of your cluster volume is checked on an hourly basis to determine your storage costs.

Storage consumed by your Neptune database is billed in per GB-month increments and I/Os consumed are billed in per million request increments. You pay only for the storage and I/Os that your Neptune database consumes, and you don't need to provision in advance.

For pricing information, see the [Neptune product page](#).

Instance Scaling in Neptune

You can scale your Neptune DB cluster as needed by modifying the DB instance class for each DB instance in the DB cluster. Neptune supports several optimized DB instance classes.

Read Scaling in Neptune

You can achieve read scaling for your Neptune DB cluster by creating up to 15 Neptune replicas in the DB cluster. Each Neptune replica returns the same data from the cluster volume with minimal replica lag (often considerably less than 100 milliseconds after the primary instance has written an update). As your read traffic increases, you can create additional Neptune replicas and connect to them directly to distribute the read load for your DB cluster. Neptune replicas don't have to be of the same DB instance class as the primary instance.

For information about adding Neptune replicas to a DB cluster, see [Adding Neptune Replicas \(p. 423\)](#).

Auto-scaling the number of replicas in an Amazon Neptune DB cluster

You can use Neptune auto-scaling to automatically adjust the number of Neptune replicas in a DB cluster, to meet your connectivity and workload requirements. Auto-scaling lets your Neptune DB cluster handle sudden increases in workload, and then, when the workload decreases, auto-scaling removes unnecessary replicas so you aren't paying for unused capacity.

You can only use auto-scaling with a Neptune DB cluster that already has one primary writer instance and at least one read-replica instance (see [Amazon Neptune DB Clusters and Instances \(p. 73\)](#)). Also, all read-replica instances in the cluster must be in an available state. If any read-replica is in a state other than available, Neptune autoscaling does nothing until every read-replica in the cluster is available.

See [Creating a DB Cluster \(p. 166\)](#) if you need to create a new cluster.

Using the AWS CLI, you define and apply a scaling policy to the DB cluster. The scaling policy specifies the following auto-scaling parameters:

- The minimum and maximum number of replicas to have in the cluster.
- A so-called *cool-down* interval between replica additions or deletions.
- The CloudWatch metric and the metric trigger value for scaling up or down.

Using the AWS CLI, you can also edit or delete your auto-scaling policy.

When the CloudWatch metric you're using reaches the threshold you specified in your policy, as long as your DB cluster doesn't already have the maximum number of replicas that you set, Neptune creates a new replica using the same instance type as the DB cluster's primary instance.

When the metric shows that the workload has decreased sufficiently, Neptune auto-scaling removes unnecessary replicas so that you don't pay for capacity you aren't using.

Note

Neptune auto-scaling only removes replicas that it created. It does not remove pre-existing replicas.

Using the [neptune_autoscaling_config \(p. 410\)](#) DB cluster parameter, you can also specify the instance type of the new read-replicas that Neptune auto-scaling creates, the maintenance windows for those read-replicas, and tags to be associated with each of the new read-replicas. You provide these configuration settings in a JSON string as the value of the `neptune_autoscaling_config` parameter, like this:

```
"{
  \"tags\": [
    { \"key\" : \"reader tag-0 key\", \"value\" : \"reader tag-0 value\" },
    { \"key\" : \"reader tag-1 key\", \"value\" : \"reader tag-1 value\" },
  ],
  \"maintenanceWindow\" : \"wed:12:03-wed:12:33\",
  \"dbInstanceClass\" : \"db.r5.xlarge\"
}"
```

Note that the quotation marks in the JSON string must all be escaped with a backslash character (\). All whitespace in the string is optional, as usual.

Any of the three configuration settings not specified in the `neptune_autoscaling_config` parameter are copied from the configuration of the DB cluster's primary writer instance.

When Application Auto Scaling API Reference auto-scaling adds a new read-replica instance, it prefixes the DB instance ID with `autoscaled-reader` (for example, `autoscaled-reader-7r7t7z3lbd-20210828`). It also adds the a tag to every read-replica that it creates with the key `autoscaled-reader` and a value of `TRUE`. You can see this tag on the **Tags** tab of the DB instance detail page in the AWS Management Console.

```
"key" : "autoscaled-reader", "value" : "TRUE"
```

The promotion tier of all the read-replica instances created by auto-scaling is the lowest priority, which is 15 by default. This means that during a failover, any replica a higher priority, such as one that was created manually, would be promoted first. See [Fault Tolerance for a Neptune DB Cluster \(p. 674\)](#).

Neptune auto-scaling is implemented using Application Auto Scaling with a [target tracking scaling policy](#) that uses a Neptune [CPU Utilization \(p. 644\)](#) CloudWatch metric as a predefined metric.

How to enable auto-scaling for Amazon Neptune

Enabling auto-scaling for a Neptune DB cluster involves three steps:

1. Register your DB cluster with Application Auto Scaling

The first step in enabling auto-scaling for a Neptune DB cluster is to register the cluster with Application Auto Scaling, using the AWS CLI or one of the Application Auto Scaling SDKs. The cluster must already have one primary instance and at least one read-replica instance:

For example, to register a cluster to be auto-scaled with from one to eight additional replicas, you could use the AWS CLI `register-scalable-target` command as follows:

```
aws application-autoscaling register-scalable-target \  
  --service-namespace neptune \  
  --resource-id cluster:(your DB cluster name) \  
  --scalable-dimension neptune:cluster:ReadReplicaCount \  
  --min-capacity 1 \  
  --max-capacity 8
```

This is equivalent to using the the `RegisterScalableTarget` Application Auto Scaling API operation.

The AWS CLI `register-scalable-target` command takes the following parameters:

- **--service-namespace** – Set to `neptune`.

This parameter is equivalent to the `ServiceNamespace` parameter in the Application Auto Scaling API.

- **resource-id** – Set this to the resource identifier for your Neptune DB cluster. The resource type is `cluster`, which is followed by a colon (':'), and then the name of your DB cluster.

This parameter is equivalent to the `ResourceID` parameter in the Application Auto Scaling API.

- **scalable-dimension** – The scalable dimension in this case is the number of replica instances in the DB cluster, so you set this parameter to `neptune:cluster:ReadReplicaCount`.

This parameter is equivalent to the `ScalableDimension` parameter in the Application Auto Scaling API.

- **min-capacity** – The minimum number of reader DB replica instances to be managed by Application Auto Scaling. This can be as little as 0 and no larger than the value of the `max-capacity` parameter.

This parameter is equivalent to the `MinCapacity` parameter in the Application Auto Scaling API.

- **max-capacity** – The maximum number of reader DB replica instances to be managed by Application Auto Scaling. This value can be no smaller than the value of the `min-capacity` parameter, and no larger than 15 minus the number of pre-existing replicas. A DB cluster can't have more than 15 replicas in total, so the maximum number of replicas managed by auto-scaling plus the number of pre-existing replicas can't exceed that limit.

The `max-capacity` AWS CLI parameter is equivalent to the `MaxCapacity` parameter in the Application Auto Scaling API.

When you register your DB cluster, Application Auto Scaling creates an `AWSServiceRoleForApplicationAutoScaling_NeptuneCluster` service-linked role. For more information, see [Service-linked roles for Application auto-scaling](#) in the *Application Auto Scaling User Guide*.

2. Define an autoscaling policy to use with your DB cluster

A target-tracking scaling policy is defined as a JSON text object that can also be saved in a text file. For Neptune this policy currently can only use the Neptune [CPUUtilization \(p. 644\)](#) CloudWatch metric as a predefined metric named `NeptuneReaderAverageCPUUtilization`.

Here is an example target tracking scaling configuration policy for Neptune:

```
{
  "PredefinedMetricSpecification": { "PredefinedMetricType":
    "NeptuneReaderAverageCPUUtilization" },
  "TargetValue": 85.0,
  "ScaleInCooldown" : 600,
  "ScaleOutCooldown" : 600,
}
```

The **TargetValue** element here contains the percentage of CPU utilization above which auto-scaling *scales out* (that is, adds more replicas) and below which it *scales in* (that is, deletes replicas). In this case, the target percentage that triggers scaling is 85.0%.

The **ScaleInCooldown** element specifies the amount of time, in seconds, after a scale-in activity completes before another scale-in can start. The default is 300 seconds. Here, the value of 600 specifies that at least ten minutes must elapse between the completion of one replica deletion and the start of another one.

The **ScaleOutCooldown** element specifies the amount of time, in seconds, after a scale-out activity completes before another scale-out can start. The default is 300 seconds. Here, the value of 600 specifies that at least ten minutes must elapse between the completion of one replica addition and the start of another one.

The **DisableScaleIn** element is a Boolean that if present and set to `true` disables scale-in entirely, meaning that auto-scaling may add replicas but will never remove any. By default, scale-in is enabled, and `DisableScaleIn` is `false`.

After registering your Neptune DB cluster with Application Auto Scaling and defining a JSON scaling policy in a text file, next apply the scaling policy to the registered DB cluster. You can use the AWS CLI `put-scaling-policy` command to do this, with parameters like the following:

```
aws application-autoscaling put-scaling-policy \
  --policy-name (name of the scaling policy) \
  --policy-type TargetTrackingScaling \
  --resource-id cluster:(name of your Neptune DB cluster) \
  --service-namespace neptune \
```



```
--scalable-dimension neptune:cluster:ReadReplicaCount \  
--target-tracking-scaling-policy-configuration (file where you saved the policy)
```

When you have applied the auto-scaling policy, auto-scaling is enabled on your DB cluster.

You can also use the AWS CLI `put-scaling-policy` command to update an existing auto-scaling policy.

See also `PutScalingPolicy` in the *Application Auto Scaling API Reference*.

Removing auto-scaling from a Neptune DB cluster

To remove auto-scaling from a Neptune DB cluster, use the AWS CLI `delete-scaling-policy` and `deregister-scalable-target` commands.

Maintaining an Amazon Neptune DB Cluster

Periodically, Neptune performs maintenance on Neptune resources. Maintenance most often involves updates to the DB cluster's underlying operating system or database engine version. Updates to the operating system most often occur for security issues and should be done as soon as possible. You can determine which engine release version you currently have installed by using the [instance-status API](#) (p. 639).

Maintenance items require that Neptune take your DB cluster offline for a short time. Maintenance that requires a resource to be offline include scale compute operations (which generally take only a few minutes from start to finish), and required operating system or database patching. Required patching is automatically scheduled only for patches that are related to security and instance reliability. Such patching occurs infrequently (typically once every few months) and seldom requires more than a fraction of your maintenance window.

DB instances are not automatically backed up when an OS update is applied. So you should back up your DB instances before you apply an update.

You can view whether a maintenance update is available for your DB cluster using the Neptune console. If an update is available, it is indicated by the word **Available** or **Required** in the **Maintenance** column for the DB cluster on the Neptune console.

If an update is available, you can do any of the following:

- Defer the maintenance items.
- Apply the maintenance items immediately.
- Schedule the maintenance items to start during your next maintenance window.
- Take no action.

Note

Certain operating system updates are marked as **Required**. If you defer a required update, you receive a notice from Neptune indicating when the update will be performed. Other updates are marked as **Available**, and these you can defer indefinitely.

The maintenance window determines when pending operations start, but it does not limit the total execution time of these operations. Maintenance operations are not guaranteed to finish before the maintenance window ends, and can continue beyond the specified end time. For more information, see [Neptune Maintenance Window](#) (p. 433).

Neptune Maintenance Window

Every DB cluster has a weekly maintenance window during which any system changes are applied. You can think of the maintenance window as an opportunity to control when modifications and software patching occur, in the event either are requested or required. If a maintenance event is scheduled for a given week, it is initiated during the 30-minute maintenance window that you identify. Most maintenance events also complete during the 30-minute maintenance window, although larger maintenance events might sometimes take more than 30 minutes to complete.

The 30-minute maintenance window is selected at random from an 8-hour block of time per Region. If you don't specify a preferred maintenance window when you create the DB cluster, Neptune assigns a 30-minute maintenance window on a randomly selected day of the week.

Neptune consumes some of the resources on your DB cluster while maintenance is being applied. You might observe a minimal effect on performance. For a DB instance, on rare occasions, a Multi-AZ failover might be required for a maintenance update to complete.

Here are the time blocks for each AWS Region from which default maintenance windows are assigned:

Region	Time Block
US West (Oregon) Region	06:00–14:00 UTC
US West (N. California) Region	06:00–14:00 UTC
US East (Ohio) Region	03:00–11:00 UTC
Europe (Ireland) Region	22:00–06:00 UTC

Adjusting the Preferred DB Cluster Maintenance Window Using the Console

The Neptune DB cluster maintenance window should fall at the time of lowest usage. Thus you might need to change it from time to time. Your DB cluster is unavailable during this time only if the updates that are being applied require an outage. The outage is for the minimum amount of time required to make the necessary updates.

To adjust the preferred DB cluster maintenance window

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Clusters**.
3. Choose the DB cluster for which you want to change the maintenance window.
4. Choose **Actions, Modify cluster**.
5. In the **Maintenance** section, update the maintenance window.
6. Choose **Continue**.

On the confirmation page, review your changes.

7. To apply the changes to the maintenance window immediately, select **Apply immediately**.
8. Choose **Modify cluster** to save your changes.

To edit your changes, choose **Back**, or to cancel your changes, choose **Cancel**.

Amazon Neptune Updates

Amazon Neptune releases database engine updates regularly. Customers can choose to apply these updates manually or automatically depending on the kind of update. During the maintenance window when updates are applied, the cluster writer and reader instances are not available to process user queries. Most maintenance events take at most 30 minutes to complete. To configure your cluster's maintenance window, see [Neptune Maintenance Window \(p. 433\)](#).

Past engine releases are listed at [Engine Releases for Amazon Neptune \(p. 718\)](#).

Note that the way Neptune numbers and handles updates changed in November 2019, as described in [Engine Version Numbers \(p. 100\)](#).

You can use the console to update the engine version manually, to set the `AutoMinorVersionUpgrade` field to `true` so that Neptune automatically updates the engine to the latest version whenever possible.

Using the Console to Upgrade Your Engine to a More Recent Version

You can perform a minor version upgrade of a Neptune DB cluster using the AWS Management Console as follows:

To upgrade the engine version of a DB cluster using the console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Clusters**, and then choose the DB cluster that you want to modify.
3. Choose **Actions**, and then choose **Modify cluster**. The **Modify DB cluster** page appears.
4. For **DB engine version**, choose the new version.
5. Choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, choose **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify Cluster** to save your changes.

Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

Setting Up Neptune to Upgrade the Engine Version Automatically

If you set the `AutoMinorVersionUpgrade` field to `true` in the writer DB instance of a DB cluster, Neptune will automatically update the engine of the cluster to the latest version whenever possible thereafter.

You can use the AWS Management Console to set this field:

To set the `AutoMinorVersionUpgrade` field of a DB cluster

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the list of DB instances, choose the primary instance (the writer instance) of the DB cluster.
3. Choose **Instance actions**, and then choose **Modify**. The **Modify DB Instance** page appears.

4. Choose **Actions**, and then choose **Modify instance**. The **Modify DB instance** page appears.
5. Set the auto minor version update field.

See [Modifying an Instance \(p. 444\)](#) for more information.

Database Cloning in Neptune

Using DB cloning, you can quickly and cost-effectively create clones of all your databases in Amazon Neptune. The clone databases require only minimal additional space when they are first created. Database cloning uses a *copy-on-write protocol*. Data is copied at the time that it changes, either on the source databases or the clone databases. You can make multiple clones from the same DB cluster. You can also create additional clones from other clones. For more information about how the copy-on-write protocol works in the context of Neptune storage, see [Copy-on-Write Protocol \(p. 438\)](#).

You can use DB cloning in a variety of use cases, especially where you don't want to have an impact on your production environment, such as the following:

- Experiment with and assess the impact of changes, such as schema changes or parameter group changes.
- Perform workload-intensive operations, such as exporting data or running analytical queries.
- Create a copy of a production DB cluster in a non-production environment for development or testing.

To create a clone of a DB cluster using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**. Choose the primary instance for the DB cluster that you want to create a clone of.
3. Choose **Instance actions**, and then choose **Create clone**.
4. On the **Create Clone** page, enter a name for the primary instance of the clone DB cluster as the **DB instance identifier**.

If you want to, configure any other settings for the clone DB cluster. For information about the different DB cluster settings, see [Launching a Cluster \(p. 412\)](#).

5. Choose **Create Clone** to launch the clone DB cluster.

To create a clone of a DB cluster using the AWS CLI

- Call the Neptune [restore-db-cluster-to-point-in-time \(p. 904\)](#) AWS CLI command and supply the following values:
 - `--source-db-cluster-identifier` – The name of the source DB cluster to create a clone of.
 - `--db-cluster-identifier` – The name of the clone DB cluster.
 - `--restore-type copy-on-write` – The `copy-on-write` value indicates that a clone DB cluster should be created.
 - `--use-latest-restorable-time` – This specifies that the latest restorable backup time should be used.

Note

The [restore-db-cluster-to-point-in-time \(p. 904\)](#) AWS CLI command only clones the DB cluster, not the DB instances for that DB cluster.

The following Linux/UNIX example creates a clone from the `source-db-cluster-id` DB cluster and names the clone `db-clone-cluster-id`.

```
aws neptune restore-db-cluster-to-point-in-time \
```

```
--region us-east-1 \  
--source-db-cluster-identifier source-db-cluster-id \  
--db-cluster-identifier db-clone-cluster-id \  
--restore-type copy-on-write \  
--use-latest-restorable-time
```

The same example works on Windows if the `\` line-end escape character is replaced by the Windows `^` equivalent:

```
aws neptune restore-db-cluster-to-point-in-time ^  
--region us-east-1 ^  
--source-db-cluster-identifier source-db-cluster-id ^  
--db-cluster-identifier db-clone-cluster-id ^  
--restore-type copy-on-write ^  
--use-latest-restorable-time
```

Limitations

DB cloning in Neptune has the following limitations:

- You can't create clone databases across AWS Regions. The clone databases must be created in the same Region as the source databases.
- A cloned database always uses the most recent patch of the Neptune engine version being used by the database it was cloned from. This is true even if the source database has not yet been upgraded to that patch version. The engine version itself does not change, however.
- Currently, you are limited to up to 15 clones based on a copy, including clones based on other clones. After that, only copies can be created. However, each copy can also have up to 15 clones.
- Cross-account DB cloning is not currently supported.
- You can provide a different virtual private cloud (VPC) for your clone. However, the subnets in those VPCs must map to the same set of Availability Zones.

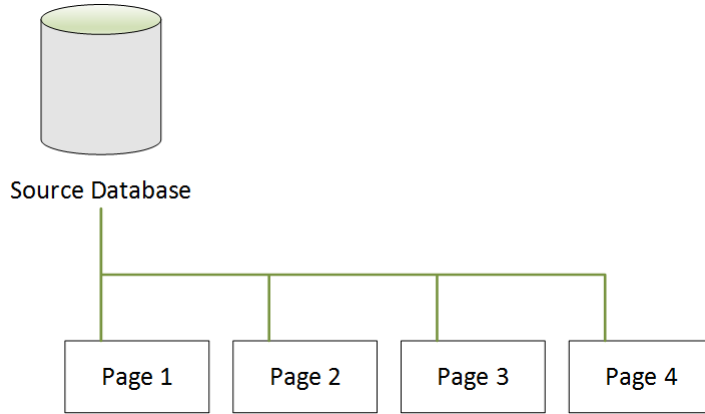
Copy-on-Write Protocol for DB Cloning

The following scenarios illustrate how the copy-on-write protocol works.

- [Neptune Database Before Cloning \(p. 438\)](#)
- [Neptune Database After Cloning \(p. 439\)](#)
- [When a Change Is Made to the Source Database \(p. 439\)](#)
- [When a Change Is Made to the Clone Database \(p. 440\)](#)

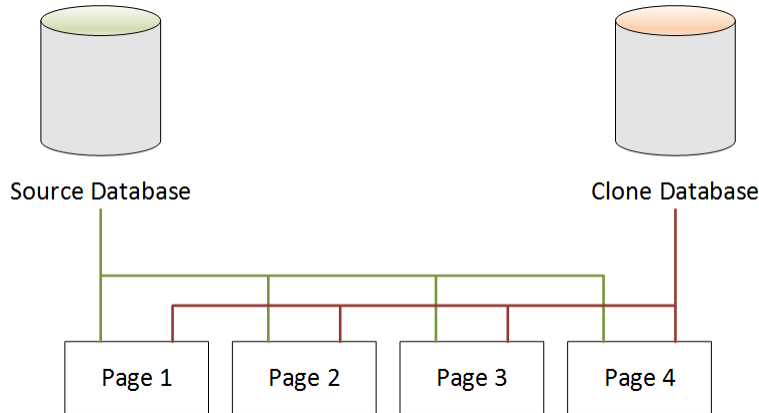
Neptune Database Before Cloning

Data in a source database is stored in pages. In the following diagram, the source database has four pages.



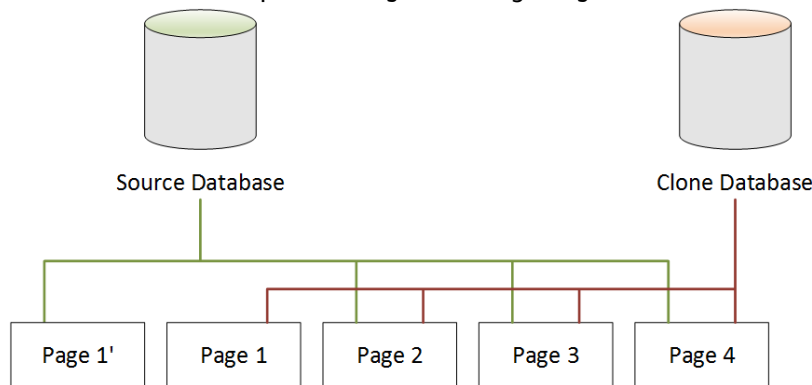
Neptune Database After Cloning

As shown in the following diagram, there are no changes in the source database after DB cloning. Both the source database and the clone database point to the same four pages. No pages have been physically copied, so no additional storage is required.



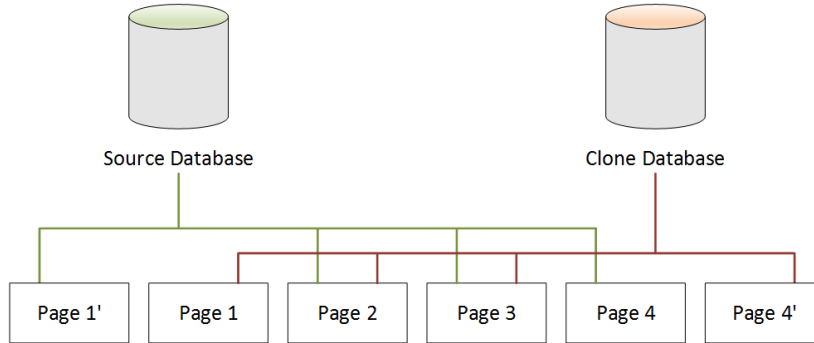
When a Change Is Made to the Source Database

In the following example, the source database makes a change to the data in Page 1. Instead of writing to the original Page 1, it uses additional storage to create a new page, called Page 1'. The source database now points to the new Page 1', and also to Page 2, Page 3, and Page 4. The clone database continues to point to Page 1 through Page 4.



When a Change Is Made to the Clone Database

In the following diagram, the clone database has also changed, this time in Page 4. Instead of writing to the original Page 4, additional storage is used to create a new page, called Page 4'. The source database continues to point to Page 1', and also Page 2 through Page 4, but the clone database now points to Page 1 through Page 3, and also Page 4'.



As shown in the second scenario, after DB cloning, there is no additional storage required at the point of clone creation. However, as changes occur in the source database and clone database, only the changed pages are created, as shown in the third and fourth scenarios. As more changes occur over time in both the source database and clone database, you need incrementally more storage to capture and store the changes.

Deleting a Source Database

Deleting a source database does not affect the clone databases that are associated with it. The clone databases continue to point to the pages that were previously owned by the source database.

Managing Amazon Neptune Instances

The following sections have information on instance-level operations.

Topics

- [Neptune T3 Burstable Instance Class \(p. 442\)](#)
- [Modifying a Neptune DB Instance \(and Applying Immediately\) \(p. 444\)](#)
- [Renaming a Neptune DB Instance \(p. 447\)](#)
- [Rebooting a DB Instance in Amazon Neptune \(p. 448\)](#)
- [Deleting a DB Instance in Amazon Neptune \(p. 449\)](#)

Neptune T3 Burstable Instance Class

In addition to fixed-performance instance classes such as R5 and R4, Amazon Neptune gives you the option of using a burstable-performance T3 instance. While you're developing your graph application, you want your database to be fast and responsive, but you don't need to use it all the time. Neptune's `db.t3.medium` instance class is just what you should use in that situation, at significantly lower cost than the least expensive fixed-performance instance class.

A burstable instance runs at a baseline level of CPU performance until a workload needs more, and then bursts well above the baseline for as long as a workload requires. Its hourly price covers the bursts, provided that the average CPU utilization doesn't exceed the baseline over a 24-hour period. For most development and test situations, that translates to good performance at a low cost.

If you start with a T3 instance class, you can easily switch later to a fixed-performance instance class when you're ready to go into production, using the AWS Management Console, AWS CLI, or one of the AWS SDKs.

T3 Bursting Is Governed by CPU Credits

A CPU credit represents the full utilization of one virtual CPU core (vCPU) for one minute. That can also translate into 50% utilization of a vCPU for two minutes, or 25% utilization of two vCPUs for two minutes, and so on.

A T3 instance accrues CPU credits when it's idle and uses them up when it's active, both measured at millisecond resolution. The `db.t3.medium` instance class has two vCPUs, each of which earns 12 CPU credits per hour when idle. This means that 20% utilization of each vCPU results in a zero CPU credit balance. The 12 CPU credits earned are spent by 20% utilization of the vCPU (since 20% of 60 minutes is also 12). This 20% utilization is thus the *baseline* utilization rate that produces neither a positive nor negative CPU-credit balance.

Idle time (CPU utilization below 20% of the total available) causes CPU credits to be stored in a credit balance bucket, up to the limit for a `db.t3.medium` instance class of 576 (the maximum number of CPU credits that could be accrued in 24 hours, namely $2 \times 12 \times 24$). Over that limit, CPU credits are simply discarded.

When necessary, CPU utilization can burst to as high as 100% for as long as needed by a workload, even after the CPU credit balance falls below zero. If the instance sustains a negative balance continuously for 24 hours, it incurs an additional charge of \$0.05 for every -60 CPU credits accrued over that period. For most development and test workloads, however, bursting is usually covered by idle time before or after the burst.

Note

Neptune's T3 instance class is configured like the Amazon EC2 [unlimited mode](#).

Using the AWS Management Console to Create a T3 Burstable Instance

In the AWS Management Console, you can create a primary DB cluster instance or a read-replica instance that uses the `db.t3.medium` instance class, or you can modify an existing instance to use the `db.t3.medium` instance class.

For example, to create a new DB cluster primary instance in the Neptune console:

- Choose **Create Database**.
- Choose a **DB engine version** equal to or later than 1.0.2.2.
- Under **Purpose**, choose **Development and Testing**.

- As the **DB instance class**, accept the default: `db.t3.medium` – 2 vCPU, 4 GiB RAM.

Using the AWS CLI to Create a T3 Burstable Instance

You can also use the AWS CLI to do the same thing:

```
aws neptune create-db-cluster \  
  --db-cluster-identifier (name for a new DB cluster) \  
  --engine neptune \  
  --engine-version "1.0.2.2"  
  
aws neptune create-db-instance \  
  --db-cluster-identifier (name of the new DB cluster) \  
  --db-instance-identifier (name for the primary writer instance in the cluster) \  
  --engine neptune \  
  --db-instance-class db.t3.medium
```

Modifying a Neptune DB Instance (and Applying Immediately)

You can apply most changes to an Amazon Neptune DB instance immediately or defer them until the next maintenance window. Some modifications, such as parameter group changes, require that you manually reboot your DB instance for the change to take effect.

Important

Some modifications result in an outage because Neptune must reboot your DB instance for the change to be applied. Review the impact to your database and applications before modifying your DB instance settings.

Effects of the Apply Immediately Option

When you modify a DB instance, you can apply the changes immediately. To apply changes immediately, choose the **Apply immediately** option on the AWS Management Console.

If you don't choose to apply changes immediately, the changes are put into the pending modifications queue. During the next maintenance window, any pending changes in the queue are applied.

Important

If you choose to apply the changes immediately, any changes in the pending modifications queue are also applied. If any of the pending modifications require downtime, choosing to apply changes immediately can cause unexpected downtime.

Common Settings and Downtime Implications

The following table contains details about which settings you can change, when the changes can be applied, and whether the changes cause downtime for the DB instance.

DB instance setting	When the change occurs	Downtime notes
Allocated storage	<p>If Apply immediately is selected, the change occurs immediately.</p> <p>If Apply immediately is not selected, the change occurs during the next maintenance window.</p>	No downtime. Performance might be degraded during the change.
Auto minor version upgrade	The change is applied asynchronously, as soon as possible. This setting ignores the Apply immediately setting.	An outage occurs if a newer minor version is available, and Neptune has enabled automatic patching for that version.
Backup retention period	<p>If Apply immediately is selected, the change occurs immediately.</p> <p>If Apply immediately is not selected, and you change the setting from a nonzero value to another nonzero value, the change is applied asynchronously, as soon as possible. Otherwise, the</p>	An outage occurs if you change from zero to a nonzero value, or from a nonzero value to zero.

DB instance setting	When the change occurs	Downtime notes
	change occurs during the next maintenance window.	
Backup window	The change is applied asynchronously, as soon as possible.	–
DB instance class	<p>If Apply immediately is selected, the change occurs immediately.</p> <p>If Apply immediately is not selected, the change occurs during the next maintenance window.</p>	An outage occurs during this change.
DB instance identifier	<p>If Apply immediately is selected, the change occurs immediately.</p> <p>If Apply immediately is not selected, the change occurs during the next maintenance window.</p>	An outage occurs during this change. The DB instance is rebooted.
DB parameter group	<p>Note: Parameter group can only be changed for an entire cluster.</p> <p>The parameter group change occurs immediately. However, parameter changes only occur when you reboot the DB instance manually without failover.</p> <p>For more information, see Rebooting a DB Instance in Amazon Neptune (p. 448).</p>	An outage doesn't occur during this change. However, parameter changes only occur when you reboot the DB instance manually without failover.
Maintenance window	The change occurs immediately. This setting ignores the Apply immediately setting.	<p>If there are one or more pending actions that cause an outage, and the maintenance window is changed to include the current time, those pending actions are applied immediately, and an outage occurs.</p> <p>If you set the window to the current time, there must be at least 30 minutes between the current time and end of the window to ensure that any pending changes are applied.</p>

DB instance setting	When the change occurs	Downtime notes
Security group	The change is applied asynchronously, as soon as possible. This setting ignores the Apply immediately setting.	–

Renaming a Neptune DB Instance

You can rename an Amazon Neptune DB instance by using the AWS Management Console. Renaming a DB instance can have far-reaching effects. The following is a list of things you should know before you rename a DB instance.

- When you rename a DB instance, the endpoint for the DB instance changes because the URL includes the name you assigned to the DB instance. You should always redirect traffic from the old URL to the new one.
- When you rename a DB instance, the old DNS name that was used by the DB instance is immediately deleted, but it can remain cached for a few minutes. The new DNS name for the renamed DB instance becomes effective after about 10 minutes. The renamed DB instance is not available until the new name becomes effective.
- You can't use an existing DB instance name when you are renaming an instance.
- All read replicas that are associated with a DB instance remain associated with that instance after it is renamed. For example, suppose that you have a DB instance that serves your production database, and the instance has several associated read replicas. If you rename the DB instance and then replace it in the production environment with a DB snapshot, the DB instance that you renamed still has the read replicas associated with it.
- Metrics and events that are associated with the name of a DB instance are maintained if you reuse a DB instance name. For example, if you promote a read replica and rename it to be the name of the previous primary instance, the events and metrics that were associated with the primary instance are then associated with the renamed instance.
- DB instance tags remain with the DB instance, regardless of renaming.
- DB snapshots are retained for a renamed DB instance.

To rename a DB instance using the Neptune console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**.
3. Choose the radio button next to the DB instance that you want to rename.
4. In the **Instance actions** menu, choose **Modify**.
5. Enter a new name in the **DB instance identifier** text box. Select **Apply immediately**, and then choose **Continue**.
6. Choose **Modify DB instance** to complete the change.

Rebooting a DB Instance in Amazon Neptune

In some cases, if you modify an Amazon Neptune DB instance, change the DB parameter group that is associated with the instance, or change a static DB parameter in a parameter group that the instance uses, you must reboot the instance to apply the changes.

Rebooting a DB instance restarts the database engine service. A reboot also applies to the DB instance any changes to the associated DB parameter group that were pending. Rebooting a DB instance results in a momentary outage of the instance, during which the DB instance status is set to *rebooting*. If the Neptune instance is configured for Multi-AZ, the reboot might be conducted through a failover. A Neptune event is created when the reboot is completed.

If your DB instance is a Multi-AZ deployment, you can force a failover from one Availability Zone to another when you choose the **Reboot** option. When you force a failover of your DB instance, Neptune automatically switches to a standby replica in another Availability Zone. It then updates the DNS record for the DB instance to point to the standby DB instance. As a result, you must clean up and re-establish any existing connections to your DB instance.

Reboot with failover is beneficial when you want to simulate a failure of a DB instance for testing or restore operations to the original Availability Zone after a failover occurs. For more information, see [High Availability \(Multi-AZ\)](#) in the *Amazon RDS User Guide*. When you reboot a DB cluster, it fails over to the standby replica. Rebooting a Neptune replica does not initiate a failover.

The time required to reboot is a function of the crash recovery process. To improve the reboot time, we recommend that you reduce database activities as much as possible during the reboot process to reduce rollback activity for in-transit transactions.

On the console, the **Reboot** option might be disabled if the DB instance is not in the **Available** state. This can be due to several reasons, such as an in-progress backup, a customer-requested modification, or a maintenance window action.

Note

Rebooting the primary instance of an Amazon Neptune DB cluster also automatically reboots the Neptune replicas for that DB cluster.

To reboot a DB instance using the Neptune console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**.
3. Choose the DB instance that you want to reboot.
4. Choose **Instance actions**, and then choose **Reboot**.
5. To force a failover from one Availability Zone to another, select **Reboot with failover?** in the **Reboot DB Instance** dialog box.
6. Choose **Reboot**. To cancel the reboot, choose **Cancel** instead.

Deleting a DB Instance in Amazon Neptune

You can delete an Amazon Neptune DB instance in any state and at any time, as long the instance has been started and deletion protection is disabled on the instance.

You Cannot Delete a DB Instance If Deletion Protection Is Enabled

You can only delete DB instances that have deletion protection disabled. Neptune enforces deletion protection regardless of whether you use the console, the AWS CLI, or the APIs to delete a DB instance.

Deletion protection is enabled by default when you create a production DB instance using the AWS Management Console.

Deletion protection is disabled by default if you use the AWS CLI or API commands to create a DB instance.

To delete a DB instance that does have deletion protection enabled, first modify the instance to set its `DeletionProtection` field to `false`.

Enabling or disabling deletion protection does not cause an outage.

Taking a Final Snapshot of Your DB Instance Before Deleting It

To delete a DB instance, you must specify the name of the instance and whether you want to have a final DB snapshot taken of the instance. If the DB instance that you're deleting has a status of **Creating**, you can't have a final DB snapshot taken. If the DB instance is in a failure state with a status of **failed**, **incompatible-restore**, or **incompatible-network**, you can only delete the instance when the `SkipFinalSnapshot` parameter is set to `true`.

If you delete all Neptune DB instances in a DB cluster using the AWS Management Console, the entire DB cluster is deleted automatically. If you are using the AWS CLI or SDK, you must delete the DB cluster manually after you delete the last instance.

Important

If you delete an entire DB cluster, all its automated backups are deleted at the same time, and cannot be recovered. This means that unless you choose to create a final DB snapshot manually, you can't restore the DB instance to its final state at a later time. Manual snapshots of an instance are not deleted when the cluster is deleted.

If the DB instance that you want to delete has a read replica, you should either promote the read replica or delete it.

In the following examples, you delete a DB instance both with and without a final DB snapshot.

Deleting a DB Instance with No Final Snapshot

If you want to quickly delete a DB instance, you can skip creating a final DB snapshot. When you delete a DB instance, all automated backups are deleted and cannot be recovered. Manual snapshots are not deleted.

To delete a DB instance with no final DB snapshot using the Neptune console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**.
3. In the **Instances** list, choose the radio button next to the DB instance that you want to delete.

4. Choose **Instance actions**, and then choose **Delete**.
5. Choose **No** in the **Create final snapshot?** box.
6. Choose **Delete**.

Deleting a DB Instance with a Final Snapshot

If you want to be able to restore a deleted DB instance at a later time, you can create a final DB snapshot. All automated backups are also deleted and cannot be recovered. Manual snapshots are not deleted.

To delete a DB instance with a final DB snapshot using the Neptune console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**.
3. In the **Instances** list, choose the radio button next to the DB instance that you want to delete.
4. Choose **Instance actions**, and then choose **Delete**.
5. Choose **Yes** in the **Create final snapshot?** box.
6. In the **Final snapshot name** box, enter the name of your final DB snapshot.
7. Choose **Delete**.

You can check the health of an instance, determine what kind of instance it is, find out which engine release version you currently have installed, and obtain other information about an instance using the [instance-status API](#) (p. 639).

Capturing Graph Changes in Real Time Using Neptune Streams

Neptune Streams logs every change to your graph as it happens, in the order that it is made, in a fully managed way. Once you enable Streams, Neptune takes care of availability, backup, security and expiry.

Note

This feature was available in [Lab Mode \(p. 85\)](#) starting with [Release 1.0.1.0.200463.0 \(2019-10-15\) \(p. 767\)](#), and is available for production use starting with [Neptune engine release 1.0.2.2.R2 \(p. 754\)](#).

The following are some of the many use cases where you might want to capture changes to a graph as they occur:

- You might want your application to notify people automatically when certain changes are made.
- You might want to maintain a current version of your graph data in another data store also, such as Amazon OpenSearch Service, Amazon ElastiCache, or Amazon Simple Storage Service (Amazon S3).

Neptune uses the same native storage for the change-log stream as for graph data. It writes change log entries synchronously together with the transaction that makes those changes. You retrieve these change records from the log stream using an HTTP REST API. (For information, see [Calling the Streams API \(p. 453\)](#).)

The following diagram shows how change-log data can be retrieved from Neptune Streams.

■

Neptune Streams Guarantees

- Changes made by a transaction are immediately available for reading from both writer and readers as soon as the transaction is complete (aside from any normal replication lag in readers).
- Change records appear strictly sequentially, in the order in which they occurred (this includes the changes made within a transaction).
- The changes streams contain no duplicates. Each change is logged only once.
- The changes streams are complete. No changes are lost or omitted.
- The changes streams contain all the information needed to determine the complete state of the database itself at any point in time, provided that the starting state is known.
- Streams can be turned on or off at any time.

Neptune Streams Operational Properties

- The change-log stream is fully managed.
- Change-log data is written synchronously as part of the same transaction that makes a change.
- When Neptune Streams are enabled, you incur I/O and storage charges associated with the change-log data.
- Change records are purged automatically one week after they are created. The retention period of one week is not currently configurable.
- Read performance on the streams scales with instances.

- You can achieve high availability and read throughput using read replicas. There is no limit on the number of stream readers that you can create and use concurrently.
- Change-log data is replicated across multiple Availability Zones, making it highly durable.
- The log data is as secure as your graph data itself. It can be encrypted at rest and in transit. Access can be controlled using IAM, Amazon VPC, and AWS Key Management Service (AWS KMS). Like the graph data, it can be backed up and later restored using point-in-time restores (PITR).
- The synchronous writing of stream data as part of each transaction causes a slight degradation in overall write performance.
- Stream data is not sharded, because Neptune is single-sharded by design.
- The log stream `GetRecords` API uses the same resources as all other Neptune graph operations. This means that clients need to load balance between stream requests and other DB requests.
- When streams are disabled, all log data becomes inaccessible immediately. This means that you must read all log data of interest to you before you disable logging.
- There is currently no native integration with AWS Lambda. The log stream does not generate an event that can trigger a Lambda function.

Topics

- [Using Neptune Streams \(p. 452\)](#)
- [Serialization Formats in Neptune Streams \(p. 456\)](#)
- [Neptune Streams Examples \(p. 459\)](#)
- [Using AWS CloudFormation to Set Up Neptune-to-Neptune Replication with the Streams Consumer Application \(p. 463\)](#)
- [Using Neptune streams cross-region replication for disaster recovery \(p. 467\)](#)

Using Neptune Streams

With the Neptune Streams feature, you can generate a complete sequence of change-log entries that record every change made to your graph data as it happens. For an overview of this feature, see [Capturing Graph Changes in Real Time Using Neptune Streams \(p. 451\)](#).

Topics

- [Enabling Neptune Streams \(p. 452\)](#)
- [Disabling Neptune Streams \(p. 453\)](#)
- [Calling the Neptune Streams REST API \(p. 453\)](#)
- [Neptune Streams API Response Format \(p. 454\)](#)
- [Neptune Streams API Exceptions \(p. 455\)](#)

Enabling Neptune Streams

You can enable or disable Neptune Streams at any time by setting the `neptune_streams` DB Cluster parameter (p. 409). Setting the parameter to 1 enables Streams, and setting it to 0 disables Streams.

Note

Up until recently, Neptune Streams has been an experimental feature that you enable or disable in Lab Mode using the DB Cluster `neptune_lab_mode` parameter (see [Neptune Lab Mode \(p. 85\)](#)). Using Lab Mode to enable Streams is now deprecated and will be disabled in the future.

After you turn Streams on, the change records in the change-log stream continue to be available for one week after they are created.

Disabling Neptune Streams

You can turn Neptune Streams off any time that it is running.

To turn Streams off, update the DB Cluster parameter group so that the value of the `neptune_streams` parameter is set to 0.

Important

As soon as Streams is turned off, you can't access the change-log data any more. Be sure to read what you are interested in *before* turning Streams off.

Calling the Neptune Streams REST API

You access Neptune Streams using a REST API that sends an HTTP GET request to one of the following local endpoints:

- For a SPARQL graph DB: `https://Neptune-DNS:8182/sparql/stream`.
- For a Gremlin or openCypher graph DB: `https://Neptune-DNS:8182/propertygraph/stream`. This endpoint can be aliased as `https://Neptune-DNS:8182/pg/stream`.

Note

As of [engine release 1.1.0.0 \(p. 718\)](#), the Gremlin stream endpoint (`https://Neptune-DNS:8182/gremlin/stream`) is being deprecated, along with its associated output format (GREMLIN_JSON). It is still supported for backward compatibility but may be removed in future releases.

Only an HTTP GET operation is allowed.

Neptune supports gzip compression of the response, provided that the HTTP request includes an `Accept-Encoding` header that specifies gzip as an accepted compression format (that is, "Accept-Encoding: gzip").

Parameters

- `limit` – long, optional. Range: 1–100,000. Default: 10.

Specifies the maximum number of records to return. There is also a size limit of 10 MB on the response that can't be modified and that takes precedence over the number of records specified in the `limit` parameter. The response does include a threshold-breaching record if the 10 MB limit was reached.

- `iteratorType` – String, optional.

This parameter can take one of the following values:

- `AT_SEQUENCE_NUMBER`(default) – Indicates that reading should start from the event sequence number specified jointly by the `commitNum` and `opNum` parameters.
- `AFTER_SEQUENCE_NUMBER` – Indicates that reading should start right after the event sequence number specified jointly by the `commitNum` and `opNum` parameters.
- `TRIM_HORIZON` – Indicates that reading should start at the last untrimmed record in the system, which is the oldest unexpired (not yet deleted) record in the change-log stream. This mode is useful during application startup, when you don't have a specific starting event sequence number.
- `LATEST` – Indicates that reading should start at the most recent record in the system, which is the latest unexpired (not yet deleted) record in the change-log stream. This is useful when there is a need to read records from current top of the streams so as not to process older records, such as during disaster recovery or a zero-downtime upgrade. Note that in this mode, there is at most only one record returned.

- `commitNum` – long, required when `iteratorType` is `AT_SEQUENCE_NUMBER` or `AFTER_SEQUENCE_NUMBER`.

The commit number of the starting record to read from the change-log stream.

This parameter is ignored when `iteratorType` is `TRIM_HORIZON` or `LATEST`.

- `opNum` – long, optional (the default is 1).

The operation sequence number within the specified commit to start reading from in the change-log stream data.

Operations that change SPARQL graph data generally only generate a single change record per operation. However, operations that change Gremlin graph data can generate multiple change records per operation, as in the following examples:

- `INSERT` – A Gremlin vertex can have multiple labels, and a Gremlin element can have multiple properties. A separate change record is generated for each label and property when an element is inserted.
- `UPDATE` – When a Gremlin element property is changed, two change records are generated: the first for removing the previous value, and the second for inserting the new value.
- `DELETE` – A separate change record is generated for each element property that is deleted. For example, when a Gremlin edge with properties is deleted, one change record is generated for each of the properties, and after that, one is generated for deletion of the edge label.

When a Gremlin vertex is deleted, all the incoming and outgoing edge properties are deleted first, then the edge labels, then the vertex properties, and finally the vertex labels. Each of these deletions generates a change record.

Neptune Streams API Response Format

A response to a Neptune Streams REST API request has the following fields:

- `lastEventId` – Sequence identifier of the last change in the stream response. An event ID is composed of two fields: A `commitNum` identifies a transaction that changed the graph, and an `opNum` identifies a specific operation within that transaction. This is shown in the following example.

```
"eventId": {
  "commitNum": 12,
  "opNum": 1
}
```

- `lastTrxTimestamp` – The time at which the commit for the transaction was requested, in milliseconds from the Unix epoch.
- `format` – Serialization format for the change records being returned. The possible values are `GREMLIN_JSON` for Gremlin change records, and `NQUADS` for SPARQL change records.
- `records` – An array of serialized change-log stream records included in the response. Each record in the `records` array contains these fields:
 - `commitTimestamp` – The time at which the commit for the transaction was requested, in milliseconds from the Unix epoch.
 - `eventId` – The sequence identifier of the stream change record.
 - `data` – The serialized Gremlin, SPARQL, or OpenCypher change record. The serialization formats of each record are described in more detail in the next section, [Serialization Formats in Neptune Streams \(p. 456\)](#).
 - `op` – The operation that created the change.

- `totalRecords` – The total number of records in the response.

For example, the following response returns Gremlin change data.

```
{
  "lastEventId": {
    "commitNum": 12,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "GREMLIN_JSON",
  "records": [
    {
      "commitTimestamp": 1560011610678,
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "v1",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD"
    }
  ],
  "totalRecords": 1
}
```

The following response returns SPARQL change data.

```
{
  "lastEventId": {
    "commitNum": 97,
    "opNum": 1
  },
  "lastTrxTimestamp": 1561489355102,
  "format": "NQUADS",
  "records": [
    {
      "commitTimestamp": 1561489355102,
      "eventId": {
        "commitNum": 97,
        "opNum": 1
      },
      "data": {
        "stmt": "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
      },
      "op": "ADD"
    }
  ],
  "totalRecords": 1
}
```

Neptune Streams API Exceptions

The following table describes Neptune Streams exceptions.

Error Code	HTTP Code	OK to Retry?	Message
InvalidParameterException	400	No	An invalid or out-of-range value was supplied as an input parameter.
ExpiredStreamException	400	No	All of the requested records exceed the maximum age allowed and have expired.
ThrottlingException	500	Yes	Rate of requests exceeds the maximum throughput.
StreamRecordsNotFoundException	404	No	The requested resource could not be found. The stream may not be specified correctly.
MemoryLimitExceededException	500	Yes	The request processing did not succeed due to lack of memory, but can be retried when the server is less busy.

Serialization Formats in Neptune Streams

Amazon Neptune uses two different formats for serializing graph-changes data to log streams, depending on whether the graph was created using Gremlin or SPARQL.

Both formats share a common record serialization format, as described in [Neptune Streams API Response Format \(p. 454\)](#), that contains the following fields:

- `commitTimestamp` – The time at which the commit for the transaction was requested, in milliseconds from the Unix epoch.
- `eventId` – The sequence identifier of the stream change record.
- `data` – The serialized Gremlin, SPARQL, or OpenCypher change record. The serialization formats of each record are described in more detail in the next sections.
- `op` – The operation that created the change.

Topics

- [PG_JSON Change Serialization Format \(p. 456\)](#)
- [SPARQL NQUADS Change Serialization Format \(p. 459\)](#)

PG_JSON Change Serialization Format

Note

As of [engine release 1.1.0.0 \(p. 718\)](#), the Gremlin stream output format (`GREMLIN_JSON`) output by the Gremlin stream endpoint (`https://Neptune-DNS:8182/gremlin/stream`) is being deprecated. It is replaced by `PG_JSON`, which is currently identical to `GREMLIN_JSON`.

A Gremlin or openCypher change record, contained in the `data` field of a log stream response, has the following fields:

- `id` – String, required.

The ID of the Gremlin or openCypher element.

- `type` – String, required.

The type of this Gremlin or openCypher element. Must be one of the following:

- `v1` – Vertex label for Gremlin; node label for openCypher.
- `vp` – Vertex properties for Gremlin; node properties for openCypher.
- `e` – Edge and edge label for Gremlin; relationship and relationship type for openCypher.
- `ep` – Edge properties for Gremlin; relationship properties for openCypher.
- `key` – String, required.

The property name. For element labels, this is "label".

- `value` – value object, required.

This is a JSON object that contains a `value` field for the value itself, and a `dataType` field for the JSON data type of that value.

```
"value": {  
  "value": "the new value",  
  "dataType": "the JSON datatype of the new value"  
}
```

- `from` – String, optional.

If this is an edge (`type="e"`), the ID of the corresponding *from* vertex or source node.

- `to` – String, optional.

If this is an edge (`type="e"`), the ID of the corresponding *to* vertex or target node.

Gremlin Examples

- The following is an example of a Gremlin vertex label.

```
{  
  "id": "an ID string",  
  "type": "v1",  
  "key": "label",  
  "value": {  
    "value": "the new value of the vertex label",  
    "dataType": "String"  
  }  
}
```

- The following is an example of a Gremlin vertex property.

```
{  
  "id": "an ID string",  
  "type": "vp",  
  "key": "the property name",  
  "value": {  
    "value": "the new value of the vertex property",  
    "dataType": "the datatype of the vertex property"  
  }  
}
```

```
}
```

- The following is an example of a Gremlin edge.

```
{  
  "id": "an ID string",  
  "type": "e",  
  "key": "label",  
  "value": {  
    "value": "the new value of the edge",  
    "dataType": "String"  
  },  
  "from": "the ID of the corresponding \"from\" vertex",  
  "to": "the ID of the corresponding \"to\" vertex"  
}
```

openCypher Examples

- The following is an example of an openCypher node label.

```
{  
  "id": "an ID string",  
  "type": "v1",  
  "key": "label",  
  "value": {  
    "value": "the new value of the node label",  
    "dataType": "String"  
  }  
}
```

- The following is an example of an openCypher node property.

```
{  
  "id": "an ID string",  
  "type": "vp",  
  "key": "the property name",  
  "value": {  
    "value": "the new value of the node property",  
    "dataType": "the datatype of the node property"  
  }  
}
```

- The following is an example of an openCypher relationship.

```
{  
  "id": "an ID string",  
  "type": "e",  
  "key": "label",  
  "value": {  
    "value": "the new value of the relationship",  
    "dataType": "String"  
  },  
  "from": "the ID of the corresponding source node",  
  "to": "the ID of the corresponding target node"  
}
```

SPARQL NQUADS Change Serialization Format

Neptune logs changes to SPARQL quads in the graph using the Resource Description Framework (RDF) N-QUADS language defined in the [W3C RDF 1.1 N-Quads](#) specification.

The data field in the change record simply contains a `stmt` field that holds an N-QUADS statement expressing the changed quad, as in the following example.

```
"stmt" : "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
```

Neptune Streams Examples

The following examples show how to access change-log stream data in Amazon Neptune.

Topics

- [AT_SEQUENCE_NUMBER Change Log \(p. 459\)](#)
- [AFTER_SEQUENCE_NUMBER Change Log \(p. 460\)](#)
- [TRIM_HORIZON Change Log \(p. 461\)](#)
- [LATEST Change Log \(p. 461\)](#)
- [Compression Change Log \(p. 462\)](#)

AT_SEQUENCE_NUMBER Change Log

The following example shows a Gremlin or openCypher `AT_SEQUENCE_NUMBER` change log.

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?
limit=1&commitNum=1&opNum=1&iteratorType=AT_SEQUENCE_NUMBER" |jq
{
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
    {
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "commitTimestamp": 1560011610678,
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "v1",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD"
    }
  ],
}
```

```
"totalRecords": 1  
}
```

This one shows a SPARQL example of an AT_SEQUENCE_NUMBER change log.

```
curl -s "https://localhost:8182/sparql/stream?  
limit=1&commitNum=1&opNum=1&iteratorType=AT_SEQUENCE_NUMBER" |jq  
{  
  "lastEventId": {  
    "commitNum": 1,  
    "opNum": 1  
  },  
  "lastTrxTimestamp": 1571252030566,  
  "format": "NQUADS",  
  "records": [  
    {  
      "eventId": {  
        "commitNum": 1,  
        "opNum": 1  
      },  
      "commitTimestamp": 1571252030566,  
      "data": {  
        "stmt": "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"  
      },  
      "op": "ADD"  
    }  
  ],  
  "totalRecords": 1  
}
```

AFTER_SEQUENCE_NUMBER Change Log

The following example shows a Gremlin or openCypher AFTER_SEQUENCE_NUMBER change log.

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?  
limit=1&commitNum=1&opNum=1&iteratorType=AFTER_SEQUENCE_NUMBER" |jq  
{  
  "lastEventId": {  
    "commitNum": 2,  
    "opNum": 1  
  },  
  "lastTrxTimestamp": 1560011633768,  
  "format": "PG_JSON",  
  "records": [  
    {  
      "commitTimestamp": 1560011633768,  
      "eventId": {  
        "commitNum": 2,  
        "opNum": 1  
      },  
      "data": {  
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",  
        "type": "v1",  
        "key": "label",  
        "value": {  
          "value": "vertex",  
          "dataType": "String"  
        }  
      },  
      "op": "REMOVE"  
    }  
  ],  
}
```

```
"totalRecords": 1
}
```

TRIM_HORIZON Change Log

The following example shows a Gremlin or openCypher TRIM_HORIZON change log.

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?limit=1&iteratorType=TRIM_HORIZON" | jq
{
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1560011610678,
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "v1",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD"
    }
  ],
  "totalRecords": 1
}
```

LATEST Change Log

The following example shows a Gremlin or openCypher LATEST change log. Note that the API parameters `limit`, `commitNum`, and `opNum` are completely optional.

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?iteratorType=LATEST" | jq
{
  "lastEventId": {
    "commitNum": 21,
    "opNum": 4
  },
  "lastTrxTimestamp": 1634710497743,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1634710497743,
      "eventId": {
        "commitNum": 21,
        "opNum": 4
      },
      "data": {
        "id": "24be4e2b-53b9-b195-56ba-3f48fa2b60ac",

```

```
    "type": "e",
    "key": "label",
    "value": {
      "value": "created",
      "dataType": "String"
    },
    "from": "4",
    "to": "5"
  },
  "op": "REMOVE"
}
],
"totalRecords": 1
}
```

Compression Change Log

The following example shows a Gremlin or openCypher compression change log.

```
curl -sH \
  "Accept-Encoding: gzip" \
  "https://Neptune-DNS:8182/propertygraph/stream?limit=1&commitNum=1" \
  -H "Accept-Encoding: gzip" \
  -v |gunzip -|jq
> GET /gremlin/stream?limit=1 HTTP/1.1
> Host: localhost:8182
> User-Agent: curl/7.64.0
> Accept: /
> Accept-Encoding: gzip
*> Accept-Encoding: gzip*
>
< HTTP/1.1 200 OK
< Content-Type: application/json; charset=UTF-8
< Connection: keep-alive
*< content-encoding: gzip*
< content-length: 191
<
{ [191 bytes data]
Connection #0 to host localhost left intact
{
  "lastEventId": "1:1",
  "lastTrxTimestamp": 1558942160603,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1558942160603,
      "eventId": "1:1",
      "data": {
        "id": "v1",
        "type": "v1",
        "key": "label",
        "value": {
          "value": "person",
          "dataType": "String"
        }
      },
      "op": "ADD"
    }
  ],
  "totalRecords": 1
}
```

Using AWS CloudFormation to Set Up Neptune-to-Neptune Replication with the Streams Consumer Application







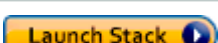
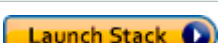
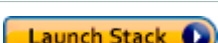
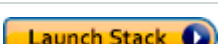
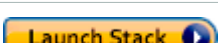
You can use an AWS CloudFormation template to set up the Neptune streams consumer application to support Neptune-to-Neptune replication.












Topics

- [Choose an AWS CloudFormation template for Your Region \(p. 463\)](#)
- [Add details About the Neptune streams consumer stack you're creating \(p. 464\)](#)
- [Run the AWS CloudFormation Template \(p. 466\)](#)
- [To update the stream poller with the latest Lambda artifacts \(p. 466\)](#)

Choose an AWS CloudFormation template for Your Region

To launch the appropriate AWS CloudFormation stack on the AWS CloudFormation console, choose one of the **Launch Stack** buttons in the following table, depending on the AWS Region that you want to use.

Region	View	View in Designer	Launch
US East (N. Virginia)	View	View in Designer	
US East (Ohio)	View	View in Designer	
US West (N. California)	View	View in Designer	
US West (Oregon)	View	View in Designer	
Canada (Central)	View	View in Designer	
South America (São Paulo)	View	View in Designer	
Europe (Stockholm)	View	View in Designer	
Europe (Ireland)	View	View in Designer	
Europe (London)	View	View in Designer	
Europe (Paris)	View	View in Designer	
Europe (Frankfurt)	View	View in Designer	

Region	View	View in Designer	Launch
Middle East (Bahrain)	View	View in Designer	Launch Stack 
Asia Pacific (Tokyo)	View	View in Designer	Launch Stack 
Asia Pacific (Hong Kong)	View	View in Designer	Launch Stack 
Asia Pacific (Seoul)	View	View in Designer	Launch Stack 
Asia Pacific (Singapore)	View	View in Designer	Launch Stack 
Asia Pacific (Sydney)	View	View in Designer	Launch Stack 
Asia Pacific (Mumbai)	View	View in Designer	Launch Stack 
China (Beijing)	View	View in Designer	Launch Stack 
China (Ningxia)	View	View in Designer	Launch Stack 
AWS GovCloud (US-West)	View	View in Designer	Launch Stack 
AWS GovCloud (US-East)	View	View in Designer	Launch Stack 

On the **Create Stack** page, choose **Next**.

Add details About the Neptune streams consumer stack you're creating

The **Specify Stack Details** page provides properties and parameters that you can use to control the setup of the application:

Stack Name – The name of the new AWS CloudFormation stack that you're creating. You can generally use the default value, `NeptuneStreamPoller`.

Under **Parameters**, provide the following:

Network Configuration for the VPC Where the Streams Consumer Runs

- **VPC** – Provide the name of the VPC where the polling Lambda function will run.
- **List of Subnet IDs** – The subnets to which a network interface is established. Add subnets corresponding to your Neptune cluster.
- **List of Security Group IDs** – Provide the IDs of security groups that grant write inbound access to your source Neptune DB cluster.
- **List of Route Table IDs** – This is needed to create an Amazon DynamoDB endpoint in your Neptune VPC, if you do not already have one. You must provide a comma-separated list of route table IDs associated with the subnets.

- **Require to create Dynamo DB VPC Endpoint** – A Boolean value that defaults to `true`. You only need to change it to `false` if you have already created a DynamoDB endpoint in your VPC.
- **Require to create Monitoring VPC Endpoint** – A Boolean value that defaults to `true`. You only need to change it to `false` if you have already created a monitoring endpoint in your VPC.

Stream Poller

- **Application Name** – You can generally leave this set to the default (`NeptuneStream`). If you use a different name, it must be unique.
- **Memory size for Lambda Poller** – Used to set the memory size available to the Lambda poller function. The default value is 2,048 megabytes.
- **Lambda Runtime** – The language used in the Lambda function that retrieves items from the Neptune stream. You can set this either to `python3.6` or to `java8`.
- **S3 Bucket having Lambda code artifacts** – Leave this blank unless you are using a custom Lambda polling function that loads from a different S3 bucket.
- **S3 Key corresponding to Lambda Code artifacts** – Leave this blank unless you are using a custom Lambda polling function.
- **Logging level for Lambda** – In general, leave this set to the default value, `INFO`.
- **Managed Policies for Lambda Execution** – In general, leave this blank unless you are using a custom Lambda polling function.
- **Stream Records Handler** – In general, leave this blank unless you are using a custom handler for the records in Neptune streams.
- **Maximum records Fetched from Stream** – You can use this parameter to tune performance. The default (100) is a good place to start. The maximum allowable is 10,000. The higher the number, the fewer network calls are needed to read records from the stream, but the more memory is required to process the records. Lower values of this parameter result in lower throughput.
- **Max wait time between two Polls (in Seconds)** – Determines how frequently the Lambda poller is invoked to poll the Neptune streams. Set this value to 0 for continuous polling. The maximum value is 3,600 seconds (1 hour). The default value (60 seconds) is a good place to start, depending on how fast your graph data changes.
- **Maximum Continuous polling period (in Seconds)** – Used to set a timeout for the Lambda polling function. It should be between 5 seconds and 900 seconds. The default value (600 seconds) is a good place to start.
- **Step Function Fallback Period** – The number of step-function-fallback-period units to wait for the poller, after which the step function is called through Amazon CloudWatch Events to recover from a failure. The default (5 minutes) is a good place to start.
- **Step Function Fallback Period Unit** – The time units used to measure the preceding Step Function Fallback Period (minutes, hours, days). The default (minutes) is generally sufficient.
- **Data replication scope** – Determines whether to replicate both nodes and edges, or only nodes to OpenSearch (this applies to Gremlin engine data only). The default value (All) is generally a good place to start.
- **Ignore Elastic Search missing document error** – Flag to determine whether a missing document error in OpenSearch can be ignored. Missing document errors occur rarely but need manual intervention if not ignored. The default value (True) is generally a good place to start.

Neptune Stream

- **Endpoint of source Neptune Stream** – (*Required*) This takes one of two forms:
 - `https://your DB cluster:port/gremlin/stream`
 - `https://your DB cluster:port/sparql/stream`

- **Neptune Query Engine** – Choose Gremlin or SPARQL.
- **Is IAM Auth Enabled?** – If your Neptune DB cluster is using IAM authentication, set this parameter to `true`.
- **Neptune Cluster Resource Id** – If your Neptune DB cluster is using IAM authentication, set this parameter to the cluster resource ID. The resource ID is not the same as the cluster ID. Instead, it takes the form: `cluster-` followed by 28 alpha-numeric characters. It can be found under **Cluster Details** in the Neptune console.

Alarm

- **Required to create Cloud watch Alarm** – Set this to `true` if you want to create a CloudWatch alarm for the new stack.
- **Email for Alarm Notifications** – The email address to which alarm notifications should be sent (only needed if alarms are enabled).

Run the AWS CloudFormation Template

Now you can complete the process of provisioning a Neptune streams consumer application instance as follows:

1. In AWS CloudFormation, on the **Specify Stack Details** page, choose **Next**.
2. On the **Options** page, choose **Next**.
3. On the **Review** page, select the first check box to acknowledge that AWS CloudFormation will create IAM resources. Select the second check box to acknowledge `CAPABILITY_AUTO_EXPAND` for the new stack.

Note

`CAPABILITY_AUTO_EXPAND` explicitly acknowledges that macros will be expanded when creating the stack, without prior review. Users often create a change set from a processed template so that the changes made by macros can be reviewed before actually creating the stack. For more information, see the AWS CloudFormation [CreateStack](#) API in the *AWS CloudFormation API Reference*.

Then choose **Create**.

To update the stream poller with the latest Lambda artifacts

You can update the stream poller with the latest Lambda code artifacts as follows:

1. In the AWS Management Console, navigate to AWS CloudFormation and select the main parent AWS CloudFormation stack.
2. Select the **Update** option for the stack.
3. Select **Replace current template**.
4. For the template source, choose **Amazon S3 URL** and enter the following S3 URL:

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/
neptune_to_elastic_search.json
```

5. Select **Next** without changing any AWS CloudFormation parameters.
6. Choose **Update Stack**.

The stack will now update the Lambda artifacts with the most recent ones.

Using Neptune streams cross-region replication for disaster recovery

Neptune provides two ways of implementing cross-region failover capabilities:

- Cross-region snapshot copy and restore
- Using Neptune streams to replicate data between two clusters in two different regions.

Cross-region snapshot copy and restore has the lowest operational overhead for recovering a Neptune cluster in a different region. However, copying a snapshot between regions can require significant data-transfer time, since a snapshot is a full backup of the Neptune cluster. As a result, cross-region snapshot copy and restore can be used for scenarios that only require a Recovery Point Objective (RPO) of hours and a Recovery Time Objective (RTO) of hours.

A Recovery Point Objective (RPO) is measured by the time in between backups. It defines how much data may be lost between the time the last backup was made and the time at which the database is recovered.

A Recovery Time Objective (RTO) is measured by the time it takes to perform a recovery operation. This is the time it takes the DB cluster to fail over to a recovered database after a failure occurs.

Neptune streams provides a way to keep a backup Neptune cluster in sync with the primary production cluster at all times. If a failure occurs, your database then fails over to the backup cluster. This reduces RPO and RTO to minutes, since data is constantly being copied to the backup cluster, which is immediately available as a failover target at any time.

The drawback of using Neptune streams in this way is that both the operational overhead required to maintain the replication components, and the cost of having a second Neptune DB cluster online all of the time, can be significant.

Important

Currently, using Neptune streams for disaster recovery only works if your target cluster in another region does not have IAM authentication enabled.

Setting up Neptune-to-Neptune replication

Your primary production DB cluster resides in a VPC in a given source region. There are three main things that you need to replicate or emulate in a different, recovery region for the purposes of disaster recovery:

- The data stored in the cluster.
- The configuration of the primary cluster. This would include whether it uses IAM authentication, whether it is encrypted, its DB cluster parameters, its instance parameters, instance sizes, and so forth).
- The networking topology it uses, including the target VPC, its security groups, and so forth.

You can use Neptune management APIs such as the following to gather that information:

- [DescribeDBClusters](#) (p. 826)
- [DescribeDBInstances](#) (p. 859)
- [DescribeDBClusterParameters](#) (p. 877)
- [DescribeDBParameters](#) (p. 875)

- [DescribeVpcs](#)

With the information you gather, you can use the following procedure to set up a backup cluster in a different region, to which your production cluster can fail over in the event of a failure.

1: Enable Neptune streams

You can use the [ModifyDBClusterParameterGroup \(p. 872\)](#) to set the `neptune_streams` parameter to 1. Then, reboot all the instances in the DB cluster so that change takes effect.

It's a good idea to perform at least one add or update operation on the source DB cluster after Neptune streams has been enabled. This populates the change stream with data points that can be referenced later when re-syncing the production cluster with the backup cluster.

2: Create a new VPC in the region where you want to set up your backup cluster

Before creating a new Neptune DB cluster in a different region from your primary cluster, you need to establish a new VPC in the target region to host the cluster. Connectivity between the primary and backup clusters is established through VPC peering, which uses traffic across private subnets in different VPCs. However, to establish VPC peering between two VPCs, they must not have overlapping CIDR blocks or IP address spaces. This that you can't just use the default VPC in both regions, because the CIDR block for a default VPC is always the same (172.31.0.0/16).

You can use an existing VPC in the target region as long as it meets the following conditions:

- It does not have a CIDR block that overlaps with the CIDR block of the VPC where your primary cluster is located.
- It is not already peered with another VPC that has the same CIDR block as the VPC where your primary cluster is located.

If there is no suitable VPC available in the target region, create one using the Amazon EC2 [CreateVpc](#) API.

3: Create a snapshot of your primary cluster and restore it to the target backup region

Now you create a new Neptune cluster in an appropriate VPC in the target backup region that is a copy of your production cluster:

Make a copy of your production cluster in the backup region

1. In your target backup region, re-create the parameters and parameter groups used by your production DB cluster. You can do this using [CreateDBClusterParameterGroup \(p. 869\)](#), [CreateDBParameterGroup \(p. 867\)](#), [ModifyDBClusterParameterGroup \(p. 872\)](#) and [ModifyDBParameterGroup \(p. 871\)](#).

Note that the [CopyDBClusterParameterGroup \(p. 866\)](#) and [CopyDBParameterGroup \(p. 865\)](#) APIs do not currently support cross-region copying.

2. Use [CreateDBClusterSnapshot \(p. 889\)](#) to create a snapshot of your production cluster in the VPC in your production region.
3. Use [CopyDBClusterSnapshot \(p. 893\)](#) to copy the snapshot to the VPC in your target backup region.

4. Use [RestoreDBClusterFromSnapshot \(p. 898\)](#) to create a new DB cluster in the VPC in your target backup region using the copied snapshot. Use the configuration settings and parameters that you copied from your primary production cluster.
5. The new Neptune cluster now exists but doesn't contain any instances. Use [CreateDBInstance \(p. 833\)](#) to create a new primary/writer instance that has the same instance type and size as your production cluster's writer instance. There's no need to create additional read-replicas at this point unless your backup instance will be used to service read I/O in the target region prior to a failover.

4: Establish VPC peering between your primary cluster's VPC and your new backup cluster's VPC

By setting up VPC peering, you enable your primary cluster's VPC to communicate with your backup cluster's VPC as if they are a single private network. To do this, take the following steps:

1. From your production cluster's VPC, call the [CreateVpcPeeringConnection](#) API to establish the peering connection.
2. From your target backup cluster's VPC, call the [AcceptVpcPeeringConnection](#) API to accept the peering connection.
3. From your production cluster's VPC, use the [CreateRoute](#) API to add a route to the VPC's route table that redirects all traffic to the target VPC's CIDR block so that it uses the VPC peering prefix list.
4. Similarly, from your target backup cluster's VPC, use the [CreateRoute](#) API to add a route to the VPC's route table that routes traffic to the primary cluster's VPC.

5: Set up the Neptune streams replication infrastructure

Now that both clusters are deployed and network communication between both regions has been established, use the [Neptune-to-Neptune AWS CloudFormation template \(p. 463\)](#) to deploy the Neptune streams consumer Lambda function with the additional infrastructure that supports data replication. Do this in your primary production cluster's VPC.

Important

This process does not currently support a target cluster that has IAM authentication enabled.

The parameters that you will need to provide for this AWS CloudFormation stack are:

- **NeptuneStreamEndpoint** – The stream endpoint for the primary cluster, in URL format. For example: `https://(cluster name):8182/gremlin/stream`.
- **QueryEngine** – This must be either `gremlin` or `sparql`.
- **RouteTableIds** – Lets you add routes for both a DynamoDB VPC Endpoint and a monitoring VPC Endpoint.

Two additional parameters, namely `CreateMonitoringEndpoint` and `CreateDynamoDBEndpoint`, must also be set to `true` if they do not already exist on the primary cluster's VPC. If they do already exist, make sure they are set to `false` or the AWS CloudFormation creation will fail.

- **SecurityGroupIds** – Specifies the security group used by the Lambda consumer to communicate with the primary cluster's Neptune stream endpoint.

In the target backup cluster, attach a security group that allows traffic originating from this security group.

- **SubnetIds** – A list of subnet ID in the primary cluster's VPC that can be used by the Lambda consumer to communicate with the primary cluster.

- **TargetNeptuneClusterEndpoint** – The cluster endpoint (hostname only) of the target backup cluster
- **VPC** – The ID of the primary cluster's VPC.

All other parameters can be left with their default values.

Once the AWS CloudFormation template has been deployed, Neptune will begin replicating any changes from the primary cluster to the backup cluster. You can monitor this replication in the CloudWatch logs generated by the Lambda consumer function.

Other considerations

- If you need to use IAM authentication between the primary and backup clusters, you can also set it up when you invoke the AWS CloudFormation template.
- If encryption at rest is enabled on your primary cluster, consider how to manage the associated KMS keys when copying the snapshot across to the target region and associate a new KMS key in the target region.
- A best practice is to use DNS CNAMEs in front of the Neptune endpoints used in your applications. Then, if you need to manually failover to the target backup cluster, these CNAMEs can be changed to point to the target cluster and/or instance endpoints.

Full text search in Amazon Neptune using Amazon OpenSearch Service (OpenSearch Service)

Neptune integrates with [Amazon OpenSearch Service \(OpenSearch Service\)](#) to support full-text search in both Gremlin and SPARQL queries. This feature is available starting in [Neptune engine release 1.0.2.1 \(p. 755\)](#), although we recommend using it with engine release 1.0.4.2 or higher to take advantage of the latest fixes.

Important

When integrating with Amazon OpenSearch Service, Neptune requires Elasticsearch version 7.1 or higher rather than OpenSearch version 1.0. Neptune is not currently compatible with OpenSearch version 1.0.

You can use Neptune with an existing OpenSearch Service cluster that has been populated according to the [Neptune Data Model for OpenSearch Data \(p. 478\)](#). Or, you can create an OpenSearch Service domain linked with Neptune using an AWS CloudFormation stack.

Topics

- [Using Apache Lucene query syntax in Neptune full-text search queries \(p. 472\)](#)
- [Amazon Neptune-to-OpenSearch replication setup \(p. 472\)](#)
- [Neptune Data Model for OpenSearch Data \(p. 478\)](#)
- [Neptune Full-Text Search Parameters \(p. 480\)](#)
- [Non-string OpenSearch indexing in Amazon Neptune \(p. 483\)](#)
- [Full-Text-Search Query Execution in Amazon Neptune \(p. 494\)](#)
- [Sample SPARQL queries using full-text search in Neptune \(p. 495\)](#)
- [Using Neptune Full-Text Search in Gremlin Queries \(p. 498\)](#)
- [Troubleshooting Neptune full-text search \(p. 505\)](#)

If you are using IAM, you need to make sure that you have an IAM user with permissions both for Neptune and for OpenSearch. This means that your user must have an OpenSearch Service access policy in place like this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::account-id:root"
      },
      "Action": "es:*",
      "Resource": "arn:aws:es:region:account-id:es-resource-id/*"
    }
  ]
}
```


Important

The Neptune to OpenSearch replication process described here does not replicate blank nodes. This is an important limitation to note.

Using Apache Lucene query syntax in Neptune full-text search queries

OpenSearch supports using [Apache Lucene syntax](#) for query_string queries. This is particularly useful for passing multiple filters in a query.

Neptune uses a nested structure for storing properties in an OpenSearch document (see [Neptune Full-Text Search Data Model \(p. 478\)](#)). When using Lucene syntax, you need to use full paths to the properties in this nested model.

Here is a Gremlin example:

```
g.withSideEffect("Neptune#fts.endpoint", "es_endpoint")
  .withSideEffect("Neptune#fts.queryType", "query_string")
  .V()
  .has("...", "Neptune#fts predicates.name.value:\"Jane Austin\" AND entity_type:Book")
```

Here is a SPARQL example:

```
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200 (http://localhost:9200/)' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*name.value:Ronak AND predicates.\\*foaf\\*surname.value:Sh*" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

Amazon Neptune-to-OpenSearch replication setup

Amazon Neptune supports full-text search in Gremlin and SPARQL queries using Amazon OpenSearch Service (OpenSearch Service). You can use an AWS CloudFormation stack to link an OpenSearch Service domain to Neptune.

Before you begin, you need an existing Neptune DB cluster with streams enabled on it to serve as the source, and an OpenSearch Service domain to serve as the replication target.

If you already have an existing target OpenSearch Service domain that can be accessed by Lambda in the VPC where your Neptune DB cluster is located, the template can use that one. Otherwise, you need to create a new one.

Note

The OpenSearch cluster and Lambda function that you create must be located in the same VPC as your Neptune DB cluster, and the OpenSearch cluster must be configured in VPC mode (not Internet mode).

We recommend that you use a newly created Neptune instance to use with OpenSearch Service. If you use an existing instance that already has data in it, you should perform an OpenSearch Service data sync

before making queries or there may be data inconsistencies. This GitHub project provides an example of how to perform the synchronization: [Export Neptune to OpenSearch](https://github.com/awslabs/amazon-neptune-tools/tree/master/export-neptune-to-elasticsearch) (https://github.com/awslabs/amazon-neptune-tools/tree/master/export-neptune-to-elasticsearch).





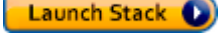
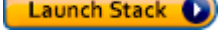
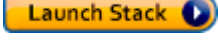
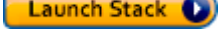






Important








When integrating with Amazon OpenSearch Service, Neptune requires Elasticsearch version 7.1 or higher rather than OpenSearch version 1.0. Neptune is not currently compatible with OpenSearch version 1.0.

The AWS CloudFormation template below then creates a streams-consumer application instance that provides Neptune-to-OpenSearch replication.

Set up Neptune-to-OpenSearch replication using an AWS CloudFormation template for your region

To launch the AWS CloudFormation stack on the AWS CloudFormation console, choose one of the **Launch Stack** buttons in the following table, depending on the AWS Region that you want to use.

Region	View	View in Designer	Launch
US East (N. Virginia)	View	View in Designer	
US East (Ohio)	View	View in Designer	
US West (N. California)	View	View in Designer	
US West (Oregon)	View	View in Designer	
Canada (Central)	View	View in Designer	
South America (São Paulo)	View	View in Designer	
Europe (Stockholm)	View	View in Designer	
Europe (Ireland)	View	View in Designer	
Europe (London)	View	View in Designer	
Europe (Paris)	View	View in Designer	
Europe (Frankfurt)	View	View in Designer	
Middle East (Bahrain)	View	View in Designer	
Asia Pacific (Hong Kong)	View	View in Designer	
Asia Pacific (Tokyo)	View	View in Designer	

Region	View	View in Designer	Launch
Asia Pacific (Seoul)	View	View in Designer	Launch Stack 
Asia Pacific (Singapore)	View	View in Designer	Launch Stack 
Asia Pacific (Mumbai)	View	View in Designer	Launch Stack 
China (Beijing)	View	View in Designer	Launch Stack 
China (Ningxia)	View	View in Designer	Launch Stack 
AWS GovCloud (US-West)	View	View in Designer	Launch Stack 
AWS GovCloud (US-East)	View	View in Designer	Launch Stack 

On the **Create Stack** page, choose **Next**.

Add Details About the new OpenSearch stack you are creating

The **Specify Stack Details** page provides properties and parameters that you can use to control the setup of full-text search:

Stack Name – The name of the new AWS CloudFormation stack that you're creating. You can generally use the default value, `NeptuneStreamPoller`.

Under **Parameters**, provide the following:

Network Configuration for the VPC Where the Streams Consumer Runs

- **VPC** – Provide the name of the VPC where the polling Lambda function will run.
- **List of Subnet IDs** – The subnets to which a network interface is established. Add subnets corresponding to your Neptune cluster.
- **List of Security Group IDs** – Provide the IDs of security groups that grant write inbound access to your source Neptune DB cluster.
- **List of Route Table IDs** – This is needed to create an Amazon DynamoDB endpoint in your Neptune VPC, if you do not already have one. You must provide a comma-separated list of route table IDs associated with the subnets.
- **Require to create Dynamo DB VPC Endpoint** – A Boolean value that defaults to `true`. You only need to change it to `false` if you have already created a DynamoDB endpoint in your VPC.
- **Require to create Monitoring VPC Endpoint** – A Boolean value that defaults to `true`. You only need to change it to `false` if you have already created a monitoring endpoint in your VPC.

Stream Poller

- **Application Name** – You can generally leave this set to the default (`NeptuneStream`). If you use a different name, it must be unique.

- **Memory size for Lambda Poller** – Used to set the memory size available to the Lambda poller function. The default value is 2,048 megabytes.
- **Lambda Runtime** – The language used in the Lambda function that retrieves items from the Neptune stream. You can set this either to `python3.6` or to `java8`.
- **S3 Bucket having Lambda code artifacts** – Leave this blank unless you are using a custom Lambda polling function that loads from a different S3 bucket.
- **S3 Key corresponding to Lambda Code artifacts** – Leave this blank unless you are using a custom Lambda polling function.
- **Logging level for Lambda** – In general, leave this set to the default value, `INFO`.
- **Managed Policies for Lambda Execution** – In general, leave this blank unless you are using a custom Lambda polling function.
- **Stream Records Handler** – In general, leave this blank unless you are using a custom handler for the records in Neptune streams.
- **Maximum records Fetched from Stream** – You can use this parameter to tune performance. The default (100) is a good place to start. The maximum allowable is 10,000. The higher the number, the fewer network calls are needed to read records from the stream, but the more memory is required to process the records.
- **Max wait time between two Polls (in Seconds)** – Determines how frequently the Lambda poller is invoked to poll the Neptune streams. Set this value to 0 for continuous polling. The maximum value is 3,600 seconds (1 hour). The default value (60 seconds) is a good place to start, depending on how fast your graph data changes.
- **Maximum Continuous polling period (in Seconds)** – Used to set a timeout for the Lambda polling function. It should be between 5 seconds and 900 seconds. The default value (600 seconds) is a good place to start.
- **Step Function Fallback Period** – The number of step-function-fallback-period units to wait for the poller, after which the step function is called through Amazon CloudWatch Events to recover from a failure. The default (5 minutes) is a good place to start.
- **Step Function Fallback Period Unit** – The time units used to measure the preceding Step Function Fallback Period (minutes, hours, days). The default (minutes) is generally sufficient.
- **Data replication scope** – Determines whether to replicate both nodes and edges, or only nodes to OpenSearch (this applies to Gremlin engine data only). The default value (`All`) is generally a good place to start.
- **Ignore OpenSearch missing document error** – Flag to determine whether a missing document error in OpenSearch can be ignored. Missing document errors occur rarely but need manual intervention if not ignored. The default value (`True`) is generally a good place to start.
- **Enable Non-String Indexing** – Flag to enable or disable indexing of fields that do not have string content. If this flag is set to `true`, non-string fields are indexed in OpenSearch, or if `false`, only string fields are indexed. The default is `true`.
- **Properties to exclude from being inserted into OpenSearch** – A comma-delimited list of property or predicate keys to exclude from OpenSearch indexing. If this CFN parameter value is left blank, all the property keys are indexed.
- **Datatypes to exclude from being inserted into OpenSearch** – A comma-delimited list of property or predicate datatypes to exclude from OpenSearch indexing. If this CFN parameter value is left blank, all the property values that can safely be converted to OpenSearch datatypes are indexed.

Neptune Stream

- **Endpoint of source Neptune Stream** – (*Required*) This takes one of two forms:
 - `https://your DB cluster:port/gremlin/stream`
 - `https://your DB cluster:port/sparql/stream`

- **Neptune Query Engine** – Choose Gremlin or SPARQL.
- **Is IAM Auth Enabled?** – If your Neptune DB cluster is using IAM authentication, set this parameter to `true`.
- **Neptune Cluster Resource Id** – If your Neptune DB cluster is using IAM authentication, set this parameter to the cluster resource ID. The resource ID is not the same as the cluster ID. Instead, it takes the form: `cluster-` followed by 28 alpha-numeric characters. It can be found under **Cluster Details** in the Neptune console.

Target OpenSearch cluster

- **Endpoint for OpenSearch service** – (Required) Provide the endpoint for the OpenSearch service in your VPC.
- **Number of Shards for OpenSearch Index** – The default value (5) is generally a good place to start.
- **Number of Replicas for OpenSearch Index** – The default value (1) is generally a good place to start.
- **Geo Location Fields for Mapping** – If you are using geolocation fields, list the property keys here.

Alarm

- **Require to create Cloud watch Alarm** – Set this to `true` if you want to create a CloudWatch alarm for the new stack.
- **Email for Alarm Notifications** – The email address to which alarm notifications should be sent (only needed if alarms are enabled).

Run the AWS CloudFormation Template

Now you can complete the process of provisioning a Neptune streams consumer application instance as follows:

1. In AWS CloudFormation, on the **Specify Stack Details** page, choose **Next**.
2. On the **Options** page, choose **Next**.
3. On the **Review** page, select the first check box to acknowledge that AWS CloudFormation will create IAM resources. Select the second check box to acknowledge `CAPABILITY_AUTO_EXPAND` for the new stack.

Note

`CAPABILITY_AUTO_EXPAND` explicitly acknowledges that macros will be expanded when creating the stack, without prior review. Users often create a change set from a processed template so that the changes made by macros can be reviewed before actually creating the stack. For more information, see the AWS CloudFormation [CreateStack](#) API operation in the *AWS CloudFormation API Reference*.

Then choose **Create**.

To update the stream poller with the latest Lambda artifacts

You can update the stream poller with the latest Lambda code artifacts as follows:

1. In the AWS Management Console, navigate to AWS CloudFormation and select the main parent AWS CloudFormation stack.
2. Select the **Update** option for the stack.
3. Select **Replace current template**.
4. For the template source, choose **Amazon S3 URL** and enter the following S3 URL:

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/  
neptune_to_elastic_search.json
```

5. Select **Next** without changing any AWS CloudFormation parameters.
6. Choose **Update Stack**.

The stack will now update the Lambda artifacts with the most recent ones.

Different approaches to enabling full text search on existing Neptune databases

The best way to enable full text search on an existing Neptune database is generally as follows, provided you can pause your write workloads. The downtime it requires is limited, because creating a clone is relatively fast.

1. Stop all write workloads on the database.
2. Create a clone of the database.
3. Enable streams on the original database.
4. Resume the write workloads.
5. Use [this Neptune tool on github](#) to perform a one-time synchronization from the clone.
6. Use [this AWS CloudFormation template](#) to synchronize from your original database for continuous updating (no configuration change is needed in the template).

If you can't afford to suspend write workloads on your database, here is an approach that requires even less downtime than the recommended approach above, but it needs to be done carefully:

1. Enable streams on the database.
2. Create a clone of the database.
3. Make note of the latest `eventID` for the streams from the clone.
4. Use [this Neptune tool on github](#) to perform a one-time synchronization from the clone.
5. Use [this AWS CloudFormation template](#) to synchronize from your original database for continuous updating. You do need to configure the AWS CloudFormation template: make sure to set the checkpoint in the DynamoDB table using the `eventID` you recorded before starting the continuous synchronization.

Querying custom fields in OpenSearch using Neptune full-text search

The current stream poller can easily be extended to write custom code for handling custom fields, as is explained in detail in the blog post: [Capture graph changes using Neptune Streams](#).

Note

When adding a custom field in OpenSearch, make sure to add the new field as an inner object of a predicate (see [Neptune Full-Text Search Data Model \(p. 478\)](#)).

Neptune Data Model for OpenSearch Data

Amazon Neptune uses a unified JSON document structure for storing both SPARQL and Gremlin data in OpenSearch Service. Each document in OpenSearch corresponds to an entity and stores all the relevant information for that entity. For Gremlin, vertexes and edges are considered entities, so the corresponding OpenSearch documents have information about vertexes, labels, and properties. For SPARQL, subjects can be considered entities, so corresponding OpenSearch documents have information about all the predicate-object pairs in one document.

Note

The Neptune-to-OpenSearch replication implementation only stores string data. However, you can modify it to store other data types.

The unified JSON document structure looks like the following.

```
{
  "entity_id": "Vertex Id/Edge Id/Subject URI",
  "entity_type": [List of Labels/rdf:type object value],
  "document_type": "vertex/edge/rdf-resource"
  "predicates": {
    "Property name or predicate URI": [
      {
        "value": "Property Value or Object Value",
        "graph": "(Only for Sparql) Named Graph Quad is present"
        "language": "(Only for Sparql) rdf:langString"
      },
      {
        "value": "Property Value 2/ Object Value 2",
      }
    ]
  }
}
```

- `entity_id` – Entity unique ID representing the document.
 - For SPARQL, this is the subject URI.
 - For Gremlin, this is the `Vertex_ID` or `Edge_ID`.
- `entity_type` – Represents one or more labels for a vertex or edge, or zero or more `rdf:type` predicate values for a subject.
- `document_type` – Used to specify whether the current document represents a vertex, edge, or `rdf-resource`.
- `predicates` – For Gremlin, stores properties and values for a vertex or edge. For SPARQL, it stores predicate-object pairs.

The property name takes the form `properties.name.value` in OpenSearch. To query it, you have to name it in that form.

- `value` – A property value for Gremlin or an object value for SPARQL.
- `graph` – A named graph for SPARQL.
- `language` – A language tag for a `rdf:langString` literal in SPARQL.

Sample SPARQL OpenSearch Document

Data

```
@prefix dt: <http://example.org/datatype#> .
@prefix ex: <http://example.org/> .
```

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:simone    rdf:type      ex:Person      ex:g1
ex:michael  rdf:type      ex:Person      ex:g1
ex:simone    ex:likes     "spaghetti"    ex:g1

ex:simone    ex:knows     ex:michael     ex:g2  # Not stored in ES
ex:simone    ex:likes     "spaghetti"    ex:g2
ex:simone    ex:status    "La vita è un sogno"@it  ex:g2

ex:simone    ex:age       "40"^^xsd:int  DG      # Not stored in ES
ex:simone    ex:dummy     "testData"^^dt:newDataType  DG
ex:simone    ex:hates     _:bnode        # Not stored in ES
_:bnode      ex:means     "coding"       DG      # Not stored in ES
```

Documents

```
{
  "entity_id": "http://example.org/simone",
  "entity_type": ["http://example.org/Person"],
  "document_type": "rdf-resource"
  "predicates": {
    "http://example.org/likes": [
      {
        "value": "spaghetti",
        "graph": "http://example.org/g1"
      },
      {
        "value": "spaghetti",
        "graph": "http://example.org/g2"
      }
    ]
    "http://example.org/status": [
      {
        "value": "La vita è un sogno",
        "language": "it" // Only present for rdf:langString
      }
    ]
  }
}
```

```
{
  "entity_id" : "http://example.org/michael",
  "entity_type" : ["http://example.org/Person"],
  "document_type": "rdf-resource"
}
```

Sample Gremlin OpenSearch Document

Data

```
# Vertex 1
simone  label    Person    <== Label
simone  likes   "spaghetti" <== Property
simone  likes   "rice"     <== Property
simone  age     40        <== Property

# Vertex 2
michael label    Person    <== Label

# Edge 1
```



```
simone knows michael <== Edge
e1 updated "2019-07-03" <== Edge Property
e1 through "company" <== Edge Property
e1 since 10 <== Edge Property
```

Documents

```
{
  "entity_id": "simone",
  "entity_type": ["Person"],
  "document_type": "vertex",
  "predicates": {
    "likes": [
      {
        "value": "spaghetti"
      },
      {
        "value": "rice"
      }
    ]
  }
}
```

```
{
  "entity_id" : "michael",
  "entity_type" : ["Person"],
  "document_type": "vertex"
}
```

```
{
  "entity_id": "e1",
  "entity_type": ["knows"],
  "document_type": "edge"
  "predicates": {
    "through": [
      {
        "value": "company"
      }
    ]
  }
}
```

Neptune Full-Text Search Parameters

Amazon Neptune uses the following parameters for specifying full-text OpenSearch queries in both Gremlin and SPARQL:

- **queryType** – (*Required*) The type of OpenSearch query. (For a list of query types, see the [OpenSearch documentation](#)). Neptune supports the following OpenSearch query types:
- **simple_query_string** – Returns documents based on a provided query string, using a parser with a limited but fault-tolerant Lucene syntax. This is the default query type.

This query uses a simple syntax to parse and split the provided query string into terms based on special operators. The query then analyzes each term independently before returning matching documents.

While its syntax is more limited than the `query_string` query, the `simple_query_string` query does not return errors for invalid syntax. Instead, it ignores any invalid parts of the query string.

- **match** – The `match` query is the standard query for performing a full-text search, including options for fuzzy matching.
- **prefix** – Returns documents that contain a specific prefix in a provided field.
- **fuzzy** – Returns documents that contain terms similar to the search term, as measured by a Levenshtein edit distance.

An edit distance is the number of one-character changes needed to turn one term into another. These changes can include:

- Changing a character (box to fox).
- Removing a character (black to lack).
- Inserting a character (sic to sick).
- Transposing two adjacent characters (act to cat).

To find similar terms, the fuzzy query creates a set of all possible variations and expansions of the search term within a specified edit distance and then returns exact matches for each of those variants.

- **term** – Returns documents that contain an exact match of a specified term in one of the specified fields.

You can use the `term` query to find documents based on a precise value such as a price, a product ID, or a username.

Warning

Avoid using the `term` query for text fields. By default, OpenSearch changes the values of text fields as part of its analysis, which can make finding exact matches for text field values difficult.

To search text field values, use the `match` query instead.

- **query_string** – Returns documents based on a provided query string, using a parser with a strict syntax (Lucene syntax).

This query uses a syntax to parse and split the provided query string based on operators, such as AND or NOT. The query then analyzes each split text independently before returning matching documents.

You can use the `query_string` query to create a complex search that includes wildcard characters, searches across multiple fields, and more. While versatile, the query is strict and returns an error if the query string includes any invalid syntax.

Warning

Because it returns an error for any invalid syntax, we don't recommend using the `query_string` query for search boxes.

If you don't need to support a query syntax, consider using the `match` query. If you need the features of a query syntax, use the `simple_query_string` query, which is less strict.

- **field** – The field in OpenSearch against which to run the search. This can be omitted only if the `queryType` allows it (as `simple_query_string` and `query_string` do), in which case the search is against all fields. In Gremlin, it is implicit.

Multiple fields can be specified if the query allows it, as do `simple_query_string` and `query_string`.

- **query** – (*Required*) The query to run against OpenSearch. The contents of this field might vary according to the `queryType`. Different `queryTypes` accept different syntaxes, as `Regex` does, for example. In Gremlin, `query` is implicit.
- **maxResults** – The maximum number of results to return. The default is the `index.max_result_window` OpenSearch setting, which itself defaults to 1000. The `maxResults` parameter can specify any number lower than that.

Important

If you set `maxResults` to a value higher than the OpenSearch `index.max_result_window` value and try to retrieve more than `index.max_result_window` results, OpenSearch fails with a Result window is too large error. However, Neptune handles this gracefully without propagating the error. Keep this in mind if you are trying to fetch more than `index.max_result_window` results.

- **minScore** – The minimum score a search result must have to be returned. See [OpenSearch relevance documentation](#) for an explanation of result scoring.
- **batchSize** – Neptune always fetches data in batches (the default batch size is 100). You can use this parameter to tune performance. The batch size cannot exceed the `index.max_result_window` OpenSearch setting, which defaults to 1000.
- **sortBy** – An optional parameter that lets you sort the results returned by OpenSearch by one of the following:

- *A particular string field in the document* –

For example, in a SPARQL query, you could specify:

```
neptune-fts:config neptune-fts:sortBy foaf:name .
```

In a similar Gremlin query, you could specify:

```
.withSideEffect('Neptune#fts.sortBy', 'name')
```

- *A particular non-string field (long, double, etc.) in the document* –

Note that when sorting on a non-string field, you need to append `.value` to the field name to differentiate it from a string field.

For example, in a SPARQL query, you could specify:

```
neptune-fts:config neptune-fts:sortBy foaf:name.value .
```

In a similar Gremlin query, you could specify:

```
.withSideEffect('Neptune#fts.sortBy', 'name.value')
```

- **score** – Sort by match score (the default).

If the `sortOrder` parameter is present but `sortBy` is not present, the results are sorted by `score` in the order specified by `sortOrder`.

- **id** – Sort by ID, which means the SPARQL subject URI or the Gremlin vertex or edge ID.

For example, in a SPARQL query, you could specify:

```
neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_id' .
```

In a similar Gremlin query, you could specify:

```
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_id')
```

- **label** – Sort by label.

For example, in a SPARQL query, you could specify:

```
neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_type' .
```

In a similar Gremlin query, you could specify:

```
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_type')
```

- `doc_type` – Sort by document type (that is, SPARQL or Gremlin).

For example, in a SPARQL query, you could specify:

```
neptune-fts:config neptune-fts:sortBy 'Neptune#fts.document_type' .
```

In a similar Gremlin query, you could specify:

```
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.document_type')
```

By default, OpenSearch results are not sorted and their order is non-deterministic, meaning that the same query may return items in a different order each time it is run. For this reason, if the result set is greater than `max_result_window`, a quite different subset of the total results could be returned every time a query is run. By sorting, however, you can make the results of different runs more directly comparable.

If no `sortOrder` parameter accompanies `sortBy`, descending (`DESC`) order from greatest to least is used.

- **`sortOrder`** – An optional parameter that lets you specify whether OpenSearch results are sorted from least to greatest or from greatest to least (the default):
 - `ASC` – Ascending order, from least to greatest.
 - `DESC` – Descending order, from greatest to least.

This is the default value, used when the `sortBy` parameter is present but no `sortOrder` is specified.

If neither `sortBy` nor `sortOrder` is present, OpenSearch results are not sorted by default.

Non-string OpenSearch indexing in Amazon Neptune

Non-string OpenSearch indexing in Amazon Neptune allows replicating non-string values for predicates to OpenSearch using the stream poller. All predicate values that can safely be converted to a corresponding OpenSearch mapping or datatype is then replicated to OpenSearch.

For non-string indexing to be enabled on a new stack, the `Enable Non-String Indexing` flag in the AWS CloudFormation template must be set to `true`. This is the default setting. To update an existing stack to support non-string indexing, see [Updating an existing stack \(p. 484\)](#) below.

Note

- It is best not to enable non-string indexing on engine versions earlier than **1.0.4.2**.
- OpenSearch queries using regular expressions for field names that match multiple fields, some of which contain string values and others of which contain non-string values, fail with an error. The same thing happens if full-text search queries in Neptune are of that type.

- When sorting by a non-string field, append ".value" to the field name to differentiate it from a string field.

Contents

- [Updating an existing Neptune full-text search stack to support non-string indexing \(p. 484\)](#)
- [Filtering what fields are indexed in Neptune full-text search \(p. 485\)](#)
 - [Filter by property or predicate name \(p. 485\)](#)
 - [Filter by property or predicate value type \(p. 486\)](#)
- [Mapping of SPARQL and Gremlin datatypes to OpenSearch \(p. 487\)](#)
- [Validation of data mappings \(p. 488\)](#)
- [Sample non-string OpenSearch queries in Neptune \(p. 492\)](#)
 - [1. Get all vertices with age greater than 30 and name starting with "Si" \(p. 492\)](#)
 - [2. Get all nodes with age between 10 and 50 and a name with a fuzzy match with "Ronka" \(p. 493\)](#)
 - [3. Get all nodes with a timestamp that falls within the last 25 days \(p. 493\)](#)
 - [4. Get all nodes with a timestamp that falls within a given year and month \(p. 493\)](#)

Updating an existing Neptune full-text search stack to support non-string indexing

If you are already using Neptune full-text search, here are the steps you need to take to support non-string indexing:

1. **Stop the stream poller Lambda function.** This ensures that no new updates are copied during export. Do this by disabling the cloud event rule that invokes the Lambda function:
 - In the AWS Management Console, navigate to CloudWatch.
 - Select **Rules**.
 - Choose the rule with the Lambda stream poller name.
 - Select **disable** to temporarily disable the rule.
2. **Delete the current Neptune index in OpenSearch.** Use the following `curl` query to delete the `amazon_neptune` index from your OpenSearch cluster:

```
curl -X DELETE "your OpenSearch endpoint/amazon_neptune"
```

3. **Start a one-time export from Neptune to OpenSearch.** It is best to set up a new OpenSearch stack at this point, so that new artifacts are picked up for the poller that performs the export.

Follow the steps listed [here in GitHub](#) to start the one-time export of your Neptune data into OpenSearch.

4. **Update the Lambda artifacts for the existing stream poller.** After the export of Neptune data to OpenSearch has completed successfully, take the following steps:
 - In the AWS Management Console, navigate to AWS CloudFormation.
 - Choose the main parent AWS CloudFormation stack.
 - Select the **Update** option for that stack.
 - Select **Replace current template from options**.
 - For the template source, select **Amazon S3 URL**.
 - For the Amazon S3 URL, enter:

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/  
neptune_to_elastic_search.json
```

- Choose **Next** without changing any of the AWS CloudFormation parameters.
 - Select **Update stack**. AWS CloudFormation will replace the Lambda code artifacts for the stream poller with the latest artifacts.
5. **Start the stream poller again.** Do this by enabling the appropriate CloudWatch rule:
- In the AWS Management Console, navigate to CloudWatch.
 - Select **Rules**.
 - Choose the rule with the Lambda stream poller name.
 - Select **enable**.

Filtering what fields are indexed in Neptune full-text search

There are two fields in the AWS CloudFormation template details that let you specify property or predicate keys or datatypes to exclude from OpenSearch indexing:

Filter by property or predicate name

You can use the optional AWS CloudFormation template parameter named `Properties` to exclude from being inserted into Elastic Search Index to provide a comma-delimited list of property or predicate keys to exclude from OpenSearch indexing.

For example, suppose you set this parameter to bob:

```
"Properties to exclude from being inserted into Elastic Search Index" : bob
```

In that case, the stream record of the following Gremlin update query would be dropped rather than going into the index:

```
g.V("1").property("bob", "test")
```

Similarly, you could set the parameter to `http://my/example#bob`:

```
"Properties to exclude from being inserted into Elastic Search Index" : http://my/  
example#bob
```

In that case, the stream record of the following SPARQL update query would be dropped rather than going into the index:

```
PREFIX ex: <http://my/example#>  
INSERT DATA { ex:s1 ex:bob "test"}.
```

If you don't enter anything in this AWS CloudFormation template parameter, all the property keys not otherwise excluded will be indexed.

Filter by property or predicate value type

You can use the optional AWS CloudFormation template parameter named `Datatypes` to exclude from being inserted into Elastic Search Index to provide a comma-delimited list of property or predicate value datatypes to exclude from OpenSearch indexing.

For SPARQL, you don't need to list the full XSD type URI, you can just list the datatype token. Valid datatype tokens that you can list are:

- `string`
- `boolean`
- `float`
- `double`
- `dateTime`
- `date`
- `time`
- `byte`
- `short`
- `int`
- `long`
- `decimal`
- `integer`
- `nonNegativeInteger`
- `nonPositiveInteger`
- `negativeInteger`
- `unsignedByte`
- `unsignedShort`
- `unsignedInt`
- `unsignedLong`

For Gremlin, valid datatypes to list are:

- `string`
- `date`
- `bool`
- `byte`
- `short`
- `int`
- `long`
- `float`
- `double`

For example, suppose you set this parameter to `string`:

```
"Datatypes to exclude from being inserted into Elastic Search Index" : string
```

In that case, the stream record of the following Gremlin update query would be dropped rather than going into the index:

```
g.V("1").property("myStringval", "testvalue")
```

Similarly, you could set the parameter to `int`:

```
"Datatypes to exclude from being inserted into Elastic Search Index" : int
```

In that case, the stream record of the following SPARQL update query would be dropped rather than going into the index:

```
PREFIX ex: <http://my/example#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
INSERT DATA { ex:s1 ex:bob "11"^^xsd:int }.
```

If you don't enter anything in this AWS CloudFormation template parameter, all the properties whose values can be safely converted to OpenSearch equivalents will be indexed. Listed types that are unsupported by the query language are ignored.

Mapping of SPARQL and Gremlin datatypes to OpenSearch

New datatype mappings in OpenSearch are created based on the datatype being use in the property or object. Because some fields contain values of different types, the initial mapping may exclude some values of the field.

Neptune datatypes map to OpenSearch datatypes as follows:

SPARQL types	Gremlin types	OpenSearch types
XSD:int	byte	long
XSD:unsignedInt	short	
XSD:integer	int	
XSD:byte	long	
XSD:unsignedByte		
XSD:short		
XSD:unsignedShort		
XSD:long		
XSD:unsignedLong		
XSD:float	float	double
XSD:double	double	
XSD:decimal		
XSD:boolean	bool	boolean
XSD:datetime	date	date
XSD:date		

SPARQL types	Gremlin types	OpenSearch types
XSD:string XSD:time	string	text
Custom datatype	N/A	text
Any other datatype	N/A	text

For example, the following Gremlin update query causes a new mapping for "newField" to be added to OpenSearch, namely { "type" : "double" }:

```
g.V("1").property("newField" 10.5)
```

Similarly, the following SPARQL update query causes a new mapping for "ex:byte" to be added to OpenSearch, namely { "type" : "long" }:

```
PREFIX ex: <http://my/example#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

INSERT DATA { ex:test ex:byte "123"^^xsd:byte }.
```

Note

As you can see, an item mapped from Neptune to OpenSearch may end up with a different datatype in OpenSearch than it has in Neptune. However, there is an explicit text field in OpenSearch, "datatype", that records the datatype that the item has in Neptune.

Validation of data mappings

Data is replicated to OpenSearch from Neptune using this process:

- If a mapping for the field in question is already present in OpenSearch:
 - If the data can be safely converted to the existing mapping using data validation rules, then store the field in OpenSearch.
 - If not, drop the corresponding stream update record.
- If there is no existing mapping for the field in question, find an OpenSearch datatype corresponding to the field's datatype in Neptune.
 - If the field data can be safely converted to the OpenSearch datatype using data validation rules, then store the new mapping and field data in OpenSearch.
 - If not, drop the corresponding stream update record.

Values are validated against equivalent OpenSearch types or existing OpenSearch mappings rather than the Neptune types. For example, validation for the value "123" in "123"^^xsd:int is done against the long type rather than the int type.

Although Neptune attempts to replicate all data to OpenSearch, there are cases where datatypes in OpenSearch are totally different from the ones in Neptune, and in such cases records are skipped rather than being indexed in OpenSearch.

For example, in Neptune one property can have multiple values of different types, whereas in OpenSearch a field must have the same type across the index.

By enabling debug logs, you can view what records have been dropped during export from Neptune to OpenSearch. An example of a debug log entry is:

```
Dropping Record : Data type not a valid Gremlin type
<Record>
```

Datatypes are validated as follows:

- **text** – All values in Neptune can safely be mapped to text in OpenSearch.
- **long** – The following rules for Neptune datatypes apply when the OpenSearch mapping type is long (in the examples below, it is assumed that "testLong" has a long mapping type):
 - **boolean** – Invalid, cannot be converted, and the corresponding stream update record is dropped.

Invalid Gremlin examples are:

```
"testLong" : true.
"testLong" : false.
```

Invalid SPARQL examples are:

```
":testLong" : "true"^^xsd:boolean
":testLong" : "false"^^xsd:boolean
```

- **datetime** – Invalid, cannot be converted, and the corresponding stream update record is dropped.

An invalid Gremlin example is:

```
":testLong" : datetime('2018-11-04T00:00:00').
```

An invalid SPARQL example is:

```
":testLong" : "2016-01-01"^^xsd:date
```

- **float, double, or decimal** – If the value in Neptune is an integer that can fit in 64 bits, it is valid and is stored in OpenSearch as a long, but if it has a fractional part, or is a NaN or an INF, or is larger than 9,223,372,036,854,775,807 or smaller than -9,223,372,036,854,775,808, then it is not valid and the corresponding stream update record is dropped.

Valid Gremlin examples are:

```
"testLong" : 145.0.
":testLong" : 123
":testLong" : -9223372036854775807
```

Valid SPARQL examples are:

```
":testLong" : "145.0"^^xsd:float
":testLong" : 145.0
":testLong" : "145.0"^^xsd:double
":testLong" : "145.0"^^xsd:decimal
":testLong" : "-9223372036854775807"
```

Invalid Gremlin examples are:

```
"testLong" : 123.45
":testLong" : 9223372036854775900
```

Invalid SPARQL examples are:

```
":testLong" : 123.45
":testLong" : 9223372036854775900
":testLong" : "123.45"^^xsd:float
":testLong" : "123.45"^^xsd:double
":testLong" : "123.45"^^xsd:decimal
```

- **string** – If the value in Neptune is a string representation of an integer that can be contained in a 64-bit integer, then it is valid and is converted to a long in OpenSearch. Any other string value is invalid for an Elasticsearch long mapping, and the corresponding stream update record is dropped.

Valid Gremlin examples are:

```
"testLong" : "123".
":testLong" : "145.0"
":testLong" : "-9223372036854775807"
```

Valid SPARQL examples are:

```
":testLong" : "145.0"^^xsd:string
":testLong" : "-9223372036854775807"^^xsd:string
```

Invalid Gremlin examples are:

```
"testLong" : "123.45"
":testLong" : "9223372036854775900"
":testLong" : "abc"
```

Invalid SPARQL examples are:

```
":testLong" : "123.45"^^xsd:string
":testLong" : "abc"
":testLong" : "9223372036854775900"^^xsd:string
```

- **double** – If the OpenSearch mapping type is double, the following rules apply (here, the "testDouble" field is assumed to have a double mapping in OpenSearch):
- **boolean** – Invalid, cannot be converted, and the corresponding stream update record is dropped.

Invalid Gremlin examples are:

```
"testDouble" : true.
"testDouble" : false.
```

Invalid SPARQL examples are:

```
":testDouble" : "true"^^xsd:boolean
":testDouble" : "false"^^xsd:boolean
```

- **datetime** – Invalid, cannot be converted, and the corresponding stream update record is dropped.

An invalid Gremlin example is:

```
":testDouble" : datetime('2018-11-04T00:00:00').
```

An invalid SPARQL example is:

```
":testDouble" : "2016-01-01"^^xsd:date
```

- **Floating-point NaN or INF** – If the value in SPARQL is a floating-point NaN or INF, then it is not valid and the corresponding stream update record is dropped.

Invalid SPARQL examples are:

```
" :testDouble" : "NaN"^^xsd:float  
":testDouble" : "NaN"^^double  
":testDouble" : "INF"^^double  
":testDouble" : "-INF"^^double
```

- **number or numeric string** – If the value in Neptune is any other number or numeric string representation of a number that can safely be expressed as a double, then it is valid and is converted to a double in OpenSearch. Any other string value is invalid for an OpenSearch double mapping, and the corresponding stream update record is dropped.

Valid Gremlin examples are:

```
"testDouble" : 123  
":testDouble" : "123"  
":testDouble" : 145.67  
":testDouble" : "145.67"
```

Valid SPARQL examples are:

```
":testDouble" : 123.45  
":testDouble" : 145.0  
":testDouble" : "123.45"^^xsd:float  
":testDouble" : "123.45"^^xsd:double  
":testDouble" : "123.45"^^xsd:decimal  
":testDouble" : "123.45"^^xsd:string
```

An invalid Gremlin example is:

```
":testDouble" : "abc"
```

An Invalid SPARQL examples is:

```
":testDouble" : "abc"
```

- **date** – If the OpenSearch mapping type is date, Neptune date and dateTime value are valid, as is any string value that can be parsed successfully to a dateTime format.

Valid examples in either Gremlin or SPARQL are:

```
Date(2016-01-01)  
"2016-01-01" "  
2003-09-25T10:49:41"  
"2003-09-25T10:49"
```

```
"2003-09-25T10"  
"20030925T104941-0300"  
"20030925T104941"  
"2003-Sep-25" "  
Sep-25-2003"  
"2003.Sep.25"  
"2003/09/25"  
"2003 Sep 25" "  
Wed, July 10, '96"  
"Tuesday, April 12, 1952 AD 3:30:42pm PST"  
"123"  
"-123"  
"0"  
"-0"  
"123.00"  
"-123.00"
```

Invalid examples are:

```
123.45  
True  
"abc"
```

Sample non-string OpenSearch queries in Neptune

Neptune does not currently support OpenSearch range queries directly. However, you can achieve the same effect using Lucene syntax and `query-type="query_string"`, as you can see in the following sample queries.

1. Get all vertices with age greater than 30 and name starting with "Si"

In Gremlin:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')  
  .withSideEffect("Neptune#fts.queryType", "query_string")  
  .V().has('*', 'Neptune#fts predicates.age.value:>30 && predicates.name.value:Si*');
```

In SPARQL:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>  
SELECT * WHERE {  
  SERVICE neptune-fts:search {  
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .  
    neptune-fts:config neptune-fts:queryType 'query_string' .  
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*age.value:>30 AND  
predicates.\\*foaf\\*name.value:Si*" .  
    neptune-fts:config neptune-fts:field '*' .  
    neptune-fts:config neptune-fts:return ?res .  
  }  
}
```

Here, `"*foaf*age` is used instead of the full URI for brevity. This regular expression will retrieve all fields have both `foaf` and `age` in the URI.

2. Get all nodes with age between 10 and 50 and a name with a fuzzy match with "Ronka"

In Gremlin:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
  .withSideEffect("Neptune#fts.queryType", "query_string")
  .V().has('*', 'Neptune#fts predicates.age.value:[10 TO 50] AND
  predicates.name.value:Ronka~');
```

In SPARQL:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*age.value:[10 TO 50] AND
    predicates.\\*foaf\\*name.value:Ronka~" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

3. Get all nodes with a timestamp that falls within the last 25 days

In Gremlin:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
  .withSideEffect("Neptune#fts.queryType", "query_string")
  .V().has('*', 'Neptune#fts predicates.timestamp.value:>now-25d');
```

In SPARQL:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*timestamp.value:>now-25d~" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

4. Get all nodes with a timestamp that falls within a given year and month

In Gremlin, using [date math expressions](#) in Lucene syntax, for December 2020:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
```

```
.withSideEffect("Neptune#fts.queryType", "query_string")  
.V().has('*', 'Neptune#fts predicates.timestamp.value:>2020-12');
```

A Gremlin alternative:

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')  
.withSideEffect("Neptune#fts.queryType", "query_string")  
.V().has('*', 'Neptune#fts predicates.timestamp.value:[2020-12 TO 2021-01]');
```

Full-Text-Search Query Execution in Amazon Neptune

In a query that includes full-text-search, Neptune tries to put the full-text-search calls first, before other parts of the query. This reduces the number of calls to OpenSearch and in most cases significantly improves performance. However, this is by no means a hard-and-fast rule. There are situations, for example, where a `PatternNode` or `UnionNode` may precede a full-text search call.

Consider the following Gremlin query to a database that contains 100,000 instances of `Person`:

```
g.withSideEffect('Neptune#fts.endpoint', 'your-es-endpoint-URL')  
.hasLabel('Person')  
.has('name', 'Neptune#fts marcello~');
```

If this query were executed in the order in which the steps appear then 100,000 solutions would flow into OpenSearch, causing hundreds of OpenSearch calls. In fact, Neptune calls OpenSearch first and then joins results with the Neptune results. In most cases, this is much faster than executing the query in the original order.

You can prevent this re-ordering of query-step execution using the [noReordering query hint \(p. 281\)](#):

```
g.withSideEffect('Neptune#fts.endpoint', 'your-es-endpoint-URL')  
.withSideEffect('Neptune#noReordering', true)  
.hasLabel('Person')  
.has('name', 'Neptune#fts marcello~');
```

In this second case, the `.hasLabel` step is executed first and the `.has('name', 'Neptune#fts marcello~')` step second.

For another example, consider a SPARQL query against the same kind of data:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>  
SELECT ?person WHERE {  
  ?person rdf:type foaf:Person .  
  SERVICE neptune-fts:search {  
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .  
    neptune-fts:config neptune-fts:field foaf:name .  
    neptune-fts:config neptune-fts:query 'mike' .  
    neptune-fts:config neptune-fts:return ?person .  
  }  
}
```

Here again, Neptune executes the `SERVICE` part of the query first, and then joins the results with the `Person` data. You can suppress this behavior using the [joinOrder query hint \(p. 359\)](#):

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?person WHERE {
  hint:Query hint:joinOrder "Ordered" .
  ?person rdf:type foaf:Person .
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mike' .
    neptune-fts:config neptune-fts:return ?person .
  }
}
```

Again, in the second query the parts are executed in the order they appear in the query.

Sample SPARQL queries using full-text search in Neptune

The following are some sample SPARQL queries that use full-text search in Amazon Neptune.

SPARQL match query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'match' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'michael' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL prefix query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'prefix' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mich' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL fuzzy query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```



```
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'fuzzy' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mikael' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL term query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'term' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'Dr. Kunal' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL query_string query example

This query specifies multiple fields.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael- OR rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:field foaf:surname .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL simple_query_string query example

The following query specifies fields using the wildcard (*) character.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'simple_query_string' .
    neptune-fts:config neptune-fts:query 'mikael- | rondelli' .
    neptune-fts:config neptune-fts:field '*'.
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL sort by string field query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael- | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy foaf:name .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL sort by non-string field query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael- | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name.value .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy dc:date.value .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL sort by ID query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael- | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_id' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL sort by label query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael- | rondelli' .
  }
}
```

```
neptune-fts:config neptune-fts:field foaf:name .
neptune-fts:config neptune-fts:sortOrder 'asc' .
neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_type' .
neptune-fts:config neptune-fts:return ?res .
}
}
```

SPARQL sort by doc_type query example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael- | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.document_type' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

Example of using Lucene syntax in SPARQL

Lucene syntax is only supported for query_string queries in OpenSearch.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'predicates.\\foaf\\name.value:micheal AND
predicates.\\foaf\\surname.value:sh' .
    neptune-fts:config neptune-fts:field '' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

Using Neptune Full-Text Search in Gremlin Queries

NeptuneSearchStep enables full-text search queries for the part of a Gremlin traversal that is not converted into Neptune steps. For example, consider a query like the following.

```
g.withSideEffect("Neptune#fts.endpoint", "your-es-endpoint-URL")
  .v()
  .tail(100)
  .has("name", "Neptune#fts mark*")           <== # Limit the search on name
```

This query is converted into the following optimized traversal in Neptune.

```
Neptune steps:
[
  NeptuneGraphQueryStep(Vertex) {
```

```
    JoinGroupNode {
      PatternNode[(?1, <-label>, ?2, <->) . project distinct ?1 .],
      {estimatedCardinality=INFINITY}
    }, annotations={path=[Vertex(?1):GraphStep], maxVarId=4}
  },
  NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [NeptuneTailGlobalStep(100),
  NeptuneTinkerpopTraverserConverterStep, NeptuneSearchStep {
    JoinGroupNode {
      SearchNode[(idVar=?3, query=mark*, field=name) . project ask .], {endpoint=your-
OpenSearch-endpoint-URL}
    }
    JoinGroupNode {
      SearchNode[(idVar=?3, query=mark*, field=name) . project ask .], {endpoint=your-
OpenSearch-endpoint-URL}
    }
  }]
}]
```

The following examples are of Gremlin queries against air-routes data:

Gremlin Basic Case-Insensitive match Query

```
g.withSideEffect("Neptune#fts.endpoint",
  "your-OpenSearch-endpoint-URL")
.V().has("city", "Neptune#fts dallas")

==>v[186]
==>v[8]
```

Gremlin match Query

```
g.withSideEffect("Neptune#fts.endpoint",
  "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'match')
.V().has("city", "Neptune#fts southampton")
.local(values('code', 'city').fold())
.limit(5)

==>[SOU, Southampton]
```

Gremlin fuzzy Query

```
g.withSideEffect("Neptune#fts.endpoint",
  "your-OpenSearch-endpoint-URL")
.V().has("city", "Neptune#fts allas~").values('city').limit(5)

==>Dallas
==>Dallas
==>Walla Walla
==>Velas
==>Altai
```

Gremlin query_string fuzzy Query

```
g.withSideEffect("Neptune#fts.endpoint",
```

```
        "your-OpenSearch-endpoint-URL")
    .withSideEffect('Neptune#fts.queryType', 'query_string')
    .V().has("city","Neptune#fts allas~").values('city').limit(5)

==>Dallas
==>Dallas
```

Gremlin query_string Regular Expression Query

```
g.withSideEffect("Neptune#fts.endpoint",
    "your-OpenSearch-endpoint-URL")
    .withSideEffect('Neptune#fts.queryType', 'query_string')
    .V().has("city","Neptune#fts /[dp]allas/").values('city').limit(5)

==>Dallas
==>Dallas
```

Gremlin Hybrid Query

This query uses a Neptune internal index and the OpenSearch index in the same query.

```
g.withSideEffect("Neptune#fts.endpoint",
    "your-OpenSearch-endpoint-URL")
    .V().has("region","GB-ENG")
        .has('city','Neptune#fts L*')
        .values('city')
        .dedup()
        .limit(10)

==>London
==>Leeds
==>Liverpool
==>Land's End
```

Gremlin Query Using '+' and '-' Operators

```
g.withSideEffect("Neptune#fts.endpoint",
    "your-OpenSearch-endpoint-URL")
    .V().has('desc','Neptune#fts regional municipal')
        .local(values('code','desc').fold())
        .limit(100)

==>[HYA, Barnstable Municipal Boardman Polando Field]
==>[SPS, Sheppard Air Force Base-Wichita Falls Municipal Airport]
==>[ABR, Aberdeen Regional Airport]
==>[SLK, Adirondack Regional Airport]
==>[BFD, Bradford Regional Airport]
==>[EAR, Kearney Regional Airport]
==>[ROT, Rotorua Regional Airport]
==>[YHD, Dryden Regional Airport]
==>[TEX, Telluride Regional Airport]
==>[WOL, Illawarra Regional Airport]
==>[TUP, Tupelo Regional Airport]
==>[COU, Columbia Regional Airport]
==>[MHK, Manhattan Regional Airport]
==>[BJI, Bemidji Regional Airport]
==>[HAS, Hail Regional Airport]
==>[ALO, Waterloo Regional Airport]
==>[SHV, Shreveport Regional Airport]
```

```
==>[ABI, Abilene Regional Airport]
==>[GIZ, Jizan Regional Airport]
==>[USA, Concord Regional Airport]
==>[JMS, Jamestown Regional Airport]
==>[COS, City of Colorado Springs Municipal Airport]
==>[PKB, Mid Ohio Valley Regional Airport]
```

Gremlin Query Using query_string With '+' and '-' Operators

Although the `query_string` query type is much less forgiving than the default `simple_query_string` type, it does allow for more precise queries. The first query below uses `query_string`, while the second use the default `simple_query_string`:

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has('desc', 'Neptune#fts +London -(Stansted|Gatwick)')
    .local(values('code', 'desc').fold())
    .limit(10)

==>[LHR, London Heathrow]
==>[YXU, London Airport]
==>[LTN, London Luton Airport]
==>[SEN, London Southend Airport]
==>[LCY, London City Airport]
```

Notice how `simple_query_string` in the examples below quietly ignores the '+' and '-' operators:

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has('desc', 'Neptune#fts +London -(Stansted|Gatwick)')
    .local(values('code', 'desc').fold())
    .limit(10)

==>[LHR, London Heathrow]
==>[YXU, London Airport]
==>[LGW, London Gatwick]
==>[STN, London Stansted Airport]
==>[LTN, London Luton Airport]
==>[SEN, London Southend Airport]
==>[LCY, London City Airport]
==>[SKG, Thessaloniki Macedonia International Airport]
==>[ADB, Adnan Menderes International Airport]
==>[BTV, Burlington International Airport]
```

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has('desc', 'Neptune#fts +(regional|municipal) -(international|bradford)')
    .local(values('code', 'desc').fold())
    .limit(10)

==>[CZH, Corozal Municipal Airport]
==>[MMU, Morristown Municipal Airport]
==>[YBR, Brandon Municipal Airport]
==>[RDD, Redding Municipal Airport]
==>[VIS, Visalia Municipal Airport]
==>[AIA, Alliance Municipal Airport]
```

```
==>[CDR, Chadron Municipal Airport]
==>[CVN, Clovis Municipal Airport]
==>[SDY, Sidney Richland Municipal Airport]
==>[SGU, St George Municipal Airport]
```

Gremlin query_string Query With AND and OR Operators

```
g.withSideEffect("Neptune#fts.endpoint",
  "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has('desc', 'Neptune#fts (St AND George) OR (St AND Augustin)')
    .local(values('code', 'desc').fold())
    .limit(10)

==>[YIF, St Augustin Airport]
==>[STG, St George Airport]
==>[SGO, St George Airport]
==>[SGU, St George Municipal Airport]
```

Gremlin term Query

```
g.withSideEffect("Neptune#fts.endpoint",
  "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'term')
  .V().has("SKU", "Neptune#fts ABC123DEF9")
    .local(values('code', 'city').fold())
    .limit(5)

==>[AUS, Austin]
```

Gremlin prefix Query

```
g.withSideEffect("Neptune#fts.endpoint",
  "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'prefix')
  .V().has("icao", "Neptune#fts ka")
    .local(values('code', 'icao', 'city').fold())
    .limit(5)

==>[AZO, KAZO, Kalamazoo]
==>[APN, KAPN, Alpena]
==>[ACK, KACK, Nantucket]
==>[ALO, KALO, Waterloo]
==>[ABI, KABI, Abilene]
```

Using Lucene Syntax in Neptune Gremlin

In Neptune Gremlin, you can also write very powerful queries using the Lucene query syntax. Note that Lucene syntax is only supported for `query_string` queries in OpenSearch.

Assume the following data:

```
g.addV("person")
  .property(T.id, "p1")
```

```
.property("name", "simone")
.property("surname", "rondelli")

g.addV("person")
  .property(T.id, "p2")
  .property("name", "simone")
  .property("surname", "sengupta")

g.addV("developer")
  .property(T.id, "p3")
  .property("name", "simone")
  .property("surname", "rondelli")
```

Using Lucene syntax, which is invoked when the `queryType` is `query_string`, you can search this data by name and surname as follows:

```
g.withSideEffect("Neptune#fts.endpoint", "es_endpoint")
  .withSideEffect("Neptune#fts.queryType", "query_string")
  .V()
  .has("*", "Neptune#fts predicates.name.value:simone AND
  predicates.surname.value:rondelli")

==> v[p1], v[p3]
```

Note that in the `has()` step above, the field is replaced by `"*"`). Actually, any value placed there is overridden by the fields that you access within the query. You access the name field using `predicates.name.value`, because that is how the data model is structured.

You can search by name, surname and label, as follows:

```
g.withSideEffect("Neptune#fts.endpoint", getEsEndpoint())
  .withSideEffect("Neptune#fts.queryType", "query_string")
  .V()
  .has("*", "Neptune#fts predicates.name.value:simone AND
  predicates.surname.value:rondelli AND entity_type:person")

==> v[p1]
```

The label is accessed using `entity_type`, again because that is how the data model is structured.

You can also include nesting conditions:

```
g.withSideEffect("Neptune#fts.endpoint", getEsEndpoint())
  .withSideEffect("Neptune#fts.queryType", "query_string")
  .V()
  .has("*", "Neptune#fts (predicates.name.value:simone AND
  predicates.surname.value:rondelli AND entity_type:person) OR
  predicates.surname.value:sengupta")

==> v[p1], v[p2]
```

Inserting a modern TinkerPop graph

```
g.addV('person').property(T.id, '1').property('name', 'marko').property('age', 29)
.addV('personr').property(T.id, '2').property('name', 'vadas').property('age', 27)
.addV('software').property(T.id, '3').property('name', 'lop').property('lang', 'java')
.addV('person').property(T.id, '4').property('name', 'josh').property('age', 32)
.addV('software').property(T.id, '5').property('name', 'ripple').property('lang', 'java')
.addV('person').property(T.id, '6').property('name', 'peter').property('age', 35)
```



```
g.V('1').as('a').V('2').as('b').addE('knows').from('a').to('b').property('weight', 0.5f).property(T.id, '7')
.V('1').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight', 0.4f).property(T.id, '9')
.V('4').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight', 0.4f).property(T.id, '11')
.V('4').as('a').V('5').as('b').addE('created').from('a').to('b').property('weight', 1.0f).property(T.id, '10')
.V('6').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight', 0.2f).property(T.id, '12')
.V('1').as('a').V('4').as('b').addE('knows').from('a').to('b').property('weight', 1.0f).property(T.id, '8')
```

Sort by string field value example

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'query_string')
.withSideEffect('Neptune#fts.sortOrder', 'asc')
.withSideEffect('Neptune#fts.sortBy', 'name')
.V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

Sort by non-string field value example

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'query_string')
.withSideEffect('Neptune#fts.sortOrder', 'asc')
.withSideEffect('Neptune#fts.sortBy', 'age.value')
.V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

Sort by ID field value example

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'query_string')
.withSideEffect('Neptune#fts.sortOrder', 'asc')
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_id')
.V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

Sort by label field value example

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'query_string')
.withSideEffect('Neptune#fts.sortOrder', 'asc')
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_type')
.V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

Sort by document_type field value example

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'query_string')
.withSideEffect('Neptune#fts.sortOrder', 'asc')
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.document_type')
.V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

Troubleshooting Neptune full-text search

To diagnose issues with replication from Neptune to OpenSearch, consult the CloudWatch Logs for your poller Lambda function. These logs provide details about the number of records read from the stream and the number of records replicated successfully to OpenSearch.

You can also change the LOGGING level for your Lambda function by changing the `LogLevel` environment variable.

Note

With `LogLevel` set to `DEBUG`, you can view additional details, such as dropped stream records and the reason why each was dropped, while replicating data by `StreamPoller` from Neptune to OpenSearch. This can be useful if you find you are missing records.

The Neptune streams consumer application publishes two metrics on CloudWatch that can also help you diagnose problems:

- `StreamRecordsProcessed` – The number of records processed by the application per unit of time. Helpful in tracking the application run rate.
- `StreamLagTime` – The time difference in milliseconds between the current time and the commit time of a stream record being processed. This metric shows how much the consumer application is lagging behind.

In addition, all the metrics related to the replication process are exposed in a dashboard in CloudWatch under the same name as the `ApplicationName` provided when you instantiated the application using the CloudWatch template.

You can also choose to create a CloudWatch alarm that is triggered whenever polling fails more than twice in a row. Do this by setting the `CreateCloudWatchAlarm` field to `true` when you instantiate the application. Then specify the email addresses that you want to be notified when the alarm is triggered.

Troubleshooting a process that fails while reading records from the stream

If a process fails while reading records from the stream, make sure that you have the following:

- The stream is enabled on your cluster.
- The Neptune stream endpoint is in the correct format:
 - For Gremlin: `https://your cluster endpoint:your cluster port/gremlin/stream`
 - For SPARQL: `https://your cluster endpoint:your cluster port/sparql/stream`
- The DynamoDB endpoint is configured for your VPC.
- The monitoring endpoint is configured for your VPC subnets.

Troubleshooting a process that fails while writing data to OpenSearch

If a process fails while writing records to OpenSearch, make sure that you have the following:

- Your OpenSearch version is 7.0 or above.
- OpenSearch can be accessed from the poller Lambda function in your VPC.

- The security policy attached to OpenSearch allows inbound HTTP/HTTPS requests.

What to do if the stream poller is continuously throwing `ExpiredEventException` errors

This problem is fixed in engine release versions **1.0.4.2** and higher.

Using AWS Lambda functions in Amazon Neptune

AWS Lambda functions have many uses in Amazon Neptune applications. Here we provide general guidance for using Lambda functions with any of the popular Gremlin drivers and language variants, and specific examples of Lambda functions written in Java, JavaScript, and Python.

Note

The best way to use Lambda functions with Neptune has changed with recent engine releases. Neptune used to leave idle connections open long after a Lambda execution context had been recycled, potentially leading to a resource leak on the server. To mitigate this, we used to recommend opening and closing a connection with each Lambda invocation. Starting with engine version 1.0.3.0, however, the idle connection timeout has been reduced so that connections no longer leak after an inactive Lambda execution context has been recycled, so we now recommend using a single connection for the duration of the execution context. This should include some error handling and back-off-and-retry boilerplate code to handle connections being closed unexpectedly.

Managing Gremlin WebSocket connections in AWS Lambda functions

If you use a Gremlin language variant to query Neptune, the driver connects to the database using a WebSocket connection. WebSockets are designed to support long-lived client-server connection scenarios. AWS Lambda, on the other hand, is designed to support relatively short-lived and stateless executions. This mismatch in design philosophy can lead to some unexpected issues when using Lambda to query Neptune.

An AWS Lambda function runs in an [execution context](#) which isolates the function from other functions. The execution context is created the first time the function is invoked and may be reused for subsequent invocations of the same function.

Any one execution context is never used to handle multiple concurrent invocations of the function, however. If your function is invoked simultaneously by multiple clients, Lambda [spins up an additional execution context](#) for each instance of the function. All these new execution contexts may in turn be reused for subsequent invocations of the function.

At some point, Lambda recycles execution contexts, particularly if they have been inactive for some time. AWS Lambda exposes the execution context lifecycle, including the `Init`, `Invoke` and `Shutdown` phases, through [Lambda extensions](#). Using these extensions, you can write code that cleans up external resources such as database connections when the execution context is recycled.

A common best practice is to [open the database connection outside the Lambda handler function](#) so that it can be reused with each handler call. If the database connection drops at some point, you can reconnect from inside the handler. However, there is a danger of connection leaks with this approach. If an idle connection stays open long after an execution context is destroyed, intermittent or bursty Lambda invocation scenarios can gradually leak connections and exhaust database resources.

Neptune connection limits and connection timeouts have changed with newer engine releases. Previously, every instance supported up to 60,000 WebSocket connections. Now, the maximum number of concurrent WebSocket connections per Neptune instance [varies with the instance type](#).

Also, starting with engine release 1.0.3.0, Neptune reduced the idle timeout for connections from one hour down to approximately 20 minutes. If a client doesn't close a connection, the connection is closed automatically after a 20- to 25-minute idle timeout. AWS Lambda doesn't document execution context lifetimes, but experiments show that the new Neptune connection timeout aligns well with inactive Lambda execution context timeouts. By the time an inactive execution context is recycled, there's a good chance its connection has already been closed by Neptune, or will be closed soon afterwards.

Recommendations for using AWS Lambda with Amazon Neptune Gremlin

We now recommend using a single connection and graph traversal source for the entire lifetime of a Lambda execution context, rather than one for each function invocation (every function invocation handles only one client request). Because concurrent client requests are handled by different function instances running in separate execution contexts, there's no need to maintain a pool of connections to handle concurrent requests inside a function instance. If the Gremlin driver you're using has a connection pool, configure it to use just one connection.

To handle connection failures, use retry logic around each query. Even though the goal is to maintain a single connection for the lifetime of an execution context, unexpected network events can cause that connection to be terminated abruptly. Such connection failures manifest as different errors depending on which driver you are using. You should code your Lambda function to handle these connection issues and attempt a reconnection if necessary.

Some Gremlin drivers automatically handle reconnections. The Java driver, for example, automatically attempts to reestablish connectivity to Neptune on behalf of your client code. With this driver, your function code only needs to back off and retry the query. The JavaScript and Python drivers, by contrast, do not implement any automatic reconnection logic, so with these drivers your function code must try to reconnect after backing off, and only retry the query once the connection has been re-established.

Code examples here do include reconnection logic rather than assume that the client is taking care of it.

Recommendations for using Gremlin write-requests in Lambda

If your Lambda function modifies graph data, consider adopting a back-off-and-retry strategy to handle the following exceptions:

- **ConcurrentModificationException** – The Neptune transaction semantics mean that write requests sometimes fail with a `ConcurrentModificationException`. In these situations, try an exponential back-off-based retry mechanism.
- **ReadOnlyViolationException** – Because the cluster topology can change at any moment as a result of planned or unplanned events, write responsibilities may migrate from one instance in the cluster to another. If your function code attempts to send a write request to an instance that is no longer the primary (writer) instance, the request fails with a `ReadOnlyViolationException`. When this happens, close the existing connection, reconnect to the cluster endpoint, and then retry the request.

Also, if you use a back-off-and-retry strategy to handle write request issues, consider implementing idempotent queries for create and update requests (for example, using `fold().coalesce().unfold()`).

Recommendations for using Gremlin read-requests in Lambda

If you have one or more read replicas in your cluster, it's a good idea to balance read requests across these replicas. One option is to use the [reader endpoint \(p. 78\)](#). The reader endpoint balances connections across replicas even if the cluster topology changes when you add or remove replicas, or promote a replica to become the new primary instance.

However, using the reader endpoint can result in an uneven use of cluster resources in some circumstances. The reader endpoint works by periodically changing the host that the DNS entry points to. If a client opens a lot of connections before the DNS entry changes, all the connection requests are sent to a single Neptune instance. This can be the case with a high-throughput Lambda scenario where a large number of concurrent requests to your Lambda function causes multiple execution contexts to be created, each with its own connection. If those connections are all created nearly simultaneously, the connections are likely to all point to the same replica in the cluster, and to stay pointing to that replica until the execution contexts are recycled.

One way you can distribute requests across instances is to configure your Lambda function to connect to an instance endpoint, chosen at random from a list of replica instance endpoints, rather than the reader endpoint. The downside of this approach is that it requires the Lambda code to handle changes in the cluster topology by monitoring the cluster and updating the endpoint list whenever the membership of the cluster changes.

If you are writing a Java Lambda function that needs to balance read requests across instances in your cluster, you can use the [Gremlin client for Amazon Neptune](#), a Java Gremlin client that is aware of your cluster topology and which fairly distributes connections and requests across a set of instances in a Neptune cluster. [This blog post](#) includes a sample Java Lambda function that uses the Gremlin client for Amazon Neptune.

Factors that may slow down cold starts of Neptune Gremlin Lambda functions



The first time an AWS Lambda function is invoked is referred to as a cold start. There are several factors that can increase the latency of a cold start:






- **Be sure to assign enough memory to your Lambda function.** – Compilation during a cold start can be significantly slower for a Lambda function than it would be on EC2 because AWS Lambda allocates CPU cycles [linearly in proportion to the memory](#) that you assign to the function. With 1,792 MB of memory, a function receives the equivalent of one full vCPU (one vCPU-second of credits per second). The impact of not assigning enough memory to receive adequate CPU cycles is particularly pronounced for large Lambda functions written in Java.
- **Be aware that [enabling IAM database authentication \(p. 117\)](#) may slow down a cold start** – AWS Identity and Access Management (IAM) database authentication can also slow down cold starts, particularly if the Lambda function has to generate a new signing key. This latency only affects the cold start and not subsequent requests, because once IAM DB auth has established the connection credentials, Neptune only periodically validates that they are still valid.

Using AWS CloudFormation to Create a Lambda Function to Use in Neptune

You can use an AWS CloudFormation template to create an AWS Lambda function that can access Neptune.

1. To launch the Lambda function stack on the AWS CloudFormation console, choose one of the **Launch Stack** buttons in the following table.

Region	View	View in Designer	Launch
US East (N. Virginia)	View	View in Designer	
US East (Ohio)	View	View in Designer	
US West (N. California)	View	View in Designer	
US West (Oregon)	View	View in Designer	
Canada (Central)	View	View in Designer	
South America (São Paulo)	View	View in Designer	
Europe (Stockholm)	View	View in Designer	
Europe (Ireland)	View	View in Designer	
Europe (London)	View	View in Designer	
Europe (Paris)	View	View in Designer	
Europe (Frankfurt)	View	View in Designer	
Middle East (Bahrain)	View	View in Designer	
Asia Pacific (Hong Kong)	View	View in Designer	
Asia Pacific (Tokyo)	View	View in Designer	
Asia Pacific (Seoul)	View	View in Designer	
Asia Pacific (Singapore)	View	View in Designer	
Asia Pacific (Sydney)	View	View in Designer	

Region	View	View in Designer	Launch
Asia Pacific (Mumbai)	View	View in Designer	
China (Beijing)	View	View in Designer	
China (Ningxia)	View	View in Designer	
AWS GovCloud (US-West)	View	View in Designer	
AWS GovCloud (US-East)	View	View in Designer	

2. On the **Select Template** page, choose **Next**.
3. On the **Specify Details** page, set the following options:
 - a. Choose the Lambda runtime, depending on what language you want to use in your Lambda function. These AWS CloudFormation templates currently support the following languages:
 - **Python 3.6** (maps to `python36` in the Amazon S3 URL)
 - **NodeJS 12** (maps to `nodejs12x` in the Amazon S3 URL)
 - **Ruby 2.5** (maps to `ruby25` in the Amazon S3 URL)
 - b. Provide the appropriate Neptune cluster endpoint and port number.
 - c. Provide the appropriate Neptune security group.
 - d. Provide the appropriate Neptune subnet parameters.
4. Choose **Next**.
5. On the **Options** page, choose **Next**.
6. On the **Review** page, select the first check box to acknowledge that AWS CloudFormation will create IAM resources.

Then choose **Create**.

If you need to make your own changes to the Lambda runtime, you can download a generic one from an Amazon S3 location in your Region:

```
https://s3.Amazon region.amazonaws.com/aws-neptune-customer-samples-Amazon region/lambda/runtime-language/lambda_function.zip.
```

For example:

```
https://s3.us-west-2.amazonaws.com/aws-neptune-customer-samples-us-west-2/lambda/python36/lambda_function.zip
```

AWS Lambda function examples for Amazon Neptune

The following example AWS Lambda functions, written in Java, JavaScript and Python, illustrate upserting a single vertex with a randomly generated ID using the `fold().coalesce().unfold()` idiom.

Much of the code in each function is boilerplate code, responsible for managing connections and retrying connections and queries if an error occurs. The real application logic and the Gremlin query are implemented in `doQuery()` and `query()` methods respectively. If you use these examples as a basis for your own Lambda functions, you can concentrate on modifying `doQuery()` and `query()`.

The functions are configured to retry failed queries 5 times, waiting 1 second between retries.

The functions require values to be present in the following Lambda environment variables:

- **NEPTUNE_ENDPOINT** – Your Neptune DB cluster endpoint.
- **NEPTUNE_PORT** – The Neptune port.
- **USE_IAM** – (`true` or `false`) If your database has AWS Identity and Access Management (IAM) database authentication enabled, set the `USE_IAM` environment variable to `true`. This causes the Lambda function to Sigv4-sign connection requests to Neptune. For such IAM DB auth requests, ensure that the Lambda function's execution role has an appropriate IAM policy attached that allows the function to connect to your Neptune DB cluster (see [Custom IAM Policies \(p. 119\)](#)).

Java Lambda function example for Amazon Neptune

Here are some things to keep in mind about Java AWS Lambda functions:

- The Java driver maintains its own connection pool, which you do not need, so configure your `Cluster` object with `minConnectionPoolSize(1)` and `maxConnectionPoolSize(1)`.
- The `Cluster` object can be slow to build because it creates one or more serializers (Gyro by default, plus another if you've configured it for additional output formats such as binary). These can take a while to instantiate.
- The connection pool is initialized with the first request. At this point, the driver sets up the Netty stack, allocates byte buffers, and creates a signing key if you are using IAM DB auth. All of which can add to the cold-start latency.
- The Java driver's connection pool monitors the availability of server hosts and automatically attempts to reconnect if a connection fails. It starts a background task to try to re-establish the connection. Use `reconnectInterval()` to configure the interval between reconnection attempts. While the driver is attempting to reconnect, your Lambda function can simply retry the query.

If the interval between retries is smaller than the interval between reconnect attempts, retries on a failed connection fail again because the host is considered unavailable. This does not apply to retries for a `ConcurrentModificationException`.

- Use Java 8 rather than Java 11. Netty optimizations are not enabled by default in Java 11.
- This example uses [Retry4j](#) for retries.
- To use the Sigv4 signing driver in your Java Lambda function, see the dependency requirements in [Connecting to Neptune Using Java and Gremlin with Signature Version 4 Signing \(p. 151\)](#).

```
package com.amazonaws.examples.social;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.evannlennick.retry4j.CallExecutor;
import com.evannlennick.retry4j.CallExecutorBuilder;
import com.evannlennick.retry4j.Status;
import com.evannlennick.retry4j.config.RetryConfig;
import com.evannlennick.retry4j.config.RetryConfigBuilder;
import org.apache.tinkerpop.gremlin.driver.Cluster;
import org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer;
import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
```

```
import org.apache.tinkerpop.gremlin.driver.ser.Serializers;
import org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource;
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;
import org.apache.tinkerpop.gremlin.structure.T;

import java.io.*;
import java.time.temporal.ChronoUnit;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.function.Function;

import static java.nio.charset.StandardCharsets.UTF_8;
import static org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.addV;
import static org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.unfold;

public class MyHandler implements RequestStreamHandler {

    private final GraphTraversalSource g;
    private final CallExecutor<Object> executor;
    private final Random idGenerator = new Random();

    public MyHandler() {

        this.g = AnonymousTraversalSource
            .traversal()
            .withRemote(DriverRemoteConnection.using(createCluster()));

        this.executor = new CallExecutorBuilder<Object>()
            .config(createRetryConfig())
            .build();

    }

    @Override
    public void handleRequest(InputStream input,
                              OutputStream output,
                              Context context) throws IOException {

        doQuery(input, output);

    }

    private void doQuery(InputStream input, OutputStream output) throws IOException {
        try {

            Map<String, Object> args = new HashMap<>();
            args.put("id", idGenerator.nextInt());

            String result = query(args);

            try (Writer writer = new BufferedWriter(new OutputStreamWriter(output, UTF_8)))
            {
                writer.write(result);
            }

        } finally {
            input.close();
            output.close();
        }

    }

    private String query(Map<String, Object> args) {

        int id = (int) args.get("id");
```

```
@SuppressWarnings("unchecked")
Callable<Object> query = () -> g.V(id)
    .fold()
    .coalesce(
        unfold(),
        addV("Person").property(T.id, id))
    .id().next();

Status<Object> status = executor.execute(query);

return status.getResult().toString();
}

private Cluster createCluster() {
    Cluster.Builder builder = Cluster.build()
        .addContactPoint(System.getenv("NEPTUNE_ENDPOINT"))
        .port(Integer.parseInt(System.getenv("NEPTUNE_PORT")))
        .enableSsl(true)
        .minConnectionPoolSize(1)
        .maxConnectionPoolSize(1)
        .serializer(Serializers.GRAPHBINARY_V1D0)
        .reconnectInterval(2000);

    if (Boolean.parseBoolean(getOptionalEnv("USE_IAM", "true"))) {
        builder = builder.channelizer(SigV4WebSocketChannelizer.class);
    }

    return builder.create();
}

private RetryConfig createRetryConfig() {
    return new RetryConfigBuilder()
        .retryOnCustomExceptionLogic(retryLogic())
        .withDelayBetweenTries(1000, ChronoUnit.MILLIS)
        .withMaxNumberOfTries(5)
        .withFixedBackoff()
        .build();
}

private Function<Exception, Boolean> retryLogic() {

    return e -> {

        StringWriter stringWriter = new StringWriter();
        e.printStackTrace(new PrintWriter(stringWriter));
        String message = stringWriter.toString();

        // Check for connection issues
        if (message.contains("Timed out while waiting for an available host") ||
            message.contains("Timed-out waiting for connection on Host") ||
            message.contains("Connection to server is no longer active") ||
            message.contains("Connection reset by peer") ||
            message.contains("SSL Engine closed already") ||
            message.contains("Pool is shutdown") ||
            message.contains("ExtendedClosedChannelException") ||
            message.contains("Broken pipe")) {
            return true;
        }

        // Concurrent writes can sometimes trigger a ConcurrentModificationException.
        // In these circumstances you may want to backoff and retry.
        if (message.contains("ConcurrentModificationException")) {
            return true;
        }
    }
}
```

```
        // If the primary fails over to a new instance, existing connections to the old
primary will
        // throw a ReadOnlyViolationException. You may want to back and retry.
        if (message.contains("ReadOnlyViolationException")) {
            return true;
        }

        return false;
    };
}

private String getOptionalEnv(String name, String defaultValue) {
    String value = System.getenv(name);
    if (value != null && value.length() > 0) {
        return value;
    } else {
        return defaultValue;
    }
}
}
```

If you want to include reconnect logic in your function, see [Java reconnect sample \(p. 272\)](#).

JavaScript Lambda function example for Amazon Neptune

Notes about this example

- The JavaScript driver doesn't maintain a connection pool. It always opens a single connection.
- The example function uses the Sigv4 signing utilities from [gremlin-aws-sigv4](#) for signing requests to an IAM authentication-enabled database.
- It uses the [retry\(\)](#) function from the open-source [async utility module](#) to handle backoff-and-retry attempts.
- Gremlin terminal steps return a JavaScript promise (see the [TinkerPop documentation](#)). For `next()`, this is a `{value, done}` tuple.
- Connection errors are raised inside the handler, and dealt with using some backoff-and-retry logic in line with the recommendations outlined here, with one exception. There is one kind of connection issue that the driver does not treat as an exception, and which cannot therefore be accommodated by this backoff-and-retry logic.

The problem is that if a connection is closed after a driver sends a request but before the driver receives a response, the query appears to complete but returns a null value. As far as the lambda function client is concerned, the function appears to complete successfully, but with an empty response.

The impact of this issue depends on how your application treats an empty response. Some applications may treat an empty response from a read request as an error, but others may mistakenly treat it as an empty result.

Write requests that encounter this connection issue will also return an empty response. Does a successful invocation with an empty response signal success or failure? If the client invoking a write function simply treats the successful invocation of the function to mean the write to the database has been committed, rather than inspecting the body of the response, the system may appear to lose data.

This issue results from how the driver treats events emitted by the underlying socket. When the underlying network socket is closed with an `ECONNRESET` error, the `WebSocket` used by the driver is

closed and emits a 'ws close' event. There's nothing in the driver, however, to handle that event in a way that could be used to raise an exception. As a result, the query simply disappears.

To work around this issue, the example lambda function here adds a 'ws close' event handler that throws an exception to the driver when creating a remote connection. This exception is not, however, raised along the Gremlin query's request-response path, and can't therefore be used to trigger any backoff-and-retry logic within the lambda function itself. Instead, the exception thrown by the 'ws close' event handler results in an unhandled exception that causes the lambda invocation to fail. This allows the client that invokes the function to handle the error and retry the lambda invocation if appropriate.

We recommend that you implement backoff-and-retry logic in the lambda function itself to protect your clients from intermittent connection issues. However, the workaround for the above issue requires the client to implement retry logic too, to handle failures that result from this particular connection issue.

JavaScript code

```
const gremlin = require('gremlin');
const async = require('async');
const {getUrlAndHeaders} = require('gremlin-aws-sigv4/lib/utils');

const traversal = gremlin.process.AnonymousTraversalSource.traversal;
const DriverRemoteConnection = gremlin.driver.DriverRemoteConnection;
const t = gremlin.process.t;
const __ = gremlin.process.statics;

let conn = null;
let g = null;

async function query(context) {

  const id = context.id;

  return g.V(id)
    .fold()
    .coalesce(
      __.unfold(),
      __.addV('User').property(t.id, id)
    )
    .id().next();
}

async function doQuery() {
  const id = Math.floor(Math.random() * 10000).toString();

  let result = await query({id: id});
  return result['value'];
}

exports.handler = async (event, context) => {

  const getConnectionDetails = () => {
    if (process.env['USE_IAM'] == 'true'){
      return getUrlAndHeaders(
        process.env['NEPTUNE_ENDPOINT'],
        process.env['NEPTUNE_PORT'],
        {},
        '/gremlin',
        'wss');
    } else {
```

```
    const database_url = 'wss://' + process.env['NEPTUNE_ENDPOINT'] + ':' +
process.env['NEPTUNE_PORT'] + '/gremlin';
    return { url: database_url, headers: {}};
  }
};

const createRemoteConnection = () => {
  const { url, headers } = getConnectionDetails();

  const c = new DriverRemoteConnection(
    url,
    {
      mimeType: 'application/vnd.gremlin-v2.0+json',
      headers: headers
    });

  c._client._connection.on('close', (code, message) => {
    console.info(`close - ${code} ${message}`);
    if (code == 1006){
      console.error('Connection closed prematurely');
      throw new Error('Connection closed prematurely');
    }
  });

  return c;
};

const createGraphTraversalSource = (conn) => {
  return traversal().withRemote(conn);
};

if (conn == null){
  console.info("Initializing connection")
  conn = createRemoteConnection();
  g = createGraphTraversalSource(conn);
}

return async.retry(
  {
    times: 5,
    interval: 1000,
    errorFilter: function (err) {

      // Add filters here to determine whether error can be retried
      console.warn('Determining whether retrieable error: ' + err.message);

      // Check for connection issues
      if (err.message.startsWith('WebSocket is not open')){
        console.warn('Reopening connection');
        conn.close();
        conn = createRemoteConnection();
        g = createGraphTraversalSource(conn);
        return true;
      }

      // Check for ConcurrentModificationException
      if (err.message.includes('ConcurrentModificationException')){
        console.warn('Retrying query because of ConcurrentModificationException');
        return true;
      }

      // Check for ReadOnlyViolationException
      if (err.message.includes('ReadOnlyViolationException')){
        console.warn('Retrying query because of ReadOnlyViolationException');
        return true;
      }
    }
  }
);
```

```
    }  
    return false;  
  }  
},  
doQuery);  
};
```

Python Lambda function example for Amazon Neptune

Here are some things to notice about the following Python AWS Lambda example function:

- It uses the [backoff module](#).
- It sets `pool_size=1` to keep from creating an unnecessary connection pool.
- It sets `message_serializer=serializer.GraphSONSerializersV2d0()`.

```
import os, sys, backoff, math  
from random import randint  
from gremlin_python import statics  
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection  
from gremlin_python.driver.protocol import GremlinServerError  
from gremlin_python.driver import serializer  
from gremlin_python.process.anonymous_traversal import traversal  
from gremlin_python.process.graph_traversal import __  
from gremlin_python.process.strategies import *  
from gremlin_python.process.traversal import T  
from tornado.websocket import WebSocketClosedError  
from tornado import httpclient  
from botocore.auth import SigV4Auth  
from botocore.awsrequest import AWSRequest  
from botocore.credentials import ReadOnlyCredentials  
from types import SimpleNamespace  
  
reconnectable_err_msgs = [  
    'ReadOnlyViolationException',  
    'Server disconnected',  
    'Connection refused'  
]  
  
retriable_err_msgs = ['ConcurrentModificationException'] + reconnectable_err_msgs  
  
network_errors = [WebSocketClosedError, OSError]  
  
retriable_errors = [GremlinServerError] + network_errors  
  
def prepare_iamdb_request(database_url):  
  
    service = 'neptune-db'  
    method = 'GET'  
  
    access_key = os.environ['AWS_ACCESS_KEY_ID']  
    secret_key = os.environ['AWS_SECRET_ACCESS_KEY']  
    region = os.environ['AWS_REGION']  
    session_token = os.environ['AWS_SESSION_TOKEN']  
  
    creds = SimpleNamespace(  
        access_key=access_key, secret_key=secret_key, token=session_token, region=region,  
    )
```

```
request = AWSRequest(method=method, url=database_url, data=None)
SigV4Auth(creds, service, region).add_auth(request)

return httpclient.HTTPRequest(database_url, headers=request.headers.items())

def is_retriable_error(e):

    is_retriable = False
    err_msg = str(e)

    if isinstance(e, tuple(network_errors)):
        is_retriable = True
    else:
        is_retriable = any(retriable_err_msg in err_msg for retriable_err_msg in
retriable_err_msgs)

    print('error: [{}] {}'.format(type(e), err_msg))
    print('is_retriable: {}'.format(is_retriable))

    return is_retriable

def is_non_retriable_error(e):
    return not is_retriable_error(e)

def reset_connection_if_connection_issue(params):

    is_reconnectable = False

    e = sys.exc_info()[1]
    err_msg = str(e)

    if isinstance(e, tuple(network_errors)):
        is_reconnectable = True
    else:
        is_reconnectable = any(reconnectable_err_msg in err_msg for reconnectable_err_msg in
reconnectable_err_msgs)

    print('is_reconnectable: {}'.format(is_reconnectable))

    if is_reconnectable:
        global conn
        global g
        conn.close()
        conn = create_remote_connection()
        g = create_graph_traversal_source(conn)

@backoff.on_exception(backoff.constant,
    tuple(retriable_errors),
    max_tries=5,
    jitter=None,
    giveup=is_non_retriable_error,
    on_backoff=reset_connection_if_connection_issue,
    interval=1)
def query(**kwargs):

    id = kwargs['id']

    return (g.V(id)
        .fold()
        .coalesce(
            __.unfold(),
            __.addV('User').property(T.id, id)
        )
        .id().next())
```



```
def doQuery(event):
    return query(id=str(randint(0, 10000)))

def lambda_handler(event, context):
    return doQuery(event)

def create_graph_traversal_source(conn):
    return traversal().withRemote(conn)

def create_remote_connection():
    print('Creating remote connection')

    return DriverRemoteConnection(
        connection_string(),
        'g',
        pool_size=1,
        message_serializer=serializer.GraphSONSerializersV2d0())

def connection_string():

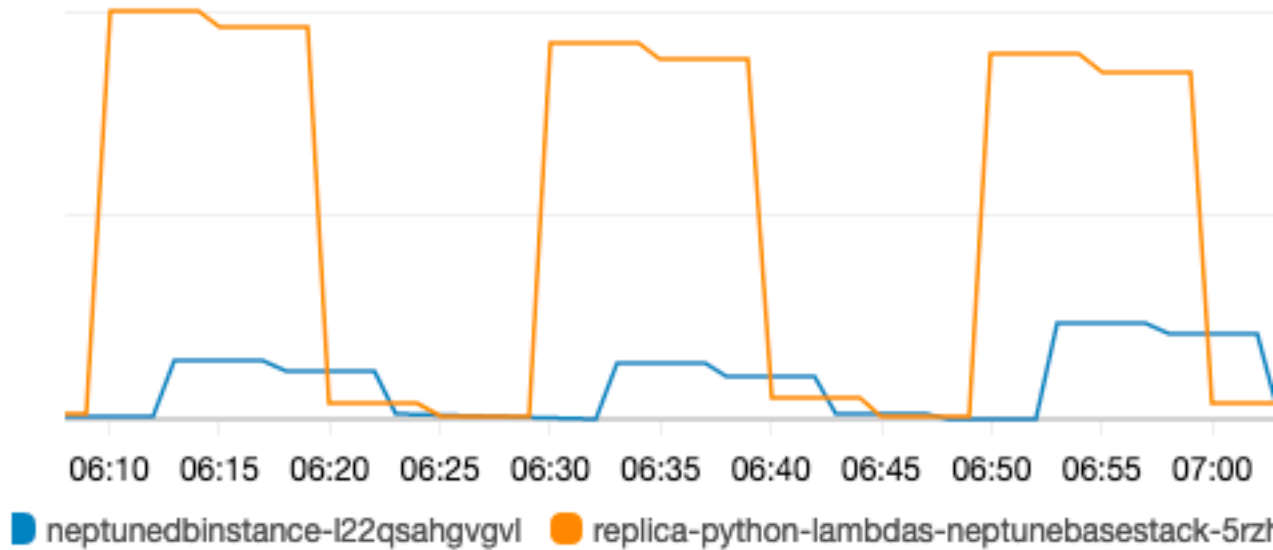
    database_url = 'wss://{}:{}/gremlin'.format(os.environ['neptuneEndpoint'],
os.environ['neptunePort'])

    if 'USE_IAM' in os.environ and os.environ['USE_IAM'] == 'true':
        return prepare_iamdb_request(database_url)
    else:
        return database_url

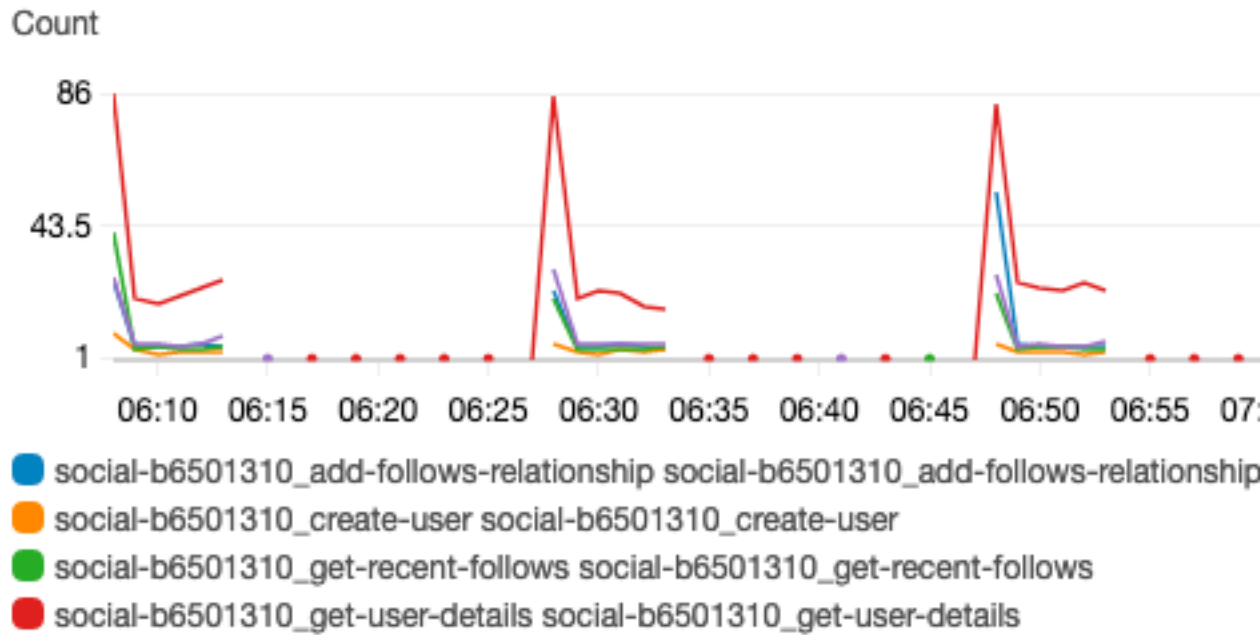
conn = create_remote_connection()
g = create_graph_traversal_source(conn)
```

Here are sample results:

GremlinWebSocketOpenConnections



Lambda-ConcurrentExecutions



Workload

Various units

API Version 2020-12-18
521

2.17k



Amazon Neptune ML for machine learning on graphs

There is often valuable information in large connected datasets that can be hard to extract using queries based on human intuition alone. Machine learning (ML) techniques can help find hidden correlations in graphs with billions of relationships. These correlations can be helpful for recommending products, predicting credit worthiness, identifying fraud, and many other things.

The Neptune ML feature makes it possible to build and train useful machine learning models on large graphs in hours instead of weeks. To accomplish this, Neptune ML uses graph neural network (GNN) technology powered by [Amazon SageMaker](#) and the [Deep Graph Library \(DGL\)](#) (which is [open-source](#)). Graph neural networks are an emerging field in artificial intelligence (see, for example, [A Comprehensive Survey on Graph Neural Networks](#)). For a hands-on tutorial about using GNNs with DGL, see [Learning graph neural networks with Deep Graph Library](#).

Note

Graph vertices are identified in Neptune ML models as "nodes". For example, vertex classification uses a node-classification machine learning model, and vertex regression uses a node-regression model.

What Neptune ML can do

Neptune ML can train machine learning models to support five different categories of inference:

Types of inference task currently supported by Neptune ML

- **Node classification** – predicting the categorical feature of a vertex property.

For example, given the movie *The Shawshank Redemption*, Neptune ML can predict its genre property as `story` from a candidate set of [`story`, `crime`, `action`, `fantasy`, `drama`, `family`, ...].

There are two types of node-classification tasks:

- **Single-class classification:** In this kind of task, each node has only one target feature. For example, the property, `Place_of_birth` of `Alan Turing` has the value `UK`.
 - **Multi-class classification:** In this kind of task, each node can have more than one target feature. For example, the property `genre` of the film *The Godfather* has the values `crime` and `story`.
- **Node regression** – predicting a numerical property of a vertex.

For example, given the movie *Avengers: Endgame*, Neptune ML can predict that its property `popularity` has a value of `5.0`.

- **Edge classification** – predicting the categorical feature of an edge property.

There are two types of edge-classification tasks:

- **Single-class classification:** In this kind of task, each edge has only one target feature. For example, a ratings edge between a user and a movie might have the property, `liked`, with a value of either "Yes" or "No".
 - **Multi-class classification:** In this kind of task, each edge can have more than one target feature. For example, a ratings between a user and movie might have multiple values for the property tag such as "Funny", "Heartwarming", "Chilling", and so on.
- **Edge regression** – predicting a numerical property of an edge.

For example, a rating edge between a user and a movie might have the numerical property, `score`, for which Neptune ML could predict a value given a user and a movie.

- **Link prediction** – predicting the most likely destination nodes for a particular source node and outgoing edge, or the most likely source nodes for a given destination node and incoming edge.

For example, with a drug-disease knowledge graph, given `Aspirin` as the source node, and `treats` as the outgoing edge, Neptune ML can predict the most relevant destination nodes as `heart disease`, `fever`, and so on.

Or, with the Wikimedia knowledge graph, given `President-of` as the edge or relation and `United-States` as the destination node, Neptune ML can predict the most relevant heads as `George Washington`, `Abraham Lincoln`, `Franklin D. Roosevelt`, and so on.

With Neptune ML, you can use machine learning models that fall in two general categories:

Types of machine learning model currently supported by Neptune ML

- **Graph Neural Network (GNN) models** – These include [Relational Graph Convolutional Networks \(R-GCNs\)](#). GNN models work for all three types of task above.
- **Knowledge-Graph Embedding (KGE) models** – These include `TransE`, `DistMult`, and `RotatE` models. They only work for link prediction.





User defined models – Neptune ML also lets you provide your own custom model implementation for all the types of tasks listed above. You can use the [Neptune ML toolkit](#) to develop and test your python-based custom model implementation before using the Neptune ML training API with your model. See [Custom models in Neptune ML \(p. 595\)](#) for details about how to structure and organize your implementation so that it's compatible with Neptune ML's training infrastructure.






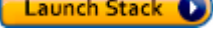
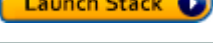
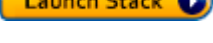
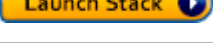

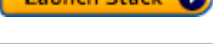

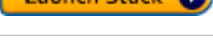



Using the Neptune ML AWS CloudFormation template to get started quickly in a new DB cluster

The easiest way to get started with Neptune ML is to use the AWS CloudFormation quick-start template. This template installs all necessary components, including a Neptune DB cluster, and sets up the necessary IAM roles.

To create the Neptune ML quick-start stack

1. To launch the AWS CloudFormation stack on the AWS CloudFormation console, choose one of the **Launch Stack** buttons in the following table:

Region	View	View in Designer	Launch
US East (N. Virginia)	View	View in Designer	
US East (Ohio)	View	View in Designer	
US West (N. California)	View	View in Designer	
US West (Oregon)	View	View in Designer	

Region	View	View in Designer	Launch
Canada (Central)	View	View in Designer	
South America (São Paulo)	View	View in Designer	
Europe (Stockholm)	View	View in Designer	
Europe (Ireland)	View	View in Designer	
Europe (London)	View	View in Designer	
Europe (Paris)	View	View in Designer	
Europe (Frankfurt)	View	View in Designer	
Middle East (Bahrain)	View	View in Designer	
Asia Pacific (Hong Kong)	View	View in Designer	
Asia Pacific (Tokyo)	View	View in Designer	
Asia Pacific (Seoul)	View	View in Designer	
Asia Pacific (Singapore)	View	View in Designer	
Asia Pacific (Sydney)	View	View in Designer	
Asia Pacific (Mumbai)	View	View in Designer	
China (Beijing)	View	View in Designer	
China (Ningxia)	View	View in Designer	
AWS GovCloud (US-West)	View	View in Designer	

2. On the **Select Template** page, choose **Next**.
3. On the **Specify Details** page, choose **Next**.
4. On the **Options** page, choose **Next**.
5. On the **Review** page, there are two check boxes that you need to check:
 - The first one acknowledges that AWS CloudFormation might create IAM resources with custom names.
 - The second acknowledges that AWS CloudFormation might require the `CAPABILITY_AUTO_EXPAND` capability for the new stack. `CAPABILITY_AUTO_EXPAND` explicitly

allows AWS CloudFormation to expand macros automatically when creating the stack, without prior review.

Customers often create a change set from a processed template so that the changes made by macros can be reviewed before actually creating the stack. For more information, see the AWS CloudFormation [CreateStack](#) API.

Then choose **Create**.

The quick-start template creates and sets up the following:

- A Neptune DB cluster.
- The necessary IAM roles (and attaches them).
- The necessary Amazon EC2 security group.
- The necessary SageMaker VPC endpoints.
- A DB cluster parameter group for Neptune ML.
- The necessary parameters in that parameter group.
- A SageMaker notebook with pre-populated notebook samples for Neptune ML. Note that not all instance sizes are available in every region, so you need to be sure that the notebook instance size selected is one that your region supports.
- The Neptune-Export service.

When the quick-start stack is ready, go to the SageMaker notebook that the template created and check out the pre-populated examples. They will help you download sample datasets to use for experimenting with Neptune ML capabilities.

They can also save you a lot of time when you are using Neptune ML. For example, see the [%neptune_ml](#) (p. 39) line magic, and the [%%neptune_ml](#) (p. 42) cell magic that these notebooks support.

You can also use the following AWS CLI command to run the quick-start AWS CloudFormation template:

```
aws cloudformation create-stack \  
  --stack-name neptune-ml-fullstack-$(date '+%Y-%m-%d-%H-%M') \  
  --template-url https://aws-neptune-customer-samples.s3.amazonaws.com/v2/cloudformation-  
templates/neptune-ml-nested-stack.json \  
  --parameters ParameterKey=EnableIAMAuthOnExportAPI,ParameterValue=(true if you have IAM  
auth enabled, or false otherwise) \  
                 ParameterKey=Env,ParameterValue=test$(date '+%H%M')\  
  --capabilities CAPABILITY_IAM \  
  --region (the AWS region, like us-east-1) \  
  --disable-rollback \  
  --profile (optionally, a named CLI profile of yours)
```

Setting up Neptune ML without using the quick-start AWS CloudFormation template

If you don't use the AWS CloudFormation quick-start template, there are a number of things you need to do to get Neptune ML working:

- **Create a Neptune DB cluster** – If you don't yet have a working Neptune DB cluster, see [Creating a DB Cluster \(p. 166\)](#) to create one. You can then load data to experiment with from a number of different public datasets. The easiest way to do that is by using the SageMaker notebooks in Neptune for getting started with Neptune ML.
- **Make sure that your Neptune DB cluster isn't running an older engine version** – If you're using an existing Neptune DB cluster, check to be sure that it's running at least engine version 1.0.5.0. If it's running an earlier engine version, you can upgrade it as described in [Neptune Engine Updates \(p. 99\)](#).
- **Set up access between Neptune, SageMaker, and Amazon S3 using IAM** – For instructions, see [Manual setup \(p. 527\)](#).
- **Create two SageMaker endpoints in your Neptune VPC** – This lets the Neptune engine access the necessary SageMaker management APIs. See the instructions in [Create two endpoints for SageMaker in your Neptune VPC \(p. 533\)](#).

Setting up IAM roles and policies for Neptune ML

You need to create the following IAM policies and attach them to the appropriate IAM roles so that Neptune ML can work with SageMaker and Amazon S3:

- **Create a `NeptuneSageMakerPolicy`** – This provides Neptune ML with the minimum access it requires to SageMaker and Amazon S3.

For an example of this policy, see [IAM policy that grants Neptune ML access to SageMaker and Amazon S3 \(p. 527\)](#).

- **Create a `SageMakerExecutionPolicy`** – This provides SageMaker with the minimum access it requires to Neptune ML and Amazon S3.

For an example of this policy, see [SageMaker execution \(p. 529\)](#).

- **Create the `NeptuneSageMakerIAMRole`** – This is the role that Neptune ML uses for access to the resources it needs.

This role needs the [NeptuneSageMakerPolicy \(p. 527\)](#) attached, as well as either an [Neptune trust policy \(p. 530\)](#) or a [Combined trust policy \(p. 531\)](#).

- **Create the `SageMakerExecutionIAMRole`** – This is the role that SageMaker uses for access to the resources it needs to work with Neptune ML.

This role needs the [SageMakerExecutionPolicy \(p. 529\)](#) attached, as well as either a [Neptune trust policy \(p. 531\)](#) or a [Combined trust policy \(p. 531\)](#).

IAM policy that grants Neptune ML access to SageMaker and Amazon S3

The following policy grants Neptune ML the access that it needs to access SageMaker and Amazon S3 resources:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/*"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "sagemaker.amazonaws.com"
          ]
        }
      }
    },
    {
      "Action": [
        "kms:CreateGrant",
        "kms:Decrypt",
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms::*:key/*",
      "Effect": "Allow"
    }
  ]
}
```



```
    },
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/sagemaker/*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:CreateEndpoint",
        "sagemaker:CreateEndpointConfig",
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateModel",
        "sagemaker:CreateProcessingJob",
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateTransformJob",
        "sagemaker>DeleteEndpoint",
        "sagemaker>DeleteEndpointConfig",
        "sagemaker:StopHyperParameterTuningJob",
        "sagemaker>DeleteModel",
        "sagemaker:StopProcessingJob",
        "sagemaker:StopTrainingJob",
        "sagemaker:StopTransformJob",
        "sagemaker:DescribeEndpoint",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:DescribeHyperParameterTuningJob",
        "sagemaker:DescribeModel",
        "sagemaker:DescribeProcessingJob",
        "sagemaker:DescribeTrainingJob",
        "sagemaker:DescribeTransformJob",
        "sagemaker:InvokeEndpoint",
        "sagemaker:ListTags",
        "sagemaker:ListTrainingJobsForHyperParameterTuningJob",
        "sagemaker:UpdateEndpoint"
      ],
      "Resource": [
        "arn:aws:sagemaker:*:*:*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:ListEndpointConfigs",
        "sagemaker:ListEndpoints",
        "sagemaker:ListHyperParameterTuningJobs",
        "sagemaker:ListModels",
        "sagemaker:ListProcessingJobs",
        "sagemaker:ListTrainingJobs",
        "sagemaker:ListTransformJobs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
```

```
        "s3:AbortMultipartUpload",
        "s3:ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::*"
    ],
    "Effect": "Allow"
}
]
```

IAM execution policy for Amazon SageMaker

The following policy grants SageMaker the access it needs to Neptune ML and Amazon S3:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:CreateVpcEndpoint",
        "ec2>DeleteNetworkInterface",
        "ec2>DeleteNetworkInterfacePermission",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ],
      "Resource": [
        "*",
        "arn:aws:ecr:*:*:repository/*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam:*:*:role/*"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "sagemaker.amazonaws.com"
          ]
        }
      }
    },
    {
      "Effect": "Allow"
    }
  ]
}
```

```

    },
    {
      "Action": [
        "kms:CreateGrant",
        "kms:Decrypt",
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms:*:*:key/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/sagemaker/*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:DescribeHyperParameterTuningJob",
        "sagemaker:ListTags",
        "sagemaker:CreateTransformJob",
        "sagemaker:DescribeTransformJob"
      ],
      "Resource": [
        "arn:aws:sagemaker:*:*:*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:AbortMultipartUpload",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::*:*"
      ],
      "Effect": "Allow"
    }
  ]
}

```

Amazon Neptune trust policy

The following policy creates a trust relationship just for Neptune ML:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {

```

```
    "Service": [
      "rds.amazonaws.com"
    ]
  },
  "Action": "sts:AssumeRole"
}
]
```

Trust policy for Amazon SageMaker

The following policy creates a trust relationship just for SageMaker:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Combined trust policy for Amazon Neptune and Amazon SageMaker

The following policy creates a combined trust relationship that you can use for both Neptune ML and SageMaker:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "rds.amazonaws.com",
          "sagemaker.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Setting up Neptune ML on your Neptune DB cluster

If you're not using the AWS CloudFormation quick-start template for setting up Neptune ML, there are a number of steps to take to enable Neptune ML on your DB cluster, set DB cluster parameters, and attach the necessary IAM roles.

Create a DB cluster parameter group for your new Neptune ML cluster

The following are AWS CLI commands you can use to create a new DB cluster parameter group and set it up to work with Neptune ML:

To create and configure a DB cluster parameter group for Neptune ML

1. Create a new DB cluster parameter group:

```
aws neptune create-db-cluster-parameter-group \  
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \  
  --db-parameter-group-family neptune1 \  
  --description "(description of your machine learning project)" \  
  --region (AWS region, such as us-east-1)
```

2. Create a `neptune_ml_iam_role` DB cluster parameter set to the ARN of the `SageMakerExecutionIAMRole` for your DB cluster to use while calling SageMaker for creating jobs and getting prediction from hosted ML models:

```
aws neptune modify-db-cluster-parameter-group \  
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \  
  --parameters "ParameterName=neptune_ml_iam_role, \  
    ParameterValue=ARN of the SageMakerExecutionIAMRole, \  
    Description=NeptuneMLRole, \  
    ApplyMethod=pending-reboot" \  
  --region (AWS region, such as us-east-1)
```

Setting this parameter allows Neptune to access SageMaker without you having to pass in the role with every call.

For information about how to create the `SageMakerExecutionIAMRole`, see [Setting up IAM roles and policies for Neptune ML \(p. 527\)](#).

3. Finally, use `describe-db-cluster-parameters` to check that all the parameters in the new DB cluster parameter group are set as you want them to be:

```
aws neptune describe-db-cluster-parameters \  
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \  
  --region (AWS region, such as us-east-1)
```

Attach the new DB cluster parameter group to the DB cluster you will use with Neptune ML

Now you can attach the new DB cluster parameter group that you just created to an existing DB cluster by using the following command:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (the name of your existing DB cluster) \  
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group)
```

```
--apply-immediately  
--db-cluster-parameter-group-name (name of your new DB cluster parameter group) \  
--region (AWS region, such as us-east-1)
```

To make all the parameters effective, you can then reboot the DB cluster:

```
aws neptune reboot-db-instance  
--db-instance-identifier (name of the primary instance of your DB cluster) \  
--profile (name of your AWS profile to use) \  
--region (AWS region, such as us-east-1)
```

Or, if you're creating a new DB cluster to use with Neptune ML, you can use the following command to create the cluster with the new parameter group attached, and then create a new primary (writer) instance:

```
cluster-name=(the name of the new DB cluster)  
aws neptune create-db-cluster  
--db-cluster-identifier ${cluster-name}  
--engine graphdb \  
--engine-version 1.0.4.1 \  
--db-cluster-parameter-group-name (name of your new DB cluster parameter group) \  
--db-subnet-group-name (name of the subnet to use) \  
--region (AWS region, such as us-east-1)  
  
aws neptune create-db-instance  
--db-cluster-identifier ${cluster-name}  
--db-instance-identifier ${cluster-name}-i \  
--db-instance-class (the instance class to use, such as db.r5.xlarge) \  
--engine graphdb \  
--region (AWS region, such as us-east-1)
```

Attach the NeptuneSageMakerIAMRole to your DB cluster so that it can access SageMaker and Amazon S3 resources

Finally, attach the NeptuneSageMakerIAMRole that will allow your DB cluster to communicate with SageMaker and Amazon S3 (for information about how to create that role, see [Manual setup \(p. 527\)](#)):

```
aws neptune add-role-to-db-cluster  
--db-cluster-identifier ${cluster-name}  
--role-arn arn:aws:iam::(the ARN number of the role's ARN):role/NeptuneMLRole \  
--region (AWS region, such as us-east-1)
```

Create two endpoints for SageMaker in your Neptune VPC

Neptune ML needs two SageMaker endpoints in your Neptune DB cluster's VPC:

- `com.amazonaws.(AWS region, like us-east-1).sagemaker.runtime`
- `com.amazonaws.(AWS region, like us-east-1).sagemaker.api`

If you haven't used the quick-start AWS CloudFormation template, which creates these automatically for you, you can use the following AWS CLI commands to create them:

This one creates the `sagemaker.runtime` endpoint:

```
create-vpc-endpoint  
--vpc-id (the ID of your Neptune DB cluster's VPC)
```

```
--service-name com.amazonaws.(AWS region, like us-east-1).sagemaker.runtime
--subnet-ids (the subnet ID or IDs that you want to use)
--security-group-ids (the security group for the endpoint network interface, or omit to
use the default)
--private-dns-enabled
```

And this one creates the `sagemaker.api` endpoint:

```
aws create-vpc-endpoint
--vpc-id (the ID of your Neptune DB cluster's VPC)
--service-name com.amazonaws.(AWS region, like us-east-1).sagemaker.api
--subnet-ids (the subnet ID or IDs that you want to use)
--security-group-ids (the security group for the endpoint network interface, or omit to
use the default)
--private-dns-enabled
```

You can also use the [VPC console](#) to create these endpoints. See [Secure prediction calls in Amazon SageMaker with AWS PrivateLink](#) and [Securing all Amazon SageMaker API calls with AWS PrivateLink](#).

Create a SageMaker inference endpoint parameter in your DB cluster parameter group

To avoid having to specify the SageMaker inference endpoint of the model that you're using in every query you make to it, create a DB cluster parameter named `neptune_ml_endpoint` in the DB cluster parameter group for Neptune ML. Set the parameter to the `id` of the instance endpoint in question.

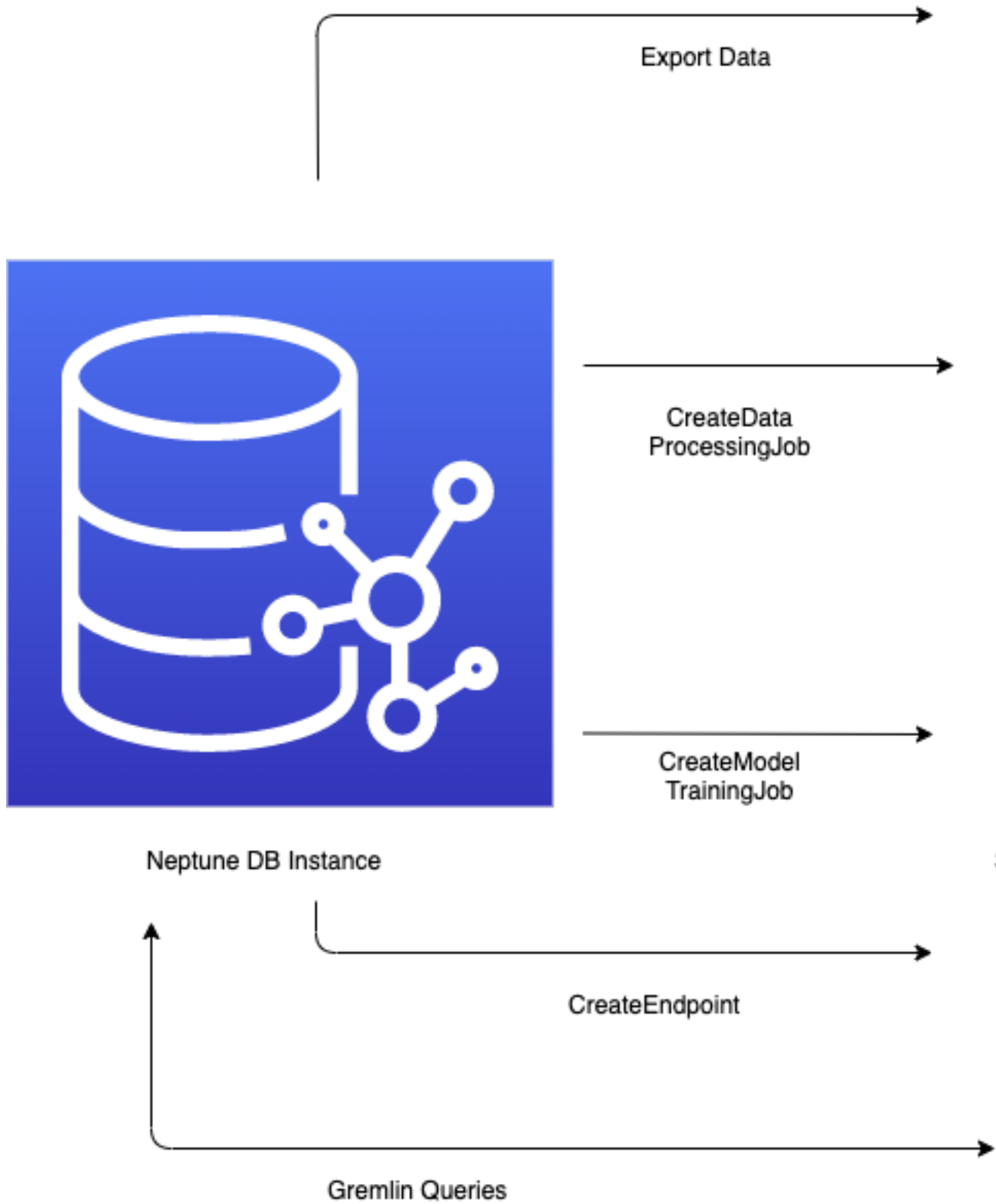
You can use the following AWS CLI command to do that:

```
aws neptune modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name neptune-ml-demo \
--parameters "ParameterName=neptune_ml_endpoint, \
ParameterValue=(the name of the SageMaker inference endpoint you want to
query), \
Description=NeptuneMLEndpoint, \
ApplyMethod=pending-reboot" \
--region (AWS region, such as us-east-1)
```

Overview of how to use the Neptune ML feature

Starting workflow for using Neptune ML

Using the Neptune ML feature in Amazon Neptune generally involves the following five steps to begin with:



1. **Data export and configuration** – The data-export step uses the Neptune-Export service or the `neptune-export` command line tool to export data from Neptune into Amazon Simple Storage Service (Amazon S3) in CSV form. A configuration file named `training-data-configuration.json` is automatically generated at the same time, which specifies how the exported data can be loaded into a trainable graph.
2. **Data preprocessing** – In this step, the exported dataset is preprocessed using standard techniques to prepare it for model training. Feature normalization can be performed for numeric data, and text features can be encoded using `word2vec`. At the end of this step, a DGL (Deep Graph library) graph is generated from the exported dataset for the model training step to use.

This step is implemented using a SageMaker processing job in your account, and the resulting data is stored in an Amazon S3 location that you have specified.

3. **Model training** – The model training step trains the machine learning model that will be used for predictions.

Model training is done in two stages:

- The first stage uses a SageMaker processing job to generate a model training strategy configuration set that specifies what type of model and model hyperparameter ranges will be used for the model training.
 - The second stage then uses a SageMaker model tuning job to try different hyperparameter configurations and select the training job that produced the best-performing model. The tuning job runs a pre-specified number of model training job trials on the processed data. At the end of this stage, the trained model parameters of the best training job are used to generate model artifacts for inference.
4. **Create an inference endpoint in Amazon SageMaker** – The inference endpoint is a SageMaker endpoint instance that is launched with the model artifacts produced by the best training job. Each model is tied to a single endpoint. The endpoint is able to accept incoming requests from the graph database and return the model predictions for inputs in the requests. After you have created the endpoint, it stays active until you delete it.
 5. **Query the machine learning model using Gremlin** – You can use extensions to the Gremlin query language to query predictions from the inference endpoint.

Note

The [Neptune workbench \(p. 30\)](#) contains a line magic and a cell magic that can save you a lot of time managing these steps, namely:

- [%neptune_ml \(p. 39\)](#)
- [%%neptune_ml \(p. 42\)](#)

Workflows for handling evolving graph data

With a continuously changing graph, you may need to update ML predictions frequently using the newest data. While you can do this simply by re-running the steps one through four (from **Data export and configuration** to **Create an inference endpoint in Amazon SageMaker**, Neptune ML supports simpler ways to update your ML predictions using new data. One is an incremental-model workflow:

Incremental-model inference workflow

In this workflow, you update the ML predictions without retraining the ML model.

Note

You can only do this when the graph data has been updated with new nodes and/or edges. It will not currently work when nodes are removed.

1. **Data export and configuration** – This step is the same as in the main workflow.
2. **Incremental data preprocessing** – This step is similar to the data preprocessing step in the main workflow, but uses the same processing configuration used previously, that corresponds to a specific trained model.
3. **Model transform** – Instead of a model training step, this model-transform step takes the trained model from the main workflow and the results of the incremental data preprocessing step, and generates new model artifacts to use for inference. The model-transform step launches a SageMaker processing job to perform the computation that generates the updated model artifacts.
4. **Update the Amazon SageMaker inference endpoint** – This step updates an existing inference endpoint with the new model artifacts generated by the model-transform step. Alternatively, you can also create a new inference endpoint with the new model artifacts.

Model re-training with a warm start

Using this workflow, you can train and deploy a new ML model for making predictions using the incremental graph data, but start from an existing model generated using the main workflow:

1. **Data export and configuration** – This step is the same as in the main workflow.
2. **Incremental data preprocessing** – This step is the same as in the incremental model inference workflow. The new graph data should be processed with the same processing method that was used previously for model training.
3. **Model training with a warm start** – Model training is similar to what happens in the main workflow, but you can speed up model hyperparameter search by leveraging the information from the previous model training task.
4. **Update the Amazon SageMaker inference endpoint** – This step is the same as in the incremental model inference workflow.

Workflows for custom models in Neptune ML

Neptune ML lets you implement, train and deploy custom models of your own for any of the tasks that Neptune ML supports. The workflow for developing and deploying a custom model is essentially the same as for the built-in models, with a few differences, as explained in [Custom model workflow](#) (p. 596).

Instance selection for the Neptune ML stages

The different stages of Neptune ML processing use different SageMaker instances. Here, we discuss how to choose the right instance type for each stage. You can find information about SageMaker instance types and pricing at [Amazon SageMaker Pricing](#).

Selecting an instance for data processing

The SageMaker [data-processing](#) (p. 564) step requires a [processing instance](#) that has enough memory and disk storage for the input, intermediate, and output data. The specific amount of memory and disk storage needed depends on the characteristics of the Neptune ML graph and its exported features.

By default, Neptune ML chooses the smallest `m1.r5` instance whose memory is ten times larger than the size of the exported graph data on disk.

Selecting an instance for model training and model transform

Selecting the right instance type for [model training](#) or [model transform](#) depends on the task type, the graph size, and your turn-around requirements. GPU instances provide the best performance. We generally recommend `p3` and `g4dn` serial instances. You can also use `p2` or `p4d` instances.

By default, Neptune ML chooses the smallest GPU instance with more memory than model training and model transform requires. You can find what that selection is in the `train_instance_recommendation.json` file, in the Amazon S3 data processing output location. Here is an example of the contents of a `train_instance_recommendation.json` file:

```
{
  "instance":      "(the recommended instance type for model training and transform)",
  "cpu_instance": "(the recommended instance type for base processing instance)",
  "disk_size":    "(the estimated disk space required)",
  "mem_size":     "(the estimated memory required)"
}
```

Selecting an instance for an inference endpoint

Selecting the right instance type for an [inference endpoint](#) (p. 602) depends on the task type, the graph size and your budget. By default, Neptune ML chooses the smallest `m1.m5d` instance with more memory the inference endpoint requires.

Note

If more than 384 GB of memory is needed, Neptune ML uses an `m1.r5d.24xlarge` instance.

You can see what instance type Neptune ML recommends in the `infer_instance_recommendation.json` file located in the Amazon S3 location you are using for model training. Here is an example of that file's contents:

```
{
  "instance" : "(the recommended instance type for an inference endpoint)",
  "disk_size" : "(the estimated disk space required)",
  "mem_size" : "(the estimated memory required)"
}
```

Using the `neptune-export` tool or Neptune-Export service to export data from Neptune for Neptune ML

Neptune ML requires that you provide training data for the [Deep Graph Library \(DGL\)](#) to create and evaluate models.

You can export data from Neptune using either the [Neptune-Export service \(p. 235\)](#), or [`neptune-export` utility \(p. 240\)](#). Both the service and the command line tool publish data to Amazon Simple Storage Service (Amazon S3) in a CSV format, encrypted using Amazon S3 server-side encryption (SSE-S3). See [Files exported by Neptune-Export and `neptune-export` \(p. 243\)](#).

In addition, when you configure an export of training data for Neptune ML the export job creates and publishes an encrypted model-training configuration file along with the exported data. By default, this file is named `training-data-configuration.json`.

Examples of using the Neptune-Export service to export training data for Neptune ML

This request exports property-graph training data for a node classification task:

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
    "command": "export-pg",
    "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
    "params": {
      "endpoint": "(your Neptune endpoint DNS name)",
      "profile": "neptune_ml"
    },
    "additionalParams": {
      "neptune_ml": {
        "version": "v2.0",
        "targets": [
          {
            "node": "Movie",
            "property": "genre",
            "type": "classification"
          }
        ]
      }
    }
  }' \
  --region us-east-1
```

This request exports RDF training data for a node classification task:

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
    "command": "export-rdf",
```

```
"outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
"params": {
  "endpoint": "(your Neptune endpoint DNS name)",
  "profile": "neptune_ml"
},
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/
genre",
        "type": "classification"
      }
    ]
  }
}
}' \
--region us-east-1
```

Fields to set in the `params` object when exporting training data

The `params` object in an export request can contain various fields, as described in the [params documentation \(p. 247\)](#). The following ones are most relevant for exporting machine-learning training data:

- **endpoint** – Use `endpoint` to specify an endpoint of a Neptune instance in your DB cluster that the export process can query to extract data.
- **profile** – The `profile` field in the `params` object must be set to `neptune-ml`.

This causes the export process to format the exported data appropriately for Neptune ML model training, in a CSV format for property-graph data or as N-Triples for RDF data. It also causes a `training-data-configuration.json` file to be created and written to the same Amazon S3 location as the exported training data.

- **cloneCluster** – If set to `true`, the export process clones your DB cluster, exports from the clone, and then deletes the clone when it is finished.
- **useIamAuth** – If your DB cluster has [IAM authentication \(p. 117\)](#) enabled, you must include this field set to `true`.

The export process also provides several ways to filter the data you export (see [these examples \(p. 255\)](#)).

Using the `additionalParams` object to tune the export of model-training information

The `additionalParams` object contains fields that you can use to specify machine-learning class labels and features for training purposes and guide the creation of a training data configuration file.

The export process cannot automatically infer which node and edge properties should be the machine learning class labels to serve as examples for training purposes. It also cannot automatically infer the best feature encoding for numeric, categorical and text properties, so you need to supply hints using fields in the `additionalParams` object to specify these things, or to override the default encoding.

For property-graph data, the top-level structure of `additionalParams` in an export request might look like this:

```
{
  "command": "export-pg",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams": {
    "neptune_ml": {
      "version": "v2.0",
      "targets": [ (an array of node and edge class label targets) ],
      "features": [ (an array of node feature hints) ]
    }
  }
}
```

For RDF data, its top-level structure might look like this:

```
{
  "command": "export-rdf",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams": {
    "neptune_ml": {
      "version": "v2.0",
      "targets": [ (an array of node and edge class label targets) ]
    }
  }
}
```

You can also supply multiple export configurations, using the `jobs` field:

```
{
  "command": "export-pg",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams" : {
    "neptune_ml" : {
      "version": "v2.0",
      "jobs": [
        {
          "name" : "(training data configuration name)",
          "targets": [ (an array of node and edge class label targets) ],
          "features": [ (an array of node feature hints) ]
        },
        {
          "name" : "(another training data configuration name)",
          "targets": [ (an array of node and edge class label targets) ],
          "features": [ (an array of node feature hints) ]
        }
      ]
    }
  }
}
```

```
}
```

Top-level elements in the `neptune_ml` field in `additionalParams`

The `version` element in `neptune_ml`

Specifies the version of training data configuration to generate.

(Optional), Type: string, Default: "v2.0".

If you do include `version`, set it to `v2.0`.

The `jobs` field in `neptune_ml`

Contains an array of training-data configuration objects, each of which defines a data processing job, and contains:

- **name** – The name of the training data configuration to be created.

For example, a training data configuration with the name "job-number-1" results in a training data configuration file named `job-number-1.json`.

- **targets** – A JSON array of node and edge class label targets that represent the machine-learning class labels for training purposes. See [The `targets` field in a `neptune_ml` object \(p. 544\)](#).
- **features** – A JSON array of node property features. See [The `features` field in `neptune_ml` \(p. 549\)](#).

The targets field in a neptune_ml object

The targets field in a JSON training data export configuration contains an array of target objects that specify a training task and the machine-learning class labels for training this task. The contents of the target objects varies depending on whether you are training on property-graph data or RDF data.

For property-graph node classification and regression tasks, target objects in the array can look like this:

```
{
  "node": "(node property-graph label)",
  "property": "(property name)",
  "type": "(used to specify classification or regression)",
  "split_rate": [0.8,0.2,0.0],
  "separator": ","
}
```

For property-graph edge classification, regression or link prediction tasks, they can look like this:

```
{
  "edge": "(edge property-graph label)",
  "property": "(property name)",
  "type": "(used to specify classification, regression or link_prediction)",
  "split_rate": [0.8,0.2,0.0],
  "separator": ","
}
```

For RDF classification and regression tasks, target objects in the array can look like this:

```
{
  "node": "(node type of an RDF node)",
  "predicate": "(predicate IRI)",
  "type": "(used to specify classification or regression)",
  "split_rate": [0.8,0.2,0.0]
}
```

For RDF link prediction tasks, target objects in the array can look like this:

```
{
  "subject": "(source node type of an edge)",
  "predicate": "(relation type of an edge)",
  "object": "(destination node type of an edge)",
  "type": "link_prediction",
  "split_rate": [0.8,0.2,0.0]
}
```

Target objects can contain the following fields:

Contents

- [Fields in a property-graph target object \(p. 545\)](#)
 - [The node \(vertex\) field in a target object \(p. 545\)](#)
 - [The edge field in a property-graph target object \(p. 545\)](#)
 - [The property field in a property-graph target object \(p. 545\)](#)
 - [The type field in a property-graph target object \(p. 545\)](#)
 - [The split_rate field in a property-graph target object \(p. 546\)](#)
 - [The separator field in a property-graph target object \(p. 546\)](#)

- [Fields in an RDF target object \(p. 546\)](#)
 - [The node field in an RDF target object \(p. 546\)](#)
 - [The subject field in an RDF target object \(p. 546\)](#)
 - [The predicate field in an RDF target object \(p. 547\)](#)
 - [The object field in an RDF target object \(p. 547\)](#)
 - [The type field in an RDF target object \(p. 547\)](#)
 - [The split_rate field in a property-graph target object \(p. 547\)](#)

Fields in a property-graph target object

The node (vertex) field in a target object

The property-graph label of a target node (vertex). A target object must contain a node element or an edge element, but not both.

A node can take either a single value, like this:

```
"node": "Movie"
```

Or, in the case of a multi-label vertex, it can take an array of values, like this:

```
"node": ["Content", "Movie"]
```

The edge field in a property-graph target object

Specifies a target edge by its start node label(s), its own label, and its end-node label(s). A target object must contain an edge element or a node element, but not both.

The value of an edge field is a JSON array of three strings that represent the start-node's property-graph label(s), the property-graph label of the edge itself, and the end-node's property-graph label(s), like this:

```
"edge": ["Person_A", "knows", "Person_B"]
```

If the start node and/or end node has multiple labels, enclose them in an array, like this:

```
"edge": [ ["Admin", "Person_A"], "knows", ["Admin", "Person_B"] ]
```

The property field in a property-graph target object

Specifies a property of the target vertex or edge, like this:

```
"property" : "rating"
```

This field is required, except when the target task is link prediction.

The type field in a property-graph target object

Indicates the type of target task to be performed on the node or edge, like this:

```
"type" : "regression"
```

The supported task types for nodes are:

- `classification`
- `regression`

The supported task types for edges are:

- `classification`
- `regression`
- `link_prediction`

This field is required.

The `split_rate` field in a property-graph target object

(*Optional*) An estimate of the proportions of nodes or edges that the training, validation, and test stages will use, respectively. These proportions are represented by a JSON array of three numbers between zero and one that add up to one:

```
"split_rate": [0.7, 0.1, 0.2]
```

If you do not supply the optional `split_rate` field, the default estimated value is `[0.9, 0.1, 0.0]`.

The `separator` field in a property-graph target object

(*Optional*) Used with a classification task.

The `separator` field specifies a character used to split a target property value into multiple categorical values when it is used to store multiple category values in a string. For example:

```
"separator": "|"
```

The presence of a `separator` field indicates that the task is a multi-target classification task.

Fields in an RDF target object

The `node` field in an RDF target object

Defines the node type of target nodes. Used with node classification tasks or node regression tasks. The node type of a node in RDF is defined by:

```
node_id, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, node_type
```

An RDF node can only take a single value, like this:

```
"node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie"
```

The `subject` field in an RDF target object

For link prediction tasks, `subject` defines the source node type of target edges.

```
"subject": "http://aws.amazon.com/neptune/csv2rdf/class/Director"
```

Note

For link prediction tasks, `subject` should be used together with `predicate` and `object`. If any of these three is not provided, all edges are treated as the training target.

The `predicate` field in an RDF target object

For node classification and node regression tasks, `predicate` defines what literal data is used as the target node feature of a target node.

```
"predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/genre"
```

Note

If the target nodes have only one predicate defining the target node feature, the `predicate` field can be omitted.

For link prediction tasks, `predicate` defines the relation type of target edges:

```
"predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/direct"
```

Note

For link prediction tasks, `predicate` should be used together with `subject` and `object`. If any of these three is not provided, all edges are treated as the training target.

The `object` field in an RDF target object

For link prediction tasks, `object` defines the destination node type of target edges:

```
"object": "http://aws.amazon.com/neptune/csv2rdf/class/Movie"
```

Note

For link prediction tasks, `object` should be used together with `subject` and `predicate`. If any of these three is not provided, all edges are treated as the training target.

The `type` field in an RDF target object

Indicates the type of target task to be performed, like this:

```
"type" : "regression"
```

The supported task types for RDF data are:

- `link_prediction`
- `classification`
- `regression`

This field is required.

The `split_rate` field in a property-graph target object

(Optional) An estimate of the proportions of nodes or edges that the training, validation, and test stages will use, respectively. These proportions are represented by a JSON array of three numbers between zero and one that add up to one:

```
"split_rate": [0.7, 0.1, 0.2]
```

If you do not supply the optional `split_rate` field, the default estimated value is `[0.9, 0.1, 0.0]`.

The features field in `neptune_ml`

Property values and RDF literals come in different formats and data types. To achieve good performance in machine learning, it is essential to convert those values to numerical encodings known as *features*.

Neptune ML performs feature extraction and encoding as part of the data-export and data-processing steps, as described in [Feature encoding in Neptune ML \(p. 566\)](#).

For property-graph datasets, the export process automatically infers `auto` features for string properties and for numeric properties that contain multiples values. For numeric properties containing single values, it infers `numerical` features. For date properties it infers `datetime` features.

If you want to override an auto-inferred feature specification, or add a bucket numerical, TF-IDF, or Word2Vec specification for a property, you can control the feature encoding using the `features` field.

Note

You can only use the `features` field to control the feature specifications for property-graph data, not for RDF data.

The `features` field contains a JSON array of node property features. Objects in the array can contain the following fields:

Contents

- [The node field in features \(p. 549\)](#)
- [The edge field in features \(p. 549\)](#)
- [The property field in features \(p. 550\)](#)
- [Possible values of the type field for features \(p. 550\)](#)
- [The norm field \(p. 551\)](#)
- [The language field \(p. 551\)](#)
- [The separator field \(p. 551\)](#)
- [The range field \(p. 552\)](#)
- [The bucket_cnt field \(p. 552\)](#)
- [The slide_window_size field \(p. 552\)](#)
- [The imputer field \(p. 552\)](#)
- [The max_features field \(p. 553\)](#)
- [The min_df field \(p. 553\)](#)
- [The ngram_range field \(p. 553\)](#)
- [The datetime_parts field \(p. 553\)](#)

The node field in features

The `node` field specifies a property-graph label of a feature vertex. For example:

```
"node": "Person"
```

If a vertex has multiple labels, use an array to contain them. For example:

```
"node": ["Admin", "Person"]
```

The edge field in features

The `edge` field specifies the edge type of a feature edge. An edge type consists of an array containing the property-graph label(s) of the source vertex, the property-graph label of the edge, and the property-

graph label(s) of the destination vertex. You must supply all three values when specifying an edge feature. For example:

```
"edge": ["User", "reviewed", "Movie"]
```

If a source or destination vertex of an edge type has multiple labels, use another array to contain them. For example:

```
"edge": [["Admin", "Person"], "edited", "Post"]
```

The property field in features

Use the property parameter to specify a property of the vertex identified by the `node` parameter. For example:

```
"property" : "age"
```

Possible values of the type field for features

The `type` parameter specifies the type of feature being defined. For example:

```
"type": "bucket_numerical"
```

Possible values of the `type` parameter

- **"auto"** – Specifies that Neptune ML should automatically detect the property type and apply a proper feature encoding. An `auto` feature can also have an optional `separator` field.

See [Auto feature encoding in Neptune ML \(p. 568\)](#).
- **"category"** – This feature encoding represents a property value as one of a number of categories. In other words, the feature can take one or more discrete values. A `category` feature can also have an optional `separator` field.

See [Categorical features in Neptune ML \(p. 566\)](#).

- **"numerical"** – This feature encoding represents numerical property values as numbers in a continuous interval where "greater than" and "less than" have meaning.

A `numerical` feature can also have optional `norm`, `imputer`, and `separator` fields.

See [Numerical features in Neptune ML \(p. 566\)](#).

- **"bucket_numerical"** – This feature encoding divides numerical property values into a set of *buckets* or categories.

For example, you could encode people's ages in 4 buckets: kids (0-20), young-adults (20-40), middle-aged (40-60), and elders (60 and up).

A `bucket_numerical` feature requires a `range` and a `bucket_cnt` field, and can optionally also include an `imputer` and/or `slide_window_size` field.

See [Bucket-numerical features in Neptune ML \(p. 566\)](#).

- **"datetime"** – This feature encoding represents a datetime property value as an array of these categorical features: year, month, weekday, and hour.

One or more of these four categories can be eliminated using the `datetime_parts` parameter.

See [Datetime features in Neptune ML \(p. 568\)](#).

- **"text_tfidf"** – This feature encoding converts property values that consist of sentences or free-form text into numeric vectors using a TF-IDF vectorizer.

A `text_tfidf` feature encoding must also have an `ngram_range` field, a `min_df` field, and a `max_features` field to properly define it.

See [Text TF-IDF features in Neptune ML \(p. 567\)](#).

- **"text_word2vec"** – This feature encoding converts property values that consist of sentences or free-form text into numeric vectors using Word2Vec models.

A `text_word2vec` feature can also have an optional `language` field.

See [Text Word2Vec features in Neptune ML \(p. 567\)](#).

- **"none"** – Using the `none` type causes no feature encoding to occur. The raw property values are parsed and saved instead.

Use `none` only if you plan to perform your own custom feature encoding as part of custom model training.

The norm field

This field is required for numerical features. It specifies a normalization method to use on numeric values:

```
"norm": "min-max"
```

The following normalization methods are supported:

- **"min-max"** – Normalize each value by subtracting the minimum value from it and then dividing it by the difference between the maximum value and the minimum.
- **"standard"** – Normalize each value by dividing it by the sum of all the values.
- **"none"** – Don't normalize the numerical values during encoding.

See [Numerical features in Neptune ML \(p. 566\)](#).

The language field

This field is required for `text_word2vec` features. It specifies the name of the language model used to encode the property value string:

```
"language" : "en_core_web_lg"
```

Neptune ML currently only supports: `"en_core_web_lg"` (for English). The specific language model can only work with its target language, so the output embedding is not guaranteed to be valid if you supply other languages.

See [Text Word2Vec features in Neptune ML \(p. 567\)](#).

The separator field

This field is used optionally with `category`, `numerical` and `auto` features. It specifies a character that can be used to split a property value into multiple categorical values or numerical values:


```
"separator": ";"
```

Only use the `separator` field when the property stores multiple delimited values in a single string, such as "Actor;Director" or "0.1;0.2".

See [Categorical features \(p. 566\)](#), [Numerical features \(p. 566\)](#), and [Auto encoding \(p. 568\)](#).

The `range` field

This field is required for `bucket_numerical` features. It specifies the range of numerical values that are to be divided into buckets, in the format [`lower-bound`, `upper-bound`]:

```
"range" : [20, 100]
```

If a property value is smaller than the lower bound then it is assigned to the first bucket, or if it's larger than the upper bound, it's assigned to the last bucket.

See [Bucket-numerical features in Neptune ML \(p. 566\)](#).

The `bucket_cnt` field

This field is required for `bucket_numerical` features. It specifies the number of buckets that the numerical range defined by the `range` parameter should be divided into:

```
"bucket_cnt": 10
```

See [Bucket-numerical features in Neptune ML \(p. 566\)](#).

The `slide_window_size` field

This field is used optionally with `bucket_numerical` features to assign values to more than one bucket:

```
"slide_window_size": 5
```

The way a slide window works is that Neptune ML takes the window size s and transforms each numeric value v of a property into a range from $v - s/2$ through $v + s/2$. The value is then assigned to every bucket that the range overlaps.

See [Bucket-numerical features in Neptune ML \(p. 566\)](#).

The `imputer` field

This field is used optionally with `numerical` and `bucket_numerical` features to provide an imputation technique for filling in missing values:

```
"imputer": "mean"
```

The supported imputation techniques are:

- "mean"
- "median"
- "most-frequent"

If you don't include the `imputer` parameter, data preprocessing halts and exits when a missing value is encountered.

See [Numerical features in Neptune ML \(p. 566\)](#) and [Bucket-numerical features in Neptune ML \(p. 566\)](#).

The `max_features` field

This field is used optionally by `text_tfidf` features to specify the maximum number of terms to encode:

```
"max_features": 100
```

A setting of 100 causes the TF-IDF vectorizer to encode only the 100 most common terms. The default value if you don't include `max_features` is 5,000.

See [Text TF-IDF features in Neptune ML \(p. 567\)](#).

The `min_df` field

This field is used optionally by `text_tfidf` features to specify the minimum document frequency of terms to encode:

```
"min_df": 5
```

A setting of 5 indicates that a term must appear in at least 5 different property values in order to be encoded.

The default value if you don't include the `min_df` parameter is 2.

See [Text TF-IDF features in Neptune ML \(p. 567\)](#).

The `ngram_range` field

This field is used optionally by `text_tfidf` features to specify what size sequences of words or tokens should be considered as potential individual terms to encode:

```
"ngram_range": [2, 4]
```

The value `[2, 4]` specifies that sequences of 2, 3 and 4 words should be considered as potential individual terms.

The default if you don't explicitly set `ngram_range` is `[1, 1]`, meaning that only single words or tokens are considered as terms to encode.

See [Text TF-IDF features in Neptune ML \(p. 567\)](#).

The `datetime_parts` field

This field is used optionally by `datetime` features to specify which parts of the datetime value to encode categorically:

```
"datetime_parts": ["weekday", "hour"]
```

If you don't include `datetime_parts`, by default Neptune ML encodes the year, month, weekday and hour parts of the datetime value. The value ["weekday", "hour"] indicates that only the weekday and hour of datetime values should be encoded categorically in the feature.

If one of the parts does not have more than one unique value in the training set, it is not encoded.

See [Datetime features in Neptune ML \(p. 568\)](#).

Examples of using parameters within `additionalParams` for tuning model-training configuration

Contents

- [Property-graph examples using `additionalParams` \(p. 555\)](#)
 - [Specifying a default split rate for model-training configuration \(p. 555\)](#)
 - [Specifying a node-classification task for model-training configuration \(p. 556\)](#)
 - [Specifying a multi-class node classification task for model-training configuration \(p. 556\)](#)
 - [Specifying a node regression task for model-training configuration \(p. 557\)](#)
 - [Specifying an edge-classification task for model-training configuration \(p. 557\)](#)
 - [Specifying a multi-class edge classification task for model-training configuration \(p. 557\)](#)
 - [Specifying an edge regression for model-training configuration \(p. 558\)](#)
 - [Specifying a link prediction task for model-training configuration \(p. 558\)](#)
 - [Specifying a numerical bucket feature \(p. 559\)](#)
 - [Specifying a Word2Vec feature \(p. 559\)](#)
 - [Specifying a TF-IDF feature \(p. 559\)](#)
 - [Specifying a datetime feature \(p. 560\)](#)
 - [Specifying a category feature \(p. 560\)](#)
 - [Specifying a numerical feature \(p. 561\)](#)
 - [Specifying an auto feature \(p. 561\)](#)
- [RDF examples using `additionalParams` \(p. 562\)](#)
 - [Specifying a default split rate for model-training configuration \(p. 562\)](#)
 - [Specifying a node-classification task for model-training configuration \(p. 562\)](#)
 - [Specifying a node regression task for model-training configuration \(p. 562\)](#)
 - [Specifying a link prediction task for particular edges \(p. 563\)](#)
 - [Specifying a link prediction task for all edges \(p. 563\)](#)

Property-graph examples using `additionalParams`

Specifying a default split rate for model-training configuration

In the following example, the `split_rate` parameter sets the default split rate for model training. If no default split rate is specified, the training uses a value of [0.9, 0.1, 0.0]. You can override the default value on a per-target basis by specifying a `split_rate` for each target.

In the following example, the default `split_rate` field indicates that a split rate of [0.7, 0.1, 0.2] should be used unless overridden on a per-target basis:"

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "split_rate": [0.7,0.1,0.2],
    "targets": [
      (...)
    ],
    "features": [
      (...)
    ]
  }
}
```

```
}  
}
```

Specifying a node-classification task for model-training configuration

To indicate which node property contains labeled examples for training purposes, add a node classification element to the `targets` array, using `"type" : "classification"`. Add a `split_rate` field if you want to override the default split rate.

In the following example, the node target indicates that the `genre` property of each `Movie` node should be treated as a node class label. The `split_rate` value overrides the default split rate:

```
"additionalParams": {  
  "neptune_ml": {  
    "version": "v2.0",  
    "targets": [  
      {  
        "node": "Movie",  
        "property": "genre",  
        "type": "classification",  
        "split_rate": [0.7,0.1,0.2]  
      }  
    ],  
    "features": [  
      (...)  
    ]  
  }  
}
```

Specifying a multi-class node classification task for model-training configuration

To indicate which node property contains multiple labeled examples for training purposes, add a node classification element to the `targets` array, using `"type" : "classification"`, and `separator` to specify a character that can be used to split a target property value into multiple categorical values. Add a `split_rate` field if you want to override the default split rate.

In the following example, the node target indicates that the `genre` property of each `Movie` node should be treated as a node class label. The `separator` field indicates that each `genre` property contains multiple semicolon-separated values:

```
"additionalParams": {  
  "neptune_ml": {  
    "version": "v2.0",  
    "targets": [  
      {  
        "node": "Movie",  
        "property": "genre",  
        "type": "classification",  
        "separator": ";"  
      }  
    ],  
    "features": [  
      (...)  
    ]  
  }  
}
```

Specifying a node regression task for model-training configuration

To indicate which node property contains labeled regressions for training purposes, add a node regression element to the `targets` array, using `"type" : "regression"`. Add a `split_rate` field if you want to override the default split rate.

The following node target indicates that the `rating` property of each `Movie` node should be treated as a node regression label:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "Movie",
        "property": "rating",
        "type": "regression",
        "split_rate": [0.7,0.1,0.2]
      }
    ],
    "features": [
      ...
    ]
  }
}
```

Specifying an edge-classification task for model-training configuration

To indicate which edge property contains labeled examples for training purposes, add an edge element to the `targets` array, using `"type" : "regression"`. Add a `split_rate` field if you want to override the default split rate.

The following edge target indicates that the `metAtLocation` property of each `knows` edge should be treated as an edge class label:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "knows", "Person"],
        "property": "metAtLocation",
        "type": "classification"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

Specifying a multi-class edge classification task for model-training configuration

To indicate which edge property contains multiple labeled examples for training purposes, add an edge element to the `targets` array, using `"type" : "classification"`, and a `separator` field to specify a character used to split a target property value into multiple categorical values. Add a `split_rate` field if you want to override the default split rate.

The following edge target indicates that the `sentiment` property of each `repliedTo` edge should be treated as an edge class label. The `separator` field indicates that each `sentiment` property contains multiple comma-separated values:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "repliedTo", "Message"],
        "property": "sentiment",
        "type": "classification",
        "separator": ",",
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

Specifying an edge regression for model-training configuration

To indicate which edge property contains labeled regression examples for training purposes, add an edge element to the `targets` array, using `"type" : "regression"`. Add a `split_rate` field if you want to override the default split rate.

The following edge target indicates that the `rating` property of each reviewed edge should be treated as an edge regression:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "reviewed", "Movie"],
        "property": "rating",
        "type" : "regression"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

Specifying a link prediction task for model-training configuration

To indicate which edges should be used for link prediction training purposes, add an edge element to the `targets` array using `"type" : "link_prediction"`. Add a `split_rate` field if you want to override the default split rate.

The following edge target indicates that `cites` edges should be used for link prediction:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Article", "cites", "Article"],
        "type" : "link_prediction"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

```
    ]  
  }  
}
```

Specifying a numerical bucket feature

You can specify a numerical data feature for a node property by adding "type": "bucket_numerical" to the features array.

The following node feature indicates that the age property of each Person node should be treated as a numerical bucket feature:

```
"additionalParams": {  
  "neptune_ml": {  
    "targets": [  
      ...  
    ],  
    "features": [  
      {  
        "node": "Person",  
        "property": "age",  
        "type": "bucket_numerical",  
        "range": [1, 100],  
        "bucket_cnt": 5,  
        "slide_window_size": 3,  
        "imputer": "median"  
      }  
    ]  
  }  
}
```

Specifying a Word2Vec feature

You can specify a Word2Vec feature for a node property by adding "type": "text_word2vec" to the features array.

The following node feature indicates that the description property of each Movie node should be treated as a Word2Vec feature:

```
"additionalParams": {  
  "neptune_ml": {  
    "version": "v2.0",  
    "targets": [  
      ...  
    ],  
    "features": [  
      {  
        "node": "Movie",  
        "property": "description",  
        "type": "text_word2vec",  
        "language": "en_core_web_lg"  
      }  
    ]  
  }  
}
```

Specifying a TF-IDF feature

You can specify a TF-IDF feature for a node property by adding "type": "text_tfidf" to the features array.

The following node feature indicates that the `bio` property of each `Person` node should be treated as a TF-IDF feature:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "bio",
        "type": "text_tfidf",
        "ngram_range": [1, 2],
        "min_df": 5,
        "max_features": 1000
      }
    ]
  }
}
```

Specifying a datetime feature

The export process automatically infers `datetime` features for date properties. However, if you want to limit the `datetime_parts` used for a `datetime` feature, or override a feature specification so that a property that would normally be treated as an `auto` feature is explicitly treated as a `datetime` feature, you can do so by adding a `"type": "datetime"` to the features array.

The following node feature indicates that the `createdAt` property of each `Post` node should be treated as a `datetime` feature:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Post",
        "property": "createdAt",
        "type": "datetime",
        "datetime_parts": ["month", "weekday", "hour"]
      }
    ]
  }
}
```

Specifying a category feature

The export process automatically infers `auto` features for string properties and numeric properties containing multiples values. For numeric properties containing single values, it infers `numerical` features. For date properties it infers `datetime` features.

If you want to override a feature specification so that a property is treated as a categorical feature, add a `"type": "category"` to the features array. If the property contains multiple values, include a `separator` field. For example:

```
"additionalParams": {
```

```
"neptune_ml": {
  "version": "v2.0",
  "targets": [
    ...
  ],
  "features": [
    {
      "node": "Post",
      "property": "tag",
      "type": "category",
      "separator": "|"
    }
  ]
}
```

Specifying a numerical feature

The export process automatically infers `auto` features for string properties and numeric properties containing multiples values. For numeric properties containing single values, it infers `numerical` features. For date properties it infers `datetime` features.

If you want to override a feature specification so that a property is treated as a `numerical` feature, add `"type": "numerical"` to the features array. If the property contains multiple values, include a `separator` field. For example:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Recording",
        "property": "duration",
        "type": "numerical",
        "separator": ","
      }
    ]
  }
}
```

Specifying an auto feature

The export process automatically infers `auto` features for string properties and numeric properties containing multiples values. For numeric properties containing single values, it infers `numerical` features. For date properties it infers `datetime` features.

If you want to override a feature specification so that a property is treated as an `auto` feature, add `"type": "auto"` to the features array. If the property contains multiple values, include a `separator` field. For example:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {

```

```
    "node": "User",
    "property": "role",
    "type": "auto",
    "separator": ",",
  }
]
}
}
```

RDF examples using `additionalParams`

Specifying a default split rate for model-training configuration

In the following example, the `split_rate` parameter sets the default split rate for model training. If no default split rate is specified, the training uses a value of `[0.9, 0.1, 0.0]`. You can override the default value on a per-target basis by specifying a `split_rate` for each target.

In the following example, the default `split_rate` field indicates that a split rate of `[0.7, 0.1, 0.2]` should be used unless overridden on a per-target basis:"

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "split_rate": [0.7,0.1,0.2],
    "targets": [
      (...)
    ]
  }
}
```

Specifying a node-classification task for model-training configuration

To indicate which node property contains labeled examples for training purposes, add a node classification element to the `targets` array, using `"type" : "classification"`. Add a `node` field to indicate the node type of target nodes. Add a `predicate` field to define which literal data is used as the target node feature of the target node. Add a `split_rate` field if you want to override the default split rate.

In the following example, the `node` target indicates that the `genre` property of each `Movie` node should be treated as a node class label. The `split_rate` value overrides the default split rate:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/genre",
        "type": "classification",
        "split_rate": [0.7,0.1,0.2]
      }
    ]
  }
}
```

Specifying a node regression task for model-training configuration

To indicate which node property contains labeled regressions for training purposes, add a node regression element to the `targets` array, using `"type" : "regression"`. Add a `node` field to indicate

the node type of target nodes. Add a `predicate` field to define which literal data is used as the target node feature of the target node. Add a `split_rate` field if you want to override the default split rate.

The following node target indicates that the `rating` property of each `Movie` node should be treated as a node regression label:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/rating",
        "type": "regression",
        "split_rate": [0.7,0.1,0.2]
      }
    ]
  }
}
```

Specifying a link prediction task for particular edges

To indicate which edges should be used for link prediction training purposes, add an edge element to the `targets` array using `"type" : "link_prediction"`. Add `subject`, `predicate` and `object` fields to specify the edge type. Add a `split_rate` field if you want to override the default split rate.

The following edge target indicates that directed edges that connect `Directors` to `Movies` should be used for link prediction:

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "subject": "http://aws.amazon.com/neptune/csv2rdf/class/Director",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/directed",
        "object": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "type": "link_prediction"
      }
    ]
  }
}
```

Specifying a link prediction task for all edges

To indicate that all edges should be used for link prediction training purposes, add an edge element to the `targets` array using `"type" : "link_prediction"`. Do not add `subject`, `predicate`, or `object` fields. Add a `split_rate` field if you want to override the default split rate.

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "type": "link_prediction"
      }
    ]
  }
}
```

Processing the graph data exported from Neptune for training

The data-processing step takes the Neptune graph data created by the export process and creates the information that is used by the [Deep Graph Library \(DGL\)](#) during training. This includes performing various data mappings and transformations:

- Parsing nodes and edges to construct the graph- and ID-mapping files required by DGL.
- Converting node and edge properties into the node and edge features required by DGL.
- Splitting the data into training, validation, and test sets.

Managing the data-processing step for Neptune ML

After you have exported the data from Neptune that you want to use for model training, you can start a data-processing job using a `curl` (or `awscli`) command like the following:

```
curl \
-X POST https://(your Neptune endpoint)/ml/dataprocessing \
-H 'Content-Type: application/json' \
-d '{
  "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input folder)",
  "id" : "(a job ID for the new job)",
  "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output folder)",
  "configFileName" : "training-job-configuration.json"
}'
```

The details of how to use this command are explained in [The dataprocessing command \(p. 620\)](#), along with information about how to get the status of a running job, how to stop a running job, and how to list all running jobs.

Processing updated graph data for Neptune ML

You can also supply a `previousDataProcessingJobId` to the API to ensure that the new data processing job uses the same processing method as a previous job. This is required when you want to get predictions for updated graph data in Neptune, either by retraining the old model on the new data, or by recomputing the model artifacts on the new data.

You do this by using a `curl` (or `awscli`) command like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/dataprocessing \
-H 'Content-Type: application/json' \
-d '{ "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input folder)",
      "id" : "(a job ID for the new job)",
      "processedDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your output folder)",
      "previousDataProcessingJobId", "(the job ID of the previous data-processing job)}'
```

Set the value of the `previousDataProcessingJobId` parameter to the job ID of the previous-data processing job that corresponds to the trained model.

Note

Node deletions in the updated graph are currently not supported. If nodes have been removed in an updated graph, you have to start a completely new data processing job rather than use `previousDataProcessingJobId`.

Feature encoding in Neptune ML

Property values come in different formats and data types. To achieve good performance in machine learning, it is essential to convert those values to numerical encodings known as *features*.

Neptune ML performs feature extraction and encoding as part of the data-export and data-processing steps, using feature-encoding techniques described here.

Note

If you plan to implement your own feature encoding in a custom model implementation, you can disable the automatic feature encoding in the data preprocessing stage by selecting `none` as the feature encoding type. No feature encoding then occurs on that node or edge property, and instead the raw property values are parsed and saved in a dictionary. Data preprocessing still creates the DGL graph from the exported dataset, but the constructed DGL graph doesn't have the pre-processed features for training.

You should use this option only if you plan to perform your custom feature encoding as part of custom model training. See [Custom models in Neptune ML \(p. 595\)](#) for details.

Categorical features in Neptune ML

A property that can take one or more distinct values from a fixed list of possible values is a categorical feature. In Neptune ML, categorical features are encoded using [one-hot encoding](#). The following example shows how the property name of different foods is one-hot encoded according to its category:

Food	Veg.	Meat	Fruit	Encoding
Apple	0	0	1	001
Chicken	0	1	0	010
Broccoli	1	0	0	100

Note

The maximum number of categories in any categorical feature is 100. If a property has more than 100 categories of value, only the most common 99 of them are placed in distinct categories, and the rest are placed in a special category named `OTHER`.

Numerical features in Neptune ML

Any property whose values are real numbers can be encoded as a numerical feature in Neptune ML. Numerical features are encoded using floating-point numbers.

You can specify a data-normalization method to use when encoding numerical features, like this: `"norm": "normalization technique"`. The following normalization techniques are supported:

- **"none"** – Don't normalize the numerical values during encoding.
- **"min-max"** – Normalize each value by subtracting the minimum value from it and then dividing it by the difference between the maximum value and the minimum.
- **"standard"** – Normalize each value by dividing it by the sum of all the values.

Bucket-numerical features in Neptune ML

Rather than representing a numerical property using raw numbers, you can condense numerical values into categories. For example, you could divide people's ages into categories such as kids (0-20), young adults (20-40), middle-aged people (40-60) and elders (from 60 on). Using these numerical buckets, you would be transforming a numerical property into a kind of categorical feature.

In Neptune ML, you can cause a numerical property to be encoded as a bucket-numerical feature, you must provide two things:

- A numerical range in the form, "range": [*a*, *b*], where *a* and *b* are integers.
- A bucket count, in the form "bucket_cnt": *c* , where *c* is the number of buckets, also an integer.

Neptune ML then calculates the size of each bucket as $(b - a) / c$, and encodes each numeric value as the number of whatever bucket it falls into. Any value less than *a* is considered to belong in the first bucket, and any value greater than *b* is considered to belong in the last bucket.

You can also, optionally, make numeric values fall into more than one bucket, by specifying a slide-window size, like this: "slide_window_size": *s* , where *s* is a number. Neptune ML then transforms each numeric value *v* of the property into a range from $v - s/2$ through $v + s/2$, and assigns the value *v* to every bucket that the range covers.

Finally, you can also optionally provide a way of filling in missing values for numerical features and bucket-numerical features. You do this using "imputer": "*imputation technique* ", where the imputation technique is one of "mean", "median", or "most-frequent". If you don't specify an imputer, a missing value can cause processing to halt.

Text Word2Vec features in Neptune ML

Neptune ML can convert a string property value consisting of a sequence of tokens into a `text_word2vec` feature. This encodes the tokens in the string as a dense vector using one of the [spaCy trained models](#) (Neptune ML currently only supports the English `en_core_web_lg` model).

Text TF-IDF features in Neptune ML

TF-IDF (term frequency – inverse document frequency) is a numerical value intended to measure how important a word is in a document set. It is calculated by dividing the number of times a word appears in a given property value by the total number of such property values that it appears in.

For example, if the word "kiss" appears twice in a given movie title (say, "kiss kiss bang bang"), and "kiss" appears in the title of 4 movies in all, then the TF-IDF value of "kiss" in the "kiss kiss bang bang" title would be $2 / 4$.

Neptune ML can encode sentences or other free-form text property values as `text_tfidf` features. This encoding converts the sequence of words in the text into a numeric vector using a TF-IDF vectorizer, then followed by a dimensionality-reduction operation.

The vector that is initially created has *d* dimensions, where *d* is the number of unique terms in all property values of that type. The dimensionality-reduction operation uses a random sparse projection to reduce that number to a maximum of 100. The vocabulary of a graph is then generated by merging all the `text_tfidf` features in it.

You can control the TF-IDF vectorizer in several ways:

- **max_features** – Using the `max_features` parameter, you can limit the number of terms in `text_tfidf` features to the most common ones. For example, if you set `max_features` to 100, only the top 100 most commonly used terms are included. The default value for `max_features` if you don't explicitly set it is 5,000.
- **min_df** – Using the `min_df` parameter, you can limit the number of terms in `text_tfidf` features to ones having at least a specified document frequency. For example, if you set `min_df` to 5, only terms that appear in at least 5 different property values are used. The default value for `min_df` if you don't explicitly set it is 2.
- **ngram_range** – The `ngram_range` parameter determines what combinations of words are treated as terms. For example, if you set `ngram_range` to [2, 4], the following 6 terms would be found in the "kiss kiss bang bang" title:

- *2-word terms*: "kiss kiss", "kiss bang", and "bang bang".
- *3-word terms*: "kiss kiss bang" and "kiss bang bang".
- *4-word terms*: "kiss kiss bang bang".

The default setting for `ngram_range` is `[1, 1]`.

Datetime features in Neptune ML

Neptune ML can convert parts of `datetime` property values into categorical features by encoding them as [one-hot arrays](#). Use the `datetime_parts` parameter to specify one or more of the following parts to encode: `["year", "month", "weekday", "hour"]`. If you don't set `datetime_parts`, by default all four parts are encoded.

For example, if the range of `datetime` values spans the years 2010 through 2012, the four parts of the `datetime` entry `2011-04-22 01:16:34` are as follows:

- **year** – `[0, 1, 0]`.

Since there are only 3 years in the span (2010, 2011, and 2012), the one-hot array has three entries, one for each year.

- **month** – `[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]`.

Here, the one-hot array has an entry for each month of the year.

- **weekday** – `[0, 0, 0, 0, 1, 0, 0]`.

The ISO 8601 standard states that Monday is the first day of the week, and since April 22, 2011 was a Friday, the corresponding one-hot weekday array is hot in the fifth position.

- **hour** – `[0, 1, 0]`.

The hour 1 AM is set in a 24-member one-hot array.

Day of the month, minute, and second are not encoded categorically.

If the total `datetime` range in question only includes dates within a single year, no `year` array is encoded.

You can specify an imputation strategy to fill in missing `datetime` values, using the `imputer` parameter and one of the strategies available for numerical features.

Auto feature encoding in Neptune ML

Instead of manually specifying the feature encoding methods to use for the properties in your graph, you can set `auto` as a feature encoding method. Neptune ML then attempts to infer the best feature encoding for each property based on its underlying data type.

Here are some of the heuristics that Neptune ML uses in selecting the appropriate feature encodings:

- If the property has only numeric values and can be cast into numeric data types, then Neptune ML generally encodes it as a numeric value. However, if the number of unique values for the property is less than 10% of the total number of values and the cardinality of those unique values is less than 100, then Neptune ML uses a categorical encoding.
- If the property values can be cast to a `datetime` type, then Neptune ML encodes them as a `datetime` feature.
- If the property values can be coerced to booleans (1/0 or True/False), then Neptune ML uses category encoding.

- If the property is a string with more than 10% of its values unique, and the average number of tokens per value is greater than or equal to 3, the Neptune ML infers the property type to be text and uses the `text_word2vec` encoding.
- If the property is a string not classified as a text feature then Neptune ML presumes it to be a categorical feature and uses category encoding.
- If each node has its own unique value for a property that is inferred to be a category feature, Neptune ML drops the property from the training graph because it is probably an ID that would not be informative for learning.
- If the property is known to contain valid Neptune separators such as semicolons (";"), then Neptune ML can only treat the property as `MultiNumerical` or `MultiCategorical`.
 - Neptune ML first tries to encode the values as numeric features. if this succeeds, Neptune ML uses numerical encoding to create numeric vector features.
 - Otherwise, Neptune ML encodes the values as multi-categorical.
- If Neptune ML cannot infer the data type of a property's values, Neptune ML drops the property from the training graph.

Editing a training data configuration file

The Neptune export process exports Neptune ML data from a Neptune DB cluster into an S3 bucket. It exports nodes and edges separately into a `nodes/` and an `edges/` folder. It also creates a JSON training data configuration file, named `training-data-configuration.json` by default. This file contains information about the schema of the graph, the types of its features, feature transformation and normalization operations, and the target feature for a classification or regression task.

There might be cases when you want to modify the configuration file directly. One such case is when you want to change the way features are processed or how the graph is constructed, without needing to rerun the export every time you want to modify the specification for the machine learning task you're solving.

To edit the training data configuration file

1. Download the file to your local machine.

Unless you specified one or more named jobs in the `additionalParams/neptune_ml` parameter passed to the export process, the file will have the default name, which is `training-data-configuration.json`. You can use an AWS CLI command like this to download the file:

```
aws s3 cp \  
  s3://(your Amazon S3 bucket)/(path to your export folder)/training-data-  
  configuration.json \  
  ./
```

2. Edit the file using a text editor.

3. Upload the modified file.

Upload the modified file back to the same location in Amazon S3 from which you downloaded it, using use an AWS CLI command like this:

```
aws s3 cp \  
  training-data-configuration.json \  
  s3://(your Amazon S3 bucket)/(path to your export folder)/training-data-  
  configuration.json
```

Example of a JSON training data configuration file

Here is a sample training data configuration file that describes a graph for a node-classification task:

```
{  
  "version" : "v2.0",  
  "query_engine" : "gremlin",  
  "graph" : [  
    {  
      "edges" : [  
        {  
          "file_name" : "edges/(movie)-included_in-(genre).csv",  
          "separator" : ",",  
          "source" : ["~from", "movie"],  
          "relation" : ["", "included_in"],  
          "dest" : [ "-to", "genre" ]  
        },  
        {  
          "file_name" : "edges/(user)-rated-(movie).csv",  
          "separator" : ",",  
          "source" : ["~from", "movie"],  
          "relation" : ["rating", "prefixname"], # [prefixname#value]        }  
      ]  
    }  
  ]  
}
```

```
    "dest" : ["-to", "genre"],
    "features" : [
      {
        "feature" : ["rating", "rating", "numerical"],
        "norm" : "min-max"
      }
    ]
  }
],
"nodes" : [
  {
    "file_name" : "nodes/genre.csv",
    "separator" : ",",
    "node" : ["-id", "genre"],
    "features" : [
      {
        "feature": ["name", "genre", "category"],
        "separator": ";"
      }
    ]
  },
  {
    "file_name" : "nodes/movie.csv",
    "separator" : ",",
    "node" : ["-id", "movie"],
    "features" : [
      {
        "feature": ["title", "title", "word2vec"],
        "language": ["en_core_web_lg"]
      }
    ]
  },
  {
    "file_name" : "nodes/user.csv",
    "separator" : ",",
    "node" : ["-id", "user"],
    "features" : [
      {
        "feature": ["age", "age", "numerical"],
        "norm" : "min-max",
        "imputation": "median",
      },
      {
        "feature": ["occupation", "occupation", "category"],
      }
    ],
    "labels" : [
      {
        "label": ["gender", "classification"],
        "split_rate" : [0.8, 0.2, 0.0]
      }
    ]
  }
],
"warnings" : [ ]
}
```

The structure of JSON training data configuration files

The training configuration file refers to CSV files saved by the export process in the nodes/ and edges/ folders.

Each file under `nodes/` stores information about nodes that have the same property-graph node label. Each column in a node file stores either the node ID or the node property. The first line of the file contains a header that specifies the `~id` or property name for each column.

Each file under `edges/` stores information about nodes that have the same property-graph edge label. Each column in a node file stores either the source node ID, the destination node ID, or the edge property. The first line of the file contains a header specifying the `~from`, `~to`, or property name for each column.

The training data configuration file has three top-level elements:

```
{
  "version" : "v2.0",
  "query_engine" : "gremlin",
  "graph" : [ ... ]
}
```

- `version` – (String) The version of configuration file being used.
- `query_engine` – (String) The query language used for exporting the graph data. Currently, only "gremlin" is valid.
- `graph` – (JSON array) lists one or more configuration objects that contain model parameters for each of the nodes and edges that will be used.

The configuration objects in the `graph` array have the structure described in the next section.

Contents of a configuration object listed in the `graph` array

A configuration object in the `graph` array can contain three top-level nodes:

```
{
  "edges" : [ ... ],
  "nodes" : [ ... ],
  "warnings" : [ ... ],
}
```

- `edges` – (array of JSON objects) Each JSON object specifies a set of parameters to define how an edge in the graph will be treated during the model processing and training. This is only used with the Gremlin engine.
- `nodes` – (array of JSON objects) Each JSON object specifies a set of parameters to define how a node in the graph will be treated during the model processing and training. This is only used with the Gremlin engine.
- `warnings` – (array of JSON objects) Each object contains a warning generated during the data export process.

Contents of an edge configuration object listed in an `edges` array

An edge configuration object listed in an `edges` array can contain the following top-level fields:

```
{
  "file_name" : "(path to a CSV file)",
  "separator" : "(separator character)",
  "source" : ["(column label for starting node ID)", "(starting node type)"],
  "relation" : ["(column label for the relationship name)", "(the prefix name for the relationship name)"],
  "dest" : ["(column label for ending node ID)", "(ending node type)"],
  "features" : [(array of feature objects)]
  "labels" : [(array of label objects)]
}
```

```
}
```

- **file_name** – A string specifying the path to a CSV file that stores information about edges having the same property-graph label.

The first line of that file contains a header line of column labels.

The first two column labels are `~from` and `~to`. The first column (the `~from` column) stores the ID of the edge's starting node, and the second (the `~to` column) stores the ID of the edge's ending node.

The remaining column labels in the header line specify, for each remaining column, the name of the edge property whose values have been exported into that column.

- **separator** – A string containing the delimiter that separates columns in that CSV file.
- **source** – A JSON array containing two strings that specify the starting node of the edge. The first string contains the header name of the column that the starting node ID is stored in. The second string specifies the node type.
- **relation** – A JSON array containing two strings that specify the edge's relation type. The first string contains the header name of the column that the relation name (`relname`) is stored in. The second string contains the prefix for the relation name (`prefixname`).

The full relation type consists of the two strings combined, with a hyphen character between them, like this: `prefixname-relname`.

If the first string is empty, all edges have the same relation type, namely the `prefixname` string.

- **dest** – A JSON array containing two strings that specify the ending node of the edge. The first string contains the header name of the column that the node ID is stored in. The second string specifies the node type.
- **features** – A JSON array of property-value feature objects. Each property-value feature object contains the following fields:
 - **feature** – A JSON array of three strings. The first string contains the header name of the column that contains the property value. The second string contains the feature name. The third string contains the feature type.
 - **norm** – (*Optional*) Specifies a normalization method to apply to the property values.
- **labels** – A JSON array of objects. Each of the objects defines a target feature of the edges, and specifies the proportions of the edges that the training and validation stages should take. Each object contains the following fields:
 - **label** – A JSON array of two strings. The first string contains the header name of the column that contains the target feature property value. The second string specifies one of the following target task types:
 - `"classification"` – An edge classification task. The property values provided in the column identified by the first string in the `label` array are treated as categorical values. For an edge classification task, the first string in the `label` array can't be empty.
 - `"regression"` – An edge regression task. The property values provided in the column identified by the first string in the `label` array are treated as numerical values. For an edge regression task, the first string in the `label` array can't be empty.
 - `"link_prediction"` – A link prediction task. No property values are required. For a link prediction task, the first string in the `label` array is ignored.

Contents of a node configuration object listed in a nodes array

A node configuration object listed in a `nodes` array can contain the following fields:

```
{
```

```
"file_name" : "(path to a CSV file)",  
"separator" : "(separator character)",  
"node"      : ["(column label for the node ID)", "(node type)"],  
"features"  : [(feature array)],  
"labels"    : [(label array)],  
}
```

- **file_name** – A string specifying the path to a CSV file that stores information about nodes having the same property-graph label.

The first line of that file contains a header line of column labels.

The first column label is `~id`, and the first column (the `~id` column) stores the node ID.

The remaining column labels in the header line specify, for each remaining column, the name of the node property whose values have been exported into that column.

- **separator** – A string containing the delimiter that separates columns in that CSV file.
- **node** – A JSON array containing two strings. The first string contains the header name of the column that stores node IDs. The second string specifies the node type in the graph, which corresponds to a property-graph label of the node.
- **features** – A JSON array of node feature objects. See [Contents of a feature object listed in a features array for a node or edge \(p. 574\)](#).
- **labels** – A JSON array of node label objects. See [Contents of a node label object listed in a node labels array \(p. 575\)](#).

Contents of a feature object listed in a features array for a node or edge

A node feature object listed in a node `features` array can contain the following top-level fields:

- **feature** – A JSON array of three strings. The first string contains the header name of the column that contains the property value for the feature. The second string contains the feature name.

The third string contains the feature type. Valid feature types are listed in [Possible values of the type field for features \(p. 550\)](#).

- **norm** – This field is required for numerical features. It specifies a normalization method to use on numeric values. Valid values are "none", "min-max", and "standard". See [The norm field \(p. 551\)](#) for details.
- **language** – This field is required for `text_word2vec` features. It specifies the name of the language model used to encode the property value string. Currently, Neptune ML only supports "en_core_web_lg" (for English).

See [Text Word2Vec features in Neptune ML \(p. 567\)](#).

- **separator** – This field is used optionally with `category`, `numerical` and `auto` features. It specifies a character that can be used to split a property value into multiple categorical values or numerical values.

See [The separator field \(p. 551\)](#).

- **range** – This field is required for `bucket_numerical` features. It specifies the range of numerical values that are to be divided into buckets.

See [The range field \(p. 552\)](#).

- **bucket_cnt** – This field is required for `bucket_numerical` features. It specifies the number of buckets that the numerical range defined by the `range` parameter should be divided into.

See [Bucket-numerical features in Neptune ML \(p. 566\)](#).

- **slide_window_size** – This field is used optionally with `bucket_numerical` features to assign values to more than one bucket.

See [The `slide_window_size` field \(p. 552\)](#).

- **imputer** – This field is used optionally with `numerical`, `bucket_numerical`, and `datetime` features to provide an imputation technique for filling in missing values. The supported imputation techniques are "mean", "median", and "most_frequent".

See [The `imputer` field \(p. 552\)](#).

- **max_features** – This field is used optionally by `text_tfidf` features to specify the maximum number of terms to encode.

See [The `max_features` field \(p. 553\)](#).

- **min_df** – This field is used optionally by `text_tfidf` features to specify the minimum document frequency of terms to encode

See [The `min_df` field \(p. 553\)](#).

- **ngram_range** – This field is used optionally by `text_tfidf` features to specify a range of numbers of words or tokens to considered as potential individual terms to encode

See [The `ngram_range` field \(p. 553\)](#).

- **datetime_parts** – This field is used optionally by `datetime` features to specify which parts of the datetime value to encode categorically.

See [The `datetime_parts` field \(p. 553\)](#).

Contents of a node label object listed in a node `labels` array

A label object listed in a node `labels` array defines a node target feature and specifies the proportions of nodes that the training, validation, and test stages will use. Each object can contain the following fields:

```
{
  "label"      : ["(column label for the target feature property value)", "(task
type)"],
  "split_rate" : [(training proportion), (validation proportion), (test proportion)],
  "separator"  : "(separator character for node-classification category values)",
}
```

- **label** – A JSON array containing two strings. The first string contains the header name of the column that stores the property values for the feature. The second string specifies the target task type, which can be:
 - "classification" – A node classification task. The property values in the specified column are used to create a categorical feature.
 - "regression" – A node regression task. The property values in the specified column are used to create a numerical feature.
- **split_rate** – A JSON array containing three numbers between zero and one that add up to one and represent an estimate of the proportions of nodes that the training, validation, and test stages will use, respectively. See [split_rate \(p. 546\)](#).
- **separator** – A string containing the delimiter that separates categorical feature values for a classification task.

Note

If no label object is provided for both edges and nodes, the task is automatically assumed to be link prediction, and edges are randomly split into 90% for training and 10% for validation.

Training a model using Neptune ML

After you have processed the data that you exported from Neptune for model training, you can start a model-training job using a `curl` (or `awscurl`) command like the following:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltraining
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
}'
```

The details of how to use this command are explained in [The modeltraining command \(p. 624\)](#), along with information about how to get the status of a running job, how to stop a running job, and how to list all running jobs.

You can also supply a `previousModelTrainingJobId` to use information from a completed Neptune ML model training job to accelerate the hyperparameter search in a new training job. This is useful during [model retraining on new graph data \(p. 538\)](#), as well as [incremental training on the same graph data \(p. 537\)](#). Use a command like this one:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltraining
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
  "previousModelTrainingJobId" : "(the model-training job-id of a completed job)"
}'
```

You can train your own model implementation on the Neptune ML training infrastructure by supplying a `customModelTrainingParameters` object, like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltraining
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
  "modelName": "custom",
  "customModelTrainingParameters" : {
    "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
module)",
    "trainingEntryPointScript": "(your training script entry-point name in the Python
module)",
    "transformEntryPointScript": "(your transform script entry-point name in the
Python module)"
  }
}'
```

See [The `modeltraining` command \(p. 624\)](#) for more information, such as about how to get the status of a running job, how to stop a running job, and how to list all running jobs. See [Custom models in Neptune ML \(p. 595\)](#) for information about how to implement and use a custom model.

Topics

- [Models and model training in Amazon Neptune ML \(p. 579\)](#)
- [Customizing model hyperparameter configurations in Neptune ML \(p. 581\)](#)
- [Model training best practices \(p. 589\)](#)

Models and model training in Amazon Neptune ML

Neptune ML uses Graph Neural Networks (GNN) to create models for the various machine-learning tasks. Graph neural networks have been shown to obtain state-of-the-art results for graph machine learning tasks and are excellent at extracting informative patterns from graph structured data.

Graph neural networks (GNNs) in Neptune ML

Graph Neural Networks (GNNs) belong to a family of neural networks that compute node representations by taking into account the structure and features of nearby nodes. GNNs complement other traditional machine learning and neural network methods that are not well-suited for graph data.

GNNs are used to solve machine-learning tasks such as node classification and regression (predicting properties of nodes) and edge classification and regression (predicting properties of edges) or link prediction (predicting whether two nodes in the graph should be connected or not).

In general, using a GNN for a machine learning task involves two stages:

- An encoding stage, where the GNN computes a d -dimensional vector for each node in the graph. These vectors are also called *representations* or *embeddings*.
- A decoding stage, which makes predictions based on the encoded representations.

For node classification and regression, the node representations are used directly for the classification and regression tasks. For edge classification and regression, the node representations of the incident nodes on an edge are used as input for the classification or regression. For link prediction, an edge likelihood score is computed by using a pair of node representations and an edge type representation.

The [Deep Graph Library \(DGL\)](#) facilitates the efficient definition and training of GNNs for these tasks.

Different GNN models are unified under the formulation of message passing. In this view, the representation for a node in a graph is calculated using the node's neighbors' representations (the messages), together with the node's initial representation. In NeptuneML, the initial representation of a node is derived from the features extracted from its node properties, or is learnable and depends on the identity of the node.

Neptune ML also provides the option to concatenate node features and learnable node representations to serve as the original node representation.

For the various tasks in Neptune ML involving graphs with node properties, we use the [Relational Graph Convolutional Network \(R-GCN\)](#) to perform the encoding stage. R-GCN is a GNN architecture that is well-suited for graphs that have multiple node and edge types (these are known as heterogeneous graphs).

The R-GCN network consists of a fixed number of layers, stacked one after the other. Each layer of the R-GCN uses its learnable model parameters to aggregate information from the immediate, 1-hop neighborhood of a node. Since subsequent layers use the previous layer's output representations as input, the radius of the graph neighborhood that influences a node's final embedding depends on the number of layers (`num-layer`), of the R-GCN network.

For example, this means that a 2-layer network uses information from nodes that are 2 hops away.

To learn more about GNNs, see [A Comprehensive Survey on Graph Neural Networks](#). For more information about the Deep Graph Library (DGL), visit the [DGL webpage](#). For a hands-on tutorial about using DGL with GNNs, see [Learning graph neural networks with Deep Graph Library](#).

Training Graph Neural Networks

In machine learning, the process of getting a model to learn how to make good predictions for a task is called model training. This is usually performed by specifying a particular objective to optimize, as well as an algorithm to use to perform this optimization.

This process is employed in training a GNN to learn good representations for the downstream task as well. We create an objective function for that task that is minimized during model training. For example, for node classification, we use [CrossEntropyLoss](#) as the objective, which penalizes misclassifications, and for node regression we minimize [MeanSquareError](#).

The objective is usually a loss function that takes the model predictions for a particular data point and compares them to the ground-truth value for that data point. It returns the loss value, which shows how far off the model's predictions are. The goal of the training process is to minimize the loss and ensure that model predictions are close to the ground-truth.

The optimization algorithm used in deep learning for the training process is usually a variant of gradient descent. In Neptune ML, we use *Adam*, which stands for [adaptive gradient descent algorithm](#).

While the model training process tries to ensure that the learned model parameters are close to the minima of the objective function, the overall performance of a model also depends on the model's *hyperparameters*, which are model settings that aren't learned by the training algorithm. For example, the dimensionality of the learned node representation, `num-hidden`, is a hyperparameter that affects model performance. Therefore, it is common in machine learning to perform hyperparameter optimization (HPO) to choose the suitable hyperparameters.

Neptune ML uses a SageMaker hyperparameter tuning job to launch multiple instances of model training with different hyperparameter configurations to try to find the best model for a range of hyperparameters settings. See [Customizing model hyperparameter configurations in Neptune ML](#) (p. 581).

Knowledge graph embedding models in Neptune ML

Knowledge graphs (KGs) are graphs that encode information about different entities (nodes) and their relations (edges). In Neptune ML, knowledge graph embedding models are applied by default for performing link prediction when the graph does not contain node properties, only relations with other nodes. Although, R-GCN models with learnable embeddings can also be used for these graphs by specifying the model type as `"rgcn"`, knowledge graph embedding models are simpler and are designed to be effective for learning representations for large scale knowledge graphs.

Knowledge graph embedding models are used in a link prediction task to predict the nodes or relations that complete a triple (h, r, t) where h is the source node, r is the relation type and t is the destination node.

The knowledge graph embedding models implemented in Neptune ML are `distmult`, `transE`, and `rotatE`. To learn more about knowledge graph embedding models, see [DGL-KE](#).

Training custom models in Neptune ML

Neptune ML lets you define and implement custom models of your own, for particular scenarios. See [Custom models in Neptune ML](#) (p. 595) for information about how to implement a custom model and how to use Neptune ML infrastructure to train it.

Customizing model hyperparameter configurations in Neptune ML

When you start a Neptune ML model-training job, Neptune ML automatically uses the information inferred from the preceding [data-processing](#) (p. 564) job. It uses the information to generate hyperparameter configuration ranges that are used to create a [SageMaker hyperparameter tuning job](#) to train multiple models for your task. That way, you don't have to specify a long list of hyperparameter values for the models to be trained with. Instead, the model hyperparameter ranges and defaults are selected based on the task type, graph type, and the tuning-job settings.

However, you can also override the default hyperparameter configuration and provide custom hyperparameters by modifying a JSON configuration file that the data-processing job generates.

Using the Neptune ML [modelTraining API](#) (p. 624), you can control several high level hyperparameter tuning job settings like `maxHPONumberOfTrainingJobs`, `maxHPOParallelTrainingJobs`, and `trainingInstanceType`. For more fine-grained control over the model hyperparameters, you can customize the `model-HPO-configuration.json` file that the data-processing job generates. The file is saved in the Amazon S3 location that you specified for processing-job output.

You can download the file, edit it to override the default hyperparameter configurations, and upload it back to the same Amazon S3 location. Do not change the name of the file, and be careful to follow these instructions as you edit.

To download the file from Amazon S3:

```
aws s3 cp \  
  s3://(bucket name)/(path to output folder)/model-HPO-configuration.json \  
  ./
```

When you have finished editing, upload the file back to where it was:

```
aws s3 cp \  
  model-HPO-configuration.json \  
  s3://(bucket name)/(path to output folder)/model-HPO-configuration.json
```

Structure of the `model-HPO-configuration.json` file

The `model-HPO-configuration.json` file specifies the model to be trained, the machine learning `task_type` and the hyperparameters that should be varied or fixed for the various runs of model training.

The hyperparameters are categorized as belonging to various tiers that signify the precedence given to the hyperparameters when the hyperparameter tuning job is invoked:

- Tier-1 hyperparameters have the highest precedence. If you set `maxHPONumberOfTrainingJobs` to a value less than 10, only Tier-1 hyperparameters are tuned, and the rest take their default values.
- Tier-2 hyperparameters have lower precedence, so if you have more than 10 but less than 50 total training jobs for a tuning job, then both Tier-1 and Tier-2 hyperparameters are tuned.
- Tier 3 hyperparameters are tuned together with Tier-1 and Tier-2 only if you have more than 50 total training jobs.
- Finally, fixed hyperparameters are not tuned at all, and always take their default values.

Example of a `model-HPO-configuration.json` file

The following is a sample `model-HPO-configuration.json` file:

```
{
  "models": [
    {
      "model": "rgcn",
      "task_type": "node_class",
      "eval_metric": {
        "metric": "acc"
      },
      "eval_frequency": {
        "type": "evaluate_every_epoch",
        "value": 1
      },
      "1-tier-param": [
        {
          "param": "num-hidden",
          "range": [16, 128],
          "type": "int",
          "inc_strategy": "power2"
        },
        {
          "param": "num-epochs",
          "range": [3, 30],
          "inc_strategy": "linear",
          "inc_val": 1,
          "type": "int",
          "node_strategy": "perM"
        },
        {
          "param": "lr",
          "range": [0.001, 0.01],
          "type": "float",
          "inc_strategy": "log"
        }
      ],
      "2-tier-param": [
        {
          "param": "dropout",
          "range": [0.0, 0.5],
          "inc_strategy": "linear",
          "type": "float",
          "default": 0.3
        },
        {
          "param": "global-norm",
          "type": "bool",
          "default": true
        }
      ],
      "3-tier-param": [
        {
          "param": "batch-size",
          "range": [128, 4096],
          "inc_strategy": "power2",
          "type": "int",
          "default": 1024
        },
        {
          "param": "sparse-embedding",
          "type": "bool",
          "default": false
        },
        {
          "param": "concat-node-embed",
          "type": "bool",
          "default": true
        }
      ]
    }
  ]
}
```

```

    },
    {
      "param": "fanout",
      "type": "int",
      "options": [[10, 30],[15, 30], [15, 30]],
      "default": [10, 15, 15]
    },
    {
      "param": "num-layer",
      "range": [1, 3],
      "inc_strategy": "linear",
      "inc_val": 1,
      "type": "int",
      "default": 2
    },
    {
      "param": "num-bases",
      "range": [2, 8],
      "inc_strategy": "linear",
      "inc_val": 2,
      "type": "int",
      "default": 2
    }
  ],
  "fixed-param": [
    {
      "param": "layer-norm",
      "type": "bool",
      "default": false
    },
    {
      "param": "use-self-loop",
      "type": "bool",
      "default": true
    },
    {
      "param": "low-mem",
      "type": "bool",
      "default": true
    },
    {
      "param": "l2norm",
      "type": "float",
      "default": 0
    }
  ]
}
]
}

```

Elements of a model-HPO-configuration.json file

The file contains a JSON object with a single top-level array named `models` that contains a single model-configuration object. When customizing the file, make sure the `models` array only has one model-configuration object in it. If your file contains more than one model-configuration object, the tuning job will fail with a warning.

The model-configuration object contains the following top-level elements:

- **model** – (*String*) The model type to be trained (**do not modify**). Valid values are:
 - "rgcn" – This is the default for node classification and regression tasks, and for heterogeneous link prediction tasks.
 - "transe" – This is the default for KGE link prediction tasks.

- "distmult" – This is an alternative model type for KGE link prediction tasks.
- "rotate" – This is an alternative model type for KGE link prediction tasks.

As a rule, don't directly modify the `model` value, because different model types often have substantially different applicable hyperparameters, which can result in a parsing error after the training job has started.

To change the model type, use the `modelName` parameter in the [modelTraining API \(p. 624\)](#) rather than change it in the `model-HPO-configuration.json` file.

A way to change the model type and make fine-grain hyperparameter changes is to copy the default model configuration template for the model that you want to use and paste that into the `model-HPO-configuration.json` file. There is a folder named `hpo-configuration-templates` in the same Amazon S3 location as the `model-HPO-configuration.json` file if the inferred task type supports multiple models. This folder contains all the default hyperparameter configurations for the other models that are applicable to the task.

For example, if you want to change the model and hyperparameter configurations for a KGE link-prediction task from the default `transe` model to a `distmult` model, simply paste the contents of the `hpo-configuration-templates/distmult.json` file into the `model-HPO-configuration.json` file and then edit the hyperparameters as necessary.

Note

If you set the `modelName` parameter in the `modelTraining` API and also change the `model` and hyperparameter specification in the `model-HPO-configuration.json` file, and these are different, the `model` value in the `model-HPO-configuration.json` file takes precedence, and the `modelName` value is ignored.

- **task_type** – (*String*) The machine learning task type inferred by or passed directly to the data-processing job (**do not modify**). Valid values are:
 - "node_class"
 - "node_regression"
 - "link_prediction"

The data-processing job infers the task type by examining the exported dataset and the generated training-job configuration file for properties of the dataset.

This value should not be changed. If you want to train a different task, you need to [run a new data-processing job \(p. 564\)](#). If the `task_type` value is not what you were expecting, you should check the inputs to your data-processing job to make sure that they are correct. This includes parameters to the `modelTraining` API, as well as in the training-job configuration file generated by the data-export process.

- **eval_metric** – (*String*) The evaluation metric should be used for evaluating the model performance and for selecting the best-performing model across HPO runs. Valid values are:
 - "acc" – Standard classification accuracy. This is the default for single-label classification tasks, unless imbalanced labels are found during data processing, in which case the default is "F1".
 - "acc_topk" – The number of times the correct label is among the top `k` predictions. You can also set the value `k` by passing in `topk` as an extra key.
 - "F1" – The [F1 score](#).
 - "mse" – [Mean-squared error metric](#), for regression tasks.
 - "mrr" – [Mean reciprocal rank metric](#).
 - "precision" – The model precision, calculated as the ratio of true positives to predicted positives: $\text{precision} = \frac{\text{true-positives}}{\text{true-positives} + \text{false-positives}}$.
 - "recall" – The model recall, calculated as the ratio of true positives to actual positives: $\text{recall} = \frac{\text{true-positives}}{\text{true-positives} + \text{false-negatives}}$.
 - "roc_auc" – The area under the [ROC curve](#). This is the default for multi-label classification.

For example, to change the metric to F1, change the `eval_metric` value as follows:

```
" eval_metric": {  
  "metric": "F1",  
},
```

Or, to change the metric to a topk accuracy score, you would change `eval_metric` as follows:

```
"eval_metric": {  
  "metric": "acc_topk",  
  "topk": 2  
},
```

- **eval_frequency** – (*Object*) Specifies how often during training the performance of the model on the validation set should be checked. Based on the validation performance, early stopping can then be initiated and the best model can be saved.

The `eval_frequency` object contains two elements, namely "type" and "value". For example:

```
"eval_frequency": {  
  "type": "evaluate_every_pct",  
  "value": 0.1  
},
```

Valid type values are:

- **evaluate_every_pct** – Specifies the percentage of training to be completed for each evaluation.

For `evaluate_every_pct`, the "value" field contains a floating-point number between zero and one which expresses that percentage.

- **evaluate_every_batch** – Specifies the number of training batches to be completed for each evaluation.

For `evaluate_every_batch`, the "value" field contains an integer which expresses that batch count.

- **evaluate_every_epoch** – Specifies the number of epochs per evaluation, where a new epoch starts at midnight.

For `evaluate_every_epoch`, the "value" field contains an integer which expresses that epoch count.

The default setting for `eval_frequency` is:

```
"eval_frequency": {  
  "type": "evaluate_every_epoch",  
  "value": 1  
},
```

- **1-tier-param** – (*Required*) An array of Tier-1 hyperparameters.

If you don't want to tune any hyperparameters, you can set this to an empty array. This does not affect the total number of training jobs launched by the SageMaker hyperparameter tuning job. It just means that all training jobs, if there is more than 1 but less than 10, will run with the same set of hyperparameters.

On the other hand, if you want to treat all your tunable hyperparameters with equal significance then you can put all the hyperparameters in this array.

- **2-tier-param** – *(Required)* An array of Tier-2 hyperparameters.

These parameters are only tuned if `maxHPONumberOfTrainingJobs` has a value greater than 10. Otherwise, they are fixed to the default values.

If you have a training budget of at most 10 training jobs or don't want Tier-2 hyperparameters for any other reason, but you want to tune all tunable hyperparameters, you can set this to an empty array.

- **3-tier-param** – *(Required)* An array of Tier-3 hyperparameters.

These parameters are only tuned if `maxHPONumberOfTrainingJobs` has a value greater than 50. Otherwise, they are fixed to the default values.

If you don't want Tier-3 hyperparameters, you can set this to an empty array.

- **fixed-param** – *(Required)* An array of fixed hyperparameters that take only their default values and do not vary in different training jobs.

If you want to vary all hyperparameters, you can set this to an empty array and either set the value for `maxHPONumberOfTrainingJobs` large enough to vary all tiers or make all hyperparameters Tier-1.

The JSON object that represents each hyperparameter in `1-tier-param`, `2-tier-param`, `3-tier-param`, and `fixed-param` contains the following elements:

- **param** – *(String)* The name of the hyperparameter (**do not change**).

See the [list of valid hyperparameter names in Neptune ML \(p. 587\)](#).

- **type** – *(String)* The hyperparameter type (**do not change**).

Valid types are: `bool`, `int`, and `float`.

- **default** – *(String)* The default value for the hyperparameter.

You can set a new default value.

Tunable hyperparameters can also contain the following elements:

- **range** – *(Array)* The range for a continuous tunable hyperparameter.

This should be an array with two values, namely the minimum and maximum of the range (`[min, max]`).

- **options** – *(Array)* The options for a categorical tunable hyperparameter.

This array should contain all the options to consider:

```
"options" : [value1, value2, ... valuen]
```

- **inc_strategy** – *(String)* The type of incremental change for continuous tunable hyperparameter ranges (**do not change**).

Valid values are `log`, `linear`, and `power2`. This applies only when the `range` key is set.

Modifying this may result in not using the full range of your hyperparameter for tuning.

- **inc_val** – *(Float)* The amount by which successive increments differ for continuous tunable hyperparameters (**do not change**).

This applies only when the `range` key is set.

Modifying this may result in not using the full range of your hyperparameter for tuning.

- **node_strategy** – (*String*) Indicates that the effective range for this hyperparameter should change based on the number of nodes in the graph (**do not change**).

Valid values are "perM" (per million), "per10M" (per 10 million), and "per100M" (per 100 million).

Rather than change this value, change the `range` instead.

- **edge_strategy** – (*String*) Indicates that the effective range for this hyperparameter should change based on the number of edges in the graph (**do not change**).

Valid values are "perM" (per million), "per10M" (per 10 million), and "per100M" (per 100 million).

Rather than change this value, change the `range` instead.

List of all the hyperparameters in Neptune ML

The following list contains all the hyperparameters that can be set anywhere in Neptune ML, for any model type and task. Because they are not all applicable to every model type, it is important that you only set hyperparameters in the `model-HPO-configuration.json` file that appear in the template for the model you're using.

- **batch-size** – The size of the batch of target nodes using in one forward pass. *Type: int.*

Setting this to a much larger value can cause memory issues for training on GPU instances.

- **call_to_consider_early_stop** – Specifies the number of the earliest evaluation call at which early stopping can kick in. This is to allow a burn-in period if necessary. *Type: int. Default: 0.*

See [Early stopping of the model training process in Neptune ML \(p. 590\)](#).

- **concat-node-embed** – Indicates whether to get the initial representation of a node by concatenating its features with its initial embeddings. *Type: bool.*

- **dropout** – The dropout probability applied to dropout layers. *Type: float.*

- **enable_early_stop** – Toggles whether or not to use the early stopping feature. *Type: bool. Default: true. See .*

- **fanout** – The number of neighbors to sample for a target node during neighbor sampling. *Type: int.*

This value is tightly coupled with `num-layers` and should always be in the same hyperparameter tier. This is because you can specify a fanout for each potential GNN layer.

Because this hyperparameter can cause model performance to vary widely, it should be fixed or set as a Tier-2 or Tier-3 hyperparameter. Setting it to a large value can cause memory issues for training on GPU instance.

- **gamma** – The margin value in the score function. *Type: float.*

This applies to `KGE` link-prediction models only.

- **global-norm** – Indicates whether to use global normalization rather than relation-based normalization. *Type: bool.*

- **l2norm** – Indicates whether to use an `L2` normalization penalty rather than weight decay. *Type: bool.*

- **layer-norm** – Indicates whether to use layer normalization for `rgcn` models. *Type: bool.*

- **low-mem** – Indicates whether to use a low-memory implementation of the relation message passing function at the expense of speed. *Type: bool.*

- **lr** – The learning rate. *Type: float.*

This should be set as a Tier-1 hyperparameter.

- **neg-share** – In link prediction, indicates whether positive sampled edges can share negative edge samples. *Type: bool.*
- **num-bases** – The number of bases for basis decomposition in a `rgcn` model. *Type: int.*
- **num-epochs** – The number of epochs of training to run. *Type: int.*

An epoch is a complete training pass through the graph.

- **num-hidden** – The hidden layer size or number of units. *Type: int.*

This also sets the initial embedding size for featureless nodes.

Setting this to a much larger value without reducing `batch-size` can cause out-of-memory issues for training on GPU instance.

- **num-layer** – The number of GNN layers in the model. *Type: int.*

This value is tightly coupled with the fanout parameter and should come after fanout is set in the same hyperparameter tier.

Because this can cause model performance to vary widely, it should be fixed or set as a Tier-2 or Tier-3 hyperparameter.

- **num-negs** – In link prediction, the number of negative samples per positive sample. *Type: int.*
- **regularization-coef** – In link prediction, the coefficient of regularization loss. *Type: float.*
- **rel-part** – Indicates whether to use relation partition for `KGE` link prediction. *Type: bool.*
- **sparse-lr** – The learning rate for sparse embeddings. The parameters of the sparse embedding layer are trained using a separate optimizer which can have a separate learning rate. *Type: float.*
- **use-self-loop** – Indicates whether to include self loops in training a `rgcn` model. *Type: bool.*
- **window-for-early-stop** – Controls the number of latest validation scores to average to decide on an early stop. The default is 3. *type=int. See also [Early stopping of the model training process in Neptune ML \(p. 590\)](#). *Type: int. Default: 3.**

See .

Customizing hyperparameters in Neptune ML

When you are editing the `model-HPO-configuration.json` file, the following are the most common kinds of changes to make:

- Edit the minimum and/or maximum values of `range` hyperparameters.
- Set a hyperparameter to a fixed value by moving it to the `fixed-param` section and setting its default value to the fixed value you want it to take.
- Change the priority of a hyperparameter by placing it in a particular tier, editing its range, and making sure that its default value is set appropriately.

Model training best practices

There are things you can do to improve the performance of Neptune ML models.

Choose the right node property

Not all the properties in your graph may be meaningful or relevant to your machine learning tasks. Any irrelevant properties should be excluded during data export.

Here are some best practices:

- Use domain experts to help evaluate the importance of features and the feasibility of using them for predictions.
- Remove the features that you determine are redundant or irrelevant to reduce noise in the data and unimportant correlations.
- Iterate as you build your model. Adjust the features, feature combinations, and tuning objectives as you go along.

[Feature Processing](#) in the Amazon Machine Learning Developer Guide provides additional guidelines for feature processing that are relevant to Neptune ML.

Handle outlier data points

An outlier is a data point that is significantly different from the remaining data. Data outliers can spoil or mislead the training process, resulting in longer training time or less accurate models. Unless they are truly important, you should eliminate outliers before exporting the data.

Remove duplicate nodes and edges

Graphs stored in Neptune may have duplicate nodes or edges. These redundant elements will introduce noise for ML model training. Eliminate duplicate nodes or edges before exporting the data.

Tune the graph structure

When the graph is exported, you can change the way features are processed and how the graph is constructed, to improve the model performance.

Here are some best practices:

- When an edge property has the meaning of categories of edges, it is worth turning it into edge types in some cases.
- The default normalization policy used for a numerical property is `min-max`, but in some cases other normalization policies work better. You can preprocess the property and change the normalization policy as explained in [Elements of a model-HPO-configuration.json file \(p. 583\)](#).
- The export process automatically generates feature types based on property types. For example, it treats `String` properties as categorical features and `Float` and `Int` properties as numerical features. If you need to, you can modify the feature type after export (see [Elements of a model-HPO-configuration.json file \(p. 583\)](#)).

Tune the hyperparameter ranges and defaults

The data-processing operation infers hyperparameter configuration ranges from the graph. If the generated model hyperparameter ranges and defaults don't work well for your graph data, you can edit the HPO configuration file to create your own hyperparameter tuning strategy.

Here are some best practices:

- When the graph goes large, the default hidden dimension size may not be large enough to contain all the information. You can change the `num-hidden` hyperparameter to control the hidden dimension size.
- For knowledge graph embedding (KGE) models, you may want to change the specific model being used according to your graph structure and budget.

`TransE` models have difficulty in dealing with one-to-many (1-N), many-to-one (N-1), and many-to-many (N-N) relations. `DistMult` models have difficulty in dealing with symmetric relations. `RotatE` is good at modeling all kinds of relations but is more expensive than `TransE` and `DistMult` during training.

- In some cases, when both node identification and node feature information are important, you should use `concat-node-embed` to tell the Neptune ML model to get the initial representation of a node by concatenating its features with its initial embeddings.
- When you are getting reasonably good performance over some hyperparameters, you can adjust the hyperparameter search space according to those results.

Early stopping of the model training process in Neptune ML

Early stopping can significantly reduce the model-training run time and associated costs without degrading model performance. It also prevents the model from overfitting on the training data.

Early stopping depends on regular measurements of validation-set performance. Initially, performance improves as training proceeds, but when the model starts overfitting, it starts to decline again. The early stopping feature identifies the point at which the model starts overfitting and halts model training at that point.

Neptune ML monitors the validation metric calls and compares the most recent validation metric to the average of validation metrics over the last `n` evaluations, where `n` is a number set using the `window-for-early-stop` parameter. As soon as the validation metric is worse than that average, Neptune ML stops the model training and saves the best model so far.

You can control early stopping using the following parameters:

- **`call_to_consider_early_stop`** – Specifies the evaluation call at which to start considering an early stop. The default value is zero.

For example, setting this parameter to 4 causes Neptune ML to skip over the first 3 evaluation calls for early stop purposes, and only to consider an early stop at the fourth call. Ignoring a few initial evaluation calls can avoid an early stop triggered by irrelevant validation-metric oscillations at the start of training.

- **`window-for-early-stop`** – The value of this parameter is an integer that specifies the number of recent validation scores to average when deciding on an early stop. The default value is 3.
- **`enable_early_stop`** – Use this Boolean parameter to turn off the early stop feature. By default, its value is `true`.

Early stopping of the HPO process in Neptune ML

The early stop feature in Neptune ML also stops training jobs that are not performing well compared to other training jobs, using the SageMaker HPO warm-start feature. This too can reduce costs and improve the quality of HPO.

See [Run a warm start hyperparameter tuning job](#) for a description of how this works.

Warm start provides the ability to pass information learned from previous training jobs to subsequent training jobs and provides two distinct benefits:

- First, the results of previous training jobs are used to select good combinations of hyperparameters to search over in the new tuning job.
- Second, it allows early stopping to access more model runs, which reduces tuning time.

This feature is enabled automatically in Neptune ML, and allows you strike a balance between model training time and performance. If you are satisfied with the performance of the current model, you can use that model. Otherwise, you run more HPOs that are warm-started with the results of previous runs so as to discover a better model.

Get professional support services

AWS offers professional support services to help you with problems in your machine learning on Neptune projects. If you get stuck, reach out to [AWS support](#).

Use a trained model to generate new model artifacts

Using the Neptune ML model transform command, you can compute model artifacts like node embeddings on processed graph data using pre-trained model parameters.

Model transform for incremental inference

In the [incremental model inference workflow \(p. 537\)](#), after you have processed the updated graph data that you exported from Neptune you can start a model transform job using a curl (or awscli) command like the following:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltransform
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "mlModelTrainingJobId": "(the ML model training job-id)",
  "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
}'
```

You can then pass the ID of this job to the create-endpoints API call to create a new endpoint or update an existing one with the new model artifacts generated by this job. This allows the new or updated endpoint to provide model predictions for the updated graph data.

Model transform for any training job

You can also supply a `trainingJobName` parameter to generate model artifacts for any of the SageMaker training jobs launched during Neptune ML model training. Since a Neptune ML model training job can potentially launch many SageMaker training jobs, this gives you the flexibility to create an inference endpoint based on any of those SageMaker training jobs.

For example:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltransform
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "trainingJobName" : "(name a completed SageMaker training job)",
  "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
}'
```

If the original training job was for a user-provided custom model, you must include a `customModelTransformParameters` object when invoking a model transform. See [Custom models in Neptune ML \(p. 595\)](#) for information about how to implement and use a custom model.

Note

The `modeltransform` command always runs the model transform on the best SageMaker training job for that training.

See [The modeltransform command \(p. 629\)](#) for more information about model transform jobs.

Artifacts produced by model training in Neptune ML

After model training, Neptune ML uses the best trained model parameters to generate model artifacts that are necessary for launching the inference endpoint and providing model predictions. These artifacts are packaged by the training job and stored in the Amazon S3 output location of the best SageMaker training job.

The following sections describe what is included in the model artifacts for the various tasks, and how the `modeltransform` command uses a pre-existing trained model to generate artifacts even on new graph data.

Artifacts generated for different tasks

The content of the model artifacts generated by the training process depends on the target machine learning task:

- **Node classification and regression** – For node property prediction, the artifacts include model parameters, node embeddings from the [GNN encoder \(p. 579\)](#), model predictions, and some configuration files for the inference endpoint. Model predictions are pre-computed for node classification and node regression to reduce query latency.
- **Edge classification and regression** – For edge property prediction, the artifacts also include model parameters and node embeddings. The parameters of the model decoder are especially important for inference because we compute the edge classification or edge regression predictions by applying the model decoder to the saved embeddings of the source and destination vertex of an edge.
- **Link prediction** – For link prediction, in addition to the artifacts generated for edge property prediction, the DGL graph is also included as an artifact because link prediction requires the training graph to perform predictions. The objective of link prediction is to predict the destination vertices that are likely to combine with a source vertex to form an edge of a particular type in the graph. In order to do this, the node embedding of the source vertex and a learned representation for the edge type are combined with the node embeddings of all possible destination vertices to produce an edge likelihood score for each of the destination vertices. The scores are then sorted to rank the potential destination vertices and return the top candidates.

Generating new model artifacts

The model artifacts generated after model training in Neptune ML are directly tied to the training graph. This means that the inference endpoint is able to provide predictions only for entities that were in the original training graph.

In order to get model predictions for new entities that have been added to the graph, new model artifacts need to be recomputed for the new graph data. This is accomplished using the `modeltransform` command.

Since model training implicitly performs a model transform at the end of the training process, model artifacts are always recomputed on the training graph data by a training job. However, the `modeltransform` command can also compute model artifacts on graph data that was not used for training a model. In order to do this, the new graph data must be processed using the same feature encodings as the original graph data and must adhere to the same graph schema.

You can accomplish this by first creating a new data processing job that is a clone of the data processing job run on the original training graph data, and running it on the new graph data (see [Processing](#)

[updated graph data for Neptune ML \(p. 564\)](#)). Then, call the `modeltransform` command with the new `dataProcessingJobId` and the old `modelTrainingJobId` to recompute the model artifacts on the updated graph data.

For node property prediction, the node embeddings and predictions are recomputed on the new graph data, even for nodes that were present in the original training graph.

For edge property prediction and link prediction, the node embeddings are also recomputed and similarly override any existing node embeddings. To recompute the node embeddings, Neptune ML applies the learned GNN encoder from the previous trained model to the nodes of the new graph data with their new features.

For nodes that do not have features, the learned initial representations from the original model training are re-used. For new nodes that do not have features and were not present in the original training graph, Neptune ML initializes their representation as the average of the learned initial node representations of that node type present in the original training graph. This can cause some performance drop in model predictions if you have many new nodes that do not have features, since they will all be initialized to the average initial embedding for that node type.

If your model is trained with `concat-node-embed` set to `true`, then the initial node representations are created by concatenating the node features with the learnable initial representation. Thus, for the updated graph, the initial node representation of new nodes also uses the average initial node embeddings, concatenated with new node features.

Additionally, node deletions are currently not supported. If nodes have been removed in the updated graph, you have to retrain the model on the updated graph data.

Recomputing the model artifacts re-uses the learned model parameters on a new graph, and should only be done when the new graph is very similar to the old graph. If your new graph is not sufficiently similar, you need to retrain the model to obtain similar model performance on the new graph data. What constitutes sufficiently similar depends on the structure of your graph data, but as a rule of thumb you should retrain your model if your new data is more than 10-20% different from the original training graph data.

For graphs where all the nodes have features, the higher end of the threshold (20% different) applies but for graphs where many nodes do not have features and the new nodes added to the graph don't have properties, then the lower end (10% different) may be even be too high.

See [The `modeltransform` command \(p. 629\)](#) for more information about model transform jobs.

Custom models in Neptune ML

Neptune ML lets you define your own custom model implementations using Python. You can train and deploy custom models using Neptune ML infrastructure very much as you do for the built-in models, and use them to obtain predictions through graph queries.

You can start implementing a custom model of your own in Python by following the [Neptune ML toolkit examples](#), and by using the model components provided in the Neptune ML toolkit. The following sections provide more details.

Contents

- [Overview of custom models in Neptune ML \(p. 596\)](#)
 - [When to use a custom model in Neptune ML \(p. 596\)](#)
 - [Workflow for developing and using a custom model in Neptune ML \(p. 596\)](#)
- [Custom model development in Neptune ML \(p. 598\)](#)
 - [Custom model training script development in Neptune ML \(p. 598\)](#)
 - [Custom model transform script development in Neptune ML \(p. 599\)](#)
 - [Custom model-hpo-configuration.json file in Neptune ML \(p. 599\)](#)
 - [Local testing of your custom model implementation in Neptune ML \(p. 600\)](#)

Overview of custom models in Neptune ML

When to use a custom model in Neptune ML

Neptune ML's built-in models handle all the standard tasks supported by Neptune ML, but there may be cases where you want to have more granular control over the model for a particular task, or need to customize the model training process. For example, a custom model is appropriate in the following situations:

- Feature encoding for text features of very large text models need to be run on GPU.
- You want to use your own custom Graph Neural Network (GNN) model developed in Deep Graph Library (DGL).
- You want to use tabular models or ensemble models for node classification and regression.

Workflow for developing and using a custom model in Neptune ML

Custom model support in Neptune ML is designed to integrate seamlessly into existing Neptune ML workflows. It works by running custom code in your source module on Neptune ML's infrastructure to train the model. Just as is the case for a built-in mode, Neptune ML automatically launches a SageMaker HyperParameter tuning job and selects the best model according to the evaluation metric. It then uses the implementation provided in your source module to generate model artifacts for deployment.

Data export, training configuration, and data preprocessing is the same for a custom model as for a built-in one.

After data preprocessing is when you can iteratively and interactively develop and test your custom model implementation using Python. When your model is production-ready, you can upload the resulting Python module to Amazon S3 like this:

```
aws s3 cp --recursive (source path to module) s3://(bucket name)/(destination path for your module)
```

Then, you can use the normal [default \(p. 535\)](#) or the [incremental \(p. 537\)](#) data workflow to deploy the model to production, with a few differences.

For model training using a custom model, you must provide a `customModelTrainingParameters` JSON object to the Neptune ML model training API to ensure that your custom code is used. The fields in the `customModelTrainingParameters` object are as follows:

- **sourceS3DirectoryPath** – *(Required)* The path to the Amazon S3 location where the Python module implementing your model is located. This must point to a valid existing Amazon S3 location that contains, at a minimum, a training script, a transform script, and a `model-hpo-configuration.json` file.
- **trainingEntryPointScript** – *(Optional)* The name of the entry point in your module of a script that performs model training and takes hyperparameters as command-line arguments, including fixed hyperparameters.

Default: `training.py`.

- **transformEntryPointScript** – *(Optional)* The name of the entry point in your module of a script that should be run after the best model from the hyperparameter search has been identified, to compute the model artifacts necessary for model deployment. It should be able to run with no command-line arguments.

Default: transform.py.

For example:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltraining
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
  "modelName": "custom",
  "customModelTrainingParameters" : {
    "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
module)",
    "trainingEntryPointScript": "(your training script entry-point name in the Python
module)",
    "transformEntryPointScript": "(your transform script entry-point name in the
Python module)"
  }
}'
```

Similarly, to enable a custom model transform, you must provide a `customModelTransformParameters` JSON object to the Neptune ML model transform API, with field values that are compatible with the saved model parameters from the training job. The `customModelTransformParameters` object contains these fields:

- **sourceS3DirectoryPath** – (Required) The path to the Amazon S3 location where the Python module implementing your model is located. This must point to a valid existing Amazon S3 location that contains, at a minimum, a training script, a transform script, and a `model-hpo-configuration.json` file.
- **transformEntryPointScript** – (Optional) The name of the entry point in your module of a script that should be run after the best model from the hyperparameter search has been identified, to compute the model artifacts necessary for model deployment. It should be able to run with no command-line arguments.

Default: transform.py.

For example:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltransform
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "trainingJobName" : "(name of a completed SageMaker training job)",
  "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
  "customModelTransformParameters" : {
    "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
module)",
    "transformEntryPointScript": "(your transform script entry-point name in the
Python module)"
  }
}'
```

Custom model development in Neptune ML

A good way to start custom model development is by following [Neptune ML toolkit examples](#) to structure and write your training module. The Neptune ML toolkit also implements modularized graph ML model components in the [modelzoo](#) that you can stack and use to create your custom model.

In addition, the toolkit provides utility functions that help you generate the necessary artifacts during model training and model transform. You can import this Python package in your custom implementation. Any functions or modules provided in the toolkit are also available in the Neptune ML training environment.

If your Python module has additional external dependencies, you can include these additional dependencies by creating a `requirements.txt` file in your module's directory. The packages listed in the `requirements.txt` file will then be installed before your training script is run.

At a minimum, the Python module that implements your custom model needs to contain the following:

- A training script entry point
- A transform script entry point
- A `model-hpo-configuration.json` file

Custom model training script development in Neptune ML

Your custom model training script should be an executable Python script like the Neptune ML toolkit's `train.py` example. It must accept hyperparameter names and values as command-line arguments. During model training, the hyperparameter names are obtained from the `model-hpo-configuration.json` file. The hyperparameter values either fall within the valid hyperparameter range if the hyperparameter is tunable, or take the default hyperparameter value if it is not tunable.

Your training script is run on a SageMaker training instance using a syntax like this:

```
python3 (script entry point) --(1st parameter) (1st value) --(2nd parameter) (2nd value) (...)
```

For all tasks, the Neptune ML AutoTrainer sends several required parameters to your training script in addition to the hyperparameters that you specify, and your script must be able to handle these additional parameters in order to work properly.

These additional required parameters vary somewhat by task:

For node classification or node regression

- **task** – The task type used internally by Neptune ML. For node classification this is `node_class`, and for node regression it is `node_regression`.
- **model** – The model name used internally by Neptune ML, which is `custom` in this case.
- **name** – The name of the task used internally by Neptune ML, which is `node_class-custom` for node classification in this case, and `node_regression-custom` for node regression.
- **target_ntype** – The name of the node type for classification or regression.
- **property** – The name of the node property for classification or regression.

For link prediction

- **task** – The task type used internally by Neptune ML. For link prediction, this is `link_predict`.
- **model** – The model name used internally by Neptune ML, which is `custom` in this case.

- **name** – The name of the task used internally by Neptune ML, which is `link_predict-custom` in this case.

For edge classification or edge regression

- **task** – The task type used internally by Neptune ML. For edge classification this is `edge_class`, and for edge regression it is `edge_regression`.
- **model** – The model name used internally by Neptune ML, which is `custom` in this case.
- **name** – The name of the task used internally by Neptune ML, which is `edge_class-custom` for edge classification in this case, and `edge_regression-custom` for edge regression.
- **target_etype** – The name of the edge type for classification or regression.
- **property** – The name of the edge property for classification or regression.

Your script should save the model parameters, as well as any other artifacts that will be needed to at the end of training.

You can use Neptune ML toolkit utility functions to determine the location of the processed graph data, the location where the model parameters should be saved, and what GPU devices are available on the training instance. See the [train.py](#) sample training script for examples of how to use these utility functions.

Custom model transform script development in Neptune ML

A transform script is needed to take advantage of the Neptune ML [incremental workflow \(p. 537\)](#) for model inference on evolving graphs without retraining the model. Even if all the artifacts necessary for model deployment are generated by the training script, you still need to provide a transform script if you want to generate updated models without retraining the model.

Your custom model transform script should be an executable Python script like the Neptune ML toolkit's [transform.py](#) example script. Because this script is invoked during model training with no command line arguments, any command line arguments that the script does accept must have defaults.

The script runs on a SageMaker training instance with a syntax like this:

```
python3 (your transform script entry point)
```

Your transform script will need various pieces of information, such as:

- The location of the processed graph data.
- The location where the model parameters are saved and where new model artifacts should be saved.
- The devices available on the instance.
- The hyperparameters that generated the best model.

These inputs are obtained using Neptune ML utility functions that your script can call. See the toolkit's sample [transform.py](#) script for examples of how to do that.

The script should save the node embeddings, node ID mappings, and any other artifacts necessary for model deployment for each task. See the [model artifacts documentation \(p. 593\)](#) for more information about the model artifacts required for different Neptune ML tasks.

Custom `model-hpo-configuration.json` file in Neptune ML

The `model-hpo-configuration.json` file defines hyperparameters for your custom model. It is in the same [format \(p. 581\)](#) as the `model-hpo-configuration.json` file used with the Neptune ML built-in

models, and takes precedence over the version that is auto-generated by Neptune ML and uploaded to the location of your processed data.

When you add a new hyperparameter to your model, you must also add an entry for the hyperparameter in this file so that the hyperparameter is passed to your training script.

You must provide a range for a hyperparameter if you want it to be tunable, and set it as a `tier-1`, `tier-2`, or `tier-3` param. The hyperparameter will be tuned if the total number of training jobs configured allow for tuning hyperparameters in its tier. For a non-tunable parameter, you must provide a default value and add the hyperparameter to the `fixed-param` section of the file. See the toolkit's sample [sample model-hpo-configuration.json file](#) for an example of how to do that.

You must also provide the metric definition that the SageMaker HyperParameter Optimization job will use to evaluate the candidate models trained. To do this, you add an `eval_metric` JSON object to the `model-hpo-configuration.json` file like this:

```
"eval_metric": {
  "tuning_objective": {
    "MetricName": "(metric_name)",
    "Type": "Maximize"
  },
  "metric_definitions": [
    {
      "Name": "(metric_name)",
      "Regex": "(metric regular expression)"
    }
  ]
},
```

The `metric_definitions` array in the `eval_metric` object lists metric definition objects for each metric that you want SageMaker to extract from the training instance. Each metric definition object has a `Name` key that lets you provide a name for the metric (such as "accuracy", "f1", and so on) The `Regex` key lets you provide a regular expression string that matches how that particular metric is printed in the training logs. See the [SageMaker HyperParameter Tuning page](#) for more details on how to define metrics.

The `tuning_objective` object in `eval_metric` then allows you to specify which of the metrics in `metric_definitions` should be used as the evaluation metric that serves as the objective metric for hyperparameter optimization. The value for the `MetricName` must match the value of a `Name` in one of the definitions in `metric_definitions`. The value for `Type` should be either "Maximize" or "Minimize" depending on whether the metric should be interpreted as greater-is-better (like "accuracy") or less-is-better (like "mean-squared-error").

Errors in this section of the `model-hpo-configuration.json` file can result in failures of the Neptune ML model training API job, because the SageMaker HyperParameter Tuning job will not be able to select the best model.

Local testing of your custom model implementation in Neptune ML

You can use the Neptune ML toolkit Conda environment to run your code locally in order to test and validate your model. If you're developing on a Neptune Notebook instance, then this Conda environment will be pre-installed on the Neptune Notebook instance. If you're developing on a different instance, then you need to follow the [local setup instructions](#) in the Neptune ML toolkit.

The Conda environment accurately reproduces the environment where your model will run when you call the [model training API \(p. 624\)](#). All of the example training scripts and transform scripts allow you to pass a command line `--local` flag to run the scripts in a local environment for easy debugging. This is a

good practice while developing your own model because it allows you to interactively and iteratively test your model implementation. During model training in the Neptune ML production training environment, this parameter is omitted.

Creating an inference endpoint to query

An inference endpoint lets you query one specific model that the model-training process constructed. The endpoint attaches to the best-performing model of a given type that the training process was able to generate. The endpoint is then able to accept Gremlin queries from Neptune and return that model's predictions for inputs in the queries. After you have created an inference endpoint, it stays active until you delete it.

Managing inference endpoints for Neptune ML

After you have completed model training on data that you exported from Neptune, you can create an inference endpoint using a `curl` (or `awscli`) command like the following:

```
curl \
-X POST https://(your Neptune endpoint)/ml/endpoints
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique ID for the new endpoint)",
  "mlModelTrainingJobId": "(the model-training job-id of a completed job)"
}'
```

You can also create an inference endpoint from a model created by a completed model transform job, in much the same way:

```
curl \
-X POST https://(your Neptune endpoint)/ml/endpoints
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique ID for the new endpoint)",
  "mlModelTransformJobId": "(the model-transform job-id of a completed job)"
}'
```

The details of how to use these commands are explained in [The endpoints command \(p. 633\)](#), along with information about how to get the status of an endpoint, how to delete an endpoint, and how to list all inference endpoints.

Inference queries in Neptune ML

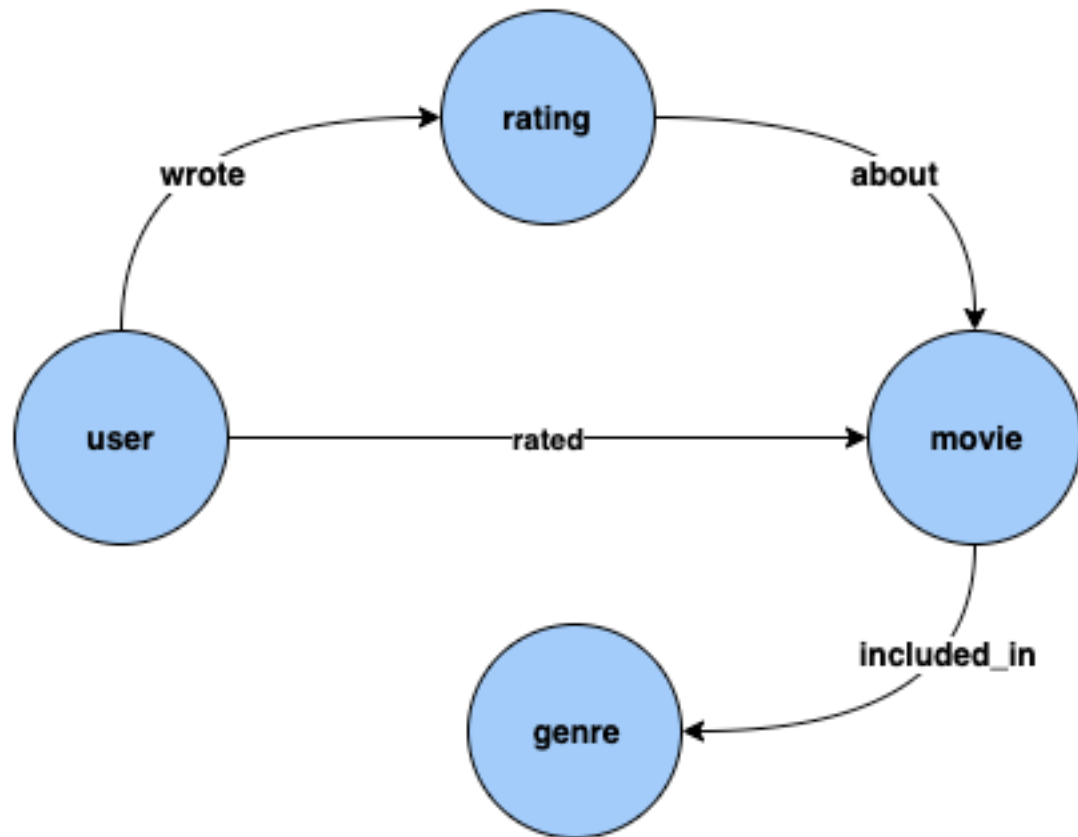
You can use either Gremlin or SPARQL to query a Neptune ML inference endpoint.

Gremlin inference queries in Neptune ML

As described in [Neptune ML capabilities \(p. 522\)](#), Neptune ML supports training models that can do the following kinds of inference tasks:

- **Node classification** – Predicts the categorical feature of a vertex property.
- **Node regression** – Predicts a numerical property of a vertex.
- **Edge classification** – Predicts the categorical feature of an edge property.
- **Edge regression** – Predicts a numerical property of an edge.
- **Link prediction** – Predicts destination nodes given a source node and outgoing edge, or source nodes given a destination node and incoming edge.

We can illustrate these different tasks with examples that use the [MovieLens 100k dataset](#) provided by [GroupLens Research](#). This dataset consists of movies, users, and ratings of the movies by the users, from which we've created a property graph like this:



Node classification: In the dataset above, **Genre** is a vertex type which is connected to vertex type **Movie** by edge **included_in**. However, if we tweak the dataset to make **Genre** a [categorical](#) feature for vertex type **Movie**, then the problem of inferring **Genre** for new movies added to our knowledge graph can be solved using node classification models.

Node regression: If we consider the vertex type **Rating**, which has properties like **timestamp** and **score**, then the problem of inferring the numerical value **Score** for a **Rating** can be solved using node regression models.

Edge classification: Similarly, for a `Rated` edge, if we have a property `Scale` that can have one of the values, `Love`, `Like`, `Dislike`, `Neutral`, `Hate`, then the problem of inferring `Scale` for the `Rated` edge for new movies/ratings can be solved using edge classification models.

Edge regression: Similarly, for the same `Rated` edge, if we have a property `Score` that holds a numerical value for the rating, then this can be inferred from edge regression models.

Link prediction: Problems like, find the top ten users who are most likely to rate a given movie, or find the top ten Movies that a given user is most likely to rate, falls under link prediction.

Note

For Neptune ML use-cases, we have a very rich set of notebooks designed to give you a hands-on understanding of each use-case. You can create these notebooks along with your Neptune cluster when you use the [Neptune ML AWS CloudFormation template \(p. 523\)](#) to create a Neptune ML cluster. These notebooks are also available on [github](#) as well.

Topics

- [Neptune ML predicates used in Gremlin inference queries \(p. 605\)](#)
- [Gremlin node classification queries in Neptune ML \(p. 607\)](#)
- [Gremlin node regression queries in Neptune ML \(p. 609\)](#)
- [Gremlin edge classification queries in Neptune ML \(p. 609\)](#)
- [Gremlin edge regression queries in Neptune ML \(p. 611\)](#)
- [Gremlin link prediction queries using link-prediction models in Neptune ML \(p. 612\)](#)
- [List of exceptions for Neptune ML Gremlin inference queries \(p. 613\)](#)

Neptune ML predicates used in Gremlin inference queries

Neptune#ml.endpoint

The `Neptune#ml.endpoint` predicate is used in a `with()` step to specify the inference endpoint, if necessary:

```
.with("Neptune#ml.endpoint", "the model's SageMaker inference endpoint")
```

You can identify the endpoint either by its `id` or its URL. For example:

```
.with( "Neptune#ml.endpoint", "node-classification-movie-lens-endpoint" )
```

Or:

```
.with( "Neptune#ml.endpoint", "https://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/node-classification-movie-lens-endpoint/invocations" )
```

Note

If you [set the `neptune_ml_endpoint` parameter \(p. 534\)](#) in your Neptune DB cluster parameter group to the endpoint `id` or URL, you don't need to include the `Neptune#ml.endpoint` predicate in each query.

Neptune#ml.iamRoleArn

`Neptune#ml.iamRoleArn` is used in a `with()` step to specify the ARN of the SageMaker execution IAM role, if necessary:

```
.with("Neptune#ml.iamRoleArn", "the ARN for the SageMaker execution IAM role")
```

For information about how to create the SageMaker execution IAM role, see [Manual setup \(p. 527\)](#).

Note

If you [set the `neptune_ml_iam_role` parameter \(p. 532\)](#) in your Neptune DB cluster parameter group to the ARN of your SageMaker execution IAM role, you don't need to include the `Neptune#ml.iamRoleArn` predicate in each query.

Neptune#ml.limit

The `Neptune#ml.limit` predicate optionally limits the number of results returned per entity:

```
.with( "Neptune#ml.limit", 2 )
```

By default, the limit is 1.

Neptune#ml.threshold

The `Neptune#ml.threshold` predicate optionally establishes a cutoff threshold for result scores:

```
.with( "Neptune#ml.threshold", 0.5D )
```

This lets you discard all results with scores below the specified threshold.

Neptune#ml.classification

The `Neptune#ml.classification` predicate is attached to the `properties()` step to establish that the properties need to be fetched from the SageMaker endpoint of the node classification model:

```
.properties( "property key of the node classification model" ).with( "Neptune#ml.classification" )
```

Neptune#ml.regression

The `Neptune#ml.regression` predicate is attached to the `properties()` step to establish that the properties need to be fetched from the SageMaker endpoint of the node regression model:

```
.properties( "property key of the node regression model" ).with( "Neptune#ml.regression" )
```

Neptune#ml.prediction

The `Neptune#ml.prediction` predicate is attached to `in()` and `out()` steps to establish that this is a link-prediction query:

```
.in("edge label of the link prediction model").with("Neptune#ml.prediction").hasLabel("target node label")
```

Neptune#ml.score

The `Neptune#ml.score` predicate is used in Gremlin node or edge classification queries to fetch a machine-learning confidence Score. The `Neptune#ml.score` predicate should be passed together with the query predicate in the `properties()` step to obtain an ML confidence score for node or edge classification queries.

You can find a node classification example with [other node classification examples \(p. 608\)](#), and an edge classification example in the [edge classification section \(p. 609\)](#).

Gremlin node classification queries in Neptune ML

For Gremlin node classification in Neptune ML:

- The model is trained on one property of the vertices. The set of unique values of this property are referred to as a set of node classes, or simply, classes.
- The node class or categorical property value of a vertex's property can be inferred from the node classification model. This is useful where this property is not already attached to the vertex.
- In order to fetch one or more classes from a node classification model, you need to use the `with()` step with the predicate `Neptune#ml.classification` to configure the `properties()` step. The output format is similar to what you would expect if those were vertex properties.

Here is a sample node classification query:

```
g.with( "Neptune#ml.endpoint", "node-classification-movie-lens-endpoint" )
  .with( "Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role" )
  .with( "Neptune#ml.limit", 2 )
  .with( "Neptune#ml.threshold", 0.5D )
  .V( "movie_1", "movie_2", "movie_3" )
  .properties("genre").with("Neptune#ml.classification")
```

The output of this query would look something like the following:

```
==>vp[genre->Action]
==>vp[genre->Crime]
==>vp[genre->Comedy]
```

In the query above, the `V()` and `properties()` steps are used as follows:

The `V()` step contains the set of vertices for which you want to fetch the classes from the node-classification model:

```
.V( "movie_1", "movie_2", "movie_3" )
```

The `properties()` step contains the key on which the model was trained, and has `.with("Neptune#ml.classification")` to indicate that this is a node classification ML inference query.

Multiple property keys are not currently supported in a `properties().with("Neptune#ml.classification")` step. For example, the following query results in an exception:

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .V( "movie_1", "movie_2", "movie_3" )
  .properties("genre", "other_label").with("Neptune#ml.classification")
```

For the specific error message, see the [list of Neptune ML exceptions \(p. 613\)](#).

A `properties().with("Neptune#ml.classification")` step can be used in combination with any of the following steps:

- `value()`
- `value().is()`
- `hasValue()`

- `has(value, "")`
- `key()`
- `key().is()`
- `hasKey()`
- `has(key, "")`
- `path()`

Other node-classification queries

If both the inference endpoint and the corresponding IAM role have been saved in your DB cluster parameter group, a node-classification query can be as simple as this:

```
g.V("movie_1", "movie_2", "movie_3").properties("genre").with("Neptune#ml.classification")
```

You can mix vertex properties and classes in a query using the `union()` step:

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1", "movie_2", "movie_3")
  .union(
    properties("genre").with("Neptune#ml.classification"),
    properties("genre")
  )
```

You can also make an unbounded query such as this:

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .V()
  .properties("genre").with("Neptune#ml.classification")
```

You can retrieve the node classes together with vertices using the `select()` step together with the `as()` step:

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1", "movie_2", "movie_3").as("vertex")
  .properties("genre").with("Neptune#ml.classification").as("properties")
  .select("vertex", "properties")
```

You can also filter on node classes, as illustrated in these examples:

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1", "movie_2", "movie_3")
  .properties("genre").with("Neptune#ml.classification")
  .has(value, "Horror")

g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1", "movie_2", "movie_3")
  .properties("genre").with("Neptune#ml.classification")
  .has(value, P.eq("Action"))

g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1", "movie_2", "movie_3")
```

```
.properties("genre").with("Neptune#ml.classification")  
.has(value, P.within("Action", "Horror"))
```

You can get a node classification confidence score using the `Neptune#ml.score` predicate:

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.V("movie_1", "movie_2", "movie_3")  
.properties("genre", "Neptune#ml.score").with("Neptune#ml.classification")
```

The response would look like this:

```
==>vp[genre->Action]  
==>vp[Neptune#ml.score->0.01234567]  
==>vp[genre->Crime]  
==>vp[Neptune#ml.score->0.543210]  
==>vp[genre->Comedy]  
==>vp[Neptune#ml.score->0.10101]
```

Gremlin node regression queries in Neptune ML

Node regression is similar to node classification, except that the value inferred from the regression model for each node is numeric. You can use the same Gremlin queries for node regression as for node classification except for the following differences:

- Again, in Neptune ML, nodes refer to vertices.
- The `properties()` step takes the form, `properties().with("Neptune#ml.regression")` instead of `properties().with("Neptune#ml.classification")`.
- The `"Neptune#ml.limit"` and `"Neptune#ml.threshold"` predicates are not applicable.
- When you filter on the value, you have to specify a numeric value.

Here is a sample vertex classification query:

```
g.with("Neptune#ml.endpoint", "node-regression-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.V("movie_1", "movie_2", "movie_3")  
.properties("revenue").with("Neptune#ml.regression")
```

You can filter on the value inferred using a regression model, as illustrated in the following examples:

```
g.with("Neptune#ml.endpoint", "node-regression-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.V("movie_1", "movie_2", "movie_3")  
.properties("revenue").with("Neptune#ml.regression")  
.value().is(P.gte(1600000))  
  
g.with("Neptune#ml.endpoint", "node-regression-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.V("movie_1", "movie_2", "movie_3")  
.properties("revenue").with("Neptune#ml.regression")  
.hasValue(P.lte(1600000D))
```

Gremlin edge classification queries in Neptune ML

For Gremlin edge classification in Neptune ML:

- The model is trained on one property of the edges. The set of unique values of this property is referred to as a set of classes.
- The class or categorical property value of an edge can be inferred from the edge classification model, which is useful when this property is not already attached to the edge.
- In order to fetch one or more classes from an edge classification model, you need to use the `with()` step with the predicate, `"Neptune#ml.classification"` to configure the `properties()` step. The output format is similar to what you would expect if those were edge properties.

Here is an example of an edge classification query that requests a confidence score using the `Neptune#ml.score` predicate:

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .E("relationship_1", "relationship_2", "relationship_3")
  .properties("knows_by", "Neptune#ml.score").with("Neptune#ml.classification")
```

The response would look like this:

```
==>p[knows_by->"Family"]
==>p[Neptune#ml.score->0.01234567]
==>p[knows_by->"Friends"]
==>p[Neptune#ml.score->0.543210]
==>p[knows_by->"Colleagues"]
==>p[Neptune#ml.score->0.10101]
```

Syntax of a Gremlin edge classification query

For a simple graph where `User` is the head and tail node, and `Relationship` is the edge that connects them, an example edge classification query is:

```
g.with("Neptune#ml.endpoint", "edge-classification-social-endpoint")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
  .E("relationship_1", "relationship_2", "relationship_3")
  .properties("knows_by").with("Neptune#ml.classification")
```

The output of this query would look something like the following:

```
==>p[knows_by->"Family"]
==>p[knows_by->"Friends"]
==>p[knows_by->"Colleagues"]
```

In the query above, the `E()` and `properties()` steps are used as follows:

- The `E()` step contains the set of edges for which you want to fetch the classes from the edge-classification model:

```
.E("relationship_1", "relationship_2", "relationship_3")
```

- The `properties()` step contains the key on which the model was trained, and has `.with("Neptune#ml.classification")` to indicate that this is an edge classification ML inference query.

Multiple property keys are not currently supported in a `properties().with("Neptune#ml.classification")` step. For example, the following query results in an exception being thrown:

```
g.with("Neptune#ml.endpoint","edge-classification-social-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .E("relationship_1","relationship_2","relationship_3")
  .properties("knows_by","other_label").with("Neptune#ml.classification")
```

For specific error messages, see [List of exceptions for Neptune ML Gremlin inference queries \(p. 613\)](#).

A `properties().with("Neptune#ml.classification")` step can be used in combination with any of the following steps:

- `value()`
- `value().is()`
- `hasValue()`
- `has(value,"")`
- `key()`
- `key().is()`
- `hasKey()`
- `has(key,"")`
- `path()`

Gremlin edge regression queries in Neptune ML

Edge regression is similar to edge classification, except that the value inferred from the ML model is numeric. For edge regression, Neptune ML supports the same queries as for classification.

Key points to note are:

- You need to use the ML predicate `"Neptune#ml.regression"` to configure the `properties()` step for this use-case.
- The `"Neptune#ml.limit"` and `"Neptune#ml.threshold"` predicates are not applicable in this use-case.
- For filtering on the value, you need to specify the value as numerical.

Syntax of a Gremlin edge regression query

For a simple graph where `User` is the head node, `Movie` is the tail node, and `Rated` is the edge that connects them, here is an example edge regression query that finds the numeric rating value, referred to as `score` here, for the edge `Rated`:

```
g.with("Neptune#ml.endpoint","edge-regression-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .E("rating_1","rating_2","rating_3")
  .properties("score").with("Neptune#ml.regression")
```

You can also filter on a value inferred from the ML regression model. For the existing `Rated` edges (from `User` to `Movie`) identified by `"rating_1"`, `"rating_2"`, and `"rating_3"`, where the edge property `Score` is not present for these ratings, you can use a query like following to infer `Score` for the edges where it is greater than or equal to 9:

```
g.with("Neptune#ml.endpoint","edge-regression-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .E("rating_1","rating_2","rating_3")
```

```
.properties("score").with("Neptune#ml.regression")  
.value().is(P.gte(9))
```

Gremlin link prediction queries using link-prediction models in Neptune ML

Link-prediction models can solve problems such as the following:

- **Head-node prediction:** Given a vertex and an edge type, what vertices is that vertex likely to link from?
- **Tail-node prediction:** Given a vertex and an edge label, what vertices is that vertex likely to link to?
- **Edge-type prediction:** Given a head and a tail vertex, what edge is likely to link them?

The first two cases belong to node prediction problems, and the last one is an edge type prediction.

Note

Edge prediction is not yet supported in Neptune ML.

For the examples below, consider a simple graph with the vertices `User` and `Movie` that are linked by the edge `Rated`.

Here is a sample head-node prediction query, used to predict the top five users most likely to rate the movies, "movie_1", "movie_2", and "movie_3":

```
g.with("Neptune#ml.endpoint", "node-prediction-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.with("Neptune#ml.limit", 5)  
.V("movie_1", "movie_2", "movie_3")  
.in("rated").with("Neptune#ml.prediction").hasLabel("user")
```

Here is a similar one for tail-node prediction, used to predict the top five movies that user "user_1" is likely to rate:

```
g.with("Neptune#ml.endpoint", "node-prediction-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.V("user_1")  
.out("rated").with("Neptune#ml.prediction").hasLabel("movie")
```

Both the edge label and the predicted vertex label are required. If either is omitted, an exception is thrown. For example, the following query without a predicted vertex label throws an exception:

```
g.with("Neptune#ml.endpoint", "node-prediction-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.V("user_1")  
.out("rated").with("Neptune#ml.prediction")
```

Similarly, the following query without an edge label throws an exception:

```
g.with("Neptune#ml.endpoint", "node-prediction-movie-lens-endpoint")  
.with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")  
.V("user_1")  
.out().with("Neptune#ml.prediction").hasLabel("movie")
```

For the specific error messages that these exceptions return, see the [list of Neptune ML exceptions \(p. 613\)](#).

Other link-prediction queries

You can use the `select()` step with the `as()` step to output the predicted vertices together with the input vertices:

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1").as("source")
  .in("rated").with("Neptune#ml.prediction").hasLabel("user").as("target")
  .select("source","target")

g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("user_1").as("source")
  .out("rated").with("Neptune#ml.prediction").hasLabel("movie").as("target")
  .select("source","target")
```

You can make unbounded queries, like these:

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("user_1")
  .out("rated").with("Neptune#ml.prediction").hasLabel("movie")

g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
  .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
  .V("movie_1")
  .in("rated").with("Neptune#ml.prediction").hasLabel("user")
```

List of exceptions for Neptune ML Gremlin inference queries

- **BadRequestException** – The credentials for the supplied role cannot be loaded.
Message: Unable to load credentials for role: *the specified IAM Role ARN*.
- **BadRequestException** – The specified IAM role is not authorized to invoke the SageMaker endpoint.
Message: User: *the specified IAM Role ARN* is not authorized to perform: sagemaker:InvokeEndpoint on resource: *the specified endpoint*.
- **BadRequestException** – The specified endpoint does not exist.
Message: Endpoint *the specified endpoint* not found.
- **InvalidParameterException** – The specified endpoint is not syntactically valid.
Message: Invalid endpoint provided for external service query.
- **InvalidParameterException** – The specified SageMaker execution IAM Role ARN is not syntactically valid.
Message: Invalid IAM role ARN provided for external service query.
- **InvalidParameterException** – Multiple property keys are specified in the `properties()` step in a query.
Message: ML inference queries are currently supported for one property key.
- **InvalidParameterException** – Multiple edge labels are specified in a query.
Message: ML inference are currently supported only with one edge label.
- **InvalidParameterException** – Multiple vertex label constraints are specified in a query.

Message: ML inference are currently supported only with one vertex label constraint.

- **InvalidParameterException** – Both Neptune#ml.classification and Neptune#ml.regression predicates are present in the same query.

Message: Both regression and classification ML predicates cannot be specified in the query.

- **InvalidParameterException** – More than one edge label was specified in the in() or out() step in a link-prediction query.

Message: ML inference are currently supported only with one edge label.

- **InvalidParameterException** – More than one property key was specified with Neptune#ml.score.

Message: Neptune ML inference queries are currently supported for one property key and one Neptune#ml.score property key.

- **MissingParameterException** – The endpoint was not specified in the query or as a DB cluster parameter.

Message: No endpoint provided for external service query.

- **MissingParameterException** – The SageMaker execution IAM role was not specified in the query or as a DB cluster parameter.

Message: No IAM role ARN provided for external service query.

- **MissingParameterException** – The property key is missing from the properties() step in a query.

Message: Property key needs to be specified using properties() step for ML inference queries.

- **MissingParameterException** – No edge label was specified in the in() or out() step of a link-prediction query.

Message: Edge label needs to be specified while using in() or out() step for ML inference queries.

- **MissingParameterException** – No property key was specified with Neptune#ml.score.

Message: Property key needs to be specified along with Neptune#ml.score property key while using the properties() step for Neptune ML inference queries.

- **UnsupportedOperationException** – The both() step is used in a link-prediction query.

Message: ML inference queries are currently not supported with both() step.

- **UnsupportedOperationException** – No predicted vertex label was specified in the has() step with the in() or out() step in a link-prediction query.

Message: Predicted vertex label needs to be specified using has() step for ML inference queries.

SPARQL inference queries in Neptune ML

Neptune ML maps the RDF graph into a property graph to model the ML Task. Currently, it supports the following use-cases:

- **Object classification** – Predicts the categorical feature of an object.
- **Object regression** – Predicts a numerical property of an object.
- **Object prediction** – Predicts an object given a subject and a relationship.
- **Subject prediction** – Predicts a subject given an object and a relationship.

Note

Neptune ML does not support subject classification and regression use cases with SPARQL.

Neptune ML predicates used in SPARQL inference queries

The following predicates are used with SPARQL inference:

`neptune-ml:timeout` predicate

Specifies the timeout for connection with the remote server. Should not be confused with the query request timeout, which is the maximum amount of time the server can take to satisfy a request.

Note that if the query timeout occurs before the service timeout specified by the `neptune-ml:timeout` predicate occurs, the service connection is canceled too.

`neptune-ml:outputClass` predicate

The `neptune-ml:outputClass` predicate is only used to define the class of the predicted object for object prediction or predicted subject for subject prediction.

`neptune-ml:outputScore` predicate

The `neptune-ml:outputScore` predicate is a positive number that represents the likelihood that the output of a machine learning model is correct.

`neptune-ml:modelType` predicate

The `neptune-ml:modelType` predicate specifies the type of machine learning model being trained:

- OBJECT_CLASSIFICATION
- OBJECT_REGRESSION
- OBJECT_PREDICTION
- SUBJECT_PREDICTION

`neptune-ml:input` predicate

The `neptune-ml:input` predicate refers to the list of URIs used as inputs for Neptune ML.

`neptune-ml:output` predicate

The `neptune-ml:output` predicate refers to the list of binding sets where Neptune ML returns results.

`neptune-ml:predicate` predicate

The `neptune-ml:predicate` predicate is used differently depending on the task being performed:

- For **object or subject prediction**: defines the type of predicate (the edge or relationship type).
- For **object classification and regression**: defines the literal (property) we want to predict.

neptune-ml:batchSize predicate

The `neptune-ml:batchSize` specifies the input size for the remote service call.

SPARQL object classification examples

For SPARQL object classification in Neptune ML, the model is trained on one of the predicate values. This is useful where that predicate is not already present with a given subject.

Only categorical predicate values can be inferred using the object classification model.

The following query seeks to predict the `<http://www.example.org/team>` predicate value for all the inputs of type `foaf:Person`:

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_CLASSIFICATION' ;
                    neptune-ml:input ?input ;
                    neptune-ml:predicate <http://www.example.org/team> ;
                    neptune-ml:output ?output .
  }
}
```

This query can be customized as follows:

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:endpoint 'node-prediction-account-balance-endpoint' ;
                    neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-role' ;

                    neptune-ml:batchSize "40"^^xsd:integer ;
                    neptune-ml:timeout "1000"^^xsd:integer ;

                    neptune-ml:modelType 'OBJECT_CLASSIFICATION' ;
                    neptune-ml:input ?input ;
                    neptune-ml:predicate <http://www.example.org/team> ;
                    neptune-ml:output ?output .
  }
}
```

SPARQL object regression examples

Object regression is similar to object classification, except that a numerical predicate value is inferred from the regression model for each node. You can use the same SPARQL queries for object regression as for object classification with the exception that the `Neptune#ml.limit` and `Neptune#ml.threshold` predicates are not applicable.

The following query seeks to predict the `<http://www.example.org/accountbalance>` predicate value for all the inputs of type `foaf:Person`:

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_REGRESSION' ;
                    neptune-ml:input ?input ;
                    neptune-ml:predicate <http://www.example.org/accountbalance> ;
  }
}
```

```
    neptune-ml:output ?output .  
  }  
}
```

This query can be customized as follows:

```
SELECT * WHERE { ?input a foaf:Person .  
  SERVICE neptune-ml:inference {  
    neptune-ml:config neptune-ml:endpoint 'node-prediction-account-balance-endpoint' ;  
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-role' ;  
  
                      neptune-ml:batchSize "40"^^xsd:integer ;  
                      neptune-ml:timeout "1000"^^xsd:integer ;  
  
                      neptune-ml:modelType 'OBJECT_REGRESSION' ;  
                      neptune-ml:input ?input ;  
                      neptune-ml:predicate <http://www.example.org/accountbalance> ;  
                      neptune-ml:output ?output .  
  }  
}
```

SPARQL object prediction example

Object prediction predicts the object value for a given subject and predicate.

The following object-prediction query seeks to predict what movie the input of type `foaf:Person` would like:

```
?x a foaf:Person .  
?x <http://www.example.org/likes> ?m .  
?m a <http://www.example.org/movie> .  
  
## Query  
SELECT * WHERE { ?input a foaf:Person .  
  SERVICE neptune-ml:inference {  
    neptune-ml:config neptune-ml:modelType 'OBJECT_PREDICTION' ;  
                      neptune-ml:input ?input ;  
                      neptune-ml:predicate <http://www.example.org/likes> ;  
                      neptune-ml:output ?output ;  
                      neptune-ml:outputClass <http://www.example.org/movie> .  
  }  
}
```

The query itself could be customized as follows:

```
SELECT * WHERE { ?input a foaf:Person .  
  SERVICE neptune-ml:inference {  
    neptune-ml:config neptune-ml:endpoint 'node-prediction-user-movie-prediction-  
endpoint' ;  
  
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-role' ;  
  
                      neptune-ml:limit "5"^^xsd:integer ;  
                      neptune-ml:batchSize "40"^^xsd:integer ;  
                      neptune-ml:threshold "0.1"^^xsd:double ;  
                      neptune-ml:timeout "1000"^^xsd:integer ;  
                      neptune-ml:outputScore ?score ;  
  
                      neptune-ml:modelType 'OBJECT_PREDICTION' ;  
                      neptune-ml:input ?input ;  
                      neptune-ml:predicate <http://www.example.org/likes> ;  
                      neptune-ml:output ?output ;
```

```
    neptune-ml:outputClass <http://www.example.org/movie> .  
  }  
}
```

SPARQL subject prediction example

Subject prediction predicts the subject given a predicate and an object.

For example, the following query predicts who (of type `foaf:User`) will watch a given movie:

```
SELECT * WHERE { ?input (a foaf:Movie) .  
  SERVICE neptune-ml:inference {  
    neptune-ml:config neptune-ml:modelType 'SUBJECT_PREDICTION' ;  
                      neptune-ml:input ?input ;  
                      neptune-ml:predicate <http://aws.amazon.com/neptune/csv2rdf/  
object_Property/rated> ;  
                      neptune-ml:output ?output ;  
                      neptune-ml:outputClass <http://aws.amazon.com/neptune/csv2rdf/  
class/User> ;  
  }  
}
```

List of exceptions for Neptune ML SPARQL inference queries

- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` expects at least 1 value for the parameter `(parameter name)`, found zero.
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` expects at most 1 value for the parameter `(parameter name)`, found `(a number)` values.
- **BadRequestException** - *Message:* Invalid predicate `(predicate name)` provided for external service `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` query.
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` expects the predicate `(predicate name)` to be defined.
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` expects the value of (parameter) `(parameter name)` to be a variable, found: `(type)"`
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` expects the input `(parameter name)` to be a constant, found: `(type)`.
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` is expected to return only 1 value.
- **BadRequestException** - *Message:* "The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` only allows StatementPatternNodes.
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` does not allow the predicate `(predicate name)`.
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` predicates cannot be variables, found: `(type)`.
- **BadRequestException** - *Message:* The SERVICE `http://aws.amazon.com/neptune/vocab/v01/services/ml#inference` predicates are expected to be part of the namespace `(namespace name)`, found: `(namespace name)`.

Neptune ML management API reference

Contents

- [Data processing using the dataprocessing command \(p. 620\)](#)
 - [Creating a data-processing job using the Neptune ML dataprocessing command \(p. 620\)](#)
 - [Getting the status of a data-processing job using the Neptune ML dataprocessing command \(p. 621\)](#)
 - [Stopping a data-processing job using the Neptune ML dataprocessing command \(p. 622\)](#)
 - [Listing active data-processing jobs using the Neptune ML dataprocessing command \(p. 622\)](#)
- [Model training using the modeltraining command \(p. 624\)](#)
 - [Creating a model-training job using the Neptune ML modeltraining command \(p. 624\)](#)
 - [Getting the status of a model-training job using the Neptune ML modeltraining command \(p. 626\)](#)
 - [Stopping a model-training job using the Neptune ML modeltraining command \(p. 627\)](#)
 - [Listing active model-training jobs using the Neptune ML modeltraining command \(p. 627\)](#)
- [Model transform using the modeltransform command \(p. 629\)](#)
 - [Creating a model-transform job using the Neptune ML modeltransform command \(p. 629\)](#)
 - [Getting the status of a model-transform job using the Neptune ML modeltransform command \(p. 631\)](#)
 - [Stopping a model-transform job using the Neptune ML modeltransform command \(p. 631\)](#)
 - [Listing active model-transform jobs using the Neptune ML modeltransform command \(p. 632\)](#)
- [Managing inference endpoints using the endpoints command \(p. 633\)](#)
 - [Creating an inference endpoint using the Neptune ML endpoints command \(p. 633\)](#)
 - [Getting the status of an inference endpoint using the Neptune ML endpoints command \(p. 634\)](#)
 - [Deleting an instance endpoint using the Neptune ML endpoints command \(p. 635\)](#)
 - [Listing inference endpoints using the Neptune ML endpoints command \(p. 635\)](#)
- [Neptune ML management API exceptions \(p. 637\)](#)

Data processing using the dataprocessing command

You use the Neptune ML `dataprocessing` command to create a data processing job, check its status, stop it, or list all active data-processing jobs.

Creating a data-processing job using the Neptune ML `dataprocessing` command

A typical Neptune ML `dataprocessing` command for creating a new job looks like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/dataprocessing \
-H 'Content-Type: application/json' \
-d '{
  "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input folder)",
  "id" : "(a job ID for the new job)",
  "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output folder)"
}'
```

A command to initiate incremental re-processing looks like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/dataprocessing \
-H 'Content-Type: application/json' \
-d '{
  "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input folder)",
  "id" : "(a job ID for this job)",
  "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output folder)"
  "previousDataProcessingJobId" : "(the job ID of a previously completed job to
update)"
}'
```

Parameters for `dataprocessing` job creation

- **id** – (Optional) A unique identifier for the new job.
Type: string. *Default:* An autogenerated UUID.
- **previousDataProcessingJobId** – (Optional) The job ID of a completed data processing job run on an earlier version of the data.
Type: string. *Default:* none.
Note: Use this for incremental data processing, to update the model when graph data has changed (but not when data has been deleted).
- **inputDataS3Location** – (Required) The URI of the Amazon S3 location where you want SageMaker to download the data needed to run the data processing job.
Type: string.
- **processedDataS3Location** – (Required) The URI of the Amazon S3 location where you want SageMaker to save the results of a data processing job.
Type: string.
- **sagemakerIamRoleArn** – (Optional) The ARN of an IAM role for SageMaker execution.
Type: string. *Note:* This must be listed in your DB cluster parameter group or an error will occur.

- **neptuneIamRoleArn** – (Optional) The Amazon Resource Name (ARN) of an IAM role that SageMaker can assume to perform tasks on your behalf.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

- **processingInstanceType** – (Optional) The type of ML instance used during data processing. Its memory should be large enough to hold the processed dataset.

Type: string. Default: the smallest m1.r5 type whose memory is ten times larger than the size of the exported graph data on disk.

Note: Neptune ML can select the instance type automatically. See [Selecting an instance for data processing \(p. 539\)](#).

- **processingInstanceVolumeSizeInGB** – (Optional) The disk volume size of the processing instance. Both input data and processed data are stored on disk, so the volume size must be large enough to hold both data sets.

Type: integer. Default: 0.

Note: If not specified or 0, Neptune ML chooses the volume size automatically based on the data size.

- **processingTimeOutInSeconds** – (Optional) Timeout in seconds for the data processing job.

Type: integer. Default: 86,400 (1 day).

- **modelType** – (Optional) One of the two model types that Neptune ML currently supports: heterogeneous graph models (heterogeneous), and knowledge graph (kge).

Type: string. Default: none.

Note: If not specified, Neptune ML chooses the model type automatically based on the data.

- **configFileName** – (Optional) A data specification file that describes how to load the exported graph data for training. The file is automatically generated by the Neptune export toolkit.

Type: string. Default: training-data-configuration.json.

- **subnets** – (Optional) The IDs of the subnets in the Neptune VPC.

Type: list of strings. Default: none.

- **securityGroupIds** – (Optional) The VPC security group IDs.

Type: list of strings. Default: none.

- **volumeEncryptionKMSKey** – (Optional) The AWS Key Management Service (AWS KMS) key that SageMaker uses to encrypt data on the storage volume attached to the ML compute instances that run the processing job.

Type: string. Default: none.

- **s3OutputEncryptionKMSKey** – (Optional) The AWS Key Management Service (AWS KMS) key that SageMaker uses to encrypt the output of the training job.

Type: string. Default: none.

Getting the status of a data-processing job using the Neptune ML dataprocessing command

A sample Neptune ML dataprocessing command for the status of a job looks like this:

```
curl -s \
```

```
"https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)" \  
| python -m json.tool
```

Parameters for dataprocessing job status

- **id** – (Required) The unique identifier of the data-processing job.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

Stopping a data-processing job using the Neptune ML dataprocessing command

A sample Neptune ML dataprocessing command for stopping a job looks like this:

```
curl -s \  
-X DELETE "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)"
```

Or this:

```
curl -s \  
-X DELETE "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)?clean=true"
```

Parameters for dataprocessing stop job

- **id** – (Required) The unique identifier of the data-processing job.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

- **clean** – (Optional) This flag specifies that all Amazon S3 artifacts should be deleted when the job is stopped.

Type: Boolean. Default: FALSE.

Listing active data-processing jobs using the Neptune ML dataprocessing command

A sample Neptune ML dataprocessing command for listing active jobs looks like this:

```
curl -s "https://(your Neptune endpoint)/ml/dataprocessing"
```

Or this:

```
curl -s "https://(your Neptune endpoint)/ml/dataprocessing?maxItems=3"
```

Parameters for dataprocessing list jobs

- **maxItems** – (Optional) The maximum number of items to return.

Type: integer. Default: 10. Maximum allowed value: 1024.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

Model training using the modeltraining command

You use the Neptune ML modeltraining command to create a model training job, check its status, stop it, or list all active model-training jobs.

Creating a model-training job using the Neptune ML modeltraining command

A Neptune ML modeltraining command for creating a completely new job looks like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltraining
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
}'
```

A Neptune ML modeltraining command for creating an update job for incremental model training looks like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltraining
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
  "previousModelTrainingJobId" : "(the job ID of a completed model-training job to
update)",
}'
```

A Neptune ML modeltraining command for creating a new job with user provided custom model implementation looks like:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltraining
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
  "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
  "modelName": "custom",
  "customModelTrainingParameters" : {
    "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
module)",
    "trainingEntryPointScript": "(your training script entry-point name in the Python
module)",
    "transformEntryPointScript": "(your transform script entry-point name in the
Python module)"
  }
}'
```

Parameters for modeltraining job creation

- **id** – (Optional) A unique identifier for the new job.
Type: string. *Default:* An autogenerated UUID.
- **dataProcessingJobId** – (Required) The job Id of the completed data-processing job that has created the data that the training will work with.
Type: string.
- **trainModelsS3Location** – (Required) The location in Amazon S3 where the model artifacts are to be stored.
Type: string.
- **previousModelTrainingJobId** – (Optional) The job ID of a completed model-training job that you want to update incrementally based on updated data.
Type: string. *Default:* none.
- **sagemakerIamRoleArn** – (Optional) The ARN of an IAM role for SageMaker execution.
Type: string. *Note:* This must be listed in your DB cluster parameter group or an error will occur.
- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.
Type: string. *Note:* This must be listed in your DB cluster parameter group or an error will occur.
- **modelName** – (Optional) The model type for training. By default the ML model is automatically based on the `modelType` used in data processing, but you can specify a different model type here.
Type: string. *Default:* `rgcn` for heterogeneous graphs and `kge` for knowledge graphs. *Valid values:* For heterogeneous graphs: `rgcn`. For `kge` graphs: `transe`, `distmult`, or `rotate`. For a custom model implementation: `custom`.
- **baseProcessingInstanceType** – (Optional) The type of ML instance used in preparing and managing training of ML models.
Type: string. *Note:* This is a CPU instance chosen based on memory requirements for processing the training data and model. See [Selecting an instance for model training and model transform \(p. 539\)](#).
- **trainingInstanceType** – (Optional) The type of ML instance used for model training. All Neptune ML models support CPU, GPU, and multiGPU training.
Type: string. *Default:* `m1.p3.2xlarge`.
Note: Choosing the right instance type for training depends on the task type, graph size, and your budget. See [Selecting an instance for model training and model transform \(p. 539\)](#).
- **trainingInstanceVolumeSizeInGB** – (Optional) The disk volume size of the training instance. Both input data and the output model are stored on disk, so the volume size must be large enough to hold both data sets.
Type: integer. *Default:* 0.
Note: If not specified or 0, Neptune ML selects a disk volume size based on the recommendation generated in the data processing step. See [Selecting an instance for model training and model transform \(p. 539\)](#).
- **trainingTimeOutInSeconds** – (Optional) Timeout in seconds for the training job.
Type: integer. *Default:* 86,400 (1 day).
- **maxHPONumberOfTrainingJobs** – Maximum total number of training jobs to start for the hyperparameter tuning job.

Type: integer. Default: 2.

Note: Neptune ML automatically tunes the hyper-parameters of the machine learning model. To obtain a model that performs well, use at least 10 jobs (in other words, set `maxHPONumberOfTrainingJobs` to 10). In general, the more tuning runs, the better the results.

- **maxHPOParallelTrainingJobs** – Maximum number of parallel training jobs to start for the hyperparameter tuning job.

Type: integer. Default: 2.

Note: The number of parallel jobs you can run is limited by the available resources on your training instance.

- **subnets** – (Optional) The IDs of the subnets in the Neptune VPC.

Type: list of strings. Default: none.

- **securityGroupIds** – (Optional) The VPC security group IDs.

Type: list of strings. Default: none.

- **volumeEncryptionKMSKey** – (Optional) The AWS Key Management Service (AWS KMS) key that SageMaker uses to encrypt data on the storage volume attached to the ML compute instances that run the training job.

Type: string. Default: none.

- **s3OutputEncryptionKMSKey** – (Optional) The AWS Key Management Service (AWS KMS) key that SageMaker uses to encrypt the output of the processing job.

Type: string. Default: none.

- **enableManagedSpotTraining** – (Optional) Optimizes the cost of training machine learning models by using Amazon Elastic Compute Cloud spot instances. For more information, see [Managed Spot Training in Amazon SageMaker](#).

Type: Boolean. Default: False.

- **customModelTrainingParameters** – (Optional) The configuration for custom model training. This is a JSON object with the following fields:
 - **sourceS3DirectoryPath** – (Required) The path to the Amazon S3 location where the Python module implementing your model is located. This must point to a valid existing Amazon S3 location that contains, at a minimum, a training script, a transform script, and a `model-hpo-configuration.json` file.
 - **trainingEntryPointScript** – (Optional) The name of the entry point in your module of a script that performs model training and takes hyperparameters as command-line arguments, including fixed hyperparameters.

Default: `training.py`.

- **transformEntryPointScript** – (Optional) The name of the entry point in your module of a script that should be run after the best model from the hyperparameter search has been identified, to compute the model artifacts necessary for model deployment. It should be able to run with no command-line arguments.

Default: `transform.py`.

Getting the status of a model-training job using the Neptune ML `modeltraining` command

A sample Neptune ML `modeltraining` command for the status of a job looks like this:

```
curl -s \  
  "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)" \  
  | python -m json.tool
```

Parameters for modeltraining job status

- **id** – (Required) The unique identifier of the model-training job.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

Stopping a model-training job using the Neptune ML modeltraining command

A sample Neptune ML modeltraining command for stopping a job looks like this:

```
curl -s \  
  -X DELETE "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)"
```

Or this:

```
curl -s \  
  -X DELETE "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)?clean=true"
```

Parameters for modeltraining stop job

- **id** – (Required) The unique identifier of the model-training job.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

- **clean** – (Optional) This flag specifies that all Amazon S3 artifacts should be deleted when the job is stopped.

Type: Boolean. Default: FALSE.

Listing active model-training jobs using the Neptune ML modeltraining command

A sample Neptune ML modeltraining command for listing active jobs looks like this:

```
curl -s "https://(your Neptune endpoint)/ml/modeltraining" | python -m json.tool
```

Or this:

```
curl -s "https://(your Neptune endpoint)/ml/modeltraining?maxItems=3" | python -m json.tool
```

Parameters for modeltraining list jobs

- **maxItems** – (Optional) The maximum number of items to return.

Type: integer. Default: 10. Maximum allowed value: 1024.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

Model transform using the modeltransform command

You use the Neptune ML modeltransform command to create a model transform job, check its status, stop it, or list all active model-transform jobs.

Creating a model-transform job using the Neptune ML modeltransform command

A Neptune ML modeltransform command for creating an incremental transform job, without model retraining, looks like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltransform
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-transform job ID)",
  "dataProcessingJobId" : "(the job-id of a completed data-processing job)",
  "mlModelTrainingJobId" : "(the job-id of a completed model-training job)",
  "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-transform"
}'
```

A Neptune ML modeltransform command for creating a job from a completed SageMaker training job looks like this:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltransform
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-transform job ID)",
  "trainingJobName" : "(name of a completed SageMaker training job)",
  "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-transform",
  "baseProcessingInstanceType" : ""
}'
```

A Neptune ML modeltransform command for creating a job that uses a custom model implementation looks like:

```
curl \
-X POST https://(your Neptune endpoint)/ml/modeltransform
-H 'Content-Type: application/json' \
-d '{
  "id" : "(a unique model-training job ID)",
  "trainingJobName" : "(name of a completed SageMaker training job)",
  "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-transform/"
  "customModelTransformParameters" : {
    "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python module)",
    "transformEntryPointScript": "(your transform script entry-point name in the Python module)"
  }
}'
```

Parameters for `modeltransform` job creation

- **id** – (Optional) A unique identifier for the new job.

Type: string. Default: An autogenerated UUID.

- **dataProcessingJobId** – The job Id of a completed data-processing job.

Type: string.

Note: You must include either both `dataProcessingJobId` and `mlModelTrainingJobId`, or `trainingJobName`.

- **mlModelTrainingJobId** – The job Id of a completed model-training job.

Type: string.

Note: You must include either both `dataProcessingJobId` and `mlModelTrainingJobId`, or `trainingJobName`.

- **trainingJobName** – The name of a completed SageMaker training job.

Type: string.

Note: You must include either both the `dataProcessingJobId` and the `mlModelTrainingJobId` parameters, or the `trainingJobName` parameter.

- **sagemakerIamRoleArn** – (Optional) The ARN of an IAM role for SageMaker execution.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

- **customModelTransformParameters** – (Optional) Configuration information for a model transform using a custom model. The `customModelTransformParameters` object contains the following fields, which must have values compatible with the saved model parameters from the training job:

- **sourceS3DirectoryPath** – (Required) The path to the Amazon S3 location where the Python module implementing your model is located. This must point to a valid existing Amazon S3 location that contains, at a minimum, a training script, a transform script, and a `model-hpo-configuration.json` file.

- **transformEntryPointScript** – (Optional) The name of the entry point in your module of a script that should be run after the best model from the hyperparameter search has been identified, to compute the model artifacts necessary for model deployment. It should be able to run with no command-line arguments.

Default: `transform.py`.

- **baseProcessingInstanceType** – (Optional) The type of ML instance used in preparing and managing training of ML models.

Type: string. Note: This is a CPU instance chosen based on memory requirements for processing the transform data and model. See [Selecting an instance for model training and model transform \(p. 539\)](#).

- **baseProcessingInstanceVolumeSizeInGB** – (Optional) The disk volume size of the training instance. Both input data and the output model are stored on disk, so the volume size must be large enough to hold both data sets.

Type: integer. Default: 0.

Note: If not specified or 0, Neptune ML selects a disk volume size based on the recommendation generated in the data processing step. See [Selecting an instance for model training and model transform \(p. 539\)](#).

- **subnets** – (Optional) The IDs of the subnets in the Neptune VPC.

Type: list of strings. *Default:* none.

- **securityGroupIds** – (Optional) The VPC security group IDs.

Type: list of strings. *Default:* none.

- **volumeEncryptionKMSKey** – (Optional) The AWS Key Management Service (AWS KMS) key that SageMaker uses to encrypt data on the storage volume attached to the ML compute instances that run the transform job.

Type: string. *Default:* none.

- **s3OutputEncryptionKMSKey** – (Optional) The AWS Key Management Service (AWS KMS) key that SageMaker uses to encrypt the output of the processing job.

Type: string. *Default:* none.

Getting the status of a model-transform job using the Neptune ML `modeltransform` command

A sample Neptune ML `modeltransform` command for the status of a job looks like this:

```
curl -s \  
  "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)" \  
  | python -m json.tool
```

Parameters for `modeltransform` job status

- **id** – (Required) The unique identifier of the model-transform job.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. *Note:* This must be listed in your DB cluster parameter group or an error will occur.

Stopping a model-transform job using the Neptune ML `modeltransform` command

A sample Neptune ML `modeltransform` command for stopping a job looks like this:

```
curl -s \  
  -X DELETE "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)"
```

Or this:

```
curl -s \  
  -X DELETE "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)?clean=true"
```


Parameters for `modeltransform stop job`

- **id** – (Required) The unique identifier of the model-transform job.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

- **clean** – (Optional) This flag specifies that all Amazon S3 artifacts should be deleted when the job is stopped.

Type: Boolean. Default: `FALSE`.

Listing active model-transform jobs using the Neptune ML `modeltransform` command

A sample Neptune ML `modeltransform` command for listing active jobs looks like this:

```
curl -s "https://(your Neptune endpoint)/ml/modeltransform" | python -m json.tool
```

Or this:

```
curl -s "https://(your Neptune endpoint)/ml/modeltransform?maxItems=3" | python -m json.tool
```

Parameters for `modeltransform list jobs`

- **maxItems** – (Optional) The maximum number of items to return.

Type: integer. Default: 10. Maximum allowed value: 1024.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role that provides Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will occur.

Managing inference endpoints using the endpoints command

You use the Neptune ML endpoints command to create an inference endpoint, check its status, delete it, or list existing inference endpoints.

Creating an inference endpoint using the Neptune ML endpoints command

A Neptune ML endpoints command for creating an inference endpoint from a model created by a training job looks like this:

```
curl \  
-X POST https://(your Neptune endpoint)/ml/endpoints  
-H 'Content-Type: application/json' \  
-d '{  
  "id" : "(a unique ID for the new endpoint)",  
  "mlModelTrainingJobId": "(the model-training job-id of a completed job)"  
}'
```

A Neptune ML endpoints command for updating an existing inference endpoint from a model created by a training job looks like this:

```
curl \  
-X POST https://(your Neptune endpoint)/ml/endpoints  
-H 'Content-Type: application/json' \  
-d '{  
  "id" : "(a unique ID for the new endpoint)",  
  "update" : "true",  
  "mlModelTrainingJobId": "(the model-training job-id of a completed job)"  
}'
```

A Neptune ML endpoints command for creating an inference endpoint from a model created by a model-transform job looks like this:

```
curl \  
-X POST https://(your Neptune endpoint)/ml/endpoints  
-H 'Content-Type: application/json' \  
-d '{  
  "id" : "(a unique ID for the new endpoint)",  
  "mlModelTransformJobId": "(the model-training job-id of a completed job)"  
}'
```

A Neptune ML endpoints command for updating an existing inference endpoint from a model created by a model-transform job looks like this:

```
curl \  
-X POST https://(your Neptune endpoint)/ml/endpoints  
-H 'Content-Type: application/json' \  
-d '{  
  "id" : "(a unique ID for the new endpoint)",  
  "update" : "true",  
  "mlModelTransformJobId": "(the model-training job-id of a completed job)"  
}'
```

Parameters for endpoints inference endpoint creation

- **id** – (Optional) A unique identifier for the new inference endpoint.
Type: string. *Default:* An autogenerated timestamped name.
- **mlModelTrainingJobId** – The job Id of the completed model-training job that has created the model that the inference endpoint will point to.
Type: string.
Note: You must supply either the `mlModelTrainingJobId` or the `mlModelTransformJobId`.
- **mlModelTransformJobId** – The job Id of the completed model-transform job.
Type: string.
Note: You must supply either the `mlModelTrainingJobId` or the `mlModelTransformJobId`.
- **update** – (Optional) If present, this parameter indicates that this is an update request.
Type: Boolean. *Default:* `false`
Note: You must supply either the `mlModelTrainingJobId` or the `mlModelTransformJobId`.
- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role providing Neptune access to SageMaker and Amazon S3 resources.
Type: string. *Note:* This must be listed in your DB cluster parameter group or an error will be thrown.
- **modelName** – (Optional) Model type for training. By default the ML model is automatically based on the `modelType` used in data processing, but you can specify a different model type here.
Type: string. *Default:* `rgcn` for heterogeneous graphs and `kge` for knowledge graphs. *Valid values:* For heterogeneous graphs: `rgcn`. For knowledge graphs: `kge`, `transe`, `distmult`, or `rotate`.
- **instanceType** – (Optional) The type of ML instance used for online servicing.
Type: string. *Default:* `m1.m5.xlarge`.
Note: Choosing the ML instance for an inference endpoint depends on the task type, the graph size, and your budget. See [Selecting an instance for an inference endpoint \(p. 539\)](#).
- **instanceCount** – (Optional) The minimum number of Amazon EC2 instances to deploy to an endpoint for prediction.
Type: integer. *Default:* 1.
- **volumeEncryptionKMSKey** – (Optional) The AWS Key Management Service (AWS KMS) key that SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the endpoints.
Type: string. *Default:* `none`.

Getting the status of an inference endpoint using the Neptune ML endpoints command

A sample Neptune ML endpoints command for the status of an instance endpoint looks like this:

```
curl -s \  
  "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)" \  
  | python -m json.tool
```

Parameters for `endpoints instance-endpoint status`

- **id** – (Required) The unique identifier of the inference endpoint.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role providing Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will be thrown.

Deleting an instance endpoint using the Neptune ML `endpoints` command

A sample Neptune ML `endpoints` command for deleting an instance endpoint looks like this:

```
curl -s \  
-X DELETE "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)"
```

Or this:

```
curl -s \  
-X DELETE "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)?  
clean=true"
```

Parameters for `endpoints deleting an inference endpoint`

- **id** – (Required) The unique identifier of the inference endpoint.

Type: string.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role providing Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will be thrown.

- **clean** – (Optional) Indicates that all artifacts related to this endpoint should also be deleted.

Type: Boolean. Default: FALSE.

Listing inference endpoints using the Neptune ML `endpoints` command

A Neptune ML `endpoints` command for listing inference endpoints looks like this:

```
curl -s "https://(your Neptune endpoint)/ml/endpoints" \  
| python -m json.tool
```

Or this:

```
curl -s "https://(your Neptune endpoint)/ml/endpoints?maxItems=3" \  
| python -m json.tool
```

Parameters for `dataprocessing list inference endpoints`

- **maxItems** – (Optional) The maximum number of items to return.

Type: integer. Default: 10. Maximum allowed value: 1024.

- **neptuneIamRoleArn** – (Optional) The ARN of an IAM role providing Neptune access to SageMaker and Amazon S3 resources.

Type: string. Note: This must be listed in your DB cluster parameter group or an error will be thrown.

Neptune ML management API exceptions

All Neptune ML management API exceptions return a 400 HTTP code. After receiving any of these exceptions, the command that generated the exception should not be retried.

- **MissingParameterException** – Error message:

Required credentials are missing. Please add IAM role to the cluster or pass as a parameter to this request.

- **InvalidParameterException** – Error messages:

- Invalid ML instance type.
- Invalid ID provided. ID can be 1-48 alphanumeric characters.
- Invalid ID provided. Must contain only letters, digits, or hyphens.
- Invalid ID provided. Please check whether a resource with the given ID exists.
- Another resource with same ID already exists. Please use a new ID.
- Failed to stop the job because it has already completed or failed.

- **BadRequestException** – Error messages:

- Invalid S3 URL or incorrect S3 permissions. Please check your S3 configuration.
- Provided ModelTraining job has not completed.
- Provided SageMaker Training job has not completed.
- Provided MLDataProcessing job is not completed.
- Provided MLModelTraining job doesn't exist.
- Provided ModelTransformJob doesn't exist.
- Unable to find SageMaker resource. Please check your input.

Neptune ML limits

- Inference queries are only supported currently using the [Gremlin \(p. 262\)](#) query language.
- The types of inference currently supported are node classification, node regression, edge classification, edge regression and link prediction (see [Neptune ML capabilities \(p. 522\)](#)).
- The maximum graph size that Neptune ML can support depends on the amount of memory and storage required during [data preparation \(p. 539\)](#), [model training \(p. 539\)](#), and [inference \(p. 539\)](#).
 - The maximum size of memory of a SageMaker data-processing instance is 768 GB. As a result, the data-processing stage fails if it needs more than 768 GB of memory.
 - The maximum size of memory of a SageMaker training instance is 732 GB. As a result, the training stage fails if it needs more than 732 GB of memory.
- Neptune ML is currently available only in Regions where Neptune and the other services it depends on (such as AWS Lambda, Amazon API Gateway and Amazon SageMaker) are all supported.

There are differences in China (Beijing) and China (Ningxia) having to do with the default use of IAM authentication, as is [explained here](#) along with other differences.

- The inference endpoints launched by Neptune ML currently can only return predictions for nodes that were present in the graph during training. For property graph models, you can update the inference endpoint to provide predictions for new nodes by following the [incremental data inference workflow \(p. 537\)](#) or the [retraining workflow \(p. 538\)](#) with the new data.

For each supported task, the following nodes can appear in inference queries:

- For vertex-property prediction (node classification or node regression), all vertices used in the queries must be available in the original graph, but the property to be predicted does not need to exist on all vertices.
- For edge-property prediction (edge classification or edge regression), all incident vertices on the edges used in the queries must be available in the original graph, but the edge property to be predicted does not need to exist on all edges.
- For link prediction (source or target vertex prediction), all vertices and edge types used in the queries must be available during training. Predictions for new edge types or vertices cannot be made.
- The KGE models supported by Neptune ML only work for link prediction tasks, and the representations are specific to vertices and edge types present in the graph during training. This means that all vertices and edge types referred to in an inference query must have been present in the graph during training. Predictions for new edge types or vertices cannot be made without retraining the model.

SageMaker resource limitations

Depending on your activities and resource usage over time, you may encounter error messages saying that [you've exceeded your quota \(ResourceLimitExceeded\)](#), and you need to scale up your SageMaker resources, follow the steps in the [Request a service quota increase for SageMaker resources](#) procedure on this page to request a quota increase from AWS Support.

SageMaker resource names correspond to Neptune ML stages as follows:

- The SageMaker `ProcessingJob` is used by Neptune data processing, model training, and model transform jobs.
- The SageMaker `HyperParameterTuningJob` is used by Neptune model training jobs.
- The SageMaker `TrainingJob` is used by Neptune model training jobs.
- The SageMaker `Endpoint` is used by Neptune inference endpoints.

Monitoring Amazon Neptune Resources

Amazon Neptune supports various methods for monitoring performance and usage:

- **Instance status** – Check the health of a Neptune cluster's graph database engine, find out what version of the engine is installed, and obtain other instance-related information using the [instance status API](#) (p. 639).
- **Amazon CloudWatch** – Neptune automatically sends metrics to CloudWatch and also supports CloudWatch Alarms. For more information, see [the section called "Using CloudWatch"](#) (p. 641).
- **Audit log files** – View, download, or watch database log files using the Neptune console. For more information, see [the section called "Audit Logs with Neptune"](#) (p. 651).
- **Publishing logs to Amazon CloudWatch Logs** – You can configure a Neptune DB cluster to publish audit log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, use CloudWatch to create alarms and view metrics, and use CloudWatch Logs to store your log records in highly durable storage. See [Neptune Logs in CloudWatch Logs](#) (p. 652).
- **AWS CloudTrail** – Neptune supports API logging using CloudTrail. For more information, see [the section called "Logging Neptune API Calls with AWS CloudTrail"](#) (p. 654).
- **Event notification subscriptions** – Subscribe to Neptune events to stay informed about what is happening. For more information, see [the section called "Event Notifications"](#) (p. 656).
- **Tagging** – Use tags to add metadata to your Neptune resources and track usage based on tags. For more information, see [the section called "Tagging Neptune Resources"](#) (p. 665).

Topics

- [Check the Health Status of a Neptune Instance](#) (p. 639)
- [Monitoring Neptune Using Amazon CloudWatch](#) (p. 641)
- [Using Audit Logs with Amazon Neptune Clusters](#) (p. 651)
- [Publishing Neptune Logs to Amazon CloudWatch Logs](#) (p. 652)
- [Logging Amazon Neptune API Calls with AWS CloudTrail](#) (p. 654)
- [Using Neptune Event Notification](#) (p. 656)
- [Tagging Amazon Neptune Resources](#) (p. 665)

Check the Health Status of a Neptune Instance

Amazon Neptune provides a mechanism to check the status of the graph database on the host. It's also a good way to confirm that you are able to connect to an instance.

To check the health of an instance using `curl`:

```
curl -G https://your-neptune-endpoint:port/status
```

If the instance is healthy, the `status` command returns the following fields:

- **status** – Set to "healthy" if the instance is not experiencing problems.

If the instance is recovering from a crash or from being rebooted and there are active transactions running from the latest server shutdown, `status` is set to "recovery".

- **startTime** – Set to the UTC time at which the current server process started.
- **dbEngineVersion** – Set to the Neptune engine version running on your DB cluster.

If this engine version has been manually patched since it was released, the version number is prefixed by "Patch-".
- **role** – Set to "reader" if the instance is a read-replica, or to "writer" if the instance is the primary instance.
- **gremlin** – Contains information about the Gremlin query language available on your cluster. Specifically, it contains a `version` field that specifies the current TinkerPop version being used by the engine.
- **sparql** – Contains information about the SPARQL query language available on your cluster. Specifically, it contains a `version` field that specifies the current SPARQL version being used by the engine.
- **labMode** – Contains [Lab Mode \(p. 85\)](#) settings being used by the engine.
- **rollingBackTrxCount** – If there are transactions being rolled back, this field is set to the number of such transactions. If there are none, the field doesn't appear at all.
- **rollingBackTrxEarliestStartTime** – Set to the start time of the earliest transaction being rolled back. If no transactions are being rolled back, the field doesn't appear at all.
- **features** – Contains status information about the features enabled on your DB cluster:
 - **lookupCache** – The current status of the [Lookup cache \(p. 64\)](#). This field only appears on R5d instance types, since those are the only instances where a lookup cache can exist. The field is a JSON object in the form:

```
"lookupCache": {  
  "status": "current lookup cache status"  
}
```

On an R5d instance:

- If the lookup cache is enabled, the status is listed as "Available".
- If the lookup cache has been disabled, the status is listed as "Disabled".
- If the disk limit has been reached on the instance, the status is listed as "Read Only Mode - Storage Limit Reached".
- **resultCache** – The current status of the [Caching query results \(p. 288\)](#). This field is a JSON object in the form:

```
"resultCache": {  
  "status": "current results cache status"  
}
```

- If the results cache has been enabled, the status is listed as "Available".
- If the cache is disabled, the status is listed as "Disabled".
- **IAMAuthentication** – Specifies whether or not AWS Identity and Access Management (IAM) authentication has been enabled on your DB cluster:
 - If IAM authentication been enabled, the status is listed as "enabled".
 - If IAM authentication is disabled, the status is listed as "disabled".
- **Streams** – Specifies whether or not Neptune streams have been enabled on your DB cluster:
 - If streams are enabled, the status is listed as "enabled".
 - If streams are disabled, the status is listed as "disabled".
- **AuditLog** – Specifies whether or the audit log has been enabled on your DB cluster:
 - If the enabled, the status is listed as "enabled".
 - If audit log is disabled, the status is listed as "disabled".

- **settings** – Contains information about the current settings on your DB cluster. For example, contains the current cluster query timeout setting:
 - `clusterQueryTimeoutInMs` – Specifies the current query timeout setting on your cluster.

Example of the output from the instance status command

The following is an example of the output from the instance status command, (in this case, run on an R5d instance):

```
{
  "status": "healthy",
  "startTime": "Thu Jun 17 17:11:09 UTC 2021",
  "dbEngineVersion": "development",
  "role": "writer",
  "gremlin": {
    "version": "tinkerpop-3.4.10"
  },
  "sparql": {
    "version": "sparql-1.1"
  },
  "labMode": {
    "ObjectIndex": "disabled",
    "DFEQueryEngine": "disabled",
    "ReadWriteConflictDetection": "enabled"
  },
  "features": {
    "LookupCache": {
      "status": "Available"
    },
    "ResultCache": {
      "status": "Disabled"
    },
    "IAMAuthentication": "disabled",
    "Streams": "disabled",
    "AuditLog": "disabled"
  },
  "settings": {
    "clusterQueryTimeoutInMs": "120000"
  }
}
```

If there is a problem with the instance, the status command returns the `HTTP 500` error code. If the host is unreachable, the request times out. Ensure that you are accessing the instance from within the virtual private cloud (VPC), and that your security groups allow you access to it.

Monitoring Neptune Using Amazon CloudWatch

Amazon Neptune and Amazon CloudWatch are integrated so that you can gather and analyze performance metrics. You can monitor these metrics using the CloudWatch console, the AWS Command Line Interface (AWS CLI), or the CloudWatch API.

CloudWatch also lets you set alarms so that you can be notified if a metric value breaches a threshold that you specify. You can even set up CloudWatch Events to take corrective action if a breach occurs. For more information about using CloudWatch and alarms, see the [CloudWatch Documentation](#).

Topics

- [Viewing CloudWatch Data \(Console\) \(p. 642\)](#)
- [Viewing CloudWatch Data \(AWS CLI\) \(p. 642\)](#)
- [Viewing CloudWatch Data \(API\) \(p. 642\)](#)
- [Using CloudWatch to monitor DB instance performance in Neptune \(p. 643\)](#)
- [Neptune CloudWatch Metrics \(p. 644\)](#)
- [Neptune CloudWatch Dimensions \(p. 650\)](#)

Viewing CloudWatch Data (Console)

To view CloudWatch data for a Neptune cluster (console)

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. In the **All Metrics** pane, choose **Neptune**, and then choose **DBClusterIdentifier**.
4. In the upper pane, scroll down to view the full list of metrics for your cluster. The available Neptune metric options appear in the **Viewing** list.

To select or deselect an individual metric, in the results pane, select the check box next to the resource name and metric. Graphs showing the metrics for the selected items appear at the bottom of the console. To learn more about CloudWatch graphs, see [Graph Metrics](#) in the *Amazon CloudWatch User Guide*.

Viewing CloudWatch Data (AWS CLI)

To view CloudWatch data for a Neptune cluster (AWS CLI)

1. Install the AWS CLI. For instructions, see the [AWS Command Line Interface User Guide](#).
2. Use the AWS CLI to fetch information. The relevant CloudWatch parameters for Neptune are listed in [Neptune CloudWatch Metrics \(p. 644\)](#).

The following example retrieves CloudWatch metrics for the number of Gremlin requests per second for the `gremlin-cluster` cluster.

```
aws cloudwatch get-metric-statistics \  
  --namespace AWS/Neptune --metric-name GremlinRequestsPerSec \  
  --dimensions Name=DBClusterIdentifier,Value=gremlin-cluster \  
  --start-time 2018-03-03T00:00:00Z --end-time 2018-03-04T00:00:00Z \  
  --period 60 --statistics=Average
```

Viewing CloudWatch Data (API)

CloudWatch also supports a `Query` action so that you can request information programmatically. For more information, see the [CloudWatch Query API documentation](#) and [Amazon CloudWatch API Reference](#).

When a CloudWatch action requires a parameter that is specific to Neptune monitoring, such as `MetricName`, use the values listed in [Neptune CloudWatch Metrics \(p. 644\)](#).

The following example shows a low-level CloudWatch request, using the following parameters:

- `Statistics.member.1 = Average`
- `Dimensions.member.1 = DBClusterIdentifier=gremlin-cluster`
- `Namespace = AWS/Neptune`
- `StartTime = 2013-11-14T00:00:00Z`
- `EndTime = 2013-11-16T00:00:00Z`
- `Period = 60`
- `MetricName = GremlinRequestsPerSec`

Here is what the CloudWatch request looks like. However, this is just to show the form of the request; you must construct your own request based on your metrics and timeframe.

```
https://monitoring.amazonaws.com/  
?SignatureVersion=2  
&Action=GremlinRequestsPerSec  
&Version=2010-08-01  
&StartTime=2018-03-03T00:00:00  
&EndTime=2018-03-04T00:00:00  
&Period=60  
&Statistics.member.1=Average  
&Dimensions.member.1=DBClusterIdentifier=gremlin-cluster  
&Namespace=AWS/Neptune  
&MetricName=GremlinRequests  
&Timestamp=2018-03-04T17%3A48%3A21.746Z  
&AWSAccessKeyId=AWS Access Key ID;  
&Signature=signature
```

Using CloudWatch to monitor DB instance performance in Neptune

You can use CloudWatch metrics in Neptune to monitor what is happening on your DB instances and keep track of query latency as observed by the client. The following metrics are particularly useful:

- **CPUUtilization** – Shows the percentage of CPU utilization.
- **VolumeWriteIOPs** – Shows the average number of disk I/O writes to the cluster volume, reported at 5-minute intervals.
- **MainRequestQueuePendingRequests** – Shows the number of requests waiting in the input queue pending execution.

You can also find out how many requests are pending on the server by using the [Gremlin query status endpoint \(p. 284\)](#) with the `includeWaiting` parameter. This will give you the status of all waiting queries.

The following indicators can help you adjust your Neptune provisioning and query strategies to improve efficiency and performance:

- Consistent latency, high `CPUUtilization`, high `VolumeWriteIOPs` and low `MainRequestQueuePendingRequests` together show that the server is actively engaged processing concurrent write requests at a sustainable rate, with little I/O wait.
- Consistent latency, low `CPUUtilization`, low `VolumeWriteIOPs` and no `MainRequestQueuePendingRequests` together show that you have excess capacity on the primary DB instance for processing write requests.
- High `CPUUtilization` and high `VolumeWriteIOPs` but variable latency and `MainRequestQueuePendingRequests` together show that you are sending more work than the

server can process in a given interval. Consider creating or resizing batch requests so as to do the same amount of work with less transactional overhead and/or scaling the primary instance up to increase the number of query threads capable of processing write requests concurrently.

- Low `CPUUtilization` with high `VolumeWriteIOPs` mean that query threads are waiting for I/O operations to the storage layer to complete. If you see variable latencies and some increase in `MainRequestQueuePendingRequests`, consider creating or resizing batch requests so as to do the same amount of work with less transactional overhead.

Neptune CloudWatch Metrics

Note

Amazon Neptune sends metrics to CloudWatch only when they have a non-zero value. For all Neptune metrics, the aggregation granularity is 5 minutes.

Topics

- [Neptune CloudWatch Metrics \(p. 644\)](#)
- [CloudWatch Metrics That Are Now Deprecated in Neptune \(p. 647\)](#)

Neptune CloudWatch Metrics

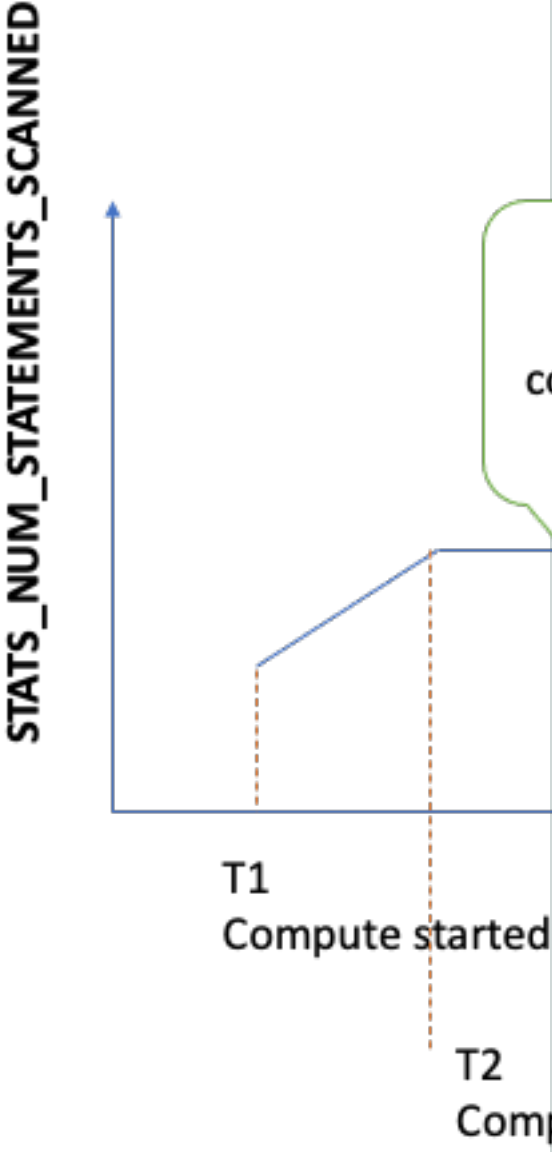
The following table lists the CloudWatch metrics that Neptune supports.

Note

All cumulative metrics are reset to zero whenever the server restarts, whether for maintenance, a reboot, or recovering from a crash.

Metric	Description
<code>BackupRetentionPeriodStorageUsed</code>	The total amount of backup storage, in bytes, used to support from the Neptune DB cluster's backup retention window. Included in the total reported by the <code>TotalBackupStorageBilled</code> metric.
<code>BufferCacheHitRatio</code>	The percentage of requests that are served by the buffer cache. This metric can be useful in diagnosing query latency, because cache misses induce significant latency. If the cache hit ratio is below 99.9, consider upgrading the instance type to cache more data in memory.
<code>ClusterReplicaLag</code>	For a read replica, the amount of lag when replicating updates from the primary instance, in milliseconds.
<code>ClusterReplicaLagMaximum</code>	The maximum amount of lag between the primary instance and each Neptune DB instance in the DB cluster, in milliseconds.
<code>ClusterReplicaLagMinimum</code>	The minimum amount of lag between the primary instance and each Neptune DB instance in the DB cluster, in milliseconds.
<code>CPUUtilization</code>	The percentage of CPU utilization.

Metric	Description
EngineUptime	The amount of time that the instance has been running, in seconds.
FreeableMemory	The amount of available random access memory, in bytes.
GremlinRequestsPerSec	Number of requests per second to the Gremlin engine.
GremlinWebSocketOpenConnections	The number of open WebSocket connections to Neptune.
LoaderRequestsPerSec	Number of loader requests per second.
MainRequestQueuePendingRequests	The number of requests waiting in the input queue pending execution. Neptune starts throttling requests when they exceed the maximum queue capacity.
NetworkReceiveThroughput	The incoming (Receive) network traffic on the DB instance, including both customer database traffic and Neptune traffic used for monitoring and replication, in bytes/second.
NetworkThroughput	The amount of network throughput both received from and transmitted to clients by each instance in the Neptune DB cluster, in bytes per second. This throughput doesn't include network traffic between instances in the DB cluster and the cluster volume.
NetworkTransmitThroughput	The outgoing (Transmit) network traffic on the DB instance, including both customer database traffic and Neptune traffic used for monitoring and replication, in bytes/second.
NumTxCommitted	The number of transactions successfully committed per second.
NumTxOpened	The number of transactions opened on the server per second.
NumTxRolledBack	The number of transactions per second rolled back on the server because of errors.
SnapshotStorageUsed	The total amount of backup storage consumed by all snapshots for a Neptune DB cluster outside its backup retention window, in bytes. Included in the total reported by the TotalBackupStorageBilled metric.
SparqlRequestsPerSec	The number of requests per second to the SPARQL engine.

Metric	Description
StatsNumStatementsScanned	<p>The total number of statements scanned for DFE statistics (p. 88) since the server started.</p> <p>Every time statistics computation is triggered, this number increases, but when no computation is happening, it remains static. As a result, if you graph it over time, you can tell when computation happened and when it didn't:</p>  <p>The graph illustrates the behavior of the <code>StatsNumStatementsScanned</code> metric. The vertical axis represents the number of statements scanned, and the horizontal axis represents time. At time <code>T1</code>, labeled "Compute started", the metric begins to increase linearly. At time <code>T2</code>, labeled "Compute ended", the metric reaches a constant value and remains static thereafter. A callout box labeled "No compute" highlights the period where the metric is constant, indicating that no statistics computation is occurring.</p>

Metric	Description
	<p>By looking at the slope of the graph in periods where the metric is increasing, you can also tell how quickly the computation was going.</p> <p>If there is no such metric, it means that the statistics feature is disabled on your DB cluster, or that the engine version you're running doesn't have the statistics feature. If the metric value is zero, it means that no statistics computation has occurred.</p>
TotalBackupStorageBilled	The total amount of backup storage for which you are billed for a given Neptune DB cluster, in bytes. Includes the backup storage measured by the BackupRetentionPeriodStorageUsed and SnapshotStorageUsed metrics.
TotalRequestsPerSec	The total number of requests per second to the server from all sources.
TotalClientErrorsPerSec	The total number per second of requests that errored out because of client-side issues.
TotalServerErrorsPerSec	The total number per second of requests that errored out on the server because of internal failures.
VolumeBytesUsed	The total amount of storage allocated to your Neptune DB cluster, in bytes. This is the amount of storage for which you are billed. It is the maximum amount of storage allocated to your DB cluster at any point in its existence, not the amount you are currently using (see Neptune storage billing (p. 75)).
VolumeReadIOPs	The average number of billed read I/O operations from a cluster volume, reported at 5-minute intervals. Billed read operations are calculated at the cluster volume level, aggregated from all instances in the Neptune DB cluster, and then reported at 5-minute intervals.
VolumeWriteIOPs	The average number of write disk I/O operations to the cluster volume, reported at 5-minute intervals.

CloudWatch Metrics That Are Now Deprecated in Neptune

Use of these Neptune metrics has now been deprecated. They are still supported, but may be eliminated in the future as new and better metrics become available.

Metric	Description
GremlinHttp1xx	Number of HTTP 1xx responses for the Gremlin endpoint per second.

Metric	Description
	We recommend that you use the new <code>Http1xx</code> combined metric instead.
<code>GremlinHttp2xx</code>	Number of HTTP 2xx responses for the Gremlin endpoint per second. We recommend that you use the new <code>Http2xx</code> combined metric instead.
<code>GremlinHttp4xx</code>	Number of HTTP 4xx errors for the Gremlin endpoint per second. We recommend that you use the new <code>Http4xx</code> combined metric instead.
<code>GremlinHttp5xx</code>	Number of HTTP 5xx errors for the Gremlin endpoint per second. We recommend that you use the new <code>Http5xx</code> combined metric instead.
<code>GremlinErrors</code>	Number of errors in Gremlin traversals.
<code>GremlinRequests</code>	Number of requests to Gremlin engine.
<code>GremlinWebSocketSuccess</code>	Number of successful WebSocket connections to the Gremlin endpoint per second.
<code>GremlinWebSocketClientErrors</code>	Number of WebSocket client errors on the Gremlin endpoint per second.
<code>GremlinWebSocketServerErrors</code>	Number of WebSocket server errors on the Gremlin endpoint per second.
<code>GremlinWebSocketAvailableConnections</code>	Number of potential WebSocket connections currently available.
<code>Http100</code>	Number of HTTP 100 responses for the endpoint per second. We recommend that you use the new <code>Http1xx</code> combined metric instead.
<code>Http101</code>	Number of HTTP 101 responses for the endpoint per second. We recommend that you use the new <code>Http1xx</code> combined metric instead.
<code>Http1xx</code>	Number of HTTP 1xx responses for the endpoint per second.
<code>Http200</code>	Number of HTTP 200 responses for the endpoint per second. We recommend that you use the new <code>Http2xx</code> combined metric instead.

Metric	Description
<code>Http2xx</code>	Number of HTTP 2xx responses for the endpoint per second.
<code>Http400</code>	Number of HTTP 400 errors for the endpoint per second. We recommend that you use the new <code>Http4xx</code> combined metric instead.
<code>Http403</code>	Number of HTTP 403 errors for the endpoint per second. We recommend that you use the new <code>Http4xx</code> combined metric instead.
<code>Http405</code>	Number of HTTP 405 errors for the endpoint per second. We recommend that you use the new <code>Http4xx</code> combined metric instead.
<code>Http413</code>	Number of HTTP 413 errors for the endpoint per second. We recommend that you use the new <code>Http4xx</code> combined metric instead.
<code>Http429</code>	Number of HTTP 429 errors for the endpoint per second. We recommend that you use the new <code>Http4xx</code> combined metric instead.
<code>Http4xx</code>	Number of HTTP 4xx errors for the endpoint per second.
<code>Http500</code>	Number of HTTP 500 errors for the endpoint per second. We recommend that you use the new <code>Http5xx</code> combined metric instead.
<code>Http501</code>	Number of HTTP 501 errors for the endpoint per second. We recommend that you use the new <code>Http5xx</code> combined metric instead.
<code>Http5xx</code>	Number of HTTP 5xx errors for the endpoint per second.
<code>LoaderErrors</code>	Number of errors from Loader requests.
<code>LoaderRequests</code>	Number of Loader Requests.

Metric	Description
<code>SparqlHttp1xx</code>	Number of HTTP 1xx responses for the SPARQL endpoint per second. We recommend that you use the new <code>Http1xx</code> combined metric instead.
<code>SparqlHttp2xx</code>	Number of HTTP 2xx responses for the SPARQL endpoint per second. We recommend that you use the new <code>Http2xx</code> combined metric instead.
<code>SparqlHttp4xx</code>	Number of HTTP 4xx errors for the SPARQL endpoint per second. We recommend that you use the new <code>Http4xx</code> combined metric instead.
<code>SparqlHttp5xx</code>	Number of HTTP 5xx errors for the SPARQL endpoint per second. We recommend that you use the new <code>Http5xx</code> combined metric instead.
<code>SparqlErrors</code>	Number of errors in the SPARQL queries.
<code>SparqlRequests</code>	Number of requests to the SPARQL engine.
<code>StatusErrors</code>	Number of errors from the status endpoint.
<code>StatusRequests</code>	Number of requests to the status endpoint.

Neptune CloudWatch Dimensions

The metrics for Amazon Neptune are qualified by the values for the account, graph name, or operation. You can use the Amazon CloudWatch console to retrieve Neptune data along with any of the dimensions in the following table.

Dimension	Description
<code>DBInstanceIdentifier</code>	Filters the data you request for a specific database instance within a cluster.
<code>DBClusterIdentifier</code>	Filters the data you request for a specific Neptune DB cluster.
<code>DBClusterIdentifier, Role</code>	Filters the data you request for a specific Neptune DB cluster, aggregating the metric by instance role (WRITER/READER). For example, you can aggregate metrics for all READER instances that belong to a cluster.
<code>DBClusterIdentifier, EngineName</code>	Filters the data by the cluster. The engine name for all Neptune instances is <code>neptune</code> .

Dimension	Description
DatabaseClass	Filters the data you request for all instances in a database class. For example, you can aggregate metrics for all instances that belong to the database class db.r4.large
EngineName	The engine name for all Neptune instances is neptune.

Using Audit Logs with Amazon Neptune Clusters

To audit Amazon Neptune DB cluster activity, enable the collection of audit logs by setting a DB cluster parameter. When audit logs are enabled, you can use it to log any combination of supported events. You can view or download the audit logs to review them.

Enabling Neptune Audit Logs

Use the `neptune_enable_audit_log` parameter to enable (1) or disable (0) audit logs.

Set this parameter in the parameter group that is used by your DB cluster. You can use the procedure shown in [Editing a DB Cluster Parameter Group or DB Parameter Group \(p. 407\)](#) to modify the parameter using the AWS Management Console, or use the `modify-db-cluster-parameter-group` AWS CLI command or the `ModifyDBClusterParameterGroup` API command to modify the parameter programmatically.

You must restart your DB cluster after modifying this parameter in order to apply the change.

Viewing Neptune Audit Logs Using the Console

You can view and download the audit logs by using the AWS Management Console. On the **Instances** page, choose the DB instance to show its details, and then scroll to the **Logs** section.

To download a log file, select that file in the **Logs** section, and then choose **Download**.

Neptune Audit Log Details

Log files are in UTF-8 format. Logs are written in multiple files, the number of which varies based on the instance size. To see the latest events, you might have to review all the audit log files.

Log entries are not in sequential order. You can use the `timestamp` value for ordering them.

Log files are rotated when they reach 100 MB in aggregate. This limit is not configurable.

The audit log files include the following comma-delimited information in rows, in the following order:

Field	Description
Timestamp	The Unix timestamp for the logged event with microsecond precision.
ServerHost	The hostname or IP of the instance that the event is logged for.
ClientHost	The hostname or IP that the user connected from.

Field	Description
ConnectionType	The connection type. Can be <code>Websocket</code> , <code>HTTP_POST</code> , or <code>HTTP_GET</code> .
Caller's IAM ARN	The ARN of the IAM user or IAM role used to sign the request. Empty if IAM authentication is disabled. Its format is: <code>arn:partition:service:region:account:resource</code> For example: <code>arn:aws:iam::123456789012:user/Anna</code> <code>arn:aws:sts::123456789012:assumed-role/AWSNeptuneNotebookRole/SageMaker</code>
Auth Context	Contains a serialized JSON object that has authentication information. The field <code>authenticationSucceeded</code> is <code>True</code> if the user was authenticated. Empty if IAM authentication is disabled.
HTTPHeader	The HTTP header information. Can contain a query. Empty for WebSocket connections.
Payload	The Gremlin or SPARQL query.

Publishing Neptune Logs to Amazon CloudWatch Logs

You can configure a Neptune DB cluster to publish audit log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage.

To publish logs to CloudWatch Logs, audit logs must be explicitly enabled (see [Enable Audit Logs \(p. 651\)](#)).

Note

Be aware of the following:

- You can't publish logs to CloudWatch Logs for the China (Beijing) or China (Ningxia) region.
- If exporting log data is disabled, Neptune doesn't delete existing log groups or log streams. If exporting log data is disabled, existing log data remains available in CloudWatch Logs, depending on log retention, and you still incur charges for stored audit log data. You can delete log streams and log groups using the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API.

Using the Console to Publish Neptune Logs to CloudWatch Logs

To publish Neptune logs to CloudWatch Logs from the console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.

2. In the navigation pane, choose **Databases**.
3. Choose the Neptune DB cluster that you want to publish the log data for.
4. For **Actions**, choose **Modify**.
5. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.
6. Choose **Continue**, and then choose **Modify DB Cluster** on the summary page.

Using the CLI to Publish Neptune Logs to CloudWatch Logs

You can create a new DB cluster that publishes audit logs to CloudWatch Logs using the AWS CLI `create-db-cluster` command with the following parameters:

```
aws neptune create-db-cluster --region us-east-1 \  
  --db-cluster-identifier my_db_cluster_id \  
  --engine neptune \  
  --enable-cloudwatch-logs-exports '["audit"]'
```

You can configure an existing DB cluster to publish audit logs to CloudWatch Logs using the AWS CLI `modify-db-cluster` command with the following parameters:

```
aws neptune modify-db-cluster --region us-east-1 \  
  --db-cluster-identifier my_db_cluster_id \  
  --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

Monitoring Neptune Log Events in Amazon CloudWatch

After enabling Neptune logs, you can monitor log events in Amazon CloudWatch Logs. A new log group is automatically created for the Neptune DB cluster under the following prefix, in which *cluster-name* represents the DB cluster name, and *log_type* represents the log type:

```
/aws/neptune/cluster-name/log_type
```

For example, if you configure the export function to include the audit log for a DB cluster named `mydbcluster`, log data is stored in the `/aws/neptune/mydbcluster/audit` log group.

All of the events from all of the DB instances in a DB cluster are pushed to a log group using different log streams.

If a log group with the specified name exists, Neptune uses that log group to export log data for the Neptune DB cluster. You can use automated configuration, such as AWS CloudFormation, to create log groups with predefined log retention periods, metric filters, and customer access. Otherwise, a new log group is automatically created using the default log retention period, **Never Expire**, in CloudWatch Logs.

You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to change the log retention period. For more information about changing log retention periods in CloudWatch Logs, see [Change Log Data Retention in CloudWatch Logs](#).

You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to search for information within the log events for a DB cluster. For more information about searching and filtering log data, see [Searching and Filtering Log Data](#).

Logging Amazon Neptune API Calls with AWS CloudTrail

Amazon Neptune is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Neptune. CloudTrail captures API calls for Neptune as events, including calls from the Neptune console and from code calls to the Neptune APIs.

CloudTrail only logs events for Neptune Management API calls, such as creating an instance or cluster. If you want to audit changes to your graph, you can use audit logs. For more information, see [Using Audit Logs with Amazon Neptune Clusters \(p. 651\)](#).

Important

Amazon Neptune console, AWS CLI, and API calls are logged as calls made to the Amazon Relational Database Service (Amazon RDS) API.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Neptune. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Neptune, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Neptune Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Neptune, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for Neptune, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts](#)

If an action is taken on behalf of your AWS account using the Neptune console, the Neptune command line interface, or the Neptune SDK APIs, AWS CloudTrail logs the action as calls made to the Amazon RDS API. For example, if you use the Neptune console to modify a DB instance or call the AWS CLI [modify-db-instance](#) command, the AWS CloudTrail log shows a call to the Amazon RDS API [ModifyDBInstance](#) action. For a list of the Neptune API actions that are logged by AWS CloudTrail, see the [Neptune API Reference](#).

Note

AWS CloudTrail only logs events for Neptune Management API calls, such as creating an instance or cluster. If you want to audit changes to your graph, you can use audit logs. For more information, see [Using Audit Logs with Amazon Neptune Clusters \(p. 651\)](#).

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Understanding Neptune Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log for a user that created a snapshot of a DB instance and then deleted that instance using the Neptune console. The console is identified by the `userAgent` element. The requested API calls made by the console (`CreateDBSnapshot` and `DeleteDBInstance`) are found in the `eventName` element for each record. Information about the user (Alice) can be found in the `userIdentity` element.

```
{
  Records:[
    {
      "awsRegion":"us-west-2",
      "eventName":"CreateDBSnapshot",
      "eventSource":"",
      "eventTime":"2014-01-14T16:23:49Z",
      "eventVersion":"1.0",
      "sourceIPAddress":"192.0.2.01",
      "userAgent":"AWS Console, aws-sdk-java\unknown-version Linux\2.6.18-kaos_fleet-1108-
prod.2 Java_HotSpot(TM)_64-Bit_Server_VM\24.45-b08",
      "userIdentity":
      {
        "accessKeyId":"",
        "accountId":"123456789012",
        "arn":"arn:aws:iam::123456789012:user/Alice",
        "principalId":"AIDAI2JXM4FBZZEXAMPLE",
        "sessionContext":
        {
          "attributes":
          {
            "creationDate":"2014-01-14T15:55:59Z",
            "mfaAuthenticated":false
          }
        },
        "type":"IAMUser",
        "userName":"Alice"
      }
    },
    {
      "awsRegion":"us-west-2",
      "eventName":"DeleteDBInstance",
      "eventSource":"",
      "eventTime":"2014-01-14T16:28:27Z",
      "eventVersion":"1.0",
      "sourceIPAddress":"192.0.2.01",
      "userAgent":"AWS Console, aws-sdk-java\unknown-version Linux\2.6.18-kaos_fleet-1108-
prod.2 Java_HotSpot(TM)_64-Bit_Server_VM\24.45-b08",
```



```
"userIdentity":
{
  "accessKeyId": "",
  "accountId": "123456789012",
  "arn": "arn:aws:iam::123456789012:user/Alice",
  "principalId": "AIDAI2JXM4FBZZEXAMPLE",
  "sessionContext":
  {
    "attributes":
    {
      "creationDate": "2014-01-14T15:55:59Z",
      "mfaAuthenticated": false
    }
  },
  "type": "IAMUser",
  "userName": "Alice"
}
]
}
```

Using Neptune Event Notification

Topics

- [Amazon Neptune event categories and event messages \(p. 657\)](#)
- [Subscribing to Neptune event notification \(p. 664\)](#)
- [Managing Neptune event notification subscriptions \(p. 665\)](#)

Amazon Neptune uses Amazon Simple Notification Service (Amazon SNS) to provide notifications when a Neptune event occurs. These notifications can be in any form that is supported by Amazon SNS for an AWS Region, such as an email, a text message, or a call to an HTTP endpoint.

Neptune groups these events into categories that you can subscribe to so that you can be notified when an event in that category occurs. You can subscribe to an event category for a DB instance, DB cluster, DB snapshot, DB cluster snapshot, or for a DB parameter group. For example, if you subscribe to the Backup category for a given DB instance, you are notified whenever a backup-related event occurs that affects the DB instance. You also receive notification when an event notification subscription changes.

Events occur at both the DB cluster and the DB instance level, so you can receive events if you subscribe to a DB cluster or a DB instance.

Event notifications are sent to the addresses you provide when you create the subscription. You might want to create several different subscriptions, such as a subscription that receives all event notifications and another subscription that includes only critical events for your production DB instances. You can easily turn off notification without deleting a subscription. To do so, set the **Enabled** radio button to **No** in the Neptune console.

Important

Amazon Neptune doesn't guarantee the order of events sent in an event stream. The event order is subject to change.

Neptune uses the Amazon Resource Name (ARN) of an Amazon SNS topic to identify each subscription. The Neptune console creates the ARN for you when you create the subscription.

Billing for Neptune event notification is through Amazon SNS. Amazon SNS fees apply when using event notification. For more information, see [Amazon Simple Notification Service Pricing](#).

Amazon Neptune event categories and event messages

Neptune generates a significant number of events in categories that you can subscribe to using the Neptune console. Each category applies to a source type, which can be a DB instance, DB snapshot, or DB parameter group.

Note

Neptune uses existing Amazon RDS event definitions and IDs.

Neptune events originating from DB instances

The following table shows a list of events by event category when a DB instance is the source type.

Category	Amazon RDS event ID	Description
availability	RDS-EVENT-0006	The DB instance restarted.
availability	RDS-EVENT-0004	DB instance shutdown.
availability	RDS-EVENT-0022	An error has occurred while restarting MySQL or MariaDB.
backup	RDS-EVENT-0001	Backing up DB instance.
backup	RDS-EVENT-0002	Finished DB Instance backup.
configuration change	RDS-EVENT-0009	The DB instance has been added to a security group.
configuration change	RDS-EVENT-0024	The DB instance is being converted to a Multi-AZ DB instance.
configuration change	RDS-EVENT-0030	The DB instance is being converted to a Single-AZ DB instance.
configuration change	RDS-EVENT-0012	Applying modification to database instance class.
configuration change	RDS-EVENT-0018	The current storage settings for this DB instance are being changed.
configuration change	RDS-EVENT-0011	A parameter group for this DB instance has changed.
configuration change	RDS-EVENT-0092	A parameter group for this DB instance has finished updating.

Category	Amazon RDS event ID	Description
configuration change	RDS-EVENT-0028	Automatic backups for this DB instance have been disabled.
configuration change	RDS-EVENT-0032	Automatic backups for this DB instance have been enabled.
configuration change	RDS-EVENT-0025	The DB instance has been converted to a Multi-AZ DB instance.
configuration change	RDS-EVENT-0029	The DB instance has been converted to a Single-AZ DB instance.
configuration change	RDS-EVENT-0014	The DB instance class for this DB instance has changed.
configuration change	RDS-EVENT-0017	The storage settings for this DB instance have changed.
configuration change	RDS-EVENT-0010	The DB instance has been removed from a security group.
creation	RDS-EVENT-0005	DB instance created.
deletion	RDS-EVENT-0003	The DB instance has been deleted.
failover	RDS-EVENT-0034	Neptune is not attempting a requested failover because a failover recently occurred on the DB instance.
failover	RDS-EVENT-0013	A Multi-AZ failover that resulted in the promotion of a standby instance has started.
failover	RDS-EVENT-0015	A Multi-AZ failover that resulted in the promotion of a standby instance is complete. It may take several minutes for the DNS to transfer to the new primary DB instance.
failover	RDS-EVENT-0065	The instance has recovered from a partial failover.

Category	Amazon RDS event ID	Description
failover	RDS-EVENT-0049	A Multi-AZ failover has completed.
failover	RDS-EVENT-0050	A Multi-AZ activation has started after a successful instance recovery.
failover	RDS-EVENT-0051	A Multi-AZ activation is complete. Your database should be accessible now.
failure	RDS-EVENT-0031	The DB instance has failed due to an incompatible configuration or an underlying storage issue. Begin a point-in-time-restore for the DB instance.
failure	RDS-EVENT-0036	The DB instance is in an incompatible network. Some of the specified subnet IDs are invalid or do not exist.
failure	RDS-EVENT-0035	The DB instance has invalid parameters. For example, if the DB instance could not start because a memory-related parameter is set too high for this instance class, the customer action would be to modify the memory parameter and reboot the DB instance.
failure	RDS-EVENT-0082	Neptune was unable to copy backup data from an Amazon S3 bucket. It is likely that the permissions for Neptune to access the Amazon S3 bucket are configured incorrectly.

Category	Amazon RDS event ID	Description
low storage	RDS-EVENT-0089	The DB instance has consumed more than 90% of its allocated storage. You can monitor the storage space for a DB instance using the Free Storage Space metric.
low storage	RDS-EVENT-0007	The allocated storage for the DB instance has been exhausted. To resolve this issue, you should allocate additional storage for the DB instance.
maintenance	RDS-EVENT-0026	Offline maintenance of the DB instance is taking place. The DB instance is currently unavailable.
maintenance	RDS-EVENT-0027	Offline maintenance of the DB instance is complete. The DB instance is now available.
maintenance	RDS-EVENT-0047	Patching of the DB instance has completed.
notification	RDS-EVENT-0044	Operator-issued notification. For more information, see the event message.
notification	RDS-EVENT-0048	Patching of the DB instance has been delayed.
notification	RDS-EVENT-0087	The DB instance has been stopped.
notification	RDS-EVENT-0088	The DB instance has been started.
notification	RDS-EVENT-0154	The DB instance is being started due to it exceeding the maximum allowed time being stopped.
notification	RDS-EVENT-0158	DB instance is in a state that can't be upgraded.

Category	Amazon RDS event ID	Description
notification	RDS-EVENT-0173	DB instance has been patched.
read replica	RDS-EVENT-0045	An error has occurred in the read replication process. For more information, see the event message.
read replica	RDS-EVENT-0046	The read replica has resumed replication. This message appears when you first create a read replica, or as a monitoring message confirming that replication is functioning properly. If this message follows an RDS-EVENT-0045 notification, then replication has resumed following an error or after replication was stopped.
read replica	RDS-EVENT-0057	Replication on the read replica was terminated.
read replica	RDS-EVENT-0062	Replication on the read replica was manually stopped.
read replica	RDS-EVENT-0063	Replication on the read replica was reset.
recovery	RDS-EVENT-0020	Recovery of the DB instance has started. Recovery time will vary with the amount of data to be recovered.
recovery	RDS-EVENT-0021	Recovery of the DB instance is complete.
recovery	RDS-EVENT-0023	A manual backup has been requested but Neptune is currently in the process of creating a DB snapshot. Submit the request again after Neptune has completed the DB snapshot.

Category	Amazon RDS event ID	Description
recovery	RDS-EVENT-0052	Recovery of the Multi-AZ instance has started. Recovery time will vary with the amount of data to be recovered.
recovery	RDS-EVENT-0053	Recovery of the Multi-AZ instance is complete.
restoration	RDS-EVENT-0008	The DB instance has been restored from a DB snapshot.
restoration	RDS-EVENT-0019	The DB instance has been restored from a point-in-time backup.

Neptune events originating from a DB cluster

The following table shows a list of events by event category when a DB cluster is the source type.

Category	RDS event ID	Description
failover	RDS-EVENT-0069	A failover for the DB cluster has failed.
failover	RDS-EVENT-0070	A failover for the DB cluster has restarted.
failover	RDS-EVENT-0071	A failover for the DB cluster has finished.
failover	RDS-EVENT-0072	A failover for the DB cluster has begun within the same Availability Zone.
failover	RDS-EVENT-0073	A failover for the DB cluster has begun across Availability Zones.
failure	RDS-EVENT-0083	Neptune was unable to copy backup data from an Amazon S3 bucket. It is likely that the permissions for Neptune to access the Amazon S3 bucket are configured incorrectly.
maintenance	RDS-EVENT-0156	The DB cluster has a DB engine minor version upgrade available.

Category	RDS event ID	Description
notification	RDS-EVENT-0076	Migration to an Neptune DB cluster failed.
notification	RDS-EVENT-0077	An attempt to convert a table from the source database to InnoDB failed during the migration to an Neptune DB cluster.
notification	RDS-EVENT-0150	The DB cluster stopped.
notification	RDS-EVENT-0151	The DB cluster started.
notification	RDS-EVENT-0152	The DB cluster stop failed.
notification	RDS-EVENT-0153	The DB cluster is being started due to it exceeding the maximum allowed time being stopped.

Neptune events originating from DB cluster snapshot

The following table shows the event category and a list of events when a Neptune DB cluster snapshot is the source type.

Category	RDS event ID	Description
backup	RDS-EVENT-0074	Creation of a manual DB cluster snapshot has started.
backup	RDS-EVENT-0075	A manual DB cluster snapshot has been created.
notification	RDS-EVENT-0162	DB cluster snapshot export task failed.
notification	RDS-EVENT-0163	DB cluster snapshot export task canceled.
notification	RDS-EVENT-0164	DB cluster snapshot export task completed.
backup	RDS-EVENT-0168	Creating automated cluster snapshot.
backup	RDS-EVENT-0169	Automated cluster snapshot created.
creation	RDS-EVENT-0170	DB cluster created.

Category	RDS event ID	Description
deletion	RDS-EVENT-0171	DB cluster deleted.
notification	RDS-EVENT-0172	Renamed DB cluster from [old DB cluster name] to [new DB cluster name].

Neptune events originating from DB cluster parameter group

The following table shows the event category and a list of events when a DB cluster parameter group is the source type.

Category	RDS event ID	Description
configuration change	RDS-EVENT-0037	The parameter group was modified.

Neptune events originating from a security group

The following table shows the event category and a list of events when a security group is the source type.

Category	RDS event ID	Description
configuration change	RDS-EVENT-0038	The security group has been modified.
failure	RDS-EVENT-0039	The security group owned by [user] does not exist; authorization for the security group has been revoked.

Subscribing to Neptune event notification

You can use the Neptune console to subscribe to event notifications, as follows:

To subscribe to Neptune event notification

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Event subscriptions**.
3. In the **Event subscriptions** pane, choose **Create event subscription**.
4. In the **Create event subscription** dialog box, do the following:
 - a. For **Name**, enter a name for the event notification subscription.
 - b. For **Send notifications to**, choose an existing Amazon SNS ARN for an Amazon SNS topic, or choose **create topic** to enter the name of a topic and a list of recipients.
 - c. For **Source type**, choose a source type.

- d. Choose **Yes** to enable the subscription. If you want to create the subscription but to not have notifications sent yet, choose **No**.
- e. Depending on the source type you selected, choose the event categories and the sources that you want to receive event notifications from.
- f. Choose **Create**.

Managing Neptune event notification subscriptions

If you choose **Event subscriptions** in the navigation pane of the Neptune console, you can view subscription categories and a list of your current subscriptions.

You can also modify or delete a specific subscription.

Modifying Neptune event notification subscriptions

To modify your current Neptune event notification subscriptions

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Event subscriptions**. The **Event subscriptions** pane shows all your event notification subscriptions.
3. In the **Event subscriptions** pane, choose the subscription that you want to modify and choose **Edit**.
4. Make your changes to the subscription in either the **Target** or **Source** section. You can add or remove source identifiers by selecting or deselecting them in the **Source** section.
5. Choose **Edit**. The Neptune console indicates that the subscription is being modified.

Deleting a Neptune event notification subscription

You can delete a subscription when you no longer need it. All subscribers to the topic will no longer receive event notifications specified by the subscription.

To delete an Neptune event notification subscription

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Event subscriptions**.
3. In the **Event subscriptions** pane, choose the subscription that you want to delete.
4. Choose **Delete**.
5. The Neptune console indicates that the subscription is being deleted.

Tagging Amazon Neptune Resources

You can use Amazon Neptune tags to add metadata to your Neptune resources. In addition, you can use tags with AWS Identity and Access Management (IAM) policies to manage access to Neptune resources and control what actions can be applied to those resources. Finally, you can use tags to track costs by grouping expenses for similarly tagged resources.

All Neptune resources can be tagged, including the following:

- DB instances

- DB clusters
- Read Replicas
- DB snapshots
- DB cluster snapshots
- Event subscriptions
- DB parameter groups
- DB cluster parameter groups
- DB subnet groups

Overview of Neptune Resource Tags

An Amazon Neptune tag is a name-value pair that you define and associate with a Neptune resource. The name is referred to as the *key*. Supplying a value for the key is optional. You can use tags to assign arbitrary information to a Neptune resource. You can use a tag key, for example, to define a category, and the tag value might be an item in that category. For example, you might define a tag key of "project" and a tag value of "Salix," indicating that the Neptune resource is assigned to the Salix project. You can also use tags to designate Neptune resources as being used for test or production by using a key such as `environment=test` or `environment =production`. We recommend that you use a consistent set of tag keys to make it easier to track metadata that is associated with Neptune resources.

Use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see [Using Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*.

Each Neptune resource has a tag set, which contains all the tags that are assigned to that Neptune resource. A tag set can contain as many as 10 tags, or it can be empty. If you add a tag to a Neptune resource that has the same key as an existing tag on resource, the new value overwrites the old value.

AWS does not apply any semantic meaning to your tags; tags are interpreted strictly as character strings. Neptune can set tags on a DB instance or other Neptune resources, depending on the settings that you use when you create the resource. For example, Neptune might add a tag indicating that a DB instance is for production or for testing.

- The tag key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "aws:" or "rds:". The string can contain only the set of Unicode letters, digits, white space, '_', ':', '/', '=', '+', '-' (Java regex: `"^([\p{L}\p{Z}\p{N}_. :/+\\-]*)$"`).
- The tag value is an optional string value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "aws:". The string can contain only the set of Unicode letters, digits, white space, '_', ':', '/', '=', '+', '-' (Java regex: `"^([\p{L}\p{Z}\p{N}_. :/+\\-]*)$"`).

Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of `project/Trinity` and `cost-center/Trinity`.

Note

You can add a tag to a snapshot. However, your bill won't reflect this grouping.

You can use the AWS Management Console, the AWS CLI, or the Neptune API to add, list, and delete tags on Neptune resources. When using the AWS CLI or the Neptune API, you must provide the Amazon

Resource Name (ARN) for the Neptune resource that you want to work with. For more information about constructing an ARN, see [Constructing an ARN for Neptune \(p. 669\)](#).

Tags are cached for authorization purposes. Because of this, additions and updates to tags on Neptune resources can take several minutes before they are available.

Copying Tags in Neptune

When you create or restore a DB instance, you can specify that the tags from the DB instance are copied to snapshots of the DB instance. Copying tags ensures that the metadata for the DB snapshots matches that of the source DB instance, and that any access policies for the DB snapshot also match those of the source DB instance. Tags are not copied by default.

You can specify that tags are copied to DB snapshots for the following actions:

- Creating a DB instance.
- Restoring a DB instance.
- Creating a Read Replica.
- Copying a DB snapshot.

Note

If you include a value for the `--tag-key` parameter of the `create-db-cluster-snapshot` AWS CLI command (or supply at least one tag to the [CreateDBClusterSnapshot \(p. 889\)](#) API action), Neptune doesn't copy tags from the source DB instance to the new DB snapshot. This is true even if the source DB instance has the `--copy-tags-to-snapshot` (`CopyTagsToSnapshot`) option enabled.

This means that you can create a copy of a DB instance from a DB snapshot and avoid adding tags that don't apply to the new DB instance. After you create your DB snapshot using the AWS CLI `create-db-cluster-snapshot` command (or the `CreateDBClusterSnapshot` Neptune API action), you can then add tags as described later in this topic.

Tagging in Neptune Using the AWS Management Console

The process to tag an Amazon Neptune resource is similar for all resources. The following procedure shows how to tag a Neptune DB instance.

To add a tag to a DB instance

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**.

Note

To filter the list of DB instances in the **Instances** pane, type a text string in the **Filter instances** box. Only DB instances that contain the string appear.

3. Choose the DB instance that you want to tag.
4. Choose **Instance actions**, and then choose **See details**.
5. In the details section, scroll down to the **Tags** section.
6. Choose **Add**. The **Add tags** window appears.
7. Type a value for **Tag key** and **Value**.
8. To add another tag, you can choose **Add another Tag** and type a value for its **Tag key** and **Value**.

Repeat this step as many times as necessary.

9. Choose **Add**.

To delete a tag from a DB instance

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**.

Note

To filter the list of DB instances in the **Instances** pane, type a text string in the **Filter instances** box. Only DB instances that contain the string appear.

3. Choose the DB instance that you want to tag.
4. Choose **Instance actions**, and then choose **See details**.
5. In the details section, scroll down to the **Tags** section.
6. Choose the tag that you want to delete.
7. Choose **Remove**, and then choose **Remove** in the **Remove tags** window.

Tagging in Neptune Using the AWS CLI

You can add, list, or remove tags for a DB instance in Neptune using the AWS CLI.

- To add one or more tags to a Neptune resource, use the AWS CLI command [add-tags-to-resource](#).
- To list the tags on a Neptune resource, use the AWS CLI command [list-tags-for-resource](#).
- To remove one or more tags from a Neptune resource, use the AWS CLI command [remove-tags-from-resource](#).

To learn more about how to construct the required Amazon Resource Name (ARN), see [Constructing an ARN for Neptune \(p. 669\)](#).

Tagging in Neptune Using the API

You can add, list, or remove tags for a DB instance using the Neptune API.

- To add a tag to a Neptune resource, use the [AddTagsToResource](#) operation.
- To list tags that are assigned to a Neptune resource, use the [ListTagsForResource](#).
- To remove tags from a Neptune resource, use the [RemoveTagsFromResource](#) operation.

To learn more about how to construct the required ARN, see [Constructing an ARN for Neptune \(p. 669\)](#).

When working with XML using the Neptune API, tags use the following schema:

```
<Tagging>
  <TagSet>
    <Tag>
      <Key>Project</Key>
      <Value>Trinity</Value>
    </Tag>
    <Tag>
      <Key>User</Key>
      <Value>Jones</Value>
    </Tag>
  </TagSet>
</Tagging>
```

```
</TagSet>
</Tagging>
```

The following table provides a list of the allowed XML tags and their characteristics. Values for `key` and `value` are case-dependent. For example, `project=Trinity` and `PROJECT=Trinity` are two distinct tags.

Tagging Element	Description
TagSet	A tag set is a container for all tags that are assigned to a Neptune resource. There can be only one tag set per resource. You work with a <code>TagSet</code> only through the Neptune API.
Tag	A tag is a user-defined key-value pair. There can be from 1 to 50 tags in a tag set.
Key	A key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with "rds:" or "aws:". The string can contain only the set of Unicode letters, digits, white space, '_', '/', '=', '+', '-' (Java regex: " <code>^\p{L}\p{Z}\p{N}_./+=\ -]*\$</code> "). Keys must be unique to a tag set. For example, you can't have a key-pair in a tag set with the key the same but with different values, such as <code>project/Trinity</code> and <code>project/Xanadu</code> .
Value	A value is the optional value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with "rds:" or "aws:". The string can contain only the set of Unicode letters, digits, white space, '_', '/', '=', '+', '-' (Java regex: " <code>^\p{L}\p{Z}\p{N}_./+=\ -]*\$</code> "). Values don't have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of <code>project/Trinity</code> and <code>cost-center/Trinity</code> .

Working with ARNs in Amazon Neptune

Resources that are created in Amazon Web Services are each uniquely identified with an Amazon Resource Name (ARN). For certain Amazon Neptune operations, you must uniquely identify a Neptune resource by specifying its ARN.

Important

Amazon Neptune shares the format of Amazon RDS ARNs. Neptune ARNs contain `rds` and not `neptune`.

Topics

- [Constructing an ARN for Neptune \(p. 669\)](#)
- [Getting an Existing ARN in Amazon Neptune \(p. 670\)](#)

Constructing an ARN for Neptune

You can construct an ARN for an Amazon Neptune resource using the following syntax. Note that Neptune shares the format of Amazon RDS ARNs.

```
arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

The following table shows the format that you should use when constructing an ARN for a particular Neptune resource type.

Resource Type	ARN Format
DB instance	<p>arn:aws:rds:<region>:<account>:db:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:db:my-instance-1</pre>
DB cluster	<p>arn:aws:rds:<region>:<account>:cluster:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster:my-cluster-1</pre>
Event subscription	<p>arn:aws:rds:<region>:<account>:es:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:es:my-subscription</pre>
DB parameter group	<p>arn:aws:rds:<region>:<account>:pg:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:pg:my-param-enable-logs</pre>
DB cluster parameter group	<p>arn:aws:rds:<region>:<account>:cluster-pg:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-pg:my-cluster-param-timezone</pre>
DB cluster snapshot	<p>arn:aws:rds:<region>:<account>:cluster-snapshot:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-snapshot:my-snap-20160809</pre>
DB subnet group	<p>arn:aws:rds:<region>:<account>:subgrp:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:subgrp:my-subnet-10</pre>

Getting an Existing ARN in Amazon Neptune

You can get the ARN of a Neptune resource by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or Neptune API.

Getting an Existing ARN Using the AWS Management Console

To get an ARN using the console, navigate to the resource that you want an ARN for, and view the details for that resource. For example, to get the ARN for a DB instance, choose **Instances** in the navigation panel, and choose the instance that you want from the list. The ARN is in the **Instance Details** section.

Getting an Existing ARN Using the AWS CLI

To use the AWS CLI to get an ARN for a particular Neptune resource, use the `describe` command for that resource. The following table shows each AWS CLI command and the ARN property that is used with the command to get an ARN.

AWS CLI Command	ARN Property
<code>describe-event-subscriptions</code>	EventSubscriptionArn
<code>describe-certificates</code>	CertificateArn
<code>describe-db-parameter-groups</code>	DBParameterGroupArn
<code>describe-db-cluster-parameter-groups</code>	DBClusterParameterGroupArn
<code>describe-db-instances</code>	DBInstanceArn
<code>describe-events</code>	SourceArn
<code>describe-db-subnet-groups</code>	DBSubnetGroupArn
<code>describe-db-clusters</code>	DBClusterArn
<code>describe-db-cluster-snapshots</code>	DBClusterSnapshotArn

For example, the following AWS CLI command gets the ARN for a DB instance.

Example

For Linux, OS X, or Unix:

```
aws neptune describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2
```

For Windows:

```
aws neptune describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2
```

Getting an Existing ARN Using the API

To get an ARN for a particular Neptune resource, call the following API actions and use the ARN properties shown.

Neptune API Action	ARN Property
<code>DescribeEventSubscriptions</code>	EventSubscriptionArn

Neptune API Action	ARN Property
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeEvents	SourceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Backing Up and Restoring an Amazon Neptune DB Cluster

This section shows how you can back up and restore Amazon Neptune DB clusters.

Topics

- [Overview of Backing Up and Restoring a Neptune DB Cluster \(p. 674\)](#)
- [Creating a DB Cluster Snapshot in Neptune \(p. 678\)](#)
- [Restoring from a DB Cluster Snapshot \(p. 679\)](#)
- [Copying a DB Cluster Snapshot \(p. 681\)](#)
- [Sharing a DB Cluster Snapshot \(p. 686\)](#)
- [Deleting a Neptune Snapshot \(p. 690\)](#)

Overview of Backing Up and Restoring a Neptune DB Cluster

This section provides top-level information about backing up and restoring data in Amazon Neptune.

Topics

- [Fault Tolerance for a Neptune DB Cluster \(p. 674\)](#)
- [Neptune Backups \(p. 674\)](#)
- [CloudWatch metrics that are useful for managing Neptune backup storage \(p. 675\)](#)
- [Restoring Data from a Neptune Backup \(p. 676\)](#)
- [Backup Window in Neptune \(p. 676\)](#)

Fault Tolerance for a Neptune DB Cluster

A Neptune DB cluster is fault tolerant by design. The cluster volume spans multiple Availability Zones in a single AWS Region, and each Availability Zone contains a copy of the cluster volume data. This functionality means that your DB cluster can tolerate a failure of an Availability Zone without any loss of data and only a brief interruption of service.

If the primary instance in a DB cluster fails, Neptune automatically fails over to a new primary instance in one of two ways:

- By promoting an existing Neptune replica to the new primary instance
- By creating a new primary instance

If the DB cluster has one or more Neptune replicas, then a Neptune replica is promoted to the primary instance during a failure event. A failure event results in a brief interruption, during which read and write operations fail with an exception. However, service is typically restored in less than 120 seconds, and often less than 60 seconds. To increase the availability of your DB cluster, we recommend that you create at least one or more Neptune replicas in two or more different Availability Zones.

You can customize the order in which your Neptune replicas are promoted to the primary instance after a failure by assigning each replica a priority. Priorities range from 0 for the highest priority to 15 for the lowest priority. If the primary instance fails, Neptune promotes the Neptune replica with the highest priority to the new primary instance. You can modify the priority of a Neptune replica at any time. Modifying the priority doesn't trigger a failover.

More than one Neptune replica can share the same priority, resulting in promotion tiers. If two or more Neptune replicas share the same priority, then Neptune promotes the replica that is largest in size. If two or more Neptune replicas share the same priority and size, then Neptune promotes an arbitrary replica in the same promotion tier.

If the DB cluster doesn't contain any Neptune replicas, then the primary instance is recreated during a failure event. A failure event results in an interruption during which read and write operations fail with an exception. Service is restored when the new primary instance is created, which typically takes less than 10 minutes. Promoting a Neptune replica to the primary instance is much faster than creating a new primary instance.

Neptune Backups

Neptune backs up your cluster volume automatically and retains restore data for the length of the *backup retention period*. Neptune backups are continuous and incremental so you can quickly restore to

any point within the backup retention period. No performance impact or interruption of database service occurs as backup data is being written. You can specify a backup retention period, from 1 to 35 days, when you create or modify a DB cluster.

To control your backup storage usage, you can reduce the backup retention interval, remove old manual snapshots when they are no longer needed, or both. To help manage your costs, you can monitor the amount of storage consumed by continuous backups and manual snapshots that persist beyond the retention period. You can reduce the backup retention interval and remove manual snapshots when they are no longer needed.

If you want to retain a backup beyond the backup retention period, you can also take a snapshot of the data in your cluster volume. Storing snapshots incurs the standard storage charges for Neptune. For more information about Neptune storage pricing, see [Amazon Neptune Pricing](#).

Neptune retains incremental restore data for the entire backup retention period. So you only need to create a snapshot for data that you want to retain beyond the backup retention period. You can create a new DB cluster from the snapshot.

Important

If you delete a DB cluster, all its automated backups are deleted at the same time and cannot be recovered. This means that unless you choose to create a final DB snapshot manually, you can't restore the DB instance to its final state at a later time. Manual snapshots are not deleted when the cluster is deleted.

Note

- For Amazon Neptune DB clusters, the default backup retention period is one day regardless of how the DB cluster is created.
- You cannot disable automated backups on Neptune. The backup retention period for Neptune is managed by the DB cluster.

CloudWatch metrics that are useful for managing Neptune backup storage

You can use the Amazon CloudWatch metrics `TotalBackupStorageBilled`, `SnapshotStorageUsed`, and `BackupRetentionPeriodStorageUsed` to review and monitor the amount of storage used by your Neptune backups, as follows:

- `BackupRetentionPeriodStorageUsed` represents the amount of backup storage used, in bytes, for storing continuous backups at the current time. This value depends on the size of the cluster volume and the amount of changes you make during the retention period. However, for billing purposes it doesn't exceed the cumulative cluster volume size during the retention period. For example, if your cluster's `VolumeBytesUsed` size is 107,374,182,400 bytes (100 GiB), and your retention period is two days, the maximum value for `BackupRetentionPeriodStorageUsed` is 214,748,364,800 bytes (100 GiB + 100 GiB).
- `SnapshotStorageUsed` represents the amount of backup storage used, in bytes, for storing manual snapshots beyond the backup retention period. Manual snapshots don't count against your snapshot backup storage while their creation timestamp is within the retention period. All automatic snapshots also don't count against your snapshot backup storage. The size of each snapshot is the size of the cluster volume at the time you take the snapshot. The `SnapshotStorageUsed` value depends on the number of snapshots you keep and the size of each snapshot. For example, suppose you have one manual snapshot outside the retention period, and the cluster's `VolumeBytesUsed` size was 100 GiB when that snapshot was taken. The amount of `SnapshotStorageUsed` is 107,374,182,400 bytes (100 GiB).
- `TotalBackupStorageBilled` represents the sum, in bytes, of `BackupRetentionPeriodStorageUsed` and `SnapshotStorageUsed`, minus an amount of free

backup storage, which equals the size of the cluster volume for one day. The free backup storage is equal to the latest volume size. For example if your cluster's `VolumeBytesUsed` size is 100 GiB, your retention period is two days, and you have one manual snapshot outside the retention period, the `TotalBackupStorageBilled` is 214,748,364,800 bytes (200 GiB + 100 GiB - 100 GiB).

You can monitor a Neptune cluster and build reports using CloudWatch metrics through the [CloudWatch console](#). For more information about how to use CloudWatch metrics, see [Monitoring Neptune \(p. 639\)](#) and the table of metrics in [Neptune CloudWatch Metrics \(p. 644\)](#).

Restoring Data from a Neptune Backup

You can recover your data by creating a new Neptune DB cluster from the backup data that Neptune retains, or from a DB cluster snapshot that you have saved. You can quickly restore a new copy of a DB cluster created from backup data to any point in time during your backup retention period. The continuous and incremental nature of Neptune backups during the backup retention period means you don't need to take frequent snapshots of your data to improve restore times.

To determine the latest or earliest restorable time for a DB instance, look for the `Latest Restorable Time` or `Earliest Restorable Time` values on the Neptune console. The latest restorable time for a DB cluster is the most recent point at which you can restore your DB cluster, typically within 5 minutes of the current time. The earliest restorable time specifies how far back within the backup retention period that you can restore your cluster volume.

You can determine when the restore of a DB cluster is complete by checking the `Latest Restorable Time` and `Earliest Restorable Time` values. The `Latest Restorable Time` and `Earliest Restorable Time` values return NULL until the restore operation is complete. You can't request a backup or restore operation if `Latest Restorable Time` or `Earliest Restorable Time` returns NULL.

To restore a DB instance to a specified time using the AWS Management Console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Instances**. Choose the primary instance for the DB cluster that you want to restore.
3. Choose **Instance actions**, and then choose **Restore to point in time**.

In the **Launch DB Instance** window, choose **Custom** under **Restore time**.

4. Specify the date and time that you want to restore to under **Custom**.
5. Type a name for the new, restored DB instance for **DB instance identifier** under **Settings**.
6. Choose **Launch DB Instance** to launch the restored DB instance.

A new DB instance is created with the name you specified, and a new DB cluster is created. The DB cluster name is the new DB instance name followed by `-cluster`. For example, if the new DB instance name is `myrestoredb`, the new DB cluster name is `myrestoredb-cluster`.

Backup Window in Neptune

Automated backups occur daily during the preferred backup window. If the backup requires more time than allotted to the backup window, the backup continues after the window ends, until it finishes. The backup window can't overlap with the weekly maintenance window for the DB instance.

During the automatic backup window, storage I/O might be suspended briefly while the backup process initializes (typically under a few seconds). You might experience elevated latencies for a few minutes during backups for Multi-AZ deployments.

The backup window is normally selected at random from an eight-hour block of time per Region by the Amazon RDS control plane underlying Neptune. The time blocks for each Region from which the default backups windows are assigned is documented in the [Backup Window](#) section of the Amazon RDS User Guide.

Creating a DB Cluster Snapshot in Neptune

Neptune creates a storage volume snapshot of your DB cluster, backing up the entire DB cluster and not just individual databases. When you create a DB cluster snapshot, you need to identify which DB cluster you are going to back up. Then give your DB cluster snapshot a name so that you can restore from it later. The amount of time it takes to create a DB cluster snapshot varies with the size of your databases. The snapshot includes the entire storage volume. So the size of files (such as temporary files) also affects the amount of time it takes to create the snapshot.

You can create a DB cluster snapshot using the AWS Management Console, the AWS CLI, or the Neptune API.

Using the Console to Create a DB Cluster Snapshot

To create a DB cluster snapshot

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Databases**.
3. In the list of DB instances, choose the primary instance for the DB cluster.
4. Choose **Instance actions**, and then choose **Take snapshot**.

The **Take DB Snapshot** window appears.

5. Enter the name of the DB cluster snapshot in the **Snapshot name** box.
6. Choose **Take Snapshot**.

Restoring from a DB Cluster Snapshot

When you create an Amazon Neptune snapshot of a DB cluster, Neptune creates a storage volume snapshot of the cluster, backing up all its data and not just individual instances. You can later create a new DB cluster by restoring from this DB cluster snapshot. When you restore the DB cluster, you provide the name of the DB cluster snapshot to restore from, and then provide a name for the new DB cluster that is created by the restore.

Contents

- [Things to keep in mind about restoring a Neptune DB cluster from a snapshot \(p. 679\)](#)
 - [You cannot restore to an existing DB cluster \(p. 679\)](#)
 - [No instances are restored \(p. 679\)](#)
 - [No custom parameter group is restored \(p. 679\)](#)
 - [No custom security groups are restored \(p. 679\)](#)
 - [You can't restore from a shared encrypted snapshot \(p. 680\)](#)
 - [A restored DB cluster uses as much storage as before \(p. 680\)](#)
- [How to restore from a snapshot \(p. 680\)](#)
 - [Using the Console to Restore from a Snapshot \(p. 680\)](#)

Things to keep in mind about restoring a Neptune DB cluster from a snapshot

You cannot restore to an existing DB cluster

The restore process always creates a new DB cluster, so you can't restore to a DB cluster that already exists.

No instances are restored

A new DB cluster created by a restore has no instances associated with it.

As soon as the restore is complete and your new DB cluster is available, explicitly create the instances that you will need. You can do this on the Neptune console, or using the [CreateDBInstance \(p. 833\)](#) API.

No custom parameter group is restored

A new DB cluster created by a restore automatically has the default DB parameter group associated with it.

As soon as the restore is complete and your new DB cluster is available, associate any custom DB parameter group that the instance you restored from was using. To do that, use the **Modify** command on the Neptune console, or the [ModifyDBInstance \(p. 846\)](#) API.

Important

We recommend that you save a custom parameter group being used in a DB cluster that you are creating a snapshot of. Then, when you restore from that snapshot, you can easily associate the correct parameter group with the restored DB cluster.

No custom security groups are restored

A new DB cluster created by a restore automatically has the default security group associated with it.

As soon as the restore is complete and your new DB cluster is available, associate any custom security groups that the instance you restored from was using. To do that, use the **Modify** command on the Neptune console, or the [ModifyDBInstance \(p. 846\)](#) API.

You can't restore from a shared encrypted snapshot

You cannot restore a DB cluster from a DB cluster snapshot that is both shared and encrypted.

Instead, make an unshared copy of the snapshot and restore from the copy.

A restored DB cluster uses as much storage as before

When you restore a DB cluster from a DB cluster snapshot, the amount of storage allocated to the new cluster is the same as was allocated to the DB cluster from which the snapshot was made, regardless of how much of that allocated storage is actually being used.

In other words, the "high water mark" for which you are billed does not change. Resetting the high water mark requires exporting the data from your graph and then reloading it onto a new DB cluster (see [Neptune storage billing \(p. 75\)](#)).

How to restore from a snapshot

You can restore a DB cluster from a DB cluster snapshot using the AWS Management Console, the AWS CLI, or the Neptune API.

Using the Console to Restore from a Snapshot

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB cluster snapshot that you want to restore from.
4. Choose **Actions, Restore Snapshot**.
5. On the **Restore DB Instance** page, in the **DB Instance Identifier** box, enter the name for your restored DB cluster.
6. Choose **Restore DB Instance**.
7. If you want to restore the functionality of the DB cluster to that of the DB cluster that the snapshot was created from, you must modify the DB cluster to use the security group. The next steps assume that your DB cluster is in a virtual private cloud (VPC). If your DB cluster is not in a VPC, use the Amazon EC2 console to locate the security group that you need for the DB cluster.
 - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
 - b. In the navigation pane, choose **Security Groups**.
 - c. Choose the security group that you want to use for your DB clusters. If necessary, add rules to link the security group to a security group for an EC2 instance.

Copying a DB Cluster Snapshot

With Neptune, you can copy automated or manual DB cluster snapshots. After you copy a snapshot, the copy is a manual snapshot.

You can copy a snapshot within the same AWS Region and across AWS Regions.

Copying an automated snapshot to another AWS account is a two-step process: First, you create a manual snapshot from the automated snapshot, and then you copy the manual snapshot to the other account.

As an alternative to copying, you can also share manual snapshots with other AWS accounts. For more information, see [Sharing a DB Cluster Snapshot \(p. 686\)](#).

Topics

- [Limitations on Copying a Snapshot \(p. 681\)](#)
- [Retention of DB Cluster Snapshot Copies \(p. 681\)](#)
- [Handling Encryption When Copying Snapshots \(p. 682\)](#)
- [Copying Snapshots Across AWS Regions \(p. 682\)](#)
- [Copying a DB Cluster Snapshot Using the Console \(p. 682\)](#)
- [Copying a DB Cluster Snapshot Using the AWS CLI \(p. 683\)](#)

Limitations on Copying a Snapshot

The following are some limitations when you copy snapshots:

- You can copy a snapshot between China (Beijing) and China (Ningxia), but you can't copy a snapshot between these China regions and other AWS Regions.
- You can copy a snapshot between AWS GovCloud (US-East) and AWS GovCloud (US-West), but you can't copy a snapshot between these AWS GovCloud (US) regions and other AWS Regions.
- If you delete a source snapshot before the target snapshot becomes available, the snapshot copy might fail. Verify that the target snapshot has a status of `AVAILABLE` before you delete a source snapshot.
- You can have up to five snapshot copy requests in progress to a single Region per account.
- Depending on the regions involved and the amount of data to be copied, a cross-region snapshot copy can take hours to complete.

If there is a large number of cross-region snapshot copy requests from a given source AWS Region, Neptune may put new cross-region copy requests from that source AWS Region into a queue until some in-progress copies complete. No progress information is displayed about copy requests while they are in that queue. Progress information is displayed only after the copy starts.

Retention of DB Cluster Snapshot Copies

Neptune deletes automated snapshots as follows:

- At the end of their retention period.
- When you disable automated snapshots for a DB cluster.
- When you delete a DB cluster.

If you want to keep an automated snapshot for a longer period, copy it to create a manual snapshot, which is then retained until you delete it. Neptune storage costs might apply to manual snapshots if they exceed your default storage space.

For more information about backup storage costs, see [Neptune Pricing](#).

Handling Encryption When Copying Snapshots

You can copy a snapshot that has been encrypted using an AWS KMS encryption key. If you copy an encrypted snapshot, the copy of the snapshot must also be encrypted. You can encrypt the copy with the same AWS KMS encryption key as the original snapshot, or you can specify a different AWS KMS encryption key.

You cannot encrypt an unencrypted DB cluster snapshot when you copy it.

For Amazon Neptune DB cluster snapshots, you can also leave the DB cluster snapshot unencrypted and instead specify a AWS KMS encryption key when restoring. The restored DB cluster is encrypted using the specified key.

Copying Snapshots Across AWS Regions

Note

This feature is available starting in [Neptune engine release 1.0.2.1 \(p. 755\)](#).

When you copy a snapshot to an AWS Region that is different from the source snapshot's AWS Region, the first copy is a full snapshot copy, even if you copy an incremental snapshot. A full snapshot copy contains all of the data and metadata required to restore the DB instance. After the first snapshot copy, you can copy incremental snapshots of the same DB instance to the same destination region within the same AWS account.

An incremental snapshot contains only the data that has changed after the most recent snapshot of the same DB instance. Incremental snapshot copying is faster and results in lower storage costs than full snapshot copying. Incremental snapshot copying across AWS Regions is supported for both unencrypted and encrypted snapshots.

Important

For shared snapshots, copying incremental snapshots is not supported. For shared snapshots, all of the copies are full snapshots, even within the same region.

Depending on the AWS Regions involved and the amount of data to be copied, a cross-region snapshot copy can take hours to complete.

Copying a DB Cluster Snapshot Using the Console

If your source database engine is Neptune, then your snapshot is a DB cluster snapshot. For each AWS account, you can copy up to five DB cluster snapshots at a time per AWS Region. Copying both encrypted and unencrypted DB cluster snapshots is supported.

For more information about data transfer pricing, see [Neptune Pricing](#).

To cancel a copy operation after it is in progress, delete the target DB cluster snapshot while that DB cluster snapshot is in **copying** status.

The following procedure works for copying encrypted or unencrypted DB cluster snapshots:

To copy a DB cluster snapshot

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.

2. In the navigation pane, choose **Snapshots**.
3. Select the check box for the DB cluster snapshot you want to copy.
4. Choose **Actions**, and then choose **Copy Snapshot**. The **Make Copy of DB Snapshot** page appears.
5. Enter the name of the DB cluster snapshot copy in **New DB Snapshot Identifier**.
6. To copy tags and values from the snapshot to the copy of the snapshot, choose **Copy Tags**.
7. For **Enable Encryption**, choose one of the following options:
 - Choose **Disable encryption** if the DB cluster snapshot isn't encrypted and you don't want to encrypt the copy.
 - Choose **Enable encryption** if the DB cluster snapshot isn't encrypted but you want to encrypt the copy. In this case, for **Master Key**, specify the AWS KMS key identifier to use to encrypt the DB cluster snapshot copy.
 - Choose **Enable encryption** if the DB cluster snapshot is encrypted. In this case, you must encrypt the copy, so **Yes** is already selected. For **Master Key**, specify the AWS KMS key identifier to use to encrypt the DB cluster snapshot copy.
8. Choose **Copy Snapshot**.

Copying a DB Cluster Snapshot Using the AWS CLI

You can copy a DB snapshot using the [copy-db-cluster-snapshot](#) (p. 893) AWS CLI command.

If you are copying the snapshot to a new AWS Region, run the command in the new Region.

Use the following parameter descriptions and examples to determine which parameters to use in copying a snapshot with the AWS CLI.

- `--source-db-cluster-snapshot-identifier` – The identifier for the source DB snapshot.
 - If the source snapshot is in the same AWS Region as the copy, specify a valid DB snapshot identifier, like `neptune:instance1-snapshot-20130805`.
 - If the source snapshot is in a different AWS Region than the copy, specify a valid DB snapshot ARN like `arn:aws:neptune:us-west-2:123456789012:snapshot:instance1-snapshot-20130805`.
 - If you are copying from a shared manual DB snapshot, this parameter must be the Amazon Resource Name (ARN) of the shared DB snapshot.
 - If you are copying an encrypted snapshot, this parameter must be in the ARN format for the source AWS Region, and must match the `SourceDBSnapshotIdentifier` in the `PreSignedUrl` parameter.
- `--target-db-cluster-snapshot-identifier` – The identifier for the new copy of the encrypted DB snapshot.
- `--kms-key-id` – The AWS KMS key ID for an encrypted DB snapshot. The AWS KMS key ID is the Amazon Resource Name (ARN), AWS KMS key identifier, or the AWS KMS key alias for the AWS KMS encryption key.
 - If you copy an encrypted DB snapshot from your AWS account, you can specify a value for this parameter to encrypt the copy with a new AWS KMS encryption key. If you don't specify a value for this parameter, then the copy of the DB snapshot is encrypted with the same AWS KMS key as the source DB snapshot.
 - You cannot use this parameter to create an encrypted copy of an unencrypted snapshot. Trying to do so will generate an error.
 - If you copy an encrypted snapshot to a different AWS Region, then you must specify a AWS KMS key for the destination AWS Region. AWS KMS encryption keys are specific to the AWS Region that they are created in, and you cannot use encryption keys from one AWS Region in another AWS Region.

- `--source-region` – The ID of the AWS Region where the source DB snapshot is. If you copy an encrypted snapshot to a different AWS Region, then you must specify this option.
- `--region` – The ID of the AWS Region into which you are copying the snapshot. If you copy an encrypted snapshot to a different AWS Region, then you must specify this option.

Example From Unencrypted, To Same Region

The following code creates a copy of a snapshot, with the new name `mydbsnapshotcopy`, from the `us-east-1` AWS region to the `us-west-2` region.

For Linux, OS X, or Unix:

```
aws neptune copy-db-cluster-snapshot \  
  --source-db-cluster-snapshot-identifier instance1-snapshot-20130805 \  
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy
```

For Windows:

```
aws neptune copy-db-cluster-snapshot ^  
  --source-db-cluster-snapshot-identifier instance1-snapshot-20130805 ^  
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy
```

Example From Unencrypted, Across Regions

The following code creates a copy of a snapshot, with the new name `mydbsnapshotcopy`, from the `us-east-1` AWS region to the `us-west-2` region. Run the command in the `us-west-2` region.

For Linux, OS X, or Unix:

```
aws neptune copy-db-cluster-snapshot \  
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-east-1:123456789012:snapshot:instance1-snapshot-20130805 \  
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy \  
  --source-region us-east-1 \  
  --region us-west-2
```

For Windows:

```
aws neptune copy-db-cluster-snapshot ^  
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-east-1:123456789012:snapshot:instance1-snapshot-20130805 ^  
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy ^  
  --source-region us-east-1 ^  
  --region us-west-2
```

Example From Encrypted, Across Regions

The following code example copies an encrypted DB snapshot from the `us-east-1` AWS region to the `us-west-2` region. Run the command in the `us-west-2` region.

For Linux, OS X, or Unix:

```
aws neptune copy-db-cluster-snapshot \  
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-west-2:123456789012:snapshot:instance1-snapshot-20161115 \  
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy \  
  --source-region us-east-1 \  
  --region us-west-2
```

```
--region us-west-2  
--kms-key-id my_us_west_2_key
```

For Windows:

```
aws neptune copy-db-cluster-snapshot ^  
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-  
west-2:123456789012:snapshot:instance1-snapshot-20161115 ^  
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy ^  
  --source-region us-east-1 ^  
  --region us-west-2  
  --kms-key-id my-us-west-2-key
```

Sharing a DB Cluster Snapshot

Using Neptune, you can share a manual DB cluster snapshot in the following ways:

- Sharing a manual DB cluster snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to copy the snapshot.
- Sharing a manual DB cluster snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to directly restore a DB cluster from the snapshot instead of taking a copy of it and restoring from that.

Note

To share an automated DB cluster snapshot, create a manual DB cluster snapshot by copying the automated snapshot, and then share that copy.

For more information about restoring a DB cluster from a DB cluster snapshot, see [How to restore from a snapshot \(p. 680\)](#).

You can share a manual snapshot with up to 20 other AWS accounts. You can also share an unencrypted manual snapshot as public, which makes the snapshot available to all AWS accounts. Take care when sharing a snapshot as public so that none of your private information is included in any of your public snapshots.

Note

When you restore a DB cluster from a shared snapshot using the AWS Command Line Interface (AWS CLI) or Neptune API, you must specify the Amazon Resource Name (ARN) of the shared snapshot as the snapshot identifier.

Topics

- [Sharing an Encrypted DB Cluster Snapshot \(p. 686\)](#)
- [Sharing a DB Cluster Snapshot \(p. 688\)](#)

Sharing an Encrypted DB Cluster Snapshot

You can share DB cluster snapshots that have been encrypted "at rest" using the AES-256 encryption algorithm. For more information, see [Encrypting Neptune Resources at Rest \(p. 109\)](#). To do this, you must take the following steps:

1. Share the AWS Key Management Service (AWS KMS) encryption key that was used to encrypt the snapshot with any accounts that you want to be able to access the snapshot.

You can share AWS KMS encryption keys with another AWS account by adding the other account to the KMS key policy. For details on updating a key policy, see [Key Policies](#) in the *AWS KMS Developer Guide*. For an example of creating a key policy, see [Creating an IAM Policy to Enable Copying of the Encrypted Snapshot \(p. 688\)](#) later in this topic.

2. Use the AWS Management Console, AWS CLI, or Neptune API to share the encrypted snapshot with the other accounts.

These restrictions apply to sharing encrypted snapshots:

- You cannot share encrypted snapshots as public.
- You cannot share a snapshot that has been encrypted using the default AWS KMS encryption key of the AWS account that shared the snapshot.

Allowing Access to an AWS KMS Encryption Key

For another AWS account to copy an encrypted DB cluster snapshot shared from your account, the account that you share your snapshot with must have access to the KMS key that encrypted the snapshot. To allow another AWS account access to an AWS KMS key, update the key policy for the KMS key with the ARN of the AWS account that you are sharing to as a `Principal` in the KMS key policy. Then allow the `kms:CreateGrant` action.

After you have given an AWS account access to your KMS encryption key, to copy your encrypted snapshot, that AWS account must create an AWS Identity and Access Management (IAM) user if it doesn't already have one. In addition, that AWS account must also attach an IAM policy to that IAM user that allows the IAM user to copy an encrypted DB cluster snapshot using your KMS key. The account must be an IAM user and cannot be a root AWS account identity due to KMS security restrictions.

In the following key policy example, user `111122223333` is the owner of the KMS encryption key, and user `444455556666` is the account that the key is being shared with. This updated key policy gives the AWS account access to the KMS key by including the ARN for the root AWS account identity for user `444455556666` as a `Principal` for the policy, and by allowing the `kms:CreateGrant` action.

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```


Creating an IAM Policy to Enable Copying of the Encrypted Snapshot

After the external AWS account has access to your KMS key, the owner of that account can create a policy that allows an IAM user created for the account to copy an encrypted snapshot encrypted with that KMS key.

The following example shows a policy that can be attached to an IAM user for AWS account 444455556666. It enables the IAM user to copy a shared snapshot from AWS account 111122223333 that has been encrypted with the KMS key c989c1dd-a3f2-4a5d-8d96-e793d082ab26 in the us-west-2 Region.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUseOfTheKey",
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:RetireGrant"
      ],
      "Resource": [ "arn:aws:kms:us-west-2:111122223333:key/c989c1dd-a3f2-4a5d-8d96-e793d082ab26" ]
    },
    {
      "Sid": "AllowAttachmentOfPersistentResources",
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": [ "arn:aws:kms:us-west-2:111122223333:key/c989c1dd-a3f2-4a5d-8d96-e793d082ab26" ],
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": true
        }
      }
    }
  ]
}
```

For details on updating a key policy, see [Key Policies](#) in the *AWS KMS Developer Guide*.

Sharing a DB Cluster Snapshot

You can share a DB cluster snapshot using the AWS Management Console, the AWS CLI, or the Neptune API.

Using the Console to Share a DB Cluster Snapshot

Using the Neptune console, you can share a manual DB cluster snapshot with up to 20 AWS accounts. You can also stop sharing a manual snapshot with one or more accounts.

To share a manual DB cluster snapshot

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the manual snapshot that you want to share.
4. Choose **Actions, Share Snapshot**.
5. Choose one of the following options for **DB snapshot visibility**.
 - If the source is unencrypted, choose **Public** to permit all AWS accounts to restore a DB cluster from your manual DB cluster snapshot. Or choose **Private** to permit only AWS accounts that you specify to restore a DB cluster from your manual DB cluster snapshot.

Warning

If you set **DB snapshot visibility** to **Public**, all AWS accounts can restore a DB cluster from your manual DB cluster snapshot and have access to your data. Do not share any manual DB cluster snapshots that contain private information as **Public**.

- If the source is encrypted, **DB snapshot visibility** is set as **Private** because encrypted snapshots can't be shared as public.
6. For **AWS Account ID**, enter the AWS account identifier for an account that you want to permit to restore a DB cluster from your manual snapshot. Then choose **Add**. Repeat to include additional AWS account identifiers, up to 20 AWS accounts.

If you make an error when adding an AWS account identifier to the list of permitted accounts, you can delete it from the list by choosing **Delete** at the right of the incorrect AWS account identifier.
 7. After you add identifiers for all of the AWS accounts that you want to permit to restore the manual snapshot, choose **Save**.

To stop sharing a manual DB cluster snapshot with an AWS account

1. Open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the manual snapshot that you want to stop sharing.
4. Choose **Actions**, and then choose **Share Snapshot**.
5. To remove permission for an AWS account, choose **Delete** for the AWS account identifier for that account from the list of authorized accounts.
6. Choose **Save**.

Deleting a Neptune Snapshot

You can delete a DB snapshot using the AWS Management Console, the AWS CLI, or the Neptune management API:

Deleting Using the Console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB snapshot that you want to delete.
4. For **Actions**, choose **Delete Snapshot**.
5. Choose **Delete** on the confirmation page.

Deleting Using the AWS CLI

You can also delete a DB snapshot using the AWS CLI `delete_db_cluster_snapshot` (p. 891) command, using the `--db-snapshot-identifier` parameter to identify the snapshot you want to delete:

For Linux, OS X, or Unix:

```
aws neptune delete-db-cluster-snapshot \  
  --db-snapshot-identifier <name-of-the-snapshot-to-delete>
```

For Windows:

```
aws neptune delete-db-cluster-snapshot ^  
  --db-snapshot-identifier <name-of-the-snapshot-to-delete>
```

Deleting Using the Neptune Management API

You can use one of the SDKs to delete a DB snapshot by calling the `DeleteDBClusterSnapshot` (p. 891) API and use the `DBSnapshotIdentifier` parameters to identify the DB snapshot to be deleted.

Best Practices: Getting the Most Out of Neptune

The following are some general recommendations for working with Amazon Neptune. Use this information as a reference to quickly find recommendations for using Amazon Neptune and maximizing performance.

Contents

- [Amazon Neptune Basic Operational Guidelines \(p. 692\)](#)
 - [Amazon Neptune Security Best Practices \(p. 693\)](#)
 - [Enable the OSPG Index if you have a large number of predicates \(p. 693\)](#)
 - [Avoid long-running transactions where possible \(p. 693\)](#)
 - [Best Practices for Using Neptune Metrics \(p. 693\)](#)
 - [Best Practices for Tuning Neptune Queries \(p. 694\)](#)
 - [Load Balancing Across Read Replicas \(p. 694\)](#)
 - [Loading Faster Using a Temporary Larger Instance \(p. 694\)](#)
 - [Resize your writer instance by failing over to a read-replica \(p. 695\)](#)
 - [Avoid different instance classes in a cluster \(p. 695\)](#)
 - [Retry Upload after Data Prefetch Task Interrupted Error \(p. 695\)](#)
- [General Best Practices for Using Gremlin with Neptune \(p. 696\)](#)
 - [Test Gremlin code in the context where you will deploy it \(p. 697\)](#)
 - [Creating Efficient Multithreaded Gremlin Writes \(p. 697\)](#)
 - [Pruning Records with the Creation Time Property \(p. 698\)](#)
 - [Using the `datetime\(\)` Method for Groovy Time Data \(p. 698\)](#)
 - [Using Native Date and Time for GLV Time Data \(p. 698\)](#)
- [Best practices using the Gremlin Java client with Neptune \(p. 699\)](#)
 - [Use the latest compatible version of the Apache TinkerPop Java client \(p. 699\)](#)
 - [Re-use the client object across multiple threads \(p. 700\)](#)
 - [Create separate Gremlin Java client objects for read and write endpoints \(p. 700\)](#)
 - [Add multiple read replica endpoints to a Gremlin Java connection pool \(p. 700\)](#)
 - [Close the client to avoid the connections limit \(p. 700\)](#)
 - [Create a new connection after failover \(p. 701\)](#)
 - [Set `maxInProcessPerConnection` and `maxSimultaneousUsagePerConnection` to the same value \(p. 701\)](#)
 - [Send queries to the server as bytecode rather than as strings \(p. 701\)](#)
 - [Always completely consume the `ResultSet` or `Iterator` returned by a query \(p. 702\)](#)
 - [Bulk add vertices and edges in batches \(p. 702\)](#)
 - [Disable DNS caching in the Java Virtual Machine \(p. 703\)](#)
 - [Optionally, set timeouts at a per-query level \(p. 703\)](#)
 - [Use the Graphbinary Serialization Protocol \(p. 703\)](#)
 - [Work around a keep-alive bug in clients earlier than 3.3.4 \(p. 704\)](#)
- [Troubleshoot Gremlin API Version 2020-12-18 \(p. 704\)](#)

- [Troubleshoot java.util.concurrent.TimeoutException \(p. 704\)](#)
- [Neptune Best Practices Using SPARQL \(p. 705\)](#)
 - [Querying All Named Graphs by Default \(p. 705\)](#)
 - [Specifying a Named Graph for Load \(p. 705\)](#)
 - [Choosing Between FILTER, FILTER...IN, and VALUES in Your Queries \(p. 705\)](#)

Amazon Neptune Basic Operational Guidelines

The following are basic operational guidelines that you should follow when working with Neptune.

- Understand Neptune DB instances so that you can size them appropriately for your performance and use-case requirements. See [Amazon Neptune DB Clusters and Instances \(p. 73\)](#).
- Monitor your CPU and memory usage. This helps you know when to migrate to a DB instance class with greater CPU or memory capacity to achieve the query performance that you require. You can set up Amazon CloudWatch to notify you when usage patterns change or when you approach the capacity of your deployment. Doing so can help you maintain system performance and availability. See [Monitoring instances \(p. 74\)](#) and [Monitoring Neptune \(p. 639\)](#) for details.

Because Neptune has its own memory manager, it is normal to see relatively low memory usage even when CPU usage is high. Encountering out-of-memory exceptions when executing queries is the best indicator that you need to increase freeable memory.

- Enable automatic backups and set the backup window to occur at a convenient time.
- Test failover for your DB instance to understand how long the process takes for your use case. It also helps ensure that the application that accesses your DB instance can automatically connect to the new DB instance after failover.
- If possible, run your client and Neptune cluster in the same region and VPC, because cross-region connections with VPC peering can introduce delays in query response times. For single-digit millisecond query responses, it is necessary to keep the client and the Neptune cluster in the same region and VPC.
- When you create a read-replica instance, it should be at least as large as the primary writer instance. This helps keep replication lag in check, and avoids replica restarts. See [Avoid different instance classes in a cluster \(p. 695\)](#).
- Before upgrading to a new major engine version, be sure to test your application on it before you upgrade. You can do this by cloning your DB cluster so that the clone cluster runs the new engine version, and then test your application on the clone.
- To facilitate failovers, all instances should ideally be the same size.

Topics

- [Amazon Neptune Security Best Practices \(p. 693\)](#)
- [Enable the OSPG Index if you have a large number of predicates \(p. 693\)](#)
- [Avoid long-running transactions where possible \(p. 693\)](#)
- [Best Practices for Using Neptune Metrics \(p. 693\)](#)
- [Best Practices for Tuning Neptune Queries \(p. 694\)](#)
- [Load Balancing Across Read Replicas \(p. 694\)](#)
- [Loading Faster Using a Temporary Larger Instance \(p. 694\)](#)
- [Resize your writer instance by failing over to a read-replica \(p. 695\)](#)
- [Avoid different instance classes in a cluster \(p. 695\)](#)
- [Retry Upload after Data Prefetch Task Interrupted Error \(p. 695\)](#)

Amazon Neptune Security Best Practices

Use AWS Identity and Access Management (IAM) accounts to control access to Neptune API actions. Control actions that create, modify, or delete Neptune resources (such as DB instances, security groups, option groups, or parameter groups), and actions that perform common administrative actions (such as backing up and restoring DB instances).

- Assign an individual IAM account to each person who manages Amazon Relational Database Service (Amazon RDS) resources. Don't use AWS account root users to manage Neptune resources. Create an IAM user for everyone, including yourself.
- Grant each user the minimum set of permissions required to perform their duties.
- Use IAM groups to effectively manage permissions for multiple users.
- Rotate your IAM credentials regularly.

For more information about using IAM to access Neptune resources, see [Security in Amazon Neptune \(p. 104\)](#). For general information about working with IAM, see [AWS Identity and Access Management](#) and [IAM Best Practices](#) in the *IAM User Guide*.

Enable the OSPG Index if you have a large number of predicates

If your data model contains a large number of distinct predicates (more than thousand in most cases), you may experience reduced performance and higher operational costs.

If this is the case, you can improve performance by enabling the [OSPG index \(p. 62\)](#). See [The OSPG index \(p. 86\)](#).

Avoid long-running transactions where possible

Long-running transactions, either read-only or read-write, can cause unexpected problems of the following kinds:

A long-running transaction on a reader instance or on a writer instance with concurrent writes can result in a large accumulation of different versions of data. This can introduce higher latencies for read queries that filter out a large portion of their results.

In some cases, the accumulated versions over hours can cause new writes to be throttled.

A long-running read-write transaction with many writes can also cause issues if the instance restarts. If an instance restarts from a maintenance event or a crash, all uncommitted writes are rolled back. Such undo operations typically run in the background and do not block the instance from coming back up, but any new writes that conflict with the operations being rolled back then fail.

For example, if the same query is retried after the connection was severed in the previous run might fail when the instance is restarted.

The time needed for undo operations is proportional to the size of the changes involved.

Best Practices for Using Neptune Metrics

To identify performance issues caused by insufficient resources and other common bottlenecks, you can monitor the metrics available for your Neptune DB cluster.

Monitor performance metrics on a regular basis to gather data about the average, maximum, and minimum values for a variety of time ranges. This helps identify when performance is degraded. Using

this data, you can set Amazon CloudWatch alarms for particular metric thresholds so you are alerted if they are reached.

When you set up a new DB cluster and get it running with a typical workload, try to capture the average, maximum, and minimum values of all of the performance metrics at a number of different intervals (for example, one hour, 24 hours, one week, two weeks). This gives you an idea of what is normal. It helps to get comparisons for both peak and off-peak hours of operation. You can then use this information to identify when performance is dropping below standard levels, and can set alarms accordingly.

See [Monitoring Neptune Using Amazon CloudWatch \(p. 641\)](#) for information about how to view Neptune metrics.

The following are the most important metrics to start with:

- **BufferCacheHitRatio** — The percentage of requests that are served by the buffer cache. Cache misses add significant latency to query execution. If the cache hit ratio is below 99.9% and latency is an issue for your application, consider upgrading the instance type to cache more data in memory.
- **CPU utilization** — Percentage of computer processing capacity used. High values for CPU consumption might be appropriate, depending on your query-performance goals.
- **Freeable memory** — How much RAM is available on the DB instance, in megabytes. Neptune has its own memory manager, so this metric may be lower than you expect. A good sign that you should consider upgrading your instance class to one with more RAM is if queries often throw out-of-memory exceptions.

The red line in the **Monitoring** tab metrics is marked at 75% for CPU and Memory Metrics. If instance memory consumption frequently crosses that line, check your workload and consider upgrading your instance to improve query performance.

Best Practices for Tuning Neptune Queries

One of the best ways to improve Neptune performance is to tune your most commonly used and most resource-intensive queries to make them less expensive to run.

For information about how to tune Gremlin queries, see [Gremlin query hints \(p. 278\)](#) and [Tuning Gremlin queries \(p. 322\)](#). For information about how to tune SPARQL queries, see [SPARQL query hints \(p. 357\)](#).

Load Balancing Across Read Replicas

The reader endpoint round-robin routing works by changing the host that the DNS entry points to. The client must create a new connection and resolve the DNS record to get a connection to a new read replica, because WebSocket connections are often kept alive for long periods.

To get different read replicas for successive requests, ensure that your client resolves the DNS entry each time it connects. This may require closing the connection and reconnecting to the reader endpoint.

You can also load balance requests across read replicas by connecting to instance endpoints explicitly.

Loading Faster Using a Temporary Larger Instance

Your load performance increases with larger instance sizes. If you're not using a large instance type, but you want increased load speeds, you can use a larger instance to load and then delete it.

Note

The following procedure is for a new cluster. If you have an existing cluster, you can add a new larger instance and then promote it to a primary DB instance.

To load data using a larger instance size

1. Create a cluster with a single `r5.12xlarge` instance. This instance is the primary DB instance.
2. Create one or more read replicas of the same size (`r5.12xlarge`).

You can create the read replicas in a smaller size, but if they are not large enough to keep up with writes made by the primary instance, they may have to restart frequently. The resulting downtime reduces performance dramatically.

3. In the bulk loader command, include `"parallelism" : "OVERSUBSCRIBE"` to tell Neptune to use all available CPU resources for loading (see [Neptune Loader Request Parameters \(p. 210\)](#)). The load operation will then proceed as fast as I/O permits, which generally requires 60-70% of CPU resources.
4. Load your data using the Neptune loader. The load job runs on the primary DB instance.
5. After the data is finished loading, be sure to scale all the instances in the cluster down to the same instance type to avoid additional charges and repeated restart problems (see [Avoid different instance sizes \(p. 695\)](#)).

Resize your writer instance by failing over to a read-replica

The best way to resize an instance in your DB cluster, including the writer instance, is to create or modify a read-replica instance so that it has the size you want, and then deliberately fail over to that read-replica. The downtime seen by your application is only the time required to change the writer's IP address, which should be around 3 to 5 seconds.

The Neptune management API that you use to fail over the current writer instance to a read-replica instance deliberately is [FailoverDBCluster \(p. 818\)](#). If you are using the Gremlin Java client, you may need to create a new Client object after the failover to pick up the new IP address, as mentioned [here \(p. 701\)](#).

Make sure to change all your instances to the same size so that you avoid a cycle of repeated restarts, as mentioned below.

Avoid different instance classes in a cluster

When your DB cluster contains instances of different classes, problems can occur over time. The most common problem is that a small reader instance can get into a cycle of repeated restarts because of replication lag. If a reader node has a weaker DB instance class configuration than that of a writer DB instance, the volume of changes can be too big for the reader to catch up.

Important

To avoid repeated restarts caused by replication lag, configure your DB cluster so that all instances have the same instance class (size).

You can see the lag between the writer instance (the primary) and the readers in your DB cluster using the `ClusterReplicaLag` metric in Amazon CloudWatch. The `VolumeWriteIOPs` metric also lets you detect bursts of write activity in your cluster that can create replication lag.

Retry Upload after Data Prefetch Task Interrupted Error

When you are loading data into Neptune using the bulk loader, a `LOAD_FAILED` status may occasionally result, with a `PARSING_ERROR` and `Data prefetch task interrupted` message reported in response to a request for detailed information, like this:


```
"errorLogs" : [
  {
    "errorCode" : "PARSING_ERROR",
    "errorMessage" : "Data prefetch task interrupted: Data prefetch task for 11467 failed",
    "fileName" : "s3://some-source-bucket/some-source-file",
    "recordNum" : 0
  }
]
```

If you encounter this error, just retry the bulk upload request again.

The error occurs when there was a temporary interruption that was typically not caused by your request or your data, and it can usually be resolved by running the bulk upload request again.

If you are using default settings, namely "mode" : "AUTO", and "failOnError" : "TRUE", the loader skips the files that it already successfully loaded and resumes loading files it had not yet loaded when the interruption occurred.

General Best Practices for Using Gremlin with Neptune

Follow these recommendations when using the Gremlin graph traversal language with Neptune. For information about using Gremlin with Neptune, see [the section called "Gremlin" \(p. 262\)](#).

Important

A change was made in TinkerPop version 3.4.11 that improves correctness of how queries are processed, but for the moment can sometimes seriously impact query performance. For example, a query of this sort may run significantly slower:

```
g.V().hasLabel('airport').
  order().
  by(out().count(), desc).
  limit(10).
  out()
```

The vertices after the limit step are now fetched in a non-optimal way because of the TinkerPop 3.4.11 change. To avoid this, you can modify the query by adding the barrier() step at any point after the order().by(). For example:

```
g.V().hasLabel('airport').
  order().
  by(out().count(), desc).
  limit(10).
  barrier().
  out()
```

TinkerPop 3.4.11 was enabled in Neptune [engine version 1.0.5.0 \(p. 725\)](#).

Topics

- [Test Gremlin code in the context where you will deploy it \(p. 697\)](#)
- [Creating Efficient Multithreaded Gremlin Writes \(p. 697\)](#)
- [Pruning Records with the Creation Time Property \(p. 698\)](#)
- [Using the datetime\(\) Method for Groovy Time Data \(p. 698\)](#)
- [Using Native Date and Time for GLV Time Data \(p. 698\)](#)

Test Gremlin code in the context where you will deploy it

In Gremlin, there are multiple ways for clients to submit queries to the server: using WebSocket, or Bytecode GLV, or through the Gremlin console using string-based scripts.

It is important to recognize that Gremlin query execution can differ depending on how you submit the query. A query that returns an empty result might be treated as having succeeded if submitted in Bytecode mode, but be treated as having failed if submitted in script mode. For example, if you include `next()` in a script-mode query, the `next()` is sent to the server, but using ByteCode the client usually processes the `next()` itself. In the first case, the query fails if no results are found, but in the second, the query succeeds whether or not the result set is empty.

If you develop and test your code in one context (for example, the Gremlin console which generally submits queries in text form), but then deploy your code in a different context (for example through the Java driver using Bytecode) you can run into problems where your code behaves differently in production than it did in your development environment.

Important

Be sure to test Gremlin code in the GLV context where it will be deployed, to avoid unexpected results.

Creating Efficient Multithreaded Gremlin Writes

There are a few guidelines for multithreaded loading of data into Neptune using Gremlin.

If possible, give each thread a set of vertices or edges to insert or modify that do not collide. For example, thread 1 addresses ID range 1–50,000, thread 2 addresses ID range 50,001–100,000, and so on. This reduces the chance of hitting a `ConcurrentModificationException`. To be safe, put a `try/catch` block around all writes. If any fail, you can retry them after a short delay.

Batching writes in a batch size between 50 and 100 (vertices or edges) generally works well. If you have a lot of properties being added for each vertex, a number closer to 50 than 100 might be a better choice. Some experimentation is worthwhile. So for batched writes, you can use something like this:

```
g.addV('test').property(id,'1').as('a').
  addV('test').property(id,'2').
  addE('friend').to('a').
```

This is then repeated in each batch operation.

Using batches is significantly more efficient than adding one vertex or edge per Gremlin round trip to the server.

If you are using a Gremlin Language Variant (GLV) client, you can create a batch programmatically by first creating a traversal. Then add to it, and finally, iterate over it; for example:

```
t.addV('test').property(id,'1').as('a')
t.addV('test').property(id,'2')
t.addE('friend').to('a')
t.iterate()
```

It's best to use the Gremlin Language Variant client if possible. But you can do something similar with a client that submits queries as text strings by concatenating strings to build up a batch.

If you are using one of the Gremlin Client libraries rather than basic HTTP for queries, the threads should all share the same client, cluster, or connection pool. You might need to tune settings to get the best

possible throughput—settings such as the size of the connection pool and the number of worker threads that the Gremlin client uses.

Pruning Records with the Creation Time Property

You can prune stale records by storing the creation time as a property on vertices and dropping them periodically.

If you need to store data for a specific lifetime and then remove it from the graph (vertex time to live), you can store a timestamp property at the creation of the vertex. You can then periodically issue a `drop()` query for all vertices that were created before a certain time; for example:

```
g.V().has("timestamp", lt(datetime('2018-10-11')))
```

Using the `datetime()` Method for Groovy Time Data

Neptune provides the `datetime` method for specifying dates and times for queries sent in the Gremlin **Groovy** variant. This includes the Gremlin Console, text strings using the HTTP REST API, and any other serialization that uses Groovy.

Important

This *only* applies to methods where you send the Gremlin query as a *text string*. If you are using a Gremlin Language Variant, you must use the native date classes and functions for the language. For more information, see the next section, [the section called "Native Date and Time"](#) (p. 698).

You can use the `datetime` method to store and compare dates:

```
g.V('3').property('date', datetime('2001-02-08'))
```

```
g.V().has('date', gt(datetime('2000-01-01')))
```

Using Native Date and Time for GLV Time Data

If you are using a Gremlin Language Variant (GLV), you must use the native date and time classes and functions provided by the programming language for Gremlin time data.

The official TinkerPop Java, Node.js (JavaScript), Python, or .NET libraries are all Gremlin Language Variant libraries.

Important

This *only* applies to *Gremlin Language Variant (GLV) libraries*. If you are using a method where you send the Gremlin query as text string, you must use the `datetime()` method provided by Neptune. This includes the Gremlin Console, text strings using the HTTP REST API, and any other serialization that uses Groovy. For more information, see the preceding section, [the section called "`datetime\(\)`"](#) (p. 698).

Python

The following is a partial example in Python that creates a single property named 'date' for the vertex with an ID of '3'. It sets the value to be a date generated using the Python `datetime.now()` method.

```
import datetime
```

```
g.V('3').property('date', datetime.datetime.now()).next()
```

For a complete example for connecting to Neptune using Python, see [Using Python to connect to a Neptune DB instance \(p. 274\)](#)

Node.js (JavaScript)

The following is a partial example in JavaScript that creates a single property named 'date' for the vertex with an ID of '3'. It sets the value to be a date generated using the Node.js `Date()` constructor.

```
g.V('3').property('date', new Date()).next()
```

For a complete example for connecting to Neptune using Node.js, see [Using Node.js to connect to a Neptune DB instance \(p. 277\)](#)

Java

The following is a partial example in Java that creates a single property named 'date' for the vertex with an ID of '3'. It sets the value to be a date generated using the Java `Date()` constructor.

```
import java.util.date  
g.V('3').property('date', new Date()).next();
```

For a complete example for connecting to Neptune using Java, see [Using a Java client to connect to a Neptune DB instance \(p. 268\)](#)

.NET (C#)

The following is a partial example in C# that creates a single property named 'date' for the vertex with an ID of '3'. It sets the value to be a date generated using the .NET `DateTime.UtcNow` property.

```
Using System;  
g.V('3').property('date', DateTime.UtcNow).next()
```

For a complete example for connecting to Neptune using C#, see [Using .NET to connect to a Neptune DB instance \(p. 276\)](#)

Best practices using the Gremlin Java client with Neptune

Use the latest compatible version of the Apache TinkerPop Java client

If you can, always use the latest version of the Apache TinkerPop Gremlin Java client supported by the engine version you are using. Newer versions contain numerous bug fixes which can improve the stability, performance and usability of the client.

See [Apache TinkerPop Java Gremlin client \(p. 267\)](#) for a list of the client versions that are compatible with various Neptune engine versions.

Re-use the client object across multiple threads

Re-use the same client (or `GraphTraversalSource`) object across multiple threads. That is, create a shared instance of a `org.apache.tinkerpop.gremlin.driver.Client` class in your application rather than doing so in every thread. The `Client` object is thread safe, and the overhead of initializing it is considerable.

This also applies to `GraphTraversalSource`, which creates a `Client` object internally. For example, the following code causes a new `Client` object to be instantiated:

```
GraphTraversalSource traversal = EmptyGraph.instance()
                                .traversal()

    .withRemote(DriverRemoteConnection.using(cluster));
```

Create separate Gremlin Java client objects for read and write endpoints

You can increase performance by only performing writes on the writer endpoint and reading from one or more read-only endpoints.

```
Client readerClient = Cluster.build("https://reader-endpoint")
    ...
    .connect()

Client writerClient = Cluster.build("https://writer-endpoint")
    ...
    .connect()
```

Add multiple read replica endpoints to a Gremlin Java connection pool

When creating a Gremlin Java `Cluster` object, you can use the `.addContactPoint()` method to add multiple read replica instances to the connection pool's contact points.

```
Cluster.Builder readerBuilder = Cluster.build()
    .port(8182)
    .minConnectionPoolSize(...)
    .maxConnectionPoolSize(...)
    .....
    .addContactPoint("reader-endpoint-1")
    .addContactPoint("reader-endpoint-2")
```

Close the client to avoid the connections limit

It is important to close the client when you are finished with it to ensure that the `WebSocket` connections are closed by the server and all resources associated with the connections are released. This happens automatically if you close the cluster using `Cluster.close()`, because `client.close()` is then called internally.

If the client is not closed properly, Neptune terminates all idle `WebSocket` connections after 20 to 25 minutes. However, if you don't explicitly close `WebSocket` connections when you're done with them and the number of live connections reaches the [WebSocket concurrent connection limit \(p. 709\)](#), additional

connections are then refused with an HTTP 429 error code. At that point, you must restart the Neptune instance to close the connections.

The advice to call `cluster.close()` does not apply to Java AWS Lambda functions. See [Managing Gremlin WebSocket connections in AWS Lambda functions \(p. 507\)](#) for details.

Create a new connection after failover

In case of failover, the Gremlin Driver might continue connecting to the old writer because the cluster DNS name is resolved to an IP address. If this occurs, you can create a new `Client` object after failover.

Set `maxInProcessPerConnection` and `maxSimultaneousUsagePerConnection` to the same value

Both the `maxInProcessPerConnection` and the `maxSimultaneousUsagePerConnection` parameters are related to the maximum number of simultaneous queries you can submit on a single WebSocket connection. Internally, these parameters are co-related and modification of one without the other could lead to a client receiving a timeout while trying to fetch a connection from the client connection pool.

We recommend keeping the default minimum in-process and simultaneous usage values, and setting `maxInProcessPerConnection` and `maxSimultaneousUsagePerConnection` to the same value.

The value to set these parameters at is a function of query complexity and the data model. A use case where the query returns a lot of data would require more connection bandwidth per query and hence, should have lower values for the parameters, and a higher value for `maxConnectionPoolSize`.

By contrast, in a case where the query returns a smaller amount of data, `maxInProcessPerConnection` and `maxSimultaneousUsagePerConnection` should be set to a higher value than `maxConnectionPoolSize`.

Send queries to the server as bytecode rather than as strings

There are advantages to using bytecode rather than a string when submitting queries:

- **Catch invalid query syntax early:** Using the bytecode variant lets you detect invalid query syntax at the compilation stage. If you use the string-based variation, you won't discover the invalid syntax until the query is submitted to the server and an error is returned.
- **Avoid string-based performance penalty:** Any string-based query submission, whether you're using WebSockets or HTTP, results in a detached vertex, which implies that the Vertex object consists of the ID, Label and all the properties associated with the Vertex (see [Properties of Elements](#)).

This can lead to unnecessary computation on the server in cases where the properties are not required. For example, if the customer is interested in getting the vertex with the ID "hakuna#1" using the query, `g.V("hakuna#1")`. If the query is sent as a string based submission, the server would spend time in retrieving the ID, label and all properties for this vertex. If the query is sent as a bytecode submission, the server only spends time retrieving the ID and the label of the vertex.

In other words, rather than submit a query like this:

```
final Cluster cluster = Cluster.build("localhost")
```

```
        .port(8182)
        .maxInProcessPerConnection(32)
        .maxSimultaneousUsagePerConnection(32)
        .serializer(Serializers.GRAPHBINARY_V1D0)
        .create();

    try {
        final Client client = cluster.connect();
        List<Result> results =
client.submit("g.V().has('name', 'pumba').out('friendOf').id()).all().get();
        System.out.println(verticesWithNamePumba);
    } finally {
        cluster.close();
    }
}
```

Instead, submit the query using bytecode, like this:

```
final Cluster cluster = Cluster.build("localhost")
    .port(8182)
    .maxInProcessPerConnection(32)
    .maxSimultaneousUsagePerConnection(32)
    .serializer(Serializers.GRAPHBINARY_V1D0)
    .create();

try {
    final GraphTraversalSource g =
traversal().withRemote(DriverRemoteConnection.using(cluster));
    List<Object> verticesWithNamePumba = g.V().has("name",
"pumba").out("friendOf").id().toList();
    System.out.println(verticesWithNamePumba);
} finally {
    cluster.close();
}
}
```

Always completely consume the ResultSet or Iterator returned by a query

The client object should always completely consume the `ResultSet` (in the case of string-based submission), or the iterator returned by `GraphTraversal`. If the query results are not completely consumed, the server holds onto them, waiting for the client to finish consuming them.

If your application only needs a partial set of results, you can use a `limit(X)` step with your query to restrict the number of results that the server generates.

Bulk add vertices and edges in batches

Every query to the Neptune DB runs in the scope of a single transaction, unless you use a session. This means that if you need to insert a lot of data using gremlin queries, batching them together in a batch size of 50-100 improves performance by reducing the number of transactions created for the load.

As an example, adding 5 vertices to the database would look like this:

```
// Create a GraphTraversalSource for the remote connection
final GraphTraversalSource g =
traversal().withRemote(DriverRemoteConnection.using(cluster));
// Add 5 vertices in a single query
g.addV("Person").property(T.id, "P1")
.addV("Person").property(T.id, "P2")
.addV("Person").property(T.id, "P3")
```

```
.addV("Person").property(T.id, "P4")  
.addV("Person").property(T.id, "P5").iterate();
```

Disable DNS caching in the Java Virtual Machine

In an environment where you want to load-balance requests across multiple read replicas, you need to disable DNS caching in the Java Virtual Machine (JVM) and provide Neptune's reader endpoint while creating the cluster. Disabling the JVM DNS cache ensures that DNS is resolved again for every new connection so that the requests are distributed across all of the read replicas. You can do this in your application's initialization code with the following line:

```
java.security.Security.setProperty("networkaddress.cache.ttl", "0");
```

However, a more complete and robust solution for load-balancing is provided by the [Amazon Gremlin Java client code](#) on GitHub. The Amazon Java Gremlin client is aware of your cluster topology and fairly distributes connections and requests across a set of instances in your Neptune cluster. See [this blog post](#) for a sample Java Lambda function that uses that client.

Optionally, set timeouts at a per-query level

Neptune provides you with the ability to set a timeout for your queries using the parameter group option `neptune_query_timeout` (see [Parameters \(p. 409\)](#)). Starting with version 3.3.7 of the Java client, however, you can also override the global timeout, with code like this:

```
final Cluster cluster = Cluster.build("localhost")  
    .port(8182)  
    .maxInProcessPerConnection(32)  
    .maxSimultaneousUsagePerConnection(32)  
    .serializer(Serializers.GRAPHBINARY_V1D0)  
    .create();  
  
try {  
    final GraphTraversalSource g =  
traversal().withRemote(DriverRemoteConnection.using(cluster));  
    List<Object> verticesWithNamePumba = g.with(ARGS_EVAL_TIMEOUT, 500L).V().has("name",  
"pumba").out("friendOf").id().toList();  
    System.out.println(verticesWithNamePumba);  
} finally {  
    cluster.close();  
}
```

Or, for string-based query submission, the code would look like this:

```
RequestOptions options = RequestOptions.build().timeout(500).create();  
List<Result> result = client.submit("g.V()", options).all().get();
```

Use the Graphbinary Serialization Protocol

The GraphBinary serialization protocol introduced in client version 3.4.0 is faster than older protocols and improves serialization performance. Because this protocol is not enabled by default (as of client version 3.4.2), it should be enabled manually while building a cluster. For example:

```
final Cluster cluster = Cluster.build("Neptune endpoint")  
    .port(8182)  
    .serializer(Serializers.GRAPHBINARY_V1D0)
```



```
.create();
```

This serializer does materialize properties even in its string submission version.

Work around a keep-alive bug in clients earlier than 3.3.4

Only for client versions 3.3.3 and below:

There is a bug in earlier version of the Gremlin client causing a new `KeepAlive` request to be sent to the server with every query instead of once per websocket connection. This consumes precious server resources as the server processes unnecessary keep-alive requests. For more information, see [TINKERPOP-2030](#).

You can avoid this problem by upgrading to Gremlin Java Client version 3.3.4 or above.

A workaround is to disable the keep-alive from the client by setting its interval to 0:

```
Cluster.Builder readerBuilder = Cluster.build()
    .port(8182)
    ...
    .keepAliveInterval(0)
```

Troubleshoot Gremlin Java client exceptions

The Gremlin Java client throws exceptions that are specific to its operation. This section provides additional information to help you troubleshoot them.

Troubleshoot `java.util.concurrent.TimeoutException`

The Gremlin Java client throws a `java.util.concurrent.TimeoutException` when a Gremlin request times out at the client itself while waiting for a slot in one of the `WebSocket` connections to become available. This timeout duration is controlled by the `maxWaitForConnection` client-side configurable parameter.

Note

Because requests that time out at the client are never sent to the server, they aren't reflected in any of the metrics captured at the server, such as `GremlinRequestsPerSec`.

This kind of timeout is generally caused in one of two ways:

- **The server actually reached maximum capacity.** If this is the case, the queue on the server fills up, which you can detect by monitoring the [MainRequestQueuePendingRequests \(p. 644\)](#) CloudWatch metric. The number of parallel queries that the server can handle depends on its instance size.

If the `MainRequestQueuePendingRequests` metric doesn't show a build-up of pending requests on the server, then the server can handle more requests and the timeout is being caused by client-side throttling.

- **Client throttling of requests.** This can generally be fixed by changing client configuration settings.

The maximum number of parallel requests that the client can send can be roughly estimated as follows:

```
maxParallelQueries = maxConnectionPoolSize * Max( maxSimultaneousUsagePerConnection,
maxInProgressPerConnection )
```

Sending more than `maxParallelQueries` to the client causes

`java.util.concurrent.TimeoutException` exceptions. You can generally fix this in several ways:

- *Increase the connection timeout duration.* If latency is not crucial for your application, increase the client's `maxWaitForConnection` setting. The client then waits longer before it times out, which in turn can increase latency.
- *Increase the maximum requests per connection.* This allows more requests to be sent using the same WebSocket connection. Do this by increasing the client's `maxSimultaneousUsagePerConnection` and `maxInProcessPerConnection` settings. These settings should generally have the same value.
- *Increase the number of connections in the connection pool.* Do this by increasing the client's `maxConnectionPoolSize` setting. The cost is increased resource consumption, because each connection uses memory and an operating-system file descriptor, and requires an SSL and WebSocket handshake during initialization.

Neptune Best Practices Using SPARQL

Follow these best practices when using the SPARQL query language with Neptune. For information about using SPARQL in Neptune, see [Accessing the Neptune graph with SPARQL \(p. 342\)](#).

Querying All Named Graphs by Default

Amazon Neptune associates every triple with a named graph. The default graph is defined as the union of all named graphs.

If you submit a SPARQL query without explicitly specifying a graph via the `GRAPH` keyword or constructs such as `FROM NAMED`, Neptune always considers all triples in your DB instance. For example, the following query returns all triples from a Neptune SPARQL endpoint:

```
SELECT * WHERE { ?s ?p ?o }
```

Triples that appear in more than one graph are returned only once.

For information about the default graph specification, see the [RDF Dataset](#) section of the SPARQL 1.1 Query Language specification.

Specifying a Named Graph for Load

Amazon Neptune associates every triple with a named graph. If you don't specify a named graph when loading, inserting, or updating triples, Neptune uses the fallback named graph defined by the URI, `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`.

If you are using the Neptune bulk loader, you can specify the named graph to use for all triples (or quads with the fourth position blank) by using the `parserConfiguration: namedGraphUri` parameter. For information about the Neptune loader `Load` command syntax, see [the section called "Loader Command" \(p. 209\)](#).

Choosing Between FILTER, FILTER...IN, and VALUES in Your Queries

There are three basic ways to inject values in SPARQL queries: `FILTER`, `FILTER...IN`, and `VALUES`.

For example, suppose that you want to look up the friends of multiple people within a single query. Using `FILTER`, you might structure your query as follows:

```
PREFIX ex: <https://www.example.com/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
WHERE {?s foaf:knows ?o. FILTER (?s = ex:person1 || ?s = ex:person2)}
```

This returns all the triples in the graph that have `?s` bound to `ex:person1` or `ex:person2` and have an outgoing edge labeled `foaf:knows`.

You can also create a query using `FILTER . . . IN` that returns equivalent results:

```
PREFIX ex: <https://www.example.com/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
WHERE {?s foaf:knows ?o. FILTER (?s IN (ex:person1, ex:person2))}
```

You can also create a query using `VALUES` that in this case also returns equivalent results:

```
PREFIX ex: <https://www.example.com/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
WHERE {?s foaf:knows ?o. VALUES ?s {ex:person1 ex:person2}}
```

Although in many cases these queries are semantically equivalent, there are some cases where the two `FILTER` variants differ from the `VALUES` variant:

- The first case is when you inject duplicate values, such as injecting the same person twice. In that case, the `VALUES` query includes the duplicates in your result. You can explicitly eliminate such duplicates by adding a `DISTINCT` to the `SELECT` clause. But there might be situations where you actually want duplicates in the query results for redundant value injection.

However, the `FILTER` and `FILTER . . . IN` versions extract the value only once when the same value appears multiple times.

- The second case is related to the fact that `VALUES` always performs an exact match, whereas `FILTER` might apply type promotion and do fuzzy matching in some cases.

For instance, when you include a literal such as `"2.0"^^xsd:float` in your values clause, a `VALUES` query exactly matches this literal, including literal value and data type.

By contrast, `FILTER` produces a fuzzy match for these numeric literals. The matches could include literals with the same value but different numeric data types, such as `xsd:double`.

Note

There is no difference between the `FILTER` and `VALUES` behavior when enumerating string literals or URIs.

The differences between `FILTER` and `VALUES` can affect optimization and the resulting query evaluation strategy. Unless your use case requires fuzzy matching, we recommend using `VALUES` because it avoids looking at special cases related to type casting. As a result, `VALUES` often produces a more efficient query that runs faster and is less expensive.

Amazon Neptune Limits

Regions

Amazon Neptune is available in the following AWS Regions:

- US East (N. Virginia): `us-east-1`
- US East (Ohio): `us-east-2`
- US West (N. California): `us-west-1`
- US West (Oregon): `us-west-2`
- Canada (Central): `ca-central-1`
- South America (São Paulo): `sa-east-1`
- Europe (Stockholm): `eu-north-1`
- Europe (Ireland): `eu-west-1`
- Europe (London): `eu-west-2`
- Europe (Paris): `eu-west-3`
- Europe (Frankfurt): `eu-central-1`
- Middle East (Bahrain): `me-south-1`
- Asia Pacific (Hong Kong): `ap-east-1`
- Asia Pacific (Tokyo): `ap-northeast-1`
- Asia Pacific (Seoul): `ap-northeast-2`
- Asia Pacific (Singapore): `ap-southeast-1`
- Asia Pacific (Sydney): `ap-southeast-2`
- Asia Pacific (Mumbai): `ap-south-1`
- China (Beijing): `cn-north-1`
- China (Ningxia): `cn-northwest-1`
- AWS GovCloud (US-West): `us-gov-west-1`
- AWS GovCloud (US-East): `us-gov-east-1`

Differences in China regions

As is true of many AWS services, Amazon Neptune operates slightly differently in China (Beijing) and China (Ningxia) than in other AWS regions.

For example, when Neptune ML uses Amazon API Gateway to create its export service, IAM authentication is enabled by default. In China regions, the process for changing that option is slightly different than it is in other regions.

These and other differences are [explained here](#).

DB Instance Sizes Supported

Neptune supports different DB instance classes in different AWS Regions. To find out what classes are supported in a given Region, see [Amazon Neptune Pricing](#) and choose the Region that you are interested in.

Limits for Each AWS Account

For certain management features, Amazon Neptune uses operational technology that is shared with Amazon Relational Database Service (Amazon RDS).

Each AWS account has limits for each Region on the number of Amazon Neptune and Amazon RDS resources that you can create. These resources include DB instances and DB clusters.

After you reach a limit for a resource, additional calls to create that resource fail with an exception.

For a list of limits shared between Amazon Neptune and Amazon RDS, see [Limits in Amazon RDS](#) in the *Amazon RDS User Guide*.

Connection to Neptune Requires a VPC

Amazon Neptune is a virtual private cloud (VPC)-only service.

Additionally, instances do not allow access from outside the VPC.

Neptune requires SSL

Beginning with engine version 1.0.4.0, Amazon Neptune only allows Secure Sockets Layer (SSL) connections through HTTPS to any instance or cluster endpoint.

Neptune requires TLS version 1.2, using the following strong cipher suites:

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

Availability Zones and DB Subnet Groups

Amazon Neptune requires a DB subnet group for each cluster that has subnets in at least two supported Availability Zones.

We recommend using three or more subnets in different Availability Zones.

HTTP Request Payload Maximum (150 MB)

The total size of Gremlin and SPARQL HTTP requests must be less than 150 MB. If a request exceeds this size, Neptune returns `HTTP 400: BadRequestException`.

This limit does not apply to Gremlin WebSockets connections.

Gremlin Implementation Differences

The Amazon Neptune Gremlin implementation has specific implementation details that might differ from other Gremlin implementations.

For more information, see [Gremlin standards compliance in Amazon Neptune \(p. 47\)](#).

SPARQL UPDATE LOAD from URI

SPARQL UPDATE LOAD from URI works only with resources that are within the same VPC.

This includes Amazon S3 URLs in the same Region as the cluster with an Amazon S3 VPC endpoint created.

The Amazon S3 URL must be HTTPS, and any authentication must be included in the URL. For more information, see [Authenticating Requests: Using Query Parameters](#) in the *Amazon Simple Storage Service API Reference*.

For information about creating a VPC endpoint, see [Creating an Amazon S3 VPC Endpoint \(p. 204\)](#).

If you need to load data from a file, we recommend that you use the Amazon Neptune loader API. For more information, see [Using the Amazon Neptune Bulk Loader to Ingest Data \(p. 190\)](#).

Note

The Amazon Neptune loader API is non-ACID.

IAM Authentication and Access Control

IAM authentication and access control are only supported at the DB cluster level. They are not supported for Gremlin or SPARQL. For more information, see [Identity and Access Management in Amazon Neptune \(p. 111\)](#)

The Amazon Neptune console requires **NeptuneReadOnlyAccess** permissions. You can restrict access to IAM users by revoking this access. For more information, see [AWS Managed \(Predefined\) Policies for Amazon Neptune \(p. 118\)](#)

Amazon Neptune does not support user name/password-based access control.

WebSocket Concurrent Connections and Maximum Connection Time

The maximum number of concurrent WebSocket connections per database instance is different for each instance type. When that limit is reached, Neptune throttles any request to open a new WebSocket connection, to prevent using all of the allocated heap memory.

Instance Type	Maximum concurrent WebSocket connections
db.t3.medium	512
db.r5.large	2,048

Instance Type	Maximum concurrent WebSocket connections
db.r5.xlarge	4,096
db.r5.2xlarge	8,192
db.r5.4xlarge	16,384
db.r5.8xlarge	32,768
db.r5.12xlarge	32,768
db.r4.large	2,048
db.r4.xlarge	4,096
db.r4.2xlarge	8,192
db.r4.4xlarge	16,384
db.r4.8xlarge	32,768

When a client properly closes a connection, the closure is immediately reflected in the open connections count.

If the client doesn't close a connection, the connection may be closed automatically after a 20- to 25-minute idle timeout (the idle timeout is the time elapsed since the last message was received from the client). However, as long as the idle timeout is not reached, Neptune keeps the connection open indefinitely.

When IAM authentication is enabled, a WebSocket connection is always disconnected a few minutes more than 10 days after it was established, if it hasn't already been closed by then.

Limits on Properties and Labels

There is no limit on the number of vertices and edges, or RDF quads you can have in a graph.

There is also no limit on the number of properties or labels that any one vertex or edge can have.

There is a size limit of 55 MB on the size of an individual property or label. In RDF terms, this means that the value in any column (S, P, O or G) of an RDF quad cannot exceed 55 MB.

If you need to associate a larger object such as an image with a vertex or node in your graph, you can store it as a file in Amazon S3 and use the Amazon S3 path as the property or label.

Limits that affect the Neptune bulk loader

You cannot queue up more than 64 Neptune bulk load jobs at a time.

Neptune only keeps track of the most recent 1,024 bulk load jobs.

Neptune only stores the last 10,000 error details per job.

Neptune Service Errors

Amazon Neptune has two different sets of errors:

- The graph engine errors that are for the Neptune DB cluster endpoints only.
- The errors that are associated with the APIs for creating and modifying Neptune resources with the AWS SDK and AWS Command Line Interface (AWS CLI).

Topics

- [Graph Engine Error Messages and Codes \(p. 711\)](#)
- [DB Cluster Management API Error Messages and Codes \(p. 715\)](#)
- [Neptune Loader Error and Feed Messages \(p. 716\)](#)

Graph Engine Error Messages and Codes

Amazon Neptune endpoints return the standard errors for Gremlin and SPARQL when encountered.

Errors that are specific to Neptune can also be returned from the same endpoints. This section documents Neptune error messages, codes, and recommended actions.

Note

These errors are for the Neptune DB cluster endpoints only. The APIs for creating and modifying Neptune resources with the AWS SDK and AWS CLI have a different set of common errors. For information about those errors, see [the section called "API Errors" \(p. 715\)](#).

Graph Engine Error Format

Neptune error messages return a relevant HTTP error code and a JSON-formatted response.

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 465
Date: Thu, 15 Mar 2017 23:56:23 GMT

{
  "requestId": "0dbcde3-a9a1-4a25-b419-828c46342e47",
  "code": "ReadOnlyViolationException",
  "detailedMessage": "The request is rejected because it violates some read-only
  restriction, such as a designation of a replica as read-only."
}
```

Graph Engine Query Errors

The following table contains the error code, message, and HTTP status.

It also indicates whether it is OK to retry the request. Generally, it is OK to retry the request if it might succeed on a new try.

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
BadRequestException	400	No	The request could not be completed.
InternalFailureException	500	Yes	The request processing has failed.
CancelledByUserException	500	Yes	The request processing was cancelled by an authorized client.
InvalidParameterException	400	No	An invalid or out-of-range value was supplied for some input parameter or invalid syntax in a supplied RDF file.
MissingParameterException	400	No	A required parameter for the specified action is not supplied.
ReadOnlyViolationException	400	No	The request is rejected because it violates some read-only restriction, such as a designation of a replica as read-only.
MalformedQueryException	400	No	The request is rejected because it contains a query that is syntactically incorrect or does not pass additional validation.
BadRequestException	400	No	Request size exceeds max allowed value of 157286400 bytes.
MemoryLimitExceededException	500	Yes	The request processing did not succeed due to lack of memory, but can be retried when the server is less busy.
QueryLimitException	400	No	Size of query exceeds system limit.
TimeLimitExceededException	500	Yes	The request processing timed out.
UnsupportedOperationException	400	No	The request uses a currently unsupported feature or construct.
ConstraintViolationException	400	Yes	The query engine discovered, during the execution of

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
			the request, that the completion of some operation is impossible without violating some data integrity constraints, such as persistence of in- and out-vertices while adding an edge. Such conditions are typically observed if there are concurrent modifications to the graph, and are transient. The client should retry the request.
QueryLimitExceededException	500	Yes	The request processing did not succeed due to the lack of a limited resource, but can be retried when the server is less busy.
ConcurrentModificationException	500	Yes	The request processing did not succeed due to a modification conflict. The client should retry the request.
QueryTooLargeException	400	No	The request was rejected because its body is too large.
MethodNotAllowedException	405	No	The request is rejected because the chosen HTTP method is not supported by the used endpoint.
AccessDeniedException	403	No	Authentication or authorization failure.
TooManyRequestsException	429	Yes	Too many requests.
ThrottlingException	500	Yes	Rate of requests exceeds the maximum throughput. OK to retry.

IAM Authentication Errors

These errors are specific to cluster that have IAM authentication enabled.

The following table contains the error code, message, and HTTP status.

Neptune Service Error Code	HTTP status	Message
Incorrect IAM User/Policy	403	You do not have sufficient access to perform this action.
Incorrect or Missing Region	403	Credential should be scoped to a valid Region, not ' <i>region</i> '.
Incorrect or Missing Service Name	403	Credential should be scoped to correct service: 'neptune-db'.
Incorrect or Missing Host Header / Invalid Signature	403	The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the service documentation for details. Host header is missing or hostname is incorrect.
Missing X-Amz-Security-Token	403	'x-amz-security-token' is named as a SignedHeader, but it does not exist in the HTTP request
Missing Authorization Header	403	The request did not include the required authorization header, or it was malformed.
Missing Authentication Token	403	Missing Authentication Token.
Old Date	403	Signature expired: <i>20181011T213907Z</i> is now earlier than <i>20181011T213915Z</i> (<i>20181011T214415Z - 5 min.</i>)
Future Date	403	Signature not yet current: <i>20500224T213559Z</i> is still later than <i>20181108T225925Z</i> (<i>20181108T225425Z + 5 min.</i>)
Incorrect Date Format	403	Date must be in ISO-8601 'basic format'. Got ' <i>date</i> '. See https://en.wikipedia.org/wiki/ISO_8601 .
Unknown/Missing Access Key or Session Token	403	The security token included in the request is invalid.
Unknown/Missing Secret Key	403	The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the service documentation for details. Host header is missing or hostname is incorrect.

DB Cluster Management API Error Messages and Codes

These Amazon Neptune errors are associated with the APIs for creating and modifying Neptune resources with the AWS SDK and AWS CLI.

The following table contains the error code, message, and HTTP status.

Neptune Service Error Code	HTTP status	Message
<code>AccessDeniedException</code>	400	You do not have sufficient access to perform this action.
<code>IncompleteSignature</code>	400	The request signature does not conform to AWS standards.
<code>InternalFailure</code>	500	The request processing has failed because of an unknown error, exception, or failure.
<code>InvalidAction</code>	400	The action or operation requested is invalid. Verify that the action is typed correctly.
<code>InvalidClientTokenId</code>	403	The X.509 certificate or AWS access key ID provided does not exist in our records.
<code>InvalidParameterCombination</code>	400	Parameters that must not be used together were used together.
<code>InvalidParameterValue</code>	400	An invalid or out-of-range value was supplied for the input parameter.
<code>InvalidQueryParameter</code>	400	An invalid or out-of-range value was supplied for the input parameter.
<code>MalformedQueryString</code>	404	The query string contains a syntax error.
<code>MissingAction</code>	400	The request is missing an action or a required parameter.
<code>MissingAuthenticationToken</code>	403	The request must contain either a valid (registered) AWS access key ID or X.509 certificate.
<code>MissingParameter</code>	400	A required parameter for the specified action is not supplied.
<code>OptInRequired</code>	403	The AWS access key ID needs a subscription for the service.
<code>RequestExpired</code>	400	The request reached the service more than 15 minutes after the

Neptune Service Error Code	HTTP status	Message
		date stamp on the request or more than 15 minutes after the request expiration date (such as for presigned URLs), or the date stamp on the request is more than 15 minutes in the future.
ServiceUnavailable	503	The request has failed due to a temporary failure of the server.
ThrottlingException	400	The request was denied due to request throttling.
ValidationError	400	The input fails to satisfy the constraints specified by an AWS service.

Neptune Loader Error and Feed Messages

The following messages are returned by the `status` endpoint of the Neptune Loader. For more information, see [Get-Status API \(p. 216\)](#).

The following table contains loader feed code and description.

Error or Feed Code	Description
LOAD_NOT_STARTED	Load has been recorded but not started.
LOAD_IN_PROGRESS	Indicates that loading is in progress and specifies the number of files currently being loaded. When the loader parses a file, it creates one or more chunks to load in parallel. Because a single file can produce multiple chunks, the file count included with this message is usually less than the number of threads being used by the bulk load process.
LOAD_COMPLETED	Load has completed without any errors or errors within an acceptable threshold.
LOAD_CANCELLED_BY_USER	Load has been cancelled by user.
LOAD_CANCELLED_DUE_TO_ERRORS	Load has been cancelled by the system due to errors.
LOAD_UNEXPECTED_ERROR	Load failed with an unexpected error.
LOAD_FAILED	Load failed as a result of one or more errors.
LOAD_S3_READ_ERROR	Feed failed due to intermittent or transient Amazon S3 connectivity issues. If any of the feeds receive this error, overall load status is set to <code>LOAD_FAILED</code> .

Error or Feed Code	Description
LOAD_S3_ACCESS_DENIED_ERROR	Access was denied to the S3 bucket. If any of the feeds receive this error, overall load status is set to LOAD_FAILED.
LOAD_COMMITTED_W_WRITE_CONFLICTS	<p>Loaded data committed with unresolved write conflicts.</p> <p>The loader will try to resolve the write conflicts in separate transactions and update the feed status as the load progresses. If the final feed status is LOAD_COMMITTED_W_WRITE_CONFLICTS, then try resuming the load and it will likely succeed without write conflicts. A write conflict is not usually related to bad input data, but duplicates in data can increase the likelihood of write conflicts.</p>
LOAD_DATA_DEADLOCK	Load was automatically rolled back due to deadlock.
LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETED	Load failed because file was deleted or updated after load start.
LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED	The load request was not executed because its dependency check fails.
LOAD_IN_QUEUE	The load request has been queued up and is waiting to be executed.
LOAD_FAILED_INVALID_REQUEST	The load failed because the request was invalid (for example, the specified source/bucket may not exist, or the file format is invalid).

Engine Releases for Amazon Neptune

Amazon Neptune releases engine updates regularly.

You can determine which engine release version you currently have installed using the [instance-status API \(p. 639\)](#). This will also tell you whether you are running an original release (implicitly `.R1`), or a later patch (`.R2`, `.R3`, and so on). For more information about release numbering, see [Engine Version Numbers \(p. 780\)](#).

Note

For minor engine releases from `1.0.1.0` onwards, the implicit starting version is `.R1`, and subsequent patches are `.R2`, `.R3`, and so on.

For information about how to update your Neptune DB engine, see [Neptune Engine Updates \(p. 99\)](#).

The following is a list of the engine releases, along with information about how to upgrade:

Topics

- [Amazon Neptune Engine Version 1.1.0.0 \(2021-11-19\) \(p. 718\)](#)
- [Amazon Neptune Engine Version 1.0.5.1 \(2021-10-01\) \(p. 722\)](#)
- [Amazon Neptune Engine Version 1.0.5.0 \(2021-07-27\) \(p. 725\)](#)
- [Amazon Neptune Engine Version 1.0.4.2 \(2021-06-01\) \(p. 729\)](#)
- [Amazon Neptune Engine Version 1.0.4.1 \(2020-12-08\) \(p. 734\)](#)
- [Amazon Neptune Engine Version 1.0.4.0 \(2020-10-12\) \(p. 739\)](#)
- [Amazon Neptune Engine Version 1.0.3.0 \(2020-08-03\) \(p. 743\)](#)
- [Amazon Neptune Engine Version 1.0.2.2 \(2020-03-09\) \(p. 748\)](#)
- [Amazon Neptune Engine Version 1.0.2.1 \(2019-11-22\) \(p. 755\)](#)
- [Amazon Neptune Engine Version 1.0.2.0 \(2019-11-08\) \(p. 761\)](#)
- [Amazon Neptune Engine Version 1.0.1.2 \(2020-06-10\) \(p. 765\)](#)
- [Amazon Neptune Engine Version 1.0.1.1 \(2020-06-26\) \(p. 766\)](#)
- [Amazon Neptune Engine Version 1.0.1.0 \(2019-07-02\) \(p. 767\)](#)
- [Upgrading Your Neptune Engine \(p. 779\)](#)
- [Testing a new version of the Neptune engine before you upgrade \(p. 780\)](#)
- [Neptune Engine Version Numbering \(p. 780\)](#)
- [Using the Console to Upgrade Your Engine to a More Recent Version \(p. 781\)](#)
- [Using the Console to Enable Automatic Engine Updates \(p. 781\)](#)
- [Manually performing a minor upgrade of the Neptune DB Engine using the CLI \(p. 781\)](#)
- [Manually performing a major upgrade of the Neptune DB Engine using the CLI \(p. 782\)](#)
- [Using the CLI to Enable Automatic Neptune Engine Updates \(p. 783\)](#)

Amazon Neptune Engine Version 1.1.0.0 (2021-11-19)

As of 2021-11-19, engine version 1.1.0.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Important

Upgrading to this engine release also triggers an operating-system upgrade on all the instances in your DB cluster.

Before upgrading the engine, Neptune generates a snapshot with a name composed of `preupgrade` followed by an autogenerated identifier based on your DB cluster information. You will not be charged for this snapshot, and you can use it to restore your DB cluster if anything goes wrong during the upgrade process.

When the engine upgrade itself has completed, the new engine version will be available briefly on the old operating system, but in less than 5 minutes all the instances in your cluster will simultaneously begin an operating-system upgrade. Your DB cluster will be unavailable at this point for around 6 minutes.

This process generates the following events:

- Per-cluster event messages:
 - Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(*autogenerated snapshot ID*)]
 - Database cluster major version has been upgraded
- Per-instance event messages:
 - Applying off-line patches to DB instance
 - DB instance shutdown
 - Finished applying off-line patches to DB instance
 - DB instance restarted

New Features in This Engine Release

- Introduced general-purpose T4g and memory-optimized R6g database instances powered by the [AWS Graviton2 processor](#). Graviton2-based instances deliver significantly better price/performance than comparable current-generation x86-based instances for a variety of workloads. Applications work as normal on these new instance types, and there is no need to port application code when you upgrade to them.

For more information on pricing and regional availability, see the [Amazon Neptune pricing page](#).

- Introduced [custom models \(p. 595\)](#) in Neptune ML.
- Added support for [SPARQL inference queries \(p. 615\)](#) in Neptune ML.
- Added [a new streams endpoint \(p. 453\)](#) for property-graph data, namely:

```
https://Neptune-DNS:8182/propertygraph/stream
```

The output format of this endpoint, named `PG_JSON`, is exactly the same as the `GREMLIN_JSON` format output by the old `gremlin/stream`.

The new `propertygraph/stream` endpoint extends Neptune stream support to openCypher and replaces the `gremlin/stream` endpoint with its associated `GREMLIN_JSON` output format.

Improvements in This Engine Release

- Made improvements to Neptune streams:
 - Added a `commitTimestamp` field to the `records` object the [Neptune streams change-log response format \(p. 454\)](#), to provide a timestamp for each record in a change-log stream.
 - Added a `LATEST` value to the `iteratorType` parameter, allowing you to retrieve the last valid `eventId` from the streams. See [Calling the Streams API \(p. 453\)](#).

- Added support for getting the [inference confidence score \(p. 606\)](#) in Gremlin node classification and regression queries.
- Added support for the `OPTIONAL MATCH` clause in openCypher.
- Added support for the `MERGE` clause in openCypher.
- Added support for using `ORDER BY` in `WITH` clauses in openCypher.
- Added support for pattern comprehension in openCypher, and extended support for pattern expression beyond existence checking.
- Extended support for the `DELETE` and `DELETE DETACH` clauses in openCypher, so that they can now be used with other update clauses.
- Extended support for `CREATE` and `UPDATE` clauses used with `RETURN` in openCypher.
- Added support in the DFE engine for the Gremlin `limit`, `range`, and `skip` steps.
- Improved query execution in the DFE engine when neither `explain` nor `profile` is requested.
- Improved query execution in the DFE engine for the `value` expression.
- Improved a number of chained Gremlin conditional insert patterns so as to avoid concurrent-modification exceptions and allow chaining of query patterns like these:
 - Conditional vertex insertion by ID, such as:

```
g.V(ID).fold().coalesce(unfold(), g.addV("L1").property(id, ID))
```

- Conditional vertex insertion with multiple labels, such as:

```
g.V(ID).fold().coalesce(unfold(), g.addV("L1:L2").property(id, ID))
```

- Conditional edge insertion by ID, such as:

```
g.E(ID).fold().coalesce(unfold(), V(from).addE(label).to(V(to)).property(id, ID))
```

- Conditional edge insertion with multiple labels, such as:

```
g.E(ID).fold().coalesce(unfold(),  
g.addE(label).from(V(from)).to(V(to)).property(id, ID))
```

- Conditional insertion followed by a query, such as:

```
g.V(ID).fold().coalesce(unfold(),  
g.addV("L1").property(id, ID)).project("myvalues").by(valueMap())
```

- Conditional insertion with added properties, such as:

```
g.V(ID).fold().coalesce(unfold(),  
g.addV("L1").property(id, ID).property("name", "pumba"))
```

Defects Fixed in This Engine Release

Important

Fixed an upgrade issue so that the auto-minor-version upgrade settings are now honored across engine versions. For example, when auto minor version upgrade is enabled on a version 1.1.0.0 engine, the engine will now be upgraded automatically to version 1.1.0.1 during the next maintenance window.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20 or 30 seconds to several minutes, after which you can resume using the DB cluster.

- Disabled the [statistics \(p. 88\)](#) feature on T3.medium instance types, which were not able to support it.
- Fixed a SPARQL bug in `explain` with an `IN` function that took non-constant values.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.1.0.0, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.11
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.1.0.0

You can manually upgrade any previous Neptune engine release to this release.

You will not automatically upgrade to this release.

Upgrading to This Release

Amazon Neptune 1.1.0.0 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine neptune \  
  --engine-version 1.1.0.0 \  
  --allow-major-version-upgrade \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine neptune ^  
  --engine-version 1.1.0.0 ^  
  --allow-major-version-upgrade ^  
  --apply-immediately
```

Instead of `--apply-immediately`, you can specify `--no-apply-immediately`. To perform a major version upgrade, the `allow-major-version-upgrade` parameter is required. Also, be sure to include the engine version or your engine may be upgraded to a different version.

If your cluster uses a custom cluster parameter group, be sure to include this parameter to specify it:

```
--db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

Similarly, if any instances in the cluster use a custom DB parameter group, be sure to include this parameter to specify it:

```
---db-instance-parameter-group-name (name of the custom instance parameter group)
```

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.5.1 (2021-10-01)

As of 2021-10-01, engine version 1.0.5.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.5.1.R2 \(2021-10-26\) \(p. 724\)](#)

New Features in This Engine Release

- Added a [results cache \(p. 288\)](#) for caching the results of specified queries.
- Added Date/time support in Neptune openCypher.
- Added support for `List` and `Map` access to elements in Neptune openCypher.

Improvements in This Engine Release

- Made Neptune openCypher endpoint names case-insensitive.
- Improved openCypher explain.
- Improved Gremlin single upsert query patterns terminating with `iterate()` and `profile()` steps.
- Improved performance in Gremlin `keys()` and `property()` functions.
- The Gremlin `dedup()` step is run in the DFE when it is used with global scope.
- The following Gremlin `HAS` predicates are run in the DFE engine when the DFE engine is enabled:
 - `EQ`
 - `NEQ`
 - `LT`
 - `LTE`
 - `GT`
 - `GTE`
 - `BETWEEN`
 - `INSIDE`
 - `OUTSIDE`
 - `WITHIN`
 - `AND` (connectives)
 - `OR` (connectives)
- Improved `LIMIT` query performance.
- Improved performance of openCypher general aggregation queries.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug that allowed an edge to be connected to another edge.
- Fixed a Gremlin bug that caused a sub-optimal join strategy to be chosen.
- Fixed a Gremlin bug that caused serialization of nodes and relationships to stall when more than 100 properties were present.
- Fixed a bug that slowed down query execution planning for queries with large graph patterns.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.5.1, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.11
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.5.1

You can manually upgrade any previous Neptune engine release to this release.

You will not automatically upgrade to this release.

Upgrading to This Release

Amazon Neptune 1.0.5.1 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.5.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.5.1 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.5.1.R2 (2021-10-26)

As of 2021-10-26, engine version 1.0.5.1.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Fixed a bug that caused a server restart when a transient error occurred while creating an older version of a graph element, under repeatable read isolation. Neptune now returns an error instead, so that the client can retry.
- Fixed a bug that caused a server restart when a transient error occurred during a single cardinality update. Neptune now returns an error instead, so that the client can retry.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.5.1.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.11
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.5.1.R2

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.5.1.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.5.1.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.5.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^ \  
  --db-cluster-identifier (your-neptune-cluster) ^ \  
  --engine-version 1.0.5.1 ^ \  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.5.0 (2021-07-27)

As of 2021-07-27, engine version 1.0.5.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.5.0.R3 \(2021-09-15\) \(p. 727\)](#)
- [Release: 1.0.5.0.R2 \(2021-08-16\) \(p. 728\)](#)

New Features in This Engine Release

- [Neptune ML \(p. 522\)](#) was released for production use with many new features, and is no longer in lab mode.
- Added initial support for the [openCypher \(p. 392\)](#) query language, in Lab Mode. **openCypher** is the open-source standard for the Cypher query language. Its syntax is specified in the [Cypher Query Language Reference \(Version 9\)](#), and is maintained by the [openCypher](#) project.

See [Accessing the Neptune Graph with openCypher \(p. 392\)](#) for information about the Neptune implementation of the language.

Support for the [Bolt protocol](#), which Neptune clients use for openCypher queries, is also supported. See [Using the Bolt protocol to make openCypher queries to Neptune \(p. 399\)](#).

Support for openCypher is now automatically enabled, but it depends on the [Neptune DFE engine \(p. 87\)](#), which is currently only available in [lab mode \(p. 85\)](#). The default `DFEQueryEngine` setting in the `neptune_lab_mode` DB cluster parameter is now `DFEQueryEngine=viaQueryHint`, which means that the engine is enabled but only used for queries that have the `useDFE` query hint present and set to `true`. If you disable the DFE engine by setting `DFEQueryEngine=disabled`, you will not be able to use openCypher.

- Added support for the [SPARQL 1.1 Graph Store HTTP Protocol](#). See [Using the SPARQL 1.1 Graph Store HTTP Protocol \(GSP\) in Amazon Neptune \(p. 368\)](#).
- Changed the default lab-mode setting for the [Neptune DFE engine \(p. 87\)](#) to `viaQueryHint`, which means that the DFE engine is now enabled by default, but only used for queries that have the `useDFE` query hint present and set to `true`.
- Added a new Amazon CloudWatch metric, `StatsNumStatementsScanned`, for monitoring the computation of statistics for the Neptune DFE engine. See [Using the StatsNumStatementsScanned CloudWatch metric to monitor statistics computation \(p. 92\)](#).

Improvements in This Engine Release

- Added support for Apache TinkerPop 3.4.11.

Important

A change was made in TinkerPop version 3.4.11 that improves correctness of how queries are processed, but for the moment can sometimes seriously impact query performance. For example, a query of this sort may run significantly slower:

```
g.V().hasLabel('airport').
  order().
    by(out().count(), desc).
  limit(10).
  out()
```

The vertices after the limit step are now fetched in a non-optimal way because of the TinkerPop 3.4.11 change. To avoid this, you can modify the query by adding the `barrier()` step at any point after the `order().by()`. For example:

```
g.V().hasLabel('airport').
  order().
    by(out().count(), desc).
  limit(10).
  barrier().
  out()
```

- The [SPARQL joinOrder query hint \(p. 359\)](#) is now supported by the Neptune DFE alternative query engine.
- The output of the [Neptune status API \(p. 639\)](#) has been expanded and reorganized to provide more clarity about your DB cluster's settings and features.

The new output has a top-level `features` object that contains status information about your DB cluster's features, and a top-level `settings` object that contains settings information. To review the new format, see [Example of the output from the instance status command \(p. 641\)](#).

- Handling of streaming change logs has been improved when `AFTER_SEQUENCE_NUMBER` streams are requested with the last event ID on the server, when that event ID has already expired. The server no longer throws an expired event ID error if the requested event ID is the most recently purged event ID on the server.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug related to the ordering of numeric values.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.5.0, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.11
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.5.0

You can manually upgrade any previous Neptune engine release to this release.

You will not automatically upgrade to this release.

Upgrading to This Release

Amazon Neptune 1.0.5.0 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.5.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.5.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.5.0.R3 (2021-09-15)

As of 2021-09-15, engine version 1.0.5.0.R3 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Fixed a bug that causes the engine to become unresponsive in either of these situations:
 - A bulk load happens at the same time as automatic statistics computation is taking place.
 - A statistics computation was requested manually at the same time that one was already occurring.
- Fixed a bug in deadlock detection and in lock acquisition that could cause the engine to crash.
- Fixed a Gremlin bug where the engine threw an error when it encountered unknown data from a remote ML endpoint in a Gremlin inference query.
- Fixed several bugs in ML model management APIs related to model transform jobs and instance recommendations.
- Fixed a bug that could cause the engine to crash when generating node and edge IDs.
- Fixed a bug that slowed down the generation of query plans for queries with large graph patterns.
- Fixed an openCypher bug that could cause a query to stall when retrieving a node that had more than 100 properties.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.5.0.R3, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.11
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.5.0.R3

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.5.0.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.5.0.R3 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.5.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.5.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.5.0.R2 (2021-08-16)

As of 2021-08-16, engine version 1.0.5.0.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Disabled an optimization made in [engine release 1.0.5.0 \(p. 725\)](#) that made the [Neptune lookup cache \(p. 64\)](#) survive engine restarts on replicas. Replica restarts now clear the lookup cache.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.5.0.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.11
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.5.0.R2

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.5.0.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.5.0.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.5.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.5.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.4.2 (2021-06-01)

Note

Engine release version 1.0.4.2.R2 was the first version of 1.0.4.2 actually to be released.

Topics

- [Amazon Neptune Engine Version 1.0.4.2.R5 \(2021-08-16\) \(p. 730\)](#)
- [Amazon Neptune Engine Version 1.0.4.2.R4 \(2021-07-23\) \(p. 730\)](#)

- [Amazon Neptune Engine Version 1.0.4.2.R3 \(2021-06-28\)](#) (p. 731)
- [Amazon Neptune Engine Version 1.0.4.2.R2 \(2021-06-01\)](#) (p. 732)
- [Amazon Neptune Engine Version 1.0.4.2.R1 \(2021-05-27\)](#) (p. 734)

Amazon Neptune Engine Version 1.0.4.2.R5 (2021-08-16)

As of 2021-08-16, engine version 1.0.4.2.R5 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Disabled an optimization made in [engine release 1.0.4.2.R4](#) (p. 730) that made the [Neptune lookup cache](#) (p. 64) survive engine restarts on replicas. Replica restarts now clear the lookup cache.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.2.R5, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.10
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.4.2.R5

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.4.2.

You can manually upgrade any previous Neptune engine release to this release.

Amazon Neptune Engine Version 1.0.4.2.R4 (2021-07-23)

As of 2021-07-23, engine version 1.0.4.2.R4 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Improvements in This Engine Release

- Improved the behavior of the lookup cache to avoid redundant cache clearing after running fast reset on a replica.
- Improved handling of streaming change logs when `AFTER_SEQUENCE_NUMBER` streams are requested with the last event ID on the server, when that event ID has already expired. The server no longer throws an expired event ID error if the requested event ID is the most recently purged event ID on the server.

Defects Fixed in This Engine Release

- Fixed a bug introduced in 1.0.4.0.R1 where queries would not return the entirety of string values larger than 760 characters. The terms affected by this bug were RDF literals and URIs, or Gremlin IDs, keys, and string values.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.2.R4, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.10
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.4.2.R4

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.4.2.

You can manually upgrade any previous Neptune engine release to this release.

Amazon Neptune Engine Version 1.0.4.2.R3 (2021-06-28)

As of 2021-06-28, engine version 1.0.4.2.R3 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Known issues in this engine release

Issue:

A SPARQL bug that fails to honor media type in an `Accept` header if there are spaces present.

For example, a query with `-H "Accept: text/csv; q=1.0, */*; q=0.1"` returns JSON output rather than CSV output.

Workaround:

If you remove the spaces in the `Accept` clause in the header, the engine returns output in the correct requested format. In other words, instead of `-H "Accept: text/csv; q=1.0, */*; q=0.1"`, use:

```
-H "Accept: text/csv;q=1.0,*/*;q=0.1"
```

Defects Fixed in This Engine Release

- Fixed a bug in clearing the lookup cache on replicas after a fast reset.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.2.R3, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.10
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.4.2.R3

This patch release is optional unless your DB cluster is using one or more R5d instances. If your cluster has R5d instances, it will automatically be upgraded in the next maintenance window. Otherwise, it will not automatically be upgraded to this patch release.

You can upgrade release 1.0.4.2.R2 to this 1.0.4.2.R3 release manually using the AWS CLI [apply-pending-maintenance-action](#) command (the [ApplyPendingMaintenanceAction](#) (p. 932) API).

Amazon Neptune Engine Version 1.0.4.2.R2 (2021-06-01)

As of 2021-06-01, engine version 1.0.4.2.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.4.2.R3 \(2021-06-28\) \(p. 731\)](#)

Known issues in this engine release

Issue:

A SPARQL bug that fails to honor media type in an `Accept` header if there are spaces present.

For example, a query with `-H "Accept: text/csv; q=1.0, */*; q=0.1"` returns JSON output rather than CSV output.

Workaround:

If you remove the spaces in the `Accept` clause in the header, the engine returns output in the correct requested format. In other words, instead of `-H "Accept: text/csv; q=1.0, */*; q=0.1"`, use:

```
-H "Accept: text/csv;q=1.0,*/*;q=0.1"
```

New Features in This Engine Release

- Added the new R5d instance type, which includes a lookup cache for speeding up reads in use cases involving a high volume of property value or RDF literal lookups. See [The Neptune lookup cache can accelerate read queries \(p. 64\)](#).
- Added a new lab-mode parameter that lets the experimental DFE engine be invoked only on a per-query basis with the `useDFE` query hint.

Improvements in This Engine Release

- Added support for TinkerPop 3.4.10.
- Added support for using the `withStrategies()` configuration step when sending Gremlin script requests. Specifically, the `SubgraphStrategy`, `PartitionStrategy`, `ReadOnlyStrategy`, `EdgeLabelVerificationStrategy`, and `ReservedKeysVerificationStrategy` are all supported.
- Added optimization for `V()` traversals in the middle of a query. Previously, such traversals were not optimized in Neptune.

- Added support for [RFC 2141 URNs](#) to be used as the `baseUri` and `namedGraphUri` parameters for a bulk load.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug in the parser where incorrect queries were treated as valid.
- Fixed a Gremlin bug where unfolding an `aggregate()` side-effect with `cap().unfold()` to a `valueMap()` would raise an exception.
- Fixed a Gremlin bug where some `property()` steps after an `addV()` step fail with a "cannot cast to String" error.
- Fixed a Gremlin bug to prevent some conditional insert patterns from raising concurrent-modification exceptions.
- Fixed a Gremlin bug so that the query request timeout now cannot exceed the session timeout.
- Fixed a SPARQL bug where updates using `LOAD` or `UNLOAD` could fail with an HTTP code 500 instead of HTTP code 400 when the remote server is unavailable.
- Fixed a bug where stream API calls were failing when `commitNum` or `opNum` values larger than the 32-bit signed integer limit (2,147,483,647) were used.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.2.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.10
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.4.2.R2

You can manually upgrade any previous Neptune engine release to this release.

You will not automatically upgrade to this release.

Upgrading to This Release

Amazon Neptune 1.0.4.2.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.4.2 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.4.2 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.4.2.R1 (2021-05-27)

Engine release 1.0.4.2.R1 was never deployed.

Amazon Neptune Engine Version 1.0.4.1 (2020-12-08)

As of 2020-12-08, engine version 1.0.4.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.4.1.R1.1 \(2021-03-22\) \(p. 736\)](#)
- [Release: 1.0.4.1.R2 \(2021-02-24\) \(p. 737\)](#)

Important

[Release: 1.0.4.0 \(2020-10-12\) \(p. 739\)](#) made TLS 1.2 and HTTPS mandatory for all connections to Amazon Neptune. However, a bug in that release has allowed HTTP connections and/or outdated TLS connections to continue to work for customers who previously set a DB cluster parameter to prevent enforcement of HTTPS connections. That bug was fixed in patch releases [1.0.4.0.R2 \(p. 741\)](#) and [1.0.4.1.R2 \(p. 737\)](#), but the fix has caused unexpected connection failures when the patches are automatically installed. For this reason, both patches have been reverted, and can only be installed manually, to give you a chance to update your setup for TLS 1.2.

Having to use SSL/TLS for all connections to Neptune affects your connections with the Gremlin console, the Gremlin driver, Gremlin Python, .NET, nodeJs, REST APIs, and also load-balancer connections. If you have been using HTTP or an older TLS version for any or all of these up until now, you must update the relevant client and drivers and change your code to use HTTPS exclusively before updating your system to the latest patches.

New Features in This Engine Release

- Introduced the Neptune ML feature, which brings powerful machine learning capabilities to Amazon Neptune. See [Amazon Neptune ML for machine learning on graphs \(p. 522\)](#).
- Added a custom SPARQL UNLOAD operation for removing data retrieved from a remote source. See [SPARQL UPDATE UNLOAD \(p. 355\)](#).

Improvements in This Engine Release

- Optimized some Gremlin conditional insert patterns to avoid concurrent-modification exceptions.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug that could cause missing results for a specific pattern of queries that used the `as()` step.
- Fixed a Gremlin bug that could cause errors when using the `project()` step nested inside another step such as `union()`.
- Fixed a Gremlin bug in the `project()` step.
- Fixed a Gremlin bug in string-based traversal where the `none()` step did not work.
- Fixed a Gremlin bug in string-based traversal where an empty map was not supported as an argument to the `inject()` step.
- Fixed a Gremlin bug in string-based traversal execution in the DFE engine where a terminal method such as `toList()` did not work properly.
- Fixed a Gremlin bug that failed to close transactions which used the `iterate()` step in String queries.
- Fixed a Gremlin bug that could cause queries using the `is(P.gte(0))` pattern to throw an exception in some situations.
- Fixed a Gremlin bug that could cause queries using the `order().by(T.id)` pattern to throw an exception in some situations.
- Fixed a Gremlin bug that could cause queries using the `addV().aggregate()` pattern to give incorrect results in some situations.
- Fixed a Gremlin bug that could cause queries using the `path()` step followed by the `project()` step pattern to throw an exception in some situations.
- Fixed a SPARQL bug where the `SUBSTR` function signals an error instead of returning an empty string.
- Fixed a bug in the DFE engine that could cause join operations in non-blocking query plans to generate incorrect results in the presence of unbound variables.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.1, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.8
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.4.1

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.4.1.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.4.1 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \
```



```
--db-cluster-identifier (your-neptune-cluster) \  
--engine-version 1.0.4.1 \  
--apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
--db-cluster-identifier (your-neptune-cluster) ^  
--engine-version 1.0.4.1 ^  
--apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.4.1.R1.1 (2021-03-22)

As of 2021-03-22, engine version 1.0.4.1.R1.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Disabled an optimization for Gremlin conditional insert patterns which can add or append to existing labels and properties.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.1.R1.1, make sure that your project is compatible with these query-language versions:

- *Gremlin version*: 3.4.8
- *SPARQL version*: 1.1

Upgrade Paths to Engine Release 1.0.4.1.R1.1

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.4.1.

Upgrading to This Release

Amazon Neptune 1.0.4.1.R1.1 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
--db-cluster-identifier (your-neptune-cluster) \  
--apply-immediately
```

```
--engine-version 1.0.4.1 \  
--apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
--db-cluster-identifier (your-neptune-cluster) ^  
--engine-version 1.0.4.1 ^  
--apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.4.1.R2 (2021-02-24)

As of 2021-02-24, engine version 1.0.4.1.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.4.1.R2.1 \(2021-03-11\) \(p. 738\)](#)

New Features in This Engine Release

- Neptune now supports compression of single files in `bzip2` format for bulk loads. See [Load Data Formats \(p. 194\)](#).

Defects Fixed in This Engine Release

- Fixed a bug in [Release: 1.0.4.0 \(2020-10-12\) \(p. 739\)](#) that allowed connections to Neptune using `HTTP` or earlier versions of `TLS`, rather than `HTTPS` and `TLS 1.2`.

Important

Having to use SSL/TLS for all connections to Neptune can be a breaking change. It affects your connections with the Gremlin console, the Gremlin driver, Gremlin Python, .NET, nodeJs, REST APIs, and also load-balancer connections. If you have been using `HTTP` or an older `TLS` version for any or all of these up until now, you must update the relevant client and drivers before installing this patch, and change your code to use `HTTPS` exclusively.

- Fixed a Gremlin bug where `InternalFailureException` was set as the response code in certain circumstances when a `ConcurrentModificationException` occurred.
- Fixed a Gremlin bug where under certain conditions updating edges or vertices could cause a transient `InternalFailureException`.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.1.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3 . 4 . 8
- *SPARQL version:* 1 . 1

Upgrade Paths to Engine Release 1.0.4.1.R2

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1 . 0 . 4 . 1 .

Upgrading to This Release

Amazon Neptune 1.0.4.1.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.4.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.4.1 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.4.1.R2.1 (2021-03-11)

As of 2021-03-11, engine version 1.0.4.1.R2.1 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Disabled an optimization for Gremlin conditional insert patterns which can add or append to existing labels and properties.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.1.R2.1, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3 . 4 . 8

- *SPARQL version: 1.1*

Upgrade Paths to Engine Release 1.0.4.1.R2.1

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.4.1.R2.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.4.1.R2.1 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.4.1.R2 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.4.1.R2 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.4.0 (2020-10-12)

As of 2020-10-12, engine version 1.0.4.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.4.0.R2 \(2021-02-24\) \(p. 741\)](#)

New Features in This Engine Release

- Added frame-level compression for Gremlin.

Improvements in This Engine Release

- Amazon Neptune now requires the use of the Secure Sockets Layer (SSL) with the TLSv1.2 protocol for all connections to Neptune in all regions, using these strong cipher suites:
 - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
 - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
 - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
 - TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
 - TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

This is true for both REST and WebSocket connections to Neptune, and means that you must use HTTPS rather than HTTP when connecting to Neptune in all regions.

Because client connections using HTTP or TLS 1.1 will no longer be supported anywhere, please make sure that your clients and code have been updated to use TLS 1.2 and HTTPS before upgrading to this engine release.

Important

Having to use SSL/TLS for all connections to Neptune can be a breaking change. It affects your connections with the Gremlin console, the Gremlin driver, Gremlin Python, .NET, nodeJs, REST APIs, and also load-balancer connections. If you have been using HTTP for any or all of these, you must now update the relevant client and drivers and change your code to use HTTPS or your connections will fail.

A bug in this release has allowed HTTP connections and/or outdated TLS connections to continue to work for customers who previously set a DB cluster parameter to prevent enforcement of HTTPS connections. That bug was fixed in patch releases [1.0.4.0.R2 \(p. 741\)](#) and [1.0.4.1.R2 \(p. 737\)](#), but the fix has caused unexpected connection failures when the patches are automatically installed.

For this reason, both patches have been reverted, and can only be installed manually, to give you a chance to update your setup for TLS 1.2.

- Upgraded TinkerPop to version 3.4.8. This is a backwards compatible upgrade. See the [TinkerPop change log](#) for what's new.
- Improved performance for the `Gremlin properties()` step.
- Added details about `BindOp` and `MultiplexerOp` in explain and profile reports.
- Added data prefetch to improve performance when there are cache misses.
- Added a new `allowEmptyStrings` setting in the bulk loader's `parserConfiguration` parameter that allows empty strings to be treated as valid property values in CSV loads (see [Neptune Loader Request Parameters \(p. 210\)](#)).
- The loader now allows an escaped semicolon in multivalued CSV columns.

Defects Fixed in This Engine Release

- Fixed a potential Gremlin memory leak related to the `both()` step.
- Fixed a bug where request metrics were missing because an endpoint ending in '/' was not being handled correctly.
- Fix a bug that caused replicas to fall behind and restart under heavy load when the DFE engine is enabled in lab mode.
- Fixed a bug that prevented the correct error message from being reported when a bulk load failed because of an out-of-memory condition.

- Fixed a SPARQL bug where the character encoding was placed in the Content-Encoding header in SPARQL query responses. Now `charset` is placed in the Content-Type header instead, enabling HTTP clients to recognize the character set being used automatically.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.0, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.8
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.4.0

You can manually upgrade any previous Neptune engine release to this release.

You will not automatically upgrade to this release.

Upgrading to This Release

Amazon Neptune 1.0.4.0 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.4.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.4.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.4.0.R2 (2021-02-24)

As of 2021-02-24, engine version 1.0.4.0.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Fixed a bug in [Release: 1.0.4.0 \(2020-10-12\) \(p. 739\)](#) that allowed connections to Neptune using HTTP or earlier versions of TLS, rather than HTTPS and TLS 1.2.

Important

Having to use SSL/TLS for all connections to Neptune can be a breaking change. It affects your connections with the Gremlin console, the Gremlin driver, Gremlin Python, .NET, nodeJs, REST APIs, and also load-balancer connections. If you have been using HTTP or an older TLS version for any or all of these up until now, you must update the relevant client and drivers before installing this patch, and change your code to use HTTPS exclusively.

- Fixed a bug in the CSV bulk load involving labels that end in #.
- Fixed a Gremlin bug where `InternalFailureException` was set as the response code in certain circumstances when a `ConcurrentModificationException` occurred.
- Fixed a Gremlin bug where under certain conditions updating edges or vertices could cause a transient `InternalFailureException`.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.4.0.R2, make sure that your project is compatible with these query-language versions:

- Gremlin version:* 3.4.8
- SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.4.0.R2

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.4.0.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.4.0.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.4.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.4.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.3.0 (2020-08-03)

As of 2020-08-03, engine version 1.0.3.0 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.3.0.R2 \(2020-10-12\) \(p. 746\)](#)
- [Release: 1.0.3.0.R3 \(2021-02-19\) \(p. 745\)](#)

New Features in This Engine Release

- Neptune has introduced a new, alternative query engine (DFE) which can significantly speed up query execution. See [The Amazon Neptune alternative query engine \(DFE\) \(p. 87\)](#).
- The DFE relies on pre-generated statistics about your Neptune graph data that are managed through new statistics endpoints. See [DFE statistics \(p. 88\)](#).
- You can now exclude queued load jobs from the list of load IDs returned by the Loader Get-Status API by setting the new `includeQueuedLoads` parameter to `FALSE`. See [Neptune Loader Get-Status request parameters \(p. 217\)](#).
- Neptune now supports trailing headers for SPARQL query responses that can contain an error code and message if a request fails after it begins to return response chunks. See [Optional HTTP trailing headers for multi-part SPARQL responses \(p. 349\)](#).
- Neptune now also lets you enable chunked response encoding for Gremlin queries. As in the SPARQL case, the response chunks have trailing headers that can contain an error code and message if a failure occurs after the query has begun to return response chunks. See [Use optional HTTP trailing headers to enable multi-part Gremlin responses \(p. 266\)](#).

Improvements in This Engine Release

- You can now provide the size of batch requests to Elasticsearch for full- text searches in Gremlin.
- Improved memory usage for SPARQL GROUP BY queries.
- Added a new Gremlin query optimizer to prune certain unbound filters.
- Increased the maximum time a WebSocket connection authenticated using IAM can stay open, from 36 hours to 10 days.

Defects Fixed in This Engine Release

- Fixed a bug where if you sent an un-encoded URL parameter in a POST request, Neptune returned an HTTP status code of 500 and an `InternalServerErrorException`. Now Neptune returns an HTTP

status code of 400 and a `BadRequestException`, with the message: `Failure to process the POST request parameters.`

- Fixed a Gremlin bug where a `WebSocket` connection failure was not correctly reported.
- Fixed a Gremlin bug involving disappearing `sideEffects`.
- Fixed a Gremlin bug where the full-text search `batchsize` parameter was not properly supported.
- Fixed a Gremlin bug to handle `toV` and `fromV` individually for each direction on `bothE`.
- Fixed a Gremlin bug involving `Edge pathType` in the `hasLabel` step.
- Fixed a SPARQL bug where join re-ordering with static bindings was not working correctly.
- Fixed a SPARQL UPDATE LOAD bug where an unavailable Amazon S3 bucket was not correctly reported.
- Fixed a SPARQL bug where an issue with a SERVICE node in a subquery was not correctly reported.
- Fixed a SPARQL bug in which queries containing nested FILTER EXISTS or FILTER NOT EXISTS conditions were not being properly evaluated.
- Fixed a SPARQL bug to correctly handle duplicate generated bindings when calling SPARQL Service endpoints through generate queries.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.3.0, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.3
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.3.0

You can manually upgrade any previous Neptune engine release to this release.

If your cluster has its `AutoMinorVersionUpgrade` parameter set to `True`, your cluster will be upgraded to this engine release automatically two to three weeks after the date of this release, during a maintenance window.

Upgrading to This Release

Amazon Neptune 1.0.3.0 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.3.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.3.0 ^
```

```
--apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.3.0.R3 (2021-02-19)

As of 2021-02-19, engine version 1.0.3.0.R3 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Fixed a bug in the CSV bulk load involving labels that end in #.
- Fixed a Gremlin bug that could cause missing results for a specific pattern of queries that use the `as()` step.
- Fixed a Gremlin bug that could cause errors when using the `project()` step nested inside another step such as `union()`.
- Fixed a Gremlin bug in string traversal execution in the experimental DFE engine when a terminal method like `toList()` is used.
- Fixed a Gremlin bug that fails to close a transaction when using the `iterate()` step in a string query.
- Fixed a Gremlin bug that could cause queries using the `is(P.gte(0))` pattern to throw an exception under certain conditions.
- Fixed a Gremlin bug where `InternalFailureException` was set as the response code in certain circumstances when a `ConcurrentModificationException` occurred.
- Fixed a Gremlin bug where under certain conditions updating edges or vertices could cause a transient `InternalFailureException`.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.3.0.R3, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.8
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.3.0.R3

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.3.0.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.3.0.R3 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.3.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.3.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.3.0.R2 (2020-10-12)

As of 2020-10-12, engine version 1.0.3.0.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Improvements in This Engine Release

- Improved performance for the Gremlin `properties()` step.
- Added details about `BindOp` and `MultiplexerOp` in explain and profile reports.
- For SPARQL query responses, added `charset` to the Content-Type header, enabling HTTP clients to recognize the charset being used automatically.

Defects Fixed in This Engine Release

- Fixed a SPARQL bug where `CancellationException` was not handled.
- Fixed a SPARQL bug where queries containing nested optionals did not work correctly.
- Fixed a SPARQL bug in LOAD where a `ConcurrentModificationException` could cause a query to hang.
- Fixed a SPARQL bug that prevented query responses from being gzip-compressed.
- Fixed a Gremlin bug in the `groupBy()` step.
- Fixed a Gremlin bug related to the use of an `aggregate()` step inside a `local()` step.
- Fixed a Gremlin bug related to using `bothE()` followed by a predicate that uses aggregate values.
- Fixed a Gremlin bug related to using the `bothE()` step with the `repeat()` step.

- Fixed a potential Gremlin memory leak related to the `both()` step.
- Fixed a bug where request metrics were missing because an endpoint ending in `'/'` was not being handled correctly.
- Fixed a bug that could raise a `ThrottlingException` even when the request queue is not full.
- Fixed a bug in fetching load status when a load fails for a reason such as `LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETE`.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.3.0.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.3
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.3.0.R2

You can manually upgrade any previous Neptune engine release to this release.

If your cluster has its `AutoMinorVersionUpgrade` parameter set to `True`, your cluster will be upgraded to this engine release automatically two to three weeks after the date of this release, during a maintenance window.

Upgrading to This Release

Amazon Neptune 1.0.3.0.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.3.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.3.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.2 (2020-03-09)

As of 2020-03-09, engine version 1.0.2.2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Subsequent Patch Releases for This Release

- [Release: 1.0.2.2.R2 \(2020-04-02\)](#) (p. 754)
- [Release: 1.0.2.2.R3 \(2020-07-22\)](#) (p. 753)
- [Release: 1.0.2.2.R4 \(2020-07-23\)](#) (p. 752)
- [Release: 1.0.2.2.R5 \(2020-10-12\)](#) (p. 751)
- [Release: 1.0.2.2.R6 \(2021-02-19\)](#) (p. 749)

Improvements in This Engine Release

- Added information to the status API about transactions that are being rolled back. See [Instance Status](#) (p. 639).
- Upgraded the version of Apache TinkerPop to 3.4.3.

Version 3.4.3 is backwards compatible with the previous version supported by Neptune (3.4.1). It does introduce one minor change in behavior: Gremlin no longer returns an error when you try to close a session that does not exist (see [Prevent error when closing sessions that don't exist](#)).

- Removed performance bottlenecks in execution of Gremlin full-text search steps.

Defects Fixed in This Engine Release

- Fixed a SPARQL bug in the handling of empty graph patterns in queries.
- Fixed a SPARQL bug in the handling of unencoded semicolons in URL-encoded queries.
- Fixed a Gremlin bug in the handling of repeated vertices in the `Union` step.
- Fixed a Gremlin bug that caused some queries with a `.simplePath()` or `.cyclicPath()` inside a `.repeat()` to return incorrect results.
- Fixed a Gremlin bug that caused `.project()` to return incorrect results if its child traversal returned no solutions.
- Fixed a Gremlin bug where errors from read-write conflicts raised an `InternalFailureException` rather than a `ConcurrentModificationException`.
- Fixed a Gremlin bug that caused `.group().by(...).by(values("property"))` failures.
- Fixed Gremlin bugs in the profile output for full-text-search steps.
- Fixed a resource leak in Gremlin sessions.
- Fixed a bug that prevented the status API from reporting the correct orderable version in some cases.
- Fixed a bulk loader bug that allowed a URL to a location other than Amazon S3 to be used as the source in a bulk load request.
- Fixed a bulk loader bug in the detailed load status.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.3
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.2

You can manually upgrade any previous Neptune engine release to this release.

If your cluster has its `AutoMinorVersionUpgrade` parameter set to `True`, your cluster will be upgraded to this engine release automatically two to three weeks after the date of this release, during a maintenance window.

Upgrading to This Release

Amazon Neptune 1.0.2.2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.2 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.2 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.2.R6 (2021-02-19)

As of 2021-02-19, engine version 1.0.2.2.R6 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug where `InternalFailureException` was set as the response code in certain circumstances when a `ConcurrentModificationException` occurred.
- Fixed a Gremlin bug where under certain conditions updating edges or vertices could cause a transient `InternalFailureException`.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.2.R6, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.8
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.2.R6

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.2.2.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.2.R6 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.2 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.2 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.2.R5 (2020-10-12)

As of 2020-10-12, engine version 1.0.2.2.R5 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Improvements in This Engine Release

- Improved performance for the Gremlin `properties()` step.
- Added details about `BindOp` and `MultiplexerOp` in explain and profile reports.
- For SPARQL query responses, added `charset` to the Content-Type header, enabling HTTP clients to recognize the charset being used automatically.

Defects Fixed in This Engine Release

- Fixed a SPARQL bug where `CancellationException` was not handled.
- Fixed a SPARQL bug where queries containing nested optionals did not work correctly.
- Fixed a SPARQL bug in `LOAD` where a `ConcurrentModificationException` could cause a query to hang.
- Fixed a SPARQL bug that prevented query responses from being gzip-compressed.
- Fixed a Gremlin bug in the `groupBy()` step.
- Fixed a Gremlin bug related to the use of an `aggregate()` step inside a `local()` step.
- Fixed a Gremlin bug related to using `bothE()` followed by a predicate that uses aggregate values.
- Fixed a Gremlin bug related to using the `bothE()` step with the `repeat()` step.
- Fixed a potential Gremlin memory leak related to the `both()` step.
- Fixed a bug where request metrics were missing because an endpoint ending in `/` was not being handled correctly.
- Fixed a bug that could raise a `ThrottlingException` even when the request queue is not full.
- Fixed a bug in fetching load status when a load fails for a reason such as `LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETE`.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.2.R5, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.3
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.2.R5

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.2.2.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.2.R5 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.2 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.2 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.2.R4 (2020-07-23)

As of 2020-07-23, engine version 1.0.2.2.R4 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Improvements in This Engine Release

- Improved memory usage by releasing unused memory back to the operating system more frequently.
- Also improved memory usage for SPARQL GROUP BY queries.
- Increased the maximum time a WebSocket connection can stay open that is authenticated using IAM, from 36 hours to 10 days.
- Added the `BufferCacheHitRatio` CloudWatch metric, which can be useful in diagnosing query latency and tuning instance types. See [Neptune Metrics \(p. 644\)](#).

Defects Fixed in This Engine Release

- Fixed a bug in closing idle or expired IAM WebSocket connections. Neptune now sends a close frame before closing the connection.
- Fixed a SPARQL bug in the evaluation of queries containing nested FILTER EXISTS and/or FILTER NOT EXISTS conditions.
- Fixed a SPARQL query termination bug that caused blocked threads on the server under certain extreme conditions.
- Fixed a Gremlin bug involving Edge pathType in the `hasLabel` step.
- Fixed a Gremlin bug to handle `toV` and `fromV` individually for each direction on `bothE`.

- Fixed a Gremlin bug involving disappearing sideEffects.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.2.R4, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.3
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.2.R4

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.2.2.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.2.R4 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.2 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.2 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.2.R3 (2020-07-22)

Engine release 1.0.2.2.R3 was incorporated into [engine release 1.0.2.2.R4 \(p. 752\)](#).

Amazon Neptune Engine Version 1.0.2.2.R2 (2020-04-02)

As of 2020-04-02, engine version 1.0.2.2.R2 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Improvements in This Engine Release

- You can now queue up to 64 bulk-load jobs, rather than having to wait for one to finish before initiating the next one. You can also make execution of a queued load request contingent on the successful completion of one or more previously queued load jobs using the `dependencies` parameter of the `load` command. See [Neptune Loader Command \(p. 209\)](#).
- Full-text-search output can now be sorted (see [Full-Text Search Parameters \(p. 480\)](#)).
- There is now a DB cluster parameter for invoking Neptune streams, and the feature has been moved out of Lab Mode. See [Enabling Neptune Streams \(p. 452\)](#).

Defects Fixed in This Engine Release

- Fixed a stochastic failure in server startup which delayed instance creation.
- Fixed an optimizer issue where `BIND` statements in the query made the optimizer start out with unselective patterns in join-order planning.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.2.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.3
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.2.R2

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.2.2.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.2.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.2 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.2 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.1 (2019-11-22)

Subsequent Patch Releases for This Release

- [Release: 1.0.2.1.R6 \(2020-04-22\) \(p. 757\)](#)
- [Release: 1.0.2.1.R5 \(2020-04-22\) \(p. 758\)](#) *This patch release was not deployed.*
- [Release: 1.0.2.1.R4 \(2019-12-20\) \(p. 758\)](#)
- [Release: 1.0.2.1.R3 \(2019-12-12\) \(p. 759\)](#)
- [Release: 1.0.2.1.R2 \(2019-11-25\) \(p. 760\)](#)

New Features in This Engine Release

- Added full-text search capabilities through integration with the Amazon OpenSearch Service. See [Neptune full text search \(p. 471\)](#)
- Added the option using lab mode to create a fourth index (an OSGP index) for large numbers of predicates. See [OSGP index \(p. 86\)](#).
- Added a *details* mode to SPARQL Explain. See [Using SPARQL explain \(p. 371\)](#) and [Details mode output \(p. 374\)](#) for details.
- Added lab mode information to the engine status report. See [Instance Status \(p. 639\)](#) for details.
- DB Cluster snapshots can now be copied across AWS Regions. See [Copying a Snapshot \(p. 681\)](#).

Improvements in This Engine Release

- Improved performance when handling a large number of predicates.
- Enhanced query optimization. While this should be entirely transparent to customers, we encourage you to test your applications before upgrading to ensure that they behave as expected.
- Minor enhancements to error reporting.
- Added optimizations for Gremlin `.project()` and `.identity()` steps.
- Added optimizations for non-terminal Gremlin `.union()` cases.
- Added native support for Gremlin `.path().by()` traversals.
- Added native support for Gremlin `.coalesce()`.

- Further optimization of bulk write.
- We now require that HTTPS connections use at least TLS version 1.2 or higher, to prevent outdated/insecure ciphers being used.

Defects Fixed in This Engine Release

- Fixed a Gremlin `addE()` inner traversal handling bug.
- Fixed a Gremlin bug caused by AST annotations leaking from child traversals to the parent.
- Fixed a bug that occurred in Gremlin when `.otherV()` was called after `select()`.
- Fixed a Gremlin bug that caused some `.hasLabel()` steps to fail if they appeared after a `bothE()` step.
- Made minor fixes for Gremlin `.sum()` and `.project()`.
- Fixed a bug in processing SPARQL queries that lack a closing brace.
- Fixed some minor bugs in SPARQL Explain.
- Fixed a bug in the handling of concurrent get load status requests.
- Reduced memory used for executing some Gremlin traversals with `.project()` steps.
- Fixed numeric comparisons of special values in SPARQL. See [Standards Compliance \(p. 47\)](#).

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.1, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.1

You can manually upgrade any previous Neptune engine release to this release.

You will not automatically upgrade to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.1 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^
```

```
--engine-version 1.0.2.1 ^  
--apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.1.R6 (2020-04-22)

As of 2020-04-22, engine version 1.0.2.1.R6 is being generally deployed. Please note that it takes several days for a new release to become available in every region.

Defects Fixed in This Engine Release

- Fixed a bug where `ConcurrentModificationConflictException` and `TransactionException` were not converted into a `NeptuneGremlinException`, causing `InternalFailureException` to be returned to customers.
- Fixed a bug where Neptune reported its status as healthy before the server was completely ready.
- Fixed a bug where dictionary and user transaction commits were out of order when two `value->id` mappings were being inserted concurrently.
- Fixed a bug in load-status serialization.
- Fixed a Gremlin sessions bug.
- Fixed a bug where Neptune failed to throw an exception when the server failed to start.
- Fixed a bug where Neptune failed to send a Web socket close frame before closing the channel.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.1.R6, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.1.R6

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.2.1.

You can manually upgrade any previous Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.1.R6 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.1 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.1.R5 (2020-04-22)

Engine release 1.0.2.1.R5 was never deployed.

Amazon Neptune Engine Version 1.0.2.1.R4 (2019-12-20)

Improvements in This Engine Release

- Neptune now tries always to place any full-text-search call first in the execution pipeline. This reduces the volume of calls to OpenSearch, which can significantly improve performance. See [Full-Text-Search Query Execution \(p. 494\)](#).
- Neptune now raises an `IllegalArgumentException` if you try to access a non-existent property, vertex, or edge. Previously, Neptune raised an `UnsupportedOperationException` in that situation.

For example, if you try to add an edge referencing a nonexistent vertex, you will now raise an `IllegalArgumentException`.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug where a `union` traversal inside a `project-by` does not return results or returns incorrect results.
- Fixed a Gremlin bug that caused nested `.project().by()` steps to return incorrect results.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.1.R4, make sure that your project is compatible with these query-language versions:

- *Gremlin version: 3.4.1*
- *SPARQL version: 1.1*

Upgrade Paths to Engine Release 1.0.2.1.R4

You can manually upgrade any previous Neptune engine release to this release.

However, **automatic updating to this release is not supported**.

Upgrading to This Release

Amazon Neptune 1.0.2.1.R4 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.1 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.1.R3 (2019-12-12)

Defects Fixed in This Engine Release

- Fixed a bug where the OSGP index was disabled even though the feature was correctly enabled using [inLab Mode \(p. 85\)](#) using the `ObjectIndex` value in the `neptune_lab_mode` parameter.
- Fixed a bug that affected Gremlin queries with a `.fold()` inside a `.project().by()` step. For example, it caused the following query to return incomplete results:

```
g.V().project("a").by(valueMap().fold())
```

- Fixed a performance bottleneck in bulk loads of RDF data.
- Fixed a bug that caused a crash on replicas when streams were enabled and the replica was restarted before the primary.

- Fixed a bug where rotated SSL certificates on instances were not picked up without an instance restart.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.1.R3, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.1.R3

You can manually upgrade any previous Neptune engine release to this release.

However, **automatic updating to this release is not supported**.

Upgrading to This Release

Amazon Neptune 1.0.2.1.R3 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.1 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.1.R2 (2019-11-25)

Defects Fixed in This Engine Release

- Fixed a bug affecting all `project().by()` queries with non round-robin by-traversals and non `path()` by-traversals.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.1.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.1.R2

You can manually upgrade any previous Neptune engine release to this release.

However, **automatic updating to this release is not supported.**

Upgrading to This Release

Amazon Neptune 1.0.2.1.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.1 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.1 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.0 (2019-11-08)

IMPORTANT: THIS VERSION IS NOW DEPRECATED

This engine version is now superseded by [version 1.0.2.1 \(p. 755\)](#), which contains all the bug fixes in this version as well as additional features such as full-text search integration, OSGP index support, and database snapshot cluster copy across AWS Regions.

Starting June 1, 2020, Neptune will automatically upgrade any cluster running this engine version to [the latest patch of version 1.0.2.1 \(p. 757\)](#) during the next maintenance window. You can upgrade manually before then, as described [here \(p. 755\)](#).

If you have any issues with the upgrade, please contact us through [AWS Support](#) or the [AWS Developer Forums](#).

Subsequent Patch Releases for This Release

- [Release: 1.0.2.0.R3 \(2020-05-05\) \(p. 763\)](#)
- [Release: 1.0.2.0.R2 \(2019-11-21\) \(p. 764\)](#)

New Features in This Engine Release

In addition to maintenance updates, this release adds new functionality to support more than one engine version at a time (see [Neptune Engine Updates \(p. 99\)](#)).

As a result, the numbering of engine releases has changed (see [Neptune Engine Version Numbering \(p. 100\)](#)).

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.0, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.0

You can manually upgrade any previous Neptune engine release to this release.

You will not automatically upgrade to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.0 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.0 ^
```

```
--apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.0.R3 (2020-05-05)

IMPORTANT: THIS VERSION IS NOW DEPRECATED

This engine version is now superseded by [version 1.0.2.1 \(p. 755\)](#), which contains all the bug fixes in this version as well as additional features such as full-text search integration, OSGP index support, and database snapshot cluster copy across AWS Regions.

Starting June 1, 2020, Neptune will automatically upgrade any cluster running this engine version to [the latest patch of version 1.0.2.1 \(p. 757\)](#) during the next maintenance window. You can upgrade manually before then, as described [here \(p. 755\)](#).

If you have any issues with the upgrade, please contact us through [AWS Support](#) or the [AWS Developer Forums](#).

Defects Fixed in This Engine Release

- Fixed a bug where `ConcurrentModificationConflictException` and `TransactionException` were reported as generic `InternalFailureExceptions`.
- Fixed bugs in health checks that caused frequent restarts of the server during start up.
- Fixed a bug where data was not visible on replicas because commits were out of order under certain conditions.
- Fixed a bug in load-status serialization where a load failed from a lack of Amazon S3 access permissions.
- Fixed a resource leak in Gremlin sessions.
- Fixed a bug in health check that hid the unhealthy status on start-up of components managing IAM authentication.
- Fixed a bug where Neptune failed to send a WebSocket close frame before closing the channel.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.0.R3, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.0.R3

Your cluster will be upgraded to this patch release automatically during your next maintenance window if you are running engine version 1.0.2.0.

You can manually upgrade any earlier Neptune engine release to this release.

Upgrading to This Release

Amazon Neptune 1.0.2.0.R3 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version 1.0.2.0 ^  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.2.0.R2 (2019-11-21)

IMPORTANT: THIS VERSION IS NOW DEPRECATED

This engine version is now superseded by [version 1.0.2.1 \(p. 755\)](#), which contains all the bug fixes in this version as well as additional features such as full-text search integration, OSGP index support, and database snapshot cluster copy across AWS Regions.

Starting June 1, 2020, Neptune will automatically upgrade any cluster running this engine version to [the latest patch of version 1.0.2.1 \(p. 757\)](#) during the next maintenance window. You can upgrade manually before then, as described [here \(p. 755\)](#).

If you have any issues with the upgrade, please contact us through [AWS Support](#) or the [AWS Developer Forums](#).

Defects Fixed in This Engine Release

- Improved the caching strategy for dirty pages on the server so that `FreeableMemory` recovers faster when the server enters a low-memory state.
- Fixed a bug that could cause a race condition and crash when many concurrent load status and/or start load requests are processed on the server.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.2.0.R2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Upgrade Paths to Engine Release 1.0.2.0.R2

You can manually upgrade any previous Neptune engine release to this release.

However, **automatic updating to this release is not supported.**

Upgrading to This Release

Amazon Neptune 1.0.2.0.R2 is now generally available.

If a DB cluster is running an engine version from which there is an upgrade path to this release, it is eligible to be upgraded now. You can upgrade any eligible cluster using the DB cluster operations on the console or by using the SDK. The following CLI command will upgrade an eligible cluster immediately:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version 1.0.2.0 \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^ \  
  --db-cluster-identifier (your-neptune-cluster) ^ \  
  --engine-version 1.0.2.0 ^ \  
  --apply-immediately
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using the DB cluster.

For more information about upgrading your engine version, see [Neptune Engine Updates \(p. 99\)](#). If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Amazon Neptune Engine Version 1.0.1.2 (2020-06-10)

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

Improvements in This Engine Release

- Neptune now raises an `IllegalArgumentException` if you try to access a non-existent property, vertex, or edge. Previously, Neptune raised an `UnsupportedOperationException` in that situation.

For example, if you try to add an edge referencing a nonexistent vertex, you will now raise an `IllegalArgumentException`.

Defects Fixed in This Engine Release

- Fixed a bug where dictionary and user transaction commits were out of order when two `value->id` mappings were being inserted concurrently.
- Fixed a bug in load-status serialization.
- Fixed a stochastic failure in server startup which delayed instance creation.
- Fixed a cursor leak.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.1.2, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.4.1
- *SPARQL version:* 1.1

Amazon Neptune Engine Version 1.0.1.1 (2020-06-26)

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

Defects Fixed in This Engine Release

- Fixed a bug where commits were out of order when inserted concurrently.
- Fixed a bug in load-status serialization.
- Fixed a stochastic failure in server startup which delayed instance creation.
- Fixed a memory leak.

Query-Language Versions Supported in This Release

Before upgrading a DB cluster to version 1.0.1.1, make sure that your project is compatible with these query-language versions:

- *Gremlin version:* 3.3.2

- *SPARQL version: 1.1*

Amazon Neptune Engine Version 1.0.1.0 (2019-07-02)

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

Amazon Neptune Engine Updates 2019-10-31

Version: 1.0.1.0.200502.0

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug in the serialization of the `tree()` step's response when clients connect to Neptune using `traversal().withRemote(...)` (in other words, using GLV bytecode).

This release addresses an issue in which clients who connected to Neptune using `traversal().withRemote(...)` received an invalid response to Gremlin queries that contained a `tree()` step.

- Fixed a SPARQL bug in `DELETE WHERE LIMIT` queries, in which the query termination process would hang because of a race condition, causing the query to time out.

Amazon Neptune Engine Updates 2019-10-15

Version: 1.0.1.0.200463.0

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

New Features in This Engine Release

- Added a Gremlin Explain/Profile feature (see [Analyzing Neptune query execution using Gremlin explain](#) (p. 292)).
- Added [Support for Gremlin sessions](#) (p. 286) to enable executing multiple Gremlin traversals in a single transaction.
- Added support for the SPARQL Federated Query extension in Neptune (see [SPARQL 1.1 Federated Query](#) and [SPARQL federated queries in Neptune using the SERVICE extension](#) (p. 390)).
- Added a feature letting you inject your own `queryId` into a Gremlin or SPARQL query, either through an HTTP URL parameter or through a SPARQL `queryId` query hint (see [Inject a Custom ID Into a Neptune Gremlin or SPARQL Query](#) (p. 83)).

- Added a [Lab Mode \(p. 85\)](#) feature to Neptune that can allow you to try out upcoming features which are not yet ready to be used in production.
- Added an upcoming [Neptune Streams \(p. 451\)](#) feature that reliably logs every change made to your database into a stream that persists for a week. This feature is available only in Lab Mode.
- Updated the formal semantics for concurrent transactions (see [Transaction Semantics in Neptune \(p. 66\)](#)). This feature provides industry-standard guarantees around concurrency.

By default, these transaction semantics are enabled. In some scenarios, this feature may change current load behavior and reduce load performance. You can use the DB Cluster `neptune_lab_mode` parameter to revert to the previous semantics by including `ReadWriteConflictDetection=disabled` in the parameter value.

Improvements in This Engine Release

- Improved the [Instance Status \(p. 639\)](#) API by reporting what version of TinkerPop and what version of SPARQL the engine is using.
- Improved Gremlin subgraph operator performance.
- Improved the performance of Gremlin response serialization.
- Improved the performance in the Gremlin Union step.
- Improved the latency of simple SPARQL queries.

Defects Fixed in This Engine Release

- Fixed a Gremlin bug where timeout was incorrectly being returned as an internal failure.
- Fixed a SPARQL bug in which ORDER BY over a partial set of variables caused an Internal Server Error.

Amazon Neptune Engine Updates 2019-09-19

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

Version: 1.0.1.0.200457.0

Amazon Neptune 1.0.1.0.200457.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200457.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Defects Fixed in This Engine Release

- Fixed a Gremlin correctness issue introduced in the previous engine release (1.0.1.0.200369.0) by removing the performance improvement to conjunctive predicate handling that caused it.
- Fixed a SPARQL bug that caused queries with `DISTINCT` and a single pattern wrapped into `OPTIONAL` to generate an `InternalServerError`.

Amazon Neptune Engine Updates 2019-08-13

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

Version: 1.0.1.0.200369.0

Amazon Neptune 1.0.1.0.200369.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200369.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

New Features in This Engine Release

- Added an `OVERSUBSCRIBE` option to the `parallelism` parameter of the [Neptune Loader Command](#) (p. 209), which causes the Neptune bulk loader to use all available threads and resources.

Improvements in This Engine Release

- Improved performance of SPARQL filters containing simple logical OR expressions.
- Improved Gremlin performance in handling conjunctive predicates.

Defects Fixed in This Engine Release

- Fixed a SPARQL bug preventing subtraction of an `xsd:duration` from an `xsd:date`.
- Fixed a SPARQL bug causing incomplete results from static inlining in the presence of a `UNION`.
- Fixed a SPARQL bug in query cancellation.
- Fixed a Gremlin bug causing overflow during type promotion.

- Fixed a Gremlin bug in the handling of vertex elements in `addE().from().to()` steps.
- Fixed a Gremlin bug (released 2019-07-26 in [Engine version 1.0.1.0.200366.0 \(p. 770\)](#)) involving the handling of NaN doubles and floats in single-cardinality inserts.
- Fixed a bug in generating query plans involving property based searches.

Amazon Neptune Engine Updates 2019-07-26

Version: 1.0.1.0.200366.0

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

New Features in This Engine Release

- Upgraded to TinkerPop 3.4.1 (see [TinkerPop Upgrade Information](#), and [TinkerPop 3.4.1 Change Log](#)).

For Neptune customers, these changes provide new functionality and improvements, such as:

- `GraphBinary` is now available as a serialization format.
- A keep-alive bug that caused memory leaks in the TinkerPop Java driver has been fixed, so a work-around is no longer necessary.

However, in a few cases, they may affect existing Gremlin code in Neptune. For example:

- `valueMap()` now returns a `Map<Object, Object>` instead of a `Map<String, Object>`.
- Inconsistent behavior of the `within()` step was fixed so it would work consistently with other steps. Previously, types had to match for comparisons to work. Now, numbers of different types can be accurately compared. For example, `33` now compares as equal to `33L`, which it did not before.
- A bug in `ReducingBarrierStep` was fixed, so it now returns no value if no elements are available for output.
- The order of `select()` scopes changed (the order is now `maps`, `side-effects`, `paths`). This changes the results of the rare queries that combine `side-effects` and `select` with the same key name for `side-effects` as for `select`.
- `bulkSet()` is now part of the GraphSON protocol. Queries that end with `toBulkSet()` won't work with older clients.
- One parameterization of the `Submit()` step was removed from the 3.4 client.

Many other changes introduced in TinkerPop 3.4 do not affect current Neptune behavior. For example, Gremlin `io()` was added as a step to `Traversal` and is now deprecated in `Graph`, but was never enabled in Neptune.

- Added support for single cardinality vertex properties to the [bulk loader for Gremlin \(p. 196\)](#), for loading property graph data.
- Added an option to overwrite the existing values for a single-cardinality property in the bulk loader.
- Added the ability to [retrieve the status of a Gremlin query \(p. 284\)](#), and to [cancel a Gremlin query \(p. 286\)](#).
- Added a [query hint for SPARQL query timeouts \(p. 363\)](#).
- Added the ability to see the instance role in the status API (see [Instance Status \(p. 639\)](#)).
- Added support for database cloning (see [Database Cloning in Neptune \(p. 437\)](#)).

Improvements in This Engine Release

- Improved the SPARQL Query Explanation to show graph variables from FROM clauses.

- Improved performance for SPARQL in filters, equal filters, VALUES clauses, and range counts.
- Improved performance for Gremlin step ordering.
- Improved performance for Gremlin `.repeat.dedup` traversals.
- Improved the performance of Gremlin `valueMap()` and `path().by()` traversals.

Defects Fixed in This Engine Release

- Fixed multiple issues with SPARQL property paths including operation with named graphs.
- Fixed an issue with SPARQL CONSTRUCT queries causing memory issues.
- Fixed an issue with the RDF Turtle parser and local names.
- Fixed an issue to correct error messages displayed to users.
- Fixed an issue with Gremlin `repeat().drop()` traversals.
- Fixed an issue with the Gremlin `drop()` step.
- Fixed an issue with Gremlin label filters.
- Fixed an issue with Gremlin query timeouts.

Amazon Neptune Engine Updates 2019-07-02

IMPORTANT: THIS ENGINE VERSION IS NOW DEPRECATED

No new instances using this engine version will be created, beginning 2021-04-27.

Defects Fixed in This Engine Release

- Fixed a bug that caused certain patterns with a property name and value bound not to be optimized.

Earlier Neptune Engine Releases

Topics

- [Amazon Neptune Engine Updates 2019-06-12 \(p. 771\)](#)
- [Amazon Neptune Engine Updates 2019-05-01 \(p. 772\)](#)
- [Amazon Neptune Engine Updates 2019-01-21 \(p. 773\)](#)
- [Amazon Neptune Engine Updates 2018-11-19 \(p. 774\)](#)
- [Amazon Neptune Engine Updates 2018-11-08 \(p. 775\)](#)
- [Amazon Neptune Engine Updates 2018-10-29 \(p. 775\)](#)
- [Amazon Neptune Engine Updates 2018-09-06 \(p. 776\)](#)
- [Amazon Neptune Engine Updates 2018-07-24 \(p. 777\)](#)
- [Amazon Neptune Engine Updates 2018-06-22 \(p. 778\)](#)

Amazon Neptune Engine Updates 2019-06-12

Version: 1.0.1.0.200310.0

Amazon Neptune 1.0.1.0.200310.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200310.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200310.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Fixes a bug where concurrent insertion and dropping of an edge can result in multiple edges with the same id.
- Other minor fixes and improvements.

Amazon Neptune Engine Updates 2019-05-01

Version: 1.0.1.0.200296.0

Amazon Neptune 1.0.1.0.200296.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200296.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200296.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Added the new `explain` feature to Neptune SPARQL queries to help you visualize the query plan and take steps to optimize it if necessary. For information, see [SPARQL explain \(p. 369\)](#).
- Improved SPARQL performance and reporting in various ways.
- Improved Gremlin performance and behavior in various ways.
- Improved the timing-out of long-running `drop()` queries.
- Improved the performance of `otherV()` queries.
- Added two fields to the information returned when you query the Neptune health status of a DB cluster or instance, namely the engine version number and the cluster or instance start time. See [Instance Status \(p. 639\)](#).
- The Neptune loader `Get-Status` API now returns a `startTime` field that records when a load job started.
- The loader command now takes an optional `parallelism` parameter that lets you restrict the number of threads the loader uses.

Amazon Neptune Engine Updates 2019-01-21

Version: 1.0.1.0.200267.0

Amazon Neptune 1.0.1.0.200267.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200267.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200267.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Neptune waits longer (within the query timeout specified) for any conflicts to get resolved. This reduces the number of concurrent modification exceptions that need to be handled by the client (see [Query Errors \(p. 711\)](#)).

- Fixed an issue where Gremlin cardinality enforcement sometimes caused the engine to restart.
- Improved Gremlin performance for `emit.times` repeat queries.
- Fixed a Gremlin issue where `repeat.until` was allowing `.emit` solutions through that should have been filtered.
- Improved error handling in Gremlin.

Amazon Neptune Engine Updates 2018-11-19

Version: 1.0.1.0.200264.0

Amazon Neptune 1.0.1.0.200264.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200264.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200264.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Added support for [the section called “Query hints” \(p. 278\)](#).
- Improved error messages for IAM authentication. For more information, see [the section called “IAM Errors” \(p. 713\)](#).
- Improved SPARQL query performance with a high number of predicates.
- Improved SPARQL property path performance.
- Improved Gremlin performance for conditional mutations, such as the `fold().coalesce(unfold(), ...)` pattern, when used with `addV()`, `addE()`, and `property()` steps.
- Improved Gremlin performance for `by()` and `sack()` modulations.
- Improved Gremlin performance for `group()` and `groupCount()` steps.
- Improved Gremlin performance for `store()`, `sideEffect()`, and `cap().unfold()` steps.
- Improved support for Gremlin single cardinality properties constraints.

- Improved enforcement of single cardinality for edge properties and vertex properties marked as single cardinality properties.
- Introduced an error if additional property values are specified for an existing edge property during Neptune Load jobs.

Amazon Neptune Engine Updates 2018-11-08

Version: 1.0.1.0.200258.0

Amazon Neptune 1.0.1.0.200258.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200258.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200258.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Added support for [SPARQL query hints](#) (p. 357).
- Improved performance for SPARQL FILTER (NOT) Exists queries.
- Improved performance for SPARQL DESCRIBE queries.
- Improved performance for the repeat until pattern in Gremlin.
- Improved performance for adding edges in Gremlin.
- Fixed an issue where SPARQL Update DELETE queries could fail in some cases.
- Fixed an issue for handling timeouts with the Gremlin WebSocket server.

Amazon Neptune Engine Updates 2018-10-29

Version: 1.0.1.0.200255.0

Amazon Neptune 1.0.1.0.200255.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200255.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200255.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Added IAM authentication information to Audit logs.
- Added Support for temporary credentials using IAM Roles and Instance Profiles.
- Added WebSocket connection termination for IAM authentication when permission is revoked or if the IAM user or role is deleted.
- Limited the maximum number of WebSocket connections to 60,000 per instance.
- Improved Bulk Load performance for smaller instance types.
- Improved performance for queries that include the `and()`, `or()`, `not()`, `drop()` operators in Gremlin.
- The NTriples parser now rejects invalid URIs, such as URIs containing whitespace.

Amazon Neptune Engine Updates 2018-09-06

Version: 1.0.1.0.200237.0

Amazon Neptune 1.0.1.0.200237.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200237.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200237.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Fixed an issue where some SPARQL `COUNT(DISTINCT)` queries failed.
- Fixed an issue where `COUNT`, `SUM`, `MIN` queries with a `DISTINCT` clause would run out of memory.
- Fixed an issue where `BLOB` type data would cause a Neptune Loader job to fail.
- Fixed an issue where duplicate inserts would cause transaction failures.
- Fixed an issue where `DROP ALL` queries could not be cancelled.
- Fixed an issue where Gremlin clients could hang intermittently.
- Updated all error codes for payloads bigger than 150M to be `HTTP 400`.
- Improved performance and accuracy of single-triple-pattern `COUNT()` queries.
- Improved performance of SPARQL `UNION` queries with `BIND` clauses.

Amazon Neptune Engine Updates 2018-07-24

Version: 1.0.1.0.200236.0

Amazon Neptune 1.0.1.0.200236.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200236.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200236.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Updated the SPARQL serialization for the `xsd:string` datatype. `xsd:string` is no longer included in JSON serialization, which is now consistent with other output formats.
- Fixed handling of `xsd:double/xsd:float` infinity. `-INF`, `NaN`, and `INF` values are now properly recognized and handled in all SPARQL data loader formats, SPARQL 1.1 UPDATE, and SPARQL 1.1 Query.
- Fixed an issue where a Gremlin query with empty string values fail unexpectedly.
- Fixed an issue where Gremlin `aggregate()` and `cap()` on an empty graph fails unexpectedly.
- Fixed an issue where incorrect error responses are returned for Gremlin when the cardinality specification is invalid, e.g. `.property(set, id, '10')` and `.property(single, id, '10')`.
- Fixed an issue where invalid Gremlin syntax was returned as an `InternalFailureException`.
- Fixed the spelling in `TimeLimitExceededException` to `TimeLimitExceededException`, in error messages.
- Changed the SPARQL and GREMLIN endpoints respond in a consistent way when no script is supplied.
- Clarified error messages for too many concurrent requests.

Amazon Neptune Engine Updates 2018-06-22

Version: 1.0.1.0.200233.0

Amazon Neptune 1.0.1.0.200233.0 is generally available. All new Neptune DB clusters, including those restored from snapshots, will be created in Neptune 1.0.1.0.200233.0 after the engine update is complete for that Region.

Existing clusters can be upgraded to this release immediately using the DB cluster operations on the console or by using the SDK. You can use the following CLI command to upgrade a DB cluster to this release immediately:

```
aws neptune apply-pending-maintenance-action \  
  --apply-action system-update \  
  --opt-in-type immediate \  
  --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

Neptune DB clusters will automatically be upgraded to engine release 1.0.1.0.200233.0 during system maintenance windows. The timing of when updates are applied depends on the Region and maintenance window setting for the DB cluster, as well as on the type of update.

Note

The instance maintenance window does not apply to engine updates.

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on all instances in a DB cluster, so you will experience downtime ranging from 20–30 seconds to several minutes, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings on the [Neptune console](#).

If you have any questions or concerns, the AWS Support team is available on the community forums and through [AWS Premium Support](#).

Improvements

- Fixed an issue where a large number of bulk load requests are issued in quick succession results in an error.

- Fixed a data-dependent issue where a query could fail with an `InternalServerError`. The following example shows the type of query affected.

```
g.V("my-id123").as("start").outE("knows").has("edgePropertyKey1",
P.gt(0)).as("myedge").inV()
    .as("end").select("start", "end", "myedge").by("vertexPropertyKey1")
    .by("vertexPropertyKey1").by("edgePropertyKey1")
```

- Fixed an issue where a Gremlin Java client cannot connect to the server using the same WebSocket connection after the timeout of a long-running query.
- Fixed an issue where the escaped sequences contained as part of the Gremlin query over HTTP or string-based queries over the WebSocket connection were not handled correctly.

Upgrading Your Neptune Engine

Note

Updates are applied to all instances in a DB cluster simultaneously. An update requires a database restart on those instances, so you experience downtime ranging from 20 or 30 seconds to several minutes, after which you can resume using the DB cluster.

Patch releases (`.R2`, `.R3`, etc.) are always installed automatically during your next maintenance window. This is independent of how the `AutoMinorVersionUpgrade` parameter is set.

If you want, you can choose to have minor engine updates also installed automatically to your entire cluster. You do this by setting the `AutoMinorVersionUpgrade` parameter of your primary writer instance to `true`. An automatic minor engine update will then be installed on every instance in your cluster during the maintenance window once the new engine version has proven stable, 2 to 3 weeks after the release.

- You can update your engine manually using the AWS Management Console as explained in [Using the Console to Upgrade Your Engine to a More Recent Version \(p. 781\)](#).
- You can also [use the CLI to update your engine manually \(p. 781\)](#).
- You can use the console to set `AutoMinorVersionUpgrade` to `true` in your primary writer instance. The result is that your cluster will always be automatically upgraded to the latest minor version during a maintenance window after a stabilization window of 2 to 3 weeks following the latest release.
- You can also [use the CLI to set `AutoMinorVersionUpgrade` \(p. 783\)](#) to `true` in your primary writer instance. Setting it to `true` in a reader instance has no effect.

If you are manually upgrading using the AWS CLI, be sure to specify the engine version to which you want to upgrade. If you do not, your engine may be upgraded to a version that is not the most recent one or the one you expect.

You can always determine what engine version your DB cluster is running by using the [Instance Status \(p. 639\)](#) API. If you are using the [CreateDBCluster \(p. 793\)](#) to create a cluster, the API returns the engine version used to create the cluster in its response.

The timing of automatic updates depends on the AWS region and maintenance window settings for your DB cluster. You can view or change your maintenance window settings on the [Neptune console](#). For more information, see [Neptune Maintenance Window \(p. 433\)](#).

Major engine releases can only be installed manually.

Note

Before performing a major upgrade, Neptune always creates a manual snapshot of your DB cluster, which lets you revert to the state of your cluster before the upgrade if you need to. This snapshot remains available until you delete it after you finish all post-upgrade testing.

Testing a new version of the Neptune engine before you upgrade

When a new major Neptune engine version is released, always test your Neptune applications on it first, before upgrading. If you do not have `AutoMinorVersionUpgrade` set, you may also want to do that for minor engine versions that introduce new features or behavior that could affect your code.

The best way to test a new version without upgrading your production DB cluster is to clone your cluster so that the clone is running the new engine version. You can then run queries on the clone without affecting your production DB cluster.

Neptune Engine Version Numbering

Neptune version numbers have 3 parts. Take version number `1.0.2.0.R2` as an example:

1. The first part (the first two numbers, namely the `1.0` in `1.0.2.0.R2`) is the database major version number.

This part only changes when a major incompatible change occurs, such as a change in the way data is stored that requires data migration when upgrading. Upgrading to a new major version often requires downtime proportional to the size of the cluster being upgraded, and can take much longer than other upgrades.

2. The second part (the third and fourth numbers, namely the `2.0` in `1.0.2.0.R2`) is the minor version number of the engine release.

You can choose to upgrade your engine to new minor versions automatically, as described below, or choose to decide for each minor version change.

3. The third part (the last number, namely the `R2` in `1.0.2.0.R2`) is the patch number for the minor version of the engine.

Patches involve urgent changes such as bug fixes or security changes that are deployed between minor engine releases. The first release of an engine version is implicitly `R0`, and patches to it are numbered `R1`, `R2`, and so forth.

Patches are always automatically applied during system maintenance windows, as described below.

You can list the available engine releases using the AWS CLI as follows:

For Linux, OS X, or Unix:

```
aws neptune describe-db-engine-versions \  
  --region <your-region> \  
  --engine neptune
```

For Windows:

```
aws neptune describe-db-engine-versions ^  
  --region <your-region> ^  
  --engine neptune
```

Available engine releases include only those releases that have a version number higher than the current one and for which an upgrade path is defined.

Using the Console to Upgrade Your Engine to a More Recent Version

You can perform a minor version upgrade of a Neptune DB cluster using the AWS Management Console as follows:

To upgrade the engine version of a DB cluster using the console

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the navigation pane, choose **Clusters**, and then choose the DB cluster that you want to modify.
3. Choose **Actions**, and then choose **Modify cluster**. The **Modify DB cluster** page appears.
4. For **DB engine version**, choose the new version.
5. Choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, choose **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify Cluster** to save your changes.

Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

Using the Console to Enable Automatic Engine Updates

If you set the `AutoMinorVersionUpgrade` field to `true` in the writer instance of a DB cluster, Neptune will automatically update the engine of the entire cluster to the latest minor version after a stabilization window of 2 to 3 weeks.

You can use the AWS Management Console to set this field:

To set the `AutoMinorVersionUpgrade` for a DB cluster

1. Sign in to the AWS Management Console, and open the Amazon Neptune console at <https://console.aws.amazon.com/neptune/home>.
2. In the list of DB instances, choose the primary instance (the writer instance) of the DB cluster.
3. Choose **Instance actions**, and then choose **Modify**. The **Modify DB Instance** page appears.
4. Choose **Actions**, and then choose **Modify instance**. The **Modify DB instance** page appears.
5. Set the auto minor version update field.

See [Modifying an Instance \(p. 444\)](#) for more information.

Manually performing a minor upgrade of the Neptune DB Engine using the CLI

You can manually perform a minor upgrade of the engine version of a DB cluster as follows, using the AWS CLI and the [ModifyDBCluster \(p. 804\)](#) API:

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine-version (new-engine-version) \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier (your-neptune-cluster) ^  
  --engine-version <new-engine-version> ^  
  --apply-immediately
```

Be sure to include the engine version that you want to upgrade to. If you do not, your engine may be upgraded to a version that is not the latest one.

Manually performing a major upgrade of the Neptune DB Engine using the CLI

To perform a major upgrade of the engine version of a DB cluster manually using the AWS CLI and the [ModifyDBCluster \(p. 804\)](#) API, you need to provide more parameters than for a minor upgrade:

It is important to include the engine version that you want to upgrade to. If you do not, your engine may be upgraded to a version that is not the latest one.

For Linux, OS X, or Unix:

```
aws neptune modify-db-cluster \  
  --db-cluster-identifier (your-neptune-cluster) \  
  --engine neptune \  
  --engine-version (new-engine-version) \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-cluster ^  
  --db-cluster-identifier <your-neptune-cluster> ^  
  --engine neptune ^  
  --engine-version <new-engine-version> ^  
  --apply-immediately
```

Instead of `--apply-immediately`, you can specify `--no-apply-immediately`.

If your cluster uses a custom cluster parameter group, be sure to include this parameter to specify it:

```
--db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

Similarly, if any instances in the cluster use a custom DB parameter group, be sure to include this parameter to specify it:

```
---db-instance-parameter-group-name (name of the custom instance parameter group)
```

Using the CLI to Enable Automatic Neptune Engine Updates

You can use the AWS CLI to enable or disable automatic engine updates for a DBcluster by setting the `AutoMinorVersionUpgrade` parameter of its primary instance.

To enable automatic engine upgrading for your DB cluster:

For Linux, OS X, or Unix:

```
aws neptune modify-db-instance \  
  --db-instance-identifier <your-cluster-write-instance> \  
  --auto-minor-version-upgrade \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-instance ^  
  --db-instance-identifier <your-cluster-write-instance> ^  
  --auto-minor-version-upgrade ^  
  --apply-immediately
```

Similarly, you can also disable automatic engine updating:

For Linux, OS X, or Unix:

```
aws neptune modify-db-instance \  
  --db-instance-identifier <your-cluster-write-instance> \  
  --no-auto-minor-version-upgrade \  
  --apply-immediately
```

For Windows:

```
aws neptune modify-db-instance ^  
  --db-instance-identifier <your-cluster-write-instance> ^  
  --no-auto-minor-version-upgrade ^  
  --apply-immediately
```


Amazon Neptune Cluster, Instance, and Resource Management API Reference

This chapter documents the Neptune APIs that you can use to manage and maintain your Neptune DB cluster.

Neptune operates on clusters of database servers that are connected in a replication topology. Thus, managing Neptune often involves deploying changes to multiple servers and making sure that all Neptune Replicas are keeping up with the primary server.

Because Neptune transparently scales the underlying storage as your data grows, managing Neptune requires relatively little management of disk storage. Likewise, because Neptune automatically performs continuous backups, a Neptune cluster does not require extensive planning or downtime for performing backups.

Contents

- [Neptune DB Clusters API \(p. 788\)](#)
 - [DBCluster \(structure\) \(p. 788\)](#)
 - [DBClusterMember \(structure\) \(p. 792\)](#)
 - [DBClusterRole \(structure\) \(p. 792\)](#)
 - [ClusterPendingModifiedValues \(structure\) \(p. 792\)](#)
 - [CreateDBCluster \(action\) \(p. 793\)](#)
 - [DeleteDBCluster \(action\) \(p. 799\)](#)
 - [ModifyDBCluster \(action\) \(p. 804\)](#)
 - [StartDBCluster \(action\) \(p. 810\)](#)
 - [StopDBCluster \(action\) \(p. 813\)](#)
 - [AddRoleToDBCluster \(action\) \(p. 817\)](#)
 - [RemoveRoleFromDBCluster \(action\) \(p. 818\)](#)
 - [FailoverDBCluster \(action\) \(p. 818\)](#)
 - [PromoteReadReplicaDBCluster \(action\) \(p. 822\)](#)
 - [DescribeDBClusters \(action\) \(p. 826\)](#)
- [Neptune Instances API \(p. 827\)](#)
 - [DBInstance \(structure\) \(p. 828\)](#)
 - [DBInstanceStatusInfo \(structure\) \(p. 831\)](#)
 - [OrderableDBInstanceOption \(structure\) \(p. 831\)](#)
 - [ValidDBInstanceModificationsMessage \(structure\) \(p. 833\)](#)
 - [CreateDBInstance \(action\) \(p. 833\)](#)
 - [DeleteDBInstance \(action\) \(p. 841\)](#)
 - [ModifyDBInstance \(action\) \(p. 846\)](#)
 - [RebootDBInstance \(action\) \(p. 855\)](#)
 - [DescribeDBInstances \(action\) \(p. 859\)](#)
 - [DescribeOrderableDBInstanceOptions \(action\) \(p. 860\)](#)
 - [DescribeValidDBInstanceModifications \(action\) \(p. 861\)](#)

- [Neptune Parameters API \(p. 862\)](#)
 - [Parameter \(structure\) \(p. 863\)](#)
 - [DBParameterGroup \(structure\) \(p. 863\)](#)
 - [DBClusterParameterGroup \(structure\) \(p. 864\)](#)
 - [DBParameterGroupStatus \(structure\) \(p. 864\)](#)
 - [CopyDBParameterGroup \(action\) \(p. 865\)](#)
 - [CopyDBClusterParameterGroup \(action\) \(p. 866\)](#)
 - [CreateDBParameterGroup \(action\) \(p. 867\)](#)
 - [CreateDBClusterParameterGroup \(action\) \(p. 869\)](#)
 - [DeleteDBParameterGroup \(action\) \(p. 870\)](#)
 - [DeleteDBClusterParameterGroup \(action\) \(p. 871\)](#)
 - [ModifyDBParameterGroup \(action\) \(p. 871\)](#)
 - [ModifyDBClusterParameterGroup \(action\) \(p. 872\)](#)
 - [ResetDBParameterGroup \(action\) \(p. 873\)](#)
 - [ResetDBClusterParameterGroup \(action\) \(p. 874\)](#)
 - [DescribeDBParameters \(action\) \(p. 875\)](#)
 - [DescribeDBParameterGroups \(action\) \(p. 876\)](#)
 - [DescribeDBClusterParameters \(action\) \(p. 877\)](#)
 - [DescribeDBClusterParameterGroups \(action\) \(p. 878\)](#)
 - [DescribeEngineDefaultParameters \(action\) \(p. 879\)](#)
 - [DescribeEngineDefaultClusterParameters \(action\) \(p. 880\)](#)
- [Neptune Subnet API \(p. 881\)](#)
 - [Subnet \(structure\) \(p. 881\)](#)
 - [DBSubnetGroup \(structure\) \(p. 882\)](#)
 - [CreateDBSubnetGroup \(action\) \(p. 882\)](#)
 - [DeleteDBSubnetGroup \(action\) \(p. 884\)](#)
 - [ModifyDBSubnetGroup \(action\) \(p. 884\)](#)
 - [DescribeDBSubnetGroups \(action\) \(p. 885\)](#)
- [Neptune Snapshots API \(p. 886\)](#)
 - [DBClusterSnapshot \(structure\) \(p. 887\)](#)
 - [DBClusterSnapshotAttribute \(structure\) \(p. 888\)](#)
 - [DBClusterSnapshotAttributesResult \(structure\) \(p. 889\)](#)
 - [CreateDBClusterSnapshot \(action\) \(p. 889\)](#)
 - [DeleteDBClusterSnapshot \(action\) \(p. 891\)](#)
 - [CopyDBClusterSnapshot \(action\) \(p. 893\)](#)
 - [ModifyDBClusterSnapshotAttribute \(action\) \(p. 896\)](#)
 - [RestoreDBClusterFromSnapshot \(action\) \(p. 898\)](#)
 - [RestoreDBClusterToPointInTime \(action\) \(p. 904\)](#)
 - [DescribeDBClusterSnapshots \(action\) \(p. 910\)](#)
 - [DescribeDBClusterSnapshotAttributes \(action\) \(p. 912\)](#)
- [Neptune Events API \(p. 912\)](#)
 - [Event \(structure\) \(p. 913\)](#)
 - [EventCategoriesMap \(structure\) \(p. 913\)](#)
 - [EventSubscription \(structure\) \(p. 914\)](#)
 - [CreateEventSubscription \(action\) \(p. 915\)](#)
 - [DeleteEventSubscription \(action\) \(p. 917\)](#)

- [ModifyEventSubscription](#) (action) (p. 918)
- [DescribeEventSubscriptions](#) (action) (p. 920)
- [AddSourceIdentifierToSubscription](#) (action) (p. 921)
- [RemoveSourceIdentifierFromSubscription](#) (action) (p. 922)
- [DescribeEvents](#) (action) (p. 923)
- [DescribeEventCategories](#) (action) (p. 925)
- [Other Neptune APIs](#) (p. 925)
 - [CloudwatchLogsExportConfiguration](#) (structure) (p. 926)
 - [DBEngineVersion](#) (structure) (p. 926)
 - [EngineDefaults](#) (structure) (p. 927)
 - [PendingMaintenanceAction](#) (structure) (p. 927)
 - [PendingModifiedValues](#) (structure) (p. 928)
 - [PendingCloudwatchLogsExports](#) (structure) (p. 929)
 - [ResourcePendingMaintenanceActions](#) (structure) (p. 929)
 - [UpgradeTarget](#) (structure) (p. 929)
 - [Tag](#) (structure) (p. 930)
 - [AddTagsToResource](#) (action) (p. 930)
 - [ListTagsForResource](#) (action) (p. 931)
 - [RemoveTagsFromResource](#) (action) (p. 931)
 - [ApplyPendingMaintenanceAction](#) (action) (p. 932)
 - [DescribePendingMaintenanceActions](#) (action) (p. 933)
 - [DescribeDBEngineVersions](#) (action) (p. 934)
- [Common Neptune Datatypes](#) (p. 935)
 - [AvailabilityZone](#) (structure) (p. 935)
 - [DBSecurityGroupMembership](#) (structure) (p. 935)
 - [DomainMembership](#) (structure) (p. 936)
 - [DoubleRange](#) (structure) (p. 936)
 - [Endpoint](#) (structure) (p. 936)
 - [Filter](#) (structure) (p. 937)
 - [Range](#) (structure) (p. 937)
 - [Timezone](#) (structure) (p. 937)
 - [VpcSecurityGroupMembership](#) (structure) (p. 937)
- [Neptune Exceptions Specific to Individual APIs](#) (p. 938)
 - [AuthorizationAlreadyExistsFault](#) (structure) (p. 939)
 - [AuthorizationNotFoundFault](#) (structure) (p. 940)
 - [AuthorizationQuotaExceededFault](#) (structure) (p. 940)
 - [CertificateNotFoundFault](#) (structure) (p. 940)
 - [DBClusterAlreadyExistsFault](#) (structure) (p. 940)
 - [DBClusterNotFoundFault](#) (structure) (p. 940)
 - [DBClusterParameterGroupNotFoundFault](#) (structure) (p. 941)
 - [DBClusterQuotaExceededFault](#) (structure) (p. 941)
 - [DBClusterRoleAlreadyExistsFault](#) (structure) (p. 941)
 - [DBClusterRoleNotFoundFault](#) (structure) (p. 941)
 - [DBClusterRoleQuotaExceededFault](#) (structure) (p. 942)
 - [DBClusterSnapshotAlreadyExistsFault](#) (structure) (p. 942)

- [DBClusterSnapshotNotFoundFault \(structure\) \(p. 942\)](#)
- [DBInstanceAlreadyExistsFault \(structure\) \(p. 942\)](#)
- [DBInstanceNotFoundFault \(structure\) \(p. 942\)](#)
- [DBLogFileNotFoundFault \(structure\) \(p. 943\)](#)
- [DBParameterGroupAlreadyExistsFault \(structure\) \(p. 943\)](#)
- [DBParameterGroupNotFoundFault \(structure\) \(p. 943\)](#)
- [DBParameterGroupQuotaExceededFault \(structure\) \(p. 943\)](#)
- [DBSecurityGroupAlreadyExistsFault \(structure\) \(p. 944\)](#)
- [DBSecurityGroupNotFoundFault \(structure\) \(p. 944\)](#)
- [DBSecurityGroupNotSupportedFault \(structure\) \(p. 944\)](#)
- [DBSecurityGroupQuotaExceededFault \(structure\) \(p. 944\)](#)
- [DBSnapshotAlreadyExistsFault \(structure\) \(p. 944\)](#)
- [DBSnapshotNotFoundFault \(structure\) \(p. 945\)](#)
- [DBSubnetGroupAlreadyExistsFault \(structure\) \(p. 945\)](#)
- [DBSubnetGroupDoesNotCoverEnoughAZs \(structure\) \(p. 945\)](#)
- [DBSubnetGroupNotAllowedFault \(structure\) \(p. 945\)](#)
- [DBSubnetGroupNotFoundFault \(structure\) \(p. 946\)](#)
- [DBSubnetGroupQuotaExceededFault \(structure\) \(p. 946\)](#)
- [DBSubnetQuotaExceededFault \(structure\) \(p. 946\)](#)
- [DBUpgradeDependencyFailureFault \(structure\) \(p. 946\)](#)
- [DomainNotFoundFault \(structure\) \(p. 946\)](#)
- [EventSubscriptionQuotaExceededFault \(structure\) \(p. 947\)](#)
- [InstanceQuotaExceededFault \(structure\) \(p. 947\)](#)
- [InsufficientDBClusterCapacityFault \(structure\) \(p. 947\)](#)
- [InsufficientDBInstanceCapacityFault \(structure\) \(p. 947\)](#)
- [InsufficientStorageClusterCapacityFault \(structure\) \(p. 948\)](#)
- [InvalidDBClusterSnapshotStateFault \(structure\) \(p. 948\)](#)
- [InvalidDBClusterStateFault \(structure\) \(p. 948\)](#)
- [InvalidDBInstanceStateFault \(structure\) \(p. 948\)](#)
- [InvalidDBParameterGroupStateFault \(structure\) \(p. 948\)](#)
- [InvalidDBSecurityGroupStateFault \(structure\) \(p. 949\)](#)
- [InvalidDBSnapshotStateFault \(structure\) \(p. 949\)](#)
- [InvalidDBSubnetGroupFault \(structure\) \(p. 949\)](#)
- [InvalidDBSubnetGroupStateFault \(structure\) \(p. 949\)](#)
- [InvalidDBSubnetStateFault \(structure\) \(p. 950\)](#)
- [InvalidEventSubscriptionStateFault \(structure\) \(p. 950\)](#)
- [InvalidOptionGroupStateFault \(structure\) \(p. 950\)](#)
- [InvalidRestoreFault \(structure\) \(p. 950\)](#)
- [InvalidSubnet \(structure\) \(p. 950\)](#)
- [InvalidVPCNetworkStateFault \(structure\) \(p. 951\)](#)
- [KMSKeyNotAccessibleFault \(structure\) \(p. 951\)](#)
- [OptionGroupNotFoundFault \(structure\) \(p. 951\)](#)
- [PointInTimeRestoreNotEnabledFault \(structure\) \(p. 951\)](#)
- [ProvisionedIopsNotAvailableInAZFault \(structure\) \(p. 952\)](#)
- [ResourceNotFoundFault \(structure\) \(p. 952\)](#)

- [SNSInvalidTopicFault \(structure\)](#) (p. 952)
- [SNSNoAuthorizationFault \(structure\)](#) (p. 952)
- [SNSTopicArnNotFoundFault \(structure\)](#) (p. 952)
- [SharedSnapshotQuotaExceededFault \(structure\)](#) (p. 953)
- [SnapshotQuotaExceededFault \(structure\)](#) (p. 953)
- [SourceNotFoundFault \(structure\)](#) (p. 953)
- [StorageQuotaExceededFault \(structure\)](#) (p. 953)
- [StorageTypeNotSupportedFault \(structure\)](#) (p. 954)
- [SubnetAlreadyInUse \(structure\)](#) (p. 954)
- [SubscriptionAlreadyExistFault \(structure\)](#) (p. 954)
- [SubscriptionCategoryNotFoundFault \(structure\)](#) (p. 954)
- [SubscriptionNotFoundFault \(structure\)](#) (p. 954)

Neptune DB Clusters API

Structures:

- [DBCluster \(structure\)](#) (p. 788)
- [DBClusterMember \(structure\)](#) (p. 792)
- [DBClusterRole \(structure\)](#) (p. 792)
- [ClusterPendingModifiedValues \(structure\)](#) (p. 792)

Actions:

- [CreateDBCluster \(action\)](#) (p. 793)
- [DeleteDBCluster \(action\)](#) (p. 799)
- [ModifyDBCluster \(action\)](#) (p. 804)
- [StartDBCluster \(action\)](#) (p. 810)
- [StopDBCluster \(action\)](#) (p. 813)
- [AddRoleToDBCluster \(action\)](#) (p. 817)
- [RemoveRoleFromDBCluster \(action\)](#) (p. 818)
- [FailoverDBCluster \(action\)](#) (p. 818)
- [PromoteReadReplicaDBCluster \(action\)](#) (p. 822)
- [DescribeDBClusters \(action\)](#) (p. 826)

DBCluster (structure)

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section](#) called “DescribeDBClusters” (p. 826) action.

Fields

- **AllocatedStorage** – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** – An array of [DBClusterRole](#) (p. 792) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** – Number (long).

Not supported by Neptune.

- **BacktrackWindow** – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** – Number (integer).

Not supported by Neptune.

- **CloneGroupId** – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** – Boolean.

If set to `true`, the DB cluster can be cloned across accounts.

- **DatabaseName** – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** – UTF-8 string.

Provides the name of the database engine to be used for this DB cluster.

- **EngineVersion** – UTF-8 string.

Indicates the database engine version.

- **HostedZoneId** – UTF-8 string.

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

- **IAMDatabaseAuthenticationEnabled** – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.

- **KmsKeyId** – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.

- **LatestRestorableTime** – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **MultiAZ** – Boolean.

Specifies whether the DB cluster has instances in multiple Availability Zones.

- **PendingModifiedValues** – A [ClusterPendingModifiedValues \(p. 792\)](#) object.

Not supported by Neptune.

- **PercentProgress** – UTF-8 string.

Specifies the progress of the operation as a percentage.

- **Port** – Number (integer).
Specifies the port that the database engine is listening on.
- **PreferredBackupWindow** – UTF-8 string.
Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.
- **PreferredMaintenanceWindow** – UTF-8 string.
Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).
- **ReaderEndpoint** – UTF-8 string.
The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.
- **ReadReplicaIdentifiers** – An array of `ReadReplicaIdentifier`— UTF-8 strings.
Contains one or more identifiers of the Read Replicas associated with this DB cluster.
- **ReplicationSourceIdentifier** – UTF-8 string.
Not supported by Neptune.
- **ReplicationType** – UTF-8 string.
Not supported by Neptune.
- **Status** – UTF-8 string.
Specifies the current state of this DB cluster.
- **StorageEncrypted** – Boolean.
Specifies whether the DB cluster is encrypted.
- **VpcSecurityGroups** – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.
Provides a list of VPC security groups that the DB cluster belongs to.

`DBCluster` is used as the response element for:

- [CreateDBCluster](#) (p. 793)
- [DeleteDBCluster](#) (p. 799)
- [FailoverDBCluster](#) (p. 818)
- [ModifyDBCluster](#) (p. 804)
- [PromoteReadReplicaDBCluster](#) (p. 822)
- [RestoreDBClusterFromSnapshot](#) (p. 898)
- [RestoreDBClusterToPointInTime](#) (p. 904)
- [StartDBCluster](#) (p. 810)
- [StopDBCluster](#) (p. 813)

DBClusterMember (structure)

Contains information about an instance that is part of a DB cluster.

Fields

- **DBClusterParameterGroupStatus** – UTF-8 string.
Specifies the status of the DB cluster parameter group for this member of the DB cluster.
- **DBInstanceIdentifier** – UTF-8 string.
Specifies the instance identifier for this member of the DB cluster.
- **IsClusterWriter** – Boolean.
Value that is `true` if the cluster member is the primary instance for the DB cluster and `false` otherwise.
- **PromotionTier** – Number (integer).
A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

DBClusterRole (structure)

Describes an Amazon Identity and Access Management (IAM) role that is associated with a DB cluster.

Fields

- **FeatureName** – UTF-8 string.
The name of the feature associated with the Amazon Identity and Access Management (IAM) role. For the list of supported feature names, see [DescribeDBEngineVersions](#).
- **RoleArn** – UTF-8 string.
The Amazon Resource Name (ARN) of the IAM role that is associated with the DB cluster.
- **Status** – UTF-8 string.
Describes the state of association between the IAM role and the DB cluster. The Status property returns one of the following values:
 - **ACTIVE** - the IAM role ARN is associated with the DB cluster and can be used to access other Amazon services on your behalf.
 - **PENDING** - the IAM role ARN is being associated with the DB cluster.
 - **INVALID** - the IAM role ARN is associated with the DB cluster, but the DB cluster is unable to assume the IAM role in order to access other Amazon services on your behalf.

ClusterPendingModifiedValues (structure)

Defines pending changes to a DB Cluster.

Fields

- **PendingCloudwatchLogsExports** – A [PendingCloudwatchLogsExports \(p. 929\)](#) object.
This `PendingCloudwatchLogsExports` structure specifies pending changes to which CloudWatch logs are enabled and which are disabled.

CreateDBCluster (action)

The Amazon CLI name for this API is: `create-db-cluster`, and the Python name is: `create_db_cluster`.

Creates a new Amazon Neptune DB cluster.

You can use the `ReplicationSourceIdentifier` parameter to create the DB cluster as a Read Replica of another DB cluster or Amazon Neptune DB instance.

Note that when you create a new cluster using `CreateDBCluster` directly, deletion protection is disabled by default (when you create a new production cluster in the console, deletion protection is enabled by default). You can only delete a DB cluster if its `DeletionProtection` field is set to `false`.

Request

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

A list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

The number of days for which automated backups are retained. You must specify a minimum value of 1.

Default: 1

Constraints:

- Must be a value from 1 to 35
- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

The name for your database of up to 64 alpha-numeric characters. If you do not provide a name, Amazon Neptune will not create a database in the DB cluster you are creating.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

The DB cluster identifier. This parameter is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- First character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster1`

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

The name of the DB cluster parameter group to associate with this DB cluster. If this argument is omitted, the default is used.

Constraints:

- If supplied, must match the name of an existing DBClusterParameterGroup.
- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

A DB subnet group to associate with this DB cluster.

Constraints: Must match the name of an existing DBSubnetGroup. Must not be default.

Example: `mySubnetgroup`

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

A value that indicates whether the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled. By default, deletion protection is enabled.

- **EnableCloudwatchLogsExports** (in the CLI: `--enable-cloudwatch-logs-exports`, or in Python: `enable_cloudwatch_logs_exports`) – An array of UTF-8 strings.

The list of log types that need to be enabled for exporting to CloudWatch Logs.

- **EnableIAMDatabaseAuthentication** (in the CLI: `--enable-iam-database-authentication`, or in Python: `enable_iam_database_authentication`) – Boolean.

If set to `true`, enables Amazon Identity and Access Management (IAM) authentication for the entire DB cluster (this cannot be set at an instance level).

Default: `false`.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – *Required*: UTF-8 string.

The name of the database engine to be used for this DB cluster.

Valid Values: `neptune`

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

The version number of the database engine to use for the new DB cluster.

Example: `1.0.2.1`

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

The Amazon KMS key identifier for an encrypted DB cluster.

The KMS key identifier is the Amazon Resource Name (ARN) for the KMS encryption key. If you are creating a DB cluster with the same Amazon account that owns the KMS encryption key used to encrypt the new DB cluster, then you can use the KMS key alias instead of the ARN for the KMS encryption key.

If an encryption key is not specified in `KmsKeyId`:

- If `ReplicationSourceIdentifier` identifies an encrypted source, then Amazon Neptune will use the encryption key used to encrypt the source. Otherwise, Amazon Neptune will use your default encryption key.
- If the `StorageEncrypted` parameter is `true` and `ReplicationSourceIdentifier` is not specified, then Amazon Neptune will use your default encryption key.

Amazon KMS creates the default encryption key for your Amazon account. Your Amazon account has a different default encryption key for each Amazon Region.

If you create a Read Replica of an encrypted DB cluster in another Amazon Region, you must set `KmsKeyId` to a KMS key ID that is valid in the destination Amazon Region. This key is used to encrypt the Read Replica in that Amazon Region.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

The port number on which the instances in the DB cluster accept connections.

Default: 8182

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

The daily time range during which automated backups are created if automated backups are enabled using the `BackupRetentionPeriod` parameter.

The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Region. To see the time blocks available, see [Adjusting the Preferred Maintenance Window](#) in the *Amazon Neptune User Guide*.

Constraints:

- Must be in the format `hh24:mi-hh24:mi`.
 - Must be in Universal Coordinated Time (UTC).
 - Must not conflict with the preferred maintenance window.
 - Must be at least 30 minutes.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

The weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Region, occurring on a random day of the week. To see the time blocks available, see [Adjusting the Preferred Maintenance Window](#) in the *Amazon Neptune User Guide*.

Valid Days: Mon, Tue, Wed, Thu, Fri, Sat, Sun.

Constraints: Minimum 30-minute window.

- **PreSignedUrl** (in the CLI: `--pre-signed-url`, or in Python: `pre_signed_url`) – UTF-8 string.

This parameter is not currently supported.

- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.

The Amazon Resource Name (ARN) of the source DB instance or DB cluster if this DB cluster is created as a Read Replica.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster is encrypted.

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to assign to the new DB cluster.

- **VpcSecurityGroupIds** (in the CLI: `--vpc-security-group-ids`, or in Python: `vpc_security_group_ids`) – An array of `VpcSecurityGroupId`— UTF-8 strings.

A list of EC2 VPC security groups to associate with this DB cluster.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called “DescribeDBClusters” \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to `true`, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB cluster.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.
Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.
- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.
True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.
- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.
If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.
- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.
Specifies the latest time to which a database can be restored with point-in-time restore.
- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.
Specifies whether the DB cluster has instances in multiple Availability Zones.
- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.
Not supported by Neptune.
- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.
Specifies the progress of the operation as a percentage.
- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).
Specifies the port that the database engine is listening on.
- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.
Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.
Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).
- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.
The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.
If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.
- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.
Contains one or more identifiers of the Read Replicas associated with this DB cluster.
- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.

Not supported by Neptune.

- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.

Not supported by Neptune.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the current state of this DB cluster.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster is encrypted.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterAlreadyExistsFault](#) (p. 940)
- [InsufficientStorageClusterCapacityFault](#) (p. 948)
- [DBClusterQuotaExceededFault](#) (p. 941)
- [StorageQuotaExceededFault](#) (p. 953)
- [DBSubnetGroupNotFoundFault](#) (p. 946)
- [InvalidVPCNetworkStateFault](#) (p. 951)
- [InvalidDBClusterStateFault](#) (p. 948)
- [InvalidDBSubnetGroupStateFault](#) (p. 949)
- [InvalidSubnet](#) (p. 950)
- [InvalidDBInstanceStateFault](#) (p. 948)
- [DBClusterParameterGroupNotFoundFault](#) (p. 941)
- [KMSKeyNotAccessibleFault](#) (p. 951)
- [DBClusterNotFoundFault](#) (p. 940)
- [DBInstanceNotFoundFault](#) (p. 942)
- [DBSubnetGroupDoesNotCoverEnoughAZs](#) (p. 945)

DeleteDBCluster (action)

The Amazon CLI name for this API is: `delete-db-cluster`, and the Python name is: `delete_db_cluster`.

The `DeleteDBCluster` action deletes a previously provisioned DB cluster. When you delete a DB cluster, all automated backups for that DB cluster are deleted and can't be recovered. Manual DB cluster snapshots of the specified DB cluster are not deleted.

Note that the DB Cluster cannot be deleted if deletion protection is enabled. To delete it, you must first set its `DeletionProtection` field to `False`.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

The DB cluster identifier for the DB cluster to be deleted. This parameter isn't case-sensitive.

Constraints:

- Must match an existing DBClusterIdentifier.
- **FinalDBSnapshotIdentifier** (in the CLI: `--final-db-snapshot-identifier`, or in Python: `final_db_snapshot_identifier`) – UTF-8 string.

The DB cluster snapshot identifier of the new DB cluster snapshot created when `SkipFinalSnapshot` is set to `false`.

Note

Specifying this parameter and also setting the `SkipFinalSnapshot` parameter to `true` results in an error.

Constraints:

- Must be 1 to 255 letters, numbers, or hyphens.
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens
- **SkipFinalSnapshot** (in the CLI: `--skip-final-snapshot`, or in Python: `skip_final_snapshot`) – Boolean.

Determines whether a final DB cluster snapshot is created before the DB cluster is deleted. If `true` is specified, no DB cluster snapshot is created. If `false` is specified, a DB cluster snapshot is created before the DB cluster is deleted.

Note

You must specify a `FinalDBSnapshotIdentifier` parameter if `SkipFinalSnapshot` is `false`.

Default: `false`

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called "DescribeDBClusters" \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to true, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to true, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB cluster.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.

- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies whether the DB cluster has instances in multiple Availability Zones.

- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.

Not supported by Neptune.

- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.
Specifies the progress of the operation as a percentage.
- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).
Specifies the port that the database engine is listening on.
- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.
Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.
Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).
- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.
The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.
- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.
Contains one or more identifiers of the Read Replicas associated with this DB cluster.
- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.
Not supported by Neptune.
- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.
Not supported by Neptune.
- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.
Specifies the current state of this DB cluster.
- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.
Specifies whether the DB cluster is encrypted.
- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.
Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterNotFoundFault](#) (p. 940)

- [InvalidDBClusterStateFault](#) (p. 948)
- [DBClusterSnapshotAlreadyExistsFault](#) (p. 942)
- [SnapshotQuotaExceededFault](#) (p. 953)
- [InvalidDBClusterSnapshotStateFault](#) (p. 948)

ModifyDBCluster (action)

The Amazon CLI name for this API is: `modify-db-cluster`, and the Python name is: `modify_db_cluster`.

Modify a setting for a DB cluster. You can change one or more database configuration parameters by specifying these parameters and the new values in the request.

Request

- **ApplyImmediately** (in the CLI: `--apply-immediately`, or in Python: `apply_immediately`) – Boolean.

A value that specifies whether the modifications in this request and any pending modifications are asynchronously applied as soon as possible, regardless of the `PreferredMaintenanceWindow` setting for the DB cluster. If this parameter is set to `false`, changes to the DB cluster are applied during the next maintenance window.

The `ApplyImmediately` parameter only affects `NewDBClusterIdentifier` values. If you set the `ApplyImmediately` parameter value to `false`, then changes to `NewDBClusterIdentifier` values are applied during the next maintenance window. All other changes are applied immediately, regardless of the value of the `ApplyImmediately` parameter.

Default: `false`

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

The number of days for which automated backups are retained. You must specify a minimum value of 1.

Default: 1

Constraints:

- Must be a value from 1 to 35
- **CloudwatchLogsExportConfiguration** (in the CLI: `--cloudwatch-logs-export-configuration`, or in Python: `cloudwatch_logs_export_configuration`) – A [CloudwatchLogsExportConfiguration](#) (p. 926) object.

The configuration setting for the log types to be enabled for export to CloudWatch Logs for a specific DB cluster.

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required:* UTF-8 string.

The DB cluster identifier for the cluster being modified. This parameter is not case-sensitive.

Constraints:

- Must match the identifier of an existing DBCluster.
- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

The name of the DB cluster parameter group to use for the DB cluster.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

A value that indicates whether the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled. By default, deletion protection is disabled.

- **EnableIAMDatabaseAuthentication** (in the CLI: `--enable-iam-database-authentication`, or in Python: `enable_iam_database_authentication`) – Boolean.

True to enable mapping of Amazon Identity and Access Management (IAM) accounts to database accounts, and otherwise false.

Default: `false`

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

The version number of the database engine to which you want to upgrade. Changing this parameter results in an outage. The change is applied during the next maintenance window unless the `ApplyImmediately` parameter is set to true.

For a list of valid engine versions, see [Engine Releases for Amazon Neptune](#), or call [DescribeDBEngineVersions](#).

- **NewDBClusterIdentifier** (in the CLI: `--new-db-cluster-identifier`, or in Python: `new_db_cluster_identifier`) – UTF-8 string.

The new DB cluster identifier for the DB cluster when renaming a DB cluster. This value is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens
- The first character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens

Example: `my-cluster2`

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

The port number on which the DB cluster accepts connections.

Constraints: Value must be 1150–65535

Default: The same port as the original DB cluster.

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

The daily time range during which automated backups are created if automated backups are enabled, using the `BackupRetentionPeriod` parameter.

The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Region.

Constraints:

- Must be in the format `hh24:mi-hh24:mi`.
- Must be in [Universal Coordinated Time \(UTC\)](#).

- Must not conflict with the preferred maintenance window.
- Must be at least 30 minutes.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

The weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Region, occurring on a random day of the week.

Valid Days: Mon, Tue, Wed, Thu, Fri, Sat, Sun.

Constraints: Minimum 30-minute window.

- **VpcSecurityGroupIds** (in the CLI: `--vpc-security-group-ids`, or in Python: `vpc_security_group_ids`) – An array of VpcSecurityGroupId— UTF-8 strings.

A list of VPC security groups that the DB cluster will belong to.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called “DescribeDBClusters” \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to `true`, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB cluster.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.

- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies whether the DB cluster has instances in multiple Availability Zones.

- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.

Not supported by Neptune.

- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.

Specifies the progress of the operation as a percentage.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

Specifies the port that the database engine is listening on.

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the [BackupRetentionPeriod](#).

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.

The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.

- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB cluster.

- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.

Not supported by Neptune.

- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.

Not supported by Neptune.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the current state of this DB cluster.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster is encrypted.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterNotFoundFault](#) (p. 940)
- [InvalidDBClusterStateFault](#) (p. 948)
- [StorageQuotaExceededFault](#) (p. 953)
- [DBSubnetGroupNotFoundFault](#) (p. 946)
- [InvalidVPCNetworkStateFault](#) (p. 951)
- [InvalidDBSubnetGroupStateFault](#) (p. 949)
- [InvalidSubnet](#) (p. 950)
- [DBClusterParameterGroupNotFoundFault](#) (p. 941)
- [InvalidDBSecurityGroupStateFault](#) (p. 949)
- [InvalidDBInstanceStateFault](#) (p. 948)

- [DBClusterAlreadyExistsFault](#) (p. 940)

StartDBCluster (action)

The Amazon CLI name for this API is: `start-db-cluster`, and the Python name is: `start_db_cluster`.

Starts an Amazon Neptune DB cluster that was stopped using the Amazon console, the Amazon CLI `stop-db-cluster` command, or the `StopDBCluster` API.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

The DB cluster identifier of the Neptune DB cluster to be started. This parameter is stored as a lowercase string.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called “DescribeDBClusters”](#) (p. 826) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole](#) (p. 792) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).
Not supported by Neptune.
- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.
Identifies the clone group to which the DB cluster is associated.
- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.
Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).
- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.
If set to true, tags are copied to any snapshot of the DB cluster that is created.
- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.
If set to true, the DB cluster can be cloned across accounts.
- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.
Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.
- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.
The Amazon Resource Name (ARN) for the DB cluster.
- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.
Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.
- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.
Provides the list of instances that make up the DB cluster.
- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.
Specifies the name of the DB cluster parameter group for the DB cluster.
- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.
The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.
- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.
Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.
- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.
Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.
- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB cluster.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.

- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies whether the DB cluster has instances in multiple Availability Zones.

- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.

Not supported by Neptune.

- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.

Specifies the progress of the operation as a percentage.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

Specifies the port that the database engine is listening on.

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.

The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.

- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB cluster.

- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.

Not supported by Neptune.

- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.

Not supported by Neptune.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the current state of this DB cluster.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster is encrypted.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterNotFoundFault](#) (p. 940)
- [InvalidDBClusterStateFault](#) (p. 948)
- [InvalidDBInstanceStateFault](#) (p. 948)

StopDBCluster (action)

The Amazon CLI name for this API is: `stop-db-cluster`, and the Python name is: `stop_db_cluster`.

Stops an Amazon Neptune DB cluster. When you stop a DB cluster, Neptune retains the DB cluster's metadata, including its endpoints and DB parameter groups.

Neptune also retains the transaction logs so you can do a point-in-time restore if necessary.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

The DB cluster identifier of the Neptune DB cluster to be stopped. This parameter is stored as a lowercase string.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called “DescribeDBClusters” \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to `true`, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.
Provides the name of the database engine to be used for this DB cluster.
- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.
Indicates the database engine version.
- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.
Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.
- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.
True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.
- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.
If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.
- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.
Specifies the latest time to which a database can be restored with point-in-time restore.
- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.
Specifies whether the DB cluster has instances in multiple Availability Zones.
- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.
Not supported by Neptune.
- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.
Specifies the progress of the operation as a percentage.
- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).
Specifies the port that the database engine is listening on.
- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.
Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.
Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).
- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.
The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.

- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.
Contains one or more identifiers of the Read Replicas associated with this DB cluster.
- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.
Not supported by Neptune.
- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.
Not supported by Neptune.
- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.
Specifies the current state of this DB cluster.
- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.
Specifies whether the DB cluster is encrypted.
- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of `VpcSecurityGroupMembership` (p. 937) `VpcSecurityGroupMembership` objects.
Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterNotFoundFault](#) (p. 940)
- [InvalidDBClusterStateFault](#) (p. 948)
- [InvalidDBInstanceStateFault](#) (p. 948)

AddRoleToDBCluster (action)

The Amazon CLI name for this API is: `add-role-to-db-cluster`, and the Python name is: `add_role_to_db_cluster`.

Associates an Identity and Access Management (IAM) role with an Neptune DB cluster.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.
The name of the DB cluster to associate the IAM role with.
- **FeatureName** (in the CLI: `--feature-name`, or in Python: `feature_name`) – UTF-8 string.
The name of the feature for the Neptune DB cluster that the IAM role is to be associated with. For the list of supported feature names, see [DBEngineVersion](#).
- **RoleArn** (in the CLI: `--role-arn`, or in Python: `role_arn`) – *Required*: UTF-8 string.
The Amazon Resource Name (ARN) of the IAM role to associate with the Neptune DB cluster, for example `arn:aws:iam::123456789012:role/NeptuneAccessRole`.

Response

- *No Response parameters.*

Errors

- [DBClusterNotFoundFault](#) (p. 940)
- [DBClusterRoleAlreadyExistsFault](#) (p. 941)
- [InvalidDBClusterStateFault](#) (p. 948)
- [DBClusterRoleQuotaExceededFault](#) (p. 942)

RemoveRoleFromDBCluster (action)

The Amazon CLI name for this API is: `remove-role-from-db-cluster`, and the Python name is: `remove_role_from_db_cluster`.

Disassociates an Identity and Access Management (IAM) role from a DB cluster.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

The name of the DB cluster to disassociate the IAM role from.

- **FeatureName** (in the CLI: `--feature-name`, or in Python: `feature_name`) – UTF-8 string.

The name of the feature for the DB cluster that the IAM role is to be disassociated from. For the list of supported feature names, see [DescribeDBEngineVersions](#).

- **RoleArn** (in the CLI: `--role-arn`, or in Python: `role_arn`) – *Required*: UTF-8 string.

The Amazon Resource Name (ARN) of the IAM role to disassociate from the DB cluster, for example `arn:aws:iam::123456789012:role/NeptuneAccessRole`.

Response

- *No Response parameters.*

Errors

- [DBClusterNotFoundFault](#) (p. 940)
- [DBClusterRoleNotFoundFault](#) (p. 941)
- [InvalidDBClusterStateFault](#) (p. 948)

FailoverDBCluster (action)

The Amazon CLI name for this API is: `failover-db-cluster`, and the Python name is: `failover_db_cluster`.

Forces a failover for a DB cluster.

A failover for a DB cluster promotes one of the Read Replicas (read-only instances) in the DB cluster to be the primary instance (the cluster writer).

Amazon Neptune will automatically fail over to a Read Replica, if one exists, when the primary instance fails. You can force a failover when you want to simulate a failure of a primary instance for testing.

Because each instance in a DB cluster has its own endpoint address, you will need to clean up and re-establish any existing connections that use those endpoint addresses when the failover is complete.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

A DB cluster identifier to force a failover for. This parameter is not case-sensitive.

Constraints:

- Must match the identifier of an existing DBCluster.
- **TargetDBInstanceIdentifier** (in the CLI: `--target-db-instance-identifier`, or in Python: `target_db_instance_identifier`) – UTF-8 string.

The name of the instance to promote to the primary instance.

You must specify the instance identifier for an Read Replica in the DB cluster. For example, `mydbcluster-replica1`.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called “DescribeDBClusters” \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to `true`, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB cluster.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.

- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies whether the DB cluster has instances in multiple Availability Zones.

- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.

Not supported by Neptune.

- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.

Specifies the progress of the operation as a percentage.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

Specifies the port that the database engine is listening on.

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the [BackupRetentionPeriod](#).

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.

The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.

- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB cluster.

- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.

Not supported by Neptune.

- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.

Not supported by Neptune.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the current state of this DB cluster.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster is encrypted.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterNotFoundFault](#) (p. 940)
- [InvalidDBClusterStateFault](#) (p. 948)
- [InvalidDBInstanceStateFault](#) (p. 948)

PromoteReadReplicaDBCluster (action)

The Amazon CLI name for this API is: `promote-read-replica-db-cluster`, and the Python name is: `promote_read_replica_db_cluster`.

Not supported.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

Not supported.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called “DescribeDBClusters” \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to `true`, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.

- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.
Provides the name of the database engine to be used for this DB cluster.
- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.
Indicates the database engine version.
- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.
Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.
- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.
True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.
- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.
If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.
- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.
Specifies the latest time to which a database can be restored with point-in-time restore.
- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.
Specifies whether the DB cluster has instances in multiple Availability Zones.
- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.
Not supported by Neptune.
- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.
Specifies the progress of the operation as a percentage.
- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).
Specifies the port that the database engine is listening on.
- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.
Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.
Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).
- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.
The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.

- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB cluster.
- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.

Not supported by Neptune.
- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.

Not supported by Neptune.
- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the current state of this DB cluster.
- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster is encrypted.
- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of `VpcSecurityGroupMembership` (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterNotFoundFault](#) (p. 940)
- [InvalidDBClusterStateFault](#) (p. 948)

DescribeDBClusters (action)

The Amazon CLI name for this API is: `describe-db-clusters`, and the Python name is: `describe_db_clusters`.

Returns information about provisioned DB clusters, and supports pagination.

Note

This operation can also return information for Amazon RDS clusters and Amazon DocDB clusters.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

The user-supplied DB cluster identifier. If this parameter is specified, information from only the specific DB cluster is returned. This parameter isn't case-sensitive.

Constraints:

- If supplied, must match an existing `DBClusterIdentifier`.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter](#) (p. 937) `Filter` objects.

A filter that specifies one or more DB clusters to describe.

Supported filters:

- `db-cluster-id` - Accepts DB cluster identifiers and DB cluster Amazon Resource Names (ARNs). The results list will only include information about the DB clusters identified by these ARNs.
- `engine` - Accepts an engine name (such as `neptune`), and restricts the results list to DB clusters created by that engine.

For example, to invoke this API from the Amazon CLI and filter so that only Neptune DB clusters are returned, you could use the following command:

Example

```
aws neptune describe-db-clusters \
    --filters Name=engine,Values=neptune
```

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous [the section called “DescribeDBClusters” \(p. 826\)](#) request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

- **DBClusters** (in the CLI: `--db-clusters`, or in Python: `db_clusters`) – An array of [DBCluster \(p. 788\)](#) DBCluster objects.

Contains a list of DB clusters for the user.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

A pagination token that can be used in a subsequent `DescribeDBClusters` request.

Errors

- [DBClusterNotFoundFault \(p. 940\)](#)

Neptune Instances API

Structures:

- [DBInstance \(structure\) \(p. 828\)](#)
- [DBInstanceStatusInfo \(structure\) \(p. 831\)](#)
- [OrderableDBInstanceOption \(structure\) \(p. 831\)](#)
- [ValidDBInstanceModificationsMessage \(structure\) \(p. 833\)](#)

Actions:

- [CreateDBInstance \(action\)](#) (p. 833)
- [DeleteDBInstance \(action\)](#) (p. 841)
- [ModifyDBInstance \(action\)](#) (p. 846)
- [RebootDBInstance \(action\)](#) (p. 855)
- [DescribeDBInstances \(action\)](#) (p. 859)
- [DescribeOrderableDBInstanceOptions \(action\)](#) (p. 860)
- [DescribeValidDBInstanceModifications \(action\)](#) (p. 861)

DBInstance (structure)

Contains the details of an Amazon Neptune DB instance.

This data type is used as a response element in the [the section called "DescribeDBInstances"](#) (p. 859) action.

Fields

- **AutoMinorVersionUpgrade** – Boolean.
Indicates that minor version patches are applied automatically.
- **AvailabilityZone** – UTF-8 string.
Specifies the name of the Availability Zone the DB instance is located in.
- **BackupRetentionPeriod** – Number (integer).
Specifies the number of days for which automatic DB snapshots are retained.
- **CACertificateIdentifier** – UTF-8 string.
The identifier of the CA certificate for this DB instance.
- **CopyTagsToSnapshot** – Boolean.
Specifies whether tags are copied from the DB instance to snapshots of the DB instance.
- **DBClusterIdentifier** – UTF-8 string.
If the DB instance is a member of a DB cluster, contains the name of the DB cluster that the DB instance is a member of.
- **DBInstanceArn** – UTF-8 string.
The Amazon Resource Name (ARN) for the DB instance.
- **DBInstanceClass** – UTF-8 string.
Contains the name of the compute and memory capacity class of the DB instance.
- **DBInstanceIdentifier** – UTF-8 string.
Contains a user-supplied database identifier. This identifier is the unique key that identifies a DB instance.
- **DBInstancePort** – Number (integer).
Specifies the port that the DB instance listens on. If the DB instance is part of a DB cluster, this can be a different port than the DB cluster port.
- **DBInstanceStatus** – UTF-8 string.
Specifies the current state of this database.

- **DbiResourceId** – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB instance. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB instance is accessed.

- **DBName** – UTF-8 string.

The database name.

- **DBParameterGroups** – An array of [DBParameterGroupStatus \(p. 864\)](#) DBParameterGroup objects.

Provides the list of DB parameter groups applied to this DB instance.

- **DBSecurityGroups** – An array of [DBSecurityGroupMembership \(p. 935\)](#) DBSecurityGroup objects.

Provides List of DB security group elements containing only `DBSecurityGroup.Name` and `DBSecurityGroup.Status` subelements.

- **DBSubnetGroup** – A [DBSubnetGroup \(p. 882\)](#) object.

Specifies information on the subnet group associated with the DB instance, including the name, description, and subnets in the subnet group.

- **DeletionProtection** – Boolean.

Indicates whether or not the DB instance has deletion protection enabled. The instance can't be deleted when deletion protection is enabled. See [Deleting a DB Instance](#).

- **DomainMemberships** – An array of [DomainMembership \(p. 936\)](#) DomainMembership objects.

Not supported

- **EnabledCloudwatchLogsExports** – An array of UTF-8 strings.

A list of log types that this DB instance is configured to export to CloudWatch Logs.

- **Endpoint** – An [Endpoint \(p. 936\)](#) object.

Specifies the connection endpoint.

- **Engine** – UTF-8 string.

Provides the name of the database engine to be used for this DB instance.

- **EngineVersion** – UTF-8 string.

Indicates the database engine version.

- **EnhancedMonitoringResourceArn** – UTF-8 string.

The Amazon Resource Name (ARN) of the Amazon CloudWatch Logs log stream that receives the Enhanced Monitoring metrics data for the DB instance.

- **IAMDatabaseAuthenticationEnabled** – Boolean.

True if Amazon Identity and Access Management (IAM) authentication is enabled, and otherwise false.

- **InstanceCreateTime** – Timestamp.

Provides the date and time the DB instance was created.

- **Iops** – Number (integer).

Specifies the Provisioned IOPS (I/O operations per second) value.

- **KmsKeyId** – UTF-8 string.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **LatestRestorableTime** – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **LicenseModel** – UTF-8 string.

License model information for this DB instance.

- **MonitoringInterval** – Number (integer).

The interval, in seconds, between points when Enhanced Monitoring metrics are collected for the DB instance.

- **MonitoringRoleArn** – UTF-8 string.

The ARN for the IAM role that permits Neptune to send Enhanced Monitoring metrics to Amazon CloudWatch Logs.

- **MultiAZ** – Boolean.

Specifies if the DB instance is a Multi-AZ deployment.

- **PendingModifiedValues** – A [PendingModifiedValues \(p. 928\)](#) object.

Specifies that changes to the DB instance are pending. This element is only included when changes are pending. Specific changes are identified by subelements.

- **PreferredBackupWindow** – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.

- **PreferredMaintenanceWindow** – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **PromotionTier** – Number (integer).

A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

- **PubliclyAccessible** – Boolean.

This flag should no longer be used.

- **ReadReplicaDBClusterIdentifiers** – An array of `ReadReplicaDBClusterIdentifier`— UTF-8 strings.

Contains one or more identifiers of DB clusters that are Read Replicas of this DB instance.

- **ReadReplicaDBInstanceIdentifiers** – An array of `ReadReplicaDBInstanceIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB instance.

- **ReadReplicaSourceDBInstanceIdentifier** – UTF-8 string.

Contains the identifier of the source DB instance if this DB instance is a Read Replica.

- **SecondaryAvailabilityZone** – UTF-8 string.

If present, specifies the name of the secondary Availability Zone for a DB instance with multi-AZ support.

- **StatusInfos** – An array of [DBInstanceStatusInfo \(p. 831\)](#) `DBInstanceStatusInfo` objects.

The status of a Read Replica. If the instance is not a Read Replica, this is blank.

- **StorageEncrypted** – Boolean.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **StorageType** – UTF-8 string.

Specifies the storage type associated with DB instance.

- **TdeCredentialArn** – UTF-8 string.

The ARN from the key store with which the instance is associated for TDE encryption.

- **Timezone** – UTF-8 string.

Not supported.

- **VpcSecurityGroups** – An array of [VpcSecurityGroupMembership \(p. 937\)](#) VpcSecurityGroupMembership objects.

Provides a list of VPC security group elements that the DB instance belongs to.

DBInstance is used as the response element for:

- [CreateDBInstance \(p. 833\)](#)
- [DeleteDBInstance \(p. 841\)](#)
- [ModifyDBInstance \(p. 846\)](#)
- [RebootDBInstance \(p. 855\)](#)

DBInstanceStatusInfo (structure)

Provides a list of status information for a DB instance.

Fields

- **Message** – UTF-8 string.

Details of the error if there is an error for the instance. If the instance is not in an error state, this value is blank.

- **Normal** – Boolean.

Boolean value that is true if the instance is operating normally, or false if the instance is in an error state.

- **Status** – UTF-8 string.

Status of the DB instance. For a StatusType of read replica, the values can be replicating, error, stopped, or terminated.

- **StatusType** – UTF-8 string.

This value is currently "read replication."

OrderableDBInstanceOption (structure)

Contains a list of available options for a DB instance.

This data type is used as a response element in the [the section called "DescribeOrderableDBInstanceOptions" \(p. 860\)](#) action.

Fields

- **AvailabilityZones** – An array of [AvailabilityZone \(p. 935\)](#) AvailabilityZone objects.

A list of Availability Zones for a DB instance.

- **DBInstanceClass** – UTF-8 string.

The DB instance class for a DB instance.

- **Engine** – UTF-8 string.

The engine type of a DB instance.

- **EngineVersion** – UTF-8 string.

The engine version of a DB instance.

- **LicenseModel** – UTF-8 string.

The license model for a DB instance.

- **MaxIopsPerDbInstance** – Number (integer).

Maximum total provisioned IOPS for a DB instance.

- **MaxIopsPerGib** – Number (double).

Maximum provisioned IOPS per GiB for a DB instance.

- **MaxStorageSize** – Number (integer).

Maximum storage size for a DB instance.

- **MinIopsPerDbInstance** – Number (integer).

Minimum total provisioned IOPS for a DB instance.

- **MinIopsPerGib** – Number (double).

Minimum provisioned IOPS per GiB for a DB instance.

- **MinStorageSize** – Number (integer).

Minimum storage size for a DB instance.

- **MultiAZCapable** – Boolean.

Indicates whether a DB instance is Multi-AZ capable.

- **ReadReplicaCapable** – Boolean.

Indicates whether a DB instance can have a Read Replica.

- **StorageType** – UTF-8 string.

Indicates the storage type for a DB instance.

- **SupportsEnhancedMonitoring** – Boolean.

Indicates whether a DB instance supports Enhanced Monitoring at intervals from 1 to 60 seconds.

- **SupportsIAMDatabaseAuthentication** – Boolean.

Indicates whether a DB instance supports IAM database authentication.

- **SupportsIops** – Boolean.

Indicates whether a DB instance supports provisioned IOPS.

- **SupportsStorageEncryption** – Boolean.

Indicates whether a DB instance supports encrypted storage.

- **Vpc** – Boolean.

Indicates whether a DB instance is in a VPC.

ValidDBInstanceModificationsMessage (structure)

Information about valid modifications that you can make to your DB instance. Contains the result of a successful call to the [the section called "DescribeValidDBInstanceModifications" \(p. 861\)](#) action. You can use this information when you call [the section called "ModifyDBInstance" \(p. 846\)](#).

Fields

- **Storage** – An array of ValidStorageOptions— a structures.

Valid storage options for your DB instance.

ValidDBInstanceModificationsMessage is used as the response element for:

- [DescribeValidDBInstanceModifications \(p. 861\)](#)

CreateDBInstance (action)

The Amazon CLI name for this API is: `create-db-instance`, and the Python name is: `create_db_instance`.

Creates a new DB instance.

Request

- **AutoMinorVersionUpgrade** (in the CLI: `--auto-minor-version-upgrade`, or in Python: `auto_minor_version_upgrade`) – Boolean.

Indicates that minor engine upgrades are applied automatically to the DB instance during the maintenance window.

Default: `true`

- **AvailabilityZone** (in the CLI: `--availability-zone`, or in Python: `availability_zone`) – UTF-8 string.

The EC2 Availability Zone that the DB instance is created in

Default: A random, system-chosen Availability Zone in the endpoint's Amazon Region.

Example: `us-east-1d`

Constraint: The AvailabilityZone parameter can't be specified if the MultiAZ parameter is set to `true`. The specified Availability Zone must be in the same Amazon Region as the current endpoint.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

The number of days for which automated backups are retained.

Not applicable. The retention period for automated backups is managed by the DB cluster. For more information, see [the section called "CreateDBCluster" \(p. 793\)](#).

Default: `1`

Constraints:

- Must be a value from 0 to 35
- Cannot be set to 0 if the DB instance is a source to Read Replicas

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

True to copy all tags from the DB instance to snapshots of the DB instance, and otherwise false. The default is false.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

The identifier of the DB cluster that the instance will belong to.

For information on creating a DB cluster, see [the section called “CreateDBCluster” \(p. 793\)](#).

Type: String

- **DBInstanceClass** (in the CLI: `--db-instance-class`, or in Python: `db_instance_class`) – *Required*: UTF-8 string.

The compute and memory capacity of the DB instance, for example, `db.m4.large`. Not all DB instance classes are available in all Amazon Regions.

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – *Required*: UTF-8 string.

The DB instance identifier. This parameter is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- First character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `mydbinstance`

- **DBName** (in the CLI: `--db-name`, or in Python: `db_name`) – UTF-8 string.

Not supported.

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – UTF-8 string.

The name of the DB parameter group to associate with this DB instance. If this argument is omitted, the default DBParameterGroup for the specified engine is used.

Constraints:

- Must be 1 to 255 letters, numbers, or hyphens.
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens
- **DBSecurityGroups** (in the CLI: `--db-security-groups`, or in Python: `db_security_groups`) – An array of DBSecurityGroupName— UTF-8 strings.

A list of DB security groups to associate with this DB instance.

Default: The default DB security group for the database engine.

- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

A DB subnet group to associate with this DB instance.

If there is no DB subnet group, then it is a non-VPC DB instance.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

A value that indicates whether the DB instance has deletion protection enabled. The database can't be deleted when deletion protection is enabled. By default, deletion protection is disabled. See [Deleting a DB Instance](#).

DB instances in a DB cluster can be deleted even when deletion protection is enabled in their parent DB cluster.

- **Domain** (in the CLI: `--domain`, or in Python: `domain`) – UTF-8 string.

Specify the Active Directory Domain to create the instance in.

- **DomainIAMRoleName** (in the CLI: `--domain-iam-role-name`, or in Python: `domain_iam_role_name`) – UTF-8 string.

Specify the name of the IAM role to be used when making API calls to the Directory Service.

- **EnableCloudwatchLogsExports** (in the CLI: `--enable-cloudwatch-logs-exports`, or in Python: `enable_cloudwatch_logs_exports`) – An array of UTF-8 strings.

The list of log types that need to be enabled for exporting to CloudWatch Logs.

- **EnableIAMDatabaseAuthentication** (in the CLI: `--enable-iam-database-authentication`, or in Python: `enable_iam_database_authentication`) – Boolean.

Not supported by Neptune (ignored).

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – *Required*: UTF-8 string.

The name of the database engine to be used for this instance.

Valid Values: `neptune`

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

The version number of the database engine to use. Currently, setting this parameter has no effect.

- **Iops** (in the CLI: `--iops`, or in Python: `iops`) – Number (integer).

The amount of Provisioned IOPS (input/output operations per second) to be initially allocated for the DB instance.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

The Amazon KMS key identifier for an encrypted DB instance.

The KMS key identifier is the Amazon Resource Name (ARN) for the KMS encryption key. If you are creating a DB instance with the same Amazon account that owns the KMS encryption key used to encrypt the new DB instance, then you can use the KMS key alias instead of the ARN for the KMS encryption key.

Not applicable. The KMS key identifier is managed by the DB cluster. For more information, see [the section called "CreateDBCluster" \(p. 793\)](#).

If the `StorageEncrypted` parameter is true, and you do not specify a value for the `KmsKeyId` parameter, then Amazon Neptune will use your default encryption key. Amazon KMS creates the default encryption key for your Amazon account. Your Amazon account has a different default encryption key for each Amazon Region.

- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.

License model information for this DB instance.

Valid values: `license-included` | `bring-your-own-license` | `general-public-license`

- **MonitoringInterval** (in the CLI: `--monitoring-interval`, or in Python: `monitoring_interval`) – Number (integer).

The interval, in seconds, between points when Enhanced Monitoring metrics are collected for the DB instance. To disable collecting Enhanced Monitoring metrics, specify 0. The default is 0.

If `MonitoringRoleArn` is specified, then you must also set `MonitoringInterval` to a value other than 0.

Valid Values: 0, 1, 5, 10, 15, 30, 60

- **MonitoringRoleArn** (in the CLI: `--monitoring-role-arn`, or in Python: `monitoring_role_arn`) – UTF-8 string.

The ARN for the IAM role that permits Neptune to send enhanced monitoring metrics to Amazon CloudWatch Logs. For example, `arn:aws:iam:123456789012:role/emaccess`.

If `MonitoringInterval` is set to a value other than 0, then you must supply a `MonitoringRoleArn` value.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies if the DB instance is a Multi-AZ deployment. You can't set the `AvailabilityZone` parameter if the `MultiAZ` parameter is set to true.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

The port number on which the database accepts connections.

Not applicable. The port is managed by the DB cluster. For more information, see [the section called "CreateDBCluster" \(p. 793\)](#).

Default: 8182

Type: Integer

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

The daily time range during which automated backups are created.

Not applicable. The daily time range for creating automated backups is managed by the DB cluster. For more information, see [the section called "CreateDBCluster" \(p. 793\)](#).

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

The time range each week during which system maintenance can occur, in Universal Coordinated Time (UTC).

Format: `ddd:hh24:mi-ddd:hh24:mi`

The default is a 30-minute window selected at random from an 8-hour block of time for each Amazon Region, occurring on a random day of the week.

Valid Days: Mon, Tue, Wed, Thu, Fri, Sat, Sun.

Constraints: Minimum 30-minute window.

- **PromotionTier** (in the CLI: `--promotion-tier`, or in Python: `promotion_tier`) – Number (integer).

A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

Default: 1

Valid Values: 0 - 15

- **PubliclyAccessible** (in the CLI: `--publicly-accessible`, or in Python: `publicly_accessible`) – Boolean.

This flag should no longer be used.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB instance is encrypted.

Not applicable. The encryption for DB instances is managed by the DB cluster. For more information, see [the section called “CreateDBCluster” \(p. 793\)](#).

Default: false

- **StorageType** (in the CLI: `--storage-type`, or in Python: `storage_type`) – UTF-8 string.

Specifies the storage type to be associated with the DB instance.

Not applicable. Storage is managed by the DB Cluster.

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to assign to the new instance.

- **TdeCredentialArn** (in the CLI: `--tde-credential-arn`, or in Python: `tde_credential_arn`) – UTF-8 string.

The ARN from the key store with which to associate the instance for TDE encryption.

- **TdeCredentialPassword** (in the CLI: `--tde-credential-password`, or in Python: `tde_credential_password`) – UTF-8 string.

The password for the given ARN from the key store in order to access the device.

- **Timezone** (in the CLI: `--timezone`, or in Python: `timezone`) – UTF-8 string.

The time zone of the DB instance.

- **VpcSecurityGroupIds** (in the CLI: `--vpc-security-group-ids`, or in Python: `vpc_security_group_ids`) – An array of `VpcSecurityGroupId`— UTF-8 strings.

A list of EC2 VPC security groups to associate with this DB instance.

Not applicable. The associated list of EC2 VPC security groups is managed by the DB cluster. For more information, see [the section called “CreateDBCluster” \(p. 793\)](#).

Default: The default EC2 VPC security group for the DB subnet group's VPC.

Response

Contains the details of an Amazon Neptune DB instance.

This data type is used as a response element in the [the section called “DescribeDBInstances” \(p. 859\)](#) action.

- **AutoMinorVersionUpgrade** (in the CLI: `--auto-minor-version-upgrade`, or in Python: `auto_minor_version_upgrade`) – Boolean.

Indicates that minor version patches are applied automatically.

- **AvailabilityZone** (in the CLI: `--availability-zone`, or in Python: `availability_zone`) – UTF-8 string.

Specifies the name of the Availability Zone the DB instance is located in.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **CACertificateIdentifier** (in the CLI: `--ca-certificate-identifier`, or in Python: `ca_certificate_identifier`) – UTF-8 string.

The identifier of the CA certificate for this DB instance.

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

Specifies whether tags are copied from the DB instance to snapshots of the DB instance.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

If the DB instance is a member of a DB cluster, contains the name of the DB cluster that the DB instance is a member of.

- **DBInstanceArn** (in the CLI: `--db-instance-arn`, or in Python: `db_instance_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB instance.

- **DBInstanceClass** (in the CLI: `--db-instance-class`, or in Python: `db_instance_class`) – UTF-8 string.

Contains the name of the compute and memory capacity class of the DB instance.

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – UTF-8 string.

Contains a user-supplied database identifier. This identifier is the unique key that identifies a DB instance.

- **DBInstancePort** (in the CLI: `--db-instance-port`, or in Python: `db_instance_port`) – Number (integer).

Specifies the port that the DB instance listens on. If the DB instance is part of a DB cluster, this can be a different port than the DB cluster port.

- **DBInstanceStatus** (in the CLI: `--db-instance-status`, or in Python: `db_instance_status`) – UTF-8 string.

Specifies the current state of this database.

- **DbiResourceId** (in the CLI: `--dbi-resource-id`, or in Python: `dbi_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB instance. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB instance is accessed.

- **DBName** (in the CLI: `--db-name`, or in Python: `db_name`) – UTF-8 string.

The database name.

- **DBParameterGroups** (in the CLI: `--db-parameter-groups`, or in Python: `db_parameter_groups`) – An array of [DBParameterGroupStatus](#) (p. 864) `DBParameterGroup` objects.

Provides the list of DB parameter groups applied to this DB instance.

- **DBSecurityGroups** (in the CLI: `--db-security-groups`, or in Python: `db_security_groups`) – An array of [DBSecurityGroupMembership](#) (p. 935) `DBSecurityGroup` objects.

Provides List of DB security group elements containing only `DBSecurityGroup.Name` and `DBSecurityGroup.Status` subelements.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – A [DBSubnetGroup \(p. 882\)](#) object.

Specifies information on the subnet group associated with the DB instance, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB instance has deletion protection enabled. The instance can't be deleted when deletion protection is enabled. See [Deleting a DB Instance](#).

- **DomainMemberships** (in the CLI: `--domain-memberships`, or in Python: `domain_memberships`) – An array of [DomainMembership \(p. 936\)](#) DomainMembership objects.

Not supported

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB instance is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – An [Endpoint \(p. 936\)](#) object.

Specifies the connection endpoint.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB instance.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **EnhancedMonitoringResourceArn** (in the CLI: `--enhanced-monitoring-resource-arn`, or in Python: `enhanced_monitoring_resource_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) of the Amazon CloudWatch Logs log stream that receives the Enhanced Monitoring metrics data for the DB instance.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if Amazon Identity and Access Management (IAM) authentication is enabled, and otherwise false.

- **InstanceCreateTime** (in the CLI: `--instance-create-time`, or in Python: `instance_create_time`) – Timestamp.

Provides the date and time the DB instance was created.

- **Iops** (in the CLI: `--iops`, or in Python: `iops`) – Number (integer).

Specifies the Provisioned IOPS (I/O operations per second) value.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.

License model information for this DB instance.

- **MonitoringInterval** (in the CLI: `--monitoring-interval`, or in Python: `monitoring_interval`) – Number (integer).

The interval, in seconds, between points when Enhanced Monitoring metrics are collected for the DB instance.

- **MonitoringRoleArn** (in the CLI: `--monitoring-role-arn`, or in Python: `monitoring_role_arn`) – UTF-8 string.

The ARN for the IAM role that permits Neptune to send Enhanced Monitoring metrics to Amazon CloudWatch Logs.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies if the DB instance is a Multi-AZ deployment.

- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [PendingModifiedValues \(p. 928\)](#) object.

Specifies that changes to the DB instance are pending. This element is only included when changes are pending. Specific changes are identified by subelements.

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **PromotionTier** (in the CLI: `--promotion-tier`, or in Python: `promotion_tier`) – Number (integer).

A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

- **PubliclyAccessible** (in the CLI: `--publicly-accessible`, or in Python: `publicly_accessible`) – Boolean.

This flag should no longer be used.

- **ReadReplicaDBClusterIdentifiers** (in the CLI: `--read-replica-db-cluster-identifiers`, or in Python: `read_replica_db_cluster_identifiers`) – An array of `ReadReplicaDBClusterIdentifier`— UTF-8 strings.

Contains one or more identifiers of DB clusters that are Read Replicas of this DB instance.

- **ReadReplicaDBInstanceIdentifiers** (in the CLI: `--read-replica-db-instance-identifiers`, or in Python: `read_replica_db_instance_identifiers`) – An array of `ReadReplicaDBInstanceIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB instance.

- **ReadReplicaSourceDBInstanceIdentifier** (in the CLI: `--read-replica-source-db-instance-identifier`, or in Python: `read_replica_source_db_instance_identifier`) – UTF-8 string.

Contains the identifier of the source DB instance if this DB instance is a Read Replica.

- **SecondaryAvailabilityZone** (in the CLI: `--secondary-availability-zone`, or in Python: `secondary_availability_zone`) – UTF-8 string.

If present, specifies the name of the secondary Availability Zone for a DB instance with multi-AZ support.

- **StatusInfos** (in the CLI: `--status-infos`, or in Python: `status_infos`) – An array of [DBInstanceStatusInfo \(p. 831\)](#) `DBInstanceStatusInfo` objects.

The status of a Read Replica. If the instance is not a Read Replica, this is blank.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **StorageType** (in the CLI: `--storage-type`, or in Python: `storage_type`) – UTF-8 string.

Specifies the storage type associated with DB instance.

- **TdeCredentialArn** (in the CLI: `--tde-credential-arn`, or in Python: `tde_credential_arn`) – UTF-8 string.

The ARN from the key store with which the instance is associated for TDE encryption.

- **Timezone** (in the CLI: `--timezone`, or in Python: `timezone`) – UTF-8 string.

Not supported.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security group elements that the DB instance belongs to.

Errors

- [DBInstanceAlreadyExistsFault](#) (p. 942)
- [InsufficientDBInstanceCapacityFault](#) (p. 947)
- [DBParameterGroupNotFoundFault](#) (p. 943)
- [DBSecurityGroupNotFoundFault](#) (p. 944)
- [InstanceQuotaExceededFault](#) (p. 947)
- [StorageQuotaExceededFault](#) (p. 953)
- [DBSubnetGroupNotFoundFault](#) (p. 946)
- [DBSubnetGroupDoesNotCoverEnoughAZs](#) (p. 945)
- [InvalidDBClusterStateFault](#) (p. 948)
- [InvalidSubnet](#) (p. 950)
- [InvalidVPCNetworkStateFault](#) (p. 951)
- [ProvisionedIopsNotAvailableInAZFault](#) (p. 952)
- [OptionGroupNotFoundFault](#) (p. 951)
- [DBClusterNotFoundFault](#) (p. 940)
- [StorageTypeNotSupportedFault](#) (p. 954)
- [AuthorizationNotFoundFault](#) (p. 940)
- [KMSKeyNotAccessibleFault](#) (p. 951)
- [DomainNotFoundFault](#) (p. 946)

DeleteDBInstance (action)

The Amazon CLI name for this API is: `delete-db-instance`, and the Python name is: `delete_db_instance`.

The `DeleteDBInstance` action deletes a previously provisioned DB instance. When you delete a DB instance, all automated backups for that instance are deleted and can't be recovered. Manual DB snapshots of the DB instance to be deleted by `DeleteDBInstance` are not deleted.

If you request a final DB snapshot the status of the Amazon Neptune DB instance is `deleting` until the DB snapshot is created. The API action `DescribeDBInstance` is used to monitor the status of this operation. The action can't be canceled or reverted once submitted.

Note that when a DB instance is in a failure state and has a status of `failed`, `incompatible-restore`, or `incompatible-network`, you can only delete it when the `SkipFinalSnapshot` parameter is set to `true`.

You can't delete a DB instance if it is the only instance in the DB cluster, or if it has deletion protection enabled.

Request

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – *Required*: UTF-8 string.

The DB instance identifier for the DB instance to be deleted. This parameter isn't case-sensitive.

Constraints:

- Must match the name of an existing DB instance.
- **FinalDBSnapshotIdentifier** (in the CLI: `--final-db-snapshot-identifier`, or in Python: `final_db_snapshot_identifier`) – UTF-8 string.

The `DBSnapshotIdentifier` of the new `DBSnapshot` created when `SkipFinalSnapshot` is set to `false`.

Note

Specifying this parameter and also setting the `SkipFinalSnapshot` parameter to `true` results in an error.

Constraints:

- Must be 1 to 255 letters or numbers.
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens
- Cannot be specified when deleting a Read Replica.
- **SkipFinalSnapshot** (in the CLI: `--skip-final-snapshot`, or in Python: `skip_final_snapshot`) – Boolean.

Determines whether a final DB snapshot is created before the DB instance is deleted. If `true` is specified, no `DBSnapshot` is created. If `false` is specified, a DB snapshot is created before the DB instance is deleted.

Note that when a DB instance is in a failure state and has a status of `'failed'`, `'incompatible-restore'`, or `'incompatible-network'`, it can only be deleted when the `SkipFinalSnapshot` parameter is set to `"true"`.

Specify `true` when deleting a Read Replica.

Note

The `FinalDBSnapshotIdentifier` parameter must be specified if `SkipFinalSnapshot` is `false`.

Default: `false`

Response

Contains the details of an Amazon Neptune DB instance.

This data type is used as a response element in the [the section called "DescribeDBInstances" \(p. 859\)](#) action.

- **AutoMinorVersionUpgrade** (in the CLI: `--auto-minor-version-upgrade`, or in Python: `auto_minor_version_upgrade`) – Boolean.

Indicates that minor version patches are applied automatically.

- **AvailabilityZone** (in the CLI: `--availability-zone`, or in Python: `availability_zone`) – UTF-8 string.

Specifies the name of the Availability Zone the DB instance is located in.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **CACertificateIdentifier** (in the CLI: `--ca-certificate-identifier`, or in Python: `ca_certificate_identifier`) – UTF-8 string.

The identifier of the CA certificate for this DB instance.

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

Specifies whether tags are copied from the DB instance to snapshots of the DB instance.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

If the DB instance is a member of a DB cluster, contains the name of the DB cluster that the DB instance is a member of.

- **DBInstanceArn** (in the CLI: `--db-instance-arn`, or in Python: `db_instance_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB instance.

- **DBInstanceClass** (in the CLI: `--db-instance-class`, or in Python: `db_instance_class`) – UTF-8 string.

Contains the name of the compute and memory capacity class of the DB instance.

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – UTF-8 string.

Contains a user-supplied database identifier. This identifier is the unique key that identifies a DB instance.

- **DBInstancePort** (in the CLI: `--db-instance-port`, or in Python: `db_instance_port`) – Number (integer).

Specifies the port that the DB instance listens on. If the DB instance is part of a DB cluster, this can be a different port than the DB cluster port.

- **DBInstanceStatus** (in the CLI: `--db-instance-status`, or in Python: `db_instance_status`) – UTF-8 string.

Specifies the current state of this database.

- **DbiResourceId** (in the CLI: `--dbi-resource-id`, or in Python: `dbi_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB instance. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB instance is accessed.

- **DBName** (in the CLI: `--db-name`, or in Python: `db_name`) – UTF-8 string.

The database name.

- **DBParameterGroups** (in the CLI: `--db-parameter-groups`, or in Python: `db_parameter_groups`) – An array of [DBParameterGroupStatus](#) (p. 864) `DBParameterGroup` objects.

Provides the list of DB parameter groups applied to this DB instance.

- **DBSecurityGroups** (in the CLI: `--db-security-groups`, or in Python: `db_security_groups`) – An array of [DBSecurityGroupMembership \(p. 935\)](#) `DBSecurityGroup` objects.

Provides List of DB security group elements containing only `DBSecurityGroup.Name` and `DBSecurityGroup.Status` subelements.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – A [DBSubnetGroup \(p. 882\)](#) object.

Specifies information on the subnet group associated with the DB instance, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB instance has deletion protection enabled. The instance can't be deleted when deletion protection is enabled. See [Deleting a DB Instance](#).

- **DomainMemberships** (in the CLI: `--domain-memberships`, or in Python: `domain_memberships`) – An array of [DomainMembership \(p. 936\)](#) `DomainMembership` objects.

Not supported

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB instance is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – An [Endpoint \(p. 936\)](#) object.

Specifies the connection endpoint.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB instance.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **EnhancedMonitoringResourceArn** (in the CLI: `--enhanced-monitoring-resource-arn`, or in Python: `enhanced_monitoring_resource_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) of the Amazon CloudWatch Logs log stream that receives the Enhanced Monitoring metrics data for the DB instance.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if Amazon Identity and Access Management (IAM) authentication is enabled, and otherwise false.

- **InstanceCreateTime** (in the CLI: `--instance-create-time`, or in Python: `instance_create_time`) – Timestamp.

Provides the date and time the DB instance was created.

- **Iops** (in the CLI: `--iops`, or in Python: `iops`) – Number (integer).

Specifies the Provisioned IOPS (I/O operations per second) value.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.

License model information for this DB instance.

- **MonitoringInterval** (in the CLI: `--monitoring-interval`, or in Python: `monitoring_interval`) – Number (integer).

The interval, in seconds, between points when Enhanced Monitoring metrics are collected for the DB instance.

- **MonitoringRoleArn** (in the CLI: `--monitoring-role-arn`, or in Python: `monitoring_role_arn`) – UTF-8 string.

The ARN for the IAM role that permits Neptune to send Enhanced Monitoring metrics to Amazon CloudWatch Logs.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies if the DB instance is a Multi-AZ deployment.

- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [PendingModifiedValues \(p. 928\)](#) object.

Specifies that changes to the DB instance are pending. This element is only included when changes are pending. Specific changes are identified by subelements.

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **PromotionTier** (in the CLI: `--promotion-tier`, or in Python: `promotion_tier`) – Number (integer).

A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

- **PubliclyAccessible** (in the CLI: `--publicly-accessible`, or in Python: `publicly_accessible`) – Boolean.

This flag should no longer be used.

- **ReadReplicaDBClusterIdentifiers** (in the CLI: `--read-replica-db-cluster-identifiers`, or in Python: `read_replica_db_cluster_identifiers`) – An array of `ReadReplicaDBClusterIdentifier`— UTF-8 strings.

Contains one or more identifiers of DB clusters that are Read Replicas of this DB instance.

- **ReadReplicaDBInstanceIdentifiers** (in the CLI: `--read-replica-db-instance-identifiers`, or in Python: `read_replica_db_instance_identifiers`) – An array of `ReadReplicaDBInstanceIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB instance.

- **ReadReplicaSourceDBInstanceIdentifier** (in the CLI: `--read-replica-source-db-instance-identifier`, or in Python: `read_replica_source_db_instance_identifier`) – UTF-8 string.

Contains the identifier of the source DB instance if this DB instance is a Read Replica.

- **SecondaryAvailabilityZone** (in the CLI: `--secondary-availability-zone`, or in Python: `secondary_availability_zone`) – UTF-8 string.

If present, specifies the name of the secondary Availability Zone for a DB instance with multi-AZ support.

- **StatusInfos** (in the CLI: `--status-infos`, or in Python: `status_infos`) – An array of [DBInstanceStatusInfo](#) (p. 831) `DBInstanceStatusInfo` objects.

The status of a Read Replica. If the instance is not a Read Replica, this is blank.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **StorageType** (in the CLI: `--storage-type`, or in Python: `storage_type`) – UTF-8 string.

Specifies the storage type associated with DB instance.

- **TdeCredentialArn** (in the CLI: `--tde-credential-arn`, or in Python: `tde_credential_arn`) – UTF-8 string.

The ARN from the key store with which the instance is associated for TDE encryption.

- **Timezone** (in the CLI: `--timezone`, or in Python: `timezone`) – UTF-8 string.

Not supported.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security group elements that the DB instance belongs to.

Errors

- [DBInstanceNotFoundFault](#) (p. 942)
- [InvalidDBInstanceStateFault](#) (p. 948)
- [DBSnapshotAlreadyExistsFault](#) (p. 944)
- [SnapshotQuotaExceededFault](#) (p. 953)
- [InvalidDBClusterStateFault](#) (p. 948)

ModifyDBInstance (action)

The Amazon CLI name for this API is: `modify-db-instance`, and the Python name is: `modify_db_instance`.

Modifies settings for a DB instance. You can change one or more database configuration parameters by specifying these parameters and the new values in the request. To learn what modifications you can make to your DB instance, call [the section called “DescribeValidDBInstanceModifications”](#) (p. 861) before you call [the section called “ModifyDBInstance”](#) (p. 846).

Request

- **AllowMajorVersionUpgrade** (in the CLI: `--allow-major-version-upgrade`, or in Python: `allow_major_version_upgrade`) – Boolean.

Indicates that major version upgrades are allowed. Changing this parameter doesn't result in an outage and the change is asynchronously applied as soon as possible.

- **ApplyImmediately** (in the CLI: `--apply-immediately`, or in Python: `apply_immediately`) – Boolean.

Specifies whether the modifications in this request and any pending modifications are asynchronously applied as soon as possible, regardless of the `PreferredMaintenanceWindow` setting for the DB instance.

If this parameter is set to `false`, changes to the DB instance are applied during the next maintenance window. Some parameter changes can cause an outage and are applied on the next call to [the section called “RebootDBInstance” \(p. 855\)](#), or the next failure reboot.

Default: `false`

- **AutoMinorVersionUpgrade** (in the CLI: `--auto-minor-version-upgrade`, or in Python: `auto_minor_version_upgrade`) – Boolean.

Indicates that minor version upgrades are applied automatically to the DB instance during the maintenance window. Changing this parameter doesn't result in an outage except in the following case and the change is asynchronously applied as soon as possible. An outage will result if this parameter is set to `true` during the maintenance window, and a newer minor version is available, and Neptune has enabled auto patching for that engine version.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Not applicable. The retention period for automated backups is managed by the DB cluster. For more information, see [the section called “ModifyDBCluster” \(p. 804\)](#).

Default: Uses existing setting

- **CACertificateIdentifier** (in the CLI: `--ca-certificate-identifier`, or in Python: `ca_certificate_identifier`) – UTF-8 string.

Indicates the certificate that needs to be associated with the instance.

- **CloudwatchLogsExportConfiguration** (in the CLI: `--cloudwatch-logs-export-configuration`, or in Python: `cloudwatch_logs_export_configuration`) – A [CloudwatchLogsExportConfiguration \(p. 926\)](#) object.

The configuration setting for the log types to be enabled for export to CloudWatch Logs for a specific DB instance or DB cluster.

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

True to copy all tags from the DB instance to snapshots of the DB instance, and otherwise false. The default is false.

- **DBInstanceClass** (in the CLI: `--db-instance-class`, or in Python: `db_instance_class`) – UTF-8 string.

The new compute and memory capacity of the DB instance, for example, `db.m4.large`. Not all DB instance classes are available in all Amazon Regions.

If you modify the DB instance class, an outage occurs during the change. The change is applied during the next maintenance window, unless `ApplyImmediately` is specified as `true` for this request.

Default: Uses existing setting

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – *Required*: UTF-8 string.

The DB instance identifier. This value is stored as a lowercase string.

Constraints:

- Must match the identifier of an existing DBInstance.
- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – UTF-8 string.

The name of the DB parameter group to apply to the DB instance. Changing this setting doesn't result in an outage. The parameter group name itself is changed immediately, but the actual parameter changes are not applied until you reboot the instance without failover. The db instance will NOT be rebooted automatically and the parameter changes will NOT be applied during the next maintenance window.

Default: Uses existing setting

Constraints: The DB parameter group must be in the same DB parameter group family as this DB instance.

- **DBPortNumber** (in the CLI: `--db-port-number`, or in Python: `db_port_number`) – Number (integer).

The port number on which the database accepts connections.

The value of the `DBPortNumber` parameter must not match any of the port values specified for options in the option group for the DB instance.

Your database will restart when you change the `DBPortNumber` value regardless of the value of the `ApplyImmediately` parameter.

Default: 8182

- **DBSecurityGroups** (in the CLI: `--db-security-groups`, or in Python: `db_security_groups`) – An array of `DBSecurityGroupName`— UTF-8 strings.

A list of DB security groups to authorize on this DB instance. Changing this setting doesn't result in an outage and the change is asynchronously applied as soon as possible.

Constraints:

- If supplied, must match existing `DBSecurityGroups`.
- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

The new DB subnet group for the DB instance. You can use this parameter to move your DB instance to a different VPC.

Changing the subnet group causes an outage during the change. The change is applied during the next maintenance window, unless you specify `true` for the `ApplyImmediately` parameter.

Constraints: If supplied, must match the name of an existing `DBSubnetGroup`.

Example: `mySubnetGroup`

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

A value that indicates whether the DB instance has deletion protection enabled. The database can't be deleted when deletion protection is enabled. By default, deletion protection is disabled. See [Deleting a DB Instance](#).

- **Domain** (in the CLI: `--domain`, or in Python: `domain`) – UTF-8 string.

Not supported.

- **DomainIAMRoleName** (in the CLI: `--domain-iam-role-name`, or in Python: `domain_iam_role_name`) – UTF-8 string.

Not supported

- **EnableIAMDatabaseAuthentication** (in the CLI: `--enable-iam-database-authentication`, or in Python: `enable_iam_database_authentication`) – Boolean.

True to enable mapping of Amazon Identity and Access Management (IAM) accounts to database accounts, and otherwise false.

You can enable IAM database authentication for the following database engines

Not applicable. Mapping Amazon IAM accounts to database accounts is managed by the DB cluster. For more information, see [the section called “ModifyDBCluster” \(p. 804\)](#).

Default: `false`

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

The version number of the database engine to upgrade to. Currently, setting this parameter has no effect. To upgrade your database engine to the most recent release, use the [the section called “ApplyPendingMaintenanceAction” \(p. 932\)](#) API.

- **Iops** (in the CLI: `--iops`, or in Python: `iops`) – Number (integer).

The new Provisioned IOPS (I/O operations per second) value for the instance.

Changing this setting doesn't result in an outage and the change is applied during the next maintenance window unless the `ApplyImmediately` parameter is set to `true` for this request.

Default: Uses existing setting

- **MonitoringInterval** (in the CLI: `--monitoring-interval`, or in Python: `monitoring_interval`) – Number (integer).

The interval, in seconds, between points when Enhanced Monitoring metrics are collected for the DB instance. To disable collecting Enhanced Monitoring metrics, specify 0. The default is 0.

If `MonitoringRoleArn` is specified, then you must also set `MonitoringInterval` to a value other than 0.

Valid Values: 0, 1, 5, 10, 15, 30, 60

- **MonitoringRoleArn** (in the CLI: `--monitoring-role-arn`, or in Python: `monitoring_role_arn`) – UTF-8 string.

The ARN for the IAM role that permits Neptune to send enhanced monitoring metrics to Amazon CloudWatch Logs. For example, `arn:aws:iam:123456789012:role/emaccess`.

If `MonitoringInterval` is set to a value other than 0, then you must supply a `MonitoringRoleArn` value.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies if the DB instance is a Multi-AZ deployment. Changing this parameter doesn't result in an outage and the change is applied during the next maintenance window unless the `ApplyImmediately` parameter is set to `true` for this request.

- **NewDBInstanceIdentifier** (in the CLI: `--new-db-instance-identifier`, or in Python: `new_db_instance_identifier`) – UTF-8 string.

The new DB instance identifier for the DB instance when renaming a DB instance. When you change the DB instance identifier, an instance reset will occur immediately if you set `Apply Immediately`

to true, or will occur during the next maintenance window if `Apply Immediately` to false. This value is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- The first character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `mydbinstance`

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

The daily time range during which automated backups are created if automated backups are enabled.

Not applicable. The daily time range for creating automated backups is managed by the DB cluster. For more information, see [the section called "ModifyDBCluster" \(p. 804\)](#).

Constraints:

- Must be in the format `hh24:mi-hh24:mi`
- Must be in Universal Time Coordinated (UTC)
- Must not conflict with the preferred maintenance window
- Must be at least 30 minutes
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

The weekly time range (in UTC) during which system maintenance can occur, which might result in an outage. Changing this parameter doesn't result in an outage, except in the following situation, and the change is asynchronously applied as soon as possible. If there are pending actions that cause a reboot, and the maintenance window is changed to include the current time, then changing this parameter will cause a reboot of the DB instance. If moving this window to the current time, there must be at least 30 minutes between the current time and end of the window to ensure pending changes are applied.

Default: Uses existing setting

Format: `ddd:hh24:mi-ddd:hh24:mi`

Valid Days: `Mon | Tue | Wed | Thu | Fri | Sat | Sun`

Constraints: Must be at least 30 minutes

- **PromotionTier** (in the CLI: `--promotion-tier`, or in Python: `promotion_tier`) – Number (integer).

A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

Default: 1

Valid Values: 0 - 15

- **PubliclyAccessible** (in the CLI: `--publicly-accessible`, or in Python: `publicly_accessible`) – Boolean.

This flag should no longer be used.

- **StorageType** (in the CLI: `--storage-type`, or in Python: `storage_type`) – UTF-8 string.

Not supported.

- **TdeCredentialArn** (in the CLI: `--tde-credential-arn`, or in Python: `tde_credential_arn`) – UTF-8 string.

The ARN from the key store with which to associate the instance for TDE encryption.

- **TdeCredentialPassword** (in the CLI: `--tde-credential-password`, or in Python: `tde_credential_password`) – UTF-8 string.

The password for the given ARN from the key store in order to access the device.

- **VpcSecurityGroupIds** (in the CLI: `--vpc-security-group-ids`, or in Python: `vpc_security_group_ids`) – An array of VpcSecurityGroupId— UTF-8 strings.

A list of EC2 VPC security groups to authorize on this DB instance. This change is asynchronously applied as soon as possible.

Not applicable. The associated list of EC2 VPC security groups is managed by the DB cluster. For more information, see [the section called “ModifyDBCluster” \(p. 804\)](#).

Constraints:

- If supplied, must match existing VpcSecurityGroupIds.

Response

Contains the details of an Amazon Neptune DB instance.

This data type is used as a response element in the [the section called “DescribeDBInstances” \(p. 859\)](#) action.

- **AutoMinorVersionUpgrade** (in the CLI: `--auto-minor-version-upgrade`, or in Python: `auto_minor_version_upgrade`) – Boolean.

Indicates that minor version patches are applied automatically.

- **AvailabilityZone** (in the CLI: `--availability-zone`, or in Python: `availability_zone`) – UTF-8 string.

Specifies the name of the Availability Zone the DB instance is located in.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **CACertificateIdentifier** (in the CLI: `--ca-certificate-identifier`, or in Python: `ca_certificate_identifier`) – UTF-8 string.

The identifier of the CA certificate for this DB instance.

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

Specifies whether tags are copied from the DB instance to snapshots of the DB instance.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

If the DB instance is a member of a DB cluster, contains the name of the DB cluster that the DB instance is a member of.

- **DBInstanceArn** (in the CLI: `--db-instance-arn`, or in Python: `db_instance_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB instance.

- **DBInstanceClass** (in the CLI: `--db-instance-class`, or in Python: `db_instance_class`) – UTF-8 string.

Contains the name of the compute and memory capacity class of the DB instance.

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – UTF-8 string.

Contains a user-supplied database identifier. This identifier is the unique key that identifies a DB instance.

- **DBInstancePort** (in the CLI: `--db-instance-port`, or in Python: `db_instance_port`) – Number (integer).

Specifies the port that the DB instance listens on. If the DB instance is part of a DB cluster, this can be a different port than the DB cluster port.

- **DBInstanceStatus** (in the CLI: `--db-instance-status`, or in Python: `db_instance_status`) – UTF-8 string.

Specifies the current state of this database.

- **DbiResourceId** (in the CLI: `--dbi-resource-id`, or in Python: `dbi_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB instance. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB instance is accessed.

- **DBName** (in the CLI: `--db-name`, or in Python: `db_name`) – UTF-8 string.

The database name.

- **DBParameterGroups** (in the CLI: `--db-parameter-groups`, or in Python: `db_parameter_groups`) – An array of [DBParameterGroupStatus](#) (p. 864) [DBParameterGroup](#) objects.

Provides the list of DB parameter groups applied to this DB instance.

- **DBSecurityGroups** (in the CLI: `--db-security-groups`, or in Python: `db_security_groups`) – An array of [DBSecurityGroupMembership](#) (p. 935) [DBSecurityGroup](#) objects.

Provides List of DB security group elements containing only `DBSecurityGroup.Name` and `DBSecurityGroup.Status` subelements.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – A [DBSubnetGroup](#) (p. 882) object.

Specifies information on the subnet group associated with the DB instance, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB instance has deletion protection enabled. The instance can't be deleted when deletion protection is enabled. See [Deleting a DB Instance](#).

- **DomainMemberships** (in the CLI: `--domain-memberships`, or in Python: `domain_memberships`) – An array of [DomainMembership](#) (p. 936) [DomainMembership](#) objects.

Not supported

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB instance is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – An [Endpoint](#) (p. 936) object.

Specifies the connection endpoint.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB instance.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.
Indicates the database engine version.
- **EnhancedMonitoringResourceArn** (in the CLI: `--enhanced-monitoring-resource-arn`, or in Python: `enhanced_monitoring_resource_arn`) – UTF-8 string.
The Amazon Resource Name (ARN) of the Amazon CloudWatch Logs log stream that receives the Enhanced Monitoring metrics data for the DB instance.
- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.
True if Amazon Identity and Access Management (IAM) authentication is enabled, and otherwise false.
- **InstanceCreateTime** (in the CLI: `--instance-create-time`, or in Python: `instance_create_time`) – Timestamp.
Provides the date and time the DB instance was created.
- **Iops** (in the CLI: `--iops`, or in Python: `iops`) – Number (integer).
Specifies the Provisioned IOPS (I/O operations per second) value.
- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.
Not supported: The encryption for DB instances is managed by the DB cluster.
- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.
Specifies the latest time to which a database can be restored with point-in-time restore.
- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.
License model information for this DB instance.
- **MonitoringInterval** (in the CLI: `--monitoring-interval`, or in Python: `monitoring_interval`) – Number (integer).
The interval, in seconds, between points when Enhanced Monitoring metrics are collected for the DB instance.
- **MonitoringRoleArn** (in the CLI: `--monitoring-role-arn`, or in Python: `monitoring_role_arn`) – UTF-8 string.
The ARN for the IAM role that permits Neptune to send Enhanced Monitoring metrics to Amazon CloudWatch Logs.
- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.
Specifies if the DB instance is a Multi-AZ deployment.
- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [PendingModifiedValues](#) (p. 928) object.
Specifies that changes to the DB instance are pending. This element is only included when changes are pending. Specific changes are identified by subelements.
- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.
Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **PromotionTier** (in the CLI: `--promotion-tier`, or in Python: `promotion_tier`) – Number (integer).

A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

- **PubliclyAccessible** (in the CLI: `--publicly-accessible`, or in Python: `publicly_accessible`) – Boolean.

This flag should no longer be used.

- **ReadReplicaDBClusterIdentifiers** (in the CLI: `--read-replica-db-cluster-identifiers`, or in Python: `read_replica_db_cluster_identifiers`) – An array of `ReadReplicaDBClusterIdentifier`— UTF-8 strings.

Contains one or more identifiers of DB clusters that are Read Replicas of this DB instance.

- **ReadReplicaDBInstanceIdentifiers** (in the CLI: `--read-replica-db-instance-identifiers`, or in Python: `read_replica_db_instance_identifiers`) – An array of `ReadReplicaDBInstanceIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB instance.

- **ReadReplicaSourceDBInstanceIdentifier** (in the CLI: `--read-replica-source-db-instance-identifier`, or in Python: `read_replica_source_db_instance_identifier`) – UTF-8 string.

Contains the identifier of the source DB instance if this DB instance is a Read Replica.

- **SecondaryAvailabilityZone** (in the CLI: `--secondary-availability-zone`, or in Python: `secondary_availability_zone`) – UTF-8 string.

If present, specifies the name of the secondary Availability Zone for a DB instance with multi-AZ support.

- **StatusInfos** (in the CLI: `--status-infos`, or in Python: `status_infos`) – An array of [DBInstanceStatusInfo](#) (p. 831) `DBInstanceStatusInfo` objects.

The status of a Read Replica. If the instance is not a Read Replica, this is blank.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **StorageType** (in the CLI: `--storage-type`, or in Python: `storage_type`) – UTF-8 string.

Specifies the storage type associated with DB instance.

- **TdeCredentialArn** (in the CLI: `--tde-credential-arn`, or in Python: `tde_credential_arn`) – UTF-8 string.

The ARN from the key store with which the instance is associated for TDE encryption.

- **Timezone** (in the CLI: `--timezone`, or in Python: `timezone`) – UTF-8 string.

Not supported.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security group elements that the DB instance belongs to.

Errors

- [InvalidDBInstanceStateFault](#) (p. 948)
- [InvalidDBSecurityGroupStateFault](#) (p. 949)
- [DBInstanceAlreadyExistsFault](#) (p. 942)
- [DBInstanceNotFoundFault](#) (p. 942)
- [DBSecurityGroupNotFoundFault](#) (p. 944)
- [DBParameterGroupNotFoundFault](#) (p. 943)
- [InsufficientDBInstanceCapacityFault](#) (p. 947)
- [StorageQuotaExceededFault](#) (p. 953)
- [InvalidVPCNetworkStateFault](#) (p. 951)
- [ProvisionedIopsNotAvailableInAZFault](#) (p. 952)
- [OptionGroupNotFoundFault](#) (p. 951)
- [DBUpgradeDependencyFailureFault](#) (p. 946)
- [StorageTypeNotSupportedFault](#) (p. 954)
- [AuthorizationNotFoundFault](#) (p. 940)
- [CertificateNotFoundFault](#) (p. 940)
- [DomainNotFoundFault](#) (p. 946)

RebootDBInstance (action)

The Amazon CLI name for this API is: `reboot-db-instance`, and the Python name is: `reboot_db_instance`.

You might need to reboot your DB instance, usually for maintenance reasons. For example, if you make certain modifications, or if you change the DB parameter group associated with the DB instance, you must reboot the instance for the changes to take effect.

Rebooting a DB instance restarts the database engine service. Rebooting a DB instance results in a momentary outage, during which the DB instance status is set to rebooting.

Request

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – *Required*: UTF-8 string.

The DB instance identifier. This parameter is stored as a lowercase string.

Constraints:

- Must match the identifier of an existing DBInstance.
- **ForceFailover** (in the CLI: `--force-failover`, or in Python: `force_failover`) – Boolean.

When `true`, the reboot is conducted through a MultiAZ failover.

Constraint: You can't specify `true` if the instance is not configured for MultiAZ.

Response

Contains the details of an Amazon Neptune DB instance.

This data type is used as a response element in the [the section called “DescribeDBInstances”](#) (p. 859) action.

- **AutoMinorVersionUpgrade** (in the CLI: `--auto-minor-version-upgrade`, or in Python: `auto_minor_version_upgrade`) – Boolean.

Indicates that minor version patches are applied automatically.

- **AvailabilityZone** (in the CLI: `--availability-zone`, or in Python: `availability_zone`) – UTF-8 string.

Specifies the name of the Availability Zone the DB instance is located in.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **CACertificateIdentifier** (in the CLI: `--ca-certificate-identifier`, or in Python: `ca_certificate_identifier`) – UTF-8 string.

The identifier of the CA certificate for this DB instance.

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

Specifies whether tags are copied from the DB instance to snapshots of the DB instance.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

If the DB instance is a member of a DB cluster, contains the name of the DB cluster that the DB instance is a member of.

- **DBInstanceArn** (in the CLI: `--db-instance-arn`, or in Python: `db_instance_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB instance.

- **DBInstanceClass** (in the CLI: `--db-instance-class`, or in Python: `db_instance_class`) – UTF-8 string.

Contains the name of the compute and memory capacity class of the DB instance.

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – UTF-8 string.

Contains a user-supplied database identifier. This identifier is the unique key that identifies a DB instance.

- **DBInstancePort** (in the CLI: `--db-instance-port`, or in Python: `db_instance_port`) – Number (integer).

Specifies the port that the DB instance listens on. If the DB instance is part of a DB cluster, this can be a different port than the DB cluster port.

- **DBInstanceStatus** (in the CLI: `--db-instance-status`, or in Python: `db_instance_status`) – UTF-8 string.

Specifies the current state of this database.

- **DbiResourceId** (in the CLI: `--dbi-resource-id`, or in Python: `dbi_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB instance. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB instance is accessed.

- **DBName** (in the CLI: `--db-name`, or in Python: `db_name`) – UTF-8 string.

The database name.

- **DBParameterGroups** (in the CLI: `--db-parameter-groups`, or in Python: `db_parameter_groups`) – An array of [DBParameterGroupStatus](#) (p. 864) `DBParameterGroup` objects.

Provides the list of DB parameter groups applied to this DB instance.

- **DBSecurityGroups** (in the CLI: `--db-security-groups`, or in Python: `db_security_groups`) – An array of [DBSecurityGroupMembership \(p. 935\)](#) `DBSecurityGroup` objects.

Provides List of DB security group elements containing only `DBSecurityGroup.Name` and `DBSecurityGroup.Status` subelements.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – A [DBSubnetGroup \(p. 882\)](#) object.

Specifies information on the subnet group associated with the DB instance, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB instance has deletion protection enabled. The instance can't be deleted when deletion protection is enabled. See [Deleting a DB Instance](#).

- **DomainMemberships** (in the CLI: `--domain-memberships`, or in Python: `domain_memberships`) – An array of [DomainMembership \(p. 936\)](#) `DomainMembership` objects.

Not supported

- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB instance is configured to export to CloudWatch Logs.

- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – An [Endpoint \(p. 936\)](#) object.

Specifies the connection endpoint.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB instance.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.

- **EnhancedMonitoringResourceArn** (in the CLI: `--enhanced-monitoring-resource-arn`, or in Python: `enhanced_monitoring_resource_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) of the Amazon CloudWatch Logs log stream that receives the Enhanced Monitoring metrics data for the DB instance.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if Amazon Identity and Access Management (IAM) authentication is enabled, and otherwise false.

- **InstanceCreateTime** (in the CLI: `--instance-create-time`, or in Python: `instance_create_time`) – Timestamp.

Provides the date and time the DB instance was created.

- **Iops** (in the CLI: `--iops`, or in Python: `iops`) – Number (integer).

Specifies the Provisioned IOPS (I/O operations per second) value.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.

- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.

License model information for this DB instance.

- **MonitoringInterval** (in the CLI: `--monitoring-interval`, or in Python: `monitoring_interval`) – Number (integer).

The interval, in seconds, between points when Enhanced Monitoring metrics are collected for the DB instance.

- **MonitoringRoleArn** (in the CLI: `--monitoring-role-arn`, or in Python: `monitoring_role_arn`) – UTF-8 string.

The ARN for the IAM role that permits Neptune to send Enhanced Monitoring metrics to Amazon CloudWatch Logs.

- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies if the DB instance is a Multi-AZ deployment.

- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [PendingModifiedValues](#) (p. 928) object.

Specifies that changes to the DB instance are pending. This element is only included when changes are pending. Specific changes are identified by subelements.

- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).

- **PromotionTier** (in the CLI: `--promotion-tier`, or in Python: `promotion_tier`) – Number (integer).

A value that specifies the order in which a Read Replica is promoted to the primary instance after a failure of the existing primary instance.

- **PubliclyAccessible** (in the CLI: `--publicly-accessible`, or in Python: `publicly_accessible`) – Boolean.

This flag should no longer be used.

- **ReadReplicaDBClusterIdentifiers** (in the CLI: `--read-replica-db-cluster-identifiers`, or in Python: `read_replica_db_cluster_identifiers`) – An array of `ReadReplicaDBClusterIdentifier`— UTF-8 strings.

Contains one or more identifiers of DB clusters that are Read Replicas of this DB instance.

- **ReadReplicaDBInstanceIdentifiers** (in the CLI: `--read-replica-db-instance-identifiers`, or in Python: `read_replica_db_instance_identifiers`) – An array of `ReadReplicaDBInstanceIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB instance.

- **ReadReplicaSourceDBInstanceIdentifier** (in the CLI: `--read-replica-source-db-instance-identifier`, or in Python: `read_replica_source_db_instance_identifier`) – UTF-8 string.

Contains the identifier of the source DB instance if this DB instance is a Read Replica.

- **SecondaryAvailabilityZone** (in the CLI: `--secondary-availability-zone`, or in Python: `secondary_availability_zone`) – UTF-8 string.

If present, specifies the name of the secondary Availability Zone for a DB instance with multi-AZ support.

- **StatusInfos** (in the CLI: `--status-infos`, or in Python: `status_infos`) – An array of [DBInstanceStatusInfo](#) (p. 831) `DBInstanceStatusInfo` objects.

The status of a Read Replica. If the instance is not a Read Replica, this is blank.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Not supported: The encryption for DB instances is managed by the DB cluster.

- **StorageType** (in the CLI: `--storage-type`, or in Python: `storage_type`) – UTF-8 string.

Specifies the storage type associated with DB instance.

- **TdeCredentialArn** (in the CLI: `--tde-credential-arn`, or in Python: `tde_credential_arn`) – UTF-8 string.

The ARN from the key store with which the instance is associated for TDE encryption.

- **Timezone** (in the CLI: `--timezone`, or in Python: `timezone`) – UTF-8 string.

Not supported.

- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security group elements that the DB instance belongs to.

Errors

- [InvalidDBInstanceStateFault](#) (p. 948)
- [DBInstanceNotFoundFault](#) (p. 942)

DescribeDBInstances (action)

The Amazon CLI name for this API is: `describe-db-instances`, and the Python name is: `describe_db_instances`.

Returns information about provisioned instances, and supports pagination.

Note

This operation can also return information for Amazon RDS instances and Amazon DocDB instances.

Request

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – UTF-8 string.

The user-supplied instance identifier. If this parameter is specified, information from only the specific DB instance is returned. This parameter isn't case-sensitive.

Constraints:

- If supplied, must match the identifier of an existing `DBInstance`.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter](#) (p. 937) `Filter` objects.

A filter that specifies one or more DB instances to describe.

Supported filters:

- `db-cluster-id` - Accepts DB cluster identifiers and DB cluster Amazon Resource Names (ARNs). The results list will only include information about the DB instances associated with the DB clusters identified by these ARNs.
- `engine` - Accepts an engine name (such as `neptune`), and restricts the results list to DB instances created by that engine.

For example, to invoke this API from the Amazon CLI and filter so that only Neptune DB instances are returned, you could use the following command:

Example

```
aws neptune describe-db-instances \
    --filters Name=engine,Values=neptune
```

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBInstances` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

- **DBInstances** (in the CLI: `--db-instances`, or in Python: `db_instances`) – An array of [DBInstance](#) (p. 828) DBInstance objects.

A list of [the section called "DBInstance" \(p. 828\)](#) instances.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Errors

- [DBInstanceNotFoundFault](#) (p. 942)

DescribeOrderableDBInstanceOptions (action)

The Amazon CLI name for this API is: `describe-orderable-db-instance-options`, and the Python name is: `describe_orderable_db_instance_options`.

Returns a list of orderable DB instance options for the specified engine.

Request

- **DBInstanceClass** (in the CLI: `--db-instance-class`, or in Python: `db_instance_class`) – UTF-8 string.

The DB instance class filter value. Specify this parameter to show only the available offerings matching the specified DB instance class.
- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – *Required*: UTF-8 string.

The name of the engine to retrieve DB instance options for.
- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

The engine version filter value. Specify this parameter to show only the available offerings matching the specified engine version.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.
- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.

The license model filter value. Specify this parameter to show only the available offerings matching the specified license model.
- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeOrderableDBInstanceOptions` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.
- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.
- **Vpc** (in the CLI: `--vpc`, or in Python: `vpc`) – Boolean.

The VPC filter value. Specify this parameter to show only the available VPC or non-VPC offerings.

Response

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `OrderableDBInstanceOptions` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.
- **OrderableDBInstanceOptions** (in the CLI: `--orderable-db-instance-options`, or in Python: `orderable_db_instance_options`) – An array of [OrderableDBInstanceOption \(p. 831\)](#) `OrderableDBInstanceOption` objects.

An [the section called "OrderableDBInstanceOption" \(p. 831\)](#) structure containing information about orderable options for the DB instance.

DescribeValidDBInstanceModifications (action)

The Amazon CLI name for this API is: `describe-valid-db-instance-modifications`, and the Python name is: `describe_valid_db_instance_modifications`.

You can call [the section called “DescribeValidDBInstanceModifications” \(p. 861\)](#) to learn what modifications you can make to your DB instance. You can use this information when you call [the section called “ModifyDBInstance” \(p. 846\)](#).

Request

- **DBInstanceIdentifier** (in the CLI: `--db-instance-identifier`, or in Python: `db_instance_identifier`) – *Required:* UTF-8 string.

The customer identifier or the ARN of your DB instance.

Response

Information about valid modifications that you can make to your DB instance. Contains the result of a successful call to the [the section called “DescribeValidDBInstanceModifications” \(p. 861\)](#) action. You can use this information when you call [the section called “ModifyDBInstance” \(p. 846\)](#).

- **Storage** (in the CLI: `--storage`, or in Python: `storage`) – An array of `ValidStorageOptions`— a structures.

Valid storage options for your DB instance.

Errors

- [DBInstanceNotFoundFault \(p. 942\)](#)
- [InvalidDBInstanceStateFault \(p. 948\)](#)

Neptune Parameters API

Structures:

- [Parameter \(structure\) \(p. 863\)](#)
- [DBParameterGroup \(structure\) \(p. 863\)](#)
- [DBClusterParameterGroup \(structure\) \(p. 864\)](#)
- [DBParameterGroupStatus \(structure\) \(p. 864\)](#)

Actions:

- [CopyDBParameterGroup \(action\) \(p. 865\)](#)
- [CopyDBClusterParameterGroup \(action\) \(p. 866\)](#)
- [CreateDBParameterGroup \(action\) \(p. 867\)](#)
- [CreateDBClusterParameterGroup \(action\) \(p. 869\)](#)
- [DeleteDBParameterGroup \(action\) \(p. 870\)](#)
- [DeleteDBClusterParameterGroup \(action\) \(p. 871\)](#)
- [ModifyDBParameterGroup \(action\) \(p. 871\)](#)
- [ModifyDBClusterParameterGroup \(action\) \(p. 872\)](#)
- [ResetDBParameterGroup \(action\) \(p. 873\)](#)
- [ResetDBClusterParameterGroup \(action\) \(p. 874\)](#)
- [DescribeDBParameters \(action\) \(p. 875\)](#)
- [DescribeDBParameterGroups \(action\) \(p. 876\)](#)
- [DescribeDBClusterParameters \(action\) \(p. 877\)](#)

- [DescribeDBClusterParameterGroups \(action\) \(p. 878\)](#)
- [DescribeEngineDefaultParameters \(action\) \(p. 879\)](#)
- [DescribeEngineDefaultClusterParameters \(action\) \(p. 880\)](#)

Parameter (structure)

Specifies a parameter.

Fields

- **AllowedValues** – UTF-8 string.
Specifies the valid range of values for the parameter.
- **ApplyMethod** – UTF-8 string (valid values: `immediate=""` | `pending-reboot=""`).
Indicates when to apply parameter updates.
- **ApplyType** – UTF-8 string.
Specifies the engine specific parameters type.
- **DataType** – UTF-8 string.
Specifies the valid data type for the parameter.
- **Description** – UTF-8 string.
Provides a description of the parameter.
- **IsModifiable** – Boolean.
Indicates whether (`true`) or not (`false`) the parameter can be modified. Some parameters have security or operational implications that prevent them from being changed.
- **MinimumEngineVersion** – UTF-8 string.
The earliest engine version to which the parameter can apply.
- **ParameterName** – UTF-8 string.
Specifies the name of the parameter.
- **ParameterValue** – UTF-8 string.
Specifies the value of the parameter.
- **Source** – UTF-8 string.
Indicates the source of the parameter value.

DBParameterGroup (structure)

Contains the details of an Amazon Neptune DB parameter group.

This data type is used as a response element in the [the section called "DescribeDBParameterGroups" \(p. 876\)](#) action.

Fields

- **DBParameterGroupArn** – UTF-8 string.
The Amazon Resource Name (ARN) for the DB parameter group.

- **DBParameterGroupFamily** – UTF-8 string.
Provides the name of the DB parameter group family that this DB parameter group is compatible with.
- **DBParameterGroupName** – UTF-8 string.
Provides the name of the DB parameter group.
- **Description** – UTF-8 string.
Provides the customer-specified description for this DB parameter group.

`DBParameterGroup` is used as the response element for:

- [CopyDBParameterGroup](#) (p. 865)
- [CreateDBParameterGroup](#) (p. 867)

DBClusterParameterGroup (structure)

Contains the details of an Amazon Neptune DB cluster parameter group.

This data type is used as a response element in the [the section called “DescribeDBClusterParameterGroups”](#) (p. 878) action.

Fields

- **DBClusterParameterGroupArn** – UTF-8 string.
The Amazon Resource Name (ARN) for the DB cluster parameter group.
- **DBClusterParameterGroupName** – UTF-8 string.
Provides the name of the DB cluster parameter group.
- **DBParameterGroupFamily** – UTF-8 string.
Provides the name of the DB parameter group family that this DB cluster parameter group is compatible with.
- **Description** – UTF-8 string.
Provides the customer-specified description for this DB cluster parameter group.

`DBClusterParameterGroup` is used as the response element for:

- [CopyDBClusterParameterGroup](#) (p. 866)
- [CreateDBClusterParameterGroup](#) (p. 869)

DBParameterGroupStatus (structure)

The status of the DB parameter group.

This data type is used as a response element in the following actions:

- [the section called “CreateDBInstance”](#) (p. 833)
- [the section called “DeleteDBInstance”](#) (p. 841)
- [the section called “ModifyDBInstance”](#) (p. 846)

- [the section called “RebootDBInstance” \(p. 855\)](#)

Fields

- **DBParameterGroupName** – UTF-8 string.

The name of the DP parameter group.

- **ParameterApplyStatus** – UTF-8 string.

The status of parameter updates.

CopyDBParameterGroup (action)

The Amazon CLI name for this API is: `copy-db-parameter-group`, and the Python name is: `copy_db_parameter_group`.

Copies the specified DB parameter group.

Request

- **SourceDBParameterGroupIdentifier** (in the CLI: `--source-db-parameter-group-identifier`, or in Python: `source_db_parameter_group_identifier`) – *Required:* UTF-8 string.

The identifier or ARN for the source DB parameter group. For information about creating an ARN, see [Constructing an Amazon Resource Name \(ARN\)](#).

Constraints:

- Must specify a valid DB parameter group.
- Must specify a valid DB parameter group identifier, for example `my-db-param-group`, or a valid ARN.
- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be assigned to the copied DB parameter group.

- **TargetDBParameterGroupDescription** (in the CLI: `--target-db-parameter-group-description`, or in Python: `target_db_parameter_group_description`) – *Required:* UTF-8 string.

A description for the copied DB parameter group.

- **TargetDBParameterGroupIdentifier** (in the CLI: `--target-db-parameter-group-identifier`, or in Python: `target_db_parameter_group_identifier`) – *Required:* UTF-8 string.

The identifier for the copied DB parameter group.

Constraints:

- Cannot be null, empty, or blank.
- Must contain from 1 to 255 letters, numbers, or hyphens.
- First character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-db-parameter-group`

Response

Contains the details of an Amazon Neptune DB parameter group.

This data type is used as a response element in the [the section called “DescribeDBParameterGroups” \(p. 876\)](#) action.

- **DBParameterGroupArn** (in the CLI: `--db-parameter-group-arn`, or in Python: `db_parameter_group_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB parameter group.

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – UTF-8 string.

Provides the name of the DB parameter group family that this DB parameter group is compatible with.

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – UTF-8 string.

Provides the name of the DB parameter group.

- **Description** (in the CLI: `--description`, or in Python: `description`) – UTF-8 string.

Provides the customer-specified description for this DB parameter group.

Errors

- [DBParameterGroupNotFoundFault \(p. 943\)](#)
- [DBParameterGroupAlreadyExistsFault \(p. 943\)](#)
- [DBParameterGroupQuotaExceededFault \(p. 943\)](#)

CopyDBClusterParameterGroup (action)

The Amazon CLI name for this API is: `copy-db-cluster-parameter-group`, and the Python name is: `copy_db_cluster_parameter_group`.

Copies the specified DB cluster parameter group.

Request

- **SourceDBClusterParameterGroupIdentifier** (in the CLI: `--source-db-cluster-parameter-group-identifier`, or in Python: `source_db_cluster_parameter_group_identifier`) – *Required:* UTF-8 string.

The identifier or Amazon Resource Name (ARN) for the source DB cluster parameter group. For information about creating an ARN, see [Constructing an Amazon Resource Name \(ARN\)](#).

Constraints:

- Must specify a valid DB cluster parameter group.
- If the source DB cluster parameter group is in the same Amazon Region as the copy, specify a valid DB parameter group identifier, for example `my-db-cluster-param-group`, or a valid ARN.
- If the source DB parameter group is in a different Amazon Region than the copy, specify a valid DB cluster parameter group ARN, for example `arn:aws:rds:us-east-1:123456789012:cluster-pg:custom-cluster-group1`.
- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be assigned to the copied DB cluster parameter group.

- **TargetDBClusterParameterGroupDescription** (in the CLI: `--target-db-cluster-parameter-group-description`, or in Python: `target_db_cluster_parameter_group_description`) – *Required:* UTF-8 string.

A description for the copied DB cluster parameter group.

- **TargetDBClusterParameterGroupIdentifier** (in the CLI: `--target-db-cluster-parameter-group-identifier`, or in Python: `target_db_cluster_parameter_group_identifier`) – *Required:* UTF-8 string.

The identifier for the copied DB cluster parameter group.

Constraints:

- Cannot be null, empty, or blank
- Must contain from 1 to 255 letters, numbers, or hyphens
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens

Example: `my-cluster-param-group1`

Response

Contains the details of an Amazon Neptune DB cluster parameter group.

This data type is used as a response element in the [the section called “DescribeDBClusterParameterGroups” \(p. 878\)](#) action.

- **DBClusterParameterGroupArn** (in the CLI: `--db-cluster-parameter-group-arn`, or in Python: `db_cluster_parameter_group_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster parameter group.

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

Provides the name of the DB cluster parameter group.

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – UTF-8 string.

Provides the name of the DB parameter group family that this DB cluster parameter group is compatible with.

- **Description** (in the CLI: `--description`, or in Python: `description`) – UTF-8 string.

Provides the customer-specified description for this DB cluster parameter group.

Errors

- [DBParameterGroupNotFoundFault \(p. 943\)](#)
- [DBParameterGroupQuotaExceededFault \(p. 943\)](#)
- [DBParameterGroupAlreadyExistsFault \(p. 943\)](#)

CreateDBParameterGroup (action)

The Amazon CLI name for this API is: `create-db-parameter-group`, and the Python name is: `create_db_parameter_group`.

Creates a new DB parameter group.

A DB parameter group is initially created with the default parameters for the database engine used by the DB instance. To provide custom values for any of the parameters, you must modify the group after

creating it using *ModifyDBParameterGroup*. Once you've created a DB parameter group, you need to associate it with your DB instance using *ModifyDBInstance*. When you associate a new DB parameter group with a running DB instance, you need to reboot the DB instance without failover for the new DB parameter group and associated settings to take effect.

Important

After you create a DB parameter group, you should wait at least 5 minutes before creating your first DB instance that uses that DB parameter group as the default parameter group. This allows Amazon Neptune to fully complete the create action before the parameter group is used as the default for a new DB instance. This is especially important for parameters that are critical when creating the default database for a DB instance, such as the character set for the default database defined by the `character_set_database` parameter. You can use the *Parameter Groups* option of the Amazon Neptune console or the *DescribeDBParameters* command to verify that your DB parameter group has been created or modified.

Request

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – *Required:* UTF-8 string.

The DB parameter group family name. A DB parameter group can be associated with one and only one DB parameter group family, and can be applied only to a DB instance running a database engine and engine version compatible with that DB parameter group family.

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – *Required:* UTF-8 string.

The name of the DB parameter group.

Constraints:

- Must be 1 to 255 letters, numbers, or hyphens.
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens

Note

This value is stored as a lowercase string.

- **Description** (in the CLI: `--description`, or in Python: `description`) – *Required:* UTF-8 string.

The description for the DB parameter group.

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be assigned to the new DB parameter group.

Response

Contains the details of an Amazon Neptune DB parameter group.

This data type is used as a response element in the [the section called "DescribeDBParameterGroups" \(p. 876\)](#) action.

- **DBParameterGroupArn** (in the CLI: `--db-parameter-group-arn`, or in Python: `db_parameter_group_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB parameter group.

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – UTF-8 string.

Provides the name of the DB parameter group family that this DB parameter group is compatible with.

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – UTF-8 string.

Provides the name of the DB parameter group.

- **Description** (in the CLI: `--description`, or in Python: `description`) – UTF-8 string.

Provides the customer-specified description for this DB parameter group.

Errors

- [DBParameterGroupQuotaExceededFault](#) (p. 943)
- [DBParameterGroupAlreadyExistsFault](#) (p. 943)

CreateDBClusterParameterGroup (action)

The Amazon CLI name for this API is: `create-db-cluster-parameter-group`, and the Python name is: `create_db_cluster_parameter_group`.

Creates a new DB cluster parameter group.

Parameters in a DB cluster parameter group apply to all of the instances in a DB cluster.

A DB cluster parameter group is initially created with the default parameters for the database engine used by instances in the DB cluster. To provide custom values for any of the parameters, you must modify the group after creating it using [the section called “ModifyDBClusterParameterGroup”](#) (p. 872). Once you've created a DB cluster parameter group, you need to associate it with your DB cluster using [the section called “ModifyDBCluster”](#) (p. 804). When you associate a new DB cluster parameter group with a running DB cluster, you need to reboot the DB instances in the DB cluster without failover for the new DB cluster parameter group and associated settings to take effect.

Important

After you create a DB cluster parameter group, you should wait at least 5 minutes before creating your first DB cluster that uses that DB cluster parameter group as the default parameter group. This allows Amazon Neptune to fully complete the create action before the DB cluster parameter group is used as the default for a new DB cluster. This is especially important for parameters that are critical when creating the default database for a DB cluster, such as the character set for the default database defined by the `character_set_database` parameter. You can use the *Parameter Groups* option of the [Amazon Neptune console](#) or the [the section called “DescribeDBClusterParameters”](#) (p. 877) command to verify that your DB cluster parameter group has been created or modified.

Request

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – *Required*: UTF-8 string.

The name of the DB cluster parameter group.

Constraints:

- Must match the name of an existing `DBClusterParameterGroup`.

Note

This value is stored as a lowercase string.

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – *Required*: UTF-8 string.

The DB cluster parameter group family name. A DB cluster parameter group can be associated with one and only one DB cluster parameter group family, and can be applied only to a DB cluster running a database engine and engine version compatible with that DB cluster parameter group family.

- **Description** (in the CLI: `--description`, or in Python: `description`) – *Required*: UTF-8 string.

The description for the DB cluster parameter group.

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be assigned to the new DB cluster parameter group.

Response

Contains the details of an Amazon Neptune DB cluster parameter group.

This data type is used as a response element in the [the section called “DescribeDBClusterParameterGroups” \(p. 878\)](#) action.

- **DBClusterParameterGroupArn** (in the CLI: `--db-cluster-parameter-group-arn`, or in Python: `db_cluster_parameter_group_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster parameter group.

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

Provides the name of the DB cluster parameter group.

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – UTF-8 string.

Provides the name of the DB parameter group family that this DB cluster parameter group is compatible with.

- **Description** (in the CLI: `--description`, or in Python: `description`) – UTF-8 string.

Provides the customer-specified description for this DB cluster parameter group.

Errors

- [DBParameterGroupQuotaExceededFault \(p. 943\)](#)
- [DBParameterGroupAlreadyExistsFault \(p. 943\)](#)

DeleteDBParameterGroup (action)

The Amazon CLI name for this API is: `delete-db-parameter-group`, and the Python name is: `delete_db_parameter_group`.

Deletes a specified DBParameterGroup. The DBParameterGroup to be deleted can't be associated with any DB instances.

Request

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – *Required*: UTF-8 string.

The name of the DB parameter group.

Constraints:

- Must be the name of an existing DB parameter group
- You can't delete a default DB parameter group
- Cannot be associated with any DB instances

Response

- *No Response parameters.*

Errors

- [InvalidDBParameterGroupStateFault](#) (p. 948)
- [DBParameterGroupNotFoundFault](#) (p. 943)

DeleteDBClusterParameterGroup (action)

The Amazon CLI name for this API is: `delete-db-cluster-parameter-group`, and the Python name is: `delete_db_cluster_parameter_group`.

Deletes a specified DB cluster parameter group. The DB cluster parameter group to be deleted can't be associated with any DB clusters.

Request

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – *Required:* UTF-8 string.

The name of the DB cluster parameter group.

Constraints:

- Must be the name of an existing DB cluster parameter group.
- You can't delete a default DB cluster parameter group.
- Cannot be associated with any DB clusters.

Response

- *No Response parameters.*

Errors

- [InvalidDBParameterGroupStateFault](#) (p. 948)
- [DBParameterGroupNotFoundFault](#) (p. 943)

ModifyDBParameterGroup (action)

The Amazon CLI name for this API is: `modify-db-parameter-group`, and the Python name is: `modify_db_parameter_group`.

Modifies the parameters of a DB parameter group. To modify more than one parameter, submit a list of the following: `ParameterName`, `ParameterValue`, and `ApplyMethod`. A maximum of 20 parameters can be modified in a single request.

Note

Changes to dynamic parameters are applied immediately. Changes to static parameters require a reboot without failover to the DB instance associated with the parameter group before the change can take effect.

Important

After you modify a DB parameter group, you should wait at least 5 minutes before creating your first DB instance that uses that DB parameter group as the default parameter group. This allows Amazon Neptune to fully complete the modify action before the parameter group is used as the default for a new DB instance. This is especially important for parameters that are critical when creating the default database for a DB instance, such as the character set for the default database defined by the `character_set_database` parameter. You can use the *Parameter Groups* option of the Amazon Neptune console or the *DescribeDBParameters* command to verify that your DB parameter group has been created or modified.

Request

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – *Required:* UTF-8 string.

The name of the DB parameter group.

Constraints:

- If supplied, must match the name of an existing DBParameterGroup.
- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – *Required:* An array of [Parameter](#) (p. 863) Parameter objects.

An array of parameter names, values, and the apply method for the parameter update. At least one parameter name, value, and apply method must be supplied; subsequent arguments are optional. A maximum of 20 parameters can be modified in a single request.

Valid Values (for the application method): `immediate` | `pending-reboot`

Note

You can use the `immediate` value with dynamic parameters only. You can use the `pending-reboot` value for both dynamic and static parameters, and changes are applied when you reboot the DB instance without failover.

Response

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – UTF-8 string.

Provides the name of the DB parameter group.

Errors

- [DBParameterGroupNotFoundFault](#) (p. 943)
- [InvalidDBParameterGroupStateFault](#) (p. 948)

ModifyDBClusterParameterGroup (action)

The Amazon CLI name for this API is: `modify-db-cluster-parameter-group`, and the Python name is: `modify_db_cluster_parameter_group`.

Modifies the parameters of a DB cluster parameter group. To modify more than one parameter, submit a list of the following: `ParameterName`, `ParameterValue`, and `ApplyMethod`. A maximum of 20 parameters can be modified in a single request.

Note

Changes to dynamic parameters are applied immediately. Changes to static parameters require a reboot without failover to the DB cluster associated with the parameter group before the change can take effect.

Important

After you create a DB cluster parameter group, you should wait at least 5 minutes before creating your first DB cluster that uses that DB cluster parameter group as the default parameter group. This allows Amazon Neptune to fully complete the create action before the parameter group is used as the default for a new DB cluster. This is especially important for parameters that are critical when creating the default database for a DB cluster, such as the character set for the default database defined by the `character_set_database` parameter. You can use the *Parameter Groups* option of the Amazon Neptune console or the [the section called "DescribeDBClusterParameters" \(p. 877\)](#) command to verify that your DB cluster parameter group has been created or modified.

Request

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – *Required:* UTF-8 string.

The name of the DB cluster parameter group to modify.

- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – *Required:* An array of [Parameter \(p. 863\)](#) Parameter objects.

A list of parameters in the DB cluster parameter group to modify.

Response

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

The name of the DB cluster parameter group.

Constraints:

- Must be 1 to 255 letters or numbers.
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens

Note

This value is stored as a lowercase string.

Errors

- [DBParameterGroupNotFoundFault \(p. 943\)](#)
- [InvalidDBParameterGroupStateFault \(p. 948\)](#)

ResetDBParameterGroup (action)

The Amazon CLI name for this API is: `reset-db-parameter-group`, and the Python name is: `reset_db_parameter_group`.

Modifies the parameters of a DB parameter group to the engine/system default value. To reset specific parameters, provide a list of the following: `ParameterName` and `ApplyMethod`. To reset the entire DB parameter group, specify the `DBParameterGroup` name and `ResetAllParameters` parameters. When resetting the entire group, dynamic parameters are updated immediately and static parameters are set to `pending-reboot` to take effect on the next DB instance restart or `RebootDBInstance` request.

Request

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – *Required*: UTF-8 string.

The name of the DB parameter group.

Constraints:

- Must match the name of an existing `DBParameterGroup`.
- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – An array of [Parameter \(p. 863\)](#) Parameter objects.

To reset the entire DB parameter group, specify the `DBParameterGroup` name and `ResetAllParameters` parameters. To reset specific parameters, provide a list of the following: `ParameterName` and `ApplyMethod`. A maximum of 20 parameters can be modified in a single request.

Valid Values (for Apply method): `pending-reboot`

- **ResetAllParameters** (in the CLI: `--reset-all-parameters`, or in Python: `reset_all_parameters`) – Boolean.

Specifies whether (`true`) or not (`false`) to reset all parameters in the DB parameter group to default values.

Default: `true`

Response

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – UTF-8 string.

Provides the name of the DB parameter group.

Errors

- [InvalidDBParameterGroupStateFault \(p. 948\)](#)
- [DBParameterGroupNotFoundFault \(p. 943\)](#)

ResetDBClusterParameterGroup (action)

The Amazon CLI name for this API is: `reset-db-cluster-parameter-group`, and the Python name is: `reset_db_cluster_parameter_group`.

Modifies the parameters of a DB cluster parameter group to the default value. To reset specific parameters submit a list of the following: `ParameterName` and `ApplyMethod`. To reset the entire DB cluster parameter group, specify the `DBClusterParameterGroupName` and `ResetAllParameters` parameters.

When resetting the entire group, dynamic parameters are updated immediately and static parameters are set to `pending-reboot` to take effect on the next DB instance restart or [the section called](#)

“[RebootDBInstance](#)” (p. 855) request. You must call [the section called “RebootDBInstance”](#) (p. 855) for every DB instance in your DB cluster that you want the updated static parameter to apply to.

Request

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – *Required*: UTF-8 string.

The name of the DB cluster parameter group to reset.

- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – An array of [Parameter](#) (p. 863) Parameter objects.

A list of parameter names in the DB cluster parameter group to reset to the default values. You can't use this parameter if the `ResetAllParameters` parameter is set to `true`.

- **ResetAllParameters** (in the CLI: `--reset-all-parameters`, or in Python: `reset_all_parameters`) – Boolean.

A value that is set to `true` to reset all parameters in the DB cluster parameter group to their default values, and `false` otherwise. You can't use this parameter if there is a list of parameter names specified for the `Parameters` parameter.

Response

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

The name of the DB cluster parameter group.

Constraints:

- Must be 1 to 255 letters or numbers.
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens

Note

This value is stored as a lowercase string.

Errors

- [InvalidDBParameterGroupStateFault](#) (p. 948)
- [DBParameterGroupNotFoundFault](#) (p. 943)

DescribeDBParameters (action)

The Amazon CLI name for this API is: `describe-db-parameters`, and the Python name is: `describe_db_parameters`.

Returns the detailed parameter list for a particular DB parameter group.

Request

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – *Required*: UTF-8 string.

The name of a specific DB parameter group to return details for.

Constraints:

- If supplied, must match the name of an existing DBParameterGroup.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBParameters` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

- **Source** (in the CLI: `--source`, or in Python: `source`) – UTF-8 string.

The parameter types to return.

Default: All parameter types returned

Valid Values: `user` | `system` | `engine-default`

Response

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – An array of [Parameter \(p. 863\)](#) Parameter objects.

A list of [the section called "Parameter" \(p. 863\)](#) values.

Errors

- [DBParameterGroupNotFoundFault \(p. 943\)](#)

DescribeDBParameterGroups (action)

The Amazon CLI name for this API is: `describe-db-parameter-groups`, and the Python name is: `describe_db_parameter_groups`.

Returns a list of `DBParameterGroup` descriptions. If a `DBParameterGroupName` is specified, the list will contain only the description of the specified DB parameter group.

Request

- **DBParameterGroupName** (in the CLI: `--db-parameter-group-name`, or in Python: `db_parameter_group_name`) – UTF-8 string.

The name of a specific DB parameter group to return details for.

Constraints:

- If supplied, must match the name of an existing `DBClusterParameterGroup`.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBParameterGroups` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

- **DBParameterGroups** (in the CLI: `--db-parameter-groups`, or in Python: `db_parameter_groups`) – An array of [DBParameterGroup \(p. 863\)](#) `DBParameterGroup` objects.

A list of [the section called "DBParameterGroup" \(p. 863\)](#) instances.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Errors

- [DBParameterGroupNotFoundFault \(p. 943\)](#)

DescribeDBClusterParameters (action)

The Amazon CLI name for this API is: `describe-db-cluster-parameters`, and the Python name is: `describe_db_cluster_parameters`.

Returns the detailed parameter list for a particular DB cluster parameter group.

Request

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – *Required*: UTF-8 string.

The name of a specific DB cluster parameter group to return parameter details for.

Constraints:

- If supplied, must match the name of an existing `DBClusterParameterGroup`.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBClusterParameters` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

- **Source** (in the CLI: `--source`, or in Python: `source`) – UTF-8 string.

A value that indicates to return only parameters for a specific source. Parameter sources can be `engine`, `service`, or `customer`.

Response

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBClusterParameters` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – An array of [Parameter \(p. 863\)](#) Parameter objects.

Provides a list of parameters for the DB cluster parameter group.

Errors

- [DBParameterGroupNotFoundFault \(p. 943\)](#)

DescribeDBClusterParameterGroups (action)

The Amazon CLI name for this API is: `describe-db-cluster-parameter-groups`, and the Python name is: `describe_db_cluster_parameter_groups`.

Returns a list of `DBClusterParameterGroup` descriptions. If a `DBClusterParameterGroupName` parameter is specified, the list will contain only the description of the specified DB cluster parameter group.

Request

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

The name of a specific DB cluster parameter group to return details for.

Constraints:

- If supplied, must match the name of an existing `DBClusterParameterGroup`.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBClusterParameterGroups` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

- **DBClusterParameterGroups** (in the CLI: `--db-cluster-parameter-groups`, or in Python: `db_cluster_parameter_groups`) – An array of [DBClusterParameterGroup](#) (p. 864) `DBClusterParameterGroup` objects.

A list of DB cluster parameter groups.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBClusterParameterGroups` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Errors

- [DBParameterGroupNotFoundFault](#) (p. 943)

DescribeEngineDefaultParameters (action)

The Amazon CLI name for this API is: `describe-engine-default-parameters`, and the Python name is: `describe_engine_default_parameters`.

Returns the default engine and system parameter information for the specified database engine.

Request

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – *Required:* UTF-8 string.

The name of the DB parameter group family.

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter](#) (p. 937) `Filter` objects.

Not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeEngineDefaultParameters` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

Contains the result of a successful invocation of the [the section called "DescribeEngineDefaultParameters" \(p. 879\)](#) action.

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – UTF-8 string.

Specifies the name of the DB parameter group family that the engine default parameters apply to.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous EngineDefaults request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – An array of [Parameter \(p. 863\)](#) Parameter objects.

Contains a list of engine default parameters.

DescribeEngineDefaultClusterParameters (action)

The Amazon CLI name for this API is: `describe-engine-default-cluster-parameters`, and the Python name is: `describe_engine_default_cluster_parameters`.

Returns the default engine and system parameter information for the cluster database engine.

Request

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – *Required*: UTF-8 string.

The name of the DB cluster parameter group family to return engine parameter information for.

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeEngineDefaultClusterParameters` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

Contains the result of a successful invocation of the [the section called "DescribeEngineDefaultParameters" \(p. 879\)](#) action.

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – UTF-8 string.

Specifies the name of the DB parameter group family that the engine default parameters apply to.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous EngineDefaults request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **Parameters** (in the CLI: `--parameters`, or in Python: `parameters`) – An array of [Parameter \(p. 863\)](#) Parameter objects.

Contains a list of engine default parameters.

Neptune Subnet API

Structures:

- [Subnet \(structure\) \(p. 881\)](#)
- [DBSubnetGroup \(structure\) \(p. 882\)](#)

Actions:

- [CreateDBSubnetGroup \(action\) \(p. 882\)](#)
- [DeleteDBSubnetGroup \(action\) \(p. 884\)](#)
- [ModifyDBSubnetGroup \(action\) \(p. 884\)](#)
- [DescribeDBSubnetGroups \(action\) \(p. 885\)](#)

Subnet (structure)

Specifies a subnet.

This data type is used as a response element in the [the section called "DescribeDBSubnetGroups" \(p. 885\)](#) action.

Fields

- **SubnetAvailabilityZone** – An [AvailabilityZone \(p. 935\)](#) object.

Specifies the EC2 Availability Zone that the subnet is in.

- **SubnetIdentifier** – UTF-8 string.

Specifies the identifier of the subnet.

- **SubnetStatus** – UTF-8 string.

Specifies the status of the subnet.

DBSubnetGroup (structure)

Contains the details of an Amazon Neptune DB subnet group.

This data type is used as a response element in the [the section called "DescribeDBSubnetGroups" \(p. 885\)](#) action.

Fields

- **DBSubnetGroupArn** – UTF-8 string.
The Amazon Resource Name (ARN) for the DB subnet group.
- **DBSubnetGroupDescription** – UTF-8 string.
Provides the description of the DB subnet group.
- **DBSubnetGroupName** – UTF-8 string.
The name of the DB subnet group.
- **SubnetGroupStatus** – UTF-8 string.
Provides the status of the DB subnet group.
- **Subnets** – An array of [Subnet \(p. 881\)](#) Subnet objects.
Contains a list of [the section called "Subnet" \(p. 881\)](#) elements.
- **VpcId** – UTF-8 string.
Provides the VpcId of the DB subnet group.

DBSubnetGroup is used as the response element for:

- [CreateDBSubnetGroup \(p. 882\)](#)
- [ModifyDBSubnetGroup \(p. 884\)](#)

CreateDBSubnetGroup (action)

The Amazon CLI name for this API is: `create-db-subnet-group`, and the Python name is: `create_db_subnet_group`.

Creates a new DB subnet group. DB subnet groups must contain at least one subnet in at least two AZs in the Amazon Region.

Request

- **DBSubnetGroupDescription** (in the CLI: `--db-subnet-group-description`, or in Python: `db_subnet_group_description`) – *Required:* UTF-8 string.
The description for the DB subnet group.
- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – *Required:* UTF-8 string.
The name for the DB subnet group. This value is stored as a lowercase string.

Constraints: Must contain no more than 255 letters, numbers, periods, underscores, spaces, or hyphens. Must not be default.

Example: `mySubnetgroup`

- **SubnetIds** (in the CLI: `--subnet-ids`, or in Python: `subnet_ids`) – *Required*: An array of SubnetIdentifier— UTF-8 strings.

The EC2 Subnet IDs for the DB subnet group.

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be assigned to the new DB subnet group.

Response

Contains the details of an Amazon Neptune DB subnet group.

This data type is used as a response element in the [the section called “DescribeDBSubnetGroups” \(p. 885\)](#) action.

- **DBSubnetGroupArn** (in the CLI: `--db-subnet-group-arn`, or in Python: `db_subnet_group_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB subnet group.

- **DBSubnetGroupDescription** (in the CLI: `--db-subnet-group-description`, or in Python: `db_subnet_group_description`) – UTF-8 string.

Provides the description of the DB subnet group.

- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

The name of the DB subnet group.

- **SubnetGroupStatus** (in the CLI: `--subnet-group-status`, or in Python: `subnet_group_status`) – UTF-8 string.

Provides the status of the DB subnet group.

- **Subnets** (in the CLI: `--subnets`, or in Python: `subnets`) – An array of [Subnet \(p. 881\)](#) Subnet objects.

Contains a list of [the section called “Subnet” \(p. 881\)](#) elements.

- **VpcId** (in the CLI: `--vpc-id`, or in Python: `vpc_id`) – UTF-8 string.

Provides the VpcId of the DB subnet group.

Errors

- [DBSubnetGroupAlreadyExistsFault \(p. 945\)](#)
- [DBSubnetGroupQuotaExceededFault \(p. 946\)](#)
- [DBSubnetQuotaExceededFault \(p. 946\)](#)
- [DBSubnetGroupDoesNotCoverEnoughAZs \(p. 945\)](#)
- [InvalidSubnet \(p. 950\)](#)

DeleteDBSubnetGroup (action)

The Amazon CLI name for this API is: `delete-db-subnet-group`, and the Python name is: `delete_db_subnet_group`.

Deletes a DB subnet group.

Note

The specified database subnet group must not be associated with any DB instances.

Request

- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – *Required*: UTF-8 string.

The name of the database subnet group to delete.

Note

You can't delete the default subnet group.

Constraints:

Constraints: Must match the name of an existing DBSubnetGroup. Must not be default.

Example: `mySubnetgroup`

Response

- *No Response parameters.*

Errors

- [InvalidDBSubnetGroupStateFault](#) (p. 949)
- [InvalidDBSubnetStateFault](#) (p. 950)
- [DBSubnetGroupNotFoundFault](#) (p. 946)

ModifyDBSubnetGroup (action)

The Amazon CLI name for this API is: `modify-db-subnet-group`, and the Python name is: `modify_db_subnet_group`.

Modifies an existing DB subnet group. DB subnet groups must contain at least one subnet in at least two AZs in the Amazon Region.

Request

- **DBSubnetGroupDescription** (in the CLI: `--db-subnet-group-description`, or in Python: `db_subnet_group_description`) – UTF-8 string.

The description for the DB subnet group.

- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – *Required*: UTF-8 string.

The name for the DB subnet group. This value is stored as a lowercase string. You can't modify the default subnet group.

Constraints: Must match the name of an existing DBSubnetGroup. Must not be default.

Example: `mySubnetgroup`

- **SubnetIds** (in the CLI: `--subnet-ids`, or in Python: `subnet_ids`) – *Required:* An array of SubnetIdentifier— UTF-8 strings.

The EC2 subnet IDs for the DB subnet group.

Response

Contains the details of an Amazon Neptune DB subnet group.

This data type is used as a response element in the [the section called “DescribeDBSubnetGroups” \(p. 885\)](#) action.

- **DBSubnetGroupArn** (in the CLI: `--db-subnet-group-arn`, or in Python: `db_subnet_group_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB subnet group.

- **DBSubnetGroupDescription** (in the CLI: `--db-subnet-group-description`, or in Python: `db_subnet_group_description`) – UTF-8 string.

Provides the description of the DB subnet group.

- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

The name of the DB subnet group.

- **SubnetGroupStatus** (in the CLI: `--subnet-group-status`, or in Python: `subnet_group_status`) – UTF-8 string.

Provides the status of the DB subnet group.

- **Subnets** (in the CLI: `--subnets`, or in Python: `subnets`) – An array of [Subnet \(p. 881\)](#) Subnet objects.

Contains a list of [the section called “Subnet” \(p. 881\)](#) elements.

- **VpcId** (in the CLI: `--vpc-id`, or in Python: `vpc_id`) – UTF-8 string.

Provides the VpcId of the DB subnet group.

Errors

- [DBSubnetGroupNotFoundFault \(p. 946\)](#)
- [DBSubnetQuotaExceededFault \(p. 946\)](#)
- [SubnetAlreadyInUse \(p. 954\)](#)
- [DBSubnetGroupDoesNotCoverEnoughAZs \(p. 945\)](#)
- [InvalidSubnet \(p. 950\)](#)

DescribeDBSubnetGroups (action)

The Amazon CLI name for this API is: `describe-db-subnet-groups`, and the Python name is: `describe_db_subnet_groups`.

Returns a list of DBSubnetGroup descriptions. If a DBSubnetGroupName is specified, the list will contain only the descriptions of the specified DBSubnetGroup.

For an overview of CIDR ranges, go to the [Wikipedia Tutorial](#).

Request

- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

The name of the DB subnet group to return details for.

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous DescribeDBSubnetGroups request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

- **DBSubnetGroups** (in the CLI: `--db-subnet-groups`, or in Python: `db_subnet_groups`) – An array of [DBSubnetGroup \(p. 882\)](#) DBSubnetGroup objects.

A list of [the section called “DBSubnetGroup” \(p. 882\)](#) instances.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Errors

- [DBSubnetGroupNotFoundFault \(p. 946\)](#)

Neptune Snapshots API

Structures:

- [DBClusterSnapshot \(structure\) \(p. 887\)](#)
- [DBClusterSnapshotAttribute \(structure\) \(p. 888\)](#)
- [DBClusterSnapshotAttributesResult \(structure\) \(p. 889\)](#)

Actions:

- [CreateDBClusterSnapshot \(action\) \(p. 889\)](#)
- [DeleteDBClusterSnapshot \(action\) \(p. 891\)](#)
- [CopyDBClusterSnapshot \(action\) \(p. 893\)](#)

- [ModifyDBClusterSnapshotAttribute \(action\)](#) (p. 896)
- [RestoreDBClusterFromSnapshot \(action\)](#) (p. 898)
- [RestoreDBClusterToPointInTime \(action\)](#) (p. 904)
- [DescribeDBClusterSnapshots \(action\)](#) (p. 910)
- [DescribeDBClusterSnapshotAttributes \(action\)](#) (p. 912)

DBClusterSnapshot (structure)

Contains the details for an Amazon Neptune DB cluster snapshot

This data type is used as a response element in the [the section called “DescribeDBClusterSnapshots”](#) (p. 910) action.

Fields

- **AllocatedStorage** – Number (integer).
Specifies the allocated storage size in gibibytes (GiB).
- **AvailabilityZones** – An array of AvailabilityZone— UTF-8 strings.
Provides the list of EC2 Availability Zones that instances in the DB cluster snapshot can be restored in.
- **ClusterCreateTime** – Timestamp.
Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).
- **DBClusterIdentifier** – UTF-8 string.
Specifies the DB cluster identifier of the DB cluster that this DB cluster snapshot was created from.
- **DBClusterSnapshotArn** – UTF-8 string.
The Amazon Resource Name (ARN) for the DB cluster snapshot.
- **DBClusterSnapshotIdentifier** – UTF-8 string.
Specifies the identifier for a DB cluster snapshot. Must match the identifier of an existing snapshot.

After you restore a DB cluster using a `DBClusterSnapshotIdentifier`, you must specify the same `DBClusterSnapshotIdentifier` for any future updates to the DB cluster. When you specify this property for an update, the DB cluster is not restored from the snapshot again, and the data in the database is not changed.

However, if you don't specify the `DBClusterSnapshotIdentifier`, an empty DB cluster is created, and the original DB cluster is deleted. If you specify a property that is different from the previous snapshot restore property, the DB cluster is restored from the snapshot specified by the `DBClusterSnapshotIdentifier`, and the original DB cluster is deleted.
- **Engine** – UTF-8 string.
Specifies the name of the database engine.
- **EngineVersion** – UTF-8 string.
Provides the version of the database engine for this DB cluster snapshot.
- **IAMDatabaseAuthenticationEnabled** – Boolean.
True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.
- **KmsKeyId** – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster snapshot.

- **LicenseModel** – UTF-8 string.

Provides the license model information for this DB cluster snapshot.

- **PercentProgress** – Number (integer).

Specifies the percentage of the estimated data that has been transferred.

- **Port** – Number (integer).

Specifies the port that the DB cluster was listening on at the time of the snapshot.

- **SnapshotCreateTime** – Timestamp.

Provides the time when the snapshot was taken, in Universal Coordinated Time (UTC).

- **SnapshotType** – UTF-8 string.

Provides the type of the DB cluster snapshot.

- **SourceDBClusterSnapshotArn** – UTF-8 string.

If the DB cluster snapshot was copied from a source DB cluster snapshot, the Amazon Resource Name (ARN) for the source DB cluster snapshot, otherwise, a null value.

- **Status** – UTF-8 string.

Specifies the status of this DB cluster snapshot.

- **StorageEncrypted** – Boolean.

Specifies whether the DB cluster snapshot is encrypted.

- **VpcId** – UTF-8 string.

Provides the VPC ID associated with the DB cluster snapshot.

`DBClusterSnapshot` is used as the response element for:

- [CreateDBClusterSnapshot](#) (p. 889)
- [CopyDBClusterSnapshot](#) (p. 893)
- [DeleteDBClusterSnapshot](#) (p. 891)

DBClusterSnapshotAttribute (structure)

Contains the name and values of a manual DB cluster snapshot attribute.

Manual DB cluster snapshot attributes are used to authorize other Amazon accounts to restore a manual DB cluster snapshot. For more information, see the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

Fields

- **AttributeName** – UTF-8 string.

The name of the manual DB cluster snapshot attribute.

The attribute named `restore` refers to the list of Amazon accounts that have permission to copy or restore the manual DB cluster snapshot. For more information, see the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

- **AttributeValues** – An array of `AttributeValue`— UTF-8 strings.

The value(s) for the manual DB cluster snapshot attribute.

If the `AttributeName` field is set to `restore`, then this element returns a list of IDs of the Amazon accounts that are authorized to copy or restore the manual DB cluster snapshot. If a value of `all` is in the list, then the manual DB cluster snapshot is public and available for any Amazon account to copy or restore.

DBClusterSnapshotAttributesResult (structure)

Contains the results of a successful call to the [the section called "DescribeDBClusterSnapshotAttributes" \(p. 912\)](#) API action.

Manual DB cluster snapshot attributes are used to authorize other Amazon accounts to copy or restore a manual DB cluster snapshot. For more information, see the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

Fields

- **DBClusterSnapshotAttributes** – An array of [DBClusterSnapshotAttribute \(p. 888\)](#) `DBClusterSnapshotAttribute` objects.

The list of attributes and values for the manual DB cluster snapshot.

- **DBClusterSnapshotIdentifier** – UTF-8 string.

The identifier of the manual DB cluster snapshot that the attributes apply to.

`DBClusterSnapshotAttributesResult` is used as the response element for:

- [DescribeDBClusterSnapshotAttributes \(p. 912\)](#)
- [ModifyDBClusterSnapshotAttribute \(p. 896\)](#)

CreateDBClusterSnapshot (action)

The Amazon CLI name for this API is: `create-db-cluster-snapshot`, and the Python name is: `create_db_cluster_snapshot`.

Creates a snapshot of a DB cluster.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required:* UTF-8 string.

The identifier of the DB cluster to create a snapshot for. This parameter is not case-sensitive.

Constraints:

- Must match the identifier of an existing `DBCluster`.

Example: `my-cluster1`

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – *Required:* UTF-8 string.

The identifier of the DB cluster snapshot. This parameter is stored as a lowercase string.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- First character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster1-snapshot1`

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be assigned to the DB cluster snapshot.

Response

Contains the details for an Amazon Neptune DB cluster snapshot

This data type is used as a response element in the [the section called “DescribeDBClusterSnapshots” \(p. 910\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

Specifies the allocated storage size in gibibytes (GiB).

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of AvailabilityZone— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster snapshot can be restored in.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Specifies the DB cluster identifier of the DB cluster that this DB cluster snapshot was created from.

- **DBClusterSnapshotArn** (in the CLI: `--db-cluster-snapshot-arn`, or in Python: `db_cluster_snapshot_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster snapshot.

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – UTF-8 string.

Specifies the identifier for a DB cluster snapshot. Must match the identifier of an existing snapshot.

After you restore a DB cluster using a `DBClusterSnapshotIdentifier`, you must specify the same `DBClusterSnapshotIdentifier` for any future updates to the DB cluster. When you specify this property for an update, the DB cluster is not restored from the snapshot again, and the data in the database is not changed.

However, if you don't specify the `DBClusterSnapshotIdentifier`, an empty DB cluster is created, and the original DB cluster is deleted. If you specify a property that is different from the previous snapshot restore property, the DB cluster is restored from the snapshot specified by the `DBClusterSnapshotIdentifier`, and the original DB cluster is deleted.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Specifies the name of the database engine.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Provides the version of the database engine for this DB cluster snapshot.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster snapshot.

- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.

Provides the license model information for this DB cluster snapshot.

- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – Number (integer).

Specifies the percentage of the estimated data that has been transferred.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

Specifies the port that the DB cluster was listening on at the time of the snapshot.

- **SnapshotCreateTime** (in the CLI: `--snapshot-create-time`, or in Python: `snapshot_create_time`) – Timestamp.

Provides the time when the snapshot was taken, in Universal Coordinated Time (UTC).

- **SnapshotType** (in the CLI: `--snapshot-type`, or in Python: `snapshot_type`) – UTF-8 string.

Provides the type of the DB cluster snapshot.

- **SourceDBClusterSnapshotArn** (in the CLI: `--source-db-cluster-snapshot-arn`, or in Python: `source_db_cluster_snapshot_arn`) – UTF-8 string.

If the DB cluster snapshot was copied from a source DB cluster snapshot, the Amazon Resource Name (ARN) for the source DB cluster snapshot, otherwise, a null value.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the status of this DB cluster snapshot.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster snapshot is encrypted.

- **VpcId** (in the CLI: `--vpc-id`, or in Python: `vpc_id`) – UTF-8 string.

Provides the VPC ID associated with the DB cluster snapshot.

Errors

- [DBClusterSnapshotAlreadyExistsFault \(p. 942\)](#)
- [InvalidDBClusterStateFault \(p. 948\)](#)
- [DBClusterNotFoundFault \(p. 940\)](#)
- [SnapshotQuotaExceededFault \(p. 953\)](#)
- [InvalidDBClusterSnapshotStateFault \(p. 948\)](#)

DeleteDBClusterSnapshot (action)

The Amazon CLI name for this API is: `delete-db-cluster-snapshot`, and the Python name is: `delete_db_cluster_snapshot`.

Deletes a DB cluster snapshot. If the snapshot is being copied, the copy operation is terminated.

Note

The DB cluster snapshot must be in the available state to be deleted.

Request

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – *Required*: UTF-8 string.

The identifier of the DB cluster snapshot to delete.

Constraints: Must be the name of an existing DB cluster snapshot in the available state.

Response

Contains the details for an Amazon Neptune DB cluster snapshot

This data type is used as a response element in the [the section called “DescribeDBClusterSnapshots” \(p. 910\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

Specifies the allocated storage size in gibibytes (GiB).

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of AvailabilityZone— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster snapshot can be restored in.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Specifies the DB cluster identifier of the DB cluster that this DB cluster snapshot was created from.

- **DBClusterSnapshotArn** (in the CLI: `--db-cluster-snapshot-arn`, or in Python: `db_cluster_snapshot_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster snapshot.

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – UTF-8 string.

Specifies the identifier for a DB cluster snapshot. Must match the identifier of an existing snapshot.

After you restore a DB cluster using a `DBClusterSnapshotIdentifier`, you must specify the same `DBClusterSnapshotIdentifier` for any future updates to the DB cluster. When you specify this property for an update, the DB cluster is not restored from the snapshot again, and the data in the database is not changed.

However, if you don't specify the `DBClusterSnapshotIdentifier`, an empty DB cluster is created, and the original DB cluster is deleted. If you specify a property that is different from the previous snapshot restore property, the DB cluster is restored from the snapshot specified by the `DBClusterSnapshotIdentifier`, and the original DB cluster is deleted.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Specifies the name of the database engine.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.
Provides the version of the database engine for this DB cluster snapshot.
- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.
True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.
- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.
If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster snapshot.
- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.
Provides the license model information for this DB cluster snapshot.
- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – Number (integer).
Specifies the percentage of the estimated data that has been transferred.
- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).
Specifies the port that the DB cluster was listening on at the time of the snapshot.
- **SnapshotCreateTime** (in the CLI: `--snapshot-create-time`, or in Python: `snapshot_create_time`) – Timestamp.
Provides the time when the snapshot was taken, in Universal Coordinated Time (UTC).
- **SnapshotType** (in the CLI: `--snapshot-type`, or in Python: `snapshot_type`) – UTF-8 string.
Provides the type of the DB cluster snapshot.
- **SourceDBClusterSnapshotArn** (in the CLI: `--source-db-cluster-snapshot-arn`, or in Python: `source_db_cluster_snapshot_arn`) – UTF-8 string.
If the DB cluster snapshot was copied from a source DB cluster snapshot, the Amazon Resource Name (ARN) for the source DB cluster snapshot, otherwise, a null value.
- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.
Specifies the status of this DB cluster snapshot.
- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.
Specifies whether the DB cluster snapshot is encrypted.
- **VpcId** (in the CLI: `--vpc-id`, or in Python: `vpc_id`) – UTF-8 string.
Provides the VPC ID associated with the DB cluster snapshot.

Errors

- [InvalidDBClusterSnapshotStateFault](#) (p. 948)
- [DBClusterSnapshotNotFoundFault](#) (p. 942)

CopyDBClusterSnapshot (action)

The Amazon CLI name for this API is: `copy-db-cluster-snapshot`, and the Python name is: `copy_db_cluster_snapshot`.

Copies a snapshot of a DB cluster.

To copy a DB cluster snapshot from a shared manual DB cluster snapshot, `SourceDBClusterSnapshotIdentifier` must be the Amazon Resource Name (ARN) of the shared DB cluster snapshot.

Request

- **CopyTags** (in the CLI: `--copy-tags`, or in Python: `copy_tags`) – Boolean.

True to copy all tags from the source DB cluster snapshot to the target DB cluster snapshot, and otherwise false. The default is false.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

The Amazon Amazon KMS key ID for an encrypted DB cluster snapshot. The KMS key ID is the Amazon Resource Name (ARN), KMS key identifier, or the KMS key alias for the KMS encryption key.

If you copy an encrypted DB cluster snapshot from your Amazon account, you can specify a value for `KmsKeyId` to encrypt the copy with a new KMS encryption key. If you don't specify a value for `KmsKeyId`, then the copy of the DB cluster snapshot is encrypted with the same KMS key as the source DB cluster snapshot.

If you copy an encrypted DB cluster snapshot that is shared from another Amazon account, then you must specify a value for `KmsKeyId`.

KMS encryption keys are specific to the Amazon Region that they are created in, and you can't use encryption keys from one Amazon Region in another Amazon Region.

You cannot encrypt an unencrypted DB cluster snapshot when you copy it. If you try to copy an unencrypted DB cluster snapshot and specify a value for the `KmsKeyId` parameter, an error is returned.

- **PreSignedUrl** (in the CLI: `--pre-signed-url`, or in Python: `pre_signed_url`) – UTF-8 string.

Not currently supported.

- **SourceDBClusterSnapshotIdentifier** (in the CLI: `--source-db-cluster-snapshot-identifier`, or in Python: `source_db_cluster_snapshot_identifier`) – *Required*: UTF-8 string.

The identifier of the DB cluster snapshot to copy. This parameter is not case-sensitive.

Constraints:

- Must specify a valid system snapshot in the "available" state.
- Specify a valid DB snapshot identifier.

Example: `my-cluster-snapshot1`

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to assign to the new DB cluster snapshot copy.

- **TargetDBClusterSnapshotIdentifier** (in the CLI: `--target-db-cluster-snapshot-identifier`, or in Python: `target_db_cluster_snapshot_identifier`) – *Required*: UTF-8 string.

The identifier of the new DB cluster snapshot to create from the source DB cluster snapshot. This parameter is not case-sensitive.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens.
- First character must be a letter.
- Cannot end with a hyphen or contain two consecutive hyphens.

Example: `my-cluster-snapshot2`

Response

Contains the details for an Amazon Neptune DB cluster snapshot.

This data type is used as a response element in the [the section called "DescribeDBClusterSnapshots" \(p. 910\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

Specifies the allocated storage size in gibibytes (GiB).

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of AvailabilityZone— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster snapshot can be restored in.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Specifies the DB cluster identifier of the DB cluster that this DB cluster snapshot was created from.

- **DBClusterSnapshotArn** (in the CLI: `--db-cluster-snapshot-arn`, or in Python: `db_cluster_snapshot_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster snapshot.

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – UTF-8 string.

Specifies the identifier for a DB cluster snapshot. Must match the identifier of an existing snapshot.

After you restore a DB cluster using a `DBClusterSnapshotIdentifier`, you must specify the same `DBClusterSnapshotIdentifier` for any future updates to the DB cluster. When you specify this property for an update, the DB cluster is not restored from the snapshot again, and the data in the database is not changed.

However, if you don't specify the `DBClusterSnapshotIdentifier`, an empty DB cluster is created, and the original DB cluster is deleted. If you specify a property that is different from the previous snapshot restore property, the DB cluster is restored from the snapshot specified by the `DBClusterSnapshotIdentifier`, and the original DB cluster is deleted.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Specifies the name of the database engine.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Provides the version of the database engine for this DB cluster snapshot.

- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster snapshot.

- **LicenseModel** (in the CLI: `--license-model`, or in Python: `license_model`) – UTF-8 string.

Provides the license model information for this DB cluster snapshot.

- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – Number (integer).

Specifies the percentage of the estimated data that has been transferred.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

Specifies the port that the DB cluster was listening on at the time of the snapshot.

- **SnapshotCreateTime** (in the CLI: `--snapshot-create-time`, or in Python: `snapshot_create_time`) – Timestamp.

Provides the time when the snapshot was taken, in Universal Coordinated Time (UTC).

- **SnapshotType** (in the CLI: `--snapshot-type`, or in Python: `snapshot_type`) – UTF-8 string.

Provides the type of the DB cluster snapshot.

- **SourceDBClusterSnapshotArn** (in the CLI: `--source-db-cluster-snapshot-arn`, or in Python: `source_db_cluster_snapshot_arn`) – UTF-8 string.

If the DB cluster snapshot was copied from a source DB cluster snapshot, the Amazon Resource Name (ARN) for the source DB cluster snapshot, otherwise, a null value.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the status of this DB cluster snapshot.

- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster snapshot is encrypted.

- **VpcId** (in the CLI: `--vpc-id`, or in Python: `vpc_id`) – UTF-8 string.

Provides the VPC ID associated with the DB cluster snapshot.

Errors

- [DBClusterSnapshotAlreadyExistsFault](#) (p. 942)
- [DBClusterSnapshotNotFoundFault](#) (p. 942)
- [InvalidDBClusterStateFault](#) (p. 948)
- [InvalidDBClusterSnapshotStateFault](#) (p. 948)
- [SnapshotQuotaExceededFault](#) (p. 953)
- [KMSKeyNotAccessibleFault](#) (p. 951)

ModifyDBClusterSnapshotAttribute (action)

The Amazon CLI name for this API is: `modify-db-cluster-snapshot-attribute`, and the Python name is: `modify_db_cluster_snapshot_attribute`.

Adds an attribute and values to, or removes an attribute and values from, a manual DB cluster snapshot.

To share a manual DB cluster snapshot with other Amazon accounts, specify `restore` as the `AttributeName` and use the `ValuesToAdd` parameter to add a list of IDs of the Amazon accounts that are authorized to restore the manual DB cluster snapshot. Use the value `all` to make the manual DB cluster snapshot public, which means that it can be copied or restored by all Amazon accounts. Do not

add the `all` value for any manual DB cluster snapshots that contain private information that you don't want available to all Amazon accounts. If a manual DB cluster snapshot is encrypted, it can be shared, but only by specifying a list of authorized Amazon account IDs for the `ValuesToAdd` parameter. You can't use `all` as a value for that parameter in this case.

To view which Amazon accounts have access to copy or restore a manual DB cluster snapshot, or whether a manual DB cluster snapshot is public or private, use the [the section called "DescribeDBClusterSnapshotAttributes" \(p. 912\)](#) API action.

Request

- **AttributeName** (in the CLI: `--attribute-name`, or in Python: `attribute_name`) – *Required:* UTF-8 string.

The name of the DB cluster snapshot attribute to modify.

To manage authorization for other Amazon accounts to copy or restore a manual DB cluster snapshot, set this value to `restore`.

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – *Required:* UTF-8 string.

The identifier for the DB cluster snapshot to modify the attributes for.

- **ValuesToAdd** (in the CLI: `--values-to-add`, or in Python: `values_to_add`) – An array of `AttributeValue`— UTF-8 strings.

A list of DB cluster snapshot attributes to add to the attribute specified by `AttributeName`.

To authorize other Amazon accounts to copy or restore a manual DB cluster snapshot, set this list to include one or more Amazon account IDs, or `all` to make the manual DB cluster snapshot restorable by any Amazon account. Do not add the `all` value for any manual DB cluster snapshots that contain private information that you don't want available to all Amazon accounts.

- **ValuesToRemove** (in the CLI: `--values-to-remove`, or in Python: `values_to_remove`) – An array of `AttributeValue`— UTF-8 strings.

A list of DB cluster snapshot attributes to remove from the attribute specified by `AttributeName`.

To remove authorization for other Amazon accounts to copy or restore a manual DB cluster snapshot, set this list to include one or more Amazon account identifiers, or `all` to remove authorization for any Amazon account to copy or restore the DB cluster snapshot. If you specify `all`, an Amazon account whose account ID is explicitly added to the `restore` attribute can still copy or restore a manual DB cluster snapshot.

Response

Contains the results of a successful call to the [the section called "DescribeDBClusterSnapshotAttributes" \(p. 912\)](#) API action.

Manual DB cluster snapshot attributes are used to authorize other Amazon accounts to copy or restore a manual DB cluster snapshot. For more information, see the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

- **DBClusterSnapshotAttributes** (in the CLI: `--db-cluster-snapshot-attributes`, or in Python: `db_cluster_snapshot_attributes`) – An array of [DBClusterSnapshotAttribute \(p. 888\)](#) `DBClusterSnapshotAttribute` objects.

The list of attributes and values for the manual DB cluster snapshot.

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – UTF-8 string.

The identifier of the manual DB cluster snapshot that the attributes apply to.

Errors

- [DBClusterSnapshotNotFoundFault](#) (p. 942)
- [InvalidDBClusterSnapshotStateFault](#) (p. 948)
- [SharedSnapshotQuotaExceededFault](#) (p. 953)

RestoreDBClusterFromSnapshot (action)

The Amazon CLI name for this API is: `restore-db-cluster-from-snapshot`, and the Python name is: `restore_db_cluster_from_snapshot`.

Creates a new DB cluster from a DB snapshot or DB cluster snapshot.

If a DB snapshot is specified, the target DB cluster is created from the source DB snapshot with a default configuration and default security group.

If a DB cluster snapshot is specified, the target DB cluster is created from the source DB cluster restore point with the same configuration as the original source DB cluster, except that the new DB cluster is created with the default security group.

Request

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the restored DB cluster can be created in.

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to true, tags are copied to any snapshot of the restored DB cluster that is created.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Not supported.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

The name of the DB cluster to create from the DB snapshot or DB cluster snapshot. This parameter isn't case-sensitive.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens

Example: `my-snapshot-id`

- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

The name of the DB cluster parameter group to associate with the new DB cluster.

Constraints:

- If supplied, must match the name of an existing DBClusterParameterGroup.
- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

The name of the DB subnet group to use for the new DB cluster.

Constraints: If supplied, must match the name of an existing DBSubnetGroup.

Example: `mySubnetgroup`

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

A value that indicates whether the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled. By default, deletion protection is disabled.

- **EnableCloudwatchLogsExports** (in the CLI: `--enable-cloudwatch-logs-exports`, or in Python: `enable_cloudwatch_logs_exports`) – An array of UTF-8 strings.

The list of logs that the restored DB cluster is to export to Amazon CloudWatch Logs.

- **EnableIAMDatabaseAuthentication** (in the CLI: `--enable-iam-database-authentication`, or in Python: `enable_iam_database_authentication`) – Boolean.

True to enable mapping of Amazon Identity and Access Management (IAM) accounts to database accounts, and otherwise false.

Default: `false`

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – *Required*: UTF-8 string.

The database engine to use for the new DB cluster.

Default: The same as source

Constraint: Must be compatible with the engine of the source

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

The version of the database engine to use for the new DB cluster.

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

The Amazon KMS key identifier to use when restoring an encrypted DB cluster from a DB snapshot or DB cluster snapshot.

The KMS key identifier is the Amazon Resource Name (ARN) for the KMS encryption key. If you are restoring a DB cluster with the same Amazon account that owns the KMS encryption key used to encrypt the new DB cluster, then you can use the KMS key alias instead of the ARN for the KMS encryption key.

If you do not specify a value for the `KmsKeyId` parameter, then the following will occur:

- If the DB snapshot or DB cluster snapshot in `SnapshotIdentifier` is encrypted, then the restored DB cluster is encrypted using the KMS key that was used to encrypt the DB snapshot or DB cluster snapshot.
- If the DB snapshot or DB cluster snapshot in `SnapshotIdentifier` is not encrypted, then the restored DB cluster is not encrypted.
- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

The port number on which the new DB cluster accepts connections.

Default: The same port as the original DB cluster.

- **SnapshotIdentifier** (in the CLI: `--snapshot-identifier`, or in Python: `snapshot_identifier`) – *Required:* UTF-8 string.

The identifier for the DB snapshot or DB cluster snapshot to restore from.

You can use either the name or the Amazon Resource Name (ARN) to specify a DB cluster snapshot. However, you can use only the ARN to specify a DB snapshot.

Constraints:

- Must match the identifier of an existing Snapshot.
- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be assigned to the restored DB cluster.

- **VpcSecurityGroupIds** (in the CLI: `--vpc-security-group-ids`, or in Python: `vpc_security_group_ids`) – An array of VpcSecurityGroupId— UTF-8 strings.

A list of VPC security groups that the new DB cluster will belong to.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called “DescribeDBClusters” \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) DBClusterRole objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of AvailabilityZone— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to `true`, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to `true`, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.

Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.

- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.

Not supported by Neptune.
- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.

Specifies the earliest time to which a database can be restored with point-in-time restore.
- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.

A list of log types that this DB cluster is configured to export to CloudWatch Logs.
- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.

Specifies the connection endpoint for the primary instance of the DB cluster.
- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

Provides the name of the database engine to be used for this DB cluster.
- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

Indicates the database engine version.
- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.
- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.

True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.
- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.
- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.

Specifies the latest time to which a database can be restored with point-in-time restore.
- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.

Specifies whether the DB cluster has instances in multiple Availability Zones.
- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.

Not supported by Neptune.
- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.

Specifies the progress of the operation as a percentage.
- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

Specifies the port that the database engine is listening on.
- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.

Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the [BackupRetentionPeriod](#).

- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.

Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).
- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.

The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.
- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.

Contains one or more identifiers of the Read Replicas associated with this DB cluster.
- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.

Not supported by Neptune.
- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.

Not supported by Neptune.
- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

Specifies the current state of this DB cluster.
- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.

Specifies whether the DB cluster is encrypted.
- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.

Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterAlreadyExistsFault](#) (p. 940)
- [DBClusterQuotaExceededFault](#) (p. 941)
- [StorageQuotaExceededFault](#) (p. 953)
- [DBSubnetGroupNotFoundFault](#) (p. 946)
- [DBSnapshotNotFoundFault](#) (p. 945)
- [DBClusterSnapshotNotFoundFault](#) (p. 942)
- [InsufficientDBClusterCapacityFault](#) (p. 947)
- [InsufficientStorageClusterCapacityFault](#) (p. 948)
- [InvalidDBSnapshotStateFault](#) (p. 949)
- [InvalidDBClusterSnapshotStateFault](#) (p. 948)
- [StorageQuotaExceededFault](#) (p. 953)

- [InvalidVPCNetworkStateFault](#) (p. 951)
- [InvalidRestoreFault](#) (p. 950)
- [DBSubnetGroupNotFoundFault](#) (p. 946)
- [InvalidSubnet](#) (p. 950)
- [OptionGroupNotFoundFault](#) (p. 951)
- [KMSKeyNotAccessibleFault](#) (p. 951)
- [DBClusterParameterGroupNotFoundFault](#) (p. 941)

RestoreDBClusterToPointInTime (action)

The Amazon CLI name for this API is: `restore-db-cluster-to-point-in-time`, and the Python name is: `restore_db_cluster_to_point_in_time`.

Restores a DB cluster to an arbitrary point in time. Users can restore to any point in time before `LatestRestorableTime` for up to `BackupRetentionPeriod` days. The target DB cluster is created from the source DB cluster with the same configuration as the original DB cluster, except that the new DB cluster is created with the default DB security group.

Note

This action only restores the DB cluster, not the DB instances for that DB cluster. You must invoke the [the section called "CreateDBInstance" \(p. 833\)](#) action to create DB instances for the restored DB cluster, specifying the identifier of the restored DB cluster in `DBClusterIdentifier`. You can create DB instances only after the `RestoreDBClusterToPointInTime` action has completed and the DB cluster is available.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – *Required*: UTF-8 string.

The name of the new DB cluster to be created.

Constraints:

- Must contain from 1 to 63 letters, numbers, or hyphens
- First character must be a letter
- Cannot end with a hyphen or contain two consecutive hyphens
- **DBClusterParameterGroupName** (in the CLI: `--db-cluster-parameter-group-name`, or in Python: `db_cluster_parameter_group_name`) – UTF-8 string.

The name of the DB cluster parameter group to associate with the new DB cluster.

Constraints:

- If supplied, must match the name of an existing `DBClusterParameterGroup`.
- **DBSubnetGroupName** (in the CLI: `--db-subnet-group-name`, or in Python: `db_subnet_group_name`) – UTF-8 string.

The DB subnet group name to use for the new DB cluster.

Constraints: If supplied, must match the name of an existing `DBSubnetGroup`.

Example: `mySubnetgroup`

- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.

A value that indicates whether the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled. By default, deletion protection is disabled.

- **EnableCloudwatchLogsExports** (in the CLI: `--enable-cloudwatch-logs-exports`, or in Python: `enable_cloudwatch_logs_exports`) – An array of UTF-8 strings.

The list of logs that the restored DB cluster is to export to CloudWatch Logs.

- **EnableIAMDatabaseAuthentication** (in the CLI: `--enable-iam-database-authentication`, or in Python: `enable_iam_database_authentication`) – Boolean.

True to enable mapping of Amazon Identity and Access Management (IAM) accounts to database accounts, and otherwise false.

Default: `false`

- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.

The Amazon KMS key identifier to use when restoring an encrypted DB cluster from an encrypted DB cluster.

The KMS key identifier is the Amazon Resource Name (ARN) for the KMS encryption key. If you are restoring a DB cluster with the same Amazon account that owns the KMS encryption key used to encrypt the new DB cluster, then you can use the KMS key alias instead of the ARN for the KMS encryption key.

You can restore to a new DB cluster and encrypt the new DB cluster with a KMS key that is different than the KMS key used to encrypt the source DB cluster. The new DB cluster is encrypted with the KMS key identified by the `KmsKeyId` parameter.

If you do not specify a value for the `KmsKeyId` parameter, then the following will occur:

- If the DB cluster is encrypted, then the restored DB cluster is encrypted using the KMS key that was used to encrypt the source DB cluster.
- If the DB cluster is not encrypted, then the restored DB cluster is not encrypted.

If `DBClusterIdentifier` refers to a DB cluster that is not encrypted, then the restore request is rejected.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).

The port number on which the new DB cluster accepts connections.

Constraints: Value must be 1150–65535

Default: The same port as the original DB cluster.

- **RestoreToTime** (in the CLI: `--restore-to-time`, or in Python: `restore_to_time`) – Timestamp.

The date and time to restore the DB cluster to.

Valid Values: Value must be a time in Universal Coordinated Time (UTC) format

Constraints:

- Must be before the latest restorable time for the DB instance
- Must be specified if `UseLatestRestorableTime` parameter is not provided
- Cannot be specified if `UseLatestRestorableTime` parameter is true
- Cannot be specified if `RestoreType` parameter is `copy-on-write`

Example: `2015-03-07T23:45:00Z`

- **RestoreType** (in the CLI: `--restore-type`, or in Python: `restore_type`) – UTF-8 string.

The type of restore to be performed. You can specify one of the following values:

- `full-copy` - The new DB cluster is restored as a full copy of the source DB cluster.
- `copy-on-write` - The new DB cluster is restored as a clone of the source DB cluster.

If you don't specify a `RestoreType` value, then the new DB cluster is restored as a full copy of the source DB cluster.

- **SourceDBClusterIdentifier** (in the CLI: `--source-db-cluster-identifier`, or in Python: `source_db_cluster_identifier`) – *Required:* UTF-8 string.

The identifier of the source DB cluster from which to restore.

Constraints:

- Must match the identifier of an existing DBCluster.
- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be applied to the restored DB cluster.

- **UseLatestRestorableTime** (in the CLI: `--use-latest-restorable-time`, or in Python: `use_latest_restorable_time`) – Boolean.

A value that is set to `true` to restore the DB cluster to the latest restorable backup time, and `false` otherwise.

Default: `false`

Constraints: Cannot be specified if `RestoreToTime` parameter is provided.

- **VpcSecurityGroupIds** (in the CLI: `--vpc-security-group-ids`, or in Python: `vpc_security_group_ids`) – An array of `VpcSecurityGroupId`— UTF-8 strings.

A list of VPC security groups that the new DB cluster belongs to.

Response

Contains the details of an Amazon Neptune DB cluster.

This data type is used as a response element in the [the section called "DescribeDBClusters" \(p. 826\)](#) action.

- **AllocatedStorage** (in the CLI: `--allocated-storage`, or in Python: `allocated_storage`) – Number (integer).

`AllocatedStorage` always returns 1, because Neptune DB cluster storage size is not fixed, but instead automatically adjusts as needed.

- **AssociatedRoles** (in the CLI: `--associated-roles`, or in Python: `associated_roles`) – An array of [DBClusterRole \(p. 792\)](#) `DBClusterRole` objects.

Provides a list of the Amazon Identity and Access Management (IAM) roles that are associated with the DB cluster. IAM roles that are associated with a DB cluster grant permission for the DB cluster to access other Amazon services on your behalf.

- **AutomaticRestartTime** (in the CLI: `--automatic-restart-time`, or in Python: `automatic_restart_time`) – Timestamp.

Time at which the DB cluster will be automatically restarted.

- **AvailabilityZones** (in the CLI: `--availability-zones`, or in Python: `availability_zones`) – An array of `AvailabilityZone`— UTF-8 strings.

Provides the list of EC2 Availability Zones that instances in the DB cluster can be created in.

- **BacktrackConsumedChangeRecords** (in the CLI: `--backtrack-consumed-change-records`, or in Python: `backtrack_consumed_change_records`) – Number (long).

Not supported by Neptune.

- **BacktrackWindow** (in the CLI: `--backtrack-window`, or in Python: `backtrack_window`) – Number (long).

Not supported by Neptune.

- **BackupRetentionPeriod** (in the CLI: `--backup-retention-period`, or in Python: `backup_retention_period`) – Number (integer).

Specifies the number of days for which automatic DB snapshots are retained.

- **Capacity** (in the CLI: `--capacity`, or in Python: `capacity`) – Number (integer).

Not supported by Neptune.

- **CloneGroupId** (in the CLI: `--clone-group-id`, or in Python: `clone_group_id`) – UTF-8 string.

Identifies the clone group to which the DB cluster is associated.

- **ClusterCreateTime** (in the CLI: `--cluster-create-time`, or in Python: `cluster_create_time`) – Timestamp.

Specifies the time when the DB cluster was created, in Universal Coordinated Time (UTC).

- **CopyTagsToSnapshot** (in the CLI: `--copy-tags-to-snapshot`, or in Python: `copy_tags_to_snapshot`) – Boolean.

If set to true, tags are copied to any snapshot of the DB cluster that is created.

- **CrossAccountClone** (in the CLI: `--cross-account-clone`, or in Python: `cross_account_clone`) – Boolean.

If set to true, the DB cluster can be cloned across accounts.

- **DatabaseName** (in the CLI: `--database-name`, or in Python: `database_name`) – UTF-8 string.

Contains the name of the initial database of this DB cluster that was provided at create time, if one was specified when the DB cluster was created. This same name is returned for the life of the DB cluster.

- **DBClusterArn** (in the CLI: `--db-cluster-arn`, or in Python: `db_cluster_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the DB cluster.

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

Contains a user-supplied DB cluster identifier. This identifier is the unique key that identifies a DB cluster.

- **DBClusterMembers** (in the CLI: `--db-cluster-members`, or in Python: `db_cluster_members`) – An array of [DBClusterMember](#) (p. 792) `DBClusterMember` objects.

Provides the list of instances that make up the DB cluster.

- **DBClusterParameterGroup** (in the CLI: `--db-cluster-parameter-group`, or in Python: `db_cluster_parameter_group`) – UTF-8 string.

Specifies the name of the DB cluster parameter group for the DB cluster.

- **DbClusterResourceId** (in the CLI: `--db-cluster-resource-id`, or in Python: `db_cluster_resource_id`) – UTF-8 string.

The Amazon Region-unique, immutable identifier for the DB cluster. This identifier is found in Amazon CloudTrail log entries whenever the Amazon KMS key for the DB cluster is accessed.

- **DBSubnetGroup** (in the CLI: `--db-subnet-group`, or in Python: `db_subnet_group`) – UTF-8 string.
Specifies information on the subnet group associated with the DB cluster, including the name, description, and subnets in the subnet group.
- **DeletionProtection** (in the CLI: `--deletion-protection`, or in Python: `deletion_protection`) – Boolean.
Indicates whether or not the DB cluster has deletion protection enabled. The database can't be deleted when deletion protection is enabled.
- **EarliestBacktrackTime** (in the CLI: `--earliest-backtrack-time`, or in Python: `earliest_backtrack_time`) – Timestamp.
Not supported by Neptune.
- **EarliestRestorableTime** (in the CLI: `--earliest-restorable-time`, or in Python: `earliest_restorable_time`) – Timestamp.
Specifies the earliest time to which a database can be restored with point-in-time restore.
- **EnabledCloudwatchLogsExports** (in the CLI: `--enabled-cloudwatch-logs-exports`, or in Python: `enabled_cloudwatch_logs_exports`) – An array of UTF-8 strings.
A list of log types that this DB cluster is configured to export to CloudWatch Logs.
- **Endpoint** (in the CLI: `--endpoint`, or in Python: `endpoint`) – UTF-8 string.
Specifies the connection endpoint for the primary instance of the DB cluster.
- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.
Provides the name of the database engine to be used for this DB cluster.
- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.
Indicates the database engine version.
- **HostedZoneId** (in the CLI: `--hosted-zone-id`, or in Python: `hosted_zone_id`) – UTF-8 string.
Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.
- **IAMDatabaseAuthenticationEnabled** (in the CLI: `--iam-database-authentication-enabled`, or in Python: `iam_database_authentication_enabled`) – Boolean.
True if mapping of Amazon Identity and Access Management (IAM) accounts to database accounts is enabled, and otherwise false.
- **KmsKeyId** (in the CLI: `--kms-key-id`, or in Python: `kms_key_id`) – UTF-8 string.
If `StorageEncrypted` is true, the Amazon KMS key identifier for the encrypted DB cluster.
- **LatestRestorableTime** (in the CLI: `--latest-restorable-time`, or in Python: `latest_restorable_time`) – Timestamp.
Specifies the latest time to which a database can be restored with point-in-time restore.
- **MultiAZ** (in the CLI: `--multi-az`, or in Python: `multi_az`) – Boolean.
Specifies whether the DB cluster has instances in multiple Availability Zones.
- **PendingModifiedValues** (in the CLI: `--pending-modified-values`, or in Python: `pending_modified_values`) – A [ClusterPendingModifiedValues](#) (p. 792) object.
Not supported by Neptune.
- **PercentProgress** (in the CLI: `--percent-progress`, or in Python: `percent_progress`) – UTF-8 string.
Specifies the progress of the operation as a percentage.

- **Port** (in the CLI: `--port`, or in Python: `port`) – Number (integer).
Specifies the port that the database engine is listening on.
- **PreferredBackupWindow** (in the CLI: `--preferred-backup-window`, or in Python: `preferred_backup_window`) – UTF-8 string.
Specifies the daily time range during which automated backups are created if automated backups are enabled, as determined by the `BackupRetentionPeriod`.
- **PreferredMaintenanceWindow** (in the CLI: `--preferred-maintenance-window`, or in Python: `preferred_maintenance_window`) – UTF-8 string.
Specifies the weekly time range during which system maintenance can occur, in Universal Coordinated Time (UTC).
- **ReaderEndpoint** (in the CLI: `--reader-endpoint`, or in Python: `reader_endpoint`) – UTF-8 string.
The reader endpoint for the DB cluster. The reader endpoint for a DB cluster load-balances connections across the Read Replicas that are available in a DB cluster. As clients request new connections to the reader endpoint, Neptune distributes the connection requests among the Read Replicas in the DB cluster. This functionality can help balance your read workload across multiple Read Replicas in your DB cluster.

If a failover occurs, and the Read Replica that you are connected to is promoted to be the primary instance, your connection is dropped. To continue sending your read workload to other Read Replicas in the cluster, you can then reconnect to the reader endpoint.
- **ReadReplicaIdentifiers** (in the CLI: `--read-replica-identifiers`, or in Python: `read_replica_identifiers`) – An array of `ReadReplicaIdentifier`— UTF-8 strings.
Contains one or more identifiers of the Read Replicas associated with this DB cluster.
- **ReplicationSourceIdentifier** (in the CLI: `--replication-source-identifier`, or in Python: `replication_source_identifier`) – UTF-8 string.
Not supported by Neptune.
- **ReplicationType** (in the CLI: `--replication-type`, or in Python: `replication_type`) – UTF-8 string.
Not supported by Neptune.
- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.
Specifies the current state of this DB cluster.
- **StorageEncrypted** (in the CLI: `--storage-encrypted`, or in Python: `storage_encrypted`) – Boolean.
Specifies whether the DB cluster is encrypted.
- **VpcSecurityGroups** (in the CLI: `--vpc-security-groups`, or in Python: `vpc_security_groups`) – An array of [VpcSecurityGroupMembership](#) (p. 937) `VpcSecurityGroupMembership` objects.
Provides a list of VPC security groups that the DB cluster belongs to.

Errors

- [DBClusterAlreadyExistsFault](#) (p. 940)
- [DBClusterNotFoundFault](#) (p. 940)
- [DBClusterQuotaExceededFault](#) (p. 941)
- [DBClusterSnapshotNotFoundFault](#) (p. 942)
- [DBSubnetGroupNotFoundFault](#) (p. 946)

- [InsufficientDBClusterCapacityFault](#) (p. 947)
- [InsufficientStorageClusterCapacityFault](#) (p. 948)
- [InvalidDBClusterSnapshotStateFault](#) (p. 948)
- [InvalidDBClusterStateFault](#) (p. 948)
- [InvalidDBSnapshotStateFault](#) (p. 949)
- [InvalidRestoreFault](#) (p. 950)
- [InvalidSubnet](#) (p. 950)
- [InvalidVPCNetworkStateFault](#) (p. 951)
- [KMSKeyNotAccessibleFault](#) (p. 951)
- [OptionGroupNotFoundFault](#) (p. 951)
- [StorageQuotaExceededFault](#) (p. 953)
- [DBClusterParameterGroupNotFoundFault](#) (p. 941)

DescribeDBClusterSnapshots (action)

The Amazon CLI name for this API is: `describe-db-cluster-snapshots`, and the Python name is: `describe_db_cluster_snapshots`.

Returns information about DB cluster snapshots. This API action supports pagination.

Request

- **DBClusterIdentifier** (in the CLI: `--db-cluster-identifier`, or in Python: `db_cluster_identifier`) – UTF-8 string.

The ID of the DB cluster to retrieve the list of DB cluster snapshots for. This parameter can't be used in conjunction with the `DBClusterSnapshotIdentifier` parameter. This parameter is not case-sensitive.

Constraints:

- If supplied, must match the identifier of an existing DBCluster.
- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – UTF-8 string.

A specific DB cluster snapshot identifier to describe. This parameter can't be used in conjunction with the `DBClusterIdentifier` parameter. This value is stored as a lowercase string.

Constraints:

- If supplied, must match the identifier of an existing DBClusterSnapshot.
- If this identifier is for an automated snapshot, the `SnapshotType` parameter must also be specified.
- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter](#) (p. 937) Filter objects.

This parameter is not currently supported.

- **IncludePublic** (in the CLI: `--include-public`, or in Python: `include_public`) – Boolean.

True to include manual DB cluster snapshots that are public and can be copied or restored by any Amazon account, and otherwise false. The default is `false`. The default is false.

You can share a manual DB cluster snapshot as public by using the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

- **IncludeShared** (in the CLI: `--include-shared`, or in Python: `include_shared`) – Boolean.

True to include shared manual DB cluster snapshots from other Amazon accounts that this Amazon account has been given permission to copy or restore, and otherwise false. The default is `false`.

You can give an Amazon account permission to restore a manual DB cluster snapshot from another Amazon account by the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribeDBClusterSnapshots` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

- **SnapshotType** (in the CLI: `--snapshot-type`, or in Python: `snapshot_type`) – UTF-8 string.

The type of DB cluster snapshots to be returned. You can specify one of the following values:

- `automated` - Return all DB cluster snapshots that have been automatically taken by Amazon Neptune for my Amazon account.
- `manual` - Return all DB cluster snapshots that have been taken by my Amazon account.
- `shared` - Return all manual DB cluster snapshots that have been shared to my Amazon account.
- `public` - Return all DB cluster snapshots that have been marked as public.

If you don't specify a `SnapshotType` value, then both `automated` and `manual` DB cluster snapshots are returned. You can include shared DB cluster snapshots with these results by setting the `IncludeShared` parameter to `true`. You can include public DB cluster snapshots with these results by setting the `IncludePublic` parameter to `true`.

The `IncludeShared` and `IncludePublic` parameters don't apply for `SnapshotType` values of `manual` or `automated`. The `IncludePublic` parameter doesn't apply when `SnapshotType` is set to `shared`. The `IncludeShared` parameter doesn't apply when `SnapshotType` is set to `public`.

Response

- **DBClusterSnapshots** (in the CLI: `--db-cluster-snapshots`, or in Python: `db_cluster_snapshots`) – An array of [DBClusterSnapshot \(p. 887\)](#) `DBClusterSnapshot` objects.

Provides a list of DB cluster snapshots for the user.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous [the section called "DescribeDBClusterSnapshots" \(p. 910\)](#) request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Errors

- [DBClusterSnapshotNotFoundFault \(p. 942\)](#)

DescribeDBClusterSnapshotAttributes (action)

The Amazon CLI name for this API is: `describe-db-cluster-snapshot-attributes`, and the Python name is: `describe_db_cluster_snapshot_attributes`.

Returns a list of DB cluster snapshot attribute names and values for a manual DB cluster snapshot.

When sharing snapshots with other Amazon accounts, `DescribeDBClusterSnapshotAttributes` returns the `restore` attribute and a list of IDs for the Amazon accounts that are authorized to copy or restore the manual DB cluster snapshot. If `all` is included in the list of values for the `restore` attribute, then the manual DB cluster snapshot is public and can be copied or restored by all Amazon accounts.

To add or remove access for an Amazon account to copy or restore a manual DB cluster snapshot, or to make the manual DB cluster snapshot public or private, use the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

Request

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – *Required*: UTF-8 string.

The identifier for the DB cluster snapshot to describe the attributes for.

Response

Contains the results of a successful call to the [the section called "DescribeDBClusterSnapshotAttributes" \(p. 912\)](#) API action.

Manual DB cluster snapshot attributes are used to authorize other Amazon accounts to copy or restore a manual DB cluster snapshot. For more information, see the [the section called "ModifyDBClusterSnapshotAttribute" \(p. 896\)](#) API action.

- **DBClusterSnapshotAttributes** (in the CLI: `--db-cluster-snapshot-attributes`, or in Python: `db_cluster_snapshot_attributes`) – An array of [DBClusterSnapshotAttribute \(p. 888\)](#) `DBClusterSnapshotAttribute` objects.

The list of attributes and values for the manual DB cluster snapshot.

- **DBClusterSnapshotIdentifier** (in the CLI: `--db-cluster-snapshot-identifier`, or in Python: `db_cluster_snapshot_identifier`) – UTF-8 string.

The identifier of the manual DB cluster snapshot that the attributes apply to.

Errors

- [DBClusterSnapshotNotFoundFault \(p. 942\)](#)

Neptune Events API

Structures:

- [Event \(structure\) \(p. 913\)](#)
- [EventCategoriesMap \(structure\) \(p. 913\)](#)
- [EventSubscription \(structure\) \(p. 914\)](#)

Actions:

- [CreateEventSubscription \(action\) \(p. 915\)](#)
- [DeleteEventSubscription \(action\) \(p. 917\)](#)
- [ModifyEventSubscription \(action\) \(p. 918\)](#)
- [DescribeEventSubscriptions \(action\) \(p. 920\)](#)
- [AddSourceIdentifierToSubscription \(action\) \(p. 921\)](#)
- [RemoveSourceIdentifierFromSubscription \(action\) \(p. 922\)](#)
- [DescribeEvents \(action\) \(p. 923\)](#)
- [DescribeEventCategories \(action\) \(p. 925\)](#)

Event (structure)

This data type is used as a response element in the [the section called "DescribeEvents" \(p. 923\)](#) action.

Fields

- **Date** – Timestamp.
Specifies the date and time of the event.
- **EventCategories** – An array of EventCategory— UTF-8 strings.
Specifies the category for the event.
- **Message** – UTF-8 string.
Provides the text of this event.
- **SourceArn** – UTF-8 string.
The Amazon Resource Name (ARN) for the event.
- **SourceIdentifier** – UTF-8 string.
Provides the identifier for the source of the event.
- **SourceType** – UTF-8 string (valid values: `db-instance=""` | `db-parameter-group=""` | `db-security-group=""` | `db-snapshot=""` | `db-cluster=""` | `db-cluster-snapshot=""`).
Specifies the source type for this event.

EventCategoriesMap (structure)

Contains the results of a successful invocation of the [the section called "DescribeEventCategories" \(p. 925\)](#) action.

Fields

- **EventCategories** – An array of EventCategory— UTF-8 strings.
The event categories for the specified source type
- **SourceType** – UTF-8 string.
The source type that the returned categories belong to

EventSubscription (structure)

Contains the results of a successful invocation of the [the section called "DescribeEventSubscriptions" \(p. 920\)](#) action.

Fields

- **CustomerAwsId** – UTF-8 string.

The Amazon customer account associated with the event notification subscription.

- **CustSubscriptionId** – UTF-8 string.

The event notification subscription Id.

- **Enabled** – Boolean.

A Boolean value indicating if the subscription is enabled. True indicates the subscription is enabled.

- **EventCategoriesList** – An array of EventCategory— UTF-8 strings.

A list of event categories for the event notification subscription.

- **EventSubscriptionArn** – UTF-8 string.

The Amazon Resource Name (ARN) for the event subscription.

- **SnsTopicArn** – UTF-8 string.

The topic ARN of the event notification subscription.

- **SourceIdsList** – An array of SourceId— UTF-8 strings.

A list of source IDs for the event notification subscription.

- **SourceType** – UTF-8 string.

The source type for the event notification subscription.

- **Status** – UTF-8 string.

The status of the event notification subscription.

Constraints:

Can be one of the following: creating | modifying | deleting | active | no-permission | topic-not-exist

The status "no-permission" indicates that Neptune no longer has permission to post to the SNS topic. The status "topic-not-exist" indicates that the topic was deleted after the subscription was created.

- **SubscriptionCreationTime** – UTF-8 string.

The time the event notification subscription was created.

EventSubscription is used as the response element for:

- [CreateEventSubscription \(p. 915\)](#)
- [ModifyEventSubscription \(p. 918\)](#)
- [AddSourceIdentifierToSubscription \(p. 921\)](#)
- [RemoveSourceIdentifierFromSubscription \(p. 922\)](#)
- [DeleteEventSubscription \(p. 917\)](#)

CreateEventSubscription (action)

The Amazon CLI name for this API is: `create-event-subscription`, and the Python name is: `create_event_subscription`.

Creates an event notification subscription. This action requires a topic ARN (Amazon Resource Name) created by either the Neptune console, the SNS console, or the SNS API. To obtain an ARN with SNS, you must create a topic in Amazon SNS and subscribe to the topic. The ARN is displayed in the SNS console.

You can specify the type of source (`SourceType`) you want to be notified of, provide a list of Neptune sources (`SourceIds`) that triggers the events, and provide a list of event categories (`EventCategories`) for events you want to be notified of. For example, you can specify `SourceType = db-instance`, `SourceIds = mydbinstance1, mydbinstance2` and `EventCategories = Availability, Backup`.

If you specify both the `SourceType` and `SourceIds`, such as `SourceType = db-instance` and `SourceIdentifier = myDBInstance1`, you are notified of all the `db-instance` events for the specified source. If you specify a `SourceType` but do not specify a `SourceIdentifier`, you receive notice of the events for that source type for all your Neptune sources. If you do not specify either the `SourceType` nor the `SourceIdentifier`, you are notified of events generated from all Neptune sources belonging to your customer account.

Request

- **Enabled** (in the CLI: `--enabled`, or in Python: `enabled`) – Boolean.

A Boolean value; set to **true** to activate the subscription, set to **false** to create the subscription but not active it.

- **EventCategories** (in the CLI: `--event-categories`, or in Python: `event_categories`) – An array of `EventCategory`— UTF-8 strings.

A list of event categories for a `SourceType` that you want to subscribe to. You can see a list of the categories for a given `SourceType` by using the **DescribeEventCategories** action.

- **SnsTopicArn** (in the CLI: `--sns-topic-arn`, or in Python: `sns_topic_arn`) – *Required*: UTF-8 string.

The Amazon Resource Name (ARN) of the SNS topic created for event notification. The ARN is created by Amazon SNS when you create a topic and subscribe to it.

- **SourceIds** (in the CLI: `--source-ids`, or in Python: `source_ids`) – An array of `SourceId`— UTF-8 strings.

The list of identifiers of the event sources for which events are returned. If not specified, then all sources are included in the response. An identifier must begin with a letter and must contain only ASCII letters, digits, and hyphens; it can't end with a hyphen or contain two consecutive hyphens.

Constraints:

- If `SourceIds` are supplied, `SourceType` must also be provided.
- If the source type is a DB instance, then a `DBInstanceIdentifier` must be supplied.
- If the source type is a DB security group, a `DBSecurityGroupName` must be supplied.
- If the source type is a DB parameter group, a `DBParameterGroupName` must be supplied.
- If the source type is a DB snapshot, a `DBSnapshotIdentifier` must be supplied.
- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.

The type of source that is generating the events. For example, if you want to be notified of events generated by a DB instance, you would set this parameter to `db-instance`. If this value is not specified, all events are returned.

Valid values: `db-instance` | `db-cluster` | `db-parameter-group` | `db-security-group` | `db-snapshot` | `db-cluster-snapshot`

- **SubscriptionName** (in the CLI: `--subscription-name`, or in Python: `subscription_name`) – *Required*: UTF-8 string.

The name of the subscription.

Constraints: The name must be less than 255 characters.

- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – An array of [Tag \(p. 930\)](#) Tag objects.

The tags to be applied to the new event subscription.

Response

Contains the results of a successful invocation of the [the section called “DescribeEventSubscriptions” \(p. 920\)](#) action.

- **CustomerAwsId** (in the CLI: `--customer-aws-id`, or in Python: `customer_aws_id`) – UTF-8 string.

The Amazon customer account associated with the event notification subscription.

- **CustSubscriptionId** (in the CLI: `--cust-subscription-id`, or in Python: `cust_subscription_id`) – UTF-8 string.

The event notification subscription Id.

- **Enabled** (in the CLI: `--enabled`, or in Python: `enabled`) – Boolean.

A Boolean value indicating if the subscription is enabled. True indicates the subscription is enabled.

- **EventCategoriesList** (in the CLI: `--event-categories-list`, or in Python: `event_categories_list`) – An array of `EventCategory`— UTF-8 strings.

A list of event categories for the event notification subscription.

- **EventSubscriptionArn** (in the CLI: `--event-subscription-arn`, or in Python: `event_subscription_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the event subscription.

- **SnsTopicArn** (in the CLI: `--sns-topic-arn`, or in Python: `sns_topic_arn`) – UTF-8 string.

The topic ARN of the event notification subscription.

- **SourceIdsList** (in the CLI: `--source-ids-list`, or in Python: `source_ids_list`) – An array of `SourceId`— UTF-8 strings.

A list of source IDs for the event notification subscription.

- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.

The source type for the event notification subscription.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

The status of the event notification subscription.

Constraints:

Can be one of the following: `creating` | `modifying` | `deleting` | `active` | `no-permission` | `topic-not-exist`

The status "no-permission" indicates that Neptune no longer has permission to post to the SNS topic. The status "topic-not-exist" indicates that the topic was deleted after the subscription was created.

- **SubscriptionCreationTime** (in the CLI: `--subscription-creation-time`, or in Python: `subscription_creation_time`) – UTF-8 string.

The time the event notification subscription was created.

Errors

- [EventSubscriptionQuotaExceededFault](#) (p. 947)
- [SubscriptionAlreadyExistFault](#) (p. 954)
- [SNSInvalidTopicFault](#) (p. 952)
- [SNSNoAuthorizationFault](#) (p. 952)
- [SNSTopicArnNotFoundFault](#) (p. 952)
- [SubscriptionCategoryNotFoundFault](#) (p. 954)
- [SourceNotFoundFault](#) (p. 953)

DeleteEventSubscription (action)

The Amazon CLI name for this API is: `delete-event-subscription`, and the Python name is: `delete_event_subscription`.

Deletes an event notification subscription.

Request

- **SubscriptionName** (in the CLI: `--subscription-name`, or in Python: `subscription_name`) – *Required*: UTF-8 string.

The name of the event notification subscription you want to delete.

Response

Contains the results of a successful invocation of the [the section called “DescribeEventSubscriptions”](#) (p. 920) action.

- **CustomerAwsId** (in the CLI: `--customer-aws-id`, or in Python: `customer_aws_id`) – UTF-8 string.

The Amazon customer account associated with the event notification subscription.

- **CustSubscriptionId** (in the CLI: `--cust-subscription-id`, or in Python: `cust_subscription_id`) – UTF-8 string.

The event notification subscription Id.

- **Enabled** (in the CLI: `--enabled`, or in Python: `enabled`) – Boolean.

A Boolean value indicating if the subscription is enabled. True indicates the subscription is enabled.

- **EventCategoriesList** (in the CLI: `--event-categories-list`, or in Python: `event_categories_list`) – An array of `EventCategory`— UTF-8 strings.

A list of event categories for the event notification subscription.

- **EventSubscriptionArn** (in the CLI: `--event-subscription-arn`, or in Python: `event_subscription_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the event subscription.

- **SnsTopicArn** (in the CLI: `--sns-topic-arn`, or in Python: `sns_topic_arn`) – UTF-8 string.

The topic ARN of the event notification subscription.

- **SourceIdsList** (in the CLI: `--source-ids-list`, or in Python: `source_ids_list`) – An array of `SourceId`— UTF-8 strings.

A list of source IDs for the event notification subscription.

- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.
The source type for the event notification subscription.
- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.
The status of the event notification subscription.
Constraints:
Can be one of the following: `creating` | `modifying` | `deleting` | `active` | `no-permission` | `topic-not-exist`
The status "no-permission" indicates that Neptune no longer has permission to post to the SNS topic.
The status "topic-not-exist" indicates that the topic was deleted after the subscription was created.
- **SubscriptionCreationTime** (in the CLI: `--subscription-creation-time`, or in Python: `subscription_creation_time`) – UTF-8 string.
The time the event notification subscription was created.

Errors

- [SubscriptionNotFoundFault](#) (p. 954)
- [InvalidEventSubscriptionStateFault](#) (p. 950)

ModifyEventSubscription (action)

The Amazon CLI name for this API is: `modify-event-subscription`, and the Python name is: `modify_event_subscription`.

Modifies an existing event notification subscription. Note that you can't modify the source identifiers using this call; to change source identifiers for a subscription, use the [section called "AddSourceIdentifierToSubscription"](#) (p. 921) and the [section called "RemoveSourceIdentifierFromSubscription"](#) (p. 922) calls.

You can see a list of the event categories for a given SourceType by using the **DescribeEventCategories** action.

Request

- **Enabled** (in the CLI: `--enabled`, or in Python: `enabled`) – Boolean.
A Boolean value; set to **true** to activate the subscription.
- **EventCategories** (in the CLI: `--event-categories`, or in Python: `event_categories`) – An array of EventCategory— UTF-8 strings.
A list of event categories for a SourceType that you want to subscribe to. You can see a list of the categories for a given SourceType by using the **DescribeEventCategories** action.
- **SnsTopicArn** (in the CLI: `--sns-topic-arn`, or in Python: `sns_topic_arn`) – UTF-8 string.
The Amazon Resource Name (ARN) of the SNS topic created for event notification. The ARN is created by Amazon SNS when you create a topic and subscribe to it.
- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.
The type of source that is generating the events. For example, if you want to be notified of events generated by a DB instance, you would set this parameter to `db-instance`. If this value is not specified, all events are returned.
Valid values: `db-instance` | `db-parameter-group` | `db-security-group` | `db-snapshot`

- **SubscriptionName** (in the CLI: `--subscription-name`, or in Python: `subscription_name`) – *Required:* UTF-8 string.

The name of the event notification subscription.

Response

Contains the results of a successful invocation of the [the section called “DescribeEventSubscriptions” \(p. 920\)](#) action.

- **CustomerAwsId** (in the CLI: `--customer-aws-id`, or in Python: `customer_aws_id`) – UTF-8 string.

The Amazon customer account associated with the event notification subscription.

- **CustSubscriptionId** (in the CLI: `--cust-subscription-id`, or in Python: `cust_subscription_id`) – UTF-8 string.

The event notification subscription Id.

- **Enabled** (in the CLI: `--enabled`, or in Python: `enabled`) – Boolean.

A Boolean value indicating if the subscription is enabled. True indicates the subscription is enabled.

- **EventCategoriesList** (in the CLI: `--event-categories-list`, or in Python: `event_categories_list`) – An array of EventCategory— UTF-8 strings.

A list of event categories for the event notification subscription.

- **EventSubscriptionArn** (in the CLI: `--event-subscription-arn`, or in Python: `event_subscription_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the event subscription.

- **SnsTopicArn** (in the CLI: `--sns-topic-arn`, or in Python: `sns_topic_arn`) – UTF-8 string.

The topic ARN of the event notification subscription.

- **SourceIdsList** (in the CLI: `--source-ids-list`, or in Python: `source_ids_list`) – An array of SourceId— UTF-8 strings.

A list of source IDs for the event notification subscription.

- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.

The source type for the event notification subscription.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

The status of the event notification subscription.

Constraints:

Can be one of the following: `creating` | `modifying` | `deleting` | `active` | `no-permission` | `topic-not-exist`

The status "no-permission" indicates that Neptune no longer has permission to post to the SNS topic. The status "topic-not-exist" indicates that the topic was deleted after the subscription was created.

- **SubscriptionCreationTime** (in the CLI: `--subscription-creation-time`, or in Python: `subscription_creation_time`) – UTF-8 string.

The time the event notification subscription was created.

Errors

- [EventSubscriptionQuotaExceededFault \(p. 947\)](#)

- [SubscriptionNotFoundFault \(p. 954\)](#)
- [SNSInvalidTopicFault \(p. 952\)](#)
- [SNSNoAuthorizationFault \(p. 952\)](#)
- [SNSTopicArnNotFoundFault \(p. 952\)](#)
- [SubscriptionCategoryNotFoundFault \(p. 954\)](#)

DescribeEventSubscriptions (action)

The Amazon CLI name for this API is: `describe-event-subscriptions`, and the Python name is: `describe_event_subscriptions`.

Lists all the subscription descriptions for a customer account. The description for a subscription includes SubscriptionName, SNSTopicARN, CustomerID, SourceType, SourceID, CreationTime, and Status.

If you specify a SubscriptionName, lists the description for that subscription.

Request

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous DescribeOrderableDBInstanceOptions request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

- **SubscriptionName** (in the CLI: `--subscription-name`, or in Python: `subscription_name`) – UTF-8 string.

The name of the event notification subscription you want to describe.

Response

- **EventSubscriptionsList** (in the CLI: `--event-subscriptions-list`, or in Python: `event_subscriptions_list`) – An array of [EventSubscription \(p. 914\)](#) EventSubscription objects.

A list of EventSubscriptions data types.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous DescribeOrderableDBInstanceOptions request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Errors

- [SubscriptionNotFoundFault \(p. 954\)](#)

AddSourceIdentifierToSubscription (action)

The Amazon CLI name for this API is: `add-source-identifier-to-subscription`, and the Python name is: `add_source_identifier_to_subscription`.

Adds a source identifier to an existing event notification subscription.

Request

- **SourceIdentifier** (in the CLI: `--source-identifier`, or in Python: `source_identifier`) – *Required:* UTF-8 string.

The identifier of the event source to be added.

Constraints:

- If the source type is a DB instance, then a `DBInstanceIdentifier` must be supplied.
- If the source type is a DB security group, a `DBSecurityGroupName` must be supplied.
- If the source type is a DB parameter group, a `DBParameterGroupName` must be supplied.
- If the source type is a DB snapshot, a `DBSnapshotIdentifier` must be supplied.
- **SubscriptionName** (in the CLI: `--subscription-name`, or in Python: `subscription_name`) – *Required:* UTF-8 string.

The name of the event notification subscription you want to add a source identifier to.

Response

Contains the results of a successful invocation of the [the section called “DescribeEventSubscriptions” \(p. 920\)](#) action.

- **CustomerAwsId** (in the CLI: `--customer-aws-id`, or in Python: `customer_aws_id`) – UTF-8 string.

The Amazon customer account associated with the event notification subscription.

- **CustSubscriptionId** (in the CLI: `--cust-subscription-id`, or in Python: `cust_subscription_id`) – UTF-8 string.

The event notification subscription Id.

- **Enabled** (in the CLI: `--enabled`, or in Python: `enabled`) – Boolean.

A Boolean value indicating if the subscription is enabled. True indicates the subscription is enabled.

- **EventCategoriesList** (in the CLI: `--event-categories-list`, or in Python: `event_categories_list`) – An array of `EventCategory`— UTF-8 strings.

A list of event categories for the event notification subscription.

- **EventSubscriptionArn** (in the CLI: `--event-subscription-arn`, or in Python: `event_subscription_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the event subscription.

- **SnsTopicArn** (in the CLI: `--sns-topic-arn`, or in Python: `sns_topic_arn`) – UTF-8 string.

The topic ARN of the event notification subscription.

- **SourceIdsList** (in the CLI: `--source-ids-list`, or in Python: `source_ids_list`) – An array of `SourceId`— UTF-8 strings.

A list of source IDs for the event notification subscription.

- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.

The source type for the event notification subscription.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

The status of the event notification subscription.

Constraints:

Can be one of the following: `creating` | `modifying` | `deleting` | `active` | `no-permission` | `topic-not-exist`

The status "no-permission" indicates that Neptune no longer has permission to post to the SNS topic. The status "topic-not-exist" indicates that the topic was deleted after the subscription was created.

- **SubscriptionCreationTime** (in the CLI: `--subscription-creation-time`, or in Python: `subscription_creation_time`) – UTF-8 string.

The time the event notification subscription was created.

Errors

- [SubscriptionNotFoundFault \(p. 954\)](#)
- [SourceNotFoundFault \(p. 953\)](#)

RemoveSourceIdentifierFromSubscription (action)

The Amazon CLI name for this API is: `remove-source-identifier-from-subscription`, and the Python name is: `remove_source_identifier_from_subscription`.

Removes a source identifier from an existing event notification subscription.

Request

- **SourceIdentifier** (in the CLI: `--source-identifier`, or in Python: `source_identifier`) – *Required:* UTF-8 string.

The source identifier to be removed from the subscription, such as the **DB instance identifier** for a DB instance or the name of a security group.

- **SubscriptionName** (in the CLI: `--subscription-name`, or in Python: `subscription_name`) – *Required:* UTF-8 string.

The name of the event notification subscription you want to remove a source identifier from.

Response

Contains the results of a successful invocation of the [the section called "DescribeEventSubscriptions" \(p. 920\)](#) action.

- **CustomerAwsId** (in the CLI: `--customer-aws-id`, or in Python: `customer_aws_id`) – UTF-8 string.

The Amazon customer account associated with the event notification subscription.

- **CustSubscriptionId** (in the CLI: `--cust-subscription-id`, or in Python: `cust_subscription_id`) – UTF-8 string.

The event notification subscription Id.

- **Enabled** (in the CLI: `--enabled`, or in Python: `enabled`) – Boolean.

A Boolean value indicating if the subscription is enabled. True indicates the subscription is enabled.

- **EventCategoriesList** (in the CLI: `--event-categories-list`, or in Python: `event_categories_list`) – An array of `EventCategory`— UTF-8 strings.

A list of event categories for the event notification subscription.

- **EventSubscriptionArn** (in the CLI: `--event-subscription-arn`, or in Python: `event_subscription_arn`) – UTF-8 string.

The Amazon Resource Name (ARN) for the event subscription.

- **SnsTopicArn** (in the CLI: `--sns-topic-arn`, or in Python: `sns_topic_arn`) – UTF-8 string.

The topic ARN of the event notification subscription.

- **SourceIdsList** (in the CLI: `--source-ids-list`, or in Python: `source_ids_list`) – An array of `SourceId`— UTF-8 strings.

A list of source IDs for the event notification subscription.

- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.

The source type for the event notification subscription.

- **Status** (in the CLI: `--status`, or in Python: `status`) – UTF-8 string.

The status of the event notification subscription.

Constraints:

Can be one of the following: `creating` | `modifying` | `deleting` | `active` | `no-permission` | `topic-not-exist`

The status "no-permission" indicates that Neptune no longer has permission to post to the SNS topic. The status "topic-not-exist" indicates that the topic was deleted after the subscription was created.

- **SubscriptionCreationTime** (in the CLI: `--subscription-creation-time`, or in Python: `subscription_creation_time`) – UTF-8 string.

The time the event notification subscription was created.

Errors

- [SubscriptionNotFoundFault \(p. 954\)](#)
- [SourceNotFoundFault \(p. 953\)](#)

DescribeEvents (action)

The Amazon CLI name for this API is: `describe-events`, and the Python name is: `describe_events`.

Returns events related to DB instances, DB security groups, DB snapshots, and DB parameter groups for the past 14 days. Events specific to a particular DB instance, DB security group, database snapshot, or DB parameter group can be obtained by providing the name as a parameter. By default, the past hour of events are returned.

Request

- **Duration** (in the CLI: `--duration`, or in Python: `duration`) – Number (integer).

The number of minutes to retrieve events for.

Default: 60

- **EndTime** (in the CLI: `--end-time`, or in Python: `end_time`) – Timestamp.

The end of the time interval for which to retrieve events, specified in ISO 8601 format. For more information about ISO 8601, go to the [ISO8601 Wikipedia page](#).

Example: 2009-07-08T18:00Z

- **EventCategories** (in the CLI: `--event-categories`, or in Python: `event_categories`) – An array of EventCategory— UTF-8 strings.

A list of event categories that trigger notifications for a event notification subscription.

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous DescribeEvents request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

- **SourceIdentifier** (in the CLI: `--source-identifier`, or in Python: `source_identifier`) – UTF-8 string.

The identifier of the event source for which events are returned. If not specified, then all sources are included in the response.

Constraints:

- If `SourceIdentifier` is supplied, `SourceType` must also be provided.
- If the source type is `DBInstance`, then a `DBInstanceIdentifier` must be supplied.
- If the source type is `DBSecurityGroup`, a `DBSecurityGroupName` must be supplied.
- If the source type is `DBParameterGroup`, a `DBParameterGroupName` must be supplied.
- If the source type is `DBSnapshot`, a `DBSnapshotIdentifier` must be supplied.
- Cannot end with a hyphen or contain two consecutive hyphens.
- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string (valid values: `db-instance=""` | `db-parameter-group=""` | `db-security-group=""` | `db-snapshot=""` | `db-cluster=""` | `db-cluster-snapshot=""`).

The event source to retrieve events for. If no value is specified, all events are returned.

- **StartTime** (in the CLI: `--start-time`, or in Python: `start_time`) – Timestamp.

The beginning of the time interval to retrieve events for, specified in ISO 8601 format. For more information about ISO 8601, go to the [ISO8601 Wikipedia page](#).

Example: 2009-07-08T18:00Z

Response

- **Events** (in the CLI: `--events`, or in Python: `events`) – An array of [Event \(p. 913\)](#) Event objects.

A list of [the section called "Event" \(p. 913\)](#) instances.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous Events request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords` .

DescribeEventCategories (action)

The Amazon CLI name for this API is: `describe-event-categories`, and the Python name is: `describe_event_categories`.

Displays a list of categories for all event source types, or, if specified, for a specified source type.

Request

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter \(p. 937\)](#) Filter objects.

This parameter is not currently supported.

- **SourceType** (in the CLI: `--source-type`, or in Python: `source_type`) – UTF-8 string.

The type of source that is generating the events.

Valid values: `db-instance` | `db-parameter-group` | `db-security-group` | `db-snapshot`

Response

- **EventCategoriesMapList** (in the CLI: `--event-categories-map-list`, or in Python: `event_categories_map_list`) – An array of [EventCategoriesMap \(p. 913\)](#) EventCategoriesMap objects.

A list of EventCategoriesMap data types.

Other Neptune APIs

Structures:

- [CloudwatchLogsExportConfiguration \(structure\) \(p. 926\)](#)
- [DBEngineVersion \(structure\) \(p. 926\)](#)
- [EngineDefaults \(structure\) \(p. 927\)](#)
- [PendingMaintenanceAction \(structure\) \(p. 927\)](#)
- [PendingModifiedValues \(structure\) \(p. 928\)](#)
- [PendingCloudwatchLogsExports \(structure\) \(p. 929\)](#)
- [ResourcePendingMaintenanceActions \(structure\) \(p. 929\)](#)
- [UpgradeTarget \(structure\) \(p. 929\)](#)
- [Tag \(structure\) \(p. 930\)](#)

Actions:

- [AddTagsToResource \(action\) \(p. 930\)](#)

- [ListTagsForResource \(action\)](#) (p. 931)
- [RemoveTagsFromResource \(action\)](#) (p. 931)
- [ApplyPendingMaintenanceAction \(action\)](#) (p. 932)
- [DescribePendingMaintenanceActions \(action\)](#) (p. 933)
- [DescribeDBEngineVersions \(action\)](#) (p. 934)

CloudwatchLogsExportConfiguration (structure)

The configuration setting for the log types to be enabled for export to CloudWatch Logs for a specific DB instance or DB cluster.

The `EnableLogTypes` and `DisableLogTypes` arrays determine which logs will be exported (or not exported) to CloudWatch Logs.

Fields

- **DisableLogTypes** – An array of UTF-8 strings.
The list of log types to disable.
- **EnableLogTypes** – An array of UTF-8 strings.
The list of log types to enable.

DBEngineVersion (structure)

This data type is used as a response element in the action [the section called "DescribeDBEngineVersions"](#) (p. 934).

Fields

- **DBEngineDescription** – UTF-8 string.
The description of the database engine.
- **DBEngineVersionDescription** – UTF-8 string.
The description of the database engine version.
- **DBParameterGroupFamily** – UTF-8 string.
The name of the DB parameter group family for the database engine.
- **Engine** – UTF-8 string.
The name of the database engine.
- **EngineVersion** – UTF-8 string.
The version number of the database engine.
- **ExportableLogTypes** – An array of UTF-8 strings.
The types of logs that the database engine has available for export to CloudWatch Logs.
- **SupportedTimezones** – An array of [Timezone](#) (p. 937) Timezone objects.
A list of the time zones supported by this engine for the `Timezone` parameter of the `CreateDBInstance` action.

- **SupportsLogExportsToCloudwatchLogs** – Boolean.

A value that indicates whether the engine version supports exporting the log types specified by `ExportableLogTypes` to CloudWatch Logs.

- **SupportsReadReplica** – Boolean.

Indicates whether the database engine version supports read replicas.

- **ValidUpgradeTarget** – An array of [UpgradeTarget](#) (p. 929) `UpgradeTarget` objects.

A list of engine versions that this database engine version can be upgraded to.

EngineDefaults (structure)

Contains the result of a successful invocation of the [the section called “DescribeEngineDefaultParameters”](#) (p. 879) action.

Fields

- **DBParameterGroupFamily** – UTF-8 string.

Specifies the name of the DB parameter group family that the engine default parameters apply to.

- **Marker** – UTF-8 string.

An optional pagination token provided by a previous `EngineDefaults` request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **Parameters** – An array of [Parameter](#) (p. 863) `Parameter` objects.

Contains a list of engine default parameters.

`EngineDefaults` is used as the response element for:

- [DescribeEngineDefaultParameters](#) (p. 879)
- [DescribeEngineDefaultClusterParameters](#) (p. 880)

PendingMaintenanceAction (structure)

Provides information about a pending maintenance action for a resource.

Fields

- **Action** – UTF-8 string.

The type of pending maintenance action that is available for the resource.

- **AutoAppliedAfterDate** – Timestamp.

The date of the maintenance window when the action is applied. The maintenance action is applied to the resource during its first maintenance window after this date. If this date is specified, any `next-maintenance` opt-in requests are ignored.

- **CurrentApplyDate** – Timestamp.

The effective date when the pending maintenance action is applied to the resource. This date takes into account opt-in requests received from the [the section called](#)

“[ApplyPendingMaintenanceAction](#)” (p. 932) API, the `AutoAppliedAfterDate`, and the `ForcedApplyDate`. This value is blank if an opt-in request has not been received and nothing has been specified as `AutoAppliedAfterDate` or `ForcedApplyDate`.

- **Description** – UTF-8 string.

A description providing more detail about the maintenance action.

- **ForcedApplyDate** – Timestamp.

The date when the maintenance action is automatically applied. The maintenance action is applied to the resource on this date regardless of the maintenance window for the resource. If this date is specified, any `immediate` opt-in requests are ignored.

- **OptInStatus** – UTF-8 string.

Indicates the type of opt-in request that has been received for the resource.

PendingModifiedValues (structure)

This data type is used as a response element in the [the section called “ModifyDBInstance”](#) (p. 846) action.

Fields

- **AllocatedStorage** – Number (integer).

Contains the new `AllocatedStorage` size for the DB instance that will be applied or is currently being applied.

- **BackupRetentionPeriod** – Number (integer).

Specifies the pending number of days for which automated backups are retained.

- **CACertificateIdentifier** – UTF-8 string.

Specifies the identifier of the CA certificate for the DB instance.

- **DBInstanceClass** – UTF-8 string.

Contains the new `DBInstanceClass` for the DB instance that will be applied or is currently being applied.

- **DBInstanceIdentifier** – UTF-8 string.

Contains the new `DBInstanceIdentifier` for the DB instance that will be applied or is currently being applied.

- **DBSubnetGroupName** – UTF-8 string.

The new DB subnet group for the DB instance.

- **EngineVersion** – UTF-8 string.

Indicates the database engine version.

- **Iops** – Number (integer).

Specifies the new `Provisioned IOPS` value for the DB instance that will be applied or is currently being applied.

- **MultiAZ** – Boolean.

Indicates that the Single-AZ DB instance is to change to a Multi-AZ deployment.

- **PendingCloudwatchLogsExports** – A [PendingCloudwatchLogsExports](#) (p. 929) object.

This `PendingCloudwatchLogsExports` structure specifies pending changes to which CloudWatch logs are enabled and which are disabled.

- **Port** – Number (integer).

Specifies the pending port for the DB instance.

- **StorageType** – UTF-8 string.

Specifies the storage type to be associated with the DB instance.

PendingCloudwatchLogsExports (structure)

A list of the log types whose configuration is still pending. In other words, these log types are in the process of being activated or deactivated.

Fields

- **LogTypesToDisable** – An array of UTF-8 strings.

Log types that are in the process of being enabled. After they are enabled, these log types are exported to CloudWatch Logs.

- **LogTypesToEnable** – An array of UTF-8 strings.

Log types that are in the process of being deactivated. After they are deactivated, these log types aren't exported to CloudWatch Logs.

ResourcePendingMaintenanceActions (structure)

Describes the pending maintenance actions for a resource.

Fields

- **PendingMaintenanceActionDetails** – An array of [PendingMaintenanceAction](#) (p. 927) `PendingMaintenanceAction` objects.

A list that provides details about the pending maintenance actions for the resource.

- **ResourceIdentifier** – UTF-8 string.

The ARN of the resource that has pending maintenance actions.

`ResourcePendingMaintenanceActions` is used as the response element for:

- [ApplyPendingMaintenanceAction](#) (p. 932)

UpgradeTarget (structure)

The version of the database engine that a DB instance can be upgraded to.

Fields

- **AutoUpgrade** – Boolean.

A value that indicates whether the target version is applied to any source DB instances that have `AutoMinorVersionUpgrade` set to true.

- **Description** – UTF-8 string.
The version of the database engine that a DB instance can be upgraded to.
- **Engine** – UTF-8 string.
The name of the upgrade target database engine.
- **EngineVersion** – UTF-8 string.
The version number of the upgrade target database engine.
- **IsMajorVersionUpgrade** – Boolean.
A value that indicates whether a database engine is upgraded to a major version.

Tag (structure)

Metadata assigned to an Amazon Neptune resource consisting of a key-value pair.

Fields

- **Key** – UTF-8 string.
A key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and can't be prefixed with `aws:` or `rds:`. The string can only contain the set of Unicode letters, digits, white-space, `'_'`, `'.'`, `'/'`, `'='`, `'+'`, `'-'` (Java regex: `"^([\p{L}\p{Z}\p{N}_.:/+\\-]*)$"`).
- **Value** – UTF-8 string.
A value is the optional value of the tag. The string value can be from 1 to 256 Unicode characters in length and can't be prefixed with `aws:` or `rds:`. The string can only contain the set of Unicode letters, digits, white-space, `'_'`, `'.'`, `'/'`, `'='`, `'+'`, `'-'` (Java regex: `"^([\p{L}\p{Z}\p{N}_.:/+\\-]*)$"`).

AddTagsToResource (action)

The Amazon CLI name for this API is: `add-tags-to-resource`, and the Python name is: `add_tags_to_resource`.

Adds metadata tags to an Amazon Neptune resource. These tags can also be used with cost allocation reporting to track cost associated with Amazon Neptune resources, or used in a Condition statement in an IAM policy for Amazon Neptune.

Request

- **ResourceName** (in the CLI: `--resource-name`, or in Python: `resource_name`) – *Required*: UTF-8 string.
The Amazon Neptune resource that the tags are added to. This value is an Amazon Resource Name (ARN). For information about creating an ARN, see [Constructing an Amazon Resource Name \(ARN\)](#).
- **Tags** (in the CLI: `--tags`, or in Python: `tags`) – *Required*: An array of [Tag \(p. 930\)](#) Tag objects.
The tags to be assigned to the Amazon Neptune resource.

Response

- *No Response parameters.*

Errors

- [DBInstanceNotFoundFault](#) (p. 942)
- [DBSnapshotNotFoundFault](#) (p. 945)
- [DBClusterNotFoundFault](#) (p. 940)

ListTagsForResource (action)

The Amazon CLI name for this API is: `list-tags-for-resource`, and the Python name is: `list_tags_for_resource`.

Lists all tags on an Amazon Neptune resource.

Request

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter](#) (p. 937) Filter objects.

This parameter is not currently supported.

- **ResourceName** (in the CLI: `--resource-name`, or in Python: `resource_name`) – *Required:* UTF-8 string.

The Amazon Neptune resource with tags to be listed. This value is an Amazon Resource Name (ARN). For information about creating an ARN, see [Constructing an Amazon Resource Name \(ARN\)](#).

Response

- **TagList** (in the CLI: `--tag-list`, or in Python: `tag_list`) – An array of [Tag](#) (p. 930) Tag objects.

List of tags returned by the ListTagsForResource operation.

Errors

- [DBInstanceNotFoundFault](#) (p. 942)
- [DBSnapshotNotFoundFault](#) (p. 945)
- [DBClusterNotFoundFault](#) (p. 940)

RemoveTagsFromResource (action)

The Amazon CLI name for this API is: `remove-tags-from-resource`, and the Python name is: `remove_tags_from_resource`.

Removes metadata tags from an Amazon Neptune resource.

Request

- **ResourceName** (in the CLI: `--resource-name`, or in Python: `resource_name`) – *Required:* UTF-8 string.

The Amazon Neptune resource that the tags are removed from. This value is an Amazon Resource Name (ARN). For information about creating an ARN, see [Constructing an Amazon Resource Name \(ARN\)](#).

- **TagKeys** (in the CLI: `--tag-keys`, or in Python: `tag_keys`) – *Required:* An array of UTF-8 strings.

The tag key (name) of the tag to be removed.

Response

- *No Response parameters.*

Errors

- [DBInstanceNotFoundFault](#) (p. 942)
- [DBSnapshotNotFoundFault](#) (p. 945)
- [DBClusterNotFoundFault](#) (p. 940)

ApplyPendingMaintenanceAction (action)

The Amazon CLI name for this API is: `apply-pending-maintenance-action`, and the Python name is: `apply_pending_maintenance_action`.

Applies a pending maintenance action to a resource (for example, to a DB instance).

Request

- **ApplyAction** (in the CLI: `--apply-action`, or in Python: `apply_action`) – *Required:* UTF-8 string.

The pending maintenance action to apply to this resource.

Valid values: `system-update`, `db-upgrade`

- **OptInType** (in the CLI: `--opt-in-type`, or in Python: `opt_in_type`) – *Required:* UTF-8 string.

A value that specifies the type of opt-in request, or undoes an opt-in request. An opt-in request of type `immediate` can't be undone.

Valid values:

- `immediate` - Apply the maintenance action immediately.
- `next-maintenance` - Apply the maintenance action during the next maintenance window for the resource.
- `undo-opt-in` - Cancel any existing `next-maintenance` opt-in requests.
- **ResourceIdentifier** (in the CLI: `--resource-identifier`, or in Python: `resource_identifier`) – *Required:* UTF-8 string.

The Amazon Resource Name (ARN) of the resource that the pending maintenance action applies to. For information about creating an ARN, see [Constructing an Amazon Resource Name \(ARN\)](#).

Response

Describes the pending maintenance actions for a resource.

- **PendingMaintenanceActionDetails** (in the CLI: `--pending-maintenance-action-details`, or in Python: `pending_maintenance_action_details`) – An array of [PendingMaintenanceAction](#) (p. 927) `PendingMaintenanceAction` objects.

A list that provides details about the pending maintenance actions for the resource.

- **ResourceIdentifier** (in the CLI: `--resource-identifier`, or in Python: `resource_identifier`) – UTF-8 string.

The ARN of the resource that has pending maintenance actions.

Errors

- [ResourceNotFoundFault](#) (p. 952)

DescribePendingMaintenanceActions (action)

The Amazon CLI name for this API is: `describe-pending-maintenance-actions`, and the Python name is: `describe_pending_maintenance_actions`.

Returns a list of resources (for example, DB instances) that have at least one pending maintenance action.

Request

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter](#) (p. 937) Filter objects.

A filter that specifies one or more resources to return pending maintenance actions for.

Supported filters:

- `db-cluster-id` - Accepts DB cluster identifiers and DB cluster Amazon Resource Names (ARNs). The results list will only include pending maintenance actions for the DB clusters identified by these ARNs.
- `db-instance-id` - Accepts DB instance identifiers and DB instance ARNs. The results list will only include pending maintenance actions for the DB instances identified by these ARNs.
- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribePendingMaintenanceActions` request. If this parameter is specified, the response includes only records beyond the marker, up to a number of records specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more records exist than the specified `MaxRecords` value, a pagination token called a marker is included in the response so that the remaining results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

- **ResourceIdentifier** (in the CLI: `--resource-identifier`, or in Python: `resource_identifier`) – UTF-8 string.

The ARN of a resource to return pending maintenance actions for.

Response

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous `DescribePendingMaintenanceActions` request. If this parameter is specified, the response includes only records beyond the marker, up to a number of records specified by `MaxRecords`.

- **PendingMaintenanceActions** (in the CLI: `--pending-maintenance-actions`, or in Python: `pending_maintenance_actions`) – An array of [ResourcePendingMaintenanceActions](#) (p. 929) `ResourcePendingMaintenanceActions` objects.

A list of the pending maintenance actions for the resource.

Errors

- [ResourceNotFoundFault](#) (p. 952)

DescribeDBEngineVersions (action)

The Amazon CLI name for this API is: `describe-db-engine-versions`, and the Python name is: `describe_db_engine_versions`.

Returns a list of the available DB engines.

Request

- **DBParameterGroupFamily** (in the CLI: `--db-parameter-group-family`, or in Python: `db_parameter_group_family`) – UTF-8 string.

The name of a specific DB parameter group family to return details for.

Constraints:

- If supplied, must match an existing `DBParameterGroupFamily`.
- **DefaultOnly** (in the CLI: `--default-only`, or in Python: `default_only`) – Boolean.

Indicates that only the default version of the specified engine or engine and major version combination is returned.

- **Engine** (in the CLI: `--engine`, or in Python: `engine`) – UTF-8 string.

The database engine to return.

- **EngineVersion** (in the CLI: `--engine-version`, or in Python: `engine_version`) – UTF-8 string.

The database engine version to return.

Example: 5.1.49

- **Filters** (in the CLI: `--filters`, or in Python: `filters`) – An array of [Filter](#) (p. 937) `Filter` objects.

Not currently supported.

- **ListSupportedCharacterSets** (in the CLI: `--list-supported-character-sets`, or in Python: `list_supported_character_sets`) – Boolean.

If this parameter is specified and the requested engine supports the `CharacterSetName` parameter for `CreateDBInstance`, the response includes a list of supported character sets for each engine version.

- **ListSupportedTimezones** (in the CLI: `--list-supported-timezones`, or in Python: `list_supported_timezones`) – Boolean.

If this parameter is specified and the requested engine supports the `TimeZone` parameter for `CreateDBInstance`, the response includes a list of supported time zones for each engine version.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

- **MaxRecords** (in the CLI: `--max-records`, or in Python: `max_records`) – Number (integer).

The maximum number of records to include in the response. If more than the `MaxRecords` value is available, a pagination token called a marker is included in the response so that the following results can be retrieved.

Default: 100

Constraints: Minimum 20, maximum 100.

Response

- **DBEngineVersions** (in the CLI: `--db-engine-versions`, or in Python: `db_engine_versions`) – An array of [DBEngineVersion](#) (p. 926) `DBEngineVersion` objects.

A list of `DBEngineVersion` elements.

- **Marker** (in the CLI: `--marker`, or in Python: `marker`) – UTF-8 string.

An optional pagination token provided by a previous request. If this parameter is specified, the response includes only records beyond the marker, up to the value specified by `MaxRecords`.

Common Neptune Datatypes

Structures:

- [AvailabilityZone](#) (structure) (p. 935)
- [DBSecurityGroupMembership](#) (structure) (p. 935)
- [DomainMembership](#) (structure) (p. 936)
- [DoubleRange](#) (structure) (p. 936)
- [Endpoint](#) (structure) (p. 936)
- [Filter](#) (structure) (p. 937)
- [Range](#) (structure) (p. 937)
- [Timezone](#) (structure) (p. 937)
- [VpcSecurityGroupMembership](#) (structure) (p. 937)

AvailabilityZone (structure)

Specifies an Availability Zone.

Fields

- **Name** – UTF-8 string.

The name of the availability zone.

DBSecurityGroupMembership (structure)

Specifies membership in a designated DB security group.

Fields

- **DBSecurityGroupName** – UTF-8 string.

The name of the DB security group.

- **Status** – UTF-8 string.

The status of the DB security group.

DomainMembership (structure)

An Active Directory Domain membership record associated with a DB instance.

Fields

- **Domain** – UTF-8 string.

The identifier of the Active Directory Domain.

- **FQDN** – UTF-8 string.

The fully qualified domain name of the Active Directory Domain.

- **IAMRoleName** – UTF-8 string.

The name of the IAM role to be used when making API calls to the Directory Service.

- **Status** – UTF-8 string.

The status of the DB instance's Active Directory Domain membership, such as joined, pending-join, failed etc).

DoubleRange (structure)

A range of double values.

Fields

- **From** – Number (double).

The minimum value in the range.

- **To** – Number (double).

The maximum value in the range.

Endpoint (structure)

Specifies a connection endpoint.

For the data structure that represents Amazon Neptune DB cluster endpoints, see `DBClusterEndpoint`.

Fields

- **Address** – UTF-8 string.

Specifies the DNS address of the DB instance.

- **HostedZoneId** – UTF-8 string.

Specifies the ID that Amazon Route 53 assigns when you create a hosted zone.

- **Port** – Number (integer).

Specifies the port that the database engine is listening on.

Filter (structure)

This type is not currently supported.

Fields

- **Name** – *Required:* UTF-8 string.

This parameter is not currently supported.

- **Values** – *Required:* An array of Value— UTF-8 strings.

This parameter is not currently supported.

Range (structure)

A range of integer values.

Fields

- **From** – Number (integer).

The minimum value in the range.

- **Step** – Number (integer).

The step value for the range. For example, if you have a range of 5,000 to 10,000, with a step value of 1,000, the valid values start at 5,000 and step up by 1,000. Even though 7,500 is within the range, it isn't a valid value for the range. The valid values are 5,000, 6,000, 7,000, 8,000...

- **To** – Number (integer).

The maximum value in the range.

Timezone (structure)

A time zone associated with a [the section called "DBInstance" \(p. 828\)](#).

Fields

- **TimezoneName** – UTF-8 string.

The name of the time zone.

VpcSecurityGroupMembership (structure)

This data type is used as a response element for queries on VPC security group membership.

Fields

- **Status** – UTF-8 string.

The status of the VPC security group.

- **VpcSecurityGroupId** – UTF-8 string.

The name of the VPC security group.

Neptune Exceptions Specific to Individual APIs

Exceptions:

- [AuthorizationAlreadyExistsFault](#) (structure) (p. 939)
- [AuthorizationNotFoundFault](#) (structure) (p. 940)
- [AuthorizationQuotaExceededFault](#) (structure) (p. 940)
- [CertificateNotFoundFault](#) (structure) (p. 940)
- [DBClusterAlreadyExistsFault](#) (structure) (p. 940)
- [DBClusterNotFoundFault](#) (structure) (p. 940)
- [DBClusterParameterGroupNotFoundFault](#) (structure) (p. 941)
- [DBClusterQuotaExceededFault](#) (structure) (p. 941)
- [DBClusterRoleAlreadyExistsFault](#) (structure) (p. 941)
- [DBClusterRoleNotFoundFault](#) (structure) (p. 941)
- [DBClusterRoleQuotaExceededFault](#) (structure) (p. 942)
- [DBClusterSnapshotAlreadyExistsFault](#) (structure) (p. 942)
- [DBClusterSnapshotNotFoundFault](#) (structure) (p. 942)
- [DBInstanceAlreadyExistsFault](#) (structure) (p. 942)
- [DBInstanceNotFoundFault](#) (structure) (p. 942)
- [DBLogFileNotFoundFault](#) (structure) (p. 943)
- [DBParameterGroupAlreadyExistsFault](#) (structure) (p. 943)
- [DBParameterGroupNotFoundFault](#) (structure) (p. 943)
- [DBParameterGroupQuotaExceededFault](#) (structure) (p. 943)
- [DBSecurityGroupAlreadyExistsFault](#) (structure) (p. 944)
- [DBSecurityGroupNotFoundFault](#) (structure) (p. 944)
- [DBSecurityGroupNotSupportedFault](#) (structure) (p. 944)
- [DBSecurityGroupQuotaExceededFault](#) (structure) (p. 944)
- [DBSnapshotAlreadyExistsFault](#) (structure) (p. 944)
- [DBSnapshotNotFoundFault](#) (structure) (p. 945)
- [DBSubnetGroupAlreadyExistsFault](#) (structure) (p. 945)
- [DBSubnetGroupDoesNotCoverEnoughAZs](#) (structure) (p. 945)
- [DBSubnetGroupNotAllowedFault](#) (structure) (p. 945)
- [DBSubnetGroupNotFoundFault](#) (structure) (p. 946)
- [DBSubnetGroupQuotaExceededFault](#) (structure) (p. 946)
- [DBSubnetQuotaExceededFault](#) (structure) (p. 946)
- [DBUpgradeDependencyFailureFault](#) (structure) (p. 946)
- [DomainNotFoundFault](#) (structure) (p. 946)
- [EventSubscriptionQuotaExceededFault](#) (structure) (p. 947)
- [InstanceQuotaExceededFault](#) (structure) (p. 947)

- [InsufficientDBClusterCapacityFault \(structure\)](#) (p. 947)
- [InsufficientDBInstanceCapacityFault \(structure\)](#) (p. 947)
- [InsufficientStorageClusterCapacityFault \(structure\)](#) (p. 948)
- [InvalidDBClusterSnapshotStateFault \(structure\)](#) (p. 948)
- [InvalidDBClusterStateFault \(structure\)](#) (p. 948)
- [InvalidDBInstanceStateFault \(structure\)](#) (p. 948)
- [InvalidDBParameterGroupStateFault \(structure\)](#) (p. 948)
- [InvalidDBSecurityGroupStateFault \(structure\)](#) (p. 949)
- [InvalidDBSnapshotStateFault \(structure\)](#) (p. 949)
- [InvalidDBSubnetGroupFault \(structure\)](#) (p. 949)
- [InvalidDBSubnetGroupStateFault \(structure\)](#) (p. 949)
- [InvalidDBSubnetStateFault \(structure\)](#) (p. 950)
- [InvalidEventSubscriptionStateFault \(structure\)](#) (p. 950)
- [InvalidOptionGroupStateFault \(structure\)](#) (p. 950)
- [InvalidRestoreFault \(structure\)](#) (p. 950)
- [InvalidSubnet \(structure\)](#) (p. 950)
- [InvalidVPCNetworkStateFault \(structure\)](#) (p. 951)
- [KMSKeyNotAccessibleFault \(structure\)](#) (p. 951)
- [OptionGroupNotFoundFault \(structure\)](#) (p. 951)
- [PointInTimeRestoreNotEnabledFault \(structure\)](#) (p. 951)
- [ProvisionedIopsNotAvailableInAZFault \(structure\)](#) (p. 952)
- [ResourceNotFoundFault \(structure\)](#) (p. 952)
- [SNSInvalidTopicFault \(structure\)](#) (p. 952)
- [SNSNoAuthorizationFault \(structure\)](#) (p. 952)
- [SNSTopicArnNotFoundFault \(structure\)](#) (p. 952)
- [SharedSnapshotQuotaExceededFault \(structure\)](#) (p. 953)
- [SnapshotQuotaExceededFault \(structure\)](#) (p. 953)
- [SourceNotFoundFault \(structure\)](#) (p. 953)
- [StorageQuotaExceededFault \(structure\)](#) (p. 953)
- [StorageTypeNotSupportedFault \(structure\)](#) (p. 954)
- [SubnetAlreadyInUse \(structure\)](#) (p. 954)
- [SubscriptionAlreadyExistFault \(structure\)](#) (p. 954)
- [SubscriptionCategoryNotFoundFault \(structure\)](#) (p. 954)
- [SubscriptionNotFoundFault \(structure\)](#) (p. 954)

AuthorizationAlreadyExistsFault (structure)

HTTP status code returned: 400.

The specified CIDRIP or EC2 security group is already authorized for the specified DB security group.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

AuthorizationNotFoundFault (structure)

HTTP status code returned: 404.

Specified CIDRIP or EC2 security group is not authorized for the specified DB security group.

Neptune may not also be authorized via IAM to perform necessary actions on your behalf.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

AuthorizationQuotaExceededFault (structure)

HTTP status code returned: 400.

DB security group authorization quota has been reached.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

CertificateNotFoundFault (structure)

HTTP status code returned: 404.

CertificateIdentifier does not refer to an existing certificate.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterAlreadyExistsFault (structure)

HTTP status code returned: 400.

User already has a DB cluster with the given identifier.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterNotFoundFault (structure)

HTTP status code returned: 404.

DBClusterIdentifier does not refer to an existing DB cluster.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterParameterGroupNotFoundFault (structure)

HTTP status code returned: 404.

DBClusterParameterGroupName does not refer to an existing DB Cluster parameter group.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterQuotaExceededFault (structure)

HTTP status code returned: 403.

User attempted to create a new DB cluster and the user has already reached the maximum allowed DB cluster quota.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterRoleAlreadyExistsFault (structure)

HTTP status code returned: 400.

The specified IAM role Amazon Resource Name (ARN) is already associated with the specified DB cluster.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterRoleNotFoundFault (structure)

HTTP status code returned: 404.

The specified IAM role Amazon Resource Name (ARN) is not associated with the specified DB cluster.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterRoleQuotaExceededFault (structure)

HTTP status code returned: 400.

You have exceeded the maximum number of IAM roles that can be associated with the specified DB cluster.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterSnapshotAlreadyExistsFault (structure)

HTTP status code returned: 400.

User already has a DB cluster snapshot with the given identifier.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBClusterSnapshotNotFoundFault (structure)

HTTP status code returned: 404.

DBClusterSnapshotIdentifier does not refer to an existing DB cluster snapshot.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBInstanceAlreadyExistsFault (structure)

HTTP status code returned: 400.

User already has a DB instance with the given identifier.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBInstanceNotFoundFault (structure)

HTTP status code returned: 404.

DBInstanceIdentifier does not refer to an existing DB instance.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

DBLogFileNotFoundFault (structure)

HTTP status code returned: 404.

LogFileName does not refer to an existing DB log file.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

DBParameterGroupAlreadyExistsFault (structure)

HTTP status code returned: 400.

A DB parameter group with the same name exists.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

DBParameterGroupNotFoundFault (structure)

HTTP status code returned: 404.

DBParameterGroupName does not refer to an existing DB parameter group.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

DBParameterGroupQuotaExceededFault (structure)

HTTP status code returned: 400.

Request would result in user exceeding the allowed number of DB parameter groups.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

DBSecurityGroupAlreadyExistsFault (structure)

HTTP status code returned: 400.

A DB security group with the name specified in *DBSecurityGroupName* already exists.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSecurityGroupNotFoundFault (structure)

HTTP status code returned: 404.

DBSecurityGroupName does not refer to an existing DB security group.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSecurityGroupNotSupportedFault (structure)

HTTP status code returned: 400.

A DB security group is not allowed for this action.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSecurityGroupQuotaExceededFault (structure)

HTTP status code returned: 400.

Request would result in user exceeding the allowed number of DB security groups.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSnapshotAlreadyExistsFault (structure)

HTTP status code returned: 400.

DBSnapshotIdentifier is already used by an existing snapshot.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSnapshotNotFoundFault (structure)

HTTP status code returned: 404.

DBSnapshotIdentifier does not refer to an existing DB snapshot.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSubnetGroupAlreadyExistsFault (structure)

HTTP status code returned: 400.

DBSubnetGroupName is already used by an existing DB subnet group.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSubnetGroupDoesNotCoverEnoughAZs (structure)

HTTP status code returned: 400.

Subnets in the DB subnet group should cover at least two Availability Zones unless there is only one Availability Zone.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSubnetGroupNotAllowedFault (structure)

HTTP status code returned: 400.

Indicates that the DBSubnetGroup should not be specified while creating read replicas that lie in the same region as the source instance.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSubnetGroupNotFoundFault (structure)

HTTP status code returned: 404.

DBSubnetGroupName does not refer to an existing DB subnet group.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSubnetGroupQuotaExceededFault (structure)

HTTP status code returned: 400.

Request would result in user exceeding the allowed number of DB subnet groups.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBSubnetQuotaExceededFault (structure)

HTTP status code returned: 400.

Request would result in user exceeding the allowed number of subnets in a DB subnet groups.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DBUpgradeDependencyFailureFault (structure)

HTTP status code returned: 400.

The DB upgrade failed because a resource the DB depends on could not be modified.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

DomainNotFoundFault (structure)

HTTP status code returned: 404.

Domain does not refer to an existing Active Directory Domain.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

EventSubscriptionQuotaExceededFault (structure)

HTTP status code returned: 400.

You have exceeded the number of events you can subscribe to.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InstanceQuotaExceededFault (structure)

HTTP status code returned: 400.

Request would result in user exceeding the allowed number of DB instances.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InsufficientDBClusterCapacityFault (structure)

HTTP status code returned: 403.

The DB cluster does not have enough capacity for the current operation.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InsufficientDBInstanceCapacityFault (structure)

HTTP status code returned: 400.

Specified DB instance class is not available in the specified Availability Zone.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InsufficientStorageClusterCapacityFault (structure)

HTTP status code returned: 400.

There is insufficient storage available for the current action. You may be able to resolve this error by updating your subnet group to use different Availability Zones that have more storage available.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBClusterSnapshotStateFault (structure)

HTTP status code returned: 400.

The supplied value is not a valid DB cluster snapshot state.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBClusterStateFault (structure)

HTTP status code returned: 400.

The DB cluster is not in a valid state.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBInstanceStateFault (structure)

HTTP status code returned: 400.

The specified DB instance is not in the *available* state.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBParameterGroupStateFault (structure)

HTTP status code returned: 400.

The DB parameter group is in use or is in an invalid state. If you are attempting to delete the parameter group, you cannot delete it when the parameter group is in this state.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBSecurityGroupStateFault (structure)

HTTP status code returned: 400.

The state of the DB security group does not allow deletion.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBSnapshotStateFault (structure)

HTTP status code returned: 400.

The state of the DB snapshot does not allow deletion.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBSubnetGroupFault (structure)

HTTP status code returned: 400.

Indicates the DBSubnetGroup does not belong to the same VPC as that of an existing cross region read replica of the same source instance.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBSubnetGroupStateFault (structure)

HTTP status code returned: 400.

The DB subnet group cannot be deleted because it is in use.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidDBSubnetStateFault (structure)

HTTP status code returned: 400.

The DB subnet is not in the *available* state.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InvalidEventSubscriptionStateFault (structure)

HTTP status code returned: 400.

The event subscription is in an invalid state.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InvalidOptionGroupStateFault (structure)

HTTP status code returned: 400.

The option group is not in the *available* state.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InvalidRestoreFault (structure)

HTTP status code returned: 400.

Cannot restore from vpc backup to non-vpc DB instance.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

InvalidSubnet (structure)

HTTP status code returned: 400.

The requested subnet is invalid, or multiple subnets were requested that are not all in a common VPC.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

InvalidVPCNetworkStateFault (structure)

HTTP status code returned: 400.

DB subnet group does not cover all Availability Zones after it is created because users' change.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

KMSKeyNotAccessibleFault (structure)

HTTP status code returned: 400.

Error accessing KMS key.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

OptionGroupNotFoundFault (structure)

HTTP status code returned: 404.

The designated option group could not be found.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

PointInTimeRestoreNotEnabledFault (structure)

HTTP status code returned: 400.

SourceDBInstanceIdentifier refers to a DB instance with *BackupRetentionPeriod* equal to 0.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.

ProvisionedIopsNotAvailableInAZFault (structure)

HTTP status code returned: 400.

Provisioned IOPS not available in the specified Availability Zone.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

ResourceNotFoundFault (structure)

HTTP status code returned: 404.

The specified resource ID was not found.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SNSInvalidTopicFault (structure)

HTTP status code returned: 400.

The SNS topic is invalid.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SNSNoAuthorizationFault (structure)

HTTP status code returned: 400.

There is no SNS authorization.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SNSTopicArnNotFoundFault (structure)

HTTP status code returned: 404.

The ARN of the SNS topic could not be found.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SharedSnapshotQuotaExceededFault (structure)

HTTP status code returned: 400.

You have exceeded the maximum number of accounts that you can share a manual DB snapshot with.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SnapshotQuotaExceededFault (structure)

HTTP status code returned: 400.

Request would result in user exceeding the allowed number of DB snapshots.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SourceNotFoundFault (structure)

HTTP status code returned: 404.

The source could not be found.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

StorageQuotaExceededFault (structure)

HTTP status code returned: 400.

Request would result in user exceeding the allowed amount of storage available across all DB instances.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

StorageTypeNotSupportedFault (structure)

HTTP status code returned: 400.

StorageType specified cannot be associated with the DB Instance.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SubnetAlreadyInUse (structure)

HTTP status code returned: 400.

The DB subnet is already in use in the Availability Zone.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SubscriptionAlreadyExistFault (structure)

HTTP status code returned: 400.

This subscription already exists.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SubscriptionCategoryNotFoundFault (structure)

HTTP status code returned: 404.

The designated subscription category could not be found.

Fields

- **message** – UTF-8 string.
A message describing the details of the problem.

SubscriptionNotFoundFault (structure)

HTTP status code returned: 404.

The designated subscription could not be found.

Fields

- **message** – UTF-8 string.

A message describing the details of the problem.