TRN4032: High Availability and Sharding Deep Dive with Next Generation Oracle Database

Wei Hu
Vice President, Product Development
High Availability Technologies
October 22, 2018

ORACLE OPEN WORLD

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Amazon.com runs Oracle

- Amazon Payments Platform runs on Oracle
  - eCommerce payments system supporting millions of Amazon customers
- Amazon Retail Systems run on Oracle
  - Multiple large databases supporting OLTP and DSS
- Amazon Digital Commerce Platform runs Oracle
  - Supports the services behind the Kindle and other Amazon digital businesses
- *Process payments at an unprecedented scale, with accuracy, speed, and mission-critical availability* – Amazon.com quote
- *Amazon spends $60 million with Oracle* – Larry Ellison, OOW 2017

Why does Amazon.com use Oracle?

- Amazon tells their customers to use Aurora, Redshift, Postgres, and MySQL
- But these databases are not good enough for Amazon.com
  - No RAC, no Active Data Guard, no Real-Application Testing, …
- Important take-away:
  - You can modify an application written to Oracle to *run* on other databases
  - But only Oracle can run applications at the availability, and security that Amazon.com (and other customers) require
  - Also, unlike Amazon's databases (Aurora, Redshift), Oracle is available on-premises and in the cloud (including AWS)
- This session will talk about the Oracle availability features that customers like Amazon.com rely on to keep their applications running 24x7

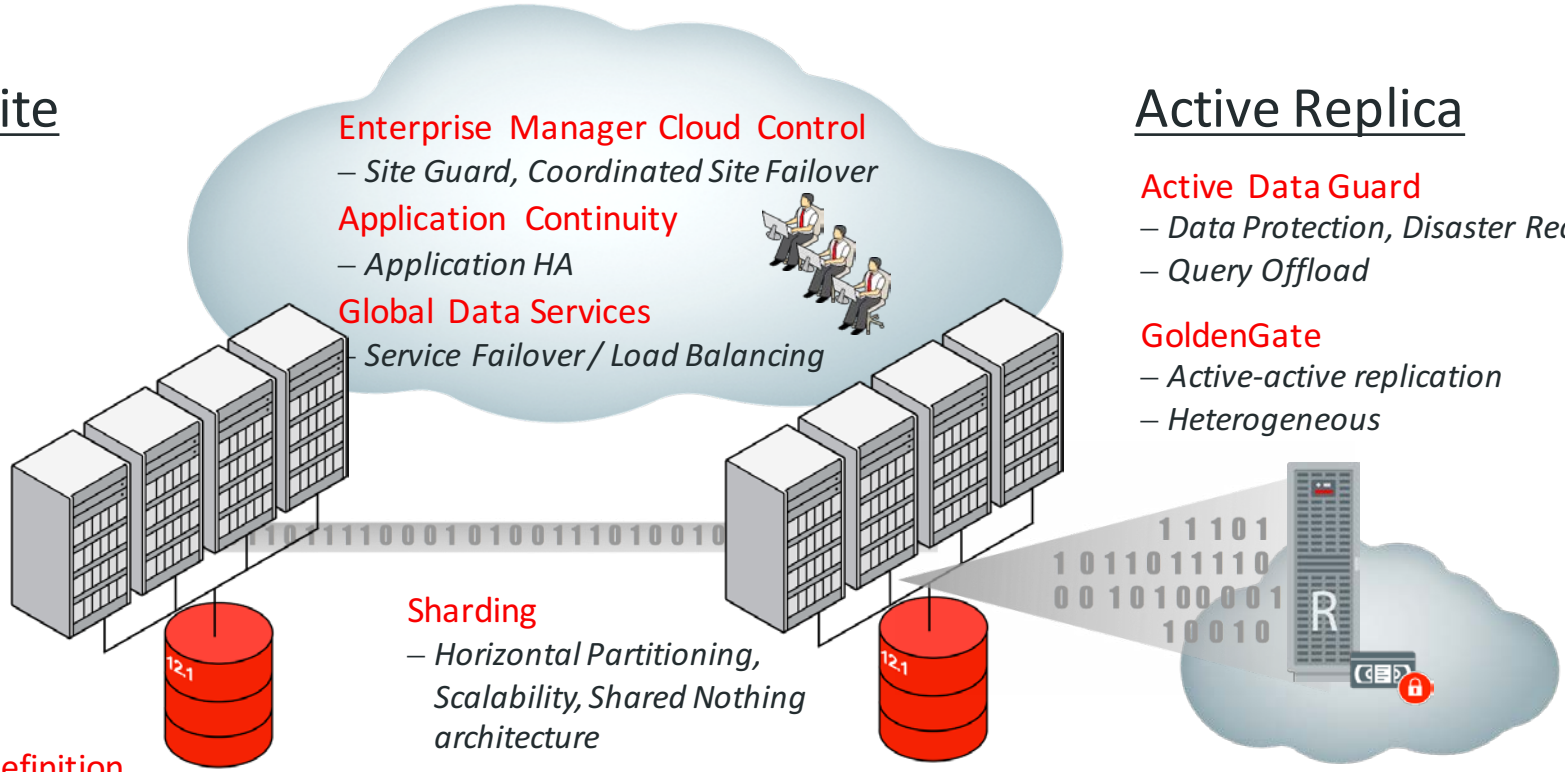
Oracle Maximum Availability Architecture (MAA)

Production Site

RAC
– *Scalability*
– *Server HA*

ASM
– *Local storage protection*

Flashback
– *Human error correction*



Enterprise Manager Cloud Control
– *Site Guard, Coordinated Site Failover*

Application Continuity
– *Application HA*

Global Data Services
– *Service Failover / Load Balancing*

Sharding
– *Horizontal Partitioning, Scalability, Shared Nothing architecture*

Active Replica

Active Data Guard
– *Data Protection, Disaster Recovery*
– *Query Offload*

GoldenGate
– *Active-active replication*
– *Heterogeneous*

Edition-based Redefinition, Online Redefinition, Data Guard, GoldenGate
– *Minimal downtime maintenance, upgrades, migrations*

RMAN, Oracle Secure Backup, Zero Data Loss Recovery Appliance
– *Backup to disk, tape or cloud*

Program Agenda

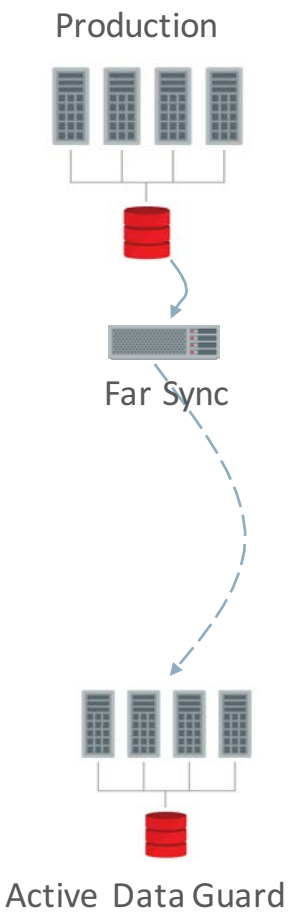
1. Active Data Guard
2. Sharding
3. Globally Distributed Database
4. Planned Downtime
5. Autonomous Database and Cloud

Active Data Guard is Critical Part of MAA

- Automatic failover in case of site failure (using Fast-Start Failover)
  - Human involvement will slow failover time (increase RTO)
- Protects against data and storage corruptions
- Automatically repairs block corruptions
- Active-active architecture automatically ensures that standby is functioning and failover-ready
- Minimum downtime rolling upgrades (using transient logical standby)

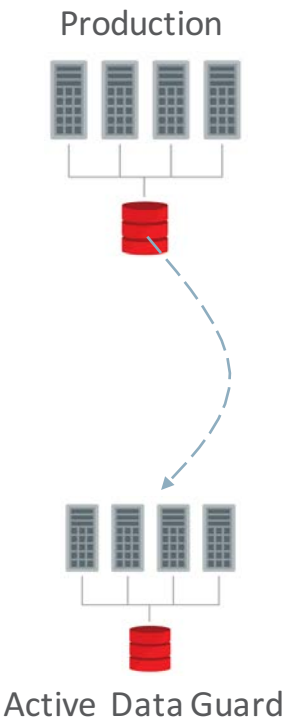
Active Data Guard features in 12.1 and 12.2

- **Zero Data Loss**
  - **Far Sync** supports zero data loss over long distances
  - **Fast Sync** supports low-latency commits for zero data loss
  - **Real-time Cascade** supports near-zero data loss even for cascaded configurations
- **Active Standby**
  - **In-memory DB on Active Data Guard** allows in-memory tables on the standby
  - Diagnostic and tuning pack (**AWR**), **SQL Plan Management** on Active Data Guard
  - **Global Temporary Tables** on Active Data Guard
  - **Global Data Services** load balances connections among primary and Active Data Guard instances
- **Fast Failover**
  - **Preserve sessions** on Active Data Guard during failover or switchover
  - **Multi-instance parallel redo apply** uses all instances in a RAC standby to apply redo
- ***… and more …***



Active Data Guard features in Oracle Database 18c

- Fast, automatic nologging loads with Active Data Guard
  - Customers use nologging loads to avoid overhead of generating redo
  - In Oracle 18*c*, the primary ships data blocks to the standby as blocks are loaded to the primary
  - Available on Engineered Systems and Oracle Cloud
- Data Guard Broker provides validation and proactive health checks. Used by Oracle Cloud to detect and fix problems before they occur
- `CREATE GLOBAL TEMPORARY TABLE` on Active Data Guard
  - Active Data Guard 12c can update global temporary tables, but the tables still needed to be created at the primary. In 18c, the standby can create its own temporary tables



Active Data Guard Features in Oracle Database 18c

Simplifying Security Management for Your Data Guard Configuration

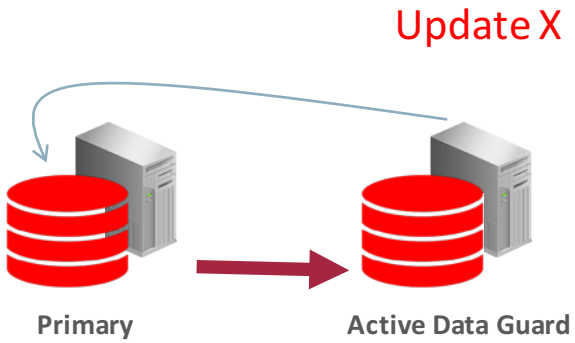
- Automatically update password file on standbys (12.2)
  - Changing Admin password on primary automatically updates standbys'
- Zero-downtime, Standby-first, encryption
  - First encrypt tablespaces on standby, switchover, then encrypt on the old primary
- New DG redo authentication protocol - uses SSL certificate for 'redo_transport_user'
- Track login failures across all databases in a Data Guard configuration. Logins anywhere will be denied when the max login count is reached

ORACLE

Updates on Active Data Guard using DML Redirection

NEW IN 19c

- Previously, you can only update Global Temp Tables on Active Data Guard. With 19c, you can update regular tables
- Updates on Active Data Guard:
  1. Update will be redirected to the primary
  2. Primary makes update, generates & sends redo for that update to all standbys
  3. Active Data Guard session sees the update in redo apply and resumes
- Preserves ACID properties for ADG session
  - Redirected update only visible to session before commit; visible to all sessions after commit
- For "Mostly Read, Occasional Updates" applications – for example, recording user logins for auditing purposes
- Enabled by **`ADG_REDIRECT_DML`** at system or session level



Program Agenda

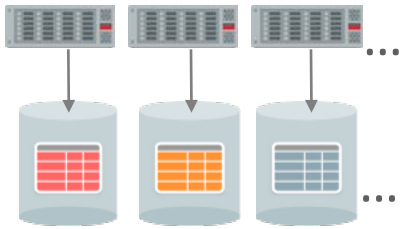
1. Active Data Guard
2. Sharding
3. Globally Distributed Database
4. Planned Downtime
5. Autonomous Database and Cloud

Oracle Database Sharding

RAC and Data Guard meet needs of most applications while preserving application transparency

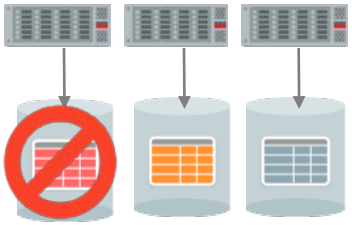
Some large scale applications want to shard across farm of independent databases *and are willing to modify application*

Linear Scalability



- Data automatically partitioned across pool of independent databases (shards)
- Add shards online to increase throughput
- Data, users and transactions scale linearly
- Single- and multi-shard queries

Fault Tolerant

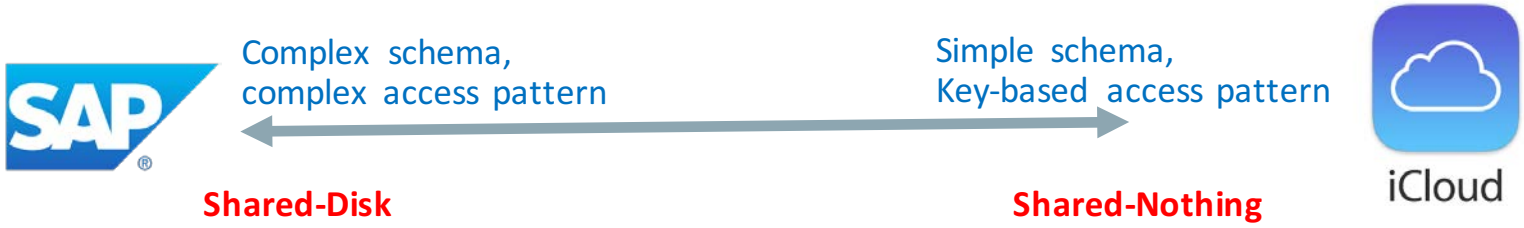


- Outage or slowdown of a shard has no impact on availability of other shards
- Shard-level HA automatically configured using database replication (RAC optional)
- Shards can be independently patched and upgraded without affecting other shards

Geographic Distribution

- Flexible shard organization - consistent hash, range, list, or composite (range-hash, list-hash)
- Data placement for availability, disaster recovery, performance, or to meet regulatory requirements

Oracle Provides Continuum of Shared-Disk and Shared-Nothing



- Shared-Disk and Shared-Nothing have their respective strengths
- Application transparency vs easy scalability and fault isolation
  - SAP vs iCloud
- Most databases only support shared-nothing because shared-disk (RAC) is hard
- Oracle offers both ends of the spectrum (RAC and Sharding)
- 18c introduces mid-point architecture that provides interesting capabilities
  - RAC Sharding

RAC Sharding

Higher performance for shard-aware RAC applications

- Affinitizes shards to RAC instances
  - Requests that specify sharding key will be routed to the RAC instance that logically holds the shard
  - Better cache utilization and reduced block pings across instances
- Requests that don't specify sharding key still work transparently
- Gives Sharded Database performance with minimal application changes
  - Just add sharding key to the most performance intensive operations
- Available on Engineered Systems and Oracle Cloud

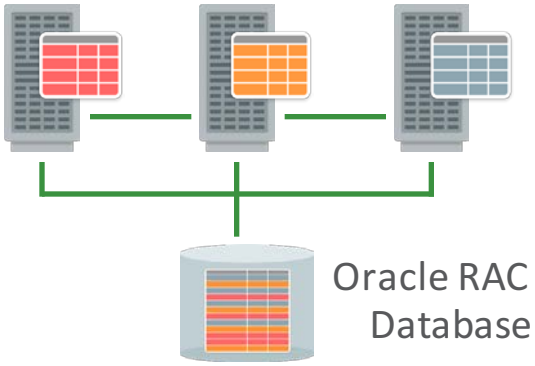
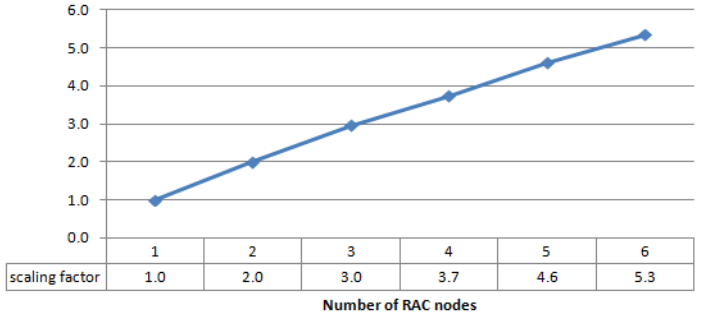


Figure 4 - RAC DB Performance with RAC Affinity



	1	2	3	4	5	6
scaling factor	1.0	2.0	3.0	3.7	4.6	5.3

Number of RAC nodes

Oracle Sharding | 19c Features

NEW IN 19c

- Support multiple PDB-shards in the same CDB
  - A PDB can be sharded across multiple CDBs
  - Allows consolidation of sharded databases
- Scale-out multi-shard query coordinators
  - Can configure shard catalog's ADG standbys to act as multi-shard query coordinators
  - Great for analytic workloads that need to query multiple shards
- Multiple Table Families
  - Allows a sharded database to support multiple table families, each of which can be sharded with a different sharding key
- Generation of globally unique sequence numbers across shards
  - For non-primary key columns (e.g. order_id)

ORACLE


Oracle Sharding POC
@ State Grid Corporation of China

国家电网公司
STATE GRID
CORPORATION OF CHINA

State Grid Corporation of China (SGCC)

- Largest electric utility in the world
  - Provides power to over 1.1 billion people across 88% of China
  - Global presence with subsidiaries in Philippines, Brazil, Portugal, Australia and Italy
  - 1.72 million employees, $585 billion in assets, $348.9 billion revenue (2017)
  - Ranked #2 in Fortune Global 500 (2nd largest company globally by revenue)
- Company mission
  - As a super-large state-owned enterprise crucial to national energy security and economic lifeline, SGCC has a mission to provide safer, cleaner, and more economical and sustainable power supply

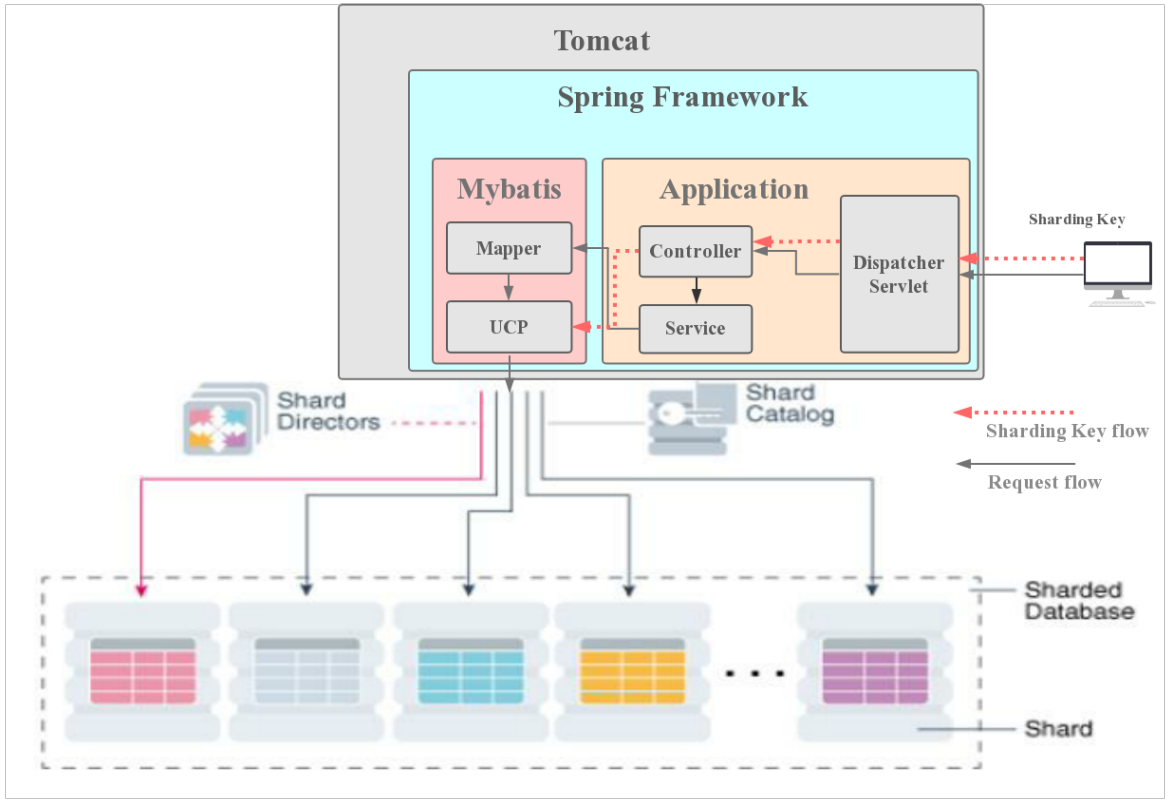
国家电网公司
STATE GRID
CORPORATION OF CHINA

Challenges

- SGCC is adopting Microservice architecture for the growing business
  - Microservice architecture provides resiliency through loose coupling and flexibility to meet growing and changing business needs
- Microservice architecture in turn requires a highly scalable, loosely-coupled, and ultra-reliable distributed database
- State Grid wanted to prove that their business-critical ordering service system can leverage Oracle Sharding to meet these requirements
  - Ordering service currently has 200-300 million users distributed across 26 subsidiaries. Goal is to build centralized ordering service
- To minimize the impact on their development processes, the new distributed database must support their current development framework based on Spring+Mybatis

国家电网公司
STATE GRID
CORPORATION OF CHINA

Architecture



- Schema sharded using Order Number
- Client uses Spring-MyBatis Java persistence framework using Oracle UCP as Data Source
- To avoid modifying MyBatis source code, local thread was added to pass the Sharding key from Controller to the UCP Data Source directly – allowing high performance direct access to shards

国家电网公司
STATE GRID
CORPORATION OF CHINA

Results

- Successfully leveraged Oracle Sharding in Spring+MyBatis Framework without modifying any application code
  - Good example for how to integrate Sharding API into Java persistence frameworks in general
- In the future, Oracle user-defined Sharding also allows data to be stored close to State Grid's customers distributed in different locations
- Oracle Sharding provides the highly scalable, loosely-coupled, and ultra reliable distributed database required by new generation microservice-based applications

Oracle Sharding Strategy

- Superset of all sharding architectures (SQL and NoSQL)
  - Pluggable replication architecture
    - Supports all the replication models (master-slave, active-active, ring, etc.)
  - Multiple sharding techniques
    - Consistent hash, range, list, composite
  - Tunable consistency
    - Within and across shards. Plus uniquely, high-performance strong consistency across shards
  - Continuum of shared-disk and shared-nothing capabilities
  - Standards-based (SQL and Java 9)
- Plus unique capabilities around **data sovereignty** and **multi-cloud**

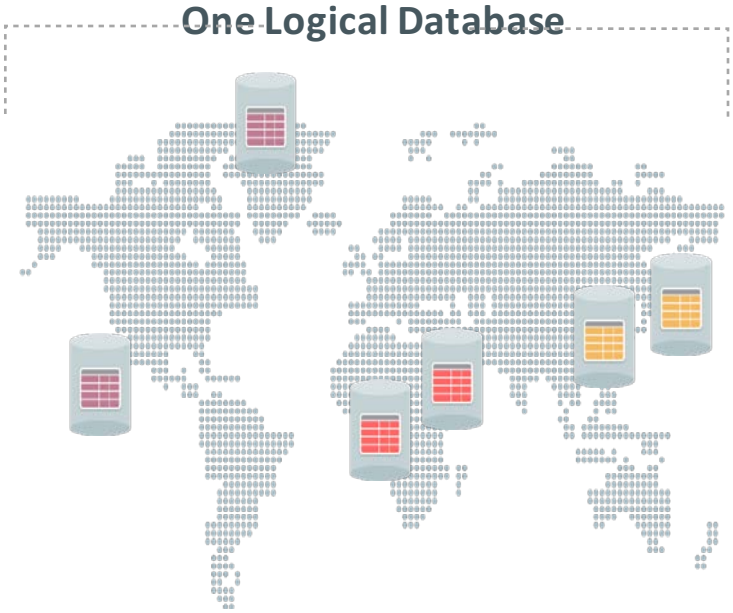
Program Agenda

1. Active Data Guard
2. Sharding
3. Globally Distributed Database
4. Planned Downtime
5. Autonomous Database and Cloud

Oracle Sharded Database is a Globally Distributed Database

Why would you want to do this?

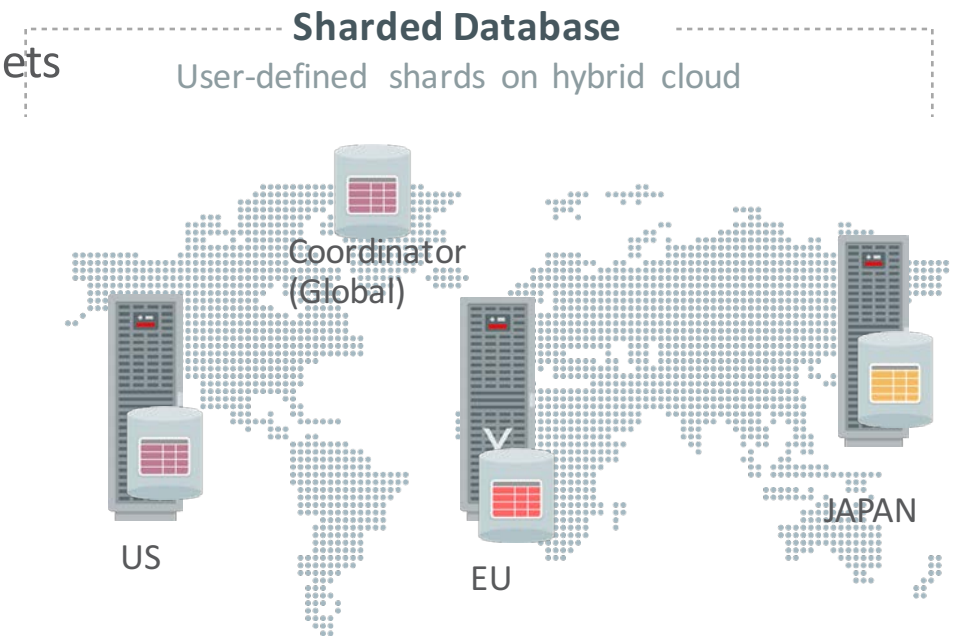
- Data sovereignty – data for country must be stored & replicated in-country
- Query data across all regions
- Disaster Recovery – primary and standby(s) can be far apart
- Low latency for reads – read from nearest replica; with load balancing and failover
- Low latency for writes if data is partitioned properly. E.g., locate European data in European data center
- Single logical database – application & DBAs see one database



Data Sovereignty (Pattern 1) and Multi-Cloud

User Defined Sharding

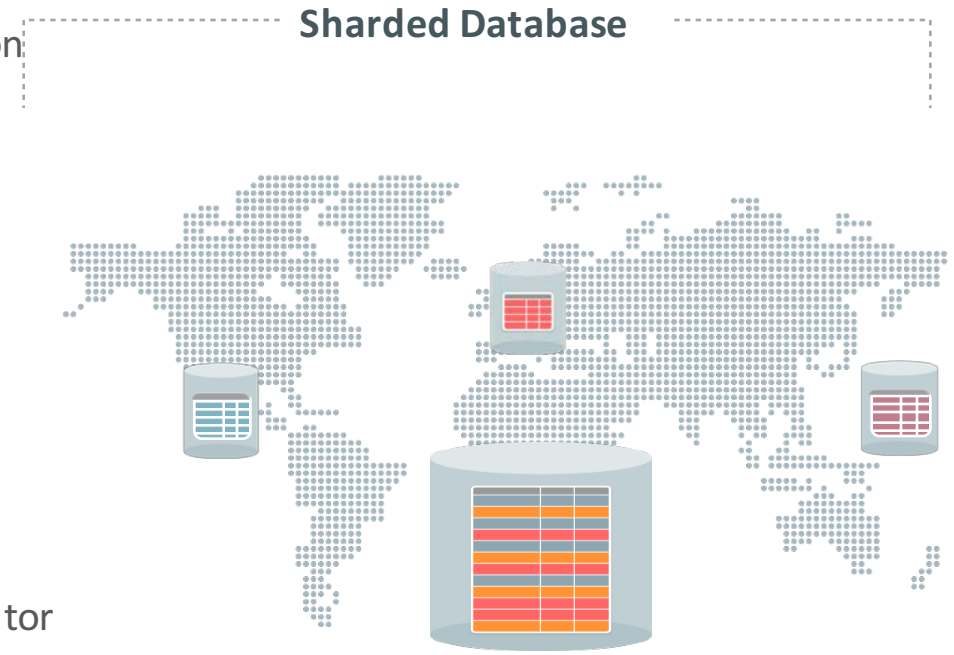
- Data Sovereignty requires data to be kept in the country
- 18c User-Defined Sharding (LIST or RANGE) lets you create a shard for each country
  - Add COUNTRY column to each table
- Applications within the country can directly access shard without a sharding key
- Query across all countries by connecting to Coordinator and issue multi-shard queries
- Other Use Cases
  - Multi-Cloud (shard across 2 or more clouds)
  - Hybrid Cloud and Cloud Bursting
    - Some shards on premises; other shards in the cloud



Data Sovereignty with Global Single Instance (Pattern 2)

Most data centralized with some data sharded by country

- Having global single instance is beneficial
  - Strong data consistency by eliminating duplication
- Centralize bulk of your data
  - Keep only subset (e.g., CUSTOMER, and other Personally Identifiable Information PII) in the countries of origin
  - Can query and run reports across all data (centralized and country-specific)
- With Oracle Sharding
  - Create all tables in a central shard catalog
  - Place certain data across country-specific shards
  - See all the data using multi-shard query coordinator
  - Application always connect to coordinator database



How Does Oracle Sharding Compare with Alternatives?

Real and Fake News

- Google Cloud Spanner *as claimed*:
  - Only enterprise-grade, globally-distributed, and strongly consistent database service for the cloud
  - Combines benefits of relational database structure with non-relational horizontal scale
  - Query data using familiar, industry-standard ANSI 2011 SQL
  - High-performance transactions and strong consistency across rows, regions, and continents with an industry-leading 99.999% availability SLA.. and enterprise grade security
- Real News: strong consistency, relational model, ANSI SQL, and transactions are cool!

Does This SQL Run on Spanner?

Transfer $500 from Savings Account to Checking Account

```
UPDATE savings_accounts
  SET balance = balance - 500
  WHERE cust_id = 12345 AND account = 3209;
```

Decrement Savings Acct
(Savings table constraint: amount > $100)

```
UPDATE checking_accounts
  SET balance = balance + 500
  WHERE cust_id = 12345 AND account = 3208;
```

Increment Checking Acct

```
INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (13, cust_id, account, balance, -500)
    FROM savings_accounts WHERE cust_id = 12345 AND account = 3209;

INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (12, cust_id, account, balance, 500)
    FROM checking_accounts WHERE cust_id = 12345 AND account = 3208;
```

Journal the transfer operations
(tran_cod 13 – transfer from
tran_code 12 – transfer to)

```
COMMIT WORK;
```

End transaction

Oracle SQL vs Spanner Program (Application Transparent! NOT)

```
/* transfer $500 from savings to checking */
/* all tables are sharded by cust_id */
/* savings table constraint: amount > $100 */
UPDATE savings_accounts
  SET balance = balance - 500
  WHERE cust_id = 12345 AND account = 3209;
UPDATE checking_accounts
  SET balance = balance + 500
  WHERE cust_id = 12345 AND account = 3208;

/* log new balances (read your own writes) */
/* tran_code = 13 – transfer from */
INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (13, cust_id, account, balance, -500)
    FROM savings_accounts WHERE cust_id = 12345 AND account = 3209;
/* tran_code = 12 – transfer to */
INSERT INTO journal (tran_code, cust_id, account, balance, change_amount)
  SELECT (12, cust_id, account, balance, 500)
    FROM checking_accounts WHERE cust_id = 12345 AND account = 3208;

COMMIT WORK;
```

```
static void writeWithTransaction(DatabaseClient dbClient) {
 dbClient
   .readWriteTransaction()
   .run(
     new TransactionCallable<Void>() {
      @Override
      public Void run(TransactionContext transaction) throws Exception {
       // Transfer $500 from SAVINGS to CHECKING. We do the transfer in a transaction to
       // ensure that the transfer is atomic.
       Struct row =
         transaction.readRow("SAVINGS_ACCOUNTS", Key.of(12345, 3209),
                             Arrays.asList("SavingsRow"));
       long savingsBalance = row.getLong(2);
       // Transaction will only be committed if sufficient funds (>$100) constraint is met
       // at the time of commit. Otherwise it will be aborted and the callable will be rerun
       // by the client library.
       if (savingsBalance >= 600) {
        long checkingBalance =
          transaction
            .readRow("CHECKING_ACCOUNTS", Key.of(12345, 3208),
                     Arrays.asList("CheckingsRow"))
            .getLong(2);
        long transfer = 500;
        checkingBalance += transfer;
        savingsBalance -= transfer;
        transaction.buffer(
          Mutation.newUpdateBuilder("CHECKING_ACCOUNTS")
            .set("cust_id")
            .to(12345)
            .set("account")
            .to(3208)
            .set("CheckingsRow")
            .to(checkingBalance)
            .build());
        transaction.buffer(
          Mutation.newUpdateBuilder("SAVINGS_ACCOUNTS")
            .set("cust_id")
            .to(12345)
            .set("account")
            .to(3209)
            .set("SavingsRow")
            .to(savingsBalance)
            .build());
        // Log new balances, cannot read your own writes
        // mutations: checkingBalance, savingsBalance
        // must fit in memory!
        transaction.buffer(
          Mutation.newInsertBuilder("JOURNAL")
            .set("tran_code")
            .to(13)
            .set("cust_id")
            .to(12345)
            .set("account")
            .to(3209)
            .set("balance")
            .to(savingsBalance)
            .set("change_amount")
            .to(transfer)
            .build());
        transaction.buffer(
          Mutation.newInsertBuilder("JOURNAL")
            .set("tran_code")
            .to(12)
            .set("cust_id")
            .to(12345)
            .set("account")
            .to(3208)
            .set("balance")
            .to(checkingBalance)
            .set("change_amount")
            .to(transfer)
            .build());
       }
       return null;
      }
     });
} // writeWithTransaction
```

Oracle Sharded Database is a Globally Distributed Database

- Single logical database with replicas across data centers and regions
- Applications sees one logical database
  - Connect to a single 'service' and be routed to the right replica
  - Routing takes into account: network latency, replica response time, and data held by each replica
  - Handles failover; i.e., route to a good replica if original is unavailable
  - Handle add and drop of replicas (that may contain disjoint data)
- Manage the replicas as a single database
  - *Administrators do not need to maintain their spreadsheet of nodes, who replicates to whom, and perform admin operations N times for each replica*
  - Global metadata and view of the physical databases (replicas), data they contain, replication topology, etc.
  - Automatic configuration of replication
  - Automatic propagation of management operations (DDLs, passwd files, …) to all replicas
  - Rebalance data and synchronize state as nodes are added/dropped
- Oracle Sharding does all this, while support data sovereignty and multi-cloud

Program Agenda

1. Active Data Guard
2. Sharding
3. Globally Distributed Database
4. **Planned Downtime**
5. Autonomous Database and Cloud

Planned Maintenance is Key to Uptime

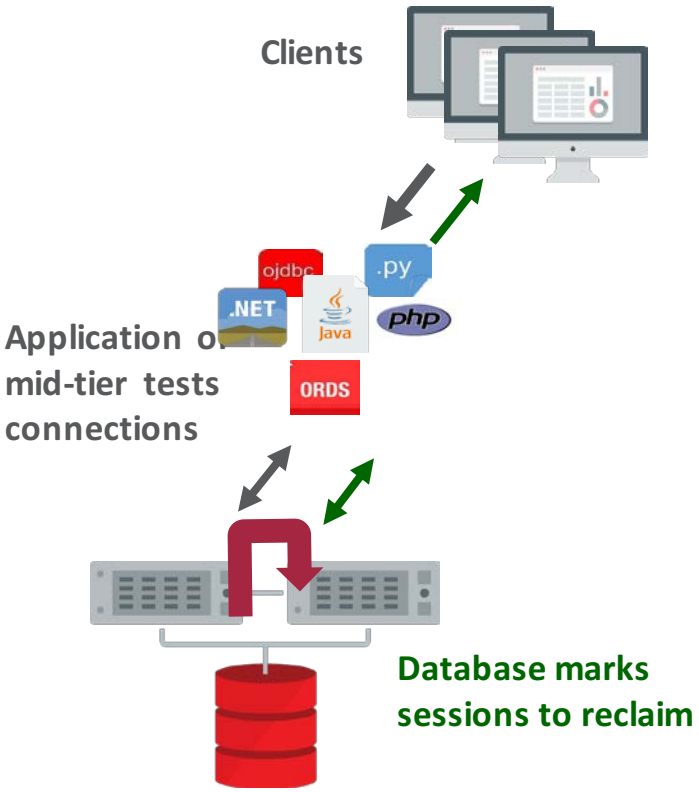
- Unplanned downtime is rare with well-run systems
  - Hardware / software failures are automatically handled by MAA
- Key to best uptime is reducing planned downtime
  - Make application schema changes online
  - Apply patches and upgrades online or rolling
- Focus on automation to make them simple
- Use Application Continuity to hide downtime from applications

Patching Improvements

- Zero-Downtime Oracle Grid Infrastructure Patching
  - Patch Oracle Grid Infrastructure without interrupting database operations.
  - Patches are applied out-of-place and in a rolling fashion with one node being patched at a time while the **database instance(s) on that node remain up and running**.
  - Supported for Oracle RAC and RAC One Node clusters with two or more nodes.
- OJVM is Oracle RAC rolling patching enabled with Oracle RAC 18c (18.4)
  - Non-Java services are available at all times.
  - Java services are available all the time, except for a ~10 seconds brownout.
    - No errors are reported during the brownout.

Applications see no errors during maintenance

Automatically reclaim Connections

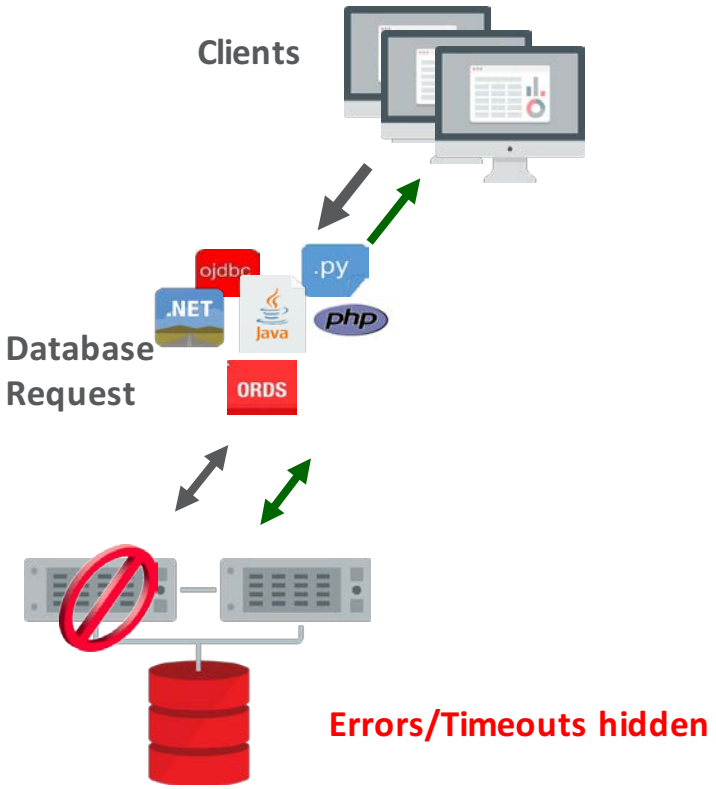


- When we shutdown an instance for maintenance, it is best if all apps release their connections
  - However, many applications do not explicitly disconnect, preventing us from shutting down instance
- 18c Database and Drivers reclaim connections where applications do not notice prior to maintenance
  - Example: End of Web requests, Transaction boundaries, Connection tests, Connection returned to pool, and Custom rules
- Applications that do not release their connections are failed over

Transparent Application Continuity

NEW IN 19c

Failover Applications that do not Release their Connections



- Sessions that do not release their connections must be killed. Transparent Application Continuity then restores session on another instance
- Key innovations
  - Track session state as it changes. This is verified during replay
  - Automatically enables and disables capture of replay state
    - Disables capture after commits since we do not replay committed transactions (or statements with side-effects)
    - Automatically re-enables
  - Automatically trims the client's replay history
  - Handles SYSDATE, sequences, …

ORACLE

Automated Transient Logical Standby (Oracle 12*c*)

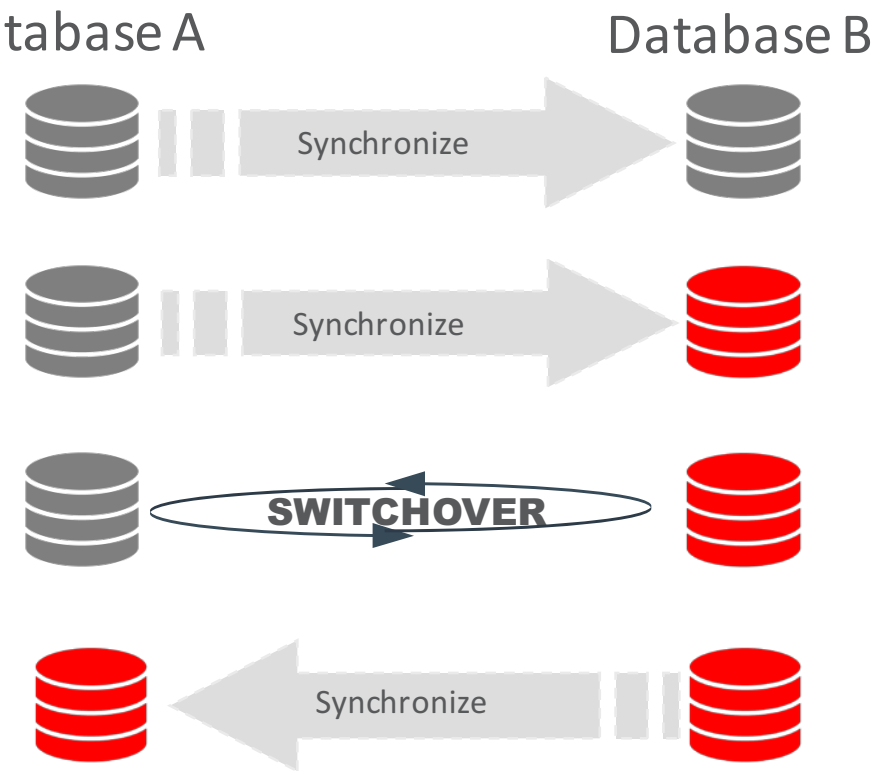
Active Data Guard Rolling Database Upgrades (DBMS_ROLLING)

DBMS_ROLLING.INIT_PLAN
DBMS_ROLLING.BUILD_PLAN
DBMS_ROLLING.START_PLAN

Do the upgrade (DBUA)

DBMS_ROLLING.SWITCHOVER

DBMS_ROLLING.FINISH_PLAN



Enhancements to DBMS_ROLLING (18c and 19c)

- DBMS_ROLLING requires supporting logical replication of each datatype
  - All the common datatypes are fully supported during rolling upgrade
  - 18c and 19c added more: BigSCN, Long Identifiers, SDO_GEORASTER, TOPOLOGY, REF, top-level varrays attributes, SDO_RDF_TRIPLE_S, …
- New approach to handle the few remaining unsupported types
  - Key observation: unsupported datatypes are ok if not *updated* during upgrade
  - In 19c, we block updates to these unsupported types (and plsql) during a rolling upgrade
    - Feasible as customers can disable certain reports and batch jobs during upgrades
  - Allows databases containing unsupported types to use DBMS_ROLLING

Zero Downtime Upgrade (ZDU)

Active Data Guard and GoldenGate

- Automatic setup/teardown and configuration of Active Data Guard (Transient Logical Standby) or GoldenGate
- Works from 11.2.0.4

Zero-Downtime Upgrade

Fully Automated Workflow

- Clone
- Switch Over
- Upgrade
- Sync Up
- Switch Back

Robust, Fast & Space Efficient

Robust

- Resume-able after failures
- Revertible for fast rollback / fall back.

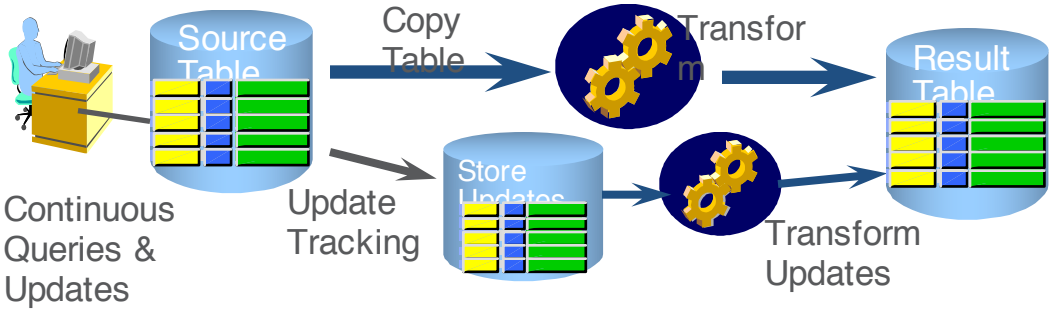
Fast & Space Efficient

- Fast cloning with Snapshots for databases hosted on ACFS
- Full Database copy also supported

ORACLE

Online Operations

Online Redefinition Improvements



- DBMS_REDEFINITION allows you to reorganize and redefine tables online
  - Add/drop/rename columns, switch physical storage structures, reorg table, transform data … online
- DBMS_REDEFINITION enhanced in 12.2 for even the largest, busiest databases:
  - Full restartability (resume at point of failure)
  - Entire redefinition process runs without acquiring Exclusive DDL lock
  - Maintain dependent MVs during redefinition
  - V$online_redef for progress monitoring
  - Support for Binary XML storage changes, BFILE, invisible columns
- 18c, 19c added support for tables with bitmap join indexes, hybrid partitioning, row archival, and both spatial index and VPD policies

Online Operations

Don't bring down Oracle to make schema changes!

11.2 & Prior	**Create index** online, **rebuild index** online, rebuild index partition online **Add Column**, Add Constraint enable novalidate
12.1	Online **move partition** **Drop index** online **Set unused** column online, **alter column visible**/invisible, alter index unusable online, alter index visible/invisible alter index parallel/noparallel
12.2	**Alter table move online for non-partitioned tables** **Alter table from non-partitioned to partitioned online** **Alter table split partition online** **Create table for exchange (usable for online partition exchange)** **Move/merge/split partition maintenance operations can now do data filtering**
18.1	Alter table modify partitioned table to a **different partitioning method** (e.g., hash to range) Alter table **merge partition/subpartition** online

Online Operations

Even better with 19c

NEW IN 19c

- 18c and 19c enhanced to reduce impact of online DDLs on running systems
- Improved concurrency between multiple partition maintenance operations
- 18c and 19c does intelligent, fine-grained cursor invalidation (avoiding blanket cursor invalidation)
  - Fine-grained invalidation (treat DML and SELECT cursors differently)
  - Rolling invalidation (existing, possibly sub-optimal cursors can still be used)
  - CREATE/DROP/ALTER INDEX, TRUNCATE TABLE, Partition Maintenance Operations, DDLs on unusable and invisible indexes, …

ORACLE

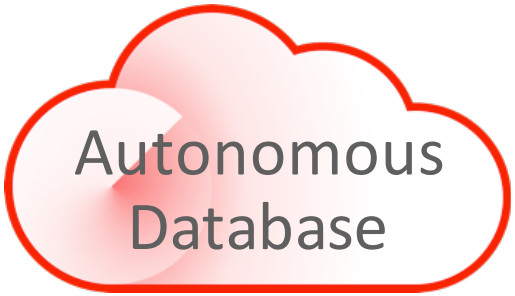
Program Agenda

1. Active Data Guard
2. Sharding
3. Globally Distributed Database
4. Planned Downtime
5. Autonomous Database and Cloud

Oracle Autonomous Database – Ingredient 1

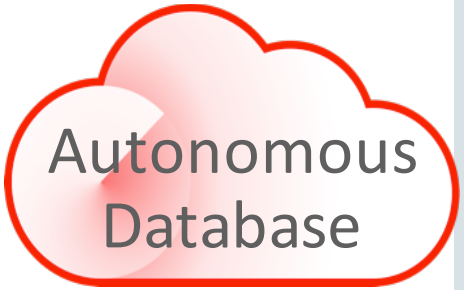
Maximum Availability Architecture (MAA)

- MAA adopted as a standard by many companies large and small
  - All 4 Top US banks (JPMC, BoA, Citibank, Wells Fargo) adopted MAA
- MAA is Oracle's *best* recommended architecture
  - If you want to a standardized architecture – pick what that Oracle recommends and that other credible customers have adopted
  - These customers standardized on MAA only after due diligence
- *Leverage the collective know-how, experience of Oracle and other companies such as JPMC, BoA, PayPal, …*



Oracle Autonomous Database – Ingredient 2
Exadata

- Exadata standardizes the hardware and configuration
- Standardization across customers brings greater reliability. Fixes found by other customers directly help you be more stable
- Exadata is the only platform that can do this
  - *Same reliability as New York Stock Exchange and top banks worldwide*
- Exadata also provides unique availability advantages
  - In-memory query on Active Data Guard, RAC Sharding, Nologging loads with Active Data Guard, Active Data Guard multi-instance parallel redo apply…
- Even if there is *zero* unique capability, Exadata would still bring greater reliability because of standardization

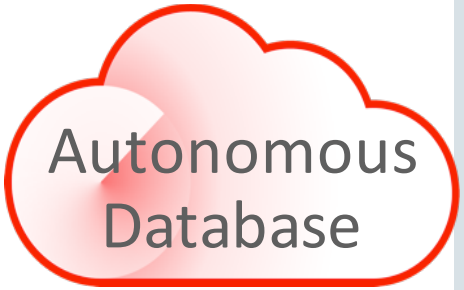


ORACLE

Oracle Autonomous Database – Ingredient 3
Cloud Management with Machine Learning

- MAA + Exadata provides a standard architecture, software, and hardware configuration
- The Autonomous Database software then makes this entire infrastructure self-managing and self-healing

- All 3 ingredients: MAA, Exadata, and Autonomous Database software combine to to uniquely provide 99.995% NX Availability



ORACLE


ORACLE®