

WebSphere eXtreme Scale V8.6

Key Concepts and Usage Scenarios

Introduction to elastic caching

Typical application scenarios

Deployment options



Jonathan Marshall
John Pape
Kristi Peterson
Greg Reid
Fabio Santos B. da Silva
Federico Senese



International Technical Support Organization

**WebSphere eXtreme Scale V8.6:
Key Concepts and Usage Scenarios**

September 2013

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

Second Edition (September 2013)

This edition applies to WebSphere eXtreme Scale Version 8.6.

© Copyright International Business Machines Corporation 2008, 2013. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
Authors	xi
Now you can become a published author, too!	xiii
Comments welcome	xiii
Stay connected to IBM Redbooks	xiii
Chapter 1. Introduction to WebSphere eXtreme Scale	1
1.1 The scalability challenge	2
1.2 Caching	4
1.2.1 Holding the cached data	5
1.2.2 Disk offload	6
1.2.3 Data grids	6
1.3 Introduction to WebSphere eXtreme Scale	6
1.3.1 Architecture and terminology	7
1.3.2 Transaction support	9
1.3.3 Securability	10
1.3.4 Scalability	10
1.3.5 High availability	11
1.4 Installation modes and deployment topologies	11
1.4.1 Installation modes	12
1.4.2 Deployment topologies	13
1.4.3 Mixing and matching	15
1.5 What is new in WebSphere eXtreme Scale Version 8.6	16
1.5.1 Architectural enhancements	16
1.5.2 Programming enhancements	17
1.5.3 Monitoring and management enhancements	19
1.6 Entry points for WebSphere eXtreme Scale	19
1.6.1 Possible entry points	19
1.6.2 Decision tree	21
1.7 Comparing WebSphere eXtreme Scale to in-memory databases	22
1.7.1 Introducing IMDBs	22
1.7.2 Explaining the difference	23
1.8 IBM WebSphere DataPower XC10 Appliance	24
Chapter 2. Architecture and topologies	27
2.1 WebSphere eXtreme Scale architecture	28
2.1.1 Grid architecture	28
2.1.2 Internal components	30
2.1.3 Grid clients and servers	32
2.1.4 WebSphere eXtreme Scale meta model	33
2.1.5 Transport protocol: IBM eXtremeIO (XIO)	34
2.2 Catalog server	35
2.2.1 Access point for clients	35
2.2.2 Maintaining the catalog service domain	35
2.2.3 Placement and balancing shards	37
2.2.4 Placement work and route table flow	39

2.3	Replication: Primary and replica shards	44
2.3.1	Selecting data to replicate: Revisioning	44
2.3.2	Synchronous replicas	45
2.3.3	Asynchronous replicas	47
2.4	APIs used to access the grid	48
2.4.1	ObjectMap API	48
2.4.2	EntityManager API	50
2.4.3	Spring framework	51
2.4.4	Microsoft .NET	53
2.4.5	REST data service	53
2.4.6	REST gateway	54
2.5	A simple example	54
2.6	Scalability sizing considerations	56
2.7	Common topologies	57
2.7.1	Managed grid	57
2.7.2	Stand-alone grid	57
2.7.3	Embedded cache	58
2.7.4	Embedded-partitioned application and cache	58
2.7.5	Remote cache (with optional near cache)	59
2.8	Common topology options	60
2.8.1	Zones	60
2.8.2	IBM eXtreme Data Format	66
2.8.3	IBM eXtremeMemory	67
2.8.4	Multi-master replication	69
Chapter 3. Application scenarios		75
3.1	Introducing the scenarios	76
3.2	Application state store scenario	77
3.2.1	HTTP session distribution	78
3.2.2	Application server elasticity	78
3.2.3	Enterprise service bus state management	79
3.3	Side cache scenario	79
3.3.1	Implementation overview	80
3.3.2	Dynamic cache provider	81
3.3.3	Service-oriented architecture caching	82
3.3.4	Mobile gateway	83
3.3.5	Enterprise data grid (cross-technology access)	83
3.4	In-line cache scenario	83
3.4.1	Implementation overview	84
3.4.2	EIS shock absorber	85
3.5	Extreme Transaction Processing scenario	85
3.5.1	Implementation overview	87
3.5.2	MapReduce programming model	87
3.5.3	Real-time business rules and event processing	89
3.6	Integration with other IBM products	90
3.6.1	WebSphere Portal Server and Web Content Manager	91
3.6.2	WebSphere Commerce Server	91
3.6.3	IBM WebSphere DataPower XG45 and XI52 appliances	91
3.6.4	WebSphere Message Broker	92
3.6.5	IBM Worklight	92
3.6.6	IBM Business Process Manager	92
3.6.7	Rational Team Concert	93
3.6.8	IBM Operational Decision Management	93

Chapter 4. Developing with WebSphere eXtreme Scale	95
4.1 Setting up a development environment.	96
4.1.1 Installing the development environment	97
4.1.2 Creating a WebSphere eXtreme Scale grid	103
4.2 Developing for WebSphere eXtreme Scale	107
4.2.1 Testing the grid with a sample web application.	107
4.2.2 Using the ObjectMap API	109
4.3 Loading data into the grid	111
4.3.1 Introducing loaders	112
4.3.2 Implementing the JPA loader	114
4.3.3 Implementing a time-based updater	117
4.3.4 Implementing a client loader	120
4.4 Querying data using Object Grid Query Language	122
4.4.1 Important considerations when using queries.	125
4.4.2 Optimizing query performance using the global index	127
4.4.3 Inverse range index	129
4.4.4 Continuous query	131
4.5 Server-side development with agents	132
4.5.1 Agent development	133
4.5.2 Liberty Profile configuration	136
4.6 Dealing with data eviction and stale data	137
4.6.1 Tolerating the data	138
4.6.2 Using time-based eviction strategies	138
4.6.3 Polling the database for updates in regular intervals	139
4.6.4 Using JMS to propagate changes	140
4.6.5 Using near cache invalidation in WebSphere eXtreme Scale version 8.6	140
4.6.6 Assuring no external changes are made to backing store	141
4.6.7 Ensuring all external change processes notify the grid.	141
4.7 Saving time with WXSUtils	143
4.7.1 Examples of usage	143
4.8 Transactions	144
4.8.1 Transaction considerations	144
4.8.2 JTA transactions	148
4.8.3 Multi-partition transactions	148
Chapter 5. Deployment scenarios	151
5.1 WebSphere eXtreme Scale in a stand-alone environment	152
5.1.1 Benefits	152
5.1.2 Limitations	152
5.1.3 Introducing the sample application	153
5.1.4 Introducing the sample topology	156
5.1.5 Creating the sample topology	158
5.1.6 Validating the sample topology	171
5.1.7 Testing the sample topology and application	174
5.2 Integrating in a WebSphere Application Server Network Deployment environment	178
5.2.1 Benefits	179
5.2.2 Limitations	182
5.2.3 Considerations	183
5.2.4 Introducing the sample application	184
5.2.5 Introducing the sample topology	187
5.2.6 Creating the sample topology	189
5.2.7 Validating the sample topology	221
5.2.8 Testing the sample topology and application	227

5.3	Integrating into WebSphere Application Server Liberty Profile	230
5.3.1	Benefits	231
5.3.2	Limitations	231
5.3.3	Considerations	231
5.3.4	Introducing the sample application	232
5.3.5	Introducing the sample topology	232
5.3.6	Creating the sample topology	233
5.3.7	Testing the sample topology and application	239
Chapter 6.	Extended HTTP session management with WebSphere eXtreme Scale	243
6.1	HTTP session management overview	244
6.1.1	Using HTTP session to hold state	244
6.1.2	Tracking who what owns each HTTP session	245
6.1.3	Session replication and session persistence	247
6.1.4	Session affinity	247
6.1.5	Commonly implemented options for HTTP session management	249
6.2	Using WebSphere eXtreme Scale for HTTP session management	250
6.2.1	Benefits of using WebSphere eXtreme Scale for HTTP session management	250
6.2.2	Integration of WebSphere eXtreme Scale session management	251
6.2.3	Configuration for WebSphere eXtreme Scale session management	252
6.2.4	Topologies for eXtreme Scale session management	253
6.2.5	Sample grid configuration files for an HTTP session	253
6.2.6	HTTP session servlet filter configuration	256
6.3	Introducing the SessionTest sample application	257
6.4	Scenario 1: Implementing with WebSphere Application Server Network Deployment	260
6.4.1	Installing the sample SessionTest application	261
6.4.2	Configuring WebSphere eXtreme Scale session management	262
6.4.3	Installing the HTTPSessionGrid application into the container servers	263
6.4.4	Reconfiguring to use WebSphere eXtreme Scale session management	270
6.4.5	Testing the application	279
6.5	Scenario 2: Implementing with WebSphere Liberty Profile	281
6.5.1	Implementing the grid configuration	281
6.5.2	Configuring HTTP session management	282
6.5.3	Running the application	286
6.5.4	Validating HTTP session failover	286
6.6	Scenario 3: Implementing with Apache Tomcat	288
6.6.1	Operational model	288
6.6.2	Implementing the web application server deployment environment	289
6.6.3	Implementing the grid topology	292
6.6.4	Configuring HTTP session management	293
6.6.5	Deploying the sample application into Apache Tomcat	295
6.6.6	Starting the deployment environment	296
6.6.7	Testing the application and session failover	299
Chapter 7.	Basic administration	303
7.1	The web console	304
7.1.1	Configuring the web console for monitoring	305
7.1.2	Monitoring your grid with the web console	307
7.1.3	Monitoring your grid with the Message Center	310
7.2	Command line access: xscmd	310
7.2.1	Placement layout: showPlacement command	311
7.2.2	Current route table: routetable command	313
7.2.3	Size of the grid: showMapSizes command	315

7.2.4 Rerun placement: triggerPlacement command	316
7.2.5 Version of grid contents: revisions command	317
7.2.6 Information about servers: showInfo command	319
7.2.7 Stopping servers efficiently: teardown command	319
7.2.8 Reviewing keys in the grid: findByKey command	320
7.2.9 Multi-master replication commands	320
7.2.10 Viewing health notifications with xscmd	323
7.3 Log analyzer	324
7.3.1 Running the log analyzer	324
7.3.2 Extending the capabilities of the log analyzer	325
7.3.3 Sample output from xsLogAnalyzer	326
7.4 JVM logging and tracing	326
7.4.1 Enabling logging and tracing in a stand-alone WebSphere eXtreme Scale environment	327
7.4.2 Enabling logging in a WebSphere Application Server managed eXtreme Scale environment	327
7.4.3 Using High Performance Extensible Logging	327
7.5 Monitoring with other products	328
7.5.1 IBM Tivoli Enterprise Monitoring Agent	328
7.5.2 CA Wily Introscope	328
7.5.3 Hyperic HQ	329
7.5.4 Java JConsole and other MBean inspection tools	329
7.6 Tuning, recovery and troubleshooting	329
7.6.1 Tuning the heartbeat frequency level	329
7.6.2 Tuning and controlling placement	330
7.6.3 Tuning container server start and stop	332
7.6.4 Off loading work from primary shards: replicaReadEnabled option	332
7.6.5 Recovery for network blips and brown outs	333
7.6.6 Recovery for client failures	333
7.6.7 Troubleshooting placement problems	334
7.6.8 Troubleshooting synchronous replication	334
7.6.9 Troubleshooting asynchronous replication	337
7.6.10 Troubleshooting multi-master replication	338
7.6.11 Testing a custom collision arbiter for multi-master replication	339
Appendix A. Additional material	341
Locating the Web material	341
Using the Web material	341
Downloading and extracting the Web material	342
Related publications	343
IBM Redbooks publications	343
Online resources	343
How to get Redbooks publications	345
Help from IBM	345

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DataPower®	IBM®	solidDB®
DB2®	Rational®	Tivoli®
developerWorks®	Rational Team Concert™	WebSphere®
Global Technology Services®	Redbooks®	
IA®	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

Worklight is trademark or registered trademark of Worklight, an IBM Company.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM WebSphere® eXtreme Scale provides a solution to scalability issues through caching and grid technology. It provides an enhanced quality of service in high performance computing environments.

This IBM® Redbooks® publication introduces WebSphere eXtreme Scale and shows how to set up and use an eXtreme Scale environment. It begins with a discussion of the issues that would lead you to an eXtreme Scale solution. It then describes the architecture of eXtreme Scale to help you understand how the product works. It provides information about potential grid topologies, the APIs used by applications to access the grid, and application scenarios that show how to effectively use the grid.

This book is intended for architects who want to implement WebSphere eXtreme Scale.

The original edition of this book was based on WebSphere eXtreme Scale version 6.1. It was published in 2008 and described as a “User’s Guide”. This second edition updates the information based on WebSphere eXtreme Scale version 8.6, and covers key concepts and usage scenarios. For more information about what has changed, see 1.5, “What is new in WebSphere eXtreme Scale Version 8.6” on page 16.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center:

Jonathan Marshall is a Consulting IT Specialist working in the UK as a WebSphere Technical Professional with 13 years experience in IBM. He has worked with a wide range of clients on WebSphere Application Server and related technologies, including Liberty Profile, WebSphere eXtreme Scale, WebSphere Virtual Enterprise, and IBM Workload Deployer. He is also currently the mobile lead in the UK WebSphere team, covering mobile technologies such as IBM Worklight®. He has published developerWorks® articles and previous Redbooks on WebSphere eXtreme Scale, including being one of the authors of the first edition of this book. Jonathan can be reached on Twitter @jmarshall1 and through his blog at https://www.ibm.com/developerworks/community/blogs/WASFAQs/entry/extreme_development_with_websphere_extreme_scale_and_liberty?lang=en.

John Pape is an advisory software engineer in Raleigh, NC in the United States. He has 13 years of experience with WebSphere Application Server, and has been with IBM since 2005. His areas of expertise include IBM WebSphere Application Server, WebSphere Virtual Enterprise, WebSphere eXtreme Scale, WebSphere DataPower® XC10 Caching Appliance, virtualization, Linux, and the Java Virtual Machine. He holds a degree in Management Information Systems. John was also one of the authors of the first edition of this book, and blogs externally at <http://johnsrandommusings.tumblr.com>.

Kristi Peterson is an advisory software engineer in Rochester, MN in the United States. She works on the WebSphere eXtreme Scale development team. Kristi has been involved in the inner workings of WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 Appliance since 2006. She began her IBM career with WebSphere Application Server doing high availability customer scenario testing. She holds a BA degree in Computer Science and

English from Luther College in Decorah, IA. Kristi was a contributor to the first edition of this book.

Greg Reid enjoyed 20 years of support and development work with IBM Canada before he joined the IBM Software Services for WebSphere (ISSW) organization in the US. After five more years as an ISSW consultant for WebSphere ND, Greg moved into the WebSphere Commerce practice, where he was first introduced to the powerful capabilities of WebSphere eXtreme Scale. He then moved back to ISSW to specialize in it. Greg holds a degree in Computer Science.

Fabio Santos B. da Silva is an IT Specialist working for IBM Global Technology Services® in Brazil. He has eight years of expertise in supporting IBM WebSphere Application Server and IBM WebSphere Portal Server. His main areas of expertise are automation for WebSphere administration, infrastructure design, implementation, and maintenance and problem determination in the WebSphere environment. He has more than 14 years of IT expertise in fields such as middleware, Java programming, open source tools, UNIX operating systems, network security, and web hosting environments. He has designed, implemented, and supported various middleware infrastructure and web hosting environments in large public and private organizations. Fabio is also an IBM Certified System Administrator for WebSphere Application Server and WebSphere Portal Server.

Federico Senese is a Consulting IT Specialist who is based in Rome, Italy. Federico works as a WebSphere Application Infrastructure technical leader in Europe. His mission is to promote the adoption of strategic WebSphere products, such as WebSphere eXtreme Scale. He has deep skills on several WebSphere products, including WebSphere Application Server, WebSphere eXtreme Scale, and IBM WebSphere DataPower XC10 Appliance, and on other related offerings such as IBM WebSphere Portal and IBM WebSphere DataPower XI45 and XI52. During his 13 year career at IBM, Federico has helped several large enterprise customers in Europe to adopt IBM WebSphere middleware solutions to deliver innovative, mission critical applications and improve the quality of service of their IT platforms. Federico holds a degree in Electronic Engineering from University of Salerno, Italy.

Thanks to the authors of the first edition of this book: Daniel Froehlich, Nitin Gaur, and Jennifer Zorza. We thank them for their excellent work, much of which is still contained in this second edition.

The second edition of this IBM Redbooks project was led by:

Debbie Landon
International Technical Support Organization, Raleigh Center

Thanks to the following people for their contributions to this project:

Charles LeVay
Paul Maridueña
Gabriel Montero
Vince Pedroza
Thomas Savage
Michael Smith
WebSphere eXtreme Scale development team, IBM Raleigh

Jared Anderson
Joshua Dettinger
Fred Kulack
Jim Krueger
Lisa Walkosz

Bob Westland
WebSphere eXtreme Scale development team, IBM Rochester

Abelard Chow
Kenny Chu
John Paul Parkin
Imran Sayyid
WebSphere eXtreme Scale development team, IBM Toronto

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Introduction to WebSphere eXtreme Scale

IBM WebSphere eXtreme Scale provides a high-performance, highly scalable caching framework. This chapter introduces WebSphere eXtreme Scale and its place in a modern application environment.

Product name: The WebSphere eXtreme Scale product name is often shortened to *eXtreme Scale*. Both names are used interchangeably throughout this document.

This chapter explains scalability challenges that exist in today's application environment, and how caching and grid technologies can help to resolve this challenge. The key features of the eXtreme Scale product are introduced, illustrating how they can help to solve these issues. Possible entry points to eXtreme Scale are also proposed. The chapter closes with a comparison of eXtreme Scale to in-memory databases, and introduces the IBM WebSphere DataPower XC10 Appliance.

This chapter includes the following sections:

- ▶ “The scalability challenge” on page 2
- ▶ “Caching” on page 4
- ▶ “Introduction to WebSphere eXtreme Scale” on page 6
- ▶ “Installation modes and deployment topologies” on page 11
- ▶ “What is new in WebSphere eXtreme Scale Version 8.6” on page 16
- ▶ “Entry points for WebSphere eXtreme Scale” on page 19
- ▶ “Comparing WebSphere eXtreme Scale to in-memory databases” on page 22
- ▶ “IBM WebSphere DataPower XC10 Appliance” on page 24

1.1 The scalability challenge

Scalability is the ability of a system to handle increasing load in a graceful manner. This implies that a system can be readily extended. A system has perfectly linear scaling capabilities if doubling its processor capacity also doubles its maximum throughput. In general, there are two ways an IT system can be scaled:

- ▶ Horizontally, by adding more hosts to a tier. This is called *scale out*.
- ▶ Vertically, by enlarging the capabilities of a single system. For example, adding processors. This is called *scale up*.

Consider a classical three tier application such as the one shown in Figure 1-1. The application server tier is both scaled out by having three hosts and scaled up by having three application servers on each host. The database tier is scaled up by using a single powerful system with many processors. The database tier is scaled out by having a shadow database that uses log shipping capability to support reports, analysis, and so forth.

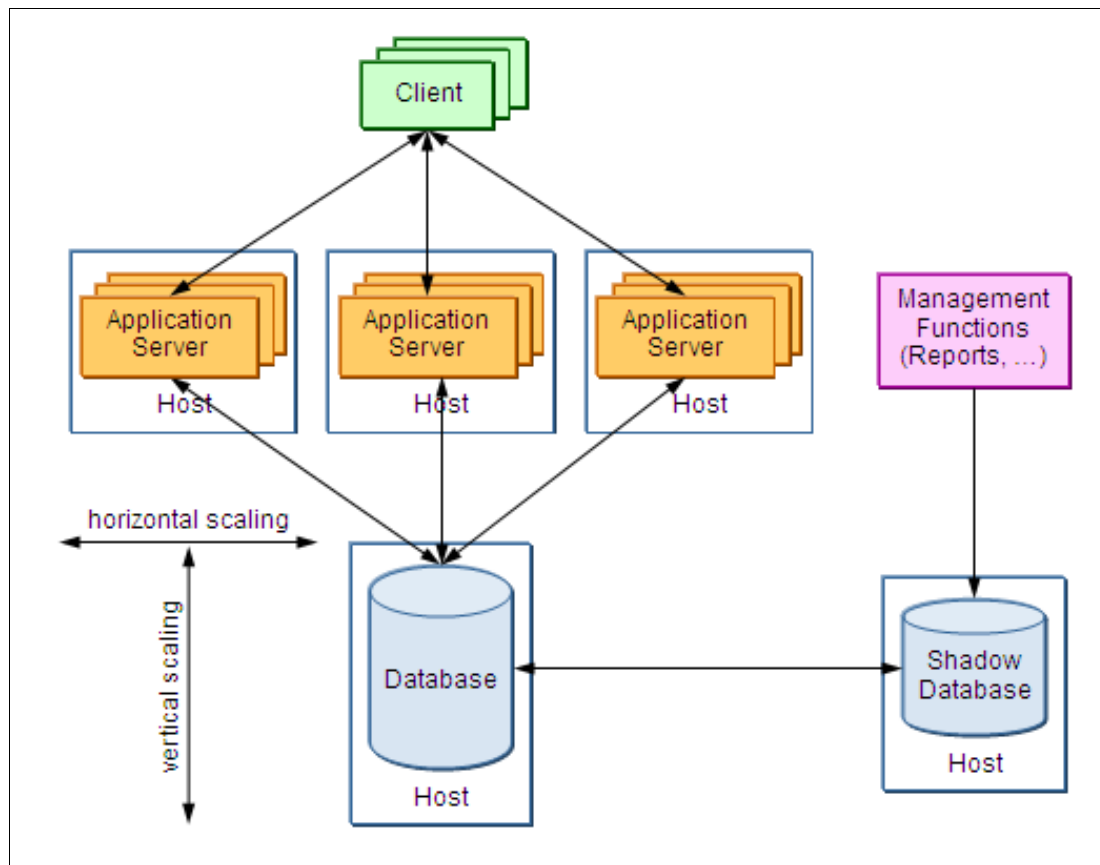


Figure 1-1 Scaling options in a traditional three tier application

Scaling is successful if all involved resources can cope with the increased load indefinitely. But at some point a resource reaches its maximum throughput, limiting the overall throughput of the system. This point is called the *saturation point* and the limiting resource is called a *bottleneck resource*.

Figure 1-2 shows the correlation between load and throughput that can typically be measured for an application.

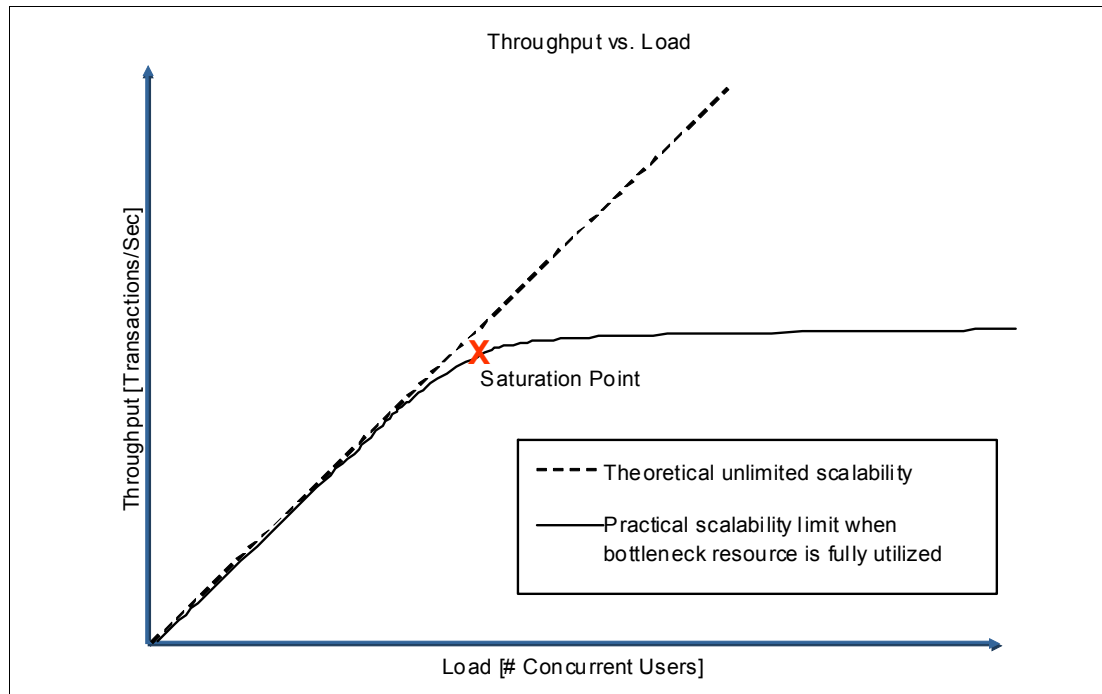


Figure 1-2 Correlation between throughput and load that shows scalability limits

In a well-crafted application, the database is often the limiting bottleneck resource. This is because when application servers become bottlenecks, they can be scaled out horizontally effectively. They scale out well because they primarily communicate only with the database, with little communication between each other. Scaling out a database is not nearly as easy nor effective.

When the load on the database increases, the usual response is to scale it up by adding more processors and/or memory to the database server. At some point, either because of practical, financial, or physical limits, enterprise databases are unable to continue to scale up. The next approach is to scale out by adding more database servers and using a high speed connection between them to provide a cluster of database servers. This, however, can be quite costly, and poses challenges in keeping the database cluster members synchronized.

It is important to ensure that the databases are kept synchronized for data integrity and crash recovery. For example, consider two concurrent transactions that modify the same row. When these transactions are run by different database servers, communication is required to ensure the atomic, consistent, isolated, and durable attributes of database transaction are preserved. This communication can grow exponentially as the number of involved database servers increases, which ultimately limits the scalability of the database back end. In fact, although examples of application server clusters using more than 100 hosts can be easily found, a database server cluster with more than four members is rare.

The scalability challenge, then, is to provide scalable access to large amounts of data. In almost all application scenarios, scalability is treated as a competitive advantage. It directly impacts the business applications and the business unit that owns the applications. This is because applications that are scalable can easily accommodate growth and aid the business functions in analysis and business development.

1.2 Caching

The obvious approach to solve the scalability challenge is to reduce the number of requests that are made to the bottlenecked resource or resources. Often the most practical way to accomplish this is by introducing a *cache*.

Cache A cache can be defined as a copy of frequently accessed data that is held in a relatively *nearby* location, such as within process memory. The intent of any caching mechanism is to reduce response time by reducing access time to data, and to increase scalability by reducing the number of requests to a constrained resource.

Virtually every major business application incorporates some caching techniques. A frequent example is a database cache, as illustrated in Figure 1-3. Here, the cache sits between the application and the database to reduce the load on the database.

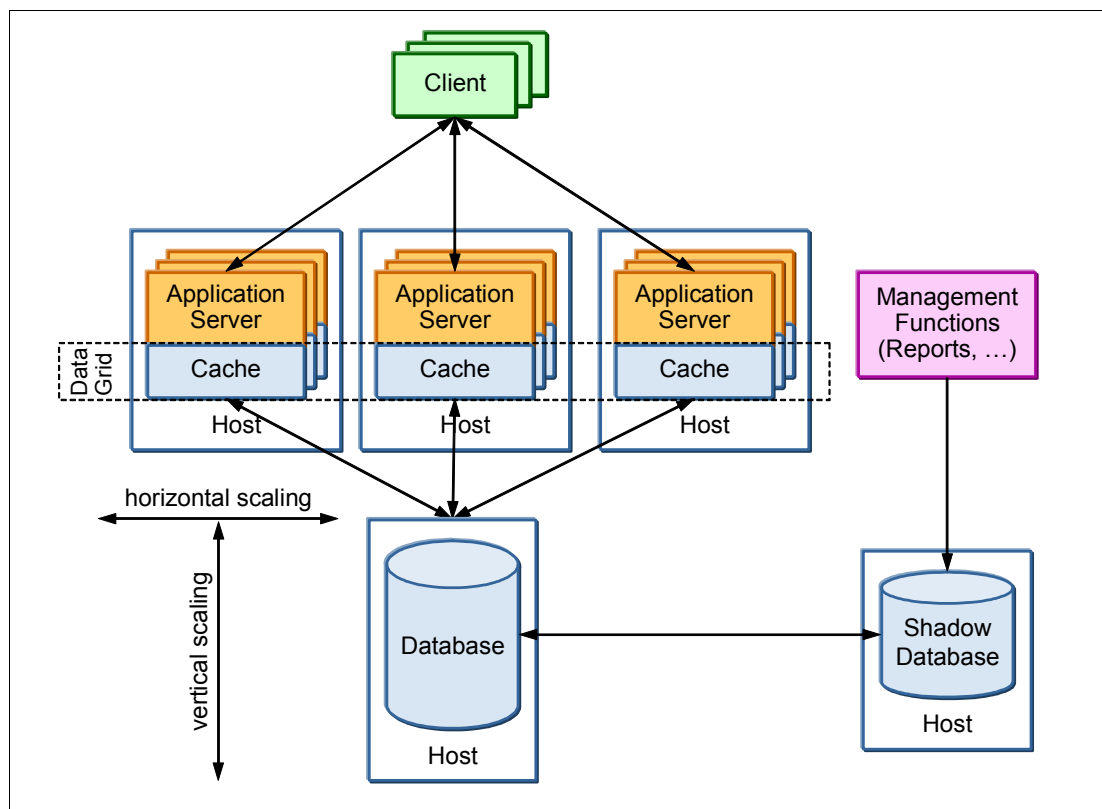


Figure 1-3 Introducing caching as a response to the scalability challenge

A cache is used to hold the syntax of prior requests to the resource and its response. This allows subsequent requests of the same syntax to be quickly served up from the cache rather than making another request to the resource.

Caches can be introduced at any tier, and in multiple places within each tier. In an end-to-end request from a browser to an e-commerce system, for example, there might be significant caching used:

- ▶ Within the browser, for images and CSS files.
- ▶ In a caching proxy within the user's company intranet, for identical image requests made by other users in that company.
- ▶ In an edge-of-network service as provided by, for example, Akamai.

- ▶ At the web server tier, for static content.
- ▶ In the application server tier, for caching various oft-used items, and copies of browser pages or page fragments already rendered for a request.
- ▶ In the database, caching past read and write data to avoid having to read them from disk.
- ▶ At other back-end systems that are used to complete the request. For example, use caches in a server that is used to handle a web service call to return appropriate municipal and state taxes for a locale.

Caching is a universally accepted and commonly implemented practice to increase performance while reducing system requirements. The appropriate uses of caching are ever expanding.

1.2.1 Holding the cached data

After you determine that caching is valuable at a particular tier, the next decision is where to hold the cached data. Usually the easiest and fastest place is within the local addressable memory of the component that is using the cache. But being easiest does not make it right for every circumstance.

At the application server tier in particular, the amount of cached data can be substantial, in the hundreds of megabytes. It often includes caching of HTTP session data, servlet/JSP page fragments, commands, inventory, prices, and many other application objects, all working to reduce back-end system loads while improving responsiveness. If the number or sizes of these caches can be increased, the *cache hit ratio* can be further improved, saving even more calls to constrained back-end resources.

The amount of cached data at the application tier can lead to several issues:

- ▶ How to ensure all *Java virtual machines* (JVMs) within the application server tier have a consistent copy of each cached object.
- ▶ How to propagate a *cache invalidation* from one JVM to the others so that none use a *stale cache* entry.
- ▶ Performing both of the above in a timely manner so that caches are synchronized as changes are made.
- ▶ How to hold all of the cached data within the limited application server JVM heap size.
- ▶ Excessive *garbage collection* (GC) pause times because of increased application server JVM heap size.

In a modern WebSphere Application Server Network Deployment environment, built-in *Data Replication Service* (DRS) features are available to address the first three bullets. But this service comes at extra cost in terms of processor and network bandwidth usage. As the number of application server JVMs in the cluster is increased, the DRS “network chatter” to keep all of the caches synchronized can become a significant portion of the overall work done by each JVM. A *point of negative return* can be reached, where adding another application server JVM to the cluster actually *decreases* responsiveness of the overall application.

Furthermore, if the cached data is held within the application server’s JVM heap, it can become constrained in the total size of its caches. In 32-bit mode, there is an addressability limit of around 2 GB as a usable heap size. Switching to 64-bit mode is not a cure either, as there can be excessive GC overhead as heap sizes grow over 4 GB or more. As the caches grow bigger, GC activity and pause times can rise to 10% or more of the total processor usage. Users can also experience excessive response time delays when lengthy *full-GC* operations freeze the entire JVM, sometimes for several seconds at a time.

1.2.2 Disk offload

Another refinement as the cache sizes become unwieldy is to *offload* some of the in-heap cached data onto disk. Disk offload functionality is also provided by a modern WebSphere Application Server Network Deployment environment's built-in dynamic caching (*DynaCache*) facilities. Dynamic cache can be configured to hold, for example, the 1000 most recently used cache items within the JVM heap. It then offloads the lesser-used cached content onto local disk or SAN.

This process can help to keep the JVM heap usage down to a manageable size. However, it involves another performance cost of writing to and reading from the disk, which is much slower than from memory. And cache invalidation still must be handled, which can now involve disk activity across all JVMs in the cluster.

1.2.3 Data grids

Taking the caching approach to the extreme leads to *data grids* as a scalable solution.

Grid A grid in general is a form of loosely coupled and heterogeneous computers that act together to run large tasks. To accomplish this task, a grid must be highly scalable. There are several different forms of grid, depending on the task at hand.

Data grid A data grid focuses on the provisioning and access of *information* in a grid manner, that is, using a large number of loosely-coupled cooperative caches to store data.

1.3 Introduction to WebSphere eXtreme Scale

WebSphere eXtreme Scale provides an extensible framework to simplify the caching of data that is used by an application. It can be used to build a highly scalable, fault tolerant data grid with virtually unlimited scaling capabilities beyond *terabyte* capacity. Because its capacity can be dynamically increased to an extreme size, it can be thought of as an *elastic cache*.

Figure 1-4 on page 7 illustrates many business and application challenges that suggest an elastic caching solution. WebSphere eXtreme Scale is the IBM response for this need in modern application environments.

WebSphere eXtreme Scale enables infrastructure with the ability to deal with extreme levels of data processing and performance. When the data and resulting transactions experience incremental or exponential growth, the business performance does not suffer because the grid is easily extended by adding more capacity (JVMs and hardware).

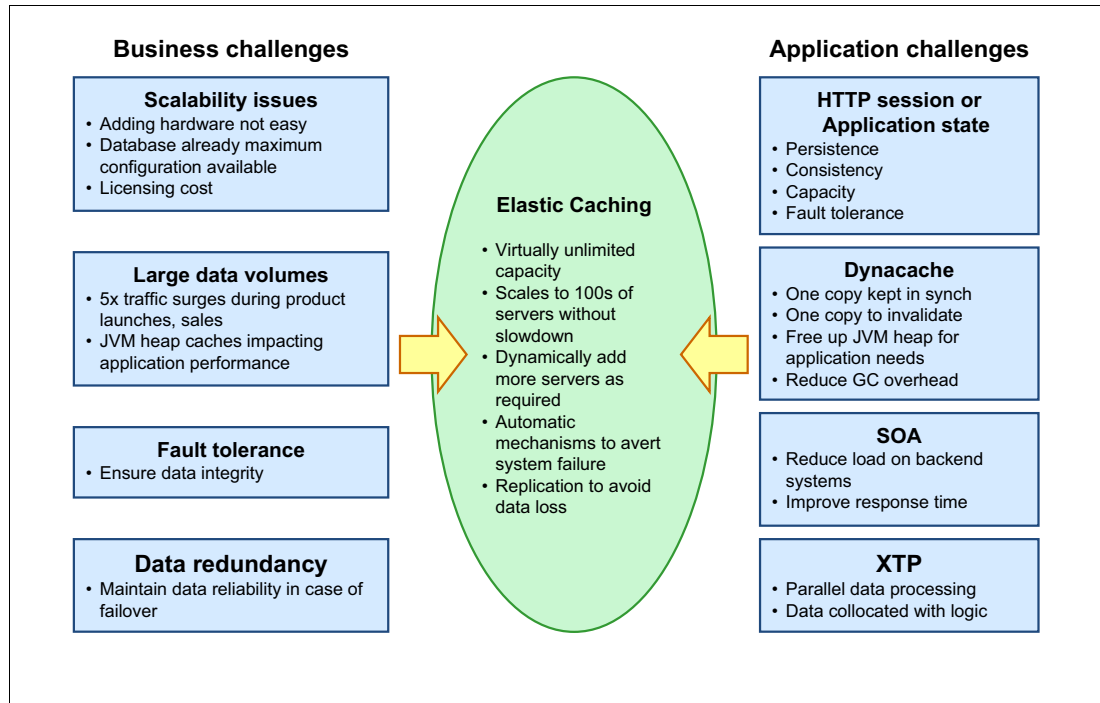


Figure 1-4 Challenges that lead to elastic caching

1.3.1 Architecture and terminology

This section provides a brief overview of how eXtreme Scale holds cached data in the grid and introduces some key terms and concepts. For more information, see Chapter 2, “Architecture and topologies” on page 27.

Map A *map* is a memory structure to hold key-value pairs. It allows an application to store and retrieve grid values that are indexed by *keys*.

Maps support optional indexes that can be used to index attributes of the key or value. These indexes are automatically used by the eXtreme Scale query engine to determine the most efficient way to run a query.

Partition For faster access to many keys, the set of keys in a map is typically split into a configurable number of *partitions*. It is also possible to configure a map that contains a single partition to hold all of your key/value pairs.

You can think of the entire set of keys as being *hashed* across the number of configured partitions, which is always a prime number.

Shard A shard represents a partition that is placed on a *container*. Multiple shards that represent different partitions can exist in a single container.

Each partition has an instance that is a *primary shard* and a configurable number of *replica shards*. The replica shards are either synchronous or asynchronous. The types and placement of replica shards are determined by eXtreme Scale by using a deployment policy, which specifies the minimum and maximum number of synchronous and asynchronous replicas.

Catalog The *catalog service* parses the deployment policy and grid configuration files. It uses their definitions to control placement of shards across the

available container servers (or simply *containers*) participating in the data grid. It also discovers and monitors the health of the containers, automatically balancing shard placement as necessary when a new container is added to the configuration, or if a container is stopped or killed.

In a normally functioning grid, there is little work for a catalog server to perform, and so it has little influence on scalability. It is built to service hundreds or thousands of container servers.

There can be one or several catalog servers running in your grid environment. However, only one is chosen automatically as the *master catalog*. All others run in standby mode in case of a failure of the master catalog server.

Container

A *container server* physically holds the application data for the grid. Each partition is represented by a primary shard hosted on a container, and optionally one or more replica shards that are hosted on other containers.

Any number of containers can be run on a host node (vertical scaling) and any number of host nodes can run extra containers (horizontal scaling).

A key feature of an eXtreme Scale grid is that more containers can be added dynamically on any node in the network. By specifying the same list of *catalog server end points* and the same grid configuration files, the new container startup is added to the same grid. It can therefore immediately participate in accepting primary and replica shards, adding to the capacity of the grid. Rebalancing of the shards across available containers is automatic, or can be requested manually at any time.

WebSphere eXtreme Scale uses logical placement rules (implemented by the catalog server) to ensure that replica shards are held in containers that are running on different host machines than the primary shard.

Client

Clients initially connect to the catalog service, retrieving a description of the current server topology. This connection is actually to a list of possible catalog servers. The current master catalog will respond.

Knowing the topology, the client logic can itself determine which key is in what partition, hosted on what container, and therefore access the appropriate container directly. Further catalog communication is no longer required unless there is an unexpected change in the container topology. Because changes rarely occur in practice, the clients are almost always talking exclusively and directly with the containers. This makes the communication protocol very efficient.

Clients can read or update keyed data within multiple partitions in a single transaction.

High availability

When the container topology changes because a new container is added or an existing container fails, the catalog service gets involved to update all client topology knowledge (known as the *routing table*) so that the client retries the request to the appropriate container. This automatic rerouting happens quickly and is transparent to the application.

If a container fails, an available replica of each of the primary shards originally hosted by the failing container is automatically promoted to

become the new primary shard. A fresh replica is then created on another available container so the system has the configured number of asynchronous and synchronous replicas.

If the master catalog server fails, one of the remaining catalog servers within the same catalog service domain is automatically promoted to become the new master catalog server. Although in this current eXtreme Scale release, extra catalog servers cannot be dynamically added to the configuration, the failed one can be restarted and act as another standby catalog.

For more information about how eXtreme Scale manages its routing tables and propagation to clients for failover, see 2.2.4, “Placement work and route table flow” on page 39.

Agents

With the WebSphere eXtreme Scale product, you can install and run user-supplied logic, which is known as an eXtreme Scale *agent*, on each shard in a grid. An agent has direct memory visibility to all key/value data within that shard, for retrieval or update. When grid data are partitioned, as is typical, a copy of the agent logic runs on every primary shard in parallel. In this way, eXtreme Scale provides powerful *grid computing* capability, where the processing logic is collocated with the data and run on parallel threads. This is also known as *eXtreme Transaction Processing (XTP)*. A tremendous amount of data processing potential is unleashed by this approach. For more information, see 4.5, “Server-side development with agents” on page 132.

eXtreme Scale has the following key features:

- ▶ Transaction support
- ▶ Securability
- ▶ Scalability
- ▶ High availability

These features are addressed in detail in the following sections.

1.3.2 Transaction support

Custom caching solutions often use a `java.util.Map` to store data. Updates are put into the map. However, a single user transaction can update several objects in the cache, and the whole transaction might be rolled back because of some business exception. The system must ensure that all changes are rolled back in the cache.

WebSphere eXtreme Scale has built-in transaction support for all changes that are made to the cached data. Changes are committed or rolled back in an atomic way. WebSphere eXtreme Scale augments the database and acts as an intermediary between the application and database. Transaction processing ensures that multiple individual operations that work in tandem are treated as a single unit of work. If even one individual operation fails, the entire transaction fails.

WebSphere eXtreme Scale uses transactions for the following reasons:

- ▶ To apply multiple changes as an atomic unit at commit time
- ▶ To ensure consistency of all cached data
- ▶ To isolate a thread from concurrent changes
- ▶ To act as the unit of replication to make the changes durable
- ▶ To implement a lifecycle for locks on change

WebSphere eXtreme Scale implements transactions by providing copies of cached objects to the application. Any changes to these copies by the application are tracked by a *difference map*. In a successful commit, the changes are applied to the objects in the grid. A locking strategy ensures that the data is not changed in between. Optimistic and pessimistic locking strategies are supported.

In a transaction rollback, the difference map is discarded and existing locks on the entries are released. As a result, no changes occur to the grid data.

For more information about eXtreme Scale transaction support, see *Integrating WebSphere eXtreme Scale transactions with other transactions* at:

http://www.ibm.com/developerworks/websphere/techjournal/1205_jolin/1205_jolin.html

1.3.3 Securability

Depending on the sensitivity of the data that is stored in the cache, security can be an important point to consider. As with a database, eXtreme Scale security provides fine-grained control on which client is allowed which grid data access. It includes the following security features:

- ▶ Authentication
 - Allows you to authenticate the identity of the requester or client. WebSphere eXtreme Scale supports both client-to-server and server-to-server authentication.
- ▶ Authorization
 - Provides the appropriate level of access control to authenticated clients. The authorization includes controlling operations such as reading, querying, and modifying the data in the grid. It also secures grid management operations such as replication of data, starting, stopping, and querying of grid containers.
- ▶ Transport security
 - Ensures secure communications between the remote clients and grid servers by using encrypted SSL protocol. Encrypted SSL communication is also used between catalog servers and container servers.
- ▶ System security
 - Includes overall system security for the access and operational management of the grid itself, such as issuing commands to start, stop, or query servers.

It is also possible to run your grid in an unsecured manner. This configuration is easier to configure and runs with somewhat reduced overheads. This approach is often acceptable where a network firewall is used to restrict communication with the grid so that only your otherwise secured application server clients can access the grid.

1.3.4 Scalability

As the product name suggests, eXtreme Scale supports substantial scale outs. It is designed to scale to thousands of grid containers, each of which are able to handle tens of gigabytes of data. That is terabytes of tested available capacity. If you find that you initially sized your grid too small, more grid containers can be added dynamically, adding capacity without interruption.

As explained in 1.1, “The scalability challenge” on page 2, the amount of communication between grid servers is a crucial limiting factor for scalability. The eXtreme Scale servers limit their communication to a bare minimum. This configuration allows large scale outs with

almost perfectly linear performance. That is, doubling your number of container servers gives you double the grid capacity with nearly equal grid performance.

Because clients communicate directly with the correct grid container for each key request, the number of containers does not affect performance whatsoever.

Necessary communication between grid containers occurs for two reasons:

- ▶ Availability management

Periodic communication occurs to track which containers are available. This communication is minimized by grouping the containers into chunks of around 20 in size, and by using small data packets for this “heartbeat” function.

- ▶ Data replication

Replication of cached content is used to ensure high availability in a container JVM or node failure. This is the only peer-to-peer communication between containers in the grid. Replication can be configured to be synchronous or asynchronous, or none at all, as the needs of the application failover behavior dictate. If no replication or asynchronous replication is specified, application writes to the grid do not wait before control is quickly returned to the application.

In tests, WebSphere eXtreme Scale ran smoothly with more than 1000 JVMs participating in a data grid managing more than a terabyte of data. Performance tests conducted did not identify any bottleneck, suggesting that it could have been increased still further. The scale out test was only limited by available hardware.

1.3.5 High availability

When the grid becomes the system of record, high availability is a crucial requirement. The grid must ensure that no loss of data will occur for a wide range of different failure cases. WebSphere eXtreme Scale accomplishes this by allowing for redundant copies of cache data, called *replicas*. An object can have a configurable number of replicas throughout the grid. A replica can be synchronous with the primary object (transactions commit only when all synchronous replicas are changed) or asynchronous (an update of replicas occurs after commit).

Furthermore, unique *zones* can be configured to ensure replicas are on different physical hosts (although this is done automatically for most topologies). And *Multi-Master Replication* (MMR) can be used to further replicate data between geographically distant data centers to ensure availability in disaster scenarios.

High availability requires careful planning, sizing, and configuration. More information about high availability and failure modes can be found in the WebSphere eXtreme Scale Version 8.6 Information Center at:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsavail.html>

1.4 Installation modes and deployment topologies

WebSphere eXtreme Scale provides for three possible ways of installing the product binary files, and for two possible ways of deploying its catalog and container servers. This section introduces these key concepts.

1.4.1 Installation modes

You install eXtreme Scale product binary files onto your host machines by using the IBM Installation Manager. In this process, you decide which of the following that you want to do:

- ▶ Install the client and server binary files into a stand-alone directory structure.

Using this installation choice, the eXtreme Scale binary files are installed into a new directory structure of your choice. There is no implicit knowledge nor integration with any existing application server environment, or any augmentation of any WebSphere Application Server profiles. This installation mode is called a *stand-alone installation*, as it provides no inherent management of your catalog and container JVMs. You are provided with some sample scripts (or Windows batch files) to start and stop your JVMs.

For more consideration about choosing this installation mode, see 5.1.1, “Benefits” on page 152 and 5.1.2, “Limitations” on page 152.

- ▶ Install the client and server binary files into an existing WebSphere Application Server environment.

In this installation mode, specify to the Installation Manager the location of your existing WebSphere Application Server environment. The required eXtreme Scale client and server binary files are installed into that directory structure. Specifically, they go into an `ObjectGrid` subdirectory within your `<was_install_dir>` location.

As a post-installation step for this type of installation, run a process called “Profile Augmentation”. Specify an existing WebSphere Application Server profile, or request that a new profile is created. In either case, that profile is extended with the following eXtreme Scale specifics, among others:

- New and modified pages of the WebSphere Administration Console that you can use to create a Catalog Service Domain for your eXtreme Scale catalogs, manage your dynamic cache offload location, and install support for eXtreme Scale HTTP session management.
- Integration with WebSphere Application Server security.
- Integration with the WebSphere Application Server’s Object Request Broker (ORB) provider, if ORB communication is used. However, use the new XIO communication if possible.

This installation mode is most commonly called a *WAS managed installation*, or simply *managed* or *integrated installation*.

Note: This is the only place in this book where you will see the term “WAS” used. It is shown here because you might see the term used in other documentation and wonder what it means. It is a convenient and oft-used acronym for the WebSphere Application Server, but is not officially sanctioned by IBM. Therefore, this book henceforth uses the term “Integrated installation”.

This is a convenient installation mode if you already have WebSphere Application Server installed and want to use the WebSphere Administration Console and `wsadmin` scripting facilities to configure and manage your eXtreme Scale topology.

An integrated installation also offers the benefit of the Performance Monitoring Infrastructure (PMI) facility and performance MBean capability within the WebSphere Application Server JVM environment. This means that you can, for example, examine PMI metrics to visualize your eXtreme Scale container thread pool usage. Or use one of the many third-party MBean collector tools to draw graphs of your container JVM heap usage.

This additional capability and convenience comes at some minor performance cost, however. With the eXtreme Scale containers running within a WebSphere Application Server provided JVM, a not insignificant amount of “extra stuff” is loaded into that JVM. This uses some additional JVM heap and increases certain code paths. In comparing a managed installation to an otherwise identical stand-alone installation, several customers have estimated a 5-10% reduction of eXtreme Scale grid performance. This is seen by many as a small price to pay for the additional convenience that the managed installation offers. However, you might consider it as wasting that 5-10% and prefer to extract as much performance as possible from your available system.

For more information about these and other considerations, see 5.2.1, “Benefits” on page 179 and 5.2.2, “Limitations” on page 182.

- ▶ Install only the client binary files into an existing WebSphere Application Server environment, with the server binary files installed elsewhere.

This installation process is similar to the integrated installation process described above. You again specify the location of an existing WebSphere Application Server installation directory, but this time choose to install only the eXtreme Scale client binary files there.

In this case, your eXtreme Scale servers (catalog and container JVMs) run as a remote deployment elsewhere, typically using a separate stand-alone installation. Profile augmentation is still performed so that you can use the WebSphere Administration Console to easily configure how your clients find and connect to the remote grid.

The client binary files that are used in this installation continue to integrate with the WebSphere Application Server security infrastructure and with its ORB, if ORB communication is used.

It is supported to install eXtreme Scale in multiple modes on the same machine. For example, an integrated client installation and a stand-alone server installation. And you can even have several stand-alone installations if you want, which must be in unique directories. Each binary installation acts independently, so they can be at different maintenance levels.

1.4.2 Deployment topologies

No matter which installation mode you choose, you have a further choice of how you start your eXtreme Scale container processes:

- ▶ Run within the same JVMs as the client

This is known as an *embedded deployment*, and can be further refined into three suboptions:

- Embedded-local

With embedded-local, the container logic runs within the same JVM as your application client and forms the entire grid. There is only one container forming your grid, and that container has only one partition.

This is useful in a development environment to test your usage of eXtreme Scale without having to set up a complete grid environment. It has no practical use outside of development and testing because it offers no advantage over the application managing data in its heap directly by using a standard HashMap.

- Embedded-partitioned

With embedded-partitioned, you typically define a cluster of several application servers, with each application server JVM also holding an eXtreme Scale container. Each of those containers participates as a peer to form your grid, with each container holding a

roughly equal number of partitions. You can also set up replication between partitions so that you have the benefit of high availability if a JVM fails.

The automatic management of the data across partitions in separate JVMs does have some advantages over what you might do with a HashMap. However, the data is still contained in your client JVMs so it offers no heap reduction. Also, in a WebSphere Application Server environment, its ObjectMap API used with the Data Replication Service offers high-availability. Therefore this deployment option has limited practical use, although it might offer more capability to a non-WebSphere Application Server environment.

- Embedded intra-domain

New with eXtreme Scale V8.6, the embedded intra-domain deployment option has a niche application of supporting a JPA Level 2 cache with higher performance than an embedded-partitioned topology. There is a single partition in each container that is a primary partition, as with embedded-local. However, the containers are configured to work together to form the overall grid. With all of those primary copies of the same data, there will be contention with their grid updates. Special “mediation” logic arbitrates between partition updates, deciding which one “wins” in an update race. Every JVM holds its own local copy of the data, so the JVM can read and write it directly within the JVM heap for best performance. The background mediation processing ensures that other primary copies are kept reasonably in sync.

- Run in their own JVMs

This is known as a *remote deployment* because the container servers run in JVMs different from (remote from) the application client JVMs. There is a complete separation between the client and grid JVMs, which offers several advantages in terms of isolation, client-side JVM heap reduction, and failover considerations.

Tip: In most cases, this is the preferred deployment topology.

One slight disadvantage when compared to an embedded-partitioned topology is that with a remote topology, every client request to the grid involves a TCP/IP network call to the container that hosts the wanted partition data. There is some performance cost to this network “hop.” However, modern systems and networks typically keep this under a few milliseconds, which is in the imperceptible “noise” range when considering a second or more for a typical website page rendering.

In an embedded-partitioned deployment with, for example, four application server JVMs, there is a one-in-four chance that the wanted partition for a grid GET or PUT operation was hosted within the same JVM making the request.

Note: This four-JVM example can be improved to approximately a one-in-two chance by using the `replicaReadEnabled=true` option of the grid configuration. This option allows the client to read from a replica copy as well as the primary copy. If the primary copy is not hosted in the same application server JVM, the replica copy might be, doubling the odds of reading local memory for a GET request. The caveat with `replicaReadEnabled=true` is that an asynchronous replica is slightly “stale” compared to the primary. But in many application usage scenarios, this is not of any practical concern.

This situation offers a fast path direct memory access to that partition content. The possibility of a direct JVM heap read is sacrificed with the remote topology. However, this slight disadvantage can be nullified by providing for a *near cache*, as described in 2.7.5, “Remote cache (with optional near cache)” on page 59.

In summary, with the possible exception of development and test environments, select a remote deployment unless there is a requirement or a proven advantage in running an embedded deployment topology.

1.4.3 Mixing and matching

An interesting yet confusing feature of eXtreme Scale is that it places virtually no limits on how you decide to combine your installation and deployment decisions. You can mix them in whatever way makes sense for your particular needs.

There is also no requirement that your catalog servers must be deployed in the same environment or topology as the container servers. You can, for example, host your catalog servers in an integrated remote topology on your WebSphere Application Server node agent JVMs, while the container servers run as stand-alone remote JVMs.

The following are the most commonly used topologies:

- ▶ Integrated client using embedded integrated servers

This is popular for a development environment, perhaps within an Eclipse framework, where you run your application logic, an eXtreme Scale catalog, and an eXtreme Scale container all within the same JVM.

- ▶ Integrated client using a remote integrated server

In this topology, you install the eXtreme Scale client and server binary files into an existing WebSphere Application Server environment, and augment the Deployment Manager and Application Server profiles. You then configure an application server cluster to run your integrated catalog servers, and another cluster to run your integrated container servers.

Finally, you configure a Catalog Server Domain to define how the eXtreme Scale client logic that is used by your existing application server JVMs connects to the integrated catalog cluster that you set up. The catalog server then tells the client how it communicates with the integrated container server cluster.

This is still termed a *remote deployment* topology, even though all of the server JVMs (application, catalog, and container) are running within the same WebSphere Application Server environment. It is “remote” because the containers are running in separate JVMs from the client application JVMs.

An example of this common topology is used in 5.2.5, “Introducing the sample topology” on page 187.

- ▶ Integrated client using a remote stand-alone server

This is a typical topology for customers who are running WebSphere Application Server Network Deployment, and want to get the maximum performance from their remote grid. This is instead of running the grid containers in the integrated environment, with the slight performance penalty as mentioned earlier.

Install the eXtreme Scale client into an existing WebSphere Application Server environment. As with the previous example, you then configure a Catalog Server Domain to connect to your remote catalog server and container servers.

Those remote JVMs are typically provided by a stand-alone installation for maximum performance. However, it is also possible that they are hosted by a separate WebSphere Application Server cell.

- ▶ Stand-alone client using a remote stand-alone server

This topology is commonly used for environments using a Java Platform, Enterprise Edition application server technology other than WebSphere Application Server such as a

Liberty profile or an Apache Tomcat application server. The application uses the client binary files that are supplied by the stand-alone installation to connect to and use the grid catalog and container JVMs supplied by the same stand-alone environment binary files.

For an example of this topology, see 6.6, “Scenario 3: Implementing with Apache Tomcat” on page 288

1.5 What is new in WebSphere eXtreme Scale Version 8.6

This book is an update to *User's Guide to WebSphere eXtreme Scale*, SG24-7683-00, which was based on WebSphere eXtreme Scale version 6.1. Many enhancements have been made to the WebSphere eXtreme Scale product, and this second edition brings things up to date current with version 8.6.

There have been so many enhancements and new features that it is no longer practical to address all of them in a comprehensive user's guide. Instead, the focus is on the key concepts that you need to know, and the most common usage patterns being used. Hence the retitling of this book to *WebSphere eXtreme Scale v8.6 Key Concepts and Usage Scenarios*, SG24-7683.

Each of the WebSphere eXtreme Scale Information Centers for the intervening product releases (v7.1, v7.1.1, v8.5, and v8.6) provides a “What's new in this release” section. See the following links for the complete detail of each WebSphere eXtreme Scale release:

- ▶ WebSphere eXtreme Scale v7.1 Information Center, What is new in this release:
<http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1/topic/com.ibm.websphere.extremescale.over.doc/rxsfeatures.html>
- ▶ WebSphere eXtreme Scale v7.1.1 Information Center, What's new in Version 7.1.1:
<http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1m1/topic/com.ibm.websphere.extremescale.doc/rxsfeatures.html>
- ▶ WebSphere eXtreme Scale v8.5 Information Center, What's new in Version 8.5:
<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/topic/com.ibm.websphere.extremescale.doc/rxsfeatures.html>
- ▶ WebSphere eXtreme Scale v8.6 Information Center, What's new in Version 8.6:
<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxsfeatures.html>

This section covers only the most important of those many updates and changes, which are broadly categorized as follows:

- ▶ Architectural enhancements
- ▶ Programming enhancements
- ▶ Monitoring and management enhancements

1.5.1 Architectural enhancements

The following are some of the key architectural enhancements:

- | | |
|----------------------|--|
| XIO transport | A new network transport protocol, <i>eXtremeIO</i> (XIO) offers significantly better performance and throughput than the original ORB protocol. For more information, see 2.1.5, “Transport protocol: IBM eXtremeIO (XIO)” on page 34. |
|----------------------|--|

XM	By configuring <i>eXtremeMemory</i> (XM), you can store objects in native memory instead of within the Java heap. By moving objects off the Java heap, you can avoid garbage collection pauses, leading to more constant performance and predictable response times. For more information, see 2.8.3, “IBM eXtremeMemory” on page 67.
XDF format	WebSphere <i>eXtreme Data Format</i> (XDF) is a new, more efficient means of representing object data. It is specifically optimized for the way that eXtreme Scale stores objects in the grid. The new format is a prerequisite for the introduction of support for Microsoft .NET, leading to the support for a true enterprise data grid. For more information, see 2.8.2, “IBM eXtreme Data Format” on page 66.
Liberty profile support	The Liberty profile is a highly composable, fast to start, dynamic application server runtime environment. WebSphere eXtreme Scale is easily added to a Liberty profile by the selection of certain eXtreme Scale features. For more information, see 5.2.8, “Testing the sample topology and application” on page 227.
JTA	WebSphere eXtreme Scale provides support for the Java Transaction API (JTA). Through JTA, client management is simplified by using Java Enterprise Edition (Java EE). Through support from an eXtreme Scale resource adapter, applications can look up eXtreme Scale client connections, and demarcate local transactions using Java EE local transactions or the eXtreme Scale APIs. For more information, see 4.8.2, “JTA transactions” on page 148.
Intra-domain for JPA	By configuring an intra-domain topology for a JPA L2 cache, a primary shard is placed on every container server in the configuration. Each primary shard contains the entire content of the partition. Performance is increased because clients can locally read and write to their own primary copy. You can find more information about this enhancement in the WebSphere eXtreme Scale Version 8.6 Information Center at: http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsjpacache.html#cxsjpacache__intradomain
MMR	<i>Multi-Master Replication</i> (MMR) can be configured to asynchronously replicate grid data between geographically distant data centers. This supports failover to a disaster recovery site with its grid content already current, satisfying another key requirement for high availability. For more information, see 2.8.4, “Multi-master replication” on page 69.

1.5.2 Programming enhancements

The following are some of the key programming enhancements:

Microsoft .NET support	Enterprise data grids use the XIO transport protocol and the XDF data format. With the new transport and serialization format, you can share data between Java and Microsoft .NET clients that are connected to the same data grid. A native .NET programming interface for accessing the data grid is now
-------------------------------	--

provided. For more information, see 2.4.4, “Microsoft .NET” on page 53.

- REST gateway** The *Representational State Transfer* (REST) gateway provides non-Java clients access to eXtreme Scale grid data. For more information, see 2.4.6, “REST gateway” on page 54.
- Near cache invalidation** You can configure near cache invalidation to remove stale data from the near cache as quickly as possible. When an update, deletion, or invalidation operation is run against the remote data grid, an asynchronous invalidation is triggered in the near cache. For more information, see 4.6.5, “Using near cache invalidation in WebSphere eXtreme Scale version 8.6” on page 140.
- Upsert method** The `upsert` and `upsertAll` methods can be used more efficiently than the original `ObjectMap` `put` and `putAll` methods. The original methods required that a `getForUpdate` be run before the `put/putAll`. With the new methods, only one call is made, reducing path length and lock contention on the grid. Example 4-7 on page 111 shows an example.
- Lock method** When you use pessimistic locking, you can use the `lock` method to lock data, or keys, without returning any data values. In previous releases, you used the `get` or `getForUpdate` APIs to lock keys in the data grid.
- Spring annotation** With Spring cache abstraction, you can enhance your existing Spring application to use eXtreme Scale as the cache provider. For more information, see 2.4.3, “Spring framework” on page 51.
- Continuous query** You can use continuous query to be automatically notified in your client application when data is added or changed in the data grid. For more information, see 4.4.4, “Continuous query” on page 131.
- Default local index** To iterate through all keys and values in a local map, an agent (running on each shard in the grid) can use the default index. This index does not require any configuration, and is always available. For more information, see the WebSphere eXtreme Scale Version 7.1 Information Center at: <http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1/topic/com.ibm.websphere.extremescale.prog.doc/cxsidxnonkey.html>.
- Global index** The global index extends the built-in `HashIndex` plug-in, and runs on shards in a distributed, partitioned data grid. Global index tracks the location of indexed attributes in the data grid, and provides an efficient means of finding partitions, keys, values, and entries in the grid. For more information, see 4.4.2, “Optimizing query performance using the global index” on page 127.
- Multi-partition transactions** WebSphere eXtreme Scale now supports updates to multiple partitions in a data grid within a single transaction. In prior releases, only a single partition could be updated within a transaction. For more information, see 4.8.3, “Multi-partition transactions” on page 148.

OSGi support

Using the OSGi framework, you can expose your plug-ins as OSGi services so they can be used by the eXtreme Scale runtime. In addition, you can start eXtreme Scale servers and clients in an OSGi container, so you can dynamically add and update eXtreme Scale plug-ins to the runtime environment. For examples of using OSGi packaging, see Chapter 4, “Developing with WebSphere eXtreme Scale” on page 95, especially 4.4, “Querying data using Object Grid Query Language” on page 122.

1.5.3 Monitoring and management enhancements

Following are some of the key monitoring and management enhancements:

xscmd	The <code>xscmd</code> utility is the new supported version of the original (and not formally supported) <code>xsadmin</code> utility. For more information about the commands that are provided by the <code>xscmd</code> utility, see 7.2, “Command line access: <code>xscmd</code> ” on page 310.
Web console	The graphical web console provides current and historical views into eXtreme Scale server statistics. You can view the keys of the cache entries in your data grid in the console (and with the <code>xscmd</code> utility). You can run UNIX-style regular expressions to query keys and their values from your grid. And you can invalidate sets of data based on the regular expression or partition. For more information, see 7.1, “The web console” on page 304.
Log analyzer	With the new <code>xsloganalyzer</code> tool, you can generate reports from your log files that can help you analyze the performance of your environment and troubleshoot issues. For more information, see 7.3, “Log analyzer” on page 324.
Message center	The message center provides an aggregated view of event notifications for log file and first-failure data capture (FFDC) messages. You can view these event notifications in the web console, the <code>xscmd</code> utility, or programmatically with MBeans. The message center is described in the WebSphere eXtreme Scale Version 8.6 Information Center at: http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxshealthmonitor.html .

1.6 Entry points for WebSphere eXtreme Scale

This section addresses how eXtreme Scale can be implemented into an existing IT infrastructure by showing possible entry points, and providing a decision tree based on typical problems an organization might face.

1.6.1 Possible entry points

Figure 1-5 on page 20 shows three possible entry points for adopting eXtreme Scale. It can be used in scenarios that range from a simple in-process cache to an enterprise-wide distributed data grid. The diagram also provides a roadmap. An organization can start with one of the lower entry points and evolve continuously to the higher levels of *grid computing*.

The first entry point uses eXtreme Scale as a sophisticated caching layer for an application. This proven and well-supported IBM product can be used instead of investing in the custom development of a home-grown solution. The replacement or augmentation of an existing caching implementation that has reached its limits (for example, transactionality, scalability, or security) is also found in this category.

The second entry point uses eXtreme Scale as a data grid to store large amounts of data. Although a traditional in-memory data cache has a practical size limit, a data grid can be configured to hold a tremendous quantity of data. Furthermore it is fault-tolerant and can be dynamically extended to accommodate an ever-increasing amount of cached data as your needs grow.

The third entry point uses the scalability eXtreme Scale supports to build an enterprise-wide complex data grid solution. It can include parallel grid-style computing by bringing the algorithms to the data. This is as opposed to the traditional distributed computing approach where the algorithms retrieve the required data from a back end.

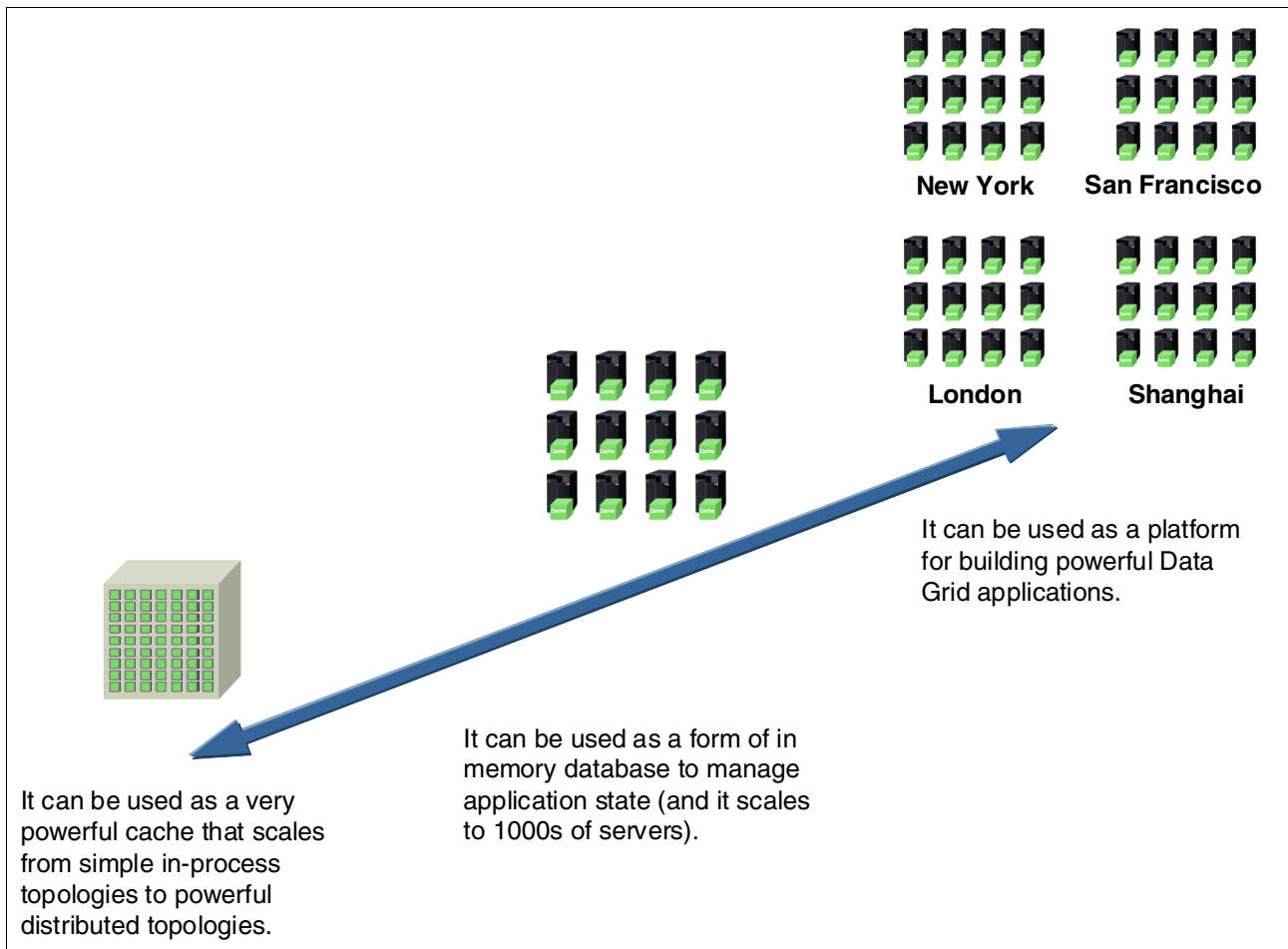


Figure 1-5 Entry points for adopting WebSphere eXtreme Scale

1.6.2 Decision tree

When you consider adopting eXtreme Scale as an enterprise-wide object caching grid, engage the IT department in detailed systems and design analysis. This is an important exercise in determining a roadmap for adoption of any new technology. Depending on the issues at hand and the findings from the analysis, the decision tree that is shown in Figure 1-6 shows possible solutions based on eXtreme Scale.

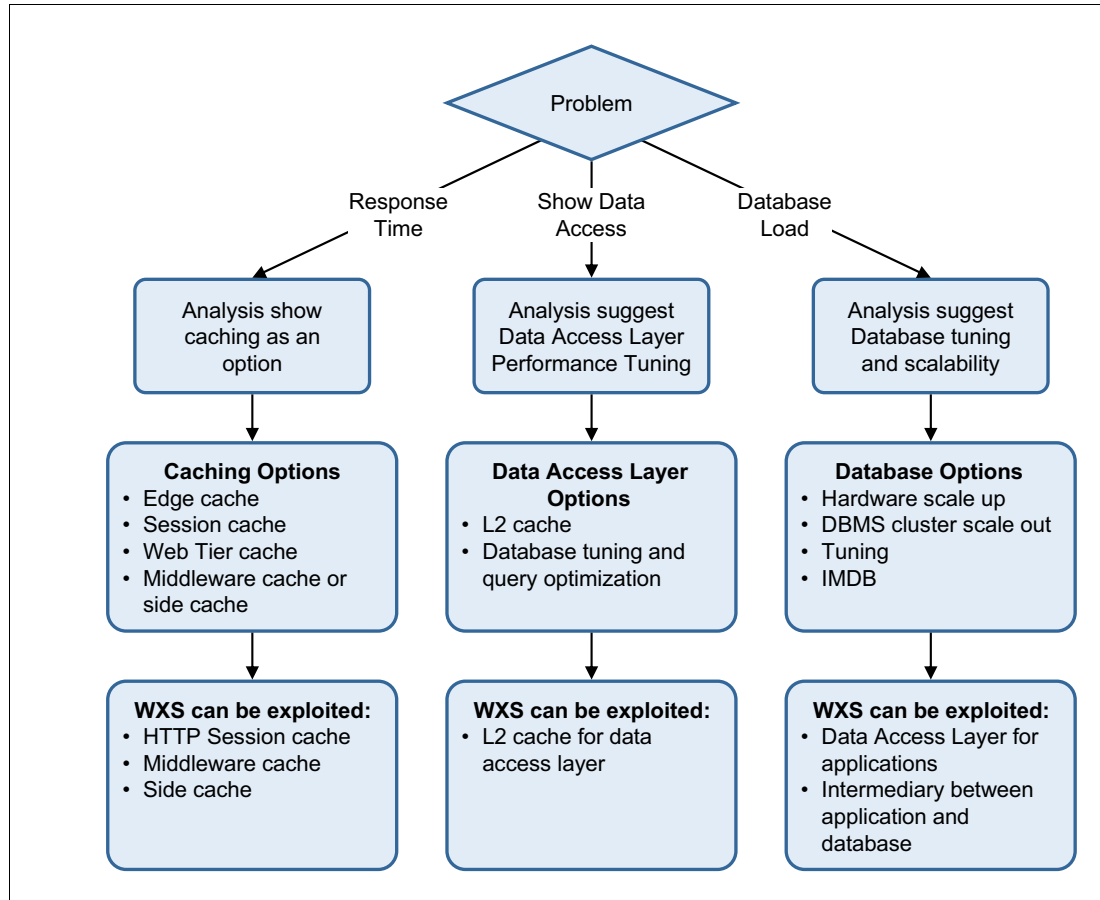


Figure 1-6 Decision tree for adopting eXtreme Scale

The decision tree focuses on the scalable distributed computing platform. With the ever-changing landscape of enterprise computing, a modular approach toward solving scalability issues becomes imperative. This necessity arises from the manageability and performance requirements of the distributed components.

The decision tree breaks down the problem components into following three broad categories:

► Caching

The intent of any caching mechanism is to improve performance by enabling easy access to data, which, in the absence of a cache, must be fetched from a database. Repeated access to a database for data can be computationally expensive and might impact the overall application performance. Caching can also be employed as a front-end web tier, and at the edge for static application contents. WebSphere eXtreme Scale can be

employed to cache HTTP sessions, which are typically cached in the same address space as the application. For more information about the HTTP session and side cache scenario, see Chapter 3, “Application scenarios” on page 75.

► Data access layer

Also known as the data persistence layer, the data access layer provides access to persisted data that is stored in a database. The data access layer is used by the applications to access the data. This layer relieves the application from dealing with the complexities inherent in this access. Because eXtreme Scale stores the data in the form of an object in the grid, the data access layer potentially acts as a loader. It therefore manages the entity relationship (Object Relational Mapping [ORM]) and converts the raw data into Java objects that can be stored in the grid.

The data access layer can be either custom-built Java Connection Architecture (JCA)-based or standard JPA-based. Other commercial ORMs are also commonly used as data access layers. WebSphere eXtreme Scale can be used with data access layers in various scenarios, such as a L2 cache or even a layer above the ORM layer, as a data access layer itself. For more information about data access layer scenarios, see Chapter 3, “Application scenarios” on page 75.

► Database scalability

Traditionally, databases were notorious culprits in hindering scalability, and were considered incapable of meeting the needs of a high-performance distributed computing design. With the new advancements in technology around the hardware that hosts the databases, network (10 GigE, and so forth), and disk access technologies (Fibre Channel, and so forth), coupled with the advancements with the multi-processing DBMS technologies, the databases have risen to the scalability challenge. WebSphere eXtreme Scale as a data grid technology enhances rather than replaces the role of an enterprise database.

WebSphere eXtreme Scale acts as an intermediary to the enterprise database. By reducing the access to the databases, it not only improves the application performance, but also relieves the database for business activities. These activities can include business intelligence analysis, data mining, and data analysis. This approach allows for an enterprise to save costs, serve as the focus from database scalability shifts to data grid scalability, and easily adapt to growth in data demand. For more information about WebSphere eXtreme Scale as a network-attached cache scenario, see Chapter 3, “Application scenarios” on page 75.

WebSphere eXtreme Scale is a versatile platform for extending scalability across all components of the enterprise architecture.

1.7 Comparing WebSphere eXtreme Scale to in-memory databases

Introducing a caching layer or data grid is not the only solution to address the scalability challenge described in 1.1, “The scalability challenge” on page 2. Another viable solution is an *in-memory database* (IMDB). This section compares a data grid solution to IMDBs.

1.7.1 Introducing IMDBs

An *in-memory database* (IMDB) has all the qualities of a traditional *relational database management system* (RDBMS), but is entirely in memory, eliminating the need for a database. Although this notion of a purely in-memory database pleases technologists, it has never

settled well with the business community that usually own the data and the application. The reasons for this dissatisfaction are simple. A business has requirements and functions such as audits, history, legal requirements, operations, and business analysis, all of which require data to be *persisted* for subsequent retrieval and mining.

IMDBs attempt to bring data closer to the application. An IMDB solution involves holding an entire database in memory, as a single entity. The application treats the IMDB layer as the primary database, and the IMDB is backed by a relational database. The advantage of this approach is faster data access times.

IBM solidDB® is a current example of a highly performance and capable IMDB. For more information about solidDB, see *IBM solidDB: Delivering Data with Extreme Speed*, SG24-7887.

IMDB solutions provide all the database-like enterprise service quality features such as ACID (Atomic, Consistent, Isolated, Durable) transactions, high availability, fail over, clustering, and SQL support.

Although IMDB technology provides much needed relief to address costly scalability issues and business needs, it does have some limitations. An IMDB can hold only a finite amount of data because the data must fit into a single address space. To address this issue, applications and data can be partitioned according to a relevant business need. For example, you can create partitions based on customer location, with one installation for east coast customers and another for west coast customers.

1.7.2 Explaining the difference

Figure 1-7 provides an overview of the difference between a data grid and an IMDB.

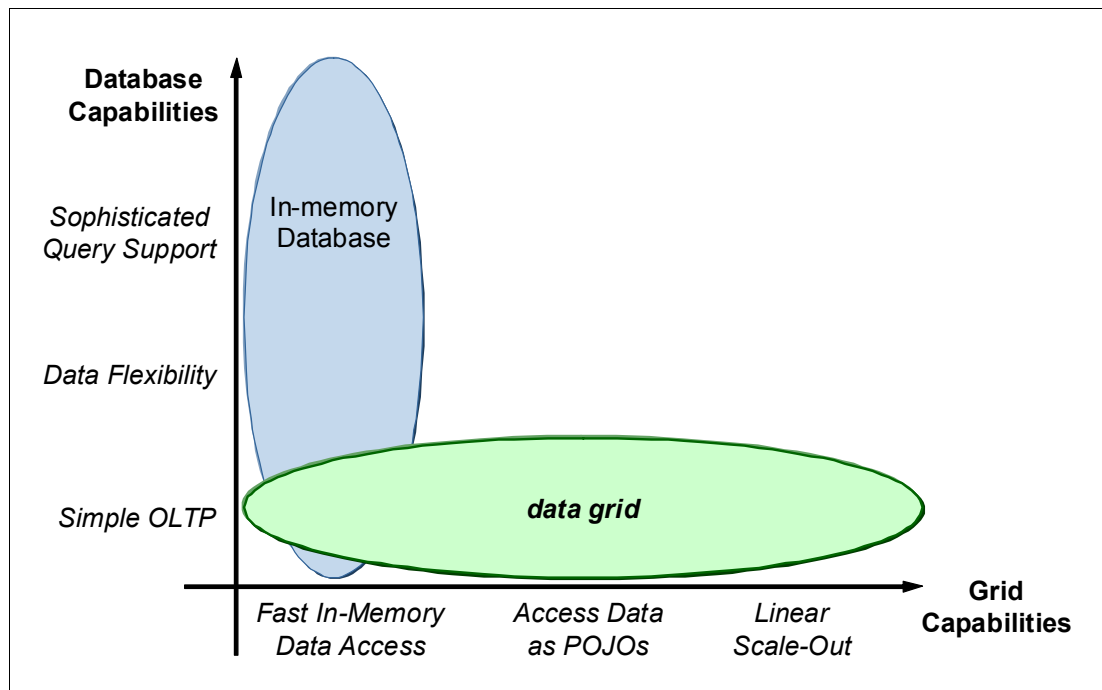


Figure 1-7 IMDB versus data grid

WebSphere eXtreme Scale is considered an *in-memory data grid* as opposed to an IMDB. The distinction is because eXtreme Scale is intended to compliment a database and not

compete with it. WebSphere eXtreme Scale provides an in-memory store of objects that is distributed and closer to the application. It provides the enterprise with a caching fabric that manifests itself as a grid that is a layer before the database.

The grid provides a scalable infrastructure for data that can expand as the needs of the application grow. Because the grid is self-managed, it can grow with the environment by adding new JVMs to the grid until there is a single partition or shard per JVM. This differentiating feature makes the adoption of eXtreme Scale attractive because grid management and maintenance becomes a routine administrative task.

Although data grids scale linearly and provide an enterprise-wide shock absorber to the back-end database, they also have their limitations. For instance, the data that are contained in a data grid are stored in the form of an object, compared to raw data stored in an IMDB. Therefore, a data grid must work with a persistence framework to convert the data into objects while pulling it into the grid. Similarly, it must be converted to raw data while writing it back to the database. This adds another framework to the overall solution, which might add extra architectural planning.

Data grids do not support a full SQL syntax for complex queries of the data. Instead, eXtreme Scale supports its simpler *Object Grid Query Language* (OGQL) which is similar to the Java Persistence Query Language (JPQL).

Although data grids and IMDBs differ in their approach, it is important to understand the pros and cons of these data caching technologies when you select an in-memory data cache solution.

1.8 IBM WebSphere DataPower XC10 Appliance

Much of the functionality that is provided by WebSphere eXtreme Scale software product is also available in an easy-to-install “appliance” called the IBM WebSphere DataPower XC10 Appliance. This section provides a brief overview of the WebSphere DataPower XC10 Appliance. It also provides guidance when you might want to consider it rather than installing the WebSphere eXtreme Scale software on your systems.

The IBM Redbooks website lists several documents that talk in detail about the WebSphere DataPower XC10 Appliance, including *Elastic Dynamic Caching with the IBM WebSphere DataPower XC10 Appliance*, SG24-4851. This book contains a chapter with a detailed chart that compares and contrasts the features available in the WebSphere eXtreme Scale software and the WebSphere DataPower XC10 Appliance. Figure 1-8 on page 25 provides a simple flowchart to determine the cases where a WebSphere DataPower XC10 Appliance can be used as an alternative to a WebSphere eXtreme Scale software installation.

The key point to consider about the WebSphere DataPower XC10 Appliance is that, as a pre-configured and secure ready-to-go “black box”, it does not allow you to install any software onto it. You can use its administration console to configure it as you want, but you cannot install any of your own software on it. This means that you cannot use some of the more advanced features that you might want to use eXtreme Scale for, such as in-line loaders, agents, or parallel grid processing. Each of these advanced features requires that your unique code is available on the grid side.

However, there are still a vast number of more common tasks at which the WebSphere DataPower XC10 Appliance excels, including all of the typical *side-cache* scenarios. As such, you can use it for caching HTTP session data, offloading dynamic cache data, or as an application side-cache. As with the eXtreme Scale software installation, the WebSphere DataPower XC10 Appliance supports transactions, scalability, security, and high availability.

If you want to explore the capability of the WebSphere DataPower XC10 Appliance before purchasing one, the IBM Elastic Caching Community blog has announced the availability of a virtual image of the appliance for download:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/introducing_a_developer_xc10_appliance_virtual_image

At the time of writing this book, the image of the WebSphere DataPower XC10 Appliance V2.5 was based on WebSphere eXtreme Scale V8.6, so it matches the software capabilities outlined in this book.

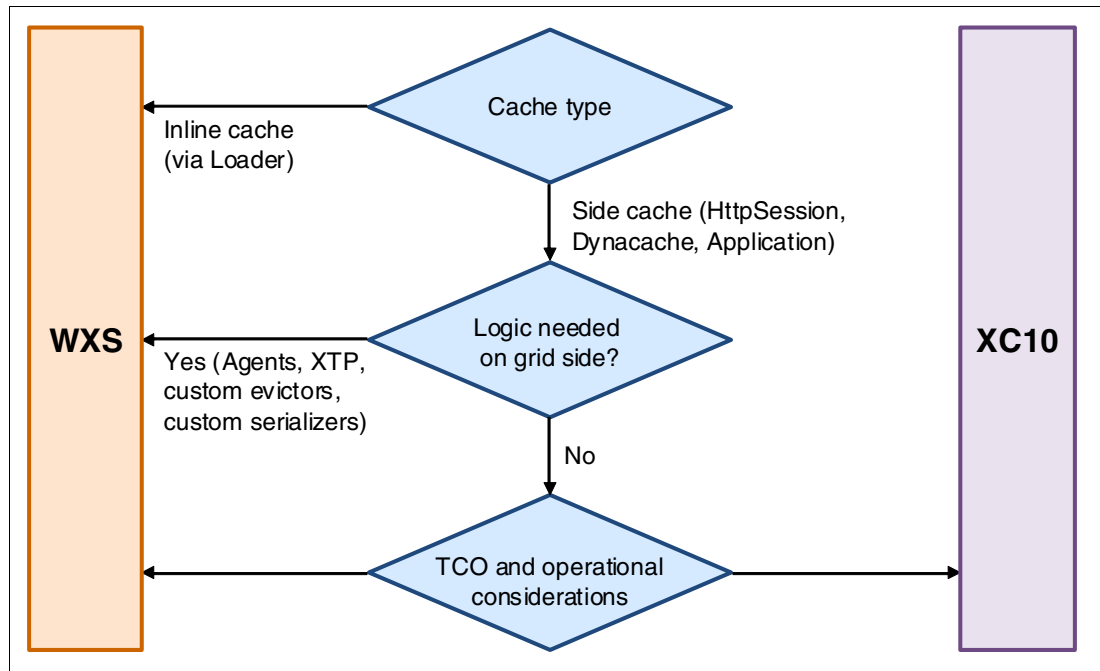


Figure 1-8 Simple flowchart for determining if a DataPower XC10 appliance can be used for caching

After your decision flow gets you to the final TCO and operational considerations diamond shown in Figure 1-8, you might decide on a solution for numerous reasons:

- ▶ A WebSphere DataPower XC10 Appliance offers a “plug in and go” solution, compared to the “install and configure” steps of the WebSphere eXtreme Scale software.
- ▶ If you already have extensive monitoring infrastructure for existing JVMs in your enterprise, the WebSphere eXtreme Scale software solution might be more attractive.
- ▶ The WebSphere eXtreme Scale software solution is licensed by processor value units (PVUs), as with most other IBM software. The WebSphere DataPower XC10 Appliance is instead purchased outright, with a smaller yearly support cost. Talk to your IBM sales representative for the details.



Architecture and topologies

This chapter provides an overview of the architecture of the IBM WebSphere eXtreme Scale product. It explains the terminology for the main components, which in turn provides a strong basis for defining eXtreme Scale topologies.

This chapter includes the following sections:

- ▶ “WebSphere eXtreme Scale architecture” on page 28
- ▶ “Catalog server” on page 35
- ▶ “Replication: Primary and replica shards” on page 44
- ▶ “APIs used to access the grid” on page 48
- ▶ “A simple example” on page 54
- ▶ “Scalability sizing considerations” on page 56
- ▶ “Common topologies” on page 57
- ▶ “Common topology options” on page 60

2.1 WebSphere eXtreme Scale architecture

This section addresses the basic concepts required to understand how a grid is structured and how applications use the grid.

2.1.1 Grid architecture

Figure 2-1 on page 29 is an illustration of the following components that are used to define a grid:

- ▶ Java virtual machine

A Java virtual machine (JVM) is an execution environment that is platform independent. In the context of eXtreme Scale, a JVM can host one or more grid containers. A JVM can be either an application server or a stand-alone JVM.

- ▶ Grid

WebSphere eXtreme Scale forms a grid for caching across a loosely coupled network of grid containers. The term *grid*, as it applies to WebSphere eXtreme Scale, means that eXtreme Scale emulates grid-like distribution and self-governance to allow a set of JVMs to act like a single entity. The grid layer provides qualities of service such as scalability and replication of data.

- ▶ Grid containers

Like a typical container in a Java EE context, grid containers provide grid application services such as security, transaction support, JNDI lookup service, remote connectivity, and so on. The grid containers house shard distribution and placement, and enable easy management of the grid infrastructure. Like other containers such as Web and EJB containers, a grid container can also take advantage of the configuration service provided by the WebSphere Application Server infrastructure in a managed environment.

- ▶ Partitions

Partitioning is the process of splitting data into smaller sections. Partitioning allows the grid to store more data than can be accommodated in a single JVM. The data is partitioned by using an application-defined schema. The grid can have many partitions, depending on the application, and these partitions must be factored in when you configure and design for a scalable infrastructure.

- ▶ Shards

The term *shard* is used to define a single instance of a partition. Each partition has a primary shard and an optional set of replica shards. The shard distribution algorithms ensure that the primary and replica shards are never in the same container to ensure fault tolerance and high availability.

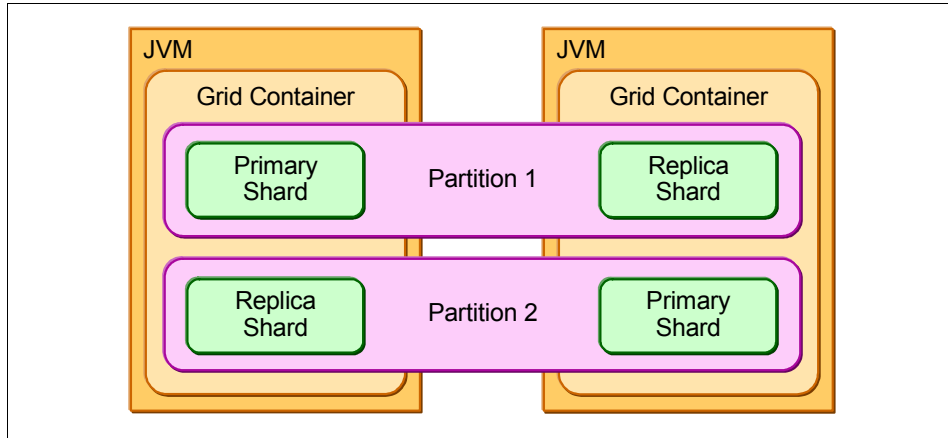


Figure 2-1 WebSphere eXtreme Scale components

Figure 2-2 shows four partitions with a single replica each, making eight shards in total. All of the shards are in two grid containers. In this case, each grid container holds four shards.

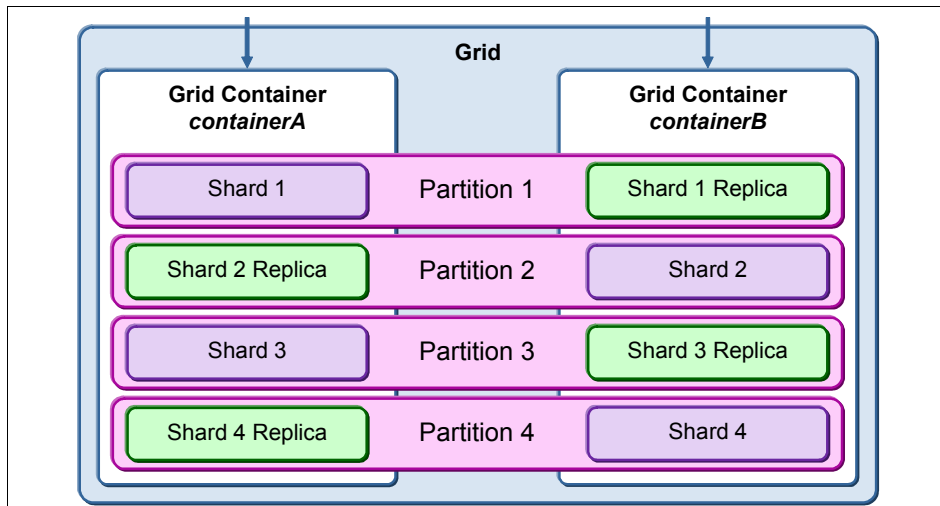


Figure 2-2 Shards distribution

Primary shards: A primary shard is sometimes referred to as the primary partition. Although you might see these two terms used interchangeably, a partition is a collection of a primary and zero or more replica shards.

Shard placement is the responsibility of catalog servers. As the grid membership changes and new JVMs are added to accommodate growth, the catalog server pulls shards from relatively overloaded containers and moves them to the new empty container. With this behavior, the grid can scale out by simply adding more JVMs.

Conversely, when the grid membership changes because of failure or planned removal of JVMs, the catalog server attempts to redistribute the shards that best fit the available JVMs. In this case, the grid is said to *scale in*. The ability of WebSphere eXtreme Scale to scale in and scale out provides tremendous flexibility to the changing nature of infrastructure.

2.1.2 Internal components

Figure 2-3 is an illustration of the internal components that are found in WebSphere eXtreme Scale.

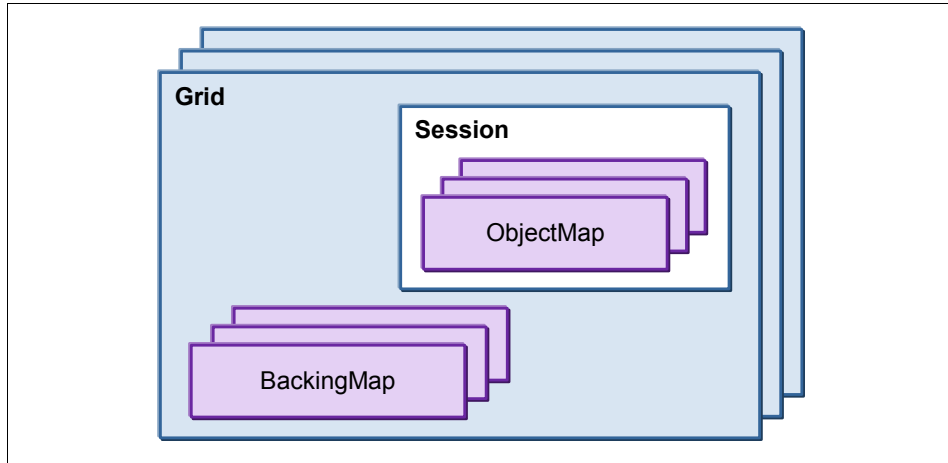


Figure 2-3 WebSphere eXtreme Scale internal components

Session

When a grid client initially interacts with the grid, a session is established. A connection to a session can be made directly by a user or through a front-end application. Sessions are single threaded. When another user connects to the grid, another session is established.

Map

A map is an interface that stores data as key/value pairs. There are no duplicate keys in a map. A map is considered an associative data structure because it associates an object with a key.

ObjectMap

An ObjectMap is a type of map that is used to store a value for a key. That value can be an object instance. An object instance can require its corresponding class file to be in the JVM because the bytecode is needed to resolve the object class if it is deserialized on the server side.

An ObjectMap is always on a client, and is used in the context of a local session.

Figure 2-4 shows three different ObjectMaps. An ObjectMap holds the following key objects and value objects:

- ▶ The first ObjectMap on the left contains a primitive key and primitive value (of primitive data types char, int, and string).
- ▶ The middle ObjectMap contains a primitive key and a compound value.
- ▶ The last ObjectMap on the right contains a compound key and compound value.

All keys and values must be of the same type.

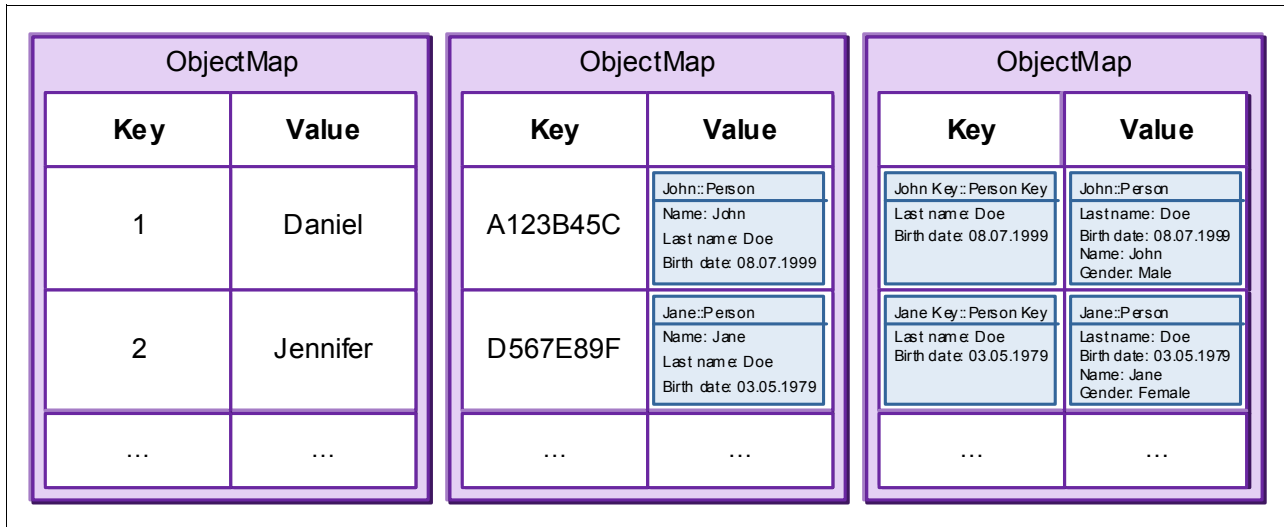


Figure 2-4 ObjectMap examples

BackingMaps

A BackingMap contains cached objects that are stored in the grid. An ObjectMap and a BackingMap are related through a grid session. The session interface is used to begin a transaction and to obtain an ObjectMap, which is required for running transactional interactions between an application and a BackingMap object.

ObjectMaps and BackingMaps can be in the same JVM that is hosting the local grid. For more information, see 2.7.4, “Embedded-partitioned application and cache” on page 58. BackingMaps can also be in a JVM separate from the ObjectMaps. In this configuration, the two maps communicate remotely to persist data. In both cases, objects are copied between an ObjectMap and a storing BackingMap.

Each entity has its own backing map. Any serializable entity attributes are persisted to the BackingMap. A BackingMap can optionally have a loader. In a loader scenario, the BackingMap requests any needed data that it does not contain from its loader, which in turn, retrieves it from the database. This process is illustrated in Figure 2-5. For more information about using loaders, see 3.4.2, “EIS shock absorber” on page 85.

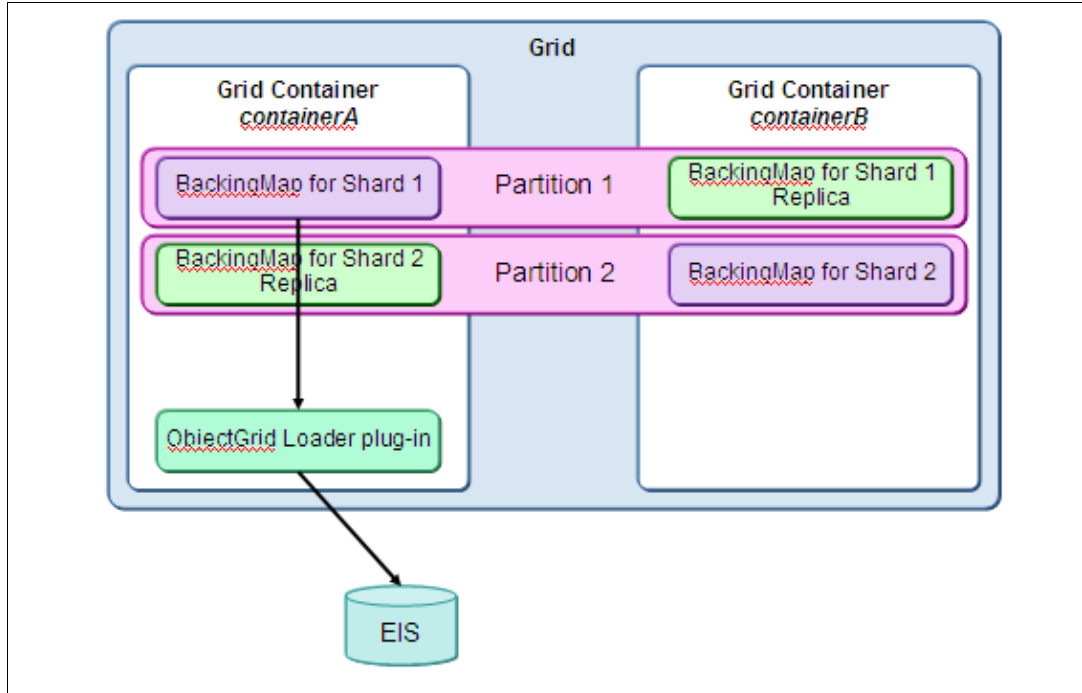


Figure 2-5 Example of using a BackingMap with a loader

2.1.3 Grid clients and servers

The following terms are used when describing how an application interacts with the grid:

- ▶ ObjectGrid Instance

Applications must obtain an ObjectGrid instance to work with a grid. This is done so that the application can interact with the grid and run various operations, such as create, retrieve, update, and delete the objects in the grid.

- ▶ Grid server

Catalog servers and the JVMs that host grid containers that hold the cache are defined as grid servers. The catalog server’s primary function is to serve routing information, whereas the other grid servers or container servers host the cache (stored in BackingMaps).

Servers: The terms “grid server” and “ObjectGrid server” are interchangeable.

- ▶ Grid client

Clients connect to a grid and are attached to the whole grid. Clients must examine the key of the application data to determine to which partition to route the request. Any entity that is attached to the grid with any kind of request becomes a client. A client contains an ObjectMap and can contain a near-cache copy of a BackingMap.

Clients: The terms “grid client” and “ObjectGrid client” are interchangeable.

A grid server and client can have independent BackingMaps (far-cache and near-cache). The server-side, or far-cache, BackingMap is always shared between clients. The client-side, or near-cache, BackingMap (if in use) is shared between all threads of the grid client. Clients can read data from multiple partitions in a single transaction. However, clients can only update a single partition in a transaction.

2.1.4 WebSphere eXtreme Scale meta model

This section explains the operational components that make up WebSphere eXtreme Scale and the relationship between these components. This model can be used to better understand the WebSphere eXtreme Scale product itself, and can be instrumental in the design of a scalable topology.

The WebSphere eXtreme Scale meta model can also be used for analysis of sizing requirements of the grid. Figure 2-6 shows the relationship between various components of the grid.

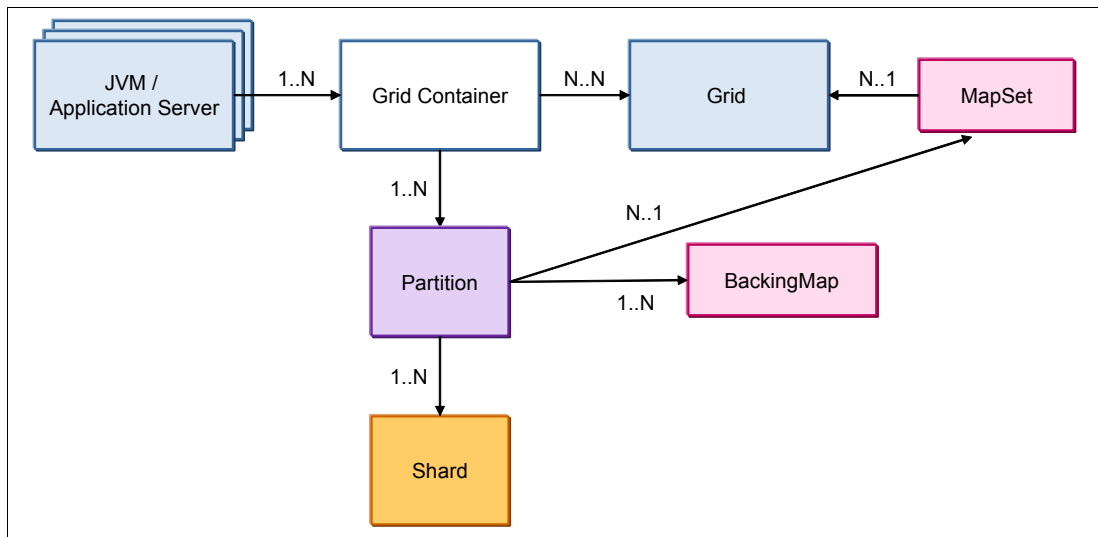


Figure 2-6 WebSphere eXtreme Scale meta model

The following describes the relationship between the components in an eXtreme Scale environment:

- ▶ The JVM can be either an application server or a stand-alone JVM and can host many grid containers. A JVM contains a run time and a number of containers, usually one.
- ▶ A grid container in a JVM can host many ObjectGrid instances, or many ObjectGrid instances that are spread across many grid containers.
- ▶ A MapSet is a collection of maps that are typically used together. Many MapSets can exist in one ObjectGrid instance.
- ▶ An ObjectGrid consists of a number of partitions. Each partition has a primary shard and N replica shards.
- ▶ One partition is a portion of the overall ObjectGrid. A partition can host many BackingMaps. A partition hosts only one MapSet. If an ObjectGrid has two MapSets defined, the ObjectGrid has two sets of partitions: Partitions hosting MapSet1 and partitions hosting MapSet2.

ObjectGrids, MapSets, the number of partitions, and the types of replicas must be defined at server startup. A finite list of maps can be defined, or a map template can be used to add maps during run time. For more information about adding maps during run time, see the *Configuring dynamic maps* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsdynmaps.html>

2.1.5 Transport protocol: IBM eXtremeIO (XIO)

WebSphere eXtreme Scale v8.6 provides a new transport protocol for catalog server, container server, and client communication called *eXtremeIO (XIO)*.

Object Request Broker (ORB): The previous transport, ORB, is now deprecated in WebSphere eXtreme Scale v8.6. Although, the ORB is a general-purpose transport, it is more common to use it with Java native clients.

IBM eXtremeIO has the following benefits:

- ▶ Enables new language integration such as Microsoft .NET.
- ▶ Performance and reliability enhancements.

IBM eXtremeIO has the following requirements:

- ▶ You must configure eXtremeIO for all the catalog servers in your catalog service domain. Container servers and clients detect the transport that is used by the catalog server and do not need to be specifically configured. Catalog and container servers must be at the WebSphere eXtreme Scale Version 8.6 release level or higher.
- ▶ On stand-alone catalog servers, XIO is enabled by default when you use the new startXsServer start script in the WebSphere eXtreme Scale installation bin directory.
- ▶ On servers running in WebSphere Application Server, enable XIO on the catalog service domain by using the administrative console:
 - a. Select **System administration** \varnothing **WebSphere eXtreme Scale** \varnothing **Catalog service domains** \varnothing *catalog_service_domain*.
 - b. Select **Enable IBM eXtremeIO (XIO)**.
- ▶ On Liberty servers, enable eXtremeIO by setting the transport attribute to eXtremeIO in the server.xml file.
- ▶ Command-line arguments and server properties can be used to configure eXtremeIO.

For more information, see the *Configuring IBM eXtremeIO (XIO)* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsconfigxstransport.html>

Also, see the *Tuning IBM eXtremeIO (XIO)* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxstunexio.html>

2.2 Catalog server

The catalog server is the engine that drives the grid operations. The catalog server maintains healthy operation of grid servers and containers. The catalog server becomes the central nervous system of the grid operation by providing the following essential operational services:

- ▶ Location service to all the clients
- ▶ Health management of the grid itself
- ▶ Shard distribution and redistribution
- ▶ Policy and rule enforcement
- ▶ High availability and group service

The term *catalog service* refers to general actions provided by a catalog server or a group of catalog servers. Some actions can occur only on the primary catalog server and some can occur on any catalog server. A *catalog service domain* is the group of catalog servers and container servers that are all running as the same logical group.

A single catalog server can run a grid, but configure a production system to use three catalog servers. Running with three catalog servers provides fault tolerance during maintenance windows and unexpected failures.

There is always a master or primary catalog server. The primary catalog server handles placement and fail over. All of the catalog servers share information. When the primary catalog server stops, a standby or replica catalog servers takes over and retains the same information about placement, the route table, the container servers, and other grid information.

2.2.1 Access point for clients

The client begins its access to the grid by obtaining a route table from the catalog servers. The route table enables the client to locate the partitions. During a JVM failure or redistribution of partitions because of a change in grid membership, the catalog server can push the route table to the client. The client can also pull a new route table when it receives a communication failure.

For more information about client routing and recovery, see 2.2.4, “Placement work and route table flow” on page 39.

2.2.2 Maintaining the catalog service domain

Container or grid servers start and connect to the catalog service. They pass their objectgrid and deployment policy files to the catalog service to indicate which ObjectGrids they can host. The catalog service registers and processes the container servers.

The catalog server is responsible for maintaining which container servers are running, and moving and replacing shards.

The speed of fail over detection is set by using the heartbeat frequency settings. For more information about tuning the heartbeat settings, see 7.6.1, “Tuning the heartbeat frequency level” on page 329.

Catalog servers heartbeat each other. When a catalog server disappears from the group, a new master or primary catalog server is selected if needed. If a WebSphere eXtreme Scale domain experiences network problems that last long enough to trigger heartbeat failures, the catalog server initiates fail over protocols. If the network problem heals itself, the catalog

service domain can be left with two master or primary catalog servers. When this happens, the catalog service domain cannot determine which catalog server is correct. This error condition can be resolved only by restarting the domain. The quorum setting resolves this scenario.

A catalog server cannot be demoted. The catalog service domain cannot determine which catalog server is correct and the domain must be restarted. Setting quorum on the catalog service can stop a domain from splitting and trying to run with multiple primary catalog servers.

When catalog servers run with quorum enabled, the catalog servers all must be in view to allow changes to happen to the grid. If the catalog servers are out of quorum, then action is not taken on server starts, stops, or failures. The grid is frozen until all catalog servers are back in view or a user takes action on the grid.

A JVM loss where the catalog server dies also puts a domain out of quorum. When the catalog server restarts, quorum is established again. A user requested stop of a catalog server does not start an out of quorum state.

Enabling quorum: Enabling quorum protects against intermittent network issues. However, it adds a responsibility to the administrator to provide manual override in the case of more serious network issues.

If a domain runs without quorum enabled, the following scenario can occur in a three catalog server domain where cat0 is the primary catalog server:

1. A network partition or brown out occurs between the catalog server, cat0, and the other two catalog servers, cat1 and cat2. The brownout also splits the container servers.
2. The catalog servers begin checking heartbeats.
3. The heartbeats fail. Cat0 thinks cat1 and cat2 are down, and cat1 and cat2 think cat0 is down.
4. Cat1 promotes itself to primary or master catalog service.
5. Cat1 rebalances any shards that were running on servers in the brown out zone.
6. Cat0 also rebalances any shards running in the brown out zone with cat1 and cat2.
7. Time passes and the network heals, restoring communication.
8. The catalog servers attempt to rejoin each other, but are now running too many primary catalog servers and too many copies of shards. The catalog service domain must be restarted.

When quorum is enabled, the same scenario heals without multiple master catalog servers:

1. A network partition or brown out occurs between the catalog server, cat0, and the other two catalog servers, cat1 and cat2. The brownout also splits the container servers.
2. The catalog servers begin checking heartbeats.
3. The heartbeats fail. Cat0 thinks cat1 and cat2 are down, and cat1 and cat2 think cat0 is down.
4. The QuorumManager determines that quorum was lost between the three catalog servers and no action can be done on the catalog servers. Placement is suspended. Depending on client access, clients might receive TargetNotAvailableException errors when they try to reach servers in the network brown out section.

5. Time passes and the network heals, restoring communication.
6. The catalog servers and containers rejoin each other. No placement action is taken.

The following is a second scenario running with quorum enabled, where the user takes action by overriding the quorum:

1. A network partition or brown out occurs between the catalog server, cat0, and the other two catalog servers, cat1 and cat2. The brownout also splits the container servers.
2. The catalog servers begin checking heartbeats.
3. The heartbeats fail. Cat0 thinks cat1 and cat2 are down, and cat1 and cat2 think cat0 is down.
4. The QuorumManager determines that a quorum has been lost between the three catalog servers and no action can be done on the catalog servers. Placement is suspended. Depending on client access, clients might receive TargetNotAvailableException errors when they try to reach servers in the network brown out section.
5. Time passes. A user notices there is a network problem. The user stops cat0 and runs the xscmd **overrideQuorum** command.
6. Cat1 promotes to primary and rebalances shards that are lost in the same network brownout as cat0.
7. The network problems are resolved. Any container servers that were heartbeat out by cat1 are told to restart.
8. The user restarts cat0. Cat0 joins the other catalog servers as a replica catalog server.

For more information about the use of quorum, see the *Catalog server quorums* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fcxsqorcatsr.html>

Also, see the *Configuring the quorum mechanism* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Ftxsqorcatsr.html>

When quorum is enabled and a network problem happens between catalog servers, clients experience routing problems only when the container servers that it needs to access are also experiencing the network problem. If container servers die during the out of quorum state, they are not replaced and the clients fail to connect to the primary shards lost with those container servers. The client asks for a new route table, but the catalog server does not balance and move shards until the network problem resolves or a user intervenes. New clients can start when a catalog service domain is out of quorum.

If you cannot tolerate an out of quorum state, use a highly available network solution between catalog servers to prevent network brown out problems.

2.2.3 Placement and balancing shards

Catalog servers play an important role in replication, distribution, and assignment of the shards to the grid containers. As the grid containers join the grid, they register themselves with the catalog server. Based on the total number of JVMs, catalog servers are aware of the total number of grid participants. Catalog servers then use this information to calculate the total number of primary shards and their replicas for each partition available for distribution.

The catalog service pushes out the placement work to the container servers and the container servers run the tasks.

The catalog service balance and placement algorithm takes several constraints and rules into consideration to distribute shards across the available container servers. The catalog service provides balanced distribution of the number of shards across containers and automatically rebalances and moves shards based on change events. Common change events include container server failure, container server start, container server stop, and user requested placement changes.

The goal of catalog server balance algorithm is to place the ideal number of shards on each container. The ideal number of shards is calculated by the total number of shards divided by the number of container servers able to host those shards. The total number of shards on each container might not be equal, especially if you are using a prime number of partitions. The difference between containers normally is one shard. In a seven partition environment with one replica defined running on three container servers, there are 14 shards to place. The spread on each container would be 4, 5, and 5 shards.

The location of primary and replica shards and whether the requested number of maximum sync or async replica shards can be met depend on several factors:

- ▶ Primary and replica shards for the same partition are not placed on the same container server.
- ▶ When the deployment policy property `developmentMode` is set to false (preferable for production and production level testing), primary and replica shards are not placed on the same host machines.
- ▶ Any defined zone rules are met. For more information, see 2.8.1, “Zones” on page 60.

As servers start, stop, or fail, shards move from one container server to another to maintain an optimal number of shards relative to the number of containers available. As more servers become available, existing shards are moved to them. If servers leave, more shards are added to existing servers.

An exact balance of primary to replica shards on each container server is not guaranteed. For example, when new servers are added, shards are moved to the new servers. The priority for the catalog service is to place any unplaced shards and to create equal shard distribution. Sometimes this process results in more primary than replica shards (or the reverse) on a container server. For example, in a two server environment with primary and asynchronous replica shards, the primary shards are placed on the first container server. When the second container server starts, replicas are placed on it. The shard types are not balanced because the catalog server’s priority was to place all of the waiting replica shards.

The balance of shard types can be adjusted by using the `xscmd balanceShardTypes` command. Using placement constraints during starts and stops help maintain shard type balancing. For more information, see 2.3, “Replication: Primary and replica shards” on page 44. For more information about placement commands, see 7.2.1, “Placement layout: showPlacement command” on page 311.

When there are server failure events, the catalog server identifies all of the lost primary shards and promotes one of the replicas. Synchronous replicas are chosen first, then asynchronous replicas. After promotion, the catalog service balances again and tries to place replicas to replace the shards that were promoted.

During a requested server stop (rather than a JVM or machine failure), the catalog server balances and move shards to maintain shard distribution. Because the system is not

experiencing a fail over, the shard movement to new containers can be done in a balanced manner instead of an emergency promotion.

There are several ways to tune the timing of placement events. For more information, see 7.6.2, “Tuning and controlling placement” on page 330.

Figure 2-7 shows four partitions, each with a primary and one replica shard, distributed across four JVMs.

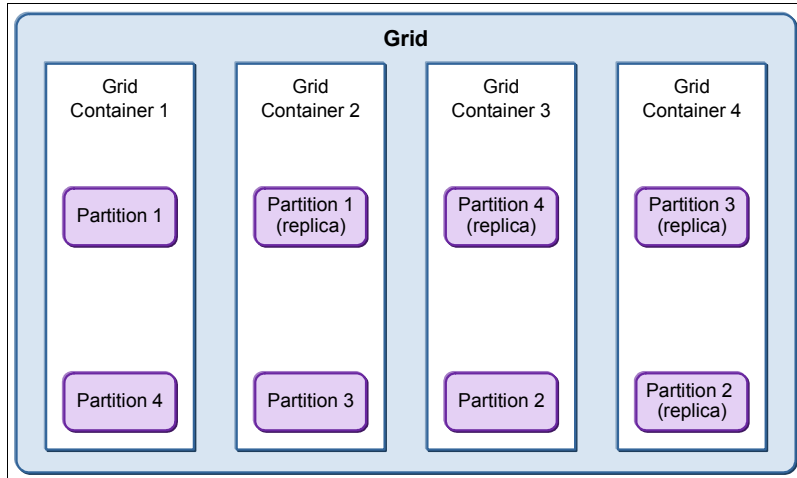


Figure 2-7 Shards placed with all available JVMs

Figure 2-8 shows how the shard placement would adjust if one of the JVMs failed and only three JVMs were available for placement.

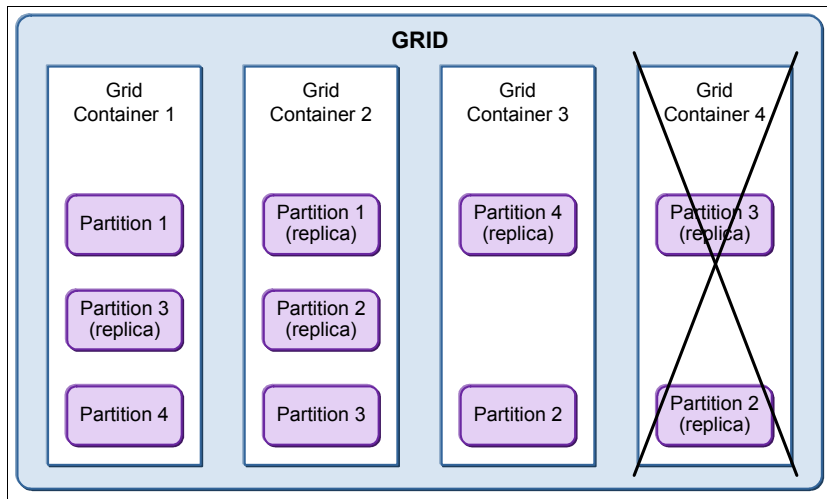


Figure 2-8 Shard placement and redistribution after grid container 4 JVM failure

2.2.4 Placement work and route table flow

This section outlines a basic placement and fail over flow using a single partition and two containers. In any placement event, placement work is sent to the primary shard or target location for a new primary shard. If the receiving container does not have an existing shard to receive the work, it creates a shard. The shard then runs the work. If the shard is new or a replica, it promotes itself to primary. A new primary shard is responsible for stopping the old

primary shard (if one exists). A primary shard is also responsible for adding or removing replicas.

The master route table details the location of all of the primary and replica shards for each partition. It is stored on the catalog servers. Each primary shard is responsible for reporting the routing information for its partition to the catalog server.

Figure 2-9 shows a newly started catalog server, container servers, and client. The catalog server sends placement work to the containers to create shards. The client asks the catalog server for a route table.

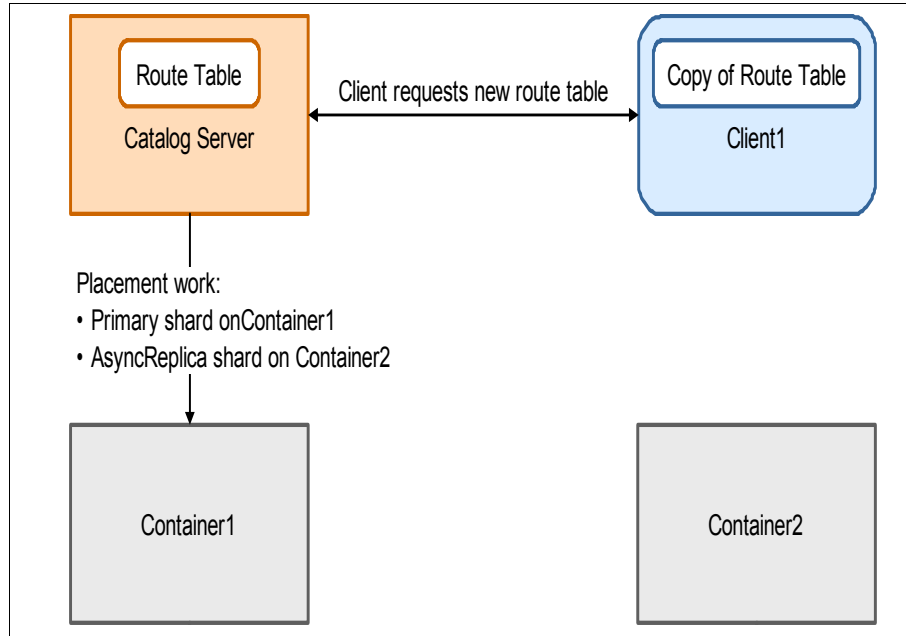


Figure 2-9 Catalog service starts placement

Figure 2-10 shows the primary shard completing the placement work from the catalog service. Container1 creates the new primary shard and starts adding an async replica shard on Container2. The primary shard reports its new route information to the catalog service. The catalog server broadcasts the new route table to the client.

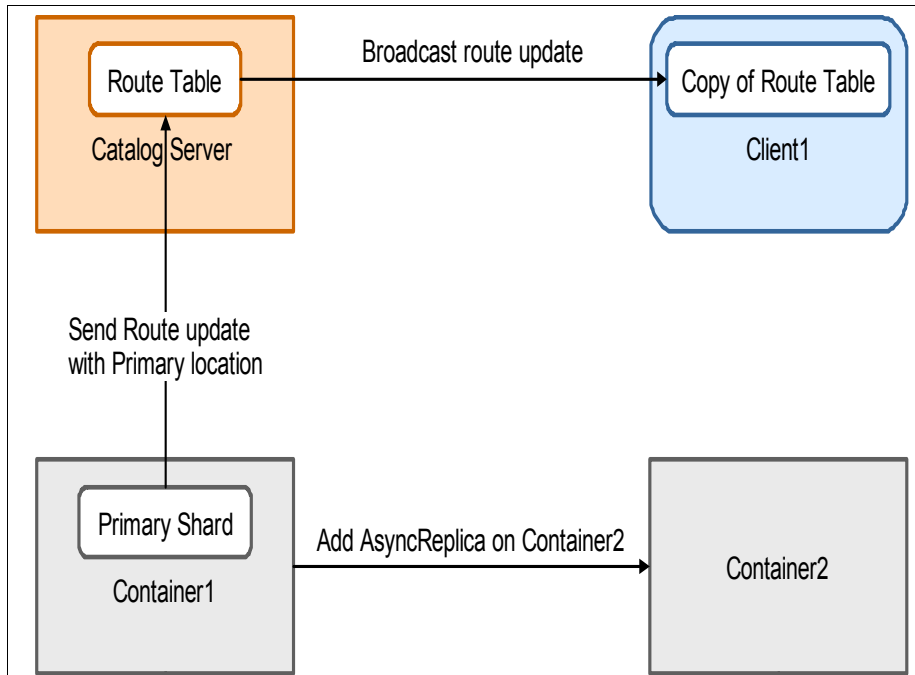


Figure 2-10 Primary shard created

Figure 2-11 details the client routing to the primary shard, the successful creation of an async replica, and the primary shard updating the route table to include the new async replica.

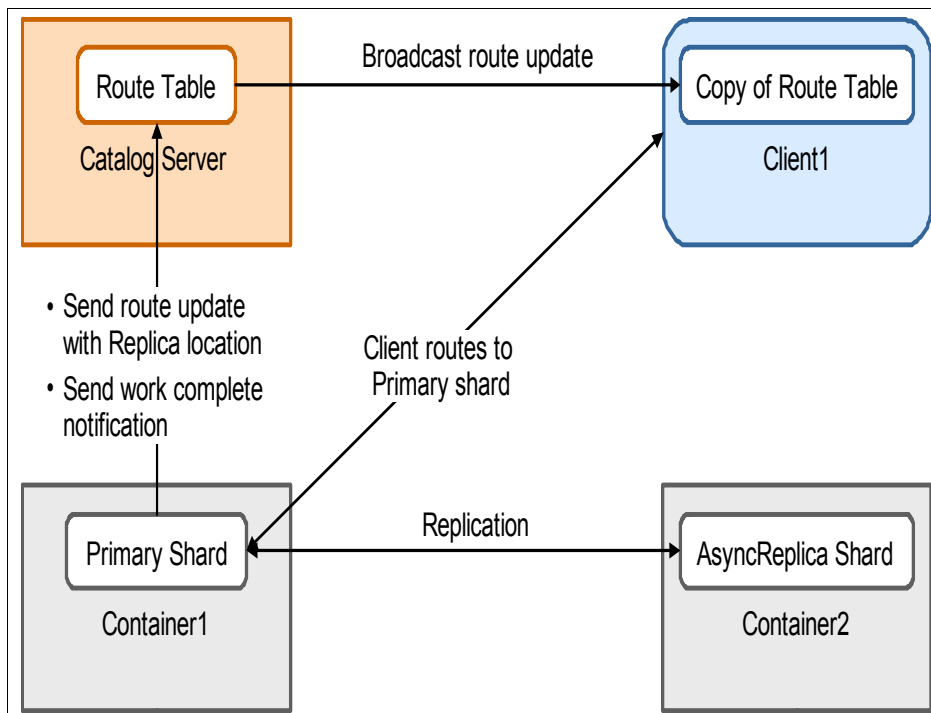


Figure 2-11 Primary shard sends a route update with the replica shard

Figure 2-12 illustrates a container server failure with a dead primary shard. The client and replication requests fail when attempting to talk to the primary shard.

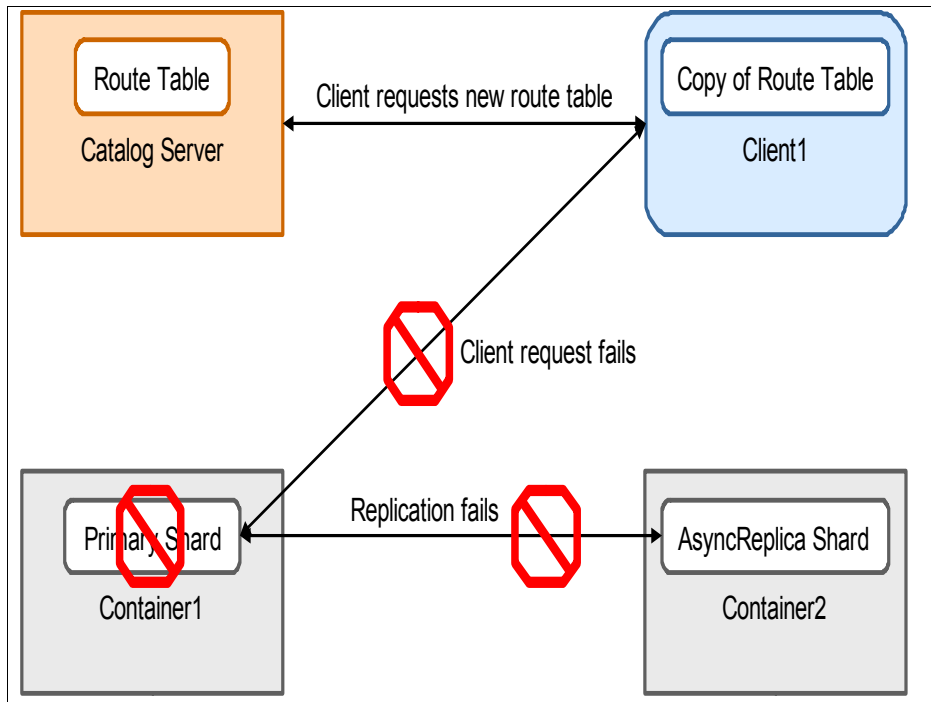


Figure 2-12 Container server failure with dead primary shard

Figure 2-13 shows the catalog service telling the async replica to promote to primary. In the meantime, the client checks for a new route table from the catalog service.

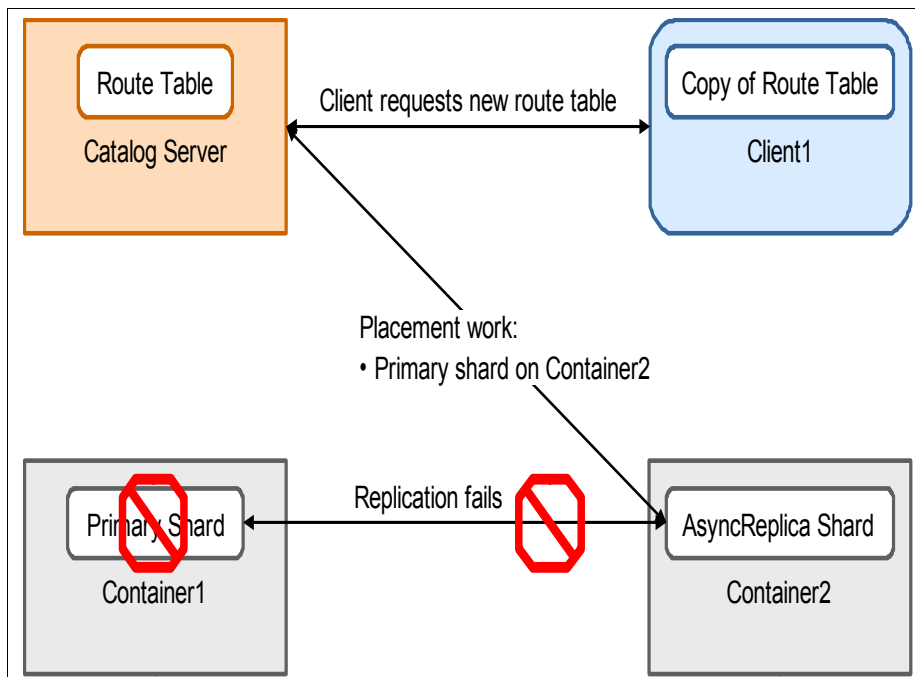


Figure 2-13 Catalog server detects failure and promotes a new primary shard

Figure 2-14 shows the promotion of a replica to the new primary, the route table being updated with the new primary location and new route updates being made available for the client.

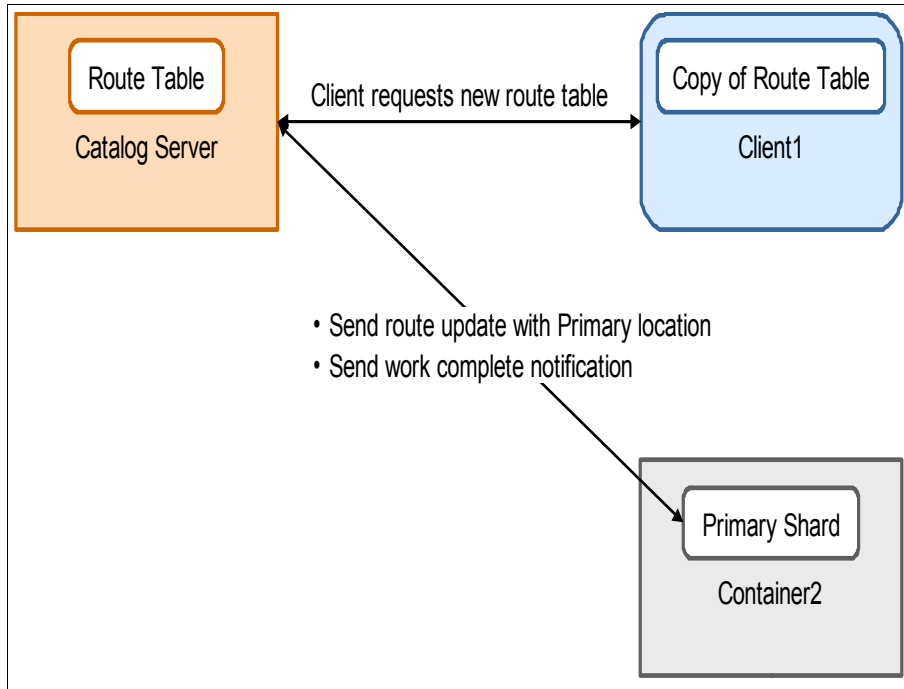


Figure 2-14 Replica shard completes promotion to primary and updates the route table

Figure 2-15 shows the client successfully routing to the new primary shard. If Container1 is restarted, the catalog service tells the primary shard on Container2 to add an async replica shard to Container1.

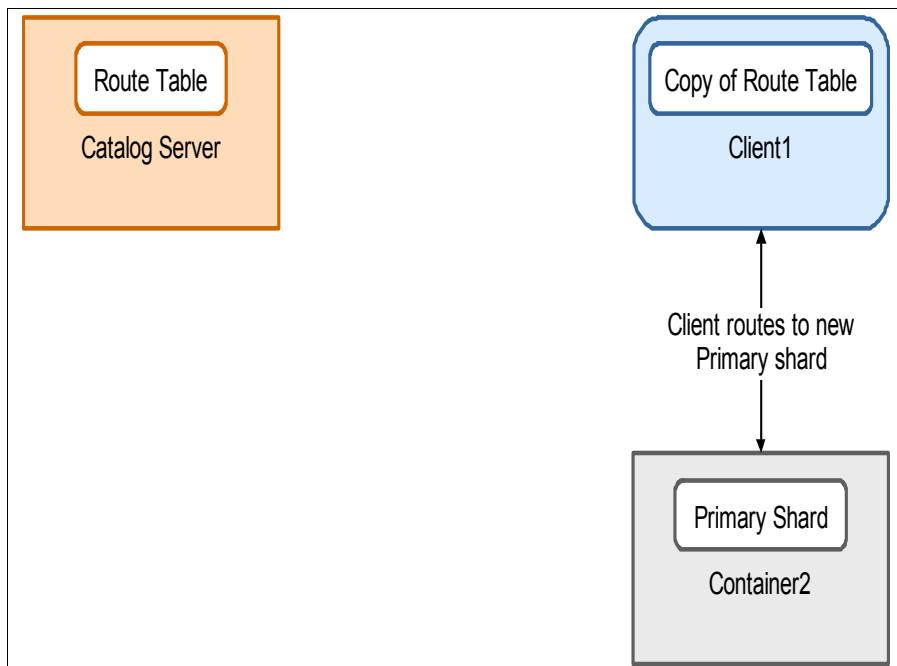


Figure 2-15 Client receives updated route table and routes to the new primary shard

Broadcasting route table updates: The catalog server broadcasts route updates to clients only in an XIO transport environment. In both XIO and ORB transport environments, the route table updates are returned on existing route requests from container servers. Route table updates can always be pulled directly from the catalog service.

For more information about client recovery from failures, see 7.6.6, “Recovery for client failures” on page 333.

2.3 Replication: Primary and replica shards

Replication of data within a partition is a vital part of WebSphere eXtreme Scale. To provide consistency of data, a partition can be defined with replicas. Replicas receive data from the primary shard and are promoted to be the new primary shard for a partition if the primary fails.

WebSphere eXtreme Scale offers synchronous and asynchronous replicas. There are advantages and disadvantages for either type:

- ▶ Synchronous replicas are best when the grid is high write and prefers consistency over speed. They are most efficient on fast and stable networks. Synchronous replicas recover from failures, but have a higher recovery processor usage than asynchronous replicas.
- ▶ Asynchronous replicas are best when some staleness in the grid is tolerated or the grid is read only. If the grid is a read only grid, an asynchronous replica provides faster fail over than preinstalling a new shard. An asynchronous replica can tolerate a slower or unstable network. Asynchronous replicas can be used in most scenarios.

Update actions can only occur on primary shards. Reads can be done on replicas if they are read enabled, which can be set by changing the `replicaReadEnabled` option to true in the deployment policy file. The `replicaReadEnabled` option is false by default.

Reading on replicas in a read only or read mostly ObjectGrid is useful for spreading out client get requests. For more information, see the *Reading from replicas* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsreadreplx.html>

2.3.1 Selecting data to replicate: Revisioning

Data in an ObjectGrid is versioned. The versioning tracks keys and values, and allows shards in a partition to be compared and the difference between them identified. The version includes data points to make comparisons and is called a *revision*. When an asynchronous shard replicates, it sends a packet of revision data that reflects its current state to the primary shard. The primary shard compares the replica shard’s revision data to its local data and creates a packet to return to the replica shard. The replica shard applies the new revision data locally.

Both a new synchronous replica entering peer mode and asynchronous and primaries in multi-master replication (MMR) use the same replication method with revisioning. For more information about the different types of replicas, see 2.3.2, “Synchronous replicas” on page 45, 2.3.3, “Asynchronous replicas” on page 47, and 2.8.4, “Multi-master replication” on page 69.

The following is the general process for replicating with revisions:

1. Replica queries the primary shard with its current revision packet.
2. The primary shard assembles a return packet.
3. The replica applies the new revision packet.
4. The replica sleeps for an interval before another query to the primary shard.

To avoid sending too much data over the network at one time, the target maximum size of the revision packet returned by the primary shard is 0.5 MB. If less than 0.5 MB is available, the primary shard still sends the revision packet. If there is no data to replicate, the primary shard returns an empty packet.

The sleep or polling interval is not a single set amount of time. WebSphere eXtreme Scale varies the polling interval in response to the time it takes to complete a revision query. It also varies the time if communication failures are detected. Finally, a shard relaxes the polling interval when it catches up with the primary and does not receive any new data for several queries.

A shard calculates poll times in three ways:

- ▶ **Active poll:** The shard receives data in its revision packet.

Each active poll interval is calculated based on the time it took to receive a response from the primary shard, with a minimum and maximum allowed time. If the primary took three seconds to return, the next active poll time is three seconds.

- ▶ **Error poll:** The shard receives communication errors.

Each error poll increases until it reaches a maximum error poll. After contacting the primary shard succeeds, the poll resets to an active poll.

- ▶ **Idle poll:** The shard receives empty packets.

Each idle poll increases until it reaches a maximum idle poll. When a query receives data, the poll resets to an active poll. If an error is received, the poll resets to an error poll.

Synchronous replica shards do not go into the idle poll. After they receive empty revision packets, they are ready to complete the peer mode process and receive data from the primary in real time. For more information about entering peer mode, see 2.3.2, “Synchronous replicas” on page 45.

Poll times: Poll times are not configurable. Poll times automatically adjust based on the replication situation.

2.3.2 Synchronous replicas

Synchronous (sync) replica shards receive data in real time during transactions on the primary shard. Each transaction must commit on the primary shard and the sync replicas before a transaction completes the commit. This process means that a transaction takes longer with a sync replica because the primary must know whether the replication was accomplished. Synchronous replicas are defined with a minimum and maximum number. When the minimum number of synchronous replicas is set, any transactions committed on the primary must also have the transaction commit successfully on the minimum number of synchronous replicas set by the deployment policy.

Multiple sync replicas can be configured, but each additional sync replica lengthens the transaction time. The performance impact for multiple synchronous replicas is linear. For most cases, a single synchronous replica provides enough fail over support.

For more information about troubleshooting sync replicas, see 7.6.8, “Troubleshooting synchronous replication” on page 334.

Lifecycle of a sync replica shard

The main two steps of a sync replica are entering peer mode and receiving data from the primary shard during transaction commit time.

Figure 2-16 shows an example of a sync replica shard’s lifecycle.

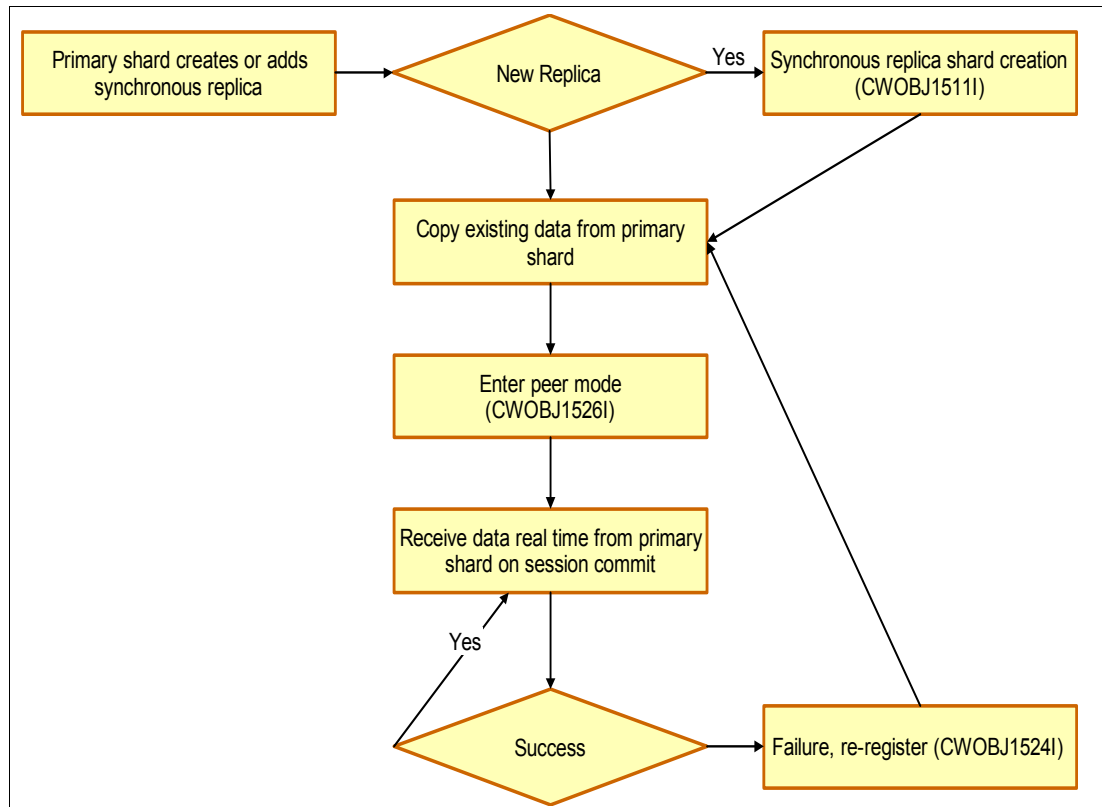


Figure 2-16 Basic lifecycle for a synchronous replica

A new sync replica shard must copy all existing data from the current primary shard before it is ready to receive data in real time. This process is called *entering peer mode*. After sync replica shard creation, the sync replica shard polls the primary for data. The polling continues until the primary shard has no additional changes for the sync replica. For more information about the polling replication model, see 2.3.1, “Selecting data to replicate: Revisioning” on page 44.

The peer mode transition happens where the grid is momentarily locked to swap the sync replica shard from polling for data to receiving data in real time. This step includes the sync replica shard sending a peer mode message to the JVM log that includes the time it took to enter peer mode. Example 2-1 shows an example of the messages that are logged on the sync replica shard’s JVM log when the shard is created and it enters peer mode.

Example 2-1 JVM logging of a new sync replica shard entering peer mode

```

SynchronousRe I CWOBJ1511I: grid:mapSet1:0 (synchronous replica) is open for
business.
SynchronousRe I CWOBJ1526I: Replica grid:mapSet1:0:payroll entering peer mode
after 0.78 seconds, replicating from primary on xsContainer1_C-0
  
```

SynchronousRe I CWOBJ1526I: Replica grid:mapSet1:0:payroll2 entering peer mode after 0.78 seconds, replicating from primary on xsContainer1_C-0

When a deployment policy defines one or more minimum synchronous replicas (`minSyncReplicas`), the success of the sync replicas determines the fate of each transaction. For example, if the deployment policy for a grid sets `minSyncReplicas="1"`, the transaction must commit on at least one sync replica shard before the transaction is successful. If the transaction does not commit on a sync replica shard, the transaction rolls back with a `ReplicationVotedToRollbackTransactionException`. For more information, see “`ReplicationVotedToRollbackTransactionException`” on page 335.

2.3.3 Asynchronous replicas

Replication for an asynchronous (async) replica does not interfere with transactions that are committing on primary shard partitions. An async replica polls the primary shard for new data at a varying interval. It receives committed data from its primary shard when it polls for data.

A maximum number of async replica shards can be defined. For most cases, one async replica shard provides enough failover capability. If there are two or more async replicas defined, they all poll the primary shard independently.

To troubleshoot async replicas, see 7.6.9, “Troubleshooting asynchronous replication” on page 337.

Lifecycle of an async replica shard

An async replica shard mainly spends its lifecycle polling for data from the primary shard. An async replica shard can be caught up with a primary shard. Whenever the primary writes new data, the async replica can be behind again.

Figure 2-17 shows the basic lifecycle of an async replica shard. If an async shard must be promoted to a primary shard, the async shard attempts to close the potential gap between itself and the primary shard by briefly becoming a sync replica. This process can occur when container servers are started or stopped and shards must move to maintain the ideal number of shards per container. If the previous primary died, the async replica promotes straight to primary.

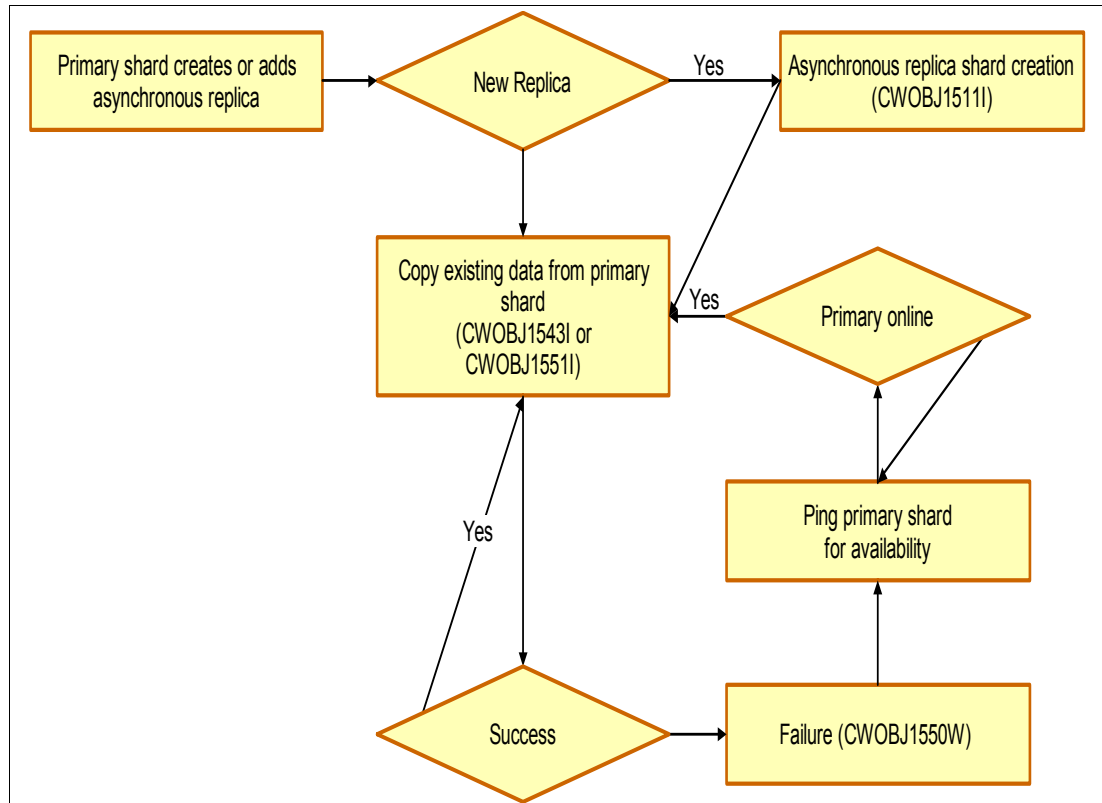


Figure 2-17 Basic lifecycle of an asynchronous replica

2.4 APIs used to access the grid

There are multiple APIs that can be used to access the grid:

- ▶ 2.4.1, “ObjectMap API” on page 48
- ▶ 2.4.2, “EntityManager API” on page 50
- ▶ 2.4.3, “Spring framework” on page 51
- ▶ 2.4.4, “Microsoft .NET” on page 53
- ▶ 2.4.5, “REST data service” on page 53
- ▶ 2.4.6, “REST gateway” on page 54

2.4.1 ObjectMap API

The ObjectMap API provides a transactional map-based API that allows typical create, read, update, and delete operations to the grid cache.

A `com.ibm.websphere.objectgrid.ObjectMap` is accessed by the client application through a session. The stored data is either targeted for or retrieved from the BackingMap.

For more information about the ObjectMap programming API, see the *Interface ObjectMap* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/ObjectMap.html>

For more information, see 4.2.2, “Using the ObjectMap API” on page 109.

Benefits

ObjectMaps provide a simple and intuitive approach for the application to store data. An ObjectMap is ideal for caching objects that have no relationships. ObjectMaps are like Java maps, allowing data to be stored as key/value pairs. Access is easy and fast with primary-key based access. Because ObjectMaps are like Java maps, they are generally familiar to programmers who know `java.util.Map`.

Using the ObjectGrid copyMode `COPY_TO_BYTES` with eXtreme Data Format (XDF) to handle serialization removes the need for Java serialization. XDF is available in WebSphere eXtreme Scale V8.6. It improves the speed and density of the stored data. XDF also provides object evolution to provide interoperability between old and new versions of objects that are stored in the grid. For more information on XDF, see 2.8.2, “IBM eXtreme Data Format” on page 66.

Limitations

ObjectMaps are not ideal if object relationships are involved in your data storage scheme. When you run in copy modes other than `COPY_TO_BYTES`, there are also performance considerations because of the reliance on Java serialization. To serialize data in eXtreme Scale, you can use Java serialization, the ObjectTransformer plug-in, or the DataSerializer plug-ins. See the serialization decision tree in the *Serialization overview* topic of the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxserializer.html>

Data that cannot take advantage of XDF must take into account changes in the objects and serialization. WebSphere eXtreme Scale depends heavily on Java object serialization to transfer object instances between JVMs. Objects in the grid might exist there for a long time, perhaps for month and years. But new releases of an application occur much more frequently.

A new release might not be able to read objects from the grid that were placed there by a previous release. This can be caused by incompatible changes in the class (for example, changing the type of an attribute from “String” to “Integer”). A simple solution would be to completely bring down the grid for redeployment and bring it back (hopefully with preinstalling) afterward.

Java offers a sophisticated class versioning mechanism for serialization. A good starting point is the Java Object Serialization Specification document (Chapter 5, “Versioning of Serializable Objects”) which can be found at the following link. Be sure to fully understand the concept of compatible and incompatible changes.

<http://java.sun.com/j2se/1.5.0/docs/guide/serialization/spec/serialTOC.html>

The Java API documentation is also worth reading and can be found at the following website:

<http://java.sun.com/j2se/1.5.0/docs/api/java/io/Serializable.html>

Leading practices for serialization versioning:

- ▶ Ensure every class that is used to store objects in the grid (that is, keys and values) has an explicitly declared `static final long serialVersionUID` with a value generated on initial release.
- ▶ Establish a build process that can verify that a new version of your application can still deserialize all object types that are serialized by the previous version. This can be accomplished by storing serialized sample objects in the version control system, and de-serializing these objects at build time to verify integrity. You can implement this as a JUnit Test, for example.
- ▶ Make sure that the object classes are available to the grid container.

2.4.2 EntityManager API

The EntityManager API is a simple and intuitive programming model for interacting with the ObjectGrid cache. As an alternative to the ObjectMap API, objects are represented as entities, which allows relationships and can help optimize performance. Relationships are defined in a schema or through Java annotations. The EntityManager API uses the existing Map-based infrastructure, but it converts entity objects to and from tuples before storing or reading them from the Map. An example of using the EntityManager API can be seen in Example 2-2.

For more information about the EntityManager programming API, see the *Caching objects and their relationships (EntityManager API)* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsemgrapi.html>

Example 2-2 EntityManager API example

```
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class EntityManagerExample {

    static public void main(String [] args)
        throws Exception
    {
        ObjectGridManager ogManager = (ObjectGridManager)
            ObjectGridManagerFactory.getObjectGridManager();
        ObjectGrid objectGrid = ogManager.createObjectGrid("Company");

        Session session = objectGrid.getSession();
        EntityManager em = session.getEntityManager();
        EntityTransaction tran = em.getTransaction();

        tran.begin();
        Manager manager=new Manager("Joe");
        Employee employee=new Employee("John", "Doe","50000");
        employee.setManager(manager);
        employee.setSSN("012-34-5678");
    }
}
```

```

    // Persist the manager to the persistence context
    // Employee is automatically cascaded by default.
    em.persist(manager);
    tran.commit();

    // Verify that we can find our employee
    tran.begin();
    Employee emp=(Employee)em.find(Employee.class, ("012-34-5678"));
    tran.commit();

// Remove the employee entity
    tran.begin();
    Employee employeetoremove =(Employee)em.find(Employee.class,
("012-34-5678");
    em.remove(employeetoremove );
    tran.commit();
}
}

```

Benefits

The EntityManager API is ideal when object relationships are involved. It provides an easy way to interact with a complex graph of related objects or Object graphs. It has a slight performance advantage over the ObjectMap API, as the EntityManager API uses tuple sets of primitives only with no reliance on serialization. There is optimized performance for queries and for loading objects from the back-end data source. The EntityManager API also might be easier to use because of its reliance on plain old Java object (POJO)-style programming, which has been adopted by most enterprise application architectures.

Limitations

The ObjectMap API is the leading practice API over the EntityManager API. The EntityManager requires the definition of schema in an `entity.xml` file. Applications might have to be rearchitected to avoid complex relationships between objects and to ensure that there is an absolute relationship between a root and its branches. This relationship is known as a constrained tree schema. Two applications might not be able to share a cache if both applications use different objects for the same data. Also, complex queries might not perform well because of the partitioned nature of the data.

2.4.3 Spring framework

WebSphere eXtreme Scale provides integration for the popular Spring framework. The Spring framework can manage WebSphere eXtreme Scale transactions and configure grid clients and servers. You can also use WebSphere eXtreme Scale as the cache provider for the cache abstraction that was introduced in Spring Framework Version 3.1.

For more information about the Spring framework, see the SpringSource website:

<http://www.springsource.org/>

For more information about the Spring framework and WebSphere eXtreme Scale, see the *Developing applications with the Spring framework* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsprogspring.html>

Benefits

Spring managed transactions can be used with local or remote WebSphere eXtreme Scale topologies. WebSphere eXtreme Scale provides an implementation of a Spring PlatformTransactionManager. With annotated POJOs, Spring automatically acquires sessions from eXtreme Scale to begin, commit, roll back, suspend, and resume eXtreme Scale transactions. For more information, see the *Managing transactions with Spring* topic in the WebSphere eXtreme Scale Version 8.6 Information Center::

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsspringint.html>

For an example, see the *Getting Started with Integrating eXtreme Scale and Spring* article on the IBM Elastic Caching Community:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/getting_started_with_spring4?lang=en

With WebSphere eXtreme Scale as a cache provider to Spring, the grid caches the return value from methods. On future invocations, the method does not need to run again. Methods are identified for caching by using annotations or XML. WebSphere eXtreme Scale runs as a remote topology and responds with the same fast fail features as using WebSphere eXtreme Scale with dynamic caching. If the grid is unavailable, cache returns are null. Normal operations continue when the grid is available again. For more information, see the *Configuring a Spring cache provider* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsspringprovide.html>

For an example, see the *Integrating eXtreme Scale and Spring on the XC10 Appliance* article on the IBM Elastic Caching Community:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/getting_started_with_spring5?lang=en

Limitations

WebSphere eXtreme Scale does not support JTA or two-phase commit. It does support the last participant as a single-phase participant. For more information about last participant support, see the *Transaction processing in Java EE applications* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxstransoverview.html>

Using WebSphere eXtreme Scale as the Spring cache is a remote topology only. The user configures WebSphere eXtreme Scale servers with Spring ObjectGrid descriptor and Spring deployment policy files. For more information, see the *Starting a container server with Spring* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txssprngcont.html>

2.4.4 Microsoft .NET

The .NET client is installed by using a self-contained, 32-bit Windows installer. After it is installed, .NET developers have access to a native .NET WebSphere eXtreme Scale data access programming interface with standard create, remove, update, and delete operations, and transactional support.

The .NET client also provides a plug-in implementation to facilitate user ID and password credential authentication with WebSphere eXtreme Scale servers, similar to the Java client. If you install the .NET client development environment feature, the .NET client provides sample .NET code, an API reference help file, and Visual Studio IntelliSense Quick Info extensions.

For more information about the WebSphere eXtreme Scale Client for .NET, see the *Client for .NET API documentation* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.netapi.doc/html/2473b261-be75-50a4-c13d-43c6d4df65be.htm>

Benefits

Previously, .NET applications, such as ASP.NET web applications, could only access the data grid by using the REST interface or the REST data service. With the .NET client, .NET applications can now use a richer .NET programming interface to access the data grid. Also, the .NET client provides full support for generic APIs and more of the batch APIs like GetAll and DeleteAll.

Limitations

The .NET client API is similar to the Java API, but initially limited features are available. The .NET client has some limitations, including no availability of near cache or EntityManager. The connection is still TCP/IP based, but all .NET client/server traffic must flow over XIO (ORB-based environments are not supported). Also, any map that is accessed by the .NET client must be configured with pessimistic locking. Also, even though WebSphere eXtreme Scale supports various "copy" modes in this release, the .NET client is limited to COPY_TO_BYTES.

2.4.5 REST data service

The WebSphere eXtreme Scale REST data service is a Java HTTP service compatible with Microsoft WCF Data Services (formally ADO.NET Data Services) and implements the Open Data Protocol (OData). The REST data service allows any HTTP client to access a data grid. RESTful applications can be developed with the rich tools provided by Microsoft Visual Studio.

For more information about the REST data service, see the *REST data services overview* topic in WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsrestintro.html>

Benefits

The REST data service is a Java servlet. The REST data service communicates with the WebSphere eXtreme Scale data grid by using the WebSphere eXtreme Scale Java APIs. It allows WCF Data Services clients or any other client that can communicate with HTTP and

XML to interact with an ObjectGrid. Various client platforms can interact with REST, such as PHP using HTTP, Atom Publishing Protocol (AtomPub), and JSON.

You can do common data access operations like create, read, update, merge, delete, and query. You can update and retrieve data from the REST data service by using a single remote procedure call to the REST data service. This process allows multiple requests to participate in a single transaction.

Limitations

The REST data service must be deployed on a web servlet container such as a WebSphere Application Server or Apache Tomcat. The WebSphere eXtreme Scale servers must be started before the REST data service web application.

2.4.6 REST gateway

The Representational State Transfer (REST) gateway allows access to a catalog server domain from non-Java environments by using HTTP requests. For more information, see the *Deploying the REST gateway* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsrestliberty.html>

Benefits

You can use the POST and GET HTTP methods to insert and get data grid map entries. You can also run operations like creating template maps, clearing the grid, and setting TTL type evictors.

Limitations

The REST gateway can only be used in Liberty or WebSphere Application Server environments (or with the WebSphere DataPower XC10 Appliance). The gateway must be deployed at the server startup. Access to the REST servlet must be secured with web application security, or unauthorized users could access and change the ObjectGrid.

2.5 A simple example

The following example shows the different components that make up a grid and illustrate the flow from the application to the grid. This example is taken from a simple distributed cache topology. Figure 2-18 on page 56 shows an application that retrieves an object for a key from the grid. It updates the object's value and commits the change to the grid.

1. The application requests an object from the grid for a key. Because this request is the first time that the object has been accessed, it is not yet present in the local session ObjectMap.
2. The grid client uses the ObjectMap to attempt to retrieve the object from the client near-cache (which is actually a local instance of a backing map). Assuming no other thread in the application has accessed this particular object, there is also a miss here.
3. The grid client determines the partition number in which the object for the key should be. Usually, the primary shard of that partition is accessed. If `readFromReplica` is enabled, a replica shard might be accessed instead. If the grid container is collocated with the application in the same JVM or host, this adjustment can increase speed. In either case, the grid client connects to the selected shard and asks for the object.

4. The shard checks whether an object for the requested key is present in its backing map. The object might be already available through preloading, but do not assume this is the case. In this case, the grid returns `null`, which indicates that the key is not available. The client is then responsible for loading the data from the back-end datastore and using a `put()` to place it into the grid. In this example, a loader has been configured for the grid. This simplifies client programming, as only the grid must be accessed to retrieve and store what is needed from the back-end datastore.
5. The loader is started to bring in the data from the back-end datastore, for example by using JDBC, and the object is constructed. All previous calls store a copy of that object in its corresponding map and return it to the caller. Therefore, the object eventually arrives in the client.
6. The client changes the object for a key and puts it back in the `ObjectMap` by using the `update()` command. The changes occur only in the session's local copy of the object. Other sessions that access the object cannot see this change yet.
7. The client commits the change, which sets off a sequence of calls to propagate the changes into the grid and down to the back-end datastore.
8. The object is updated in the client near cache.
9. The update is propagated to the backing map of the primary shard.
10. The backing map updates the entry for the key. If replicas are configured, they are now contacted and informed of the update (not shown in this example). This can happen in synchronization (the operation returns only after the replica has been updated successfully) or asynchronization (the operation returns immediately). The backing map informs the loader to update the value. By default, this updating is also a synchronous operation. If write-behind is activated, this updating happens asynchronously. The client sees a successful update operation even when the back-end datastore is offline.
11. The loader writes the changes to the back end. All calls unwind and return.

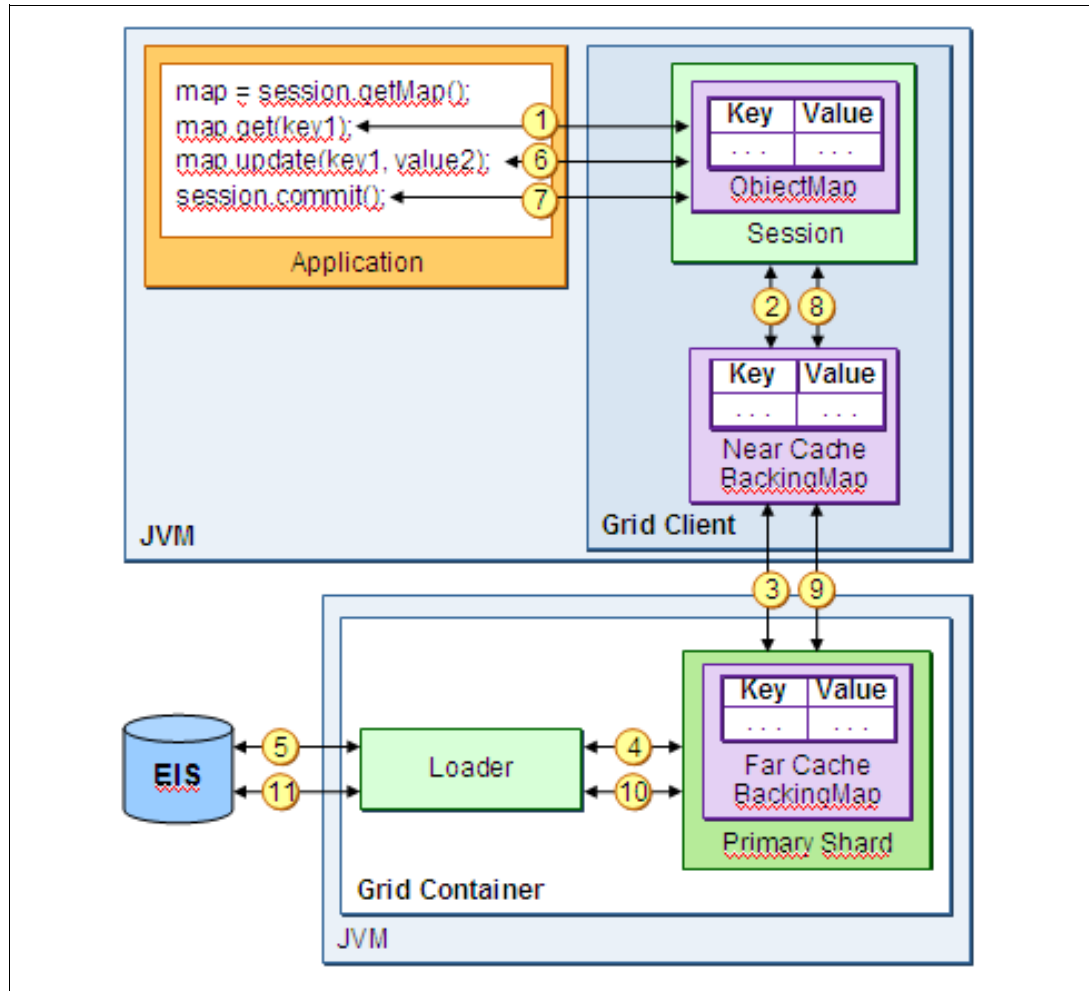


Figure 2-18 Component interactions for simple grid access

2.6 Scalability sizing considerations

WebSphere eXtreme Scale provides a scalable framework with a choice of runtime topologies to linearly extend the scalability of a data-driven transactional application. Linear scalability is the ability to grow the grid with increase in demand and data without increasing resource usage. The discussion around performance, metrics, and sizing is beyond the scope of this book, but the following are some significant sizing considerations. Sizing is an important topic, as stability and health of the grid is of paramount importance in ensuring a reliable enterprise caching system.

For detailed JVM and ObjectGrid as well as other capacity planning information, see the Capacity planning and tuning chapter in the IBM Redbooks publication *WebSphere eXtreme Scale Best Practices for Operation and Management*, SG24-7964.

Catalog servers play a central role in eXtreme Scale grid management. It is vital to plan for catalog server high availability and sizing requirements. Three catalog servers provide a good balance between reliability and scalability.

The catalog servers replicate information about the catalog service domain. They compress the data for replication. In a large topology that scales to several hundred container servers,

the default compression algorithm performance degrades. The compression algorithm for catalog server replication can be changed for better performance. Current options include `Optimized1` or turning off the compression.

For more information about catalog server properties, see the *Server properties file* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/topic/com.ibm.websphere.extremescale.doc/rxscontprops.html>

In a large-scale environment, review the the heartbeat settings periodically to maintain a balance between a stable environment and fail over detection. For more information, see 7.6.1, “Tuning the heartbeat frequency level” on page 329.

2.7 Common topologies

Before you design your topology, consider what type of software that you will install on your servers to house your grid. There are two types of servers: Stand-alone and managed (WebSphere Application Server or Liberty). You can use a combination of both types of servers. This section reviews the following common topologies:

- ▶ Managed grid
- ▶ Stand-alone grid
- ▶ Embedded cache
- ▶ Embedded-partitioned application and cache
- ▶ Remote cache (with optional near cache)

2.7.1 Managed grid

You can use WebSphere Application Server servers to host your eXtreme Scale JVMs. The main benefit of this configuration is that you can more easily manage your environment using the administrative capabilities available in the WebSphere Application Server Network Deployment product.

WebSphere eXtreme Scale can use the security that is provided by WebSphere Application Server, such as the LDAP plug-ins. The clustering and high availability management features offered by the managed environment can be used. Grid extensibility becomes easier in a managed environment because creating grid servers to extend the grid is only a matter of a few clicks. However, this is true only if the capacity supports the grid expansion.

Additionally, any commonly available monitoring tools that are already employed to monitor the performance and availability of your environment can be used to monitor the grid servers.

2.7.2 Stand-alone grid

Stand-alone servers can use the client JVM of your choice to host the ObjectGrid. The main benefit of this configuration is the use of less expensive Java SE (JSE) containers as grid servers that can hold the cache. Environments that already use JSE for their applications might also be inclined to use JSE/stand-alone containers. In this case, the inclusion of WebSphere eXtreme Scale as a platform might be the only new addition.

While there are obvious cost advantages to using stand-alone servers, the downside is the possible lack of available management and monitoring solutions for the JSE containers that host the grid servers. The advantage is less memory usage of a whole application server.

A common scenario is the use of a WebSphere Application Server Network Deployment managed environment to host enterprise applications, and the use of a readily available JSE runtime environment as the grid layer.

Regardless of which type of server is chosen, the core capability of WebSphere eXtreme Scale, which is a transactional, secure, and scalable application cache fabric, is available to be used. The issues around management and monitoring are environment-specific and a personal choice. Each topology that is described below can be implemented on either type of server or by using a combination of servers of both types.

2.7.3 Embedded cache

The embedded cache topology (Figure 2-19) is only recommended for a development environment when the application logic runs in the same JVM as the data in the grid. Each application only accesses the local ObjectGrid instance to store or retrieve data from its cache. WebSphere eXtreme Scale, in this case, is used as a simple near cache.

This topology can be faster than using a database if the needed data can be found in the embedded ObjectGrid. This process avoids a Remote Procedure Call (RPC) to the back-end datastore. Using WebSphere eXtreme Scale as an embedded cache can also reduce the load on your back-end datastore.

This topology is not recommended for fault tolerance or high availability. Also, you might find duplicated data if you have more than one grid server acting as an embedded cache. For more information, see 4.6, “Dealing with data eviction and stale data” on page 137.

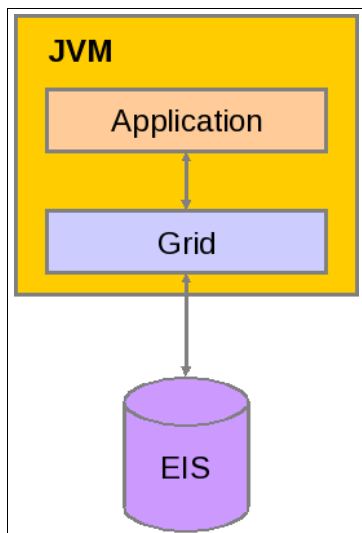


Figure 2-19 Embedded cache topology

2.7.4 Embedded-partitioned application and cache

In the embedded-partitioned application and cache topology (Figure 2-20 on page 59), the application logic runs in the same JVM as the data in the grid. However, the data that are stored in the grid are spread across all the JVMs that have WebSphere eXtreme Scale installed and configured.

This topology can be faster than using a database because an application can take advantage of the embedded-partitioned to compensate for the RPC calls made when the

requested data is stored on another server in the grid, or can be found only in the back-end datastore. This topology can also reduce the load on your back-end datastore.

With this topology, replica shards that sit in a JVM other than the primary can be used to ensure fault tolerance and high availability.

For some applications, sharing the JVM between the application and grid container can limit scaling or performance. For more information, see 2.7.5, “Remote cache (with optional near cache)” on page 59.

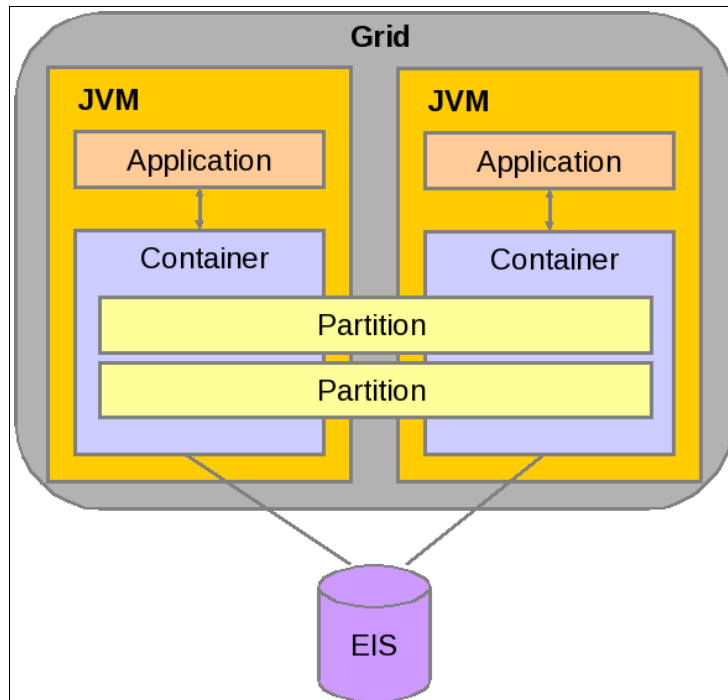


Figure 2-20 Collocated application and cache topology

2.7.5 Remote cache (with optional near cache)

In the distributed cache topology (Figure 2-21 on page 60), the application logic runs on application servers separate from the grid servers. The application servers host a grid client with an optional near cache. A near cache provides local in-memory access to a subset of the entire cached data set that is stored remotely and must be memory managed with an evictor. The data that is stored in the grid is spread across all the JVMs that have WebSphere eXtreme Scale installed and configured. In this case, the WebSphere eXtreme Scale distributed grid is more like a data service proxy for the back-end system. This topology can also reduce the load on your back-end datastore.

This is the default WebSphere eXtreme Scale topology. There are several benefits to using this topology. Replica shards can sit in JVMs separate from the primary shards to ensure fault tolerance and high availability. The application servers can be restarted without interrupting the availability of the grid servers. If you want to employ clustering, cluster the application servers that host client applications separately from the grid servers. This configuration allows the application cluster to be restarted without affecting the grid layer.

Clients can optionally have a local, in-line cache when eXtreme Scale is used in a distributed topology. This optional cache is called a near cache. It is an independent ObjectGrid on each client that serves as a cache for the remote, server-side cache. The near cache is enabled by

default when locking is disabled. The near cache is also enabled when it is configured as optimistic. It cannot be used if it is configured as pessimistic. The near cache can be set on optimistic or no locking maps by using the ObjectGrid descriptor `nearCacheEnabled` property. The size of the near cache can be managed by using evictors. For more information, see 4.6.2, “Using time-based eviction strategies” on page 138.

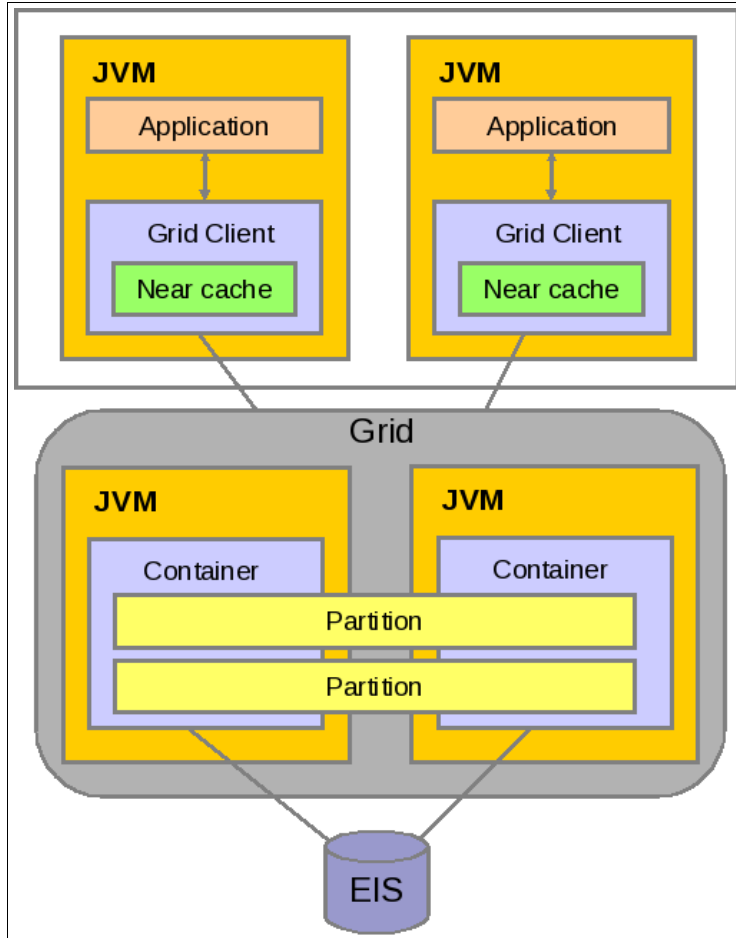


Figure 2-21 Distributed cache topology

2.8 Common topology options

With stand-alone or managed topologies, there are several topology options that can be used separately or together. The following topology options are addressed in this section:

- ▶ Zones
- ▶ IBM eXtreme Data Format (XDF)
- ▶ IBM eXtremeMemory (XM)
- ▶ Multi-master replication

2.8.1 Zones

WebSphere eXtreme Scale provides high availability of partitions by not placing replica shards on the same container server or host machine based on host name or IP address when the deployment policy `developmentMode` option is false. To further tune placement decisions, WebSphere eXtreme Scale provides zones to create collections of administratively

grouped grid containers. Instead of limiting placement by machines, a group of machines or servers on virtual machines can be grouped to improve high availability. Zone-based replication factors in the possibility of network partitions, latencies, and network congestion.

In prior releases of WebSphere eXtreme Scale, zones were used to improve high availability of data cache across geographies. The current leading practice is configuring multi-master replication to support data centers in remote locations. For more information about multi-master replication, see 2.8.4, “Multi-master replication” on page 69.

Multiple zones can be envisioned to exist across WANs and LANs, or even in the same LAN, but one zone is not intended to span across a WAN. This limitation stems from maintenance and management constraints that are imposed by group service and HAManager. However, this internal limitation can be overcome by defining multiple zones across a WAN. Adding zones to a topology includes the following advantages:

- ▶ Primary and replica can be placed in different zones, satisfying disaster recovery requirements.
- ▶ Placing synchronous or asynchronous replicas in different zones based on network latency or geographic location.
- ▶ Controlling placement grouping with virtual machines, blade centers, or other groups of systems.

Zones are configured by assigning container servers to two or more zones. In a stand-alone topology, the zone for container servers is defined at start up. In a WebSphere Application Server managed topology, node groups are used to create zones. For more information, see the *Defining zones for container servers* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txszonedef.html>

The catalog service stripes primary and replica shards across zones. For more control, the zone rules listed in the deployment policy file dictate placement of synchronous or asynchronous replica shards in different zones. For examples of zone rules, see the *Example: Zone definitions in the deployment policy descriptor XML file* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxszonedepl.html>

Figure 2-22 shows two zones with several containers grouped in each zone. A primary shard in zone A is not replicated to a container in the same zone. Each primary shard in zone A has a replica shard in zone B. Primary shards that are placed in zone B must have replicas placed in zone A.

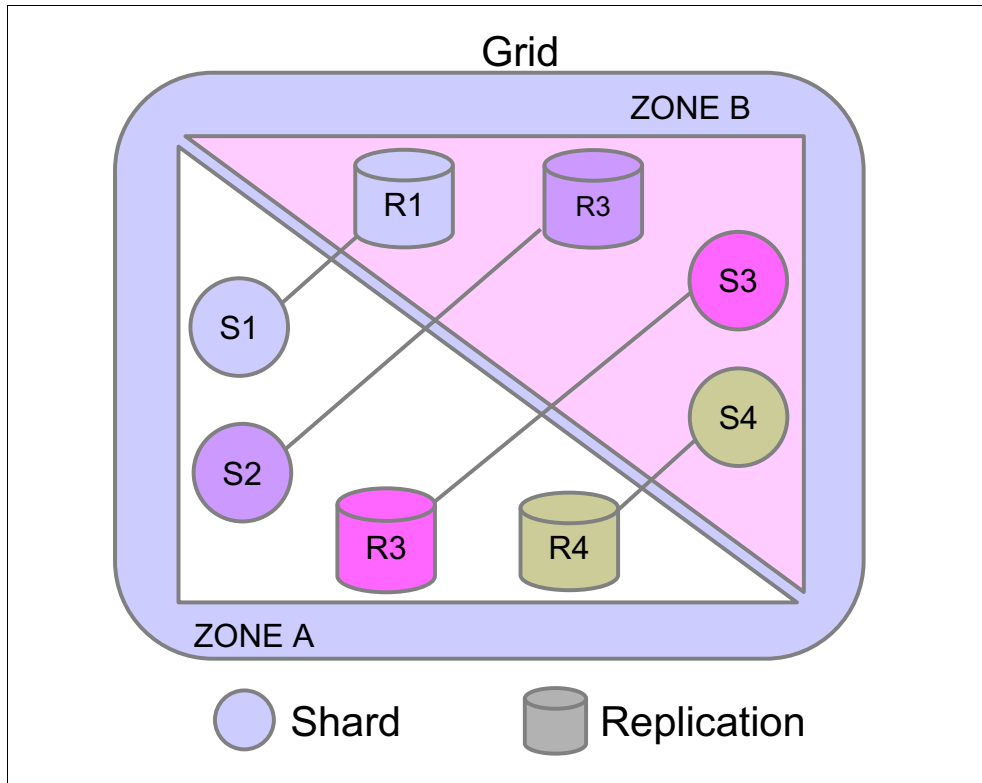


Figure 2-22 Zone placement

A typical environment (Figure 2-23) has two Linux server machines, with several virtual machines (VM). Each one of them has one or more partitions. The VMs look like different machines to the catalog service, and it does not recognize that two VMs are actually on the same physical machine. If the primary and replica shards are placed in different VMs on the same machine, all data for some partitions might be lost. To avoid this, you can create a zone and put all the container servers on the same machine together at the same zone. WebSphere eXtreme Scale places primary and replica shards in different zones to avoid data loss.

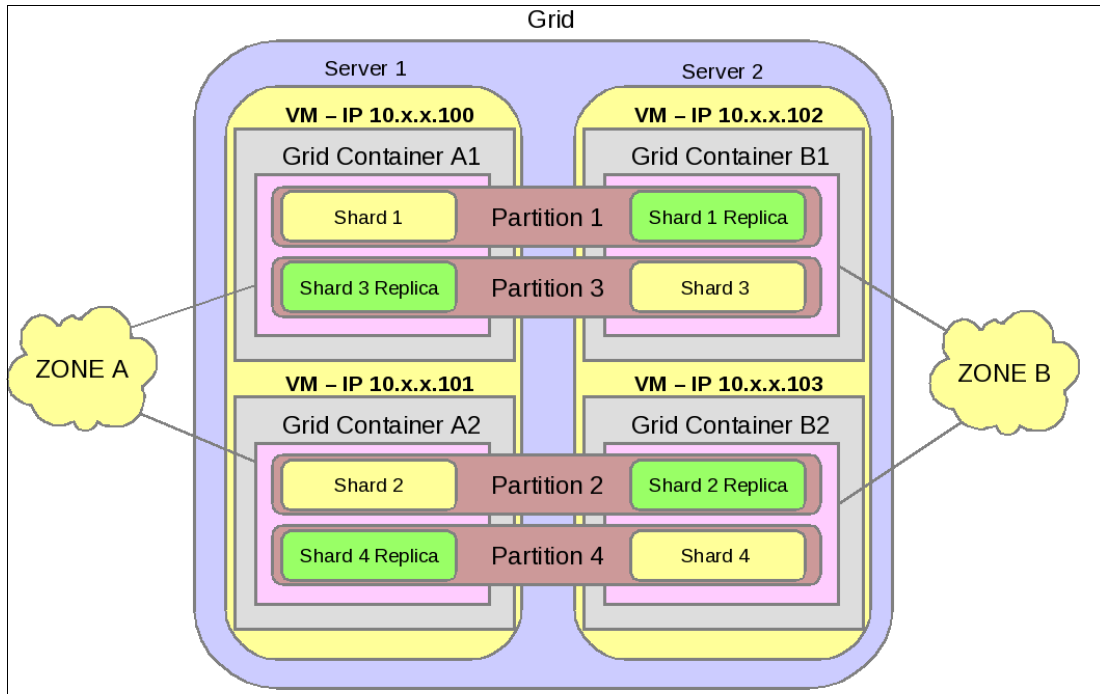


Figure 2-23 Virtual machines and zone placement

Zone capability is useful to ensure that replicas and primaries are placed in different zones. Zone rules can be configured for more precise control for which types of shards are placed in which zones. For example, rules can be set so that the replica shards are placed in separate zones (in a different building or even in a different geographic location) from the zone that hosts the primary shard.

It is a standard practice to place a synchronous replica in the same LAN (for high availability) and an asynchronous replica when you are using WAN. This configuration ensures high availability for a JVM failure in a local zone (synchronous replica) and ensures high availability in a complete data center failure (asynchronous replica).

This flexibility assures the availability of data to the application regardless of its zoned location. The catalog servers provide up-to-date routing information about the location of an object if the object is not found in the zone with the closest proximity to the application container. See Figure 2-24.

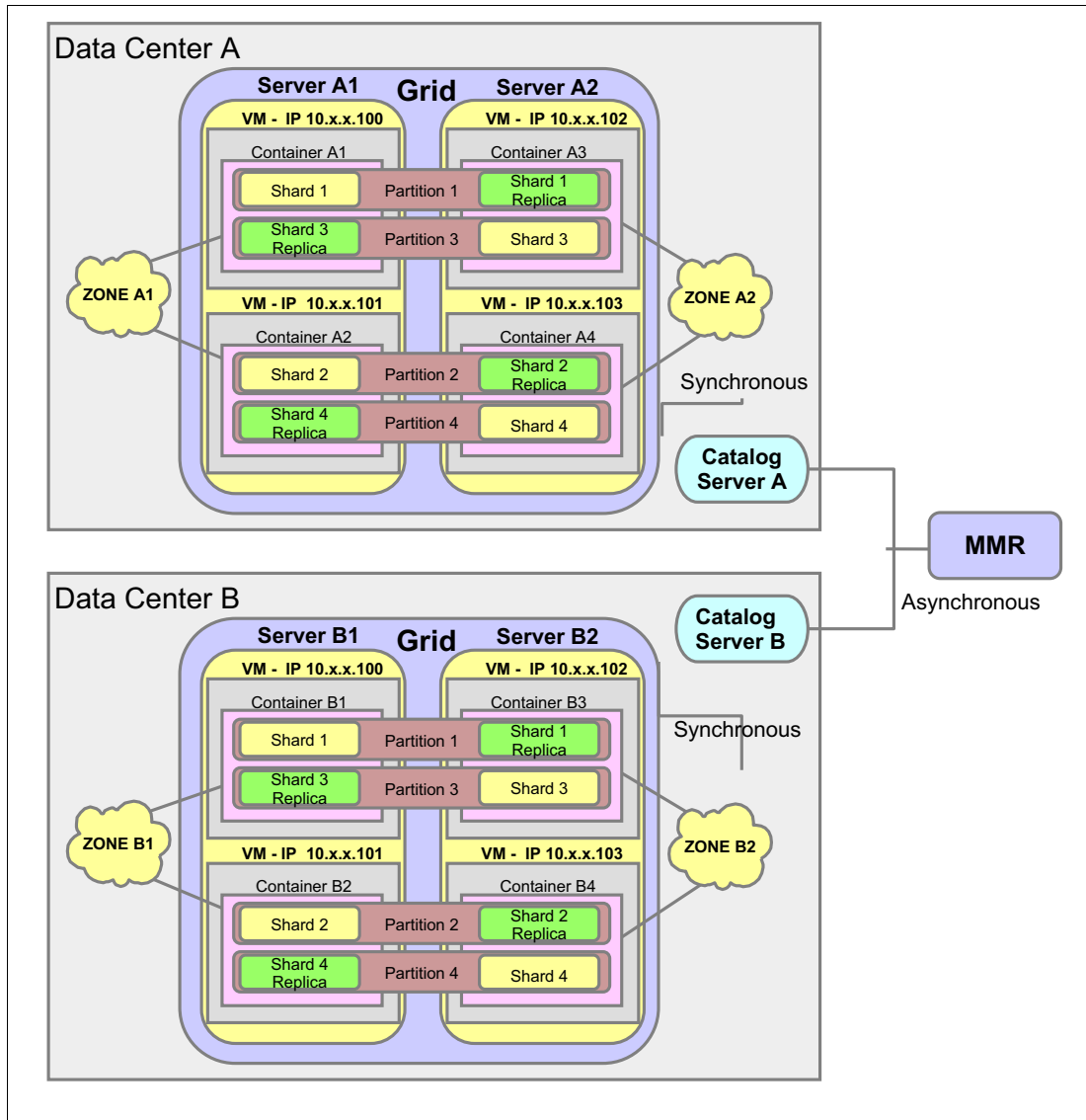


Figure 2-24 Topology using zones and multi-master replication

Zone-based routing

WebSphere eXtreme Scale provides a mechanism for clients to set preferences on how their requests are routed. WebSphere eXtreme Scale supports a routing preference for zones, local host, and local process. This preference applies to both hash-based fixed partitions and per-container partitions.

Proximity-based routing provides the capacity to minimize client traffic across zone boundaries, to minimize client traffic across machines, and to minimize client traffic across processes.

ClientProperties interface

The ClientProperties interface (Example 2-3) provides the mechanism to specify a preference for how requests are routed. You can use the `objectGridClient.properties` file or you can directly use the interface.

Example 2-3 ClientProperties interface

```
public interface ClientProperties {
    public void setPreferZones(String[] zones);
    public String[] getPreferZones();
    void setPreferLocalProcess(boolean localProcess);
    void setPreferLocalHost(boolean localHost);
    public boolean isPreferLocalProcess();
    public boolean isPreferLocalHost();
}
```

The ClientProperties object is cached in the ClientClusterContext. You can retrieve or set the ClientProperties through ClientClusterContext:

- ▶ `getClientProperties(ogName)`
This method retrieves the ClientProperties for this ObjectGrid. Each ObjectGrid can have a ClientProperties.
- ▶ `setClientProperties(ogName, url)`
This method tells whether the client properties file can be loaded with a specific ObjectGrid.

Client properties file example

The first step is to create a client properties file. A sample `objectGridClient.properties` file is shown in Example 2-4.

Example 2-4 objectGridClient.properties file

```
# eXtreme Scale client config
preferLocalProcess = false
preferLocalhost = false
preferZones = Zone1, Zone2
```

The second step is to make the client aware of the client properties file.

The default client property file name is `objectGridClient.properties`. WebSphere eXtreme Scale searches for a client property file with this name during the client startup and during client connect process in the client class path. If the file is found in the client class path, it is loaded automatically. If you use the default client properties file name, and if you also put that file in the root of your class path, it is loaded automatically and you do not need to do anything else.

If you use a different client properties file name, or if you put the client properties file outside of the class path, you have two ways to make the client aware of your properties file:

- ▶ Load the file in the application during your client connect process as shown in Example 2-5:

Example 2-5 Loading the client properties file during the client connect process

```
ClientClusterContext ccc = manager.connect("localhost:2809", null, null);
    URL clientPropsURL = Thread.currentThread().
```

```
getContextClassLoader().getResource("etc/myObjectGridClient.properties");
ClientProperties props = ccc.setClientProperties("myOGName", clientPropsURL);
```

- ▶ Use the following system property to make client aware of your client properties file:

```
-Dcom.ibm.websphere.objectgrid.ClientProperties=<fileName>
```

Programming example

Example 2-6 shows how to use `getClientClusterContext`, how to use `getClientProperties`, and how to set the routing preference to zones programmatically. This example also illustrates how you can override the preferences client property file sets programmatically.

Example 2-6 Setting zone-based routing

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
ClientClusterContext ccc = manager.connect("localhost:2809", null, null);
ClientProperties cp=ccc.getClientProperties("accounting");

String [] preferZones= new String[]{"Zone1", "Zone2"};
cp.setPreferZones(preferZones);

ObjectGrid objectGrid = manager.getObjectGrid(ccc, "accounting");
Session session = objectGrid.getSession();

ObjectMap map = session.getMap("payroll");
session.begin();
map.insert("key", "value");
session.commit();
```

Routing behaviors

When a client is set to prefer zones, the following routing behavior occurs:

1. WebSphere eXtreme Scale looks for a target in the first zone in the preferred zones array. If a target is found and is available, eXtreme Scale routes to this target.
2. If the target is not available in that zone, eXtreme Scale tries the second zone in the preferred zones array. If a target is found, eXtreme Scale routes to that target.
3. If the target is not available in that zone, eXtreme Scale attempts to find an available target in each of the zones in the preferred zones array.
4. If eXtreme Scale cannot find an available target in any of the preferred zones, it routes the requests to other zones that are not in the preferred zones.
5. If eXtreme Scale cannot find an available target in any zone, it issues a `TargetNotAvailableException` error.

When the `preferZones` preference is enabled, the `preferLocalProcess` and `preferLocalhost` settings are disabled automatically. Otherwise, the `preferLocalProcess` and `preferLocalhost` settings are enabled by default.

2.8.2 IBM eXtreme Data Format

WebSphere eXtreme Scale provides a new data serialization format called IBM eXtreme Data Format (XDF). The format includes the serialization capabilities and the wire format that are used by WebSphere eXtreme Scale when you are using eXtremeIO as the transport and the `COPY_TO_BYTES` copyMode.

Benefits

Following are some of the benefits of using XDF:

- ▶ More efficient metadata for serialization.
- ▶ Improve serialization performance of 20% to 30% that is faster than the default Java serialization.
- ▶ More compact representation.
- ▶ Common serialization format for .NET and Java applications.
- ▶ Usable without having to change object definitions. Classes do not need to be defined as serializable to use eXtreme Data Format.

Requirements

The following are some of the requirements to use XDF:

- ▶ In the ObjectGrid descriptor XML file, set the `copyMode` attribute to `COPY_TO_BYTES` in the `backingMap` element of the ObjectGrid descriptor XML file:

```
<backingMap name="Employee" lockStrategy="PESSIMISTIC"
copyMode="COPY_TO_BYTES">
```
- ▶ If Java and C# clients are used, the type that is used in the classes must be compatible. For example, `boolean` and `bool`.

XDF is not set when a map is configured with an `ObjectGridTransformer` or `DataSerializer`. It is not used if the Java class implements the `externalizable` interface or implements the `serializable` interface and the `readObject` and `writeObject` methods. Similarly, if you use a C# class that implements the `ISerializable` interface and `getObjectData` method, XDF is not used.

2.8.3 IBM eXtremeMemory

WebSphere eXtreme Scale provides IBM eXtremeMemory (XM) to store objects in native memory instead of the Java heap. By avoiding long garbage collection pauses that are caused by storing data in the Java heap, the grid can have a more constant performance and response time. When an application uses eXtremeMemory, the response time is lower and more constant compared with storing objects in the heap.

XM uses the eXtremeIO transport. Objects are serialized into bytes on the container server. If you have a catalog server and a container server that are collocated, the container servers use eXtremeMemory, but the catalog server does not.

Figure 2-25 shows a comparison of using eXtreme Scale process and eXtremeMemory process.

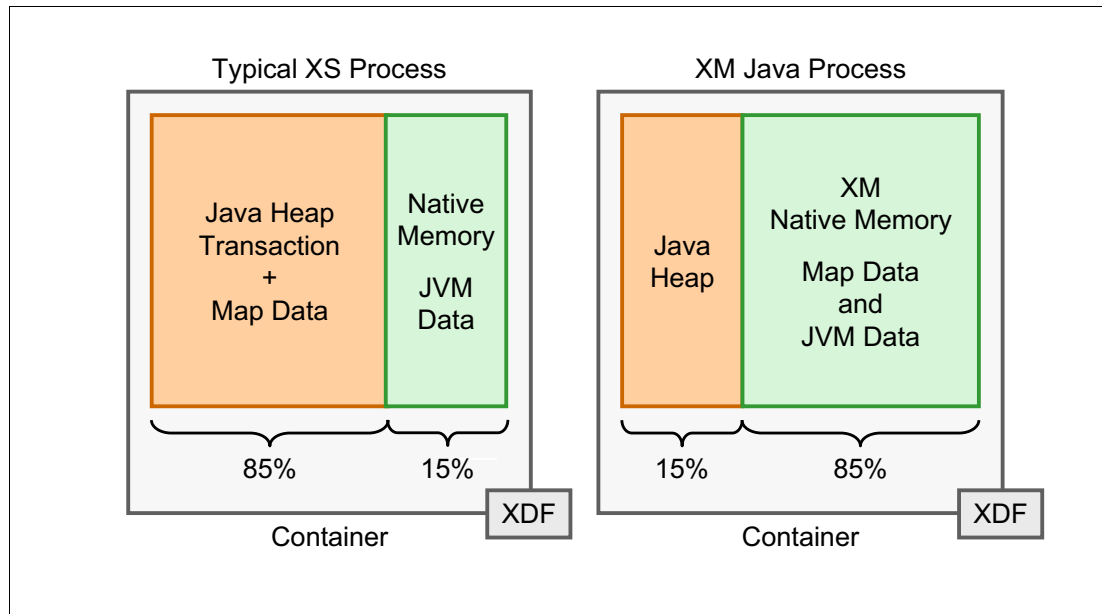


Figure 2-25 Comparison of using eXtreme Scale process and eXtremeMemory process

Benefits

Following are some of the benefits of using XM:

- ▶ More consistence response time by avoiding long garbage collection pauses.
- ▶ The lifecycle of the objects using eXtremeMemory is managed by the well-known lifecycle events of GET, INSERT, DELETE, and UPDATE. Key and value allocation occurs during INSERT. Key and value deallocate during DELETE.

Figure 2-26 is an example of the data flow when you are using XM and XDF. As shown, a Microsoft .NET application can update a value in the grid and the Java application can get that data.

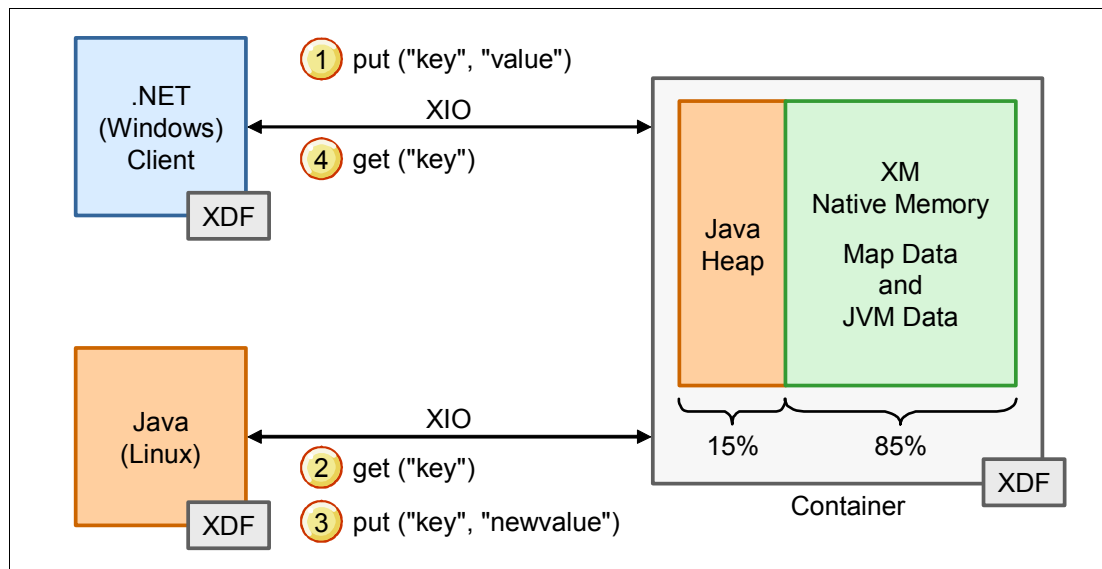


Figure 2-26 eXtremeMemory topology

The flow starts when an application, using a Microsoft .NET platform, inserts a key into the grid. The Microsoft .NET client serializes the object using XDF, then sends it to the container server by using the eXtremeIO transport. The data is stored in the native memory.

The Java client can request the same key from the grid. The object is deserialized by XDF, and the data is then available to the Java client.

For more information about XDF, see 2.8.2, “IBM eXtreme Data Format” on page 66.

Requirements

The following are some of the requirements to use XM:

- ▶ XM is supported on Linux x64, Solaris 10+ Sparc, and AIX 6.1+ (ppc).
- ▶ All maps in the map set must use either `COPY_TO_BYTES` or `COPY_TO_BYTES_RAW` copy mode. If any maps within the map set are not using one of these copy modes, objects are stored on the Java heap.
- ▶ The shared resource, `libstdc++.so.5`, must be installed. Use the package installer of your 64-bit Linux distribution to install the required resource file.
- ▶ XM in WebSphere eXtreme Scale v8.6 currently is not supported when using the following items:
 - Custom evictor plug-ins
 - Built-in write-behind loaders
- ▶ WebSphere eXtreme Scale version 8.6.0.2 or later must be installed if you are using dynamic or composite indexes.
- ▶ An existing data grid is required to determine the total map sizes.
- ▶ Ensure that all data grids have the same eXtremeMemory setting. All container servers must be started with the `settingenabl eXM=true` or `enabl eXM=false`. You cannot mix these settings.

For more information about the requirements for XM, see the *Configuring IBM eXtremeMemory* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsconfigxm.html>

2.8.4 Multi-master replication

Large-scale applications in multiple data centers, often geographically diverse, provide high performance and fail over and disaster recovery. The WebSphere eXtreme Scale *multi-master replication* (MMR) topology is a good candidate for multiple data center deployments. Each data center is a catalog service domain. It is common to use zones within a catalog service domain and multi-master between catalog service domains.

Multi-master replication is a topology option to connect two or more catalog service domains. In earlier versions of WebSphere eXtreme Scale, the catalog service domain used zones to span data centers. A single catalog service domain can be run with catalog servers and containers that are on the east and west coasts of the United States. However, network latency makes it hard to maintain a stable grid with fast fail over. MMR allows catalog service domains to operate independently while still exchanging data.

MMR uses asynchronous replication for the primary shards in each catalog service domain. MMR provides a new availability model (AP) to eXtreme Scale. Before MMR, eXtreme Scale only supported the CAP theorem (CP).

CAP Theorem: The CAP theorem states that a distributed computer system cannot support more than two of the following three properties: Consistency (of data across nodes), Availability, and Partition tolerance. In these terms, the multi-master topology is considered to be an availability and partition-tolerant (AP) topology with eventual consistency. The zone topology is considered a consistency and availability (CP) topology. The third combination of consistency and availability (CA) does not scale and is not appropriate for an extreme transaction processing environment.

For example, the MMR topology can host a catalog data shopping cart or a merchandise catalog where some stale data can be tolerated.

The rules for compatible ObjectGrid configurations that can be linked with MMR can be found in the *Configuration considerations for multi-master topologies* topic of the WebSphere eXtreme Scale Information Center Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsinitconsd.html>

Multi-master replication linking

In a simple configuration of two catalog service domains, the domains replicate to each other. Figure 2-27 shows the simple flow of domains linking together and starting replication between the shards.

In any MMR topology, the basic flow of a replication starts with establishing a link between catalog servers. This can be done by setting properties in the catalog servers properties file or at run time by using the xscmd **establishLinks** command.

The catalog servers exchange placement information for primary shards in compatible ObjectGrids. Each catalog service sends work to its primary shards to initiate replication to the foreign domain. The primary shards replicate asynchronously to their counterparts, or foreign primary, in other domains.

Figure 2-27 illustrates two domains that are running independently.

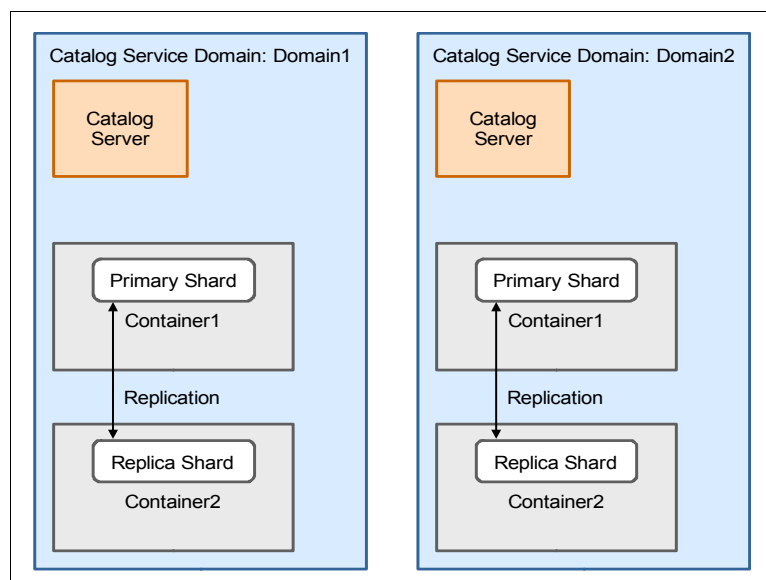


Figure 2-27 A two domain topology, ready to be linked together

Figure 2-28 demonstrates when two domains are linked and the catalog services start sending placement work to their local primary shards.

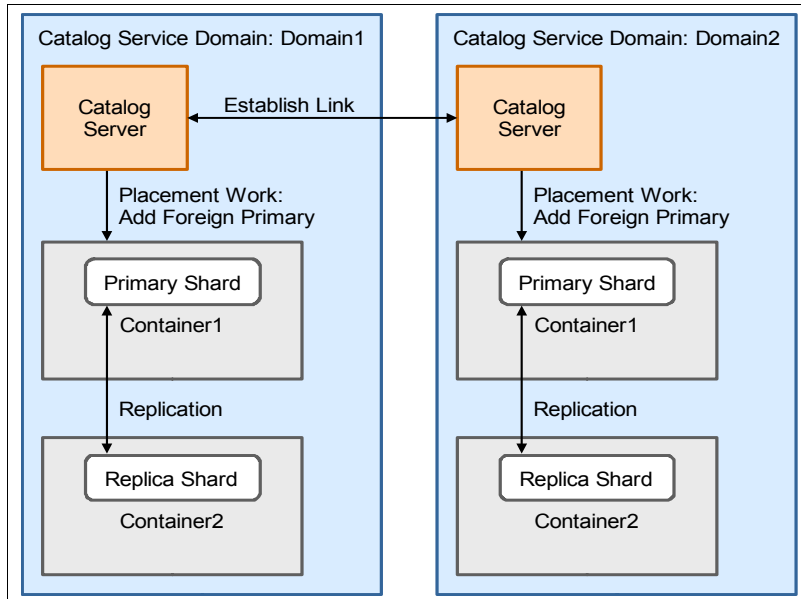


Figure 2-28 Establishing a link between two domains, catalog service sends work to primary shards

Figure 2-29 shows the primary shards doing replication with their foreign primary shards in the other domain.

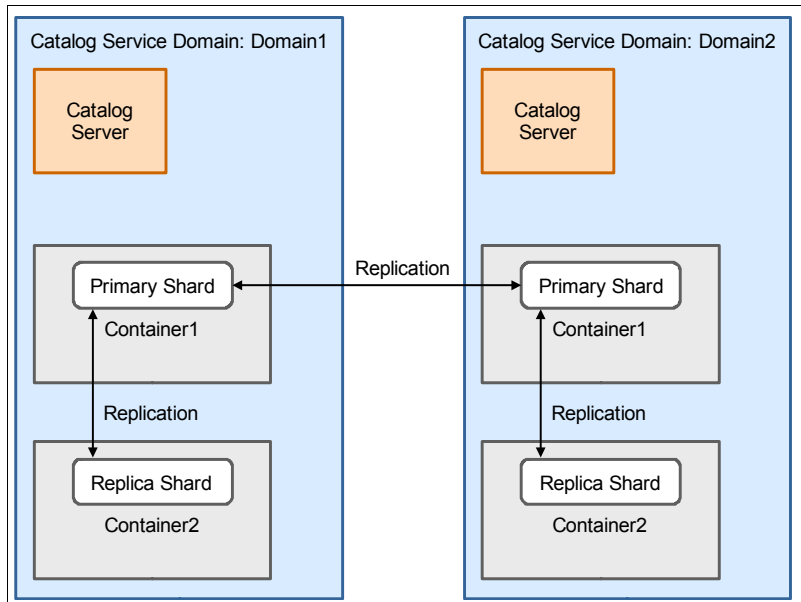


Figure 2-29 Asynchronous replication established between primaries in both domains

When the primary shards replicate to each other asynchronously, they might insert or update the same key at the nearly the same time. When they replicate the changes, the changes conflict or collide. MMR provides arbitration to resolve collisions. For more information, see “Collision arbitration” on page 72.

Collision arbitration

In MMR, data can be written on primary shards in any domain. When the primary shards replicate to each other, they can encounter conflicts or collisions where both primary shards have changed the same key. WebSphere eXtreme Scale uses *arbitration* to decide how to resolve collisions.

Users of MMR can implement an arbiter that is called when a conflict occurs. The arbiter must make a consistent decision on all domains. If domains make different decisions, there will be data inconsistency.

WebSphere eXtreme Scale provides a default arbiter that uses the lowest lexicographical domain name to select what data to keep. For example, if keys from Domain1 and Domain2 are in conflict, both domains choose to keep the data from Domain1.

A collision arbiter implements the `com.ibm.websphere.objectgrid.revision.CollisionArbiter` interface. An arbiter makes a decision on the two entries, the local entry and the incoming entry from the foreign domain.

The arbiter returns a null value, a Resolution type, or an Exception type:

- ▶ Return null: The arbiter abstains, which allows the default arbiter to decide.
- ▶ Resolution.KEEP: Keep the existing entry, and delete the collision entry.
- ▶ Resolution.OVERRIDE: Override the existing entry with the collision entry.
- ▶ Resolution.REVISE: Modify the entry either by merging the entries or removing the entry.
- ▶ Exception: The revision transaction is rolled back. No revision is applied and a message is logged.

For more information about writing arbiters, see the *Design considerations for multi-master replication* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxstopolmmr.html>

An example of a custom collision arbiter is provided in 7.6.11, “Testing a custom collision arbiter for multi-master replication” on page 339.

Multi-domain topologies

When three or more domains are linked together, there are several linking topology layouts:

- ▶ Ring
- ▶ Line
- ▶ Hub and spoke
- ▶ Tree

A ring topology is straight forward to connect. For domain1, domain2, and domain3, they each point to a neighbor to create a ring. The disadvantage is that collisions might have to arbitrate around the ring.

A hub and spoke topology can reduce collision arbitration because all replications must clear through the hub or center domain. The hub domain also must handle a larger burden of replication from all of the spokes.

For more information about these topologies, see the *Topologies for multi-master replication* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fcxslinkscsd.html>

Also, see the *Multi-master replication topology* section in the IBM Redbooks publication *WebSphere eXtreme Scale Best Practices for Operation and Management*, SG24-7964.

Replication between domains

After primary shards link to their respective foreign primaries (the primary shards in the other domains), each primary shard polls the remote primary for data on an adjusting interval. For more information about replication speed, see 2.3.1, “Selecting data to replicate: Revisioning” on page 44.

A primary shard replicates with a foreign primary shard like an asynchronous replica replicates with a primary shard. The additional step is to do arbitration when keys collide. For more information, see “Collision arbitration” on page 72.

When domains link, the replication between primary shards begins quickly. Each primary shard replicates independently and in parallel. On a cold start or restart, this means data are replicating on all primary shards. The replication polling varies on response time from the foreign primary, but the goal is to copy existing data between primary shards quickly. There can be high network usage when a link is established.

Figure 2-30 demonstrates primary shards in different domains replicating with each other. Each primary shard queries the foreign primary shard.

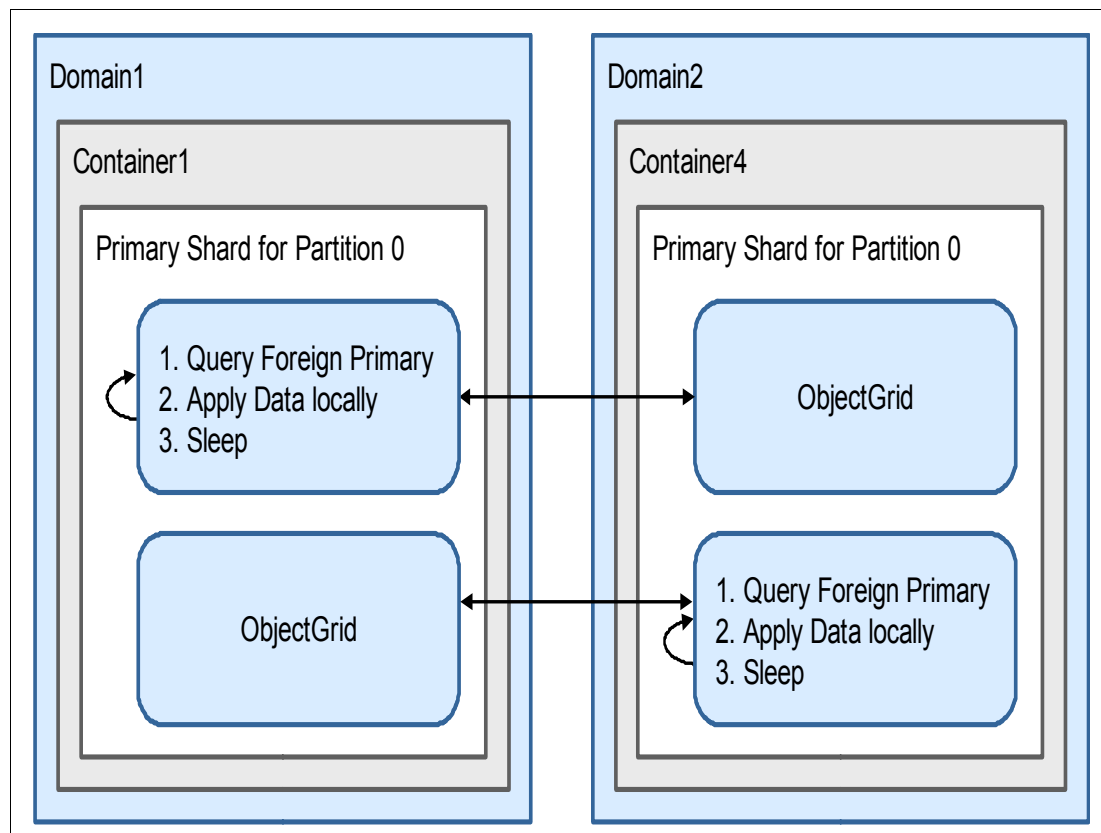


Figure 2-30 Primary shards replicating across domains

Preinstalling with multiple domains does not need to happen on every domain. A preinstall can run against one domain and the MMR linking replicates the preinstalled domain to the rest of the domains. If preinstalling from a database, the preinstall completes before the domains are linked.

For more information about loader considerations, see the *Loader considerations in a multi-master topology* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:
<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsloadermultimaster.html>

Configuring and monitoring MMR

Configuring MMR can be done one of two ways:

- ▶ Using the `xscmd establishLink` command. The user provides the name of the foreign domain and the catalog service endpoints of the foreign domain. The catalog service endpoints are a host name and listener port list of the catalog servers. It is the same catalog server endpoint list used to start container servers and clients.
- ▶ Configuring the catalog server system properties. Add a `foreignDomains` property with the name of the foreign domain. Then, add a `foreignDomainName.endpoints` property with the catalog service endpoints of the foreign domain.

Using the `establishLink` and `dismissLink` commands gives the user the most control on when to link domains. When the catalog server properties are set, the catalog service starts attempts to link at startup. It leaves less flexibility to change the order and timing of the MMR link.

There are several `xscmd` options to view and change the state of the MMR topology. For more information, see 7.2.9, “Multi-master replication commands” on page 320.

Also, see the *Troubleshooting multiple data center configurations* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Ftxsmultidctrb.html>



Application scenarios

With each WebSphere eXtreme Scale scenario, there are a range of implementations, from basic implementations that require no code to advanced scenarios that require custom development. This chapter describes a sampling of implementations along with the benefits and considerations of using each approach.

The scenarios described are largely introduced in a product-agnostic way because eXtreme Scale can apply to a wide range of implementing technologies. For more information about which IBM products integrate with eXtreme Scale, see 3.6, “Integration with other IBM products” on page 90.

This chapter includes the following sections:

- ▶ 3.1, “Introducing the scenarios” on page 76
- ▶ 3.2, “Application state store scenario” on page 77
- ▶ 3.3, “Side cache scenario” on page 79
- ▶ 3.4, “In-line cache scenario” on page 83
- ▶ 3.5, “Extreme Transaction Processing scenario” on page 85
- ▶ 3.6, “Integration with other IBM products” on page 90

3.1 Introducing the scenarios

Today, caching is typically performed manually by using code and is stored locally to the application. The requirement for caching is growing as the amount of data available continues to grow exponentially. It is worth considering the following challenges of storing the cache locally to the application:

- ▶ A local cache does not scale. It is restricted to the size of the application JVM, and must leave room for the actual application processing.
- ▶ A local cache is not fault tolerant or highly available. If the application JVM fails, the application state is lost.
- ▶ A failure of an application server causes the application to hit a “cold cache”, which can cause serious stability issues under peak load situations.
- ▶ Attempts at replicating state across application servers introduces scalability restrictions.
- ▶ Many separate local caches cause data consistency challenges.
- ▶ The local cache memory requirements can degrade performance because of garbage collection cycles on large JVM heap sizes.

This chapter introduces solutions to these challenges applied in the following application scenarios:

- ▶ Application state store
- ▶ Side cache
- ▶ In-line cache
- ▶ Extreme Transaction Processing or Data grid

The application state store and side cache scenarios are typically simple to implement and are widely used. These two scenarios can be applied to use with the WebSphere DataPower XC10 Appliance and the WebSphere eXtreme Scale.

The in-line cache and Extreme Transaction Processing scenarios are more advanced use cases, typically involving some bespoke code running on the grid itself. Therefore, these scenarios apply only to WebSphere eXtreme Scale, and not to the WebSphere DataPower XC10 Appliance.

Each scenario has different characteristics as shown in Table 3-1.

Table 3-1 Scenarios compared by characteristics

Scenario	Complexity	Application change required	System of record
Application state store	Simple	Depends	Grid
Side cache	Simple	Depends	Back end
In-line cache	Medium	Yes	Grid
Extreme transaction processing	Medium to advanced	Yes	Grid

3.2 Application state store scenario

The application state store scenario solves the fundamental question of server-side application development of “*What should I do with my application state?*”

Solutions to this problem are typically some variation of holding and managing application state within the application server. All of these suffer from the challenges of a local cache as outlined in 3.1, “Introducing the scenarios” on page 76.

WebSphere eXtreme Scale can be used instead to offload the application state into a single, coherent, highly available, and scalable cache as illustrated in Figure 3-1. Doing so provides the following benefits:

- ▶ It is a single replacement for multiple local caches, reducing overall memory requirement.
- ▶ Removes the need to manage state replication and consistency.
- ▶ Stateless applications can scale linearly. This fits well with cloud applications that must grow rapidly to meet peak demand.
- ▶ Improved memory utilization leaves more memory for applications.
- ▶ If required, application state can be shared across data centers for high availability.

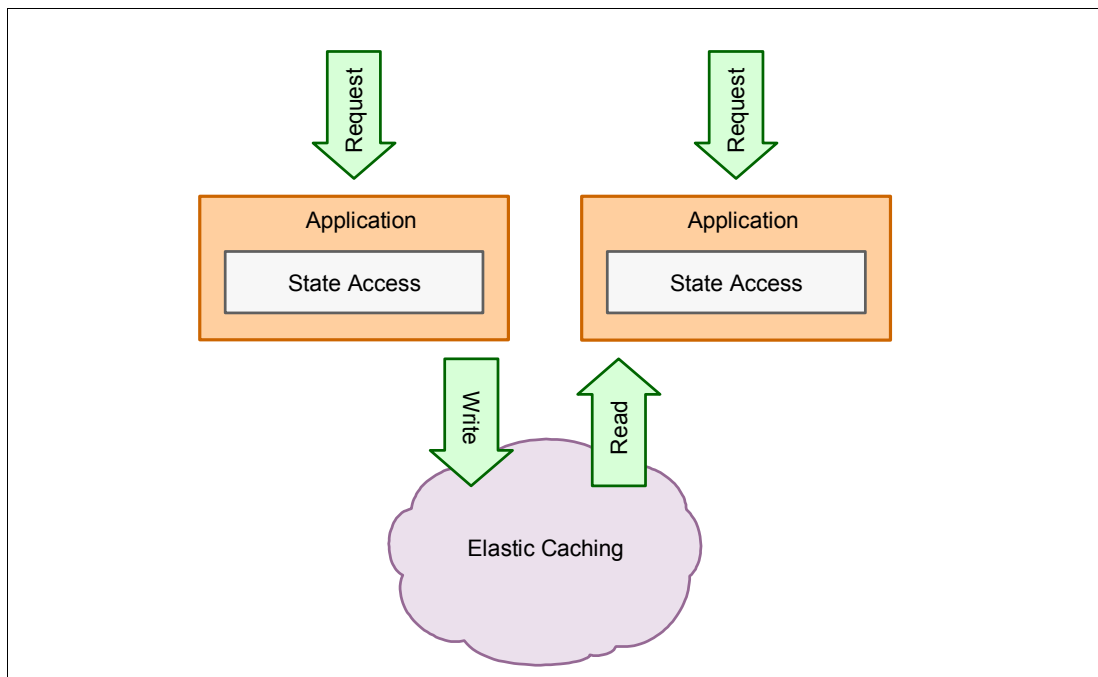


Figure 3-1 Application state store scenario

In terms of implementation, the following technical characteristics typically apply:

Complexity	Simple, and in some cases, is completely transparent to the application.
Application change	Might not need code change, depending on implementation specifics.
Technology	The application state store scenario can be implemented on both WebSphere eXtreme Scale and the WebSphere DataPower XC10 Caching appliance.

3.2.1 HTTP session distribution

User experience is critical. Without HTTP session replication, a planned or unplanned failure of an application server at best requires the user to log on again. At worst it can cause the user a significant amount of work to be done again, or in the case of a consumer application, go to a competitor website.

WebSphere eXtreme Scale takes a copy of the user HTTP sessions to provide session fail over in the event of failure. This copy provides an alternative mechanism for HTTP session replication over those available with WebSphere Application Server, which are database and memory to memory session replication. WebSphere eXtreme Scale also works with other application servers.

HTTP session replication has these key benefits:

- ▶ Greater scale than existing mechanisms in terms of amount of storage and cluster size.
- ▶ Provides a compelling solution for active/active data center HTTP session fail over, which is challenging with existing WebSphere Application Server mechanisms.
- ▶ Simpler administration for transient data than a database.

No coding is required to implement this.

Note: Before WebSphere eXtreme Scale v8.6, there was the requirement for HTTP session objects to implement Serializable. This is no longer required.

The implementation of this HTTP session management is described in Chapter 6, “Extended HTTP session management with WebSphere eXtreme Scale” on page 243.

3.2.2 Application server elasticity

Application server estates must become more flexible to respond to user demand and to cope with widely varying throughput. This can be managed by cloud and virtualization, or more intelligently with the WebSphere Application Server v8.5 Intelligent Management capability, which can dynamically adjust cluster sizes to meet business service level requirements.

This elasticity provides scale on demand, but also brings the requirement to be stateless to ensure scale and versatility. WebSphere eXtreme Scale provides the solution for the application state store.

Application server elasticity has the following key benefits:

- ▶ Allows application tier to be stateless and therefore scale as needed.
- ▶ The elastic caching tier can also scale as required to meet demand for data quantity and throughput.
- ▶ Application state is made highly available.

Using the ObjectMap API, minimal coding is required to implement application server elasticity. For more information about developing with the ObjectMap APIs, see 4.2, “Developing for WebSphere eXtreme Scale” on page 107.

3.2.3 Enterprise service bus state management

Typically the design point of an enterprise service bus (ESB) architecture is to keep it stateless. This provides benefits such as a scalable deployment model. However, as with any state management, this can be difficult with what is typically an asynchronous or event-driven deployment.

WebSphere Message Broker v8.0 now includes embedded WebSphere eXtreme Scale functionality to provide a simple way to use shared state between brokers that require only WebSphere Message Broker flow development. If needed, WebSphere Message Broker can also talk to external WebSphere eXtreme Scale grids and to the WebSphere DataPower XC10 Appliance.

WebSphere eXtreme Scale provides a sensible place for managing state in an asynchronous environment with the following key benefits:

- ▶ Managing ESB state lifecycle provides tools to manage data freshness, such as time-to-live evictors.
- ▶ Independent scale and availability of state data.

No coding is required for WebSphere Message Broker, although some flow development is needed. Other ESBs require minimal coding dependent on the technology using either the ObjectMap API or REST interface.

For more information about the use of the WebSphere Message Broker global cache, see 3.6.4, “WebSphere Message Broker” on page 92.

3.3 Side cache scenario

The purpose of the side cache scenario is to offload back-end data systems. This can be for reasons of performance and scale, or just for cost reasons. If you for example have a third party service that costs for every query, it makes sense to cache it.

WebSphere eXtreme Scale is responsible for holding a copy of all or a subset of the back-end data. This provides similar benefits to the application state store scenario that is described in 3.2, “Application state store scenario” on page 77.

In a side cache scenario, the application continues to make calls to the data system. It is the application responsibility to access the grid to retrieve, store, and possibly invalidate cached data. Therefore, the grid is portrayed as being to the side as shown in Figure 3-2.

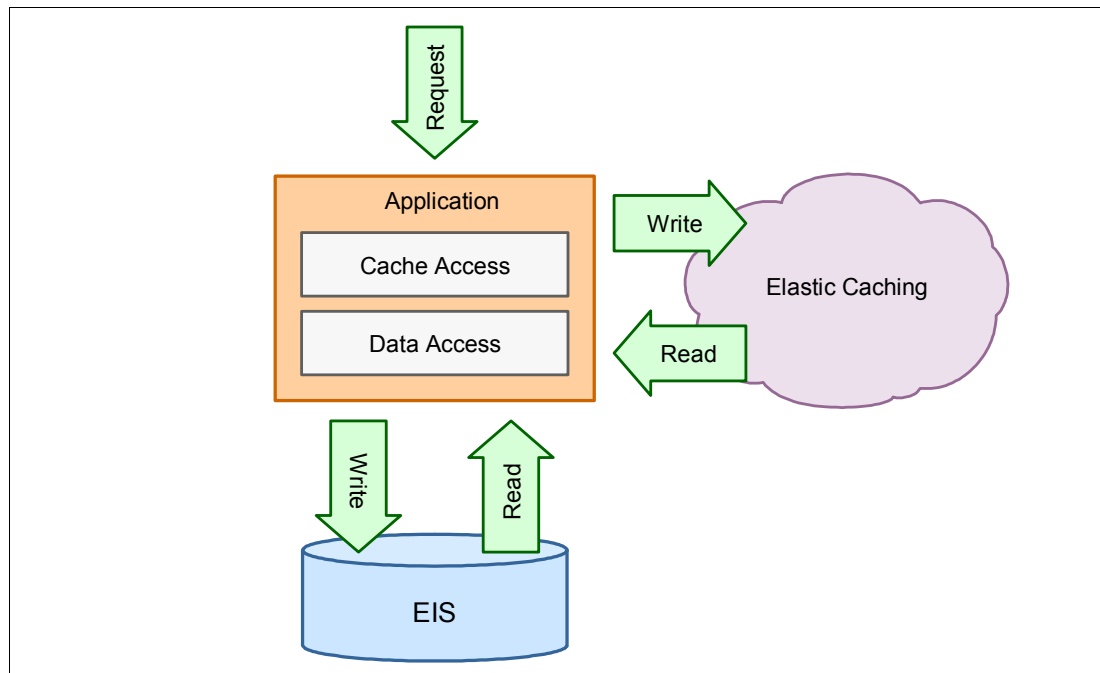


Figure 3-2 Side cache scenario

In addition to the benefits outlined in 3.2, “Application state store scenario” on page 77, the side cache adds the following benefits:

- ▶ Reduce cost of expensive back-end systems.
- ▶ Allow existing back-end systems to scale to cope with modern-day workloads.
- ▶ Maintain high performance and consistent response times.

In terms of implementation, the following technical characteristics typically apply to the side cache scenario:

Complexity	The side cache is typically simple to implement because it often relies on simple create, retrieve, update, and delete operations.
Application change	Might not need code change, depending on implementation specifics.
Technology	The side cache scenario can be implemented on both WebSphere eXtreme Scale and the WebSphere DataPower XC10 Caching appliance.

3.3.1 Implementation overview

Figure 3-3 on page 81 shows a sequence flow that is typical for the side cache scenario. Because simple get/put logic is used to access the grid, the ObjectMap API is a natural choice. The data is accessed only by primary key, and usually no complex relationships between objects must be considered. The back end remains the system of record.

Before retrieving data from the data source, the application checks the grid. If the data is in the grid (cache hit), it is returned to the user. If the data is not in the grid (cache miss), the application retrieves it from the data source, stores it in the grid, and returns it to the user.

At application startup, it might be beneficial to manually fill the cache. This is called *cache warming*, and is described in 4.3, “Loading data into the grid” on page 111.

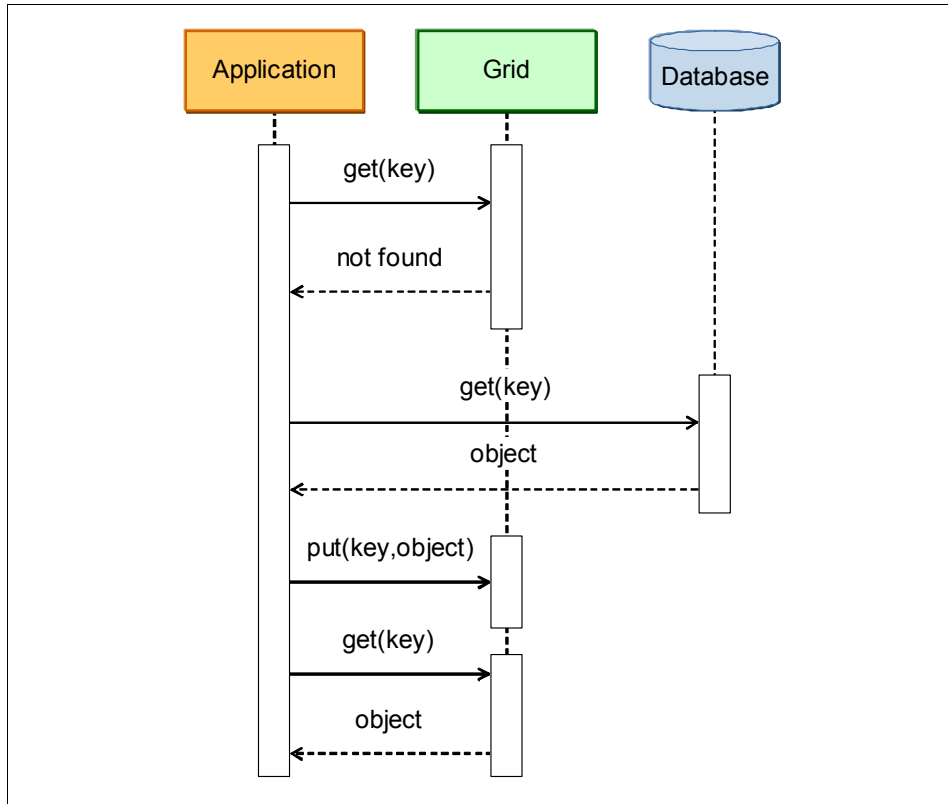


Figure 3-3 Sequence diagram that shows the interaction in a side cache scenario

3.3.2 Dynamic cache provider

The dynamic cache provider or *DynaCache* is part of WebSphere Application Server. DynaCache provides an API for caching dynamic web content (servlets and JSPs) and Java objects. It is a public API available for all WebSphere applications. It is widely used by the WebSphere Commerce Server and WebSphere Portal Server products.

DynaCache provides a powerful performance optimization for WebSphere applications. However, it suffers from the challenges of the local cache as outlined in 3.1, “Introducing the scenarios” on page 76.

With large e-commerce environments, DynaCache has the following challenges:

- ▶ Cache size limit restricts caching of growing product inventories.
- ▶ Significant redundant work is being done in the system to populate and manage the many duplicated caches.
- ▶ Site availability can be compromised in the case of application server failure because of hitting a cold cache.
- ▶ Managing cache consistency in a commerce environment can be critical.

By contrast, offloading the cache to a single, coherent, scalable, and fault-tolerant system can remove these challenges, providing the following tangible benefits:

- ▶ Performance gains are possible as data is now all in-memory and does not require the DynaCache disk offload.
- ▶ Site throughput gains are possible as a result of removing cache redundancy.
- ▶ Scalability of cache (product catalog) size.
- ▶ Dramatic improvement in the time to reach steady-state (warm cache) after full or partial site restart, or after full cache invalidation.
- ▶ Cache is always coherent and consistent.

For more information about the benefits of using WebSphere Commerce Server with WebSphere eXtreme Scale, see *Enhancing WebSphere Commerce performance with WebSphere eXtreme Scale* at:

http://www.ibm.com/developerworks/websphere/techjournal/1008_genkin/1008_genkin.html

No coding is required to implement DynaCache. Sometimes some tuning of the DynaCache configuration is needed, but typically it just requires configuration.

For more information about configuring DynaCache, see 3.6.2, “WebSphere Commerce Server” on page 91.

3.3.3 Service-oriented architecture caching

Building an architecture out of reusable and composable services has been a trend in the IT industry over recent years. It is often called *service-oriented architecture* (SOA) and provides many benefits for productivity, flexibility, and reuse. At the same time, it can quickly introduce performance challenges and scalability challenges for older or heavily used systems.

Using WebSphere eXtreme Scale in this context provides a range of benefits:

- ▶ Significantly reduces the load on the back-end system by eliminating redundant requests.
- ▶ Improves overall response time. Response time from the elastic cache is in milliseconds.
- ▶ Minimizes the need to scale hardware to increase processing capacity because the back-end system no longer must handle redundant requests.

This is a simple implementation, typically involving little configuration.

To provide a couple of examples, this pattern can be used with these applications:

- ▶ WebSphere Message Broker v8.0 functionality, which requires the simple addition of a caching node to a broker flow.
- ▶ DataPower requires a little configuration to make a REST call to a WebSphere DataPower XC10 Appliance or WebSphere eXtreme Scale endpoint.

For more information about configuring service-oriented architecture, see 3.6, “Integration with other IBM products” on page 90.

3.3.4 Mobile gateway

Although the mobilization of technology provides immense business opportunities, the dramatic increase in workload is a significant concern to those operating IT environments. In addition to the general increase in workload, mobile use cases can drive huge spikes in workload, requiring far more elastic environments.

A mobile gateway is an architectural construct that is used in an IT architecture to provide an access point for mobile applications to access enterprise data in a managed and secure way. Examples of this are IBM Worklight and IBM DataPower XG45.

By integrating WebSphere eXtreme Scale with a Mobile Gateway, users can see improved performance without having to scale the gateway and related back-end enterprise systems.

This is a simple implementation, typically involving little configuration or some simple coding.

3.3.5 Enterprise data grid (cross-technology access)

The term *enterprise data grid* is used to articulate the wide range of use cases in which an elastic cache applies.

Enterprise data grid introduces the benefit that the cache can be accessed from a range of technologies that include:

- ▶ Java technology that ranges from application servers to stand-alone Java applications.
- ▶ .NET applications can benefit from the data (new in WebSphere eXtreme Scale v8.6).
- ▶ REST endpoints allow any HTTP device or technology to access the data.

Enterprise data grid introduces the idea that it is beneficial to cache a wide range of back-end technologies that include databases, web services, and offline data sources such as files.

Enterprise data grid is a simple implementation, typically involving a little configuration or some simple code.

The enterprise data grid concepts also apply well to the in-line cache scenario as addressed in 3.4, “In-line cache scenario” on page 83.

3.4 In-line cache scenario

The in-line cache scenario is a more sophisticated version of the side cache. In addition to the side cache challenges, it addresses the following challenges:

- ▶ Further offload EIS where there is a higher ratio of writes.
- ▶ Removal of cache-management code in the application.

The in-line cache scenario moves the responsibility of the data management and cache management into the grid as illustrated in Figure 3-4.

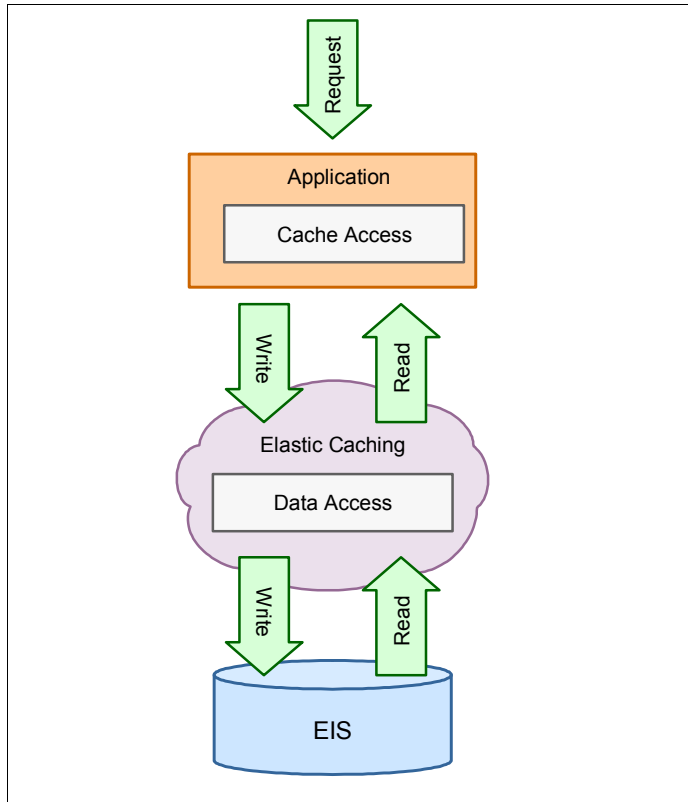


Figure 3-4 In-line cache scenario

During implementation, the following technical characteristics typically apply:

- Complexity** The in-line cache typically requires coding to implement. There is less coding in the client application because it now only must access the grid and not the EIS as well. However, the grid now must access the EIS and run cache management, which involves some coding on the grid.
- Application change** The application must be developed to interface directly to the grid instead of the EIS.
- Technology** The in-line cache scenario can only be implemented on WebSphere eXtreme Scale, and not the WebSphere DataPower XC10 Appliance because it requires code on the grid.

3.4.1 Implementation overview

The in-line cache scenario changes the way that the data is accessed from the EIS.

Whenever the application needs access to data, it always goes directly to the grid. It is the responsibility of the grid to retrieve and update the data. If the requested data is not in the grid, the grid directly accesses the EIS, populates the grid, and returns the data to the application. It uses the loader technology to access the EIS.

In the in-line scenario, the grid is also typically responsible for managing stale data through an eviction mechanism that is based on time, or by querying the EIS.

3.4.2 EIS shock absorber

The EIS “shock absorber” optimizes and offloads read and write operations to a data system. This pattern typically applies to a database system, but because bespoke coding can be used, the pattern can equally apply to the database, NoSQL document database, web service, or older systems.

The in-line cache has the following benefits:

- ▶ Whether the data is in the cache or not becomes transparent to the application. It just experiences fast back-end access.
- ▶ Perform eager loading by using a preinstall to have a more efficient read of the data.
- ▶ Perform lazy loading to store a subset of the EIS data in memory and allow the cache loader to automatically populate the cache on a “cache miss”.
- ▶ Writing data to the grid can automatically replicate the change back to the EIS.
- ▶ The write-behind configuration option batches writes to the EIS so that they can be committed in a single update, reducing the load on the EIS. This process decouples the application availability from back-end availability. Even when the back end is down, the application can write changes. They are buffered in the grid to be written back when the back-end system comes online again. The grid can be configured to replicate data to ensure that the data are not lost in case of grid container failure.

Implementation overview

To implement an in-line cache, a loader must be implemented. There are ready to use loaders provided with WebSphere eXtreme Scale, but typically some bespoke code is needed to load the data according to the application use. Generally, transform the data from the EIS schema to a form readily accessible by the application before storing it in the grid. This process helps ensure that the application access times are optimal.

The development work that is needed to implement an in-line cache is described in 4.3, “Loading data into the grid” on page 111.

When you use write-behind caching, special error-handling might be required. If a write-behind fails, these errors cannot be reported back to the client because the client request has already been completed. Information about failures is stored and can be handled.

From a maintenance point of view, the in-line caching scenario requires that you have application code on the grid. Processes for managing and updating grid code must be introduced.

3.5 Extreme Transaction Processing scenario

Extreme Transaction Processing (XTP) is the term used for high-end transactional processing requirements. It typically includes large quantities of data and large amounts of concurrent processing. It might or might not also involve large numbers of concurrent client queries.

The data grid introduces a different paradigm that is based on the *collocation* of application logic with the application data as shown in Figure 3-5. Application processing is run on the grid with the data. This provides some significant benefits that include:

- ▶ High degree of parallelization is possible. Very large data sets can be processed quickly.
- ▶ Data processing is run in the container where the data is held. There is no need to move the data to the client to process it. For XTP scenarios, this configuration is far more efficient.
- ▶ XTP applications use partitioned data that is balanced across many grid containers. If greater processing or data capacity is needed, more containers can be added to the grid.

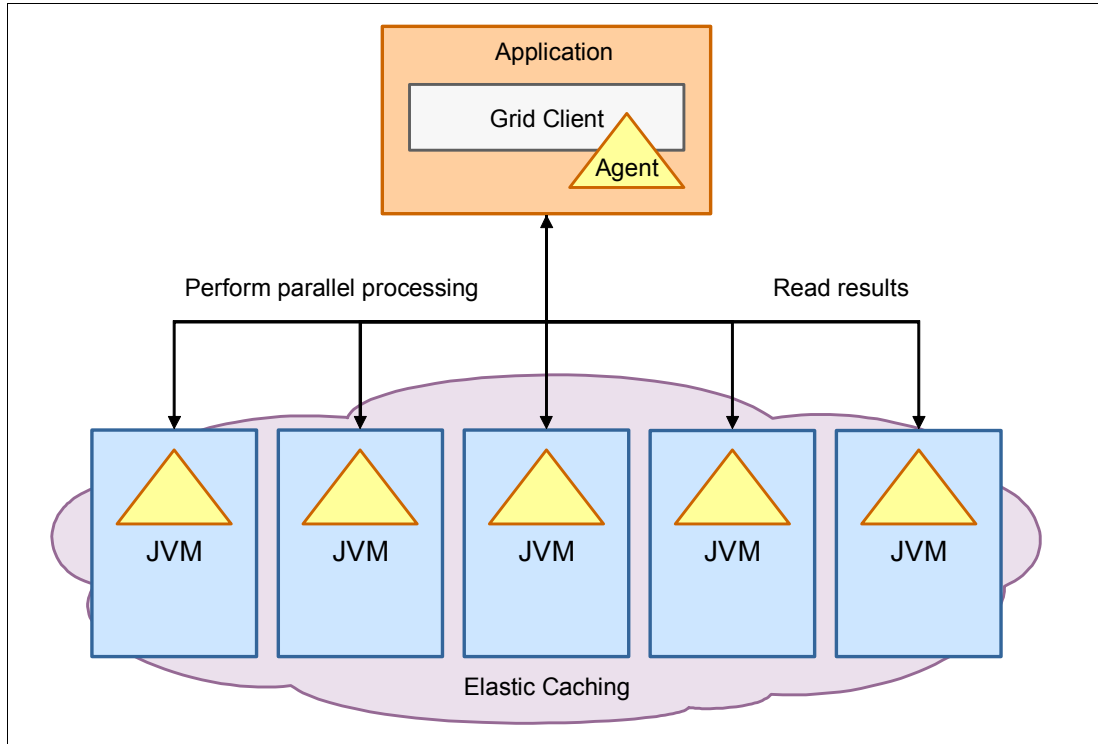


Figure 3-5 Extreme Transaction Processing scenario

In terms of implementation, the following technical characteristics typically apply:

Complexity

Application complexity can vary significantly with XTP applications. Each application is bespoke to the requirements and, because of the high-end processing requirements, can be complex to implement.

Keep in mind that accessing the grid and running queries over the data contained in the grid is not like using a database. In many cases, it is not as straightforward as working with the database because of the use of partitioning in the grid. Put careful thought and planning into the design of the partitioning of the data in the grid to optimize the path for data access, data querying, and data processing.

Finally, a bespoke solution for preloading and managing data freshness is typically required in this scenario.

This additional complexity is the trade-off for the linear scalability of the grid over the database solution.

Application change

XTP applications must be developed specifically to meet the requirements of the scenario.

Technology

The XTP scenario can only be implemented on WebSphere eXtreme Scale, and not the WebSphere DataPower XC10 Appliance because it requires code on the grid where the core processing logic of the application is.

NoSQL: The following link is an excellent introduction to the WebSphere eXtreme Scale world of NoSQL as compared to the relational database world:

<http://www.infoq.com/presentations/Enterprise-NoSQL>

3.5.1 Implementation overview

At the heart of the data grid implementation is the concept of using *agents*. Agents contain logic that can be run in parallel in every partition in the grid, or in a subset of the partitions of the grid, as defined by a set of keys.

Agent development is done with WebSphere eXtreme Scale DataGrid APIs. The DataGrid APIs provide a vehicle to run logic collocated with the data and aggregate large result sets into a single result set.

Two major query patterns exist for DataGrid applications:

- ▶ Parallel map, which processes a set of objects and returns a result set containing an entry for each object processed.
- ▶ Parallel reduction, which processes a set of objects and calculates a single result for the set.

MapReduce: These APIs conform to the pattern known as *MapReduce*, which is common across XTP and big data scenarios. The model was originally developed by Google:

<http://research.google.com/archive/mapreduce.html>

3.5.2 MapReduce programming model

MapReduce is at the heart of XTP on WebSphere eXtreme Scale, and is the mechanism by which high degrees of parallelization can be obtained. This section provides more detail on the MapReduce programming model, and introduces the two data grid agent APIs: MapGridAgent and ReduceGridAgent.

Using the data grid agents in WebSphere eXtreme Scale provides the following benefits:

- ▶ More efficiently query data across the grid because of the partition-aware processing capabilities of the agents. Without the agents, the application must query each partition individually, take the result sets for each query, and manually collate them into a single result set.
- ▶ Significantly reduce the amount of data that is transported (and therefore serialized) between the grid and the client.
- ▶ Scale application processing with a linear scaling model.
- ▶ The elastic cache is typically the system of record, so there can be little load on the back-end data stores.

MapGridAgent API

Parallel map is used to run some logic on the data in the grid, and returns a set of the processed objects.

In the example in Figure 3-6, a client application must get some data from the grid, which is represented by partitions one and two. The application creates an agent, which is an instance of `com.ibm.websphere.objectgrid.datagrid.MapGridAgent`. It passes the parameters needed to query the grid for the wanted data (keys, and so on).

When the application starts the agent, the agent class is serialized and sent to the grid partitions. The class is deserialized in each partition, and runs the agent logic in parallel against the data. After the data is processed by the agent logic, the result set is then serialized and sent back to the client application for further processing. A result set from each partition is returned and aggregated into a single, larger result set by the client-side agent.

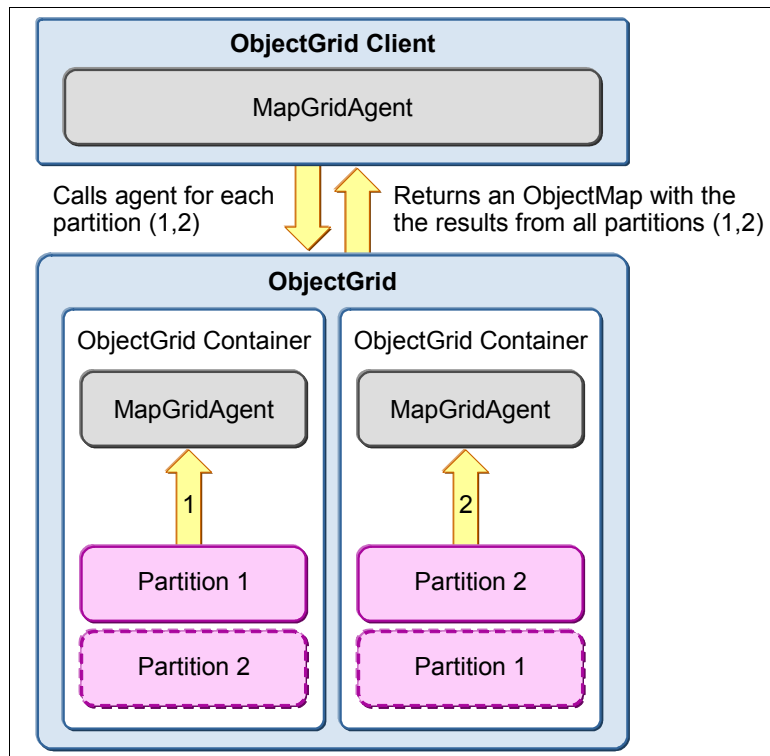


Figure 3-6 Parallel map using the MapGridAgent API

ReduceMapAgent API

Parallel reduction is similar to the parallel map, with one important difference. Instead of returning a result set from each partition, it returns a *reduction*, which is a single result from each partition.

Figure 3-7 shows an agent that is an implementation of *com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent*. The ReduceGridAgent is also serialized and sent to each partition (or a subset if so defined), and the agent runs against the data that are contained in the partitions. The returned data is a single result from each partition. The client-side agent code can further reduce this to a single result from the entire grid.

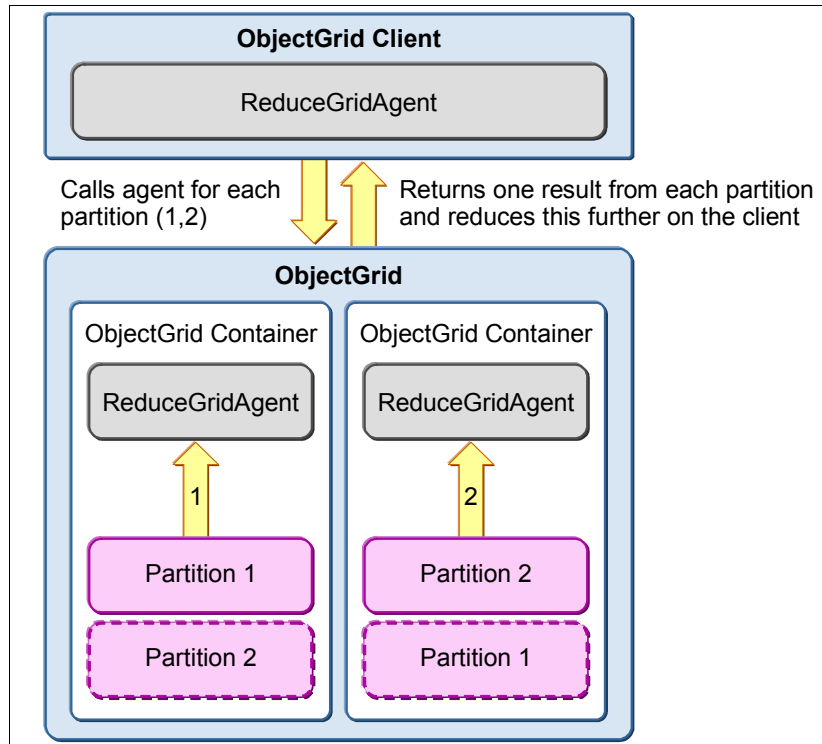


Figure 3-7 Parallel reduction using the ReduceGridAgent API

For more information about implementing agents, see 4.5, “Server-side development with agents” on page 132

3.5.3 Real-time business rules and event processing

Real-time business rules and event processing is an example where the XTP processing paradigm can be combined with other technologies to provide powerful processing capabilities. Business rule processing provides a high-level expression of the processing of input data.

Real-time business rules and event processing provide these benefits:

- ▶ High performance rule execution and parallelization.
- ▶ Improved maintenance of business rules. They do not need to be articulated in code.
- ▶ Allow existing rule and event processing systems to provide highly scale solutions.

Rule processing can be combined with an in-memory data grid to provide rapid access to large amounts of data without causing bottlenecks at the data source. This is possible with two approaches:

- ▶ Rules run against a remote grid as shown in Figure 3-8.

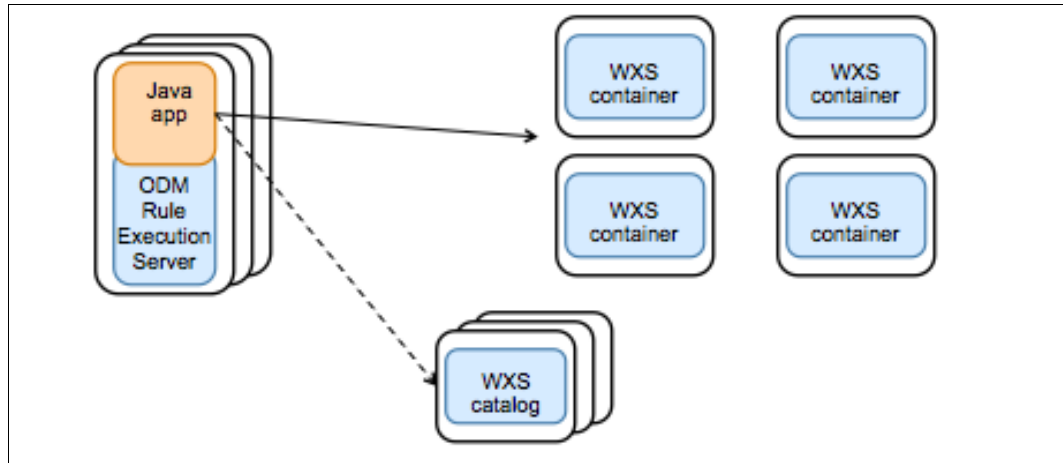


Figure 3-8 Rule execution server running with data in WebSphere eXtreme Scale

- ▶ Rules run within the grid in an Extreme Transaction Processing manner as shown in Figure 3-9.

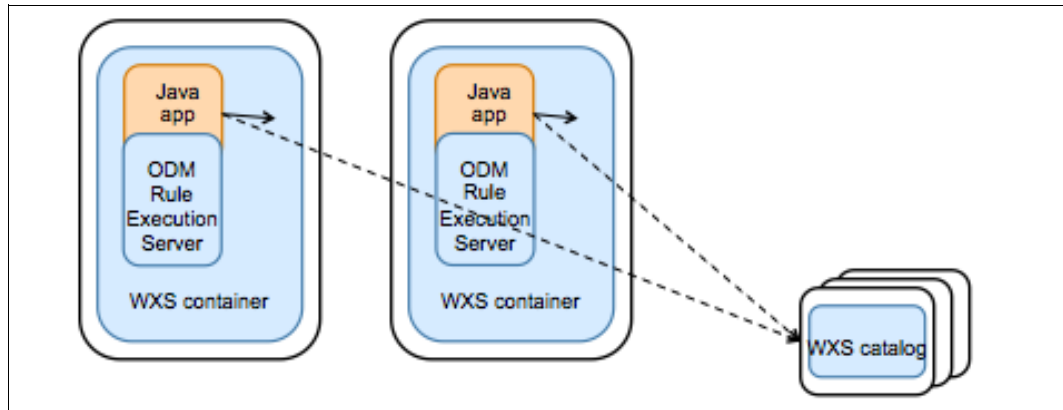


Figure 3-9 Rules running within the grid

3.6 Integration with other IBM products

WebSphere eXtreme Scale integrates with a growing range of IBM products. This section provides a brief summary of these products and provides links to further information. This list will continue to grow over time. It is worth reiterating that WebSphere eXtreme Scale can be used with any Java SE platform. Also, WebSphere Application Server is not referenced, as that solution is addressed widely in this book.

The following products currently have specific integration WebSphere eXtreme Scale:

- ▶ WebSphere Portal Server and Web Content Manager
- ▶ WebSphere Commerce Server
- ▶ IBM WebSphere DataPower XG45 and XI52 appliances
- ▶ WebSphere Message Broker

- ▶ IBM Worklight
- ▶ IBM Business Process Manager
- ▶ Rational Team Concert
- ▶ IBM Operational Decision Management

3.6.1 WebSphere Portal Server and Web Content Manager

WebSphere Portal Server and Web Content Manager integrate with WebSphere eXtreme Scale using a side cache to provide HTTP session and dynamic cache offload as described in 3.3, “Side cache scenario” on page 79. Both WebSphere eXtreme Scale and the WebSphere DataPower XC10 Appliance can be used in this scenario.

For more information about configuring the Dynamic cache, see the *Innovations within reach: Using WebSphere eXtreme Scale to enhance WebSphere Portal and IBM Web Content Manager performance* article:

http://www.ibm.com/developerworks/websphere/techjournal/1206_inreach/1206_inreach.html

More information about HTTP session offload can be found in Chapter 6, “Extended HTTP session management with WebSphere eXtreme Scale” on page 243, and in *Scalable, Integrated Solutions for Elastic Caching Using IBM WebSphere eXtreme Scale*, SG24-7926.

3.6.2 WebSphere Commerce Server

WebSphere Commerce Server integrates with WebSphere eXtreme Scale to enable scaling for product catalog caching. This integration uses a side cache to offload the dynamic cache as described in 3.3, “Side cache scenario” on page 79. Both WebSphere eXtreme Scale and the WebSphere DataPower XC10 Appliance can be used in this scenario.

For more information about dynamic cache, see *Scalable, Integrated Solutions for Elastic Caching Using IBM WebSphere eXtreme Scale*, SG24-7926.

Similarly, this information is summarized in the developerWorks article entitled *Configure WebSphere Commerce with WebSphere eXtreme Scale to improve performance, scale, and your competitive edge*:

http://www.ibm.com/developerworks/websphere/techjournal/1108_bohn/1108_bohn.html

The WebSphere DataPower XC10 Appliance is particularly suited to this use case with WebSphere Commerce Server DynaCache offload. The following are some specific WebSphere DataPower XC10 Appliance references:

- ▶ IBM WebSphere DataPower XC10 V2.0 WebSphere Commerce integration presentation:
 - http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wdatapower/wdatapower/2.0/xc10/XC10_Commerce_Integration/player.html
- ▶ *Integrating WebSphere Commerce with IBM WebSphere DataPower XC10*, REDP-4823.

3.6.3 IBM WebSphere DataPower XG45 and XI52 appliances

WebSphere DataPower XG45 and XI52 appliances rapidly integrates with the WebSphere DataPower XC10 Appliance using REST to provide a side cache for SOA caching as described in 3.3, “Side cache scenario” on page 79.

For more information about configuring this scenario, see the developerWorks article entitled *Integrating WebSphere DataPower XC10 and XI50 Appliances*:

http://www.ibm.com/developerworks/websphere/library/techarticles/1111_nijhawan/1111_nijhawan.html

This integration is also described in *Enterprise Caching Solutions using IBM WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale*, SG24-8043.

3.6.4 WebSphere Message Broker

WebSphere Message Broker v8.0 provides a global cache facility that is integrated into the product. It is based on WebSphere eXtreme Scale to provide both state management (see 3.2, “Application state store scenario” on page 77) and SOA caching (see 3.3, “Side cache scenario” on page 79). If further scaling is needed, it can connect to an external grid on WebSphere eXtreme Scale or the WebSphere DataPower XC10 Appliance.

The broad product support for caching is detailed in the *Managing data caching* topic in the WebSphere Message Broker Version 8.0.0.2 Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v8r0m0/index.jsp?topic=%2Fcom.ibm.etools.mft.doc%2Fbn23725_.htm

This integration is also described in *Enterprise Caching Solutions using IBM WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale*, SG24-8043.

3.6.5 IBM Worklight

IBM Worklight provides adapters to provide mobile applications with access to enterprise data in a secured and managed manner. These application run on the Worklight server. This provides an opportunity to handle caching of frequently requested data to enable internal data systems to cope with the peak workloads of mobile use cases.

This integration is described in detail in *Enterprise Caching Solutions using IBM WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale*, SG24-8043.

3.6.6 IBM Business Process Manager

IBM Business Process Manager typically uses WebSphere eXtreme Scale to provide SOA caching (see 3.3, “Side cache scenario” on page 79). It provides integrated mediations to store to and retrieve from WebSphere eXtreme Scale or the WebSphere DataPower XC10 Appliance.

For more information, see the *Common WebSphere eXtreme Scale scenarios* topic of the IBM Business Process Manager, V8.0.1, All platforms Information Center:

http://pic.dhe.ibm.com/infocenter/dmndhelp/v8r0m1/topic/com.ibm.wbpm.main.doc/topics/esbprog_wxsscenarios.html

A tutorial on integrating with Business Process Manager can be found at the following WebSphere Business Process Management and Connectivity integration developerWorks website:

https://www.ibm.com/developerworks/community/wikis/home?lang=en/wiki/W30b21440b0d9_432c_8e75_b16bac9c5427/page/WebSphere%20Business%20Process%20Management%20and%20Connectivity%20integration

3.6.7 Rational Team Concert

WebSphere eXtreme Scale can be used as a side cache to improve scaling and high availability of IBM Rational® Jazz-based products, such as IBM Rational Team Concert™.

For more information, see *Scalable, Integrated Solutions for Elastic Caching Using IBM WebSphere eXtreme Scale*, SG24-7926.

3.6.8 IBM Operational Decision Management

IBM Operational Decision Management provides the capability to perform decision management across rules and events.

Event processing often must handle and filter large quantities of data. The integration with WebSphere eXtreme Scale allows for sophisticated XTP-style processing of the data. This enables rapid, parallel processing of large amounts of event data.

For more information, see the *Integrating WebSphere Business Events with WebSphere eXtreme Scale* topic of the WebSphere Business Events Information Center:

<http://publib.boulder.ibm.com/infocenter/wbevents/v7r0m1/index.jsp?topic=%2Fcom.ibm.wbe.integrating.doc%2Fdoc%2Fintegrating-wxs.html>

This integration is also described in *Scalable, Integrated Solutions for Elastic Caching Using IBM WebSphere eXtreme Scale*, SG24-7926.



Developing with WebSphere eXtreme Scale

This chapter covers a range of topics that are related to developing with WebSphere eXtreme Scale. The initial sections focus on setting up the development environment and basic development hands-on steps to get you up and running. The latter sections address important facets of WebSphere eXtreme Scale development.

This chapter includes the following sections:

- ▶ 4.1, “Setting up a development environment” on page 96
- ▶ 4.2, “Developing for WebSphere eXtreme Scale” on page 107
- ▶ 4.3, “Loading data into the grid” on page 111
- ▶ 4.4, “Querying data using Object Grid Query Language” on page 122
- ▶ 4.5, “Server-side development with agents” on page 132
- ▶ 4.6, “Dealing with data eviction and stale data” on page 137
- ▶ 4.7, “Saving time with WXSUtils” on page 143
- ▶ 4.8, “Transactions” on page 144

4.1 Setting up a development environment

To introduce the basic development concepts, it is helpful to set up a productive development environment. This section introduces a couple of new (and free) developer tools available for developing with WebSphere eXtreme Scale based on the following environments:

- ▶ Eclipse
- ▶ WebSphere Liberty Profile
- ▶ WebSphere eXtreme Scale Developer Tools

WebSphere Liberty Profile: The WebSphere Liberty Profile is a lightweight WebSphere Application Server that is approximately 40 Mb in size and starts quickly. This section introduces the improved development experience based WebSphere Application Server v8.5 Liberty Profile (referred to as the *Liberty Profile*). If you are not familiar with the Liberty Profile, see the following website:

https://www.ibm.com/developerworks/community/blogs/wasdev/entry/learn_wlp_v8500

WebSphere eXtreme Scale integrates nicely with the Liberty Profile by providing a feature to run the grid. In a development environment, this provides the option to run an entire grid in a single lightweight application server, which improves the development experience.

For completeness, there are some alternatives for developing on eXtreme Scale. Your choice will largely be driven by personal preference. In general, you can develop by using the Liberty Profile, as described in this chapter, and deploy to a different target environment. The following are development alternatives:

- ▶ Java Standard Edition (SE)

The trial for Java SE development can be found on the following website:

<http://www.ibm.com/developerworks/downloads/ws/wsdg/>

- ▶ WebSphere Application Server deployment

The WebSphere Application Server deployment requires the full eXtreme Scale product for a fully integrated deployment. This deployment is addressed in Chapter 5, “Deployment scenarios” on page 151.

- ▶ XC10 Appliance Virtual Image for Developers

Developing for the IBM WebSphere DataPower XC10 Appliance requires the Java SE eXtreme Scale client, but instead connects to the virtual appliance. For more information, refer to the following website:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/introducing_a_developer_xc10_appliance_virtual_image?lang=en

- ▶ Rational Application Developer for WebSphere

The Rational Application Developer for WebSphere product comes with the WebSphere Application Server Developer Tools, so these tools do not need to be installed. Apart from that, the information that is presented in this chapter applies in general.

4.1.1 Installing the development environment

The development environment (shown in Figure 4-1) consists of the following tools. All of these tools are available at no charge for development purposes.

- ▶ Developer tools:
 - Eclipse Juno (4.2)
 - WebSphere Application Server v8.5 Liberty Profile Developer Tools
 - WebSphere eXtreme Scale v8.6 Liberty Profile Developer Tools
- ▶ A runtime server:
 - WebSphere Application Server v8.5 Liberty Profile
 - WebSphere eXtreme Scale v8.6 Liberty Profile Feature

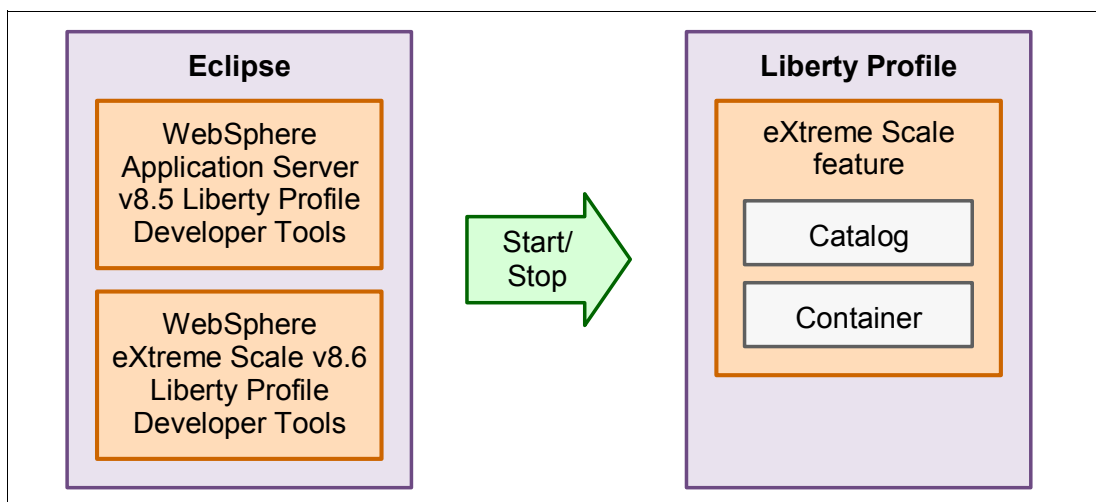


Figure 4-1 Eclipse development with eXtreme Scale and the Liberty Profile

It does not take long to get an eXtreme Scale development environment up and running. However, there are a few pieces to install for the developer tools, which are easily accessed from the Eclipse marketplace and the Liberty server.

Installing the developer tools

Complete the following steps to install the developer tools:

1. Download a copy of Eclipse (Indigo or Juno) from the following website and start it up.
<http://www.eclipse.org/downloads>
2. Go to the Eclipse marketplace by clicking **Help** → **Eclipse Marketplace**.

3. Search for the term *WebSphere* and install the following two features related to eXtreme Scale development:
- WebSphere Application Server v8.5 Liberty Profile Developer Tools, as shown in Figure 4-2.

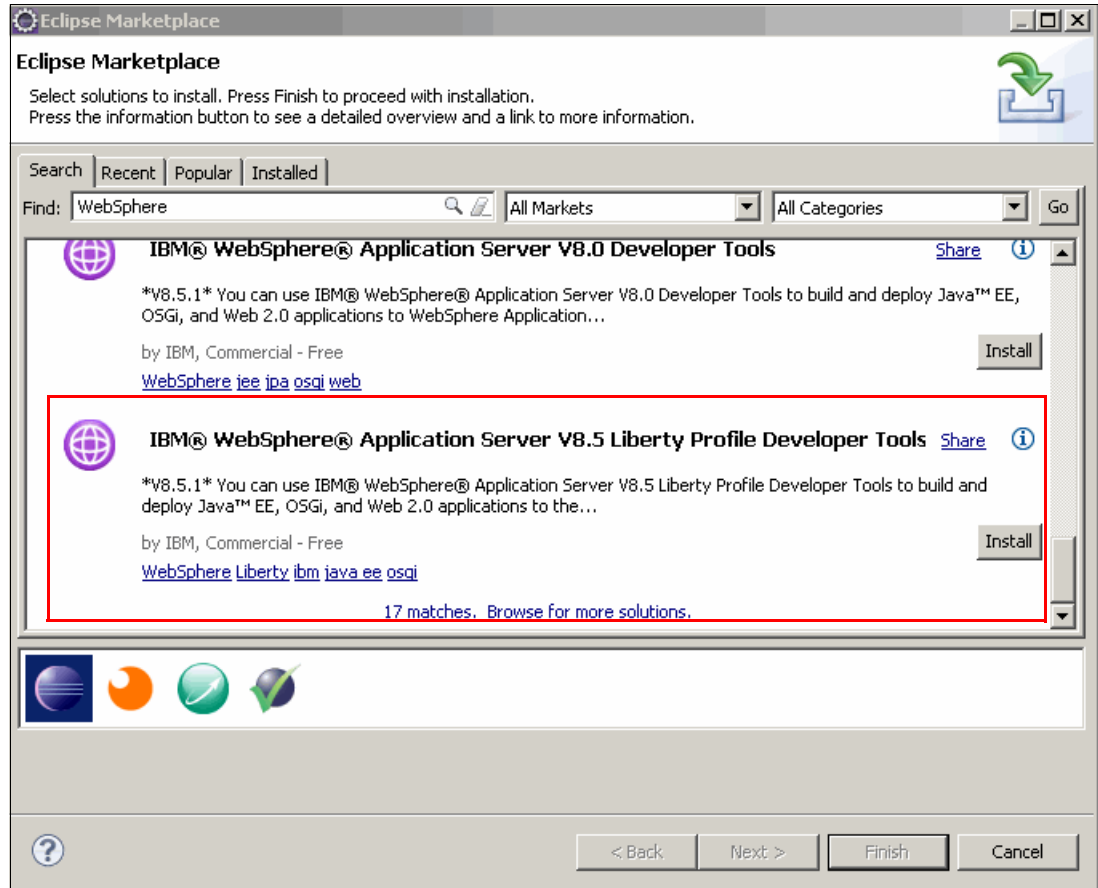


Figure 4-2 WebSphere Application Server v8.5 Liberty Profile Developer tools

- WebSphere eXtreme Scale v8.6 Liberty Profile Developer Tools, as shown in Figure 4-3.

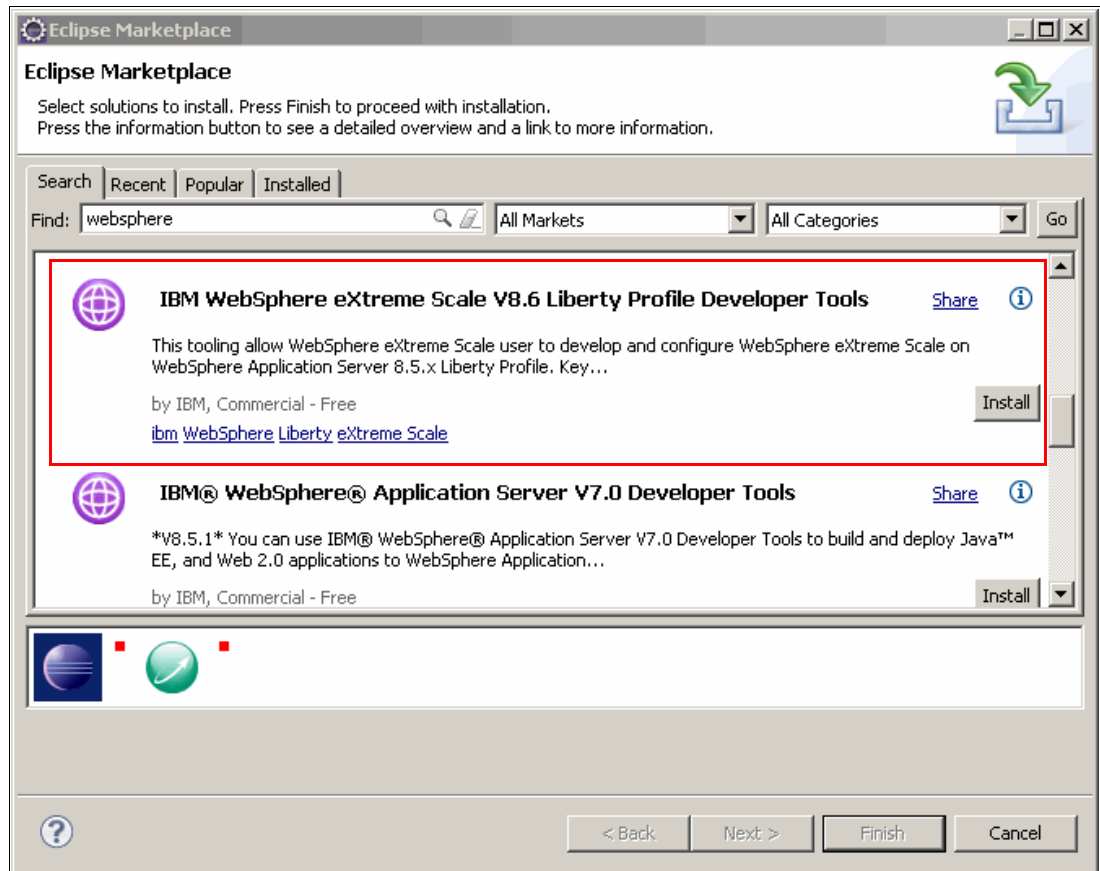


Figure 4-3 WebSphere eXtreme Scale v8.6 Liberty Profile Developer tools

4. Restart Eclipse.

Setting up the Liberty Profile and WebSphere eXtreme Scale feature

Download the WebSphere Application Server v8.5 Liberty Profile (commonly referred to as the Liberty Profile) and WebSphere eXtreme Scale feature by completing the following steps:

1. On the following IBM developer Works WASdev website, download the Liberty Profile by selecting **Downloads** → **Final Releases**. See Figure 4-4 on page 100. The file is approximately 40 Mb.

<http://wasdev.net>

Information: It is worth taking a look at wasdev.net. It is a developer portal for all things that are related to developing for WebSphere Application Server, and has many samples and articles.

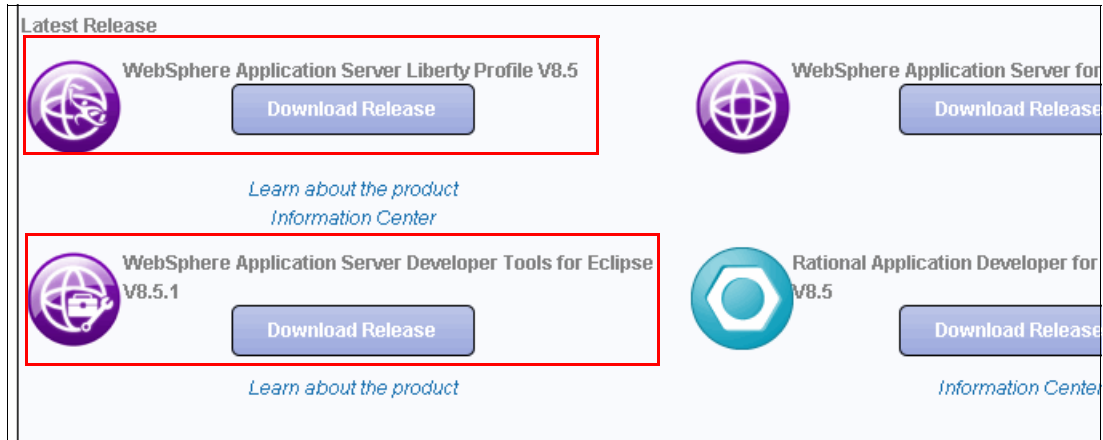


Figure 4-4 wasdev.net Liberty Profile and WebSphere eXtreme Scale downloads

2. Download WebSphere eXtreme Scale V8.6 for Developers - Liberty Profile from the following website. This is the eXtreme Scale feature for the Liberty Profile. The JAR file is approximately 30 Mb.

<http://www.ibm.com/developerworks/downloads/ws/wsdg>

3. Install the Liberty Profile. Open a command prompt and, from the download location, run the following command:

```
java -jar wlp-developers-8.5.0.2.jar
```

Note: Your jar version number might vary.

4. Accept the license agreement and provide the installation location into which you want to install the Liberty Profile. For example, use the following location:

```
c:\wlp
```

5. From the download location, install the eXtreme Scale feature. Open a command prompt and run the following command:

```
java -jar wxs-wlp_8.6.0.1.jar
```

6. Accept the license agreement and provide the installation location into which you installed the Liberty Profile. In this example, the location is c:\wlp.

The Liberty Profile server is now installed with the eXtreme Scale feature. Configure it for use with Eclipse as described in “Creating a Liberty Profile server” on page 100.

Creating a Liberty Profile server

Set up the Liberty Profile in Eclipse to host the grid by completing the following steps:

1. In Eclipse, ensure that you are in the Java EE perspective.
2. Select the **Servers** tab (or if not visible, click **Window** → **Show View** → **Servers**).
3. Right-click in the Servers pane and select **New** → **Servers**.

4. Select **IBM** → **WebSphere Application Server 8.5 Liberty Profile** as shown in Figure 4-5. Click **Next**.

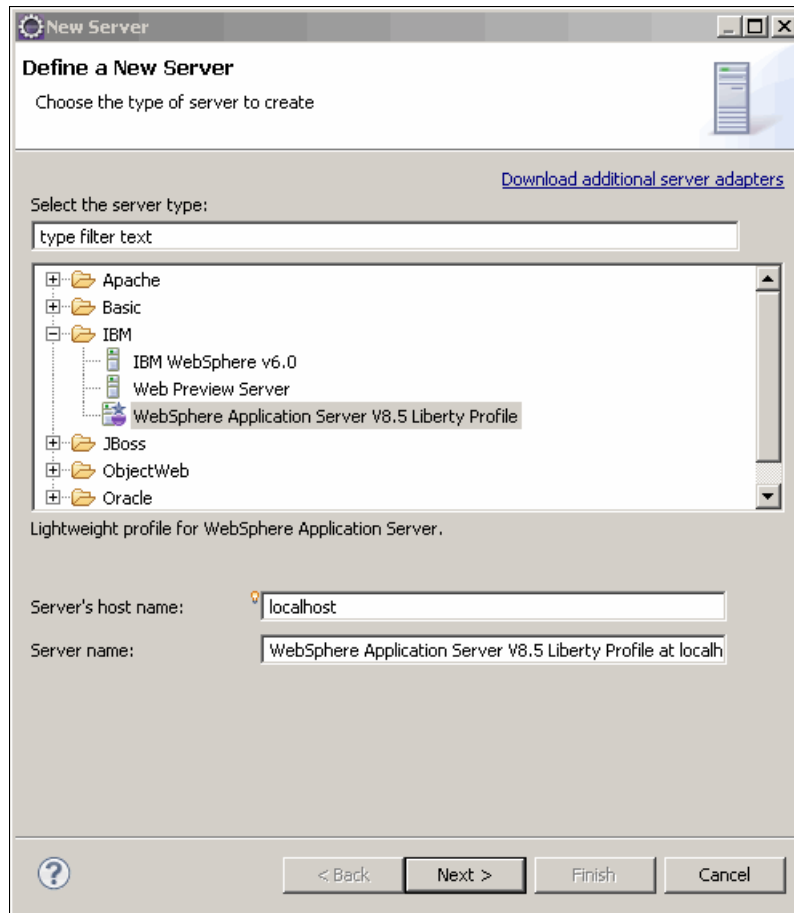


Figure 4-5 Selecting WebSphere Application Server v8.5 Liberty Profile

5. An empty installation folder is displayed, and you are given the option to download and install a new copy of the Liberty Profile. If you want, you can use the Eclipse tools to download the Liberty Profile directly from here. However, because of the requirement for the eXtreme Scale feature, this was downloaded and installed manually.

Enter the Liberty Profile installation location (for the example, this is `c:\wlp`) into the *Installation Folder* field.

This process detects your installation and enables the **Next** button. Click **Next**.

6. Enter an appropriate Server name, such as `wxsServer`, or accept the default. Click **Finish**

You now have a newly configured Liberty Profile server. If you have not used the Liberty profile before, you might want to briefly investigate the new server such as the minimal configuration it needs to run, in the `server.xml` file. Try to start and stop it to see how quickly it can do so. To start the Liberty Profile server, click the **Start** icon on the right side of the window as shown in Figure 4-6.

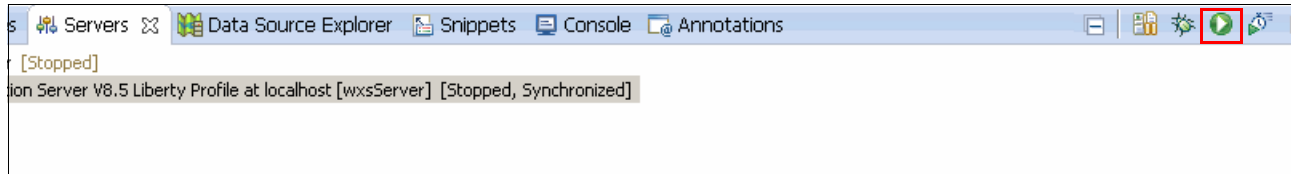


Figure 4-6 Starting the Liberty Profile

Configuring the eXtreme Scale feature in the Liberty Profile server

This section describes how to configure the Liberty Profile server to run the eXtreme Scale feature installed. This process is also documented in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txslibertytoolconfig.html>

Complete the following steps:

1. Click **File** → **New** → **Other**.
2. Select **WebSphere eXtreme Scale** → **Container server configuration file**.
3. Click **Browse** and select the Liberty Profile installation location. For the example this is `c:\wlp`. You should see your Liberty Profile server instance in the server drop-down dialog box.
4. Go through the detailed wizard of eXtreme Scale options. There are many options available. At this stage, keep the eXtreme Scale configuration simple and run a catalog server and the grid container in the same Liberty Profile instance. To accomplish this, enable the catalog server feature to run the catalog server. The container is not created by doing but rather when the `objectgrid.xml` file is detected, which is introduced later.

Click **Next** and select **Include Catalog Server**.

5. Optionally, review the remaining configuration options in the wizard. These become more important as you become familiar with eXtreme Scale. Click **Finish**.
6. From the Enterprise Explorer, click **WebSphere Application Server v8.5 Liberty Profile** → **Servers** → **wxsServer** and open the file `xsServerConfig.xml`. If it is not selected, click the **Design** tab on the editor.
7. Click **Server Configuration** → **IBM WebSphere eXtreme Scale Server Configuration**.
8. There are many options. On the right side, scroll down and change the content of Transport for server to server configuration to **XIO**.

eXtremeIO (XIO): A new transport was introduced in WebSphere eXtreme Scale version 8.6, called eXtremeIO (XIO). The XIO transport removes the dependency on ORB. Further discussion on XIO can be found in 2.1.5, “Transport protocol: IBM eXtremeIO (XIO)” on page 34.

9. Save and close the configuration file.

The new eXtreme Scale configuration needs to be included into the Liberty Profile server configuration. Complete the following steps:

1. Open the Servers view in Eclipse to see your Liberty Profile instance. The default name for it is *WebSphere Application Server v8.5 Liberty Profile at localhost*.
2. Expand the Liberty Profile instance and double-click the server configuration. As you can see, the basic Liberty configuration is small. Configuration is done “by exception” when the configuration deviates from the defaults.
3. To add the eXtreme Scale configuration to the Liberty Profile instance, click **Add**.
4. From the list of Liberty configuration options, select **Include**.
5. Click **Browse** and select the new eXtreme Scale configuration file, which is called `xsServerConfig.xml`.
6. Click **OK** and save the `server.xml` file.
7. Optionally, review the `xsServerConfig.xml` file to see what configuration options are configured. You can access this directly from the `server.xml` file by clicking the **Location** link. There are many configuration options available.

Tip: This configuration can be added to the main `server.xml` file if wanted. However, having the configuration in a separate file can make it reusable elsewhere.

The environment is now ready to create a grid configuration.

4.1.2 Creating a WebSphere eXtreme Scale grid

The last step to set up the development environment is to provide a simple grid configuration.

You need two configuration files, `objectgrid.xml` and `objectGridDeployment.xml`, to perform your grid definition and configuration with eXtreme Scale. These files are the core of the grid configuration, and are required whether you are deploying to the Liberty Profile, Java SE, or WebSphere Application Server:

- ▶ `objectgrid.xml`: This file defines the grid and the maps that are contained in the grid.

Important: This file name is case-sensitive when used with the Liberty Profile. Where the filename `objectGrid.xml` is referred to, it is typically case-sensitive and expected in the form shown. The exception is when the configuration file is used with the Liberty Profile. In this case, the file must be lowercase, `objectgrid.xml`. The convention that is used in this chapter is to use `objectGrid.xml`, except where specifically referring to examples used with the Liberty Profile.

More configuration that defines the behavior of the grid is also in the `objectgrid.xml` file, such as evictors that control when data is removed from the grid and indexes that enable fast queries. See Example 4-1.

Example 4-1 objectgrid.xml file

```
<objectGridConfig xmlns:xsi=".....">
  <objectGrids>
    <objectGrid name="Grid" txTimeout="30">

      <backingMap name="Map1" copyMode="COPY_TO_BYTES"
        lockStrategy="PESSIMISTIC"
        nullValuesSupported="false" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

```

    <backingMap name="Map2" copyMode="COPY_TO_BYTES"
      lockStrategy="PESSIMISTIC"
      nullValuesSupported="false" />
  </objectGrid>
</objectGrids>
</objectGridConfig>

```

The following are the key `objectgrid.xml` values:

objectGrid name	The name of the grid. This must be the same name as in the <code>deployment.xml</code> file. It is a collection of maps.
backingMap	Defines a map within the grid with a specific set of behaviors. This name must be the same as in the <code>deployment.xml</code> file.
txTimeout	Define the transaction time-out.
copyMode	The <code>COPY_TO_BYTES</code> value that stores values in the map in a serialized form, such as eXtreme Data Format (XDF), which is more efficient for storage.

- ▶ `objectGridDeployment.xml`: This file defines the non-functional characteristics of the grid. For example, the data availability is configured through the number of replicas. The scalability limits are configured through the number of partitions. See Example 4-2.

Example 4-2 objectGridDeployment.xml file

```

<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi= .....>
  <objectgridDeployment objectgridName="Grid">
    <mapSet name="mapSet" numberOfPartitions="13"
      minSyncReplicas="0"
      maxSyncReplicas="1"
      maxAsyncReplicas="0">
      <map ref="Map1"/>
      <map ref="Map2"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>

```

The following are the key `objectGridDeployment.xml` values:

objectgridName	The name of the grid as defined in the <code>objectgrid.xml</code> file.
mapSet	Defines a collection of maps from the <code>objectgrid.xml</code> file that have the same deployment properties.
numberOfPartitions	Defines the total number of partitions across the containers. In Example 4-2, this is 13. This number is typically a prime number, which helps balance data across the grid.
minSyncReplicas	The minimum number of synchronous replicas. If the grid cannot meet this minimum availability, it cannot come online. This sample uses 0, which allows the grid to come online even if not replicated.
maxSyncReplicas	The maximum number of synchronous replicas the grid can sustain.

maxAsyncReplicas	The maximum number of asynchronous replicas the grid can sustain.
map ref	Lists the backing maps as defined in the objectgrid.xml file.

Creating the grid

You can configure many options in the `objectgrid.xml` and `objectGridDeployment.xml` files. They define all of the features that are described in Chapter 2, “Architecture and topologies” on page 27. However, the eXtreme Scale Developer Tools provides a nice way to provide a default configuration to get you started.

Complete the following steps to create the grid:

1. Before starting the grid configuration wizard, create a folder in the project to contain the files. In the Enterprise Explorer, expand the Liberty Profile project **WebSphere Application Server v8.5 Liberty Profile** and right-click the server name `wxsServer`. Click **New** → **Folder** and call it `grids`.

Important: The grid configuration files must go into a directory with the specific name `grids`. This is detected by eXtreme Scale when it starts and creates your grid instance.

2. Click **File** → **New** → **Other**.
3. Select **WebSphere eXtreme Scale** → **Object Grid Configuration XMLs**.
4. In the Enterprise Explorer, select the Liberty Profile project instance, **WebSphere Application Server v8.5 Liberty Profile**.
5. Browse for the `grids` folder that you created as shown in Figure 4-7.

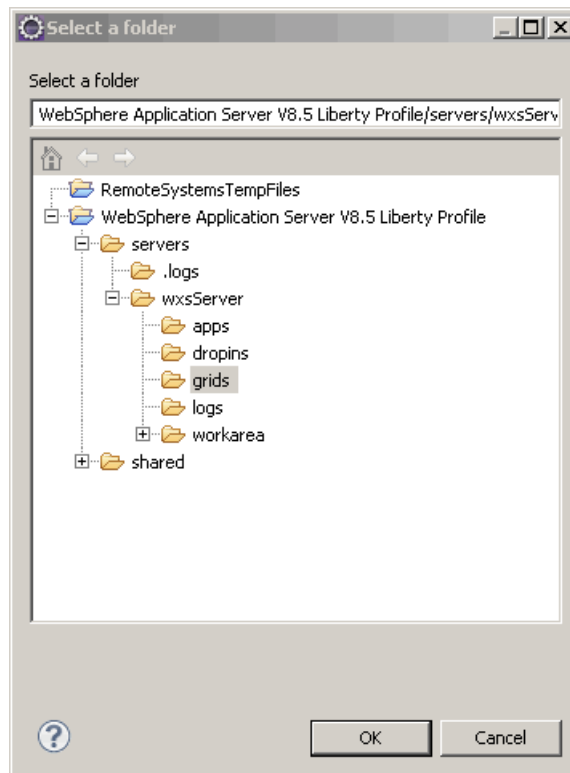


Figure 4-7 Browsing to the `grids` folder

6. A simple grid configuration is sufficient here, so click **Finish**. Consider the following important variables:
 - Grid name.
 - Map name.
 - Copy mode. Set this variable to `COPY_TO_BYTES`. This is a prerequisite for using XDF. For more information about copy modes, see 4.8.1, “Transaction considerations” on page 144.
 - Lock strategy. For more information, see 4.8.1, “Transaction considerations” on page 144.
 - Number of partitions. This variable depends on the size of the grid. For more information, see 2.6, “Scalability sizing considerations” on page 56.
 - Number of replicas. This variable depends on the level of data availability that is needed in the deployment. For more information, see 2.3, “Replication: Primary and replica shards” on page 44.

Important: This wizard creates two files; `objectgrid.xml` and `objectGridDeployment.xml`. The file names are important and case sensitive so eXtreme Scale can detect and use the grid configuration found in them.

Where the file name `objectGrid.xml` is referred to, it is typically case-sensitive and expected in the form shown. The exception is when the configuration file is used with the Liberty Profile. In this case, the file must be lowercase, `objectgrid.xml`. The convention used in this chapter is to use `objectGrid.xml`, except where specifically referring to examples used with the Liberty Profile.

Starting the Liberty Profile

Complete the following steps to start the Liberty Profile:

1. From the Servers view in Eclipse, select the Liberty Profile instance, **WebSphere Application Server v8.5 Liberty Profile at localhost**, and click the **Start** icon.
2. Review the logs to confirm that the eXtreme Scale grid has started. The logs are contained in the `messages.log` file, found in the logs directory of the Liberty Profile project, as shown in Figure 4-7 on page 105.

Something similar to Example 4-3 is displayed with a message for each partition:

Example 4-3 Log file messages confirming that the grid has started successfully

<code>com.ibm.ws.objectgrid.replication.ReplicatedPartition (primary) is open for business.</code>	<code>I CW0BJ1511I: Grid:gridMapSet:2</code>
<code>com.ibm.ws.objectgrid.replication.ReplicatedPartition (primary) is open for business.</code>	<code>I CW0BJ1511I: Grid:gridMapSet:0</code>
<code>com.ibm.ws.objectgrid.replication.ReplicatedPartition (primary) is open for business.</code>	<code>I CW0BJ1511I: Grid:gridMapSet:1</code>

You now have a fully working eXtreme Scale environment running on a Liberty Profile installation.

4.2 Developing for WebSphere eXtreme Scale

This section introduces basic grid programming concepts. It allows you to test the grid with a sample application, then introduces the programming APIs behind the sample application.

4.2.1 Testing the grid with a sample web application

Deploy the sample available from the following website to test your installation and review some simple source code:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/websphere_extreme_scale_getting_started_with_websphere_application_server_sample5

Also, there are more samples, tutorials, and a forum available on the IBM Elastic Caching community:

<http://www.ibm.com/developerworks/connect/caching>

Complete the following steps to import and run the sample application in Eclipse:

1. Click **File** → **Import** → **Existing projects into workspace**.
2. Select the downloaded file, **GettingStartedWeb.zip**.
3. Select the project **GettingStartedWebClientWAR** as shown in Figure 4-8. Click **Finish**.

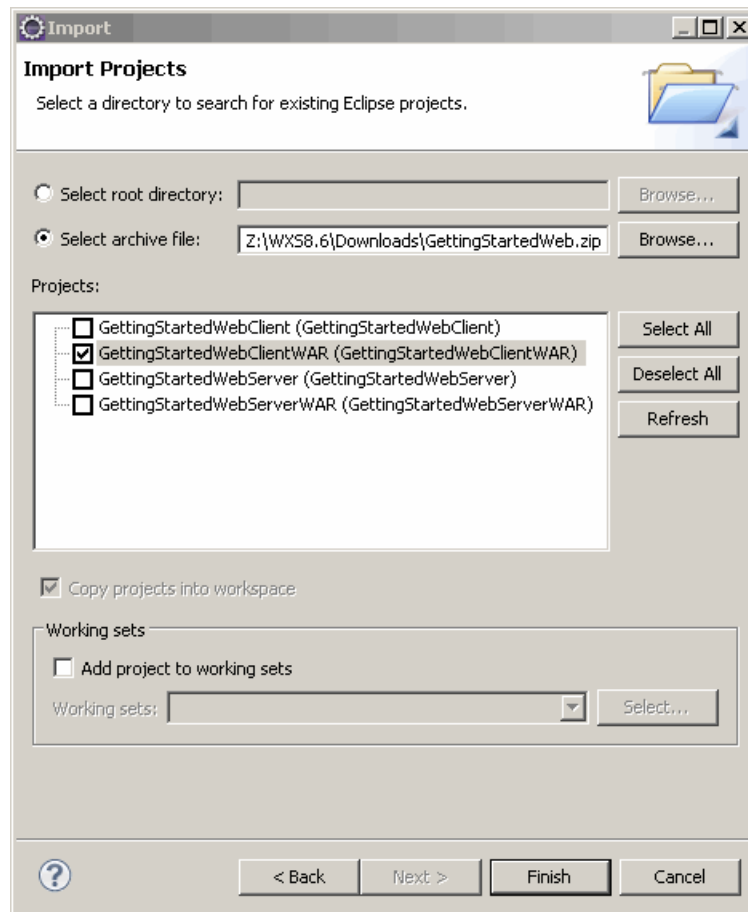


Figure 4-8 Importing WAR file for the sample application

4. At this stage, you might have compilation errors because the project does not have the server libraries on the class path. Right-click the **GettingStartedWebClientWAR** project and select **Build Path** → **Configure Build Path**.
 - a. On the Library tab, click **Add Library**.
 - b. Select **Server Runtime** and click **Next**.
 - c. Select **WebSphere Application Server V8.5 Liberty Profile** and click **Finish**.
 - d. Select the Build Path by clicking **OK**. This should remove the compilation errors.
5. Right-click the GettingStartedWebClientWAR project and select **Run as** → **Run on server**. This should add the project to the Liberty server and start it up.
6. If the web browser does not open with the test client URL, open a browser and go to `http://localhost:9080/GettingStartedWebClientWAR`.
 This opens the test page where you can test the main concepts of interfacing with an eXtreme Scale grid.
7. Connect to the grid, verifying that the Catalog Service Endpoints has **Endpoints** selected and `localhost:2809` as the endpoint. Click **Connect**.
8. Perform data insertions and retrievals by using the CRUD (Create, Retrieve, Update, Delete) operation section as shown in Figure 4-9.

The screenshot shows a web application interface with the following sections:

- Catalog Service Endpoints:** Contains two radio buttons: "Endpoints" (selected) and "Catalog Service Domain ID". Below them is a text input field for "Endpoints" containing "localhost:2809". Underneath are labels for "Remote: <host>:<port>" and "Embedded: <blank>".
- Grid:** Contains two text input fields: "Grid" (containing "Grid") with a note "(For XC10 use Simple Grid Name)" and "Map" (containing "Map1").
- Connection Status:** A bar showing a "Disconnect" button and the text "connected to Grid on localhost:2809".
- CRUD Operation:** Contains two text input fields: "Key" (containing "101") and "Value" (containing "123456789") with a note "(Used for Insert and Update only)". Below these are four buttons: "Insert", "Update", "Delete", and "Get".

Figure 4-9 Getting Started web sample for interacting with the grid

4.2.2 Using the ObjectMap API

This section introduces basic eXtreme Scale development for interacting with data in grids. For an overview of the concepts used in this section and how they interrelate, see 2.5, “A simple example” on page 54. The source code from the sample is available for inspection.

Connecting to the grid

To use a grid, a client application needs a connection to the grid. Example 4-4 shows the APIs required to access the grid.

Example 4-4 Sample code to connect to the grid

```
ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();
ClientClusterContext ccc = ogm.connect(cep,(ClientSecurityContext) null, (URL) null);
ObjectGrid grid = ogm.getObjectGrid(ccc, gridName);
```

To obtain a connection to the grid, two important parameters are needed:

- ▶ Catalog service endpoints (shown in the code sample as the variable *cep*)
This is a simple list of host names and ports of the catalog server cluster. If there is more than one available catalog server, they are each listed, separated by commas, as "hostname1:port1,hostname2:port2,...". In this example, there is only one Liberty profile catalog server configured, so the “list” is a single item: localhost:2809.
- ▶ Grid name as defined in the `objectgrid.xml` file, which is represented in the example by *gridName*.

Extra parameters to the `ogm.connect()` method can be important:

- ▶ `ClientSecurityContext` to provide configuration for a secure connection to the grid.
- ▶ URL to provide an overriding `objectgrid.xml` configuration. In general, the client `objectgrid.xml` configuration mirrors that of the server instance. However, there can be important differences. For example, if a near cache is in use on the client, a different evictor to the grid can be used. This configuration prevents too much data from being cached in the constrained memory of the client environment.

Important: The `ObjectGrid` instance can and should be cached within your application so that the `getObjectGrid` method is called only once.

When deploying grids on WebSphere products, you might see reference to the term *Catalog Service Domain*. This is configured in a WebSphere Application Server environment as a resource for application code to reference. It is administratively configured to point to the actual endpoints of a selection of catalog servers. Code simply has to refer to the name of the Catalog Service Domain instead of hardcoding host and port names

Interacting with the grid

Having obtained a connection to the grid, the application can now interact with maps within the grid. To do so, an instance of the `ObjectMap` must be obtained. Access to an `ObjectMap` is through a transactional context that is provided by the `Session` class.

The following three examples are shown to illustrate the key CRUD `ObjectMap` APIs.

Example 4-5 shows a simple example for inserting data into the grid by using the `insert` method. A similar piece of code can be used to retrieve data using the `get` method. It is worth highlighting a couple of important things here:

- ▶ A `Session` object provides the transactional context for interacting with the grid. A session therefore cannot be shared between multiple threads. Close them after use to return them to the pool.
- ▶ The `insert` method is implicitly transactional (it runs a transaction begin and end). Where more complex interaction with the grid is required, explicitly code the transaction semantics by using `begin` and `commit/rollback`.
- ▶ The `insert` semantically refers to an insert, and fails if the data already exists in the grid (which would therefore be an update).

Example 4-5 Sample code to illustrate data insertion

```
// Get a session
Session session = grid.getSession();

// Get the ObjectMap
ObjectMap map1 = session.getMap(mapName);

// Insert data into the map.
// This is implicitly transactional
map1.insert(key, val);

// Close the session so that it can be managed in a pool
session.close()
```

Example 4-6 shows a slightly more complex example where an update to existing data in the grid is run. This applies both updates and removes. It highlights some useful concepts:

- ▶ Explicit transaction control using `begin` and `commit`.
- ▶ Use of `getForUpdate` when running updates. This obtains an *unshareable* lock, which helps manage deadlock situations where there are concurrent requests to the same data.
- ▶ The update fails if the data does not exist in the grid. You must use the `insert` method to initially put data into the grid as shown in Example 4-5.

Example 4-6 Sample code to illustrate grid updates

```
// Get a session
Session session = grid.getSession();

// Get the ObjectMap
ObjectMap map1 = session.getMap(mapName);

// Explicitly begin the transaction because the update API
// relies on the fact that the get was done in the same transaction
session.begin();

// Use the getForUpdate since we know we will be doing an update
String value = map1.getForUpdate(key);

// Make some change to the value

// Perform update to the grid
map1.update(key, value);
```

```
// End the transaction
session.commit();

// Close the session so that it can be managed in a pool
session.close();
```

WebSphere eXtreme Scale version 8.6 introduced two useful new ObjectMap methods: `upsert` and `lock`. These methods provide a more flexible mechanism to manage data updates and explicit locking of the grid.

Example 4-7 illustrates the new `upsert` API, which runs an `insert` if the data does not exist in the grid, or an update if the data exists in the grid. It highlights simplification in the code:

- ▶ There is no need for the use of `getForUpdate` to test for the presence of the data in the grid to choose whether to insert or update or to obtain the correct lock level.
- ▶ If a single update is run, explicit transaction control is not required.

Example 4-7 Sample code to illustrate new WebSphere eXtreme Scale v8.6 upsert API

```
// Get a session
Session session = grid.getSession();

// Get the ObjectMap
ObjectMap map1 = session.getMap(mapName);

// Use the upsert API to perform update to the grid
map1.upsert(key, value);

// Close the session so that it can be managed in a pool
session.close();
```

For a complete overview of the ObjectMap API, see the *Introduction to ObjectMap* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxsubjectmapinterface.html>

The examples that are described in this section are simple. For information about more advanced use cases and programming techniques, see 4.7, “Saving time with WXSUtils” on page 143.

4.3 Loading data into the grid

This section introduces some options for populating the grid using loaders on the grid and client loaders. In particular, it focuses on some of the ready to use Java Persistence API (JPA) based loaders that eXtreme Scale provides to use JPA to interact with a database. Solutions using loaders are all implementations of the data access scenario found in 3.4, “In-line cache scenario” on page 83.

The following are the ready to use JPA-based features in eXtreme Scale. Their use is illustrated in Figure 4-10.

- ▶ Java Persistence API (JPA) loader
- ▶ Time-based updater
- ▶ Client loader

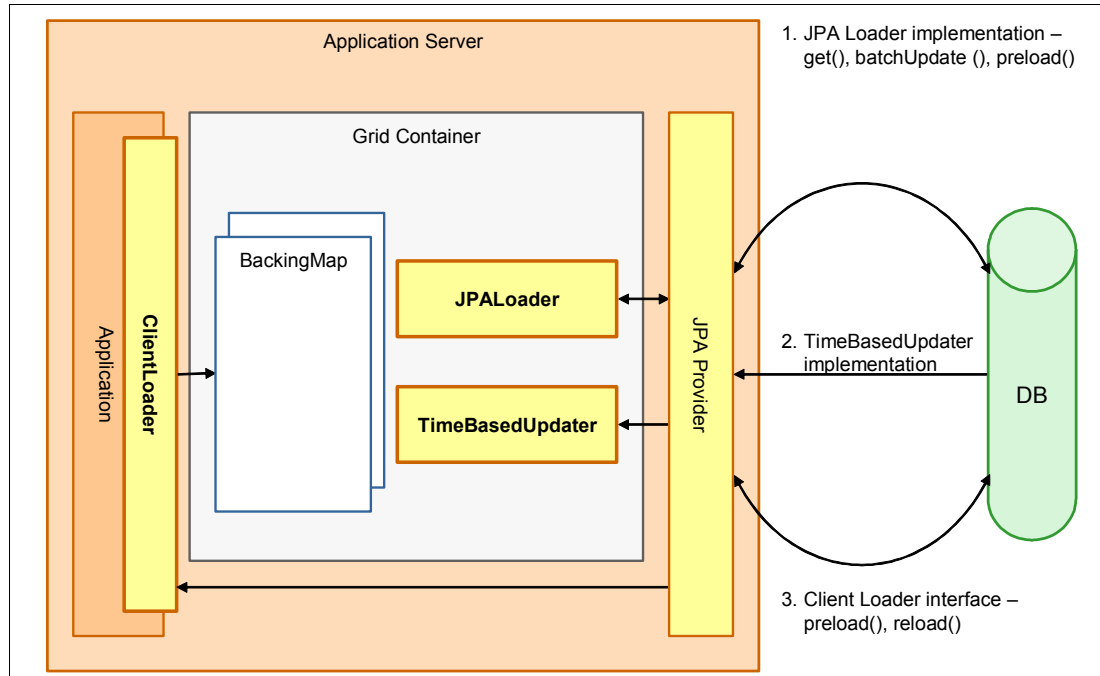


Figure 4-10 WebSphere eXtreme Scale features for using OpenJPA to interact with an RDBMS

4.3.1 Introducing loaders

Before describing the specifics of the JPA loader techniques, it is worth introducing loaders in general.

As described in 3.4, “In-line cache scenario” on page 83, the loader is used in an in-line cache scenario. In this scenario, the cache client always interfaces directly to the cache for data. It never directly accesses the data source. It is the responsibility of the grid to read from the data source to populate the grid, and write to the data source when the grid is updated. It is the loader that provides this interface between the grid and the data source.

When the client application requests data from the grid for a key, if the data is in the grid (cache hit), it is returned. If the data is not in the grid (cache miss), the loader accesses the data store for the data and populate the grid based on that. If the key is not in the data store, the client receives a `KeyNotFoundException`.

Queries: Loaders are not triggered by queries. Any data that needs to be available for queries needs to already be in the grid.

A loader is an implementation of the simple interface that is shown in Example 4-8.

Example 4-8 Loader interface

```
public interface Loader{
    /**
```



```

    * Returns the list of values corresponding to the set of keys passed in.
    */
    List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException;
    /**
    * Called to tell the Loader to write the provided changes to the backend.
    */
    void batchUpdate(TxID txid, LogSequence sequence) throws
        LoaderException, OptimisticCollisionException;
    /**
    * Signals the Loader to preload the data into the map.
    */
    void preloadMap(Session session, BackingMap backingMap) throws
        LoaderException;
}

```

A loader is configured for use in the grid by editing the `objectgrid.xml` definition file to include the code shown in Example 4-9.

Example 4-9 Loader definition in the `objectgrid.xml` file

```

<?xml version="1.0" encoding="UTF-8" ?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid">
        <backingMap name="map1" pluginCollectionRef="map1"
            lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPluginCollection id="map1">
        <bean id="Loader" className="com.myapplication.MyLoader">
            <property name="dataBaseName" type="java.lang.String"
                value="testdb" description="database name" />
            <property name="isolationLevel" type="java.lang.String"
                value="read committed" description="iso level" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

The benefit of the generic loader interface is that it can be flexible. It is typically used with a database, but it is not constrained to that. For example, it can be used to read and write data by using a web service.

The following sections describe loader implementations that are provided by eXtreme Scale to reduce the coding that is required:

- ▶ 4.3.2, “Implementing the JPA loader” on page 114
- ▶ 4.3.3, “Implementing a time-based updater” on page 117
- ▶ 4.3.4, “Implementing a client loader” on page 120

4.3.2 Implementing the JPA loader

The JPA loader provides a means to write to and read from a database from the eXtreme Scale grid using JPA entities. It is used to manage the link between the data grid and the back-end database. When a request for data arrives at the grid and cannot be satisfied, the loader is responsible for contacting the back-end data store and fetching the data. This new piece of data is first inserted into the data grid, and then returned to the client that requested the data. A loader also updates the back-end data store to reflect changes to data made in the grid.

The JPA loader function can be used in either a write-through or a write-behind capacity. Write-through is the default behavior, and ensures that every update to the grid is immediately persisted to the database. Write-behind batches updates for either an amount of time or a number of updates before they are persisted to the database.

The JPA loader also provides a preload implementation, which is run within a designated partition on the grid server. However, the use of a container-side preloader is generally not recommended. The preferred method to pre-populate data into the grid is to use a client loader implementation. For more information, see 4.3.4, "Implementing a client loader" on page 120.

To implement a JPA loader, provide an implementation of the following methods:

- ▶ `get()`: Access the data for a specific list of keys
- ▶ `batchUpdate()`: Persist changes to the database
- ▶ `preloadMap()`: Pre-populate the grid

At a high level, use these steps to implement the JPA loader:

- ▶ WebSphere eXtreme Scale configuration:
 - Configure the grid to use the JPALoader plug-in in the `objectgrid.xml` file
- ▶ JPA configuration and development:
 - JPA provider libraries/run time
 - JPA Entity definitions
 - JPA persistence unit for database connectivity

Example 4-10 highlights the eXtreme Scale configuration that is required to implement the JPA loader.

Example 4-10 Sample `objectGrid.xml` file configured to use the JPA loader

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="BranchGrid" txTimeout="60">
1)      <bean id="TransactionCallback"
className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"
value="myPersistenceUnit"/>
      </bean>
      <backingMap name="Owner" pluginCollectionRef="Owner" />
    </objectGrid>
  </objectGrids>
```

```

<backingMapPluginCollections>
2)   <backingMapPluginCollection id="Owner" >
      <bean id="Loader" className="com.ibm.websphere.objectgrid.jpa.JPALoader">
        <property name="entityClassName" type="java.lang.String"
value="com.ibm.itso.redbook.Owner"/>
      </bean>
    </backingMapPluginCollection>

  </backingMapPluginCollections>
</objectGridConfig>

```

The two plug-ins that are configured in Example 4-10 on page 114 refer to the following items:

- ▶ The JPA persistence unit, which describes how to connect to the database and provides access to the entity metadata. A persistence configuration file called `persistence.xml` is included in the JPA application. The string value that is mentioned in the `JPATXCallBack` bean definition in the example points to a persistence unit name of `myPersistenceUnit`, which exists in the `persistence.xml` file.
- ▶ A JPA loader is configured for each backing map. A JPA loader is configured by configuring a *backingMapPluginCollection* per backingMap, where the ID is that of the *plug-inCollectionRef* of the backingMap.

The example JPA loader configuration has a single property configured called `entityClassName`. The `entityClassName` property describes which JPA entity is going to be used to manage the database persistence for the backingMap.

With the configuration sample in Example 4-10 on page 114 in place, the JPA loader can now handle the cache miss.

Using write-behind with the JPALoader

By default, all loaders, including JPA loaders, immediately run an update to the database when a change is made to the grid. This is called the *write-through* scenario. Alternatively, a second scenario is available by which eXtreme Scale acts as a buffer for the updates. The updates are more efficiently batched into fewer, larger updates. This scenario can greatly improve the scalability of a database. It is referred to as the *write-behind* scenario.

The write-behind scenario might be preferable for several reasons:

- ▶ Performance: Each application transaction to the grid is quicker because the transaction does not have to wait for the data to be persisted to the back end.
- ▶ Isolation of back-end failures: If the back-end database fails, it normally affects the availability of an application. However, in this case, the application can keep running. The write-behind function has a built-in retry mechanism, which can persist the data when the back-end comes back online.
- ▶ Reduce load on back-end: Instead of multiple small database accesses, eXtreme Scale acts as a buffer for the updates. This efficiently batches changes into fewer, larger updates. This process can greatly improve the scalability of a database.

An illustration of the write-behind and write-through concepts can be seen in Figure 4-11.

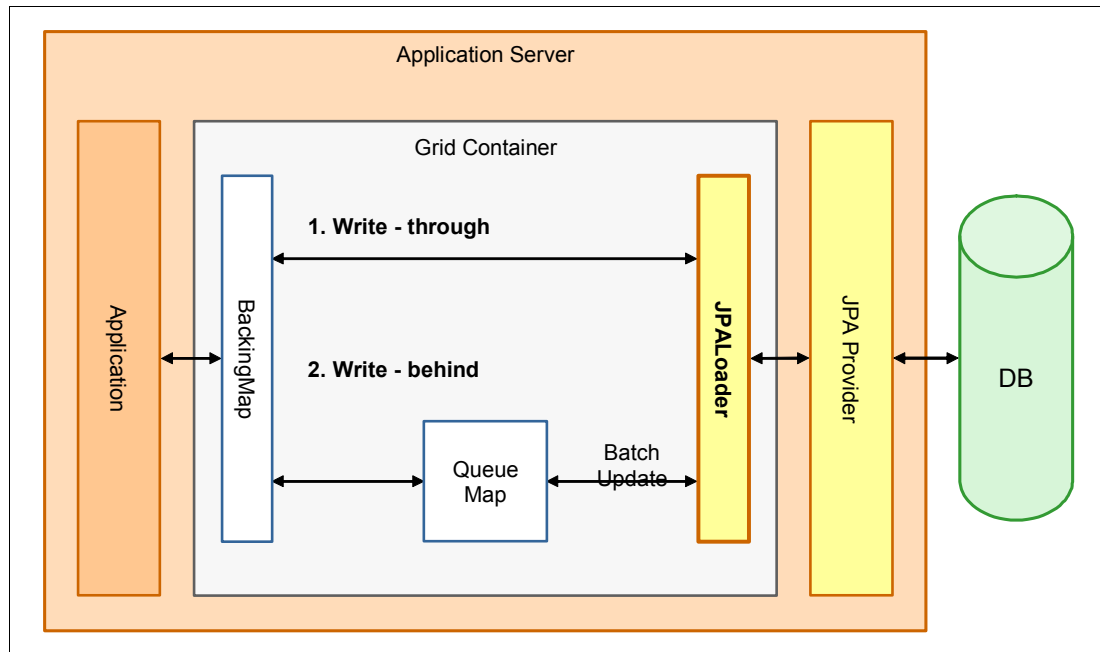


Figure 4-11 Write-through versus write-behind

The write-behind function is enabled for a specific backing map and applies to all loaders. This example involves the JPA loader.

Example 4-11 shows a portion of the `objectGrid.xml` definition file that configures the write-behind on the Owner backing map. In this case, the write-behind is configured to make updates based on two criteria:

- ▶ **Time-based update:** Runs an update to the database if 60 seconds have elapsed since the last database update. This setting can be configured to the required number of seconds between updates. If no value is provided, the default is 300 seconds.
- ▶ **Count-based update:** Perform an update if a count of 100 updates to the grid have occurred since the last database update. If no value is provided, the default count is 1000.

If both of these criteria are configured, the first criteria to be matched, time or count, triggers the update.

Example 4-11 Portion of `objectGrid.xml` file that enables write-behind

```
<backingMap name="Owner" writeBehind="T60;C100" pluginCollectionRef="Owner" />
```

If you enable trace for the object grid by using `ObjectGrid=all`, observe the output shown in Example 4-12 where it shows the write-behind thread counting elapsed time and running updates with any outstanding changes.

Example 4-12 Output from trace log file that shows information from write-behind

```
[15/09/08 13:08:42:453 BST] 0000006f WriteBehindLo 1 WBLoader - 59907ms passed since the last update. Will update the database.
[15/09/08 13:08:42:468 BST] 0000006f WriteBehindLo 1 WBLoader - pushed 2 changes to real loader.
```

Handling write-behind failure

Although there are some strong benefits to the write-behind function, there are also some additional considerations. When there is a failure writing to the back end, the application transaction has already been committed. A good example is adding an entry to the backing map where the entry exists in the database. The backing map transaction successfully commits, but when the write-behind tries to write it to the database, it fails with a duplicate key exception. Other, similar failure scenarios that must be handled as well.

A special map stores these failed write-behind updates. This map serves as an event queue for failed updates, which can be retrieved and handled. The name of the map that stores these failures uses the following naming convention:

IBM_WB_FAILED_UPDATES_<map name>

Example 4-13 shows sample code, with a map called `Owner`, for accessing the failed write-behind update map, and the failed keys and values.

Example 4-13 Sample code for handling write-behind failures

```
session.begin();
ObjectMap failedMap = session.getMap(
WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Owner");
Object key = null;

while(key = failedMap.getNextKey(ObjectMap.QUEUE_TIMEOUT_NONE)) {
LogElement element = (LogElement) fMap.get(key);
Object failedKey = element.getCacheEntry().getKey();
Object failedValue = element.getAfterImage();
fMap.remove(key);
//Handle the failure key and value.
}
session.commit();
```

4.3.3 Implementing a time-based updater

A means to refresh the contents of the grid is essential if the grid does not have exclusive access to the database. If another application can update data on the database, a time-based updater can check for these updates and notify the grid. The grid can then invalidate or refresh the stale data.

The previous section (4.3.2, "Implementing the JPA loader" on page 114), configured the sample application to use a JPA loader. You can now use the JPA loader to do another useful task. WebSphere eXtreme Scale provides a time-based updater to periodically check a database for updates. If another application makes an update to the data, the updater can detect this update and do one of three things:

- ▶ Invalidate the stale data, which is then removed.
- ▶ Update the stale data with fresh data.
- ▶ Add any recent insertions.

Basically, a time-based updater loader checks a field in the database to determine whether changes have been made. Modern commercial databases provide a function to update a time stamp field whenever that row is changed. When you have a list of time stamps, a couple of simple queries can establish what has changed since you last looked. This is an important function to help manage stale caches. For more information about options for dealing with stale cache information, see 4.6, "Dealing with data eviction and stale data" on page 137.

The time-based updater uses a heuristic to determine how often to poll the database table. This ensures that you are not polling too often, which can be resource intensive on a large database table.

The query in Example 4-14 supplies all rows that have changed in the period since the last time the time stamp was checked.

Example 4-14 Query to identify rows changes in a database

```
SELECT o FROM Owner o where o.rowChgTs>?
```

After loading the changed rows, the query in Example 4-15 can get the time stamp of the last change on the database.

Example 4-15 Query to identify last change in the database

```
SELECT MAX(o.rowChgTs) FROM Owner o
```

This process is illustrated in Figure 4-12.

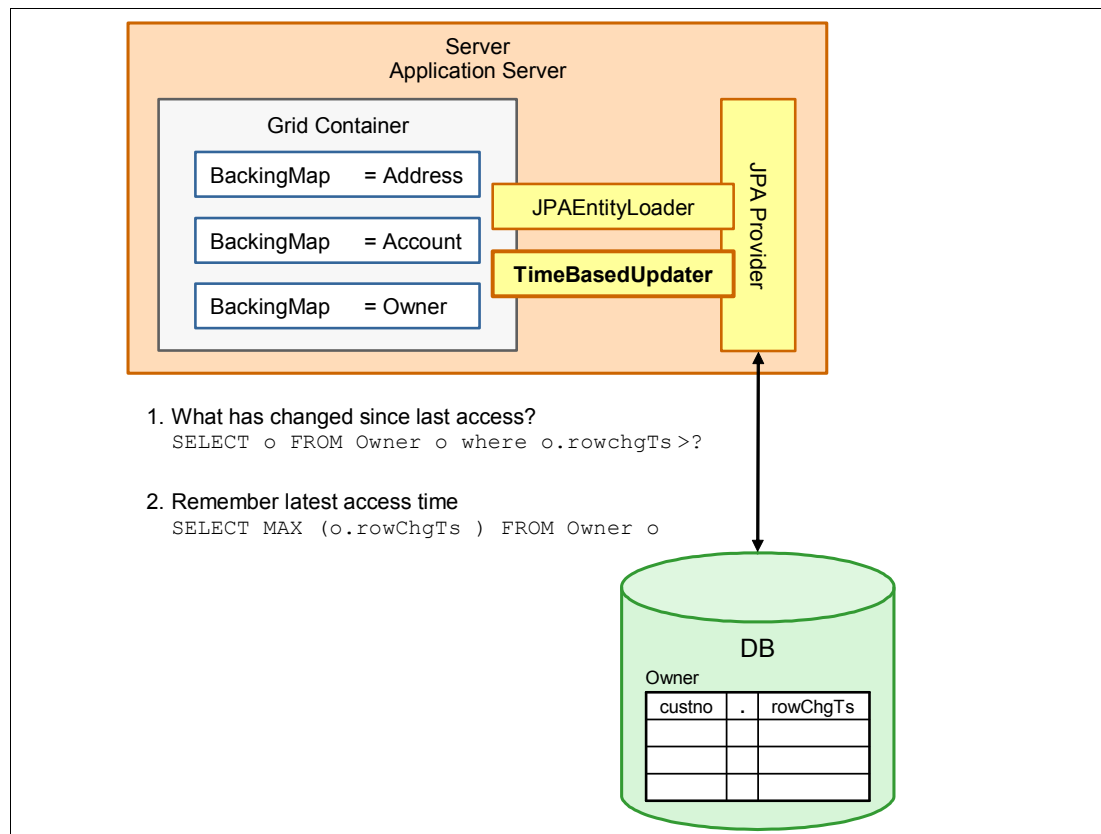


Figure 4-12 Time-based updater usage

After the time-based updater detects that data has changed, there are these possible courses of action:

- ▶ **Invalidate stale data:** This is the quickest of the options. It is ideal if you have a lot of stale data that is expensive to load at once, and if you do not mind loading the data again when you need it.

- ▶ Update stale data: The updater loads the new data into the grid, which is good if you do not want to take the hit of individual requests reloading data.
- ▶ Checking the database for any additional inserts: This extends the previous option and inserts any new rows into the grid.

The options that are provided by the time-based updater allow for detection of changes to data and inserts, but not for the deletion of data because this does not show in the time stamp query. An alternative is to provide a deleted field in the database, where applications mark rows as deleted instead of removing them. This is detected by the time-based updater, which can run the appropriate invalidations in the grid, and then explicitly delete the marked rows from the database.

Before configuring the time-based updater in eXtreme Scale, the database and the JPA entities must be changed. For example, the database might add a time stamp field to pertinent tables. The appropriate JPA entities are changed to include the new time stamp fields.

Having set up the JPA loader in 4.3.1, "Introducing loaders" on page 112, update that configuration with the time-based updater. Example 4-16 shows how this is added to the configuration of the Owner backing map.

Example 4-16 Time-based updaters sample configuration

```

<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="BranchGrid" txTimeout="60">
1)
      <bean id="TransactionCallback"
className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"
value="myPersistenceUnit"/>
      </bean>
      <backingMap name="Owner" pluginCollectionRef="Owner" >
        <timeBasedDBUpdate timestampField="rowChgTs" persistenceUnitName="myPersistenceUnit"
entityClass="com.ibm.itso.redbook.Owner" mode="UPDATE_ONLY" />
      </backingMap>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
2)
    <backingMapPluginCollection id="Owner" >
      <bean id="Loader" className="com.ibm.websphere.objectgrid.jpa.JPALoader">
        <property name="entityClassName" type="java.lang.String"
value="com.ibm.itso.redbook.Owner"/>
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>

```

Provide the configuration for each backing map that is to be kept up to date. It specifies the following items:

- ▶ `entityClass`: This is the class name of the entity to keep up to date,
- ▶ `timestampField`: This maps to the time stamp instance variable that was added to the JPA entity.
- ▶ `persistenceUnitName`: The JPA persistence unit that already defined.
- ▶ `mode`: What type of update to run on the grid. It can be one of these values:
 - `INVALIDATE_ONLY`: This removes the stale entries from the grid. However, the entities must be loaded if needed again.
 - `UPDATE_ONLY`: This updates the grid with any entities that have changed.
 - `INSERT_UPDATE`: As well as updating any stale entities in the grid, this populates the grid with any new entities that have been added to the database since.

4.3.4 Implementing a client loader

The ability to preload data is provided with the loader interface, specifically the JPA loader implementation mentioned in 4.3.3, “Implementing a time-based updater” on page 117. However, the JPA-specific implementations are restricted to preloading an entire database table for a corresponding JPA entity. It might be that your application does not want to, or for reasons of size, cannot load an entire table.

The client loader implementation provides a flexible means of bulk-loading, or preloading, data to populate the grid at start, or in the case that much of the data in the grid needs refreshing, from the database. Instead of having to write your own code for the preload, as is the case with the generic loader interface, the client loader provides an implementation that can take a specific JPA query or load the whole table without further coding. The loading can be run on demand within the client or as part of a grid preload in a custom loader implementation.

An example of the client loader scenario is shown in Figure 4-13.

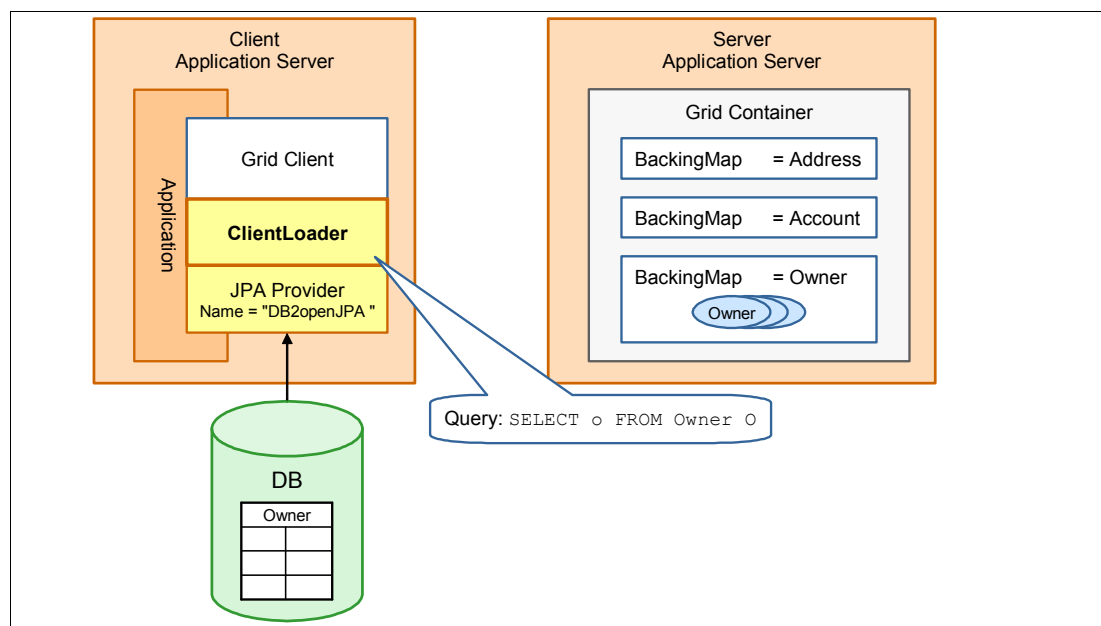


Figure 4-13 Using a client loader to populate the grid

JPA loader or client loader for preloading: The preload function provided by the JPA loaders and the client loader is similar, with the obvious difference being that you can provide a custom query to the client loader and the location of the code execution.

The use cases for these two tools vary because the JPA loader is behind the grid and server-side as opposed to the client loader, which is in front of the grid and client-side.

The JPA loader preload is important when you must pre-populate the grid without using a client application. As the implementations do not allow a custom query to the database, you can create a custom loader and implement the `preload()` method to use the client loader to populate the grid.

The client loader can provide more benefit because you can run more than one and populate the grid in parallel, general reducing overall loading time. Generally, the client loader implementation is preferable to a server-side loader and is the most widely used scenario.

The code that is required to implement a basic Client Loader implementation is fairly simple as shown in Example 4-17.

Example 4-17 Sample client loader implementation

```
StateManager stateMan = StateManagerFactory.getStateManager();
stateMan.setObjectGridState(AvailabilityState.PRELOAD, grid);
ClientLoader c = ClientLoaderFactory.getClientLoader();
//get the desired database data and conver to Java object form.
c.load(grid, "Owner", "myPersistenceUnit", null ,Owner.class, null, null, preload,
null);
stateMan.setObjectGridState(AvailabilityState.ONLINE, grid);
```

Some leading practices are demonstrated in Example 4-17. Before using the `ClientLoader`, the grid is put into `PRELOAD` state. This takes the grid offline. Therefore, check the status of the grid before using it, but if not, the status check ensures that the request fails quickly. The grid is then populated and brought back into the `ONLINE` state.

The following options are passed to the `load()` method in Example 4-17:

- ▶ `grid`: The `ObjectGrid` to populate.
- ▶ `Owner`: The map to populate.
- ▶ `myPersistenceUnit`: The name of the persistence unit to use from the `persistence.xml` file.
- ▶ `null`: This can be a `persistenceProps` object, where you can supplement the `persistence.xml` file with more properties.
- ▶ `Owner.class`: The class name of the JPA entity THAT IS being placed into the grid. It is optional as, by default, the classname is taken from the entity metadata file for the eXtreme Scale grid. It is not necessary in this scenario, as this example is using the same POJO as both the eXtreme Scale and the JPA entity.
- ▶ `null`: This is the optional load SQL string to specify the data that you want to preload. In this example it is null, so the query equates to "SELECT o FROM Owner o". An example in this scenario might be something such as the following:

```
SELECT o FROM Owner o WHERE
o.customerNumber >= :lowNum AND o.customerNumber <= :highNum
```

- ▶ `null`: This is the query parameter map, which is not needed in the example application. However, if you provide a query as shown in the previous point, you need two values for the `lowNum` and `highNum` parameters.
- ▶ `preload`: This is a boolean value to define whether the load is going to provide a preload or not. If it is true, the preload first clears the grid before it loads the data.
- ▶ `null`: If required, this can be a `ClientLoaderCallback` implementation, which provides `preStart()` and `postFinish()` call back methods.

4.4 Querying data using Object Grid Query Language

WebSphere eXtreme Scale offers functions to run SELECT type queries using Object Grid Query Language (OGQL). OGQL is based on Java Persistence Query Language (JPQL). Like JPQL, objects and attributes are used instead of tables and columns. When the `ObjectMap` API is used, the query schema must be defined programmatically or through XML.

The query engine of eXtreme Scale has the following capabilities:

- ▶ Single and multi-valued results
- ▶ Aggregate functions
- ▶ Sorting and grouping
- ▶ Joins
- ▶ Conditional expressions with sub queries
- ▶ Named and positional parameters
- ▶ Index utilization
- ▶ Path expression syntax for object navigation
- ▶ Pagination

To illustrate the information described in this section, a sample application is provided. Details about how to obtain this sample can be found in Appendix A, “Additional material” on page 341.

To introduce the query language, a simple example is shown in Example 4-18. It shows a query that retrieves a list of inventory objects that have prices above a certain limit. To run the query, the following artifacts that must be developed:

- ▶ Application code that includes the OGQL query
- ▶ Configuration defined in the `objectgrid.xml` file to include the map schema and index
- ▶ Deploying the data Java classes to the eXtreme Scale container

Example 4-18 Query retrieving a list of inventory objects with quantity of a specific limit

```
SELECT i FROM Map1 i WHERE i.quantity =?1
```

Example 4-19 shows a sample code snippet of how the query APIs can be used to run the sample query. A query can only run against one partition. The code that is shown in Example 4-19 must be run against *every* partition.

Example 4-19 Sample code using the query API to run a query

```
// since query works on multiple objects, need to start a new transaction
Session sess = grid.getSession();
sess.begin();

// build query and set query parameters
ObjectQuery q = sess.createObjectQuery(queryString);
```

```

q.setParameter(1, quantity );

// client side query can only run on 1 partition.
q.setPartition(partitionNumber);

// process results
Iterator resIter =q.getResultIterator();
while(resIter.hasNext()) {
    InventoryBean bean = (InventoryBean)resIter.next();
    list.add(bean);
}
// just a read operation, so rollback
sess.rollback();

```

To use a query with the ObjectMap API, two configuration elements are needed in the objectgrid.xml file. These are illustrated in bold in Example 4-20.

- ▶ Map schema: The Map schema allows eXtreme Scale to understand the structure of the Java class in the grid and its primary key attribute.
- ▶ Index on the attribute to be queried: To improve performance, an index can be defined on the attribute to be queried. In this example, it is the *quantity* attribute. For more information about indexes and the use of the global index, see 4.4.2, “Optimizing query performance using the global index” on page 127.

Example 4-20 Sample objectgrid.xml file

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://ibm.com/ws/objectgrid/config">

<objectGrids>
  <objectGrid name="Grid">
    <backingMap name="Map1" copyMode="COPY_TO_BYTES"
      pluginCollectionRef="GridPlugins"/>
    <querySchema>
      <mapSchemas>
        <mapSchema
          valueClass="com.ibm.itso.sample.InventoryBean"
          mapName="Map1"
          primaryKeyField="sku"/>
        </mapSchema>
      </mapSchemas>
    </querySchema>
  </objectGrid>
</objectGrids>

<backingMapPluginCollections>
  <backingMapPluginCollection id="GridPlugins">
    <bean id="MapIndexPlugin"
      className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
      <property name="Name" type="java.lang.String" value="QUANTITY" />
      <property name="RangeIndex" type="boolean" value="true" />
      <property name="AttributeName" type="java.lang.String" value="qty" />
    </bean>
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Provide the Liberty Profile with a copy of the Java class of the data to be stored in the grid. This is so the grid can inflate the Java objects for querying. Ultimately, the XDF will negate the need for this in the future. There are other reasons for providing Java classes for use on the grid, for example for eXtreme Scale plug-ins and agents. However, this should only be done if necessary because it adds complexity, and scenarios that require code on the grid prevent you from using the WebSphere DataPower XC10 Appliance.

When you provide eXtreme Scale classes for use with the Liberty Profile, they must be placed in an OSGi bundle and dropped into the grids directory.

OSGi is a standard that provides services to enforce modularity, versions, and a dynamic component model in a Java platform. Both Eclipse and the Liberty Profile are based on it. For further information, see *WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076.

The following steps provide a simple way to provide the OSGi bundle to deploy the required classes onto the Liberty Profile grid class path:

1. Create a MANIFEST.MF file in the source META-INF directory that contains information similar to that found in Example 4-21.

The important information is the *Export-Package*, which defines which package is exposed to the Liberty Profile container. Any other package in the bundle is not visible to code outside the bundle. This is the benefit of OSGi.

Example 4-21 Sample MANIFEST.MF file

```

Manifest-Version: 1.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-Name: QueryInventory
Bundle-SymbolicName: QueryInventory
Export-Package: com.ibm.itso.sample
Bundle-Version: 1.0.0
Bundle-ManifestVersion: 2
  
```

2. Export the classes and MANIFEST.MF as a JAR file. The example here has just one class and the MANIFEST.MF, so the file can be simple. Eclipse provides the option to save the JAR description to make it easy to regenerate it when necessary.

Place the JAR file into the eXtreme Scale grids directory with the grid configuration as described in 4.1.2, “Creating a WebSphere eXtreme Scale grid” on page 103. The contents of the grids directory look similar to Figure 4-14.

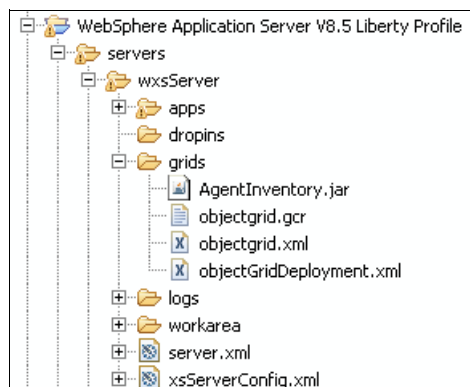


Figure 4-14 Deployment artifacts in the grids directory

4.4.1 Important considerations when using queries

Although the grid query language resembles SQL, the underlying concepts are quite different because you are running against a NoSQL platform.

Important: The design of the data schema of the grid is important to ensure high performance. In a NoSQL deployment, various design guidelines that apply in other disciplines such as with relational databases should not be followed.

Design the data schema such that the application can access it as optimally as possible. This might introduce data duplication and de-normalization and other practices that are discouraged in the relational database world.

These principles are described in the *Single-partition and cross-data-grid transactions* topic on the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxscrossgrid.html>

This section covers the following considerations when you use queries:

- ▶ “Handling partitions” on page 125
- ▶ “Using joins” on page 126
- ▶ “Limiting query results” on page 126
- ▶ “Using indexes” on page 126
- ▶ “Understanding query performance” on page 127

Handling partitions

A query is run against a single partition. The partition can be explicitly set by using `Query.setPartition()`. Otherwise, the query is run against the partition the session is pinned to by previous operations.

When the data is spread over multiple partitions, the query does not work as it normally does when querying a database. To run a query over the whole grid, consider the following options:

- ▶ Iterate over each partition: This is easiest to code as the code iterates across each partition and runs the query. However, it is the slowest and least scalable.
- ▶ Use agents to run the query in parallel: Using agents (such as `MapGridAgent` and `ReduceGridAgent`) to run queries can be fast because each agent is run in parallel across all partitions. Although it is more complex to code, this is a powerful use case for agents. For more information, see 4.5, “Server-side development with agents” on page 132.

Aggregation functions: When you use queries with agents, remember that aggregation functions, such as `count()`, work only on the local partition. The total result must be computed by applying the function to the results from the agent.

- ▶ Use a global index: The global index is a new feature in WebSphere eXtreme Scale version 8.6. The global index is an API that helps maintain an index across the grid of the wanted attributes. It returns a list of partitions that contain the objects of interest so that the query can be targeted to just the partitions that contain the data. For more information, see 4.4.2, “Optimizing query performance using the global index” on page 127.
- ▶ Use multi-partition transaction support: When using the multi-partition transaction support (as described in 4.8.3, “Multi-partition transactions” on page 148), queries can span multiple partitions. This makes development simpler, but it does come with the performance penalty of using two-phase commit transactions. The *Data access and*

transactions topic in the WebSphere eXtreme Scale Version 8.6 Information Center provides further information about this feature:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsdataacc.html>

Which approach is best depends on your requirements. However, the global index feature is powerful and makes a good place to start.

To illustrate the information described in this section, a sample application is provided. Information about how to obtain this sample can be found in Appendix A, “Additional material” on page 341.

Using joins

Relational databases often use a JOIN statement to link associated child entities. WebSphere eXtreme Scale also provides a JOIN, but this powerful construct must be used wisely:

- ▶ Avoid cartesian products in the FROM clause: Use JOIN instead.

Joining of entities occurs in memory. Be careful when you are using cartesian products (FROM A, B), because the resulting temporary data can easily become very large, potentially causing out of memory issues. Use JOINS to reduce the size and reduce the query execution time dramatically (FROM A JOIN B). For example, include the JOIN criteria in the FROM clause, as shown in Example 4-22.

Using different syntax usually does not make a difference for a SQL database, as the query optimizer creates the same execution plan. But the syntax you use can make a huge difference for the grid when cartesian products are involved.

Example 4-22 Expressing a JOIN statement in SQL in the FROM clause

```
SELECT * FROM A JOIN B ON (b.fk = a.pk)
```

- ▶ Remember that a query that includes JOINS is run against a single partition. If the joined child data is in a different partition, the child is not included in the result.

If a JOIN must retrieve all child objects, ensure that the children are placed in the same partition as the parent. This task can be performed by placing the data into a single map set that shares a partitioning key.

Limiting query results

In SQL, you often limit the result set size to avoid being overloaded with a huge result set. In eXtreme Scale, this can be accomplished by using `setFirstResult()` and `setMaxResults()` to control the size of the result.

Special care must be taken when you are using agents. All elements from the agent result are sent to the client, no matter how large or small that set of results might be.

Using indexes

As with relational databases, indexes are important to speed up queries. WebSphere eXtreme Scale supports indexes on any attributes of the data.

Also, as with relational databases, there is a cost that is associated with an index. An index is stored in memory and must be considered when you calculate heap sizes for a grid container. Also, an index must be kept up to date when data is changed, which affects write speeds.

Understanding query performance

SQL databases usually include a highly sophisticated query optimizer to find an optimal query execution plan. WebSphere eXtreme Scale does not include any query optimization component. Based on the query string, a straightforward execution plan is created and run. The FROM clause is evaluated from left to right, so place entities with low cardinality first.

The same applies to WHERE clauses, so you should place conditions with high selectivity, and that are supported by an index, first. As an example, when querying a global company's data for all male employees who live in England, query on England first (one of many countries) and then on males (one of two genders).

Use rollback transactions when you run a read-only query to avoid expensive commit processing (known as walking the graph).

A helpful tip is to check the query execution plan for efficiency. This can be performed by logging the plan from the application (see Example 4-23) or by enabling the trace setting `QueryEnginePlan=debug=enabled`.

Example 4-23 Code to log the query plan

```
public void search(...) {
    String queryText = ...;
    Query query = ...

    if (log.isDebugEnabled()) {
        log.debug("Going to execute query "+queryText
            +" with plan " + query.getPlan());
    }
}
```

For more information about query tuning and query plans, see the following topics in the IBM WebSphere eXtreme Scale Version 8.6 Information Center:

► *Tuning query performance*

<http://pic.dhe.ibm.com/infocenter/wxinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsqryperf.html>

► *Query plan*

<http://pic.dhe.ibm.com/infocenter/wxinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsqueryplan.html>

4.4.2 Optimizing query performance using the global index

The global index capability is a new function in WebSphere eXtreme Scale version 8.6. The global index is a solution to the challenges articulated in 4.4.1, “Important considerations when using queries” on page 125. It addresses the question as to the best way to query across a large, partitioned environment where cached objects are spread across all partitions.

When you use regular indexes, queries, and agents, the query must run on all partitions. This is an expensive operation to retrieve data. Ideally, run these operations only on applicable partitions, and therefore eliminate unnecessary processor usage. This is the capability of the global index.

Global index tracks the location of indexed attributes, and therefore can determine which partitions contain the data you are interested in.

The Global Index provides these benefits:

- ▶ Efficient cross-partition access to index data.
- ▶ The index is automatically updated.
- ▶ Using secondary, global/reverse index maps to store global index data is no longer needed. This previously had to be done with bespoke development.
- ▶ Clients can query an index with predictable performance because only a subset of the partitions is queried.

The global index is an enhancement to the HashIndex plug-in introduced in 4.4, “Querying data using Object Grid Query Language” on page 122. It provides an API to allow:

- ▶ Clients to retrieve the partitions in which an indexed attribute exists, which is useful for routing agents.
- ▶ Index queries are enhanced to return keys and values without using an agent or multiple remote calls.

The following items introduce the changes that must be made to the original query example found in 4.4, “Querying data using Object Grid Query Language” on page 122.

- ▶ Enable the global index in the `objectgrid.xml`.

The change to the grid definition is trivial. Add a property to the existing index definition to enable the global index, as shown in Example 4-24.

Example 4-24 Addition to the `objectgrid.xml` file to enable global index

```
<backingMapPluginCollection id="GridPlugins">
  <bean id="MapIndexPlugin"
        className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
    <property name="Name" type="java.lang.String" value="QUANTITY" />
    <property name="RangeIndex" type="boolean" value="true"/>
    <property name="AttributeName" type="java.lang.String" value="qty" />
    <property name="GlobalIndexEnabled" type="boolean" value="true" />
  </bean>
</backingMapPluginCollection>
```

- ▶ Use the global index API to optimize the query logic.

The query code remains the same as introduced at the start of this section. Example 4-25 shows how to get the partitions that contain the data of interest. In this example, the application wants to query the QUANTITY value and the global index narrows the search to only those partitions that contain it.

Example 4-25 Code to use the global index

```
Session session = grid.getSession();
ObjectMap map = session.getMap(mapName);
MapGlobalIndex quantityIndex = (MapGlobalIndex) map.getIndex("QUANTITY");
//Find partitions of items with a quantity of the variable quantity
Collection<Integer> partitions = quantityIndex.findPartitions(quantity);

// just run query against partitions that have the keys we're interested in
for (Integer partitionNumber : partitions) {

    // run existing query logic on partition partitionNumber
    ...
}
```

Now, apply the global index to the query used previously as shown in Example 4-26. In this scenario, the query is looking to find items that have a quantity of 0 (they have run out). Without a global index, the client runs a query for each partition. On a large system, this can easily be over 50 queries.

Example 4-26 Query retrieving a list of Inventory objects with quantity of a specific limit.

```
SELECT i FROM Map1 i WHERE i.quantity =?1
```

Compare this to a global index query, where there might be only a handful of items with quantity of 0. By way of example, this might perform only five to ten queries, which is far more efficient.

Optimizing further

On a large data set, the spread of data of a non-primary attribute can be sufficiently random across the grid that a global index might not narrow down the number of partitions enough to benefit from it.

In this case, consider providing a custom partition key. WebSphere eXtreme Scale version 8.6 introduces a new feature, called `PartitionKey`, to specify the field or attributes on which a hash code calculation is run using a Java annotation.

For more information, see the *Mapping keys to partitions with PartitionKey annotations* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txspartitionkey.html>

4.4.3 Inverse range index

The inverse range index provides an index capability to search ranged data. Consider the following data sets, which can be searchable:

- ▶ Products that store promotion start and end dates. The inverse range index allows a search to be run to find products that are on promotion on a specific date.
- ▶ Insurance policies that only apply to a range of ages. You might have different policies, some apply to the people over 50, some for 40 to 60, and others for 35 to 55. The inverse range index enables a search to find policies that are applicable to a certain age, in this example, for someone who is 51.

To implement an inverse range index, there are two constraints, as it is a more complex search:

- ▶ The data must be defined and partitioned using a complex key. Example 4-27 on page 130 illustrates this. Specifically, all cache entries and search keys with the same value of non-range attributes must route to the same partition. This can be done by having the same non-range attributes in both the cache data key and the search data key.
- ▶ The search must be run on the server side, running in the grid using code such as an agent to run it. This is because the inverse range index implements a `MapIndexPlugin`, which cannot be run from the client.

This can be illustrated by extending the stock inventory example used elsewhere in this chapter with some range data, namely a promotion start and end data. To do this, there are two *key* Java objects needed:

- ▶ For the cache data key itself, Example 4-27 shows a simple example:
 - Non-range data such as product id
 - Range data such as promotion start and end dates

Example 4-27 Sample cache class with range data

```
public class ProductKey {

    private String sku;
    private Date startPromotionDate;
    private Date endPromotionDate;

    //getters and setters
}
```

3. For the search, Example 4-28 shows the corresponding search key:
 - All non-range attributes from the data key, so in this case the product id. This ensures that the search key routes to the same partition as the data key.
 - The search value for the range data, such as a specific date to search for promotions with.

Example 4-28 Sample search key for specific promotion date

```
public class ProductSearchKey {

    private String sku;
    private Date promotionDate;

    //getters and setters
}
```

The configuration for the inverse range index is similar to the other index configurations in the `objectgrid.xml` as shown in Example 4-29.

The important line of the configuration is the `AttributeName`. This string defines the search parameters (as obtained from the search keys `sku` and `promotionDate`) and how they relate to the range values from the data key. In this example, the search parameter string is:

```
sku[key.sku], promotionDate[key.startPromotionDate, key.endPromotionDate]
```

Example 4-29 objectgrid.xml file with inverse range index configuration shown

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://ibm.com/ws/objectgrid/config">

<objectGrids>
  <objectGrid name="Grid">
    <backingMap name="Map1" copyMode="COPY_TO_BYTES"
      pluginCollectionRef="GridPlugins"/>
    <querySchema>
      <mapSchemas>
        <mapSchema valueClass="com.ibm.itso.sample.InventoryBean"
          mapName="Map1" primaryKeyField="sku"/>
      </mapSchemas>
    </querySchema>
  </objectGrid>
</objectGrids>
```

```

    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
  <backingMapPluginCollection id="GridPlugins">
    <bean id="MapIndexPlugin"
      className="com.ibm.websphere.objectgrid.plugins.index.InverseRangeIndex">
      <property name="Name" type="java.lang.String" value="productData"/>
      <property name="AttributeName" type="java.lang.String"
        value="sku[key.sku],
        promotionDate[key.startPromotionDate, key.endPromotionDate]"/>
    </bean>
  </backingMapPluginCollection>
</backingMapPluginCollections>

</objectGridConfig>

```

After the data and configuration is set up, the code to use the index to run a find is easy and efficient. The code in Example 4-30 shows an example using the index. The implementation that is shown is in an agent process method. For more information about developing agents, see 4.5, “Server-side development with agents” on page 132.

Example 4-30 Agent code snippet to run the Inverse Range Index

```

MapIndex codeIndex = (MapIndex) map.getIndex("InverseRangeIndex");
//return keys that match search for processing
Iterator result = codeIndex.findAll(searchKey);
//process objects for found keys

```

To deploy the code onto Liberty, it must be packaged into an OSGi bundle (a JAR file with specific contents in the MANIFEST.MF file) and placed in the grids directory of the server. This simple process is explained in 4.5.2, “Liberty Profile configuration” on page 136.

The important configuration needed to allow the sample to work in an OSGi bundle is to specify the correct import packages in the MANIFEST.MF file, such as:

```

Import-Package: com.ibm.websphere.objectgrid,
com.ibm.websphere.objectgrid.datagrid, com.ibm.websphere.objectgrid.plugins.index

```

The IBM Elastic Caching Community provides a sample application for using the inverse range index that is based on the scenario in this section:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/inverse_range_index?lang=en

4.4.4 Continuous query

A new feature in WebSphere eXtreme Scale version 8.6 is the continuous query. The purpose of a continuous query is to proactively notify the client when a change to the data set of interest occurs.

Compared to a traditional query, where the client run perform the query (or *pull* the information), with the continuous query, the client registers a query filter and gets notified (*push* the information) when the result set matching that query changes. This feature uses the

publish/subscribe model, which is used internally to perform the near cache invalidation. There is no further infrastructure required within the eXtreme Scale configuration.

A simple example is illustrated in Figure 4-15, where the filter is `stock.value > 100`. That is, the application wants to be notified when any stock value goes over 100.

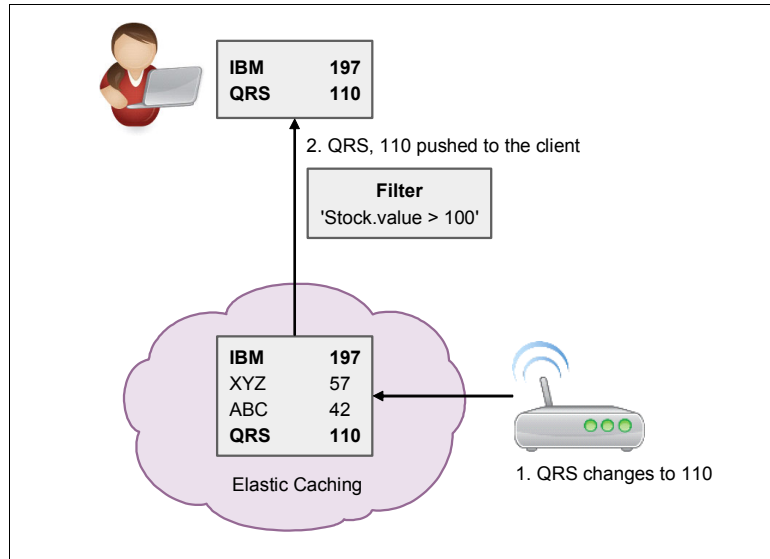


Figure 4-15 Continuous query

To configure a continuous query, complete these steps. The process to set this up requires some code.

1. Define a filter chain that passes the keys/values of interest.
2. Implement a custom filter or use the provided basic filters of EQ, NEQ, GT, GTE, NotNull, IsNull, LT, LTE, RegEx, NotRegEx, And, Not, Or
3. Handle the changes that match the filter. The following range of events can be handled when the data changes:
 - Insert: There is a new object that matches the query.
 - Update: An object that did not match the query, matches it now.
 - Update: An object that did match the query, does not match it now.
 - Update: An object that did match the query, still matches it, but value has changed. This is optional.
 - Delete/Clear/Invalidate: An object that did match the query has been removed.

The IBM Elastic Caching Community provides a sample application for using the continuous query:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/continuous_query_sample

4.5 Server-side development with agents

This section describes an implementation of the XTP pattern from 3.5, “Extreme Transaction Processing scenario” on page 85 using the MapReduce programming model. This is a fundamental shift in the approach to writing applications because this runs code directly on

the eXtreme Scale grid. This section therefore does not apply to the WebSphere DataPower XC10 Appliance.

Running logic on the grid is done by developing *agents*. A copy of an agent runs on each defined partition in parallel. Each agent has direct read/write memory access to all of the data that are held in its partition. This approach to developing can facilitate high-scale performance and scale applications, but can also be used as approach to improve normal side cache and in-line cache scenarios as well.

There are two WebSphere eXtreme Scale APIs to provide an implementation of the MapReduce, a technique for parallel processing and querying large quantities of data:

- ▶ `com.ibm.websphere.objectgrid.datagrid.MapGridAgent`
Returns a map of results from a partition. For example, a `MapGridAgent` can be useful for retrieving data with some calculation performed on each result.
- ▶ `com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent`
Returns a single result from a partition. For example, a `ReduceGridAgent` can be useful for working out the lowest or highest value for a partition.

The use of agents can be illustrated with an example using the `ReduceGridAgent`. It is an enhancement to the inventory scenario used in 4.4, “Querying data using Object Grid Query Language” on page 122. In the scenario, the grid maintains the status of a product inventory. It might be reasonable to query the grid to ascertain the total stock value of the inventory. Using the techniques described so far, the client must query each partition for each object stored in the grid and return the objects to the client side for processing. This is not efficient.

With agents, the processing can be pushed to the grid and run in parallel. The logic for this is simple, but it illustrates clearly what logic is run where.

- ▶ On the grid, each partition is responsible for calculating the value of the inventory in its partition and return the result for aggregation.
- ▶ On the client, the agent waits for the results from each partition and aggregate the totals to provide the complete inventory total.

4.5.1 Agent development

Example 4-31 shows the `ReduceGridAgent` interface. The two reduce methods are run on the grid, and `reduceResults` is the method that is run on the client after the grid-side execution is complete.

Example 4-31 `com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent`

```
public interface ReduceGridAgent {  
  
    /**  
     * Provides the logic to be run on each partition.  
     * @return Object containing the result of the processing  
     */  
    public Object reduce(Session s, ObjectMap map);  
  
    /**  
     * Provides the logic to be run on each partition.  
     * In this case, the entries for the partition are constrained  
     * further by providing a collection of the keys to be processed  
     * @return Object containing the result of the processing  
     */  
}
```

```

*/
public Object reduce(Session s, ObjectMap map, Collection keys);

/**
 * Run once for the whole grid.
 * It is run after the reduce methods have been run.
 * @param A Collection of Java Objects, which are the results from
 * each reduce method.
 */
public Object reduceResults(Collection results);
}

```

Example 4-31 on page 133 calculates the overall value of stock that is held by an organization, where the stock inventory is represented by data in the grid.

To illustrate this example, a sample is provided. For more information about how to obtain this example, see Appendix A, “Additional material” on page 341.

To implement the interface for this example, the following two methods can be implemented:

- ▶ `reduce()` to calculate the inventory total for a partition. This is run on the grid on each partition.
- ▶ `reduceResults()` to aggregate the inventory total that is returned from the partitions. This is run on the client.

The process is illustrated in Figure 4-16. The third method, `reduce`, is not implemented here, but can be implemented when a range of keys can be provided to the agent for processing the corresponding values.

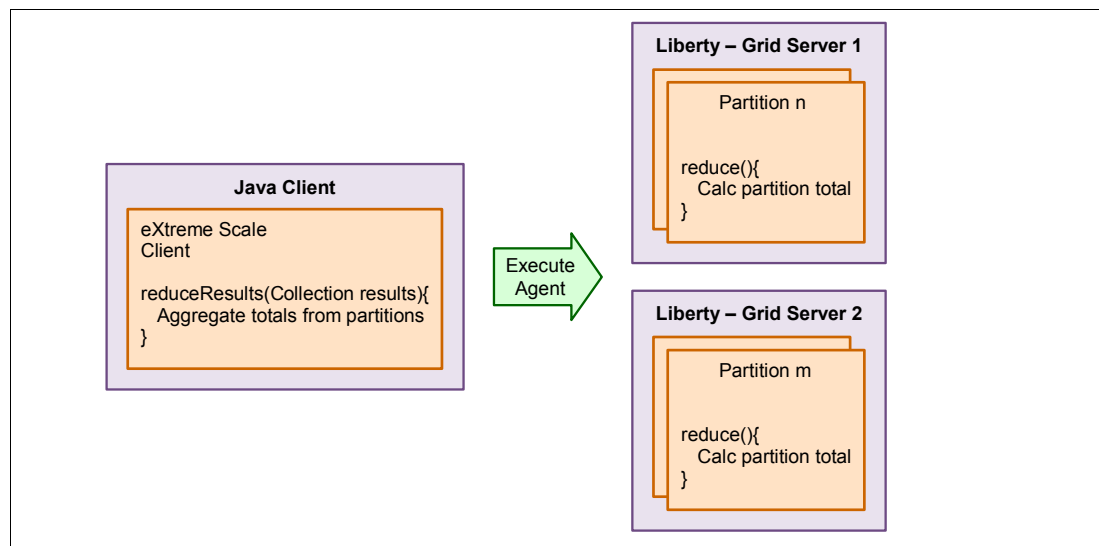


Figure 4-16 Overview of agent execution

Example 4-32 shows sample code from this scenario. For more information about the simple query implemented in the reduce method, see 4.4, “Querying data using Object Grid Query Language” on page 122. In reality, queries are normally more complex.

Example 4-32 StockValueReduceAgent implementation

```
public class StockValueReduceAgent implements ReduceGridAgent{

    /*
    * Server-side calculation of value of stock held in each partition
    */
    public Object reduce(Session session, ObjectMap map) {
        //Query everything in the partition (in reality this would be smarter!)
        String queryString = " select i from Map1 i";
        ObjectQuery query = session.createObjectQuery(queryString);

        // process results
        BigDecimal value = new BigDecimal(0);
        for (Iterator i = query.getResultIterator(); i.hasNext();) {
            InventoryBean inventory = (InventoryBean)i.next();
            value = value.add(
                inventory.getUnitPrice().multiply(
                    BigDecimal.valueOf(inventory.getQty())));
        }

        System.out.println("Returning total value for partition: "+value);
        return value;
    }

    //unused
    public Object reduce(Session arg0, ObjectMap map, Collection keys) {
        return null;
    }

    /*
    * Client-side aggregate results from each partition
    */
    public Object reduceResults(Collection results) {
        BigDecimal totalValue = new BigDecimal(0);
        for (BigDecimal partitionValue : (Collection<BigDecimal>)results) {
            totalValue = totalValue.add(partitionValue);
        }
        return totalValue;
    }
}
```

After you develop the agent code, it must be started by the client application. Example 4-33 illustrates how to do this by obtaining a reference to an AgentManager from the map.

Example 4-33 Client-side code to execute an agent

```
public BigDecimal executeAgent(String mapName) throws ObjectGridException{
    Session session = grid.getSession();
    ObjectMap map = session.getMap(mapName);
    AgentManager amgr = map.getAgentManager();
```

```

StockValueReduceAgent agent = new StockValueReduceAgent();

BigDecimal total = (BigDecimal) amgr.callReduceAgent(agent);
session.close();
System.out.println("Total stock value is calculated as: "+total);
return total
}

```

A MapGridAgent would be implemented similar to this example. The difference with the MapGridAgent is that it returns a Map of processed entries. Be careful that this does not become a large amount of data being serialized back to the client.

To run the agent code, any Java classes that the agent needs to run must be provided on the grid server-side. This is straight forward in a stand-alone Java deployment of eXtreme Scale, as it can be added to the class path. There are some different packaging requirements if you are deploying it in Liberty. For more information, see 4.5.2, “Liberty Profile configuration” on page 136.

4.5.2 Liberty Profile configuration

The grid configuration (`objectgrid.xml` and `objectGridDeployment.xml`) for the sample code remain the same as that defined in Example 4-20 on page 123. The grid configuration files do not contain anything specific to the use agents, although the map schema define is used by the query in the agent implementation.

To deploy the agent in the Liberty Profile, complete the steps in 4.4, “Querying data using Object Grid Query Language” on page 122 to package all Java classes referred to by the agent code and inventory object. Deploy this package as an OSGi bundle (JAR file with a special `MANIFEST.MF` file) in the `grids` directory of the Liberty Profile server instance.

The `MANIFEST.MF` file (Example 4-21 on page 124) needs revising to include the dependencies of the agent code. If you are not familiar with OSGi, a key concept is that of explicitly managing module imports and exports. This is where OSGi helps to regulate situations where the class path is not managed.

Example 4-34 illustrates how the classes required by the agent are explicitly defined.

Example 4-34 Extended MANIFEST.MF file

```

Manifest-Version: 1.0
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Bundle-Name: QueryInventory
Bundle-SymbolicName: QueryInventory
Export-Package: com.ibm.itso.sample
Import-Package: com.ibm.websphere.objectgrid,
    com.ibm.websphere.objectgrid.datagrid, com.ibm.websphere.objectgrid.query
Bundle-Version: 1.0.0
Bundle-ManifestVersion: 2

```

The Java classes and MANIFEST.MF file are ready to be exported to the grids directory of the Liberty Profile instance as shown in Figure 4-17.

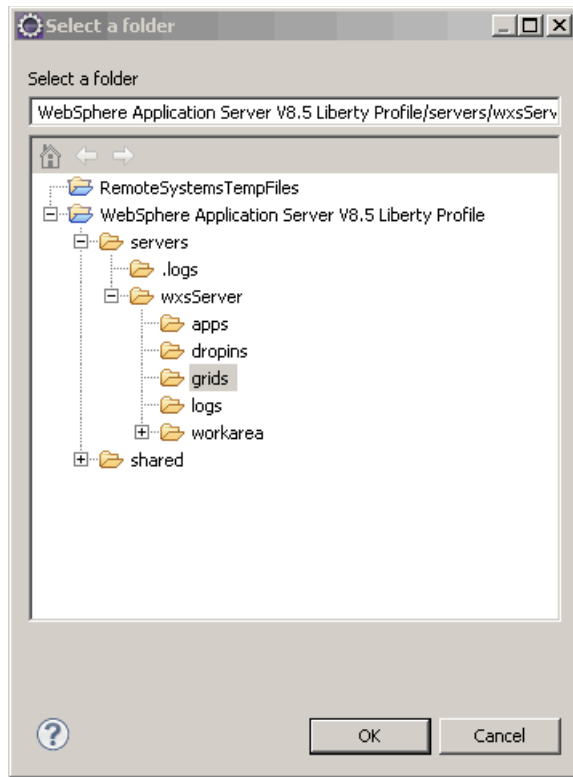


Figure 4-17 OSGi bundle (JAR) deployment in the Liberty Profile

The client code can be run from any Java environment, such as a web application in the Liberty Profile or a stand-alone Java Standard Edition (SE). If you are accessing the agent from a Java SE client, ensure that the client and server are at the same product and fix level.

To run code with a Java SE client, download a full (or trial) WebSphere eXtreme Scale environment because the eXtreme Scale Liberty feature download does not provide this. For development, this is available as a trial from the following website:

<http://www.ibm.com/developerworks/downloads/ws/wsdg>

The IBM Elastic Caching Community provides a sample application for using agents:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/running_queries_in_parallel_using_a_reducegridagent1

4.6 Dealing with data eviction and stale data

This section describes the methods for dealing with state cache data and data eviction:

- ▶ 4.6.1, “Tolerating the data” on page 138
- ▶ 4.6.2, “Using time-based eviction strategies” on page 138
- ▶ 4.6.3, “Polling the database for updates in regular intervals” on page 139
- ▶ 4.6.4, “Using JMS to propagate changes” on page 140
- ▶ 4.6.5, “Using near cache invalidation in WebSphere eXtreme Scale version 8.6” on page 140

- ▶ 4.6.6, “Assuring no external changes are made to backing store” on page 141
- ▶ 4.6.7, “Ensuring all external change processes notify the grid” on page 141

Whenever an application uses a cache or grid function with a hardened back-end data store, the problem of stale data in the cache must be considered. The application might read old data from the cache when the data has already been updated in the back end. This situation is called a *dirty read* because of a stale cache. This situation can have severe impacts for the application and the business it supports. It occurs frequently when many external changes (that is, not managed by the application) are applied to the back-end data store.

Determine your requirements for cached data accuracy by answering the following questions:

- ▶ How acceptable is a dirty read?
- ▶ What are the consequences if one occurs?
- ▶ What is the maximum acceptable time between an update in the back-end store and an update in the cache?

The answers to these questions depend on the type of data and the type of use cases the application drives against it. The answers can range from “dirty reads can occur and do not really matter” to “dirty reads are not acceptable in any case.”

For example, a social community web application that uses a cache from which to read profile images of users can tolerate a dirty read easily. Showing an out of date image is a minor inconvenience, and probably not even noticed by other users.

A different example would be an online stock trading application that reads stock exchange rates from the cache for actual trades. A dirty read can result in a wrong price calculation, imposing a loss to the brokerage. That makes dirty reads intolerable, and the solution must ensure that they cannot occur.

Establish accuracy requirements for each entity that is stored in the cache. Different entities might have different requirements for a single application. Most applications that use a cache are developed to handle stale data, regardless of how small the likelihood, where appropriate. For example, an e-commerce site might have small intervals of stale data in the cache of quantities of product stock, but before purchase it can review the stock quantity directly with the data source to get a definitive number.

The following sections address different options and what eXtreme Scale offers to deal with this issue.

4.6.1 Tolerating the data

If requirements allow, dirty reads can simply be accepted.

4.6.2 Using time-based eviction strategies

You can use a time-based eviction strategy (for example, every 30 minutes or every day at 23:00 and 05:00) to establish an upper bound for the dirty read time. WebSphere eXtreme Scale provides a *time to live* (TTL) evictor that can be used to expire old data from the cache.

Example 4-35 on page 139 provides a sample `objectGrid.xml` file with a TTL evictor configured to expire data 30 minutes after the last access or update time. In this example, eXtreme Scale maintains two timers for last *access* time and for last *update* time. They can be used separately or combined with an OR as shown.

Note: This strategy can cause peak loads at the back end when a lot of invalidated data is re-read at a certain point in time. It can also cause performance degradation because of numerous cache misses after invalidation occurred.

Example 4-35 Configuring a TTL evictor

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid1">
    <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME|LAST_UPDATE_TIME"
      timeToLive="1800" />
  </objectGrid>
</objectGrids>
```

4.6.3 Polling the database for updates in regular intervals

The cache application polls the back-end store for the latest changes. This application can either invalidate the objects from the cache or directly update it. Information provided by the back-end store is used to determine changes. Examples can be a time stamp column “last modified” that denotes the point in time. In this case, it is easy to select all rows that have been changed since the last row. It is common for a business application to have a “last modified” time stamp for audit reason that can be used right away.

If such a column is not already present, it might be introduced without changing the application by using back-end store features to create and maintain this column. For example, IBM DB2® UDB V9.5 supports a column that is defined as follows:

```
last_modified TIMESTAMP NOT NULL
  GENERATED ALWAYS
  FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP
```

DB2 ensures that the column *last_modified* is always updated.

WebSphere eXtreme Scale supports polling the database for changes for JPA using a TimeBasedDBUpdater. For more information, see 4.3.3, “Implementing a time-based updater” on page 117. Also, see the *Starting the JPA time-based updater* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsjpadbudprog.html>

When JPA is not used, a simple approach can be used, with the application creating a background “worker thread” to poll changes from the database and update the grid:

- ▶ This approach does not invalidate objects that have been deleted from the back end. Deleted objects are hard to select from a database for obvious reasons.
- ▶ Stale data can still exist with polling-based invalidation. There can be quite a large time period between the time data changes and the next poll, during which clients will get stale data.

4.6.4 Using JMS to propagate changes

Stale data can arise when several clients use a near cache and have local copies for the same key, and then one client modifies this data. By default, the other clients are not notified of the modification, so they have a stale object in their own near cache.

JMS messaging, especially publish/subscribe using topics, can be used to push changes from the grid to near caches on each client. WebSphere eXtreme Scale supports this feature by using the `JMSObjectGridEventListener`. For more information, see the *Class `JMSObjectGridEventListener`* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/plugins/builtins/JMSObjectGridEventListener.html>

Leading practice: Use small near caches with quick eviction strategy. Stale near caches can be avoided by configuring a small cache size and an aggressive eviction strategy that promptly removes the objects from the near cache. Keep in mind that there is a large grid behind the near cache, and it is not expensive to get the data from the grid again. So the benefits from a near cache are fairly small. The grid has low redundancy of objects to free more space for caching, and a clever replication mechanism to ensure availability and consistency.

The `JMSObjectGridEventListener` can be used to synchronize any number of grids. This approach can be useful when strong separation between grids (for example, data centers in different locations) prohibit tight coupling using IIOP. However, data exchange using message-oriented middleware is a valid option.

The JMS `publish` and `subscribe` methods of invalidating data in near cache has effectively been replaced as of WebSphere eXtreme Scale version 8.6 with the introduction of in-product near cache invalidation. This topic is described in 4.6.5, “Using near cache invalidation in WebSphere eXtreme Scale version 8.6” on page 140.

4.6.5 Using near cache invalidation in WebSphere eXtreme Scale version 8.6

With WebSphere eXtreme Scale version 8.6, a new feature was introduced that simplifies near cache invalidation. The new feature provides notification to clients by using near cache upon any `remove()`, `update()`, `invalidate()`, or `clear()` `ObjectMap` operations on the grid. When these events occur, the server sends a list of affected keys to interested clients. The clients then invalidate their local near caches based on the key list. The invalidation events are propagated from the server to the clients asynchronously. Because the invalidation is done asynchronously, there is still a possibility of a client using stale data from the near cache.

No client code changes are required to enable the near cache invalidation. All near cache invalidation configuration is done in the `objectGrid.xml` configuration file.

Near cache evictors: Continue to use near cache evictors to prevent unbounded growth of the near cache instance.

To use near cache invalidation, you must use XIO and near caching must be enabled.

To use near cache, the `lockStrategy` value must be set to `NONE` or `OPTIMISTIC`. By default near cache is enabled because `lockStrategy="OPTIMISTIC"` in the `objectGrid.xml` file. To

confirm this, the `BackingMap.isNearCacheEnabled()` API call can also be used (Example 4-36):

1. Configure the `objectGrid.xml` file

Example 4-36 Setting up near cache invalidation in the `objectGrid.xml` file

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="Grid">
      <backingMap name="Owner" nearCacheInvalidationEnabled="true" lockStrategy="OPTIMISTIC"
copyMode="COPY_TO_BYTES"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

2. Restart the container servers and clients.

4.6.6 Assuring no external changes are made to backing store

This approach prohibits external changes to be directly applied to the backing store. Instead, every change to the data is only allowed through applications that participate in the cache. This approach can be difficult if other components or applications that are not aware of the cache have write access to the backing store. For example, subsystems, interfaces to external systems, and data cleansing scripts would not be allowed to change data.

However, for those rare situations where this restriction can be accommodated and enforced, this approach can be highly efficient. By making the grid the “system of record”, all reads and updates are made exclusively from and to the grid. The grid itself updates the database as a write-through configuration. In this manner, the grid content is always the most current.

4.6.7 Ensuring all external change processes notify the grid

Instead of *prohibiting* all external changes as in 4.6.6, “Assuring no external changes are made to backing store” on page 141, this approach envisions that the grid is *notified* of all external changes. Figure 4-18 on page 142 gives an example of this approach. All database scripts or other external modifications must be equipped with a grid client that notifies the grid about changes. The easiest notification would be to invalidate the data in the cache. A more sophisticated solution is to directly update the data.

The downside of this approach is that it might be difficult to ensure that all scripts, applications, and so forth, are aware of the grid.

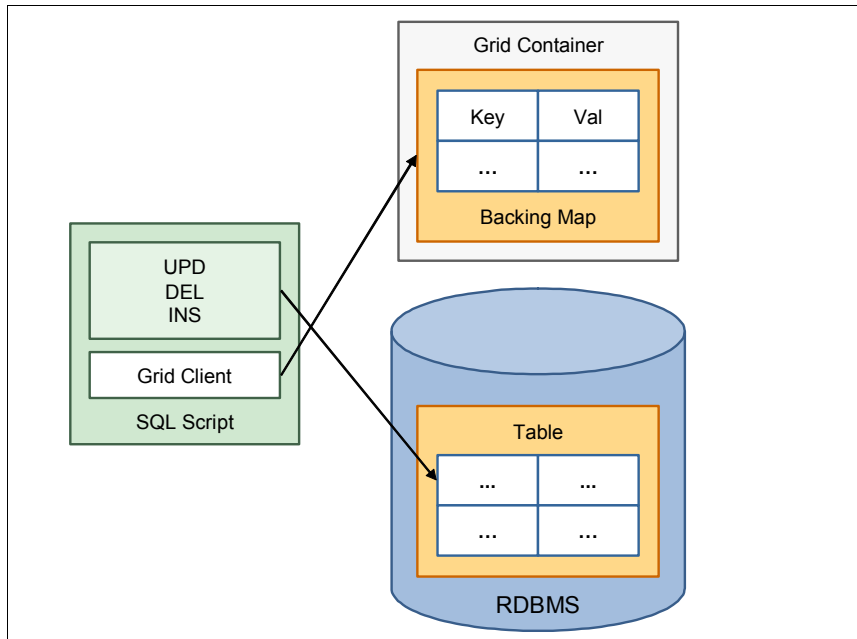


Figure 4-18 Notification of external changes to grid

For SQL databases, post-insert, update, and delete triggers can be used to invalidate or update the grid as depicted in Figure 4-19. This approach requires a grid client to be embedded into the database. To do so, the database needs to support triggers that are implemented in Java.

The triggers are usually run before the transaction is committed. This circumstance can lead to an unnecessary cache invalidation when the transaction is rolled back. But this invalidation is acceptable when dirty reads cannot be tolerated. It is better to have an unnecessary invalidation than a stale object.

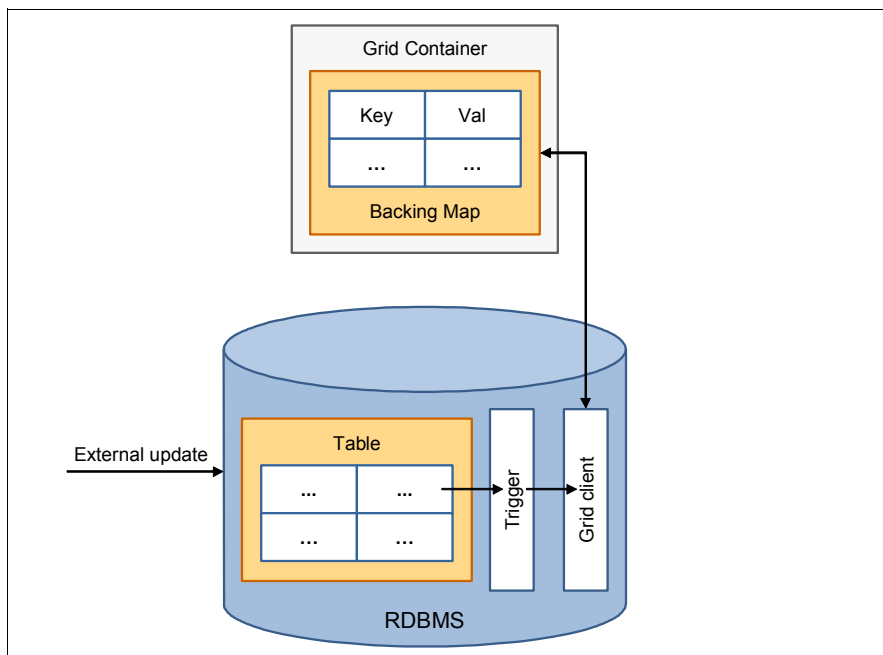


Figure 4-19 Pushing changes from the database to the grid by using triggers

4.7 Saving time with WXSUtils

The WXSUtils Java library is a convenience library that is designed to simplify various operations against an eXtreme Scale data grid instance.

Attention: The WXSUtils package is not shipped with WebSphere eXtreme Scale, nor is it supported as part of the product. Consider it *technology preview* code.

WXSUtils requires eXtreme Scale version 7.1.1 or higher and Java version 5 or higher. Java version 5 or later is required because of the use of Java generics.

WXSUtils provides the following operation simplifications:

- ▶ Client-to-grid connections using a simple parameterized method or a configuration file approach.
- ▶ Obtaining references to ObjectMaps.
- ▶ Bulk update, insert, remove, and invalidate operations. For example, inserting a batch of 1000 key/value pairs into the grid with a single API call, and having the corresponding actions on the grid side done in parallel across all partitions. This feature can simplify the writing of an efficient client-side grid preloader application.
- ▶ Conditional put operations that are based on a key value or presence.
- ▶ Data filtering.
- ▶ Working with maps as Lists.
- ▶ Working with maps as Sets.
- ▶ Running jobs that touch every partition in the grid (and all the data within the partition).
- ▶ Snapshot map and partition data to a file.

For a complete listing of the features that are offered by the WXSUtils library and how to use them, see the *WXSUtils Programming Guide V0.1* at:

<https://github.com/bnewport/Samples/blob/master/wxsutils/WXSUtilsProgramming.pdf>

Note: WXSUtils makes extensive use of Java ThreadLocals and static variables. Therefore, use WXSUtils in a Java Platform, Enterprise Edition Web or EJB module with care. Starting or stopping a Web or EJB module is only supported if the bundled servlet listener, `com.devwebsphere.wxsutils.WXSUtilsServletListener`, is used. Unless this listener is specified and registered, the JVM containing the Java Platform, Enterprise Edition Web or EJB module must be restarted if the module is updated to correctly reinitialize the WXSUtils library.

4.7.1 Examples of usage

This section provides examples to demonstrate how to use WXSUtils for common purposes. Your application can mix WXSUtil APIs with conventional ObjectGrid APIs. For example, you can use the bulk loading capability of WXSUtils to preload batches of data into the grid, and use a traditional ObjectGrid get method to retrieve information from the grid.

Connecting to a grid

Example 4-37 demonstrates a concise, easy to use means of connecting an eXtreme Scale client to a grid instance.

Example 4-37 Establishing a client connection

```
ogclient = WXSUtils.connectClient("host:2809", "Grid",
"/multijob/multijob_objectgrid.xml");
WXSUtils utils = new WXSUtils(ogclient);
```

Making a bulk update

Example 4-38 demonstrates how to run a bulk update on a grid. The example shows just two key/value pairs being put in bulk, but this can easily be extended to 200 or 2000.

Example 4-38 Performing a bulk update

```
WXSSMap<String,String> map = ...;
Map<String, String> batch = new HashMap<String, String>();
batch.put("K1", "V1");
batch.put("K2", "V2");
map.putAll(batch);
```

Starting a development mode test grid instance

Example 4-39 demonstrates an easy way to start a grid instance in the same JVM as the application, allowing for easy debugging and deployment.

Example 4-39 Starting a development mode test grid instance

```
ogclient = WXSUtils.startTestServer("Grid", "/objectgrid.xml", "/deployment.xml");
```

4.8 Transactions

This section introduces transactional programming within WebSphere eXtreme Scale, and particularly highlights these concepts:

- ▶ 4.8.1, "Transaction considerations" on page 144
- ▶ 4.8.2, "JTA transactions" on page 148
- ▶ 4.8.3, "Multi-partition transactions" on page 148

4.8.1 Transaction considerations

WebSphere eXtreme Scale runs as a transaction manager to allow transactional updates to the grid. The following are reasons why you need transactional updates:

- ▶ To apply multiple changes as an atomic unit. Either all changes get persisted, or none do. There is no situation where just some of the changes persist.
- ▶ To allow all changes to be rolled back in the case of an error.
- ▶ To ensure consistency of cached data in an environment where there can be many concurrent accesses to the same data.
- ▶ To act as the unit of replication to make the changes durable.

Following are some decisions to make when you are using transactions:

- ▶ Using implicit or explicit transactions in the application
- ▶ Determining lock strategy
- ▶ Determining copy mode
- ▶ Using locks

Using implicit or explicit transactions in the application

In an application, transactions can be implicit (auto-commit) or explicit (demarcated by `begin` and `commit` or `rollback`).

With implicit transactions, the application developer does not specify a transaction scope. Rather, each method of accessing the data in the grid is wrapped in a single transaction. This is simple to code. However, if there are many updates to go into the grid, this performs more slowly than explicitly grouping or batching updates into a single transaction.

As a general guideline, use explicit transactional control within the application. Fewer, larger transactions are faster than many smaller transactions. It improves the performance of any synchronous replication too, as these are committed in a batch. A side effect of larger transactions is that locks are held for longer, which can adversely affect concurrency.

If you are using the EntityManager API, use explicit transactions because otherwise the entity is disconnected from the grid.

To improve the performance when preloading data, use explicit transactions to group updates into batch updates for each partition.

WebSphere eXtreme Scale v8.6 introduced a JCA resource adapter that can integrate with the transactional control of a Java EE environment. For more information, see 4.8.2, “JTA transactions” on page 148. Use this adapter if you are integrating with transactions in a Java EE environment.

Determining lock strategy

A key decision when you are implementing transactions is to decide what to do with the data you are performing an update on *while* you are doing it. This is where the lock strategy comes in. This allows you to configure whether anyone can access the data that you are using, while you are using it.

Similar to relational databases, WebSphere eXtreme Scale provides the following lock strategies:

- ▶ Pessimistic
- ▶ Optimistic
- ▶ No locking

Pessimistic locking is where you assume that there are going to be frequent concurrent updates of the same data. The grid holds a lock on the data you are using and prevents anyone else from accessing it. Pessimistic locking has the greatest impact on the throughput of the grid because concurrent threads are kept waiting while existing transactions finish. However, it is particularly useful if there are many concurrent updates to the same data. It provides the best consistency of data.

Important: Pessimistic locking cannot be used with the near cache.

Optimistic locking is where you assume that there are *not* going to be frequent concurrent updates of the same data. That is, concurrent updates are relatively rare, though are still

possible and supported. The grid does not hold a lock on the data you are using. Instead, it takes a copy of the version of data you are using when you start your transaction. It uses this copy to compare with the version of data in the grid *before* you commit the transaction. If the data has not changed, the optimistic transaction commits. If the data *has* changed, there has been a concurrent update to the data. The transaction is rolled back with a `OptimisticCollisionException`. Use optimistic locking if there is a low chance of concurrent writes to the same data. The application must handle the rare scenario where there are concurrent updates by setting up a `try/catch` block to detect the possible `OptimisticLockException`. In this case, it gets the newly updated value from the grid, updates it locally, and attempts to commit the change again. The grid can benefit from higher throughput as a result of not holding locks. However, there is a small processing effect due to the need to copy more version data.

No locking is where eXtreme Scale performs no lock on the data. It assumes that the application is using the preloaded grid in read-only mode, or is providing its own concurrency control. Use this with caution.

Determining copy mode

WebSphere eXtreme Scale sometimes needs to make copies of the data it is operating upon. This is needed for various reasons, but primarily so that the application can change the data in a temporary “work area” without exposing the `BackingMap` data, and therefore other threads, to changing and inconsistent data. To make copies takes processing time, so avoid doing it if not necessary. This is especially relevant if you are using Java serialization, which can be slow.

This section provides a brief review of the copy mode. Use `COPY_TO_BYTES` or `COPY_TO_BYTES_RAW` whenever possible because they provide the most efficient means of storing data within the grid. This is not the default copy mode, so it must be explicitly configured as such in your `objectgrid.xml` configuration file. For example:

```
<backingMap name="myMap" copyMode="COPY_TO_BYTES" />
```

The copy mode is less relevant when you are using a remote grid without a near cache because the cached data must be copied to the client on `ObjectMap.get()` and, if the data is updated, copied back to the grid. The copying here cannot be minimized using copy modes. The following `copyMode` options create two copies:

- ▶ `COPY_TO_BYTES`
Stores the data on the grid as bytes (XDF) and inflates to Java objects on the client.
- ▶ `COPY_TO_BYTES_RAW`
Stores the data on the grid as bytes (XDF) and makes the bytes available directly to the client.
- ▶ `COPY_ON_READ_AND_COMMIT`
Stores the data on the grid as Java objects. It by default uses Java serialization to take a copy of the Java object when persisting it to the grid. Serialization can be optimized by using the `DataSerializer` plug-in. This is the default copy mode.

If the data is *local to the application* (near cache or agent) and the grid holds Java objects (as opposed to XDF), the following `copyMode` options can optimize the processing:

- ▶ `COPY_ON_READ`
A copy is made when using `ObjectMap.get()`, but it is not copied at the end of the transaction. Rather, the grid data is replaced with the reference to the changed data. Use this mode only if the application will not edit the copied Java object after it is committed.

► COPY_ON_WRITE

Does not make a copy of the object on `ObjectMap.get()`, but instead uses a proxy to the data, which can be updated. The data from the proxy are copied into the grid on transaction commit only if the data has changed.

► NO_COPY

No copies of the cached data are made, so this option requires the least processing. However, any changes are made directly to the data held in the `BackingMap`. This can cause data inconsistencies, so use with caution.

For more information about the use of the copy mode, see *WebSphere eXtreme Scale Best Practices for Operation and Management* at:

<http://www.redbooks.ibm.com/redpieces/abstracts/sg247964.html>

Using locks

Concurrent access to data is handled by locks. Whether a read or write can occur to a cached object depends on the locks that are held on that object. There are three levels of lock as shown in Table 4-1.

Table 4-1 Lock modes

Lock mode	Obtained by method	Description
SHARED	<code>get()</code>	A SHARED lock allows some concurrency. Another thread wanting to read data by using <code>get()</code> or <code>getForUpdate()</code> can still do so. Other threads cannot update the data while a SHARED lock is held.
UPGRADEABLE	<code>getForUpdate()</code>	An UPGRADEABLE lock restricts concurrent access to the data. Another thread can still read the data, but can only perform a <code>get()</code> . They cannot perform a <code>getForUpdate()</code> or update the data.
EXCLUSIVE	<code>commit()</code> , <code>getNextKey()</code>	An EXCLUSIVE lock prevents all concurrent access to the object. It is not possible for another thread even to read the data while an EXCLUSIVE lock is held.

The correct use of locks can prevent deadlocks. For more information, see the *Locks* topic of the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxslock.html>

WebSphere eXtreme Scale Version 8.6 added two new APIs to explicitly lock objects in the grid:

- `boolean ObjectMap.lock(Object key, LockMode lockMode);`
- `List<Boolean> ObjectMap.lockAll(List keys, LockMode lockMode);`

These APIs apply only when you are using the pessimistic locking strategy. Before version 8.6, a lock can be obtained only by retrieving the data, which is wasteful if the `get()` was not needed. Instead of using `get()` or `getForUpdate()`, one of the lock methods can be used instead.

For more information about transactions within WebSphere eXtreme Scale, see the *Transaction processing overview* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsvertxn.html>

4.8.2 JTA transactions

In versions of WebSphere eXtreme Scale before version 8.6, the only way to propagate transactionality into the grid was by using the *WebSphereTransactionCallback* plug-in as demonstrated programmatically in Example 4-40.

Example 4-40 WebSphereTransactionCallback plug-in for JTA transactions

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

In WebSphere eXtreme Scale version 8.6, a *Java EE Connector Architecture* (JCA) adapter was included in the product to facilitate tighter Java Platform, Enterprise Edition transaction integration. By using the JCA adapter, grid operations can be included alongside XA enlisted resource managers using *Last Participant Support* (LPS). LPS describes an extension to the transactional support model that allows a single one-phase resource to participate in a two-phase transaction with one or more two-phase resources.

When you use the JCA adapter, your Java Platform, Enterprise Edition application can perform these tasks:

- ▶ Look up or inject eXtreme Scale resource adapter connection factories within a Java EE application component.
- ▶ Obtain standard connection handles to the eXtreme Scale client and share them between application components using Java EE conventions.
- ▶ Demarcate eXtreme Scale transactions by using either the `javax.resource.cci.LocalTransaction` API or the `com.ibm.websphere.objectgrid.Session` interface.
- ▶ Use the entire eXtreme Scale client API, such as the `ObjectMap` API and `EntityManager` API.

Additionally, the following extra capabilities are possible with WebSphere Application Server:

- ▶ Enlist eXtreme Scale connections with a global transaction as a last participant with other two-phase commit resources. The eXtreme Scale resource adapter provides local transaction support, with a single-phase commit resource. With WebSphere Application Server, your applications can enlist one single-phase commit resource into a global transaction through last participant support.
- ▶ Automatic resource adapter installation when the profile is augmented.
- ▶ Automatic security principal propagation.

The IBM Elastic Caching Community provides a sample application for using the JTA resource adapter:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/websphere_extreme_scale_jta_sample_using_the_wxs_resource_adapter

4.8.3 Multi-partition transactions

WebSphere eXtreme Scale provides the capability to spread data across multiple partitions in a data grid. WebSphere eXtreme Scale Version v8.6 adds the ability to read and update

several partitions in a single transaction. This type of transaction is called a *multi-partition transaction*. It uses the two-phase commit protocol to coordinate and recover the transaction in case of failure. A prerequisite to using this is that you must be using the XIO protocol.

In some scenarios, an application might need to interact with data on two or more maps in the same transaction. However, when interacting with data across a grid, it is likely that the application is using multiple partitions. If it is mandatory that an update is performed to multiple partitions in a single transaction, you can do this by enabling the two-phase commit protocol by using the following code:

```
session.setTxCommitProtocol(Session.TxCommitProtocol.TWOPHASE)
```

Example 4-41 illustrates how to create, retrieve, update, and delete operations in a grid with a two-phase commit protocol.

Example 4-41 Using multi-partition transactions

```
Session session = objectGrid.getSession();
Objectmap countryMap = session.getMap("Country");
Objectmap ibmOfficeMap = session.getMap("IBMOffice");

session.setTxCommitProtocol(Session.TxCommitProtocol.TWOPHASE);

session.begin();
countryMap.insert("DDD55", "Brazil");
ibmOfficeMap.insert("DDD5511", "Sao Paulo");
ibmOfficeMap.insert("DDD5519", "Hortolandia");
ibmOfficeMap.insert("DDD5521", "Rio de Janeiro");
session.commit();
```

When an application uses a `Session` object, the session must be in the context of a transaction. A transaction begins and commits, or begins and rolls back, by using the `begin`, `commit`, and `rollback` methods on the `Session` object.

Applications can also work in auto-commit mode, in which the `Session` automatically begins and commits a transaction whenever an operation is run on the map. Auto-commit mode cannot group multiple operations into a single transaction. Therefore, it is the slower option if you are creating a batch of multiple operations into a single transaction. However, for transactions that contain only one operation, auto-commit is simpler.

When your application is finished with the `Session`, use the optional `Session.close()` method to close the session. Closing the `Session` releases it from the heap and allows subsequent calls to the `getSession()` method to be reused, improving performance.

Important: The two-phase commit affects performance for data access. Use it only when absolutely necessary. Optimal performance is achieved by using one-phase commit interactions and designing the data schema to rely on that.



Deployment scenarios

WebSphere eXtreme Scale supports a wide array of deployment options to support different technical and business requirements, including the following environments:

- ▶ Stand-alone environment
- ▶ Integrated installation with WebSphere Application Server Network Deployment
- ▶ Integrated installation with WebSphere Application Server Liberty Profile

This chapter includes the following topics:

- ▶ 5.1, “WebSphere eXtreme Scale in a stand-alone environment” on page 152
- ▶ 5.2, “Integrating in a WebSphere Application Server Network Deployment environment” on page 178
- ▶ 5.3, “Integrating into WebSphere Application Server Liberty Profile” on page 230

5.1 WebSphere eXtreme Scale in a stand-alone environment

This section describes the deployment option of installing WebSphere eXtreme Scale into a stand-alone Java Platform, Standard Edition (JSE) environment. The 5.1.1, “Benefits” on page 152 and 5.1.2, “Limitations” on page 152 sections review the benefits and limitations of implementing WebSphere eXtreme Scale in a stand-alone environment. Consider these factors when you are deciding the best deployment option for your needs.

A sample getting started application (5.1.3, “Introducing the sample application” on page 153) is used as a simple JSE grid application that is deployed and tested over a stand-alone, multi-server WebSphere eXtreme Scale topology.

The rest of the sections deal with the following topics of implementing a stand-alone environment:

- ▶ 5.1.4, “Introducing the sample topology” on page 156
- ▶ 5.1.5, “Creating the sample topology” on page 158
- ▶ 5.1.6, “Validating the sample topology” on page 171
- ▶ 5.1.7, “Testing the sample topology and application” on page 174

5.1.1 Benefits

There are both benefits and limitations to any deployment scenario. By implementing a WebSphere eXtreme Scale stand-alone environment, you gain the following benefits:

- ▶ **Simplicity**

The stand-alone deployment option is the least complex installation and configuration option. The deployment is merely a basic run time of WebSphere eXtreme Scale libraries and a Java virtual machine.
- ▶ **Scale with ease**

Grid capacity can be increased by simply configuring more containers on existing hosts (vertical scaling) or on more hosts (horizontal scaling). WebSphere eXtreme Scale takes care of setting up the correct communication infrastructure, based on core groups according to the number of running containers.
- ▶ **Minimum memory requirements**

By operating a Java Standard Edition application, the amount of memory available to WebSphere eXtreme Scale run time to store objects in the grid instances is maximized.

5.1.2 Limitations

Consider these limitations when considering the stand-alone environment:

- ▶ **Administration and deployment**

The stand-alone environment does not provide specific features to support a centralized administration and deployment of grid-related configurations. System administrators are supposed to operate directly on grid hosts to deploy a new grid configuration or modify an existing one. However, WebSphere eXtreme Scale does provide the `xscmd` command to support some level of diagnostic and management capabilities. For more information, see 5.1.6, “Validating the sample topology” on page 171.
- ▶ **Java virtual machine**

WebSphere eXtreme Scale version 8.6 introduces the support for eXtremeIO (XIO) as an efficient and more inter-operable transport mechanism. XIO can be used with IBM and

non-IBM Java Runtime Environment (JRE) version 6 and 7 by supplying the WebSphere eXtreme Scale client library (`objectgridclient.jar`). However, for grid configuration based on RMI/IIOP as the transport protocol, more configurations should be implemented on the client side. For more information, see the *Configuring a custom Object Request Broker* topic of the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txscfgorb.html>

- ▶ Java Enterprise Edition managed resources

WebSphere eXtreme Scale in a stand-alone JSE environment does not have access to managed resources such as JDBC connection pooling mechanisms, JNDI calls, JCA adapters, and so on.

5.1.3 Introducing the sample application

One of the key application scenarios that are supported by WebSphere eXtreme Scale is the Application State Store pattern, where the grid is used to hold and manage application data.

WebSphere eXtreme Scale version 8.6 provides a ready to use, “getting started” sample application that implements such a pattern. The getting started application is typically used to verify the product installation. The provided code sample and start scripts are suited to implementing a simple grid topology that consists of one catalog server and one or more containers servers that run on the same host.

The getting started sample application is in the following directory:

<WXS_install_root>/ObjectGrid/gettingstarted

For more information about using this example, see the *Tutorial: Getting started with WebSphere eXtreme Scale* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsgetstart.html>

This section uses a slightly modified version of this getting started application that allows you to complete these tasks:

- ▶ Familiarize yourself with the WebSphere eXtreme Scale configuration and deployment procedures.
- ▶ Configure and run multiple catalog and container servers on different hosts.
- ▶ Run multiple grid clients concurrently to validate data consistency.

Application architecture and design

The getting started application (Figure 5-1 on page 154) consists of two main components:

- ▶ Grid client

A Java Standard Edition application, a *client*, that runs basic operations (get, insert, update, and delete) against data that is managed within a map called *Map1*. The client application uses ObjectMap APIs to interact with Map1. This application is packaged as a standard Java archive (JAR) called `GettingStartedClient.jar`.

- ▶ Grid

The grid provides a single, coherent, highly available, and scalable store for Map1.

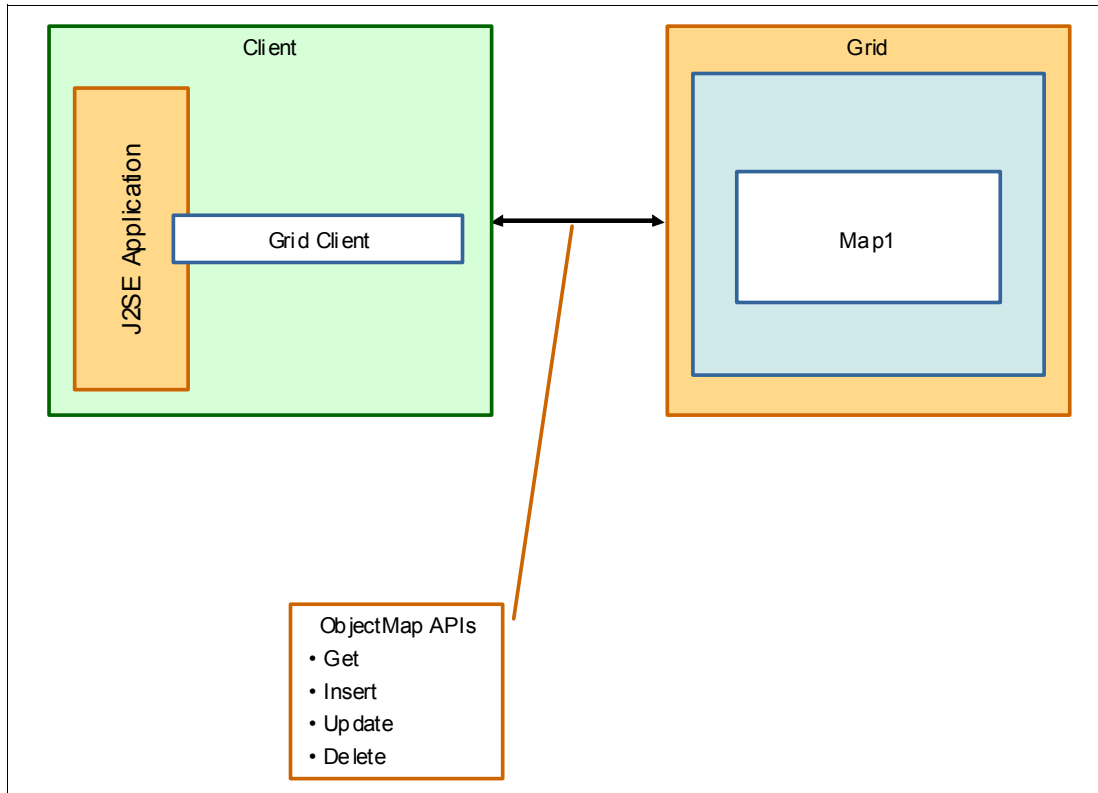


Figure 5-1 Getting started sample application

Sample application: The sample application and grid configuration files used in this section are provided in the STANDALONE folder of DeploymentScenarios.zip available as additional material for this Redbooks publication. For more information about downloading this material, see Appendix A, “Additional material” on page 341.

Client connection to the grid

In a stand-alone environment, connecting to the grid requires knowledge of the host name and ports of the catalog servers. Example 5-1 shows a code snippet that demonstrates how to establish a connection to the grid.

Example 5-1 Connecting to the grid

```
//The ObjectGrid instance is cached for the life of the application.
private ObjectGrid grid;

// Connect to the Catalog Server. The security and client override XML are not
// specified.
ClientClusterContext ccc =
ObjectGridManagerFactory.getObjectGridManager().connect(cep, null, null);

// Retrieve the ObjectGrid client connection and return it.
ObjectGrid grid =
ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc, gridName);
```

More coding examples can be found in Chapter 4, “Developing with WebSphere eXtreme Scale” on page 95.

Client interaction with the grid

The client interacts with Map1 in the grid using ObjectMap APIs to insert, update, retrieve, and remove data in the grid. You can use the *upsert(key, value)* method to insert or update data in a simplified way. If the key already exists, it is updated with the newly provided value. Otherwise, a new value is inserted into the grid.

Example 5-2 shows how to interact with Map1 using the ObjectMap APIs.

Example 5-2 Interacting with the grid using ObjectMap APIs

```
private String mapName = "Map1";
private ObjectGrid grid;
Session sess = grid.getSession();
ObjectMap map1 = sess.getMap(mapName);

map1.insert(k, v);
...
map1.upsert(k, v);
...
map1.update(k, v);
...
map1.remove(k);
...
Object value = (Object) map1.get(k);
```

Client transaction management

To protect the data grid from concurrent changes, clients can programmatically use transaction demarcation when inserting or updating data into the grid. The client can start a new transaction by issuing the *begin()* method on the *Session* object, and control transaction outcomes through *commit()* and *rollback()* methods.

By default, WebSphere eXtreme Scale uses a one-phase commit protocol that allows the client to include operations toward a single partition as part of a transaction, as shown in the Example 5-3.

Example 5-3 Single partition data operations

```
ObjectMap map1 = sess.getMap(mapName);
sess.setTxCommitProtocol(TxCommitProtocol.ONEPHASE);
sess.begin();
...
if(...)
    sess.commit();

else
    sess.rollback();
```

Clients can optionally select a two-phase commit protocol when modifying data managed within multiple partitions, as shown in Example 5-4.

Example 5-4 Multiple partitions data operations

```
ObjectMap map1 = sess.getMap(mapName);
sess.setTxCommitProtocol(TxCommitProtocol.TWOPHASE);
sess.begin();
...
```

```
if(...)
    sess.commit();

else
    sess.rollback();
```

Multi-partition transactions: Multi-partition transactions require XIO as the transport mechanism, and the configuration of a locking strategy (pessimistic or optimistic) on the backing maps. The following limitations apply to the WebSphere eXtreme Scale topology and programming models when you use multi-partition updates:

- ▶ Multi-master replication cannot be enabled.
- ▶ The Microsoft .NET client cannot manage multi-partition updates.
- ▶ BackingMaps configured with a loader plug-in cannot participate in a multi-partition transaction update.

Grid configuration

Example 5-5 shows the content of the `objectGrid.xml` file used to define the grid configuration. This configuration file is responsible for defining all the `objectGrid` and `backingMap` elements that make up the application data store. In this example, a grid called `Grid` contains a single backing map definition called `Map1`.

Example 5-5 Grid configuration within the `objectGrid.xml` file

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="Grid" txTimeout="30">
      <backingMap name="Map1" copyMode="COPY_TO_BYTES" lockStrategy="PESSIMISTIC"
        nullValuesSupported="false" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

5.1.4 Introducing the sample topology

The getting started application that is described in 5.1.3, “Introducing the sample application” on page 153 requires a topology on which to run. This section describes the sample topology created for this example to demonstrate WebSphere eXtreme Scale in a stand-alone environment.

Operational model

The basic operational model of the sample topology is shown in Figure 5-2. The sample topology includes four nodes (whost01, whost02, whost03, and whost04) that host client, catalog, and container services.

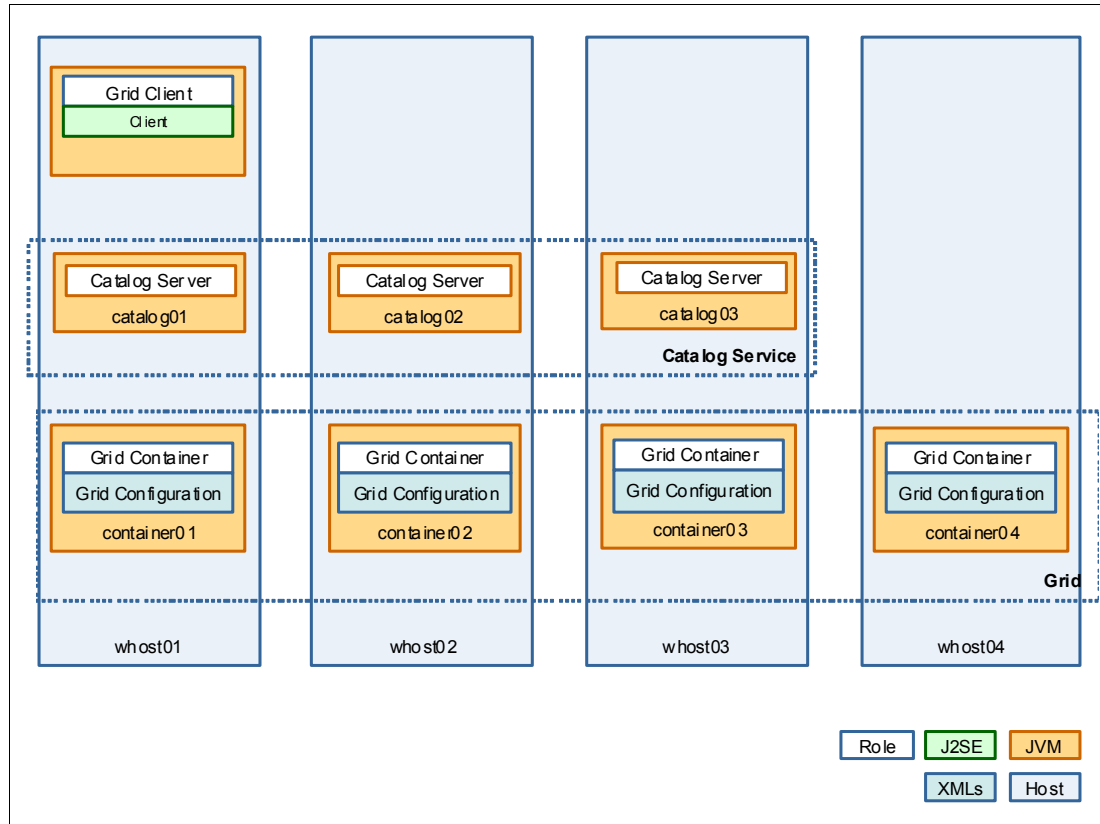


Figure 5-2 Sample topology for WebSphere eXtreme Scale in a stand-alone environment

Specifically, the topology has the following characteristics:

- ▶ The *catalog service* is supported by three catalog servers (catalog01, catalog02, and catalog03) deployed over three hosts (whost01, whost02, and whost03).
- ▶ The *grid* is supported by four container servers (container01, container02, container03, and container04) deployed over the four hosts (whost01, whost02, whost03, and whost04).
- ▶ The *grid client* is deployed on the host whost01.

Note for smaller-scale environments: If your test environment includes fewer hosts, you can still implement and test the sample topology. Either activate a reduced number of catalog or container servers, or co-locate multiple catalog and container servers on the same host if sufficient system resources are available.

Grid topology

In this example, the grid is operating within four container servers (container01, container02, container03, and container04). The grid is configured to manage thirteen partitions, with one optional synchronous replication for each primary shard.

A possible placement is shown in Figure 5-3. The actual shard placement can vary depending on the availability of container servers and placement decision made by the catalog server.

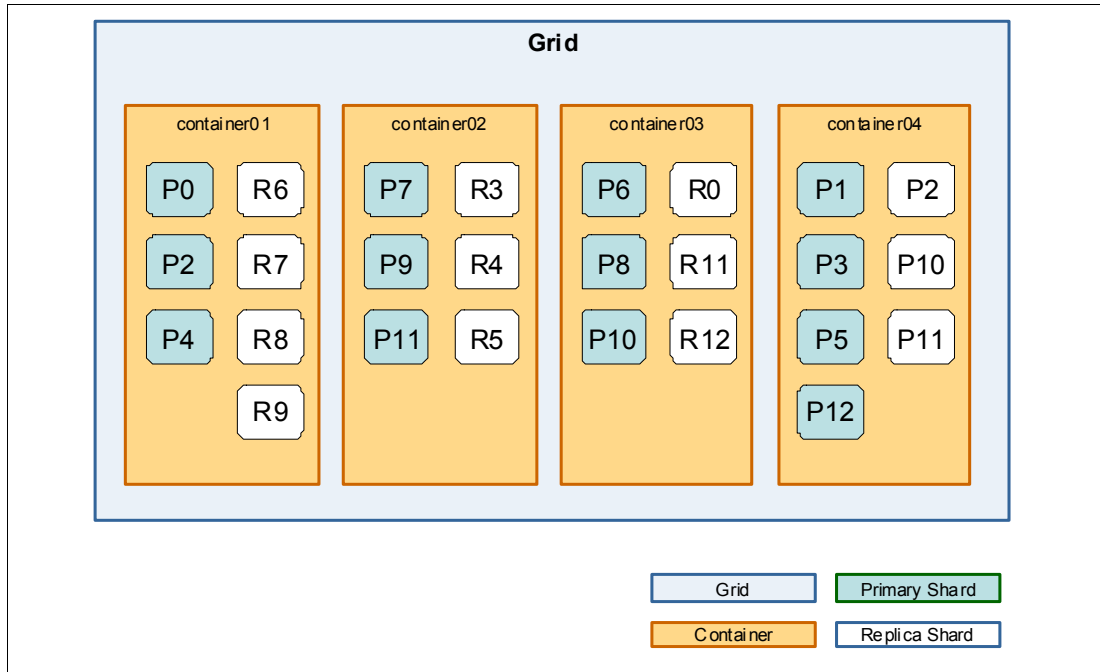


Figure 5-3 Grid topology for the stand-alone example scenario

This configuration is defined in the objectGridDeployment.xml file as shown in Example 5-6.

Example 5-6 Grid deployment configuration within the objectGridDeployment.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy../deploymentPolicy.xsd" xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Grid">
    <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0"
maxSyncReplicas="1" >
      <map ref="Map1"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

5.1.5 Creating the sample topology

This section describes the required steps to create the sample stand-alone topology described in 5.1.4, “Introducing the sample topology” on page 156.

Installing WebSphere eXtreme Scale

The installation process for a stand-alone environment consists of the following steps:

Remember: These steps must be repeated for every node in the topology.

1. Install WebSphere eXtreme Scale version 8.6.0.1 using IBM Installation Manager onto your supported platform of choice.
2. Configure the WebSphere eXtreme Scale 8.6.0.1 environment.

Installing WebSphere eXtreme Scale 8.6.0.1 using IBM Installation Manager

Install WebSphere eXtreme Scale by using the IBM Installation Manager, which is a common installation program that is used across many IBM software products.

Complete these steps to install WebSphere eXtreme Scale version 8.6.0.1:

1. Install the IBM Installation Manager, if not already present on the deployment nodes.

For each supported platform, IBM Installation Manager is included in the WebSphere product media in the following directory:

```
<WXS_product_media>/InstallationManager/agent.installer.<platform>
```

The installation instructions for the IBM Installation Manager can be found in the WebSphere eXtreme Scale Version 8.6 Information Center:

http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Ftxs_installation_dist_gui.html

2. Download the latest available WebSphere eXtreme Scale Fixpack (8.6.0.1 at the time of writing) from the IBM Fix Central at:

<http://www-933.ibm.com/support/fixcentral/>

3. Extract the Fixpack into a temporary directory, for example:

```
/tmp/WXS_8601_fixpack
```

4. Start the IBM Installation Manager by issuing the following command:

```
<IM_Installation_root>/IBMIM
```

5. The IBM Installation Manager needs to know the location of the WebSphere eXtreme Scale installation binary files. Click **File** → **Preferences**.
6. From the Repository view, click **Add Repositories**.

7. Click **Browse**, navigate to the WebSphere eXtreme Scale installation directory, and select **repository.config** as shown in Figure 5-4:
<WXS_product_media>/WXS_8600

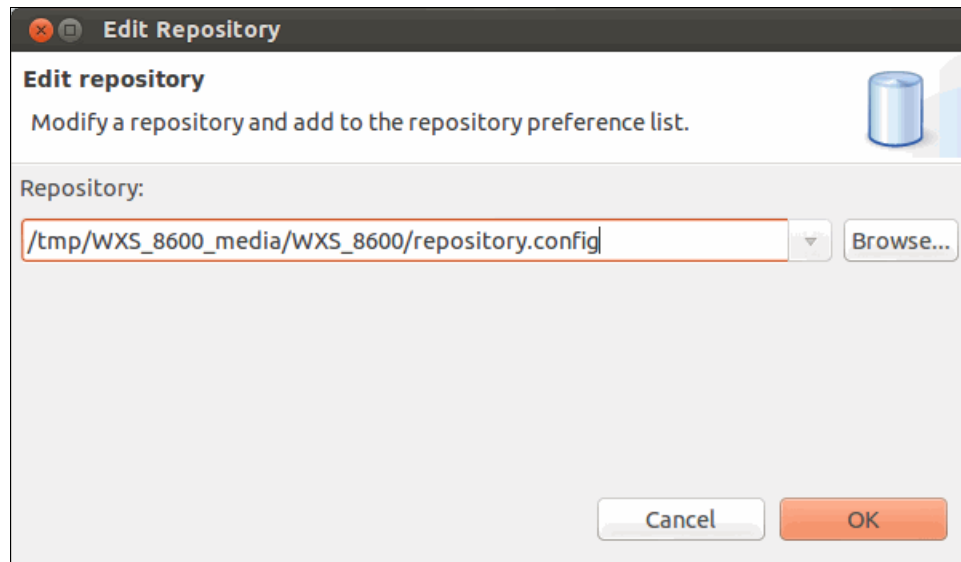


Figure 5-4 Selecting the WebSphere eXtreme Scale 8.6.0.0 repository location

- Repeat the steps 6 on page 159 and 7 on page 160 to add the repository for Fixpack1, /tmp/WXS_8601_fixpack/repository.config. Figure 5-5 shows both repositories have been added. Click **OK** to close the Preferences dialog box.

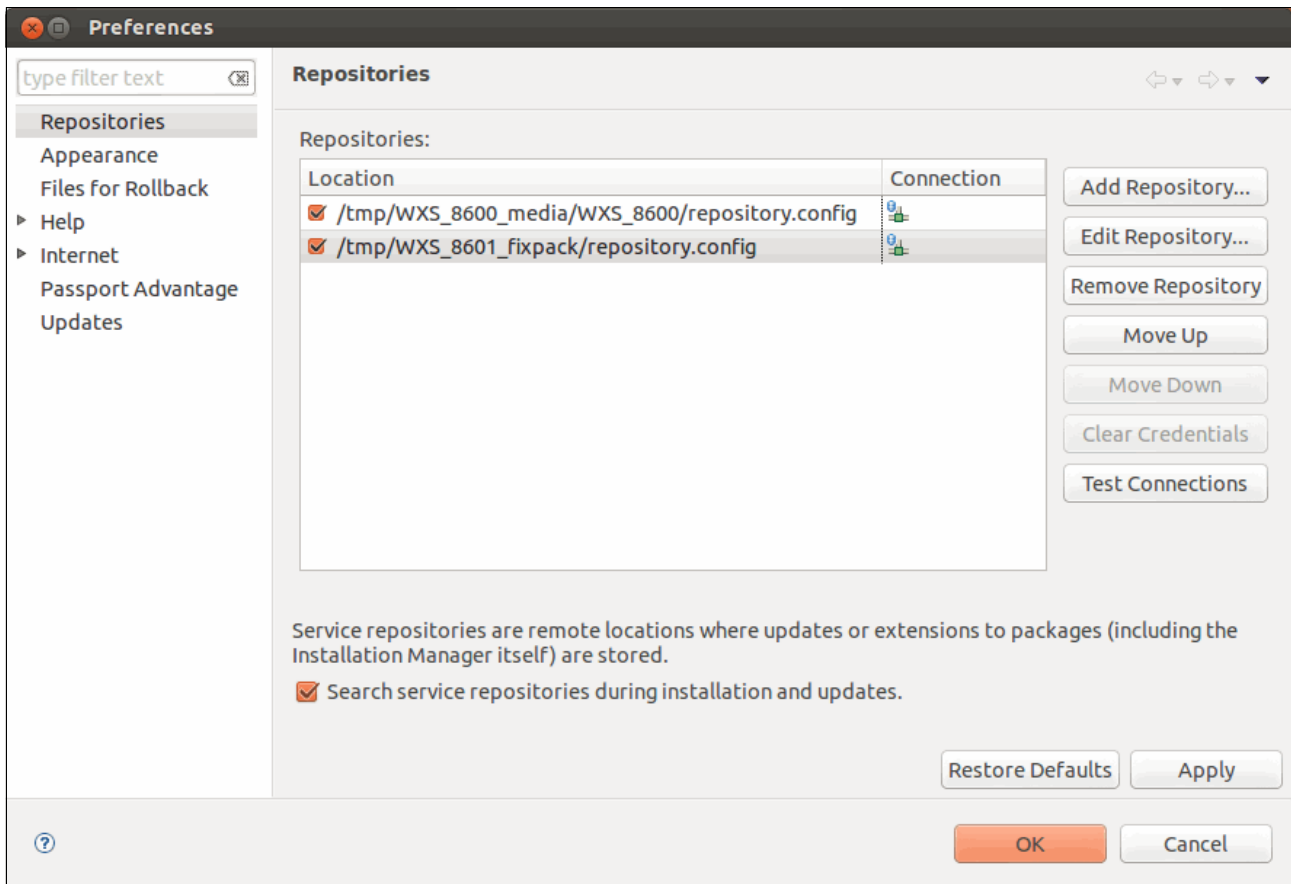


Figure 5-5 Selecting both WebSphere eXtreme Scale 8.6.0.0 and Fixpack 1 repositories

- From the main Installation Manager window, click **Install**.

10. On the Install Packages window, there are a range of installation options available. Select **IBM WebSphere eXtreme Scale in a stand-alone environment Version 8.6.0.1** installation package as shown in Figure 5-6. Click **Next**.

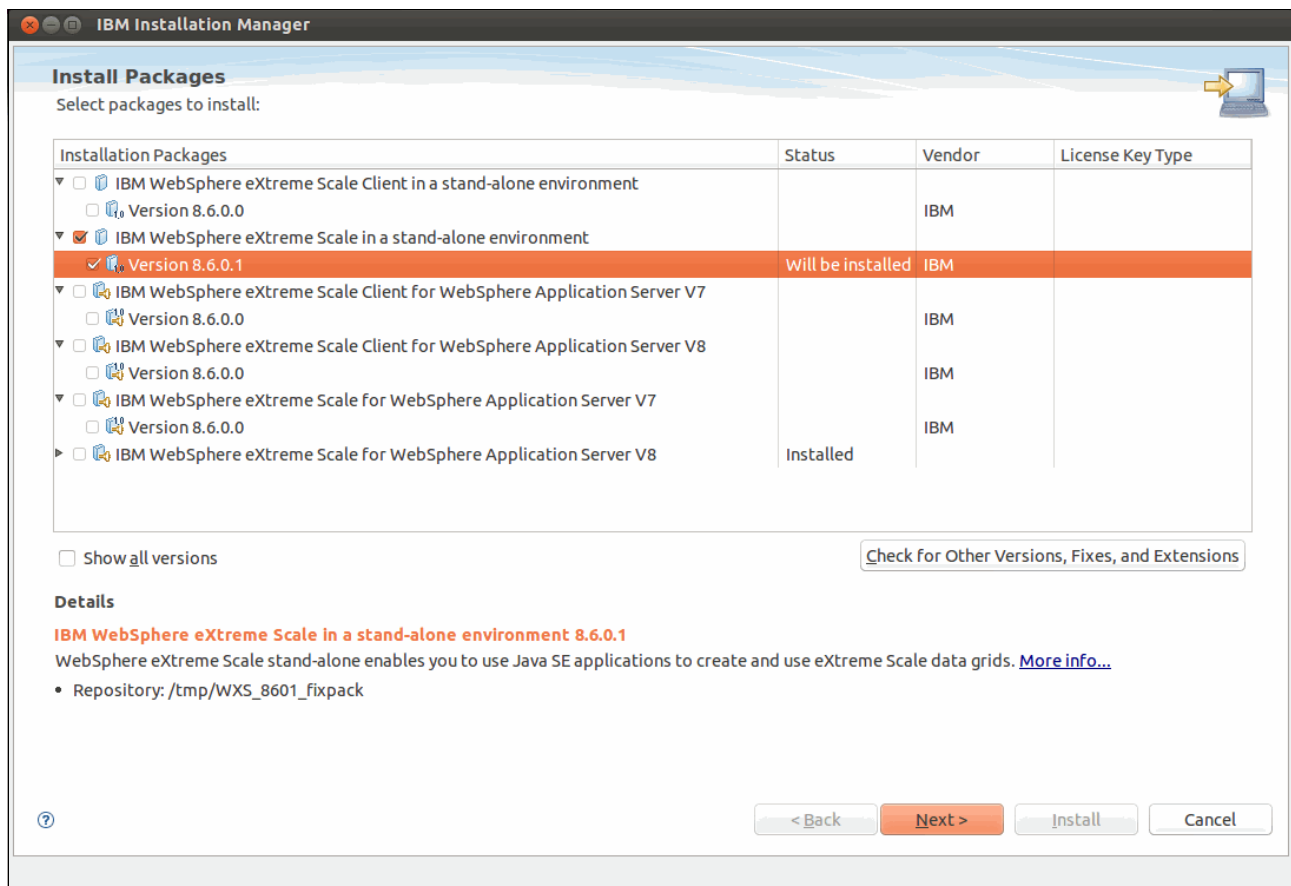


Figure 5-6 Selecting the IBM WebSphere eXtreme Scale in a stand-alone environment Version 8.6.0.1 package

11. Accept the software license agreement and click **Next**.

12. On the Install Packages window (Figure 5-7), keep the default option of **Create a new package group**. Specify the **Installation Directory** (/opt/IBM/WebSphere/XS86) and select **64-bit** for the architecture. Click **Next**.

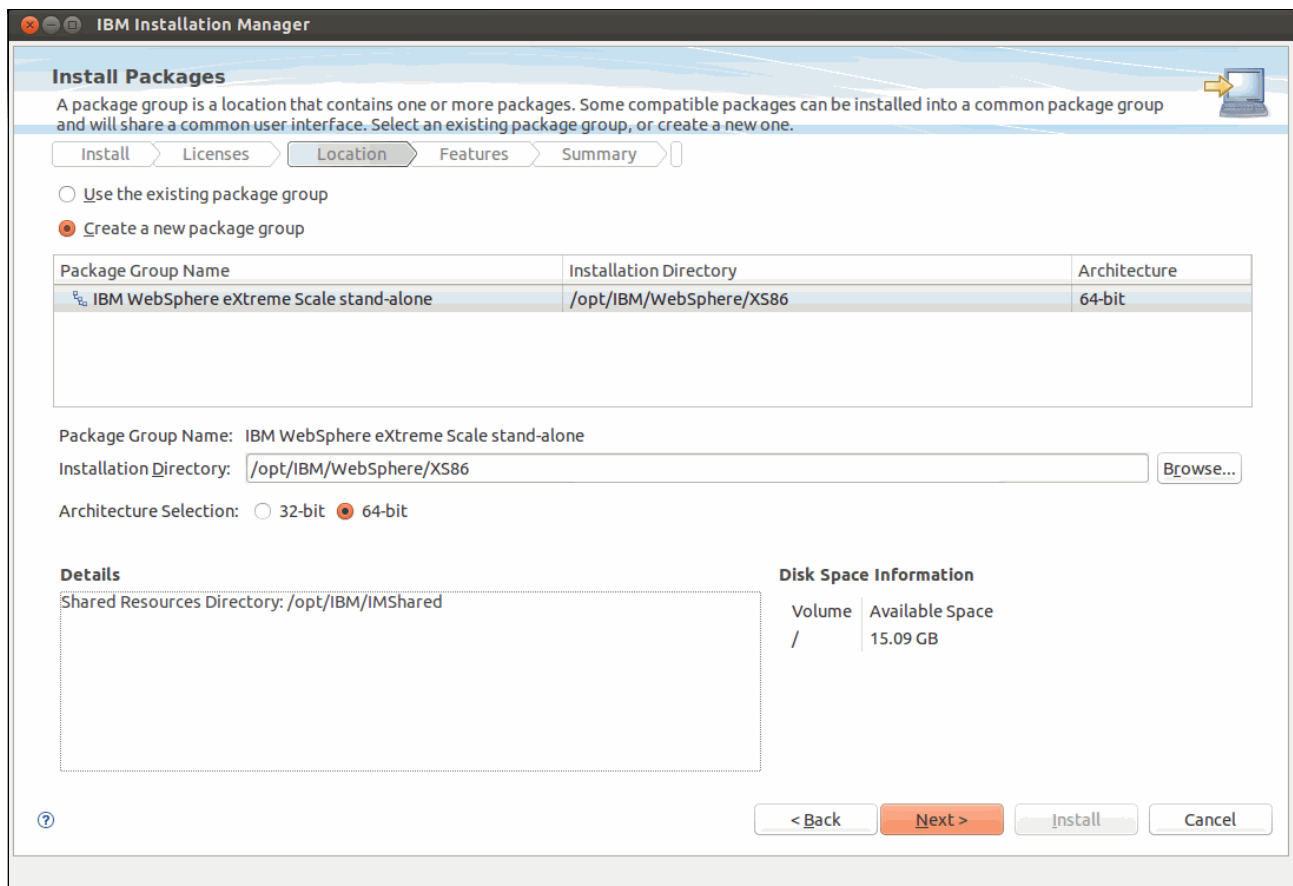


Figure 5-7 WebSphere eXtreme Scale package group, installation directory, and architecture selection

13. Select the **Samples** option, as shown in Figure 5-8. Select all the options and click **Next**.

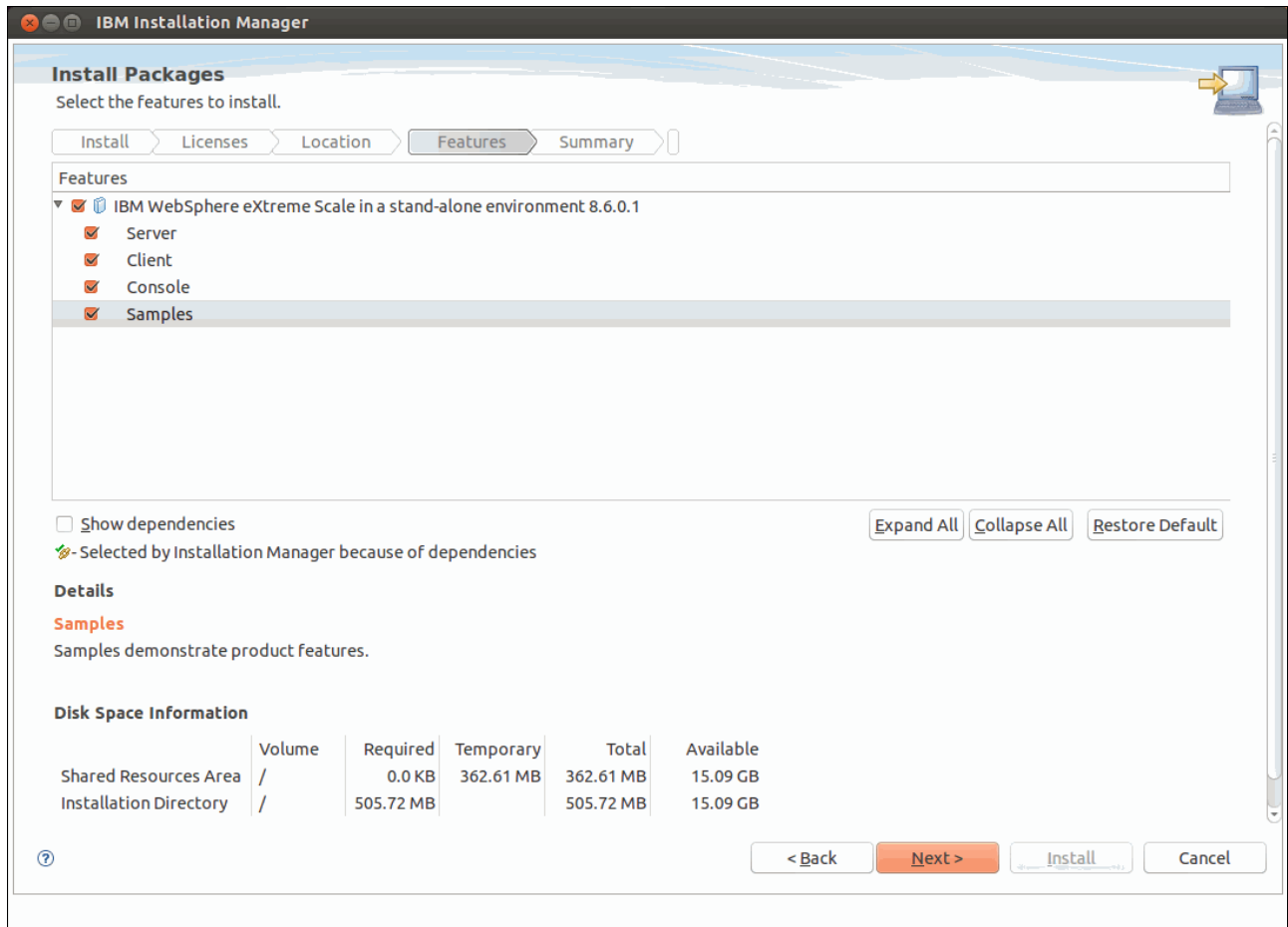


Figure 5-8 Selecting the WebSphere eXtreme Scale features to install

14. Review the installation summary and click **Install** to start the installation, as shown in Figure 5-9.

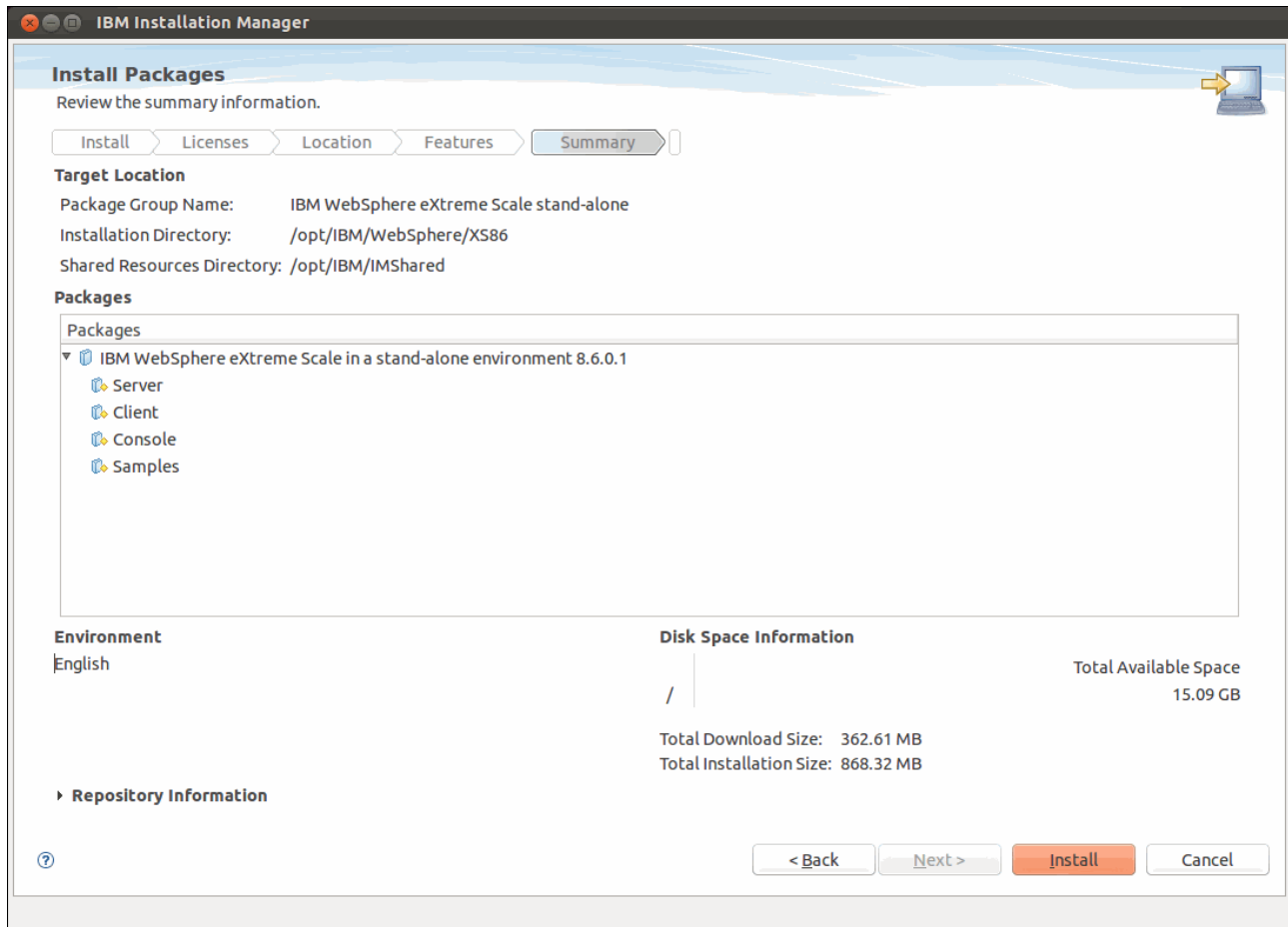


Figure 5-9 WebSphere eXtreme Scale 8.6.0.1 installation summary

15. After the installation completes successfully, click **Finish** and close the Installation Manager as shown in Figure 5-10.

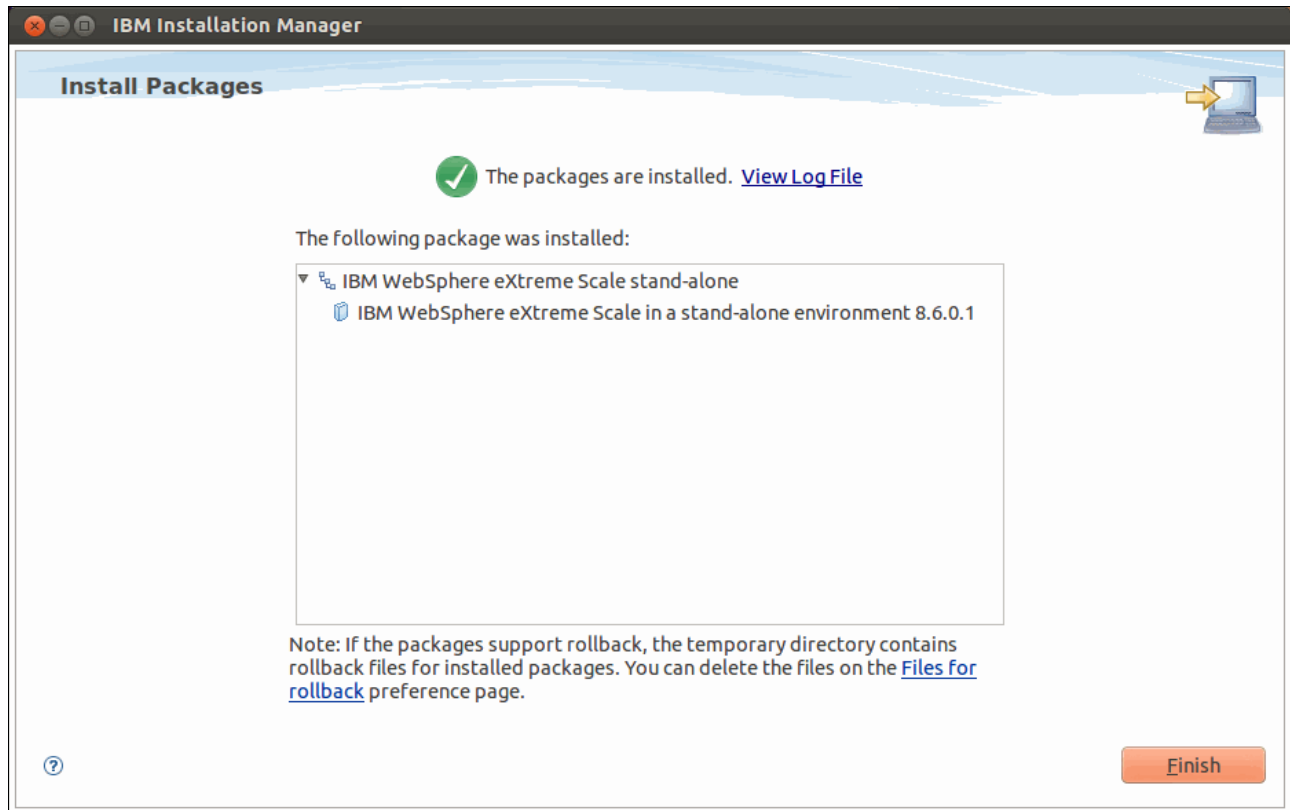


Figure 5-10 WebSphere eXtreme Scale 8.6.0.1 installation confirmation

Configuring the runtime environment

This section shows you how to configure and start the catalog servers and container server that comprise the sample stand-alone environment.

Starting the catalog servers

As described in "Operational model" on page 157, the sample topology includes three catalog servers as shown in Figure 5-11.

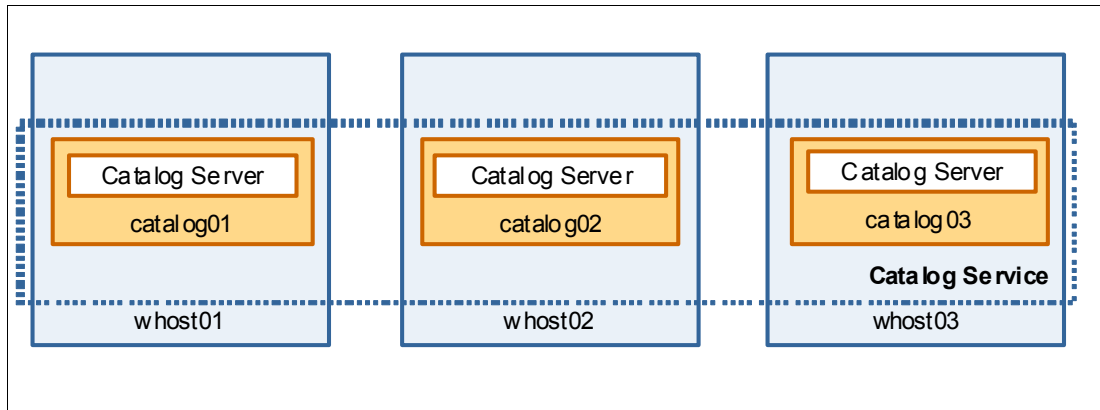


Figure 5-11 Catalog servers for the sample topology

The `startXsServer` script in the `<WXS_install_root>/ObjectGrid/bin` directory is used to start the catalog and container servers that use the IBM XIO transport mechanism.

The `startXsServer` script accepts an extensive set of options to configure catalog servers. For more information, see the *startXsServer script (XIO)* topic in the IBM WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Frxsstartxsserver.html>

For the sample topology, start the catalog server by using the options that are shown in Example 5-7.

Example 5-7 startXsServer script options to start the catalog servers

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh <server_name> [with the
following options]
  -catalogServiceEndpoints <server:host:clientPort:peerPort,...>
  -listenerHost <hostname>
  -listenerPort <port>
  -JMXServicePort <port>
```

Table 5-1 provides a summary of the TCP/IP port configuration that is defined for the three catalog servers.

Table 5-1 Catalog server TCP/IP port configuration

Catalog server	Host name	XIO listener port	Client port, Peer port	JMX port
catalog01	whost01.rtp.ibm.com	2809	6600, 6601	1099
catalog02	whost02.rtp.ibm.com	2809	6600, 6601	1099
catalog03	whost03.rtp.ibm.com	2810	6602, 6603	1100

Following are the default values for the XIO listener, client, and peer port numbers:

- ▶ XIO listener port: 2809
- ▶ Client port: 6600
- ▶ Peer port: 6601
- ▶ JMX port: 1099

Generally, explicitly set port numbers as part of the catalog server script to have better control and understanding of TCP/IP ports that are used for intra-process communications.

The `catalogServiceEndpoints` parameter passed to the `startXsServer` script to start a catalog server must have the same value for all the catalog servers. In the sample topology, this value is the following:

```
catalog01:whost01.rtp.ibm.com:6600:6601,catalog02:whost02.rtp.ibm.com:6600:6601,
catalog03:whost03.rtp.ibm.com:6602:6603
```

Smaller-scale environments: If your test environment includes fewer catalog servers, the configuration procedure in this section is still valid. When you start the catalog server, check that the provided value for the `catalogServiceEndpoints` is appropriate for the reduced catalog service.

Complete the following steps to start the catalog service:

1. Start catalog01 on the whost01 host by issuing the command shown in the Example 5-8.

Example 5-8 Starting the catalog01 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh catalog01 -catalogServiceEndPoints  
catalog01:whost01.rtp.ibm.com:6600:6601,catalog02:whost02.rtp.ibm.com  
:6600:6601,catalog03:whost03.rtp.ibm.com:6602:6603 -listenerHost  
whost01.rtp.ibm.com -listenerPort 2809 -JMXServicePort 1099
```

2. Start catalog02 on the whost02 host by issuing the command shown in Example 5-9.

Example 5-9 Starting the catalog02 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh catalog02  
-catalogServiceEndPoints  
catalog01:whost01.rtp.ibm.com:6600:6601,catalog02:whost02.rtp.ibm.com:6600:6601  
,catalog03:whost03.rtp.ibm.com:6602:6603 -listenerHost whost02.rtp.ibm.com  
-listenerPort 2809 -JMXServicePort 1099
```

3. Start catalog03 on the whost03 host by issuing the command shown in Example 5-10.

Example 5-10 Starting the catalog03 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh catalog03  
-catalogServiceEndPoints  
catalog01:whost01.rtp.ibm.com:6600:6601,catalog02:whost02.rtp.ibm.com:6600:6601  
,catalog03:whost03.rtp.ibm.com:6602:6603 -listenerHost whost03.rtp.ibm.com  
-listenerPort 2810 -JMXServicePort 1100
```

Starting all the catalog servers in parallel: The catalog service is a critical infrastructure to the grid. It is based on grid technology itself, using the grid replication mechanisms to ensure data availability. The internal partitioning configuration requires at least one replica to be available. When only a single cluster member is started, this rule cannot be established and the catalog server cannot start.

You can check that each catalog server has successfully started by checking that SystemOut.log contains the message that is shown in the Example 5-11 for catalog01.

Example 5-11 Message indicating a successful catalog server start for catalog01

```
[5/3/13 14:09:38:121 EDT] 00000001 ServerImpl I CWOBJ1001I: ObjectGrid Server  
catalog01 is ready to process requests.
```

One of the catalog servers is elected to be the master. The master catalog server generates the message shown in the Example 5-12.

Example 5-12 Message indicating that the catalog server is elected to be the master

```
[5/3/13 14:09:37:194 EDT] 00000001 CatalogService I CWOBJ8106I: The master  
catalog service cluster activated with cluster DefaultDomain
```

Example 5-13 shows the messages you see when the catalog service cannot start because of a missing peer catalog server.

Example 5-13 A catalog server cannot start without all cluster members

```
[5/3/13 14:25:47:109 EDT] 00000001 ServerRuntime I   CWOBJ8401I: The system is
waiting for a server replica to be started.
...
5/3/13 14:25:47:820 EDT] 00000065 ServerService W   CWOBJ1668W: A client is making
a request to a server that has not completely started.
```

The catalog server start fails if a peer catalog server does not become available within a predefined window (default is 300 seconds). Example 5-14 shows the message that is logged by the startXsServer script.

Example 5-14 Catalog server start failure due to missing cluster members

```
[5/3/13 14:30:32:328 EDT] 00000016 ProcessLaunchE CWOBJ2509E: Timedout after
waiting 300 seconds for the ObjectGrid server to start.
```

Starting the container JVMs

As described in “Operational model” on page 157, the sample topology includes four container servers as shown in Figure 5-12.

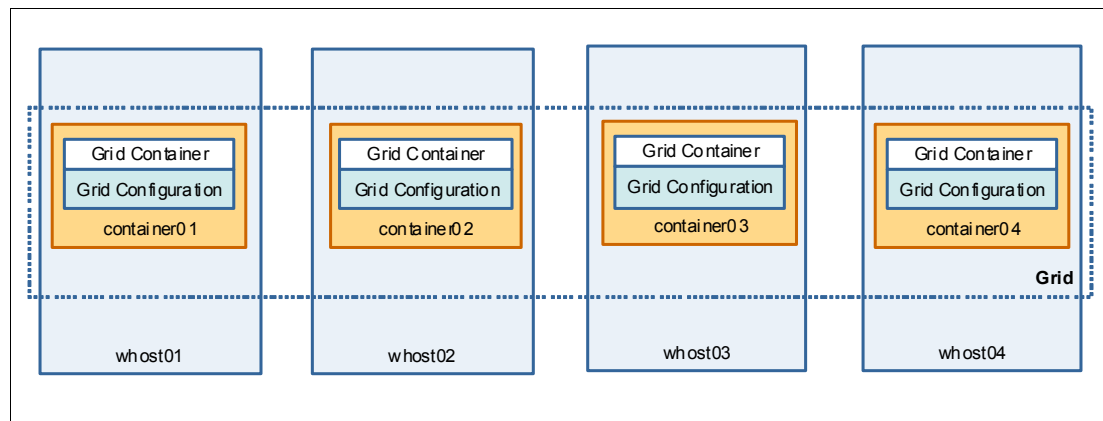


Figure 5-12 Container servers for the sample topology

For the sample topology, start the container servers by using the options shown in Example 5-15.

Example 5-15 startXsServer script options to start container servers

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh <server_name> [with the
following options]
-catalogServiceEndpoints <host:listeningPort,...>
-objectgridFile <relative or absolute path to objectGrid.xml file>
-deploymentPolicyFile <relative or absolute path to objectGridDeployment.xml>
```

The catalogServiceEndpoints parameter passed to the startXsServer script to start a container server must have the same value for all the container servers. In the sample topology, this value is the following:

```
whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809,whost03.rtp.ibm.com:2810
```

Smaller-scale environments: If your test environment includes fewer container servers, the configuration procedure in this section is still valid. When you start the container server, check that the provided value for `catalogServiceEndPoints` parameter is consistent with the catalog service previously configured.

Complete the following steps to start the container servers:

1. Start container01 on whost01 by issuing the command shown in the Example 5-16.

Example 5-16 Starting the container01 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh container01
-catalogServiceEndPoints
whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809,whost03.rtp.ibm.com:2810
-objectgridFile xml/objectGrid.xml -deploymentPolicyFile
xml/objectGridDeployment.xml
```

2. Start container02 on whost02 by issuing the command shown in the Example 5-17.

Example 5-17 Starting the container02 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh container02
-catalogServiceEndPoints
whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809,whost03.rtp.ibm.com:2810
-objectgridFile xml/objectGrid.xml -deploymentPolicyFile
xml/objectGridDeployment.xml
```

3. Start container03 on whost03 by issuing the command shown in the Example 5-18.

Example 5-18 Starting the container03 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh container03
-catalogServiceEndPoints
whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809,whost03.rtp.ibm.com:2810
-objectgridFile xml/objectGrid.xml -deploymentPolicyFile
xml/objectGridDeployment.xml
```

4. Start container04 on whost04 by issuing the command shown in the Example 5-19.

Example 5-19 Starting the container04 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh container04
-catalogServiceEndPoints
whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809,whost03.rtp.ibm.com:2810
-objectgridFile xml/objectGrid.xml -deploymentPolicyFile
xml/objectGridDeployment.xml
```

Example 5-20 shows the message reported by `startXsServer` script upon a successful start of a container server,

Example 5-20 Message indicating a successful container server start

```
[5/3/13 15:13:40:481 EDT] 00000001 ServerImpl I CWOBJ1001I: ObjectGrid Server
container01 is ready to process requests.
```

5.1.6 Validating the sample topology

The `xscmd` command can be used to validate the operating status for the grid. The `xscmd` script must be run from the node directory by using the options shown in Example 5-21.

Example 5-21 `xscmd` options

```
<WXS_install_root>/ObjectGrid/bin/xscmd.sh [with the following options]
  -cep <host:XIO_listening_port,...>
  -c <command>
```

Multiple catalog service endpoints can be passed to `xscmd` using the `-cep` parameter. The `xscmd` command tries to establish a connection to the first available catalog server as specified in the list passed as the `-cep` argument. Although this feature is useful to improve the resiliency of the `xscmd` command, this section addresses the following commands that are issued against a single catalog server (catalog01):

- ▶ `showInfo`
- ▶ `showPlacement`
- ▶ `routetable`

showInfo

The `showInfo` command is useful to validate that the running WebSphere eXtreme services are consistent with the topology to be implemented. Example 5-22 shows the output of `showInfo` that contains information about the environment specifications, including the installed product version and JVM information.

Example 5-22 `showInfo` command output

```
<WXS_install_root>/ObjectGrid/bin/xscmd.sh -cep whost01.rtp.ibm.com:2809 -c
showInfo
*** Server Name: catalog01 ***
Client Port                6600 (configured)
Domain name                DefaultDomain
Host name                  whost01.rtp.ibm.com
IP Address                 9.42.171.46
JMX Connector Port        1099 (derived)
JMX Service Port          1099 (configured)
JMX Service URL           service:jmx:rmi://9.42.171.46:
                          1099/jndi/rmi://9.42.171.46:
                          1099/objectgrid/MBeanServer
JAVA Directory             /opt/IBM/WebSphere/XS86/java/jre
JAVA Vendor                IBM Corporation
JAVA Version               1.7.0
JAVA Bit Mode              64
JVM Runtime Version        pxa6470sr2ifx-20121102_01
                          (SR2+IV31201)
JVM Version                JRE 1.7.0 Linux amd64-64
                          20121031_126871 (JIT enabled, AOT
                          enabled), J9VM -
                          R26_Java726_SR2_iFix_3_20121031_1658
                          _B126871, JIT -
                          r11.b01_20120808_24925_ifx2, GC -
                          R26_Java726_SR2_iFix_3_20121031_1658
                          _B126871, J9CL - 20121031_126871
Listener Port (XIO)        2809 (configured)
```

```

OS Architecture          amd64
Operating System        Linux
Operating System Version 3.5.0-17-generic
Process ID              16202
Peer Port               6601 (configured)
Server Type             Catalog Server
Time Stamp from Server  05/03/13 15:23:22.096 EDT
Transport               eXtremeIO
WebSphere eXtreme Scale Product Directory /opt/IBM/WebSphere/XS86/ObjectGrid
WebSphere eXtreme Scale Version v7.0.0 (8.6.0.0) [cf11305.31182928]
XIO Timeout (seconds)  30 (default)

```

```

*** Server Name: catalog02 ***
...
...
*** Server Name: catalog03 ***
...
*** Server Name: container01 ***
...
...
*** Server Name: container03 ***
...
...
*** Server Name: container02 ***
...
*** Server Name: container04 ***
...
...

```

showPlacement

The **showPlacement** command returns information about the list of all containers and shards deployed on the hosts that comprise the grid. Such information is a representation of the primary and replica shard topology as *planned* by the catalog service. Example 5-23 reports an example of placement information across all four containers (container01 to container04).

Example 5-23 showPlacement command output

```

<WXS_install_root>/ObjectGrid/bin/xscmd.sh -cep whost01.rtp.ibm.com:2809 -c
showPlacement

```

```

CWXSIO068I: Executing command: showPlacement
*** Show all online container servers for Grid data grid and mapSet map set.
Host: whost01.rtp.ibm.com

```

```

Container: container01_C-1, Server:container01, Zone:DefaultZone

```

Partition	Shard Type	Reserved
1	Primary	false
3	Primary	false
7	Primary	false
8	SynchronousReplica	false
9	SynchronousReplica	false
10	SynchronousReplica	false
11	SynchronousReplica	false

```
Host: whost02.rtp.ibm.com
Container: container02_C-1, Server:container02, Zone:DefaultZone
Partition Shard Type          Reserved
-----
0          Primary            false
4          Primary            false
6          Primary            false
5          SynchronousReplica false
7          SynchronousReplica false
12         SynchronousReplica false
```

```
Host: whost03.rtp.ibm.com
Container: container03_C-0, Server:container03, Zone:DefaultZone
Partition Shard Type          Reserved
-----
2          Primary            false
5          Primary            false
8          Primary            false
9          Primary            false
0          SynchronousReplica false
1          SynchronousReplica false
3          SynchronousReplica false
```

```
Host: whost04.rtp.ibm.com
Container: container04_C-0, Server:container04, Zone:DefaultZone
Partition Shard Type          Reserved
-----
10         Primary            false
11         Primary            false
12         Primary            false
2          SynchronousReplica false
4          SynchronousReplica false
6          SynchronousReplica false
```

```
Number of containers matching = 4
Total known containers       = 4
Total known hosts            = 4
```

routetable

The **routetable** command is used to display the routing table information managed by the catalog service and pushed to the clients (Example 5-24). This is a representation of the *actual* placement of primary and replica shards across available containers. This placement can differ, during placement operations, from the information returned from the **showPlacement** command, which shows the *planned* placement.

The **routetable** command also provides useful information about the status of primary and replica shards. In fact, the **xscmd** command uses the routing table to check whether partitions are reachable or not from the workstation where the **xscmd** program is.

Example 5-24 routetable command output

```
<WXS_install_root>/ObjectGrid/bin/xscmd.sh -cep whost01.rtp.ibm.com:2809 -c routetable
Starting at: 2013-05-03 17:00:55.368
CWXSI0068I: Executing command: routetable
*** Displaying routing information for data grid: Grid:mapSet
```

```

Placement Scope: Domain
Primary          0      reachable whost02.rtp.ibm.com DefaultZone container02_C-1
SynchronousReplica 0    reachable whost03.rtp.ibm.com DefaultZone container03_C-0
Primary          1      reachable whost01.rtp.ibm.com DefaultZone container01_C-1
SynchronousReplica 1    reachable whost03.rtp.ibm.com DefaultZone container03_C-0
Primary          2      reachable whost03.rtp.ibm.com DefaultZone container03_C-0
SynchronousReplica 2    reachable whost04.rtp.ibm.com DefaultZone container04_C-0
Primary          3      reachable whost01.rtp.ibm.com DefaultZone container01_C-1
SynchronousReplica 3    reachable whost03.rtp.ibm.com DefaultZone container03_C-0
Primary          4      reachable whost02.rtp.ibm.com DefaultZone container02_C-1
SynchronousReplica 4    reachable whost04.rtp.ibm.com DefaultZone container04_C-0
Primary          5      reachable whost03.rtp.ibm.com DefaultZone container03_C-0
SynchronousReplica 5    reachable whost02.rtp.ibm.com DefaultZone container02_C-1
Primary          6      reachable whost02.rtp.ibm.com DefaultZone container02_C-1
SynchronousReplica 6    reachable whost04.rtp.ibm.com DefaultZone container04_C-0
Primary          7      reachable whost01.rtp.ibm.com DefaultZone container01_C-1
SynchronousReplica 7    reachable whost02.rtp.ibm.com DefaultZone container02_C-1
Primary          8      reachable whost03.rtp.ibm.com DefaultZone container03_C-0
SynchronousReplica 8    reachable whost01.rtp.ibm.com DefaultZone container01_C-1
Primary          9      reachable whost03.rtp.ibm.com DefaultZone container03_C-0
SynchronousReplica 9    reachable whost01.rtp.ibm.com DefaultZone container01_C-1
Primary         10      reachable whost04.rtp.ibm.com DefaultZone container04_C-0
SynchronousReplica 10    reachable whost01.rtp.ibm.com DefaultZone container01_C-1
Primary         11      reachable whost04.rtp.ibm.com DefaultZone container04_C-0
SynchronousReplica 11    reachable whost01.rtp.ibm.com DefaultZone container01_C-1
Primary         12      reachable whost04.rtp.ibm.com DefaultZone container04_C-0
SynchronousReplica 12    reachable whost02.rtp.ibm.com DefaultZone container02_C-1

```

CWXSIO040I: The routetable command completed successfully.

5.1.7 Testing the sample topology and application

WebSphere eXtreme Scale clients connect to the catalog service, retrieve a description of the server topology, and then communicate directly to each container server as needed. When the server topology changes because new servers are added or existing servers fail, the client is automatically routed to the appropriate server that is hosting the data.

Figure 5-13 shows the sample topology that is used in the example.

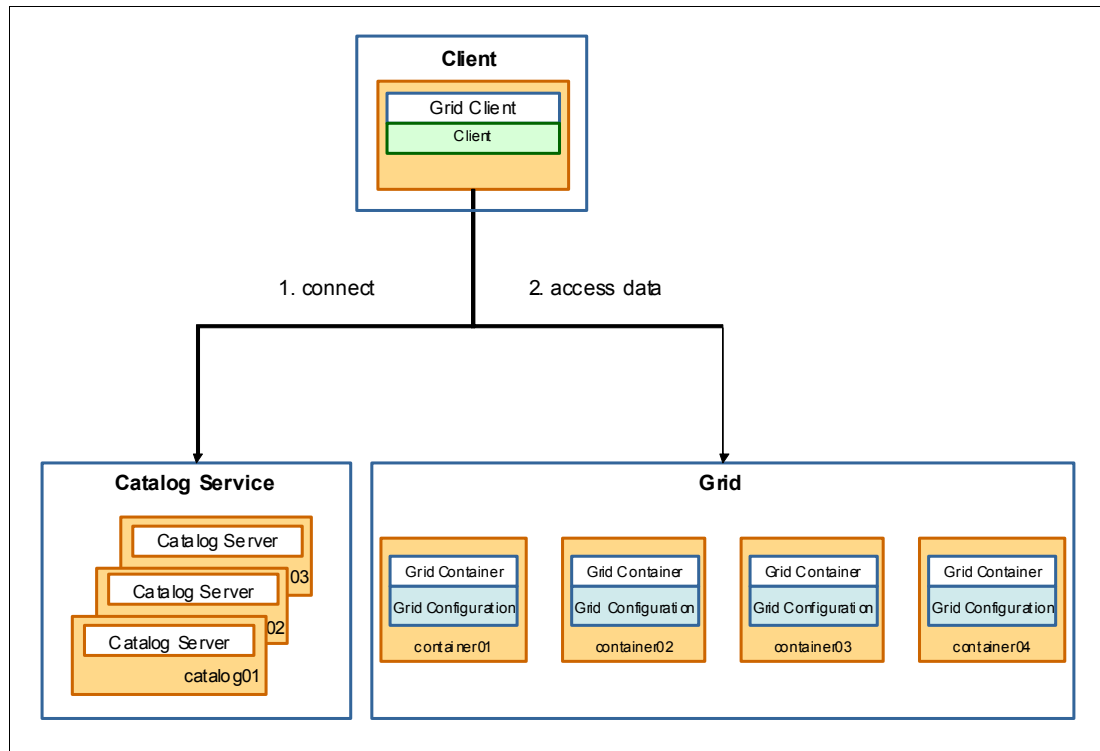


Figure 5-13 Sample topology grid client, catalogs, and containers

Test the grid by using the client program that is supplied as part of the getting started application. This client is useful to insert, update, and delete data from the newly deployed grid with the option to have transactional control over grid operations. You can start the client by issuing the command shown in the Example 5-25.

Example 5-25 Starting the client program

```
java -cp
./lib/GettingStartedClient.jar:<WXS_install_root>/ObjectGrid/lib/ogclient.jar
com.ibm.websphere.xs.sample.gettingstarted.Client whost01.rtp.ibm.com:2809
...
...
Connected to ObjectGrid: Grid

Entercommand: <[exit | [begin | begin2pc | commit | rollback] | [n<number>] |
<[i(nsert) | g(et) | u(pdate) | (u)p(sert) | d(elete)]> <key> [<value>]
```

After you start the client, complete the following steps to test the grid:

1. Add an entry into the grid by issuing the **i <key> value** command as shown Example 5-26.

Example 5-26 Inserting a new entry into the grid

```
> i key1 value1
SUCCESS: Inserted TestValue [value=value1] with key TestKey [key=key1],
partitionId=3
```

2. Retrieve an entry from the grid. You can check that an entry is in the grid and retrieve its associated value by issuing the **g <key>** command as shown in Example 5-27.

Example 5-27 Inserting a new entry into the grid

```
> g key1  
Value is TestValue [value=value1], partitionId=3
```

3. Update an entry into the grid. You can update an existing entry in the grid by issuing the **u <key> <value>** command as shown in Example 5-28.

Example 5-28 Updating an existing entry in the grid

```
> u key1 value11  
SUCCESS: Updated key TestKey [key=key1] with value TestValue [value=value11],  
partitionId=3
```

4. Delete an entry from the grid. You can delete an existing entry from the grid by issuing the **d <key>** command as shown in Example 5-29.

Example 5-29 Deleting an entry from the grid

```
> d key1  
SUCCESS: Deleted value with key TestKey [key=key1], partitionId=3
```

5. Insert two entries using multi-partition transaction support. Multiple entries can be inserted into the grid by taking advantage of the new multi-partition transaction support that was introduced in WebSphere eXtreme Scale version 8.6. For more information, see 4.8.3, “Multi-partition transactions” on page 148.
 - a. Begin a new transaction by issuing the **begin2pc** command as shown in Example 5-30.

Example 5-30 Starting a new multi-partition transaction

```
> begin2pc  
SUCCESS: 2-Phase capable, multi-partition transaction started.
```

- b. Insert the first entry into the grid by issuing the **i key1 value1** command as shown in Example 5-31.

Example 5-31 Inserting the first entry as part of a multi-partition transaction

```
> i key1 value1  
SUCCESS: Inserted TestValue [value=value1] with key TestKey [key=key1],  
partitionId=3
```

- c. Insert the second entry into the grid by issuing the **i key2 value2** command as shown in Example 5-32.

Example 5-32 Inserting the second entry as part of a multi-partition transaction

```
> i key2 value2  
SUCCESS: Inserted TestValue [value=value2] with key TestKey [key=key2],  
partitionId=2
```

- d. Commit the multi-partition transaction by issuing the **commit** command as shown in Example 5-33.

Example 5-33 Inserting both entries as part of a multi-partition transaction

```
> commit
SUCCESS: Transaction committed.
```

Note: Each transaction needs to be complete within the default time-out, specified as part of grid configuration (objectGrid.xml). In this scenario, the time-out is 30 seconds. In case the multi-partition transaction is not completed within 30 seconds, the transaction will be eventually rolled-back due to a time-out condition.

- e. Exit from the getting started client application by issuing the **exit** command as shown in Example 5-34.

Example 5-34 Exiting from the getting started client application

```
> exit
Exiting...
```

6. Display the partition information. You can verify the distribution of entries (key1 and key2) into the Map1 partitions and display the partition size by using the **xscmd -c showMapSizes** command as shown in Example 5-35.

Example 5-35 showMapSizes command output

```
<WXS_install_root>/ObjectGrid/bin/xscmd.sh -cep
whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809,whost03.p.ibm.com:2810 -c
showMapSizes
Starting at: 2013-05-06 12:04:17.009
```

```
CWXSIO068I: Executing command: showMapSizes
*** Displaying results for Grid data grid and mapSet map set.
```

```
*** Listing maps for container01 ***
```

Map Name	Partition	Map Entries	Used Bytes	Shard Type	Container
Map1	1	0	0	Primary	container01_C-1
Map1	3	1	392 B	Primary	container01_C-1
Map1	7	0	0	Primary	container01_C-1
Map1	8	0	0	SynchronousReplica	container01_C-1
Map1	9	0	0	SynchronousReplica	container01_C-1
Map1	10	0	0	SynchronousReplica	container01_C-1
Map1	11	0	0	SynchronousReplica	container01_C-1

Server total: 1 (392 B)

```
*** Listing maps for container04 ***
```

Map Name	Partition	Map Entries	Used Bytes	Shard Type	Container
Map1	2	1	392 B	SynchronousReplica	container04_C-0
Map1	4	0	0	SynchronousReplica	container04_C-0
Map1	6	0	0	SynchronousReplica	container04_C-0
Map1	10	0	0	Primary	container04_C-0
Map1	11	0	0	Primary	container04_C-0
Map1	12	0	0	Primary	container04_C-0

Server total: 1 (392 B)

*** Listing maps for container02 ***

Map Name	Partition	Map Entries	Used Bytes	Shard Type	Container
Map1	0	0	0	Primary	container02_C-1
Map1	4	0	0	Primary	container02_C-1
Map1	5	0	0	SynchronousReplica	container02_C-1
Map1	6	0	0	Primary	container02_C-1
Map1	7	0	0	SynchronousReplica	container02_C-1
Map1	12	0	0	SynchronousReplica	container02_C-1

Server total: 0 (0 B)

*** Listing maps for container03 ***

Map Name	Partition	Map Entries	Used Bytes	Shard Type	Container
Map1	0	0	0	SynchronousReplica	container03_C-0
Map1	1	0	0	SynchronousReplica	container03_C-0
Map1	2	1	392 B	Primary	container03_C-0
Map1	3	1	392 B	SynchronousReplica	container03_C-0
Map1	5	0	0	Primary	container03_C-0
Map1	8	0	0	Primary	container03_C-0
Map1	9	0	0	Primary	container03_C-0

Server total: 2 (784 B)

Total catalog service domain count: 4 (1 KB)

(The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.)

CWXS10040I: The showMapSizes command completed successfully.

Ending at: 2013-05-06 12:04:20.019

5.2 Integrating in a WebSphere Application Server Network Deployment environment

WebSphere eXtreme Scale can easily be integrated into a WebSphere Application Server Network Deployment environment. This deployment scenario offers several advantages over the stand-alone environment. It allows you to use Java Platform, Enterprise Edition programming model and supporting services, such as transaction management, security, and monitoring.

This section reviews the benefits and limitations (5.2.1, “Benefits” on page 179, 5.2.2, “Limitations” on page 182, and 5.2.3, “Considerations” on page 183) of WebSphere eXtreme Scale integrated into a WebSphere Application Server environment. It also introduces a sample application and topology that is used to demonstrate the features (5.2.4, “Introducing the sample application” on page 184 and 5.2.5, “Introducing the sample topology” on page 187). This section then provides detailed instructions about how to install and configure the sample environment (5.2.6, “Creating the sample topology” on page 189). In the final sections (5.2.7, “Validating the sample topology” on page 221 and 5.2.8, “Testing the sample topology and application” on page 227), a sample getting started web application is used to validate the implemented topology.

5.2.1 Benefits

Integrating WebSphere eXtreme Scale into WebSphere Application Server Network Deployment environment provides several advantages that enhance system administration and simplify the development of grid applications:

- ▶ Catalog and container servers as managed application servers
- ▶ Catalog service domain configuration
- ▶ Simplified connection to the grid
- ▶ Simplified grid container management
- ▶ Simplified administration and management
- ▶ Managed environment for loader implementations
- ▶ Transparent transaction demarcation
- ▶ Use underlying security infrastructure
- ▶ Use PMI for monitoring

Catalog and container servers as managed application servers

Catalog and container servers are defined as standard application server instances within a WebSphere Application Server cell.

For high availability reasons, use dedicated clusters for catalog and container servers. Best results can be achieved by having at least three catalog server instances that are in different physical locations. Three catalog servers provide a high-available catalog service even when a catalog server is down for planned or unplanned maintenance.

Also, use a distinct WebSphere cluster to host the grid infrastructure (container servers) to decouple it from catalog servers and applications.

Catalog service domain configuration

Catalog service domains define a group of catalog servers that are associated to a grid infrastructure. Catalog service domains can be easily defined as administered resources within a WebSphere Application Server Network Deployment environment as shown in Figure 5-14.

Catalog service domains > CSD01

A WebSphere eXtreme Scale catalog service domain is a highly available collection of catalog servers. This collection of catalog servers can run in WebSphere Application Server server processes within a single cell and core group. The catalog service domain can also define a group of remote servers that run in different WebSphere Application Server cells or as Java SE processes. The catalog service controls the placement of shards and discovers and monitors the health of the container servers in the data grid.

[Test connection](#)

General Properties

* Name

Enable this catalog service domain as the default unless another catalog service domain is explicitly specified.

Enable IBM eXtremeIO (XIO) communication. This configuration option is not applicable for domains that are configured with remote server members.

Additional Properties

- [Client security properties](#)
- [Custom properties](#)

Catalog Servers

[New](#) [Edit](#) [Delete](#)

Select	Catalog Server Endpoint	Client Port	Listener Port	Status
<input type="checkbox"/>	WCell01\WNode01\catalog01	6601		
<input type="checkbox"/>	WCell01\WNode02\catalog02	6602		
<input type="checkbox"/>	WCell01\WNode03\catalog03	6603		

Figure 5-14 Catalog service domain configuration

Simplified connection to the grid

In a stand-alone environment, connecting to the grid requires knowledge of the host name and port of the catalog servers. In an integrated environment, this requirement can be simplified by using a catalog service domain. More specifically, the catalog service domain can be queried to obtain the list of catalog service endpoints that client uses to establish a connection to the catalog service. Example 5-36 shows a code snippet of how this simplification can be done.

Example 5-36 Connecting to a grid without specifying a catalog server host and port

```
//Use ObjectGridManagerFactory to get the reference to the ObjectGridManager API
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();

//Get CatalogDomainManager through objectGridManager
```

```
CatalogDomainManager catalogDomainManager = objectGridManager.getCatalogDomainManager();

//Get catalogDomainInfo for CSD01 domain
CatalogDomainInfo catalogDomainInfo = catalogDomainManager.getDomainInfo("CSD01");

//Get the catalog server endpoints for catalogs within CSD01
String cep = catalogDomainInfo.getClientCatalogServerEndpoints();

//Obtain client context using catalog service endpoints
ClientClusterContext ccc = objectGridManager.connect(cep, (ClientSecurityConfiguration) null,
(URL) null);

//Get the objectGrid instance
ObjectGrid objectGrid = objectGridManager.getObjectGrid(ccc, "Grid");
```

Simplified grid container management

Grid configurations can be packaged within a Java Platform, Enterprise Edition module (Web or EJB module) and deployed as a standard enterprise archive (EAR) file. When a Java Platform, Enterprise Edition application starts, WebSphere eXtreme Scale checks for grid configuration files (`objectGrid.xml` and `objectGridDeployment.xml`) within the META-INF folder of the EJB and WEB modules. If only `objectGrid.xml` is found, the application server is assumed to be a grid client. If both `objectGrid.xml` and `objectGridDeployment.xml` are present, the application server acts as a container by implementing the specified deployment policies.

Simplified administration and management

In an integrated environment, the WebSphere Application Server Network Deployment infrastructure is responsible for deployment operations. More specifically, it ensures that grid configurations packaged within EARs are consistently deployed over all the managed nodes. The WebSphere administrative console can be used to manage grid containers like other application servers, which simplifies administration and management.

Managed environment for loader implementations

Implementing custom loaders often requires special resources to gain access to the back-end information system. In an integrated environment, a loader implementation can use all available Java Platform, Enterprise Edition resources. For example, JDBC connections, JMS providers, or JCA adapters can be used in a custom loader implementation using connection pooling, lookup and discovery (JNDI) and security (providing necessary authentications and authorizations).

Transparent transaction demarcation

WebSphere eXtreme Scale offers transactional access to a grid. In a stand-alone environment, the grid client must demarcate transaction boundaries. Therefore, this transaction management requires more programming effort.

In an integrated environment, WebSphere Application Server can act as transaction manager to manage transactions that involve grid and non-grid resources, such as relational databases. This capability can be used by interacting with the grid using the WebSphere eXtreme Scale Resource Adapter. This resource adapter is compliant to Java EE Connector Architecture (JCA) specifications, and provides local transaction support. It is a single-phase commit resource from the perspective of the transaction manager. WebSphere Application Server offers the capability to manage transactions that involve multiple two-phase capable

resource managers and one single-phase capable resource manager, by using *last participant support* feature.

Note: WebSphere eXtreme Scale version 7.1.1 and prior versions supported an integration with WebSphere Application Server Transaction Manager using the WebSphereTransactionCallBack plug-in. This plug-in has been deprecated in WebSphere eXtreme Scale version 8.6.

Use underlying security infrastructure

WebSphere Application Server security infrastructure can be used to propagate the client authentication credential from a grid client to catalog and container servers within a WebSphere security domain. A secured Java Platform, Enterprise Edition component that is deployed into WebSphere Application Server can include a client security token as part of its interaction with catalog and container servers. Catalog and container servers can extract client credentials from WebSphere Application Server security token to apply the correct authentication and authorization controls.

Use PMI for monitoring

WebSphere Application Server provides a sophisticated performance monitoring infrastructure (PMI). It can be used to gather performance statistics of all components inside an application server. WebSphere eXtreme Scale takes advantage of this infrastructure to collect performance statistics relevant to the grid. Metrics are grouped within the following PMI modules:

objectGridModule	Provides metrics about transaction response time.
mapModule	Provides metrics about map and index count and time statistics. These include hit rate per map, number of entries per map, number of results per index and so on.
agentManagerModule	Provides statistics related to map-based agents. These include number of agent executions, number of partitions involved, time required to run map and reduce operations, time required to serialize agents, and related results.
queryModule	Provides statistics about query processing and execution. These include the amount of time to create a query plan and run the query, the number of times that a query has been run, and so on.

5.2.2 Limitations

There are limitations when using WebSphere eXtreme Scale integrated in a WebSphere Application Server environment. Consider the following topics.

Larger footprint

In an integrated environment, the grid shares the available memory with application server or application components. This factor translates into less memory available for storing objects in the grid.

Avoiding deploying enterprise applications and grid on the same cluster

When the grid container is deployed within the enterprise application in the same application server, the grid container becomes more vulnerable. Issues in the application (for example, memory leaks or deadlock situations) can bring the grid container down. To increase high availability, separate the application and the grid into different clusters.

Inter-process communications

High Availability Manager and core groups provided by WebSphere Application Server Network Deployment infrastructure are used by WebSphere eXtreme Scale for health checks and for intercommunication among catalog servers. Given the point-to-point communications between managed processes, the system resource consumption (processor, memory, and network bandwidth) does not increase linearly and exhibits quadratic growth as the size of a core group increases. To optimize core group communications and system resource consumption, distribute the managed processes across multiple core groups.

Typically, the maximum number of cluster members within a single core group is 50 - 100, according to application design and incoming workload pattern. This means that a large grid infrastructure should be deployed across multiple clusters, each one using a dedicated core group.

It is also possible to implement a zone-based topology with multiple clusters and core groups by associating a zone to a specific cluster/core group.

For more information about core group configuration and scalability, see the *Core group scaling considerations* topic in the WebSphere Application Server Network Deployment version 8.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/crun_ha_cgscale.html

5.2.3 Considerations

The section addresses factors that need to be considered when you use WebSphere eXtreme Scale with WebSphere Application Server.

Topology selection

WebSphere Application Server offers a large variety of deployment topologies to support many scalability and high availability requirements. Grid requirements place an extra constraint on this topology. Carefully plan the overall target topology to match all requirements. For more information grid topologies, see Chapter 2, “Architecture and topologies” on page 27 and *WebSphere Application Server V8.5 Concepts, Planning, and Design Guide*, SG24-8022.

Configuration files

An EJB module or WAR module using a grid must have corresponding grid XML configuration files in its META-INF directory. Depending on what configuration files are found by the WebSphere eXtreme Scale runtime environment, the application runs in one of the following two modes:

- ▶ Client only: When only an `objectGrid.xml` file is detected, the application is a grid client only. The grid configuration from the XML file is used to control grid operations at the client side, such as evictors and near-cache configurations.
- ▶ Grid container: When the META-INF directory contains both `objectGrid` and `objectGridDeployment.xml` files, a grid container is automatically started inside the application server. The grid container registers with the catalog server and becomes eligible for shard placement.

Tip: Configuration file names are case sensitive. If the grid container does not start automatically, be sure to check the file name spelling.

5.2.4 Introducing the sample application

This section introduces the sample application that is used for WebSphere eXtreme Scale integrated into a WebSphere Application Server Network Deployment environment example.

Application architecture and design

A getting started web application is used to demonstrate the integrated topology. This application contains two main components as shown in Figure 5-15.

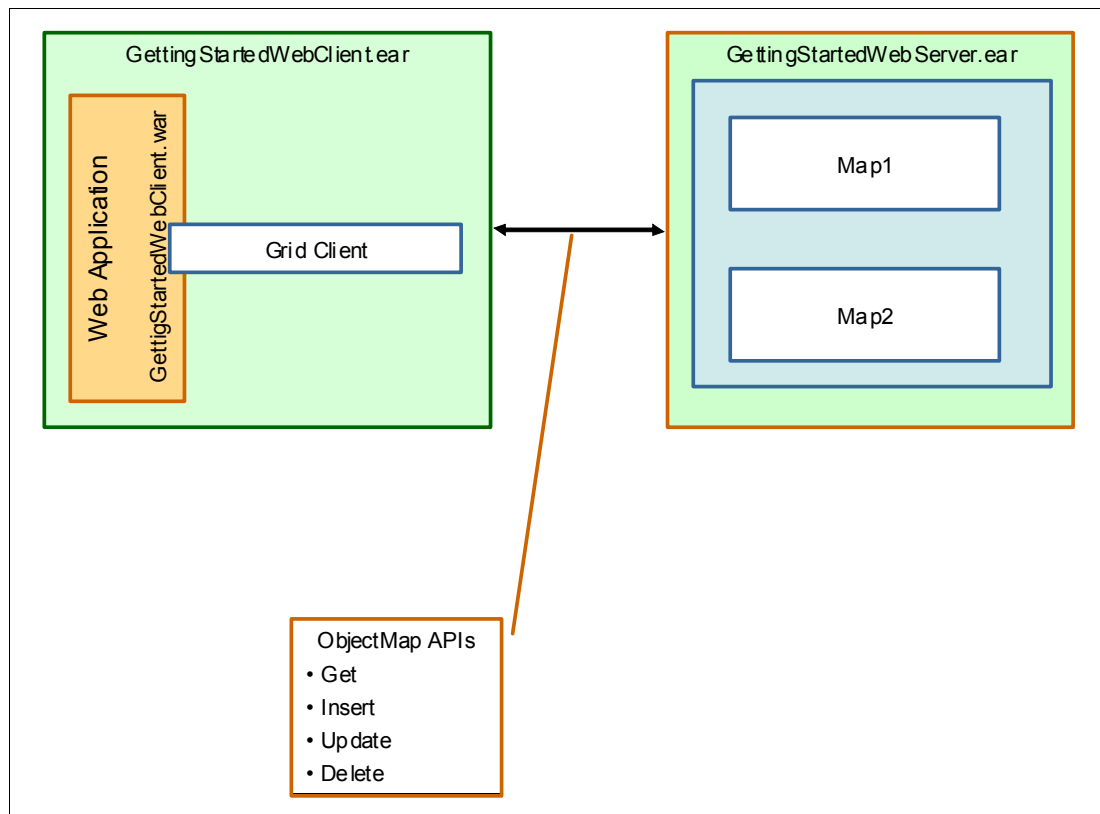


Figure 5-15 Sample application used for integrating into a WebSphere Application Server environment

The getting started application consists of two main components:

- ▶ **Grid client application:** A simple Java Platform, Enterprise Edition web application, `GettingStartedWebClient.ear`, uses a servlet to connect to a grid and run basic operations like get, update, insert, and delete against two maps, `Map1` and `Map2`.
- ▶ **Grid configuration:** The grid provides a single, coherent, highly available, and scalable store for `Map1` and `Map2`. Grid configuration files are packaged within the `META-INF` folder of a web module (`GettingStartedWebServer.war`) that is incorporated into the `GettingStartedWebServer.ear`.

The sample applications in this section, `GettingStartedWebClient.ear` and `GettingStartedWebServer.ear`, are provided in the `WASND` folder of the `DeploymentScenarios.zip` file available as additional material. For more information about downloading this material, see Appendix A, “Additional material” on page 341.

Client connection to the grid

In an integrated installation with WebSphere Application Server Network Deployment, the client can connect to the grid by either using catalog service endpoints or the catalog service domains as shown in the Example 5-37.

Example 5-37 Client connection to the grid

```
protected synchronized String connectClient(String cep, String gridName, String domainId) {
    if (grid == null) {
        // Connect to the Catalog Service. The security and client override
        // XML are not specified.
        // use catalog domain manager instead of cep if defined
        cdm = ObjectGridManagerFactory.getObjectGridManager().getCatalogDomainManager();
        if (cdm != null) {
            if(cdm.getDomainInfo(domainId) != null) {
                try {
                    String endpoints = cdm.getDomainInfo(domainId).getClientCatalogServerEndpoints();
                    cep = endpoints;
                }
                catch (Exception e) {
                    System.out.println(e);
                    e.printStackTrace();
                }
            }
        }
    }
    try {
        // If the catalog service end points are null, then call the connect
        // method without the catalog service end points parameter to connect to
        // the catalog service configured for this process.
        ClientClusterContext ccc;
        if (cep != null) {
            ccc = ObjectGridManagerFactory.getObjectGridManager().connect(cep, null, null);
        } else {
            ccc = ObjectGridManagerFactory.getObjectGridManager().connect(null, null);
            // for display purposes
            cep = getCatalogServiceEndpoints();
        }
        // Set a custom client properties file
        URL clientPropsURL =
        Thread.currentThread().getContextClassLoader().getResource("properties/objectGridClient.properties");
        ccc.setClientProperties(gridName, clientPropsURL);
        // Retrieve the ObjectGrid client connection and cache it.
        grid = ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc, gridName);
    } catch (Exception e) {
        throw new ObjectGridRuntimeException("Unable to connect to catalog service at
        endpoints:" + cep, e);
    }
    return cep;
}
```

Client interaction with the grid

The client interacts with Map1 and Map2 using the ObjectMap APIs to insert, update, retrieve, and remove data in the grid as shown in the Example 5-38.

Example 5-38 Client interaction with the grid

```
Session sess = grid.getSession();
// Get the ObjectMap
ObjectMap map1 = sess.getMap(mapName);
if (crudOperation.equals("insert")) {
    map1.insert(key, val);
    pw.println("SUCCESS: Inserted " + val + " with key " + key + EOL);
} else if (crudOperation.equals("update")) {
    map1.update(key, val);
    pw.println("SUCCESS: Updated key " +key + " with value " + val + EOL);
} else if (crudOperation.equals("delete")) {
    String oldValue = (String) map1.remove(key);
    pw.println("SUCCESS: Deleted value with key " + key + " and old value " +
oldValue + EOL);
} else if (crudOperation.equals("get")) {
    String value = (String) map1.get(key);
    if (value != null)
        pw.println("SUCCESS: Value is: " + value + EOL);
    else
        pw.println("SUCCESS: Value is: Not found" + EOL);
}
}
```

Grid configuration

Example 5-39 shows the content of objectGrid.xml file that is being used to define the grid configuration. This configuration file is responsible for defining all the objectGrid and backingMap elements that make up the application data store. In this example, a grid called Grid contains two backing map definitions, called Map1 and Map2.

Example 5-39 Grid configuration within objectGrid.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="Grid">
        <backingMap name="Map1" />
        <backingMap name="Map2" />
    </objectGrid>
</objectGrids>
</objectGridConfig>
```

The grid is configured to manage thirteen partitions, with one optional synchronous replication for each primary shard. This configuration is defined in the objectGridDeployment.xml file as shown in Example 5-40.

Example 5-40 Grid deployment configuration within objectGridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
../deploymentPolicy.xsd"
xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
<objectgridDeployment objectgridName="Grid">
    <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0"
maxSyncReplicas="1" >
        <map ref="Map1"/>
        <map ref="Map2"/>
    </mapSet>
</objectgridDeployment>
</deploymentPolicy>

```

These deployment descriptors are packaged into the META-INF folder of the GettingStartedWebServer web application as shown in Figure 5-16.

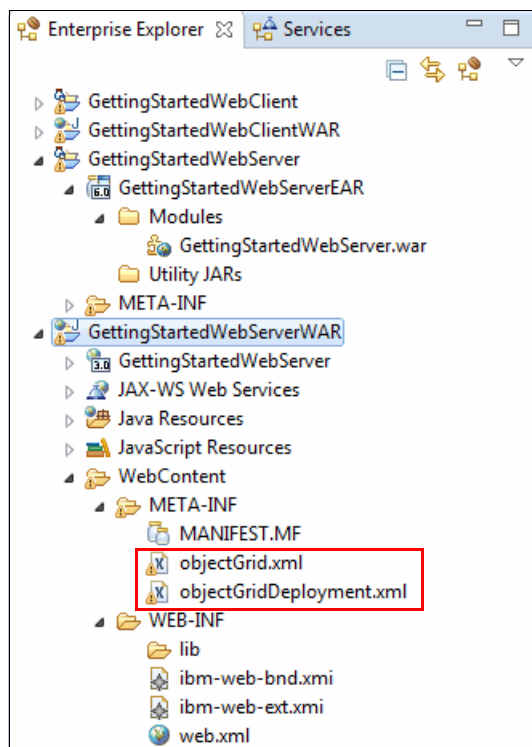


Figure 5-16 Grid configuration packaged into GettingStartedWebServer

5.2.5 Introducing the sample topology

The getting started web application in 5.2.4, “Introducing the sample application” on page 184 requires deployment topology on which to run. This section describes the sample topology that demonstrates WebSphere eXtreme Scale integrated into a WebSphere Application Server Network Deployment environment.

Operational model

The basic operational model of the sample topology is shown in Figure 5-17. Like the stand-alone topology in 5.1, “WebSphere eXtreme Scale in a stand-alone environment” on page 152, the sample topology includes four hosts (whost01, whost02, whost03, and whost04) where the client, catalog, and container services are run.

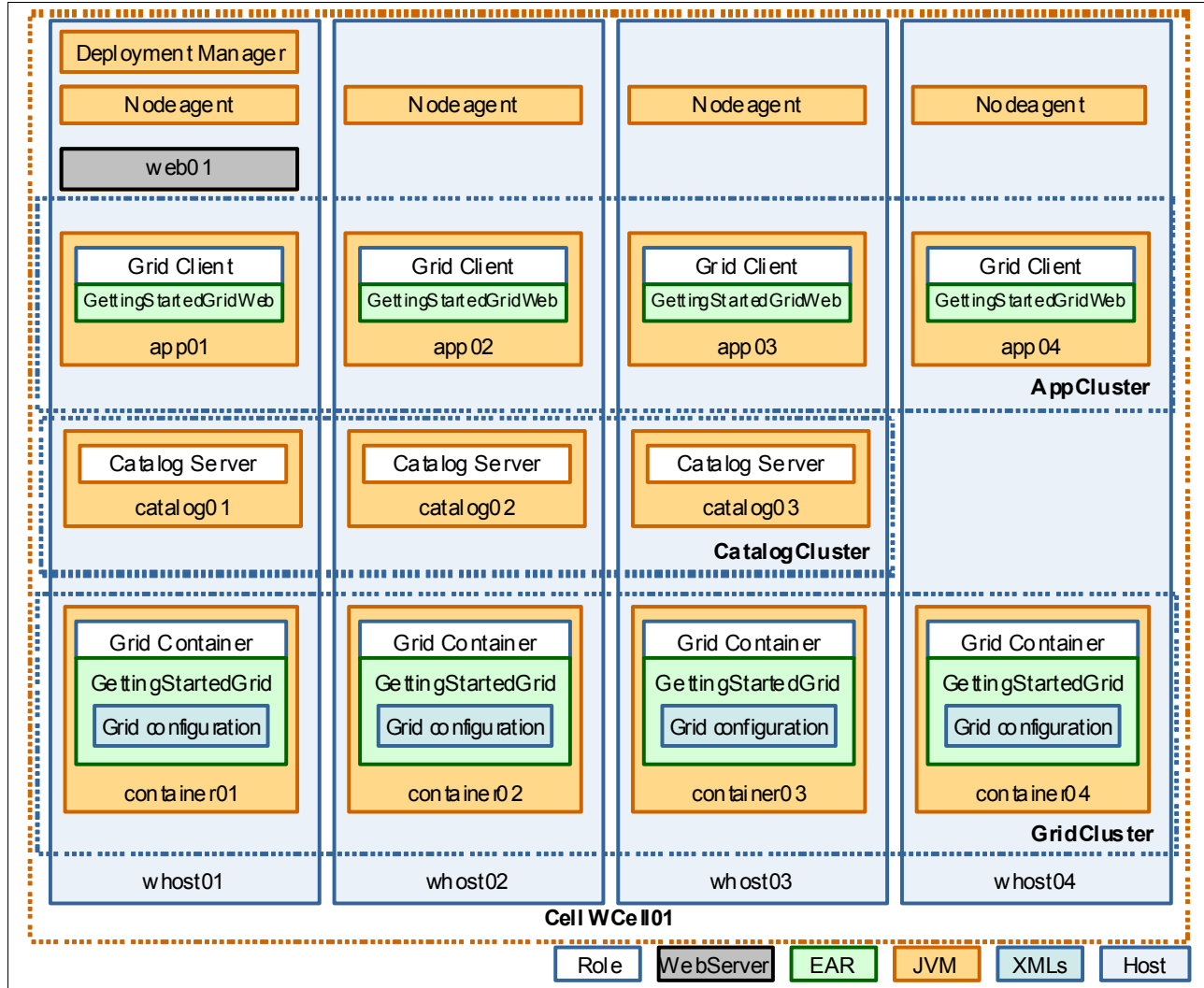


Figure 5-17 Sample topology for integrating into a WebSphere Application Server Network Deployment environment

More specifically, the sample integrated environment has the following characteristics:

- ▶ One WebSphere Deployment Manager is deployed on whost01.
- ▶ One IBM HTTP server instance (web01) is deployed on whost01.
- ▶ A catalog service is hosted by *CatalogCluster*, comprising three members (catalog01, catalog02, and catalog03) that are deployed on three hosts (whost01, whost02, and whost03).
- ▶ A grid is hosted by the *GridCluster*, comprising four members (container01, container02, container03, and container04) that are deployed over four hosts (whost01, whost02, whost03, and whost04).

- ▶ The getting started grid client application is hosted by the *AppCluster*, comprising four members (app01, app02, app03, and app04) that are deployed over four hosts (whost01, whost02, whost03, and whost04).

Grid topology

The grid is operating within the GridCluster. As described in “Grid configuration” on page 156, the grid is configured to manage thirteen partitions, with one optional synchronous replication for each primary shard. A possible placement is shown in the Figure 5-18. The actual shard placement can vary, depending upon the availability of container cluster members and the placement decisions made by the catalog server.

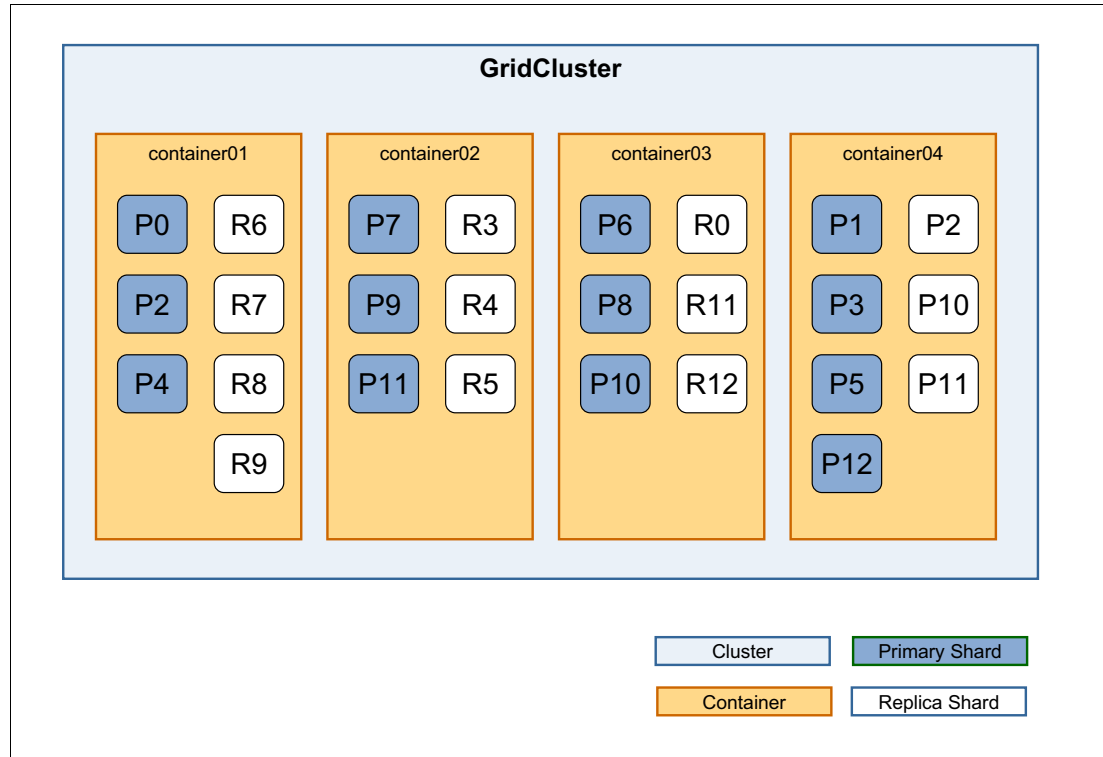


Figure 5-18 Grid topology

5.2.6 Creating the sample topology

This section describes the required steps to create an integrated environment based on WebSphere eXtreme Scale and WebSphere Application Server Network Deployment.

The following products are required to be installed on the four hosts (whost01, whost02, whost03, and whost04) included in the sample topology:

- ▶ IBM WebSphere Application Server Network Deployment version 8.5.0.2
- ▶ IBM WebSphere eXtreme Scale version 8.6.0.1

On whost01, the following products must also be installed:

- ▶ IBM HTTP server version 8.5.0.2
- ▶ IBM WebSphere plug-ins for IBM HTTP server version 8.5.0.2

The installation of the required software stack is run by using the IBM Installation Manager, which is a common installation program that is used across many IBM software products.

After the products are installed, they can be configured to match the wanted topology. This process includes the creation of application server profiles, nodes federated under the deployment manager, and the creation of clusters.

Installing WebSphere Application Server

To install WebSphere Application Server Network Deployment version 8.5.0.2, complete the following steps:

Note: These steps must be repeated for every node in the topology.

1. Install the IBM Installation Manager, if not already present on deployment hosts. The IBM Installation Manager is incorporated as part of the WebSphere Application Server Network Deployment package.
2. Install WebSphere Application Server Network Deployment version 8.5.0.0 on each node by using IBM Installation Manager.

For more information about installation procedures, see the *Installing and uninstalling the product on distributed operating systems* topic in the WebSphere Application Server Network Deployment version 8.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.installation.nd.doc/ae/tins_installation_dist.html

3. Update WebSphere Application Server Network Deployment to the latest fix pack (8.5.0.2 at the time of writing):
 - a. Download the latest available WebSphere Application Server Network Deployment Fixpack from IBM Fix Central:
<http://www-933.ibm.com/support/fixcentral/>
 - b. Apply the latest fix pack by using the IBM Installation Manager.

For more information about locating and applying the fix pack, see the *Installing interim fixes on distributed operating systems using the GUI* topic in the WebSphere Application Server Network Deployment version 8.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.installation.nd.doc/ae/tins_install_fixes_dist_gui.html

Installing IBM HTTP server and plug-in

You can install the IBM HTTP server and plug-in by using the IBM Installation Manager.

Note: Perform these steps on the whost01 host only.

To install IBM HTTP server and Plug-in version 8.5.0.2, complete these steps:

1. Configure the IBM Installation Manager repository to point to the WebSphere Application Server version 8.5 *Supplements* package.
2. Select the following packages for installation:
 - IBM HTTP server for WebSphere Application Server
 - Web Server Plug-ins for IBM WebSphere Application Server
3. Specify an installation directory for each package. You can keep the default paths or update them to match file system conventions in your environment.

4. Configure a port number for the IBM HTTP server to listen on. Keep the default port 80 if IBM HTTP server is running as root. Otherwise, select an available port greater than 1024 (for example, 8080).

Installing WebSphere eXtreme Scale

Install WebSphere eXtreme Scale integrated into a WebSphere Application Server Network Deployment environment by using the IBM Installation Manager.

Complete these steps to install WebSphere eXtreme Scale version 8.6.0.1:

Note: These steps must be repeated for every node in the topology.

1. Download the latest available WebSphere eXtreme Scale 8.6 fix pack for WebSphere Application Server 8 (8.6.0.1 at the time of writing) from IBM Fix Central:

<http://www.ibm.com/support/fixcentral/>

Fixpacks: WebSphere eXtreme Scale provides separate fix packs according to the chosen deployment scenario (stand-alone or integrated installation with WebSphere Application Server):

- ▶ 8.6.0-WS-WXS-FP0000001 is the fix pack for the stand-alone environment
- ▶ 8.6.0-WS-WXS-WAS8-FP0000001 is the fix pack for integrated installation with WebSphere Application Server

2. Extract the downloaded fix pack into a temporary directory, for example:
/tmp/WXS_8601_WAS8_fixpack
3. Issue the IBMIM command in the <IM_Installation_root> directory to start the IBM Installation Manager.
4. The IBM Installation Manager needs to know the location of WebSphere eXtreme Scale installation binary files. Click **File** → **Preferences**.
5. From the Repository view, click **Add Repositories**.

6. Click **Browse**, navigate to the WebSphere eXtreme Scale installation directory <WXS_product_media>/WXS_8600, and select repository.config as shown in Figure 5-19.

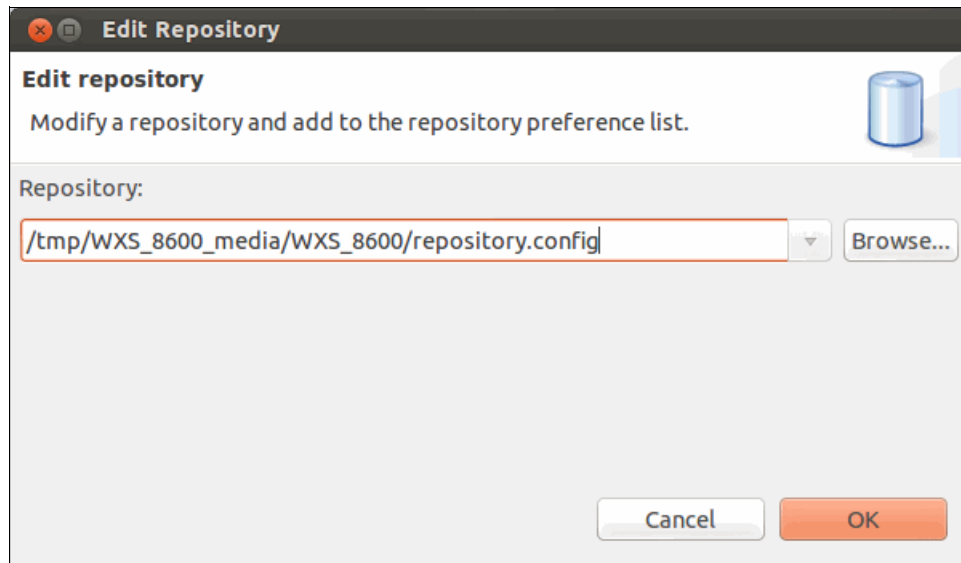


Figure 5-19 WebSphere eXtreme Scale 8.6.0.0 repository location

- Repeat steps 5 on page 191 and 6 on page 192 to add the repository for WebSphere eXtreme Scale 8.6 for WebSphere Application Server 8 Fixpack 1, which is `/tmp/WXS_8601_WAS8_fixpack/repository.config`. Both repositories have being added as shown Figure 5-20. Click **OK** to close the Preferences dialog window.

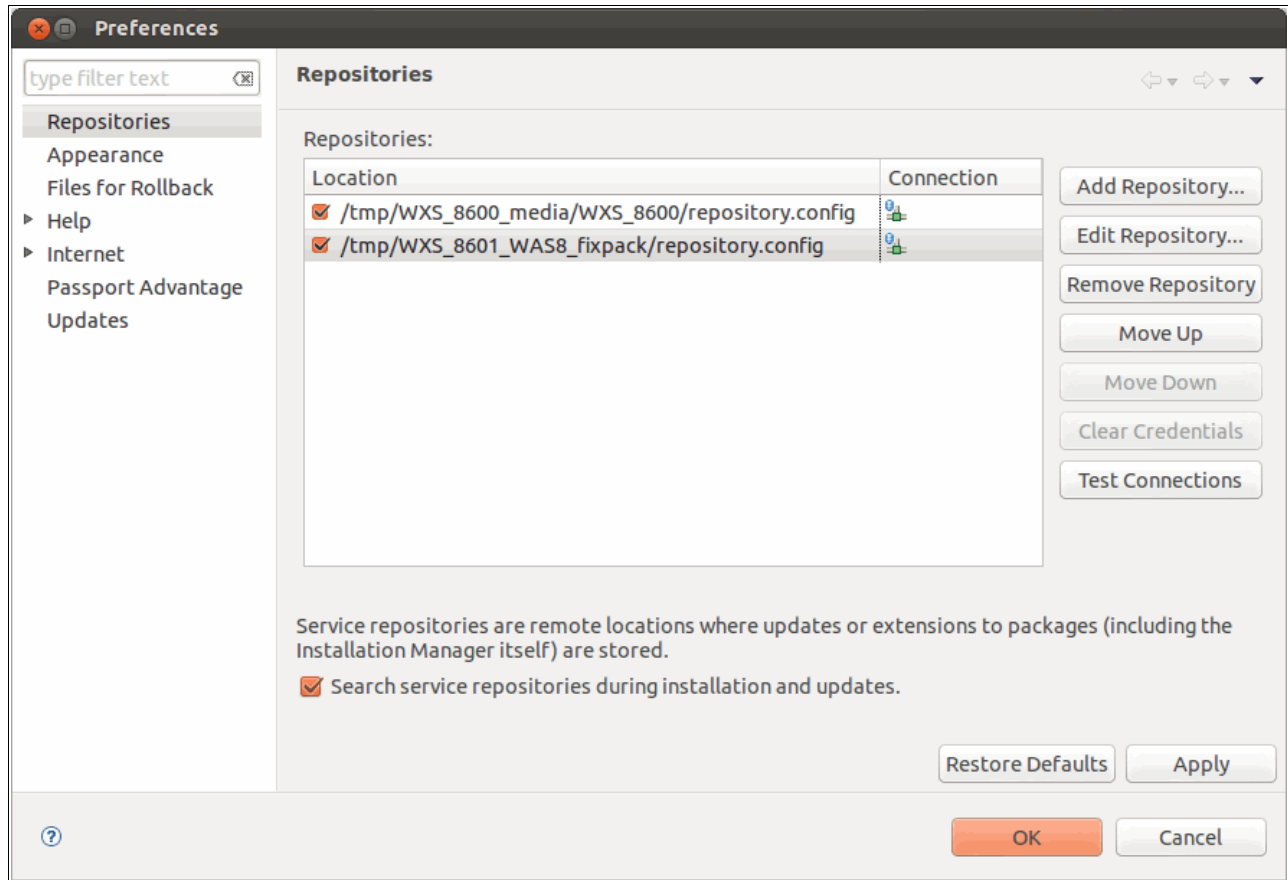


Figure 5-20 Adding repositories

- From the main Installation Manager page, select **Install**.

- On the Install Packages window, a range of installation options is available. Select **IBM WebSphere eXtreme Scale for WebSphere Application Server V8** as shown in Figure 5-21, and click **Next**.

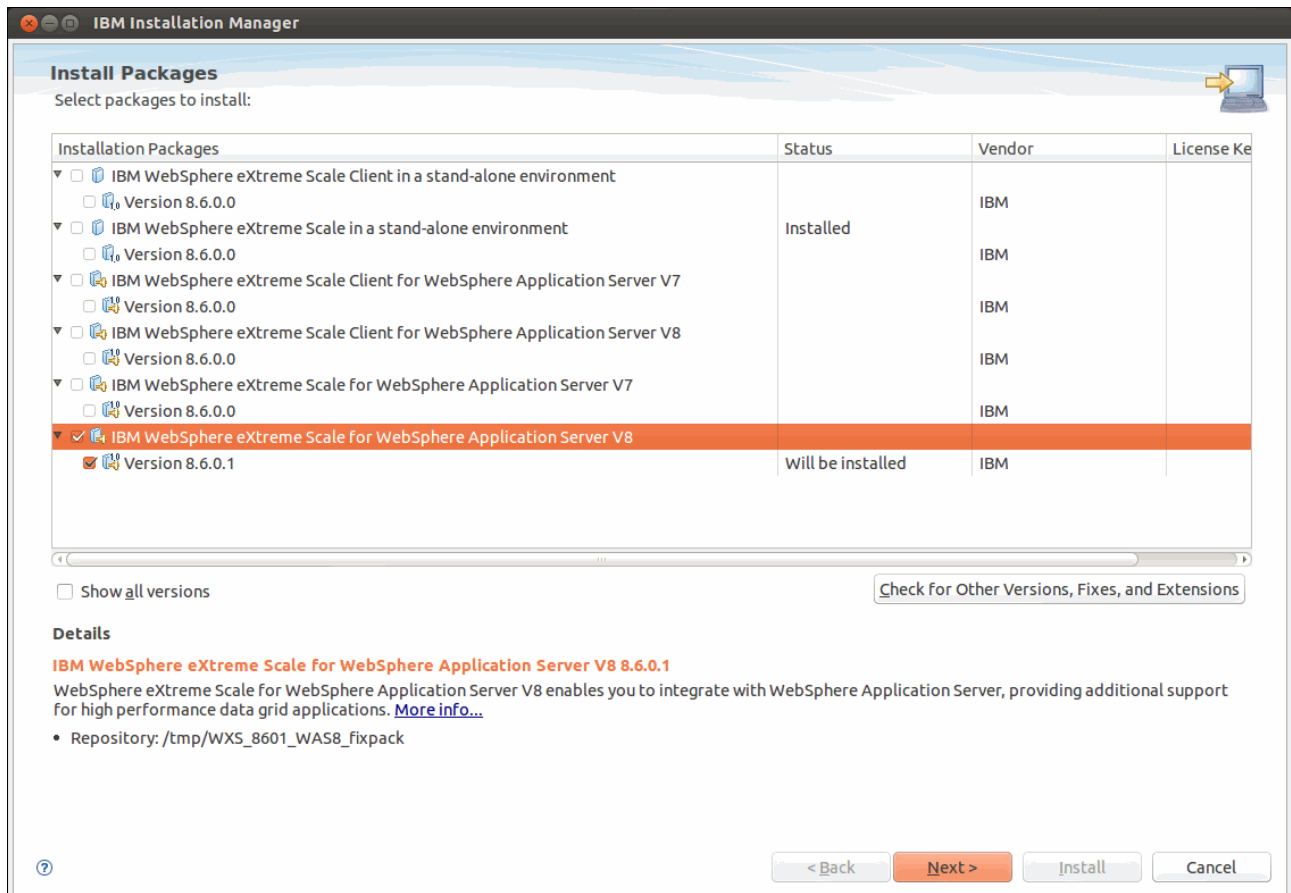


Figure 5-21 Selecting the WebSphere eXtreme Scale 8.6.0.1 for WebSphere Application Server 8 installation package

10. Keep the default option **Use the existing package group**, select the package **IBM WebSphere Application Server V8.5**, and click **Next** as shown in Figure 5-22.

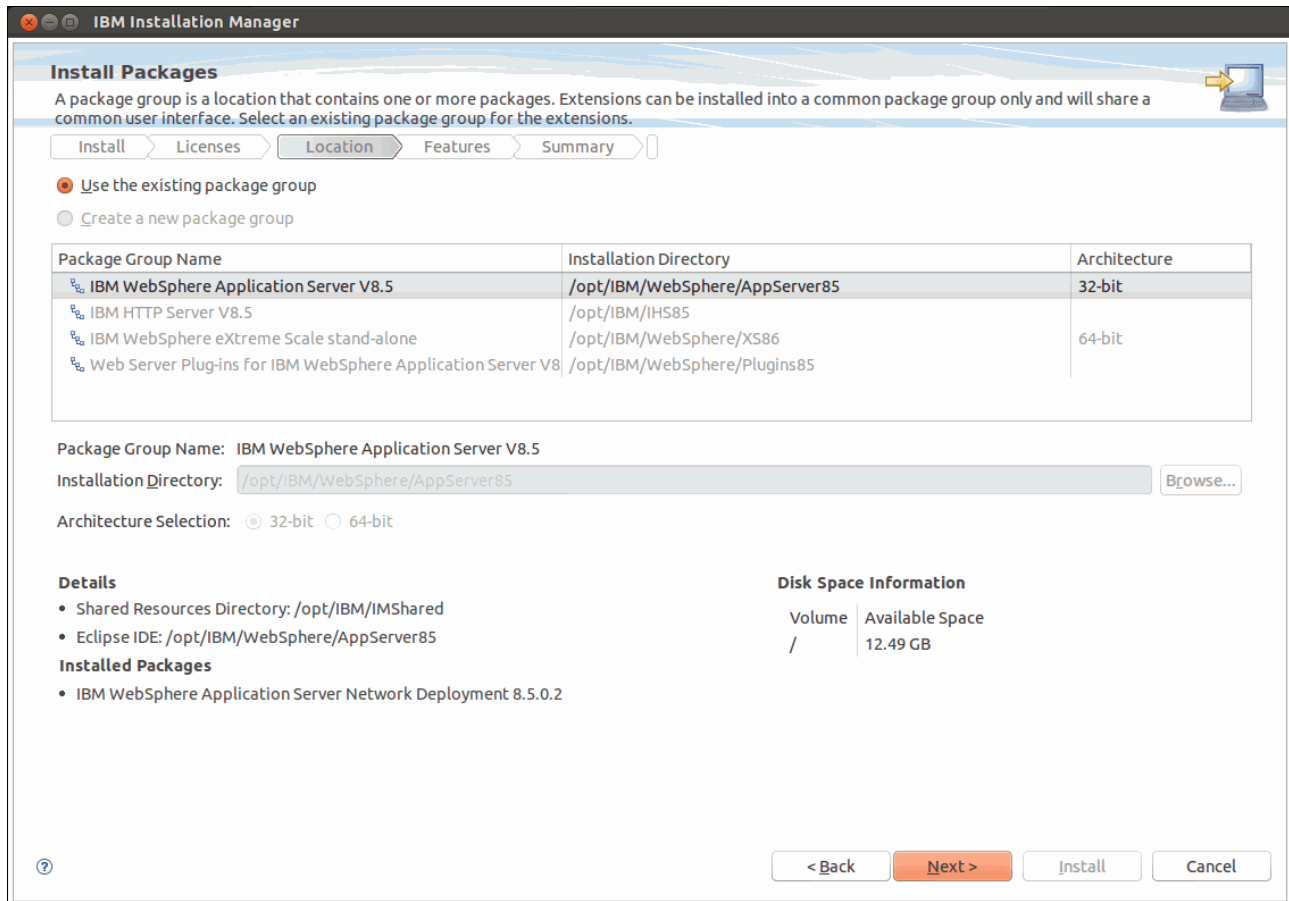


Figure 5-22 WebSphere eXtreme Scale for WebSphere Application Server V8 package group selection

11. Make sure that you select the **Samples** option, as shown in Figure 5-23. Select all of the options and click **Next**.

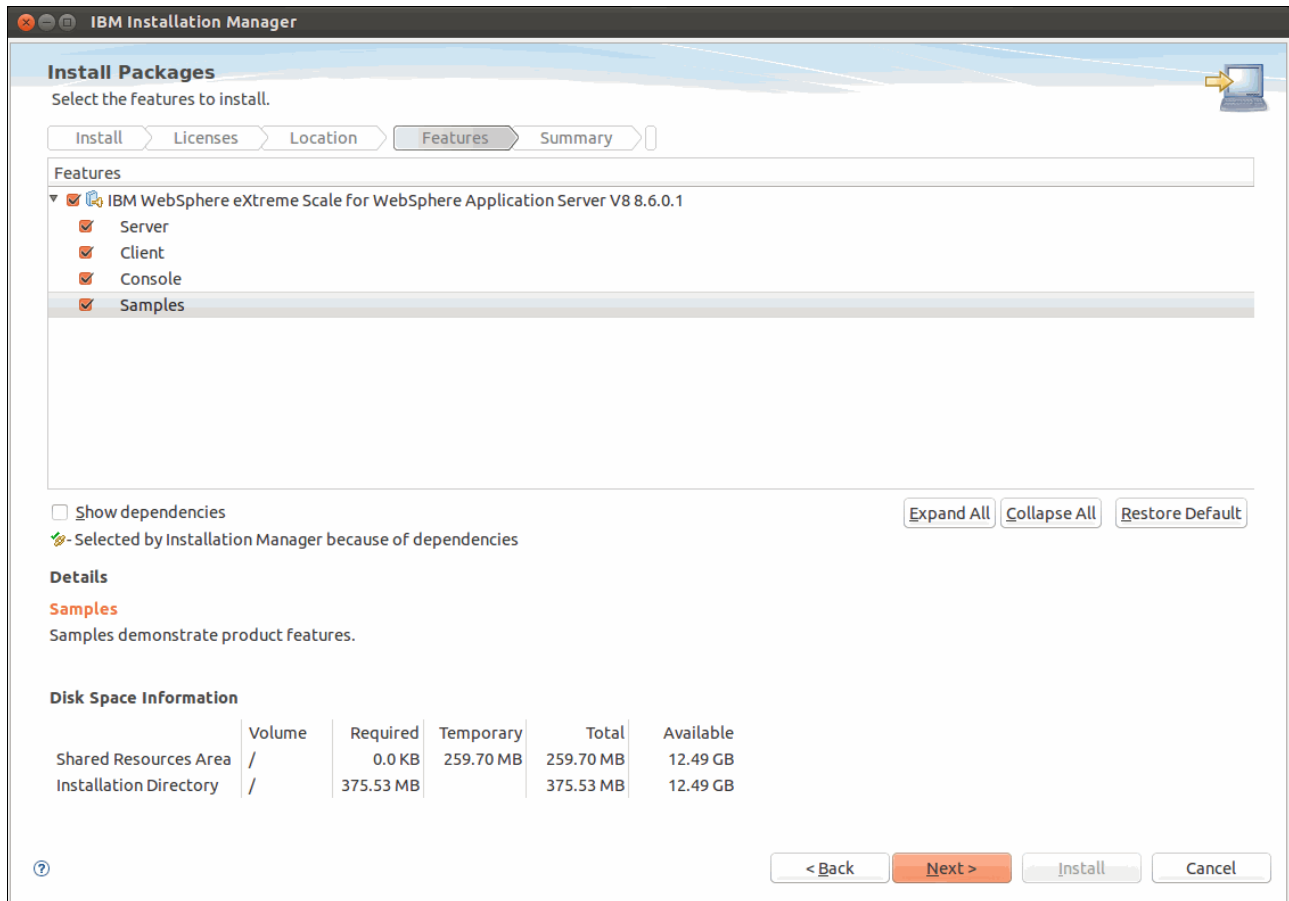


Figure 5-23 WebSphere eXtreme Scale features to install

12. Review the installation summary and click **Install** to start the installation as shown in Figure 5-24.

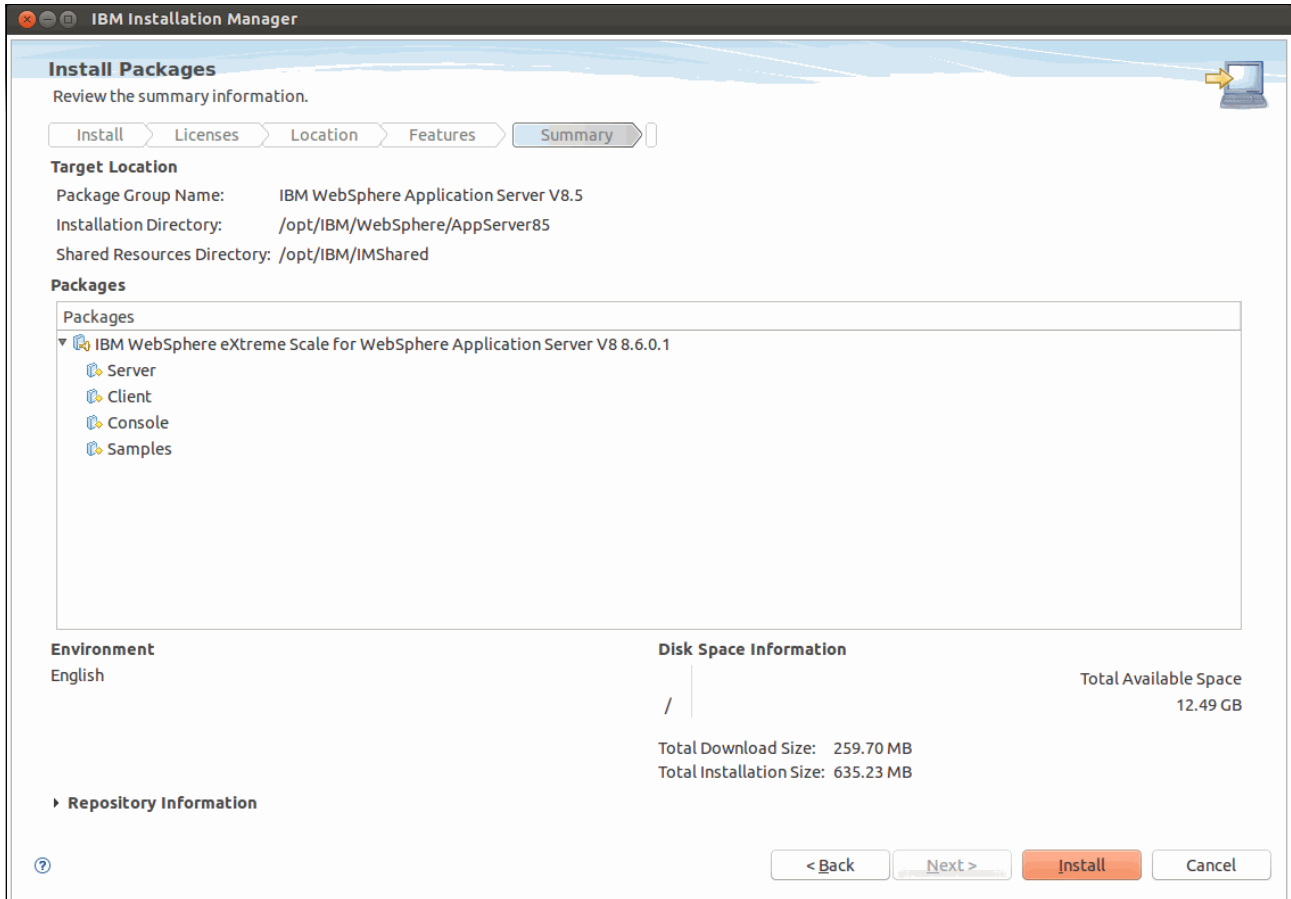


Figure 5-24 WebSphere eXtreme Scale 8.6.0.1 on WebSphere Application Server 8.5 installation summary

13. After the installation completes successfully, select **None** for the **Which program do you want to start** option. Click **Finish** as shown in Figure 5-25.

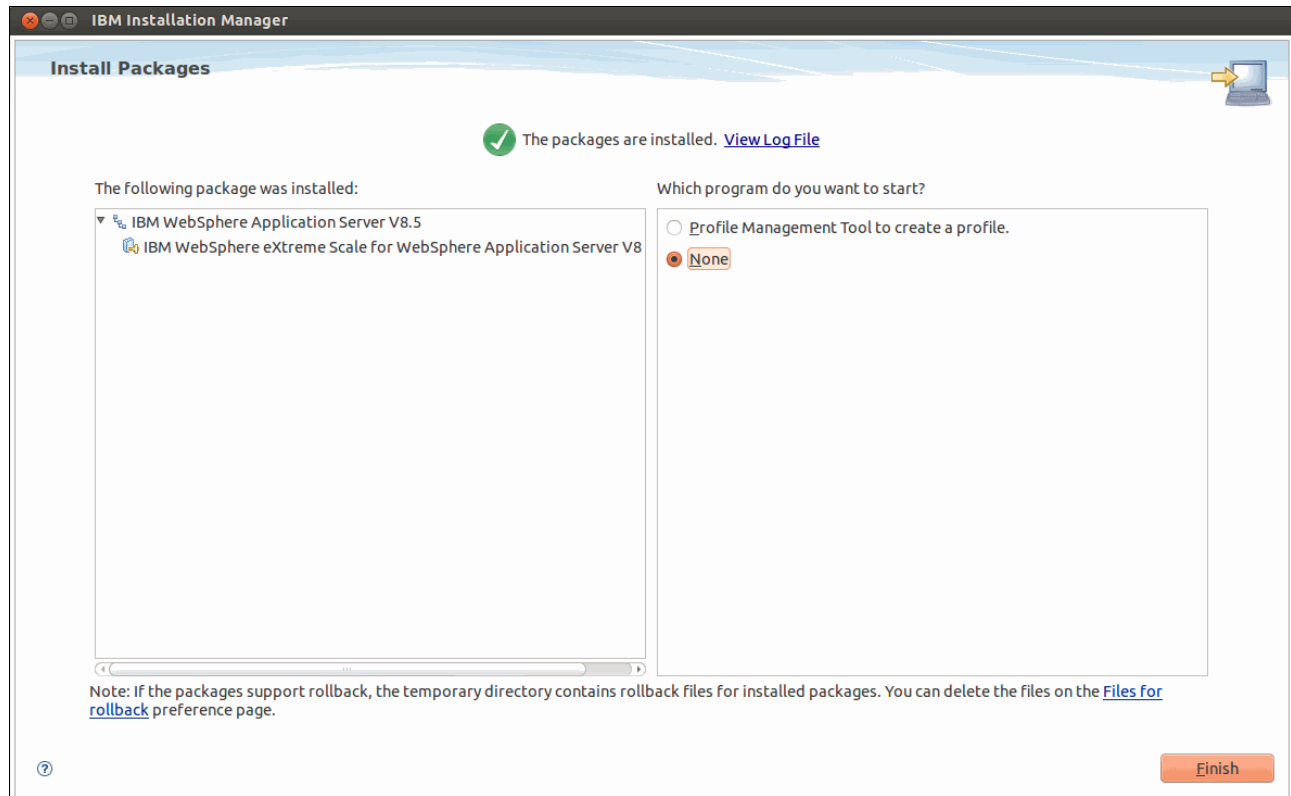


Figure 5-25 WebSphere eXtreme Scale 8.6.0.1 installation confirmation

14. Close the IBM Installation Manager.

Creating WebSphere eXtreme Scale profiles

You can use the WebSphere customization toolbox to create profiles for WebSphere eXtreme Scale. A profile is a set of files that define the configuration for the application server runtime environment. When WebSphere eXtreme Scale is integrated into WebSphere Application Server, the capabilities that are offered by WebSphere eXtreme Scale are added to WebSphere Application Server run time and made accessible through the administration console.

The sample topology includes five profiles as shown in Figure 5-26.

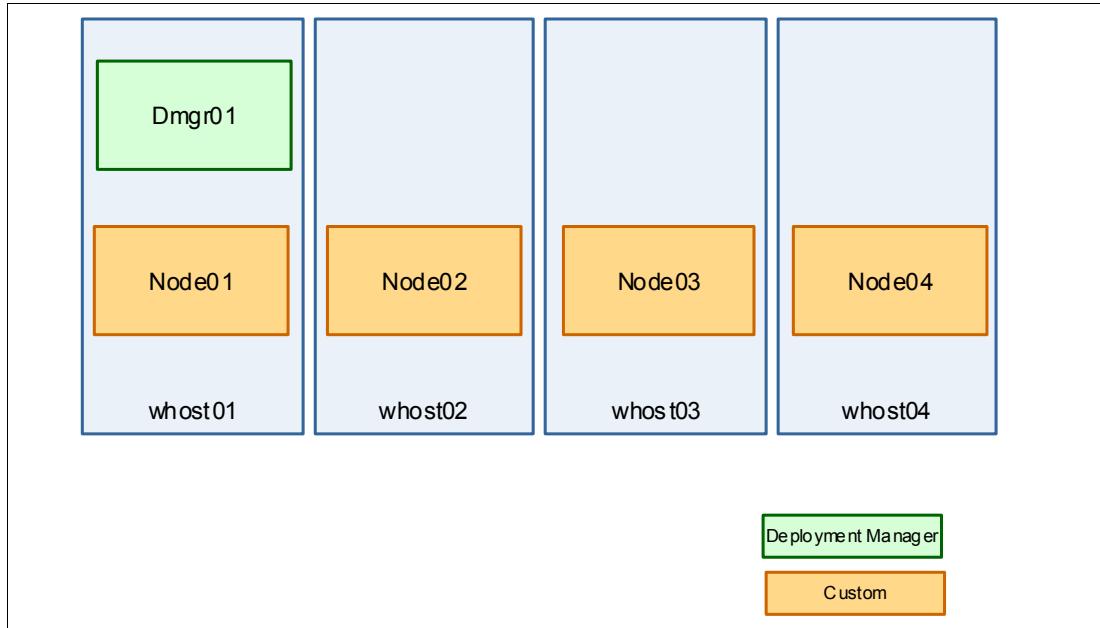


Figure 5-26 Profiles

Complete the following steps to create these profiles:

Note: These steps must be repeated for every host in the topology.

1. Start the WebSphere Application Server Profile Management tool by using the `wct` command in the `<WAS_install_root>/bin/ProfileManagement` directory.
2. On the WebSphere Customization Toolbox window (Figure 5-27), click **Create**.

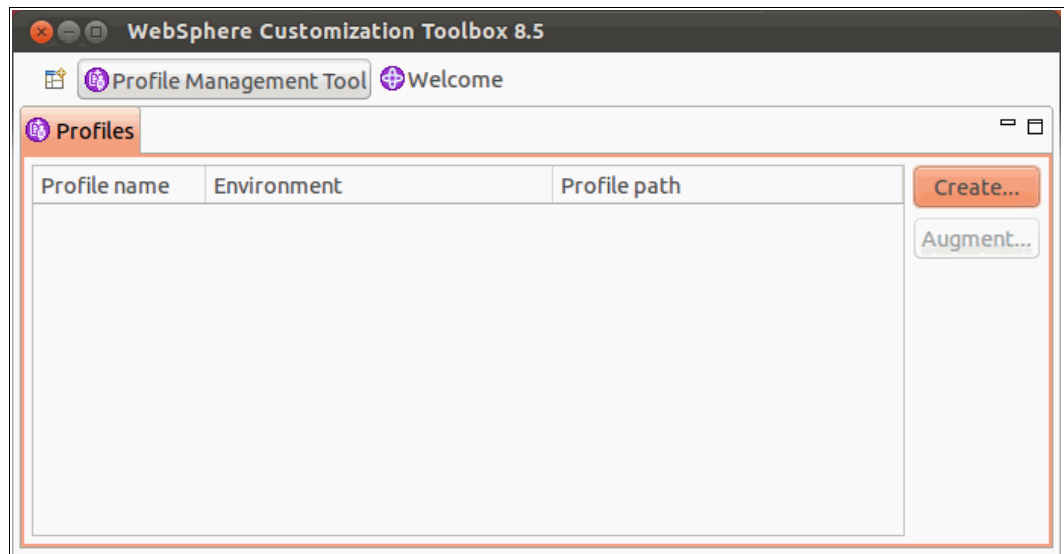


Figure 5-27 Creating a profile by using WebSphere Customization Toolbox

3. The Profile Management Tool supports multiple profile templates. WebSphere eXtreme Scale provides more templates that are grouped within the WebSphere eXtreme Scale Environment Selection as shown in Figure 5-28. Select **WebSphere eXtreme Scale** and click **Next**.

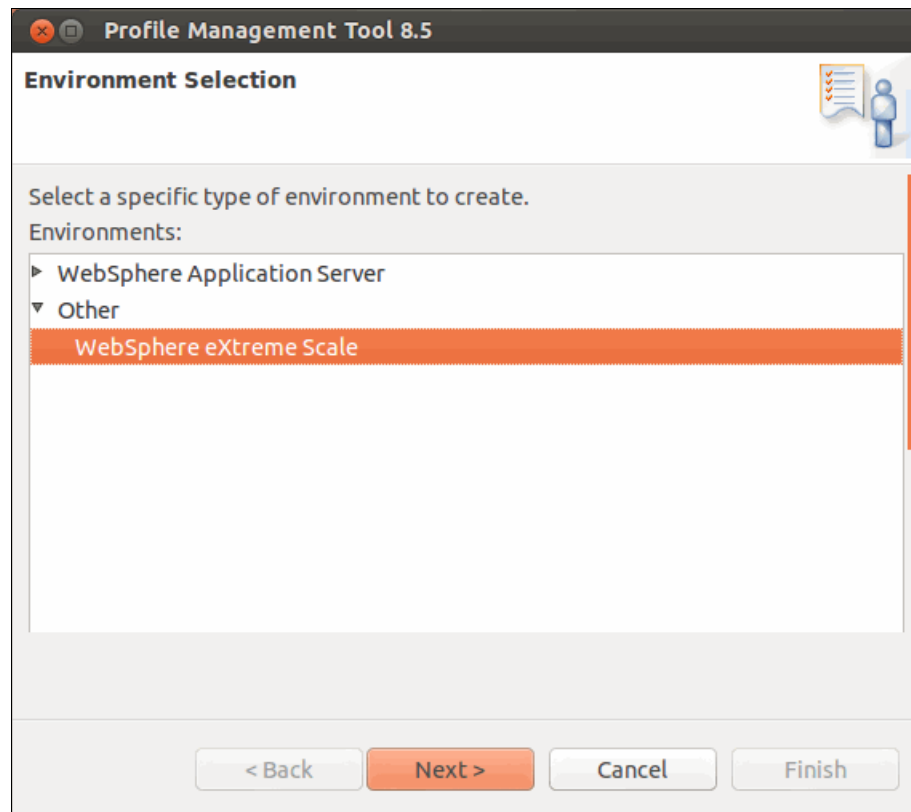


Figure 5-28 WebSphere eXtreme Scale environment selection

The specific WebSphere eXtreme Scale templates are shown in Figure 5-29.

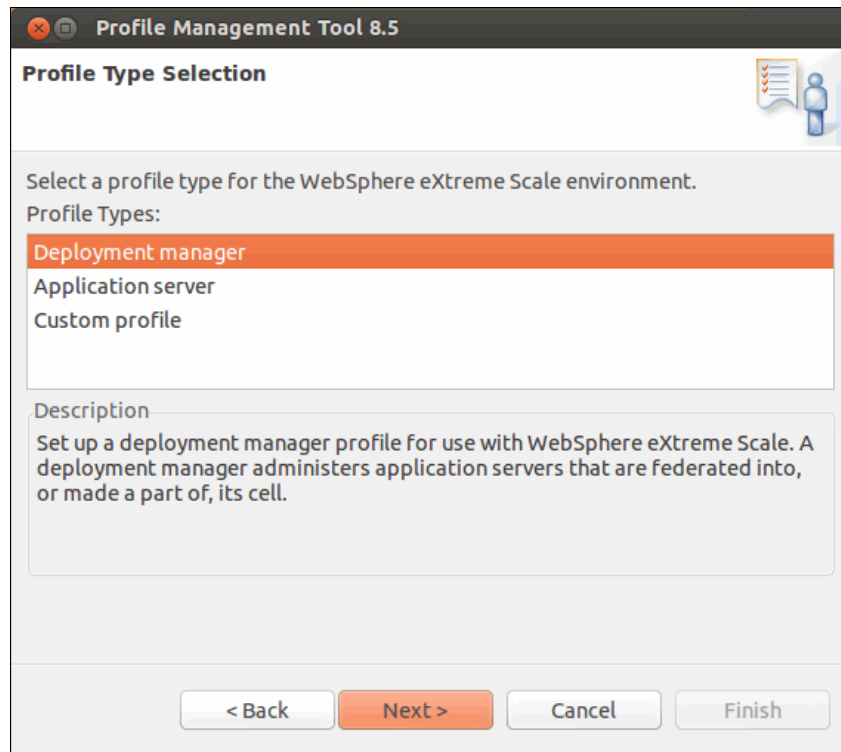


Figure 5-29 Profile type selection

- When you define a new profile, you are offered the option whether to create using a simplified path (a typical profile creation) or a more sophisticated one (advanced profile creation) as shown in Figure 5-30.

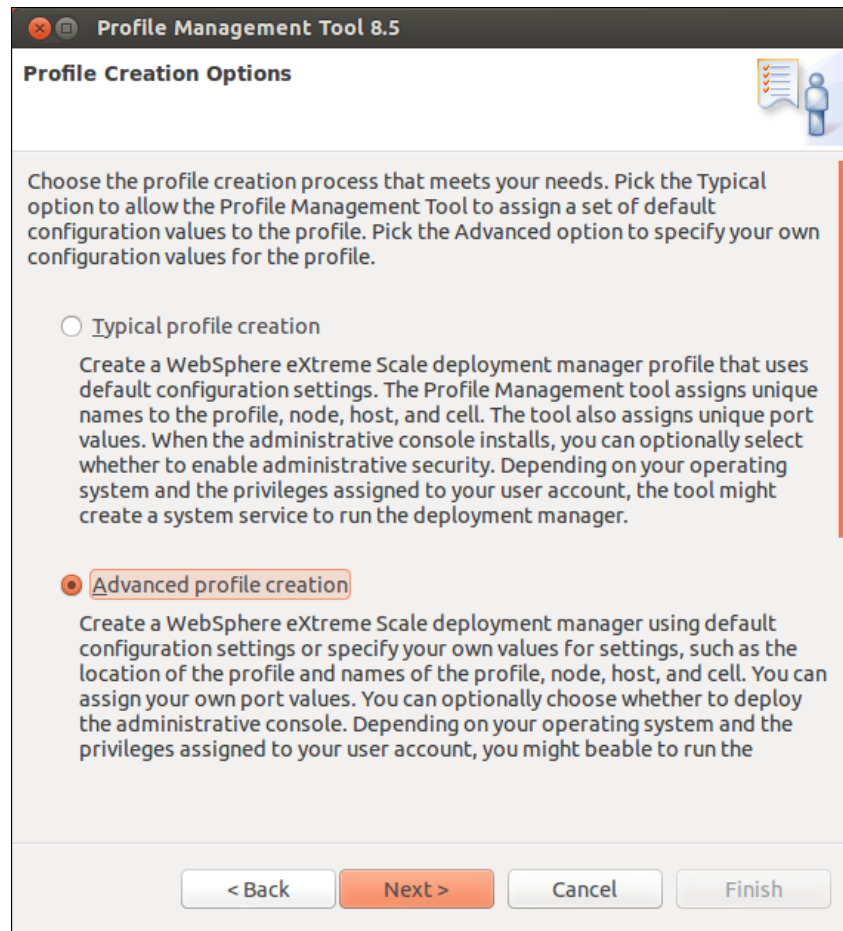


Figure 5-30 Advanced profile creation

The preferred option is **Advanced profile creation** because it allows you to define name and settings for the entire profile configuration.

Complete the creation of the five profiles on the sample topology hosts according to schema shown in the Table 5-2.

Table 5-2 Sample topology node, profile type, profile name, and node mappings

Host	Profile type	Profile name	Node name
whost01	Deployment Manager	Dmgr01	WCellManager01
whost01	Custom profile	Node01	WNode01
whost02	Custom profile	Node02	WNode02
whost03	Custom profile	Node03	WNode03
whost04	Custom profile	Node04	WNode04

For more information about profile creation, see *WebSphere Application Server V8.5 Administration and Configuration Guide for the Full Profile*, SG24-8056.

Federating profiles under the deployment manager

After you complete the product installation, product updates, and profile creation, federate the nodes under the deployment manager cell by completing these steps:

1. Start the deployment manager on whost01 by issuing the following command:

```
<WAS_install_root>/profiles/Dmgr01/bin/startManager.sh
```
2. Federate each custom profile (Node01, Node02, Node03, and Node04) into the cell by using the `<WAS_install_root>/profiles/<Profile Name>/bin/addNode.sh <dmgr host> <dmgr port>` command as shown in Example 5-41.

Example 5-41 addNode script example

```
addNode.sh whost01.rtp.ibm.com 8879
```

Important: Perform this operation on one profile at a time because the deployment manager does not support concurrent addNode operations.

3. Ensure that all node agents are up and running and the cell is fully synchronized. You can check this from the administrative console by clicking **System Administration** → **Nodes**. The result looks like Figure 5-31 where the Version column shows the installed products and version levels.

Nodes

Use this page to manage nodes in the application server environment. A node corresponds to a physical computer system with a distinct IP host address. The following table lists the managed and unmanaged node this cell. The first node is the deployment manager. Add new nodes to the cell and to this list by clicking Add Node.

⊕ Preferences

Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	WCellManager01	whost01.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	+
<input type="checkbox"/>	WNode01	whost01.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	+
<input type="checkbox"/>	WNode02	whost02.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	+
<input type="checkbox"/>	WNode03	whost03.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	+
<input type="checkbox"/>	WNode04	whost04.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	+

Figure 5-31 Sample topology after product installation and node federation

Configuring the runtime environment

After WebSphere Application Server V8.5.0.2 and WebSphere eXtreme Scale V8.6.0.1 are installed on all hosts, you can proceed with the configuration of the runtime environment.

Configuring the web server

In the sample topology, one IBM HTTP server instance called *web01* is deployed on whost01 and is configured as a local web server on the node WNode01, as shown in Figure 5-32.

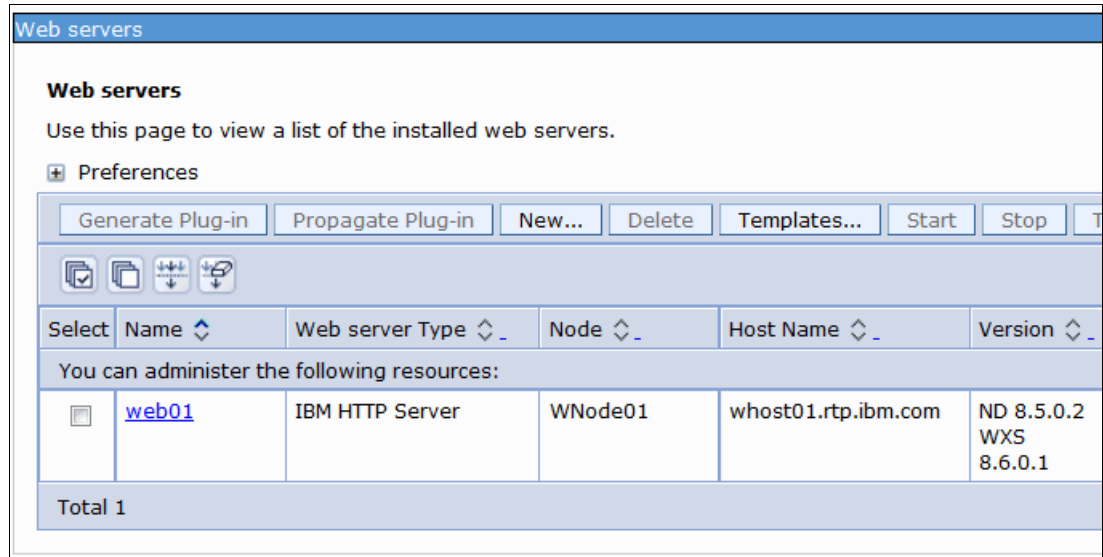


Figure 5-32 Local web server *web01*

For more information about how to configure a local web server, see the *Setting up a local web server* topic in the WebSphere Application Server Network Deployment version 8.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/t_ihs_localsetup.html

Creating the clusters

The sample topology requires three clusters:

- ▶ AppCluster hosting the getting started web application that acts as the grid client
- ▶ GridCluster containing the grid containers
- ▶ CatalogCluster hosting the catalog service

To create the clusters, complete these steps:

1. Log on to the administrative console and click **Servers** → **Clusters** → **WebSphere application server clusters** → **New**.
2. In the Step 1 window, name the cluster AppCluster.
3. Create a first cluster member app01 on node WNode01.
4. Create all required cluster members as shown in Figure 5-33 on page 205:
 - app02 on WNode02
 - app03 on WNode03
 - app04 on WNode04



Figure 5-33 Creating the AppCluster with all cluster members

- Repeat the previous steps to create a cluster called CatalogCluster on nodes WNode01, WNode02, WNode03, and WNode04.
- Repeat the previous steps to create a cluster called GridCluster on nodes WNode01, WNode02, WNode03, and WNode04.

For more information about cluster creation, see *WebSphere Application Server V8.5 Administration and Configuration Guide for the Full Profile*, SG24-8056.

Defining the catalog service domain

Catalog service domains define a group of catalog servers that manage the placement of shards and monitor the health of container servers in your data grid. In the sample topology, the CatalogCluster cluster members are part of the same catalog service domain called CSD01 as shown in Figure 5-34.

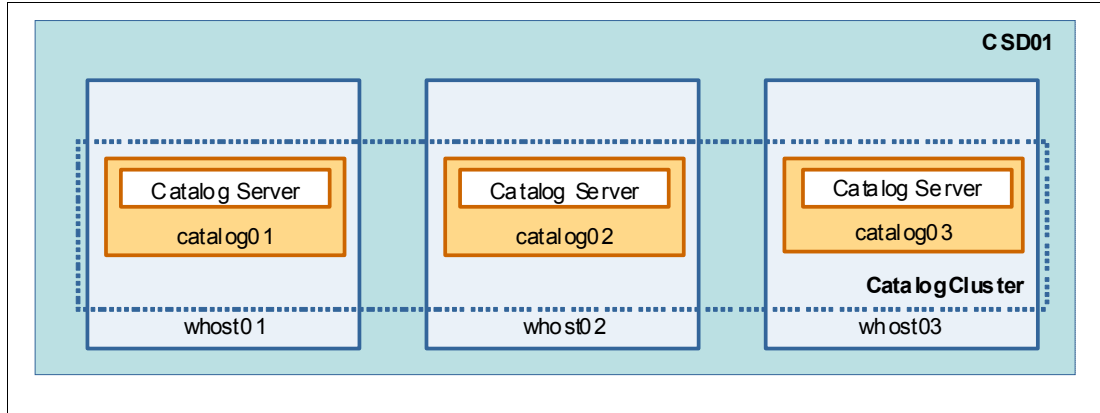


Figure 5-34 Catalog service domain

When you define a catalog service domain that places catalog servers on the application servers within the cell, the core group mechanisms of WebSphere Application Server are used. The catalog service automatically starts on the application servers in the cell. As a result, the members of a single catalog service domain cannot span the boundaries of a core group. A catalog service domain therefore cannot span cells. However, WebSphere eXtreme Scale container servers and clients can span cells by connecting to a catalog server across cell boundaries, such as a stand-alone catalog service domain or a catalog service domain embedded in another cell.

You can configure the catalog service domain by completing the following steps:

1. From the WebSphere Application Server administrative console, click **System administration** → **WebSphere eXtreme Scale** → **Catalog service domains**.
2. On the Catalog service domains window, click **New** as shown in Figure 5-35.

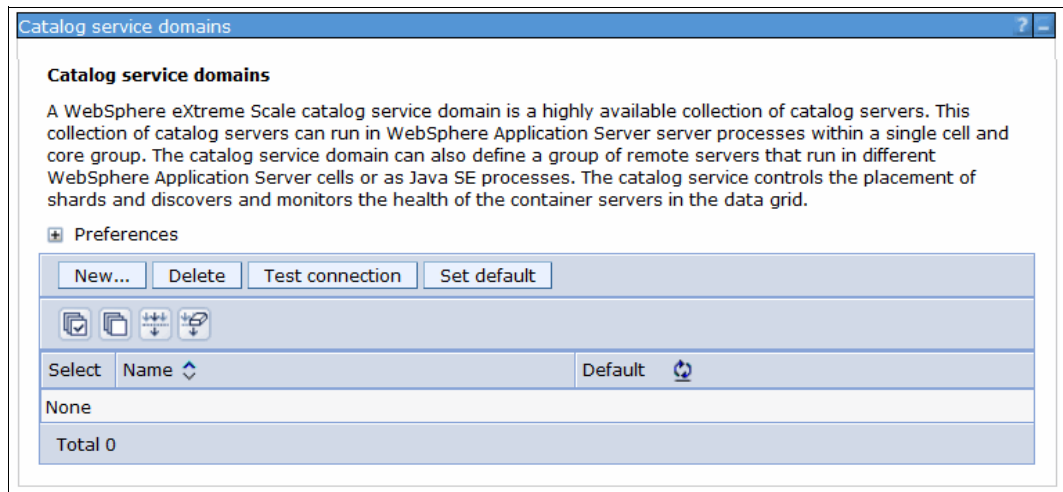


Figure 5-35 Creating a catalog service domain

- Specify CSD01 as catalog service domain name as shown in Figure 5-36.

Catalog service domains

Catalog service domains > New...

A WebSphere eXtreme Scale catalog service domain is a highly available collection of catalog servers. This collection of catalog servers WebSphere Application Server server processes within a single cell and core group. The catalog service domain can also define a group that run in different WebSphere Application Server cells or as Java SE processes. The catalog service controls the placement of shards monitors the health of the container servers in the data grid.

General Properties

* Name
CSD01

Enable this catalog service domain as the default unless another catalog service domain is explicitly specified.

Enable IBM extremeIO (XIO) communication. This configuration option is not applicable for domains that are configured with remote server members.

Catalog Servers

New Delete

Select	Catalog Server Endpoint	Client Port	Listener Port
<input checked="" type="radio"/>	Existing application server WCell01\WNode01\catalog01	6601	
<input type="radio"/>	Remote server		

Apply OK Reset Cancel

Figure 5-36 Naming the catalog service domain

- The catalog service domain CSD01 is hosted by CatalogCluster members, which are existing application servers within the cell topology. Specify a client port number for each catalog server according to the schema defined in Table 5-3.

Table 5-3 CSD01 catalog servers

Catalog server	Catalog server endpoint	Client port
catalog01	WCell01\WNode01\catalog01	6601
catalog02	WCell01\WNode02\catalog02	6602
catalog03	WCell01\WNode03\catalog03	6603

Note: By default, deployment manager is added as member of the catalog server domain. Because the catalog service is hosted by CatalogCluster members, remove the deployment manager from the catalog service domain.

- Add catalog01 as a catalog service endpoint as shown in Figure 5-36.

- Repeat the previous step to add the additional catalog servers (catalog02 and catalog03). The configuration for CSD01 looks like Figure 5-37.

Catalog service domains

[Catalog service domains](#) > CSD01

A WebSphere eXtreme Scale catalog service domain is a highly available collection of catalog servers. This collection of catalog servers can run in WebSphere Application Server server processes within a single cell and core group. The catalog service domain can also define a group of remote servers that run in different WebSphere Application Server cells or as Java SE processes. The catalog service controls the placement of shards and discovers and monitors the health of the container servers in the data grid.

Test connection

General Properties

* Name

Enable this catalog service domain as the default unless another catalog service domain is explicitly specified.

Enable IBM eXtremeIO (XIO) communication. This configuration option is not applicable for domains that are configured with remote server members.

Catalog Servers

Select	Catalog Server Endpoint	Client Port	Listener Port	Status
<input type="checkbox"/>	WCell01\WNode01\app01	6601		?
<input type="checkbox"/>	WCell01\WNode02\app02	6602		?
<input type="checkbox"/>	WCell01\WNode03\app03	6603		?

Additional Properties

- [Client security properties](#)
- [Custom properties](#)

Figure 5-37 Catalog service domain configuration

Installing grid applications

Grid applications are deployed in the sample topology shown in Figure 5-38.

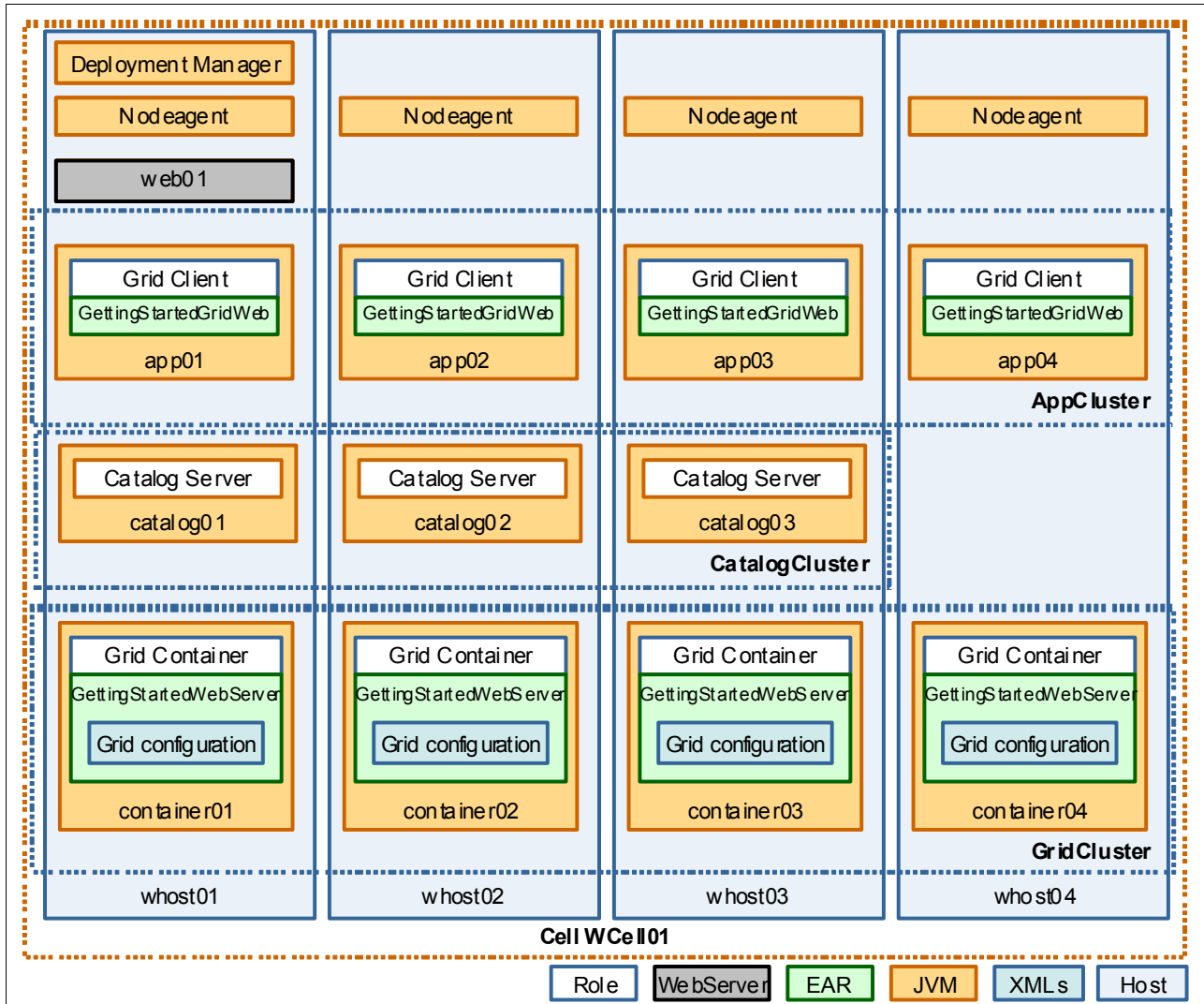


Figure 5-38 Grid applications

The mapping between applications and target servers is described in Table 5-4.

Table 5-4 Mapping between applications and servers

Application type	Enterprise Application	Target cluster / server
Grid configuration	GettingStartedWebServer.ear	GridCluster web01
Grid client	GettingStartedClient.ear	AppCluster web01

Installing grid application

As described in “Grid configuration” on page 186, the grid configuration is packaged within a standard EAR. Complete these steps to install the `GettingStartedWebServer.ear` application:

1. Log in on the WebSphere Application Server console.
2. In the left pane, expand **Applications**, and click **New Application**.
3. In the New Application window, click **Next Enterprise Application** as shown in Figure 5-39.

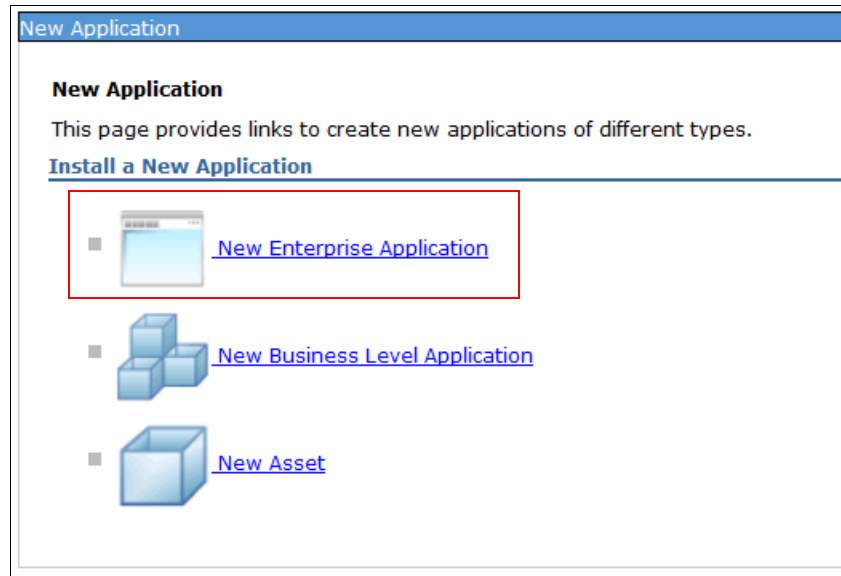


Figure 5-39 Creating a enterprise application

4. In the Preparing for the application installation window (Figure 5-40), click **Browse** and select the `GettingStartedWebServer.ear` file. Click **Next**.

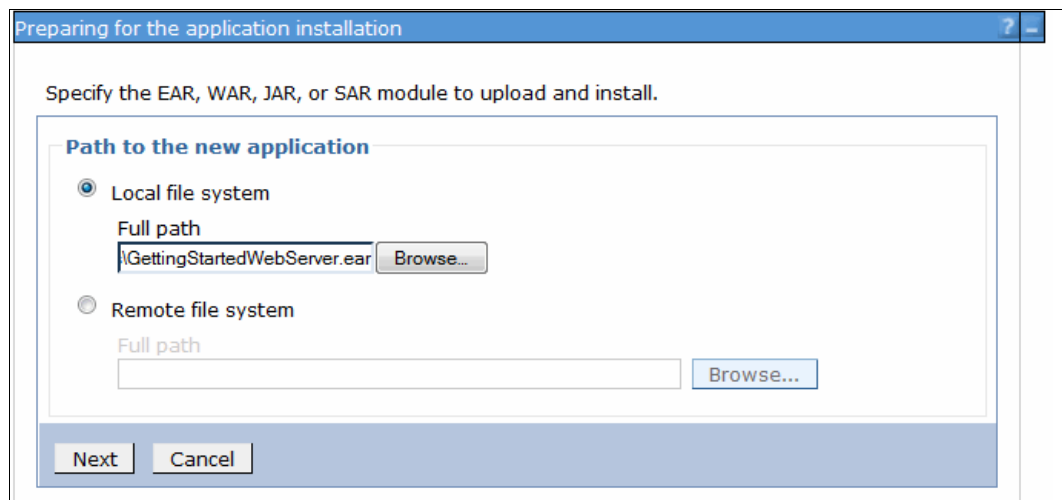


Figure 5-40 Selecting `GettingStartedWebServer.ear`

5. Select **Fast Path** and click **Next** as shown in Figure 5-41.

Preparing for the application installation

How do you want to install the application?

Fast Path - Prompt only when additional information is required.

Detailed - Show all installation options and parameters.

Choose to generate default bindings and mappings

Previous Next Cancel

Figure 5-41 Preparing for the application installation

6. On the Install New Application - Step 1: Select installation options window (Figure 5-42), accept the default installation options. Click **Next**.

Install New Application

Specify options for installing enterprise applications and modules.

→ **Step 1: Select installation options**

Step 2 Map modules to servers

✦ Step 3 Map virtual hosts for Web modules

✦ Step 4 Metadata for modules

Step 5 Summary

Select installation options

Specify the various options that are available for your application.

Precompile JavaServer Pages files

Directory to install application

Distribute application

Use Binary Configuration

Deploy enterprise beans

Application name

GettingStartedWebServerEAR

Application edition

Edition description

Create MBeans for resources

Override class reloading settings for Web and EJB modules

Reload interval in seconds

Deploy Web services

Validate Input off/warn/fail

warn

Process embedded configuration

Figure 5-42 Installing a new application, step 1, selecting the installation options

7. On the Install New Application - Step 2: New modules to servers window (Figure 5-43), complete the following steps to install the GettingStartedWebServer modules on GridCluster and web01 as summarized in Figure 5-43.
 - a. From the Clusters and servers list, select:
 - **WebSphere:cell=WCell01,node=WNode01,server=web01**
 - **WebSphere:cell=WCell01,cluster=GridCluster**
 Press Ctrl+Shift to select both the cluster (GridCluster) and web server (web01).
 - b. Select the **GettingStartedWebServer** module.
 - c. Click **Apply**, then click **Next**.

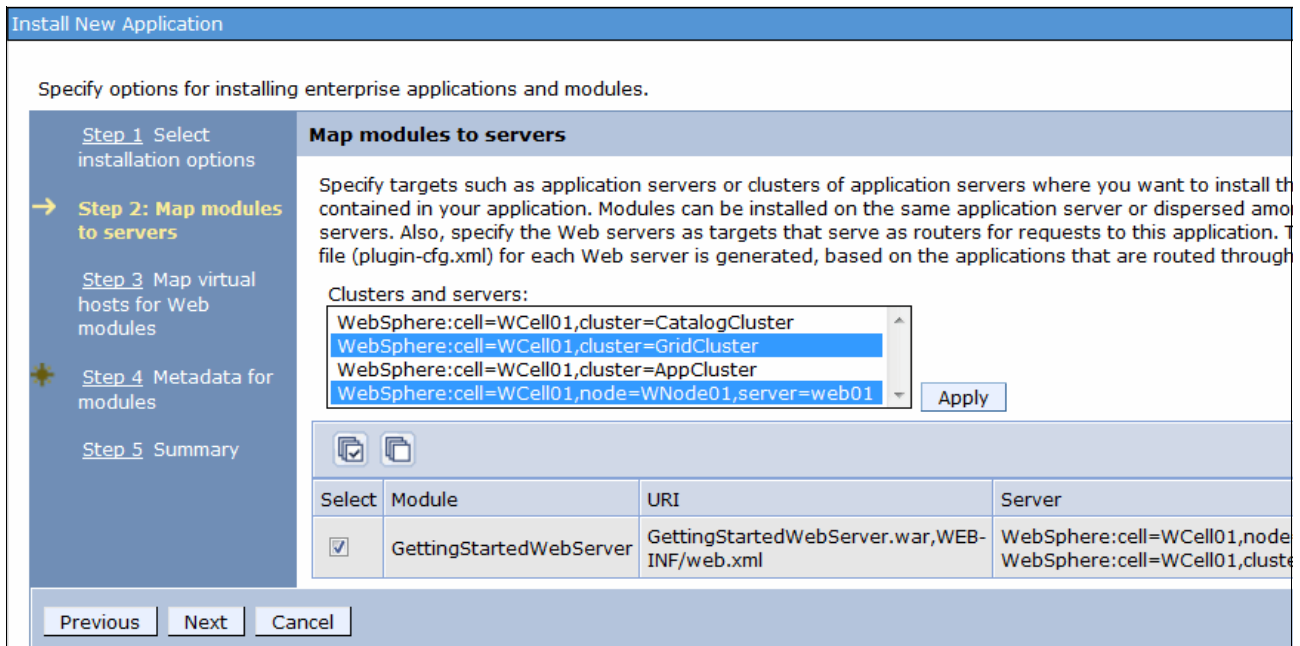


Figure 5-43 Grid application installation on AppCluster and web01

8. On the Install New Application - Step 3: Map virtual hosts for web modules window (Figure 5-44), accept the default virtual hosts for web module. Click **Next**.

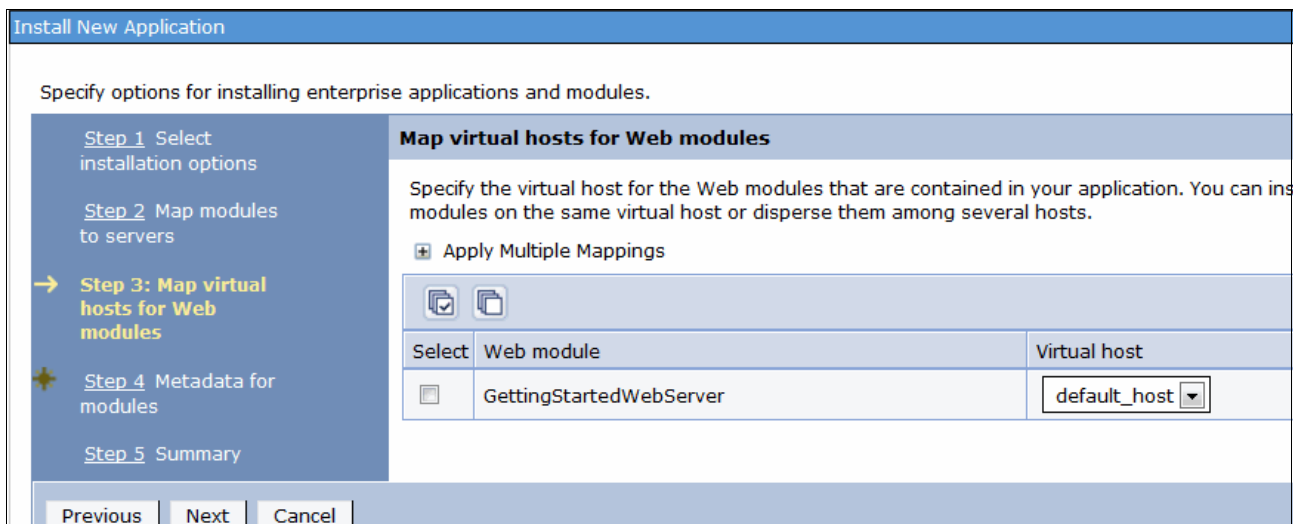


Figure 5-44 Virtual host configuration

9. On the Install New Application - Step 4: Metadata for modules window (Figure 5-45), accept the default metadata for modules. Click **Next**.

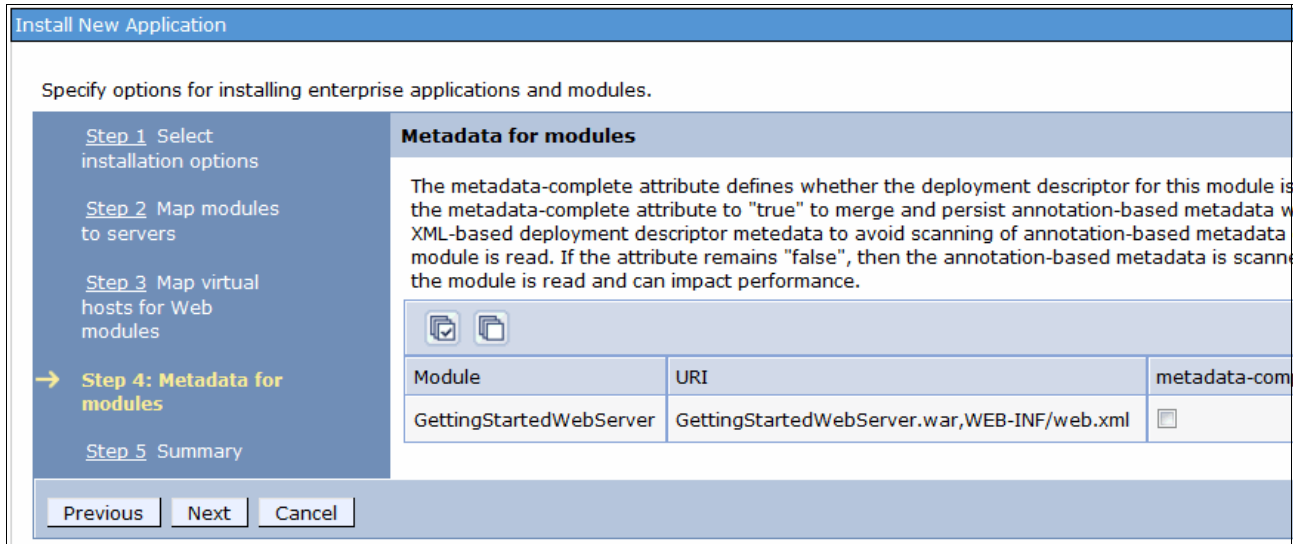


Figure 5-45 Metadata configuration

10. On the Install New Application - Step 5: Summary window (Figure 5-46), verify that all the information provided is consistent with Figure 5-46, then click **Finish**.

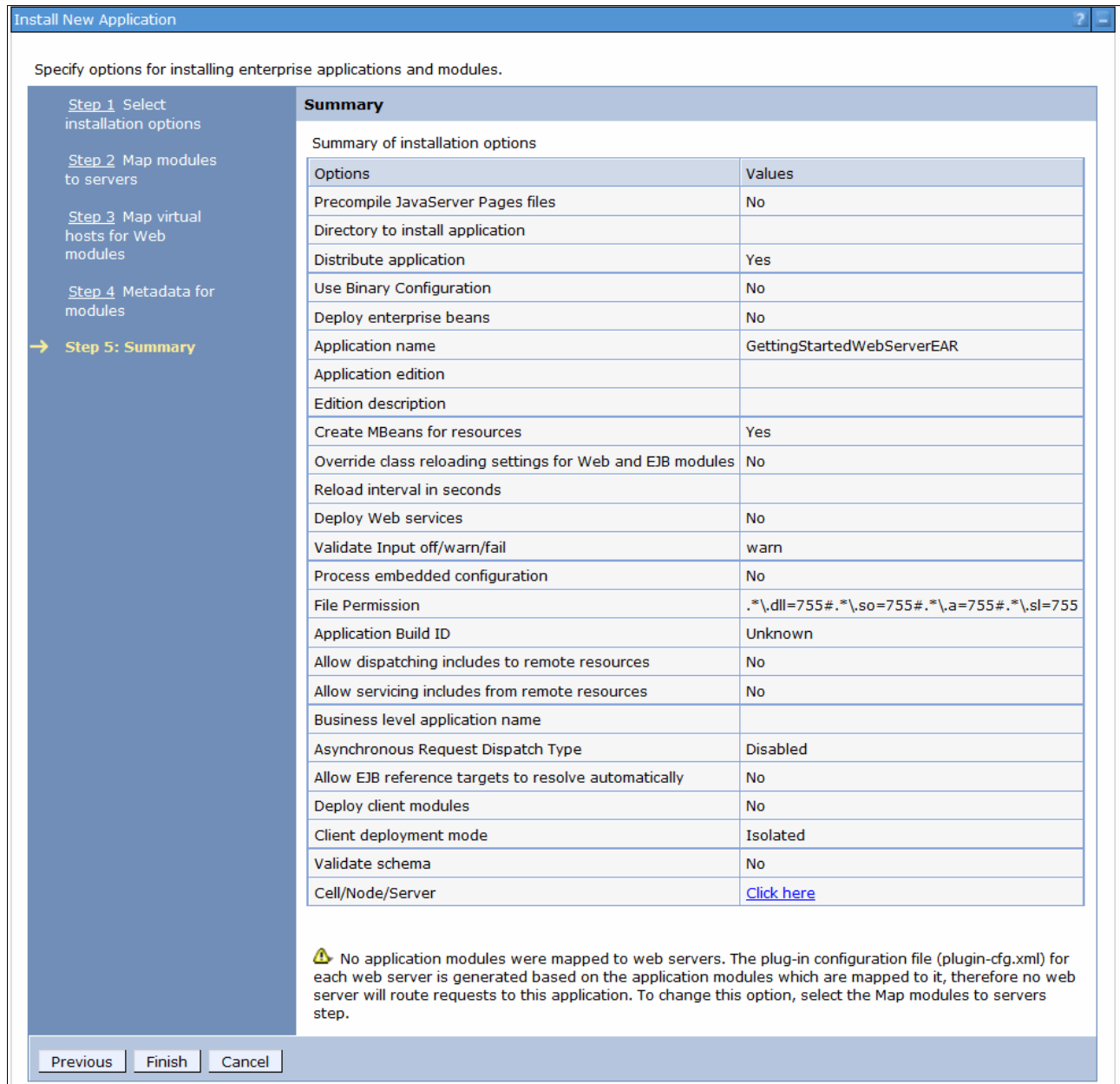


Figure 5-46 Installing a new application summary

11. After the application has been installed, click the **Save** link to save the application to the master configuration as shown in Figure 5-47.

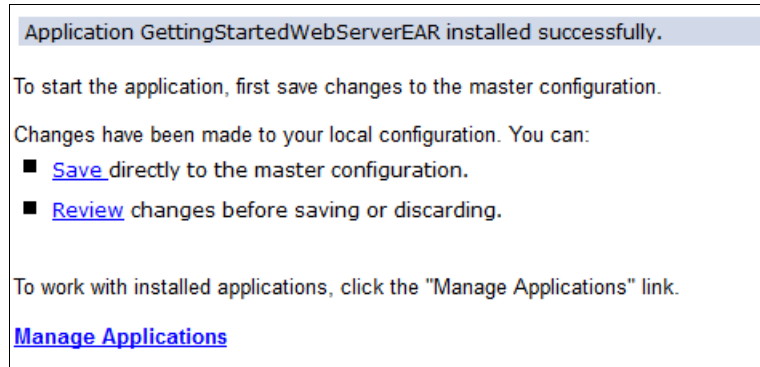


Figure 5-47 Saving the new application to the master configuration

Installing the grid client application

The client application is packaged as a standard EAR called `GettingStartedClient.ear`. Follow the steps in “Installing grid application” on page 210 to install the `GettingStartedClient.ear` on `web01` and `AppCluster` as shown in Figure 5-48.

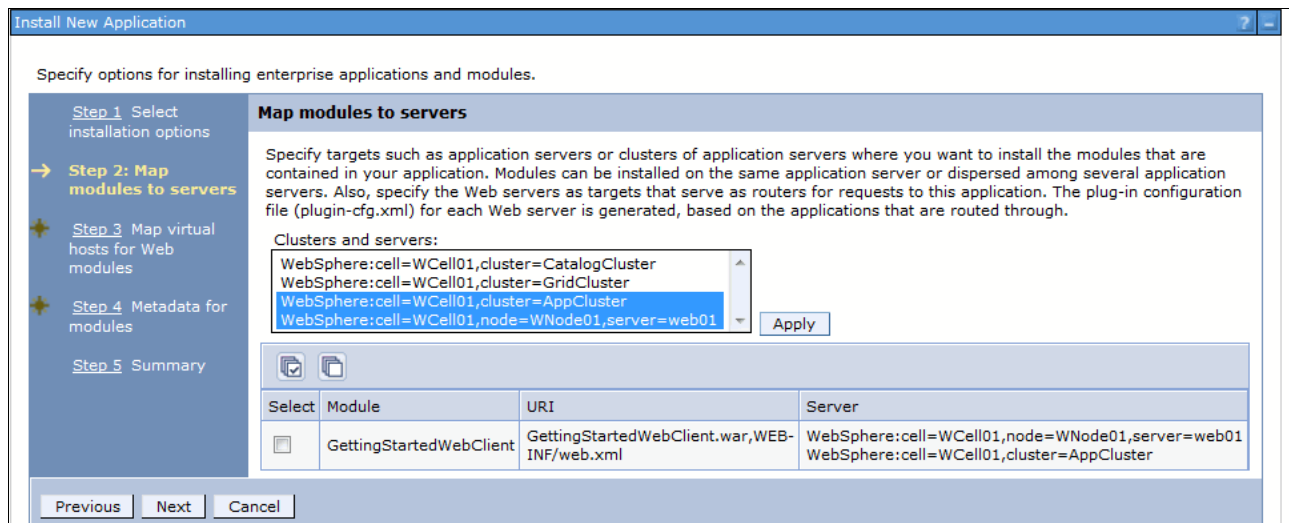


Figure 5-48 Installing `GettingStartedWebClient.ear` on `web01` and `AppCluster`

Starting the runtime environment

After you complete the configuration of the integrated environment and deploy the grid and client applications, you can start the catalog, container, and application clusters.

Starting the catalog cluster

1. Log in on the WebSphere Application Server administrative console.
2. In the left navigation pane, click **Servers** → **Clusters** → **WebSphere application server clusters**. You can check the cluster runtime status as shown in Figure 5-49. Select **CatalogCluster** and click **Start**.

The screenshot displays the 'WebSphere application server clusters' administrative console. At the top, there is a title bar and a header section with the title 'WebSphere application server clusters' and a brief description of what a server cluster is. Below this, there is a 'Preferences' section with several action buttons: 'New...', 'Delete', 'Start', 'Stop', 'Ripplestart', and 'ImmediateStop'. A toolbar with icons for selection and navigation is also present. The main area is a table with columns for 'Select', 'Name', and 'Status'. The table lists three clusters: 'AppCluster', 'CatalogCluster', and 'GridCluster'. Each cluster has a checkbox in the 'Select' column and a red 'X' icon in the 'Status' column, indicating that all clusters are currently stopped. A 'Total 3' summary is shown at the bottom of the table.

Select	Name	Status
<input type="checkbox"/>	AppCluster	✘
<input type="checkbox"/>	CatalogCluster	✘
<input type="checkbox"/>	GridCluster	✘
Total 3		

Figure 5-49 Clusters runtime status

- The start operation on the CatalogCluster catalog is started as shown in Figure 5-50. Wait for CatalogCluster to be up and running as shown in Figure 5-52 on page 218. You can monitor runtime status for CatalogCluster by refreshing the status view.

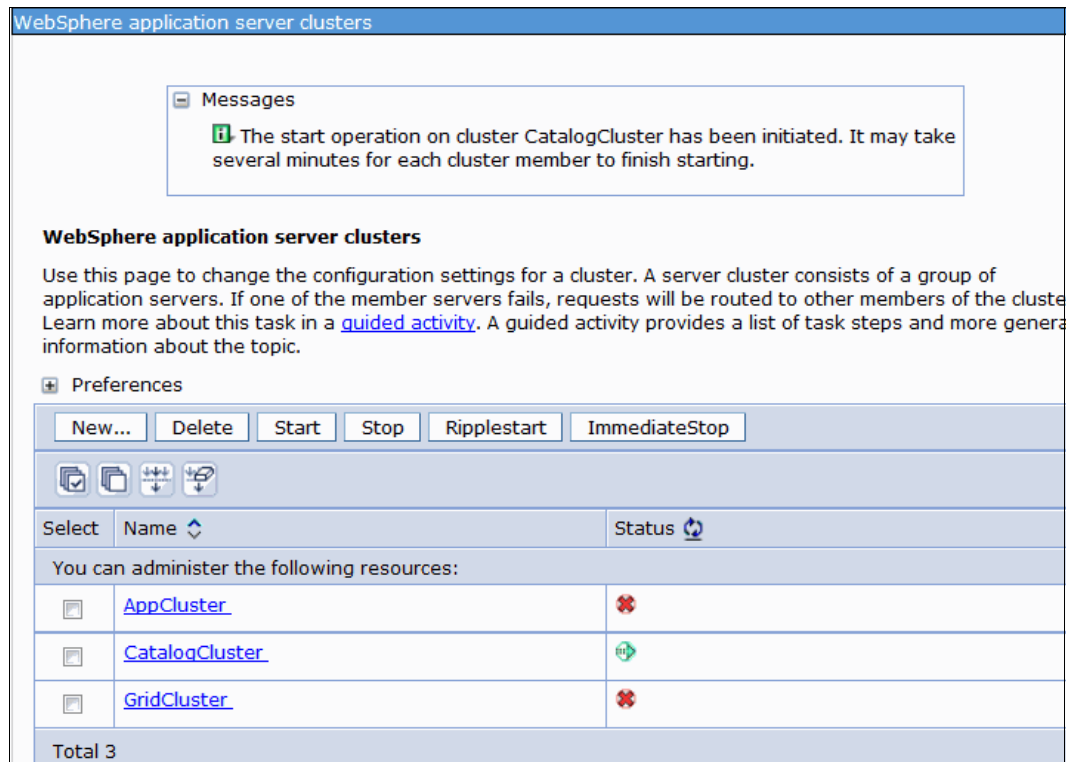


Figure 5-50 Starting the CatalogCluster

- Check the status of the Catalog Service domain. In the left navigation pane, click **System Administration** → **WebSphere eXtreme Scale** → **Catalog Service Domains**. The Catalog Service Domain list contains CSD01 configured in “Defining the catalog service domain” on page 206.

- On the Catalog service domains window, select **CSD01** and click **Test Connection** to verify that CSD01 is operating correctly as shown in Figure 5-51.

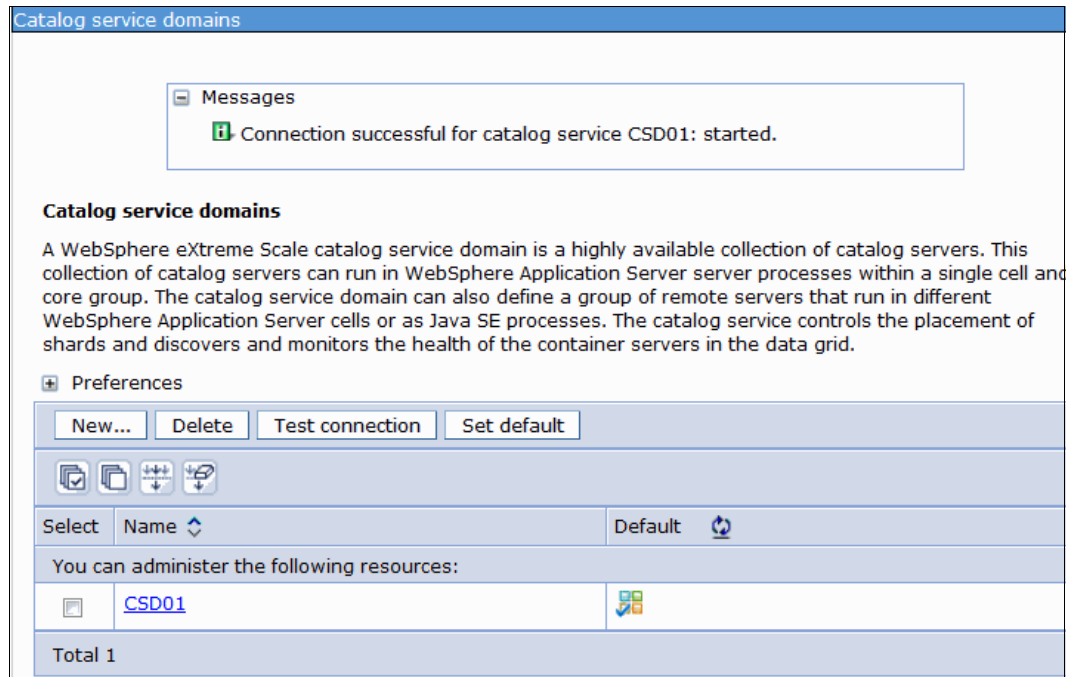


Figure 5-51 Testing connection to CSD01

Starting the GridCluster

The GridCluster hosts the grid configuration. You can start the GridCluster by following the steps in “Starting the catalog cluster” on page 216. Wait for GridCluster to start as shown in Figure 5-52.

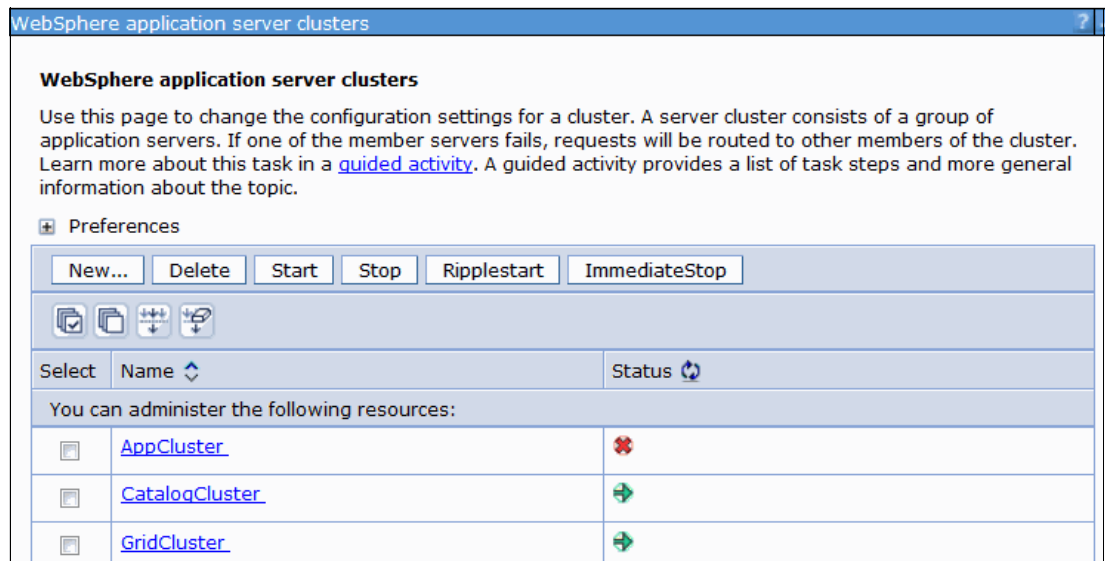


Figure 5-52 Starting the GridCluster

Container servers start receiving shard placement work from catalog servers as soon as they become available. The excerpt from the SystemOut.log file for container01 shown in Example 5-42 demonstrates that three partitions (Grid:mapSet:1, Grid:mapSet:2, and Grid:mapSet:12) are placed in container01.

Example 5-42 Shard placement work assigned from catalog server to container servers

```
[5/10/13 16:35:39:111 EDT] 00000001 WsServerImpl A WSVR0001I: Server
container01 open for e-business
```

```
5/10/13 16:35:54:032 EDT] 000000c5 ObjectGridCon I CWOBJ7509I: Placement work,
workId 1, from catalog server for partition Grid:mapSet:1 intended for container
WCell01\WNode01\container01_C-5 was received.
```

```
[5/10/13 16:35:54:039 EDT] 000000c6 ObjectGridCon I CWOBJ7509I: Placement work,
workId 2, from catalog server for partition Grid:mapSet:2 intended for container
WCell01\WNode01\container01_C-5 was received.
```

```
[5/10/13 16:35:54:040 EDT] 000000c4 ObjectGridCon I CWOBJ7509I: Placement work,
workId 12, from catalog server for partition Grid:mapSet:12 intended for
container WCell01\WNode01\container01_C-5 was received.
```

Containers run shard placement operations that are requested by the catalog. Example 5-43 shows an excerpt of the container01 SystemOut.log that shows placement execution (shard activation) for Grid:mapSet:1, Grid:mapSet:2, and Grid:mapSet:12.

Example 5-43 Shard activation run by container01

```
...
[5/10/13 16:35:54:520 EDT] 000000cb SynchronousRe I CWOBJ1511I: Grid:mapSet:1
(synchronous replica) is open for business.
```

```
...
[5/10/13 16:35:57:555 EDT] 000000cb SynchronousRe I CWOBJ1511I: Grid:mapSet:2
(synchronous replica) is open for business.
```

```
...
[5/10/13 16:36:00:618 EDT] 000000cb SynchronousRe I CWOBJ1511I: Grid:mapSet:12
(synchronous replica) is open for business.
```

Starting the application cluster

The AppCluster hosts the grid client configuration. You can start the AppCluster by following the steps in “Starting the catalog cluster” on page 216. Wait for AppCluster to start as shown in Figure 5-53.

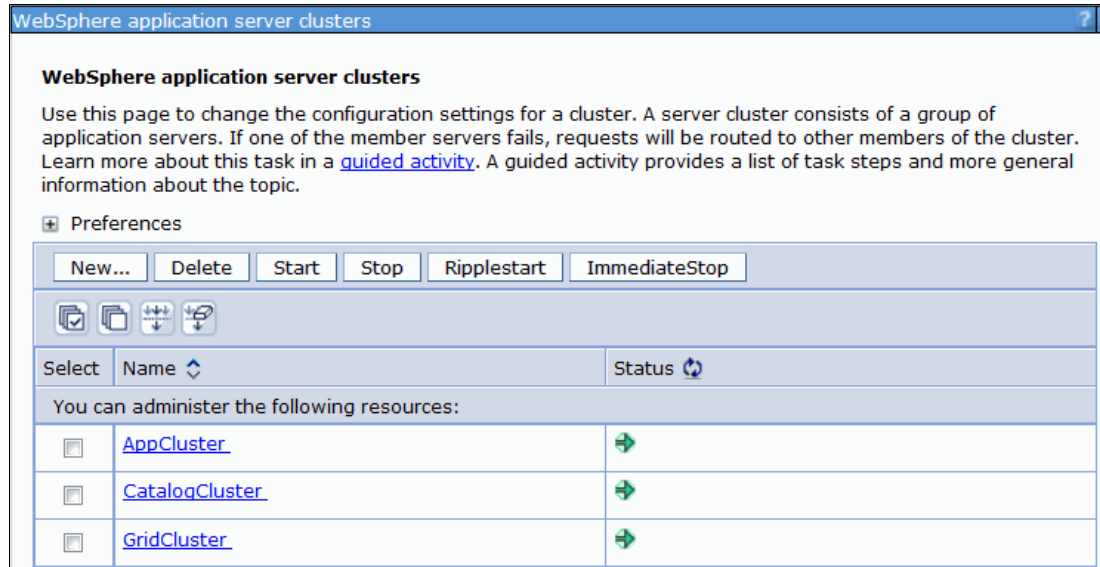


Figure 5-53 Starting the AppCluster

Starting the web server

The web server web01 is a single entry point to access to the grid applications. Because the getting started web application deployed on AppCluster is accessed through web01, generate a plug-in configuration to manage this routing. To generate plug-in configuration and start web01, complete these steps.

1. Log on to the WebSphere Application Server administrative console and click **Server Types** → **Web servers**.
2. The web server list contains the web server *web01*. Click **Generate Plug-in** and then **Start**. Wait for web01 to complete the start as shown in Figure 5-54.

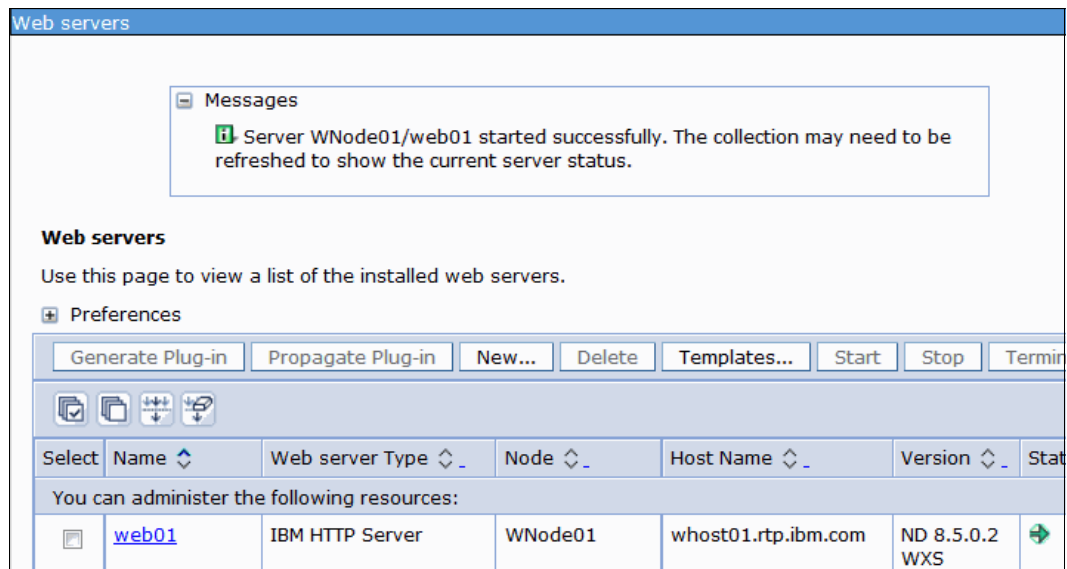


Figure 5-54 Generating plug-in and starting web01

5.2.7 Validating the sample topology

The `xscmd` command can be used to validate the operating status of the grid. The `xscmd` script must be run from the node directory by using the options shown in Example 5-44.

Example 5-44 `xscmd` options

```
<WAS_install_root>/profiles/<Node name>/bin/xscmd.sh [with the following options]
  -cep <host:XIO_listening_port,...>
  -c <command>
```

When WebSphere eXtreme Scale is integrated into WebSphere Application Server, the `xscmd` script is available in different locations:

- ▶ `<WAS_install_root>/bin` as part of WebSphere Application Server binary
- ▶ `<WAS_install_root>/profiles/<Profile name>/bin` as part of profile binary

The `xscmd` script starts the `setupCmdLine` script to set up the correct execution context before issuing a JMX call against the catalog server. Because some relevant configuration information used by `setupCmdLine` (for example, security) is defined within node profiles, start `xscmd` from one of the custom node profiles.

Determining XIO ports

The XIO listening port is dynamically assigned to catalog servers when they were created as `CatalogCluster` members. You can determine the XIO listening port for the `catalog01` by completing the following steps:

1. Log in to the Deployment Manager console and click **Servers** → **Server Types** → **WebSphere Application Servers** to display the list of servers as shown in Figure 5-55.

Select	Name	Node	Host Name	Version	Cluster Name	Status
<input type="checkbox"/>	app01	WNode01	whost01.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	AppCluster	➔
<input type="checkbox"/>	app02	WNode02	whost02.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	AppCluster	➔
<input type="checkbox"/>	app03	WNode03	whost03.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	AppCluster	➔
<input type="checkbox"/>	app04	WNode04	whost04.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	AppCluster	➔
<input type="checkbox"/>	catalog01	WNode01	whost01.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	CatalogCluster	➔
<input type="checkbox"/>	catalog02	WNode02	whost02.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	CatalogCluster	➔
<input type="checkbox"/>	catalog03	WNode03	whost03.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	CatalogCluster	➔
<input type="checkbox"/>	container01	WNode01	whost01.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	GridCluster	➔
<input type="checkbox"/>	container02	WNode02	whost02.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	GridCluster	➔
<input type="checkbox"/>	container03	WNode03	whost03.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	GridCluster	➔

Figure 5-55 List of available servers

2. Click **catalog01** server to display application server properties.
3. Under the Communications section, click **Ports** to display the list of TCP/IP ports that are assigned to this server as shown in Figure 5-56.

Port Name	Port Number
BOOTSTRAP_ADDRESS	9811
SOAP_CONNECTOR_ADDRESS	8881
ORB_LISTENER_ADDRESS	9102
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9407
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	9408
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	9409
WC_adminhost	9062
WC_defaulthost	9081
DCS_UNICAST_ADDRESS	9355
WC_adminhost_secure	9045
WC_defaulthost_secure	9444
SIP_DEFAULTHOST	5062
SIP_DEFAULTHOST_SECURE	5063
OVERLAY_UDP_LISTENER_ADDRESS	11009
OVERLAY_TCP_LISTENER_ADDRESS	11010
IPC_CONNECTOR_ADDRESS	9634
SIB_ENDPOINT_ADDRESS	7278
SIB_ENDPOINT_SECURE_ADDRESS	7287
SIB_MQ_ENDPOINT_ADDRESS	5559
SIB_MQ_ENDPOINT_SECURE_ADDRESS	5579
XIO_ADDRESS	4811

Figure 5-56 List of TCP/IP ports that are assigned to the catalog01 server

4. Take note of the XIO port, called XIO_ADDRESS (port 4811 in this example).
5. Repeat these steps for the other catalog servers (catalog02 and catalog03).

In the sample topology, XIO ports are configured as summarized in Table 5-5.

Table 5-5 XIO ports

Catalog server	Hostname	XIO port
catalog01	whost01.rtp.ibm.com	4811
catalog02	whost02.rtp.ibm.com	4812
catalog03	whost03.rtp.ibm.com	4814

Multiple catalog service endpoints can be passed to `xscmd` using the `-cep` parameter. The `xscmd` command tries to establish a connection to the first available catalog server as specified in the list passed as a `-cep` argument. Although this feature is useful to improve the resiliency of the `xscmd` command, this section covers these commands that are issued against a single catalog server (catalog01):

- ▶ `showInfo`
- ▶ `showPlacement`
- ▶ `routetable`

User credentials

The `xscmd` command communicates with WebSphere Administrative Services by using the Remote Method Invocation (RMI). When WebSphere Global Security is enabled, a valid WebSphere Application Server credential is required to run the `xscmd` command. The provided user credential must have at least the operator role.

For more information about administrative roles, see the *Administrative roles* topic in the WebSphere Application Server Network Deployment V8.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.doc/ae/rsec_adminroles.html

By default, `xscmd` prompts the user to provide user name and password from the command line. To simplify `xscmd` command execution, consider the option to store user credentials within `sas.client.props` file as outlined in the following steps:

1. Set the properties shown in the Example 5-45 in the `<WAS_install_root>/profiles/<Profile name>/sas.client.props` file.

Example 5-45 Setting user credentials in the sas.client.props file

```
com.ibm.CORBA.loginSource=properties
com.ibm.CORBA.loginUserid=wasadmin
com.ibm.CORBA.loginPassword=passw0rd
```

2. Encode the user password by issuing the following commands:

```
<WAS_install_root>/profiles/<Profile name>/PropFilePasswordEncoder.sh
<WAS_install_root>/profiles/<Profile name>/sas.client.props
com.ibm.CORBA.loginPassword
```

showInfo command

The `showInfo` command is useful to validate that the running WebSphere eXtreme Scale services are consistent with the topology to be implemented. Example 5-46 shows the output of `showInfo` that contains information about the environment specifications, including the installed product version and JVM information.

Example 5-46 showInfo command output

```
<WAS_install_root>/profiles/Node01/bin/xscmd.sh -cep whost01.rtp.ibm.com:4811 -c
showInfo
*** Server Name: WCell01\WNode01\catalog01 ***
Client Port                6601 (configured)
Domain name                 CSD01
Host name                   9.42.171.46
IP Address                  9.42.171.46
JMX Connector Port         4811 (derived)
JMX Service Port           4811 (configured)
JMX Service URL             service:jmx:iiop:
```

```

//9.42.171.46/jndi/corbaname:
iiop:9.42.171.46:
9811/WsnAdminNameService#JMXCon
nector
JAVA Directory /opt/IBM/WebSphere/AppServer85/
java/jre
JAVA Vendor IBM Corporation
JAVA Version 1.6.0
JAVA Bit Mode 64
JVM Runtime Version pxa6460_26sr5ifx-20130308_02
(SR5)
JVM Version JRE 1.6.0 Linux amd64-64
Compressed References
20130204_137148 (JIT enabled,
AOT enabled), J9VM -
R26_Java626_SR5_20130204_0851_B
137148, JIT -
r11.b03_20130131_32403, GC -
R26_Java626_SR5_20130204_0851_B
137148_CMPRSS, J9CL -
20130204_137148
Listener Port (XIO) 4811 (configured)
OS Architecture amd64
Operating System Linux
Operating System Version 3.5.0-17-generic
Process ID 17868
Peer Port 0 (configured)
Server Type Catalog Server
Time Stamp from Server 05/17/13 13:18:45.267 EDT
Transport eXtremeIO
WebSphere Application Server Product Directory /opt/IBM/WebSphere/AppServer85
IBM WebSphere Application Server - ND Version 8.5.0.2
WebSphere Application Server Full Server Name WCell01\WNode01\catalog01
IBM WebSphere Application Server - XD Version 8.6.0.1
WebSphere eXtreme Scale Version v7.0.0 (8.6.0.0)
[cf11305.31182928]
XIO Timeout (seconds) 30 (default)

*** Server Name: WCell01\WNode02\catalog02 ***
...
*** Server Name: WCell01\WNode03\catalog03 ***
...
*** Server Name: WCell01\WNode01\container01 ***
...
*** Server Name: WCell01\WNode02\container02 ***
...
*** Server Name: WCell01\WNode03\container03 ***
...
*** Server Name: WCell01\WNode04\container03 ***

```

showPlacement command

The **showPlacement** command returns information about the list of all containers and shards that are deployed on the hosts that comprise the grid. Such information is a representation of the primary and replica shard topology as *planned* by the catalog service. Example 5-47 shows an example of placement information across all four containers (container01 to container04).

Example 5-47 showPlacement command output

```
<WAS_install_root>/profiles/Node01/bin/xscmd.sh -cep whost01.rtp.ibm.com:4811 -c showPlacement  
Starting at: 2013-05-20 07:22:51.108
```

```
CWXSIO068I: Executing command: showPlacement
```

```
Command showPlacement is a technology preview. The command usage and output is subject to  
change.
```

```
*** Show all online container servers for Grid data grid and mapSet map set.
```

```
Host: 9.42.171.46
```

```
Container: WCell01\WNode01\container01_C-2, Server:WCell01\WNode01\container01,  
Zone:DefaultZone
```

Partition	Shard	Type	Reserved
3		Primary	false
4		Primary	false
5		Primary	false
1		SynchronousReplica	false
2		SynchronousReplica	false
10		SynchronousReplica	false
12		SynchronousReplica	false

```
Container: WCell01\WNode03\container03_C-2, Server:WCell01\WNode03\container03,  
Zone:DefaultZone
```

Partition	Shard	Type	Reserved
6		Primary	false
7		Primary	false
8		Primary	false
3		SynchronousReplica	false
4		SynchronousReplica	false
5		SynchronousReplica	false

```
Host: 9.42.171.47
```

```
Container: WCell01\WNode02\container02_C-2, Server:WCell01\WNode02\container02,  
Zone:DefaultZone
```

Partition	Shard	Type	Reserved
1		Primary	false
2		Primary	false
12		Primary	false
0		SynchronousReplica	false
8		SynchronousReplica	false
9		SynchronousReplica	false
11		SynchronousReplica	false

```
Container: WCell01\WNode04\container04_C-2, Server:WCell01\WNode04\container04,  
Zone:DefaultZone
```

Partition	Shard	Type	Reserved
0		Primary	false
9		Primary	false

```

10      Primary          false
11      Primary          false
6       SynchronousReplica false
7       SynchronousReplica false

```

```

Number of containers matching = 4
Total known containers       = 4
Total known hosts           = 4

```

CWXSIO040I: The showPlacement command completed successfully.
Ending at: 2013-05-20 07:23:02.013

routetable command

The **routetable** command is used to display the routing table information managed by the catalog service and pushed to the clients (Example 5-48). This table is a representation of the *actual* placement of primary and replica shards across available containers. The actual placement can differ during placement operations from the information returned from **showPlacement** command, which shows the *planned* placement.

The **routetable** command also provides useful information about the status of primary and replica shards. The **xscmd** command uses the routing table to check whether partitions are reachable or not from the workstation where the **xscmd** program is.

Example 5-48 routetable command output

```

<WAS_install_root>/profiles/Node01/bin/xscmd.sh -cep whost01.rtp.ibm.com:4811 -c routetable
Starting at: 2013-05-20 07:25:25.645
CWXSIO068I: Executing command: routetable

```

```

Realm/Cell Name: defaultWIMFileBasedRealm
User Identity: wasadmin
User Password:
*** Displaying routing information for data grid: Grid:mapSet
Placement Scope: Domain

```

Shard Type	Partition	State	Host	Zone	Container
Primary	0	reachable	9.42.171.49	DefaultZone	WCell01\WNode04\container04_C-2
SynchronousReplica	0	reachable	9.42.171.47	DefaultZone	WCell01\WNode02\container02_C-2
Primary	1	reachable	9.42.171.47	DefaultZone	WCell01\WNode02\container02_C-2
SynchronousReplica	1	reachable	9.42.171.46	DefaultZone	WCell01\WNode01\container01_C-2
Primary	2	reachable	9.42.171.47	DefaultZone	WCell01\WNode02\container02_C-2
SynchronousReplica	2	reachable	9.42.171.46	DefaultZone	WCell01\WNode01\container01_C-2
Primary	3	reachable	9.42.171.46	DefaultZone	WCell01\WNode01\container01_C-2
Primary	4	reachable	9.42.171.46	DefaultZone	WCell01\WNode01\container01_C-2
Primary	5	reachable	9.42.171.46	DefaultZone	WCell01\WNode01\container01_C-2
Primary	6	reachable	9.42.171.48	DefaultZone	WCell01\WNode03\container03_C-2
Primary	7	reachable	9.42.171.48	DefaultZone	WCell01\WNode03\container03_C-2
Primary	8	reachable	9.42.171.48	DefaultZone	WCell01\WNode03\container03_C-2
Primary	9	reachable	9.42.171.49	DefaultZone	WCell01\WNode04\container04_C-2
SynchronousReplica	9	reachable	9.42.171.47	DefaultZone	WCell01\WNode02\container02_C-2
Primary	10	reachable	9.42.171.49	DefaultZone	WCell01\WNode04\container04_C-2
SynchronousReplica	10	reachable	9.42.171.46	DefaultZone	WCell01\WNode01\container01_C-2
Primary	11	reachable	9.42.171.49	DefaultZone	WCell01\WNode04\container04_C-2
SynchronousReplica	11	reachable	9.42.171.47	DefaultZone	WCell01\WNode02\container02_C-2
Primary	12	reachable	9.42.171.47	DefaultZone	WCell01\WNode02\container02_C-2
SynchronousReplica	12	reachable	9.42.171.46	DefaultZone	WCell01\WNode01\container01_C-2

5.2.8 Testing the sample topology and application

The getting started web application is a grid application that establishes a connection to the catalog service through a catalog service domain. It retrieves a description of server topology, and then communicates directly with each container server as required to run create, read, update, and delete operations.

Figure 5-57 shows the sample topology that is used in the example.

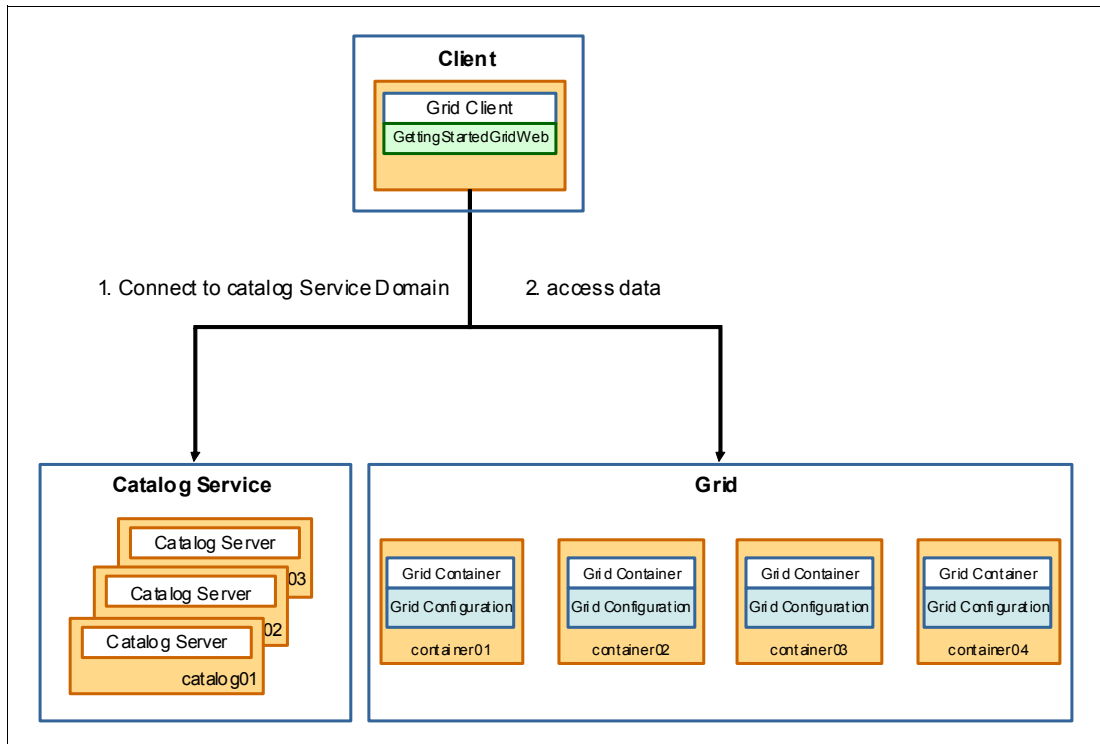


Figure 5-57 Grid client, catalog service, and grid

In this section, test the grid using the getting started web application by completing the following steps:

1. Access the application by establishing a connection to the following URL:
<http://whost01.rtp.ibm.com:8080/GettingStartedWebClient/>

The default input page for getting started web application is displayed, as shown in Figure 5-58.

The screenshot shows the 'WebSphere eXtreme Scale Getting Started Client' web application. The page has a dark header with 'IBM WebSphere eXtreme Scale Samples' and the IBM logo. Below the header is a blue navigation bar with 'WebSphere eXtreme Scale Getting Started Sample'. The main content area is titled 'WebSphere eXtreme Scale Getting Started Client' and contains the following elements:

- A paragraph: 'This sample servlet provides simple CRUD operations against an eXtreme Scale grid.'
- A paragraph: 'This sample servlet can also be used for simple CRUD operations against an XC10 DataPower appliance Simple Grid.'
- A section titled 'Catalog Service Endpoints' with two radio buttons: 'Endpoints' (selected) and 'Catalog Service Domain ID'. Below these are input fields for 'Endpoints' (containing 'whost01.rtp.ibm.com:4811'), 'Remote: <host>:<port>', and 'Embedded: <blank>'.
- A section titled 'Grid' with input fields for 'Grid' (containing 'Grid') and 'Map' (containing 'Map1'). A note next to the Grid field says '(For XC10 use Simple Grid Name)'.
- A status bar showing a 'Disconnect' button and the text 'connected to Grid on whost01.rtp.ibm.com:4811,whost02.rtp.ibm.com:4811,whost03.rtp.ibm.com:4814'.
- A section titled 'CRUD Operation' with input fields for 'Key' and 'Value' (with a note '(Used for Insert and Update only)'). Below these are four buttons: 'Insert', 'Update', 'Delete', and 'Get'.

Figure 5-58 Getting started web application

2. Connect the web client to the grid by specifying CSD01 as the Catalog Service Domain ID and clicking **Connect** as shown in Figure 5-59.

WebSphere eXtreme Scale Getting Started Client

This sample servlet provides simple CRUD operations against an eXtreme Scale grid.

This sample servlet can also be used for simple CRUD operations against an XC10 DataPower appliance Simple

Catalog Service Endpoints

Endpoints

Catalog Service Domain ID

Domain ID

Grid

Grid (For XC10 use Simple Grid Name)

Map

no connection

Figure 5-59 Connecting to CSD01

3. On a successful connection to catalog service, the message shown in Figure 5-60 is displayed in the web client.

connected to Grid on whost01.rtp.ibm.com:4811,whost02.rtp.ibm.com:4811,whost03.rtp.ibm.com:4

Figure 5-60 Client successfully connected to CSD01

4. Add an entry into the grid by providing a key and value pair and clicking **Insert**. The client application shows the outcome of the insert operation as shown in Figure 5-61.

CRUD Operation

Key

Value (Used for Insert and Update only)

SUCCESS: Inserted value1 with key key1

Figure 5-61 Inserting a new entry into the grid

- Retrieve an entry from the grid by providing a key and clicking **Get**. The client application shows the outcome of the retrieve operation as shown in Figure 5-62.

The screenshot shows a web form titled "CRUD Operation". It contains two input fields: "Key" with the value "key1" and "Value" which is empty. A note next to the Value field says "(Used for Insert and Update only)". Below the fields are four buttons: "Insert", "Update", "Delete", and "Get".

Figure 5-62 Retrieving an entry from the grid

- Update an existing entry in the grid by providing a key and value pair and clicking **Update**. The client application shows the outcome of the update operation as shown in Figure 5-63.

The screenshot shows the same "CRUD Operation" form. The "Key" field contains "key1" and the "Value" field contains "value11". The note "(Used for Insert and Update only)" is present. The "Update" button is highlighted, indicating it is the selected operation.

Figure 5-63 Updating an existing entry in the grid

- Delete an entry from the grid by providing a key and clicking **Delete**. The client application shows the outcome of the delete operation as shown in Figure 5-64.

The screenshot shows the "CRUD Operation" form. The "Key" field contains "key1" and the "Value" field is empty. The note "(Used for Insert and Update only)" is present. The "Delete" button is highlighted, indicating it is the selected operation.

Figure 5-64 Deleting an entry from the grid

5.3 Integrating into WebSphere Application Server Liberty Profile

This section describes the benefits, limitations, and details for using the Liberty Profile as a WebSphere eXtreme Scale container or catalog server. The following topics are covered in this section:

- ▶ 5.3.1, "Benefits" on page 231
- ▶ 5.3.2, "Limitations" on page 231
- ▶ 5.3.3, "Considerations" on page 231

- ▶ 5.3.4, “Introducing the sample application” on page 232
- ▶ 5.3.5, “Introducing the sample topology” on page 232
- ▶ 5.3.6, “Creating the sample topology” on page 233
- ▶ 5.3.7, “Testing the sample topology and application” on page 239

5.3.1 Benefits

The WebSphere Application Server Liberty Profile, introduced with WebSphere Application Server version 8.5, provides a highly composable, fast-starting, dynamic application server environment.

For more information about the Liberty Profile, see the *Liberty profile* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fcxsliberty.html>

Using WebSphere eXtreme Scale with the Liberty Profile allows you to take advantage of the simplicity that the Liberty Profile provides and provide top-tier enterprise data grid functions to clients. For example, to deploy a grid container in the form of a Liberty Profile server, you only need to drop a well-formed OSGi bundle into a directory and start the server. Even easier, at the minimum, you can simply copy your ObjectGrid configuration files into the appropriate directory and start the server.

5.3.2 Limitations

When you use the Liberty Profile as a WebSphere eXtreme Scale container or catalog server, you have full access to all of the features of WebSphere eXtreme Scale.

5.3.3 Considerations

If you decide to run the Liberty Profile with the JVM that Oracle provides, special considerations must be taken:

- ▶ Classloader deadlock

When you use the Oracle JVM, cases of deadlocks in the BundleLoader bundle have been reported. To mitigate this, see the *Classloader deadlock* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fcxsliberty.html>

- ▶ IBM ORB

WebSphere eXtreme Scale requires the use of the IBM ORB implementation unless you choose to run your grid environment using XIO only. The Liberty Profile does not provide the IBM ORB like a full WebSphere Application Server installation does.

To use the IBM ORB implementation, you must reference the IBM ORB JARs by adding the `java.endorsed.dirs` Java custom property to your JVM startup arguments. For more information about configuring a custom ORB implementation with WebSphere eXtreme Scale, see the *Configuring a custom Object Request Broker* topic in the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Ftxscfgorb.html>

5.3.4 Introducing the sample application

The sample application that is used in the Liberty Profile hosted sample topology, `GettingStartedWebClient.ear`, is a simple Java Platform, Enterprise Edition web application. The application uses a servlet to connect to a grid and run basic operations like `get`, `update`, `insert`, and `delete`. Figure 5-65 provides an overview of the basic application topology.

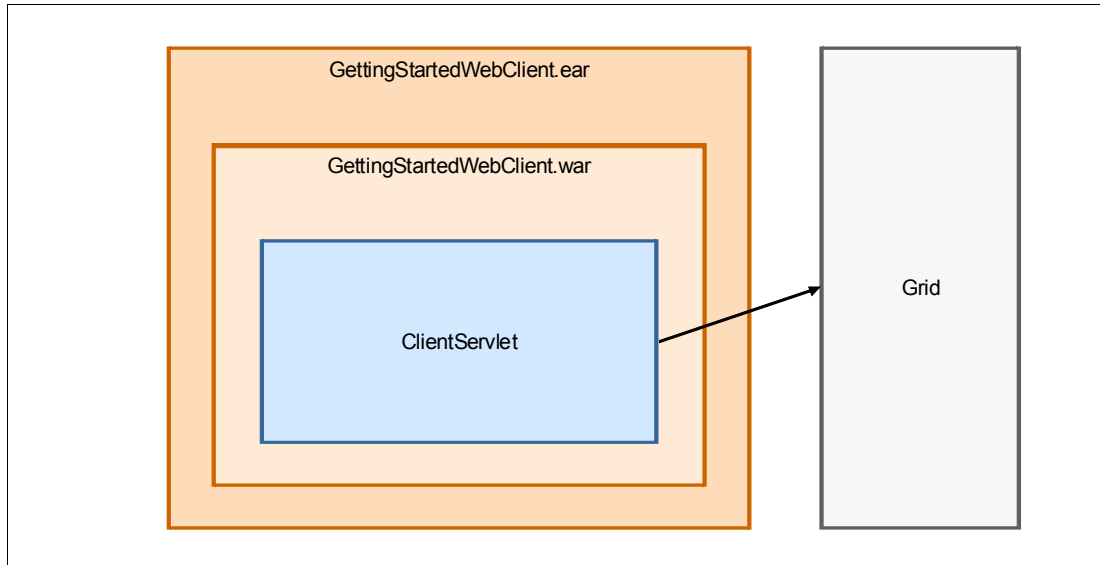


Figure 5-65 Overview of the sample application for integrating with a Liberty Profile

The sample application, `GettingStartedWebClient.ear` can be downloaded from the following website:

https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/websphere_extreme_scale_getting_started_with_websphere_application_server_sample5?lang=en

5.3.5 Introducing the sample topology

A sample topology must be built to accommodate the Getting Started application. This section describes setting up a sample topology based on using WebSphere eXtreme Scale with the Liberty Profile.

For this example, a sample topology of three Liberty Profile based catalog server instances is used. Four Liberty Profile based container servers are used to hold data. Finally, a single Liberty Profile server instance acts as a client to the grid. The server instances are spread evenly across four hosts as depicted in Figure 5-66.

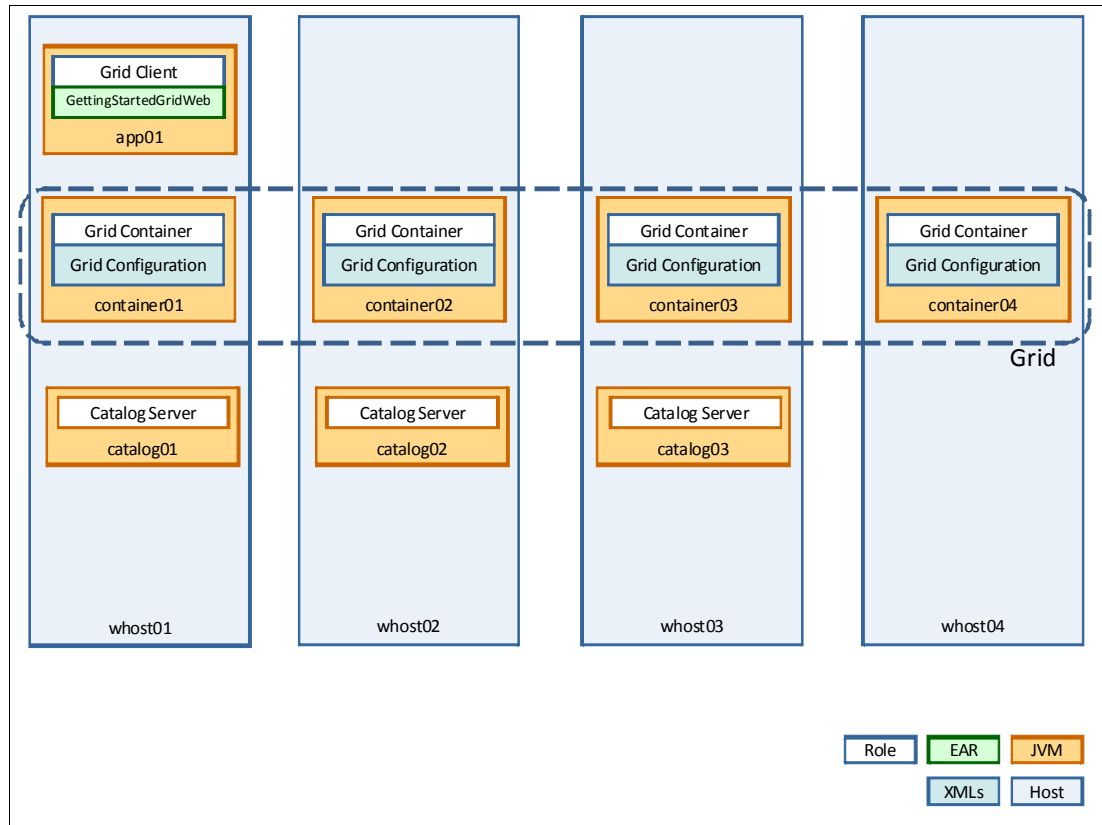


Figure 5-66 Topology for Liberty sample deployment

5.3.6 Creating the sample topology

This section describes how to install and configure the sample topology using the Liberty Profile and WebSphere eXtreme Scale.

Installing the WebSphere Liberty Profile for WebSphere eXtreme Scale


The Liberty Profile is typically installed as part of a WebSphere Application Server version 8.5 product installation or by using a direct download from IBM. The fastest way to get a WebSphere eXtreme Scale enhanced WebSphere Application Server Liberty Profile installation is to download the archives available on the WASDev Community website:


<https://www.ibm.com/developerworks/community/blogs/wasdev/entry/download>


The Downloads page is shown in Figure 5-67.

Downloads

Latest Release

 **WebSphere Application Server V8.5 Liberty Profile**
[Download Release](#)
[Learn about the product Information Center](#)

 **WebSphere Application Server Developer Tools for Eclipse V8.5.1**
[Download Release](#)
[Learn about the product](#)

 **WebSphere Application Server for Developers V8.5**
[Download Release](#)

Rational Application Developer for WebSphere Software V8.5
[Download Release](#)
[Information Center](#)


 **WebSphere eXtreme Scale V8.6 for Developers - Liberty Profile**
[Download Release](#)
[Information Center](#)

Figure 5-67 Downloading the Liberty Profile and the WebSphere eXtreme Scale enhancements

After you download the archives, complete the following steps to install the software:

1. Install the WebSphere Application Server Liberty Profile by locating the downloaded archive `wlp-developers-8.5.0.2.jar` and run the following command:

```
java -jar wlp-developers-8.5.0.2.jar
```
2. Provided you have Java included in your path, the command issued in step 1 presents an option to review the license agreement (Example 5-49). Press Enter to continue.

Example 5-49 License acceptance for the Liberty Profile

Before you can use, extract, or install IBM WebSphere Application Server 8.5 - Liberty Profile, you must accept the terms of International License Agreement for Non-Warranted Programs and additional license information. Please read the following license agreements carefully.

The license agreement is separately viewable using the `--viewLicenseAgreement` option.

Press Enter to display the license terms now, or 'x' to skip. x
Additional license information is separately viewable using the `--viewLicenseInfo` option.

Press Enter to display additional license information now, or 'x' to skip.

3. Accept the license agreement by pressing 1 as shown in Example 5-50.

Example 5-50 Accepting the product licenses

By choosing the "I Agree" option below, you agree to the terms of the license agreement and non-IBM terms, if applicable. If you do not agree, select "I do not Agree".

Select [1] I Agree, or [2] I do not Agree:

4. Select a directory to install the Liberty Profile into as shown in Example 5-51.

Example 5-51 Selecting an installation path for the Liberty Profile

Enter directory for product files or leave blank to accept the default value.
The default target directory is /home/root/Downloads
Target directory for product files?

Note: Depending on your chosen installation directory, you might need to run the Java command in Step 1 on page 234 by prefacing the command with `sudo`.

5. When the installation completes, you see the message shown in Example 5-52.

Example 5-52 Successful installation of the Liberty Profile

Successfully extracted all product files.

6. After you install the Liberty Profile, install the WebSphere eXtreme Scale enhancements. Locate the WebSphere eXtreme Scale version 8.6 Liberty Profile archive that you downloaded (Figure 5-67 on page 234) and run the following command:

```
java -jar wxs-wlp_8.6.0.1.jar
```

As with the Liberty Profile install, you are presented with the option to review the various licenses that must be accepted before installing (Example 5-53). Press Enter to continue.

Example 5-53 WebSphere eXtreme Scale for WebSphere Liberty Profile license review

Before you can use, extract, or install IBM International License Agreement for Non-Warranted Programs (IBM form number Z125-5589-05)., you must accept the terms of International License Agreement for Non-Warranted Programs and additional license information. Please read the following license agreements carefully.

The license agreement is separately viewable using the `--viewLicenseAgreement` option.

Press Enter to display the license terms now, or 'x' to skip.

7. Accept the license agreement by pressing 1.
8. Enter the same product installation directory that you specified in step 4 on page 235.

Note: Depending on your chosen installation directory, you might need to run the Java command in step 6 on page 235 by prefacing the command with **sudo**.

9. When the installation completes, you see the message shown in Example 5-54.

Example 5-54 Successful installation

```
Successfully extracted all product files.  
Created the jvm.options file to include eXtreme Scale binaries on the Java  
endorsed directories path at /opt/ibm/wlp/etc/jvm.options.
```

You now have a working WebSphere eXtreme Scale enhanced Liberty Profile deployment.

Configuring the runtime environment

When the installation completes, the grid topology can be constructed. The first step is to create the three catalog servers arranged on three different hosts as depicted in Figure 5-66 on page 233.

1. Create three liberty servers (catalog01, catalog02, and catalog03) by running the following command on each of the three catalog server hosts from the Liberty Profile installation directory:

```
<wlp_install_dir>/bin/server create <server_name>
```

2. To verify that the servers were created successfully, browse to the `<wlp_install_dir>/usr/servers` directory and confirm that a new directory named after the server name provided was created. See Example 5-55.

Example 5-55 Verifying new server creation for catalog01

```
[root@whost01 servers]$ pwd  
/opt/ibm/wlp/usr/servers  
[root@whost01 servers]$ ls -ltra  
total 16  
drwxr-xr-x. 4 root root 4096 May  3 16:47 ..  
drwxr-xr-x. 4 root root 4096 May  3 17:39 .  
drwxr-xr-x. 5 root root 4096 May  3 17:39 catalog01  
drwxr-xr-x. 2 root root 4096 May  3 17:39 .logs  
[root@whost01 servers]$
```

3. Create four container servers using the same command structure as used in step 1. The container servers that are used in this sample topology are called container01, container02, container03, and container04.
4. Create a single server instance to act as the client for the sample grid topology. The application server name that is used in this sample topology is app01.

In total, you create the following eight new Liberty server instances:

- ▶ catalog01
- ▶ catalog02
- ▶ catalog03
- ▶ container01
- ▶ container02
- ▶ container03
- ▶ container04
- ▶ app01

Thus far you have only created server definitions that are used to host WebSphere eXtreme Scale catalog and container server logic. However, certain configurations must be completed to designate the appropriate Liberty Profile server instances as catalog servers. This can be accomplished by editing the `server.xml` file found in the `<wlp_install_dir>/usr/servers/<server_name>` directory.

Example 5-56 demonstrates how to configure the Liberty Profile instance as a catalog server.

Example 5-56 Configuring a Liberty Profile server to be a catalog server

```
<server description="new server">
<!-- Enable features -->
  <featureManager>
    <feature>eXtremeScale.server-1.1</feature>
  </featureManager>
  <com.ibm.ws.xs.server.config catalogServer="true" listenerPort="2809" transport="XIO"/>
</server>
```

The default new server configuration includes the basic features for serving JSP pages and opening an HTTP endpoint port. Catalog servers in this sample topology do not serve JSPs. Therefore, that configuration can be eliminated from the `server.xml` file. This leaves only the configuration that is shown in Example 5-56. Because the sample topology runs only one catalog server per host, the value for `listenerPort` parameter can be the same across all three server instances.

Starting the catalog servers

Catalog servers must always be started before containers and the catalog servers themselves must be started within a relatively short amount of time of each other. To start the catalog server instances, run the following command:

```
<wlp_install_dir>/bin/server start <server_name>
```

The Liberty Profile server instance starts. Verify that the server is running by reviewing the message.log file that is in the following directory:

```
<wlp_install_dir>/usr/servers/<server_name>/logs
```

Generally, if you find a message like the one shown in Example 5-57, the server started successfully.

Example 5-57 Successful server start message

```
[5/3/13 18:37:15:068 EDT] 00000017 com.ibm.ws.kernel.feature.internal.FeatureManager
A CWWKF0011I: The server catalog01 is ready to run a smarter planet.
```

Starting the container servers

Container servers are configured much the same way as the catalog servers. In fact, the same `server.xml` configuration file can be used without the parameter that activates the catalog server logic. See Example 5-58.

Example 5-58 Configuring a Liberty Profile server to function as a container server

```
<server description="new server">
<!-- Enable features -->
  <featureManager>
    <feature>eXtremeScale.server-1.1</feature>
  </featureManager>
</server>
```

Altering the Liberty Profile server configuration is not enough to create a working grid. The Liberty Profile container server requires `objectgrid.xml` and `objectGridDeployment.xml` files to successfully host a grid instance.

To accomplish this, create a directory called `grids` in the `<wlp_install_dir>/usr/servers/<server_name>` directory. Within this new directory, create the `objectgrid.xml` file shown in Example 5-5 on page 156 and the `objectGridDeployment.xml` file shown in Example 5-6 on page 158.

Tip: In most cases, the capitalization of the `objectgrid.xml` and `objectGridDeployment.xml` file names is done purposefully. For the Liberty Profile, the `objectgrid.xml` file name is not `objectGrid.xml` as is typically seen in other WebSphere eXtreme Scale deployment scenarios.

Start the container servers with the following command:

```
<wlp_install_dir>/bin/server start <server_name>
```

You can validate that the server started correctly and that WebSphere eXtreme Scale has detected the grid configuration files by reviewing the logs in the following directory:

```
<wlp_install_dir>/usr/servers/<server_name>/logs
```

The sample topology grid infrastructure has now been set up and is running. The last operation that is required to test the sample application is to start the Liberty Profile server instance that functions as the grid client and hosts the Getting Started application.

Starting the client application server

To set up the client side of the sample topology, the Liberty Profile server, `app01`, must be started and the sample application deployed by dropping the application archives into the `dropins` folder.

Complete the following steps to start the Liberty Profile client server and deploy the sample application:

1. Start the client server with the following command:

```
<wlp_install_dir>/bin/server start <server_name>
```

In this sample topology, `app01` is used for the `<server_name>` parameter.

2. After the server starts, copy the sample application, `GettingStartedWebClient.ear`, into the `dropins` folder that is in the following directory:

```
<wlp_install_dir>/usr/servers/<server_name>/dropins
```

The server run time detects the new archive and installs the application automatically. You can review the server log as shown in Example 5-59 to confirm the installation.

Example 5-59 Installing the GettingStartedWebClient.ear sample application

```
[5/3/13 22:46:35:777 EDT] 00000012 com.ibm.ws.app.manager.internal.statemachine.StartAction
I CWWKZ0018I: Starting application GettingStartedWebClient.
[5/3/13 22:46:36:161 EDT] 00000012 com.ibm.ws.webcontainer.osgi.webapp.WebGroup
I SRVE0169I: Loading Web Module: GettingStartedWebClient.
[5/3/13 22:46:36:162 EDT] 00000012 com.ibm.ws.webcontainer
I SRVE0250I: Web Module GettingStartedWebClient has been bound to default_host.
[5/3/13 22:46:36:164 EDT] 00000012 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host):
http://host1:9080/GettingStartedWebClient/*
[5/3/13 22:46:36:165 EDT] 00000012 com.ibm.ws.webcontainer.osgi.WebContainer
I SRVE9998I: Application GettingStartedWebClient added to web container.
[5/3/13 22:46:36:166 EDT] 00000012 com.ibm.ws.app.manager.internal.statemachine.StartAction
A CWWKZ0001I: Application GettingStartedWebClient started in 0.89 seconds.
```

The environment is now ready to be tested.

5.3.7 Testing the sample topology and application

To test the sample topology, complete the following steps:

1. From a web client, navigate to the following URL:

```
http://hostname:port/GettingStartedWebClient/ClientServlet
```

In this URL, *hostname* and *port* are the host name and HTTP port for your Liberty Profile server. The default HTTP endpoint point for the Liberty Profile run time is 9080.

If you connect to the correct host and port combination, you see a web page like the one shown in Figure 5-68.

IBM WebSphere eXtreme Scale Samples

WebSphere eXtreme Scale Getting Started Sample

WebSphere eXtreme Scale Getting Started Client

This sample servlet provides simple CRUD operations against an eXtreme Scale grid.

This sample servlet can also be used for simple CRUD operations against an XC10 DataPower appliance Simple Grid.

Catalog Service Endpoints

Endpoints

Catalog Service Domain ID

Endpoints

Remote: <host>:<port>
Embedded: <blank>

Grid

Grid (For XC10 use Simple Grid Name)

Map

no connection

CRUD Operation

Key

Value (Used for Insert and Update only)

Figure 5-68 Home page for the GettingStartedWebClient sample application

2. Connect the web client to the grid by specifying the catalog service endpoint that is used in the grid. There are three possible catalog servers to connect to in this sample topology:
 - whost01:2809
 - whost02:2809
 - whost03:2809

After you specify the catalog server endpoint to connect to, click **Connect** as shown in Figure 5-69.

Catalog Service Endpoints

Endpoints

Catalog Service Domain ID

Endpoints

Remote: <host>:<port>

Embedded: <blank>

Grid

Grid (For XC10 use Simple Grid Name)

Map

no connection

Figure 5-69 Connecting the web client to the grid

The client application is connected to the grid as shown in Figure 5-70.

Catalog Service Endpoints

Endpoints

Catalog Service Domain ID

Endpoints

Remote: <host>:<port>

Embedded: <blank>

Grid

Grid (For XC10 use Simple Grid Name)

Map

connected to Grid on whost01:2809

Figure 5-70 Successful grid connection

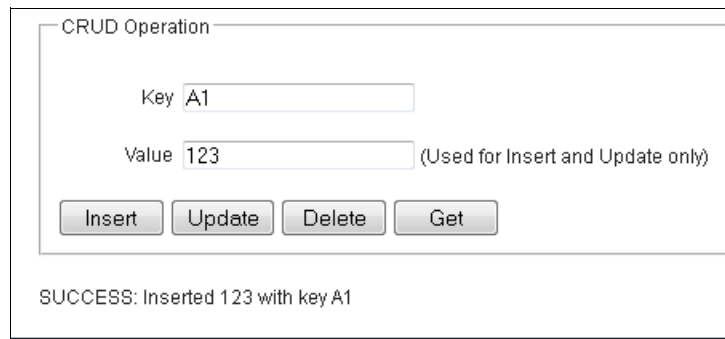
3. With a successful connection established, the web client can now issue operations on the grid itself. Click **Insert** as shown in Figure 5-71.



The screenshot shows a web form titled "CRUD Operation". It contains two input fields: "Key" with the value "A1" and "Value" with the value "123". The "Value" field has a note "(Used for Insert and Update only)" to its right. Below the fields are four buttons: "Insert", "Update", "Delete", and "Get". The "Insert" button is highlighted with a darker background, indicating it is the active operation.

Figure 5-71 Inserting data into the sample grid

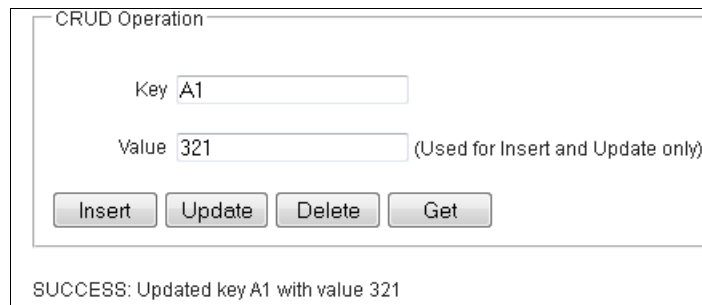
4. To complete the operation, click **Insert**. The client application shows the outcome of the operation (Figure 5-72).



The screenshot shows the same "CRUD Operation" form as in Figure 5-71. The "Key" field contains "A1" and the "Value" field contains "123". The "Insert" button is still highlighted. Below the form, a message box displays "SUCCESS: Inserted 123 with key A1".

Figure 5-72 Successful insert operation


5. Change the text in the Value field from 123 to 321 and click **Update** (Figure 5-73).



The screenshot shows the "CRUD Operation" form with the "Key" field containing "A1" and the "Value" field containing "321". The "Value" field has the note "(Used for Insert and Update only)". The "Update" button is highlighted with a darker background. Below the form, a message box displays "SUCCESS: Updated key A1 with value 321".

Figure 5-73 Successful update to grid

6. To remove the key, make sure that A1 is still in the Key field, and click **Delete**.



Extended HTTP session management with WebSphere eXtreme Scale

This chapter describes how WebSphere eXtreme Scale can be used to provide HTTP session management in both a WebSphere and non-WebSphere Application Server environment.

It introduces the main concepts behind implementing WebSphere eXtreme Scale HTTP session management. It begins with a general background of HTTP session management and introduces a sample application to demonstrate these concepts. The chapter continues by detailing how you can configure WebSphere eXtreme Scale session management support for the sample application in three typical environments.

This chapter includes the following sections:

- ▶ 6.1, “HTTP session management overview” on page 244
- ▶ 6.2, “Using WebSphere eXtreme Scale for HTTP session management” on page 250
- ▶ 6.3, “Introducing the SessionTest sample application” on page 257
- ▶ 6.4, “Scenario 1: Implementing with WebSphere Application Server Network Deployment” on page 260
- ▶ 6.5, “Scenario 2: Implementing with WebSphere Liberty Profile” on page 281
- ▶ 6.6, “Scenario 3: Implementing with Apache Tomcat” on page 288

6.1 HTTP session management overview

HTTP session management with WebSphere eXtreme Scale is just one of several deployment scenarios outlined in Chapter 3, “Application scenarios” on page 75. Although this scenario is rarely the primary reason for purchasing WebSphere eXtreme Scale, it often finds use after WebSphere eXtreme Scale is installed for other reasons. It can be used, for example, to offload dynamic cache or as an application side cache. This chapter describes HTTP session management in detail because it illustrates a number of key points when you use WebSphere eXtreme Scale:

- ▶ It applies equally whether you are holding your grid within a WebSphere eXtreme Scale software installation or on an IBM WebSphere DataPower XC10 Appliance.
- ▶ It can be used across all Java Platform, Enterprise Edition web containers. This scenario provides a convenient means of introducing how WebSphere eXtreme Scale integrates with various architectures.
- ▶ It is one of two “plug in” scenarios for WebSphere eXtreme Scale, meaning that your application logic does not change. Rather, only the configuration changes.
- ▶ The more popular “plug in” scenario, the WebSphere eXtreme Scale dynamic cache plug-in, is already covered in many other articles available on IBM developerWorks, including the following article. The Resources section at the bottom of this article lists dozens of other links that are related to dynamic caching.

<http://www.ibm.com/developerworks/library/co-wxs-dynamic-cache>

HTTP session management is a basic requirement of the Java Platform, Enterprise Edition specification for application servers. It allows Java Platform, Enterprise Edition applications to store *state* for a user across a series of HTTP requests. The specification provides an API to store and retrieve information specific to that user. The classic example of this is a website shopping cart, where transient data or state about intended purchases is saved across several browser pages until final checkout.

All application servers provide this basic function, but the qualities of service that underpin this can vary widely.

This section provides a brief overview of several session manager concepts:

- ▶ How an application server typically maps an incoming user (or “client”) request to the HTTP session associated with that client.
- ▶ How session replication is used to provide high availability in case of an application server failure.
- ▶ How session affinity can be used to aid performance.

Note: Some information about each of these topics is intentionally omitted to keep this discussion at a conceptual level. There are a great many web resources available to gain a deeper understanding of these topics.

6.1.1 Using HTTP session to hold state

The Hypertext Transfer Protocol (HTTP) is *stateless* by design. That means that every incoming request from a web browser has no association with any previous requests from the same or any other user. Its GET method (and other methods) is *idempotent*, meaning that multiple identical requests produce the same effect as a single request. Using a stateless protocol has many advantages for server design. However, it requires some consideration

when an application must maintain some knowledge about a user (*stateful*) as the user navigates through pages of the website.

When a servlet is started within an application server as a result of a client request, the servlet can call a method on the incoming servlet request object (`request.getSession`) to retrieve the HTTP session that is associated with that user. On the first request to the application by this user, this API call fails to find the session. It then by default implicitly creates an empty `HttpSession` object, returning it to the servlet's `getSession` call.

The servlet then uses the `HttpSession` to store (`session.putAttribute`) various objects that are related to this user's interaction with the application. For example, it can store a "breadcrumb trail" to track the user's flow through the application.

On subsequent requests from the same user, the servlet's `request.getSession` call to `request` the associated session finds the `HttpSession` object and returns it for use. The servlet can retrieve (`session.getAttribute`) values that were previously stored into the session. It can also modify these values, delete them, or add new ones to the session.

Note that JavaServer Pages (JSP) are internally compiled into a servlet for execution on the application server, so although this discussion talks about servlets, it applies equally to JSPs. One interesting yet commonly unknown aspect of a JSP is that it automatically issues a `request.getSession(true)` at its entry. The `true` parameter, which is the default, says that if the `HttpSession` is not found (a first-time call from this client), a new `HttpSession` is created. If for whatever reason you do *not* want an `HttpSession` to be automatically retrieved or implicitly created, add the following to the top of the JSP to override the default behavior:

```
<%@ page session="false" %>
```

Doing so can slightly improve performance in JSPs that have no need for `HttpSession`.

6.1.2 Tracking who what owns each HTTP session

But how does the `request.getSession` logic know how to find the requesting client's `HttpSession`? There are two techniques that are commonly used, each with their advantages and disadvantages:

- ▶ Cookie-based session tracking
- ▶ URL rewriting

Cookie-based session tracking

Using *cookies* is the simplest and most common way to track HTTP sessions. When session management is enabled and a client makes a request, the `HttpSession` object is created and a unique session ID is generated for the client. This is sent to the client browser as a specially named cookie as part of the HTTP response. On subsequent client requests to the same host domain name, the browser sends all cookies that are stored for that domain as part of the client request. The application server's session manager logic can then find that cookie in the request, extract its content (the session ID), and use that unique value to locate the associated HTTP session for this client.

The advantages of using cookies are that it is straightforward and offers relatively compact communication with the client. A disadvantage is that it is possible for a user to configure their browser to reject cookies. That user finds that they have a hard time accessing most websites, as cookie-based tracking is by far the most prevalent technique for HTTP session tracking.

The cookie name is unimportant if the same name is used consistently through all tiers of the website flow. A common cookie name for session tracking, and the default name that is used by the WebSphere Application Server, is JSESSIONID. There is rarely any need to change this name.

URL rewriting

Where the application wants to support users of browsers with their cookies disabled, an alternative technique called *URL rewriting* can be used. In this technique, every link that is constructed in a web page gets a parameter appended to it. The appended parameter contains the session ID.

For example, the following link in a web page:

```
<a href="/store/catalog">
```

might instead be extended as:

```
<a href="store/catalog;jsessionid=DA32242SSGE2">
```

When the user clicks the link in their browser, that extended form of the URL is sent to the application server as part of the request. The session manager within the application server can find and recognize the `jsessionid=DA32242SSGE2` as the session ID. The manager then uses it (as with the cookie-based approach) to locate the associated `HttpSession` object for this client.

The advantage of URL rewriting is that this technique always works because it does not depend on cookie support.

However, there are several disadvantages:

- ▶ Larger HTML pages for transmission in both directions (and consider that pages often have dozens or even hundreds of links to be rewritten).
- ▶ The user can directly see these URLs (with the `jsessionid` parameters) in the browser address bar, and inadvertently (or deviously) alter the session ID. It is also possible to change a cookie's content, but not inadvertently. Even so, there is virtually no chance of an altered session ID being successful at "hijacking" another user's session. Actual session IDs are 20 or more alphanumeric characters long, and generated with a highly random and unpredictable algorithm.
- ▶ If the user bookmarks the page, the bookmarked URL includes the `jsessionid` value for the current `HttpSession`. This session will not be found when the bookmark is used in a subsequent browser session, possibly causing unpredictable behavior.
- ▶ The URL extension is only possible for pages that are dynamically constructed by a servlet or JSP. Static HTML pages cannot be rewritten and therefore cannot be used. All pages must be dynamically created.

Other techniques for HTTP session tracking

There are a couple of other possible techniques that can be used for HTTP session tracking, but they are rarely used:

- ▶ Hidden form fields

This can be thought of as a variant on URL rewriting. Rather than rewriting page links, an HTML form on every page is used with an HTTP POST operation. That form contains a hidden form field to provide the session ID. This technique is rarely used because it has most of the same disadvantages of URL rewriting and is significantly more cumbersome to use.

- ▶ SSL ID session tracking

This technique can be used when a user is required to be encrypted (HTTPS). As a side-effect of the “SSL handshake” used in this protocol switch, an SSL session ID is maintained between the browser and the application. This can be thought of as similar to cookie session tracking, but the SSL ID cannot be disabled by the browser user. It is therefore possible to uniquely map the SSL ID to an HTTP session. However, there are several complications and caveats in doing so, given a typical multitier environment through which this SSL ID must flow. In practical terms, this technique is not often used.

6.1.3 Session replication and session persistence

You can enable session persistence in your application server tier to cause HTTP session content to be written to a database, or in a WebSphere Application Server Network Deployment environment, replicated to another application server JVM in the cluster. Both of these techniques keep a recent copy of the session data that can be retrieved by this or any other application server in your cluster. This configuration provides two important benefits:

- ▶ High availability

If the original application server fails and the user’s next request is routed to another application server in the cluster, it can retrieve the replicated or persisted session data and carry on the user’s session without impact.

If you do not have support, the new application server fails to find the user’s session in its memory, and so considers this a new user of the website. The user must start their interactions over again from the beginning, for example with a fresh login.

- ▶ Offloading of sessions to reduce JVM heap contention

An application server keeps `HttpSession` data within its JVM heap for quick access. As the number and size of sessions grows, this heap usage can add to garbage collection times, causing performance slow downs. In an extreme case, it can even lead to a JVM out-of-memory exception. By setting up session replication or persistence, the application server can be configured to keep only the most recently used 1000 (for example) sessions in memory. Earlier sessions will have already been persisted to the database or replicated to another application server in the cluster, so they can be reinstated in JVM heap when required.

6.1.4 Session affinity

For best performance, modern application servers such as WebSphere Application Server, keep the most recently used `HttpSession` objects for user sessions within their JVM heap memory. The default configuration for WebSphere Application Server is to keep the most recent 1000 sessions in memory, but this can be easily changed.

In a typical highly available application server topology, there are two or more (sometimes many more, even hundreds) application server “clones”, each with the same applications installed. This is called a *cluster* of application servers. A web server tier in front of the application server tier provides for routing of inbound requests to an available application server to service the request.

To provide the highest level of performance when HTTP sessions are used, the web servers are typically configured to preserve *session affinity* to a particular application server. This is sometimes also called *session stickiness*.

If a specific user “Bob” was originally routed by the web server tier to application server “App04” and established an HTTP session on that application server, then it is best if “Bob” is

routed to the same App04 application server later. This process allows App04 to directly retrieve Bob's session from its JVM heap, rather than having to fetch Bob's session content from the database or from another JVM. If session persistence or replication are not enabled, session affinity becomes critical for maintaining a session across user requests.

There are several means by which a web server tier can provide session affinity. The most commonly used is session cookie-based affinity, which is the only method that is described in this section. You can search the web for "http session affinity" to learn more about other possible approaches such as IP-based affinity or SSL-based affinity. You will find that these approaches have several disadvantages, which is why session cookie-based affinity is often preferred.

Session cookie-based affinity

As described in "Cookie-based session tracking" on page 245, the session manager uses a cookie, usually JSESSIONID, to track the user session. In a WebSphere Application Server Network Deployment environment, the session manager also appends an indicator of the current application server to the value, using the following conceptual format:

```
JSESSIONID=sessionid123:appid04
```

Given this format, the `appid04` portion provides a one-to-one reference to a specific application server, in a form that is understood by the web server routing logic implemented by the WebSphere web server plug-in. The plug-in can inspect inbound HTTP requests for their cookies, and upon finding a JSESSIONID cookie, further inspect it for its `appid04` portion. It can then route this request to that same application server, App04. What this example shows as `appid04` is known as the application server's *clone ID*. It is a cryptic string of letters and numbers that uniquely identifies an application server in the topology.

After the request arrives on the application server, its session manager can then inspect the inbound JSESSIONID cookie content, find its `sessionid123` value, and use that to locate the appropriate `HttpSession` object for this user.

On the first request from a new browser user to an application, there is no JSESSIONID accompanying the inbound request. In this case, the web server plug-in, failing to find the cookie, resorts to its configured scheme for routing of a non-affinity request. In a WebSphere Application Server plug-in environment, this is specified in the plug-in configuration file as either round-robin (each application server in order) or random (pick one randomly). After the chosen application server services the request, a new JSESSIONID cookie is returned along with the response. The cookie is then returned by the browser on subsequent requests to the same domain name.

Application server fail over

What happens if an application server crashes, for example, due to an out-of-memory (OOM) situation? If the web server plug-in finds that the affinity application server is not available, the plug-in automatically *fails over* to another available application server in the cluster. That new application server uses the `sessionid123` to read the `HttpSession` from the persisted or replicated copy, then continues as normal. In the eventual response to the browser, the Session Manager appends its own `:appid02` (or whatever) to uniquely identify this new application server clone in the JSESSIONID content. So now the cookie might look like (conceptually):

```
JSESSIONID=sessionid123:appid04:appid02
```

After this fail over, the next inbound request through the web server plug-in uses the *last* `appid` from the cookie, `appid02`. It now maintains session affinity to the App02 application

server. The original `app04` is kept in the string to allow for optional *fail back* processing, if that behavior is wanted, when `App04` becomes available again.

6.1.5 Commonly implemented options for HTTP session management

Most application servers provide some sort of session persistence or replication function. This means that HTTP session objects are made available to other application servers in the environment. This is normally done either by writing (or persisting) the in-memory HTTP session state to a database, or by using memory-to-memory replication technology that targets another application server in the cluster to hold a copy.

During an application server failure or a request being routed to another application server in the cluster, the session state is copied to the appropriate application server and made available to the application. This process avoids requiring the user to log in again and repeat what they did up to that point.

However, there are compromises and limitations to the replication facilities provided with application servers. For example, to keep performance at a reasonable level, replication is typically done periodically, perhaps every 10 seconds. This gives a failure window that might be unacceptable to your application.

This delay is configurable within the WebSphere Application Server environment. It can be set to 0 to indicate to persist or replicate immediately at the end of the servlet execution. Doing so costs slightly in performance degradation when compared to the batching updates every 10 seconds. The chosen interval is a balance between how many seconds of session updates it is permissible to lose (worst case) in a fail over situation versus the possible performance effect of writing too frequently.

Another possible consideration is how many HTTP sessions can be held within the JVM heap at a time. Good HTTP session practice keeps session size small (to a few kilobytes), but there is no automatic enforcement of this recommendation. A single session can grow to several megabytes in size, sometimes without the application programmer even realizing that this has happened. Sometimes an object is written into `HttpSession` that inadvertently brings in another referenced object, which brings in another, and so on.

With traditional HTTP session management, all HTTP sessions for the application server are kept within its JVM heap. Under high periods of load, the application can have several hundred or even thousands of sessions in memory. If the accumulated size grows too large, this accumulation can potentially lead to an out-of-memory exception.

WebSphere Application Server provides a tunable option to control the number of in-memory sessions per application server JVM. Its default is 1000. When the number grows beyond this value, the least-recently-used sessions are offloaded to a database (if database persistence is configured), replicated to another JVM in cluster (if replication is configured), or discarded if neither. This prevents having to worry about session sizes and number of sessions, and how they might affect the application server's heap and health.

For more information about HTTP session replication options in WebSphere Application Server, see *WebSphere Application Server V8.5 Administration and Configuration Guide for the Full Profile*, SG24-8056.

6.2 Using WebSphere eXtreme Scale for HTTP session management

Now that you have an understanding of the key HTTP session management concepts, this section describes how WebSphere eXtreme Scale can be used to improve on session management. An overview of how eXtreme Scale session management is implemented is also provided.

6.2.1 Benefits of using WebSphere eXtreme Scale for HTTP session management

When using WebSphere eXtreme Scale for HTTP session management, you can benefit from the following advantages:

- ▶ High qualities of service

WebSphere eXtreme Scale provides faster and more robust replication than WebSphere Application Server. WebSphere eXtreme Scale provides an assured level of replication as it retries saving a session object if the initial attempt fails. Additionally, HTTP session data can be replicated asynchronously without the performance effect seen when you use the synchronous replication within WebSphere Application Server.

- ▶ No practical limit on HTTP session size or quantity

The WebSphere eXtreme Scale grid capacity can be configured to any reasonable size, and extended dynamically as required.

- ▶ Consistent support for any Java Platform, Enterprise Edition web container

Various vendors of web containers can implement HTTP session management in slightly different ways. For example, Apache Tomcat does not directly offer session replication. WebSphere eXtreme Scale's session management can integrate a number of disparate web container technologies into one common framework for session management.

- ▶ Highly scalable

The default WebSphere Application Server session manager tends to slow down as more sessions are created. WebSphere eXtreme Scale session manager has flat performance characteristics from 1 to 10000 or more active sessions.

- ▶ Session data is not restricted to a WebSphere Application Server cell boundary

In WebSphere Application Server, sessions are typically restricted to a cell boundary. Although it is technically possible to share sessions across a cell using a database, there are management drawbacks to this. However, it is straightforward to set up a remote WebSphere eXtreme Scale grid that many applications and cells can share.

- ▶ Cross application and platform access

You can configure WebSphere eXtreme Scale to act as a centralized HTTP session store for multiple applications that are running on different application servers. These servers include WebSphere Application Server, WebSphere Liberty, and Apache Tomcat.

- ▶ No need to implement `Serializable`

With standard application server support for replicating or persisting a session, each of the objects (or session attributes) stored within the `HttpSession` is required to implement `Serializable`. This is needed so that the objects can be serialized as they are marshalled across the *Object Request Broker* (ORB) communication protocol that is used for memory-to-memory replication or database offload persisting.

Tip: There is no warning if you place non-serializable data into your `HttpSession` because it is supported to do so, if you ensure that the session is kept only in memory. The serializable requirement arises when you want to persist or replicate the session. At the point of actually attempting the serialization, you receive a `NotSerializableException` if the serializable requirement is not met.

If instead you use WebSphere eXtreme Scale version 8.6 to replicate your session, you can use its XIO communication protocol and its XDF data format, which do not require serialization. Therefore, your application objects stored within `HttpSession` do not have to implement `Serializable`. Although this might seem like a small thing, the traditional need for implementing `Serializable` has prevented several clients from using session persistence or replication because of the development cost and risk in changing an existing application.

- ▶ No requirement for session affinity

With the traditional session manager, unless database persistence or replication is enabled, application server session affinity is required. Affinity is necessary so that the same application server is repeatedly used for the same user session so that the `HttpSession` can be located within that application server's JVM heap.

When using WebSphere eXtreme Scale for session management, all `HttpSession` data is always available to every application server in the cluster because it is stored in one common place on the shared grid. Session affinity is a feature of most web servers and other front-end appliances, but it is not universally available. WebSphere eXtreme Scale session management can lessen the impact in environments where affinity cannot be ensured.

6.2.2 Integration of WebSphere eXtreme Scale session management

WebSphere eXtreme Scale provides non-invasive integration with HTTP session management so that the application does not have to be changed. It does so by using an HTTP servlet filter that is a standard part of the Java Platform, Enterprise Edition servlet specification.

As shown in Figure 6-1 on page 252, the WebSphere eXtreme Scale HTTP servlet filter intercepts every request to the web application. Before passing control to the application servlet or JSP, it wraps the `HttpServletRequest` and `HttpServletResponse` objects that the servlet developer uses to access the request's session state. Therefore, when the servlet makes a `request.getSession()` call, it is processed by WebSphere eXtreme Scale logic rather than the default application server session manager.

The filter ensures that the session data is synchronized with the grid. On each request, if HTTP session attributes have changed, they are written back into the grid.

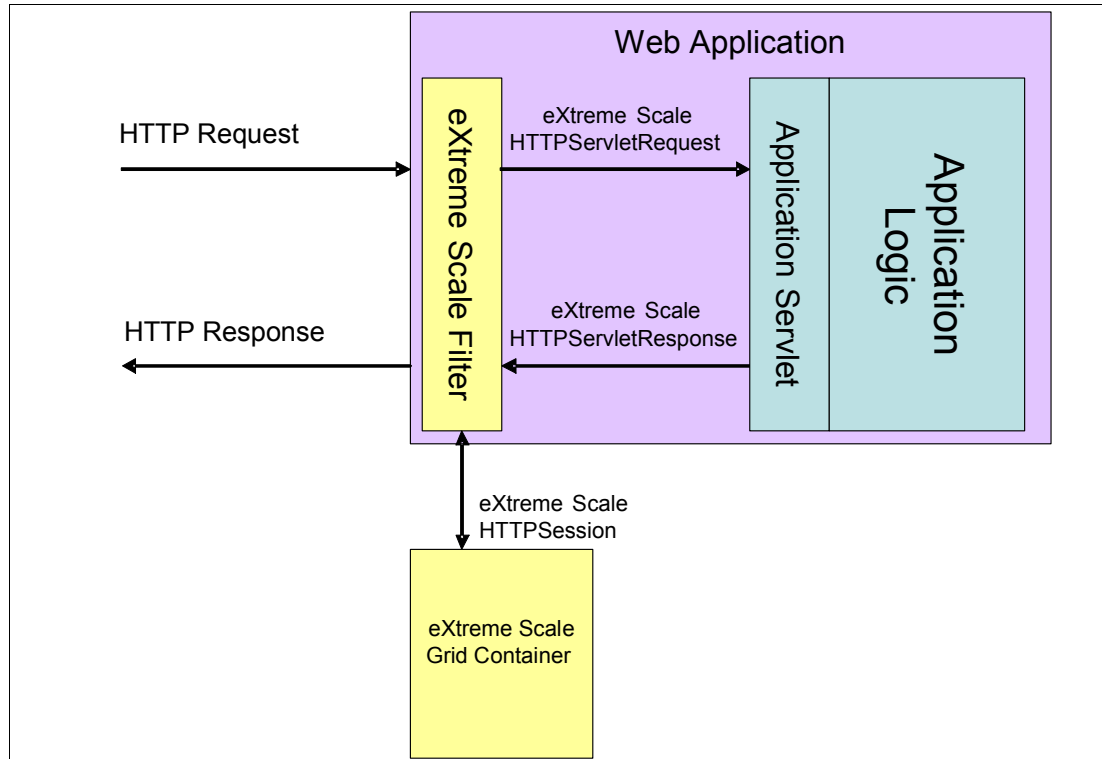


Figure 6-1 WebSphere eXtreme Scale servlet filter

6.2.3 Configuration for WebSphere eXtreme Scale session management

The servlet filter technique that is used by WebSphere eXtreme Scale integration for session management can be used by any Java Platform, Enterprise Edition-compliant web container. The web container must support the rudimentary session management APIs and support configurable servlet filters, which are both requirements of the Java Platform, Enterprise Edition specification.

Whichever Java Platform, Enterprise Edition web container is chosen, WebSphere eXtreme Scale is provided with the same basic configuration information. However, the method of providing the information varies somewhat for each scenario. The configuration that is required in all cases includes:

- ▶ Introduction of the WebSphere eXtreme Scale servlet filter and session listener into the execution path.
- ▶ Configuration of the WebSphere eXtreme Scale grid in which the session data is stored.
- ▶ Specific tuning for how the session manager within the web container behaves.

Three common scenarios for WebSphere eXtreme Scale used as the session manager are described in this chapter. Using the same sample application to demonstrate HTTP session usage, the following scenarios are described:

- ▶ “Scenario 1: Implementing with WebSphere Application Server Network Deployment” on page 260.
- ▶ “Scenario 2: Implementing with WebSphere Liberty Profile” on page 281
- ▶ “Scenario 3: Implementing with Apache Tomcat” on page 288.

The techniques described in this chapter generally apply equally to other Java Platform, Enterprise Edition compliant web containers and application servers.

6.2.4 Topologies for eXtreme Scale session management

As with any other implementations of a WebSphere eXtreme Scale data grid, you have two choices as to where to host your container servers:

- ▶ **Embedded:** The container runs in the same JVM as the web container client. Several JVMs can participate to form an embedded-partitioned grid, or you can use only one container in each JVM as an embedded-local grid topology.
- ▶ **Remote:** The containers run as JVMs separate from the web container client JVM.

Although an embedded topology is fully supported, the preferred topology for a production environment is remote. This topology provides for isolation of application client and container server functions, and helps prevent memory and other JVM resource contention. It allows for unique and appropriate JVM tuning for the application client and WebSphere eXtreme Scale container JVMs. You can decide how many JVMs are necessary for the grid, independent of the number of application servers you are running. There is little use of an embedded topology, except for possible use in your development and test environments.

Note that you can configure a remote topology by using an integrated or stand-alone installation. You can only create an embedded topology in an integrated installation.

6.2.5 Sample grid configuration files for an HTTP session

Regardless of the web container that you use, your HTTP session data ultimately is stored in a WebSphere eXtreme Scale grid. Therefore, you must configure the grid using the following configuration files:

- ▶ `objectGrid.xml`

This file contains the definition of the grid itself including its maps, locking behavior, plug-ins, and so on.

- ▶ `objectGridDeployment.xml`

This file contains a description of the grid's deployment that includes how many partitions, what replication strategy to use, and so on.

Although these configuration file names are common to any type of grid, there are some unique and specific requirements for a grid to support HTTP session data. To make this configuration easier, the WebSphere eXtreme Scale installation provides some pre-configured sample files that you can use with little or no modification.

In your `<wxs_install_root>/ObjectGrid/session/samples` directory, you will find the sample files as shown in Figure 6-2.

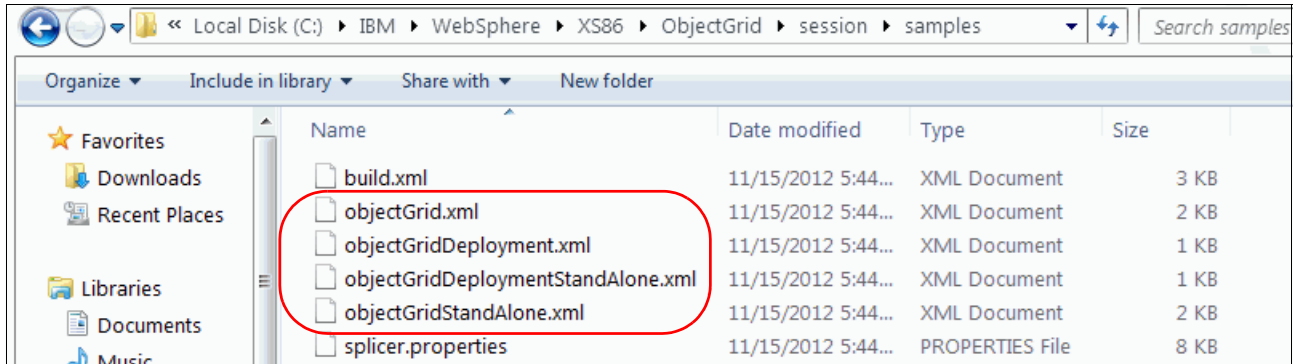


Figure 6-2 Sample configuration files for WebSphere eXtreme Scale session management

These sample files that are used for the configuration of the grid, and provide grid and cluster deployment definitions for WebSphere eXtreme Scale. The `build.xml` and the `splicer.properties` files are used for web application configuration, and are described in the scenarios described later in this chapter.

Two versions of the grid configuration files are provided as samples:

- ▶ The `objectGrid.xml` and `objectGridDeployment.xml` files are used with a WebSphere eXtreme Scale embedded topology.
- ▶ The `objectGridStandAlone.xml` and `objectGridDeploymentStandAlone.xml` files are used for a remote topology. If used, copy the remote topology files as simply `objectGrid.xml` and `objectGridDeployment.xml`.

Note: WebSphere eXtreme Scale remote topology was called a “stand-alone” topology in the past, which explains the “Stand-alone” term used in these sample file names.

For reference, the contents of these four sample files are shown in the following examples. Example 6-1 shows the `objectGrid.xml` file.

Example 6-1 objectGrid.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://ibm.com/ws/objectgrid/config"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd">
  <objectGrids>
    <objectGrid name="session" txTimeout="30">
      <bean id="ObjectGridEventListener"
className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
      <backingMap name="objectgridSessionMetadata"
pluginCollectionRef="objectgridSessionMetadata" readOnly="false" lockStrategy="PESSIMISTIC"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600" copyMode="NO_COPY"/>
      <backingMap name="objectgridSessionAttribute.*" template="true" readOnly="false"
lockStrategy="PESSIMISTIC" ttlEvictorType="NONE" copyMode="NO_COPY"/>
      <backingMap name="objectgridSessionTTL.*" template="true" readOnly="false"
lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600"
copyMode="NO_COPY"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

```

<backingMapPluginCollections>
  <backingMapPluginCollection id="objectgridSessionMetadata">
    <bean id="MapEventListener" className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Example 6-2 shows the objectGridDeployment.xml file.

Example 6-2 objectGridDeployment.xml file

```

<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd">
  <objectgridDeployment objectgridName="session">
    <mapSet name="sessionMapSet" numberOfPartitions="5" minSyncReplicas="0" maxSyncReplicas="0"
maxAsyncReplicas="1" developmentMode="false" placementStrategy="PER_CONTAINER">
      <map ref="objectgridSessionMetadata"/>
      <map ref="objectgridSessionAttribute.*"/>
      <map ref="objectgridSessionTTL.*"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>

```

Example 6-3 shows the objectGridStandalone.xml file.

Example 6-3 objectGridStandalone.xml file

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://ibm.com/ws/objectgrid/config"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd">
  <objectGrids>
    <objectGrid name="session" txTimeout="30">
      <bean id="ObjectGridEventListener"
className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
      <backingMap name="objectgridSessionMetadata"
pluginCollectionRef="objectgridSessionMetadata" readOnly="false" lockStrategy="PESSIMISTIC"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600" copyMode="COPY_TO_BYTES"/>
      <backingMap name="objectgridSessionAttribute.*" template="true" readOnly="false"
lockStrategy="PESSIMISTIC" ttlEvictorType="NONE" copyMode="COPY_TO_BYTES"/>
      <backingMap name="objectgridSessionTTL.*" template="true" readOnly="false"
lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600"
copyMode="COPY_TO_BYTES"/>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="objectgridSessionMetadata">
      <bean id="MapEventListener" className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>

```

Example 6-4 shows the `objectGridDeploymentStandalone.xml` file.

Example 6-4 objectGridDeploymentStandalone.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd">
  <objectgridDeployment objectgridName="session">
    <mapSet name="sessionMapSet" numberOfPartitions="47" minSyncReplicas="0" maxSyncReplicas="0"
maxAsyncReplicas="1" developmentMode="false" placementStrategy="FIXED_PARTITIONS">
      <map ref="objectgridSessionMetadata"/>
      <map ref="objectgridSessionAttribute.*"/>
      <map ref="objectgridSessionTTL.*"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

The content shown in these examples has been reformatted slightly (using the XMLlint tool) for consistent indenting, which makes direct file comparison easier. The following are the only differences between the files that are used for an embedded topology and for the remote topology:

- ▶ The `objectGrid.xml` (for embedded use) file uses `NO_COPY` on its three `backingMap` definitions. Whereas the `objectGridStandalone.xml` (for remote use) file uses `COPY_TO_BYTES`. When embedded, there is no point in making a copy of the session data, as it is updated directly in the JVM heap.
- ▶ For the `sessionMapSet` definition, the `objectGridDeployment.xml` (for embedded use) file uses five partitions per application server JVM. Conversely, the `objectGridDeploymentStandalone.xml` (for remote use) file uses a fixed number of 47 partitions that are spread across all available WebSphere eXtreme Scale container JVMs.

Other than these minor differences, the two sets of files are identical. You can tune them to your own environment, but these sample files are typically used as-is and renamed if necessary.

Both of the sample deployment files configure `maxAsynchReplicas=1`. This provides a copy of each `HttpSession` held in the grid, allowing for fail over of a WebSphere eXtreme Scale container server JVM.

Each application server client request for `HttpSession` uses the primary partition copy of that data. Fail over of an application server does not require any extra work on the grid, as all application servers use the same primary `HttpSession` data for a specific session. You do not have to worry about concurrent writes to the same `HttpSession` because only one application server at a time is used to service an incoming request from a user (normally the “affinity” application server).

6.2.6 HTTP session servlet filter configuration

No matter which Java Platform, Enterprise Edition web container technology is used, the WebSphere eXtreme Scale HTTP session servlet filter, running on the web container client, requires certain topology and tuning for your usage. The manner in which this information is supplied differs by scenario. However, the basic list of required information is the same, and is summarized here.

The names shown here exactly match those required in the `splicer.properties` file in 6.4, “Scenario 1: Implementing with WebSphere Application Server Network Deployment” on page 260. The same general names are also used in configuration for the other scenarios described in 6.5, “Scenario 2: Implementing with WebSphere Liberty Profile” on page 281 and 6.6, “Scenario 3: Implementing with Apache Tomcat” on page 288.

objectGridType	The grid topology, either <code>REMOTE</code> or <code>EMBEDDED</code> .
objectGridName	The name of the grid, for example <code>session</code> .
catalogHostPort	How to connect to the WebSphere eXtreme Scale catalog servers, specified as a string of <code>hostname:port<hostname2:port2<...>></code>
replicationInterval	The number of seconds to wait between batching of writes to the grid. Using a longer interval helps improve performance, whereas using a shorter interval reduces the potential amount of session data that is lost during a fail over. It can be set to 0 to indicate no delay, which means that any changed session data is written immediately upon the end of each servlet execution.
sessionTableSize	For the <code>REMOTE</code> topology, specifies the number of sessions to keep locally in the client JVM, providing a local cache for better performance. It is ignored for an <code>EMBEDDED</code> topology.
fragmentedSession	If set to <code>true</code> (the default), each session attribute is written as a separate key/value pair to the grid. If <code>false</code> , the entire session is written as one large key/value pair. There are performance pros and cons to each approach, and the correct value depends on how your application uses <code>HttpSession</code> . However, for most applications, the default setting works best.
reuseSessionId	If set to <code>true</code> , web containers on different hosts reuse the same session id for the same user session. If <code>false</code> , each host creates its own unique session id to represent the same user session. The default is <code>false</code> , but is then overridden to <code>true</code> in certain installation scenarios. It is best to check this and explicitly set it yourself to make sure that you are getting the behavior that you want.

This lists only the most commonly used configuration settings. The complete list is available in the *splicer.properties file* topic in the WebSphere eXtreme Scale version 8.5 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/topic/com.ibm.websphere.extremescale.doc/rxsssplice.html>

6.3 Introducing the SessionTest sample application

A simple application has been created to demonstrate the functionality of HTTP session for the scenarios explained later in this chapter. The application is a JSP called `test.jsp` that displays a page of data on your browser about the environment in which it is running.

The source code for the `test.jsp` application is provided in the samples compressed file that is described in Appendix A, “Additional material” on page 341.

For this discussion, the pertinent part of the JSP is a snippet of Java code that is shown in Example 6-5.

Example 6-5 Snippet of the sample test.jsp application

```
...
Integer counter = (Integer) session.getAttribute("counter");
if (counter == null) {
    counter = new Integer(0);
} else {
    counter = new Integer(counter.intValue() + 1);
}
session.setAttribute("counter", counter);
...
```

The logic shown in Example 6-5 gets (using the `getAttribute` API) a `counter` value from the user `HttpSession`, then increments that value for display on the JSP output page.

If this is the first-time use of the `test.jsp` application, there will not have been an `HttpSession` created for it yet. The default behavior of any JSP upon entry is to implicitly run a `session=request.getSession(true)` call to find the associated `HttpSession` for this user, and if not found, create an empty `HttpSession`. In either case (existing or new), it is assigned to the `session` object for use within the JSP logic.

The `test.jsp` application uses this default behavior. Therefore, if it finds that the `counter` fetched from `HttpSession` is null, it knows that this is the first execution of the JSP, and it initializes the `counter` to 0. The `counter` (either 0 for first-time use, or increased by 1 for subsequent uses) is then saved (`setAttribute` API) into the `HttpSession`. This `counter` value is then shown on the page of data produced by the JSP.

Other logic within the JSP is used to find and display various other interesting factors about the environment in which it runs. These include the host name, IP address, and port used to access the application, and the session ID extracted from the content of the `JSESSIONID` cookie.

The JSP also displays the value obtained from a configured JVM custom property called `com.ibm.websphere.servlet.application.host`. If running in a WebSphere Application Server environment or a WebSphere Liberty profile environment, this JVM custom property is automatically added to the application server's custom properties when the application server is created, as a convenience to provide the unique name of each application server.

If running in a non-WebSphere environment, this custom property likely will not be found, so it is shown by the `test.jsp` as blank. You can manually add it to your non-WebSphere Application Server's custom properties if you want.

Figure 6-3 shows the first use of the `test.jsp` application. Note that the counter is set to 0.



The screenshot shows a web browser window with the title "HTTP Sample application". The address bar displays "whost01.rtp.ibm.com:8080/SessionTestWeb/test.jsp". The page content is divided into two sections: "Host information" and "Session information".

Host information	
Hostname	whost02.rtp.ibm.com
IP address	9.42.171.47
Port	9080
Application server (WAS and Liberty only)	app02

Session information	
Session ID	_Oh20v2FFkZOZs9WZJl0vnD
Counter:	0

Figure 6-3 Example of the first use (counter = 0) of the `test.jsp` application

If the JSP is run a second time (by refreshing your browser), the following is true:

1. The plug-in logic running on the web server finds the `JSESSIONID` cookie in the incoming request and parses it to find the session-id and clone-id portions from that cookie.
In Figure 6-3, only the true session-id portion is shown. This is because that is the only part of the `JSESSIONID` that is returned by the `session.getId()` API used by the `test.jsp` application.
If you want to see the entire `JSESSIONID` content, including its epoch prefix and clone-id suffix portions, you can use `request.getHeader("Cookie")` to retrieve the raw cookie content and then parse the results. There is no guarantee that the current syntax for `JSESSIONID` will remain the same in future releases, so do not build any application dependency on it.
2. The plug-in then routes the incoming request to the same affinity application server, in this example `app02`.
3. The `test.jsp` application running again on `app02` uses the session ID from the `JSESSIONID` cookie to locate the original `HttpSession` within its JVM heap.
4. The `test.jsp` logic increments the counter found in that `HttpSession`. The counter now shows 1 for its value.

Figure 6-4 shows the results after five executions, all running on the same app02 application server, incrementing the counter within the HttpSession to 5.



Host information	
Hostname	whost02.rtp.ibm.com
IP address	9.42.171.47
Port	9080
Application server (WAS and Liberty only)	app02
Session information	
Session ID	_Oh20v2FFkZOZs9WZJl0vnD
Counter:	5

Figure 6-4 Example of the test.jsp application after running five times

6.4 Scenario 1: Implementing with WebSphere Application Server Network Deployment

This section shows you how to introduce WebSphere eXtreme Scale session management into an existing web application running within a WebSphere Application Server Network Deployment environment. This scenario illustrates a remote topology that uses an integrated WebSphere eXtreme Scale client and server installation.

Because the intent in this example is to show how an already deployed application can be converted to using WebSphere eXtreme Scale, the instructions start by deploying a SessionTest application (for more information, see 6.3, “Introducing the SessionTest sample application” on page 257) into an existing cluster of application servers. This is the same cluster that was created in 5.2, “Integrating in a WebSphere Application Server Network Deployment environment” on page 178. This installation is completely normal, and uses the default session manager that is provided by the WebSphere Application Server web container.

After the application is tested to work properly with the default session manager, details are provided on how to switch to WebSphere eXtreme Scale for its session management. A simple fail over scenario is also tested.

This scenario starts with the premise that you have already created the topology that is described in 5.2.5, “Introducing the sample topology” on page 187. It then walks through the additional steps that are required to install and configure a sample application to use WebSphere eXtreme Scale as its HTTP session manager.

If you already have WebSphere eXtreme Scale installed and know when first deploying your application that you wanted to use WebSphere eXtreme Scale as its session manager, there are a few short cuts. As a new application is deployed, you can use the deployment pages to select and configure WebSphere eXtreme Scale for its session management. For more information, see the *Configuring WebSphere Application Server HTTP session persistence to a data grid* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/common/tsessionapp.html>

6.4.1 Installing the sample SessionTest application

As described in “Installing the grid client application” on page 215, you must install the SessionTest.ear application into the AppCluster application server cluster just as you installed the GettingStartedClient.ear. You can leave the GettingStartedClient application installed on the same AppCluster.

Sample application: The sample application described in this section, SessionTest.ear, is provided in the WASND folder of HTTPSession.zip file available as additional material for this book. For information about downloading this material, see Appendix A, “Additional material” on page 341. For more information about this sample application, see 6.3, “Introducing the SessionTest sample application” on page 257.

Regenerate the web server plug-in and restart the web server to pick up the routing definition for the newly installed application.

Plug-in regeneration: This plug-in regeneration and web server restart process is not always necessary. You can configure your environment to automatically regenerate the plug-in whenever a configuration change is made. You can configure it to automatically propagate the plug-in to the web server tier. And you can configure the plug-in to periodically check and refresh its configuration if the configuration file changes. Therefore, if you have all of this set up properly, you do not need to perform the manual restart steps. But in this simple example, it is better to follow the complete steps to make certain that everything is refreshed.

Generally, ensure that the application works properly using the initial configuration before you continue with the WebSphere eXtreme Scale configuration changes described in the remainder of this section.

Open a web browser and try accessing the test.jsp application within the deployed SessionTest application using the following URL, where the *hostname* and *port* are as you defined them during the creation of the environment described in Chapter 5, “Deployment scenarios” on page 151:

`http://hostname:port/SessionTestWeb/test.jsp`

A window similar to that shown in Figure 6-3 on page 259 is displayed. The counter shown on the window is 0 because this is the first use of the test.jsp application. Therefore, a new, empty HttpSession was created. If you refresh the browser several times, observe that the session remains “sticky” on the same application server, and that the counter continues to increment. If this occurs, you have correct functionality of the default session manager, and correct web server plug-in affinity.

If you observe that the hostname, port, or application server name changes as you refresh the browser, session affinity is not working properly. The most likely cause of this is that you

neglected to regenerate the web server plug-in and restart the web server after an application server was added or removed from the cluster configuration. The plug-in logic is unable to match the clone-id from an incoming JSESSIONID cookie to an application server in the cluster that it knows about. Therefore, it defaults to choosing any available application server in the cluster.

No session persistence or replication has been defined yet for the default session manager. Therefore, if you were to intentionally fail the application server that is being used (app02 in this example) by stopping that application server and then refreshing the browser again, the following steps would occur:

1. A new request for the same URL arrives at the web server, with the same JSESSIONID content that was saved by the browser from the earlier response.
2. The plug-in parses that incoming JSESSIONID cookie, and finds the clone-id representing the application server app02 in the cookie.
3. The plug-in attempts to maintain session affinity by passing this new inbound request to the matching application server, app02.
4. The attempted network connection to the app02 application server fails because the application server is no longer available.
5. Automatically handling the failure, the plug-in tries the request again using another application server within the same defined application server cluster, AppCluster. This is either to the next-in-line application, app03, or a random application server in the cluster, according to the configured plug-in round-robin or random setting. For this example, it chooses app03.
6. App03 receives the request, parses the incoming JSESSIONID, extracts the session ID portion, and attempts to find the corresponding HttpSession within its JVM heap. But it fails to locate the session because the session was created on a different application server and no session replication nor persistence has been configured.
7. A new, empty HttpSession is returned to the test.jsp application that runs on App03 application server, which therefore considers this a first-time request by this user and reinitializes the counter to 0.

These steps simulate what happens in the case of a fail over when there is no session persistence nor replication configured. An example of this in the real e-commerce world is that the user might lose their shopping cart content. This lost data might cause them to abandon the website altogether, therefore losing a sale.

The default session manager within the WebSphere Application Server can be configured to persist HTTP session content to a database or replicate it in-memory to another JVM in the cluster. Either of these configurations solves this fail over scenario. App03 can recover the original HttpSession and maintain a smooth incrementing of the counter.

The Information Center for WebSphere Application Server Network Deployment documents how you can set up traditional session persistence or replication. However, the intent of this section is to document how you can instead use WebSphere eXtreme Scale session management to accomplish a similar highly available environment for your HttpSession data with the added benefits described in 6.2.1, “Benefits of using WebSphere eXtreme Scale for HTTP session management” on page 250.

6.4.2 Configuring WebSphere eXtreme Scale session management

To switch to using eXtreme Scale for HTTP session management in a WebSphere Application Server environment, complete the following high-level steps. Several of these steps were

already done when creating the sample topology in 5.2, “Integrating in a WebSphere Application Server Network Deployment environment” on page 178.

1. Install the WebSphere eXtreme Scale product binary files on the client side and the grid side. This was already done for the sample topology in “Installing WebSphere eXtreme Scale” on page 191.
2. Augment your existing Deployment Manager and Application Server profiles with WebSphere eXtreme Scale. Or alternatively, if this is a brand new WebSphere Application Server Network Deployment installation, you can initially create the Deployment Manager and Application Server profiles using the eXtreme Scale augmentation. This second approach was used in “Creating WebSphere eXtreme Scale profiles” on page 198.
3. Create your WebSphere eXtreme Scale catalog and container server clusters. This was completed in “Creating the clusters” on page 204.
4. Define a catalog server domain so that the WebSphere Application Server clients know how to connect to the WebSphere eXtreme Scale catalog servers. For WebSphere eXtreme Scale session management, this step is only required if the catalogs are outside of the client’s cell, which is not true in the sample topology that is used here. In any case, it was done in “Defining the catalog service domain” on page 206.
5. Deploy a specially constructed application into the container cluster. This application configures the containers to support the WebSphere eXtreme Scale HTTP session management needs. This process is described in 6.4.3, “Installing the HTTPSessionGrid application into the container servers” on page 263.
6. Reconfigure the existing client application to use WebSphere eXtreme Scale for its session management. This process is described in 6.4.4, “Reconfiguring to use WebSphere eXtreme Scale session management” on page 270.
7. Test the new configuration, including fail over testing, which is covered in 6.4.5, “Testing the application” on page 279.

As noted, steps 1 through 4 were already done in the detailed setup for the sample WebSphere Application Server Network Deployment with WebSphere eXtreme Scale topology in Chapter 5.

For step 6, a sample application has already been deployed and tested with the default session manager, providing an existing client application to reconfigure. All that remains to be done is the work in steps 5, 6, and 7.

6.4.3 Installing the HTTPSessionGrid application into the container servers

The container servers defined within the GridCluster must be started with the special remote versions of the sample HTTP session manager grid configuration files. That is, the `objectGridStandalone.xml` and `objectGridDeploymentStandalone.xml` files that were introduced in 6.2.5, “Sample grid configuration files for an HTTP session” on page 253.

Note: Although this sample environment is running with an integrated (not stand-alone) remote topology, the “Standalone” named files are used because they are appropriate for a remote (rather than embedded) topology. These sample files should be more properly named `objectGridRemote.xml` and `objectGridDeploymentRemote.xml`.

If you want to host the container servers in a truly *stand-alone* remote environment separate from your WebSphere Application Server installation, start those containers using the same `objectGridStandalone.xml` and `objectGridDeploymentStandalone.xml` files. This is a topology that is often used by WebSphere eXtreme Scale customers. However, in the this

scenario, the catalogs and containers are configured from an *integrated* WebSphere eXtreme Scale installation. In this topology, the easiest way of starting the container servers with the appropriate grid configuration files is to package and deploy them onto the container cluster in a simple application that contains them.

Structure of the HTTPSessionGrid application

After a WebSphere Application Server profile is augmented with WebSphere eXtreme Scale, every application startup is monitored for the presence of an `objectGrid.xml` and `objectGridDeployment.xml` file in its META-INF directory. This is described in “Simplified grid container management” on page 181. If only the `objectGrid.xml` is found, it is considered to be an eXtreme Scale *client* application. If *both* files are found, it is considered to be an eXtreme Scale *container*. This convenient method of container configuration is used by this scenario.

Figure 6-5 shows the HTTPSessionGrid application as it is displayed in an Eclipse workspace. It is exported and deployed onto the grid container cluster as a standard enterprise archive (EAR), HTTPSessionGrid.ear, which contains a single web archive (WAR), HTTPSessionGridWeb.war. The WAR file in turn contains the necessary `objectGrid.xml` and `objectGridDeployment.xml` files that are used by the container startup as its default configuration file names.

Important: For this remote grid scenario, these two files have been copied and renamed from the original `objectGridStandalone.xml` and `objectGridDeploymentStandalone.xml` files.

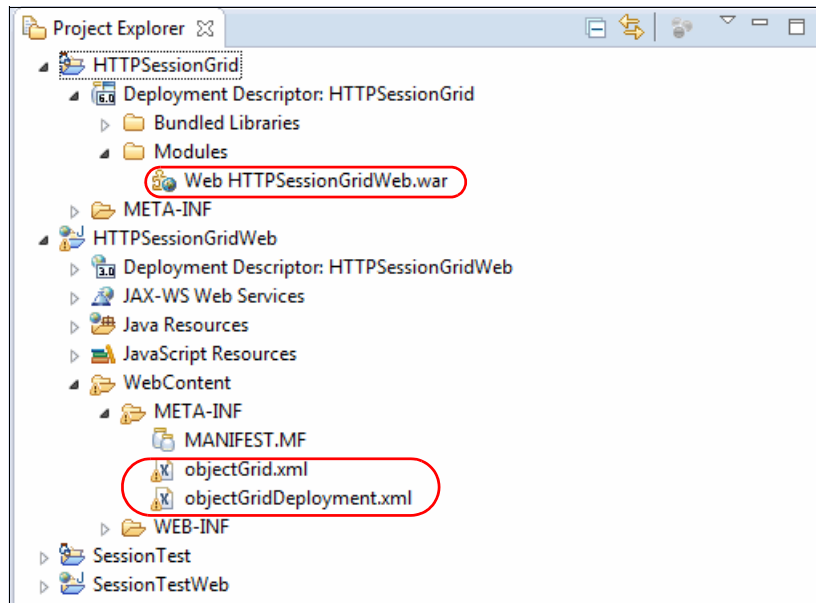


Figure 6-5 Grid configuration files packaged into a simple HTTPSessionGrid application

There is no actual application logic within this trivial HTTPSessionGrid application. It is just a convenient means of defining the container configuration in an integrated environment. There are no class files, no library jars, and no resources. Also, there is nothing in any of the non-expanded directories as shown in Figure 6-5. The application is simply these two XML files, which are packaged into the standard WAR-within-EAR structure that is required by Java Platform, Enterprise Edition application deployment.

Deploying the HTTPSessionGrid application to the GridCluster

The HTTPSessionGrid project is exported from Eclipse as an EAR. It can be placed anywhere. In this scenario, which is running in a Windows environment, it is exported into the C:\Apps\HTTPSessionGrid.ear directory.

The following steps show you how the application is then deployed into the GridCluster. There is nothing unusual here. It is just another enterprise application being deployed. The only thing to pay attention to is on the deployment step where you choose which servers to deploy to, you must choose the GridCluster.

1. Log in to the Deployment Manager console and navigate to **Applications** → **New application**.
2. Browse to and select the **HTTPSessionGrid.ear** application to install (Figure 6-6).

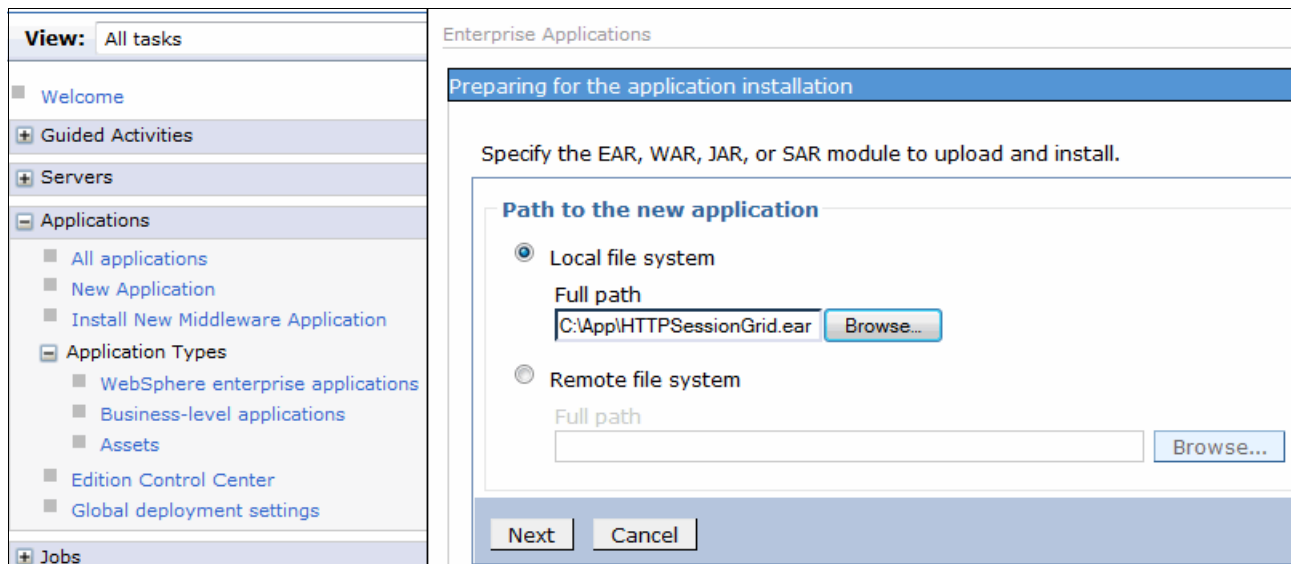


Figure 6-6 Enterprise application deployment, selecting the application EAR file to be deployed

3. As shown in Figure 6-7, select the **Fast Path** deployment, which bypasses several deployment detail pages. Again, this is a normal application deployment that needs no special handling here. Click **Next**.

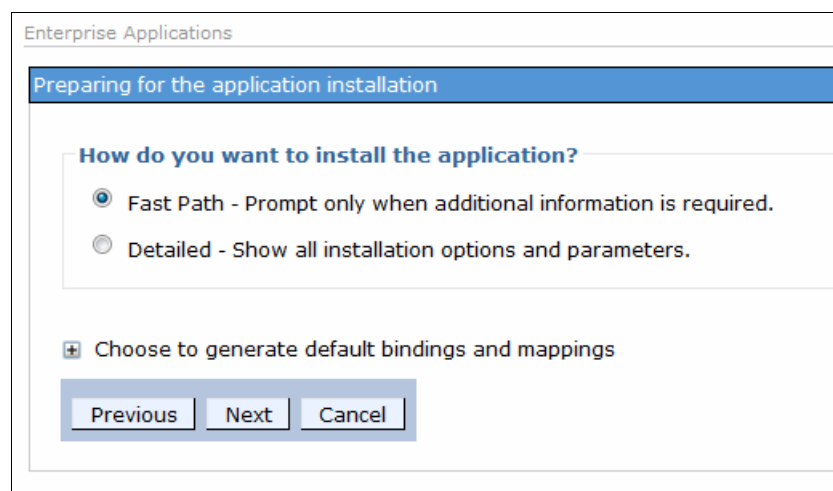


Figure 6-7 Selecting fast path deployment

4. On the deployment Step 1 page (Figure 6-8), leave all of the defaults and click **Next**.

Install New Application

Specify options for installing enterprise applications and modules.

→ **Step 1: Select installation options**

[Step 2: Map modules to servers](#)

✦ [Step 3: Metadata for modules](#)

[Step 4: Summary](#)

Select installation options

Specify the various options that are available for your application.

- Precompile JavaServer Pages files

Directory to install application

- Distribute application
- Use Binary Configuration
- Deploy enterprise beans

Application name

Application edition

Edition description

- Create MBeans for resources
- Override class reloading settings for Web and EJB modules

Reload interval in seconds

- Deploy Web services

Validate Input off/warn/fail

- Process embedded configuration

File Permission

Application Build ID

- Allow dispatching includes to remote resources
- Allow servicing includes from remote resources

Business level application name

Asynchronous Request Dispatch Type

- Allow EJB reference targets to resolve automatically
- Deploy client modules
Client deployment mode
- Validate schema

Figure 6-8 Deployment step 1

- On the deployment Step 2 page (Figure 6-9), select the GridCluster as the target of the deployment, and click **Apply** then click **Next**.

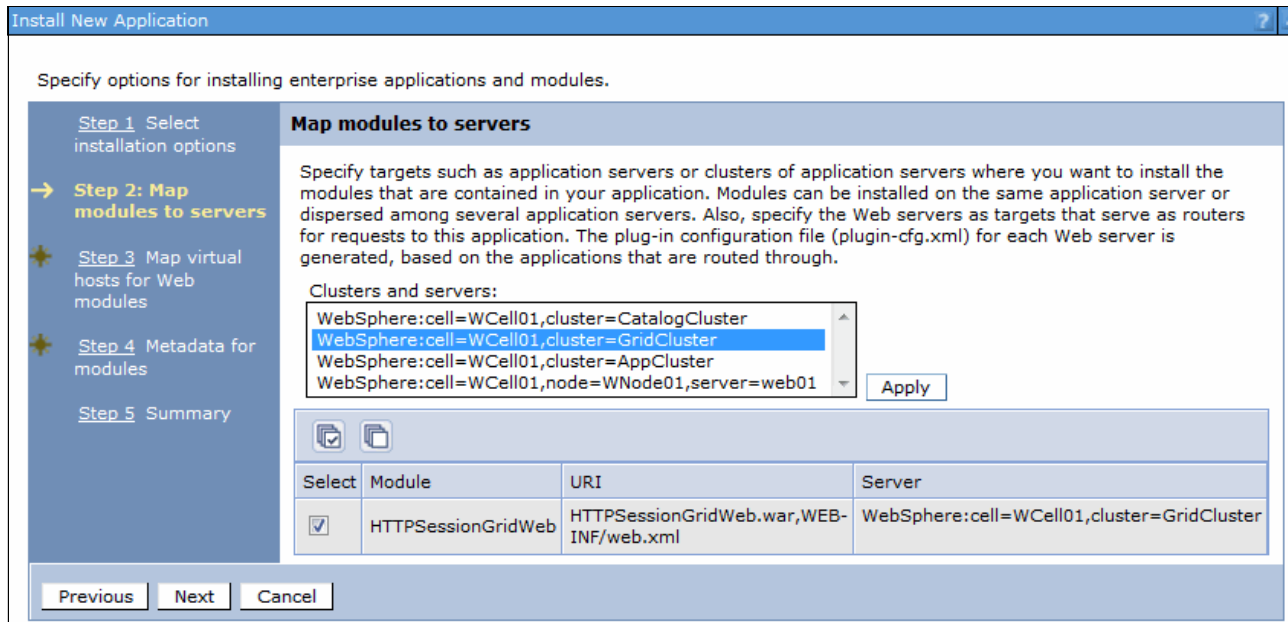


Figure 6-9 Selecting the appropriate container cluster to deploy the application

- On the deployment Step 3 page (Figure 6-10), accept the defaults and click **Next**.

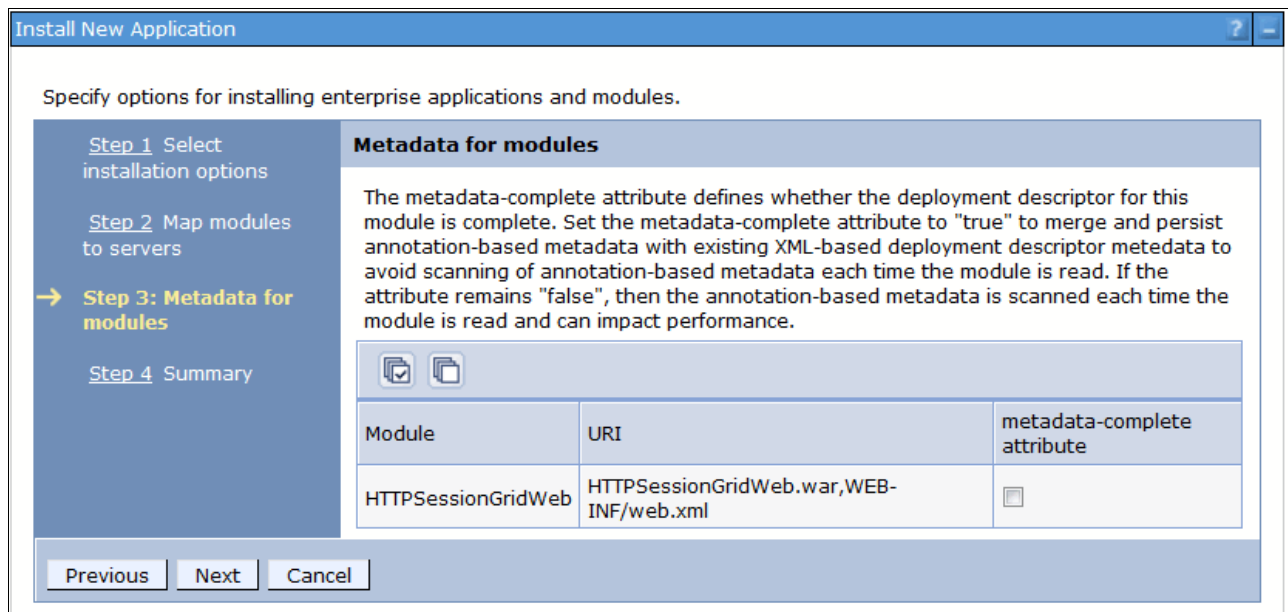


Figure 6-10 Accepting the default for metadata for modules

- The deployment Step 4 page (Figure 6-11) provides a summary of all of the deployment options selected. Review the data, and, if correct, click **Finish** to run the deployment.

Install New Application

Specify options for installing enterprise applications and modules.

[Step 1](#) Select installation options

[Step 2](#) Map modules to servers

[Step 3](#) Metadata for modules

→ Step 4: Summary

Summary

Summary of installation options

Options	Values
Precompile JavaServer Pages files	No
Directory to install application	
Distribute application	Yes
Use Binary Configuration	No
Deploy enterprise beans	No
Application name	HTTPSessionGrid
Application edition	
Edition description	
Create MBeans for resources	Yes
Override class reloading settings for Web and EJB modules	No
Reload interval in seconds	
Deploy Web services	No
Validate Input off/warn/fail	warn
Process embedded configuration	No
File Permission	.*\,dll=755#.*\,so=755#.*\,a=755
Application Build ID	Unknown
Allow dispatching includes to remote resources	No
Allow servicing includes from remote resources	No
Business level application name	
Asynchronous Request Dispatch Type	Disabled
Allow EJB reference targets to resolve automatically	No
Deploy client modules	No
Client deployment mode	Isolated
Validate schema	No
Cell/Node/Server	Click here

No application modules were mapped to web servers. The plug-in configuration file (plugin- each web server is generated based on the application modules which are mapped to it, theref server will route requests to this application. To change this option, select the Map modules to step.

Previous
Finish
Cancel

Figure 6-11 Application deployment summary page

- The deployment proceeds, and a window similar to that shown in Figure 6-12 is displayed as the application is deployed and synchronized across all nodes that host the container cluster.

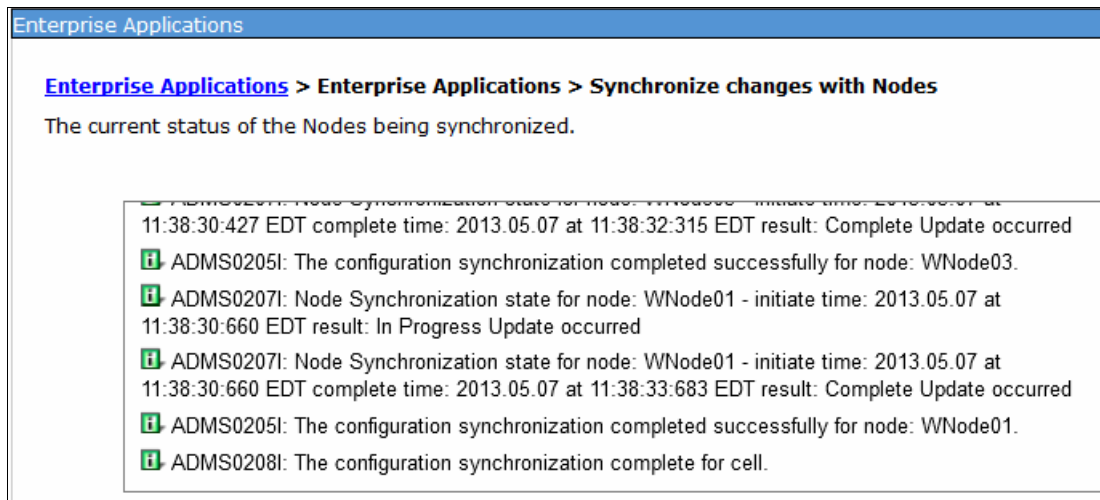


Figure 6-12 Application deployment and synchronization completed

- If your grid container cluster is already started, you can start the new HTTPSessionGrid application within it as shown in Figure 6-13. Click **Applications** → **Enterprise applications**, select HTTPSessionGrid, and click **Start**. If the container cluster is not already started, then start it now, which also starts the deployed HTTPSessionGrid application within it.

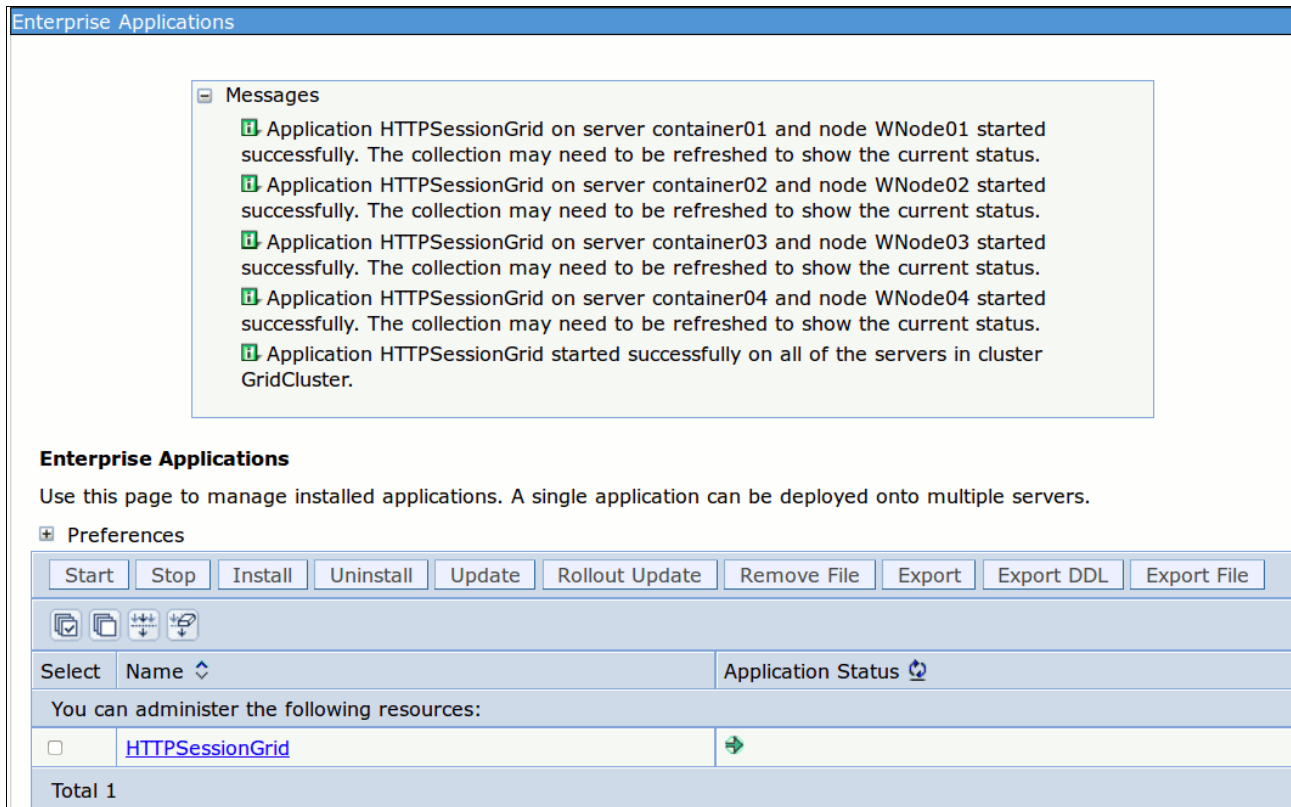


Figure 6-13 Starting the HTTPSessionGrid application on the container cluster

For this new HTTPSessionGrid application, there is no need to regenerate the web server plug-in or restart the web servers because the application is never accessed from a browser.

6.4.4 Reconfiguring to use WebSphere eXtreme Scale session management

The SessionTest application is now deployed into the AppCluster, but it is using the default session management that is provided by WebSphere Application Server. The WebSphere eXtreme Scale container cluster has been configured and started to hold the HTTP session data. The WebSphere eXtreme Scale catalog server cluster is also presumed started and ready.

Choice of session manager configuration scope

The SessionTest application must be reconfigured to use WebSphere eXtreme Scale session management. However, there are three ways of approaching this configuration, each with its own advantages and disadvantages:

- ▶ Configuring session management at enterprise application scope

You might have several applications deployed within your application server cluster. That is the case with the example topology because the original GettingStartedClient application is installed alongside the SessionTest application in the AppCluster. It might be that you do not want all of the applications using WebSphere eXtreme Scale as the session manager. Configuring session management per application at the application scope level gives you this flexibility.

However, the more practical advantage in configuring session management at application scope is that, when the application is deployed into many application servers in a cluster, the application-scoped configuration applies equally across those application servers. That is, you only need to do the configuration in one place, saving time when compared to the application server scope described in the next bullet.

Reach the application-scoped session configuration window by opening in the WebSphere Application Server administration console and clicking **Applications** → **Application Types (expand)** → **WebSphere enterprise applications** → *appName* → **Session management**.

The disadvantage with this approach is when you have several applications deployed into your application server cluster and want them to all use the same WebSphere eXtreme Scale session management configuration, you must configure each application separately.

- ▶ Configuring session management at application server scope

You can do your session management configuration at the application server scope by opening the WebSphere Application Server administration console and clicking **Servers** → **Server types** → **WebSphere application servers** → *serverName* → **Session management**.

Any session management configuration that is done at the application server scope affects all applications that are running in that application server. That includes all current and any future applications that are deployed into the application server.

- ▶ Combining both approaches, application scope and server scope

It is also possible to use a combination of both application scope and server scope. You can configure your default session management behavior for all applications at the application server scope, and then specifically override that configuration for certain applications at their enterprise application scope.

Important: Consider carefully before proceeding with this option, as it might be difficult for future administrators to figure out. If you do choose to proceed, make sure that you maintain detailed reference documentation for the configuration.

Override session management check box

You will find that both of the session manager configuration pages reached by either the application scope or server scope look identical. However, there is one important difference. The application-scoped session manager configuration page starts with a check box that is selected to override session management.

If the override session management check box is not checked, any application-scoped session manager configuration is ignored. If the check box is checked, the application-scoped configuration completely overrides (effectively replaces) the same settings configured at server scope for this application.

In the end, there is no correct answer for which approach to choose, other than to use the one that works best for your needs. In the setup described in “Steps to reconfigure session management for SessionTest application” on page 272, the session management configuration is done at the SessionTest enterprise application scope.

WebSphere Application Server administrative console fields

Section 6.2.2, “Integration of WebSphere eXtreme Scale session management” on page 251 details the means by which WebSphere eXtreme Scale session management gains control for HTTP session processing. Because eXtreme Scale overrides most of the default session manager processing, many fields of the session manager configuration pages work differently when you select WebSphere eXtreme Scale as the session manager.

Figure 6-14 on page 272 is annotated to show these important differences. Note that this is of a session-scoped configuration page, but the discussion applies equally to the application-scoped page.

With WebSphere eXtreme Scale session management, many of these session management configurations are done elsewhere. Specifically, a new `splicer.properties` file is used. For more information, see “The `splicer.properties` file” on page 276.

Allow overflow check box: Pay particular attention to the fact that you must always select the **Allow overflow** check box when you use WebSphere eXtreme Scale. Without this option selected, the application server imposes a cap on the number of sessions that WebSphere eXtreme Scale can hold in memory. This limit can interfere with the WebSphere eXtreme Scale management of its own configured in-memory size (done with a `splicer.properties` file setting).

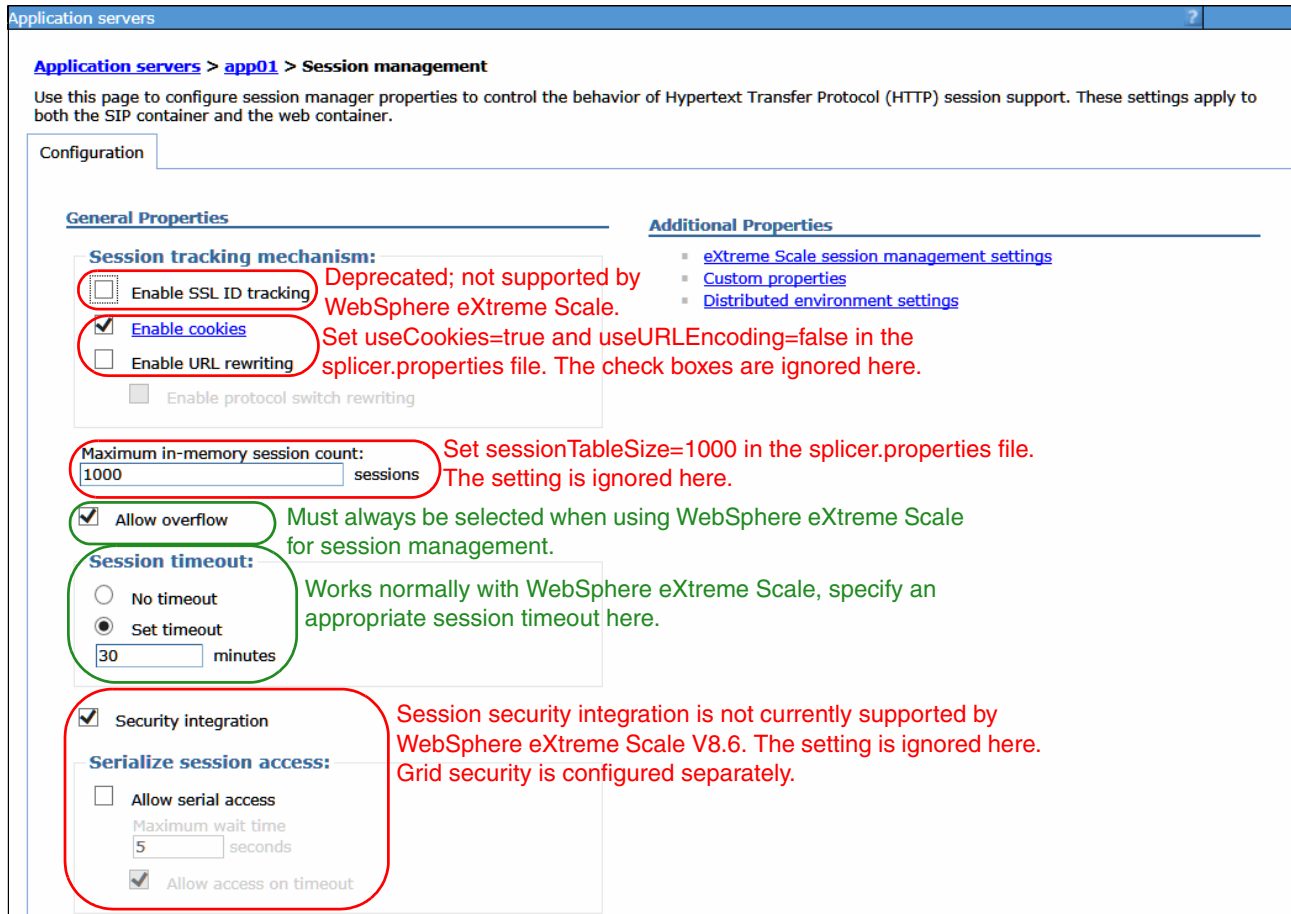


Figure 6-14 Using WebSphere eXtreme Scale session management

Steps to reconfigure session management for SessionTest application

Complete the following steps to reconfigure session management for the SessionTest application to use WebSphere eXtreme Scale:

1. In the WebSphere Application Server administration console, click **Applications** → **Application Types (expand)** → **WebSphere enterprise applications** → **SessionTest** → **Session management**, as shown in Figure 6-15 on page 273.
 - Select the **Override session management** check box on this application-scoped session management configuration page. This overrides any application server-scoped session management settings.
 - Click the **eXtreme Scale session management settings** link.

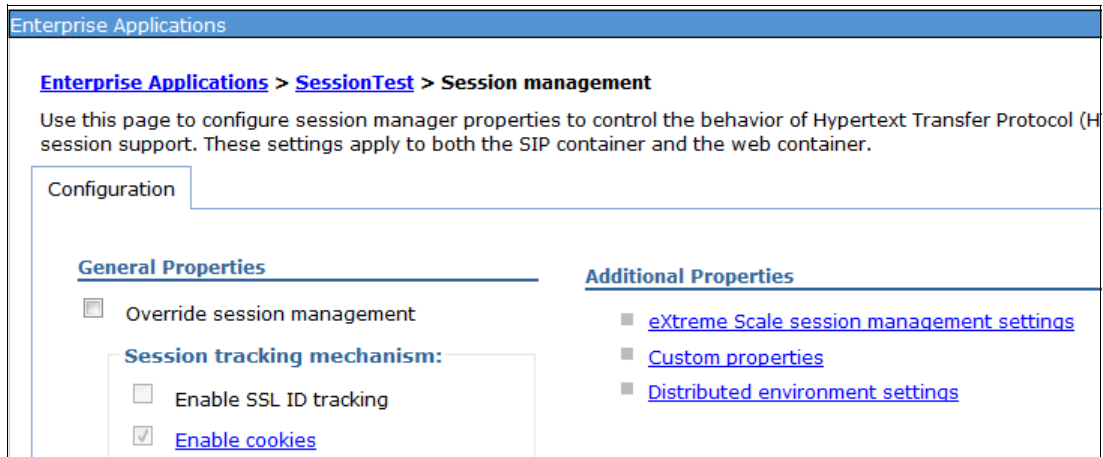


Figure 6-15 The Session management page of the SessionTest application

2. You are now on a page shown in Figure 6-16:
 - Select the **Enable session management** check box. If this is not selected, eXtreme Scale is not used for session management and instead the default session manager is used.
 - Select **Remote eXtreme Scale data grid** from the list as the topology that is being configured.
 - Select the configured catalog server domain, **CSD01**, to be used by the application server clients to locate the appropriate catalog server cluster. This catalog server domain was configured in “Defining the catalog service domain” on page 206.

Click **Browse**.

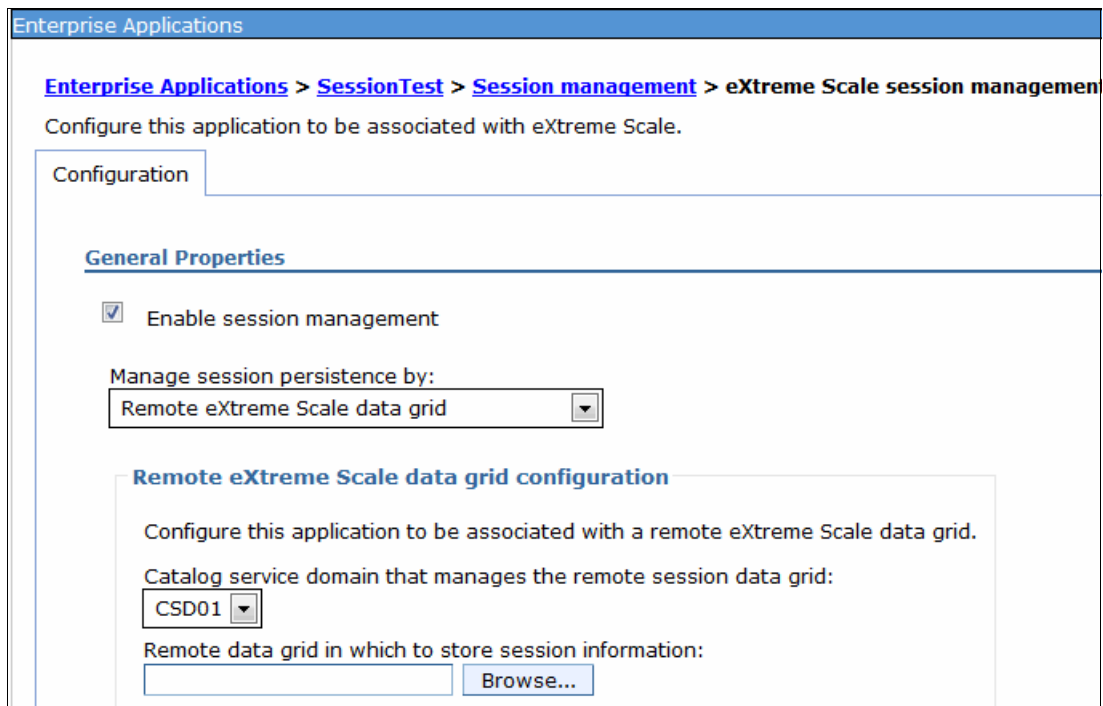


Figure 6-16 WebSphere eXtreme Scale session management configuration page

3. A window is displayed that lists all the grids that are currently defined in the environment as shown in Figure 6-17. In this example, there is only one. This is the grid called `session` by way of the `objectgridName="session"` line specified in the `objectGridDeployment.xml` file that is used to create this grid (Example 6-4 on page 256).

Select the `session` remote data grid. You are returned to the previous page with its name filled into the **Remote data grid name** field.

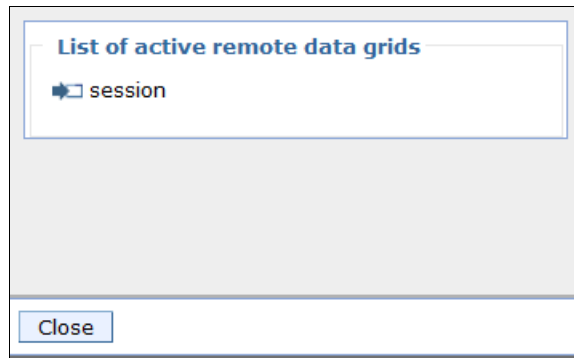


Figure 6-17 Selecting the remote data grid

4. On the eXtreme Scale session management configuration page, click **Apply** to accept these changes.
5. On the Session management configuration page, save your work as shown in Figure 6-18. Click **Save** to save the configuration to the master repository. By default (unless this is explicitly turned off), this also synchronizes the changes to all of the nodes.

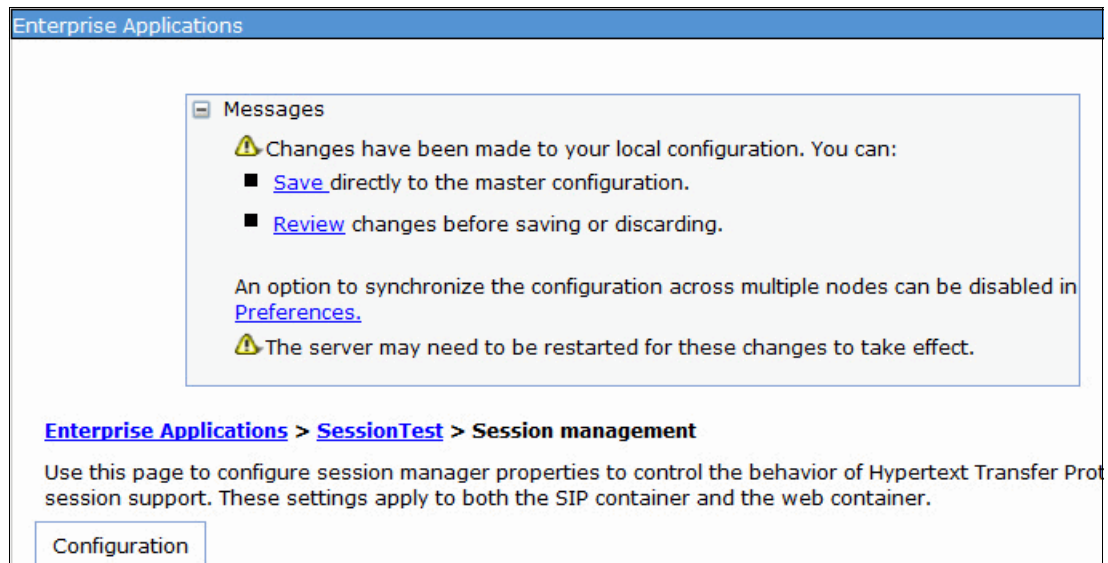


Figure 6-18 Confirmation to save the changes to the configuration

Syncing node configuration with the master copy

You now need to make sure that each of the node configurations is synchronized with the Deployment Manager configuration. This is not unique to this HTTP session scenario. It applies any time that you change a WebSphere Application Server Network Deployment configuration.

When you make changes using the WebSphere Application Server administration console and save them, the changes are actually saved to a configuration directory structure on the Deployment Manager node. This is the *master configuration repository*. Each of the nodes in the cell contains a copy of those portions of the master configuration that are relevant to that node. The node, and any application servers and applications that run on that node, use the node copy of the configuration data.

In a standard WebSphere Application Server Network Deployment setup, the Deployment Manager is configured to push any master repository changes to each of the nodes. The NodeAgent process running on each node also periodically (default is every minute) polls the Deployment Manager to see whether there have been any changes. If so, it initiates a “pull” of the updated configuration to their node.

The sample environment set up in 5.2, “Integrating in a WebSphere Application Server Network Deployment environment” on page 178 used all default settings, so the node configurations can be automatically synchronized. However, for the sake of completeness of this scenario, and as a good general practice, make certain that the node configuration files are in sync with the master copy.

Navigate to **System Administration (expand) → Nodes** to see a window like that shown in Figure 6-19. In the Status column to the right, a green circle indicates that the node is in sync with the master repository. If any of the nodes do not show green circles, force a synchronization. To accomplish this, select the check box beside the nodes that need synchronizing, and click **Synchronize**. You can also click **Full Resynchronize**, which takes longer but is more thorough.

Nodes

Use this page to manage nodes in the application server environment. A node corresponds to a physical computer system with a distinct IP host address. The following table lists the managed and unmanaged nodes in this cell. The first node is the deployment manager. Add new nodes to the and to this list by clicking Add Node.

Preferences

Add node Remove node Force delete Synchronize Full Resynchronize Stop

Select	Name	Host Name	Version	Discovery Protocol	Status
	WCellManager01	whost01.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	↔
<input type="checkbox"/>	WNode01	whost01.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	↔
<input type="checkbox"/>	WNode02	whost02.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	↔
<input type="checkbox"/>	WNode03	whost03.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	↔
<input type="checkbox"/>	WNode04	whost04.rtp.ibm.com	ND 8.5.0.2 WXS 8.6.0.1	TCP	↔

Total 5

Figure 6-19 Verifying all node configurations are up to date

Restarting the application to pick up the changes

Because you made configuration changes to an application that is running, it is important to stop and restart the application to pick up the configuration changes.

If you make configuration changes at the application server scope rather than enterprise application scope, you must restart the entire application server. Restarting an application

server restarts all of the applications that are installed on it, so doing this ensures that all configuration changes are made.

In this scenario, you made changes to the session manager at the enterprise application scope, so you are safe to restart the SessionTest application. Select one of the following approaches:

- ▶ To restart only the affected application:
 - a. Click **Applications (expand)** → **Application types (expand)** → **WebSphere enterprise applications**.
 - b. Select the **SessionTest** application and click **Stop**. Wait for the application to stop.
 - c. Select the SessionTest application again and click **Start**.
- ▶ To restart the cluster of application servers that host the SessionTest application:
 - a. Click **Servers (expand)** → **Clusters (expand)** → **WebSphere application server clusters**.
 - b. Select the **AppCluster** cluster and click **Stop**. Wait for all of the application servers to stop.
 - c. Select the **AppCluster** cluster again and click **Start**.

The splicer.properties file

WebSphere eXtreme Scale augmentation has now done some background processing to set the application up for eXtreme Scale session management. The reconfigured SessionTest application can actually be used now.

However, before you test it, this section introduces the `splicer.properties` file that was automatically created.

WebSphere custom property: The location of the WebSphere custom property is specific to configuring WebSphere eXtreme Scale session management at the application level. However, had the eXtreme Scale session management configuration instead been done at the application server level, the custom property is located elsewhere. Specifically, it is in the defined custom properties of the application server itself. The result is that the variable's value contains the location of the `splicer.properties` file that is used for the eXtreme Scale session management configuration and tuning.

The following processes happen automatically when you click **Save** on the confirmation window in Figure 6-18 on page 274:

1. The configuration changes are saved to the Deployment Manager master repository, as usual for a save operation in the WebSphere Application Server administration console.
2. A new custom property is added to the cell level. You can see this by clicking **System Administration (expand)** → **Cell** → **Custom properties**, as shown in Figure 6-20 on page 277.

As you can see, this new custom property name starts with the name of the enterprise application that has been “spliced.” Therefore, each application can have its own `splicer.properties` file for its own unique tuning needs. The variable's value provides the location of a `splicer.properties` file that was automatically constructed.



Figure 6-20 A new cell-scoped custom property that provides the location of the splicer.properties

3. The `${USER_INSTALL_ROOT}` shown in the value column is a reference to a WebSphere environment variable. These variables can be defined at application server, cluster, node, and cell level. If the same variable is defined at multiple levels, the earliest definition in that order is used.
4. To find the actual value of the variable as it will be used by the SessionTest application, click **Environment (expand)** → **WebSphere**. Start at the bottom of the variable hierarchy, which is the application server level, and search up the hierarchy until you find a matching name. In the example scenario, the search involves these steps:
 - a. On the WebSphere Variables page, use the Scope menu to select one of the application servers on which the SessionTest application is installed. For example, select the **Node=WNode01,Server=app01** application server scope. The window that lists the variables is refreshed to show you only the variables that are defined at that server scope. You will find that there is no `USER_INSTALL_ROOT` defined here, so continue your search with the next hierarchy up.
 - b. The next level up the hierarchy is cluster scope. Use the menu on the variables page to select the **Cluster=AppCluster** scope. In this case, there are no variables that are defined at this scope, so continue your search.
 - c. Next, select the node scope for one of the nodes hosting the application servers that run the SessionTest application. For example, select the **Node=WNode01** node. Here you find that there are several pages of WebSphere variables defined. They are ordered alphabetically. Click the arrow at the bottom of the page to progress to page 2 of the list.
 - d. As you page through the Node=WNode01 scoped variables, you find that there is a `USER_INSTALL_ROOT` variable defined at this level of the hierarchy.

The process is shown in Figure 6-21.

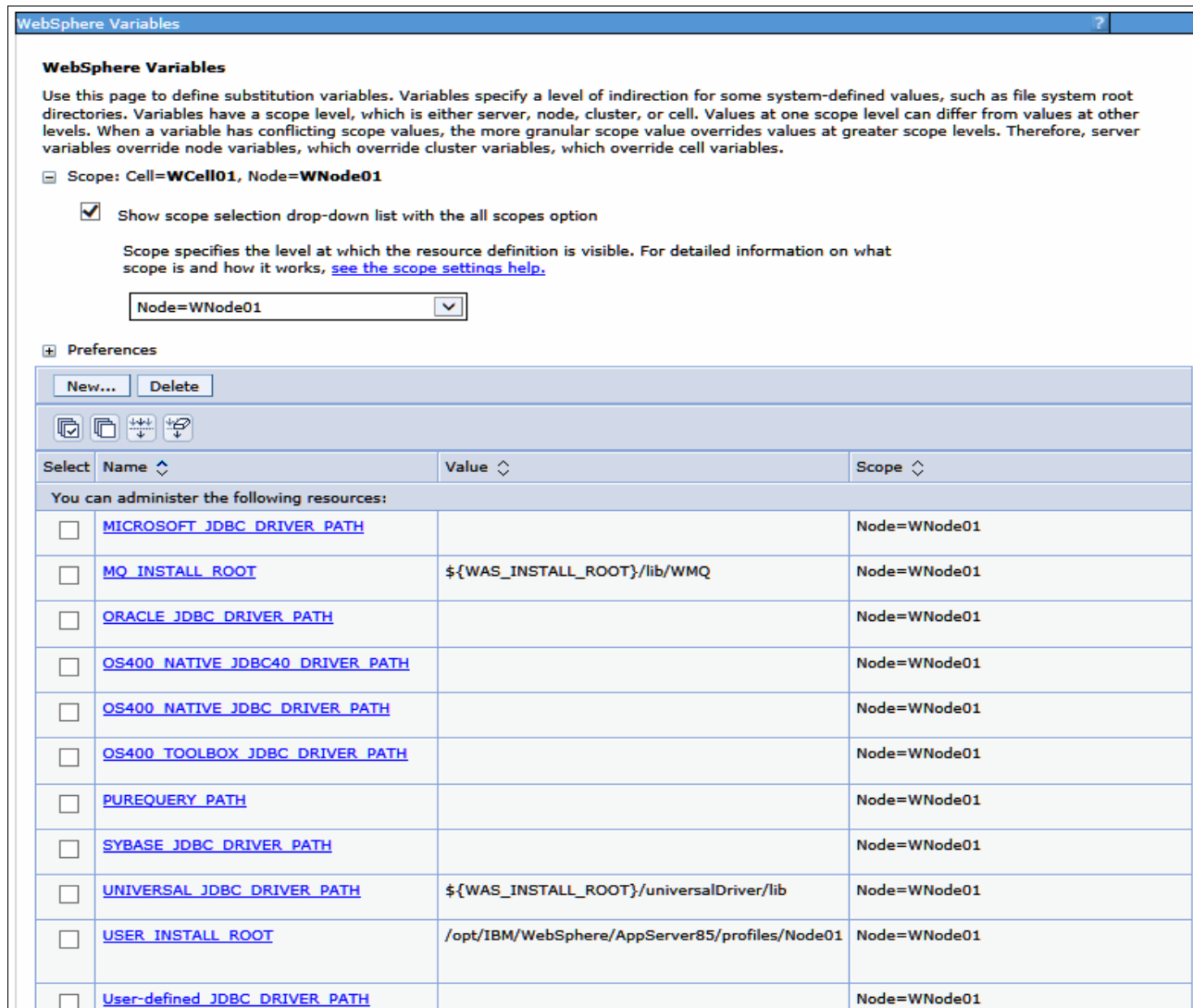


Figure 6-21 WebSphere variables for node=WNode1

- The value of the USER_INSTALL_ROOT variable that is used by the application on this node is /opt/IBM/WebSphere/AppServer85/profiles/Node01. Now you can go back to the original variable found in Figure 6-20 on page 277, and perform the appropriate substitution into that variable.
- The following is the fully expanded variable:


```
SessionTest.com.ibm.websphere.xs.sessionFilterProps =
/opt/IBM/WebSphere/AppServer85/profiles/Node01/config/cells/WCell01/application
s/SessionTest.ear/deployments/SessionTest/splicer.properties
```
- If you examine the splicer.properties file now, you find that it contains what is shown in Example 6-6.

Example 6-6 The splicer.properties file referenced by the cell-scoped custom property

```
# Generated by writeSplicerFile() at Tue May 07 13:43:39 EDT 2013
useURLencoding=false
useCookies=true
```

```
objectGridType=REMOTE
reuseSessionId=true
webSphereCatalogDomain=CSD01
objectGridName=session
replicationInterval=10
sessionTableSize=1000
fragmentedSession=true
sessionManagementType=XSRemoteSessionManagement
```

You need to know how to find the `splicer.properties` file so you can change its configuration if necessary.

This `splicer.properties` file is used by the servlet filter that was injected into the path of all servlet calls to the `SessionTest` application. The file provides the servlet filter with the information that it needs to know, such as the following information:

- ▶ Connect with the remote session grid using the CSD01 catalog server domain list.
- ▶ Hold 1000 sessions locally within the application server as a local cache.
- ▶ Use a `JSESSIONID` cookie, not URL rewriting, as the session tracking mechanism.
- ▶ Batch up writes to the grid for 10 milliseconds to reduce the impact of writing every update immediately.
- ▶ Write individual `HttpSession` attributes as key/value pairs to the grid, rather than the whole `HttpSession` as one monolithic key/value.

There are many other possible configuration and tuning properties that can be put into the `splicer.properties` file. For more information, see the WebSphere eXtreme Scale Version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxsssplice.html>

If you want to closely mimic the current tuning of a default session manager configuration in a WebSphere eXtreme Scale session manager configuration, see the *Configuring WebSphere eXtreme Scale to use your previous configuration settings* topic in the WebSphere eXtreme Scale Version 8.6 Information Center, which provides a mapping of the traditional session manager configuration field names to the `splicer.properties` property names:

http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsmigration_configurationapplication.html

6.4.5 Testing the application

You can now test the `SessionTest` application again, but this time it is using WebSphere eXtreme Scale session management.

For initial verification, you can follow the same steps from 6.4.1, “Installing the sample `SessionTest` application” on page 261. You see a similar window, with the initial count of 0. It is zero because this is the first touch of the application after it was restarted, so a new `HttpSession` is created for it.

Refresh your browser several times to ensure that the application remains “sticky” on the same application server, and that the counter properly increments.

To prove that it is using the WebSphere eXtreme Scale grid for its session management, log on to one of the nodes that hosts an eXtreme Scale catalog server (for example, WNode01 in the sample topology). Run an xscmd **showMapSizes** command to see the grid content.

Example 6-7 shows an example of the xscmd **showMapSizes** command and a small portion of its output. The fact that there is a primary partition that contains at least one map entry for objectgridSessionAttribute proves that the grid is being used to hold the HTTP session attribute data.

Example 6-7 Example of showMapSizes output, showing the HTTP session is being stored in the grid

```
/opt/IBM/WebSphere/AppServer85/profiles/Node01/bin/xscmd.sh -cep
whost01.rtp.ibm.com:4811 -c showMapSizes
```

```
*** Displaying results for session data grid and sessionMapSet map set.
```

```
*** Listing maps for WCell101\WNode02\container02 ***
```

Map Name	Partition	Map Entries	Used Bytes	Shard Type
objectgridSessionAttribute	26	1	280 B	Primary

Validating HTTP session failover

You might be interested in observing the behavior when an application server fails. Ensure that any HTTP session data used for user sessions on that failed application server is automatically recovered on another application server. Complete the following steps to verify HTTP session failover:

1. Continue refreshing your browser a couple of times to increase the counter. Make note of the final value and of the application server that it is running on.
2. Use the WebSphere Application Server administration console to stop the application server that the test.jsp was running on. Wait for it to finish stopping.
3. Refresh your browser again to restart the test.jsp. This time, the web server plug-in first attempts to send the request to the same application server (due to JSESSIONID cookie session affinity), but this attempt fails. It then automatically tries the request to another application server in the cluster.
4. A result panel from the test.jsp is displayed that shows that the count has been incremented properly, but that the application server name has changed. This confirms that the application server fail over worked properly. In a real-world application server failover, the browser user is typically unaware that any failure has occurred.

Validating WebSphere eXtreme Scale container failover

Another possible fail over scenario that you might want test is what happens if a WebSphere eXtreme Scale container server fails. In the example topology, there is a single asynchronous replica configured that keeps a copy of all of the primary partition data. Therefore, if the primary partition fails, your testing is unaffected. Run the following steps to confirm this configuration:

1. Run an xscmd **showMapSizes** command as shown in Example 6-7. Make note of the following attributes:
 - Partition number of one of the partitions that contains the Primary data for objectgridSessionAttribute. In this example, it is partition 26.
 - The container number that is hosting the Primary shard. In this example, it is container02, running on host WNode02.

- The container number that is hosting the `AsynchronousReplica` shard for the same numbered partition as the primary shard. For example, if the Primary shard is number 26 hosted by `container02`, find the `AsynchronousReplica` partition 26, and make note of its container number. In this example, this is `container03`.
- 2. Log in to the node that is hosting the primary shard. In the example, this is `WNode02`. Use the same login user ID as was used to start the container JVMs (for example, `wasadmin`).
- 3. Run a `ps -ef | grep java` command to list all of the processes that are running Java.
- 4. In the list displayed, find the Java process running the container that holds the primary partition (`container02` in this example), and record its process ID (PID) number.
- 5. Issue a `kill -9 pid` command, where `pid` is the process ID of the primary partition. This is used to simulate an unexpected hard failure of that container JVM.
- 6. Issue another `xscmd showMapSizes` command. Notice that the shard that was originally holding the `AsynchronousReplica` shard has been “promoted” and is now the primary shard for that partition number. Furthermore, a fresh `AsynchronousReplica` has been made on another available container (for example, on `container04`).
- 7. Refresh your browser and check whether the counter increments are running normally. The WebSphere eXtreme Scale session manager running on behalf of the `test.jsp` has automatically located the correct HTTP session in the new primary partition. The browser user sees no impact whatsoever.

6.5 Scenario 2: Implementing with WebSphere Liberty Profile

HTTP session distribution can also be accomplished by using the WebSphere Liberty Profile. To use remote grid HTTP session storage, the `eXtremeScale_webApp-1.1` feature can be used. This section demonstrates how to accomplish HTTP session distribution by using WebSphere eXtreme Scale and the WebSphere Liberty Profile.

6.5.1 Implementing the grid configuration

For this example, the sample grid topology differs from the topology that was introduced in 5.3.5, “Introducing the sample topology” on page 232. In this topology, only two hosts are used (rather than four) and a second client server instance, `app02`, is added along with a new HTTP server instance. The new HTTP server instance is used to distribute HTTP requests across the two client server instances.

The new topology is shown in Figure 6-22.

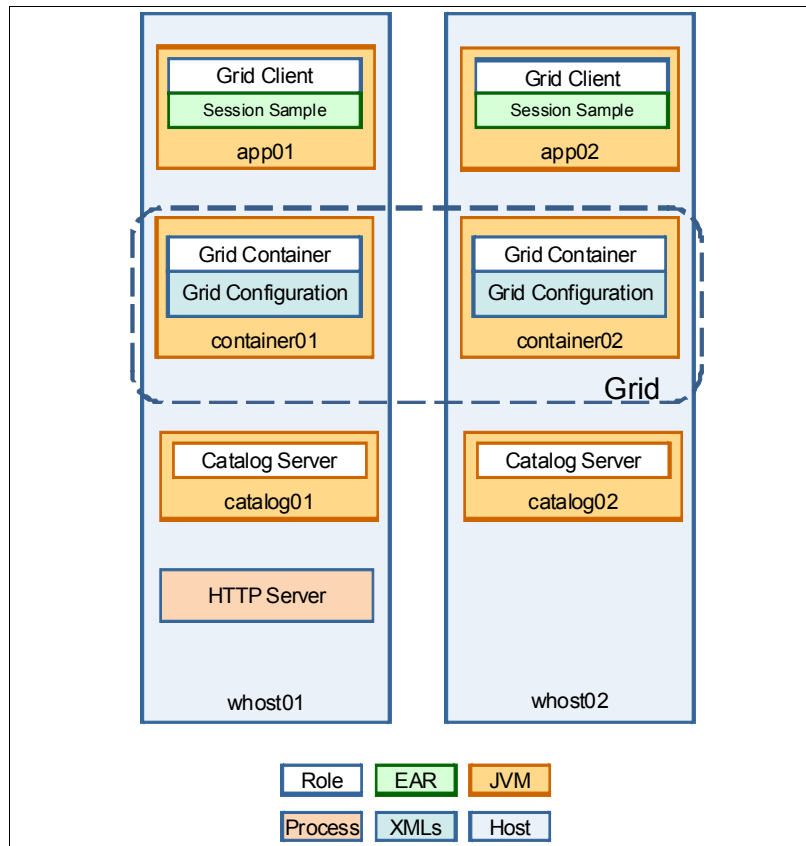


Figure 6-22 Liberty server topology for HTTP session distribution example

6.5.2 Configuring HTTP session management

Two main operations must be addressed to enable HTTP session distribution using the WebSphere Liberty Profile:

- ▶ Configuring the WebSphere Liberty Profile server instances
- ▶ Instrumenting the HTTP session sample application with a special ServletFilter that accommodates session management through WebSphere eXtreme Scale

To configure the example grid topology, complete the following steps:

1. Create all required server instances:
 - Run the following commands on the first host:


```
<wlp_install_root>/bin/server create app01
<wlp_install_root>/bin/server create catalog01
<wlp_install_root>/bin/server create container01
```
 - Run the following commands on the second host:


```
<wlp_install_root>/bin/server create app02
<wlp_install_root>/bin/server create catalog02
<wlp_install_root>/bin/server create container02
```

This process creates all of the required WebSphere Liberty Profile instances for this topology.

2. Create the `objectgrid.xml` and `objectgridDeployment.xml` files in the appropriate directory for each container server. Example 6-8 shows the `objectgrid.xml` file.

Example 6-8 objectgrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="session" txTimeout="30">
      <bean id="ObjectGridEventListener"
className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
      <backingMap name="objectgridSessionMetadata"
pluginCollectionRef="objectgridSessionMetadata" readOnly="false" lockStrategy="PESSIMISTIC"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600" copyMode="COPY_TO_BYTES"/>
      <backingMap name="objectgridSessionAttribute.*" template="true" readOnly="false"
lockStrategy="PESSIMISTIC" ttlEvictorType="NONE" copyMode="COPY_TO_BYTES"/>
      <backingMap name="objectgridSessionTTL.*" template="true" readOnly="false"
lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600"
copyMode="COPY_TO_BYTES"/>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="objectgridSessionMetadata">
      <bean id="MapEventListener"
className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Example 6-9 shows the `objectgridDeployment.xml` file

Example 6-9 objectgridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
../deploymentPolicy.xsd"
xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="session">
    <mapSet name="sessionMapSet" numberOfPartitions="11" minSyncReplicas="0"
maxSyncReplicas="0" maxAsyncReplicas="1" developmentMode="false"
placementStrategy="FIXED_PARTITIONS">
      <map ref="objectgridSessionMetadata"/>
      <map ref="objectgridSessionAttribute.*"/>
      <map ref="objectgridSessionTTL.*"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

3. Create a directory called `grids` in the root of each container server, and place the `objectgrid.xml` and `objectgridDeployment.xml` files in this directory:

```
<wlp_install_root>/usr/servers/container01/grids
<wlp_install_root>/usr/servers/container02/grids
```

4. Open the `server.xml` file in the root of each container server instance and configure it as shown in Example 6-10. Substitute your host name and ports as needed. Note the bold areas of the configuration in Example 6-10:
 - **eXtremeScale.webGrid-1.1**: This feature is required to support the HTTP session activity on the grid container.
 - **serverName**: This is the name of the container server. It differs on each container server instance in the topology. In this example, `container01` and `container02` are valid.
 - **catalogServiceBootstrap**: These values represent the host and port combinations for the two catalog servers in the sample topology.

Configuration for the session grid container servers is complete.

Example 6-10 Sample container server configuration

```
<server description="new server">
<!-- Enable features -->
  <featureManager>
    <feature>eXtremeScale.webGrid-1.1</feature>
  </featureManager>
  <xsServer serverName="container01" transport="XI0"
catalogServiceBootstrap="whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809"/>
</server>
```

5. Open the `server.xml` file in the root of each client server instance (`app01` and `app02`) and configure it as shown in Example 6-11. Substitute your host name, port, and cookie domain values.

Example 6-11 Adding eXtreme Scale client features to the Liberty server instances

```
<server description="App01">
  <!-- Enable features -->
  <featureManager>
    <feature>servlet-3.0</feature>
    <feature>jsp-2.2</feature>
    <feature>eXtremeScale.webapp-1.1</feature>
  </featureManager>
  <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" httpsPort="9443" />
  <httpSession cookieDomain=".rtp.ibm.com" idReuse="true" />
  <xsWebApp objectGridName="session"
catalogHostPort="whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809" securityEnabled="false"
replicationInterval="0" reuseSessionId="true" cookieDomain=".rtp.ibm.com" />
</server>
```

The bold elements at the bottom of Example 6-11 describe the configuration values required to integrate with the remote grid instance:

- **eXtremeScale.webapp-1.1**: This is the name of the feature that enables the local Liberty server instance to use HTTP session distribution with WebSphere eXtreme Scale.
- **idReuse**: This value instructs the Liberty server instance to use the session ID sent from the browser to preserve session data across web applications.
- **objectGridName**: This is the name of the ObjectGrid instance that is used to store the HTTP session data.

- **catalogHostPort:** These are the host and port combinations that are used by the WebSphere eXtreme Scale client code for connecting to the remote grid and obtaining an ObjectGrid reference.
 - **securityEnabled:** This setting enables WebSphere eXtreme Scale client security.
 - **cookieDomain:** Use the same name across all hosts if you want sessions to be available across different hosts
 - **replicationInterval:** An integer value (in seconds) that defines the time between writing of updated sessions to the grid.
6. After you modify both client server instances, copy the `SessionTest.ear` sample application file into the `dropins` folder of each client server instance.

Sample application: The sample application that is described in this section, `SessionTest.ear`, is provided in the `LIBERTY` folder of `HTTPSession.zip` available as additional material. For more information about downloading this material, see Appendix A, “Additional material” on page 341.

7. When the server is started, the web application is deployed automatically. Configuration of the client server instances and the deployment of the sample application are completed.
8. Open the `server.xml` file in the root of each catalog server instance and configure it as shown in Example 6-12. Substitute your host name and port values.

Example 6-12 Sample configuration for the catalog server instances

```
<server description="new server">
<!-- Enable features -->
  <featureManager>
    <feature>eXtremeScale.server-1.1</feature>
  </featureManager>
<xsServer serverName="catalog01" isCatalog="true" transport="XIO"
listenerHost="whost01.rtp.ibm.com" listenerPort="2809"
catalogClusterEndpoints="catalog01:whost01.rtp.ibm.com:6600:6601,catalog02:whost02.rtp.ibm.com:6600:6601"/>
</server>
```

Note the bold configuration elements of Example 6-12:

- **eXtremeScale.server-1.1:** This is the basic WebSphere eXtreme Scale enabling feature to enable a Liberty Profile server to function as a grid server.
- **serverName:** This is the name of the grid server process. It is general practice to match this value with the Liberty server name for simplicity.
- **isCatalog:** This value enables the Liberty server to function as a grid catalog server instance.
- **transport:** This field can be set to ORB or XIO, enabling either technology for grid communication purposes.

Configuration of the catalog servers in the sample topology is completed.

After you complete the steps in this section, you have replicated the sample topology. The next action to take is to start the environment and validate that it works as expected.

6.5.3 Running the application

To run the sample application and verify that HTTP sessions are distributed by using the WebSphere eXtreme Scale grid, start the runtime environment. This section covers the steps to run this procedure. The basic overview of the steps are:

1. Start the catalog server instances.
2. Start the container server instances.
3. Start the client server instances.

Tip: All of the catalog server instances that are defined in the `catalogClusterEndpoints` parameter must be started within a reasonable amount of time. By default, if all defined servers are not available within five minutes, the catalog server cluster stops.

The commands to start the entire environment are listed in Example 6-13. After all servers report that they are running, you are ready to test session failover and distribution.

Example 6-13 Starting the sample topology servers

```
<<On whost01>>
user@host1:cd <wlp_install_root>/usr/servers

user@whost01:../../bin/server start catalog01
Starting server catalog01.
Server catalog01 started with process ID 9043.

<<On whost02>>
user@whost02:../../bin/server start catalog02
Starting server catalog02.
Server catalog02 started with process ID 4522.

user@whost02:../../bin/server start container02
Starting server container02.
Server container02 started with process ID 4621.

user@whost02:../../bin/server start app02
Starting server app02.
Server app02 started with process ID 5100.

<<On whost01>>
user@whost01:../../bin/server start container01
Starting server container01.
Server container01 started with process ID 12991.

user@whost01:../../bin/server start app01
Starting server app01.
Server app01 started with process ID 30686.
```

6.5.4 Validating HTTP session failover

To test the system, complete the following steps:

1. Open a web browser and navigate to the sample application running on app01 by using the following URL:

`http://whost01.rtp.ibm.com:9080/SessionTestWeb/test.jsp`

In Figure 6-23, note that the port value of 19080 was used rather than port 9080 as documented in the previous steps. This was done to eliminate port conflicts with the WebSphere Application Server Network Deployment installation present on whost01 and whost02.



Host information	
Hostname	whost01.rtp.ibm.com
IP address	9.42.171.46
Port	19080
Application server (WAS and Liberty only)	SMF WebContainer

Session information	
Session ID	8zTpIT2D89yMdkrbp6KVeNx
Counter:	0

Figure 6-23 Initial view of SessionTestWeb output

2. Note the Session ID value. This session ID was issued from the session manager in the WebSphere Liberty Profile run time. Note that the count is 0. Refresh the page several times and verify that the counter is incremented as shown in Figure 6-24.



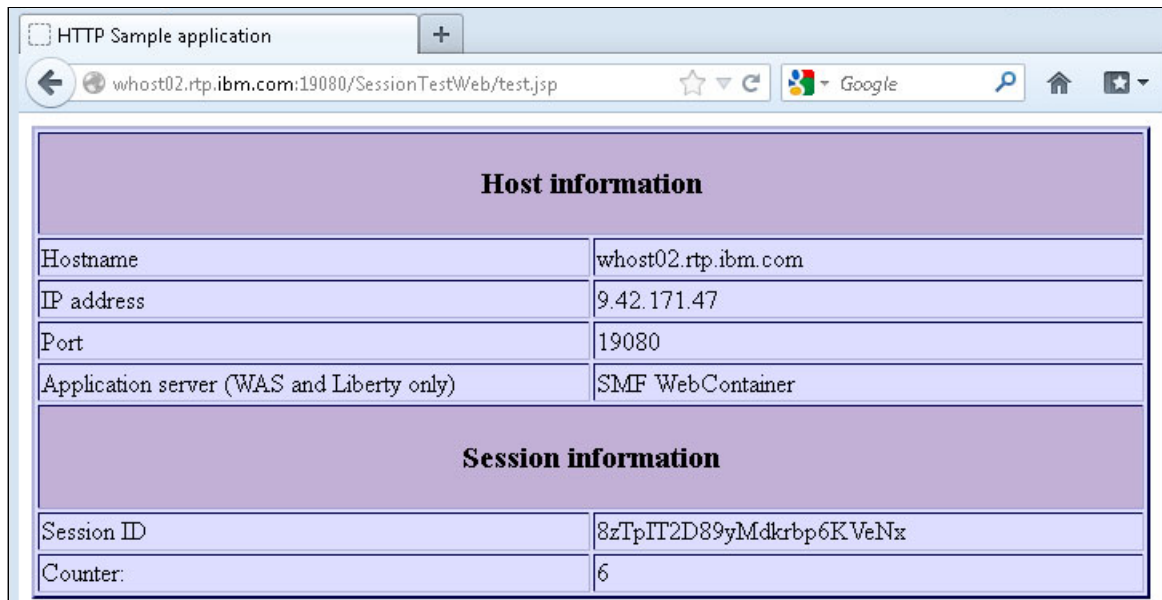
Host information	
Hostname	whost01.rtp.ibm.com
IP address	9.42.171.46
Port	19080
Application server (WAS and Liberty only)	SMF WebContainer

Session information	
Session ID	8zTpIT2D89yMdkrbp6KVeNx
Counter:	5

Figure 6-24 SessionTestWeb output after refreshing several times

3. To verify that the HTTP session data is being stored in the grid, navigate to the sample application running on app02 by using the following URL:
<http://whost02.rtp.ibm.com:9080/SessionTestWeb/test.jsp>

Even though you have switched to the app02 server instance, the data that are contained in the HTTP session are still available because of the eXtreme Scale integration. See Figure 6-25.



The screenshot shows a web browser window with the title "HTTP Sample application". The address bar displays "whost02.rtp.ibm.com:19080/SessionTestWeb/test.jsp". The main content area is divided into two sections: "Host information" and "Session information".

Host information	
Hostname	whost02.rtp.ibm.com
IP address	9.42.171.47
Port	19080
Application server (WAS and Liberty only)	SMF WebContainer

Session information	
Session ID	8zTpIT2D89yMdkrbp6KVeNx
Counter:	6

Figure 6-25 Session data is maintained even when switching between app01 and app02

To further verify session distribution with WebSphere eXtreme Scale, you can configure an HTTP server to load balance requests to the Liberty server instances. For more information about for generating a WebSphere HTTP plug-in configuration file, see the *Configuring a web server plug-in for the Liberty profile* topic of the WebSphere Application Server, Network Deployment, Version 8.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.wlp.nd.multiplatform.doc%2Fae%2Ftwlp_admin_webserver_plugin.html

6.6 Scenario 3: Implementing with Apache Tomcat

This section describes how to use WebSphere eXtreme Scale to efficiently manage HTTP sessions that are generated by web applications running on Apache Tomcat. This scenario illustrates a remote topology that uses the WebSphere eXtreme Scale client installation on a non-WebSphere application server environment.

Note: The sample topology and configuration illustrated in this section can be easily applied to any Java Platform, Enterprise Edition compliant application server. Supported application servers include Oracle WebLogic and JBoss Application Server.

6.6.1 Operational model

The basic operational model of the sample topology of implementing WebSphere eXtreme Scale with Apache Tomcat is shown in Figure 6-26 on page 289.

More specifically, the sample topology is deployed on two hosts (whost01 and whost02) and has the following characteristics:

- ▶ One Apache HTTP server instance is deployed on whost01.
- ▶ The Apache Connector Module (mod_JK) is configured within the Apache HTTP server to forward the requests for dynamic web resources to Apache Tomcat instances.
- ▶ Two Apache Tomcat instances (tomcat01 and tomcat02) are deployed on both hosts. The SessionTest web application that is described in 6.3, “Introducing the SessionTest sample application” on page 257 is deployed on both Apache Tomcat instances.
- ▶ The catalog service is supported by two catalog servers (catalog01 and catalog02) deployed on both hosts.
- ▶ The grid is operating within two container servers (container01 and container02) deployed on both hosts.

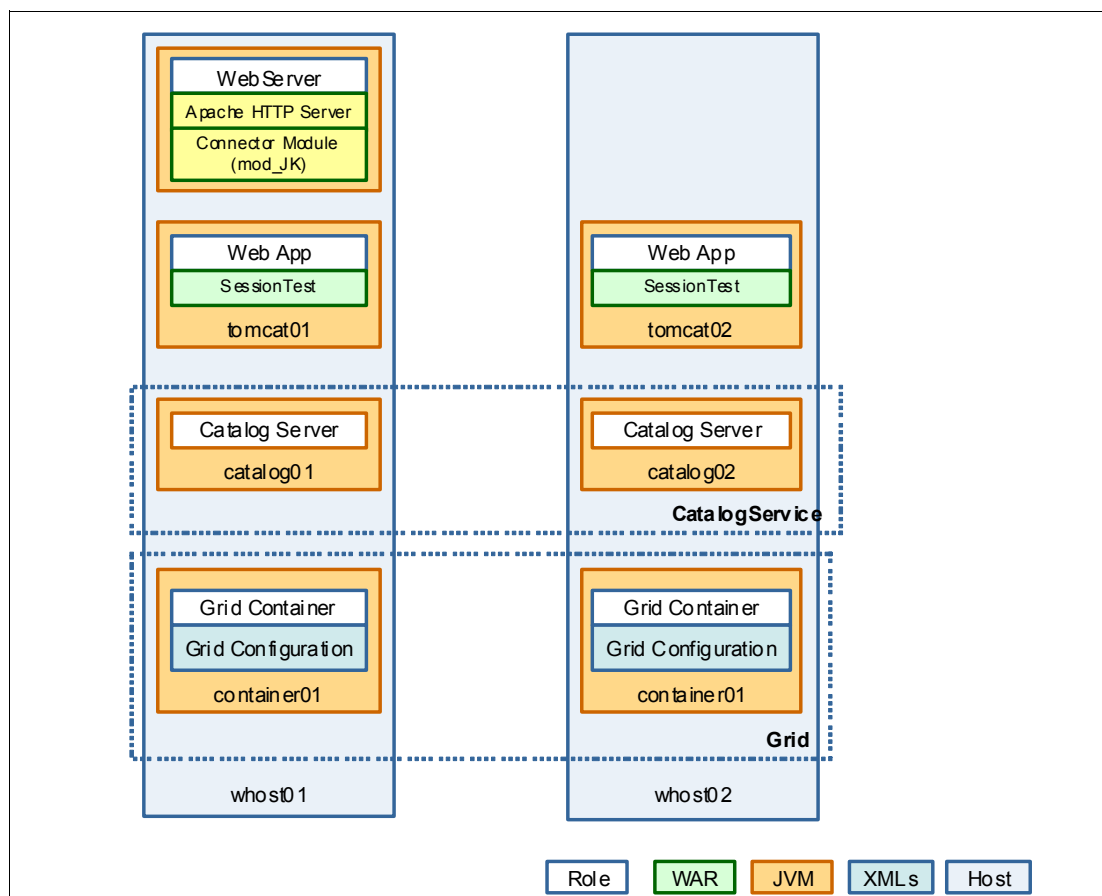


Figure 6-26 Sample topology for WebSphere eXtreme Scale for Apache Tomcat

6.6.2 Implementing the web application server deployment environment

This section describes the required steps to create the sample deployment environment based on Apache Web Server and Apache Tomcat.

Installing the required software components

The installation process consists of the following high-level steps to be completed on whost01 and whost02:

1. Download and install the latest Apache HTTP server release on whost01. The product package and documentation are available on the following website:

<http://httpd.apache.org/>

The sample topology that is used in this section is based on Apache Web Server 2.2.2 deployed on Linux x86-64.

2. Download and install the latest Apache Tomcat release on whost01 and whost02. The product package and related documentation is available at the following website:

<http://tomcat.apache.org/index.html>

The sample topology that is used in this section is based on Apache Tomcat 7.0.40 deployed on Linux x86-64.

3. Download and install the latest Apache Tomcat Connector module (mod_JK) on whost01. The module package and related documentation is available at the following website:

<http://tomcat.apache.org/connectors-doc/>

The sample topology that is used in this section is based on Apache Tomcat Connector (mod_JK) version 1.2.37.

Configuring Apache Tomcat

The default configuration is ready to use for applications that are directly served by Apache Tomcat without the intermediation of a web server. The Apache Tomcat Connector module communicates with Apache Tomcat by using the Apache JServ Protocol (AJP). This protocol is enabled by specifying the directive shown in Example 6-14 within the <TOMCAT_install_root>/config/server.xml configuration file.

Example 6-14 Enabling AJP protocol

```
Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

Incoming requests for dynamic web resources are forwarded to Apache Tomcat instances by applying a session affinity policy based on the same principles described in “Session cookie-based affinity” on page 248.

To support the session affinity policy, configure each Apache Tomcat instance to incorporate a unique identifier within the JSESSIONID cookie. This identifier tells the Apache Connector module to which Apache Tomcat instance to forward incoming requests.

This unique identifier is specified as the `jvmRoute` attribute of the Engine configuration within <tomcat_install_root>/config/server.xml as shown in Example 6-15.

Example 6-15 jvmRoute configuration for Apache Tomcat instance on whost01

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat01">
```

Example 6-16 shows the identifier on whost02.

Example 6-16 jvmRoute configuration for Apache Tomcat instance on whost02

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat02">
```

Configuring Apache Connector module

Apache Connector module configuration is based on two main entities:

► `mod_jk.conf`

This file contains module-specific configuration options. The default configuration options are appropriate for most common deployments. However, check the configured value for the following directives:

– `JkMount`

This directive contains a list of Uniform Resource Identifiers (URIs) of dynamic web resources to be forwarded to Apache Tomcat instances, which are configured as *workers* within the `mod_jk` configuration.

In the sample topology, `JkMount` directive is configured as shown in Example 6-17. In this configuration, `/SessionTestWeb*` is the URI comprising all web resources included within the `SessionTest` application and `loadbalancer` is a top-level worker that represents the two Apache Tomcat instances.

Example 6-17 JkMount directive

```
JkMount /SessionTestWeb* loadbalancer
```

– `JkWorkersFile`

This directive specifies the configuration file that contains information about the Apache Tomcat instances serving dynamic web resources, as shown in the Example 6-18.

Example 6-18 JkWorkersFile directive

```
JkWorkersFile /etc/apache2/workers.properties
```

► `workers.properties`

The `workers.properties` configuration file contains information about Apache Tomcat instances (*workers*), and related hosts and ports. Example 6-19 shows the `workers.properties` configuration file that is used to dispatch incoming requests to `tomcat01` and `tomcat02` instances.

Example 6-19 workers.properties file

```
# Load-balancing configuration
worker.list=loadbalancer,status
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=tomcat01,tomcat02
worker.loadbalancer.sticky_session=1

# Apache Tomcat instance on whost01
worker.tomcat01.host=whost01.rtp.ibm.com
worker.tomcat01.port=8009
worker.tomcat01.type=ajp13
worker.tomcat01.lbfactor=1

# Apache Tomcat instance on whost02
worker.tomcat02.host=whost02.rtp.ibm.com
worker.tomcat02.port=8009
worker.tomcat02.type=ajp13
worker.tomcat02.lbfactor=1
```

```
# Status worker for managing load balancer
worker.status.type=status
```

The following are some relevant information about the directives that are set within `workers.properties` file:

- The top-level workers are `loadbalancer` and `status`. The `loadbalancer` worker name must match the value of the `JkMount` directive in the `mod_jk.conf` configuration file.
- The top-level `loadbalancer` worker distributes incoming requests to two workers that are called `tomcat01` and `tomcat02` by applying a session affinity policy (`worker.loadbalancer.sticky_session=1`).
- Each worker is configured to communicate with a corresponding Apache Tomcat instance.
 - The `worker.<worker_name>` must match the `jvmRoute` identifier that is specified in the `<TOMCAT_install_root>/config/server.xml` configuration file as shown in Example 6-15 on page 290 and Example 6-16 on page 290.
 - AJP configuration is specified by setting the `worker.<worker_name>.host` and `worker.<worker_name>.port` directives.
- Incoming requests are balanced across the available Apache Tomcat instances (`worker.loadbalancer.balance_workers=tomcat01,tomcat02`). Both instances have the same weight (`worker.<worker_name>.lbfactor=1`).

For more information about how to configure this module, see the Apache Tomcat Connector - Webserver HowTo documentation at:

http://tomcat.apache.org/connectors-doc/webserver_howto/apache.html

6.6.3 Implementing the grid topology

The HTTP session grid is implemented as a stand-alone topology, similar to the one described in 5.1, “WebSphere eXtreme Scale in a stand-alone environment” on page 152.

Figure 6-27 shows more details about the sample grid topology.

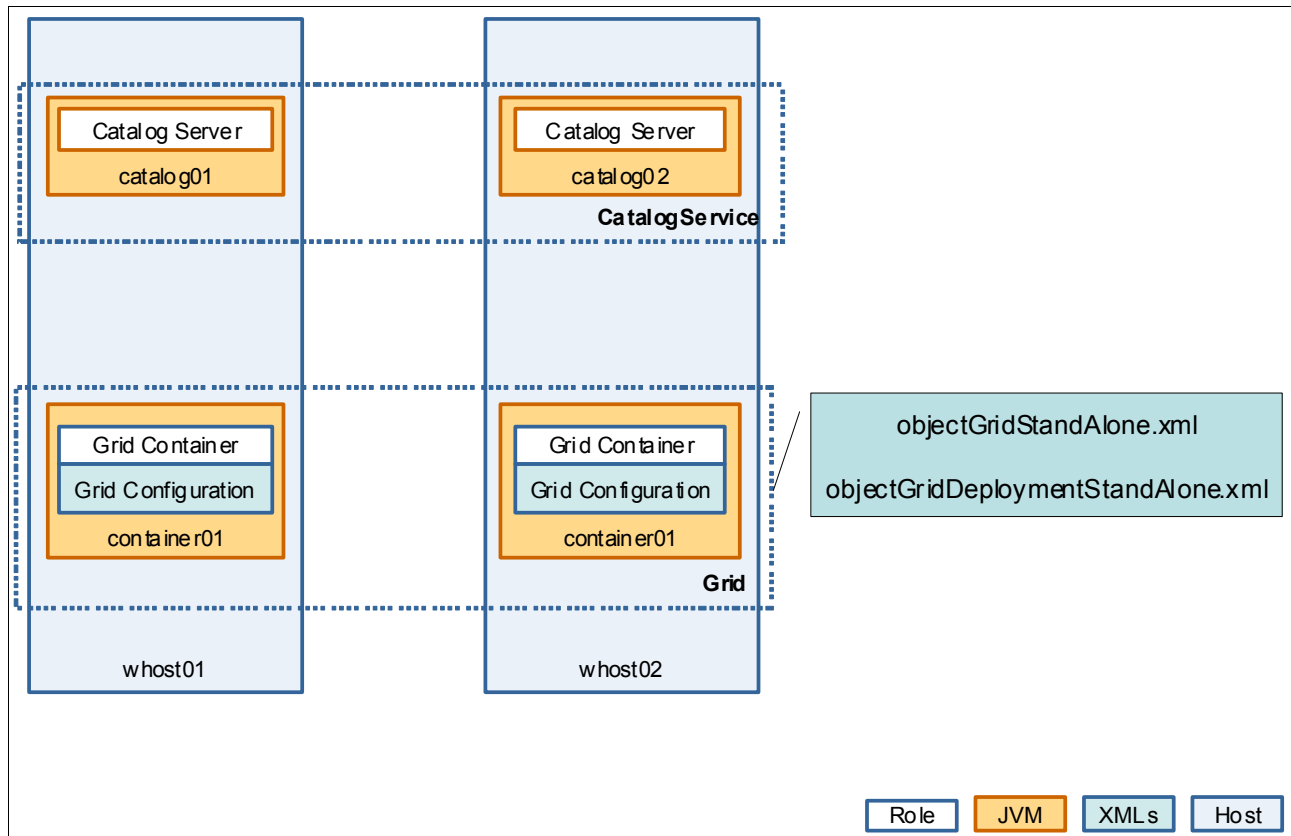


Figure 6-27 Sample topology for the HTTP session grid

The grid configuration is based on the sample configurations for HTTP session grids that are provided in `<WXS_install_root>/ObjectGrid/session/samples`. For the remote topology, the grid configuration uses `objectGridStandAlone.xml` and `objectGridDeploymentStandAlone.xml`, as described in 6.2.5, “Sample grid configuration files for an HTTP session” on page 253.

6.6.4 Configuring HTTP session management

Two main operations must be addressed to enable HTTP session management:

- ▶ Instrumenting the HTTP session sample application with a special grid servlet filter that accommodates session management through WebSphere eXtreme Scale.
- ▶ Configuring the Apache Tomcat classpath libraries.

Instrumenting the sample application with the grid servlet filter

Instrument the `SessionTestWeb` application to use the WebSphere eXtreme Scale filter for session management as described in 6.2.2, “Integration of WebSphere eXtreme Scale session management” on page 251.

Sample application: The sample application that is described in this section, `SessionTestWeb.war`, is provided in the `TOMCAT` folder of `HTTPSession.zip`, which is available as additional material. For information about downloading this material, see Appendix A, “Additional material” on page 341.

WebSphere eXtreme Scale provides a command-line script that is called `addObjectgridSessionFilter` that creates the appropriate servlet filter declarations within the web deployment descriptor.

The script is in the `<WXS_install_root>/ObjectGrid/session/bin` folder, and can be run by providing the parameter options shown in Example 6-20.

Example 6-20 addObjectgridSessionFilter options

```
addObjectgridSessionFilter.sh[.bat] [with the following options]
  <EAR_or_WAR_file>
  <splicer_properties_file>
```

The grid servlet filter configuration is provided through the `splicer.properties` file that is described in 6.2.6, “HTTP session servlet filter configuration” on page 256.

Example 6-21 shows the `splicer.properties` file that is used to configure WebSphere eXtreme Scale session filter for the sample topology. Note that `catalogHost` is configured with the catalog service endpoints of `catalog01` and `catalog02` servers.

Example 6-21 splicer.properties file

```
objectGridType=REMOTE
objectGridName=session
catalogHostPort=whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809
replicationInterval=10
sessionTableSize=1000
fragmentedSession=true
securityEnabled = false
```

You can now configure the WebSphere eXtreme Scale session filter for the `SessionTestWeb` application by running the `addObjectgridSessionFilter` command as shown in Example 6-22.

Example 6-22 Adding the grid servlet filter to SessionTestWeb application

```
/opt/IBM/WebSphere/XS86/ObjectGrid/session/bin/addObjectGridFilter.sh
SessionTestWeb.war splicer.properties
```

```
CWWSM0023I: Reading properties file: splicer.properties
CWWSM0021I: Reading archive: SessionTestWeb.war
CWWSM0027I: Processing .war file: SessionTestWeb
CWWSM0028I: Context parameters are:
CWWSM0029I: Context name: securityEnabled Value: false
CWWSM0029I: Context name: fragmentedSession Value: true
CWWSM0029I: Context name: sessionTableSize Value: 1000
CWWSM0029I: Context name: objectGridType Value: REMOTE
CWWSM0029I: Context name: replicationInterval Value: 10
CWWSM0029I: Context name: catalogHostPort Value:
whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809
CWWSM0029I: Context name: objectGridName Value: session
CWWSM0030I: Application splicing completed successfully.
```

Configuring Apache Tomcat classpath libraries

At run time, the WebSphere eXtreme Scale session filter operates as a grid client to interact with the grid to insert, retrieve, update, and delete HTTP session data.

Copy the following JAR files into the Apache Tomcat server library path in the <TOMCAT_install_root>/lib directory:

- ▶ **sessionobjectgrid.jar:** This JAR file contains the WebSphere eXtreme Scale session filter implementation and is available at <WXS_install_root>/session/lib/sessionobjectgrid.jar.
- ▶ **ogclient.jar:** This JAR file contains the WebSphere eXtreme Scale client implementation, and is available at <WXS_install_root>/ObjectGrid/lib/ogclient.jar.

Example 6-23 provides a listing for the Apache Tomcat classpath library.

Example 6-23 Apache Tomcat server classpath library

```
ls -ltr /opt/apache-tomcat-7.0.40/lib/
total 35244
-rw-r--r-- 1 wasadmin wasadmin 23173 May 5 08:55 tomcat-util.jar
-rw-r--r-- 1 wasadmin wasadmin 123986 May 5 08:55 tomcat-jdbc.jar
-rw-r--r-- 1 wasadmin wasadmin 51677 May 5 08:55 tomcat-i18n-ja.jar
-rw-r--r-- 1 wasadmin wasadmin 48692 May 5 08:55 tomcat-i18n-fr.jar
-rw-r--r-- 1 wasadmin wasadmin 77363 May 5 08:55 tomcat-i18n-es.jar
-rw-r--r-- 1 wasadmin wasadmin 235410 May 5 08:55 tomcat-dbcp.jar
-rw-r--r-- 1 wasadmin wasadmin 795027 May 5 08:55 tomcat-coyote.jar
-rw-r--r-- 1 wasadmin wasadmin 6872 May 5 08:55 tomcat-api.jar
-rw-r--r-- 1 wasadmin wasadmin 177599 May 5 08:55 servlet-api.jar
-rw-r--r-- 1 wasadmin wasadmin 88690 May 5 08:55 jsp-api.jar
-rw-r--r-- 1 wasadmin wasadmin 599413 May 5 08:55 jasper.jar
-rw-r--r-- 1 wasadmin wasadmin 123240 May 5 08:55 jasper-el.jar
-rw-r--r-- 1 wasadmin wasadmin 46085 May 5 08:55 el-api.jar
-rw-r--r-- 1 wasadmin wasadmin 1801636 May 5 08:55 ecj-4.2.2.jar
-rw-r--r-- 1 wasadmin wasadmin 255181 May 5 08:55 catalina-tribes.jar
-rw-r--r-- 1 wasadmin wasadmin 1564343 May 5 08:55 catalina.jar
-rw-r--r-- 1 wasadmin wasadmin 133468 May 5 08:55 catalina-ha.jar
-rw-r--r-- 1 wasadmin wasadmin 54175 May 5 08:55 catalina-ant.jar
-rw-r--r-- 1 wasadmin wasadmin 15264 May 5 08:55 annotations-api.jar
-rw-rw-r-- 1 wasadmin wasadmin 29269199 May 26 07:50 ogclient.jar
-rw-rw-r-- 1 wasadmin wasadmin 558515 May 26 08:02 sessionobjectgrid.jar
```

6.6.5 Deploying the sample application into Apache Tomcat

After you configure HTTP session management, you can deploy the SessionTest application within both Apache Tomcat instances. This task can be accomplished by copying the instrumented application SessionTestWeb.war into the <TOMCAT_install_root>/webapps folder of each Apache Tomcat instance on whost01 and whost02. The content of webapp folder looks like that in Example 6-24.

When the Apache Tomcat servers are started, the web application is deployed automatically.

Example 6-24 Apache Tomcat webapp folder

```
ls -latr /opt/apache-tomcat-7.0.40/webapps/
total 32
drwxr-xr-x 3 wasadmin wasadmin 4096 May 5 08:55 ROOT
drwxr-xr-x 5 wasadmin wasadmin 4096 May 5 08:55 manager
drwxr-xr-x 5 wasadmin wasadmin 4096 May 5 08:55 host-manager
drwxr-xr-x 6 wasadmin wasadmin 4096 May 5 08:55 examples
```

6.6.6 Starting the deployment environment

You can now start all of the software components that comprise the deployment environments. The basic steps involved are:

1. Start the grid infrastructure.
2. Start the web application server infrastructure.

Starting the grid infrastructure

This section addresses the steps that are required to start the grid infrastructure.

Starting catalog servers

Start the catalog servers on the hosts whost01 and whost02 by completing the following steps:

1. Start catalog01 on whost01 by issuing the command shown in the Example 6-25.

Example 6-25 Starting the catalog01 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh catalog01 -catalogServiceEndPoints catalog01:whost01.rtp.ibm.com:6600:6601,catalog02:whost02.rtp.ibm.com:6600:6601 -listenerHost whost01.rtp.ibm.com -listenerPort 2809 -JMXServicePort 1099
```

2. Start catalog02 on whost02 by issuing the command shown in Example 6-26.

Example 6-26 Starting the catalog02 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh catalog02 -catalogServiceEndPoints catalog01:whost01.rtp.ibm.com:6600:6601,catalog02:whost02.rtp.ibm.com:6600:6601 -listenerHost whost02.rtp.ibm.com -listenerPort 2809 -JMXServicePort 1099
```

Starting all the catalog servers in parallel: Start the catalog servers in parallel to ensure that internal partitioning information is available with at least one replica. When only a single cluster member is started, this rule cannot be established and the catalog server will not start.

Starting container servers

You can start the container servers on whost01 and whost02 by completing the following steps:

1. Start container01 on the host whost01 by issuing the command shown in the Example 6-27.

Example 6-27 Starting the container01 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh container01 -catalogServiceEndPoints whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809 -objectgridFile <WXS_install_root>/ObjectGrid/session/samples/objectGridStandAlone.xml
```



```
-deploymentPolicyFile
<WXS_install_root>/ObjectGrid/session/samples/objectGridDeploymentStandAlone.xml
```

2. Start container02 on the host whost02 by issuing the command shown in the Example 6-28.

Example 6-28 Starting the container02 server

```
<WXS_install_root>/ObjectGrid/bin/startXsServer.sh container02
-catalogServiceEndPoints whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809
-objectgridFile
<WXS_install_root>/ObjectGrid/session/samples/objectGridStandAlone.xml
-deploymentPolicyFile
<WXS_install_root>/ObjectGrid/session/samples/objectGridDeploymentStandAlone.xml
```

Validating the grid topology

The `xscmd -c showMapSizes` command can be used to validate the operating status for the grid. Example 6-29 shows the output of this command.

Example 6-29 showMapSizes command output

```
<WXS_install_root>/ObjectGrid/bin/xscmd.sh -cep whost01.rtp.ibm.com:2809,whost02.rtp.ibm.com:2809 -c
showMapSizes
```

```
*** Displaying results for session data grid and sessionMapSet map set.
```

```
*** Listing maps for container01 ***
```

Map Name	Partition	Map	Entries	Used Bytes	Shard Type	Container
objectgridSessionAttributeEvicted	0	0	0	0	AsynchronousReplica	container01_C-0
objectgridSessionAttributeEvicted	1	0	0	0	Primary	container01_C-0
objectgridSessionAttributeEvicted	2	0	0	0	AsynchronousReplica	container01_C-0
....						
objectgridSessionAttributeEvicted	46	0	0	0	AsynchronousReplica	container01_C-0
objectgridSessionMetadata	0	0	0	0	AsynchronousReplica	container01_C-0
objectgridSessionMetadata	1	0	0	0	Primary	container01_C-0
objectgridSessionMetadata	2	0	0	0	AsynchronousReplica	container01_C-0
...						
objectgridSessionMetadata	46	0	0	0	AsynchronousReplica	container01_C-0

Server total: 0 (0 B)

```
*** Listing maps for container02 ***
```

Map Name	Partition	Map	Entries	Used Bytes	Shard Type	Container
objectgridSessionAttributeEvicted	0	0	0	0	Primary	container02_C-0
objectgridSessionAttributeEvicted	1	0	0	0	AsynchronousReplica	container02_C-0
objectgridSessionAttributeEvicted	2	0	0	0	Primary	container02_C-0
...						
objectgridSessionAttributeEvicted	46	0	0	0	Primary	container02_C-0
objectgridSessionMetadata	0	0	0	0	Primary	container02_C-0
objectgridSessionMetadata	1	0	0	0	AsynchronousReplica	container02_C-0
objectgridSessionMetadata	2	0	0	0	Primary	container02_C-0
...						
objectgridSessionMetadata	46	0	0	0	Primary	container02_C-0

Server total: 0 (0 B)

Total catalog service domain count: 0 (0 B)

(The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.)

Starting the web application server infrastructure

This section covers the steps that are required to start the web application server infrastructure.

Starting the Apache HTTP server

You can start the Apache HTTP server on whost01 by using the Apache HTTP server control Interface to issue the `apachectl start` command. The log file `error.log` contains information about Apache Connector module (`mod_jk`) loaded during HTTP server startup as shown in Example 6-30.

Example 6-30 Apache HTTP server error.log file

```
cat /var/log/apache2/error.log
[Wed May 29 05:05:48 2013] [notice] Apache/2.2.22 (Ubuntu) mod_jk/1.2.37
configured -- resuming normal operations
```

Note: On some operating systems and Linux distributions, the Apache HTTP server is registered as a managed operating system service. In such case, start the Apache HTTP server using the operating system directory, such as `service apache2 start` on Linux.

Starting the Apache Tomcat server

You can start the Apache HTTP server on whost01 and whost02 by using the startup script in the `<TOMCAT_install_root>/bin` folder as shown in Example 6-31.

Example 6-31 Apache Tomcat start

```
<TOMCAT_install_root>/bin/startup.sh
Using CATALINA_BASE:   /opt/apache-tomcat-7.0.40
Using CATALINA_HOME:   /opt/apache-tomcat-7.0.40
Using CATALINA_TMPDIR: /opt/apache-tomcat-7.0.40/temp
Using JRE_HOME:        /usr/lib/jvm/java-7-openjdk-amd64
Using CLASSPATH:
/opt/apache-tomcat-7.0.40/bin/bootstrap.jar:/opt/apache-tomcat-7.0.40/bin/tomcat-j
uli.jar
```

Apache Tomcat start information is logged in the `<TOMCAT_install_root>/logs/catalina.out` file. Example 6-32 contains a portion of the `catalina.out` log file that shows the WebSphere eXtreme Scale client initialization during `SessionTestWeb` application startup.

Example 6-32 WebSphere eXtreme Scale start-up information in catalina.out log

```
May 29, 2013 6:16:27 AM org.apache.catalina.startup.HostConfig deployWAR
INFO: Deploying web application archive /opt/apache-tomcat-7.0.40/webapps/SessionTestWeb.war
May 29, 2013 6:16:30 AM com.ibm.ws.xs.size.JvmMemoryUtils
INFO: MEMORYSTATS_DEFAULT_SIZING_IN_USE_CWOBJ4542
May 29, 2013 6:16:30 AM com.ibm.ws.objectgrid.runtime.RuntimeInfo
INFO: INTERNAL_OBJECTGRID_VERSION_CWOBJ0903
[5/29/13 6:16:30:856 EDT] 0000000c RuntimeInfo I CWOBJ0903I: The internal version of
WebSphere eXtreme Scale is v7.0.0 (8.6.0.0) [cf11305.31182928].
May 29, 2013 6:16:30 AM com.ibm.ws.objectgrid.WXSProperties
INFO: WXS_PROPERTY_CWOBJ0054
[5/29/13 6:16:30:867 EDT] 0000000c WXSProperties I CWOBJ0054I: The value of the
"com.ibm.websphere.objectgrid.container.reconnect.block.reconnect.time" property is "30000".
May 29, 2013 6:16:30 AM com.ibm.ws.objectgrid.WXSProperties
INFO: WXS_PROPERTY_CWOBJ0054
```

```
[5/29/13 6:16:30:870 EDT] 0000000c WXSProperties I CWOBJ0054I: The value of the
"com.ibm.websphere.objectgrid.container.reconnect.min.successful.heartbeats" property is "10".
May 29, 2013 6:16:30 AM com.ibm.ws.objectgrid.WXSProperties
INFO: WXS_PROPERTY_CWOBJ0054
[5/29/13 6:16:30:872 EDT] 0000000c WXSProperties I CWOBJ0054I: The value of the
"com.ibm.websphere.objectgrid.container.reconnect.restart" property is "true".
May 29, 2013 6:16:30 AM com.ibm.ws.objectgrid.WXSProperties
INFO: WXS_PROPERTY_CWOBJ0054
[5/29/13 6:16:30:876 EDT] 0000000c WXSProperties I CWOBJ0054I: The value of the
"com.ibm.websphere.objectgrid.container.reconnect.restart.delay" property is "2000".
May 29, 2013 6:16:30 AM com.ibm.ws.objectgrid.WXSProperties
INFO: WXS_PROPERTY_CWOBJ0054
[5/29/13 6:16:30:880 EDT] 0000000c WXSProperties I CWOBJ0054I: The value of the
"com.ibm.websphere.objectgrid.container.reconnect.restart.parent.timeout" property is "180000".
May 29, 2013 6:16:30 AM com.ibm.ws.objectgrid.WXSProperties
INFO: WXS_PROPERTY_CWOBJ0054
[5/29/13 6:16:30:883 EDT] 0000000c WXSProperties I CWOBJ0054I: The value of the
"com.ibm.websphere.objectgrid.container.reconnect.retry.forever" property is "false".
May 29, 2013 6:16:31 AM com.ibm.ws.xs.sessionmanager.HttpSessionFilter
AUDIT: session.objectgrid
[5/29/13 6:16:31:020 EDT] 0000000c HttpSessionFi Z CWWSM0007I: Using the ObjectGrid based
SessionManager.
```

6.6.7 Testing the application and session failover

After you complete the configuration steps and start up the deployment environment, test the SessionTest application to validate session management behavior. The test case is described in 6.4.1, “Installing the sample SessionTest application” on page 261. To run the test, complete the following steps:

1. Open a web browser, and navigate to the sample application by using the following URL:

<http://whost01.rtp.ibm.com/SessionTestWeb/test.jsp>

The main page of SessionTest web application is displayed as shown in Figure 6-28 on page 300.

Note the value for the Hostname, Session ID, and Counter fields. The incoming browser request from Apache HTTP server has been dispatched to Apache Tomcat instance running on whost01.rtp.ibm.com (hostName) and has been served by the tomcat01 instance (jvmRoute is appended to sessionID value). The counter value is 0 because this is the first request issued by the browser.



Host information	
Hostname	whost01.rtp.ibm.com
IP address	172.16.1.201
Port	80
Application server (WAS and Liberty only)	null

Session information	
Session ID	EBFFE99313A7DD27C2CF4937847E9A51.tomcat01
Counter:	0

Figure 6-28 Initial view of the SessionTestWeb output

2. Refresh the web browser page several times. The main page is updated as shown in Figure 6-29.

Note that the Apache HTTP server routes the incoming request to the same “affinity” Apache Tomcat server, in this case, tomcat01. The counter value is incremented according to the number of page refreshes.



Host information	
Hostname	whost01.rtp.ibm.com
IP address	172.16.1.201
Port	80
Application server (WAS and Liberty only)	null

Session information	
Session ID	EBFFE99313A7DD27C2CF4937847E9A51.tomcat01
Counter:	5

Figure 6-29 SessionTestWeb after refreshing several times

3. To verify that the HTTP session data is being stored and there is transparent session failover, complete the following steps:
 - a. Stop the Apache Tomcat instance that served previous requests, in this case, tomcat01, by issuing the shutdown script in <TOMCAT_install_root>/bin directory as shown in Example 6-33.

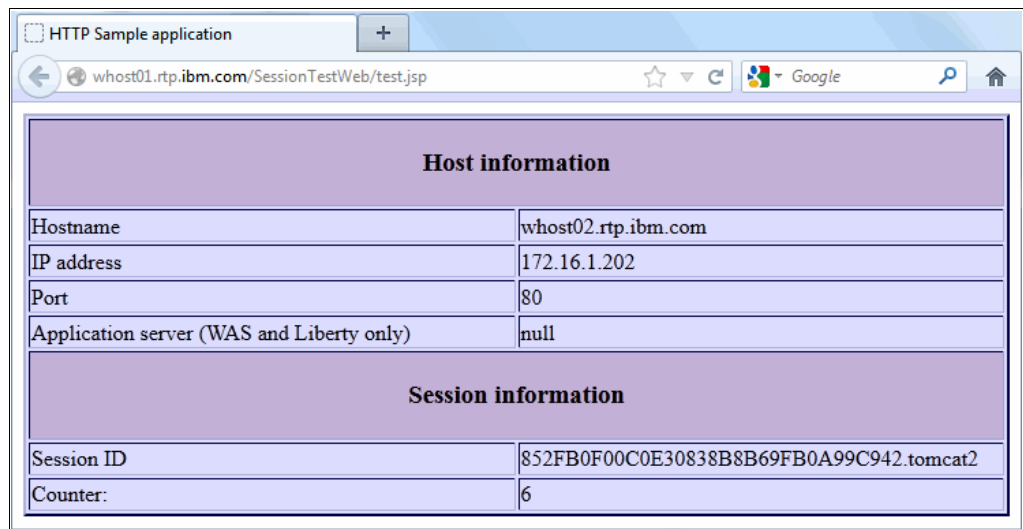
Example 6-33 Apache Tomcat instance shutdown

```
<TOMCAT_install_root>/bin/shutdown.sh
Using CATALINA_BASE: /opt/apache-tomcat-7.0.40
```

```
Using CATALINA_HOME: /opt/apache-tomcat-7.0.40
Using CATALINA_TMPDIR: /opt/apache-tomcat-7.0.40/temp
Using JRE_HOME: /usr/lib/jvm/java-7-openjdk-amd64
Using CLASSPATH:
/opt/apache-tomcat-7.0.40/bin/bootstrap.jar:/opt/apache-tomcat-7.0.40/bin/to
mcat-juli.jar
```

- b. Refresh your browser again to invoke the test.jsp again, as shown in Figure 6-30.

The Apache Connector module first attempts to send the request to the same Apache Tomcat instance as originally (tomcat01). Upon the failure, the connector automatically tries the request to the other Apache Tomcat instance (tomcat02). Note that the incoming request has been forwarded to the tomcat02 instance as shown by the values of the Hostname and SessionID fields. The counter has been incremented properly to 6.



Host information	
Hostname	whost02.rtp.ibm.com
IP address	172.16.1.202
Port	80
Application server (WAS and Liberty only)	null

Session information	
Session ID	852FB0F00C0E30838B8B69FB0A99C942.tomcat2
Counter:	6

Figure 6-30 Session data is maintained even when switching between app01 and app02



Basic administration

This chapter introduces several ways to administer, monitor, and troubleshoot your WebSphere eXtreme Scale environment.

This chapter includes the following sections:

- ▶ 7.1, “The web console” on page 304
- ▶ 7.2, “Command line access: xscmd” on page 310
- ▶ 7.3, “Log analyzer” on page 324
- ▶ 7.4, “JVM logging and tracing” on page 326
- ▶ 7.5, “Monitoring with other products” on page 328
- ▶ 7.6, “Tuning, recovery and troubleshooting” on page 329

7.1 The web console

Basic monitoring and simple cache querying are made easy with the web console component included with WebSphere eXtreme Scale version 8.6. The web console provides many key performance indicators that include the following functions:

- ▶ Current data grid used capacity distribution
- ▶ Capacity usage over time
- ▶ Most active data grids
- ▶ Average throughput
- ▶ Average transaction time
- ▶ Used capacity versus number of cache entries
- ▶ Cache hit rates

The web console is ready to run and can be started with the following command:

```
wxs_install_root/ObjectGrid/bin/startConsole.sh/bat
```

If you are starting the web console for the first time, use these default credentials to access the console:

- ▶ User name: *admin*
- ▶ Password: *admin*

Figure 7-1 shows the home page of the web console.

The screenshot shows the 'eXtreme Scale Monitor' web console. The top navigation bar includes 'Home', 'Monitor', 'Management', and 'Settings'. The 'Get Started' section contains two cards: 'Register catalog servers' and 'Group catalog servers into catalog service domains'. The 'eXtreme Scale Overview' section is divided into three columns: 'Catalog server', 'ObjectGrid', and 'Map'. Each column provides a brief description and an icon. The 'Catalog server' column explains its role in controlling data storage and hosting a catalog service. The 'ObjectGrid' column describes it as an in-memory database for applications. The 'Map' column explains that maps store application data in key-value pairs and are grouped into map sets, which are further divided into partitions and shards.

Figure 7-1 WebSphere eXtreme Scale web console home page

7.1.1 Configuring the web console for monitoring

Before you use the web console, it must be configured to connect to the catalog servers for your grid deployment.

To configure your catalog servers, complete the following steps:

1. Click **Settings** → **Catalog Servers** from the home page. On the Catalog Servers page, click the **Plus** icon that is shown in Figure 7-2 to define a new catalog server.

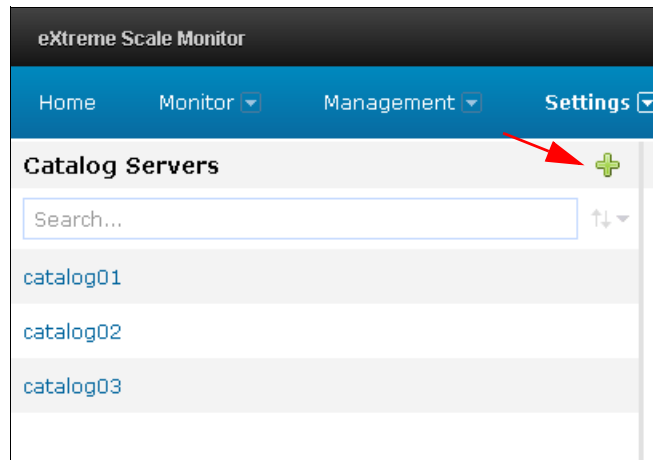


Figure 7-2 Adding a catalog server

2. On the Registering an existing Catalog Server page, define the following values for the new catalog server:
 - **Name:** This is an arbitrary name to use in the console. Generally, use the same server name that was given at server startup (for stand-alone deployment) or server definition (for a WebSphere Application Server-managed deployment).
 - **Host name:** The name of the host that the catalog server is running. Generally, use a fully qualified domain name here.
 - **Listener port:** The bootstrap port number that the catalog server is listening on. The default listener port is 2809.

Figure 7-3 shows a sample catalog server definition.

Registering an existing Catalog Server.

* Name:

* Host name:

* Listener port:

OK Cancel

Figure 7-3 Sample catalog server definition

- After you define your catalog servers, add them to a catalog service domain by clicking **Settings** → **Catalog Service Domains**. From the Catalog Service Domains page, define a new domain by clicking the **Plus** icon shown in Figure 7-4.

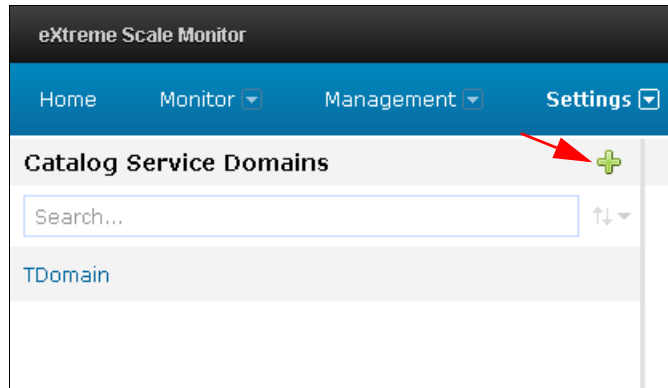


Figure 7-4 Adding a catalog service domain

- In the resulting dialog window, enter a name for your new domain and then click in the **Catalog servers** text box. The web console presents a list of defined catalog servers for you to add to the domain as shown in Figure 7-5.

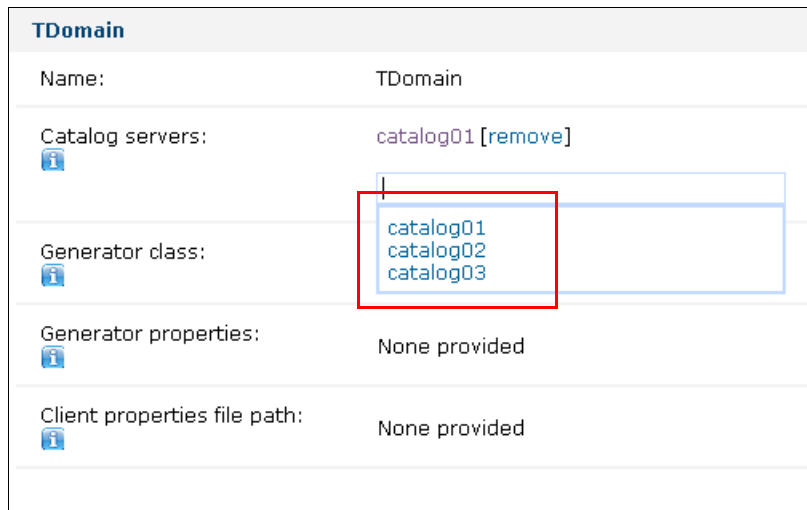


Figure 7-5 Adding catalog servers to the catalog service domain

- After you configure catalog servers and add them to a catalog service domain, select your active catalog service domain in the menu at the upper right of the page. You can verify a successful connection by looking for the “Connected” indicator to the right of the catalog service domain menu. Figure 7-6 shows the portion of the page where the connection status is shown.

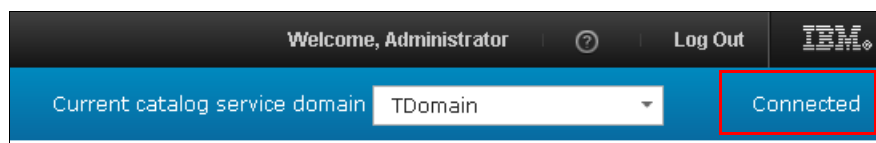


Figure 7-6 Successful connection to the catalog service domain

Now that the web console is configured and connected to a catalog service domain, you are ready to monitor your grid.

7.1.2 Monitoring your grid with the web console

The web console is a versatile monitoring tool with many uses. This section covers only a few key performance indicators.

Data grid overview

The Data Grid Overview page of the web console can be used to get a quick overview of key metrics for your grid deployment. A sample window is shown in Figure 7-7.

The Data Grid Overview page can also be used to review cache usage and average throughput values.

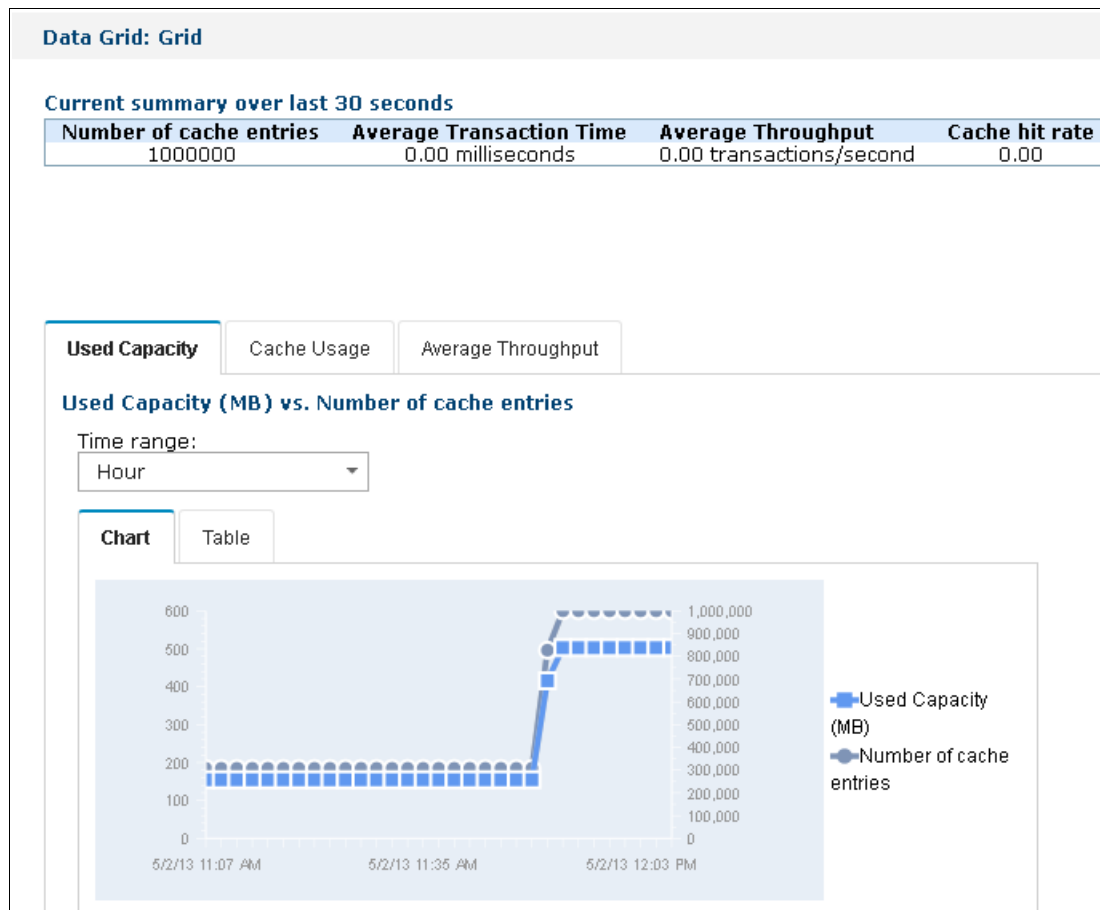


Figure 7-7 Sample data grid overview page output

Data grid details

You can review your grid with more detail by clicking **Monitor** → **Data Grid Details**. From here, you can review your grid-level and map-level details. Figure 7-8 shows sample output between map instances at the grid level.

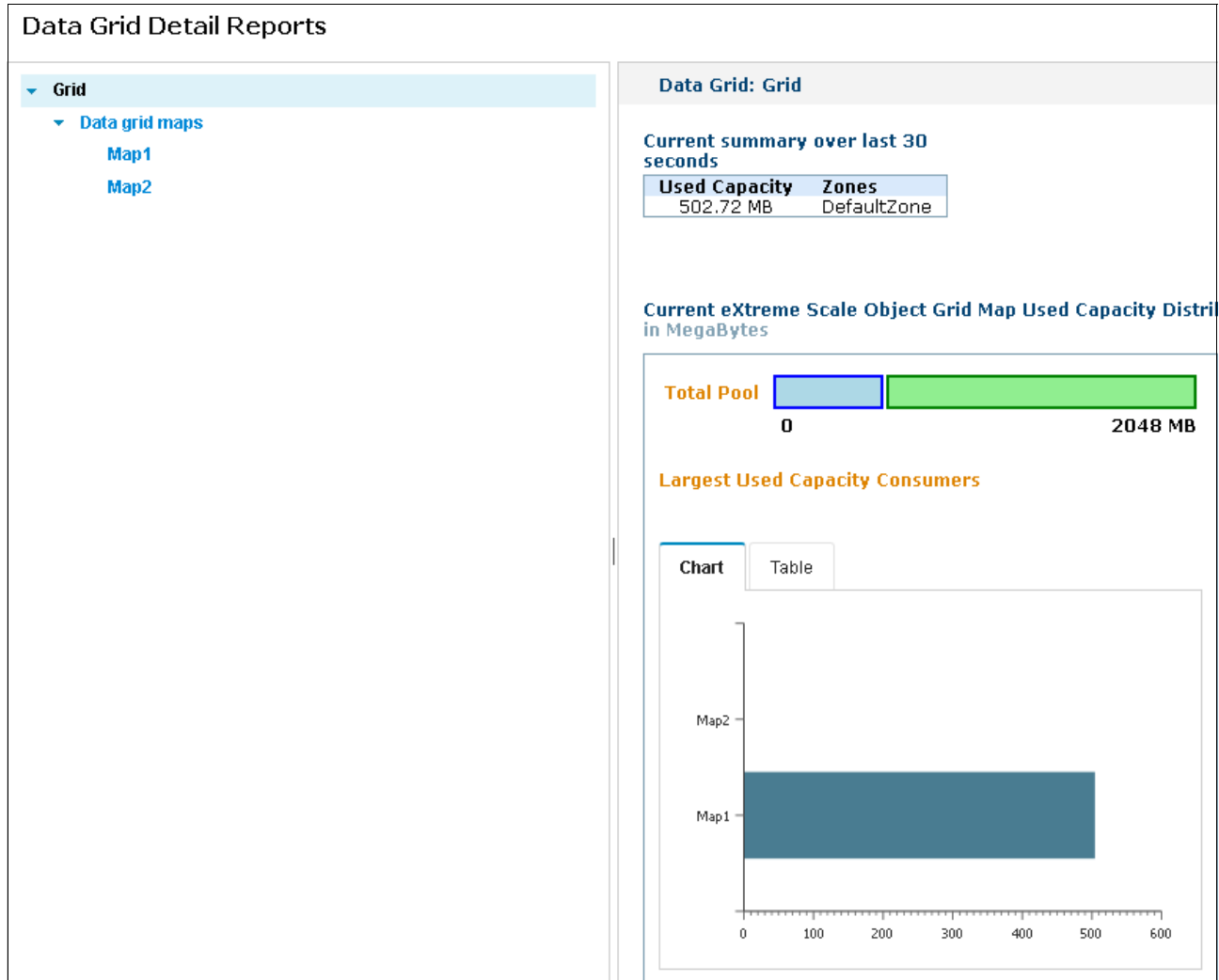


Figure 7-8 Data grid detail information showing capacity and data distribution between map instances at the grid level

Figure 7-9 shows sample output between partitions at the map level.

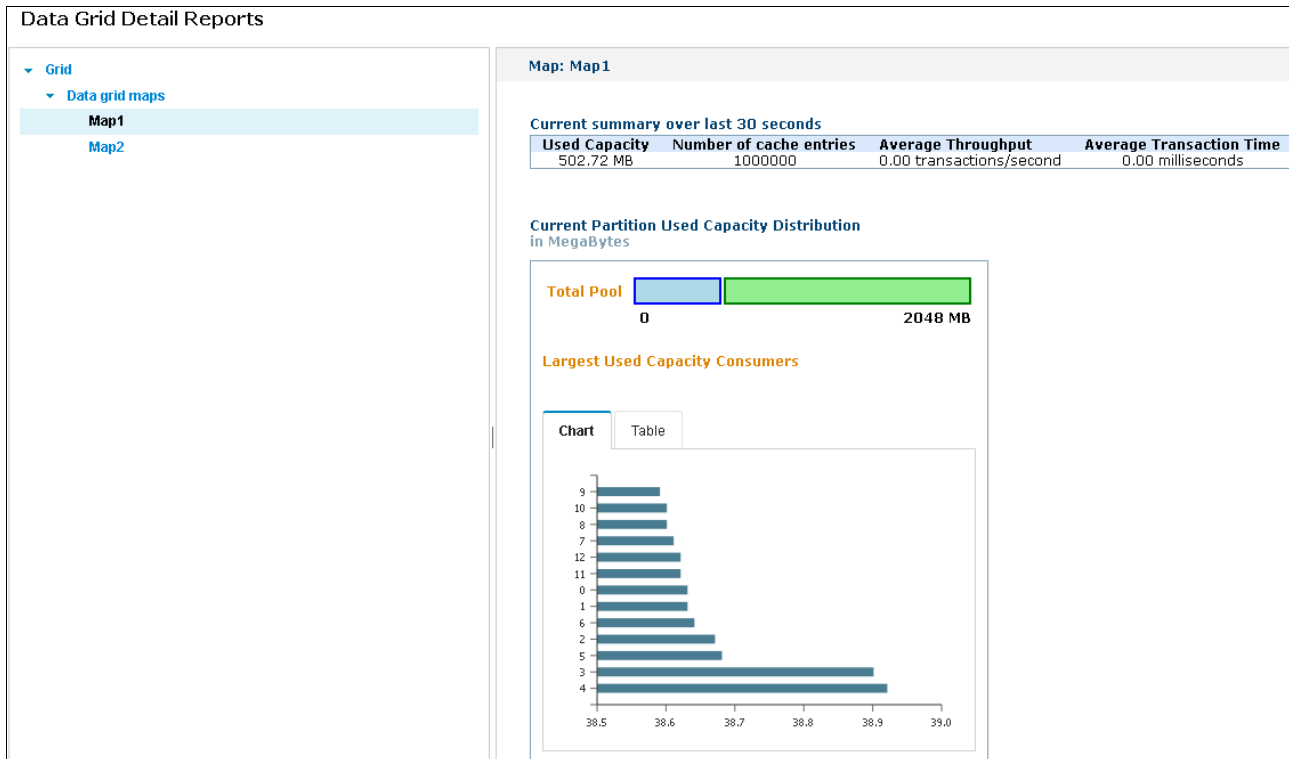


Figure 7-9 Data grid detail information showing capacity and data distribution between partitions at the map level

Data grid content inspection

If you want to review a particular object in the grid, for example to validate that a key actually exists in the grid or to verify a field, review the contents of the grid maps. Click **Management** → **Query Data Grid Contents**. From the Query Data Grid Contents page, select a map to query. The page displays controls to run various operations on the grid as shown in Figure 7-10.

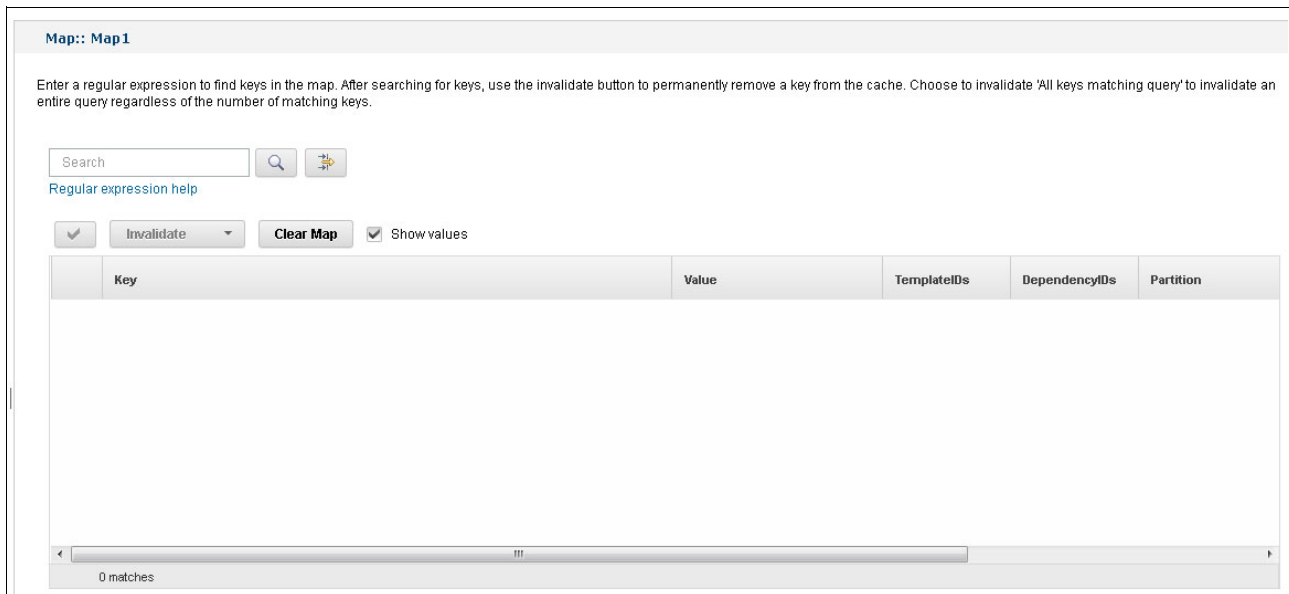


Figure 7-10 The map query interface

Using the Query Data Grid Contents page, you can query the grid for data using regular expressions. For example, you can use `Key.*` to look for all keys that start with `Key`. You can also invalidate individual or groups of data objects in the grid. Additionally, you can issue a **Clear Map** command that purges all the data from currently selected map instance.

For more information, see the *Regular expression syntax* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxsregsyntax.html>

7.1.3 Monitoring your grid with the Message Center

The web console also provides a feature called the *Message Center*. The Message Center page displays event messages from the catalog and container servers of the grid. Figure 7-11 shows an example of server start notifications.

ID	Type	Date	Source	Message
No filter applied				
5	Information	May 7, 2013 2:45:29 PM	cat1	CWOBJ8250I: Server started: container3
4	Information	May 7, 2013 2:45:28 PM	cat1	CWOBJ8252I: Core group membership changed: DefaultZoneC
3	Information	May 7, 2013 2:44:03 PM	cat1	CWOBJ8250I: Server started: container2
2	Information	May 7, 2013 2:44:02 PM	cat1	CWOBJ8252I: Core group membership changed: DefaultZoneC
1	Information	May 7, 2013 2:43:49 PM	cat1	CWOBJ8250I: Server started: container1
0	Information	May 7, 2013 2:43:48 PM	cat1	CWOBJ8252I: Core group membership changed: DefaultZoneC

Figure 7-11 Message Center in the web console

7.2 Command line access: xscmd

WebSphere eXtreme Scale provides a command line tool called `xscmd`. The `xscmd` tool allows the user to view and change the state of the catalog servers, container servers, and grid data.

xscmd versus xsadmin: Prior releases of WebSphere eXtreme Scale provided a sample command line tool called `xsadmin`. The commands in `xsadmin` are also in `xscmd`, although they use a slightly different syntax. A mapping of `xsadmin` to `xscmd` can be found in the *xsadmin tool to xscmd tool migration* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxsxsadmintoxscmd.html>

For more information about using `xscmd`, see the *Administering with the xscmd utility* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsxscmd.html>

Running xscmd: If your grid runs in a secured WebSphere Application Server environment, `xscmd` must be run from the profile directory. Running `xscmd` from the profile directory prompts the user for the correct WebSphere Application Server administrative security credentials.

Running `xscmd` in a stand-alone installation can result in errors when connecting to the catalog server, such as:

```
CWXSI0062E: Catalog service is unavailable at the
service:jmx:iiop://9.42.171.46/jndi/corbaname:iiop:9.42.171.46:9811/WsnAdminNameService#JMXConnector endpoint
```

To provide security parameters to `xscmd`, see the *Configuring security profiles for the xscmd utility* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsxscmdsec.html>

Following are the most frequently used commands that show the state of the grid:

- ▶ `showPlacement`
- ▶ `routetable`
- ▶ `showMapsizes`
- ▶ `revisions`
- ▶ `showInfo`

These commands show different aspects of the grid that include the catalog server perspective, the container side, and the size of the grid. They also provide a good starting point for tracking down problems within the grid.

Available commands: For a fast list of all the commands available in `xscmd`, run `xscmd -lc` to list the commands. For help on individual commands, use `-h` instead of `-c`. For example, to see the options available for the `showPlacement` command, run:

```
xscmd -h showPlacement
```

7.2.1 Placement layout: `showPlacement` command

The `showPlacement` command lists the catalog service placement plan. It is the layout of the primary and replica shards that are stored in the catalog server. The information that is

displayed by **showPlacement** is the last placement layout plan that was pushed out to the container servers.

The **showPlacement** command displays the containers with their shard assignments as demonstrated in Example 7-1.

Example 7-1 Output from xscmd command showPlacement

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c showPlacement
```

```
CWXSIO068I: Executing command: showPlacement
```

Command showPlacement is a technology preview. The command usage and output is subject to change.

*** Show all online container servers for Grid data grid and mapSet map set.

```
Host: mach01
```

```
Container: xsServer01_C-0, Server:xsServer01, Zone:DefaultZone
```

```
Partition Shard Type Reserved
```

```
-----
```

0	Primary	false
1	Primary	false
2	Primary	false
3	Primary	false

```
Number of containers matching = 1
```

```
Total known containers = 1
```

```
Total known hosts = 1
```

```
CWXSIO040I: The showPlacement command completed successfully.
```

If some shards are not yet placed, they are listed in the unassigned container. To view unassigned shards, add the shard filter option, **-sf** with a **U** for unassigned shards. Example 7-2 shows an example of this command.

Example 7-2 xscmd command showPlacement -sf U with unassigned shards

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c showPlacement -sf U
```

```
CWXSIO068I: Executing command: showPlacement
```

Command showPlacement is a technology preview. The command usage and output is subject to change.

*** Showing unassigned container servers for Grid data grid and mapSet map set.

```
Partition Shard Type
```

```
-----
```

0	SynchronousReplica
1	SynchronousReplica
2	SynchronousReplica
3	SynchronousReplica

```
CWXSIO040I: The showPlacement command completed successfully.
```

Shards remain on the unassigned container for several reasons:

- ▶ Waiting for more containers to start to meet placement constraints:
 - The `numInitialContainers` constraint was set in the deployment policy. Start more containers to match the `numInitialContainers`.
 - Replicas defined as `maxAsyncReplicas` or `maxSyncReplicas` in the deployment policy can only be placed when there are enough containers servers to host them. Start more container servers.
 - The `developmentMode` constraint in the deployment policy is set to `false`. Start containers on more hosts. Replicas cannot be placed on the same hosts as primary shards if `developmentMode` is set to `false`.
 - The `minSyncReplicas` is set to one or more. Placement occurs only if there are enough containers to host the primary shard plus the number of `numSyncReplicas` defined. Start more container servers.
 - Zones are defined. Placement occurs only if there are enough containers to host shards in the different zones.
 - Waiting for the `placementDeferralInterval` set in the catalog server properties file to expire. After the last server is started and the `placementDeferralInterval` expires, placement starts.
- ▶ Individual shards failed to place and the containers reported the failed shards to the catalog server. These shards can be placed again by using the **`triggerPlacement`** command.
- ▶ Balance is suspended with the **`suspendBalancing`** command. The **`resumeBalancing`** command initiates placement of unassigned shards.

The command **`placementServiceStatus`** can be used to review the current placement constraints on the grid.

There are several commands that can affect grid placement. If the output of **`showPlacement`** does not look balanced, the **`balanceShardTypes`** command can be used to move primary and replica shards.

These commands can affect placement and balancing:

- ▶ **`triggerPlacement`**
- ▶ **`suspendBalancing`**
- ▶ **`resumeBalancing`**
- ▶ **`balanceShardTypes`**
- ▶ **`reserveShard`**
- ▶ **`releaseShard`**

7.2.2 Current route table: **`routetable`** command

The **`routetable`** command shows both the route table that is stored in the catalog server and the result of pinging each shard entry in the route table status. The route table represents the last reported location of primary and replica shards. The **`showPlacement`** command provides the placement plan, and the route table provides a view of the results.

The **`routetable`** command output shows the master route table list that is stored on the catalog server. It lists the shard types, their host name, and the container server.

Example 7-3 demonstrates sample output from the **rutetable** command.

Example 7-3 Example of xscmd rutetable command output

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c rutetable

CWXSI0068I: Executing command: rutetable

*** Displaying routing information for data grid: Grid:mapSet

Placement Scope: Domain
Shard Type          Partition State    Host    Zone        Container
-----
Primary             0         reachable mach01 DefaultZone S02_C-0
SynchronousReplica 0         reachable mach01 DefaultZone S01_C-1
Primary             1         reachable mach01 DefaultZone S01_C-1
SynchronousReplica 1         reachable mach01 DefaultZone S02_C-0
Primary             2         reachable mach01 DefaultZone S02_C-0
SynchronousReplica 2         reachable mach01 DefaultZone S01_C-1
Primary             3         reachable mach01 DefaultZone S01_C-1
SynchronousReplica 3         reachable mach01 DefaultZone S02_C-0

CWXSI0040I: The rutetable command completed successfully.
```

Running the **rutetable** command provides the status of each shard. When **xscmd** receives the route table from the catalog service, **xscmd** pings all of the shards to determine whether they are online and in the correct state. The following states are listed by the **rutetable** command:

- ▶ Reachable
- ▶ Unreachable
- ▶ Invalid

The *invalid* state means that the shard can be reached, but the state that it reported does not match the state in the route table. For example, if the route table has a state of *AsynchronousReplica* for a shard, but the shard reports it is a *Primary*, the *invalid* state is displayed.

The states of *unreachable* or *invalid* are often temporary states. For example, a shard that is in the current route table as an *AsynchronousReplica*, but was promoted to be the *Primary* shard for the partition updates the catalog service with the new and correct route table. Running the **rutetable** command again displays that the shard is *reachable*.

Unreachable or *invalid* states in the **rutetable** command that persist indicate problems with fail over. If the catalog server was not able to relocate shards after a failover, the route table still points to previous shard locations. Compare the output of the **rutetable** command to the **showPlacement** command to check for disparity between the planned location of the shard and the last location reported in the route table.

The **triggerPlacement** command can also be used to resolve *unreachable* or *invalid* partitions.

The **rutetable** command can report unassigned shards. For more information about unassigned shards, see 7.2.1, “Placement layout: showPlacement command” on page 311.

Example 7-4 shows an example of the **routetable** command.

Example 7-4 Running the routetable command to list unassigned shards

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c routetable

CWXSI0068I: Executing command: routetable

*** Displaying routing information for data grid: Grid:mapSet

Placement Scope: Domain
Shard Type          Partition State      Host      Zone      Container
-----
Primary             0          reachable mach1     DefaultZone c1_C-1
SynchronousReplica 0          unassigned unassigned unassigned unassigned
Primary             1          reachable mach1     DefaultZone c1_C-1
SynchronousReplica 1          unassigned unassigned unassigned unassigned
Primary             2          reachable mach1     DefaultZone c1_C-1
SynchronousReplica 2          unassigned unassigned unassigned unassigned
Primary             3          reachable mach1     DefaultZone c1_C-1
SynchronousReplica 3          unassigned unassigned unassigned unassigned

CWXSI0040I: The routetable command completed successfully.

Ending at: 2013-04-26 15:32:14.945
```

7.2.3 Size of the grid: showMapSizes command

The **showMapSizes** command provides the number and size of objects in the ObjectGrid from a map view. For example, running the **showMapSizes** command without any filters displays the sizes of all of the maps for all of the user grids.

The **showMapsizes** command also calculates the total number and sizes of objects in the grid. The command is useful to determine whether primary shards and replica shards match in size.

Different options can be used on the **showMapSizes** command to give a smaller set of data. For example, instead of all maps, the command can be set to return a single map. The command can also filter by shard type. For example, the **showMapsizes** command can be run twice, where the first run captures only the primary shard totals and a second run captures only the asynchronous replica totals. The results can be compared to find any differences between the primary and replicas.

Example 7-5 demonstrates the output of the **showMapSizes** command.

Example 7-5 Example showMapSizes command output

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c showMapSizes

CWXSI0068I: Executing command: showMapSizes

*** Displaying results for Grid data grid and mapSet map set.

*** Listing maps for S01 ***
Map Name Partition Map Entries Used Bytes Shard Type      Container
-----
-----
```

```

Map1      0      1      264 B      SynchronousReplica S01_C-1
Map1      1      5      1 KB       Primary           S01_C-1
Map1      2      1      256 B      SynchronousReplica S01_C-1
Map1      3      0      0          Primary           S01_C-1
Server total: 7 (1 KB)

```

*** Listing maps for S02 ***

```

Map Name Partition Map Entries Used Bytes Shard Type      Container
-----
Map1      0      1      264 B      Primary         S02_C-0
Map1      1      5      1 KB       SynchronousReplica S02_C-0
Map1      2      1      256 B      Primary         S02_C-0
Map1      3      0      0          SynchronousReplica S02_C-0
Server total: 7 (1 KB)

```

Total catalog service domain count: 14 (3 KB)

(The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.)

CWXSI0040I: The showMapSizes command completed successfully.

Using the **showMapSizes** command is also a good tool to see the result of placement. Running **routeTable** and **showMapSizes** after placement events gives the user a picture of the grid state.

If maps are missing from **showMapSizes**, the maps are likely template maps. Template maps are only created when accessed by the user. When maps are created based on the template maps defined in the deployment policy file, the maps are displayed in the **showMapSizes** output.

7.2.4 Rerun placement: triggerPlacement command

The **triggerPlacement** command can be useful in several situations. The command asks the catalog service to push out a new placement plan that includes placing any shards that are not assigned. If the **showPlacement** command does not reveal any shards in an unassigned state, the catalog service pushes out the current placement plan without any changes. The placement work is processed by the containers. If the containers have no new work to do, they send a route table update.

Following are some examples of when to use the **triggerPlacement** command:

- ▶ There are some shards in the Unassigned section of **showPlacement** command output. The catalog server tries to place those shards. The shards can be unassigned if they failed to place during the first request.
- ▶ The route table is missing shard entries or has shards with an invalid state. During a **triggerPlacement** request, all of the primary shards resubmit their route table information, which refreshes the master copy on the catalog server.
- ▶ Grid placement is controlled by **numInitialContainers** in a deployment policy, but the number of containers listed cannot be started. The **triggerPlacement** command starts placement and overrides the **numInitialContainers** constraint.

The **triggerPlacement** command runs against the catalog service, and the catalog server pushes work to the container servers. When the **triggerPlacement** command returns, the

work that is pushed to the containers can still be ongoing. Use the **roustable** and **showMapsizes** commands to monitor the progress.

7.2.5 Version of grid contents: revisions command

The **revisions** command is often paired with the **showMapsizes** command. They both display information about the state of the shards in the grids. Although the **showMapsizes** command returns the number and size of the maps, the **revisions** command shows the version of the data in the shards. Whenever data is acted on in the grid, the revision changes. After a key is updated several times, the revision or version of the ObjectGrid is updated several times.

Example 7-6 shows an example of the **revisions** command output.

Example 7-6 Running revisions, only the partitions with data inserted are displayed in the output

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c revisions

CWXSI0068I: Executing command: revisions

*** Revisions for catalog service domain: DefaultDomain, Grid: Grid, MapSet: mapSet
Partition Type          Domain          Lifetime ID    Revision Lifetime
                        Owner
-----
0 AsynchronousReplica  DefaultDomain  1367005130001  1 c1
0 Primary              DefaultDomain  1367005130001  1 c1
1 AsynchronousReplica  DefaultDomain  1367005130001  2 c1
1 Primary              DefaultDomain  1367005130001  2 c1
9 AsynchronousReplica  DefaultDomain  1367005130002  1 c1
9 Primary              DefaultDomain  1367005130002  1 c1
10 AsynchronousReplica DefaultDomain  1367005130001  2 c1
10 Primary             DefaultDomain  1367005130001  2 c1
11 AsynchronousReplica DefaultDomain  1367005130001  1 c1
11 Primary             DefaultDomain  1367005130001  1 c1
12 AsynchronousReplica DefaultDomain  1367005130004  4 c1
12 Primary             DefaultDomain  1367005130004  4 c1

CWXSI0040I: The revisions command completed successfully.
```

After the primary for a partition moves to different locations, the revisions keep their history. For example, if a primary moves three times, it can have historical lifetime numbers from the primary shards' prior locations. This maintains the correct history of the data. It does not indicate rogue data from other containers.

Example 7-7 shows an example of the **revisions** command output after shards have moved. In this example, the primary shard for partition 0 moved from c1 to c2 and now has revisions with lifetimes from both locations.

Example 7-7 Running the revisions command after the shards have moved

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c revisions

CWXSI0068I: Executing command: revisions
*** Revisions for catalog service domain: DefaultDomain, Grid: Grid, MapSet:
mapSet
Partition Type          Domain          Lifetime ID    Revision Lifetime
                        Owner
-----
0 AsynchronousReplica  DefaultDomain  1367005130001  2 c1
```

```

0 AsynchronousReplica DefaultDomain 1367006630850      1 c2
0 Primary              DefaultDomain 1367005130001      2 c1
0 Primary              DefaultDomain 1367006630850      1 c2
1 AsynchronousReplica DefaultDomain 1367005130001      2 c1
1 Primary              DefaultDomain 1367005130001      3 c1
3 AsynchronousReplica DefaultDomain 1367005130003      1 c1
3 Primary              DefaultDomain 1367005130003      1 c1
5 AsynchronousReplica DefaultDomain 1367005130002      1 c1
5 Primary              DefaultDomain 1367005130002      1 c1
7 AsynchronousReplica DefaultDomain 1367005130003      1 c1
7 Primary              DefaultDomain 1367005130003      1 c1
9 AsynchronousReplica DefaultDomain 1367005130002      1 c1
9 Primary              DefaultDomain 1367005130002      1 c1
10 AsynchronousReplica DefaultDomain 1367005130001      2 c1
10 Primary              DefaultDomain 1367005130001      2 c1
11 AsynchronousReplica DefaultDomain 1367005130001      2 c1
11 Primary              DefaultDomain 1367005130001      2 c1
12 AsynchronousReplica DefaultDomain 1367005130004      8 c1
12 Primary              DefaultDomain 1367005130004      8 c1

```

CWXSIO040I: The revisions command completed successfully.

Using the revision check or **-rc** option with the **revisions** command tells xscmd to calculate the state of the replicas compared to the primary. For example, if the async replica has half of the primary's data, it reports the async replica is at 50%. The **-rc** option can be used to monitor replicas during grid loading or after a placement event where replicas are starting from scratch in new locations.

The **-rc** option is shown in Example 7-8. In this example, the replica shards match the primary shards.

Example 7-8 Running revisions -rc when the replica shards match the primary shards

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c revisions -rc
```

```

CWXSIO068I: Executing command: revisions
*** Revision check for catalog service domain: DefaultDomain, Grid: Grid, MapSet: mapSet
Partition Type Domain Lifetime ID Revision Lifetime Owner
-----

```

```
Grid replication: 100.000%
```

CWXSIO040I: The revisions command completed successfully.

Example 7-9 shows running the **revisions -rc** command when the asynchronous replica is being the primary.

Example 7-9 Running the revisions -rc command when the asynchronous replica is being the primary

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c revisions -rc
```

```

CWXSIO068I: Executing command: revisions
*** Revision check for catalog service domain: DefaultDomain, Grid: Grid, MapSet: mapSet
Partition Type Domain Lifetime ID Revision Lifetime Owner
-----

```

```

10 AsynchronousReplica DefaultDomain 1367005130001      1 c1
10 Primary              DefaultDomain 1367005130001      2 c1

```

Grid replication: 75.000%

CWXSIO040I: The revisions command completed successfully.

7.2.6 Information about servers: showInfo command

The **showInfo** command displays several important details about the catalog and container servers in a catalog service domain. The details include the WebSphere eXtreme Scale version, the installation location, the transport type, ports that are used, and the Java version used by the WebSphere eXtreme Scale installation.

The **showInfo** command is handy to ensure that all servers are running and accessible. It can be used to ensure that the same version is on every server. It can also be used to review the ports that are used in the catalog service domain.

The **showInfo** command returns information for all the servers by default. There are different filter options to request a subset of the servers. For more information, see the *Retrieving eXtreme Scale environment information with the xscmd utility* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsshowinfo.html>

7.2.7 Stopping servers efficiently: teardown command

The **teardown** command allows servers to stop in batches instead of selecting a single server to stop using the stop stopXsServer script.

Example 7-10 shows an example of running the **teardown** command on a catalog server.

Example 7-10 Running the teardown command on a catalog server

```
wxs_install_root\ObjectGrid\bin>xscmd -c teardown -sl cat1
```

```
CWXSIO068I: Executing command: teardown
***The following servers are being stopped:
  cat1
```

```
Do you want to tear down the listed servers? (Y/N)
```

```
y
```

```
Server Result    Message
-----
cat1    succeeded
```

```
CWXSIO040I: The teardown command completed successfully.
```

For more information, see the *Stopping servers gracefully with the xscmd utility* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txstardown.html>

7.2.8 Reviewing keys in the grid: findByKey command

The **findByKey** command shows keys in the grid and the partition that stores the key. The **findByKey** command requires an ObjectGrid name, map name, and a search term.

Example 7-11 shows an example of running the **findByKey** command to search for keys that start with key8.

Example 7-11 Running the findByKey command to search for keys that start with key8

```
wxs_install_root\ObjectGrid\bin>xscmd -c findByKey -g accounting -m payroll -fs key8.*  
Starting at: 2013-05-10 17:09:51.608
```

```
CWXSIO068I: Executing command: findByKey
```

```
11 matching keys were found.
```

```
Partition Key
```

```
-----  
0          key80  
0          key81  
0          key82  
0          key83  
0          key84  
0          key85  
0          key86  
0          key87  
0          key88  
0          key89  
0          key8
```

```
CWXSIO040I: The findByKey command completed successfully.
```

For more information about using the **findByKey** command, see the *Querying, displaying, and invalidating data* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsquerygrid.html>

7.2.9 Multi-master replication commands

There are several *multi-master replication* (MMR) commands to change and monitor the MMR linking. The MMR commands reflect the state of the grid for the catalog service domain that **xscmd** was run against. It is often useful to run the monitoring commands against both catalog service domains to compare the results.

Linking the grid: establishLink and dismissLink commands

The **establishLink** and **dismissLink** commands need to be run on only one side of a catalog service domain. The **dismissLink** command can be used on either catalog service domain to stop the link between domains.

When **establishLink** completes, the catalog servers completed the exchange of placement information to start shard linking. The linking process between the primary shards on the container servers can be ongoing. Use the **showLinkedPrimaryes** command to monitor the linking process.

Checking the links: showLinkedDomains and showLinkedPrimaries commands

The **showLinkedDomains** command can be used at any time to verify the linked foreign domains. Example 7-12 shows sample output from the **showLinkedDomains** command.

Example 7-12 Running showLinkedDomains command in a two domain topology

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c showLinkedDomains
```

```
CWXSIO068I: Executing command: showLinkedDomains
```

```
*** Retrieving foreign catalog service domains linked to the following catalog  
service domain: domain1
```

```
Foreign catalog service domain
```

```
-----
```

```
domain2
```

```
CWXSIO040I: The showLinkedDomains command completed successfully.
```

The **showLinkedPrimaries** commands lists and checks the state of all of the individual links between primary shards. Running the command without options lists all of the links. This is demonstrated in Example 7-13.

Example 7-13 Running the showLinkedPrimaries command

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c showLinkedPrimaries
```

```
CWXSIO068I: Executing command: showLinkedPrimaries
```

The output for command **showLinkedPrimaries** is a technology preview. The command output is subject to change.

```
*** Displaying results for Grid data grid and mapSet map set.
```

```
*** Listing Primary Shards for local domain: domain1,  
Container: c1_C-0, Server: c1, Host: hydra.rchland.ibm.com ***
```

```
Grid Name Map Set Name Partition Domain Container Status
```

```
-----  
Grid      mapSet      0          domain2 c2_C-0    online  
Grid      mapSet      1          domain2 c2_C-0    online  
Grid      mapSet      2          domain2 c2_C-0    online  
Grid      mapSet      3          domain2 c2_C-0    online
```

```
CWXSIO040I: The showLinkedPrimaries command completed successfully.
```

The **linkhealthcheck** or **-hc** option lists any primary shards that do not report the correct number of links or a healthy or online state for the links. A state of online indicates that the primary shard is able to replicate with the foreign primary. This is shown in Example 7-14.

Example 7-14 Running the showLinkedPrimaries -hc command with a healthy grid

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c showLinkedPrimaries -hc
```

```
CWXSIO068I: Executing command: showLinkedPrimaries
```

The output for command **showLinkedPrimaries** is a technology preview. The command output is subject to change.

CWXSIO091I: Verifying the primary shards have the correct number of links to foreign primary shards.

*** Displaying results for Grid data grid and mapSet map set. Expected number of online links: 1.

CWXSIO092I: All primary shards for Grid data grid and mapSet map set have the correct number of links to foreign primary shards.

CWXSIO040I: The showLinkedPrimaries command completed successfully.

A state of recovery, demonstrated in Example 7-15, indicates that the primary shard encountered an error while replicating with the foreign primary. A link in recovery state is tried on an interval until communication is established again or the link is removed.

Example 7-15 Running the showLinkedPrimaries -hc command on a grid encountering errors

```
wxs_install_root\ObjectGrid\bin>xscmd.bat -c showLinkedPrimaries -hc  
CWXSIO068I: Executing command: showLinkedPrimaries
```

The output for command showLinkedPrimaries is a technology preview. The command output is subject to change.

CWXSIO091I: Verifying the primary shards have the correct number of links to foreign primary shards.

*** Displaying results for Grid data grid and mapSet map set. Expected number of online links: 1.

*** Listing Primary Shards with the incorrect number of links for local domain:
domain1,Container: c1_C-0, Server: c1, Host: hydra.rchland.ibm.com ***

Grid	Map Set	Name	Partition	Domain	Container	Status
Grid	mapSet		0	domain2	c2_C-0	recovery
Grid	mapSet		1	domain2	c2_C-0	recovery
Grid	mapSet		2	domain2	c2_C-0	recovery
Grid	mapSet		3	domain2	c2_C-0	recovery

CWXSIO040I: The showLinkedPrimaries command completed successfully.

Usually a link that is marked as recovery experiences a temporary problem. The foreign primary might be moving or there might be a network problem between the domains.

Checking the replication state: showMapSizes, revisions, and showDomainReplicationState commands

The **showMapSizes** and **revisions** commands only run against a single catalog server domain. However, the output can be compared to check that the map sizes and revisions match in each domain. The output of each command must be compared based the ObjectGrid name, the map set name, and the partition number. With the **showMapSizes** command, the map names must also be matched.

The **showDomainReplicationState** command shows snapshots of the replication state for a local and foreign domain. Sample command output can be seen in Example 7-16. The data that are displayed in the command are collected at intervals and can be reviewed when the **showDomainReplicationState** command is used.

Example 7-16 Running the showDomainReplicationState command in a two domain topology

CWXSIO068I: Executing command: showDomainReplicationState

Command showDomainReplicationState is a technology preview. The command usage and output is subject to change.

```
Domain domain1
Container Outstanding Inbound Revisions Outstanding Outbound Revisions
-----
c1          0                               0
```

```
Domain domain2
Container Outstanding Inbound Revisions Outstanding Outbound Revisions
-----
c2          0                               0
```

CWXSIO040I: The showDomainReplicationState command completed successfully.

7.2.10 Viewing health notifications with xscmd

During run time, WebSphere eXtreme Scale gathers a set of notifications and log messages. It makes them available through a single access stream such as xscmd or the web console. By default, the health notifications include a set of general health messages and warning or error messages from the JVM logs of the catalog and container servers.

For more information about the messages in the web console, see 7.1.3, “Monitoring your grid with the Message Center” on page 310.

Using xscmd, you can view the last several gathered messages or change the notification type. In Example 7-17, the notification history includes a replication warning. To follow up on the warning message, the JVM log for server testDualPrimaryMinSync21 can be reviewed for more problems or replication recovery.

Example 7-17 Running the showNotificationHistory xscmd command

CWXSIO068I: Executing command: showNotificationHistory

Message ID	Severity	Server source	Message
-----	-----	-----	-----
...			
23	WARNING	testDualPrimaryMinSync21	CW0BJ1550W: The primary (accounting:mapSet1:0) shard received exceptions while replicating from the primary shard on the domain1: testDualPrimaryMinSync11_C-0 primary container. The primary shard contin
24	INFO	testDualPrimaryMinSync2Cat0	CW0BJ8252I: Core group

```
membership changed:
DefaultZoneCGO
25          INFO      testDualPrimaryMinSync2Cat0 CWOBJ8250I: Server started:
                                     testDualPrimaryMinSync20
```

CWXSIO040I: The showNotificationHistory command completed successfully.

For more information about viewing and changing the health notifications, see the *Viewing health notifications with the xscmd utility* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txshealthxscmd.html>

7.3 Log analyzer

WebSphere eXtreme Scale version 8.6 includes a command-line log analyzer tool that can be used to analyze current performance and status, and troubleshoot system problems. The log analyzer tool can perform the following tasks:

- ▶ Display the total number of failover messages seen in the grid over time.
- ▶ Display all critical grid messages, including the top five exceptions seen in the grid.
- ▶ Display how many times an exception occurred, and what server and container the exception was sent from.
- ▶ Display a diagram that displays the grid topology.
- ▶ Display ORB configuration settings between JVM instances to ensure consistent configuration values are maintained.
- ▶ Display a timeline diagram that shows all grid lifecycle events, exceptions, messages, and *first-failure data capture* (FFDC) events.

XIO: If you are using XIO as the grid transport, the **xsLogAnalyzer** tool does not display any ORB setting information on the Topology Consistency tab of the analysis output.

7.3.1 Running the log analyzer

The log analyzer tool can be found in the following directories:

- ▶ Stand-alone installation: *wxs_install_root*/ObjectGrid/bin
- ▶ WebSphere Application Server managed installation: *was_root*/bin

The tool executable file is called `xsLogAnalyzer`. An example is shown in Example 7-18.

Example 7-18 Running the log analyzer tool

```
xsLogAnalyzer.sh|bat -logsRoot c:\myxslogs -outDir c:\myxslogs\out
```

In Example 7-18 on page 324, you see several useful parameters are being specified:

► -logsRoot

Specifies the absolute path to the log directory to be analyzed. This is a required parameter. Copy the log and trace files to the computer from which you are planning to use the log analyzer tool.

► -outDir

Specifies an existing directory to which the log output is directed. Omitting this parameter results in the log analyzer output being set to the root location of the log analyzer tool.

A complete set of parameters that can be used with the **xsLogAnalyzer** command can be found in the *Running log analysis* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txslogvisconfig.html>

7.3.2 Extending the capabilities of the log analyzer

You can create customized log scanners for log analysis. Using regular expressions, you can extend the capabilities of the log analyzer to capture virtually any event message of interest. For more information about how to implement a custom scanner, see the *Running log analysis* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txslogvisconfig.html>

7.3.3 Sample output from xsLogAnalyzer

Figure 7-12 depicts sample output from the xsLogAnalyzer tool.

Log Analysis Report Time Range Processed: 4/26/13 10:26:40:660 EDT - 4/29/13 17:02:42:385 EDT

All WXS Critical Messages

All Exceptions

Topology Summary

Topology Consistency

Event Timeline View

▼ Top 5 Incidents

Top 5 Incidents from process startup
[Click here for full Exception report](#)

Message Type	Count	Percent	Infected	Time Range
CWOBJ82611	6	40.00	container1, cat1	4/26/13 10:26:46:899 EDT - 4/29/13 4:48:20:822 EDT
ClassNotFoundException processing class com.ibm.ws.objectgrid.jpa.batch.QueryClearAgent. Message=com.ibm.itso.redbook.wxs.Owner	3	20.00	container1	4/29/13 4:47:10:265 EDT - 4/29/13 4:47:10:373 EDT
CWOBJ0006W	3	20.00	container1	4/29/13 4:47:10:275 EDT - 4/29/13 4:47:10:460 EDT
FFDC1003I	2	13.33	cat1	4/29/13 4:48:27:497 EDT - 4/29/13 4:48:40:500 EDT
ClientCommunicatorAdmin restart Failed to restart: java.io.IOException: Failed to get a RMI stub: javax.naming.NameNotFoundException: objectgrid/MBeanServer	1	6.67	cat1	4/29/13 4:44:16:141 EDT - 4/29/13 4:44:16:141 EDT

▼ Details for Top 5 Incidents

Details for Top 5 Incidents

▼ CWOBJ82611

Incident: CWOBJ82611

Infection Summary
 Distribution of event across log files: [Click here for full Exception report](#)

Process	Log File	Count	Percent	Time Range
cat1	/opt/ibm/WebSphere/eXtremeScale/ObjectGrid/bin/logs/cat1/SystemOut.log	2	33.33	4/26/13 10:26:46:899 EDT - 4/29/13 4:45:28:898 EDT
container1	/opt/ibm/WebSphere/eXtremeScale/ObjectGrid/bin/logs/container1/SystemOut.log	4	66.67	4/26/13 10:29:46:694 EDT - 4/29/13 4:48:20:822 EDT

Top 5 Stack Variations
 Percent of Total Event Count: First Line of Full Stack

- ▶ 16.67% : CWOBJ82611: Critical log message and first-failure data capture (FFDC) exception notifications are enabled for the cat1 container server.
- ▶ 16.67% : CWOBJ82611: Critical log message and first-failure data capture (FFDC) exception notifications are enabled for the container1 container server.

Figure 7-12 Sample xsLogAnalyzer HTML output

7.4 JVM logging and tracing

Log and trace files are a key aspect to troubleshooting the grid environment. Generally, log files contain informational, warning, and critical error messages. Trace files are used to pinpoint exactly what is occurring in the grid at various levels of verbosity. Additionally, a third type of logging called First-Fail Data Capture (FFDC) can provide value in the troubleshooting process by logging more verbose data for certain exceptions.

Enabling logging for the grid depends on how you install WebSphere eXtreme Scale. This section addresses how to set up logging in a WebSphere Application Server managed environment and in a stand-alone WebSphere eXtreme Scale environment.

7.4.1 Enabling logging and tracing in a stand-alone WebSphere eXtreme Scale environment

By default, all grid servers, catalog, and containers generate log files. The directory in which the log files are generated depends on the current working directory when the start server script is run. For example, if you run the start server script from the `wxs_install_root/ObjectGrid/bin` directory, the logs are generated in the `wxs_install_root/ObjectGrid/bin/logs` directory.

To customize logging to specify a different logging location, create a property file that contains the customizations. An example property file is shown in Example 7-19.

Example 7-19 Sample logging configuration property file

```
workingDirectory=<directory>
traceSpec= <trace_string>
systemStreamToFileEnabled=true
```

In Example 7-19, the following properties are set:

- ▶ The **workingDirectory** setting describes the root directory where all logging is generated. WebSphere eXtreme Scale creates a subdirectory in the path for `workingDirectory` named after the server name provided by the user. To use the property file with a new server instance, the **-serverProps** parameter can be added to the start server command as shown in Example 7-20.

Example 7-20 Using a server properties file

```
startXsServer.sh container1 -catalogServiceEndpoints hostname:port -serverProps
/path/to/server.props
```

- ▶ To enable tracing for IBM Support use, provide a value for the **<trace_string>** parameter shown in Example 7-19.
- ▶ The **systemStreamToFileEnabled** property enables the JVM to log to a file. If it is set to `false`, the JVM logs to the console.

7.4.2 Enabling logging in a WebSphere Application Server managed eXtreme Scale environment

Enabling logging in a WebSphere Application Server managed eXtreme Scale environment is fairly simple. If you are familiar with WebSphere Application Server administration, you have likely enabled logging and have specified custom trace settings. For more information, see the *Enabling and disabling logging* topic in the Network Deployment (Distributed operating systems), Version 7.0 Information Center:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.base.doc/ae/tprf_enablelog.html

For a WebSphere Application Server managed environment, tracing is configured by using the standard manner.

7.4.3 Using High Performance Extensible Logging

High Performance Extensible Logging (HPEL) is a log and trace component that can be used in both the WebSphere Application Server managed and stand-alone deployment scenarios.

HPEL generates logs in a binary format and uses simple tools and APIs to access the binary data in a human-readable format. HPEL is a fast and efficient manner of logging. For more information about HPEL and instructions for configuring it in a WebSphere Application Server managed environment, see the following IBM Education Assistant website:

http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.was_v8/was/8.0/ProblemDetermination/WASv8_HPEL/player.html

To enable HPEL logging in a stand-alone WebSphere eXtreme Scale environment, add the `hpe1Enable` and `hpe1RepositoryLocation` options to a custom log configuration property file.

Example 7-21 depicts the same log configuration from Example 7-19 on page 327 with the new HPEL setting added.

Example 7-21 Enabling HPEL logging in the server properties file

```
workingDirectory=<directory>
traceSpec=
systemStreamToFileEnabled=true
hpe1Enable=true
hpe1RepositoryLocation=<directory>
```

7.5 Monitoring with other products

Other products can help monitor a WebSphere eXtreme Scale environment. This section covers some of these options.

7.5.1 IBM Tivoli Enterprise Monitoring Agent

The IBM Tivoli® Enterprise Monitoring Agent is a feature-rich monitoring solution that you can use to monitor databases, operating systems, and servers in distributed and host environments. WebSphere eXtreme Scale includes a customized agent that you can use to inspect eXtreme Scale management beans. The monitoring agent can be used with both stand-alone eXtreme Scale and WebSphere Application Server deployments.

For more information, see the *Monitoring with the IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsmonitortivoli.html>

7.5.2 CA Wily Introscope

CA Wily Introscope is a monitoring and performance management product. To monitor WebSphere eXtreme Scale applications with CA Wily Introscope, you must supply instrumentation points or code points. CA Wily Introscope uses ProbeBuilderDirective (PBD) files to do monitoring. WebSphere eXtreme Scale provides example PBD files for catalog and container servers in the WebSphere eXtreme Scale Information Center.

For more information, see the *Monitoring eXtreme Scale applications with CA Wily Introscope* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxsmonitorintroscope.html>

7.5.3 Hyperic HQ

Hyperic HQ is a monitoring and performance management product that is available as an open source or an Enterprise product. WebSphere eXtreme Scale includes a plug-in that allows Hyperic HQ agents to discover container servers, and to report and aggregate statistics. The plug-in uses eXtreme Scale Managed Beans (MBeans).

For more information, see the *Monitoring eXtreme Scale with Hyperic HQ* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsmonitorhyperic.html>

7.5.4 Java JConsole and other MBean inspection tools

WebSphere eXtreme Scale provides a set of MBeans that can be accessed programmatically or by an MBean tool such as Java JConsole.

The MBeans can view or change the grid. Many of the xscmd commands are also available through MBeans.

For more information, see the *Administering with Managed Beans (MBeans)* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxsmbean.html>

7.6 Tuning, recovery and troubleshooting

This section covers some general tuning troubleshooting for the infrastructure of a grid. It includes the following topics:

- ▶ 7.6.1, “Tuning the heartbeat frequency level” on page 329
- ▶ 7.6.2, “Tuning and controlling placement” on page 330
- ▶ 7.6.3, “Tuning container server start and stop” on page 332
- ▶ 7.6.4, “Off loading work from primary shards: replicaReadEnabled option” on page 332
- ▶ 7.6.5, “Recovery for network blips and brown outs” on page 333
- ▶ 7.6.6, “Recovery for client failures” on page 333
- ▶ 7.6.7, “Troubleshooting placement problems” on page 334
- ▶ 7.6.8, “Troubleshooting synchronous replication” on page 334
- ▶ 7.6.9, “Troubleshooting asynchronous replication” on page 337
- ▶ 7.6.10, “Troubleshooting multi-master replication” on page 338
- ▶ 7.6.11, “Testing a custom collision arbiter for multi-master replication” on page 339

There are also several troubleshooting articles in the WebSphere eXtreme Scale version 8.6 Information Center *Troubleshooting* topic:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/rxstrouble.html>

7.6.1 Tuning the heartbeat frequency level

Three heartbeat settings can be tuned to affect how aggressively the availability of the containers is monitored for fail over purposes. These settings use the following heartbeat frequency levels:

- ▶ Level 0 uses a 30-second failure detection time
- ▶ Level -10 uses a 15-second failure detection time
- ▶ Level -1 uses a 5-second failure detection time
- ▶ Level 10 using a 90-second failure detection time
- ▶ Level 1 uses a 180-second failure detection time

By the default, eXtreme Scale sets the heartbeat frequency to level 0. Level -10 and 10 are new to WebSphere eXtreme Scale v8.6.0.2.

If you want a more aggressive fail over and highly responsive system, specify `-heartbeat -10` when you start the catalog server. Many systems cannot balance the most aggressive heartbeat without false positives on heartbeat failures. If a server experiences a long JVM garbage collection pause, the more aggressive heartbeat options can trigger and servers will be incorrectly marked as down. If the servers are marked as down, they must shut down and be restarted. For more information, see 7.6.2, “Tuning and controlling placement” on page 330.

If you want the system to have less processor impact and to be more resource friendly, start the catalog server with `-heartbeat 10`. Doing so makes fail over occur slower, but generates less heartbeat traffic. A long fail over time causes longer response times on clients. If the clients are running with a `clientRetryTimeout` less than 90 seconds, they experience `TargetNotAvailableException` errors more frequently than a shorter time-out.

For more information about setting the heartbeat interval, see the *Tuning the heartbeat interval setting for failover detection* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Ftxsfailover.html>

7.6.2 Tuning and controlling placement

There are several ways to tune the timing of placement events. Depending on the scale of an environment and the startup sequence, different placement control choices create an efficient start. Without a placement constraint, shard thrashing or bouncing shards from server to server can occur when you start or stop several container servers. In a small test environment, the impact is small. In a large environment, moving shards around uses memory, processor, and network resources to promote primary shards and move replica shards.

The following are the pause or hold placement options:

- ▶ The `xscmd suspendBalancing` command (also available on the `PlacementServiceMBean`). This command pauses all normal placement for a grid. It can be used before any containers are started or between container server stops and starts. If balance is suspended during normal run time, server failure events still trigger placement. Suspending balance gives you precise control.
- ▶ The `placementDeferralInterval` catalog server property (set to 15 seconds by default). This property sets a timer whenever a container starts. If another server starts within the timer, the timer resets. When it expires (no new containers starting), placement begins. It is a good control for servers that are starting during the same time frame.
- ▶ The `numInitialContainers` deployment policy option. This option waits until the number of containers started match the `numInitialContainers` setting. This setting only provides control at the initial startup.

The preferred practice is to use a single placement constraint to control placement. One constraint reduces confusion on when placement occurs.

There is a precedence when using more than one constraint. A **suspendBalancing** overrides both **numInitialContainers** and the **placementDeferralInterval**. The **numInitialContainers** overrides the **placementDeferralInterval**.

The following actions initiate placement:

- ▶ The xscmd **triggerPlacement** command.
- ▶ The xscmd **resumeBalancing** command (also available on the PlacementServiceMBean).
- ▶ Number of container servers started being equal or greater than the **numInitialContainers** deployment policy option started.
- ▶ The **placementDeferralInterval** catalog server property expires.
- ▶ Server changes, starting servers, or stopping servers unless a placement constraint is used.
- ▶ Server failure.

The **triggerPlacement** command overrides all placement constraints. If balance is suspended and the **numInitialContainers** property is set, the **numInitialContainers** property is checked even if a **resumeBalancing** command is issued. For example:

1. The grid `myGrid` has `numInitialContainers="5"` in its deployment policy file.
2. The **suspendBalancing** command is run.
3. Four container servers are started.
4. The **resumeBalancing** command is run.
5. Placement does not occur until a fifth container is started or the **triggerPlacement** command is used.

To use suspending and resuming balance during initial startup of a catalog server domain, complete the following steps:

1. Start the catalog servers.
2. Run the **suspendBalancing** command:

```
xscmd -c suspendBalancing -g grid_name -ms mapSet_name
```
3. Start the container servers.
4. After the container servers are all started, run the **resumeBalancing** command:

```
xscmd -c resumeBalancing -g grid_name -ms mapSet_name
```
5. When the **resumeBalancing** command returns, the execution of the placement work is ongoing. Use the **showPlacement**, **routeTable**, and **showMapSizes** commands to monitor the work.

Other commands can modify placement, such as the **balanceShardTypes** command. This command redistributes the balance of primary and replica shards across containers.

For more information about using xscmd to view and control placement, see 7.2.1, “Placement layout: showPlacement command” on page 311.

Also, see the *Controlling placement* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsplacement.html>

7.6.3 Tuning container server start and stop

Container servers start and join the same catalog service domain by providing the catalog service endpoints, `catalogServiceEndpoints`, list of catalog server hosts, and listener ports. The container also provides an ObjectGrid descriptor and deployment policy file.

Container servers are best started or stopped in groups. Using a placement constraint as described in 7.6.2, “Tuning and controlling placement” on page 330 helps reduce server churn. Stopping or starting several servers individually causes shards to move around on every server change. For example, stopping several servers back to back can force a shard to move several times if it gets moved to the next server to stop.

To stop container servers in batches, use the **teardown** command. The catalog service stops a group of servers and bundles the shard movement into one set of moves instead of several. For more information, see 7.2.7, “Stopping servers efficiently: teardown command” on page 319 and the *Stopping servers gracefully with the xscmd utility* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsteardown.html>

The **teardown** command can be used on stand-alone servers and WebSphere Application Server managed servers. When the **teardown** command is used against servers in a WebSphere Application Server managed environment, the WebSphere eXtreme Scale containers that are running in the WebSphere Application Server stop. The application server remains running, and can be stopped in the administrative console.

When servers stop individually, the **showMapSizes** or **routeTable** commands can be used to check that shards moved clean before the next container server stop or start.

7.6.4 Off loading work from primary shards: replicaReadEnabled option

A replica shard can take some of the work from a primary shard by handling read operations. If a grid is read mostly, allowing reads on replicas improves performance.

When replicas are read enabled by setting the `replicaReadEnabled` option to `true` in the deployment policy file, they process get requests from clients. The `replicaReadEnabled` option is `false` by default.

Reading on replicas in a read only or read mostly ObjectGrid is useful for spreading out client get requests. If the grid is high write, the replica can return stale data. If the key is not yet on a replica, the replica shard forwards to the primary shard.

For more information, see the *Reading from replicas* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/cxsreadreplx.html>

7.6.5 Recovery for network blips and brown outs

When the network is unavailable for an amount of time, servers can appear to be dead or gone. WebSphere eXtreme Scale starts fail over recovery when servers do not respond to heartbeats. The catalog service removes container servers and rebalances shards to account for failures.

The network can return without actual JVM failures. When servers are removed from the catalog service based on heartbeat failures, but return later, they cannot be directly added back to the catalog service. The servers likely have shards on them that have moved to other servers in the domain.

To provide recovery, the catalog server tells the server that it was removed from the group. The server then goes through a reconnect process where it stops and restarts, and then joins the catalog server again. The container server looks like a new server to the catalog server and can have shards balanced to it. In a stand-alone environment, the JVM restarts. In a WebSphere Application Server managed environment, the WebSphere eXtreme Scale container restarts, but the entire application server does not restart.

To manually recover servers that disconnect from the catalog service domain, the reconnect option can be disabled by using a system property. If reconnect is disabled, the container servers must be manually restarted when they are disconnected from the catalog service domain.

Messages are logged when servers leave and try to rejoin. Example 7-22 shows a heartbeat failure message for a container server that is logged on the catalog server.

Example 7-22 Catalog server JVM log message, removing a server from the core group view

```
CatalogServic I   CWOBJ7211I: As a result of a heartbeat (view heartbeat type)
from leader mach1.ibm.com:53416 for core group DefaultZoneCGO with member list
mach2.ibm.com:53416, the server xsServer01 is being removed from the core group
view.
```

The container server also logs messages when it reconnects. Example 7-23 shows two messages that occur on a container server during container reconnect.

Example 7-23 Container server JVM messages on container reconnect

```
CWOBJ1234I: A container reconnect message was received from server
catalogServer01.
CWOBJ1213I: Server was disconnected from the primary catalog server but was able
to reconnect.
```

For more information about the container reconnect properties, see the *Container server reconnect properties* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fconfigurecontainerreconnect.html>

7.6.6 Recovery for client failures

When the primary shard dies, the client request can be tried again transparently to the user. Each client request has a time-out window. If an exception was received on a request and the exception indicates that it is safe to try again, the client asks for a new route table and tries

the request again until the time out is reached. When the time out is reached, a `TargetNotAvailableException` error message is generated.

Most communication exceptions that are received during client requests can be tried again. If the client receives a `MessageTimeoutException` (or a `NO_RESPONSE SystemException`), the request is immediately returned. A `MessageTimeoutException` indicates the remote side (a shard) received the request, but did not complete it before the client side timed out. The request is not tried again because the remote side might be processing a long running transaction. If the client submits the request again automatically, a duplicate key or other transaction exception might occur. A `TargetNotAvailableException` message is generated in this case with a cause by of `MessageTimeoutException`.

7.6.7 Troubleshooting placement problems

The catalog service monitors placement progress. Placement problems can occur if there are network problems and the work does not reach its intended target. A container server might be experiencing high processor or GC problems in a resource constrained environment and become unresponsive. If there are problems placing shards in one location, the catalog service attempts to place them in a new location. If shards cannot be placed, the catalog service leaves them on a virtual unassigned container that can be reviewed by using the **showPlacement** command. Shards that are unassigned are waiting for enough container servers to be running to be placed or another placement event. For more information about placement events, see 7.6.2, “Tuning and controlling placement” on page 330. For more information about using the **showPlacement** command, see 7.2.1, “Placement layout: showPlacement command” on page 311.

When individual shards cannot be placed on container servers, the catalog service marks the container server as disabled. Because the container cannot successfully host a shard, the catalog service does not try to place more shards on the container.

To review the list of disabled container servers, use the `xscmd listDisabledForPlacement` command. Review the status of any container servers in the list to ensure that they are available and healthy.

When the container servers recover, add them back using the **enableForPlacement** command and provide the name of container server to join:

```
xscmd -c enableForPlacement -ct container_server
```

The `xscmd showInfo` command can be used to check whether a container server is still running. Review the container server’s JVM logs for CPU starvation messages with long delays:

```
HAControllerI W HMGR0152W: CPU Starvation detected. Current thread scheduling delay is 15 seconds.
```

For more information, see the *Controlling placement* topic in the WebSphere eXtreme Scale version 8.6 Information Center:

<http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/txsplacement.html>

7.6.8 Troubleshooting synchronous replication

Synchronous (sync) replicas must commit with the primary shard. When sync replicas cannot commit, they go through a recovery period. When there are not enough sync replicas to

commit a transaction, the primary shard generates a `ReplicationVotedToRollbackTransactionException` error message.

ReplicationVotedToRollbackTransactionException

A `ReplicationVotedToRollbackTransactionException` message often indicates a temporary error state. If the primary shard fails, the sync replica shard is promoted to provide client access as soon as possible. Then, a new sync replica shard is added. During this short time, a client receives the `ReplicationVotedToRollbackTransactionException` error message. As soon as the new sync replica is placed and running, new transactions are successful again.

When a `ReplicationVotedToRollbackTransactionException` error message continues to happen, one of the following scenarios might be the cause:

- ▶ There are not enough container servers to host a new sync replica shard. Use the `xscmd showPlacement` command to review which shards are placed. For more information, see 7.2.1, “Placement layout: showPlacement command” on page 311.
- ▶ The sync replica shard failed to commit because of a network or JVM error. For more information, see “Replication failures” on page 335.
- ▶ The sync replica shard was placed, but has a large volume of data to replicate before it is available for new transactions. Monitor for the `CWOBJ1526 enter peer mode` message in the JVM log of the sync replica. Also, use the `xscmd showMapSizes` and `revisions` commands. For more information, see 7.2, “Command line access: xscmd” on page 310.
- ▶ There was a failure during replication and the sync replica must re-register with the primary shard. For more information, see “Replication failures” on page 335. Monitor for the `CWOBJ1526 enter peer mode` messages in the JVM log of the sync replica. Also, use the `xscmd showMapSizes` and `revisions` commands. For more information about these commands, see 7.2.3, “Size of the grid: showMapSizes command” on page 315 and 7.2.5, “Version of grid contents: revisions command” on page 317.

Replication failures

For synchronous replica shards, there can be communication failures between the primary shard and the sync replica. There can also be failures while applying the data to the sync replica. Both of these failures have the same recovery path.

If the primary receives a communication error from the sync replica during replication, the primary stops trying to commit data to it. This frees the primary from more network delays. In the meantime, the primary shard pings the replica and recovers the sync replica shard when it is available again.

If the sync replica shard encounters an error applying data, the sync replica also goes into a recovery path.

Whenever a sync replica goes into the recovery path, it must register with the primary shard and get a fresh set of data. The sync replica shard cannot fix part of its data and maintain its synchronous peer mode state with the primary shard.

The sync replica shard automatically goes through the process to re-register. During this time, the sync replica logs a `CWOBJ1524I` message that indicates that it must re-register. Then, the sync replica enters peer mode again.

Example 7-24 shows an example of the CWOBJ1524 message that is logged when the primary shard experienced a communication problem with the replica shard.

Example 7-24 Primary shard lost and regained communication with sync replica

```
SynchronousRe I CWOBJ1524I: Replica listener grid:mapSet1:0 must re-register with the primary. Reason: Replica was disconnected from primary on containerName for an unknown length of time and must be reregistered to restart replication
```

Example 7-25 shows an example of the CWOBJ1524 when the sync replica shard experienced a problem applying data from the primary.

Example 7-25 Sync replica experiences a failure and successfully re-registers with the primary shard

```
SynchronousRe I CWOBJ1524I: Replica listener grid:mapSet1:0 must register again with the primary. Reason: com.ibm.websphere.objectgrid.DuplicateKeyException: ObjectGrid: accounting, Map: payroll, Key: dupKey
```

Troubleshooting

Exceptions can occur on a sync replica shard when it is applying the data from the primary shard. In all of these cases, the sync replica shard tries to recover by performing a re-register. If a sync replica shard does several re-register attempts without successfully committing data, it stops trying to recover or replicate. The sync replica shard generates a CWOBJ1537E message to the log and alerts the catalog service to its failure.

Example 7-26 shows an example of a sync replica shard that re-registered several times without successfully committing the data.

Example 7-26 Sync replica shard attempted to recover, but was unsuccessful after several tries

```
SynchronousRe E CWOBJ1537E: grid:mapSet1:0 exceeded the maximum number of times to reregister (3) without successful transactions.
```

If the replica shards are missing, see 7.6.7, “Troubleshooting placement problems” on page 334 for more information.

To review the amount of data in a sync replica shard, use the `xscmd showMapSizes` and `revisions` commands. For more information about these commands, see 7.2.3, “Size of the grid: showMapSizes command” on page 315 and 7.2.5, “Version of grid contents: revisions command” on page 317.

KeyNotFoundException or DuplicateKeyException

A `KeyNotFoundException` message indicates that the key that the replica is trying to update or remove is not on the replica shard. A `DuplicateKeyException` message indicates that the key that the replica is trying to insert is already on the replica shard.

The most common causes of `KeyNotFoundExceptions` and `DuplicateKeyExceptions` are when using a lock strategy (`lockStrategy`) of `NONE` or no locking (versus `Pessimistic` or `Optimistic`) in the `ObjectGrid` descriptor file. When `NONE` is selected as the lock strategy, the application must control the locking. If locking is not done properly in the application, the replica shard can encounter a `KeyNotFoundException` when the primary shard processes an update and does a remove or insert and update at the same time. The lock strategy of `NONE` is therefore not recommended.

When the `hashCode()` or `equals()` methods on a custom key are incorrect or incomplete, `KeyNotFoundExceptions` and `DuplicateKeyExceptions` can also occur. Routing to the correct

partition and accessing the correct key requires a consistent and accurate `hashCode()` and `equals()` method. For example, if the hash code is not consistent, a `KeyNotFoundException` can occur on a replica. The key might hash one way for a primary, but differently on the replica.

`KeyNotFoundException`s can also occur when the application reuses a key object in the same transaction. The transaction displays incorrectly on the replica.

OptionalDataException and ClassCastException

If the replica shard does not have the correct class path or cannot properly process the keys or values in the transaction, a replica can generate an `OptionalDataException` or `ClassCastException` error message. An `OptionalDataException` message can also indicate the `readObject()` or `writeObject()` implementations are incorrect on the key or value class.

LockTimeoutException

A `LockTimeoutException` message can occur when transactions are flowing slowly from the primary to the replica. The cause might be high processor or network usage that slows the completion of transactions on the replica.

A special case: Temporary sync replica

If you look closely at the JVM logs, you see a temporary synchronous replica (temp sync replica) shard displayed, especially during container server stops and starts. Example 7-27 shows an example `CWOBJ1511I` message with a temporary sync replica.

Example 7-27 A replica shard marked as temporary, placed to copy data before primary promotion

```
SynchronousRe I  CWOBJ1511I: accounting:mapSet1:2 (temporary synchronous replica)
is open for business.
```

The temp sync replica shard acts like a normal sync replica shard. However, it is only used when the primary moves to a new location where there is no existing shard to promote. The temp sync replica copies all the data from the old primary before it becomes the new primary. When the temp sync replica enters peer mode, the shard completes its promotion to primary. The temp sync replica is unrelated to the deployment policy replica settings.

The temp sync replica shard can also appear as an extra sync replica shard in the output of the `showMapSizes` command until it completes its promotion to primary shard.

7.6.9 Troubleshooting asynchronous replication

Asynchronous (async) replicas poll the primary shard for data. Network errors or brief communication errors during failover are the most frequent problems.

Replication failures

If the async replica shard loses contact with the primary shard, it pings the primary shard until the primary shard is available again. After the primary shard is available, the async replica replicates again. If the primary loses contact with the replica when it returns the new packet of data, the primary does not attempt any recovery. The async replica polls again when the network problems are resolved. The async replica shard does not have to enter a special recovery path to re-register like a synchronous replica shard. The async replica shard can start again where it left off on the last query.

Troubleshooting

When async replica shards experience communication problems with the primary shard, usually the primary shard died and the replica shard polled it before the catalog service initiated fail over. If the primary shard moves, the new primary shard attaches the async replica shard to itself and restarts replication. If there is a temporary networking problem, the replica shard can also experience communication problems.

To review the amount of data in an async replica, use the `xscmd showMapSizes` and `revisions` commands. For more information about these commands, see 7.2.3, “Size of the grid: showMapSizes command” on page 315 and 7.2.5, “Version of grid contents: revisions command” on page 317.

If the replica shards are missing, see 7.6.7, “Troubleshooting placement problems” on page 334 for more information.

7.6.10 Troubleshooting multi-master replication

During MMR, primary shards poll remote or foreign primary shards for data. Network errors or brief communication errors during failover are the most frequent problems.

Replication failures

If a primary shard loses contact with its foreign primary shard, it switches to pinging the foreign primary shard until the foreign shard is available again. After the foreign shard is available, the replication restarts. The primary shard can start again where it left off on the last query.

If a minimum number of synchronous replicas are configured, the data that are retrieved from the foreign primary must be successfully applied on the primary and on the sync replica shard. If a primary’s sync replica shard is down or fails to apply the data from the foreign primary, the transaction fails. The primary shard then stops querying its foreign primary. When the sync replica shards come back online, the primary shard restarts replication with its foreign primary.

Troubleshooting

When primary shards experience communication problems with their foreign shards, usually the foreign primary shard moved. When servers stop or start, the catalog servers exchange placement information. The primary shard receives a new location for its foreign primary shard and restarts replication. If there is a temporary networking problem, the primary shard can also experience communication problems.

To review the amount of data in each domain, use the `xscmd showMapSizes` and `revisions` commands. For more information about these commands, see 7.2.3, “Size of the grid: showMapSizes command” on page 315 and 7.2.5, “Version of grid contents: revisions command” on page 317. These commands must be run on each domain. The total map sizes can be compared directly. The shard types and partitions can be compared to inspect each primary partition.

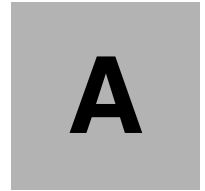
If you find using the `xscmd showMapSizes` command that data from one domain is missing in the other domain, use the `xscmd showLinkedPrimaries` command with the `-hc` option to check whether all the primary shard links are online. For more information, see 7.2.9, “Multi-master replication commands” on page 320.

7.6.11 Testing a custom collision arbiter for multi-master replication

Testing a custom collision arbiter can be done in a small environment by doing updates on keys after dismissing the link. To do so, complete the following steps:

1. Start two domains, domainA and domainB, with enough containers to place primary partitions.
2. Link the domains by running the `xscmd establishLink` command from one domain pointing to the other domain.
3. Insert a key into one of the domains.
4. Run the `xscmd -c showMapSizes` command on both domains until the key exists in both.
5. Close the link by running the `xscmd dismissLink` command from one domain.
6. Update the recently inserted key on domainA, and review the revision data by using the `xscmd revisions` command running against domainA.
7. Update the recently inserted key on domainB, and review the revision data by using the `xscmd revisions` command running against domainB.
8. Link the domains again.
9. Review the revisions on both domains again.
10. Get the key from both domains and verify that the key has the expected contents on both domains. The default collision arbiter keeps the change from domainA.

A custom collision arbiter must be consistent regardless of the domain that uses it. If the arbiter gives different results in domainA versus domainB, the grid data is inconsistent.



Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247683>

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG24-7683-01.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
Developing.zip	This zip file includes the sample code used in Chapter 4, “Developing with WebSphere eXtreme Scale” on page 95. This sample code shows how to assemble a module for development with the Liberty profile. It also illustrates the configuration required for using indexes and code for deploying a sample agent.
DeploymentScenario.zip	This zip file includes the Getting Started application and grid configurations used in the different deployment scenarios described in Chapter 5, “Deployment scenarios” on page 151. This includes the stand-alone environment, integrated installation

with WebSphere Application Server Network Deployment, and integrated installation with WebSphere Application Server Liberty profile environments.

HTTPSession.zip

This zip file contains a test application and grid configuration used in Chapter 6, “Extended HTTP session management with WebSphere eXtreme Scale” on page 243. This sample code demonstrates the proper functionality of an HTTP session for a WebSphere Application Server Network deployment, WebSphere Application Server Liberty profile, and Apache Tomcat environments.

Downloading and extracting the Web material

Create a subdirectory (folder) on your workstation, and extract the contents of the Web material .zip file into this folder.

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get Redbooks publications” on page 345. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *Enterprise Caching Solutions using IBM WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale*, SG24-8043
- ▶ *Elastic Dynamic Caching with the IBM WebSphere DataPower XC10 Appliance*, SG24-4851
- ▶ *IBM solidDB: Delivering Data with Extreme Speed*, SG24-7887
- ▶ *Integrating WebSphere Commerce with IBM WebSphere DataPower XC10*, REDP-4823
- ▶ *Scalable, Integrated Solutions for Elastic Caching Using IBM WebSphere eXtreme Scale*, SG24-7926
- ▶ *WebSphere Application Server Liberty Profile Guide for Developers*, SG24-8076
- ▶ *WebSphere Application Server V8.5 Concepts, Planning, and Design Guide*, SG24-8022
- ▶ *WebSphere Application Server V8.5 Administration and Configuration Guide for the Full Profile*, SG24-8056
- ▶ *WebSphere eXtreme Scale Best Practices for Operation and Management*, SG24-7964

Online resources

These Web sites are also relevant as further information sources:

- ▶ WebSphere eXtreme Scale product home page
<http://www.ibm.com/software/webservers/appserv/extremescale>
- ▶ WebSphere eXtreme Scale wiki documentation
http://www-128.ibm.com/developerworks/wikis/x/_IMF
- ▶ IBM WebSphere eXtreme Scale Version 8.6 Information Center
http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6/topic/com.ibm.websphere.extremescale.doc/kc_welcome-xs.html
- ▶ IBM Elastic Caching community
<http://www.ibm.com/developerworks/connect/caching>
- ▶ WebSphere Application Server documentation
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/welcome_nd.html

- ▶ Integrating WebSphere eXtreme Scale transactions with other transactions
http://www.ibm.com/developerworks/websphere/techjournal/1205_jolin/1205_jolin.html
- ▶ Java Object Serialization Specification version 1.5.0
<http://java.sun.com/j2se/1.5.0/docs/guide/serialization/spec/serialTOC.html>
- ▶ SpringSource
<http://www.springsource.org/>
- ▶ *Getting Started with Integrating eXtreme Scale and Spring*
https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/getting_started_with_spring4?lang=en
- ▶ *Integrating eXtreme Scale and Spring on the XC10 Appliance*
https://www.ibm.com/developerworks/community/blogs/714470bb-75c8-4f99-8aca-766c0d55a21c/entry/getting_started_with_spring5?lang=en
- ▶ *Enhancing WebSphere Commerce performance with WebSphere eXtreme Scale*
http://www.ibm.com/developerworks/websphere/techjournal/1008_genkin/1008_genkin.html
- ▶ *Innovations within reach: Using WebSphere eXtreme Scale to enhance WebSphere Portal and IBM Web Content Manager performance* article:
http://www.ibm.com/developerworks/websphere/techjournal/1206_inreach/1206_inreach.html
- ▶ *Configure WebSphere Commerce with WebSphere eXtreme Scale to improve performance, scale, and your competitive edge:*
http://www.ibm.com/developerworks/websphere/techjournal/1108_bohn/1108_bohn.html
- ▶ IBM WebSphere DataPower XC10 V2.0 WebSphere Commerce integration presentation
http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wdatapower/wdatapower/2.0/xc10/XC10_Commerce_Integration/player.html
- ▶ *Integrating WebSphere DataPower XC10 and XI50 Appliances*
http://www.ibm.com/developerworks/websphere/library/techarticles/1111_nijhawan/1111_nijhawan.html
- ▶ *WebSphere eXtreme Scale Trial and WebSphere eXtreme Scale for Developers - Liberty Profile*
<http://www.ibm.com/developerworks/downloads/ws/wsdg/>
- ▶ Eclipse downloads
<http://www.eclipse.org/downloads>
- ▶ *WXSUtils Programming Guide V0.1*
<https://github.com/bnewport/Samples/blob/master/wxsutils/WXSUtilsProgramming.pdf>
- ▶ *Tips and techniques for WebSphere eXtreme Scale DynaCache in WebSphere Commerce environments, Part 1: General caching and WebSphere eXtreme Scale servers*
<http://www.ibm.com/developerworks/library/co-wxs-dynamic-cache>

- ▶ WebSphere Application Server, Network Deployment, Version 8.5 Information Center
http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/welcome_ndmp.html
- ▶ Apache Tomcat Connector - Webserver HowTo documentation
http://tomcat.apache.org/connectors-doc/webserver_howto/apache.html

How to get Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Redbooks

WebSphere eXtreme Scale V8.6: Key Concepts and Usage Scenarios

(0.5" spine)

0.475" x 0.873"

250 <-> 459 pages



WebSphere eXtreme Scale V8.6

Key Concepts and Usage Scenarios



Redbooks®

Introduction to elastic caching

IBM WebSphere eXtreme Scale provides a solution to scalability issues through caching and grid technology. It provides an enhanced quality of service in high performance computing environments.

Typical application scenarios

This IBM Redbooks publication introduces WebSphere eXtreme Scale and shows how to set up and use an eXtreme Scale environment. It begins with a discussion of the issues that would lead you to an eXtreme Scale solution. It then describes the architecture of eXtreme Scale to help you understand how the product works. It provides information about potential grid topologies, the APIs used by applications to access the grid, and application scenarios that show how to effectively use the grid.

Deployment options

This book is intended for architects who want to implement WebSphere eXtreme Scale.

The original edition of this book was based on WebSphere eXtreme Scale version 6.1. It was published in 2008 and described as a “User’s Guide”. This second edition updates the information based on WebSphere eXtreme Scale version 8.6, and covers key concepts and usage scenarios.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7683-01

ISBN 073843860X