Oracle® NoSQL Database

Administrator's Guide

Release 25.1
E85373-55
June 2025

Oracle NoSQL Database Administrator's Guide, Release 25.1

E85373-55

Contents

4 Administer

5 Reference

Preface

This document describes how to install and configure Oracle NoSQL Database (Oracle NoSQL Database).

This book is aimed at the systems administrator responsible for managing an Oracle NoSQL Database installation.

Conventions Used in This Book

The following typographical conventions are used within this manual:

Information that you are to type literally is presented in `monospaced` font.

Variable or non-literal text is presented in *italics*. For example: "Go to your *KVHOME* directory."

> **Note:**
>
> Finally, notes of special interest are represented using a note block such as this.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Introduction

The article in this section gives an introduction to Oracle NoSQL Database.

Introduction to Oracle NoSQL Database

Oracle NoSQL Database is a distributed, shared-nothing, non-relational database that provides large-scale storage and access to key/value, JSON, and tabular data. It can deliver predictable, low latencies to simple queries at any scale and is designed from the ground up for high availability.

Oracle NoSQL Database's shared-nothing architecture allows it to scale horizontally to meet exceptionally high throughput demands while delivering predictable low latencies. Oracle NoSQL Database requires minimal administration and contains many ***self-healing*** features that enable it to remain ***always-on*** during failures- hardware failures, network partition failures, or even entire data center disasters.

Oracle NoSQL Database offers highly flexible deployment and various methods to access the data store from your application. For applications that require an embedded, ultra-low latency, zero administration database, it can be directly embedded into a Java application. In this deployment scenario, applications can start and stop the database using APIs. When using the Java Direct Driver , applications can read and write data to the database. In most scenarios, Oracle NoSQL is deployed on a cluster of commodity computers connected by a high-speed network. In this deployment scenario, applications must choose a programming language SDK to communicate with the Oracle NoSQL Database cluster. Oracle NoSQL Database offers two types of language SDKS:

1. **Direct driver:** This type of SDK will connect directly to every Oracle NoSQL node in the cluster using TPC/IP. Hence, care must be taken to ensure a network route between the application and every Oracle NoSQL node in the database cluster. Currently, the only supported programming language for direct drivers is Java .
2. **Standard:** This type of SDK will connect to the database using the HTTP protocol via the Oracle NoSQL HTTP proxy. Since standard SDKs use HTTP, you need only ensure a network route between your application code and the load balancer if using one, or between the application and the HTTP proxy if not using a load balancer.

Oracle NoSQL Database supports many of the most popular programming languages and frameworks with idiomatic language APIs and data structures, giving your application language native access to data stored in NoSQL Database. It currently supports the following programming languages and frameworks: Javascript (Node.js), Python, Java, Golang, C#/.NET, Rust, and Spring Data. You can also navigate the database as you develop your code with plugins for one of the following supported integrated development environments: Visual Studio Code , IntelliJ, or Eclipse .

2

Install and Upgrade

The articles in this section include steps to install a new Oracle NoSQL Database or upgrade the software of your Oracle NoSQL Database deployment.

Installing Oracle NoSQL Database

This article describes the process for installing Oracle NoSQL Database. If you already know the number of Storage Nodes you will use in your data store, follow the subsequent instructions. If you need help with estimating the resources required for installing the database and configuring your data store, see Initial Capacity Planning and then follow the subsequent instructions outlined in this topic. The capacity planning will help you estimate the number of Storage Nodes you need to use to install the software. You can come up with an estimate based on your application's requirements and the characteristics of the hardware available to you. The Oracle NoSQL Database will make the best use of the Storage Nodes you provide.

- Installation Prerequisites
- Installation

Installation Prerequisites

Make sure that you have Java SE 11 or later installed on all of the Storage Nodes that you are going to use for the Oracle NoSQL Database installation. From your Linux operating system, run the following command to verify the existing Java version in your Linux machine:

```
java -version
```

> **Note:**
>
> Oracle NoSQL Database is compatible with Java SE 11 (64 bit) or later versions. It is tested and certified against Oracle Java SE 17 (64 bit). It is recommended that you upgrade your systems to the latest Java releases to take advantage of all bug fixes and performance improvements. See Release Notes - Overview for more details on Java requirements.

Linux is the officially supported platform for Oracle NoSQL Database. Running the Oracle NoSQL Database requires a 64-bit JVM. You do not necessarily need root access on each Storage Node for the installation process. Be sure that the `jps` tool is working. Installing the JDK makes the `jps` tool available for use by the Storage Node Agent (SNA). The `jps` tool can be used to verify the Oracle NoSQL Database processes that are currently running in your Storage Node.

If the JDK is installed correctly, the output from invoking jps should list at least one Java process (the jps process itself). Use this command to verify successful installation of java in your Linux machine.

```
% jps
```

Output:

```
16216 Jps
```

> **Note:**
>
> You must run the command listed above as the same OS user who will run the Oracle NoSQL Database SNA processes.

Finally, make sure that each of the Storage Nodes is running some sort of reliable clock synchronization. Clock synchronization is necessary for timestamp continuity and synchronized coordination between storage nodes. Generally, a synchronization delta of less than half a second is required. Network Time Protocol `(ntp)` is sufficient for this purpose.

Installation

Before you install Oracle NoSQL Database, decide on the directories to store the various database package files and to store data. Set the following environment variables with the appropriate directory path.

- – `$KVHOME` - This is the directory to store all the Oracle NoSQL Database package files (libraries, Javadoc, scripts, and so forth). It is recommended that you use the same directory path for `$KVHOME` on each of the Storage Nodes in the installation. To make future software upgrades easier, adopt a convention for `$KVHOME` that includes the release number. For example, use a `$KVHOME` location such as `/var/kv/kv-M.N.O`, where `M.N.O` represent the software release.major.minor numbers.
- – `$KVROOT` - This is the directory to store Oracle NoSQL Database data.

It is recommended that both the `$KVHOME` and `$KVROOT` directories are local to the Storage Node, and not on a Network File System.

> **Note:**
>
> Use different directories for `$KVHOME` and `$KVROOT`. An example is shown below.
>
> ```
> export $KVHOME=$HOME/nosql/kv-24.1.11
> export $KVROOT=$KVHOME/kvroot
> ```

Steps to install the Oracle NoSQL Database:

1. Download the Oracle NoSQL Database bundle. You can download either Community Edition or Enterprise Edition software.

- **Community Edition:** Oracle NoSQL Database Community Edition (CE) software is licensed pursuant to the Apache 2.0 License (Apache 2.0).
- **Enterprise Edition:** Oracle NoSQL Database Enterprise Edition (EE) software is licensed pursuant to the Oracle commercial license.

To understand the difference between editions, see NoSQL Database Option Differences. If you have more than one Storage Node, copy the downloaded software to each of the Storage Nodes.

2. Extract the contents of the Oracle NoSQL Database package (`kv-M.N.O.zip` or `kv-M.N.O.tar.gz`) to `$KVHOME`. If `$KVHOME` resides on a shared network directory (which is not recommended) you need to only unpack it once. If `$KVHOME` is local to each Storage Node, unpack the package on each Storage Node. Unzipping the package installs the Oracle NoSQL Database.

   ```
   unzip kv-ee-24.1.11.zip
   ```

3. Set the appropriate values for`$KVHOME` (where you have unzipped the Oracle NoSQL Database package) and `$KVROOT`.
   Example:

   ```
   export $KVHOME=$HOME/nosql/kv-24.1.11
   export $KVROOT=$KVHOME/kvroot
   ```

4. Verify the software installation using the following command:

   ```
   java -Xmx64m -Xms64m -jar $KVHOME/lib/kvclient.jar
   ```

   You should see output that looks like this:

   ```
   24.1.11 2024-04-05 21:25:44 UTC
   Build id: 477e7f102ab4 Edition:Client
   ```

   where `24.1.11` is the database version number.

The software installation is complete. You can continue to configure your data store.

Upgrading an Existing Oracle NoSQL Database Deployment

This article describes how to upgrade your Oracle NoSQL Database software to a new release.

Upgrading a data store from an existing release to a new release can be accomplished one Storage Node at a time. This is because Storage Nodes running a mix of two releases are permitted to run simultaneously in the same data store. This allows you to strategically upgrade Storage Nodes in the most efficient manner. Installing new software requires that you restart each Storage Node.

Rolling Upgrade:

Upgrading a data store while the store remains online and available to clients is called rolling upgrade. A rolling upgrade is very useful, since downtime is undesirable in any system.

You can perform a rolling upgrade if the data store's replication factor is greater than two. With a replication factor greater than two, shards can maintain their majorities and continue reading

and writing data on behalf of clients. Meanwhile, you can restart and upgrade software on each Storage Node, one at a time.

Offline Upgrade:

Upgrading a system after shutting down the data store is called offline upgrade. In this case your data store is unavailable for the duration of the upgrade. Even if your data store can support a rolling upgrade, you may sometimes want to perform an offline upgrade, which involves these steps:

1. Shutting down all nodes.
2. Installing new software on each Storage Node.
3. Restarting each node.

Steps to upgrade an existing database (offline update or rolling update):

- General Upgrade Notes
- Preparing to Upgrade
- Steps to Upgrade - Examples
- Upgrading the XRegion Service Agent
- Upgrading the Oracle NoSQL Database Proxy
- Upgrading JDK on your Oracle NoSQL Database deployment

General Upgrade Notes

The upgrade information given below is generally true for all versions of Oracle NoSQL Database.

- Installing new software requires that each Storage Node be restarted.
- You do not need to invoke `makebootconfig` command while upgrading your data store.
- When your data store has more than one Storage Node, you can use the following command to understand the order of upgrade. For example,

  ```
  show upgrade-order
  s2
  s3
  ```

- When upgrading the software while the Storage Node is stopped, it is recommended to move the existing log files under `$KVROOT` and `$KVROOT/<storename>/log` to another directory.
- When upgrading your data store, place the new software version in a new `$KVHOME` directory on a Storage Node running the admin service. Here the new `$KVHOME` directory is referred to as `$NEW_KVHOME`. If the `$KVHOME` and `$NEW_KVHOME` directories are shared by multiple Storage Nodes (for example, using NFS), maintain both directories while the upgrade is in progress. After the upgrade is complete, you no longer need the original `$KVHOME` directory. Before removing the original `$KVHOME` directory you must modify the start up scripts on each Storage Node (e.g. `~/.bashrc` where you have defined `$KVHOME`) to modify the value of existing `$KVHOME` and replace it with the value of `$NEW_KVHOME` so that the Storage Node uses the new software.

Preparing to Upgrade

Oracle NoSQL Database supports upgrades from releases for the current year and prior few calendar years. For example, to upgrade a data store to the 24.x release, the data store must be running release 20.x or later. See Release Notes for specific information on the latest version of the software and the minimum older version needed for the upgrade.

Before beginning the upgrade process, create a backup of the store by making a snapshot. See Taking a Snapshot. During the upgrade process, you should not create any plans until the admin and managed services in the data store have been upgraded.

Plan to upgrade any application programs that use the Java Direct Driver after upgrading the service components. You need to re-link the application using the libraries of the new release of Oracle NoSQL Database.

Steps to Upgrade - Examples

Upgrading a data store from an existing release to a new release can be accomplished one Storage Node at a time. Different scenarios of upgrade are captured in the examples below. Each example shows how to upgrade your Oracle NoSQL Database software to a new release.

Example1: In this example you are upgrading your Oracle NoSQL Database from version **22.1.7** to version **24.1.11**. The data store has a capacity 1 and replication factor 1. Here `$KVHOME` is `/var/kv/kv-22.1.7`, `$KVROOT` is `/var/kv/kv-22.1.7/kvroot` and `$KVHOST` is the hostname of your Storage Node.

- Invoke the `runadmin` command to start the Admin command line interface (CLI) utility on the Storage Node. This command starts the `kv` prompt.

  ```
  java -jar $KVHOME/lib/kvstore.jar runadmin -host $KVHOST
  -port 5000 -security $KVROOT/security/root.login
  ```

  Use ping command to list the admin service and all managed services of your Storage Node.

  ```
  kv→ping
  ```

  Output:

  ```
  Pinging components of store mystore based upon topology sequence #154
  150 partitions and 1 storage nodes
  Time: 2024-04-05 06:13:33 UTC   Version: 22.1.7
  Shard Status: healthy:1 writable-degraded:0 read-only:0 offline:0 total:1
  Admin Status: healthy
  Zone [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  RN Status: online:1 read-only:0 offline:0
  Storage Node [sn1] on localhost:5000
  Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  Status: RUNNING   Ver:  2024-04-05 06:14:33 UTC
  Build id: 1473c1dac49c Edition: Enterprise
          Admin [admin1]          Status: RUNNING,MASTER
  ```

```
Rep Node [rg1-rn1]      Status: RUNNING,MASTER
sequenceNumber:330 haPort:5006 available storage size:1023 MB
```

- Verify that the Storage node are at or above the prerequisite software version needed to upgrade to the current version.

  ```
  kv-> verify prerequisite
  ```

  Output:

  ```
  Verify: starting verification of store mystore based upon topology
  sequence #154
  150 partitions and 1 storage nodes
  Time: 2024-04-05 09:27:43 UTC   Version: 22.1.7
  See localhost:/var/kv/kv-22.1.7/kvroot/mystore/log/mystore_{0..N}.log
  for progress messages

  Verify prerequisite: Storage Node [sn1] on localhost: 5000
  Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  Status: RUNNING   Ver: 22.1.7 2024-04-05 16:36:54 UTC  Build id:
  61b68fb1a3ec
  Edition: Enterprise    isMasterBalanced: true
  serviceStartTime: 2024-04-05 08:55:49 UTC
  Verification complete, no violations
  ```

- To upgrade your data store, you need to install the latest software in your Storage Node. See Install and verify your NoSQL Database installation for more details.
- Stop the Oracle NoSQL Database Storage Node Agent and services related to the root directory of the current Oracle NoSQL Database (`22.1.7`).

  ```
  java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop
  -root $KVROOT
  ```

- Restart the Storage Node using the updated software release(`24.1.11`). Here `$NEW_KVHOME` is `/var/kv/kv-24.1.11` and `$KVROOT` is `/var/kv/kv-22.1.7/kvroot`

  ```
  nohup java -Xmx64m -Xms64m -jar $NEW_KVHOME/lib/kvstore.jar
  start -root $KVROOT &
  ```

- Invoke the `runadmin` command to start the Admin command line interface (CLI) utility on the Storage Node which is now running the updated software release. This command starts the `kv` prompt.

  ```
  java -Xmx64m -Xms64m -jar $NEW_KVHOME/lib/kvstore.jar
  runadmin -port 5000 -host $KVHOST
  -security $KVROOT/security/root.login
  ```

- Verify the store configuration to check if the upgrade is completed successfully.

  ```
  kv-> verify configuration
  ```

Output:

```
Verify: starting verification of store mystore based upon topology
sequence #154
150 partitions and 1 storage nodes
Time: 2024-04-05 09:33:10 UTC   Version: 24.1.11
See localhost:/var/kv/kv-22.1.7/kvroot/mystore/log/mystore_{0..N}.log for
progress messages
Verify: Shard Status: healthy: 1 writable-degraded: 0 read-only: 0
offline: 0 total: 1
Verify: Admin Status: healthy
Verify: Zone [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
RN Status: online: 1 read-only: 0 offline: 0
Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on localhost: 5000
Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:25:44 UTC  Build id:
477e7f102ab4
Edition: Enterprise    isMasterBalanced: true
serviceStartTime: 2024-04-05 09:32:29 UTC
Verify:         Admin [admin1]
Status: RUNNING,MASTER  serviceStartTime: 2024-04-05 09:32:41 UTC
stateChangeTime: 2024-04-05 09:32:39 UTC
availableStorageSize: 2 GB
Verify:  rg1-rn1: Storage directory on rg1-rn1 is running low
[/var/kv/test1 size: 1 GB available: 1023 MB]
Verify:  Rep Node [rg1-rn1]
Status: RUNNING,MASTER sequenceNumber: 4,244 haPort: 5006
availableStorageSize: 1023 MB storageType: HD
serviceStartTime: 2024-04-05 09:32:52 UTC
stateChangeTime: 2024-04-05 09:32:56 UTC

Verification complete, 0 violations, 1 note found.
Verification note: [rg1-rn1]
Storage directory on rg1-rn1 is running low [/var/kv/test1
size: 1 GB available: 1023 MB].
```

Example 2: Error while directly upgrading Oracle NoSQL Database from a very old version to the current version.

In this example you want to upgrade Oracle NoSQL Database from version **19.5.9** to version **24.1.11**. Here `$KVHOME` is `/var/kv/kv-19.5.9` and `$KVROOT` is `/var/kv/kv-19.5.9/kvroot`.

The following example shows the error you encounter when you try to upgrade from version **19.5.9** to version **24.1.11**.

- To upgrade your data store, you need to install the latest software in your Storage Node. See Install and verify your NoSQL Database installation for more details.
- Stop the Oracle NoSQL Database Storage Node Agent and services related to the root directory of the current Oracle NoSQL Database (`19.5.9`).

```
java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop
-root $KVROOT
```

- Restart the Storage Node using the updated software release(`24.1.11`).

  ```
  nohup java -Xmx64m -Xms64m -jar $NEW_KVHOME/lib/kvstore.jar start
  -root $KVROOT &
  ```

  You get an output as shown below, which implies an error has occurred.

  ```
  [1]+  Exit 1 nohup java -Xmx64m -Xms64m -jar
  /var/kv/kv-24.1.11/lib/kvstore.jar start
  -root /var/kv/kv-19.5.9/kvroot
  ```

  Open the `nohup.out` file to view the error.

  ```
  vi nohup.out
  Failed to start SNA: The previous software version 19.5.9 does not satisfy
  the prerequisite for 24.1.11 which requires version 20.1.12 or later.
  ```

  To avoid this error, you need to upgrade the data store from 19.5.9 to any 20.*.* release and then upgrade it to 24.1.11.

Example 3: Upgrading a data store with more than one Storage Node.

In this example you are upgrading your Oracle NoSQL Database from version **22.1.7** to version **24.1.11**. The data store has a capacity 1 and replication factor 2. You have two Storage Nodes in your data store. Here `$KVHOME` is `/var/kv/kv-22.1.7`, `$KVROOT` is `/var/kv/kv-22.1.7/kvroot` and `$KVHOST` is the hostname of your first Storage Node.

- Invoke the `runadmin` command to start the Admin command line interface (CLI) utility on the Storage Node which is now running the existing software release(22.1.7). This command starts the `kv` prompt.

  ```
  java -jar $KVHOME/lib/kvstore.jar runadmin -host $KVHOST
  -port 5000 -security $KVROOT/security/root.login
  ```

- Use ping command to return information about the runtime entities of your data store.

  ```
  kv→ping
  ```

  Output:

  ```
  Pinging components of store mystore based upon topology sequence #156
  150 partitions and 2 storage nodes
  Time: 2024-04-05 15:21:02 UTC   Version: 22.1.7
  Shard Status: healthy: 1 writable-degraded: 0 read-only: 0 offline: 0
  total: 1
  Admin Status: healthy
  Zone [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  RN Status: online: 2 read-only: 0 offline: 0 maxDelayMillis: 1
  maxCatchupTimeSecs: 0
  Storage Node [sn1] on <XX>.com: 5000
  Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  Status: RUNNING   Ver: 22.1.7 2024-04-05 21:25:44 UTC  Build id:
  ```

```
477e7f102ab4
Edition: Enterprise isMasterBalanced: true serviceStartTime: 2024-04-05
10:29:33 UTC
        Admin [admin1] Status: RUNNING,MASTER  serviceStartTime:
2024-04-05 10:29:44 UTC
        stateChangeTime: 2024-04-05 10:29:42 UTC availableStorageSize: 2 GB
        Rep Node [rg1-rn1] Status: RUNNING,MASTER sequenceNumber: 4,259
haPort: 5006
        availableStorageSize: 1023 MB storageType: HD
        serviceStartTime: 2024-04-05 10:29:56 UTC
        stateChangeTime: 2024-04-05 13:29:29 UTC
Storage Node [sn2] on <XX>.com: 5000
Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 22.1.7 2024-04-05 21:25:44 UTC  Build id:
477e7f102ab4
Edition: Enterprise  isMasterBalanced: true  serviceStartTime: 2024-04-05
13:29:18 UTC
        Admin [admin2] Status: RUNNING,REPLICA serviceStartTime:
2024-04-05 13:29:24 UTC
        stateChangeTime: 2024-04-05 13:29:23 UTC availableStorageSize: 2 GB
        Rep Node [rg1-rn2] Status: RUNNING,REPLICA sequenceNumber: 4,259
haPort: 5006
        availableStorageSize: 99 MB storageType: HD
        serviceStartTime: 2024-04-05 13:29:25 UTC
        stateChangeTime: 2024-04-05 13:29:29 UTC delayMillis: 1
        catchupTimeSecs: 0
```

- Verify that the Storage nodes are at or above the prerequisite software version needed to upgrade to the current version.

  ```
  kv-> verify prerequisite
  ```

  Output:

  ```
  Verify: starting verification of store mystore based upon topology
  sequence #156
  150 partitions and 2 storage nodes
  Time: 2024-04-05 15:30:45 UTC   Version: 22.1.7
  See <XX>.com:/var/kv/kv-22.1.7/kvroot/mystore/log/mystore_{0..N}.log for
  progress messages
  Verify prerequisite: Storage Node [sn1] on <XX>.com: 5000
  Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  Status: RUNNING   Ver: 22.1.7 2024-04-05 21:25:44 UTC  Build id:
  477e7f102ab4
  Edition: Enterprise isMasterBalanced: true serviceStartTime: 2024-04-05
  10:29:33 UTC

  Verify prerequisite: Storage Node [sn2] on <XX>.com: 5000
  Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  Status: RUNNING   Ver: 22.1.7 2024-04-05 21:25:44 UTC  Build id:
  477e7f102ab4
  Edition: Enterprise    isMasterBalanced: true
  ```

```
serviceStartTime: 2024-04-05 13:29:18 UTC

Verification complete, no violations.
```

- To upgrade your data store, you need to install the latest software in all your Storage Nodes. In your first Storage Nodes (sn1), install the new version of the software. See Install and verify your NoSQL Database installation for more details.
- Stop the Oracle NoSQL Database Storage Node Agent and services related to the root directory of the current Oracle NoSQL Database (`22.1.7`).

```
java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop
-root $KVROOT
```

- Restart the first Storage Node using the updated software release(`24.1.11`). Here `$NEW_KVHOME` is `/var/kv/kv-24.1.11` and `$KVROOT` is `/var/kv/kv-22.1.7/kvroot`.

```
nohup java -Xmx64m -Xms64m -jar $NEW_KVHOME/lib/kvstore.jar start
-root $KVROOT &
```

- Invoke the `runadmin` command to start the Admin command line interface (CLI) utility on the Storage Node which is now running the updated software release. This command starts the `kv` prompt.

```
java -Xmx64m -Xms64m -jar $NEW_KVHOME/lib/kvstore.jar
runadmin -port 5000 -host $KVHOST -security
$KVROOT/security/root.login
```

- Verify the store configuration to check if the upgrade for the first Storage Node (sn1) is completed successfully.

```
kv-> verify upgrade
```

Output:

```
Verify: starting verification of store mystore based upon topology
sequence #156
150 partitions and 2 storage nodes
Time: 2024-04-05 10:35:44 UTC   Version: 24.1.11
See <XX>.com:/var/kv/kv-22.1.7/kvroot/mystore/log/mystore_{0..N}.log for
progress messages
Verify upgrade: Storage Node [sn1] on <XX>.com: 5000
Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:25:44 UTC  Build id:
477e7f102ab4
Edition: Enterprise    isMasterBalanced: true
serviceStartTime: 2024-04-05 10:29:33 UTC
Verify: sn2: Node needs to be upgraded from 22.1.7 to version 24.1.11 or
newer
Verify upgrade: Storage Node [sn2] on <XX>.com: 5000
Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 22.1.7 2024-04-05 16:36:54 UTC  Build id:
61b68fb1a3ec
```

```
Edition: Enterprise    isMasterBalanced: true
serviceStartTime: 2024-04-05 10:18:09 UTC

Verification complete, 0 violations, 1 note found.
Verification note: [sn2]
Node needs to be upgraded from 22.1.7 to version 24.1.11 or newer
```

- Obtain an ordered list of the Storage Nodes to upgrade. The output below shows that the Storage Node sn2 needs to be upgraded.

  ```
  kv-> show upgrade-order
  ```

  Output:

  ```
  Calculating upgrade order,target version: 24.1.11,prerequisite: 20.1.12
  sn2
  ```

- In your second Storage Node (sn2), install the new version of the software. See Install and verify your NoSQL Database installation for more details.

  > **Note:**
  >
  > The software (`kv-24.1.11.zip`) has already been copied from Storage Node sn1 to the Storage Node sn2.

- Stop the Oracle NoSQL Database Storage Node Agent and services related to the root directory of the current Oracle NoSQL Database (`22.1.7`).

  ```
  java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop
  -root $KVROOT
  ```

- Restart the second Storage Node using the updated software release(`24.1.11`). Here `$NEW_KVHOME` is `/var/kv/kv-24.1.11` and `$KVROOT` is `/var/kv/kv-22.1.7/kvroot`.

  ```
  nohup java -Xmx64m -Xms64m -jar $NEW_KVHOME/lib/kvstore.jar start
  -root $KVROOT &
  ```

- Invoke the `runadmin` command to start the Admin command line interface (CLI) utility on the Storage Node which is now running the updated software release. Here`$KVHOST` is the host name of the first Storage Node(sn1).

  ```
  java -Xmx64m -Xms64m -jar $NEW_KVHOME/lib/kvstore.jar
  runadmin -port 5000 -host $KVHOST -security
  $KVROOT/security/root.login
  ```

- Verify the store configuration to check if the upgrade for the second Storage Node (sn2) is completed successfully.

  ```
  kv-> verify upgrade
  ```

Output:

```
Verify: starting verification of store mystore based upon topology
sequence #156
150 partitions and 2 storage nodes
Time: 2024-04-05 13:32:24 UTC   Version: 24.1.11
See <XX>.com:/var/kv/kv-22.1.7/kvroot/mystore/log/mystore_{0..N}.log for
progress messages
Verify upgrade: Storage Node [sn1] on <XX>.com: 5000
Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:25:44 UTC  Build id:
477e7f102ab4
Edition: Enterprise    isMasterBalanced: true
serviceStartTime: 2024-04-05 10:29:33 UTC
Verify upgrade: Storage Node [sn2] on <XX>.com: 5000
Zone: [name=zone1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:25:44 UTC  Build id:
477e7f102ab4
Edition: Enterprise    isMasterBalanced: true
serviceStartTime: 2024-04-05 13:29:18 UTC

Verification complete, no violations.
```

- Check if any other Storage Nodes need to be upgraded.

```
kv-> show upgrade-order


Calculating upgrade order, target version: 24.1.11,
prerequisite: 20.1.12
There are no nodes that need to be upgraded
```

The output shows the upgrade for all Storage Nodes is complete.

Upgrading the Xregion Service Agent

If you are using XRegion Service Agent, then you should upgrade your data store first before upgrading the XRegion Service agent. If the agent is upgraded first before the data store is upgraded, the agent may get blocked when accessing the new system table and wait for the data store to be upgraded. To configure the XRegion Service agent see, Configure XRegion Service.

Upgrading the Oracle NoSQL Database Proxy

If you have configured Oracle NoSQL Database Proxy, make sure to upgrade the proxy. A compatible proxy jar file for a given database server, httpproxy.jar, is included in the database server bundle's lib directory. See Configuring the Proxy for more details.

Upgrading JDK on your Oracle NoSQL Database deployment

Consider that you have a JDK version, say JDK 11 SE, installed on all the Storage Nodes in your data store deployment. But after some time, Oracle releases a new version of the JDK,

say JDK 17 SE, that includes security enhancements and bug fixes. Now, you want to upgrade the existing JDK to a newer version of the JDK.

Additionally, during the upgrade, you want to ensure that the data store remains online and available to clients.

Consider that your existing Oracle NoSQL Database is deployed on 3 Storage Nodes (SN1, SN2, and SN3).

To update the JDK on your Oracle NoSQL Database deployment:

1. Based on the OS architecture, download and install the required version of JDK from Java SE Downloads.
2. Update the `$JAVA_HOME` and `$PATH` environment variables to point to the updated JDK directory.
3. Verify that in the Storage Node, the JDK is now pointing to the new JDK by running the `java -version` command and verifying the output.
4. Stop the SNA (Storage Node Agent) process in SN1 by running the following command:

   ```
   java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop -root $kvroot
   ```

5. Restart the SNA process in SN1 by running the following command:

   ```
   nohup java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar start -
   root $kvroot &
   ```

6. Repeat steps 1 through 5 for each Storage Node. Make sure these steps are run sequentially on all the Storage Nodes. For example, run steps 1 to 5 on SN1, followed by SN2, and so on.

3
Configure

The articles in this section provide steps on how to configure Oracle NoSQL Database.

Configuration Basics

Once you have installed Oracle NoSQL Database on each of the Storage Nodes that you are using in your data store (see Installing Oracle NoSQL Database), you must configure the data store. To do this, you use the Administration command line interface (CLI).

- Installation Configuration Parameters
- Configuring the Firewall

Installation Configuration Parameters

Before you configure Oracle NoSQL Database, you should determine the following parameters for each Storage Node in the data store. Each of these parameters are directives to use with the `makebootconfig` utility:

- **root**

  Where the KVROOT directory should reside. The KVROOT directory stores the data of your data store and security related information. There should be enough disk space on each Storage Node to hold the data to be stored in your data store. The KVROOT disk space requirements can be reduced if the *storagedir* parameter is used to store the data at a different location outside the KVROOT directory. It is recommended that you make the KVROOT directory the same local directory path on each Storage Node (but not a shared or NFS mounted directory).

- **port**

  The TCP/IP port through which the Storage Node connects to the Oracle NoSQL Database. This port should be free (unused) on each Storage Node. The default port used in all examples is 5000. This port is sometimes referred to as the *registry port*.

- **harange**

  The replication nodes and Admin process use the harange (high availability range) ports to communicate between each other. For each Storage Node in the data store, specify sequential port numbers, one port for each replication node on the Storage Node, plus an additional port if the Storage Node hosts an Admin. Specify the port range as `startPort,endPort`.

- **servicerange**

  A range of ports that a Storage Node uses to communicate with other administrative services and its managed services. Some of the managed services are the replication nodes for every Storage Node. This optional parameter is useful when Storage Node services must use specific ports for a firewall or other security purposes. By default, the services use anonymous ports. Specify the port range as `startPort,endPort`. For more information, see Storage Node Parameters.

- **store-security**

Specifies whether security is in use. While this is an optional parameter, it is strongly advised that you configure Oracle NoSQL Database with security enabled.

Specifying `none` indicates that security will not be in use.

Specifying `configure` indicates that you want to configure a secure data store. The `makebootconfig` process will then invoke the `securityconfig` utility as part of its operation.

Specifying `enable` indicates security will be in use. However, you will need to either explicitly configure security by utilizing the security configuration utility(`securityconfig`), or copy a previously created security configuration from another system.

> **Note:**
>
> If you do not specify the `-store-security` parameter, security is configured by default. To complete a secure installation, you must use the `securityconfig` utility to create the security folder before starting up the Storage Node Agents.

- **capacity**

The total number of replication nodes the Storage Node can support. Capacity is an optional, but extremely important parameter, representing the number of replication nodes. If the Storage Node you are configuring has the resources to support more than one replication node, set the capacity value to the appropriate number. To host a replication node successfully to handle peak run time demand, you need sufficient disk, cpu, memory, and network bandwidth .

To have your Storage Node host Arbiter Nodes, set the capacity to `0` . A Storage Node with capacity 0 will be allocated as Arbiter Nodes whenever required. For more information see Deploying an Arbiter Node Enabled Topology.

Consider the following configuration settings for Storage Nodes with a capacity greater than one:

1. It is recommended that you configure each Storage Node with a capacity equal to the number of available disks on the machine. Such a configuration permits the placement of each replication node on its own disk, ensuring that replication nodes on the Storage Node are not competing for I/O resources. The `-storagedir` parameter lets you specify the directory location for each replication node disk.

   For example:

```
> java -Xmx64m -Xms64m \
    -jar $KVHOME/lib/kvstore.jar makebootconfig \
    -root /opt/ondb/var/kvroot \
    -port 5000 \
    -host node10
    -harange 5010,5025 \
    -capacity 3 \
    -admindir /disk1/ondb/admin01 \
    -admindirsize 5000_MB \
    -storagedir /disk1/ondb/data \
    -storagedir /disk2/ondb/data \
    -storagedir /disk3/ondb/data \
    -storagedirsize 1_tb \
    -rnlogdir /disk1/ondb/rnlog01 \
```

```
-rnlogdir /disk2/ondb/rnlog02 \
-rnlogdir /disk3/ondb/rnlog03
```

where `-capacity 3` represents the number of replication nodes on the Storage Node (`node10`). The three replication nodes are in the corresponding disks (`disk1`, `disk2`, `disk3`).

2. Increase the `-harange` parameter to support additional ports required for the replication and Admin Nodes.
3. Increase the `-servicerange` parameter to account for the additional ports required by the replication nodes.

If no value for capacity is specified, it defaults to `1`.

- **storage-type**

  Specifies the type of disk on which the storage directories reside. You can specify storage type only for a Storage Node and not for replication nodes. You can set one value for this parameter for a Storage Node. The valid values are HD, SSD, NVME and UNKNOWN.

  > **Note:**
  >
  > The parameters `storagedir` and `storagedirsize` are specific to every replication node, whereas storage-type is specific to a Storage Node.

- **admindir**

  The directory path to contain the environment associated with a Storage Node Admin process.

  It is strongly recommended that the Admin directory path resolves to a separate disk. You can accomplish this by creating suitable entries in the `/etc/fstab` directory that attaches the file system on disk to an appropriate location in the overall directory hierarchy. Placing the Admin environment on a separate disk ensures that the Admin is not competing for I/O resources. It also isolates the impact of a disk failure to a single environment.

  If you do not specify an explicit directory path for `-admindir`, the Admin environment files are located in this directory:

  ```
  $KVROOT/KVSTORE/<SNID>/<AdminId>/
  ```

- **admindirsize**

  The size of the Admin storage directory. This is optional but recommended. For more information, see Managing Admin Directory Size.

- **storagedir**

  A directory path that will contain the environment associated with a replication node. When the `-capacity` parameter is greater than 1, it is recommended that you specify a multiple set of `-storagedir` parameter values, one for each replication node that the Storage Node hosts. Each directory path should resolve to a separate disk. You can accomplish this by creating suitable entries in the `/etc/fstab` directory that attaches the file system on disk to an appropriate location in the overall directory hierarchy. Placing each environment on a separate disk ensures that the shards are not competing for I/O resources. It also isolates the impact of a disk failure to a single location.

- **storagedirsize**

The size of each storage directory. It is strongly recommended that you specify this parameter for each replication node. The Oracle NoSQL Database uses the storage directory size to enforce disk usage, using the `-storagedirsize` parameter value to calculate how much data to store on disk before suspending write activity. For more information, see Managing Storage Directory Sizes.

- **rnlogdir**

  The directory path to contain the log files associated with a replication node. For capacity values greater than one, specify multiple `rnlogdir` parameters, one for each replication node that the Storage Node is hosting.

  It is recommended that each `rnlogdir` path resolves to a separate disk partition on a replication node. You can accomplish this by creating suitable entries in the `/etc/fstab` directory that attaches the file system on a disk to an appropriate location in the overall directory hierarchy. Placing `rnlogdir` in a distinct partition on the replication node ensures that metrics and errors can be reported and retained, even if the partition containing the data store log files is full. Separating the `rnlogdir` on a distinct partition also isolates the impact of losing complete replication node log files from a `kvroot` disk failure.

  If you do not specify a location for `rnlogdir`, logs are placed under the `$KVROOT/KVSTORE/log` directory by default.

- **num_cpus**

  The total number of processors on the machine available to the replication nodes. This is an optional parameter, used to coordinate the use of processors across replication nodes. If the value is `0`, the system queries the Storage Node to determine the number of processors on the machine. The default value for `num_cpus` is `0`, and examples in this document use that value.

- **memory_mb**

  The total number of megabytes of memory available to the replication node. The system uses the `memory_mb` value to guide specification of the replication node's heap and cache sizes. This calculation is more critical if a Storage Node hosts multiple replication nodes, and must allocate memory between these processes. If the value is `0`, the system attempts to determine the amount of memory on the replication node. The default value for `memory_mb` is `0`, and examples in this document use that value.

- **force**

  Specifies that the command generates the boot configuration files, even if verifying the configuration against the system finds any inaccurate parameters. That means you force the creation of the boot configuration file, even if the value of any of the parameters discussed above (like port, harange etc) is inaccurate.

See makebootconfig for more details.

Configuring the Firewall

Most of the Storage Nodes, either physical or virtual machines, have built-in firewalls. Additionally, you may have separate firewalls in-between machines. In a NoSQL topology, the Storage Nodes need to communicate with one another, so communication must pass through the firewalls. You need to open the firewall ports used by the communication channels in the data store. To make sure your network firewall works with your topology, you should set the ports specified by the `-port` , `-harange`, `-servicerange`, and `-admin-web-port` parameters of the `makebootconfig` command. These four parameters are used to constraint the data store to a limited set of ports. Setting the ports is usually done for security or data center policy

reasons. By default the services in your data store use anonymous ports. To specify a range of ports, you use the format of `startPort,endPort`.

Configuring security in a data store

Basics of data store security

Oracle NoSQL Database can be configured securely.

In a secure configuration, network communications between NoSQL clients, utilities, and NoSQL data store components are encrypted using SSL/TLS, and all processes must authenticate themselves to the components to which they connect. It is strongly advised that you configure Oracle NoSQL Database with security enabled.

When you configure the Oracle NoSQL Database, the parameter `store-security` specifies whether security is in use. Specifying ***none*** indicates that security will not be in use. Specifying ***configure*** indicates that you want to configure security. When you specify ***configure*** or do not specify the `store-security` parameter, then the **makebootconfig** process will invoke the **securityconfig** utility as part of its operation. Specifying ***enable*** indicates security will be in use. When you specify ***enable*** , you will need to either explicitly configure security by utilizing the security configuration utility(`securityconfig`), or copy a previously created security configuration from another system.

> **Note:**
>
> If you do not specify the `-store-security` parameter, security is configured by default. To complete a secure installation, you must use the `securityconfig` utility to create the security folder before starting up the Storage Node agents.

Configuring security using securityconfig tool

You can run the `securityconfig` tool before or after the `makebootconfig` process. This tool creates the security directory and also creates security related files. The `makebootconfig` utility automatically invokes the `securityconfig` tool in one of the following two scenarios.

- You specify `store-security configure` in the `makebootconfig` command explicitly requesting to configure a secure data store.
- You omit the `store-security` parameter in the `makebootconfig` command. A secure data store is then configured by default.

Invoke the `securityconfig` tool as shown below:

```
java -Xmx64m -Xms64m
-jar $KVHOME/lib/kvstore.jar
securityconfig \
config create -root $KVROOT -kspwd (******)
Created files
```

```
$KVROOT/security/security.xml
$KVROOT/security/store.keys
$KVROOT/security/store.trust
$KVROOT/security/client.trust
$KVROOT/security/client.security
$KVROOT/security/store.passwd (Generated in CE version)
$KVROOT/security/store.wallet/cwallet.sso (Generated in EE version)
Created
```

See Configuring Security with Securityconfig in the *Security Guide* for more details.

If you have more than one Storage Node in your data store, then the security configuration is configured in the first Storage Node using `-store-security configure`). The security directory and all files contained in it should be copied from the first Storage Node to other Storage Nodes to setup security. Zip all the security related files from the first Storage Node to `security.zip`.

```
cd ;
zip -r $HOME/security.zip $KVROOT/security;
cd -
```

Copy the `security.zip` from first Storage Node to other Storage Nodes. In the other Storage Nodes, you will unzip the `security.zip` file and use this security information (copied from the first Storage Node). You then use `-store-security enable` while configuring the remaining Storage Nodes.

Create users and configure security with remote access

You must create users for a secure cluster.

To configure security with remote access, perform the following steps:

- Invoke the `runadmin` command to start the Admin command line interface (CLI) utility on the Storage Node. This command starts the `kv` prompt.

  ```
  java -jar $KVHOME/lib/kvstore.jar runadmin -host $KVHOST -port 5000 -
  security $KVROOT/security/client.security
  ```

- Create the first admin user. In this case, user `root` is defined.

  ```
  kv->execute 'CREATE USER root IDENTIFIED BY "password" ADMIN'
  ```

- Grant the `readwrite` role to the first admin user:

  ```
  kv->execute "GRANT readwrite TO USER root"
  ```

- Exit the Admin command line interface (CLI) utility.

  ```
  kv-> exit
  ```

- Generate a password store for the first admin user. This step creates a `root.passwd` file in the `$KVROOT/security` directory. These are the commands to create `root.passwd`.

```
java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar securityconfig \
pwdfile create -file $KVROOT/security/root.passwd


java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar securityconfig \
pwdfile secret \
-file $KVROOT/security/root.passwd -set -alias root -secret password
```

- Copy the `client.security` file to another file named `root.login`. This `client.security` was created by the `securityconfig` utility earlier. For details, see Configuring security using securityconfig tool.

```
cp $KVROOT/security/client.security $KVROOT/security/root.login
```

- Zip all the user security files. This must be copied to all the Storage Nodes of the data store.

```
cd $KVROOT/security;
zip -r root.zip root.* client.trust ;
cd -
```

- From every Storage Node (other than the first Storage Node in the data store), unzip the user security files into the `$KVROOT/security` directory.

```
unzip -o $KVROOT/security/root.zip -d $KVROOT/security
```

- You can now access the Admin node running on a Storage Node from another Storage Node remotely as follows:

```
java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar runadmin \
-port 5000 -host node01 \
-security $KVROOT/security/root.login
```

Configure a single node KVLite

KVLite is a simplified version of the Oracle NoSQL Database.

KVLite is a single shard data store, that is not replicated. It runs in a single process without requiring any administrative interface. You configure, start, and stop KVLite using a command line interface.

KVLite is intended for use by application developers who need to develop and unit test their Oracle NoSQL Database applications. It can be used as a development platform for developers to get familiar with Oracle NoSQL APIs, and test different ways of interacting with these APIs. This is the simplest configuration of a NoSQL database and helps you get started quickly as it does not need any detailed configuration steps. However it is not intended for production deployment, or for performance measurements.

Start KVLite in a secure mode as shown below.

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar kvlite
```

`kvstore` is the name of the data store that gets configured and `kvroot` is the directory where Oracle NoSQL Database data is placed.

Also, KVLite is secure by default. If you want to run KVLite in unsecure mode, you will have to explicitly provide parameters to disable security while installing KVLite as shown below.

```
 java -jar lib/kvstore.jar kvlite -secure-config disable
```

Configuring a single region data store

At a high level, configuring your store requires these steps:

- Configuring your data store installation
- Using Plans
- Start the Administration CLI
- Name your data store
- Create a Zone
- Create an Administration Process on a Specific Storage Node
- Create a Storage Node Pool
- Create the Remainder of your Storage Nodes
- Create and Deploy Replication Nodes
- Smoke Testing the System
- Create a script to configure the data store
- Troubleshooting

Configuring your data store installation

Once you determine your data store's configuration information as described in the previous section (see Installation Configuration Parameters), complete the following tasks to configure your data store.

1. Create the initial bootconfig configuration file using the `makebootconfig` command. Do this on each Storage Node.

> **Note:**
>
> Using the `makebootconfig` command to create the configuration file is integrated with the Storage Node on which you run the command. Such integration, checks and validates all parameters and their values against the Storage Node environment before generating the boot configuration files. To bypass verifying any parameters or values for the boot configuration files, use the ***-force*** flag (`makebootconfig -force`).

Following is an example of using `makebootconfig`, using a sample set of parameters and values. For a list of all the `makebootconfig` parameters, see makebootconfig.

```
> mkdir -p $KVROOT


> java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar \
makebootconfig -root $KVROOT \
               -port 5000 \
               -host $KVHOST \
               -harange 5010,5020 \
               -capacity 1 \
               -admindir /export/admin \
               -admindirsize 5000_MB \
               -storagedir /export/data1 \
               -storagedirsize 1_tb \
               -rnlogdir /export/rnlogs
```

> **Note:**
>
> It is strongly recommended that you specify both `storagedir` and `storagedirsize`. If you specify the `-storagedir` parameter, but not `-storagedirsize`, `makebootconfig` displays a warning.

When the `store-security` parameter is omitted from the `makebootconfig` command , a secure data store is configured by default. The `makebootconfig` command internally invokes the `securityconfig` tool to create the security directory and security related files. To configure a non secure data store, specify `store-security none` in the `makebootconfig` command. However it is recommended to configure a secure data store in production environments.

2. Start the Oracle NoSQL Database Storage Node Agent (SNA) on each of the Oracle NoSQL Database Storage Nodes. The SNA manages the Oracle NoSQL Database administrative processes on each Storage Node. It also owns and manages the registry port, which is the main way to communicate with Oracle NoSQL Database on that Storage Node. Before starting the SNA, on each Storage Node, set the environment variable `MALLOC_ARENA_MAX` to 1. Doing this ensures that memory usage is restricted to the specified heap size. To start the SNA on each Storage Node use the `start` command as follows:

```
nohup java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar start -root $KVROOT &
```

> **Note:**
>
> If the replication node or the Admin Service crashes, the SNA restarts the processes.

3. Use the `jps -m` command to verify that the Oracle NoSQL Database processes are running :

   ```
   > jps -m
   ```

   Output:

   ```
   2830534 kvstore.jar start -root $KVROOT
   2830645 ManagedService -root $KVROOT -secdir $KVROOT/security -class Admin
   -service BootstrapAdmin.5000 -config config.xml
   ```

4. Using `ssh` to reach the node, issue a `ping` command (in security mode) to be sure that the Oracle NoSQL Database client library can contact the Oracle NoSQL Database Storage Node Agent.

   > **Note:**
   >
   > If your data store is a non secure one, the `-security` option in the below command can be omitted.

   ```
   ssh node01
   java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar ping -host $KVHOST -port
   5000
   -security $KVROOT/security/client.security

   Login as: Anonymous (Enter any user name here)
   Anonymous's password: (Enter any password)

   SNA at hostname: node01, registry port: 5000 is not registered.
   No further information is available
   Can't find store topology:
   Could not contact any RepNode at: [node01:5000]
   ```

   This return informs you, that only the Storage Node process is running on the Storage Node `node01`. Once Oracle NoSQL Database is fully configured, you can use the `ping` command again to get more details.

   If the client library cannot contact the SNA, the `ping` command displays this message:

   ```
   Unable to connect to the storage node agent at host <hostname>,
   port 5000, which may not be running; nested exception is:

   java.rmi.ConnectException: Connection refused to host: <hostname>;
   nested exception is:
   java.net.ConnectException: Connection refused
   Can't find store topology:
   Could not contact any RepNode at: [<hostname>:5000]
   ```

If the Storage Nodes do not start up, review the `adminboot` and `snaboot` logs in the `$KVROOT` directory to investigate what occurred and to help identify the problem. When the Storage Nodes have all started successfully, you can configure the data store.

> **Note:**
>
> For best results, configure your Storage Nodes so that the SNA starts automatically when the Storage Node boots up. The details of how to do this are beyond the scope of this document, because they depend on how your operating system is designed. See your operating system documentation for information about launching an application automatically at bootup.

Using Plans

You use plans to configure your data store. A plan consists of administrative operations. Plans can modify the state managed by the Admin service, and issue requests to data store components such as Storage Nodes and replication nodes. Some plans consist of simple state-changing operations, while others perform a set of tasks that affect every Storage Node and replication nodes in the data store. For example, you use a plan to create a zone or Storage Node, or to reconfigure parameters on a replication node.

You use the `plan` command, available from the administrative command line interface, to both create and execute plans, as well as to perform many other tasks. For more about using the `plan` command, see CLI Command Reference.

By default, running a `plan` command executes asynchronously in the background. The command line prompt returns as soon as the background process begins. You can check the progress of a running plan using the `show plan` *`id`* command.

You can run a `plan` command synchronously in two ways:

```
plan action_to_complete –wait

plan wait -id plan_id
```

Using either the `-wait` flag or the `plan wait` command, causes the command line prompt to return only after the command completes.

The `-wait` flag and the `plan wait` command are useful when executing plans from scripts, which typically expect each command to finish before processing the next command.

You can also create a plan, but defer its execution using the optional `-noexecute` flag, as follows:

```
plan action –name plan-name -noexecute
```

Later, you can execute the plan on demand as follows:

```
plan execute -id id_num
```

Tracking Plan Progress

There are several ways to track the progress of a plan.

- The `show plan -id` command provides information about the progress of a running plan. Use the optional `-verbose` flag to get more details.

- The CLI `verify` command gives service status information as the plan is executing and services start.

> **Note:**
>
> The `verify` command is of interest for only topology-related plans. If the plan is modifying parameters, such changes may not be visible using the verify command.

- The CLI's `logtail` command lets you follow the store-wide log.

Plan States

Plans can be in any of the following states. A plan can be in only one state at a time. These are the possible states:

Name	Description
APPROVED	The plan exists with correct operations, but is not running.
CANCELED	A plan that is manually `INTERRUPTED` or that experiences an `ERROR` can be terminated. Use the `cancel` command to terminate a plan.
ERROR	If a plan in the `RUNNING` state encounters a problem, it transitions to this state and ends processing without successfully completing. Storage Nodes and replication nodes can encounter an error before the plan processes the error and transitions to an `ERROR` state.
INTERRUPTED	A `RUNNING` plan transitions to this state after the `interrupt` command in the CLI.
INTERRUPT REQUESTED	When a running plan receives an interrupt request, the plan may have to cleanup or reverse previous steps taken during its execution. If the plan transitions to this state, it is to make sure that the data store remains in a consistent state.
RUNNING	The plan is currently executing its commands.
SUCCEEDED	The plan has completed successfully.

You can use the `plan execute` command whenever a plan enters the `INTERRUPTED`, `INTERRUPT REQUESTED` or `ERROR` state. Retrying is appropriate if the underlying problem was transient or has been rectified. When you retry a Plan, it processes the steps again. Each step is idempotent, and can be safely repeated.

Reviewing Plans

You can use the CLI `show plans` command to review the execution history of plans. The command also lists the plan ID numbers, plan names, and the state of each plan. With the plan ID, use the `show plan -id <plan number>` command to see more details about a specific plan.

The next example shows the output of both the `show plans` command and then the `show plan -id <plan number>` command. The `show plan` command returns the plan name, the number of

attempts, the start and end date and times, the total number of tasks the plan completed, and the whether the plan completed successfully.

```
kv-> show plans
```

Output:

```
1 Deploy KVLite         SUCCEEDED
2 Deploy Storage Node   SUCCEEDED
3 Deploy Admin Service SUCCEEDED
4 Deploy KVStore        SUCCEEDED


kv-> show plan -id 3
```

Output:

```
Plan Deploy Admin Service (3)
Owner: null
State:          SUCCEEDED
Attempt number: 1
Started:        2024-04-05 22:05:31 UTC
Ended:          2024-04-05 22:05:31 UTC
Total tasks:    1
Successful:     1
```

Plan Ownership

In a secure Oracle NoSQL Database deployment, each plan command is associated with its creator as the owner. Only the plan owner can see and operate it. If a plan is created in an earlier version of Oracle NoSQL Database, or in an non secure data store, the owner is `null`.

> **Note:**
>
> The SYSOPER privilege allows a user to perform cancel, execute, interrupt, and wait on any plan.

Users with the `SYSVIEW` privilege can see plans owned by other users, plans with a `null` owner, and plans whose owners have been removed from the Oracle NoSQL Database.

For more information about user privileges and on configuring Oracle NoSQL Database securely, see the Security Guide.

Pruning Plans

The system automatically prunes plans that should be removed. Plans are removed from the Admin Store if they match both of these conditions:

- Are in a terminal state (`SUCCEEDED` or `CANCELLED`)
- Have a Plan ID number that is 1000 less than the most recent Plan ID

For example, if the most recent Plan ID is 2000, the system prunes all plans with ID numbers 1000 or less that are in a terminal state . The system does not remove plans in a non-terminal state.

While pruning plans occurs automatically, you can detect that pruning has occurred in these situations:

- Attempting to show a plan with a specific ID that has been pruned.
- Specifying a range of plans that contains one or more removed plans.

Start the Administration CLI

Before running the Admin CLI and continuing further, you must have already completed all of the configuration steps described in Configuring your data store installation.

The `runadmin` utility provides the Admin command line interface (CLI). You can use the `runadmin` utility for a number of purposes. In this section, you use it to administer the Storage Nodes in your data store. First, you supply the details of the Storage Node and registry port that `runadmin` can use to connect to the data store.

If this is the first Storage Node you are connecting to that data store using the CLI, the Storage Node is designated as the one on which the master copy of the administration database resides.

> **Note:**
>
> You cannot change whatever Storage Node you use to initially configure the data store, such as `node01` in this example. Carefully plan the Storage Node to which `runadmin` first connects.

In the example below, you use `$KVHOST` to represent the network name of the Storage Node to which `runadmin` connects, and you use `5000` as the registry port.

One of the most important aspects of this Storage Node is that it must run the Storage Node Agent (SNA). All Storage Nodes should have an SNA running on them at this point. If any do not, complete the instructions in Installing Oracle NoSQL Database before proceeding further.

To start `runadmin` to use the Admin command line interface (CLI) for administration purposes, use these commands:

```
ssh node01
> java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar runadmin \
-host $KVHOST -port 5000 \
-security $KVROOT/security/client.security
```

With this `runadmin` example, you specify a single host and port (`-host node01 -port 5000`), permitting one Storage Node host to run an Admin process. The Admin process lets you run Admin CLI commands. If you want more than one Storage Node to support CLI commands, use the `runadmin` utility `-helper-hosts` flag and list two or more Storage Nodes and ports, rather than `-host <name> -port <value>`. For example, the next command starts an Admin

process on three different Storage Nodes, which can then service CLI commands (<host2>, <host3>, and <host4>):

```
ssh node01
> java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts <host2>:5000, <host3>:5000, <host4>:5000 \
-security $KVROOT/security/client.security
```

> **Note:**
>
> You need to complete the steps in Create users and configure security with remote access , to use the`runadmin` command to access the Admin node running on a Storage Node from any another Storage node.

After starting the Admin CLI, you can invoke the `help` command to describe all of the CLI commands.

You can collect the configuration steps that this section describes into a file, and then pass the script to the CLI utility using the `-script`option. See Create a script to configure the data store for more information.

Name your data store

When you start the Admin CLI , the `kv->` prompt appears. Once you see this, you can name your data store by using the `configure -name` command. The only information this command needs is the name of the data store that you want to configure.

Note that the name of your data store is essentially used to form a path to records kept in the data store. For this reason, you should avoid using characters in the data store name that might interfere with its use within a file path. The command line interface does not allow an invalid data store name. Valid characters are alphanumeric, '-', '_', and '.'.

For example:

```
kv-> configure -name mystore
```

Output:

```
Store configured: mystore
```

> **Note:**
>
> The data store name must be unique across all instances of NoSQL Database.

Create a Zone

After starting the Admin command line interface (CLI) and naming your data store, you need to create at least one zone. It is possible, and even desirable, to create more than one zone. Because zones are complete copies of your data store, using multiple zones improves your data store's availability. This section describes an installation with a single zone. For more directions about creating a store deployment with multiple zones, see Configuring with Multiple Zones.

> **Note:**
>
> Once you add Storage Nodes to a zone, you cannot remove the zone from your data store.

To create a zone, use the `plan deploy-zone` with this usage:

```
plan deploy-zone -name <zone name>
-rf <replication factor>
[-type [primary | secondary]]
[-arbiters | -no-arbiters ]
[-json ]
[-master-affinity | -no-master-affinity]
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

where:

- `-arbiters`

  Specifies that you can allocate Arbiter Nodes on the Storage Node in the zone.

- `-no-arbiters`

  Specifies that you cannot allocate Arbiter Nodes on the Storage Node in the zone. You can specify this flag only on a primary zone.

  > **Note:**
  >
  > Only primary zones can host Arbiter Nodes.

- `-rf`

  A number specifying the Zone Replication Factor. A primary zone can have a Replication Factor equal to zero.

- `-name`

  Identifies the zone name, as a string.

- `-json`

  Formats the command output in JSON.

> **Note:**
>
> Only primary zones can host Arbiter Nodes.

- `-master-affinity`

  Indicates that this zone is a Master Affinity zone.

- `-no-master-affinity`

  Specifies that this zone is not a Master Affinity zone.

- `-type`

  Specifies the type of zone to create. If you do not specify a –type, the `plan` utility creates a Primary zone.

For more information on Primary and Secondary Replication Factors, see Configuring with Multiple Zones.

When you execute the `plan deploy-zone` command, the CLI returns the plan number. It also returns instructions on how to check the plan's status, or to wait for it to complete. For example:

```
kv-> plan deploy-zone -name "Boston" -rf 1 -wait
```

Output:

```
Executed plan 1, waiting for completion...
Plan 1 ended successfully
```

You can show the plans and their status using the `show plans` command.

```
kv-> show plans
```

Output:

```
1 Deploy Zone (1)          SUCCEEDED
```

A zero Replication Factor zone is useful to host only Arbiter Nodes. You would add zero capacity Storage Nodes to this zone in order to host Arbiter Nodes. For more information see Deploying an Arbiter Node Enabled Topology.

You can also create Master Affinity Zones, which let you prioritize master nodes in primary zones. See Master Affinity Zones for details.

Create an Administration Process on a Specific Storage Node

Every data store has an administration database. The Admin CLI is currently connected to the Storage Node `node01`. Use the `deploy-sn` command to deploy the Storage Node `node01`. You then use the command `deploy-admin` to deploy an Administration process on the same Storage Node `node01` to continue configuring this data store.

The `deploy-admin` command creates an Administration process, with the same type as the Storage Node (SN) zone — if the zone is primary, the Admin is a primary Admin; if a secondary zone, so is the Admin.

Secondary Admins support failover. If a primary Admin fails, it converts to an offline secondary to re-establish quorum using existing Admins. A secondary Admin converts to a primary to take over for the failed primary. For more information on how quorum is applied, see the *Concepts Guide*.

To support failover, ensure that any zones used to continue data store operation after a failure contain at least one Admin node.

> **Note:**
>
> A deployed Admin must be the same type (`PRIMARY` or `SECONDARY`) as its zone. Also, the number of deployed Admins in a zone should be equal to the Zone Replication Factor.

The `deploy-sn` command requires a Zone ID. You can get this ID by using the `show topology` command:

```
kv-> show topology
```

Output:

```
store=mystore  numPartitions=0 sequence=1
zn: id=zn1 name=Boston repFactor=1 type=PRIMARY
allowArbiters=false masterAffinity=false
```

The zone ID is `zn1` in the output.

When you deploy the Storage Node, provide the zone ID, the node's network name, and its registry port number. For example:

```
kv-> plan deploy-sn -zn zn1 -host <hostname> -port 5000 -wait
```

Output:

```
Executed plan 2, waiting for completion...
Plan 2 ended successfully
```

Having deployed the Storage Node, create the Admin process on the Storage Node that you just deployed, using the `deploy-admin` command. This command requires the Storage Node ID (which you can obtain using the `show topology` command) and an optional plan name.

```
kv-> plan deploy-admin -sn sn1 -wait
```

Output:

```
Executed plan 3, waiting for completion...
Plan 3 ended successfully
```

Create a Storage Node Pool

Once you have created your Administration process, you can create a Storage Node Pool. This pool is used to contain all the Storage Nodes in your data store. A Storage Node pool is used for resource distribution when creating or modifying a data store. You use the `pool create` command to create this pool, then you join Storage Nodes to the pool using the `pool join` command.

Note that a default pool called **`AllStorageNodes`** will be created automatically and all SNs will be added to it during the topology deployment process. Therefore, the `pool` commands are optional if you use the `AllStorageNodes` pool as the default pool during deployment. You may have multiple kinds of Storage Nodes in different zones that vary by processor type, speed and/or disk capacity. So the Storage Node pool lets you define a logical grouping of Storage Nodes by whatever specification you pick.

> **Note:**
>
> This section is only to demonstrate how to explicitly create a Storage Node pool. Skip this section if you want to use the default pool `AllStorageNodes` during the topology deployment process.

Remember that you already have a Storage Node created. You did that in the previous step where you used the `deploy-sn` command to deploy the Storage Node. Therefore, after you add the pool, you can immediately join that first Storage Node to the pool.

The `pool create` command only requires you to provide the name of the pool.

The `pool join` command requires the name of the pool to which you want to join the Storage Node, and the Storage Node's ID. You can obtain the Storage Node's ID using the `show topology` command.

For example:

```
kv-> pool create -name BostonPool
```

Output:

```
Added pool BostonPool


kv-> show topology
```

Output:

```
store=mystore  numPartitions=0 sequence=2
zn: id=zn1 name=Boston repFactor=1 type=PRIMARY
```

```
allowArbiters=false masterAffinity=false
sn=[sn1] zn:[id=zn1 name=Boston] <hostname>:5000 capacity=1 RUNNING


kv-> pool join -name BostonPool -sn sn1
```

Output:

```
Added Storage Node(s) [sn1] to pool BostonPool
```

Create the Remainder of your Storage Nodes

This section is only applicable if you are configuring multiple Storage Nodes. Skip this section if you are configuring a single Storage Node.

Having created your Storage Node Pool, you can create the remainder of your Storage Nodes. Every Storage Node hosts various Oracle NoSQL Database admin and managed services in the data store. Consequently, you must use the `deploy-sn` command in the same way as you did in Create an Administration Process on a Specific Storage Nodeto add each new Storage Node to your data store. As you deploy each Storage Node, join it to your Storage Node Pool as described in the previous section.

Hint: Storage Node ID numbers increment sequentially with each Storage Node you add. So you do not have to repetitively look up the IDs with `show topology`. If the last Storage Node you created was assigned an ID of `10`, then the next Storage Node is automatically assigned ID `11`.

```
kv-> plan deploy-sn -zn zn1 -host <host2> -port 5000 -wait
Executed plan 4, waiting for completion...
Plan 4 ended successfully
kv-> pool join -name BostonPool -sn sn2
Added Storage Node(s) [sn2] to pool BostonPool
kv-> plan deploy-sn -zn zn1 -host <host3> -port 5000 -wait
Executed plan 5, waiting for completion...
Plan 5 ended successfully
kv-> pool join -name BostonPool -sn sn3
Added Storage Node(s) [sn3] to pool BostonPool
kv->
....
```

Repeat this process for all new Storage Nodes in your data store.

Create and Deploy Replication Nodes

The final step in your configuration process is to create replication nodes on every Storage Node in your data store. You do this using the `topology create` and `plan deploy-topology` commands. The `topology create` command takes the following arguments:

- *topology name* - A string to identify the topology.
- *pool name* - A string to identify the pool.
- *number of partitions*

The initial configuration is based on the number of Storage Nodes specified by the pool. This number is fixed once the topology is created and it cannot be changed. The command will automatically create an appropriate number of shards and replication nodes based upon the Storage Nodes in the pool.

You should make sure the number of partitions you select is more than the largest number of shards you ever expect your data store to contain, because the total number of partitions is static and cannot be changed. For simpler use cases, you can use the following formula to arrive at a very rough estimate for the number of partitions:

```
(Total number of disks hosted by the Storage Nodes  /
 Replication Factor) * 10
```

To get a more accurate estimate for production use, see Number of Partitions.

The `plan deploy-topology` command requires a topology name.

Once you issue the following commands, your data store is fully installed and configured:

```
kv-> topology create -name topo -pool BostonPool -partitions 300
```

Output:

```
Created: topo
```

```
kv-> plan deploy-topology -name topo -wait
```

Output:

```
Executed plan 6, waiting for completion...
Plan 6 ended successfully
```

> **Note:**
>
> If you have not created an explicit Storage pool , use `-pool AllStorageNodes` in the above command.

As a final sanity check, you can confirm that all of the plans succeeded using the `show plans` command:

```
kv-> show plans
```

Output:

```
1 Deploy Zone (1)           SUCCEEDED
2 Deploy Storage Node (2)   SUCCEEDED
3 Deploy Admin Service (3)  SUCCEEDED
4 Deploy-RepNodes (4)       SUCCEEDED
```

You can then exit the command line interface.

```
kv-> exit
```

Smoke Testing the System

There are several things you can do to ensure that your data store is up and fully functional. You verify your data store using the `verify configuration` command in the CLI.

1. The `verify configuration` command inspects all the components of the data store. It also checks whether all store services are available. For the available store services, the command also checks for any version or metadata mismatches. The command requires no parameters, and runs in verbose mode, by default. For example:

   ```
   kv-> verify configuration
   ```

   Output:

   ```
   Verify: starting verification of store mystore based upon topology
   sequence #2
   0 partitions and 1 storage nodes
   Time: 2024-04-05 10:41:15 UTC   Version: 24.1.11
   See <hostname>:$KVROOT/mystore/log/mystore_{0..N}.log for progress messages
   Verify: Shard Status: healthy: 0 writable-degraded: 0 read-only: 0
   offline: 0 total: 0
   Verify: Admin Status: healthy
   Verify: Zone [name=Boston id=zn1 type=PRIMARY allowArbiters=false
   masterAffinity=false]
   RN Status: online: 0 read-only: 0 offline: 0
   Verify: == checking storage node sn1 ==
   Verify:         sn1: sn1 has 0 RepNodes and is under its capacity limit of
   1
   Verify: Storage Node [sn1] on <hostname>: 5000
   Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
   masterAffinity=false]
   Status: RUNNING   Ver: 24.1.11 2024-04-05 21:25:44 UTC  Build id:
   477e7f102ab4
   Edition: Enterprise    isMasterBalanced: unknown     serviceStartTime:
   2024-04-05 10:37:28 UTC
   Verify:         Admin [admin1]   Status: RUNNING,MASTER  serviceStartTime:
   2024-04-05 10:38:21 UTC
   stateChangeTime: 2024-04-05 10:38:21 UTC    availableStorageSize: 999 MB

   Verification complete, 0 violations, 1 note found.
   Verification note: [sn1]  sn1 has 0 RepNodes and is under its capacity
   limit of 1
   ```

   If the output shows all Storage Nodes and replication nodes as running without any errors, then the data store is configured well and all Storage Nodes are up and active.

2. Run the ping command as shown below:

   ```
   kv-> ping
   ```

Output:

```
Pinging components of store mystore based upon topology sequence #2
0 partitions and 1 storage nodes
Time: 2024-04-05 11:36:06 UTC   Version: 24.1.11
Shard Status: healthy: 0 writable-degraded: 0 read-only: 0 offline: 0
total: 0
Admin Status: healthy
Zone [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
RN Status: online: 0 read-only: 0 offline: 0
Storage Node [sn1] on <hostname>: 5000
Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:25:44 UTC  Build id:
477e7f102ab4
Edition: Enterprise  isMasterBalanced: unknown serviceStartTime:
2024-04-05 10:37:28 UTC
Admin [admin1]  Status: RUNNING,MASTER  serviceStartTime: 2024-04-05
10:38:21 UTC
stateChangeTime: 2024-04-05 10:38:21 UTC availableStorageSize: 999 MB
```

If the output shows all Storage Nodes and replication nodes as running without any errors, then the data store is configured well and all Storage Nodes are up and active.

If you run into installation problems or want to start over with a new data store, then on every Storage node in the data store:

1. Stop the Storage Node using:

   ```
   java -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar stop -root $KVROOT
   ```

2. Remove the contents of the KVROOT directory:

   ```
   rm -rf $KVROOT
   ```

3. Start over with the steps described in Installation Configuration Parameters.

Create a script to configure the data store

> **Note:**
>
> You must follow the configuration steps as mentioned in Configuring your data store installation before running the Admin CLI.

You now know how to configure a data store using an interactive command line interface session. However, you can collect all of the commands used in the prior sections into a script file, and then run the script in a single batch operation. To do this, use the `load` command in the command line interface. For example:

Using the `load -file` command line option:

```
ssh node01
> java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar runadmin -port 5000 -host $KVHOST \
-security $KVROOT/security/client.security \
load -file script.txt
```

Using directly the `load -file` command. First start `runadmin` to use the Admin command line interface (CLI) for administration purposes:

```
 java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar runadmin -port 5000 -host node01 \
-security $KVROOT/security/client.security


kv-> load -file <path to file>
```

Using this command you can load the named file and interpret its contents as a script of commands to be executed.

The file, `script.txt`, would contain content like the code snippet shown below. Note that the name of the store in this example is `mystore`.

```
### Begin Script ###
configure -name mystore
plan deploy-zone -name "Boston" -rf 3 -wait
plan deploy-sn -zn zn1 -host <hostname> -port 5000 -wait
plan deploy-admin -sn sn1 -wait
pool create -name BostonPool
pool join -name BostonPool -sn sn1
plan deploy-sn -zn zn1 -host <host2> -port 5000 -wait
pool join -name BostonPool -sn sn2
plan deploy-sn -zn zn1 -host <host3> -port 5000 -wait
pool join -name BostonPool -sn sn3
topology create -name topo -pool BostonPool -partitions 300
plan deploy-topology -name topo -wait
exit
### End Script ###
```

Troubleshooting

Typical errors when bringing up a data store are typos and misconfiguration. It is also possible to run into network port conflicts, especially if the deployment failed and you are starting over. Processes associated with a data store are reported by `jps -m` command. Some examples of them are :

- `kvstore.jar start -root $KVROOT` (SNA process)
- ManagedService

If you kill the SNA process it should also kill its managed processes.

There are detailed log files available in `$KVROOT/storename/log` as well as logs of the bootstrap process in `$KVROOT/*.log`. The bootstrap logs are most useful in diagnosing initial startup problems. The logs in `storename/log` appear once the data store has been configured. The logs on the Storage Node chosen for the admin process are the most detailed and include a store-wide consolidated log file: `$KVROOT/storename/log/storename_*.log`

Each line in the log file is prefixed with the date of the message, its severity, and the name of the component which issued it. For example:

```
2024-04-05 14:28:26.982 UTC INFO [admin1]
Initializing Admin for store: mystore
```

When looking for more context for events at a given time, use the timestamp and component name to narrow down the section of log to peruse.

Error messages in the logs show up with **SEVERE** in them so you can grep for that if you are troubleshooting. SEVERE error messages are also displayed in the CLI's `show events` command, and when you use the `ping` command.

In addition to log files, the log directories may also contain *.perf files, which are performance files for the replication nodes.

In general, `verify configuration` is the tool of choice for understanding the state of the data store. In addition to contacting the data store components, it will cross check each component's parameters against the Admin database. For example, `verify configuration` might report that a replication node's helperHosts parameter was at odds with the Admin. If this was the case then it might explain why a replication node cannot come up. The `Verify configuration` tool also checks on Admins. It also verifies the configuration of Arbiter Nodes in the topology.

Additionally, in order to catch configuration errors early, you can use the diagnostics tool when troubleshooting your data store. Also, you can use this tool to package important information and files to be able to send them to Oracle Support. For more information, see Diagnostics Utility.

Where to Find Error Information

As your data store operates, you can discover information about any problems that may be occurring by looking at the plan history and by looking at error logs.

The plan history indicates if any configuration or operational actions you attempted to take against the store encountered problems. This information is available as the plan executes and finishes. Errors are reported in the plan history each time an attempt to run the plan fails. The plan history can be seen using the CLI `show plan` command.

Other problems may occur asynchronously. You can learn about unexpected failures, service downtime, and performance issues through the CLI's `show events` command. Events come with a time stamp, and the description may contain enough information to diagnose the issue. In other cases, more context may be needed, and the administrator may want to see what else happened around that time.

The store-wide log consolidates logging output from all services. Browsing this file might give you a more complete view of activity during the problem period. It can be viewed using the CLI's `logtail` command, or by directly viewing the <storename>_N.log file in the $KVHOME/<storename>/log directory.

Service States

Oracle NoSQL Database uses four different types of services, all of which should be running correctly in order for your store to be in a healthy state. The four service types are the Admin, Storage Nodes, Replication Nodes and Arbiters Nodes. You should have multiple instances of these services running throughout your store.

Each service has a status that can be viewed using any of the following:

- The `show topology` command in the Administration CLI.
- Using the `ping` command.

The status values can be one of the following:

Name	Description
ERROR_NO_RESTART	The service is in an error state and is not automatically restarted. Administrative intervention is required.
ERROR_RESTARTING	The service is in an error state. Oracle NoSQL Database attempts to restart the service.
RUNNING	The service is running normally.
STARTING	The service is coming up.
STOPPED	The service was stopped intentionally and cleanly.
STOPPING	The service is stopping. This may take some time as some services can be involved in time-consuming activities when they are asked to stop.
SUCCEEDED	The plan has completed successfully.
UNREACHABLE	The service is not reachable by the Admin. If the status was seen using a command issued by the Admin, this state may mask a STOPPED or ERROR state. If an SN is UNREACHABLE, or an RN is having problems and its SN is UNREACHABLE, the first thing to check is the network connectivity between the Admin and the SN. However, if the managing SNA is reachable and the managed Replication Node is not, we can guess that the network is OK and the problem lies elsewhere.
WAITING_FOR_DEPLOY	The service is waiting for commands or acknowledgments from other services during its startup processing. If it is a Storage Node, it is waiting for the initial deploy-SN command. Other services should transition out of this phase without any administrative intervention from the user.

A healthy service begins with `STARTING`. It may transition to `WAITING_FOR_DEPLOY` for a short period before going on to `RUNNING`.

`ERROR_RESTARTING` and `ERROR_NO_RESTART` indicate that there has been a problem that should be investigated. An `UNREACHABLE` service may only be in that state temporarily, although if that state persists, the service may be truly in an `ERROR_RESTARTING` or `ERROR_NO_RESTART` state.

Useful Commands

The following commands may be useful to you when troubleshooting your KVStore.

- ```
  java -Xmx64m -Xms64m \
  -jar kvstore.tmp/kvstore.jar ping -host node01 -port 5000 \
  -security USER/security/admin.security
  ```

  Reports the status of the store running on the specified host and port. This command can be used against any of the host and port pairs used for Storage Nodes.

  > **Note:**
  >
  > This assumes that you have completed the steps in Create users and configure security with remote access .

- `jps -m`

  Reports the Java processes running on a machine. If the Oracle NoSQL Database processes are running, they are reported by this command.

- `ps -eaf | grep kv`

  You can view the list of kvstore processes that are running.

Configure data store - Advanced scenarios

Create Additional Admin Processes

If you have deployed more than one Storage Node, you can add additional Admin processes using the `deploy-admin` plan. You are responsible for creating the appropriate number of Admins.

For example, currently you have a single Admin process deployed in your data store. So far, this has been sufficient to proceed with the data store configuration. However, to increase your data store's reliability, you should deploy multiple Admin processes, each running on a different Storage Node. This way, you can continue to administer your data store even if one Storage Node becomes unreachable and ends its Admin process. Having multiple Admin processes also means that you can continue to monitor your data store, even if you lose a Storage Node that is running an Admin process.

Create the Admin process on a Storage Node you just deployed, using the `plan deploy-admin` command. This command requires the Storage Node ID, which you can get from the `show topology` command:. Below is an example.

```
kv-> show topology
```

Output:

```
store=mystore  numPartitions=0 sequence=2
zn: id=zn1 name=Boston repFactor=1 type=PRIMARY allowArbiters=false
masterAffinity=false
```
```

```
sn=[sn1] zn:[id=zn1 name=Boston] <hostname>.oraclevcn.com:5000 capacity=1
RUNNING
numShards=0


kv-> plan deploy-admin -sn sn1 -wait
```

Output:

```
Executed plan 3, waiting for completion...
Plan 3 ended successfully
```

Although Admins are not required for normal data operations on the data store, they are needed to perform various administrative operations, including DDL operations. For example to create or modify tables, and for security operations involving users and roles. It is very important that the Admin services remain available.

Consideration for Admin Quorum
The full availability of the Admin service depends on having a quorum of the total Admin services available at a given time. Having a quorum of Admins operates similarly to the quorum for replication nodes in a shard. For replication nodes, the replication factor controls how many replication nodes can fail and still maintain the service. For example, with a replication factor of 3, the following table describes how failure numbers affect availability:

Failures	Availability
0	Full
1	Full
2	Read-only
3	None

The same failure and availability values exist for Admins. It is strongly recommended that you use the store replication factor to determine how many Admins should exist. This means that the Admin service has the same availability as the data store does for data operations. It is recommended that you use 3 Admins (matching the typical replication factor).

As with the store replication factor, when you use an even number of replicas, to maintain quorum (which is majority of the total number), you need more than half of the total replicas to be available. That means for a total of 4 replicas you need at least 2 replicas to be available to maintain quorum. For example, a replication factor of 4 has this behavior with failures and availability:

Failures	Availability
0	Full
1	Full
2	Read-only
3	Read-only
4	None

So, with a replication factor of 4, the group can still tolerate only a single failure and maintain full availability. Moreover, in addition to the higher Replication Factor value having no benefit during failures, now one more node exists that can fail, and the chance of losing quorum increases. The replication factor described here are for primary Storage Nodes associated with primary zones. For data stores with secondary zones, the nodes in the secondary zones are not included in the quorum.

Available Admins in Zones
Making sure that Admins are available in the right zones is another important consideration. If a data store has multiple primary zones, the zones were presumably set up to provide better availability. In this case, the admins should reflect the same arrangement. It is recommended that each zone has the same number of admins as the zone's replication factor. Unlike replication nodes, where all nodes in the shard can handle read operations, only the admin master responds to admin operations (unless there is no master). So, putting admins in a secondary zone is only useful to support failure recovery.

For example, if a store has primary and secondary zones, and all of the primary zones are lost, the administrator can use the `repair-admin-quorum` and `plan failover` commands to resume operations by converting the secondary zone to a primary zone. But these operations can occur only if an Admin node is available. For this reason, stores with secondary zones should include Admins in the secondary zones.

Configuring with Multiple Zones

To achieve optimal use of all available physical facilities, deploy your data store across multiple zones. Multiple zones provide fault isolation and availability for your data if a single zone fails. Each zone has a copy of your complete data store, including a copy of all the shards. With this configuration, reads are always possible, as long as your data's consistency guarantees can be met, because at least one replica is located in every zone. Writes can also occur in the event of a zone loss, as long as the database maintains quorum. See *Concepts Guide*.

You can specify a different replication factor to each zone. A replication factor is quantified as one of the following:

Zone Replication Factor
The number of copies, or replicas, maintained in a zone.

Primary Replication Factor
The total number of replicas in all primary zones. This replication factor controls the number of replicas that participate in elections and acknowledgments. For additional information on how to identify the **Primary Replication Factor** and its implications, see Replication Factor.

Secondary Replication Factor
The total number of replicas in all secondary zones. Secondary replicas provide additional read-only copies of the data.

Store Replication Factor
The total number of replicas across the entire data store.

Zones that are located near each other physically benefit by avoiding bottlenecks from throughput limitations, and by reducing latency during elections and commits.

> **Note:**
>
> There are two types of zones: Primary, and Secondary.
>
> *Primary* zones contain nodes which can serve as masters or replicas. Zones are created as primary Zones by default. For good performance, primary zones should be connected by low latency networks so that they can participate efficiently in master elections and commit acknowledgments. Primary zones can also become Master Affinity zones.
>
> *Secondary* zones contain nodes which can only serve as replicas. Secondary zones can be used to provide low latency read access to data at a distant location, or to maintain an extra copy of the data to increase redundancy or increase read capacity. Because the nodes in secondary zones do not participate in master elections or commit acknowledgments, secondary zones can be connected to other zones by higher latency networks, because additional latency will not interfere with those time critical operations.

Using high throughput and low latency networks to connect primary zones leads to better results and improved performance. You can use networks with higher latency to connect to secondary zones so long as the connections provide sufficient throughput to support replication and sufficient reliability that temporary interruptions do not interfere with network throughput.

> **Note:**
>
> Because any primary zone can host master nodes, you can reduce write performance by connecting primary zones through a limited throughput or a high latency network link.

The following steps walk you through the process of deploying six Storage Nodes across three primary zones. You can then verify that each shard has a replica in every zone; service can be continued in the event of a zone failure. You will configure secure data store in all the six Storage Nodes. In the first Storage Node, security will be configured and the security directory and all files contained in it will be copied from the first Storage Node to other Storage Nodes to setup security.

Follow the steps below in the first Storage Node `(node01)`:

- Execute the following command:

```
java -jar $KVHOME/lib/kvstore.jar makebootconfig \
-root $KVROOT \
-port 5000 \
-host $KVHOST \
-harange 5010,5020 \
-store-security configure \
-capacity 1 \
-storagedir ${KVDATA}/disk1 \
-storagedirsize 5500-MB \
```

- Start the Storage Node Agent:

```
java -jar $KVHOME/lib/kvstore.jar start -root $KVROOT &
```

- Create a zip file of all the security files created:

```
cd ;
zip -r $HOME/security.zip $KVROOT/security;
cd -
```

- Copy `$HOME/security.zip` from this node (`node01`) to the other five nodes.

Follow these steps in each of the other Storage Nodes (`node02`, `node03`, `node04`, `node05`, `node06`).

- Unzip the security files copied from the first Storage Node (`node01`).

```
cd;
unzip -o security.zip -d /;
cd -;
```

- Execute the following command:

```
java -jar $KVHOME/lib/kvstore.jar makebootconfig \
-root $KVROOT \
-port 5000 \
-host $KVHOST \
-harange 5010,5020 \
-store-security enable \
-capacity 1 \
-storagedir ${KVDATA}/disk1 \
-storagedirsize 5500-MB \
```

- Start the Storage Node Agent:

```
java -jar $KVHOME/lib/kvstore.jar start -root $KVROOT &
```

From the first Storage Node (`node01`) deploy your data store using the following steps:

- Start the Admin CLI. Here $KVHOST is `node01`.

```
java -Xmx64m -Xms64m \
-jar  $KVHOME/lib/kvstore.jar runadmin \
-port 5000 -host $KVHOST
-security $KVROOT/security/client.security
```

- Name your data store and deploy three primary zones:

```
configure -name MetroArea;
plan deploy-zone -name "Manhattan" -rf 1 -wait;
plan deploy-zone -name "JerseyCity" -rf 1 -wait;
plan deploy-zone -name "Queens" -rf 1 -wait;
```

- Deploy the first Storage Node with administration process in the Manhattan zone.

```
plan deploy-sn -znname Manhattan -host node01 -port 5000 -wait;
plan deploy-admin -sn sn1 -wait;
```

  Deploy a second Storage Node in Manhattan zone:

```
plan deploy-sn -znname Manhattan -host node02 -port 5000 -wait;
```

- Deploy the first Storage Node with administration process in the JerseyCity zone:.

```
plan deploy-sn -znname JerseyCity -host node03 -port 5000 -wait;
plan deploy-admin -sn sn3 -wait;
```

  Deploy a second Storage Node in JerseyCity zone:

```
plan deploy-sn -znname JerseyCity -host node04 -port 5000 -wait;
```

- Deploy the first Storage Node with administration process in the Queens zone:.

```
plan deploy-sn -znname Queens -host node05 -port 5000 -wait;
plan deploy-admin -sn sn5 -wait;
```

  Deploy a second Storage Node in Queens zone:

```
plan deploy-sn -znname JerseyCity -host node06 -port 5000 -wait;
```

- Create and deploy a topology:

```
topology create -name Topo1 -pool AllStorageNodes -partitions 300;
plan deploy-topology -name Topo1 -wait;
```

- Follow the instructions mentioned in Create users and configure security with remote access to create the access for the users in the multiple zones.
- Check service status with the `show topology` command:

```
kv-> show topology
```

  Output:

```
store=MetroArea  numPartitions=100 sequence=117
  zn: id=zn1 name=Manhattan repFactor=1 type=PRIMARY
   allowArbiters=false masterAffinity=false
  zn: id=zn2 name=JerseyCity repFactor=1 type=PRIMARY
   allowArbiters=false masterAffinity=false
  zn: id=zn3 name=Queens repFactor=1 type=PRIMARY
   allowArbiters=false masterAffinity=false

  sn=[sn1] zn:[id=zn1 name=Manhattan] node01:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
          No performance info available
  sn=[sn2] zn:[id=zn1 name=Manhattan] node02:5000 capacity=1 RUNNING
```

```
    [rg2-rn1] RUNNING
          No performance info available
  sn=[sn3] zn:[id=zn2 name=JerseyCity] node03:5000 capacity=1 RUNNING
    [rg1-rn2] RUNNING
          No performance info available
  sn=[sn4] zn:[id=zn2 name=JerseyCity] node04:5000 capacity=1 RUNNING
    [rg2-rn2] RUNNING
          No performance info available
  sn=[sn5] zn:[id=zn3 name=Queens] node05:5000 capacity=1 RUNNING
    [rg1-rn3] RUNNING
          No performance info available
  sn=[sn6] zn:[id=zn3 name=Queens] node06:5000 capacity=1 RUNNING
    [rg2-rn3] RUNNING
          No performance info available

  numShards=2
  shard=[rg1] num partitions=50
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn3
    [rg1-rn3] sn=sn5
  shard=[rg2] num partitions=50
    [rg2-rn1] sn=sn2
    [rg2-rn2] sn=sn4
    [rg2-rn3] sn=sn6
```

- Verify that each shard has a replica in every zone:

  ```
  kv-> verify configuration
  ```

  Output:

  ```
  Verify: starting verification of store MetroArea
  based upon topology sequence #117
  100 partitions and 6 storage nodes
  Time: 2024-04-05 10:41:15 UTC   Version: 24.1.11
  See node01:
  $KVROOT/MetroArea/log/MetroArea_{0..N}.log
  for progress messages
  Verify: Shard Status: healthy:2
  writable-degraded:0 read-only:0 offline:0 total:2
  Verify: Admin Status: healthy
  Verify: Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]   RN Status: online:2 read-only:0 offline:0
  Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
  masterAffinity=false]   RN Status: online:2 read-only:0 offline:0
  maxDelayMillis:1 maxCatchupTimeSecs:0
  Verify: Zone [name=Queens id=zn3 type=PRIMARY allowArbiters=false
  masterAffinity=false]   RN Status: online:2 read-only:0 offline:0
  maxDelayMillis:4 maxCatchupTimeSecs:0
  Verify: == checking storage node sn1 ==
  Verify: Storage Node [sn1] on node01:5000
  Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
  masterAffinity=false]
  Status: RUNNING   Ver: 24.1.11 2024-04-05 10:41:15 UTC
  ```

```
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Admin [admin1]        Status: RUNNING,MASTER
Verify:     Rep Node [rg1-rn1]    Status: RUNNING,MASTER
sequenceNumber:1,261 haPort:5011 available storage size:31 GB
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on node02:5000
Zone: [name=Manhattan id=zn1 type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:41:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Rep Node [rg2-rn1]    Status: RUNNING,MASTER
sequenceNumber:1,236 haPort:5012 available storage size:31 GB
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:5000
Zone: [name=JerseyCity id=zn2 type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:41:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Admin [admin2]        Status: RUNNING,REPLICA
Verify:     Rep Node [rg1-rn2]    Status: RUNNING,REPLICA
sequenceNumber:1,261 haPort:5011 available storage size:31 GB
delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:5000
Zone: [name=JerseyCity id=zn2 type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:41:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Rep Node [rg2-rn2]    Status: RUNNING,REPLICA
sequenceNumber:1,236 haPort:5012 available storage size:31 GB
delayMillis:0 catchupTimeSecs:0
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:5000
Zone: [name=Queens id=zn3 type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:41:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Admin [admin3]        Status: RUNNING,REPLICA
Verify:     Rep Node [rg1-rn3]    Status: RUNNING,REPLICA
sequenceNumber:1,261 haPort:5011 available storage size:31 GB
delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:5000
Zone: [name=Queens id=zn3 type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:41:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Rep Node [rg2-rn3]    Status: RUNNING,REPLICA
sequenceNumber:1,236 haPort:5012 available storage size:31 GB
delayMillis:4 catchupTimeSecs:0

Verification complete, no violations.
```

In the above example there are three zones (zn1 = Manhattan, zn2 = JerseyCity, zn3=Queens) with six replication nodes (two masters and four replicas) in the data store. This means that this topology is not only highly available because you have three replicas within each shard,

but it is also able to recover from a single zone failure. If any zone fails, the other two zones are enough to elect the new master, so service continues without any interruption.

Adding Secondary Zone to the Existing Topology

This section shows how to add a secondary zone to an existing topology that was created in Configuring with Multiple Zones . The following example adds a secondary zone in a different geographical location, Europe, allowing the users to read the data from the secondary zone either because it is physically located closer to the client or because the primary zone in the New York metro area is unavailable due to a disaster. The steps involve creating and starting two new Storage Nodes with capacity 1, creating a secondary zone, deploying the new Storage Nodes in the secondary zone, and doing a redistribute of the topology so that a replica for each shard is placed in the secondary zone.

Follow these steps in both the new Storage Nodes (`node07` and `node08`).

1. Copy the security zipped files from the first node and unzip the files.

   ```
   unzip -o security.zip -d /;
   ```

2. Invoke the `makebootconfig` utility for the first new Storage Node that will be deployed in the Frankfurt zone. The security configuration will be enabled while invoking the `makebootconfig` utility.

   ```
   java -jar $KVHOME/lib/kvstore.jar makebootconfig \
   -root $KVROOT \
   -port 5000 \
   -host $KVHOST \
   -harange 5010,5020 \
   -store-security enable \
   -capacity 1 \
   -storagedir ${KVDATA}/disk1 \
   -storagedirsize 5500-MB
   ```

3. Start the Storage Node Agent.

   ```
   java -jar $KVHOME/lib/kvstore.jar start -root $KVROOT &
   ```

To create a secondary zone and deploy the new Storage Nodes, do the following steps:

1. Start the Admin CLI. Here $KVHOST is `node01`.

   ```
   java -Xmx64m -Xms64m \
   -jar  $KVHOME/lib/kvstore.jar runadmin \
   -port 5000 -host $KVHOST
   -security $KVROOT/security/client.security
   ```

2. Create a secondary zone in Frankfurt.

   ```
   kv-> plan deploy-zone -name Frankfurt -rf 1 -type secondary -wait
   ```

Output:

```
Executed plan 14, waiting for completion...
Plan 14 ended successfully
```

3. Deploy Storage Node sn7 in the Frankfurt zone.

```
kv-> plan deploy-sn -znname Frankfurt -host node07 -port 5000 -wait
```

Output:

```
Executed plan 15, waiting for completion...
Plan 15 ended successfully
```

4. Deploy the Storage Node sn7 with administration process in the Frankfurt zone.

```
kv-> plan deploy-admin -sn sn7 -wait
```

Output:

```
Executed plan 16, waiting for completion...
Plan 16 ended successfully
```

5. Deploy Storage Node sn8 in the Frankfurt zone.

```
kv-> plan deploy-sn -znname Frankfurt -host node08  -port 5000 -wait
```

Output:

```
Executed plan 17, waiting for completion...
Plan 17 ended successfully
```

6. Do redistribute and then deploy the new topology to create one replica for every shard in the secondary Frankfurt zone.

```
kv-> topology clone -current -name topo_secondary
```

Output:

```
Created topo_secondary


kv-> topology redistribute -name topo_secondary -pool AllStorageNodes
```

Output:

```
Redistributed: topo_secondary


kv-> topology preview -name topo_secondary
```

Output:

```
Topology transformation from current deployed topology to topo_secondary:
Create 2 RN shard rg1 1 new RN : rg1-rn4
            shard rg2 1 new RN : rg2-rn4


kv-> plan deploy-topology -name topo_secondary -wait
```

Output:

```
Executed plan 19, waiting for completion...
Plan 19 ended successfully
```

7. Follow the instructions mentioned in Create users and configure security with remote access to copy user security files in the new Storage Nodes created.
8. Check service status with the show topology command.

```
kv-> show topology
```

Output:

```
store=MetroArea  numPartitions=100 sequence=120
  zn: id=zn1 name=Manhattan repFactor=1 type=PRIMARY
    allowArbiters=false masterAffinity=false
  zn: id=zn2 name=JerseyCity repFactor=1 type=PRIMARY
    allowArbiters=false masterAffinity=false
  zn: id=zn3 name=Queens repFactor=1 type=PRIMARY
    allowArbiters=false masterAffinity=false
  zn: id=zn4 name=Frankfurt repFactor=1 type=SECONDARY
    allowArbiters=false masterAffinity=false

  sn=[sn1] zn:[id=zn1 name=Manhattan] node01:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
    single-op avg latency=0.21372496 ms multi-op avg latency=0.0 ms
  sn=[sn2] zn:[id=zn1 name=Manhattan] node02:5000 capacity=1 RUNNING
    [rg2-rn1] RUNNING
    single-op avg latency=0.30840763 ms multi-op avg latency=0.0 ms
  sn=[sn3] zn:[id=zn2 name=JerseyCity] node03:5000 capacity=1 RUNNING
    [rg1-rn2] RUNNING
    No performance info available
  sn=[sn4] zn:[id=zn2 name=JerseyCity] node04:5000 capacity=1 RUNNING
    [rg2-rn2] RUNNING
    No performance info available
  sn=[sn5] zn:[id=zn3 name=Queens] node05:5000 capacity=1 RUNNING
    [rg1-rn3] RUNNING
    No performance info available
  sn=[sn6] zn:[id=zn3 name=Queens] node06:5000 capacity=1 RUNNING
    [rg2-rn3] RUNNING
    No performance info available
  sn=[sn7] zn:[id=zn4 name=Frankfurt] node07:5000 capacity=1 RUNNING
    [rg1-rn4] RUNNING
    No performance info available
  sn=[sn8] zn:[id=zn4 name=Frankfurt] node07:5000 capacity=1 RUNNING
```

```
    [rg2-rn4] RUNNING
    No performance info available

  numShards=2
  shard=[rg1] num partitions=50
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn3
    [rg1-rn3] sn=sn5
    [rg1-rn4] sn=sn7
  shard=[rg2] num partitions=50
    [rg2-rn1] sn=sn2
    [rg2-rn2] sn=sn4
    [rg2-rn3] sn=sn6
    [rg2-rn4] sn=sn8
```

9. Verify that the secondary zone has a replica for each shard.

```
kv-> verify configuration
```

Output:

```
Verify: starting verification of store MetroArea
based upon topology sequence #120
100 partitions and 7 storage nodes
Time: 2024-04-05 10:52:15 UTC   Version: 24.1.11
See node01:
$KVROOT/Disk1/MetroArea/log/MetroArea_{0..N}.log
for progress messages
Verify: Shard Status: healthy:2
writable-degraded:0 read-only:0 offline:0 total:2
Verify: Admin Status: healthy
Verify: Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]   RN Status: online:2 read-only:0 offline:0
Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]   RN Status: online:2 read-only:0 offline:0
maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]   RN Status: online:2 read-only:0 offline:0
maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=Frankfurt id=zn4 type=SECONDARY allowArbiters=false
masterAffinity=false]   RN Status: online:1 read-only:0 offline:0
maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on node01:5000
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Admin [admin1]        Status: RUNNING,MASTER
Verify:     Rep Node [rg1-rn1]    Status: RUNNING,MASTER
sequenceNumber:1,261 haPort:5011 available storage size:31 GB
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on node02:5000
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
```

```
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Rep Node [rg2-rn1]    Status: RUNNING,MASTER
sequenceNumber:1,236 haPort:5012 available storage size:31 GB
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:5000
Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Admin [admin2]        Status: RUNNING,REPLICA
Verify:     Rep Node [rg1-rn2]    Status: RUNNING,REPLICA
sequenceNumber:1,261 haPort:5011 available storage size:31 GB
delayMillis:0 catchupTimeSecs:0
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:5000
Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Rep Node [rg2-rn2]    Status: RUNNING,REPLICA
sequenceNumber:1,236 haPort:5012 available storage size:31 GB
delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:5000
Zone: [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Admin [admin3]        Status: RUNNING,REPLICA
Verify:     Rep Node [rg1-rn3]    Status: RUNNING,REPLICA
sequenceNumber:1,261 haPort:5011 available storage size:31 GB
delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:5000
Zone: [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING  Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Rep Node [rg2-rn3]    Status: RUNNING,REPLICA
sequenceNumber:1,236 haPort:5012 available storage size:31 GB
delayMillis:0 catchupTimeSecs:0
Verify: == checking storage node sn7 ==
Verify: Storage Node [sn7] on node07:5000
Zone: [name=Frankfurt id=zn4 type=SECONDARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify:     Admin [admin4]        Status: RUNNING,REPLICA
Verify:     Rep Node [rg1-rn4]    Status: RUNNING,REPLICA
sequenceNumber:1,261 haPort:5011 available storage size:31 GB
delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn8 ==
Verify: Storage Node [sn8] on node08:5000
Zone: [name=Frankfurt id=zn4 type=SECONDARY allowArbiters=false
masterAffinity=false]
```

```
Status: RUNNING Ver: 24.1.11 2024-04-05 10:52:15 UTC
Build id: c8998e4a8aa5 Edition: Enterprise
Verify: Rep Node [rg2-rn4] Status: RUNNING,REPLICA
sequenceNumber:1,238 haPort:5012 available storage size:31 GB
delayMillis:0 catchupTimeSecs:0

Verification complete, no violations.
```

Oracle NoSQL Database Proxy

Learn how to set up Oracle NoSQL Database Proxy in Oracle NoSQL Database.

Topics:

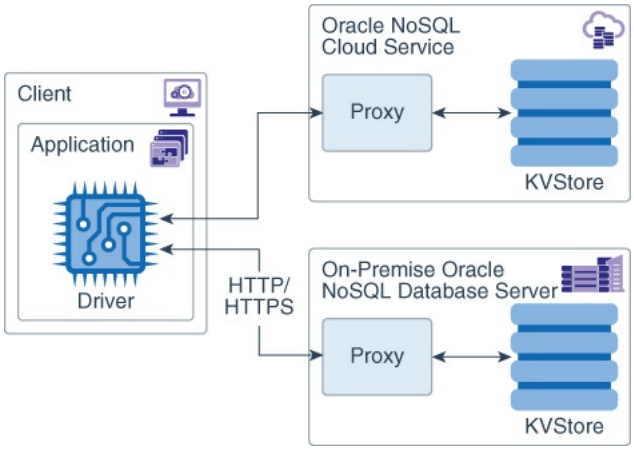
About the Oracle NoSQL Database Proxy

The Oracle NoSQL Database Proxy is a middle-tier component that lets the Oracle NoSQL Database drivers communicate with the Oracle NoSQL Database data store. The Oracle NoSQL Database drivers are available in various programming languages that are used in the client application. Currently, Java, Python, Go, Node.js, C#, Rust, and Spring Data language drivers are supported.

The Oracle NoSQL Database Proxy is a server that accepts requests from Oracle NoSQL Database drivers and processes them using the Oracle NoSQL Database. The Oracle NoSQL Database drivers can be used to access either the Oracle NoSQL Database Cloud Service or an on-premises installation via the Oracle NoSQL Database Proxy. Since the drivers and APIs are identical, applications can be moved between these two options. However, an application connecting simultaneously to both the on-premises and Oracle NoSQL Database Cloud Service is not recommended.

For example, you can deploy a local Oracle NoSQL Database data store first for a prototype project, and move forward to Oracle NoSQL Database Cloud Service for a production project.

Figure 3-1 Oracle NoSQL Database Proxy and Driver



The JAR file for the Oracle NoSQL Database Proxy is included in the Enterprise Edition distribution and the Community Edition distribution of Oracle NoSQL Database. Users can download the JAR for the Oracle NoSQL Database Proxy from the Oracle Technology Network.

Configuring the Proxy

You configure the Oracle NoSQL Database Proxy after deploying a data store.

Obtain the following information from the secure data store deployment:

- data store name. See ping.
- data store helper host:port list. See Obtaining a KVStore Handle in the *Java Direct Driver Developer's Guide*.

Here is the general usage to start the proxy:

```
java -jar lib/httpproxy.jar [-async <default: false>] [-config <>] [-
helperHosts <>] [-hostname <>] [-httpPort <default: 80>] [-httpsPort
<default: 443>] [-idleReadTimeout <default: 0>] [-kvConsistency <default:
NONE_REQUIRED>] [-kvDurability <default: COMMIT_NO_SYNC>] [-kvRequestTimeout
<default: -1>] [-monitorStatsEnabled <default: false>] [-numAcceptThreads
<default: 3>] [-numRequestThreads <default: 32>] [-sslCertificate <>] [-
sslCiphers <>] [-sslPrivateKey <>] [-sslPrivateKeyPass <>] [-sslProtocols
<default: TLSv1.2,TLSv1.1,TLSv1>] [-sslSecurityDir <>] [-storeName <>] [-
storeSecurityFile <>] [-verbose <default: false>]
```

Proxy Parameters

You can provide the following parameters as the command line arguments to start the proxy.

Parameter	Required ?	Default Value	Description
-async	No	false	Defines whether or not the proxy uses the asynchronous interfaces of the data store. This option instructs the proxy to use asynchronous requests, which can reduce the number of processing threads needed.

-config	No	No Default value	Specifies a Java properties file that contains the proxy configuration options. Supply the options in the following format:
			
			`property_name=value`
			
			For example:
			
			`async=true`
			`storeName=mystore`
			`helperHosts=node01:5000`
			`httpPort=8080`
			`monitorStatsEnabled=true`
			
			You can pass this configuration file as a single parameter to the `httpproxy.jar` command from the CLI.
			
			Note:
			Ensure that you include all the options marked as *Required* in this table.

-idleReadTimeout	No	30	Specifies the time duration (in seconds) for the server to terminate the unused connections.

Parameter	Required ?	Default Value	Description
-kvConsistency	No	NONE_REQUIRED	Configures the default read consistency used for this session. The Oracle NoSQL Database Proxy uses this parameter if the request does not specify a consistency. The consistency values are defined in the Consistency class of Java APIs. The read operations are serviced either on a master or a replica node depending on the configured value. The following policies are supported: • ABSOLUTE - The read operation is serviced on a master node. With ABSOLUTE consistency, you are guaranteed to obtain the latest updated data. • NONE-REQUIRED - The read operation can be serviced on a replica node. Data read from the replica can be older than data on the master. For more details on consistency, see Consistency Guarantees.
-kvDurability	No	COMMIT_NO_SYNC	Configures the default write durability setting used in this session. The `kvDurability` parameter defines the durability policy to be used by a master when committing a transaction. The Oracle NoSQL Database Proxy uses the `kvDurability` parameter if the request does not specify a durability. • COMMIT_NO_SYNC - The data is written to the host's in-memory cache, but the master node does not wait for the data to be written to the file system's data buffers or subsequent physical storage. • COMMIT_SYNC - The data is written to the in-memory cache, transferred to the file system's data buffers, and then synchronized to a stable storage before the write operation completes normally. • COMMIT_WRITE_NO_SYNC - The data is written to the in-memory cache, and transferred to the file system's data buffers, but not necessarily into physical storage. For more details on durability, see Durability Guarantees.

Parameter	Required ?	Default Value	Description
-kvRequestTimeout	No	-1	Configures the default request timeout used for this session (in milliseconds). The Oracle NoSQL Database Proxy adjusts this parameter on a specific request depending on the timeout specified in the request. Default value of -1 configures a timeout value of 5000.
-monitorStatsEnabled	No	false	Enables statistics collection in the proxy. If enabled, the statistics are collected in the log file *proxy_metric.log*.
-sslCiphers	No	No Default value	Advanced configuration related to SSL.
-sslProtocols	No	TLSv1.2,TLSv1.1,TLSv1	Advanced configuration related to SSL.
-sslSecurityDir	No	No Default value	Advanced configuration related to SSL.
-helperHosts	Required	No Default value	Specifies the host name and port pairs that identify how to contact helper nodes within the data store. Use an array of strings to identify multiple helper hosts . Typically, you will get these host name and port pairs from the data store's deployer or administrator. Example pattern: `hostname1:port1,hostname2:port2,...hostnameX:portX` Confirm that the ports in helper host list are left open by the firewall rules for connection between the proxy and data store server.
-storeName	Required	No Default value	Specifies the name of the data store. you can obtain this name from the data store deployment process.
-hostname	No	localhost	Specifies the host name of the machine which is starting up the proxy instance. By default, the proxy listens on all the available network interfaces on the host. Specifying a host name allows the proxy to listen on a specific interface.

Parameter	Required ?	Default Value	Description
-httpPort	No	80	Specifies the HTTP port of the proxy machine, which the proxy uses to accept non-secure connections from HTTP requests. This parameter is mutually exclusive with the `-httpsPort` parameter. Only one of these parameters can be specified. Confirm that the port is left open by the firewall rules for connection between the proxy and the driver. **Note:** Using port 80 requires root privilege. You can use port **8080** if you do not have root privilege.
-httpsPort	No	443	Specifies the HTTPS port of the proxy machine, which the proxy uses to accept secure connections from HTTPS requests. This parameter is mutually exclusive with the `-httpPort` parameter. Only one of these parameters can be specified. Confirm that the port is left open by the firewall rules for connection between the proxy and the driver. **Note:** Using port 443 requires root privilege. You can use port **8443** if you do not have root privilege.
-numAcceptThreads	No	3	This value determines the thread pool size for the threads that are used to handle the incoming connections to the proxy.

Parameter	Required ?	Default Value	Description
-numRequestThreads	No	32	Determines the thread pool size for the threads that are used to handle the request input/output traffic, after the connection has been registered by the "AcceptThread" and handed over to the "RequestThread".
-verbose	No	false	Displays the proxy start-up information. Can take either "true" or "false" as values.
-sslCertificate	Required for secure proxy only.	No Default value	Specifies the path to the SSL certificate file in pem file format. You can either generate a self-signed certificate using OpenSSL, or send a request to a public CA to generate a certificate. See Generating Certificate and Private Key for the Oracle NoSQL Database Proxy in the *Security Guide*.

Note:

The path to the SSL certificate file can be an absolute path or a path relative to the current directory (from where the proxy is started).

Parameter	Required ?	Default Value	Description
-sslPrivateKey	Required for secure proxy only.	No Default value	Specifies the path to the SSL private key file. You can either generate a private key using OpenSSL, or send a request to a public CA to generate a private key. See Generating Certificate and Private Key for the Oracle NoSQL Database Proxy in the *Security Guide*. **Note:** The path to the SSL private key file can be an absolute path or a path relative to the current directory (from where the proxy is started).
-sslPrivateKeyPass	Required for secure proxy only.	No Default value	Specifies the password for the private key, if the private key is encrypted. This parameter is not required if the private key is not encrypted.

Parameter	Required ?	Default Value	Description
-storeSecurityFile	Required for secure proxy only.	No Default value	Specifies the path to the security log in file which is generated by the client user of the data store. The client user of the data store should be a non-admin proxy user.

Note:

Note: The path to the store security file can be an absolute path or a path relative to the current directory (from where the proxy is started).

Note:

The Oracle NoSQL Database Proxy can run in one or multiple dedicated hosts. It can be hosted inside the nodes of the data store. You can use a load balancer as the front end, which has a back end set of multiple NoSQL proxies on different hosts. While configuring a load balancer, you can add an HTTP health check. The Oracle NoSQL Database Proxy provides the following URI **/V2/health** for the HTTP health check. An HTTP request to this URI returns a successful response **200 OK**. You can find an example for configuring HA proxy in the GitHub.

Using the Proxy in a non-secure data store

Starting up the Proxy

Use the following command to start up the proxy for a non-secure data store.

```
java -jar lib/httpproxy.jar \
-storeName <kvstore_name> \
-helperHosts <kvstore_helper_host> \
[-hostname <proxy_host>] \
[-httpPort <proxy_http_port>]
```

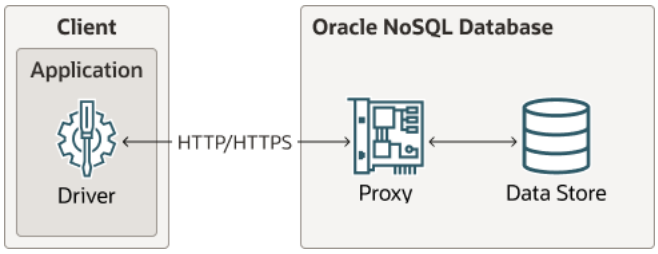
where,

- `kvstore_name` is the data store name obtained from the data store deployment. See ping.

- `kvstore_helper_host` is the data store's helper host:port list obtained from the data store deployment. See Obtaining a KVStore Handle in the *Java Direct Driver Developer's Guide*.
- `proxy_host` is the hostname of the machine to host the proxy service. This parameter is optional and defaults to `localhost`. You can also specify the complete hostname of the machine running the proxy.
- `proxy_http_port` is the port on which the proxy is listening for requests. This is an optional parameter and defaults to 80.

> **Note:**
>
> Using port 80 requires root privilege. You can use port **8080** if you do not have root privilege.

Connecting an application to the non-secure data store



Oracle NoSQL Database drivers are available in various programming languages that are used in the client application. Currently, Java, Python, Go, Node.js, C#, and Rust are supported. Oracle NoSQL Database Proxy is a server that accepts requests from the client application and processes them using the Oracle NoSQL Database.

- Java
- Python
- Go
- Node.js
- C#
- Rust

Java

The Oracle NoSQL Database Java Driver contains the jar files that enable a Java application to communicate with the proxy.

Install the Java driver in the application's classpath and use the following code to connect to the proxy.

```
String endpoint = "http://<proxy_host>:<proxy_http_port>";
StoreAccessTokenProvider atProvider = new StoreAccessTokenProvider();
```

```
NoSQLHandleConfig config = new NoSQLHandleConfig(endpoint);
config.setAuthorizationProvider(atProvider);
NoSQLHandle handle = NoSQLHandleFactory.createNoSQLHandle(config);
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the host you configured earlier.
- `proxy_http_port` is the port on which the proxy is listening for requests. This should match the http port you configured earlier.

Python

The on-premises configuration requires a running instance of the Oracle NoSQL database. In addition a running proxy service is required.

If the data store is not secure, an empty instance of `borneo.kv.StoreAccessTokenProvider` is used. For example:

```
from borneo import NoSQLHandle, NoSQLHandleConfig
from borneo.kv import StoreAccessTokenProvider
endpoint = 'http://<proxy_host>:<proxy_http_port>'
# Create the AuthorizationProvider for a not secure store:
ap = StoreAccessTokenProvider()
# create a configuration object
config = NoSQLHandleConfig(endpoint).set_authorization_provider(ap)
# create a handle from the configuration object
handle = NoSQLHandle(config)
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the host you configured earlier.
- `proxy_http_port` is the port on which the proxy is listening for requests. This should match the http port you configured earlier.

Go

The on-premises configuration requires a running instance of the Oracle NoSQL database. In addition a running proxy service is required. In this case, the `Endpoint` config parameter should point to the NoSQL proxy host and port location.

Use the following code to connect to the proxy.

```
...cfg:= nosqldb.Config{
    // EDIT: set desired endpoint for the Proxy server accordingly in your
environment.
    Endpoint: "http:<proxy_host>:<proxy_http_port>",
    Mode:     "onprem",
}
client, err:=nosqldb.NewClient(cfg)
iferr!=nil {
    fmt.Printf("failed to create a NoSQL client: %v\n", err)
    return
}
```

```
deferclient.Close()
// Perform database operations using client APIs.// ...
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the host you configured earlier.
- `proxy_http_port` is the port on which the proxy is listening for requests. This should match the http port you configured earlier.

Node.js

Your application will connect and use a running NoSQL database via the proxy service.

In non-secure mode, the driver communicates with the proxy via the HTTP protocol. The only information required is the communication `endpoint`. For on-premise NoSQL Database, the endpoint specifies the url of the proxy, in the form `http://proxy_host:proxy_http_port`

Use the following code to connect to the proxy.

```
const NoSQLClient = require('oracle-nosqldb').NoSQLClient;
const ServiceType = require('oracle-nosqldb').ServiceType;
const client = new NoSQLClient({
    serviceType: ServiceType.KVSTORE,
    endpoint: '<proxy_host>:<proxy_http_port>'
});
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the host you configured earlier.
- `proxy_http_port` is the port on which the proxy is listening for requests. This should match the http port you configured earlier.

You may also choose to store the same configuration in a file. Create file config.json with following contents:

```
{
    "serviceType": "KVSTORE",
    "endpoint": "<proxy_host>:<proxy_http_port>",
}
```

Then you may use this file to create NoSQLClient instance:

```
const NoSQLClient = require('oracle-nosqldb').NoSQLClient;
const client = new NoSQLClient('</path/to/config.json>');
```

C#

Your application will connect and use a running NoSQL database via the proxy service.

In non-secure mode, the driver communicates with the proxy via the HTTP protocol. The only information required is the communication `endpoint`. For on-premise NoSQL Database, the endpoint specifies the url of the proxy, in the form `http://proxy_host:proxy_http_port`

To connect to the proxy in non-secure mode, you need to specify communication endpoint and the service type as `ServiceType.KVStore`. You can provide an instance of NoSQLConfig either directly or in a JSON configuration file.

```
var client = new NoSQLClient(
    new NoSQLConfig
    {
        ServiceType = ServiceType.KVStore,
        Endpoint = "<proxy_host>:<proxy_http_port>"
    });
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the host you configured earlier.
- `proxy_http_port` is the port on which the proxy is listening for requests. This should match the http port you configured earlier.

You may also choose to provide the same configuration in JSON configuration file. Create file config.json with following contents:

```
{
    "ServiceType": "KVStore",
    "Endpoint": "<proxy_host>:<proxy_http_port>"
}
```

Then you may use this file to create NoSQLClient instance:

```
var client = new NoSQLClient("</path/to/config.json>");
```

Rust

Your application will connect and use a running NoSQL database via the proxy service.

In non-secure mode, the driver communicates with the proxy via the HTTP protocol. The only information required is the communication endpoint. For on-premise NoSQL Database, the endpoint specifies the URL of the proxy, in the form *http://proxy_host:proxy_http_port*.

```
let handle = Handle::builder()
.mode(HandleMode::Onprem)?
.endpoint("http://<proxy_host>:<proxy_http_port>")?
.build().await?;
```

where

- `proxy_host` is the hostname of the machine running the proxy service. This should match the proxy host you configured earlier.
- `proxy_http_port` is the port on which the proxy is listening for requests. This should match the proxy http port configured earlier.

Using the Proxy in a secure data store

Starting up the Proxy

Configuring and starting the Oracle NoSQL Database Proxy is part of the data store administration. The proxy can be started on a secure data store using the following steps.

1. Before you start up the proxy, you need to create a user (`proxy_user`) as the proxy needs an identity to connect to the secure data store. This proxy user identity (`proxy_user`) is never used for actual data operations. It is only needed for the initial connection to the store.
   In SQL shell, the following command will create a bootstrap user for the proxy. See Developers Guide for getting started with SQL commands.

   Create the `proxy_user` as shown below:

   ```
   sql-> CREATE USER proxy_user IDENTIFIED BY "<proxyuser_password>"
   ```

2. Create a new password file to store the credentials needed to login as the database user (`proxy_user`).

   ```
   java -Xmx64m -Xms64m -jar lib/kvstore.jar securityconfig pwdfile \
   create -file $KVROOT/security/login.passwd

   java  -Xmx64m -Xms64m -jar lib/kvstore.jar securityconfig pwdfile \
   secret  -file $KVROOT/security/login.passwd -set -alias proxy_user
   ```

   > **Note:**
   >
   > The secret value to store (that you enter in the above step) must match the value of <`proxyuser_password`> that you have set in the previous step.

3. Create a login file `proxy.login` for the bootstrap user with the following information in it.

   ```
   oracle.kv.auth.username=<proxy_user>
   oracle.kv.auth.pwdfile.file=login.passwd
   oracle.kv.transport=ssl
   oracle.kv.ssl.trustStore=client.trust
   ```

   where,

   - `login.passwd` is the file to store the password value of the `proxy_user` user.
   - `client.trust` is the certificate trust file obtained from the data store deployment.

4. Self-signed certificates can be used to securely connect to the Oracle NoSQL Database Proxy. Use the openSSL command to generate the self-signed certificate and private key. When prompted, provide a secure passphrase of your choice for the certificate file.

   ```
   openssl req -x509 -days 365 -newkey rsa:4096 -keyout key.pem -out
   certificate.pem \
   -subj "/C=US/ST=CA/L=San/CN=${HOSTNAME}/emailAddress=xxxx.xxxx@oracle.com"
   ```

Convert the private key to `PKCS#8` format. When prompted, first enter the passphrase that you set in the previous step and then provide a secure password of your choice for the encryption. You will use the encryption password that you set here in the step below while starting the proxy.

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -in key.pem \
-out key-pkcs8.pem -v1 PBE-SHA1-3DES
```

Additionally, a `driver.trust` file is also required if you are using the Java driver. This `driver.trust` file is not required for other language drivers. To generate the `driver.trust` file, import the certificate to the Java keystore. When prompted, provide the keystore password.

```
keytool -import -alias example -keystore driver.trust -file certificate.pem
```

5. Use the following command to start up the proxy for a secure data store:

```
java -jar lib/httpproxy.jar \
-storeName <kvstore_name> \
-helperHosts <kvstore_helper_host> \
[-hostname <proxy_host>] \
[-httpsPort <proxy_https_port>] \
-storeSecurityFile $KVROOT/security/proxy.login \
-sslCertificate certificate.pem \
-sslPrivateKey key-pkcs8.pem \
-sslPrivateKeyPass <privatekey_password> \
[-verbose true]
```

where,

- `kvstore_name` is the data store name obtained from the data store deployment. See ping to obtain information about the runtime entities (Storage Nodes and Replication Nodes) of the data store..
- `kvstore_helper_host` is the data store's helper host:port list obtained from the data store deployment. See Obtaining a KVStore Handle in the *Java Direct Driver Developer's Guide*.
- `proxy_host` is the hostname of the machine running the proxy service. This parameter is optional and defaults to `localhost`. You can also specify the complete hostname of the machine running the proxy.
- `proxy_https_port` is the port on which the proxy is listening for requests. This is an optional parameter and defaults to 443.

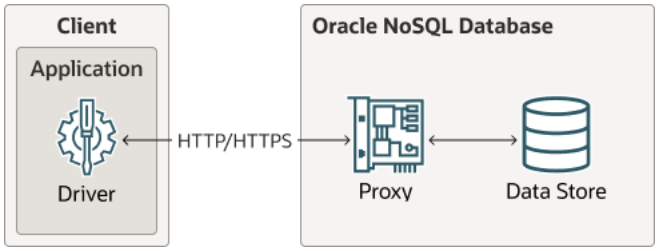
> **Note:**
>
> Using port 443 requires root privilege. You can use port **8443** if you do not have root privilege.

- `proxy.login` is the security login file generated in the earlier step for accessing the secure kvstore.
- `certificate.pem` is the certificate file generated in the previous step.
- `key-pkcs8.pem` is the private key file generated in the previous step.

- `privatekey_password` is the password for the encrypted `key-pkcs8.pem` file.

> **Note:**
>
> The proxy start-up only accepts private key file in PKCS#8 format. If your private key is already in PKCS#8 (start with -----BEGIN ENCRYPTED PRIVATE KEY----- or -----BEGIN PRIVATE KEY-----), you don't need any additional conversion. Otherwise, you can use OpenSSL to do the conversion.

Connecting an application to the secure data store



Oracle NoSQL Database drivers are available in various programming languages that are used in the client application. Currently, Java, Python, Go, Node.js, C#, and Rust are supported. Oracle NoSQL Database Proxy is a server that accepts requests from the client application and processes them using the Oracle NoSQL Database.

Authentication:

Within a secure Oracle NoSQL Database, access to the database is limited to authenticated users. You need to create a user for your application (`appln_user`) and pass the credentials of this user while connecting to your data store through the proxy from your drivers. The user will be authenticated using the credentials supplied(username/password) in the data store.

In the driver configuration, the application user name and password are provided. When the handle is created from the secure configuration the driver will send a login request to the proxy. The proxy uses this identity (`appln_user/applnuser_password`) to log into the data store, authenticating the user. This user identity (`appln_user`) must already exist in the store.

Create the user (`appln_user`) as shown below for your application to access the secure data store.

```
sql-> CREATE USER <appln_user> IDENTIFIED BY "<applnuser_password>"
```

Authorization:

Oracle NoSQL Database provides role-based authorization which enables the user to assign kvstore roles to user accounts to define accessible data and allow database administrative operations for each user account. Users can acquire desired privileges by role-granting. Your application should be given a role based on the *least privilege* access, carefully balancing the needs of the application with security concerns. See Configuring privileges and roles for more details.

After successful authentication of the application user, every driver request from the handle created sends the identity (`appln_user`) which is used by the proxy and data store to authorize any data operations such as table creation, put, get, query, etc. The data store is the entity that

does the authorization. The authorization is based on the identity and operation, and the data store uses the roles assigned to the `appln_user` to make authorization decisions.

Any request from the driver to the data store goes through the following steps:

- The driver sends the request to the proxy and the request contains the authenticated identity.
- The proxy uses the authenticated identity to send the request to the data store.
- The data store validates the identity and authorizes the identity for the desired operation (put, get, query, etc) using the privileges set in the data store for that identity.

- Java
- Python
- Go
- Node.js
- C#
- Rust

Java

The Oracle NoSQL Database Java Driver contains the jar files that enable an application to communicate with the Oracle NoSQL Database Proxy. You can connect to the proxy using the following steps.

1. Create an application user (`appln_user`) to access the data store through the secure proxy as shown in the above section.
2. For secure access, create an instance of the `StoreAccessTokenProvider` class with the parameterized constructor. Provide the reference of `StoreAccessTokenProvider` class to the `NoSQLHandleConfig` class to establish the appropriate connection. Install the Oracle NoSQL Database Java Driver in the application's classpath and use the following code to connect to the data store.

   ```
   String endpoint = "https://<proxy_host>:<proxy_https_port>";
   StoreAccessTokenProvider atProvider =
       new
   StoreAccessTokenProvider("<appln_user>","<applnuser_password>".toCharArray(
   ));
   NoSQLHandleConfig config = new NoSQLHandleConfig(endpoint);
   config.setAuthorizationProvider(atProvider);
   NoSQLHandle handle = NoSQLHandleFactory.createNoSQLHandle(config);
   ```

   where,

   - `proxy_host` is the hostname of the machine running the proxy service. This should match the proxy host you configured earlier.
   - `proxy_https_port` is the port on which the proxy is listening for requests. This should match the proxy https port configured earlier.

- `appln_user` is the user created to connect to the secure store. This should match the user created in the above section.
- `applnuser_password` is the password of the `appln_user`.

3. You can specify the details of the trust store containing the SSL certificate for the proxy in one of the following two ways.
   You can set it as part of your Java code as shown below:

```
/* the trust store containing SSL cert for the proxy */
System.setProperty("javax.net.ssl.trustStore", trustStore);
if (trustStorePassword != null) {

System.setProperty("javax.net.ssl.trustStorePassword",trustStorePassword);
}
```

   Alternatively, you can start-up the application program and set the `driver.trust` file's path to the Java trust store by using the following command. This is required as the proxy is already set up using the `certificate.pem` and `key-pkcs8.pem` files.

```
java -Djavax.net.ssl.trustStore=driver.trust \
-Djavax.net.ssl.trustStorePassword=<password of driver.trust> \
-cp .:lib/nosqldriver.jar application_program
```

   The `driver.trust` contains the `certificate.pem` or `rootCA.crt` certificate. If the certificate `certificate.pem` is in a chain signed by a trusted CA that is listed in `JAVA_HOME/jre/lib/security/cacerts`, then you don't need to append Java environment parameter `-Djavax.net.ssl.trustStore` in the Java command.

Python

The Oracle NoSQL Database Python driver contains the files that enable an application to communicate with the Oracle NoSQL Database Proxy. You can connect to a secure data store using the following steps.

1. Create an application user (`appln_user`) to access the data store through the secure proxy as shown in the above section.
2. If running a secure store, a certificate path should be specified through the `REQUESTS_CA_BUNDLE` environment variable:

```
$ export REQUESTS_CA_BUNDLE=
<fully_qualified_path_to_certificate>/certificate.pem:$REQUESTS_CA_BUNDLE
```

   or `borneo.NoSQLHandleConfig.set_ssl_ca_certs()`.

3. Use the following code to connect to the proxy.

```
from borneo import NoSQLHandle, NoSQLHandleConfig
from borneo.kv import StoreAccessTokenProvider
endpoint = 'https://<proxy_host>:<proxy_https_port>'
# Create the AuthorizationProvider for a secure store:
ap = StoreAccessTokenProvider('<appln_user>','<applnuser_password>')
# create a configuration object
config = NoSQLHandleConfig(endpoint).set_authorization_provider(ap)
# set the certificate path if running a secure store
```

```
config.set_ssl_ca_certs(<ca_certs>)
# create a handle from the configuration object
handle = NoSQLHandle(config)
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the proxy host you configured earlier.
- `proxy_https_port` is the port on which the proxy is listening for requests. This should match the proxy https port configured earlier.
- `appln_user` is the user created to connect to the secure store. This should match the user created in the above section.
- `applnuser_password` is the password of the `appln_user`.

Go

The Oracle NoSQL Database Go driver contains the files that enable an application to communicate with the Oracle NoSQL Database Proxy. You can connect to a secure data store using the following steps.

1. Create an application user (`appln_user`) to access the data store through the secure proxy as shown in the above section.
2. Use the following code to connect to the proxy. To connect an application to a secure NoSQL database, you need to provide user credentials used to authenticate with the server. If the Proxy server is configured with a self-signed certificate or a certificate that is not trusted by the default system CA, you also need to specifiy *CertPath* and *ServerName* for the certificate path and server name used to verify server's certificate.

```
import (
    "fmt"
    "github.com/oracle/nosql-go-sdk/nosqldb"
    "github.com/oracle/nosql-go-sdk/nosqldb/httputil"
)
...cfg:= nosqldb.Config{
    Endpoint: "https://<proxy_host>:<proxy_https_port>",
    Mode:     "onprem",
    Username: "<appln_user>",
    Password: "<applnuser_password>>",
   },
    // Specify the CertPath and ServerName
    // ServerName is used to verify the hostname for self-signed
certificates.
    // This field is set to the "CN" subject value from the certificate
specified by CertPath.
    HTTPConfig: httputil.HTTPConfig{
       CertPath: "<fully_qualified_path_to_cert>",
       ServerName: "<server_name>",
    },
}
client, err:=nosqldb.NewClient(cfg)
iferr!=nil {
    fmt.Printf("failed to create a NoSQL client: %v\n", err)
    return
}
```

```
deferclient.Close()
// Perform database operations using client APIs.
// ...
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the proxy host you configured earlier.
- `proxy_https_port` is the port on which the proxy is listening for requests. This should match the proxy https port configured earlier.
- `appln_user` is the user created to connect to the secure store. This should match the user created in the above section.
- `applnuser_password` is the password of the `appln_user`.

Node.js

The Oracle NoSQL Database Node.js driver contains the files that enable an application to communicate with the Oracle NoSQL Database Proxy. You can connect to a secure data store using the following steps.

1. Create an application user (`appln_user`) to access the data store through the secure proxy as shown in the above section.
2. In secure mode the proxy requires the SSL Certificate and Private key. The proxy certificate was created when configuring the proxy as explained here. If the root certificate authority (CA) for your proxy certificate is not one of the trusted root CAs , the driver needs the certificate chain file (e.g. *certificates.pem*) or a root CA certificate file (e.g. *rootCA.crt*) in order to connect to the proxy. If you are using self-signed certificate instead, the driver will need the certificate file (e.g. *certificate.pem*) for the self-signed certificate in order to connect.
   To provide the certificate or certificate chain to the driver, you have two options , either specifying in the code or setting as environment variables.

   You can specify the certificates through `httpOpt` property while creating the NoSQL handle. Inside `httpOpt` you can use `ca` property to specify the CA as shown below.

```
const client = new NoSQLClient({ .....,
  httpOpt: {
    ca: fs.readFileSync(<caCertFile>)
  },.....
});
```

> **Note:**
>
> If a file path is supplied, the path can be absolute or relative to the current working directory of the application.

Alternatively, before running your application, set the environment variable `NODE_EXTRA_CA_CERTS` as shown below.

```
export NODE_EXTRA_CA_CERTS="<fully_qualified_path_to_driver.trust>"
```

where *driver.trust* is either a certificate chain file (*certificates.pem*) for your CA, your root CA's certificate (*rootCA.crt*) or a self-signed certificate (*certificate.pem*).

3. To connect to the proxy in secure mode, in addition to communication endpoint, you need to specify user name and password of the driver user. This information is passed in `Config#auth` object under `kvstore` property and can be specified in one of 3 ways as described below.
You may choose to specify user name and password directly:

```
const NoSQLClient = require('oracle-nosqldb').NoSQLClient;
const client = new NoSQLClient({
    endpoint: 'https://<proxy_host>:<proxy_https_port>',
    auth: {
        kvstore: {
            user: '<appln_user>',
            password: '<applnuser_password>'
        }
    }
});
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the proxy host you configured earlier.
- `proxy_https_port` is the port on which the proxy is listening for requests. This should match the proxy https port configured earlier.
- `appln_user` is the user created to connect to the secure store. This should match the user created in the above section.
- `applnuser_password` is the password of the `appln_user`.

This option is less secure because the password is stored in plain text in memory.

You may choose to store credentials in a separate file which is protected by file system permissions, thus making it more secure than previous option, because the credentials will not be stored in memory, but will be accessed from this file only when login is needed. Credentials file should have the following format:

```
{
    "user":     "<appln_user>",
    "password": "<applnuser_password>"
}
```

Then you may reference this credentials file as following:

```
const NoSQLClient = require('oracle-nosqldb').NoSQLClient;
const client = new NoSQLClient({
    endpoint: 'https://<proxy_host>:<proxy_https_port>',
    auth: {
        kvstore: {
            credentials: '<path/to/credentials.json>'
        }
    }
});
```

You may also reference `credentials.json` in the configuration file that is used to create `NoSQLClient` instance.

Contents of `config.json`

```
{
    "endpoint": "https://<proxy_host>:<proxy_https_port>",
    "auth": {
        "kvstore": {
            "credentials": "<path/to/credentials.json>"
        }
    }
}
```

```
const NoSQLClient = require('oracle-nosqldb').NoSQLClient;
const client = new NoSQLClient('</path/to/config.json>');
```

> **Note:**
>
> If a file path is supplied, the path can be absolute or relative to the current working directory of the application.

C#

The Oracle NoSQL Database Dotnet driver contains the files that enable an application to communicate with the Oracle NoSQL Database Proxy. You can connect to a secure data store using the following steps.

1. Create an application user (`appln_user`) to access the data store through the secure proxy as shown in the above section.
2. To connect to the proxy in secure mode, in addition to communication endpoint, you need to specify the details of the user connecting to the secure data store. This information is passed in the instance of `KVStoreAuthorizationProvider` and can be specified in any of the ways as described below.
   You may choose to specify user name and password directly:

```
var client = new NoSQLClient(
    new NoSQLConfig
    {
        Endpoint = "https://<proxy_host>:<proxy_https_port>",
        AuthorizationProvider = new KVStoreAuthorizationProvider(
            <appln_user, // user name as string
            <applnuser_password>) // password as char[]
    });
```

where,

- `proxy_host` is the hostname of the machine running the proxy service. This should match the proxy host you configured earlier.
- `proxy_https_port` is the port on which the proxy is listening for requests. This should match the proxy https port configured earlier.

- `appln_user` is the user created to connect to the secure store. This should match the user created in the above section.
- `applnuser_password` is the password of the `appln_user`.

This option is less secure because the password is stored in plain text in memory for the lifetime of `NoSQLClient` instance. Note that the password is specified as char[] which allows you to erase it after you are finished using NoSQLClient.

You may choose to store credentials in a separate file which is protected by file system permissions, thus making it more secure than the previous option, because the credentials will not be stored in memory, but will be accessed from this file only when the login to the store is required. Credentials file should have the following format:

```
{
    "UserName": "<appln_user>",
    "Password": "<applnuser_password>"
}
```

Then you may use this credentials file as following:

Contents of `config.json`

```
var client = new NoSQLClient(
    new NoSQLConfig
    {
        Endpoint: 'https://<proxy_host>:<proxy_https_port>',
        AuthorizationProvider = new KVStoreAuthorizationProvider(
            "<path/to/credentials.json>")
    });
```

You may also reference credentials.json in the JSON configuration file that is used to create `NoSQLClient` instance:

Contents of `config.json`

```
{
    "Endpoint": "https://<proxy_host>:<proxy_https_port>",
    "AuthorizationProvider": {
        "AuthorizationType": "KVStore",
        "CredentialsFile": "<path/to/credentials.json>"
    }
}


var client = new NoSQLClient("</path/to/config.json>");
```

> **Note:**
>
> If a file path is supplied, the path can be absolute or relative to the current working directory of the application.

Note that in `config.json` the authorization provider is represented as a JSON object with the properties for `KVStoreAuthorizationProvider` and an additional AuthorizationType

property indicating the type of the authorization provider, which is KVStore for the secure on-premises store.

You need to provide the trusted root certificate to the driver if the certificate chain for your proxy certificate is not rooted in one of the well known CAs. The provided certificate may be either your custom CA or self-signed proxy certificate. It must be specified using TrustedRootCertificateFile property, which sets a file path (absolute or relative to the current working directory) to a PEM file containing one or more trusted root certificates (multiple roots are allowed in this file). This property is specified as part of ConnectionOptions in NoSQLConfig.

```
var client = new NoSQLClient(
new NoSQLConfig {
   Endpoint: 'https://<proxy_host>:<proxy_https_port>',
   AuthorizationProvider = new KVStoreAuthorizationProvider( "<path/to/
credentials.json>"),
   ConnectionOptions: { "TrustedRootCertificateFile": "<path/to/
certificates.pem>" }
});
```

Rust

The Oracle NoSQL Database Rust driver contains the files that enable an application to communicate with the Oracle NoSQL Database Proxy. You can connect to a secure data store using the following steps.

1. Create an application user (`appln_user`) to access the data store through the secure proxy as discussed above.
2. To connect to the proxy in secure mode, in addition to communication endpoint, you need to specify the user name and password of the driver user. This information is passed as shown below.

```
{
    let handle = Handle::builder()
    .endpoint("https://<proxy_host>:<proxy_https_port>")?
    .mode(HandleMode::Onprem)?
    .onprem_auth("<appln_user>", "<applnuser_password>")?
    .build().await?;
}
```

where

- `proxy_host` is the host name of the machine running the proxy service. This should match the proxy host you configured earlier.
- `proxy_https_port` is the port on which the proxy is listening for requests. This should match the proxy https port configured earlier.
- `appln_user` is the user created to connect to the secure store. This should match the user created in the above section.
- `applnuser_password` is the password of the `appln_user`.

Instead of providing the credentials directly, you can also store the credentials in a separate file which is protected by file system permissions. Credentials file should have the following format:

```
{
    "UserName": "<appln_user>",
    "Password": "<applnuser_password>"
}
```

Then you may use this credentials file as following:

```
{
    let handle = Handle::builder()
    .endpoint("https://<proxy_host>:<proxy_https_port>")?
    .mode(HandleMode::Onprem)?
    .onprem_auth_from_file("/path/to/user_pass_file")?
    .build().await?;
}
```

> **Note:**
>
> If a file path is supplied, the path can be absolute or relative to the current working directory of the application.

You need to provide trusted root certificate to the driver if the certificate chain for your proxy certificate is not rooted in one of the well known certificate authorities (CA). The provided certificate may be either your custom certificate authority or self-signed proxy certificate.

```
{
        let handle = Handle::builder()
        .endpoint("https://<proxy_host>:<proxy_https_port>")?
        .mode(HandleMode::Onprem)?
        .onprem_auth_from_file("/path/to/user_pass_file")?
        .add_cert_from_pemfile("/path/to/certificate.pem")?
        .build().await?;
}
```

> **Note:**
>
> If a file path is supplied, the path can be absolute or relative to the current working directory of the application.

If you want to instruct the client to skip verifying the server's certificate, you can specify as follows:

```
{
        let handle = Handle::builder()
```

```
        .endpoint("https://<proxy_host>:<proxy_https_port>")?
        .mode(HandleMode::Onprem)?
        .onprem_auth_from_file("/path/to/user_pass_file")?
        .danger_accept_invalid_certs(true)?
        .build().await?;
}
```

Example: Configuring Multiple Oracle NoSQL Database Proxies for Redundancy

Learn how to configure multiple Oracle NoSQL Database proxies to work with a load balancer.

You can set up a configuration using multiple Oracle NoSQL Database proxies to create redundancy. Redundancy ensures that at least one proxy continues to function through different demand loads and failure types.

The Oracle NoSQL Database Proxy can run in one or multiple dedicated hosts. It can also be hosted inside the nodes of the data store. You can use a load balancer as the front end, which has a back end set of multiple Oracle NoSQL Database proxies for redundancy on different hosts.

HAProxy is an open-source software that offers a load balancer for HTTP and TCP applications. You can use the *HAProxy* software as a load balancer in front of multiple Oracle NoSQL Database proxies.

> **Note:**
>
> There are other load balancers available. This topic demonstrates the concepts using *HAProxy* as the load balancer.

For example, consider a data store with three dedicated proxy hosts: `proxy1-nosql, proxy2-nosql, proxy3-nosql`. To set up multiple proxies with redundancy, you can configure the hosts `proxy1-nosql, proxy2-nosql, proxy3-nosql` as Oracle NoSQL Database proxies in the back end. Install and configure the *HAProxy* software as the load balancer. The load balancer routes requests to the proxies.

Configuring Oracle NoSQL Database Proxy in the hosts

Ensure that you have deployed the data store.

- Start the HTTP proxy on each of the hosts, `proxy1-nosql, proxy2-nosql, proxy3-nosql` as follows:

  Non-secure data store:

  ```
  java -jar $KVHOME/lib/httpproxy.jar -helperHosts
  <kvstore_helper_host:5000> -storeName <kvstore_name> -httpPort 8080 -
  verbose true
  ```

  For details, see Using the Proxy in a non-secure data store.

Secure data store:

```
java -jar $KVHOME/lib/httpproxy.jar -helperHosts
<kvstore_helper_host:5000> -storeName <kvstore_name> -httpsPort 8443 -
storeSecurityFile $KVROOT/security/proxy.login -sslCertificate
certificate.pem -sslPrivateKey key-pkcs8.pem -sslPrivateKeyPass
<privatekey_password> -verbose true
```

For details, see Using the Proxy in a secure data store.

> **Note:**
>
> Instead of creating a certificate for each Oracle NoSQL Database Proxy, you can create only one certificate with Subject Alternative Names (SAN). This simplifies the configuration in the following scenarios:
>
> – When you need to rotate the certificate. You only have one certificate to manage and share.
>
> – When a server has multiple names.
>
> – When using the IPs.
>
> For more details on using SAN, see Generating Certificate and Private Key for Proxy.

- Verify if the proxy is functioning.

  Non-secure data store:

  ```
  http://<proxy1-nosql>:8080/V2/health
  ```

  Secure data store:

  ```
  https://<proxy1-nosql>:8443/V2/health
  ```

Configuring the load balancer

- Install and configure the *HAProxy* software as a load balancer.

> **Note:**
>
> Oracle NoSQL Database documentation does not provide instructions to set up *HAProxy* as a load balancer. You must implement it as a prerequisite before configuring the Oracle NoSQL Database Proxy set up with redundancy.

- Configure the *HAProxy* software:

  The examples serve as a guideline to configure an open-source load balancer in the Oracle NoSQL Database HTTP proxy context.

  Add the following lines at the end of the file `/etc/haproxy/haproxy.cfg`.

  This configures the *HAProxy* to route requests to the proxies: `proxy1-nosql, proxy2-nosql, proxy3-nosql`.

Example: Non-secure data store

```
# Configure HAProxy to listen on port 8080
frontend http_front
   bind *:8080
   stats uri /haproxy?stats
   default_backend http_back

# Configure HAProxy to route requests to Oracle NoSQL Database Proxy hosts
on port 8080
backend http_back
   balance roundrobin
   server proxy1-nosql <IP_node1>:8080 check
   server proxy2-nosql <IP_node2>:8080 check
   server proxy3-nosql <IP_node3>:8080 check
```

Example: Secure data store

Depending on your load balancer, you can use one of the following sample configurations:

- **SSL passthrough configuration:**

  The load balancer passes encrypted HTTPS traffic directly to the back end servers without decrypting the traffic on the load balancer. Here, the load balancer and proxies use the same SSL certificate.

  ```
  # Configure HAProxy to listen on port 8443
  frontend http_front
     bind *:8443 ssl crt /etc/haproxy/certs/full.pem
     timeout http-keep-alive 20s
     stats uri /haproxy?stats
     default_backend http_back

  # Configure HAProxy to route requests to Oracle NoSQL Database Proxy
  hosts on port 8443
  backend http_back
     balance roundrobin
     timeout http-keep-alive 20s
     server proxy1-nosql <IP_node1>:8443 check maxconn 20 ssl verify none
     server proxy2-nosql <IP_node2>:8443 check maxconn 20 ssl verify none
     server proxy3-nosql <IP_node3>:8443 check maxconn 20 ssl verify none
  ```

- **SSL Bridging configuration:**

  The load balancer decrypts all HTTPS traffic when it arrives at the load balancer, and encrypts the traffic to the destination server. This configuration allows load balancer and proxies to use different SSL certificates.

  ```
  # Configure HAProxy to listen on port 8443
  frontend http_front
    bind *:8443 ssl crt /etc/haproxy/certs/full.pem
    timeout http-keep-alive 20s
    stats uri /haproxy?stats
    default_backend http_back


  # Configure HAProxy to route requests to Oracle NoSQL Database Proxy
  ```

```
hosts on port 8443
backend http_back
  balance roundrobin
  timeout http-keep-alive 20s
  server proxy1-nosql <IP_node1>:8443 check maxconn 20 ssl ca-file /
root/proxy1-nosql.pem
  server proxy2-nosql <IP_node2>:8443 check maxconn 20 ssl ca-file /
root/proxy2-nosql.pem
  server proxy3-nosql <IP_node3>:8443 check maxconn 20 ssl ca-file /
root/proxy3-nosql.pem
```

- Restart the *HAProxy* and validate the status:

```
sudo systemctl stop haproxy.service
sudo systemctl start haproxy.service
sudo systemctl status haproxy.service
```

- Verify if the load balancer is working.

  Here, *<LB-hostname>* is the host on which the *HAProxy* software is installed.

  Non-secure data store:

```
http://<LB-hostname>:8080/V2/health
```

  Secure data store:

```
https://<LB-hostname>:8443/V2/health
```

Configuring Multi-Region Data Stores

Oracle NoSQL Database supports Multi-Region Architecture in which you can create tables in multiple data stores, and still maintain consistent data across these clusters. Each data store in a Multi-Region Oracle NoSQL Database setup is called a *Region*. A Multi-Region Table or *MR Table* is a global logical table that is stored and maintained in different regions. MR Tables maintain consistent data in all the regions. That is, any updates made to an MR Table in one region automatically applies to the corresponding MR Table in all the other participating regions. To learn more about Oracle NoSQL Database Multi-Region Architecture and MR Tables, see Multi-Region Architecture in the *Concepts Guide*.

Replication in a Multi-Region Table:

All writes to the table, including insert, update, and delete would be replicated. All DDL operations (Create Table, Alter Table and Drop Table, Create Index, Alter Index and Drop Index) and operations that change the table metadata like TTL, will not be replicated. For example, the following actions will not be replicated.

- Index creation in one region
- Altering the definition of an existing index from one region
- Dropping the index from one region
- Changing the schema definition in one region
- Changing the Table's default Table Time to Live (TTL) in one region

You can configure a Multi-Region Oracle NoSQL Database, and create and manipulate the MR Tables using the Oracle NoSQL Database command-line interface (CLI).The remainder of this chapter is organized into four use cases to demonstrate the different features of the Multi-Region Oracle NoSQL Database and MR Tables. The examples provided show you which commands to use and how. For a complete list of all the commands available in the CLI, see Admin CLI Reference.

Child MR Tables:

You can create child tables in the Multi-Region architecture. That means an existing Multi-Region table can have child tables. You can add a child table to a top-level table that is already a Multi-Region table, and the child table will be automatically Multi-Region enabled . That is an entire table hierarchy is Multi-Region or none of it is. You can drop a child from a top-level table, and the child table will be removed from the hierarchy. The child table will also be removed from the MR Table graph such that it no longer participates in cross region replication.

Use Cases

- Use Case 1: Set up Multi-Region Environment
- Use Case 2: Expand a Multi-Region Table
- Use Case 3: Contract a Multi-Region Table
- Use Case 4: Drop a Region

Use Case 1: Set up Multi-Region Environment

An organization deploys two on-premise data stores, one each at Frankfurt and London. As per their requirement, they create a few MR Tables in both the regions. The `Users` table is one of the many MR Tables created and maintained by this organization. In the next few topics, let us discuss how to set up the Frankfurt and London regions and how to create and work with an MR Table called `Users` in these two regions.

To configure a Multi-Region NoSQL Database, you need to execute the below listed tasks in each region. For the use case under discussion, you must execute all the below listed steps in both the participating regions, Frankfurt and London.

1. Deploy the data store
2. Set Local Region Name
3. Configure XRegion Service
4. Start XRegion Service
5. Create Remote Regions
6. Create Multi-Region Tables
7. Access and Manipulate Multi-Region Tables

Deploy the data store

In each region in the Multi-Region NoSQL Database setup, you must deploy its own data store independently.

Steps:

To deploy the data store:

1. Follow the instructions given in Configuring your data store installation.

2. After deploying the data store of your desired topology, you can check the health of the data store by executing the `ping` command from the command line interface.

   ```
   [~]$ java -jar $KVHOME/lib/kvstore.jar ping -port <port number> -host
   <host name>
   ```

3. You can also verify the topology of the data store by executing the `show topology` command from the `kv` prompt. See show topology.

   ```
   kv-> show topology
   ```

Example:

For the use case under discussion, you must set up data stores for the two regions proposed.

```
# Connect to the data stores deployed at host1, host2, and host3 from the kv
prompt
[~]$java -jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts host1:5000,host2:5000,host3:5000

# View the topology of the data store
kv-> show topology
store=mrtstore  numPartitions=1000 sequence=1008
  zn: id=zn1 name=zn1 repFactor=3 type=PRIMARY allowArbiters=false
masterAffinity=false

  sn=[sn1] zn:[id=zn1 name=zn1] host1:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
             single-op avg latency=0.8630216 ms   multi-op avg
latency=1.7694647 ms
  sn=[sn2] zn:[id=zn1 name=zn1] host2:5000 capacity=1 RUNNING
    [rg1-rn2] RUNNING
             single-op avg latency=0.0 ms   multi-op avg latency=2.0211697 ms
  sn=[sn3] zn:[id=zn1 name=zn1] host3:5000 capacity=1 RUNNING
    [rg1-rn3] RUNNING
             single-op avg latency=0.0 ms   multi-op avg latency=1.8524266 ms

  numShards=1
  shard=[rg1] num partitions=1000
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn2
    [rg1-rn3] sn=sn3


# Connect to the data store deployed at host4, host5, and host6 from the kv
prompt
[~]$java -jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts host4:5000,host5:5000,host6:5000

# View the topology of the data store
kv-> show topology
store=mrtstore  numPartitions=1000 sequence=1008
  zn: id=zn1 name=zn1 repFactor=3 type=PRIMARY allowArbiters=false
masterAffinity=false
```

```
  sn=[sn1] zn:[id=zn1 name=zn1] host4:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
              single-op avg latency=0.7519707 ms   multi-op avg
latency=2.000658 ms
  sn=[sn2] zn:[id=zn1 name=zn1] host5:5000 capacity=1 RUNNING
    [rg1-rn2] RUNNING
              single-op avg latency=0.0 ms   multi-op avg latency=3.2067895 ms
  sn=[sn3] zn:[id=zn1 name=zn1] host6:5000 capacity=1 RUNNING
    [rg1-rn3] RUNNING
              single-op avg latency=0.0 ms   multi-op avg latency=1.9516457 ms

  numShards=1
  shard=[rg1] num partitions=1000
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn2
    [rg1-rn3] sn=sn3
```

Set Local Region Name

Learn how to set a name to the local region in a Multi-Region NoSQL Database.

After deploying the data store and before creating the first MR Table in each participating region, you must set a *local region name*. You can change the local region name as long as no MR Tables are created in that region. After creating the first MR Table, the local region name becomes immutable.

Steps:

To set the local region name:

1. Connect to the `sql` prompt from the local region, and connect to the local data store.
2. Execute the following command from the `sql` prompt.

   ```
   sql-> SET LOCAL REGION <local region name>;
   ```

3. Optionally, you can execute the following command to verify that the local region name is set successfully.

   ```
   sql-> SHOW REGIONS;
   ```

Example:

Set the local region name for the two proposed regions, Frankfurt and London.

```
# Connect to the data store deployed at host1, host2, and host3 from the SQL
shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- Set the local region name to 'fra'
sql-> SET LOCAL REGION fra;
Statement completed successfully

-- List the regions
```

```
sql-> SHOW REGIONS;
regions
    fra (local, active)


# Connect to the data store deployed at host4, host5, and host6 from the SQL
shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

-- Set the local region name to 'lnd'
sql-> SET LOCAL REGION lnd;
Statement completed successfully

-- List the regions
sql-> SHOW REGIONS;
regions
    lnd (local, active)
```

Configure XRegion Service

Learn how to configure the XRegion Service in a Multi-Region Oracle NoSQL Database

Before creating any MR Table, you must deploy an XRegion Service. In simple terms, this is also called an agent. The XRegion Service runs independently with the local data store and it is recommended to deploy it close to the local data store. To know more about agent and agent groups, see Cross-Region Service in the *Concepts Guide*.

You can achieve horizontal scalability by dividing the shards across multiple XRegion Service agents based on the CPU and memory of the host node of the agent and the latency and throughput requirements of the application. The mapping of data store shards to XRegion Service agents is determined in a round-robin manner in order to balance the load of agents.

> **Note:**
>
> Each XRegion Service agent will map to at least one shard. If users configure more agents than the number of shards, the XRegion Service agent would not be able to start.

Each XRegion Service group consists of a group of independent XRegion Service agents, and each agent in the group is running on a host node and is responsible to handle one or more shards of the data store. The agents in XRegion Service Group are completely independent of each other, that is, each agent does not talk directly to any other agent in the group. Any agent can be shut down and restarted without impacting other agents. It is recommended that you add XRegion Service agents on individual hosts that do not contain any Storage Node configured.

Steps:

To configure the XRegion Service, execute the following tasks in each region:

1. Create a home directory for the XRegion Service.

2. Create a JSON config file in the home directory created in the step 1. The structure of the `json.config` file is shown below.

```
{
  "path" : "<entire path to the home directory for the XRegion Service>",
  "agentGroupSize" : <number of service agents>,
  "agentId" : <agent id using 0-based numbering>,
  "region" : "<local region name>",
  "store" : "<local store name>",
  "helpers" : [
    "<host1>:<port>",
    "<host2>:<port>",
    …
    "<hostn>:<port>"
  ],
  "security" : "<entire path to the security file of the local store>",
  "regions" : [
    {
        "name" : "<remote region name>",
        "store" : "<remote store name>",
        "security" : "<entire path to the security file of the remote
store>",
        "helpers" : [
            "<host1>:<port>",
            "<host2>:<port>",
            …
            "<hostn>:<port>
         ]
    },
    {
        "name" : "<remote region name>",
        "store" : "<remote store name>",
        "security" : "<entire path to the security file of the remote
store>",
        "helpers" : [
            "<host1>:<port>",
            "<host2>:<port>",
            …
            "<hostn>:<port>
        ]
    },
    ...
  ]
  "durability" : "<durability setting>"
}
```

Where each attribute in the `json.config` file is explained below:

`path`	This is the root directory of the XRegion Service. The agents use this directory to dump logs, statistics and other auxiliary files. The directory shall be readable and writable to the agents.

`agentGroupSize` and `agentId`	Specifies the number of service agents and the Agent ID in the agent group. The Agent ID is specified as numbers starting from 0. These details are used to form a group of agents that serve the local region. Forming a group of agents achieves horizontal scalability.
`security`	Specifies the security file used by the agent. This attribute must be defined for the local store as well as the remote stores.
`region`	Specifies the local region name defined for the region where you are configuring the agent.
`store`	Specifies the name of the store in the local region.
`helpers`	Specifies the list of host and port numbers used for configuring the local store. These helper hosts are those you used to create a KV client. For XRegion Service to connect to the local and remote regions, each region's firewall must be configured to open the registry port and HA ports.
`regions`	After defining the local region, you must define a list of remote regions. At least one remote region shall be defined in order to create an MR Table. Specifies the region name, store name, and helper hosts for each remote region you want to include. **Note:** The remote region names listed here must be same as the local region names defined for them.
`durability`	This is an optional parameter. It specifies the durability setting for Master commit synchronization. The possible values are: • `COMMIT_NO_SYNC` • `COMMIT_SYNC` • `COMMIT_WRITE_NO_SYNC` The default durability setting is `COMMIT_NO_SYNC`.

3. Grant the following privileges to the XRegion Service Agent:

- CREATE_ANY_TABLE
- Write permission to system table
- Read and Write permission to all the user tables
- DELETE_ANY_TABLE

> **Note:**
>
> Inserting data into a Multi-Region table (using a put operation) requires both INSERT and DELETE privileges.

```
— create role for the agent --
CREATE ROLE <Agent Role>

— grant privileges to the role --
GRANT CREATE_ANY_TABLE to <Agent Role>
GRANT WRITE_SYSTEM_TABLE to <Agent Role>
GRANT READ_ANY_TABLE to <Agent Role>
GRANT INSERT_ANY_TABLE to <Agent Role>
GRANT DELETE_ANY_TABLE to <Agent Role>

— grant role to the agent user --
GRANT <Agent Role> to user <Agent User>
```

> **Note:**
>
> This step is required only for secure data stores. In a non-secure data store setup, this step can be skipped.

Adding additional XRegion Service Agents

You can achieve horizontal scalability by adding more XRegion Service agents in a group.

To add horizontal scalability to your agents, do the following:

- Identify a host node for the agent. It is recommended that the agent will be the only process running on that node.
- Download the Oracle NoSQL Database bundle in the host node identified above and extract the contents of the Oracle NoSQL Database package (`kv-M.N.O.zip` or `kv-M.N.O.tar.gz`) to `$KVHOME`. Unzipping the package installs the Oracle NoSQL Database.

  ```
  unzip kv-ee-24.1.11.zip
  ```

- Create a home directory for the XRegion Service.
- Create a JSON config file in the home directory created in the above step.

> **Note:**
>
> Steps to create a config file is given above. The `agentId` starts with **0** and is incremented by one. For example if `"agentGroupSize" : 2`, and there is already one agent and you are adding the second one, then the value of `agentId` is **1**.

Example:

Create a `json.config` file for each proposed region, Frankfurt and London.

```
# Contents of the configuration file in the host1 in 'fra' Region
{
  "path": "<path to the json config file>",
  "agentGroupSize": 2,
  "agentId": 0,
  "region": "fra",
  "store": "<storename at the fra region>",
  "security": "<path to the security file>",
  "helpers": [
    "host1:5000",
    "host2:5000",
    "host3:5000"
  ],
  "regions": [
    {
      "name": "lnd",
      "store": "<storename at the lnd region>",
      "security": "<path to the security file>",
      "helpers": [
        "host4:5000",
        "host5:5000",
        "host6:5000"
      ]
    }
  ]
}


# Contents of the configuration file in the host7 in 'fra' Region.
# This host is used only to run an additional XRegion agent
{
  "path": "<path to the json config file>",
  "agentGroupSize": 2,
  "agentId": 1,
  "region": "fra",
  "store": "<storename at the fra region>",
  "security": "<path to the security file>",
  "helpers": [
    "host1:5000",
    "host2:5000",
    "host3:5000"
  ],
  "regions": [
    {
```

```
      "name": "lnd",
      "store": "<storename at the lnd region>",
      "security": "<path to the security file>",
      "helpers": [
        "host4:5000",
        "host5:5000",
        "host6:5000"
      ]
    }
  ]
}
```

```
# Contents of the configuration file in the 'lnd' Region
{
  "path": "<path to the json config file>",
  "agentGroupSize": 2,
  "agentId": 0,
  "region": "lnd",
  "store": "<storename at the lnd region>",
  "security": "<path to the security file>",
  "helpers": [
    "host4:5000",
    "host5:5000",
    "host6:5000"
  ],
  "regions": [
    {
      "name": "fra",
      "store": "<storename at the fra region>",
      "security": "<path to the security file>",
      "helpers": [
        "host1:5000",
        "host2:5000",
        "host3:5000"
      ]
    }
  ]
}
```

```
# Contents of the configuration file in host 8 in the 'lnd' Region,
# This host is used only to run an additional XRegion agent
{
  "path": "<path to the json config file>",
  "agentGroupSize": 2,
  "agentId": 1,
  "region": "lnd",
  "store": "<storename at the lnd region>",
  "security": "<path to the security file>",
  "helpers": [
    "host4:5000",
    "host5:5000",
    "host6:5000"
  ],
  "regions": [
```

```
    {
      "name": "fra",
      "store": "<storename at the fra region>",
      "security": "<path to the security file>",
      "helpers": [
        "host1:5000",
        "host2:5000",
        "host3:5000"
      ]
    }
  ]
}
```

Start XRegion Service

The Xregion service in each region can be started using the `xrstart` command. The `xrstart` command has to be executed for each data store separately. The status of the `xrstart` command execution can be viewed by reading the contents of `status.<number of agents>.<agentId>.txt` file which contains the process ID of the successfully started agent. To get more details about `xrstart` command and its various parameters, see xrstart .

The command `xrstart` can be used in two ways:

- The command for starting the Xregion service in the foreground is:

  ```
  java -Xms1G -Xmx1G -jar <path to jar file> xrstart -config <path to config file>
  ```

- The command for starting the Xregion service in the background is:

  ```
  java -Xms1G -Xmx1G -jar <path to jar file> xrstart -config <path to config file> -bg
  ```

Example:

Start the XRegion Service in both the regions, Frankfurt and London.

There are two XRegion Service agents in both the regions, Frankfurt and London. In Frankfurt the XRegion agents are in host1 and host7 and in London, the XRegion agents are in host4 and host8. Both the XRegion agents have to be started in each region.

```
# Start the first XRegion Service in host1 in the 'fra' Region
[oracle@host1 xrshome]$ java -Xms256m -Xmx2048m -jar $KVHOME/lib/kvstore.jar
xrstart \
-config <path to the json config file> -bg

[1] 24356

# View the status of the xrstart command in host1 in the 'fra' Region
[oracle@host1 xrshome]$ cat status.<number of agents>.<agentId>.txt
Cross-region agent (region=fra,store=mrtstore, helpers=[host1:5000,
host2:5000, host3:5000])
starts up from config file=/home/oracle/xrshome/ json.config at 2024-04-05
08:57:34 UTC
```

Similarly start the XRegion agent in host 7.

```
# Start the second XRegion Service in host 7 in the 'fra' Region

[oracle@host7 xrshome]$ java -Xms256m -Xmx2048m -jar $KVHOME/lib/kvstore.jar
xrstart \
-config <path to the json config file> -bg

[1] 24489

 # View the status of the xrstart command in host7 in the 'fra' Region
[oracle@host7 xrshome]$ cat status.<number of agents>.<agentId>.txt
Cross-region agent (region=fra,store=mrtstore, helpers=[host1:5000,
host2:5000, host3:5000])
starts up from config file=/home/oracle/xrshome/ json.config at 2024-04-05
08:59:34 UTC


# Start the first XRegion Service in host4 in the 'lnd' Region
[oracle@host4 xrshome]$ java -Xms256m -Xmx2048m -jar $KVHOME/lib/kvstore.jar
xrstart \
-config <path to the json config file> -bg

[1] 17587

# View the status of the xrstart command in host4 in the 'lnd' Region
[oracle@host4 xrshome]$ cat status.<number of agents>.<agentId>.txt
Cross-region agent (region=lnd,store=mrtstore, helpers=[host4:5000,
host5:5000, host6:5000])
starts up from config file=/home/oracle/xrshome/ json.config at 2024-04-05
08:57:34 UTC
```

Similarly start the XRegion agent in host8.

```
# Start the second XRegion Service in host8 in the 'lnd' Region

[oracle@host8 xrshome]$ java -Xms256m -Xmx2048m -jar $KVHOME/lib/kvstore.jar
xrstart \
-config <path to the json config file> -bg

[1] 17587

# View the status of the xrstart command in the 'lnd' Region
[oracle@host8 xrshome]$ cat status.<number of agents>.<agentId>.txt
Cross-region agent (region=lnd,store=mrtstore, helpers=[host4:5000,
host5:5000, host6:5000])
starts up from config file=/home/oracle/xrshome/ json.config at 2024-04-05
09:09:34 UTC
```

Status of XRegion Agent

To check the current status of the XRegion Agent, use the command `xrstatus`. This command provides information on the status as well as the exit codes indicating whether the agent has started, is already running, or has stopped, etc, see xrstatus

The below command will check the status of the agent.

```
java -Xms256m -Xmx2048m -jar $KVHOME/lib/kvstore.jar xrstatus -config <path
to the json config file>
```

The status can be checked:

- After starting the agent using the `xrstart` command.
- When the agent is already running, to be sure if there was no failure in the connection.
- After the agent has stopped using the `xrstop` command.

To view the status of the agent in the form of exit codes, use the command:

```
echo $?
```

For example, after starting the agent, check the status of the agent as follows:

```
# Start the agent
bash-4.4$ java -Xms256m -Xmx2048m -jar ./lib/kvstore.jar xrstart -config
<path to the jsonconfig file>

[1] 102148

# View the status of the agent
bash-4.4$ java -Xms256m -Xmx2048m -jar ./lib/kvstore.jar xrstatus -config
<path to the json config file>
Agent running

# View the exit code
bash-4.4$ echo $?
0
```

Running the `xrstatus` command shows the agent's status and using the `echo $?` command reveals the exit code corresponding to the agent's status. Look at the table to understand more.

Here are the exit codes and their descriptions for the agent's status after starting, while running, and after stopping:

1. Agent status after Starting the agent:

Exit Code	Description
0	The agent has started successfully.
1	The agent has failed to start.
2	The agent failed to start with timeout, meaning it started successfully but did not generate a status file with the process ID.
3	The agent crashed after starting.
4	The agent is already running and another agent cannot be started.

2. Agent Status of a running agent:

Exit Code	Description

0	The agent is currently running.
1	The agent is currently not running.
2	The agent has crashed after running for a few seconds.

3. Agent status after stopping the agent:

Exit Code	Description
0	The agent has stopped successfully.
1	The agent has failed to stop.

Create Remote Regions

Learn to create remote regions from each region in a Multi-Region NoSQL Database.

Before creating and operating on an MR table, you must define the remote regions. You have already set the local region name for each region, in an earlier step. In this step, you define all the remote regions for each region. A remote region is different from the local region where the command is executed.

Steps:

To create the remote regions:

1. Connect to the `sql` prompt from the local region, and connect to the local data store.
2. Execute the following command from the `sql` prompt.

   ```
   sql-> CREATE REGION <remote region name>;
   ```

3. Optionally, you can execute the following command to list the remote regions that are created successfully.

   ```
   sql-> SHOW REGIONS;
   ```

Example:

Create the remote regions in both the regions, Frankfurt and London.

```
# Connect to the data store deployed in the 'fra' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- Create a remote region 'lnd'
sql-> CREATE REGION lnd;
Statement completed successfully

- List the regions
sql-> SHOW REGIONS;
regions
```

```
    fra (local, active)
    lnd (remote, active)
```

```
# Connect to the data store deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

-- Create a remote region 'fra'
sql-> CREATE REGION fra;
Statement completed successfully

- List the regions
sql-> SHOW REGIONS;
regions

    lnd (local, active)
    fra (remote, active)
```

Create Multi-Region Tables

You must create an MR Table on each data store in the connected graph, and specify the list of regions that the table should span. For the use case under discussion, you must create the `users` table as an MR Table at both the regions, in any order.

Steps:

To create an MR Table:

1. To create a table definition, use a `CREATE TABLE` statement. See Create Table in the Developers Guide.
2. Optionally, you can verify the regions associated with the MR Table by executing the following command from the `kv` prompt.

```
kv-> SHOW TABLE -NAME <table name>
```

Example:

Create an MR Table called `users` in both the regions, Frankfurt and London.

```
# Connect to the data store deployed in the 'fra' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- Create the users MR Table
sql-> CREATE TABLE users(
   -> id INTEGER,
   -> name STRING,
   -> team STRING,
   -> PRIMARY KEY (id))
   -> IN REGIONS fra,lnd;
Statement completed successfully
```

```
# Connect to the data store deployed in the 'fra' region from the kv prompt
[~]$java -jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

# Verify the regions associated with the users MR table
kv-> SHOW TABLE -NAME users
{
  "json_version": 1,
  "type": "table",
  "name": "users",
  "regions": {
    "1": "fra",
    "2": "lnd"
  },
  "fields": [
    {
      "name": "id",
      "type": "INTEGER",
      "nullable": false
    },
    {
      "name": "name",
      "type": "STRING",
      "nullable": true
    },
    {
      "name": "team",
      "type": "STRING",
      "nullable": true
    }
  ],
  "primaryKey": [
    "id"
  ],
  "shardKey": [
    "id"
  ]
}


# Connect to the data store deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

-- Create the users MR Table
sql-> CREATE TABLE users(
   -> id INTEGER,
   -> name STRING,
   -> team STRING,
   -> PRIMARY KEY (id))
   -> IN REGIONS lnd,fra;
Statement completed successfully

# Connect to the data store deployed in the 'lnd' region from the kv prompt
```

```
[~]$java -jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

# Verify the regions associated with the users MR table
kv-> SHOW TABLE -NAME users
{
  "json_version": 1,
  "type": "table",
  "name": "users",
  "regions": {
    "2": "fra",
    "1": "lnd"
  },
  "fields": [
    {
      "name": "id",
      "type": "INTEGER",
      "nullable": false
    },
    {
      "name": "name",
      "type": "STRING",
      "nullable": true
    },
    {
      "name": "team",
      "type": "STRING",
      "nullable": true
    }
  ],
  "primaryKey": [
    "id"
  ],
  "shardKey": [
    "id"
  ]
}
```

Create multi-region table with an MR_COUNTER column

You can create a multi-region table containing a column of MR_COUNTER datatype. MR_COUNTER datatype is used in such situations to take care of conflict resolution that may arise when the same data is modified across different regions. MR_COUNTER ensures that though data modifications happen simultaneously in different regions, the data can always be merged into a consistent state. This merge is performed automatically by the MR_COUNTER data type without requiring any special conflict resolution code or user intervention. To learn more about MR_COUNTER datatype, see Using CRDT datatype in a multi-region table section in the Concepts Guide.

Example:

Create an MR Table called `users` with a `MR_COUNTER` datatype in both the regions, Frankfurt and London.

```
-- Create the users MR Table
sql-> CREATE TABLE users(
   -> id INTEGER,
   -> name STRING,
   -> team STRING,
   -> count INTEGER AS MR_COUNTER,
   -> PRIMARY KEY (id))
   -> IN REGIONS fra,lnd;
Statement completed successfully


# Verify the regions associated with the users MR table
sql-> DESC AS JSON TABLE users
{
   "json_version": 1,
   "type": "table",
   "name": "users",
   "regions": {
      "1": "fra",
      "2": "lnd"
   },
   "fields": [
   {
      "name": "id",
      "type": "INTEGER",
      "nullable": false
   },
   {
      "name": "name",
      "type": "STRING",
      "nullable": true
   },
   {
       "name": "team",
       "type": "STRING",
       "nullable": true
   },
   {
       "name" : "count",
       "type" : "INTEGER",
       "nullable" : false,
       "default" : 0,
       "MRCounter" : true
  }
],
   "primaryKey": [
       "id"
   ],
   "shardKey": [
       "id"
```

```
   ]
}


# Connect to the KVStore deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
     -helper-hosts host4:5000,host5:5000,host6:5000 \
     -store mrtstore
-- Create the users MR Table
sql-> CREATE TABLE users(
   -> id INTEGER,
   -> name STRING,
   -> team STRING,
   -> count INTEGER AS MR_COUNTER,
   -> PRIMARY KEY (id))
   -> IN REGIONS lnd,fra;
Statement completed successfully


# Verify the regions associated with the users MR table
sql-> DESC AS JSON TABLE users
{
   "json_version": 1,
   "type": "table",
   "name": "users",
   "regions": {
      "2": "fra",
      "1": "lnd"
   },
   "fields": [
   {
      "name": "id",
      "type": "INTEGER",
      "nullable": false
   },
   {
      "name": "name",
      "type": "STRING",
      "nullable": true
   },
   {
       "name": "team",
       "type": "STRING",
       "nullable": true
   },
   {
        "name" : "count",
        "type" : "INTEGER",
        "nullable" : false,
        "default" : 0,
        "MRCounter" : true
   }
],
   "primaryKey": [
       "id"
   ],
```

```
   "shardKey": [
       "id"
   ]
}
```

To know more details about how to create and use an MR_COUNTER datatype, See Using the MR_COUNTER datatype section in the SQL Reference Guide.

You can use the MR_COUNTER data type in a schema-less JSON field, which means if your Multi-Region table has a JSON column, you can use MR_COUNTER data type inside the JSON column. One or more fields in the JSON column can be of MR_COUNTER data type. The MR_COUNTER data type is a subtype of the INTEGER or LONG or NUMBER data type.

Example:

```
CREATE TABLE demoJSONMR(name STRING,
  jsonWithCounter JSON(counter as INTEGER MR_COUNTER,
                       person.count as LONG MR_COUNTER),
  PRIMARY KEY(name)) IN REGIONS fra,lnd;
```

Access and Manipulate Multi-Region Tables

After creating the MR Table, you can perform read or write operations on the table using the existing data access APIs or DML statements. There is no change to any existing data access APIs or DML statements to work with the MR Tables. See Data Row Management in the *SQL Reference Guide*.

Example:

Perform DML operations on the `users` table in one region, and verify if the changes are replicated to the other region.

```
# To be executed in the fra region
-- Insert two rows into the users MR Table
sql-> INSERT INTO users(id,name,team) VALUES(1,"Amy","HR");
{"NumRowsInserted":1}
1 row returned
sql-> INSERT INTO users(id,name,team) VALUES(2,"Jack","HR");
{"NumRowsInserted":1}
1 row returned

# To be executed in the lnd region
-- Verify if the rows are replicated from the fra region
sql-> SELECT * FROM users;
{"id":1,"name":"Amy","team":"HR"}
{"id":2,"name":"Jack","team":"HR"}

2 rows returned

-- Update the row with id = 2 in the users MR Table
sql-> UPDATE users SET team = "IT" WHERE id = 2;
{"NumRowsUpdated":1}
1 row returned

-- Delete the row with id = 1 from the users MR Table
```

```
sql-> DELETE FROM users WHERE id = 1;
{"NumRowsDeleted":1}
1 row returned

# To be executed in the fra region
-- Verify if the changes are replicated from the lnd region
sql-> SELECT * FROM users;
{"id":2,"name":"Jack","team":"IT"}
1 row returned
```

Stop XRegion Service

In a multi-region setup, you can stop any running Xregion service using `xrstop` command. To get more details about the `xrstop` command, see xrstop.

Example:

Stop the XRegion Service in both the regions, Frankfurt and London.

```
# Stopping the XRegion Service in the fra region
[~]$ java -Xmx1024m -Xms256m -jar $KVHOME/lib/kvstore.jar xrstop \
-config <path to the json config file>
```

Similarly, you must stop the XRegion Service in the other region, lnd.

Use Case 2: Expand a Multi-Region Table

An organization deploys two on-premise data stores, one each at Frankfurt and London. As per their requirement, they create a few MR Tables in both the regions. The `users` table is one of the many MR Tables created and maintained by this organization. Now, they decide to expand their organization by adding another NoSQL Database in Dublin. After creating Dublin as the new region, they want to expand the existing MR Tables to the new region. In the next few topics, you learn how to add the Dublin region to the `users` table that you already created in the previous use case.

If you have not created the `users` MR Table earlier, execute the steps outlined in Use Case 1: Set up Multi-Region Environment.

Prerequisites

Steps:

Before expanding the `users` table to the new region, you must have set up the new region by executing the following tasks:

1. Set up a multi-region NoSQL Database with two regions Frankfurt (`fra`) and London (`lnd`). See Use Case 1: Set up Multi-Region Environment.
2. Deploy a local data store with store name as `dubstore` in the new region. See Configuring your data store installation.
3. Set the new region's local region name to `dub`. See Set Local Region Name.
4. Configure and start the XRegion Service in the `dub` region. See Configure XRegion Service and Start XRegion Service.

5. Update the `json.config` file in the existing regions, that is, Frankfurt (`fra`) and London (`lnd`) to include `dub` (Dublin) as a remote region.

> **Note:**
>
> You must restart the agent at existing regions to pick up the new region (`dub`) from the `json.config` file.

6. Create two remote regions, `fra` and `lnd` in the new region `dub`. See Create Remote Regions.

Example:

1. Set the local region name for the new region, Dublin.

```
# Connect to the data store deployed at host7, host8, and host9 from the
SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host7:5000,host8:5000,host9:5000 \
-store dubstore

-- Set the local region name to 'dub'
sql-> SET LOCAL REGION dub;
Statement completed successfully

-- List the regions
sql-> SHOW REGIONS;
regions
    dub (local, active)
```

2. Create a `json.config` file for the new region, Dublin.

```
# Contents of the configuration file in the 'dub' Region
{
  "path": "<entire path to the home directory for the XRegion Service>",
  "agentGroupSize": 1,
  "agentId": 0,
  "region": "dub",
  "store": "<storename at the dub region>",
  "security": "<path to the security file>",
  "helpers": [
    "host7:5000",
    "host8:5000",
    "host9:5000"
  ],
  "regions": [
    {
      "name": "fra",
      "store": "<store name at the fra region>",
      "security": "<path to the security file>",
      "helpers": [
        "host1:5000",
        "host2:5000",
        "host3:5000"
```

```
      ]
    },
    {
      "name": "lnd",
      "store": "<store name at the lnd region>",
      "security": "<path to the security file>",
      "helpers": [
        "host4:5000",
        "host5:5000",
        "host6:5000"
      ]
    }
  ]
}
```

3. Start the XRegion Service in the new region, Dublin.

```
# Start the XRegion Service in the 'dub' Region
[oracle@host7 xrshome]$ java -Xms256m -Xmx2048m -jar $KVHOME/lib/
kvstore.jar xrstart \
-config <path to the json config file> -bg

[1] 24123

# View the status of the xrstart command in the 'dub' Region
[oracle@host7 xrshome]$ cat status.<numbe of agents>.<agentId>.txt
Cross-region agent (region=fra,store=mrtstore, helpers=[host7:5000,
host8:5000, host9:5000])
starts up from config file=/home/oracle/xrshome/ json.config at 2024-04-05
08:57:34 UTC
```

4. Modify the `json.config` files in the existing regions (Frankfurt and London) to include Dublin as a remote region.

```
# Contents of the configuration file in the 'fra' Region
{
  "path": "<path to the json config file>",
  "agentGroupSize": 1,
  "agentId": 0,
  "region": "fra",
  "store": "<storename at the fra region>",
  "security": "<path to the security file>",
  "helpers": [
    "host1:5000",
    "host2:5000",
    "host3:5000"
  ],
  "regions": [
    {
      "name": "lnd",
      "store": "<storename at the lnd region>",
      "security": "<path to the security file>",
      "helpers": [
        "host4:5000",
        "host5:5000",
        "host6:5000"
```

```
      ]
    },
    {
      "name": "dub",
      "store": "<storename at the dub region>",
      "security": "<path to the security file>",
      "helpers": [
        "host7:5000",
        "host8:5000",
        "host9:5000"
      ]
    }
  ]
}
```

 # Contents of the configuration file in the 'lnd' Region

```
{
  "path": "<path to the json config file>",
  "agentGroupSize": 1,
  "agentId": 0,
  "region": "lnd",
  "store": "<storename at the lnd region>",
  "security": "<path to the security file>",
  "helpers": [
    "host4:5000",
    "host5:5000",
    "host6:5000"
  ],
  "regions": [
    {
      "name": "fra",
      "store": "<storename at the fra region>",
      "security": "<path to the security file>",
      "helpers": [
        "host1:5000",
        "host2:5000",
        "host3:5000"
      ]
    },
    {
      "name": "dub",
      "store": "<storename at the dub region>",
      "security": "<path to the security file>",
      "helpers": [
        "host7:5000",
        "host8:5000",
        "host9:5000"
      ]
    }
  ]
}
```

5. Create two remote regions, `fra` and `lnd` in the new region, Dublin.

```
# Connect to the data store deployed in the 'dub' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host7:5000,host8:5000,host9:5000 \
-store dubstore

-- Create remote regions 'fra' and 'lnd'
sql-> CREATE REGION fra;
Statement completed successfully
sql-> CREATE REGION lnd;
Statement completed successfully

- List the regions
sql-> SHOW REGIONS;
regions

    dub (local, active)
    fra (remote, active)
    lnd (remote, active)
```

Create MR Table in New Region

Steps:

As a first step in expanding an MR Table to a new region, you must create the MR Table in the new region using the `CREATE TABLE` statement. See Create Multi-Region Tables.

> **Note:**
>
> Creating the MR Table in the new region alone does not ensure replicating the data from the existing regions. This is because you have not yet linked the new region to this MR Table from the existing regions.

Example:

Create the `users` MR Table in the new region, Dublin.

```
# Connect to the data store deployed in the 'dub' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host7:5000,host8:5000,host9:5000 \
-store dubstore

-- Create the users MR Table
sql-> CREATE TABLE users(
   -> id INTEGER,
   -> name STRING,
   -> team STRING,
   -> PRIMARY KEY (id))
   -> IN REGIONS dub,fra,lnd;
Statement completed successfully

# Connect to the data store deployed in the 'dub' region from the kv prompt
```

```
[~]$java -jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts host7:5000,host8:5000,host9:5000 \
-store dubstore

# Verify the regions associated with the users MR table
kv-> SHOW TABLE -NAME users
{
  "json_version": 1,
  "type": "table",
  "name": "users",
  "regions": {
    "1": "dub",
    "2": "fra"
    "3": "lnd"
  },
  "fields": [
    {
      "name": "id",
      "type": "INTEGER",
      "nullable": false
    },
    {
      "name": "name",
      "type": "STRING",
      "nullable": true
    },
    {
      "name": "team",
      "type": "STRING",
      "nullable": true
    }
  ],
  "primaryKey": [
    "id"
  ],
  "shardKey": [
    "id"
  ]
}
```

Add New Region to Existing Regions

As a next step, you must create the new region as a remote region in the existing regions. Then, you must associate the new region with the MR Table in the existing regions.

Steps:

Execute the following steps *from each existing region*:

1. Add the new region as a remote region. See Create Remote Regions.
2. Associate the new region with the existing MR Table using the DDL command shown below.

```
ALTER TABLE <table name> ADD REGIONS <region name>;
```

Example:

1. Add the new region, Dublin as a remote region from the existing regions, Frankfurt and London.

```
# Connect to the data store deployed in the 'fra' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- Create a remote region 'dub'
sql-> CREATE REGION dub;
Statement completed successfully

- List the regions
sql-> SHOW REGIONS;
regions

    fra (local, active)
    lnd (remote, active)
    dub (remote, active)


# Connect to the data store deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

-- Create a remote region 'dub'
sql-> CREATE REGION dub;
Statement completed successfully

- List the regions
sql-> SHOW REGIONS;
regions

    lnd (local, active)
    fra (remote, active)
    dub (remote, active)
```

2. In the existing regions, alter the `users` MR Table to add the new region, Dublin.

```
# Connect to the data store deployed in the 'fra' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- Add the 'dub' region to the users MR Table
sql-> ALTER TABLE users ADD REGIONS dub;
Statement completed successfully

# Connect to the data store deployed in the 'fra' region from the kv prompt
[~]$java -jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore
```

```
# Verify the regions associated with the users MR table
kv-> SHOW TABLE -NAME users
{
  "json_version": 1,
  "type": "table",
  "name": "users",
  "regions": {
    "1": "fra",
    "2": "lnd"
    "3": "dub"
  },
  "fields": [
    {
      "name": "id",
      "type": "INTEGER",
      "nullable": false
    },
    {
      "name": "name",
      "type": "STRING",
      "nullable": true
    },
    {
      "name": "team",
      "type": "STRING",
      "nullable": true
    }
  ],
  "primaryKey": [
    "id"
  ],
  "shardKey": [
    "id"
  ]
}


# Connect to the data store deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

-- Add the 'dub' region to the users MR Table
sql-> ALTER TABLE users ADD REGIONS dub;
Statement completed successfully

# Connect to the data store deployed in the 'lnd' region from the kv prompt
[~]$java -jar $KVHOME/lib/kvstore.jar runadmin \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

# Verify the regions associated with the users MR table
kv-> SHOW TABLE -NAME users
{
  "json_version": 1,
```

```
  "type": "table",
  "name": "users",
  "regions": {
    "1": "lnd",
    "2": "fra"
    "3": "dub"
  },
  "fields": [
    {
      "name": "id",
      "type": "INTEGER",
      "nullable": false
    },
    {
      "name": "name",
      "type": "STRING",
      "nullable": true
    },
    {
      "name": "team",
      "type": "STRING",
      "nullable": true
    }
  ],
  "primaryKey": [
    "id"
  ],
  "shardKey": [
    "id"
  ]
}
```

Access MR Table in New and Existing Regions

After performing the tasks discussed in the previous sections, you can perform read/write operations on the MR Table from the new region without any disruption. However, the table may not return the complete data from the existing regions until the initialization completes in the background. Especially if the MR Table has a huge volume of data in the existing regions, it may take a while for the new table to see the data from the remote regions.

Similarly, you can continue performing read/write operations on the MR Table from the existing regions without any disruption. Adding a new region is transparent to the customers accessing the MR Table from the existing regions. However, the MR Table at the existing regions may also need initialization to see the writes from the new region. If the table at the new region is empty or small, the existing regions will quickly sync up with it. To learn how to access the MR Tables, see Access and Manipulate Multi-Region Tables.

Use Case 3: Contract a Multi-Region Table

An organization deploys three on-premise data stores, one each at Frankfurt, London, and Dublin. As per their requirement, they created a few MR Tables in all three regions. The `users` table is one of the many MR tables created and maintained by this organization. As per some changes in their business requirements, they decided to remove the `users` table from the Dublin region. In the next few topics, you learn how to contract an MR Table, that is, how to remove an MR Table from specific regions.

If you have not created the `users` MR table earlier, execute the steps outlined in Use Case 1: Set up Multi-Region Environment.

If you have not added the Dublin region to the `users` MR table, execute the steps outlined in Use Case 2: Expand a Multi-Region Table.

Alter MR Table to Drop Regions

Learn how to contract a Multi-Region table and reduce the regions it spans across.

Steps:

To remove an MR Table from a specific region in a Multi-Region NoSQL Database setup, you must execute the following steps from all the other participating regions.

1. Execute the following command from the `sql` prompt.

   ```
   ALTER TABLE <table name> DROP REGIONS <comma separated list of regions>
   ```

2. Optionally, you can execute the following command from the kv prompt to verify that the region is dropped successfully.

   ```
   SHOW TABLE -NAME <table name>
   ```

> **Note:**
>
> Suppose you drop region A from an MR table created in region B. Then:
>
> - Region B can't see any new writes on this MR table from the region A.
> - Region A continues to see the writes on this MR Table from the region B.
>
> If you want to isolate the MR table in the region A from other regions, you must drop those regions from the MR table created in region A. This is only a recommendation and not a mandatory step in contracting an MR Table.

Example:

Drop the Dublin region from the `users` MR table in the other two regions, Frankfurt and London.

```
# Connect to the data store deployed in the 'fra' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- drop the 'dub' region from the 'users' MR table
sql-> ALTER TABLE users DROP REGIONS dub;
Statement completed successfully


# Connect to the data store deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore
```

```
-- drop the 'dub' region from the 'users' MR table
sql-> ALTER TABLE users DROP REGIONS dub;
Statement completed successfully
```

Use Case 4: Drop a Region

An organization deploys three on-premise data stores, one each at Frankfurt, London, and Dublin. As per their requirement, they created a few MR Tables in all three regions. As part of business down-sizing, they decided to exclude the Dublin region resulting in a two-region NoSQL Database. In the next few topics, you learn how to drop an existing region from the NoSQL environment that you had set up in the previous sections.

If you have not set up a Multi-Region NoSQL Database with three regions already, execute the steps outlined in:

- Use Case 1: Set up Multi-Region Environment
- Use Case 2: Expand a Multi-Region Table

Prerequisites

Learn about the conditions to be satisfied before dropping a region from a Multi-Region NoSQL Database.

Before removing a region from a Multi-Region NoSQL Database, it is recommended to:

- Stop writing to all the MR Tables linked to that region.
- Ensure that all writes to the MR Tables in that region have replicated to the other regions. This helps in maintaining consistent data across the different regions.

> **Note:**
>
> As of the current release, there is no provision in Oracle NoSQL Database to make a table read-only. Hence, you must stop writes to the identified MR Tables at the application level.

Isolate the Region

Learn how to isolate a region from a Multi-Region NoSQL Database.

When you decide to drop a region, it is a good practice to isolate that region from all the other participating regions. Isolating a region disconnects it from all the MR Tables in the Multi-Region NoSQL Database.

Isolating a region ensures that:

- The isolated region cannot see writes from the other regions.
- The other regions cannot see writes from the isolated region.

> **Note:**
>
> Even though it is not mandatory to isolate the region before dropping it from a Multi-Region NoSQL Database, this is considered a cleaner approach and hence suggested.

Steps:

Isolating a region from the Multi-Region NoSQL Database environment involves two tasks. They are:

1. Drop the target region from all the MR Tables in the other regions using the DDL command shown below.
2. Drop all the other regions from all the MR Tables in the region to be isolated.

See Alter MR Table to Drop Regions.

Example:

1. Drop the Dublin region from the `users` MR table in the other two regions, Frankfurt and London.

```
# Connect to the data store deployed in the 'fra' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- drop the 'dub' region from the 'users' MR table
sql-> ALTER TABLE users DROP REGIONS dub;
Statement completed successfully


# Connect to the data store deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

-- drop the 'dub' region from the 'users' MR table
sql-> ALTER TABLE users DROP REGIONS dub;
Statement completed successfully
```

2. Drop the other regions (Frankfurt and London) from the `users` MR table in the Dublin region.

```
# Connect to the data store deployed in the 'dub' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host7:5000,host8:5000,host9:5000 \
-store dubstore

-- drop 'fra' and 'lnd' regions from the 'users' MR table
sql-> ALTER TABLE users DROP REGIONS fra,lnd;
Statement completed successfully
```

Drop MR Tables in the Isolated Region

After you ensure that the region to be dropped is isolated, you can drop all the MR Tables created in that region safely. Dropping an MR Table is exactly similar to dropping a local table.

Example:

Drop `users` MR table from the isolated region, Dublin.

```
# Connect to the data store deployed in the 'dub' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host7:5000,host8:5000,host9:5000 \
-store dubstore

-- drop the 'users' MR table
sql-> DROP TABLE users;
Statement completed successfully
```

Drop the Isolated Region

Finally, you can drop the isolated region from all the other regions.

> **Note:**
>
> Dropping an isolated region is not mandatory. You can retain the isolated region without dropping from other regions, for any future use.

Steps:

To drop the isolated region from other regions:

1. Connect to the `sql` prompt, and connect to the local KVStore.
2. Execute the following DDL command from the SQL prompt.

   ```
   DROP REGION <region name>;
   ```

3. Optionally, you can execute the following command to verify that the isolated region is dropped successfully.

   ```
   SHOW REGIONS;
   ```

Example:

Drop the Dublin region from the other two regions, Frankfurt and London.

```
# Connect to the data store deployed in the 'fra' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host1:5000,host2:5000,host3:5000 \
-store mrtstore

-- drop the 'dub' region
sql-> DROP REGION dub;
```

```
Statement completed successfully

- List the regions
sql-> SHOW REGIONS;
regions

    fra (local, active)
    lnd (remote, active)


# Connect to the data store deployed in the 'lnd' region from the SQL shell
[~]$java -jar $KVHOME/lib/sql.jar \
-helper-hosts host4:5000,host5:5000,host6:5000 \
-store mrtstore

-- drop the 'dub' region
sql-> DROP REGION dub;
Statement completed successfully

- List the regions
sql-> SHOW REGIONS;
regions

    lnd (local, active)
    fra (remote, active)
```

Use Case 5: Backup and Restore a Multi-Region Table

An organization deploys three on-premise data stores, one each at Frankfurt, London, and Dublin, and they have created MR Tables spanning all three regions. A multi-region table is a single table that spans multiple regions and is kept in sync all the time.

One day due to an application bug or illegal modifications, the organization suffers table-level data loss or corruption for an MR table in one region. Since the system keeps the MR table in sync in all the regions, the corruption or data loss gets replicated that to the other regions as well. Therefore, the organization wants to restore the MR table in all the regions from the MR table backup that they have been regularly creating as part of their data safety policy.

In this topic, you learn how to backup and restore an MR Table.

Backup a Multi-Region Table

Creating a backup of MR tables helps you in restoring the table data later in case you suffer inadvertent application corruption of the data.

To create a backup of an MR table:

1. Using multi-region table statistics, find the most up-to-date region for the table that you want to backup. Run the following command in the Admin Command Line Interface (CLI):

   ```
   show mrtable-agent-statistics -agent 0 -json
   ```

   For more information about MR table statistics, see show mrtable-agent-statistics.

2. In the statistics returned by the show mrtable-agent-statistics command, locate `"returnValue"[]."statistics"."regionStat"[]."laggingMs"."max"` attribute and find

the region that has the smallest value for the `max` attribute in the `laggingMS` field. That region contains the most up-to-date data of your MR table.
In the example below, the Frankfurt region has the smallest value for the `max` attribute in the `laggingMS` field, and hence it has the most up-to-date data for the MR table.

```
kv-> show mrtable-agent-statistics -agent 0 -json
```

Output:

```
{
  "operation": "show mrtable-agent-statistics",
  "returnCode": 5000,
  "description": "Operation ends successfully",
  "returnValue": {
    "XRegionService-1_0": {
      "timestamp": 1592901180001,
      "statistics": {
        "agentId": "XRegionService-1_0",
        "beginMs": 1592901120001,
        "dels": 1024,
        "endMs": 1592901180001,
        "incompatibleRows": 100,
        "intervalMs": 60000,
        "localRegion": "slc1",
        "persistStreamBytes": 524288,
        "puts": 2048,
        "regionStat": {
          "fra": {
            "completeWriteOps": 10,
            "laggingMs": {
              "avg": 502,
              "max": 885,
              "min": 26
            },
            "lastMessageMs": 1591594977587,
            "lastModificationMs": 1591594941686,
            "latencyMs": {
              "avg": 20,
              "max": 40,
              "min": 10
            }
          },
          "lnd": {
            "completeWriteOps": 10,
            "laggingMs": {
              "avg": 512,
              "max": 998,
              "min": 31
            },
            "lastMessageMs": 1591594977587,
            "lastModificationMs": 1591594941686,
            "latencyMs": {
              "avg": 20,
              "max": 40,
              "min": 10
```

```
            }
          },
          "dub": {
            "completeWriteOps": 20,
            "laggingMs": {
              "avg": 535,
              "max": 1024,
              "min": 45
            },
            "lastMessageMs": 1591594978254,
            "lastModificationMs": 1591594956786,
            "latencyMs": {
              "avg": 30,
              "max": 45,
              "min": 15
            }
          }
        },
        "requests": 12,
        "responses": 12,
        "streamBytes": 1048576,
        "winDels": 1024,
        "winPuts": 2048
      }
    }
  }
}
```


3. Using the Oracle NoSQL Database Migrator connect to the region identified in **Step#2** in the Source configuration to export the MR tables. And use the appropriate Sink type based on your requirement to import the MR tables. For more information on the source and sink see, Using Oracle NoSQL Data Migrator.

> **Note:**
>
> Make sure that you save the backup of the MR table on remote storage, which is not local to a NoSQL Storage Node in the NoSQL topology.

Restore a Multi-Region Table

You can restore an MR table from an MR table backup in case you suffer data loss or data corruption and want to revert to the most up-to-date version of the MR table.

> **Tip:**
>
> Oracle recommends that you stop all the write activity to the MR tables that are being restored.

To restore an MR table from backup:

1. Find the list of regions associated with the MR Table by executing the following command from the kv prompt.

   ```
   kv-> SHOW TABLE -NAME <table name>
   ```

   For example,

   ```
   kv-> SHOW TABLE -NAME users
   ```

   Output:

   ```
   {
     "json_version": 1,
     "type": "table",
     "name": "users",
     "regions": {
       "1": "fra",
       "2": "lnd"
     },
    .....
    .....
   }
   ```

2. Using the `DROP TABLE` statement, drop the MR table in each region with which the MR table is associated. For more information on how to drop an MR table, see Using TableRequest API to drop table.
3. Re-create the MR table in every region, specifying the remote regions you want to associate with the MR table. For more information, see Create Multi-Region Tables.
4. Using the Oracle NoSQL Database Migrator connect to any one region identified in Step#1 in the Sink configuration to restore the MR tables. And use the appropriate Source configuration type based on the where the MR table backup resides. During the loading of the backup, the Oracle NoSQL Database synchronizes the table in each remote region. For more information on see, Using Oracle NoSQL Data Migrator.

Troubleshooting multi-region data store setup

1. **Find agent logs for a multi-region setup:**
   Users can find the logs of an XRegion agent at the path specified in the JSON config file. The agent logs, like data store logs, contain all diagnostic information from the service agent. To learn more about the JSON config file used by the XRegion agent, see Configure XRegion Service.
2. **Access the statistics of an XRegion agent**
   The XRegion agent collects statistics periodically and posts it to a system table in the local region. You can query the system table for XRegion agent statistics by using the standard CLI command “SHOW” that returns a JSON string of agent statistics.

   ```
   show mrtable-agent-statistics
           [-agent <agentID>][-table <tableName>][-json]
   ```

   The `show` command with `mrtable-agent-statistics` option shows the latest statistics as of the last one minute for the XRegion agent. With no arguments, this command shows the combined statistics over all regions that the multi-region table spans. You can limit the

statistics to a particular agent by specifying the agent id. If a table name is specified in the command, the statistics is limited to a particular multi-region table. To understand more details about using the show command to obtain statistics for a multi-region setup, see show mrtable-agent-statistics.

3. **Display the status of a multi-region table syncing up with remote regions**
The statistic `lastModificationMs` in the `show mrtable-agent-statistics` command is the timestamp of the last operation performed in each remote region, in milliseconds. By comparing the values of this statistic of the local region and the remote region, you can determine if the remote region has caught up with the local region or still lagging behind.

   For example, suppose the time of the last write made to a remote region is T1, while the statistic lastModificationMs for the local region is T2. If T2 < T1, it means that the multi-region table has caught up with that remote region for all writes up to T2 and will continue catching up for all writes made in between T2 and T1. If T2 = T1, that means the multi-region table has caught up with all writes made at the remote region. However T2 can never be greater than T1.

```
# MR table agent statistics for a specific agent
kv-> show mrtable-agent-statistics -agent 0 -json
```

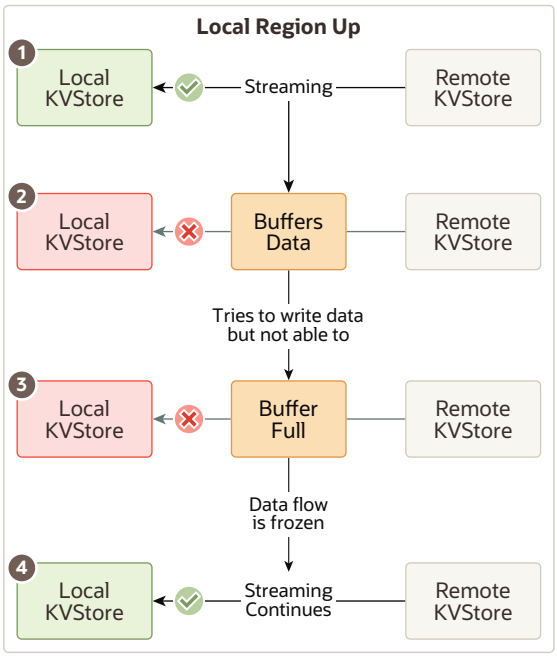
Output:


```
{
   "operation": "show mrtable-agent-statistics",
   "returnCode": 5000,
   "description": "Operation ends successfully",
   "returnValue": {
      "XRegionService-1_0": {
      "timestamp": 1592901180001,
      "statistics": {
         "agentId": "XRegionService-1_0",
         "beginMs": 1592901120001,
         "dels": 1024,
         "endMs": 1592901180001,
         "incompatibleRows": 100,
         "intervalMs": 60000,
         "localRegion": "slc1",
         "persistStreamBytes": 524288,
         "puts": 2048,
         "regionStat": {
            "lnd": {
               "completeWriteOps": 10,
               "laggingMs": {
                  "avg": 512,
                  "max": 998,
                  "min": 31
               },
               "lastMessageMs": 1591594977587,
               "lastModificationMs": 1591594941686,
               "latencyMs": {
                  "avg": 20,
                  "max": 40,
                  "min": 10
               }
            },
```

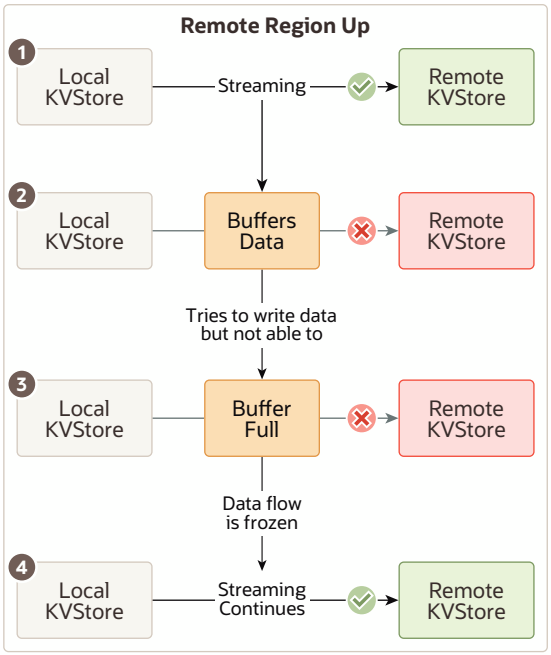
```
                "dub": {
                   "completeWriteOps": 20,
                   "laggingMs": {
                      "avg": 535,
                      "max": 1024,
                      "min": 45
                   },
                   "lastMessageMs": 1591594978254,
                   "lastModificationMs": 1591594956786,
                   "latencyMs": {
                      "avg": 30,
                      "max": 45,
                      "min": 15
                   }
                }
             },
             "requests": 12,
             "responses": 12,
             "streamBytes": 1048576,
             "winDels": 1024,
             "winPuts": 2048
              }
           }
        }
}
```

4. **Troubleshoot problems with XRegion Agent**
   If the XRegion agent encounters a problem, for example if the network connection is dropped, you should investigate the reason of the connection failure and come up with a solution to fix the connection. Meanwhile the XRegion agent would try to re-connect to the remote region until the remote region is up again. After successfully re-connecting to the remote region, the XRegion agent will resume from the stream position or the last checkpoint made, before the connection was dropped. During re-connection, the agent may dump warning messages in the log to alert users that the connection to a region or a shard in that region is lost.

5. **Troubleshoot when the local region or remote region goes down**
   The XRegion agent streams changes to the multi-region table from each remote region and persists them in the local region. Therefore, if the local region is down, the agent will keep retrying but won't be able to write any changes. After a period of time, when the buffer in the XRegion agent is full, the XRegion agent will stop streaming data from the remote regions and the data flow gets frozen. When the local region is back, the XRegion agent will just resume the stream and the workflow. No manual intervention to the XRegion agent is needed here. However you may have to fix the issue with the local region manually.

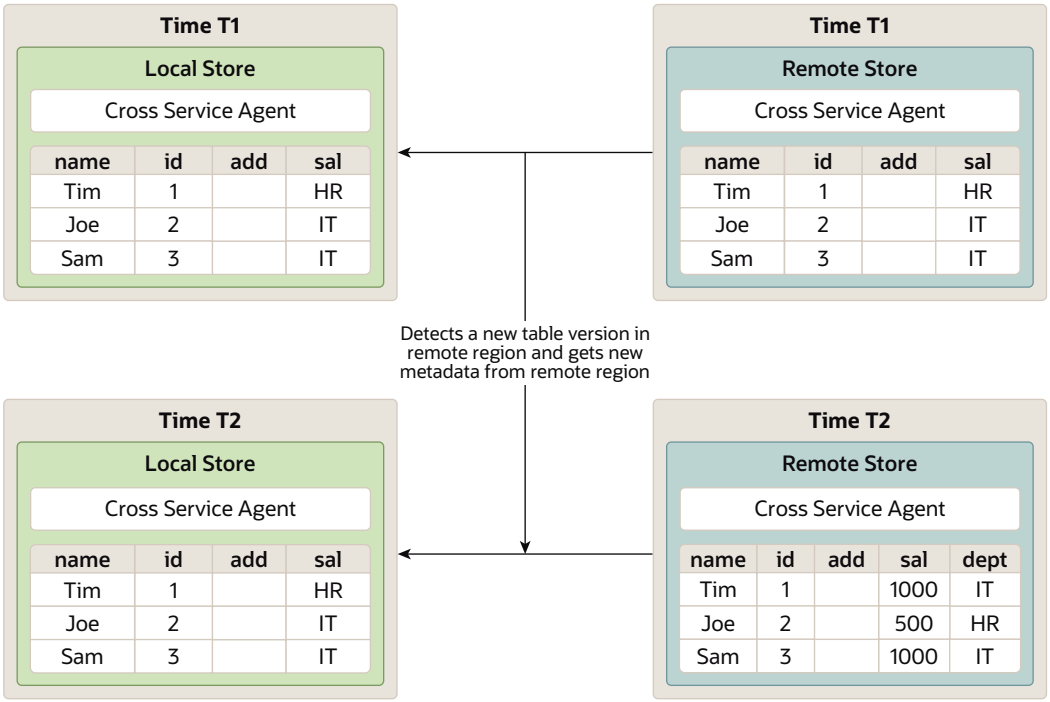


If a particular remote region is down, the XRegion agent will just keep retrying till that remote region is back. This issue is similar to any network connection problem with the XRegion agent. Until the connection to the remote region is established again, the multi-region table at the local region won't be able to see the changes in that remote region. But changes in the other remote regions are not affected as long as the XRegion agent is able to maintain the connection to these regions.

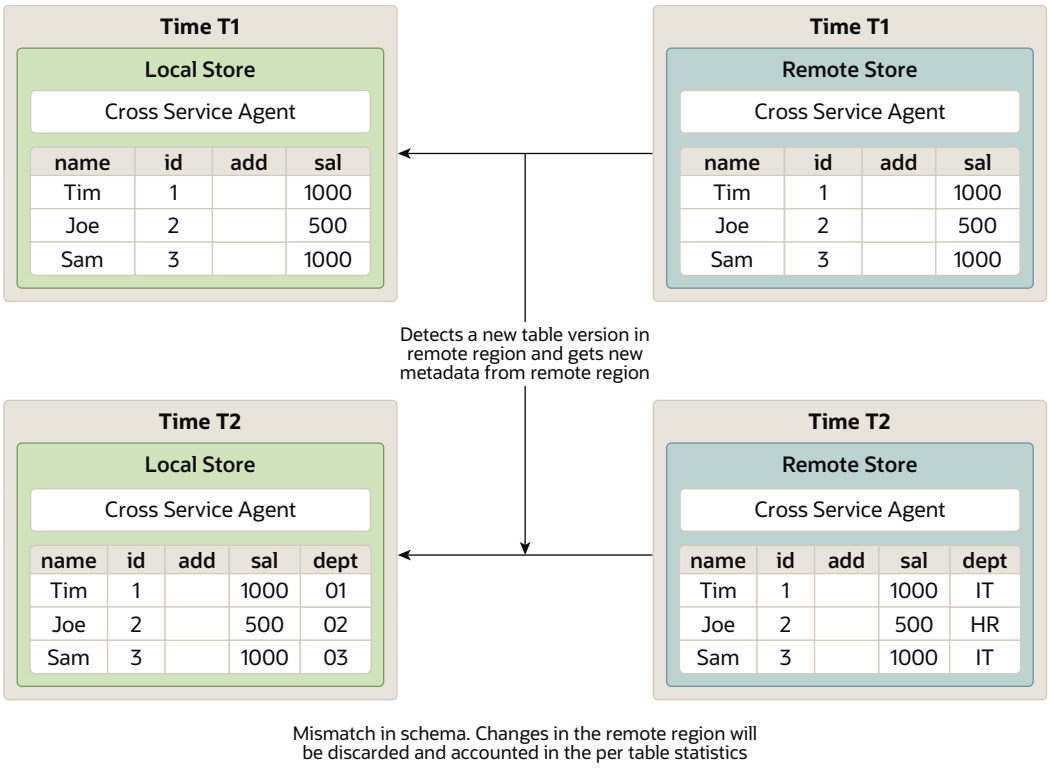


6. **Handle schema evolution in a multi-region setup**
   Schema evolution happens when there is a schema change in any of the remote regions. Then the schema of a multi-region table at the local region differs from that in the remote region. In such a situation the XRegion agent will try to solve the difference by converting a row from the remote region to the schema of local region. For example, if you add a new

column to a multi-region table at a remote region but this new column is not yet added in the local region. The multi-region table at the local region will not be able to see the new column in the changes streamed from the remote region, but the local region should still see the other columns. This would last until you fix the problem by adding the same column in the local region to end the schema divergence. In a multi-region table, there is no automatic notification to other regions when a schema changes in one region. The XRegion agent of local region is able to detect the change when it sees the data from a remote region with higher table version, and it will refresh its table metadata from the remote region to get the latest schema.



Consider the situation when the schema in different regions diverge in a way that the agent is not able to fix the schema differences by refreshing the local region table metadata from the remote region. For example, if you add a new column “Foo” with type “STRING” to the remote region but adds the same column with type “LONG” in the local region, these changes at the remote region are considered incompatible to the local region, and the agent cannot fix this difference. These changes from the remote region will not be persisted locally. Consequently the changes in the remote regions will be discarded and accounted in the per-table statistic `incompatibleRows`. See the details about persistence of remote data in the show mrtable-agent-statistics section.



7. **Handle difference in software versions between regions**
   For any particular region, you need to upgrade the data store first and then upgrade the agent to the same version. If a multi-region table has different versions of software on different regions, the agent with old version may not be able to process the rows streamed from regions with a newer version of the software correctly, and some data may be treated by the old agent as incompatible for operations. For example, if the local region is upgraded to support TTL (Time to Live) while the remote region has not yet upgraded, the changes made to the remote region will be persisted to the local region, but without any expiration information, that means the row will never expire. The same is the case if the remote region has upgraded to support TTL while the local region has not. Then all changes to the remote region with TTL will lose their TTL when applied to the local region, which means these rows will never expire. If this is undesirable, you should upgrade all regions first before writing the data to the table to ensure every region can process the data correctly. Any feature will be completely available to a multi-region table only after all the regions have upgraded to the same version.

4
Administer

The articles in this section include various tasks to administer an Oracle NoSQL Database.

Changing the Store's Topology

Topics:

Determining Your Store's Configuration

A store consists of a number of Storage Nodes. Each Storage Node can host one or more Replication Nodes, based on its storage capacity. The term `topology` is used to describe the distribution of Replication Nodes. A topology is derived from the number and capacity of available Storage Nodes, the number of partitions in the store, and the replication factors of the store's zones. Topology layouts are also governed by a set of rules that maximize the availability of the store.

All topologies must adhere to the following rules:

1. Each Replication Node from the same shard must reside on a different Storage Node. This rule prevents a single Storage Node failure causing multiple points of failure for a single shard.
2. The number of Replication Nodes assigned to a Storage Node must be less than or equal to the capacity of Storage Nodes.
3. A zone must have one or more Replication Nodes from each shard.
4. A valid Arbiter Node distribution is one in which the Arbiter Node is hosted on a Storage Node that does not contain other members of its shard.

The store’s initial configuration, or topology, is set when you create the store. Over time, it can be necessary to change the store topology. There are several reasons for such a change:

1. You need to replace or upgrade an existing Storage Node.
2. You need to increase read throughput. This is done by increasing the replication factor and creating more copies of the store's data which can be used to service read only requests.
3. You need to increase write throughput. Since each shard has a single master node, distributing the data in the store over a larger number of shards provides the store with more nodes to execute write operations.

You change the store's configuration by changing the number or capacity of available Storage Nodes, or the replication factor of a zone. To change from one configuration to another, you either create a new initial topology, or `clone` an existing topology and modify it into your target topology. You then deploy this target topology.

> **Note:**
>
> Deploying a target topology can be a lengthy operation. Plus, the time required scales with the amount of data to move. During the deployment, the system updates the topology at each step. Because of that, the store passes through intermediate topologies which you did not explicitly create.

This chapter discusses how to make configuration or topological changes to a store. It also describes how to deploy a topology enabled with Arbiter Nodes.

> **Note:**
>
> Do not make configuration changes while a taking a snapshot, or take a snapshot when changing the configuration. Before making configuration changes, we recommend you first create a snapshot as a backup. For additional information on creating snapshots, see Taking a Snapshot.

Steps for Changing the Store's Topology

When you change your topology, you should go through these steps:

1. Make the Topology Candidate
2. Transforming the Topology Candidate
3. View the Topology Candidate
4. Validate the Topology Candidate
5. Preview the Topology Candidate
6. Deploy the Topology Candidate
7. Verify the Store's Current Topology

Creating a new topology is typically an iterative process, trying different options to see what is best before deploying changes. After trying options, examine the topology candidate and decide if it is satisfactory. If not, apply more transformations, or start over with different parameters. You can view and validate topology candidates to determine if they are appropriate.

The possible transformations to expand the store include redistributing data, increasing the replication factor, and rebalancing. These are described in Transforming the Topology Candidate.

You can also decrease the current topology by removing Storage Nodes. See Contracting a Topology.

The following sections walk you through the process of changing your store’s configuration using the Administration CLI.

Make the Topology Candidate

To create the first topology candidate for an initial deployment, before any Replication Nodes exist, use the `topology create` command. The `topology create` command requires a topology name, a pool name and the number of partitions as arguments.

> **Note:**
>
> Avoid using the dollar sign ($) character in topology candidate names. The CLI displays a warning if you try to create or clone topologies whose names contain the reserved character.

For example:

```
kv-> topology create -name firstTopo -pool BostonPool
-partitions 300
```

Output:

```
Created: firstTopo
```

Use the `plan deploy-topology` command to deploy this initial topology candidate without further transformations.

After your store is initially deployed, you can create candidate topologies with the `topology clone` command. The source of a clone can be another topology candidate, or the current, deployed topology. The `topology clone` command takes the following arguments:

- `-from <from topology>`

  The name of the source topology candidate.

- `-name <to topology>`

  The name of the clone.

For example:

```
kv-> topology clone -from topo -name CloneTopo
```

Output:

```
Created CloneTopo
```

This variant of the topology clone command that takes the following arguments:

- `-current`

  Specifies using the current deployed topology as a source, so the argument requires no name.

- `-name <to topology>`

The name of the topology clone.

For example:

```
kv-> topology clone -current -name ClonedTopo
```

Output:

```
Created ClonedTopo
```

Transforming the Topology Candidate

After you initially deploy your store, you can change it by deploying another topology candidate that differs from the current topology. This target topology is generated by transforming a topology candidate to expand the store by using these commands:

- `topology redistribute`
- `rebalance`
- `change-repfactor`

Alternatively, you can contract the target topology candidate using the `topology contract` command.
Transformations follow the topology rules described in the previous section.

The topology rebalance, redistribute or change-repfactor commands can only make changes to the topology candidate if there are additional, or changed, Storage Nodes available. It uses the new resources to rearrange Replication Nodes and partitions so the topology complies with the topology rules and the store improves on read or write throughput.

The following are scenarios in how you might expand or contract the store.

Increase Data Distribution

Use the `topology redistribute` command to increase data distribution to enhance write throughput. The redistribute command works only if new Storage Nodes are added to make creating new replication nodes possible for new shards. With new shards, the system distributes partitions across the new shards, resulting in more Replication Nodes to service write operations.

The following example demonstrates adding a set of Storage Nodes (`node04` — `node07`) and redistributing the data to those nodes. Four Storage Nodes are required to meet the zone's replication factor of four and the new shards require four nodes to satisfy the replication requirements:

```
kv-> plan deploy-sn -zn zn1 -host node04 -port 5000 -wait
Executed plan 7, waiting for completion...
Plan 7 ended successfully

kv-> plan deploy-sn -zn zn1 -host node05 -port 5000 -wait
Executed plan 8, waiting for completion...
Plan 8 ended successfully

kv-> plan deploy-sn -zn zn1 -host node06 -port 5000 -wait
Executed plan 9, waiting for completion...
Plan 9 ended successfully
```

```
kv-> plan deploy-sn -zn zn1 -host node07 -port 5000 -wait
Executed plan 10, waiting for completion...
Plan 10 ended successfully

kv-> pool join -name BostonPool -sn sn4
Added Storage Node(s) [sn4] to pool BostonPool
kv-> pool join -name BostonPool -sn sn5
Added Storage Node(s) [sn5] to pool BostonPool
kv-> pool join -name BostonPool -sn sn6
Added Storage Node(s) [sn6] to pool BostonPool
kv-> pool join -name BostonPool -sn sn7
Added Storage Node(s) [sn7] to pool BostonPool

kv-> topology clone -current -name newTopo
Created newTopo

kv-> topology redistribute -name newTopo -pool BostonPool
Redistributed: newTopo

kv-> plan deploy-topology -name newTopo -wait
Executed plan 11, waiting for completion...
Plan 11 ended successfully
```

The redistribute command incorporates the new Storage Node capacity that you added to the BostonPool, and creates new shards. The command also migrates partitions to the new shards. If the number of new shards is less than or equal to the current number of shards, the `topology redistribute` command fails.

> **Note:**
>
> Do not execute the `topology redistribute` command against a store with mixed shards. A mixed shard store has shards whose Replication Nodes are operating with different software versions of Oracle NoSQL Database.

The system goes through these steps when it is redistributing a topology candidate:

1. The topology redistribute command creates new Replication Nodes (RNs) for each shard, assigning the nodes to Storage Nodes according to the topology rules. While creating new RNs, the topology command might move existing RNs to different Storage Nodes, to best use available resources while complying with the topology rules.
2. The topology command distributes Partitions evenly among all shards. The partitions in over populated shards are moved to shards with the least number of partitions.

   You cannot specify which partitions the command moves.

Increase Replication Factor

You can use the `topology change-repfactor` command to increase the replication factor. Increasing the replication factor creates more copies of the data and improves read throughput and availability. More Replication Nodes are added to each shard so that it has the requisite number of nodes. The new Replication Nodes are populated from existing nodes in the shard.

Since every shard in a zone has the same replication factor, and a large number of shards, this command may require a significant number of new Storage Nodes to succeed.

For additional information on how to identify your primary replication factor and to understand the implications of the factor value, see Replication Factor.

The following example increases the replication factor of the store to 4. The administrator deploys a new Storage Node and adds it to the Storage Node pool. The admin then clones the existing topology and transforms it to use a new replication factor of 4.

```
kv-> plan deploy-sn -zn zn1 -host node08 -port 5000 -wait
Executed plan 12, waiting for completion...
Plan 12 ended successfully

kv-> pool join -name BostonPool -sn sn8
Added Storage Node(s) [sn8] to pool BostonPool

kv-> topology clone -current -name repTopo
Created repTopo

kv-> topology change-repfactor -name repTopo -pool BostonPool -rf 4 -zn zn1
Changed replication factor in repTopo

kv-> plan deploy-topology -name repTopo -wait
Executed plan 13, waiting for completion...
Plan 13 ended successfully
```

The `change-repfactor` command fails if either of the following occurs:

1. The new replication factor is less than or equal to the current replication factor.
2. The Storage Nodes specified by the storage node pool do not have enough capacity to host the required new Replication Nodes.

Balance a Non-Compliant Topology

Topologies must obey the rules described in Determining Your Store's Configuration. Changes to the physical characteristics of the store can cause the current store topology to violate those rules. For example, after performance tuning, you want to decrease the capacity of a Storage Node (SN). If that SN is already hosting the maximum permissible number of Replication Nodes, reducing its capacity will make the store non-compliant with the capacity rules. To decrease the capacity of an SN before using the `topology rebalance` command , use the `change-parameters` command for the storage node capacity. See plan change-parameters.

You can balance a non-compliant configuration using the `topology rebalance` command. This command requires a topology candidate name and a Storage Node pool name.

Before rebalancing your topology, use the `topology validate` command for any violations to the topology rules in your `repTopo` plan:

```
kv-> topology validate -name repTopo
```

Output:

```
Validation for topology candidate "repTopo":
4 warnings.
```

```
sn7 has 0 RepNodes and is under its capacity limit of 1
sn8 has 0 RepNodes and is under its capacity limit of 1
sn5 has 0 RepNodes and is under its capacity limit of 1
sn6 has 0 RepNodes and is under its capacity limit of 1
```

In this case, there are anticipated warnings, but you do not need improvements to the topology. However, if improvements are needed, then the `topology rebalance` command will move or create Replication Nodes, using the Storage Nodes in the BostonPool pool, to correct any violations. The command does not create additional shards under any circumstances. See Shard Capacity.

```
kv-> topology rebalance -name repTopo -pool BostonPool
```

Output:

```
Rebalanced: repTopo
```

If there are insufficient Storage Nodes, or if an insufficient storage directory size is allocated, the `topology rebalance` command may be unable to correct all violations. In that case, the command makes as much progress as possible, and warns of remaining issues.

Contracting a Topology

You can contract a topology by using the `topology contract` command. This command requires a topology candidate name and a Storage Node pool name. This command supports the removal of Storage Nodes and contracts the topology by relocating Replication Nodes, deleting shards, and migrating partitions.

> **Note:**
>
> Decreasing the replication factor is not currently supported. Also, Admin relocation is not supported. If an admin is present on a contracted Storage Node, the contraction operation will fail.

The following example contracts the topology by removing 3 Storage Nodes (sn2, sn5 and sn8). First, you clone the pool using the `pool clone command` and remove the Storage Nodes from the cloned pool using the `pool leave command`. Then, the topology is contracted and deployed using the contracted pool. Finally, the Storage Nodes can be removed using the `plan remove-sn` command. This command automatically stops Storage Nodes before removal.

```
# Clone the existing Storage Node pool as to be contractedPool
kv-> pool clone -name contractedPool -from AllStorageNodes
Cloned pool contractedPool
kv-> pool leave -name contractedPool -sn sn2
Removed Storage Node(s) [sn2] from pool contractedPool
kv-> pool leave -name contractedPool -sn sn5
Removed Storage Node(s) [sn5] from pool contractedPool
kv-> pool leave -name contractedPool -sn sn8
Removed Storage Node(s) [sn8] from pool contractedPool

# Generate a contracted candidate topology
```

```
kv-> topology clone -current -name contractedTopology
Created contractedTopology
kv-> topology contract -name contractedTopology -pool contractedPool
Contracted: contractedTopology

# Deploy the contracted candidate topology as the real topology.
kv-> plan deploy-topology -name contractedTopology -wait
Executed plan 16, waiting for completion...
Plan 16 ended successfully

# Remove to-be-deleted SNs
kv-> plan remove-sn -sn sn2 -wait
Executed plan 17, waiting for completion...
Plan 17 ended successfully
kv-> plan remove-sn -sn sn5 -wait
Executed plan 18, waiting for completion...
Plan 18 ended successfully
kv-> plan remove-sn -sn sn8 -wait
Executed plan 19, waiting for completion...
Plan 19 ended successfully
```

View the Topology Candidate

You can view details of the topology candidate or a deployed topology by using the `topology view` command. The command takes a topology name as an argument. With the topology view command, you can view all at once: the store name, number of partitions, shards, replication factor, host name and capacity in the specified topology.

For example:

```
kv-> topology view -name repTopo
```

Output:

```
store=mystore  numPartitions=300 sequence=315
  zn: id=zn1 name=Boston repFactor=4 type=PRIMARY

  sn=[sn1]  zn:[id=zn1 name=Boston] node01:5000 capacity=1
    [rg1-rn1]
  sn=[sn2]  zn:[id=zn1 name=Boston] node02:5000 capacity=1
    [rg1-rn2]
  sn=[sn3]  zn:[id=zn1 name=Boston] node03:5000 capacity=1
    [rg1-rn3]
  sn=[sn4]  zn:[id=zn1 name=Boston] node04:5000 capacity=1
    [rg1-rn4]
  sn=[sn5]  zn:[id=zn1 name=Boston] node05:5000 capacity=1
  sn=[sn6]  zn:[id=zn1 name=Boston] node06:5000 capacity=1
  sn=[sn7]  zn:[id=zn1 name=Boston] node07:5000 capacity=1
  sn=[sn8]  zn:[id=zn1 name=Boston] node08:5000 capacity=1

  shard=[rg1] num partitions=300
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn2
```

```
    [rg1-rn3] sn=sn3
    [rg1-rn4] sn=sn4
```

Validate the Topology Candidate

You can validate the topology candidate or a deployed topology by using the `topology validate` command. The topology validate command takes a topology name as an argument. If no topology is specified, the current topology is validated. Validation makes sure that the topology candidate obeys the topology rules described in Determining Your Store's Configuration. Validation generates "violations" and "notes".

Violations are issues that can cause problems and should be investigated.

Notes are informational and highlight configuration oddities that may be potential issues, but may be expected.

For example:

```
kv-> topology validate -name repTopo
Validation for topology candidate "repTopo":
4 warnings.
sn7 has 0 RepNodes and is under its capacity limit of 1
sn8 has 0 RepNodes and is under its capacity limit of 1
sn5 has 0 RepNodes and is under its capacity limit of 1
sn6 has 0 RepNodes and is under its capacity limit of 1
```

Preview the Topology Candidate

You should preview the changes that would be made for the specified topology candidate relative to a starting topology. You use the `topology preview` command to do this. This command takes the following arguments:

- **name**

  A string to identify the topology.

- **start <from topology>**

  If -start topology name is not specified, the current topology is used. This command should be used before deploying a new topology.

For example:

```
kv-> topology clone -current -name redTopo
Created redTopo
kv-> topology redistribute -name redTopo -pool BostonPool
Redistributed: redTopo
kv-> topology preview -name redTopo
Topology transformation from current deployed topology to redTopo:
Create 1 shard
Create 4 RNs
Migrate 150 partitions

shard rg2
  4 new RNs: rg2-rn1 rg2-rn2 rg2-rn3 rg2-rn4
  150 partition migrations
kv-> topology validate -name redTopo
```

```
Validation for topology candidate "redTopo":
No problems
```

Deploy the Topology Candidate

When your topology candidate is satisfactory, use the Admin service to generate and execute a plan that migrates the store to the new topology.

Deploy the topology candidate with the `plan deploy-topology` command, which takes a topology name as an argument.

While the plan is executing, you can monitor the plan's progress. You have several options:

- The plan can be interrupted then retried, or canceled.
- Other, limited plans may be executed while a transformation plan is in progress to deal with ongoing problems or failures.

By default, the `plan deploy-topology` command will not deploy a topology candidate if deployment would introduce new violations of the topology rules. You can override this behavior using the optional `-force` plan flag. Do not use the `-force` plan without consideration. Introducing a topology rule violation can have many adverse effects.

The next example shows the topology differences before and after plan deployment. The first `show topology` output lists four Storage Nodes running in Zone 1, with one shard (`rg1`) storing 300 partitions. Storage nodes `sn5 –sn8` are available.

After deploying the plan, the `show topology` output lists storage nodes `sn5 – sn8` as running. Another shard exists (`rg2`), and the partitions are split between the two shards, each with 150 partitions.

```
kv-> show topology
store=mystore  numPartitions=300 sequence=315
  zn: id=zn1 name=Boston repFactor=4 type=PRIMARY

  sn=[sn1]  zn=[id=zn1 name=Boston] node01:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
           No performance info available
  sn=[sn2]  zn=[id=zn1 name=Boston] node02:5000 capacity=1 RUNNING
    [rg1-rn2] RUNNING
           No performance info available
  sn=[sn3]  zn=[id=zn1 name=Boston] node03:5000 capacity=1 RUNNING
    [rg1-rn3] RUNNING
           No performance info available
  sn=[sn4]  zn=[id=zn1 name=Boston] node04:5000 capacity=1 RUNNING
    [rg1-rn4] RUNNING
           No performance info available
  sn=[sn5]  zn=[id=zn1 name=Boston] node05:5000 capacity=1
  sn=[sn6]  zn=[id=zn1 name=Boston] node06:5000 capacity=1
  sn=[sn7]  zn=[id=zn1 name=Boston] node07:5000 capacity=1
  sn=[sn8]  zn=[id=zn1 name=Boston] node08:5000 capacity=1

  shard=[rg1] num partitions=300
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn2
    [rg1-rn3] sn=sn3
    [rg1-rn4] sn=sn4
```

```
kv-> plan deploy-topology -name redTopo -wait
Executed plan 14, waiting for completion...
Plan 14 ended successfully

kv-> show topology
store=mystore  numPartitions=300 sequence=470
  zn: id=zn1 name=Boston repFactor=4 type=PRIMARY

  sn=[sn1]  zn:[id=zn1 name=Boston] node01:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
          No performance info available
  sn=[sn2]  zn:[id=zn1 name=Boston] node02:5000 capacity=1 RUNNING
    [rg1-rn2] RUNNING
          No performance info available
  sn=[sn3]  zn:[id=zn1 name=Boston] node03:5000 capacity=1 RUNNING
    [rg1-rn3] RUNNING
          No performance info available
  sn=[sn4]  zn:[id=zn1 name=Boston] node04:5000 capacity=1 RUNNING
    [rg1-rn4] RUNNING
          No performance info available
  sn=[sn5]  zn:[id=zn1 name=Boston] node05:5000 capacity=1 RUNNING
    [rg2-rn1] RUNNING
          No performance info available
  sn=[sn6]  zn:[id=zn1 name=Boston] node06:5000 capacity=1 RUNNING
    [rg2-rn2] RUNNING
          No performance info available
  sn=[sn7]  zn:[id=zn1 name=Boston] node07:5000 capacity=1 RUNNING
    [rg2-rn3] RUNNING
          No performance info available
  sn=[sn8]  zn:[id=zn1 name=Boston] node08:5000 capacity=1 RUNNING
    [rg2-rn4] RUNNING
          No performance info available

  shard=[rg1] num partitions=150
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn2
    [rg1-rn3] sn=sn3
    [rg1-rn4] sn=sn4
  shard=[rg2] num partitions=150
    [rg2-rn1] sn=sn5
    [rg2-rn2] sn=sn6
    [rg2-rn3] sn=sn7
    [rg2-rn4] sn=sn8
```

Verify the Store's Current Topology

You can verify the store's current topology by using the `verify` command. The verify command checks the current, deployed topology to make sure it adheres to the topology rules described in Determining Your Store's Configuration.

You should examine the new topology and decide if it is satisfactory. If it is not, you can apply more transformations, or start over with different parameters.

For example:

```
kv-> verify configuration
```

Output:

```
Verify: starting verification of store mystore based upon
    topology sequence #470
300 partitions and 8 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
See localhost:KVROOT/mystore/log/mystore_{0..N}.log for progress messages
Verify: Shard Status: healthy:2 writable-degraded:0 read-only:0 offline:0
Verify: Admin Status: healthy
Verify: Zone [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    RN Status: online:8 offline:0 maxDelayMillis:0 maxCatchupTimeSecs:0
Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on node01:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin1]          Status: RUNNING,MASTER
Verify:         Rep Node [rg1-rn1]      Status: RUNNING,MASTER ...
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on node02:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg1-rn2]      Status: RUNNING,REPLICA ...
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg1-rn3]      Status: RUNNING,REPLICA ...
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg1-rn4]      Status: RUNNING,REPLICA ...
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn1]      Status: RUNNING,MASTER ...
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
```

```
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn2]      Status: RUNNING,REPLICA ...
Verify: == checking storage node sn7 ==
Verify: Storage Node [sn7] on node07:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn3]      Status: RUNNING,REPLICA ...
Verify: == checking storage node sn8 ==
Verify: Storage Node [sn8] on node08:5000
    Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING
    Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn4]      Status: RUNNING,REPLICA ...
Verification complete, no violations.
```

Deploying an Arbiter Node Enabled Topology

An `Arbiter Node` is a service that supports write availability when the store replication factor is two and a single Replication Node becomes unavailable. The role of an `Arbiter Node` is to participate in elections and respond to acknowledge requests if one of the two Replication Nodes in a shard becomes unavailable.

Arbiter Nodes are automatically configured in a topology if the store replication factor is two and a primary zone is configured to host `Arbiter Nodes`.

For example, suppose a store consists of a primary zone, "Manhattan" with two Storage Nodes deployed in the same shard. In this example, an Arbiter Node is deployed in the third Storage Node (capacity = 0) in order to provide write availability even if one of the two Replication Nodes in the shard becomes unavailable.

> **Note:**
>
> Durability.ReplicaAckPolicy must be set to SIMPLE_MAJORITY, so that writes can succeed if a Replication Node becomes unavailable in a shard. For more information on `ReplicaAckPolicy`, see this Javadoc page.

1. Create, start, and configure the store. Note that a Storage Node with capacity equal to zero is deployed, which will host the Arbiter Node.
   - Create the store:

     ```
     java -Xmx64m -Xms64m \
     -jar kv/lib/kvstore.jar makebootconfig \
     -root KVROOT \
     -host node01 \
     -port 8000 \
     ```

```
-harange 8010,8020 \
-capacity 1


java -Xmx64m -Xms64m \
-jar kv/lib/kvstore.jar makebootconfig \
-root KVROOT \
-host node02 \
-port 9000 \
-harange 9010,9020 \
-capacity 1


java -Xmx64m -Xms64m \
-jar kv/lib/kvstore.jar makebootconfig \
-root KVROOT \
-host node03 \
-port 10000 \
-harange 1000,10020 \
-capacity 0 \
```

- Create and copy the security directories:

```
java -Xmx64m -Xms64m \
-jar kv/lib/kvstore.jar
securityconfig \
config create -root KVROOT -kspwd password
Created files
KVROOT/security/security.xml
KVROOT/security/store.keys
KVROOT/security/store.trust
KVROOT/security/client.trust
KVROOT/security/client.security
KVROOT/security/store.passwd (Generated in CE version)
KVROOT/security/store.wallet/cwallet.sso (Generated in EE version)

Created
scp -r KVROOT/security node02:KVROOT/
scp -r KVROOT/security node03:KVROOT/
```

- Start the store by running the following command on each Storage Node:

```
java -Xmx64m -Xms64m -jar KVHOME/lib/kvstore.jar \
start -root KVROOT &
```

2. Load the following script `conf.txt` to deploy the zone, admin and Storage Nodes. To host an Arbiter Node, the zone must be primary and should have the `-arbiters` flag set.

```
ssh node01
java -Xmx64m -Xms64m -jar KVHOME/lib/kvstore.jar runadmin \
-port 8000 -host node01 load -file conf.txt \
-security KVROOT/security/client.security
```

The file, `conf.txt`, would then contain content like this:

```
### Begin Script ###
plan deploy-zone -name "Manhattan" -type primary -arbiters -rf 2 -wait
plan deploy-sn -zn zn1 -host node01 -port 8000 -wait
pool create -name SNs
pool join -name SNs -sn sn1
plan deploy-admin -sn sn1 -port 8001 -wait
plan deploy-sn -zn zn1 -host node02 -port 9000 -wait
pool join -name SNs -sn sn2
plan deploy-sn -zn zn1 -host node03 -port 10000 -wait
pool join -name SNs -sn sn3
### End Script ###
```

3. Create a topology, preview it, and then deploy it:

```
kv-> topology create -name arbTopo -pool SNs -partitions 300
```

Output:

```
Created: arbTopo
```

```
kv-> topology preview -name arbTopo
```

Output:

```
Topology transformation from current deployed topology to arbTopo:
Create 1 shard
Create 2 RNs
Create 300 partitions
Create 1 AN

shard rg1
  2 new RNs : rg1-rn1 rg1-rn2
  1 new AN : rg1-an1
  300 new partitions
```

```
kv-> plan deploy-topology -name arbTopo -wait
```

Output:

```
Executed plan 6, waiting for completion...
Plan 6 ended successfully
```

4. Verify that the Arbiter Node is running.

```
kv-> verify configuration
```

Output:

```
Verify: starting verification of store mystore
based upon topology sequence #308
300 partitions and 3 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
See node01:KVROOT/mystore/log/mystore_{0..N}.log
for progress messages
Verify: Shard Status: healthy:1 writable-degraded:0
                                            read-only:0 offline:0
Verify: Admin Status: healthy
Verify: Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false]
RN Status: online:2 offline:0 maxDelayMillis:6 maxCatchupTimeSecs:0
Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on node01:8000
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin1]          Status: RUNNING,MASTER
Verify:         Rep Node [rg1-rn1]
Status: RUNNING,MASTER sequenceNumber:635 haPort:8011 available storage
size:11 GB
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on node02:9000
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg1-rn2]
Status: RUNNING,REPLICA
sequenceNumber:635 haPort:9010 available storage size:12 GB delayMillis:6
catchupTimeSecs:0
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:10000
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Arb Node [rg1-an1]
Status: RUNNING,REPLICA sequenceNumber:0 haPort:node03:10010
...
```

5. Now suppose node02 is unreachable. Verify this by using `verify configuration`:

```
kv-> verify configuration
```

Output:

```
Verify: starting verification of store mystore
based upon topology sequence #308
300 partitions and 3 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
See node01:KVROOT/mystore/log/mystore_{0..N}.log
```

```
for progress messages
Verify: Shard Status: healthy:0 writable-degraded:1
                                                read-only:0 offline:0
Verify: Admin Status: healthy
Verify:
      Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false]
RN Status: online:1 offline:1
Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on node01:8000
Zone:
    [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false]
 Status: RUNNING
 Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin1]          Status: RUNNING,MASTER
Verify:         Rep Node [rg1-rn1]
Status: RUNNING,MASTER sequenceNumber:901 haPort:8011 available storage
size:12 GB
Verify: == checking storage node sn2 ==
Verify:         sn2: ping() failed for sn2 : Unable to connect to
the storage node agent at host node02, port 9000, which may not be
running; nested exception is:
        java.rmi.ConnectException: Connection refused to
        host: node02; nested exception is:
        java.net.ConnectException: Connection refused
Verify: Storage Node [sn2] on node02:9000
Zone:
    [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false] UNREACHABLE
Verify:         rg1-rn2: ping() failed for rg1-rn2 : Unable to connect
to the storage node agent at host node02, port 9000, which may not
be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
        nested exception is:
        java.net.ConnectException: Connection refused
Verify:         Rep Node [rg1-rn2]      Status: UNREACHABLE
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:10000
 Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=true
masterAffinity=false]
 Status: RUNNING
Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Arb Node [rg1-an1]
Status: RUNNING,REPLICA sequenceNumber:901 haPort:node03:10010 available
storage size:16 GB delayMillis:? catchupTimeSecs:?
Verification complete, 3 violations, 0 notes found.
Verification violation: [rg1-rn2]
ping() failed for rg1-rn2 : Unable to connect to the storage node
agent at host node02, port 9000, which may not be running;
nested exception is:
        java.rmi.ConnectException: Connection refused to
        host: node02; nested exception is:
        java.net.ConnectException: Connection refused
Verification violation: [sn2]   ping() failed for sn2 : Unable to
connect to the storage node agent at host node02, port 9000, which
```

```
may not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
nested exception is:
        java.net.ConnectException: Connection refused
...
```

In this case the Arbiter Node supports write availability so you can still perform write operations while node02 is repaired or replaced. Once node02 is restored, any written data will be migrated.

6. Test that you can still write to the store with the help of the Arbiter Node. For example, run the script file `test.kvsql` (see below for test.kvsql) using the Oracle NoSQL Database Shell utility (see below example). To do this, use the `load` command in the Query Shell:

```
> java -jar KVHOME/lib/sql.jar -helper-hosts node01:8000 \
-store mystore -security USER/security/admin.security


kvsql-> load -file ./test.kvsql
```

Output:

```
Statement completed successfully.
Statement completed successfully.
Loaded 3 rows to users.
```

> **Note:**
>
> For the Enterprise Edition (EE) installation, make sure the kvstore-ee.jar is added in the classpath.

The following commands are collected in `test.kvsql`:

```
### Begin Script ###
load -file test.ddl
import -table users -file users.json
### End Script ###
```

Where the file `test.ddl` would contain content like this:

```
DROP TABLE IF EXISTS users;
CREATE TABLE users(id INTEGER, firstname STRING, lastname STRING,
age INTEGER, primary key (id));
```

And the file `users.json` would contain content like this:

```
{"id":1,"firstname":"Dean","lastname":"Morrison","age":51}
{"id":2,"firstname":"Idona","lastname":"Roman","age":36}
{"id":3,"firstname":"Bruno","lastname":"Nunez","age":49}
```

Backup and Recovery

Topics:

Backing Up the Store

To make backups of your KVStore, use the CLI `snapshot` command to copy nodes in the store. To maintain consistency, no topology changes should be in process when you create a snapshot. Restoring a snapshot relies on the system configuration having exactly the same topology that was in effect when you created the snapshot.

When you create a snapshot, it is stored in a subdirectory of the SN. But these snapshots don't become persistent backups unless they are copied to separate storage. It is your responsibility to copy each of the snapshots to another location, preferably on a different machine, for data safety.

Due to the distributed nature and scale of Oracle NoSQL Database, it is unlikely that a single machine has the resources to contain snapshots for the entire store. This document does not address where and how you should store your snapshots.

Taking a Snapshot

> **Note:**
>
> To avoid any snapshot from being inconsistent or unusable, do not take snapshots while any configuration (topological) changes are in process. At the time of the snapshot, use the `ping` command and save the output information that identifies Masters for later use during a load or restore. For more information, see Managing Snapshots.

To create a snapshot from the Admin CLI, use the `snapshot create` command:

```
kv-> snapshot create -name <snapshot name>
```

A snapshot consists of a set of hard links to data files in the current topology, specifically, all partition records within the same shard. The snapshot does not include partitions in independent shards. To minimize any potential inconsistencies, the snapshot utility performs its operations in parallel as much as possible.

To create a snapshot with a name of your choice, use `snapshot create -name <name>`.

```
kv-> snapshot create -name Thursday
```

Output:

```
Created snapshot named 110915-153514-Thursday on all 3 nodes
Successfully backup configurations on sn1, sn2, sn3
```

Snapshot Activities

Creating a snapshot of the Oracle NoSQL Database store performs these activities:

- Backs up the data files
- Backs up the configuration and environment files required for restore activities

To complete a full set of snapshot files, the `snapshot` command attempts to backup the storage node data files, configuration files, and adds other required files. Following is a description of the various files and directories the `snapshot` command creates or copies:

Creates a `snapshots` directory as a peer to the `env` directory. Each `snapshots` directory contains one subdirectory for each snapshot you create. That subdirectory contains the `*.jdb` files.
 	
The snapshot name subdirectory with a *date-time-name* prefix has the name you supply with the `-name` parameter. The *date-time* prefix consists of a 6-digit, year, month, day value in YYMMDD format, and a 6-digit hour, minute, seconds timestamp as HHMMSS. The date and time values are separated from each other with a dash (-), and include a dash (-) suffix before the snapshot name.
	`kvroot/mystore/sn1/rg1-rn1/snapshots/`
	`170417-104506-snapshotName/*.jdb`
	`kvroot/mystore/sn1/rg1-rn1/env/*.jdb`
	`kvroot/mystore/sn1/admin1/snapshots/`
	`170417-104506-snapshotName/*.jdb`
	`kvroot/mystore/sn1/admin1/env/*.jdb`

Copies the root `config.xml` file to the *date-time-name* directory.	`kvroot/config.xml >`
	`kvroot/snapshots/170417-104506-snapshotName/`
	`config.xml`

Creates a status file in the *date-time-name* subdirectory. The contents of this file, `snapshot.stat`, indicate whether creating a snapshot was successful. When you restore to a snapshot, the procedure first validates the status file contents, continuing only if the	`kvroot/snapshots/170417-104506-snapshotName/` `snapshot.stat`

file contains the string `SNAPSHOT=COMPLETED`.	
Creates a lock file in the *date-time-name* subdirectory. The lock file, `snapshot.lck`, is used to avoid concurrent modifications from different SN Admins within the same root directory.	`kvroot/snapshots/170417-104506-snapshotName/snapshot.lck`
Creates a subdirectory of the *date-time-name* subdirectory, `security`. This subdirectory has copies of security information copied from `kvroot/security`.	`kvroot/snapshots/170417-104506-snapshotName/security`
Copies the root security policy from `kvroot/security.policy`, to the *date-time-name* subdirectory.	`kvroot/snapshots/170417-104506-snapshotName/security.policy`
Copies the store security policy to *date-time-name* subdirectory, into another subdirectory, `mystore`.	`kvroot/snapshots/170417-104506-snapshotName/mystore/security.policy`
Copies the Storage Node configuration file, `config.xml`, from `kvroot/mystore/sn1/config.xml` to a corresponding SN subdirectory in the *date-time-name* directory.	`kvroot/snapshots/170417-104506-snapshotName/mystore/sn1/config.xml`

Copying a Snapshot

Keeping a snapshot in place for a short time so that it can be used to rollback the store after an upgrade is a reasonable thing to do. In such a scenario, it might be sufficient to delete the snapshot without copying it if the upgrade can be verified relatively quickly and the snapshot is no longer needed.

For ensuring data safety during disk or hardware failures, it is recommended that you convert these snapshots into persistent backups. Otherwise, if the machine suffers a disk or other hardware failure, or if store files are deleted or overwritten, the snapshot will be lost along with the live data for the store maintained on that machine.

To convert the snapshot into a persistent backup, the snapshot needs to be copied to another location on a different machine. Later, you can use the persistent backup to restore the store after a disk or hardware failure.

Deleting a Snapshot

To remove an existing snapshot, use `snapshot remove <name>`.

```
kv-> snapshot remove -name 110915-153514-Thursday
Removed snapshot 110915-153514-Thursday
```

To remove all snapshots currently stored in the store, use `snapshot remove -all`.

```
kv-> snapshot create -name Thursday
Created snapshot named 110915-153700-Thursday on all 3 nodes
kv-> snapshot create -name later
Created snapshot named 110915-153710-later on all 3 nodes
kv-> snapshot remove -all
Removed all snapshots
```

Managing Snapshots

When you create a snapshot, the utility collects data from every Replication Node in the system, including Masters and replicas. If the operation does not succeed for any one node in a shard, the entire snapshot fails.

When you are preparing to take the snapshot, you can use the `ping` command to identify which nodes are currently running as the Master. Each shard has a Master, identified by the `MASTER` keyword. For example, in the sample output, replication node `rg1-rn1`, running on Storage Node `sn1`, is the current Master:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar ping -port 5000 -host node01 \
-security USER/security/admin/security
```

Output:

```
Pinging components of store mystore based upon topology sequence #316
300 partitions and 3 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
Shard Status: healthy:3 writable-degraded:0 read-only:0 offline:0
Admin Status: healthy
Zone [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
RN Status: online:9 offline:0 maxDelayMillis:1 maxCatchupTimeSecs:0
Storage Node [sn1] on node01:5000
   Zone: [name=Boston id=zn1 type=PRIMARY]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
      Admin [admin1]          Status: RUNNING,MASTER
      Rep Node [rg1-rn1]      Status: RUNNING,REPLICA
        sequenceNumber:231 haPort:5011 available storage size:14 GB
delayMillis:1 catchupTimeSecs:0
      Rep Node [rg2-rn1]      Status: RUNNING,REPLICA
        sequenceNumber:231 haPort:5012 available storage size:12 GB
delayMillis:1 catchupTimeSecs:0
      Rep Node [rg3-rn1]      Status: RUNNING,MASTER
        sequenceNumber:227 haPort:5013 available storage size:13 GB
Storage Node [sn2] on node02:6000
   Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
      Rep Node [rg1-rn2]      Status: RUNNING,MASTER
        sequenceNumber:231 haPort:6010 available storage size:15 GB
```

```
      Rep Node [rg2-rn2]      Status: RUNNING,REPLICA
        sequenceNumber:231 haPort:6011 available storage size:18 GB
delayMillis:1 catchupTimeSecs:0
      Rep Node [rg3-rn2]      Status: RUNNING,REPLICA
        sequenceNumber:227 haPort:6012 available storage size:12 GB
delayMillis:1 catchupTimeSecs:0
Storage Node [sn3] on node03:7000
   Zone: [name=Boston id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
      Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
        sequenceNumber:231 haPort:7010 available storage size:11 GB
delayMillis:1 catchupTimeSecs:0
      Rep Node [rg2-rn3]      Status: RUNNING,MASTER
        sequenceNumber:231 haPort:7011 available storage size:11 GB
      Rep Node [rg3-rn3]      Status: RUNNING,REPLICA
        sequenceNumber:227 haPort:7012 available storage size:10 GB
delayMillis:1 catchupTimeSecs:0
```

You should save the above information and associate it with the respective snapshot, for later use during a load or restore. If you decide to create an off-store copy of the snapshot, you should copy the snapshot data for only one of the nodes in each shard. If possible, copy the snapshot data taken from the node that was serving as the Master at the time the snapshot was taken.

> **Note:**
>
> Snapshots include the admin database, which may be required if the store needs to be restored from this snapshot.

Snapshot data for the local Storage Node is stored in a directory inside of the `KVROOT` directory. For each Storage Node in the store, you have a directory named:

```
KVROOT/<store>/<SN>/<resource>/snapshots/<snapshot_name>/files
```

where:

- <store> is the name of the store.
- <SN> is the name of the Storage Node.
- <resource> is the name of the resource running on the Storage Node. Typically, this is the name of a replication node.
- <snapshot_name> is the name of the snapshot.

Snapshot data consists of a number of files. For example:

```
 > ls /var/kvroot/mystore/sn1/rg1-rn1/snapshots/110915-153514-Thursday
00000000.jdb 00000002.jdb 00000004.jdb 00000006.jdb
00000001.jdb 00000003.jdb 00000005.jdb 00000007.jdb
```

> **Note:**
>
> To preserve storage, purge obsolete snapshots on a periodic basis.

Impact of Erasure with snapshots

Snapshot based backups create hard-links to original files. Until these backups are copied to their target location (complete off-store copy work) and the corresponding hard-links are removed (performing a snapshot remove command), erasure doesn't process obsolete data in those files. Erasure ignores files with hard-links to them.

Avoiding Disk Usage Violation

The storage engine does not consider the data consumed by snapshots when it collects information about disk space usage. Initially, the files in the snapshot are considered to be part of the live data of the store. Over time, though, as older files are cleaned and deleted, their presence in the snapshot causes the files to be retained and use the disk space that is not taken into account by the storage engine. It could cause a disk usage violation, in which case further writes to the store are disabled. To avoid this problem, users should delete snapshot files at regular intervals.

Recovering the Store

There are two ways to recover your store from a previously created snapshot:

1. Use a snapshot to create a store with any topology with the `Load` utility.
2. Restore a snapshot using the exact topology you were using when you created the snapshot.

This section describes and explains both ways to recover your store.

> **Note:**
>
> If you need to recover due to a hardware problem, such as a failed Storage Node, that qualifies as a topology change, so you must use the `Load` utility to recover. For information about replacing a failed Storage Node, see Replacing a Failed Storage Node.

Using the Load Program

You can use the `oracle.kv.util.Load` program to restore a store from a previously created snapshot. You can run this program directly, or you can access it using `kvstore.jar`, as shown in the examples in this section.

Using this tool lets you restore to any topology, not just the topology in effect when you created the snapshot.

This Load mechanism works by iterating through all records in a snapshot, putting each record into a target store as it proceeds through the snapshot. Use Load to populate a new, empty store. Do not use this with an existing store. Load only writes records if they do not already exist.

Note that to recover the store, you must load records from snapshot data captured for each shard in the store. For best results, you should load records using snapshot data captured from the replication nodes that were running as Master at the time the snapshot was taken. (If you have three shards in your store, then there are three Masters at any given time, and so you need to load data from three sets of snapshot data). To identify the Master, use ping at the time the snapshot was taken.

You should use snapshot data taken at the same point in time; do not, for example, use snapshot data for shard 1 that was taken on Monday, and snapshot data for shard 2 that was taken on Wednesday. Such actions will restore your store to an inconsistent state.

Also, the Load mechanism can only process data at the speed necessary to insert data into a new store. Because you probably have multiple shards in your store, you should restore your store from data taken from each shard. To do this, run multiple instances of the `Load` program in parallel, having each instance operate on data from different replication nodes.

The program's usage to load admin metadata is:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar load \
-store <storeName> -host <hostname> port <port>  \
-load-admin \
-source <admin-backup-dir> \
[-force] [-username <user>] \
[-security <security-file-path>]
```

The program's usage to load store data is:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar load [-verbose] \
-store <storeName> -host <hostname> \
-port <port> -source <shard-backup-dir> \
[, <shard-backup-dir>]* \
[-checkpoint <checkpoint-files-directory>] \
[-username <user>] [-security <security-file-path>]
```

where:

- `-load-admin` Loads the store metadata from the snapshot to the new store. In this case the `-source` directory must point to the environment directory of the admin node from the snapshot. The store must not be available for use by users at the time of this operation.

  > **Note:**
  >
  > This option should not be used on a store unless that store is being restored from scratch. If `-force` is specified in conjunction with `-load-admin`, any existing metadata in the store, including tables and security metadata, will be overwritten. For more information, see Load Program and Metadata.

- `-host <hostname>` identifies the host name of a node in your store.
- `-port <port>` identifies the registry port in use by the store's node.
- `-security <security-file-path>` identifies the security file used to specify properties for login.

- `-source <admin-backup-dir> | <shard-backup-dir> [,<shard-backup-dir>]* admin-backup-dir` specifies the admin snapshot directory containing the contents of the admin metadata that is to be loaded into the store.

  `Shard-backup-dir` specifies the backup directories that represent the contents of snapshots created using the snapshot commands described at Taking a Snapshot.
- `-store <storeName>` identifies the name of the store.
- `-username <user>` identifies the name of the user to login to the secured store.

Any administrative user who wants to restore records into a secure data store needs to have the required privileges. Before restoring data using the load command, run the following command to grant the user with the required privileges as shown below.

```
GRANT writesystable,readwrite TO USER <admin_user>
```

See Grant Roles and Privileges for more details.

> **Note:**
>
> If the user performing the restore does not have sufficient privilege, the Load program fails with the below exception.
>
> ```
> Load operation failed with exception: oracle.kv.FaultException:
> Insufficient access rights granted (24.1.0) on [YYYY-MM-DD HH:MI:SS
> UTC]
> Fault class name: oracle.kv.UnauthorizedException
> ```

For example, if a snapshot exists in `/var/backups/snapshots/110915-153828-later`, and a new store named "mystore" on host "host1" using registry port 5000, run the `Load` program on the host that has the `/var/backups/snapshots` directory:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar load \
-source /var/backups/snapshots/110915-153514-Thursday -store mystore \
-host host1 -port 5000 -security KVROOT/security/client.security
```

> **Note:**
>
> Before you load records into the new store, make sure that the store is deployed. For more information, see Configuring a single region data store.

Load Program and Metadata

You can use the Load program to restore a store with metadata (tables, security) from a previously created snapshot.

The following steps describe how to load from a snapshot with metadata to a newly created store:

1. Create, start and configure the new store (target). Do not make the store accessible to applications yet.

   - Create the new store:

     ```
     java -Xmx64m -Xms64m \
     -jar KVHOME/lib/kvstore.jar makebootconfig \
     -root KVROOT \
     -host NewHost -port 8000 \
     -harange 8010,8020 \
     -capacity 1
     ```

   - Create security directory:

     ```
     java -Xmx64m -Xms64m \
     -jar kv/lib/kvstore.jar securityconfig \
     config create
     -root KVROOT -kspwd password
     Created files
     KVROOT/security/security.xml
     KVROOT/security/store.keys
     KVROOT/security/store.trust
     KVROOT/security/client.trust
     KVROOT/security/client.security
     KVROOT/security/store.passwd (Generated in CE version)
     KVROOT/security/store.wallet/cwallet.sso (Generated in EE version)

     Created
     ```

   - Start the new store:

     ```
     java -Xmx64m -Xms64m \
     -jar KVHOME/lib/kvstore.jar start \
     -root KVROOT &
     ```

   - Configure the new store:

     ```
     java -Xmx64m -Xms64m \
     -jar KVHOME/lib/kvstore.jar runadmin \
     -port 8000 -host NewHost \
     -security KVROOT/security/client.security
     kv-> configure -name NewStore
     Store configured: NewStore
     ```

   > **Note:**
   >
   > Loading security metadata requires the names of the source store and the target store to be the same, otherwise the security metadata cannot be used later.

2. Locate the snapshot directories for the source store. There should be one for the admin nodes plus one for each shard. For example in a 3x3 store there should be 4 snapshot directories used for the load. The load program must have direct file-based access to each

snapshot directory loaded. In this case, the snapshot source directory is in `/var/kvroot/mystore/sn1/admin1/snapshots/110915-153514-Thursday`.

3. Load the store metadata using the `-load-admin` option. Host, port, and store refer to the target store. In this case the `-source` directory must point to the environment directory of the admin node from the snapshot.

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar load \
-source \
/var/kvroot/mystore/sn1/admin1/snapshots/110915-153514-Thursday \
-store NewStore -host NewHost -port 8000 \
-load-admin \
-security KVROOT/security/client.security
```

> **Note:**
>
> This command can be run more than once if something goes wrong, as long as the store is not accessible to applications.

4. Deploy the store. For more information, see Configuring a single region data store.
5. Once the topology is deployed, load the shard data for each shard. To do this, run the Load program in parallel, with each instance operating on data captured from different replication nodes. For example, suppose there is a snapshot of `OldStore` in `var/backups/snapshots/140827-144141-back`.

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar load \
-source var/backups/snapshots/140827-144141-back -store NewStore \
-host NewHost -port 8000 \
-security KVROOT/security/client.security
```

> **Note:**
>
> This step may take a long time or might need to be restarted. In order to significantly reduce retry time, the use of a status file is recommended.
>
> If the previous store has been configured with username and password, the program will prompt for username and password here.

6. The store is now ready for applications.

Restoring Directly from a Snapshot

You can restore a store directly from a snapshot. This mechanism is faster than using the `Load` program. However, you can restore from a snapshot only to the *exact same* topology as was in use when the snapshot was taken. This means that all ports and host names or IP addresses (depending on your configuration) must be exactly the same as when you took the snapshot.

To restore from a snapshot, complete these steps:

1. Run this command on each of the Storage Nodes (SNs) to shut down the store:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar stop -root $KVROOT
```

2. When each SN is stopped, run this command on each SN in the store to restore to the backup (using `-update-config true`):

```
> java -jar KVHOME/lib/kvstore.jar start -root /var/kvroot \
-restore-from-snapshot 170417-104506-mySnapshot -update-config true
```

3. To restore to the backup, but not override the existing configurations, run this command on each SN (with `-update-config false`):

```
> java -jar KVHOME/lib/kvstore.jar start -root /var/kvroot \
-restore-from-snapshot 170417-104506-mySnapshot -update-config false
```

The *170417–104506–mySnapshot* value represents the directory name of the snapshot to restore.

> **Note:**
>
> This procedure recovers the store to the time you created the snapshot. If your store was active after snapshot creation, all modifications made since the last snapshot are lost.

Use case to Demonstrate Backup and Restoration of Store

This is a use case to demonstrate backup and restoration of store.

Consider that you have a secure data store. In this use case, we will show you how to backup this store and then restore it using the `load` program.

When you restore a data store, both the metadata and data are restored. To demonstrate this, we will create a privileged user. This is required to write data to a secure store. We will then create a simple table and insert some data into it. We will backup the store by creating a snapshot. While restoring the store, you can then see how this metadata, table and data are successfully restored.

Backing Up the Store

This section describes the example use case.

In this example, let us consider a secure KVStore that is deployed on 2 storage nodes with capacity 3 and replication factor 2. The storage nodes are on the host `phoenix581601` at ports 8000 and 7000. Let us call the storage nodes as storage node 1 (sn1) and storage node 2 (sn2).

You can check the topology of this store by pinging it using the runadmin utility. This will help you identify the name of the store, the replication masters and the running status of the store.

```
kv->ping
```

Output:

```
Pinging components of store mystore based upon topology sequence #112
100 partitions and 2 storage nodes
Time: 2025-05-14 07:21:38 UTC   Version: 25.1.1
Shard Status: healthy: 3 writable-degraded: 0 read-only: 0 offline: 0 total: 3
Admin Status: healthy
Zone [name=Cloud id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]   RN Status: online: 6 read-only: 0 offline: 0
maxDelayMillis: 0 maxCatchupTimeSecs: 0
Storage Node [sn1] on phoenix581601: 8000    Zone: [name=Cloud id=zn1
type=PRIMARY allowArbiters=false masterAffinity=false]    Status: RUNNING
Ver: 25.1.1 2025-01-09 18:45:42 UTC  Build id: f6749f469c66 Edition:
Enterprise    isMasterBalanced: true        serviceStartTime: 2025-05-14
06:04:25 UTC
              Admin [admin1]                            Status:
RUNNING,MASTER             serviceStartTime: 2025-05-14 06:04:52 UTC
stateChangeTime: 2025-05-14 06:04:51 UTC            availableStorageSize: 2 GB
              Rep Node [rg1-rn1]         Status: RUNNING,REPLICA
sequenceNumber: 1,156 haPort: 8011 availableStorageSize: 323 GB storageType:
HD              serviceStartTime: 2025-05-14 06:44:04 UTC
stateChangeTime: 2025-05-14 06:44:46 UTC delayMillis: 0 catchupTimeSecs: 0
              Rep Node [rg2-rn1]         Status: RUNNING,MASTER
sequenceNumber: 1,456 haPort: 8012 availableStorageSize: 323 GB storageType:
HD              serviceStartTime: 2025-05-14 06:44:34 UTC
stateChangeTime: 2025-05-14 06:44:37 UTC
              Rep Node [rg3-rn1]         Status: RUNNING,MASTER
sequenceNumber: 1,392 haPort: 8013 availableStorageSize: 323 GB storageType:
HD              serviceStartTime: 2025-05-14 06:45:05 UTC
stateChangeTime: 2025-05-14 06:45:08 UTC
Storage Node [sn2] on phoenix581601: 7000    Zone: [name=Cloud id=zn1
type=PRIMARY allowArbiters=false masterAffinity=false]    Status: RUNNING
Ver: 25.1.1 2025-01-09 18:45:42 UTC  Build id: f6749f469c66 Edition:
Enterprise    isMasterBalanced: true        serviceStartTime: 2025-05-14
06:28:18 UTC
              Admin [admin2]                            Status:
RUNNING,REPLICA              serviceStartTime: 2025-05-14 06:33:02
UTC         stateChangeTime: 2025-05-14 06:33:02 UTC
availableStorageSize: 2 GB
              Rep Node [rg1-rn2]         Status: RUNNING,MASTER
sequenceNumber: 1,156 haPort: 7011 availableStorageSize: 323 GB storageType:
HD              serviceStartTime: 2025-05-14 06:44:17 UTC
stateChangeTime: 2025-05-14 06:44:45 UTC
              Rep Node [rg2-rn2]         Status: RUNNING,REPLICA
sequenceNumber: 1,456 haPort: 7012 availableStorageSize: 323 GB storageType:
HD              serviceStartTime: 2025-05-14 06:44:49 UTC
stateChangeTime: 2025-05-14 06:44:53 UTC delayMillis: 0 catchupTimeSecs: 0
              Rep Node [rg3-rn2]         Status: RUNNING,REPLICA
sequenceNumber: 1,392 haPort: 7013 availableStorageSize: 323 GB storageType:
```

```
HD              serviceStartTime: 2025-05-14 06:45:17 UTC
stateChangeTime: 2025-05-14 06:45:23 UTC delayMillis: 0 catchupTimeSecs: 0
```

You can see that the name of the store is `mystore` and it has 3 shards. The replication masters are `rg1-rn2`, `rg2-rn1` and `rg3-rn1`. The store is in `Running` state.

Write Data to Store

In this section, we will write some sample data to the secure store. To do this, we will create a user and grant the required privileges.

Create a user and grant `readwrite` privileges to the user. For more details, see User Management and Managing Roles, Privileges and Users.

Open the SQL shell. For more details, see Starting the SQL shell. In the command below, `kvroot/security` is the security directory. The security directory contains all the security related files that were automatically generated by the securityconfig tool while creating the secure store using the `makebootconfig` command. For more details, see Configuring security in a data store.

```
java -Xmx64m -Xms64m -jar lib/sql.jar -helper-hosts phoenix581601:8000 -store
mystore
      -security kvroot/security/client.security
```

Login using the user and password you created in the previous step.

```
Login as:root
root's password: <password>
```

Output:

SQL prompt is displayed.

```
sql->
```

Create a table and add some data to the table

```
/*Create a table*/
create table IF NOT EXISTS  ticket (ticketNo LONG,confNo STRING,PRIMARY
KEY(ticketNo))
Output:
Statement completed successfully

/*Load data into the table*/
import -table ticket  -file ticket.json
Output:
Loaded 3 rows to ticket.

/*Read data from the table*/
select * from ticket
Output:
{"ticketNo":34567890,"confNo":"DFGRTH"}
{"ticketNo":12345678,"confNo":"LEJDFG"}
```

```
{"ticketNo":45367892,"confNo":"FGRTFB"}
3 rows returned
```

Create Snapshot of the Store

This section describes how you can create a snapshot of the store.

Invoke the runadmin command to start the admin CLI and securely connect to the store running on host `phoenix581601` at port 8000. In the command below, `kvroot/security` is the security directory. The security directory contains all the security related files that were automatically generated by the securityconfig tool while creating the secure store using the `makebootconfig` command. For more details, see Configuring security in a data store.

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar runadmin -port 8000 -host
phoenix581601localhost -security kvroot/security/client.security
```

Output:

```
Logged in to Admin as anonymous
Connected to Admin in read-only mode
kv->
```

Create a snapshot of the store. For more details, see Backing Up the Store.

```
snapshot create -name Wednesday
```

Output:

```
Created data snapshot named 250514-072244-Wednesday on all 8 components
Successfully created snapshot on RNs: admin1, admin2, rg1-rn1, rg1-rn2, rg2-
rn1, rg2-rn2, rg3-rn1, rg3-rn2
Successfully backed up configurations on: sn1, sn2
```

As indicated in the output message, snapshot folders are created in all the admin and replication nodes that you saw in the ping output. This confirms that the entire store has been successfully backed up.

The following folders are created:

```
kvroot/mystore/sn1/admin1/snapshots/250514-072244-Wednesday
kvroot/mystore/sn1/rg1-rn1/snapshots/250514-072244-Wednesday
kvroot/mystore/sn1/rg2-rn1/snapshots/250514-072244-Wednesday
kvroot/mystore/sn1/rg3-rn1/snapshots/250514-072244-Wednesday
kvroot1/mystore/sn2/admin2/snapshots/250514-072244-Wednesday
kvroot1/mystore/sn2/rg1-rn2/snapshots/250514-072244-Wednesday
kvroot1/mystore/sn2/rg2-rn2/snapshots/250514-072244-Wednesday
kvroot1/mystore/sn2/rg3-rn2/snapshots/250514-072244-Wednesday
```

Restoring the Store

This section talks about restoring a data store from the backup.

We will now see how we can restore the backed-up data on to a new store. Let us create the new store with storage node 3 and storage node 4.

The storage nodes are on the host `phoenix581601` at ports 5000 and 6000. Start the storage node agents. For more details, see Configuring a single region data store.

Use the `configure -name` command to specify the name of the store.

The name of the new store must be same as the name of the store you backed up. In this case, use `mystore`.

Invoke the runadmin command to start the admin CLI utility to securely connect to the store running on the host `phoenix581601` at port 5000. In the command below, `kvroot2/security` is the security directory. The security directory contains all the security related files that were automatically generated by the securityconfig tool while creating the secure store using the `makebootconfig` command. For more details, see Configuring security in a data store.

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar runadmin -port 5000 -host
phoenix581601localhost -security kvroot2/security/client.security
```

Output:

```
Logged in to Admin as anonymous
Connected to Admin in read-only mode
kv->
```

Use the `configure -name` command to specify the name of the store.

```
configure -name mystore
```

Output:

```
Connected to Admin in read-only mode
configured: mystore
```

Exit the admin CLI utility

```
exit
```

Load Store Metadata from Backup

This section tells you how to load the store metadata from backup.

Load the store metadata from the backup using the `-load-admin` command. In the below command, `kvroot/mystore/sn1/admin1/snapshots/250514-072244-Wednesday` is the

snapshot of the admin of the store we backed up. The values provided for `store`, `host` and `port` are that of the new (target) store. For more details, see Recovering the Store.

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar load -source kvroot/mystore/sn1/
admin1/snapshots/250514-072244-Wednesday -store mystore -host phoenix581601 -
port 5000 -load-admin -security kvroot2/security/client.security
```

Output:

```
May 14, 2025 8:37:39 AM oracle.kv.impl.admin.AdminDatabase <init>
INFO: Open: AdminTableDatabase
May 14, 2025 8:37:39 AM oracle.kv.impl.admin.AdminDatabase close
INFO: Closing AdminTableDatabase
May 14, 2025 8:37:39 AM oracle.kv.impl.admin.AdminDatabase <init>
INFO: Open: AdminSecurityDatabase
May 14, 2025 8:37:39 AM oracle.kv.impl.admin.AdminDatabase close
INFO: Closing AdminSecurityDatabase
Logged in to Admin as anonymous
```

Deploy Store

Now, deploy the new store. For more details, see Configuring a single region data store. When prompted for user and password, provide the same user and password that you created in the store you backed up.

Grant Required Privileges to Users

This section is about granting required privileges to user before loading the shard data from the backup to a secure data store.

Any user who wants to restore records into a secure data store needs to have the required privileges. Before restoring data using the load command, run the following commands to grant the user with the required privileges. For more details on roles and privileges, see Managing Roles, Privileges and Users.

Login to the SQL shell. When prompted for user and password, provide the same user and password that you created in the store you backed up. For more details, see Starting the SQL shell. In the command below, `kvroot2/security` is the security directory. The security directory contains all the security related files that were automatically generated by the securityconfig tool while creating the secure store using the `makebootconfig` command. For more details, see Configuring security in a data store.

```
java -Xmx64m -Xms64m -jar lib/sql.jar -helper-hosts phoenix581601:5000 -store
mystore -security kvroot2/security/client.security
Login as:root
root's password:<password>
```

The SQL prompt appears

```
sql->
```

Grant `readwrite` and `writesystable` roles to the user.

```
GRANT readwrite TO USER root

Output:
Statement completed successfully

GRANT writesystable TO USER root

Output:
Statement completed successfully

exit
```

Load Shard Data from Backup

This section explains how you can load shard data from the backup.

Load the shard data from the replication masters of all the 3 shards. The snapshots of the masters are available in the following locations:

```
kvroot1/mystore/sn2/rg1-rn2/snapshots/250514-072244-Wednesday
kvroot/mystore/sn1/rg2-rn1/snapshots/250514-072244-Wednesday
kvroot/mystore/sn1/rg2-rn1/snapshots/250514-072244-Wednesday
```

Load data from the first shard:

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar load -source kvroot1/mystore/sn2/
rg1-rn2/snapshots/250514-072244-Wednesday -store mystore -host phoenix581601 -
port 5000 -security kvroot2/security/client.security
Login as:root
root's password: <password>
```

Output:

```
2025-05-16 06:15:46.204 UTC [LOAD] Load 9 records, 9 records skipped, 0 pre-
existing records from p1 of kvroot1/mystore/sn2/rg1-rn2/snapshots/
250514-072244-Wednesday: 0.017s
2025-05-16 06:15:46.239 UTC [LOAD] Load 9 records, 9 records skipped, 0 pre-
existing records from p10 of kvroot1/mystore/sn2/rg1-rn2/snapshots/
250514-072244-Wednesday: 0.014s
2025-05-16 06:15:46.242 UTC [LOAD] Load 14 records, 14 records skipped, 0 pre-
existing records from p11 of kvroot1/mystore/sn2/rg1-rn2/snapshots/
250514-072244-Wednesday: 0.013s
2025-05-16 06:15:46.243 UTC [LOAD] Load 9 records, 9 records skipped, 0 pre-
existing records from p13 of kvroot1/mystore/sn2/rg1-rn2/snapshots/
250514-072244-Wednesday: 0.001s
....
....
....
2025-05-16 06:15:46.319 UTC [LOAD] Load 8 records, 7 records skipped, 1 pre-
existing records from p32 of kvroot1/mystore/sn2/rg1-rn2/snapshots/
250514-072244-Wednesday: 0.036s
```

```
Load succeeded, wrote 0 records
```

Load data from the second shard

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar load -source kvroot/mystore/sn1/rg2-
rn1/snapshots/250514-072244-Wednesday -store mystore -host phoenix581601 -
port 5000 -security kvroot2/security/client.security
Login as:root
root's password:<password>
```

Output:

```
2025-05-14 09:36:22.315 UTC [LOAD] Load 24 records, 24 records skipped, 0 pre-
existing records from p36 of kvroot/mystore/sn1/rg2-rn1/snapshots/
250514-072244-Wednesday: 0.037s
2025-05-14 09:36:22.351 UTC [LOAD] Load 18 records, 18 records skipped, 0 pre-
existing records from p35 of kvroot/mystore/sn1/rg2-rn1/snapshots/
250514-072244-Wednesday: 0.039s
2025-05-14 09:36:22.357 UTC [LOAD] Load 12 records, 12 records skipped, 0 pre-
existing records from p39 of kvroot/mystore/sn1/rg2-rn1/snapshots/
250514-072244-Wednesday: 0.001s
....
....
....
2025-05-14 09:36:22.492 UTC [LOAD] Load 27 records, 13 records skipped, 14
pre-existing records from p63 of kvroot/mystore/sn1/rg2-rn1/snapshots/
250514-072244-Wednesday: 0.064s

Load succeeded, wrote 0 records
```

Load data from the third shard

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar load -source kvroot/mystore/sn1/rg3-
rn1/snapshots/250514-072244-Wednesday -store mystore -host phoenix581601 -
port 5000 -security kvroot2/security/client.security
Login as:root
root's password:<password>
```

Output:

```
2025-05-14 09:37:04.554 UTC [LOAD] Load 18 records, 18 records skipped, 0 pre-
existing records from p68 of kvroot/mystore/sn1/rg3-rn1/snapshots/
250514-072244-Wednesday: 0.017s
2025-05-14 09:37:04.572 UTC [LOAD] Load 17 records, 17 records skipped, 0 pre-
existing records from p69 of kvroot/mystore/sn1/rg3-rn1/snapshots/
250514-072244-Wednesday: 0.011s
2025-05-14 09:37:04.575 UTC [LOAD] Load 20 records, 20 records skipped, 0 pre-
existing records from p70 of kvroot/mystore/sn1/rg3-rn1/snapshots/
250514-072244-Wednesday: 0.001s
....
....
....
```

```
2025-05-14 09:37:04.699 UTC [LOAD] Load 12 records, 11 records skipped, 0 pre-
existing records from p89 of kvroot/mystore/sn1/rg3-rn1/snapshots/
250514-072244-Wednesday: 0.046s

Load succeeded, wrote 1 records
```

Now, we have loaded data from all the 3 shards.

Verify Restored Data

This section explains how you can verify that you have successfully restored the data from the backup.

Start the SQL shell on storage node 3:

```
java -Xmx64m -Xms64m -jar lib/sql.jar -helper-hosts phoenix581601:5000 -store
mystore -security kvroot2/security/client.security
Login as:root
root's password:<password>
```

SQL prompt is displayed

```
sql->
```

Enter command to show the tables

```
show tables
```

Output:

```
tables
  SYS$IndexStatsLease
  SYS$MRTableAgentStat
  SYS$MRTableInfo
  SYS$MRTableInitCheckpoint
  SYS$PartitionStatsLease
  SYS$SGAttributesTable
  SYS$StreamRequest
  SYS$StreamResponse
  SYS$TableMetadata
  SYS$TableStatsIndex
  SYS$TableStatsPartition
  SYS$TopologyHistory
  ticket
```

Read the data in table ticket.

```
select * from ticket
```

Output:

```
{"ticketNo":34567890,"confNo":"DFGRTH"}
{"ticketNo":12345678,"confNo":"LEJDFG"}
{"ticketNo":45367892,"confNo":"FGRTFB"}

3 rows returned
```

You can see that the metadata, tables and data that you had backed up are now restored and available on the new store.

Recovering from Data Corruption

Oracle NoSQL Database can automatically detect data corruption in the database store. When it detects data corruption, Oracle NoSQL Database automatically shuts down the associated Admin or Replication Nodes. Manual administrative action is then required before the nodes can be brought back online.

Detecting Data Corruption

Oracle NoSQL Database Admin or Replication Node processes will exit when they detect data corruption. This is caused by a background task which detects data corruption caused by a disk failure, or similar physical media or I/O subsystem problem. Typically, the corruption is detected because of a checksum error in a log entry in one of the data (*.jdb) files contained in an Admin or Replication Node database environment. A data corruption error generates output in the debug log similar to this:

```
2024-04-05 16:59:52.265 UTC SEVERE [rg1-rn1] Process exiting
com.sleepycat.je.EnvironmentFailureException: (JE 7.3.2)
rg1-rn1(-1):kvroot/mystore/sn1/rg1-rn1/env
com.sleepycat.je.log.ChecksumException:
Invalid log entry type: 102 lsn=0x0/0x0 bufPosition=5
bufRemaining=4091 LOG_CHECKSUM:
Checksum invalid on read, log is likely invalid. Environment is
invalid and must be closed
...
2024-04-05 16:59:52.270 UTC SEVERE [rg1-rn1] Exception creating
service rg1-rn1:
(JE 7.3.2) rg1-rn1(-1):kvroot/mystore/sn1/rg1-rn1/env
com.sleepycat.je.log.ChecksumException:
Invalid log entry type: 102 lsn=0x0/0x0 bufPosition=5
bufRemaining=4091 LOG_CHECKSUM:
Checksum invalid on read, log is likely invalid. Environment is
invalid and must be closed. (12.1.4.3.0): oracle.kv.FaultException:
(JE 7.3.2) rg1-rn1(-1):kvroot/mystore/sn1/rg1-rn1/env
com.sleepycat.je.log.ChecksumException: Invalid log entry type: 102
lsn=0x0/0x0 bufPosition=5 bufRemaining=4091 LOG_CHECKSUM: Checksum
invalid on read, log is likely invalid. Environment is invalid and
must be closed. (12.1.4.3.0)
Fault class name: com.sleepycat.je.EnvironmentFailureException
...
2024-04-05 16:59:52.272 UTC INFO [rg1-rn1] Service status changed
from STARTING to ERROR_NO_RESTART
```

The `EnvironmentFailureException` will cause the process to exit. Because the exception was caused by log corruption, the service status is set to `ERROR_NO_RESTART`, which means that the service will not restart automatically.

Data Corruption Recovery Procedure

If an Admin or Replication Node has been stopped due to data corruption, then manual administration intervention is required in order to restart the Node:

1. Optional: Archive the corrupted environment data files.

   If you want to send the corrupted environment to Oracle support for help in identifying the root cause of the failure, archive the corrupted environment data files. These are usually located at:

   ```
   <KVROOT>/<STORE_NAME>/<SNx>/<Adminx>/"
   ```

   or

   ```
   <KVROOT>/<STORE_NAME>/<SNx>/<rgx-rnx>"
   ```

   However, if you used the plan change-storagedir CLI command to change the storage directory for your Replication Node, then you will find the environment in the location that you specified to that command.

   You can use the show topology CLI command to display your store's topology. As part of this information, the storage directory for each of your Replication Nodes are identified.

2. Confirm that a non-corrupted version of the data is available.

   Before removing the files associated with the corrupted environment, confirm that another copy of the data is available either on another node or via a previously save snapshot. For a Replication Node, you must be using a Replication Factor greater than 1 and also have a properly operating Replication Node in the store in order for the data to reside elsewhere in the store. If you are using a RF=1, then you must have a previously saved snapshot in order to continue.

   If the problem is with an Admin Node, there must be to be another Admin available in the store that is operating properly.

   Use the ping or verify configuration commands to check if the available nodes are running properly and healthy.

3. Remove all the data files that reside in the corrupted environment.

   Once the data files associated with a corrupted environment have been saved elsewhere, and you have confirmed that another copy of the data is available, delete all the data files in the enviroment directory. Make sure you only delete the files associated with the Admin or Replication Node that has failed due to a corrupted environment error.

   ```
   # ls <KVROOT>/mystore/sn1/rg1-rn1/env
   00000000.jdb  00000001.jdb  00000002.jdb  je.config.csv
   je.info.0 je.lck  je.stat.csv

   # rm <KVROOT>/mystore/sn1/rg1-rn1/env/*.jdb
   ```

4. Perform recovery using either Network Restore, or from a backup. Be aware the recovery from a backup will not work to recover an Admin Node.

- Recovery using Network Restore

  Network restore can be used to recover from data corruption if the corrupted node belongs to a replication group that has other replication nodes available. Network restore is automatic recovery task. After removing all of the database files in the corrupted environment, you only need to connect to CLI and restart the corrupted node.

  For a Replication Node:

  ```
  kv-> plan start-service -service rg1-rn1
  ```

  For an Admin:

  ```
  kv-> plan start-service -service rg1-rn1
  ```

- Recovery from a backup (RNs only)

  If the store does not have another member in the Replication Node's shard or if all of the nodes in the shard have failed due to data corruption, you will need to restore the node's environment from a previously created snapshot. See Recovering the Store for details.

  Note that to recover an Admin that has failed due to data corruption, you must have a working Admin somewhere in the store. Snapshots do not capture Admin data.

Replacing a Failed Disk

You can replace a disk that is either in the process of failing, or has already failed. Disk replacement procedures are necessary to keep the store running. These are the steps required to replace a failed disk to preserve data availability.

The following example deploys a KVStore to a set of three machines, each with 3 disks. Use the `storagedir` flag of the `makebootconfig` command to specify the storage location of the disks.

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar makebootconfig \
    -root /opt/ondb/var/kvroot \
    -port 5000  \
    -host node09
    -harange 5010,5020 \
    -num_cpus 0  \
    -memory_mb 0 \
    -capacity 3  \
    -admindir /disk1/ondb/admin -admindirsize 1_gb \
    -storagedir /disk1/ondb/data \
    -storagedir /disk2/ondb/data \
    -storagedir /disk3/ondb/data \
    -rnlogdir /disk1/ondb/rnlog01
```

With a boot configuration such as the previous example, the directory structure created and populated on each machine is as follows:

```
 - Machine 1 (SN1) -      - Machine 2 (SN2) -     - Machine 3 (SN3) -
/opt/ondb/var/kvroot   /opt/ondb/var/kvroot  /opt/ondb/var/kvroot
```

```
  /security              /security              /security
  /store-name             /store-name            /store-name
    /sn1                    /sn2                   /sn3
      config.xml              config.xml             config.xml

/disk1/ondb/admin       /disk1/ondb/admin      /disk1/ondb/admin
  /admin1                 /admin2                /admin3
    /env                    /env                   /env

/disk1/ondb/data        /disk1/ondb/data       /disk1/ondb/data
  /rg1-rn1                /rg1-rn2               /rg1-rn3
    /env                    /env                   /env

/disk2/ondb/data        /disk2/ondb/data       /disk2/ondb/data
  /rg2-rn1                /rg2-rn2               /rg2-rn3
    /env                    /env                   /env

/disk3/ondb/data        /disk3/ondb/data       /disk3/ondb/data
  /rg3-rn1                /rg3-rn2               /rg3-rn3
    /env                    /env                   /env

/disk1/ondb/rnlog01     /disk1/ondb/rnlog01     /disk1/ondb/rnlog01
  /log                    /log                    /log
```

In this case, configuration information and administrative data is stored in a location that is separate from all of the replication data. The replication data itself is stored by each distinct Replication Node service on separate, physical media as well. Storing data in this way provides failure isolation and will typically make disk replacement less complicated and time consuming. For information on how to deploy a store, see Configuring a single region data store.

To replace a failed disk:

1. Determine which disk has failed. To do this, you can use standard system monitoring and management mechanisms. In the previous example, suppose disk2 on Storage Node 3 fails and needs to be replaced.
2. Then given a directory structure, determine which Replication Node service to stop. With the structure described above, the store writes replicated data to disk2 on Storage Node 3, so `rg2-rn3` must be stopped before replacing the failed disk.
3. Use the `plan stop-service` command to stop the affected service (rg2-rn3) so that any attempts by the system to communicate with it are no longer made; resulting in a reduction in the amount of error output related to a failure you are already aware of.

   ```
   kv-> plan stop-service -service rg2-rn3
   ```

4. Remove the failed disk (disk2) using whatever procedure is dictated by the operating system, disk manufacturer, and/or hardware platform.
5. Install a new disk using any appropriate procedures.
6. Format the disk to have the same storage directory as before; in this case, `/disk2/ondb/var/kvroot`.

7. With the new disk in place, use the `plan start-service` command to start the `rg2-rn3` service.

```
kv-> plan start-service -service rg2-rn3
```

> **Note:**
>
> Depending on the amount of data stored on the disk before it failed, recovering that data can take a considerable amount of time. Also, the system may encounter unexpected or additional network traffic and load while repopulating the new disk. If so, such events add even more time to completion.

Replacing a Failed Storage Node

You can replace a failed Storage Node, or one that is in the process of failing. Upgrading a healthy machine to another one with better specifications is also a common Storage Node replacement scenario. Generally, you should repair the underlying problem (be it hardware or software related) before proceeding with this procedure.

There are two ways to replace a failed Storage Node:

- A new, different Storage node
- An identical Storage Node

This section describes both replacement possibilities.

> **Note:**
>
> Replacing a Storage Node qualifies as a topology change. This means that if you want to restore your store from a snapshot taken before the Storage Node was replaced, you must use the `Load` program. See Using the Load Program for more information.

Using a New Storage Node

To replace a failed Storage Node by using a new, different Storage Node (node uses different host name, IP address, and port as the failed host):

1. If you are replacing hardware, bring it up and make sure it is ready for your production environment.
2. On the new, replacement node, create a "boot config" configuration file using the `makebootconfig` utility with the following commands. Enable the security configuration option in the new node. Do this on the hardware where your new Storage Node runs.

```
> mkdir -p KVROOT     (if it doesn't already exist)
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar makebootconfig -root KVROOT \
                                           -port 5000 \
                                           -host <hostname> \
```

```
-harange 5010,5020 \
-capacity 1 \
-admindir /export/admin1 \
-admindirsize 3_gb \
-store-security enable \
-storagedir /export/data1 \
-storagedirsize 1_tb \
-rnlogdir /export/rnlog1
```

3. Create the security directory under KVROOT in your new node.

```
> cd KVROOT
> mkdir security
```

4. Copy the security directory from a healthy node to the failed node:

```
scp -r <sec dir> node02:KVROOT/security
```

5. Start the Oracle NoSQL Database software on the new node:

```
> nohup java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar start -root KVROOT &
```

6. Deploy the new Storage Node to the new node. To do this using the CLI:

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar runadmin \
-port <5000> -host <host> \
 -security security/client.security
kv-> plan deploy-sn -zn <id> -host <host> -port <5000> -wait
```

7. Add the new Storage Node to the Storage Node pool. (You created a Storage Node pool when you installed the store, and you added all your Storage Nodes to it, but it is otherwise not used in this version of the product.)

```
kv-> show pools
AllStorageNodes: sn1, sn2, sn3, sn4 ... sn25, sn26
BostonPool: sn1, sn2, sn3, sn4 ... sn25
kv-> pool join -name BostonPool -sn sn26
AllStorageNodes: sn1, sn2, sn3, sn4 ... sn25, sn26
BostonPool: sn1, sn2, sn3, sn4 ... sn25
```

8. Make sure the old Storage Node is not running. If the problem is with the hardware, then turn off the broken machine. You can also stop just the Storage Node software by:

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar stop -root KVROOT &
```

9. Migrate the services from one Storage Node to another. The syntax for this plan is:

```
kv-> plan migrate-sn -from <old SN ID> -to <new SN ID> -wait
```

Assuming that you are migrating from Storage Node 25 to 26, you would use:

```
kv-> plan migrate-sn -from sn25 -to sn26 -wait
```

10. The old Storage Node is shown in the topology and is reported as UNREACHABLE. The source SNA should be removed and its rootdir should be hosed out. Bringing up the old SNA will also bring up the old Replication Nodes and admins, which are no longer members of their replication groups. This should be harmless to the rest of the store, but it produces log error messages that might be misinterpreted as indicating a problem with the store. Use the `plan remove-sn` command to remove the old and unused Storage Node in your deployment.

    ```
    kv-> plan remove-sn sn sn25 -wait
    ```

11. Use the ping command to verify the migration to the new node is complete and all services are running well.

    ```
    > java -Xmx64m -Xms64m \
            -jar KVHOME/lib/kvstore.jar ping \
            -port <5000> -host <host> \
            -security security/client.security
    ```

> **Note:**
>
> Replacing a Storage Node qualifies as a topology change. This means that if you want to restore your store from a snapshot taken before the Storage Node was replaced, you must use the `Load` program. See Using the Load Program for more information.

Task for an Identical Node

To replace a failed Storage Node with an identical node, i.e. the target node uses the same host name, internet address, and port as the failed host.

1. Prerequisite information:
    a. The hostname and port number (registry port) of the machine in the cluster where the admin process is running (e.g “host1” and 5000).
    b. The ID of the Storage Node to replace (e.g. "sn1").

    > **Note:**
    >
    > The user can use the Admin CLI ping command to get the registry port and Storage Node Identifier of any failed Storage Node.

    c. Before starting the new Storage Node, the Storage Node to be replaced must be taken down. This can be done administratively or via failure.

> **Note:**
>
> The instructions below assume that the KVROOT in the target host is empty and has no valid data. When the new Storage Node Agent begins it starts the services that it hosts, which recovers their data from other hosts. The time taken for the recovery depends on the size of the shards involved and it happens in the background.

2. Create the configuration file of the failed host using the `generateconfig` command. The `generateconfig` command can be executed from any active host (machine) in the NoSQL cluster.

   The generateconfig's usage is:

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar generateconfig \
-host <hostname> -port <port> -sn <StorageNodeId> -target <zipfile> \
-security  <path to security login file>
```

Parameter	Required	Description
`host`	Yes	The host name of the failed storage node for which the config file is generated.
`port`	Yes	The registry port of the failed storage node for which the config file is generated.
`sn`	Yes	Identifier of the failed storage node.
`target`	Yes	Full path of the zip file to be created.
`security`	No	The client security configuration file. This parameter is only required if your store is secure. A fully qualified path to a file containing security information can be specified.

   For more information on generateconfig command, See generateconfig

   For example:

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar generateconfig -host adminhost \
-port 13230 -sn sn1 -target /tmp/sn1.config.zip \
-security USER/security/admin.security
```

   The command above creates the target "`/tmp/sn1.config.zip`" . This is a zip file with the required configuration to re-create the failed Storage Node. The top-level directory in the newly created zip file (`sn1.config.zip`) is the store's KVROOT.

> **Note:**
>
> This assumes that you must have followed the steps as mentioned in Create users and configure security with remote access .

3. Restore the Storage Node configuration on the target host:
   a. Copy the zip file "sn1.config.zip" to the target host.
   b. Unzip the archive into your target host's KVROOT directory. That is, if KVROOT is /opt/kvroot, then do the following:

```
> cd /opt
> unzip <path-to-sn1.config.zip>
```

> **Note:**
>
> If kvroot already exists under /opt directory , remove all the contents in the kvroot directory before unzipping the config file.

4. Restart the Storage Node on the target host.

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar start -root KVROOT
```

> **Note:**
>
> The hostname, port number and internet address of the target host and the failed node are the same. So no changes have to be done in the Storage Node pool and the topology of the store.

Repairing a Failed Zone by Replacing Hardware

If all of the machines belonging to a zone fail, and quorum is maintained, you can replace them by using new, different Storage Nodes deployed to the same zone.

If a zone fails but quorum is lost, you can perform a failover instead. To do this, see Performing a Failover.

For example, suppose a store consists of three zones; zn1, deployed to the machines on the first floor of a physical data center, zn2, deployed to the machines on the second floor, and zn3, deployed to the third floor. Additionally, suppose that a fire destroyed all of the machines on the second floor, resulting in the failure of all of the associated Storage Nodes. In this case, you need to replace the machines in the zn2 zone; which can be accomplished by doing the following:

1. Replace each individual Storage Node in the failed zone with new, different Storage Nodes belonging to same zone (zn2), although located in a new physical location. To do this, follow the instructions in Replacing a Failed Storage Node. Make sure to remove each old Storage Node after performing the replacement.

2. After replacing and then removing each of the targeted SNs, the zone to which those SNs belonged should now contain the new SNs.

Managing your kvstore

Topics:

Increasing Storage Node Capacity

You can increase the capacity of a Storage Node by adding additional hard disks. Adding hard disks to a Storage Node permits the placement of each Replication Node on its own disk, ensuring that the Replication Nodes on the SN are not competing for I/O resources. Specify the location of the storage directory on the new disk using the `storagedir` parameter.

> **Note:**
>
> When you specify a storage directory, Oracle strongly recommends you also specify the storage directory size using the `-storagedirsize` parameter. See Managing Storage Directory Sizes for details. The system uses the configured directory sizes to enforce disk usage. Be sure to specify a storage directory size for every storage node in the store.

The following example demonstrates deploying a new store and adding two more disks to a Storage Node, increasing the capacity from 1 to 3:

1. Create, start and configure the new store.
   - Create the new store:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT \
-host node20 -port 5000 \
-harange 5010,5030 \
-capacity 1 \
-memory_mb 200 \
-storagedir /disk1/ondb/data
```

- Create and copy the security directory:

```
java -Xmx64m -Xms64m \
-jar kv/lib/kvstore.jar \
securityconfig config create -root KVROOT -kspwd password
Created files
KVROOT/security/security.xml
KVROOT/security/store.keys
KVROOT/security/store.trust
KVROOT/security/client.trust
KVROOT/security/client.security
KVROOT/security/store.passwd (Generated in CE version)
KVROOT/security/store.wallet/cwallet.sso (Generated in EE version)

Created
```

- Start the new store:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar start \
-root KVROOT &
```

- Configure the new store:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar runadmin \
-port 5000 -host node20 \
-security KVROOT/security/client.security


kv-> configure -name kvstore
```

Output:

```
Store configured: kvstore
```

2. Create a zone. Then create an administration process on a specific host:

```
kv-> plan deploy-zone -name Houston -rf 1 -wait
```

Output:

```
Executed plan 1, waiting for completion...
Plan 1 ended successfully


kv-> plan deploy-sn -znname "Houston" -port 5000 -wait -host node20
```

Output:

```
Executed plan 2, waiting for completion...
Plan 2 ended successfully
```

```
kv-> plan deploy-admin -sn sn1 -port 5001 -wait
```

Output:

```
Executed plan 3, waiting for completion...
Plan 3 ended successfully
```

3. Create the storage node pool. Then add the storage node to the pool:

```
kv-> pool create -name AllStorageNodes
```

```
kv-> pool join -name AllStorageNodes -sn sn1
```

4. Create a topology, preview it, and then deploy it:

```
kv-> topology create -name 1x1 -pool AllStorageNodes -partitions 120
```

Output:

```
Created: 1x1
```

```
kv-> topology preview -name 1x1
```

Output:

```
Topology transformation from current deployed topology to 1x1:
Create 1 shard
Create 1 RN
Create 120 partitions

shard rg1
  1 new RN : rg1-rn1
  120 new partitions
```

```
kv-> plan deploy-topology -name 1x1 -wait
```

Output:

```
Executed plan 4, waiting for completion...
Plan 4 ended successfully
```

5. Add two more disk drives to the Storage Node, mounted as `disk2` and `disk3`. Add the storage directories using the `plan change-storagedir` command. Be sure to add the Storage Directory size, such as `-storagedirsize "1 tb"`.

```
kv-> plan change-storagedir -sn sn1 -storagedir /disk2/ondb/data \
-storagedirsize "1 tb" -add -wait
```

Output:

```
Executed plan 5, waiting for completion...
Plan 5 ended successfully
```

```
kv-> plan change-storagedir -sn sn1 -storagedir /disk3/ondb/data \
-storagedirsize "1 tb" -add -wait
```

Output:

```
Executed plan 6, waiting for completion...
Plan 6 ended successfully
```

> **Note:**
>
> Because we specified storage directory sizes in the previous example, it is necessary to provide that information to your other nodes **if** you have not already done so. See Managing Storage Directory Sizes for more information.

6. Change the capacity equal to the total number of disks now available on the Storage Node (3).

```
kv-> plan change-parameters -service sn1 -wait -params capacity=3
```

Output:

```
Executed plan 7, waiting for completion...
Plan 7 ended successfully
```

> **Note:**
>
> You need to perform last two steps on all the Storage Nodes (in your cluster) to add the disk drives and increase the capacity of each Storage Node. In this case, it is a single node deployment, so the topology is now ready to be redistributed.

7. Redistribute your topology to expand the cluster in order to use the new capacity (3) of the Storage Node.

```
kv-> topology clone -current -name 3x1
```

Output:

```
Created 3x1
```

```
kv-> topology redistribute -name 3x1 -pool AllStorageNodes
```

Output:

```
Redistributed: 3x1
```

```
kv-> topology preview -name 3x1
```

Output:

```
Topology transformation from current deployed topology to 3x1:
Create 2 shards
Create 2 RNs
Migrate 80 partitions

shard rg2
  1 new RN : rg2-rn1
  40 partition migrations
shard rg3
  1 new RN : rg3-rn1
  40 partition migrations
```

```
kv-> plan deploy-topology -name 3x1 -wait
```

Output:

```
Executed plan 8, waiting for completion...
Plan 8 ended successfully
```

Managing Storage Directory Sizes

We strongly recommend that you always specify storage directory sizes for each Replication Node on every Storage Node in the store. Doing so sets disk threshold levels for each replication node, even when your store has hardware with varying disk capacities. This section describes this topic, and others.

Managing Disk Thresholds

It is very important to configure each storage directory with a specific amount of available disk space. The Oracle NoSQL Database uses the configured Storage Directory sizes to enforce disk space limits. Without configuring how much disk space is available, the store opportunistically uses all available space, less 5 GB free disk space. The system maintains 5 GB of free space to allow manual recovery if the Storage Node exceeds its configured disk limit. Be sure to monitor disk usage regularly using the statistics provided, as described in Monitoring Disk Usage.

Storage Nodes use their available disk space for two purposes:

- To store your data.
- To save reserved files.

Reserved files consist of data that has already been replicated to active replica nodes. The purpose of storing a copy of this data is to use for Replica Nodes that lose contact with the Master Node. Losing contact typically occurs because Replica nodes are shut down, or a network partition event occurs, or because another transient problem occurs. The Storage Node is primarily designed to consume the amount of disk space you assign it, and to use the remaining disk space to save the reserved files. Each Storage Node manages its available disk space, leaving 5 GB free for recovery purposes. Your intervention is typically not required in this disk management process, unless a storage node exceeds its available disk capacity.

> **Note:**
>
> If a Storage Node (SN) consumes more than what is assigned as *storagedirsize*, including leaving 5 GB of space free, the SN automatically attempts to free up disk space by deleting reserved files (**not** your data files), until more than 5 GB of space is available. If the Storage Node is unable to free up enough space, it suspends write operations to the node. Read operations continue as normal. Write operations resume automatically once the node obtains sufficient free disk space.

You can limit how much disk space the store consumes on a node by node basis, by explicitly specifying a storage directory size for each storage node, as described in Specifying Storage Directory Sizes. Storage nodes can then consume all of their configured disk space as needed, leaving free the required 5 GB. However, if you do **not** indicate a storage directory size, the Storage Node uses disk space until it consumes the disk, except for the required 5 GB for manual recovery.

Consider a storage node with a 200 GB disk. Without configuring a *storagedirsize* for that disk, the store keeps consuming up to 195 GB of disk space (leaving only the 5 GB for manual recovery). If your standard policy requires a minimum 20 GB available space on each disk, you must configure the storage node with a *storagedirsize* of 175 GB, leaving 20 GB available, and 5 GB for store recovery.

The most common reason a node's storage directory fills up is because of reserved files. If the Storage Node exceeds its disk threshold, it continues to delete the reserved files until the threshold is no longer exceeded.

Specifying Storage Directory Sizes

Use the makebootconfig `storagedirsize` parameter to specify Storage Node (SN) capacity when you initially install your store. See Configuring your data store installation and makebootconfig for details. Additionally, if your SN has the capacity to support more than one Replication Node, specify a storage directory location and storage directory size for each Replication Node.

To specify or change storage capacity after you have installed the store, use `plan change-storagedir`. When you use `plan change-storagedir` be sure to specify the `-storagedirsize` parameter to indicate how large the new storage directory is.

> **Note:**
>
> If you specify the `-storagedir` parameter, but not `-storagedirsize`, `makebootconfig` displays a warning. Always specify both parameters for control and tracking.

The value specified for the `storagedirsize` parameter must be a long, optionally followed by a unit string. Accepted unit strings are: KB, MB, GB, and TB, corresponding to 1024, 1024^2, 1024^3, 1024^4 respectively. Acceptable strings are case insensitive. Valid delimiters between the long value and the unit string are " ", "-", or "_". If you specify the delimiter as " ", your value should be enclosed in double quotes, For example "10 GB". If you have any other delimiter double quotes is not mandatory. For example 10_GB or 10-GB.

For example:

```
kv-> verbose
Verbose mode is now on
kv-> show topology
store=mystore  numPartitions=300 sequence=308
  zn: id=zn1 name=Manhattan repFactor=3 type=PRIMARY
allowArbiters=false

  sn=[sn1] zn:[id=zn1 name=Manhattan] node1:9000 capacity=1 RUNNING
    [rg1-rn1] RUNNING  /storage-dir/sn1 0
              No performance info available
  sn=[sn2] zn:[id=zn1 name=Manhattan] node2:9000 capacity=1 RUNNING
    [rg1-rn2] RUNNING  /storage-dir/sn2 0
             single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
  sn=[sn3] zn:[id=zn1 name=Manhattan] node3:9000 capacity=1 RUNNING
    [rg1-rn3] RUNNING  /storage-dir/sn3 0
              No performance info available

  shard=[rg1] num partitions=300
    [rg1-rn1] sn=sn1 haPort=node1:9010
    [rg1-rn2] sn=sn2 haPort=node2:9010
    [rg1-rn3] sn=sn3 haPort=node3:9010
    partitions=1-300

kv-> plan change-storagedir -sn sn1 -storagedir /storage-dir/sn1 \
-storagedirsize "200 gb" -add -wait
Executed plan 7, waiting for completion...
Plan 7 ended successfully
kv-> plan change-storagedir -sn sn2 -storagedir /storage-dir/sn2 \
-storagedirsize "300 gb" -add -wait
Executed plan 8, waiting for completion...
Plan 8 ended successfully
kv-> plan change-storagedir -sn sn3 -storagedir /storage-dir/sn3 \
-storagedirsize "400 gb" -add -wait
Executed plan 9, waiting for completion...
Plan 9 ended successfully
kv-> show topology
store=mystore  numPartitions=300 sequence=308
  zn: id=zn1 name=Manhattan repFactor=3 type=PRIMARY
allowArbiters=false
```

```
sn=[sn1] zn:[id=zn1 name=Manhattan] node1:9000 capacity=1 RUNNING
  [rg1-rn1] RUNNING  /storage-dir/sn1 214748364800
            No performance info available
sn=[sn2] zn:[id=zn1 name=Manhattan] node2:9000 capacity=1 RUNNING
  [rg1-rn2] RUNNING  /storage-dir/sn2 322122547200
          single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
sn=[sn3] zn:[id=zn1 name=Manhattan] node3:9000 capacity=1 RUNNING
  [rg1-rn3] RUNNING  /storage-dir/sn3 429496729600
          single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms

shard=[rg1] num partitions=300
  [rg1-rn1] sn=sn1 haPort=node1:9010
  [rg1-rn2] sn=sn2 haPort=node2:9010
  [rg1-rn3] sn=sn3 haPort=node3:9010
  partitions=1-300
```

Note:

If any Storage Node stores its data in the root directory (not recommended), then instead of `plan change-storagedir`, set the `rootDirSize` parameter. For example:

```
kv-> plan change-parameters -service sn1 -params rootDirSize=200_gb
```

Specifying Differing Disk Capacities

By default, Oracle NoSQL Database evenly distributes data across all the Storage Nodes in your store. No check is made in advance. The store expects all of the hardware in your storee to be homogenous, and so all Storage Nodes would have the same disk capacity.

However, more likely, you are running a store in an environment where some Storage Nodes have more disk capacity than others. In this case, you must specify appropriate disk capacity for each storage node. Oracle NoSQL Database will then place more data on higher capacity Storage Nodes. Be aware that specifying greater disk capacity to a storage node can result in an increased workload. Storage Nodes with more capacity than others could then serve more read and/or write activity. Be sure to size your storage nodes accordingly to support additional workload, if any.

Monitoring Disk Usage

If a Storage Node exceeds its disk usage threshold value (*storagedirsize* - 5GB), then all write activity for that node is suspended until sufficient disk space is made available. The store makes disk space available by removing reserved files to satisfy the threshold requirement. No data files are removed. Read activity continues while reserved data is being removed.

To ensure that your Storage Node can continue to service write requests, monitor the `availableLogSize` JMX statistic. This represents the amount of space that can be used by write operations. This value is **not** necessarily representative of the amount of disk space currently in use, since quite a lot of disk space can, and is, used for reserved files, which are not included in the `availableLogSize` statistic.

Reserved files are data files that have already been replicated, but which are retained for replication to nodes that are out of contact with the master node. Because Oracle NoSQL

Database liberally reserves files, all available storage will frequently be consumed by reserved data. However, reserved data is automatically deleted as necessary by the Storage Node to continue write operations. For this reason, monitoring the actual disk usage is not meaningful.

If `availableLogSize` reaches zero, writes are suspended for the Storage Node. Earlier, as `availableLogSize` approaches zero, the node has less and less space for reserved data files. The result is that the store becomes increasingly less resilient in the face of a prolonged but temporary node outage because there are increasingly fewer historical log files that the store can use to gracefully bring a node up to date once it is available again.

The following tables lists some other useful statistics about disk usage. These statistics are stored in the stats file, or you can monitor them using the JMX `oracle.kv.repnode.envmetric` type. (Xref)

Statistic	Description
`availableLogSize`	Disk space available (in bytes) for write operations. This value is calculated with consideration fo reserved data files, which are deleted automatically whenever space is required to perform write operations: `free space + reservedLogSize - protectedLogSize` In general, monitoring disk usage in the file system is not meaningful, because of the presence of reserved files that can be deleted automatically.
`activeLogSize`	Bytes used by all active data files: files required for basic operation.
`reservedLogSize`	Bytes used by all reserved data files: files that have been cleaned and can be deleted if they are not protected.
`protectedLogSize`	Bytes used by all protected data files: the subset of reserved files that are temporarily protected and cannot be deleted.
`ProtectedLogSizeMap`	A breakdown of protectedLogSize as a map of protecting entity name to protected size in bytes.
`TotalLogSize`	Total bytes used by data files on disk: activeLogSize + reservedLogSize.

The following list from part of some JMX output, shows an example of how you will see each statistic. All of these statistic names have a `Cleaning_` prefix, indicating that they may be in the log cleaning statistics group (for garbage collection):

```
.
.
.
"Cleaning_nRepeatIteratorReads": 0,
"Cleaning_nLNsExpired": 0,
"Cleaning_nCleanerRuns": 0,
"Cleaning_nBINDeltasDead": 0,
"Cleaning_nCleanerDisksReads": 0,
"Cleaning_protectedLogSizeMap": "",
"Cleaning_nCleanerDeletions": 0,
"Cleaning_nCleanerEntriesRead": 0,
"Cleaning_availableLogSize": 48942137344,
"Cleaning_nLNsDead": 0,
"Cleaning_nINsObsolete": 0,
"Cleaning_activeLogSize": 112716,
"Cleaning_nINsDead": 0,
"Cleaning_nINsMigrated": 0,
```

```
"Cleaning_totalLogSize": 112716,
"Cleaning_nBINDeltasCleaned": 0,
"Cleaning_nLNsObsolete": 0,
"Cleaning_nLNsCleaned": 0,
"Cleaning_nLNQueueHits": 0,
"Cleaning_reservedLogSize": 0,
"Cleaning_protectedLogSize": 0,
"Cleaning_nClusterLNsProcessed": 0,
"Node Compression_processedBins": 0,
.
.
.
```

You can tell if writes have been suspended for a Storage Node using the `ping` command from the CLI. In the following sample output, the Shard Status shows `read-only:1`. This indicates that one of the Storage Nodes is in read-only mode. The likeliest reason for that is that it has exceeded its disk threshold.

```
kv-> ping
```

Output:

```
Pinging components of store istore based upon topology sequence #11
3 partitions and 3 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
Shard Status: healthy:0 writable-degraded:0 read-only:1 offline:0
Admin Status: healthy
Zone [name=dc1 id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
RN Status: online:1 offline:2
Storage Node [sn1] on sn1.example.com:5000 Zone: [name=dc1
id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false] Status: RUNNING
Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Admin [admin1] Status: RUNNING,MASTER
Rep Node [rg1-rn1] Status: RUNNING,MASTER (non-authoritative)
sequenceNumber:39,177,477 haPort:5011 available storage size:6 GB
Storage Node [sn2] on sn2.example.com:5000 Zone:
[name=dc1 id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Rep Node [rg1-rn2] Status: RUNNING,UNKNOWN sequenceNumber:39,176,478
haPort:5010 available storage size:NOT AVAILABLE delayMillis:?
catchupTimeSecs:?
Storage Node [sn3] on sn3.example.com:5000 Zone: [name=dc1
id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false] Status: RUNNING
Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Rep Node [rg1-rn3] Status: RUNNING,UNKNOWN sequenceNumber:39,166,804
haPort:5010 available storage size:NOT AVAILABLE delayMillis:?
catchupTimeSecs:?
```

For information on JMX monitoring the store, see Java Management Extensions (JMX) Notifications.

Handling Disk Limit Exception

If a Storage Node exceeds its disk usage threshold value (`<storagedirsize> - 5 GB`), then the store suspends all write activities on that node, until sufficient data is removed to satisfy the threshold requirement. In such a situation, there are two ways to bring the store back to read and write availability, without deleting user data.

- Increasing `storagedirsize` on one or more Replication Nodes if there is available disk space
- Expanding the store by adding a new shard

If there is enough space left on the disk or if the complete disk size is not set as the size of `storagedirsize`, you can bring back the write availability (without any additional need of the hardware) by simply increasing the storage directory size for one or more Replication Nodes.

If there is not enough space left on disk or if the complete disk size is set as the size of the storage directory, then you should follow the store expansion procedure, where you will need additional hardware to increase the number of shards by one.

> **Note:**
>
> If you are following the store expansion procedure, it is important to check the performance files to see if the cleaner is working well, by monitoring the `minUtilization` statistics. If the `minUtilization` statistics is less than 30%, it may mean that the cleaner is not keeping up. In this case it is not possible to perform store expansion.
> Store expansion can only be performed if the `minUtilization` statistics percentage is not less than 30%.
>
> For example:
>
> ```
> 2024-04-05 16:07:12.499 UTC INFO [rg1-rn1] JE: Clean file 0x2b:
> predicted min util is below minUtilization, current util min: 39 max:
> 39,
> predicted util min: 39 max: 39, chose file with util min: 30 max: 30
> avg: 30
>
> 2024-04-05 16:07:04.029 UTC INFO [rg1-rn2] JE: Clean file 0x27:
> predicted min util is below minUtilization, current util min: 39 max:
> 39,
> predicted util min: 39 max: 39, chose file with util min: 30 max: 30
> avg: 30
>
> 2024-04-05 16:05:44.960 UTC INFO [rg1-rn3] JE: Clean file 0x27:
> predicted min util is below minUtilization, current util min: 39 max:
> 39,
> predicted util min: 39 max: 39, chose file with util min: 30 max: 30
> avg: 30
> ```

Increasing Storage Directory Size

To increase the storage directory size in one or more Replication Nodes, open the CLI and execute the following commands:

1. Disable write operations on the store or on the failed shard.

   ```
   plan enable-requests -request-type READONLY \
   {-shard <shardId[,shardId]*> | -store}
   ```

   Here, `-request-type READONLY` is the option which disables write operations on a shard. You can disable write operations on one or more shards by using the `-shard` option, or on the entire store by using the `–store` option.

   > **Note:**
   >
   > Though Replication Nodes are already in non-write availability mode whenever they hit an out of disk limit exception, it is important to disable user write operations explicitly. Disabling the user write operations ensures that the Replication Nodes are brought back up in the correct manner.

2. Execute the `PING` command to analyze the state of one or more Replication Nodes.

   ```
   kv-> ping
   ```

   Usually, when Replication Nodes hit an out of disk limit exception, Replica Replication Nodes are in the `RUNNING, UNKNOWN` state, and Master Replication Nodes are in the `RUNNING, MASTER (non-authoritative)` state.

3. To display the current, deployed topology, execute the `show topology –verbose` command. Make note of the current storage directory size allocated to each Replication Node.

   ```
   show topology -verbose [-zn] [-rn] [-an] [-sn] [-store] [-status] [-json]
   ```

4. To ensure that other Replication nodes in the store do not hit a disk limit exception while you increase the `storagedirsize`, reduce the JE free disk space on all Replication Nodes to 2 GB or 3 GB. You can use the `-all-rns` option to reduce the JE free disk space on all Replication Nodes at once, or the `-service -rgx-rgy` option to reduce the free disk space on a specific Replication Node.

   ```
   kv-> plan change-parameters [-all-rns|-service -rgx-rgy] \
   -params "configProperties=je.freeDisk=XXX"
   ```

   After executing this command with either option, the system will stop the Replication Nodes, update parameters, and restart Replication Nodes with the JE free disk space parameter you specify.

5. To increase the storage directory size on one or more Replication Nodes.

   ```
   kv-> plan change-storagedir -wait -sn snX \
   -storagedir <storagedirpath> –add -storagedirsize X_GB
   ```

Here `snX` is the Storage Node whose directory size you want to increase, and X is the new storage size in GB.

6. After the `plan change-parameters` command executes successfully, verify the new `storagedirsize` value is assigned to one or more Replication Nodes in the store.

```
show topology -verbose [-zn] [-rn] [-an] [-sn] [-store] [-status] [-json]
```

7. Lastly, reset the JE free disk space back to 5 GB. Also, enable write operations back on the store or a specific shard.

```
kv-> plan change-parameters [-all-rns|-service -rgx-rgy] \
-params "configProperties=je.freeDisk=5368709120"
```

```
kv-> plan enable-requests -request-type ALL {-shard <shardId[,shardId]*> |
-store}
```

The `-request-type ALL` option re-enables write operations on the store or on a specific shard.

Example

Let us consider a store with 1x3 topology, hitting a disk limit exception. Perform the following steps to increase the storage directory size of all Replication Nodes in the store from 16 GB to 25 GB.

1. Stop the write operations on the store level:

```
kv-> plan enable-requests -request-type READONLY -store;
```

2. Ping the store to analyze the state of one or more Replication Nodes.

```
kv-> ping
```

Output:

```
Pinging components of store istore based upon topology sequence #11
3 partitions and 3 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
Shard Status: healthy:0 writable-degraded:0 read-only:1 offline:0 total:1
Admin Status: healthy
Zone [name=dc1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
RN Status: online:1 offline:2   Storage Node [sn1] on node21:port1
Zone: [name=dc1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id:
a72484b8b33c
        Admin [admin1]          Status: RUNNING,MASTER
        Rep Node [rg1-rn1]      Status: RUNNING,MASTER (non-authoritative)
        sequenceNumber:27,447,667 haPort:5011 available storage size:12 GB
Storage Node [sn2] on node22:port1
Zone: [name=dc1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
```

```
Status: RUNNING   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id:
a72484b8b33c
        Rep Node [rg1-rn2]      Status: RUNNING,UNKNOWN
        sequenceNumber:27,447,667 haPort:5010 available storage size:10 GB
delayMillis:? catchupTimeSecs:?
Storage Node [sn3] on node23:port1
Zone: [name=dc1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id:
a72484b8b33c
        Rep Node [rg1-rn3]      Status: RUNNING,UNKNOWN
        sequenceNumber:27,447,667 haPort:5010 available storage size:9 GB
delayMillis:? catchupTimeSecs:?
```

The example shows that the Replication Nodes are in `RUNNING, UNKNOWN` state and Master Replication Node is in `RUNNING, MASTER(non-authoritative)` state.

3. View the current, deployed topology.

```
kv-> show topology -verbose
```

Output:

```
store=istore  numPartitions=3 sequence=11
  zn: id=zn1 name=dc1 repFactor=3 type=PRIMARY allowArbiters=false \
  masterAffinity=false

  sn=[sn1] zn:[id=zn1 name=dc1] node21:port1 capacity=1 RUNNING
    [rg1-rn1] RUNNING  /scratch/kvroot 16 GB
                  single-op avg latency=36.866146 ms   multi-op avg
latency=0.0 ms
    [rg1-rn1] RUNNING  /scratch/kvroot 16 GB
                  single-op avg latency=36.866146 ms   multi-op avg
latency=0.0 ms
  sn=[sn2] zn:[id=zn1 name=dc1] node22:port1 capacity=1 RUNNING
    [rg1-rn2] RUNNING  /scratch/kvroot 16 GB
                  single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
    [rg1-rn2] RUNNING  /scratch/kvroot 16 GB
                  single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
  sn=[sn3] zn:[id=zn1 name=dc1] node23:port1 capacity=1 RUNNING
    [rg1-rn3] RUNNING  /scratch/kvroot 16 GB
                  single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
    [rg1-rn3] RUNNING  /scratch/kvroot 16 GB
                  single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms

  numShards=1
  shard=[rg1] num partitions=3
    [rg1-rn1] sn=sn1 haPort=node21:port2
    [rg1-rn2] sn=sn2 haPort=node22:port3
    [rg1-rn3] sn=sn3 haPort=node23:port3
    partitions=1-3
```

You see that 16 GB of disk space is assigned as the storage directory size for each Replication Node.

4. Reduce the JE free disk space from 5 GB to 2 GB for all Replication Nodes in the store.

```
kv-> plan change-parameters -all-rns -params \
"configProperties=je.freeDisk=2147483648";
Started plan 70. Use show plan -id 70 to check status.
To wait for completion, use plan wait -id 70
```

5. For each Replication Node, increase the storage directory size to 25 GB.

```
kv-> plan change-storagedir -wait -sn sn1 -storagedir /scratch/kvroot \
-add -storagedirsize 25_GB -wait
Executed plan 72, waiting for completion...
Plan 72 ended successfully


kv-> plan change-storagedir -wait -sn sn2 -storagedir /scratch/kvroot \
-add -storagedirsize 25_GB -wait
Executed plan 73, waiting for completion...
Plan 73 ended successfully


kv-> plan change-storagedir -wait -sn sn3 -storagedir /scratch/kvroot \
-add -storagedirsize 25_GB -wait
Executed plan 74, waiting for completion...
Plan 74 ended successfully
```

6. View the topology again to verify that the new value is assigned to `storagedirsize`.

```
kv-> show topology -verbose
```

Output:

```
store=istore  numPartitions=3 sequence=11
  zn: id=zn1 name=dc1 repFactor=3 type=PRIMARY allowArbiters=false \
masterAffinity=false

  sn=[sn1] zn:[id=zn1 name=dc1] node21:port1 capacity=1 RUNNING
    [rg1-rn1] RUNNING  /scratch/kvroot 25 GB
                  single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
  sn=[sn2] zn:[id=zn1 name=dc1] node22:port1 capacity=1 RUNNING
    [rg1-rn2] RUNNING  /scratch/kvroot 25 GB
                  single-op avg latency=552.51996 ms   multi-op avg
latency=0.0 ms
  sn=[sn3] zn:[id=zn1 name=dc1] node23:port1 capacity=1 RUNNING
    [rg1-rn3] RUNNING  /scratch/kvroot 25 GB
                  single-op avg latency=14.317171 ms   multi-op avg
latency=0.0 ms

  numShards=1
  shard=[rg1] num partitions=3
    [rg1-rn1] sn=sn1 haPort=node21:port2
    [rg1-rn2] sn=sn2 haPort=node22:port3
    [rg1-rn3] sn=sn3 haPort=node23:port3
    partitions=1-3
```

The example now shows that 25 GB is assigned as the storage directory size for each Replication Node.

7. Reset the JE free disk space to 5 GB and enable write operations back on the store.

```
kv-> plan change-parameters [-all-rns|-service -rgx-rgy] \
-params "configProperties=je.freeDisk=5368709120"
```

Output:

```
kv-> plan enable-requests -request-type READONLY -store;
```

Adding a New Shard

Apart from increasing the storage directory size, you can also handle disk limit exceptions by adding a new shard and expanding your store.

The following example demonstrates adding three new Storage Nodes (Storage Nodes 21, 22, and 23) and deploying the new store to recover from disk limit exception:

1. Disable write operations on the store.

```
kv-> plan enable-requests -request-type READONLY -store;
```

Here, `-request-type READONLY` disables write operations on a store and allows only read operations.

2. Reduce the JE free disk space to 2 GB on all nodes and increase the `je.cleaner.minUtilization` configuration parameter from 40 (the default in a KVStore) to 60.

```
kv-> plan change-parameters -all-rns \
-params "configProperties=je.cleaner.minUtilization 60; \
je.freeDisk 2147483648";
```

Executing this command creates more free space for store expansion. Replication Nodes will be stopped, parameters will be updated, and the Replication Nodes will be restarted with the new parameters.

3. Create, start, and configure the new nodes for expanding the store.
   - Create the new node. Run the `makebookconfig` utility to configure each Storage Node in the store:

```
java -Xmx256m -Xms256m -jar KVHOME/kvstore.jar makebootconfig \
-root sn1/KVROOT \
-store-security none -capacity 1 \
-port port1 -host node21 \
-harange 5010,5020 \
-storagedir /scratch/sn1/u01 -storagedirsize 20-Gb


java -Xmx256m -Xms256m -jar KVHOME/kvstore.jar makebootconfig \
-root sn2/KVROOT \
-store-security none -capacity 1 \
-port port1 -host node22 \
```

```
-harange 5010,5020 \
-storagedir /scratch/sn2/u01 -storagedirsize 20-Gb


java -Xmx256m -Xms256m -jar KVHOME/kvstore.jar makebootconfig \
-root sn3/KVROOT \
-store-security none -capacity 1 \
-port port1 -host node23 \
-harange 5010,5020 \
-storagedir /scratch/sn3/u01 -storagedirsize 20-Gb
```

- Restart the Storage Node Agent (SNA) on each of the Oracle NoSQL Database nodes using the `start` utility:

```
kv-> nohup java -Xmx256m -Xms256m -jar \
KVHOME/lib/kvstore.jar start -root KVROOT &
```

- Configure the new store:

```
java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar runadmin \
-port port1 -host node21


java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar runadmin \
-port port1 -host node22


java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar runadmin \
-port port1 -host node23
```

4. Redistribute the store according to its new configuration.

```
kv-> java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar runadmin \
-port port1 -host host1


kv-> plan deploy-sn -zn zn1 -host node21 -port port1 -wait
Executed plan 7, waiting for completion...
Plan 7 ended successfully
kv-> plan deploy-sn -zn zn1 -host node22 -port port1 -wait
Executed plan 8, waiting for completion...
Plan 8 ended successfully
kv-> plan deploy-sn -zn zn1 -host node23 -port port1 -wait
Executed plan 9, waiting for completion...
Plan 9 ended successfully
Plan 11 ended successfully


kv-> pool join -name ExamplePool -sn sn4
Added Storage Node(s) [sn4] to pool ExamplePool
kv-> pool join -name ExamplePool -sn sn5
Added Storage Node(s) [sn5] to pool ExamplePool
```

```
kv-> pool join -name ExamplePool -sn sn6
Added Storage Node(s) [sn6] to pool ExamplePool


kv-> topology clone -current -name newTopo
Created newTopo


kv-> topology redistribute -name newTopo -pool ExamplePool
Redistributed: newTopo


kv-> plan deploy-topology -name newTopo -wait
Executed plan 11, waiting for completion...
```

5. Restore the Replication Nodes to its original configuration.

```
plan change-parameters -all-rns \
-params "configProperties=je.cleaner.minUtilization 40; \
je.freeDisk 5368709120";
```

6. Enable write operations back on the store.

```
kv-> plan enable-requests -request-type ALL -store;
```

Here, `–request-type ALL` enables both read and write operations on the store.

Managing Admin Directory Size

You should specify a sufficient directory size for the Admin database when you initially install your store, using the `makebootconfig admindirsize` parameter. If you do not specify a value, the system allocates a default of 3 GB as the size of the Admin directory. See Configuring your data store installation and makebootconfig for details.

Specify the value for the `-admindirsize` parameter as a long, optionally followed by a unit string. Accepted unit strings are: KB, MB, and GB, corresponding to 1024, 1024^2, and 1024^3 respectively. Acceptable strings are case insensitive. Valid delimiters between the long value and the unit string are " ", "-", or "_". If you specify the delimiter as " ", your value should be enclosed in double quotes, For example "10 GB". If you have any other delimiter double quotes is not mandatory. For example 10_GB or 10-GB.

Also, if the admin directory fills up its allotted storage space with reserved files, see Managing Disk Thresholds for more information.

If the Admin completely uses up its storage space, it will not be able to start. This condition is unlikely to occur, but in the event that your Admin cannot start, you should check its available disk space. If the directory is full, then you should increase the available disk space to the Admin. For the Admin to completely fill its storage space with actual data files, the store would have to be configured in some unexpected way — such as with an extraordinarily large number of tables, or have been allotted a very small Admin directory size.

The procedure that you use to change an Admin's allocated disk space differs depending on whether the Admin is in working condition.

Admin is Working

To increase or decrease the Admin's disk space when the Admin is functional, use the CLI to execute the following plan:

```
plan change-parameters -all-admins -params \
"configProperties=je.maxDisk=<size>"
```

where <size> is the desired storage size in bytes.

Admin is not Working

To increase or decrease the Admin's disk space when the Admin is *not* functional:

1. Set the value of `je.maxDisk` to the desired value in `config.xml` for all Admins manually:
   a. For each Storage Node that is hosting an Admin, locate the `config.xml` file in the Storage Node's root directory:

      ```
      <kvroot dir>/<store name>/<SN name>/config.xml
      ```

      and edit it as follows.
   b. Locate the admin section of the `config.xml` file. This is the section that begins with:

      ```
      <component name="ADMIN-NAME" type="adminParams" validate="true">
          ...
      </component>
      ```

   c. Add the following line to the admin section of each `config.xml` file:

      ```
      <propertyname="configProperties" value="je.maxDisk=<size>"
      type="STRING"/>
      ```

      where <size> is the desired storage size in bytes for your Admin.
2. Stop/start these Storage Nodes one by one, using the following commands:

   ```
   java -Xmx64m -Xms64m \
   -jar kvstore.jar stop -root <root dir> \
   -config <config file name>

   java -Xmx64m -Xms64m \
   -jar kvstore.jar start -root <root dir> \
   -config <config file name>
   ```

3. Wait for the status of these Storage Nodes to change to RUNNING. You can use the ping command to get the Storage Node status:

   ```
   java -Xmx64m -Xms64m \
   -jar kvstore.jar runadmin -host <host name> -port <port> ping
   ```

4. If any Admins are unreachable (you cannot get a response using the ping command), start them from the CLI using the following command:

   ```
   kv-> plan start-service -service <ADMIN_NAME> -wait
   ```

5. Once all the Admins are running, execute the following command using the CLI:

   ```
   plan change-parameters -all-admins -params \
   "configProperties=je.maxDisk=<size>"
   ```

   where <size> is the desired storage size in bytes for your Admin. This value should match the value you provided in the `config.xml` file.

Disabling Storage Node Agent Hosted Services

To disable all services associated with a stopped SNA use the `-disable-services` flag. This helps isolate failed services to avoid hard rollbacks during a failover. Also, in this way, the configuration can be updated during recovery after a failover. The usage is:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar {start | stop | restart}
[-disable-services] [-verbose]
-root KVROOT [-config <bootstrapFileName>]
```

where:

- `start -disable-services`

  Starts an Oracle NoSQL Database Storage Node Agent with all of its hosted services disabled. If the SNA is already running, the command will fail.

- `stop -disable-services`

  Stops an Oracle NoSQL Database Storage Node Agent, marking all of its services disabled so that they will not start when starting up the SNA in the future or until the services are reenabled.

- `restart -disable-services`

  Restarts an Oracle NoSQL Database Storage Node Agent with all of its hosted services disabled.

Verifying the Store

Use the Admin CLI verify command to complete these tasks:

- Perform general troubleshooting of the store.

  The `verify` command inspects all store components. It also checks whether all store services are available. For the available store services, the command also checks for any version or metadata mismatches.

- Check the status of a long-running plan

  Some plans require many steps and may take some time to execute. The administrator can verify plans to check on the plan progress. For example, you can verify a `plan deploy-sn` command while it is running against many Storage Nodes. The `verify`

command can report at each iteration to confirm that additional nodes have been created and come online.

For more about managing plans, see Using Plans.

- Get additional information to help diagnose a plan in an ERROR state.

You verify your store using the `verify` command in the CLI. The command requires no parameters, and runs in verbose mode, by default. For example:

```
kv-> verify configuration
```

Output:

```
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
See node01:Data/virtualroot/datacenter1/kvroot/MetroArea/
                                          log/MetroArea_{0..N}.log for
                                          progress messages
Verify: Shard Status: healthy:2 writable-degraded:0
                                            read-only:0 offline:0
Verify: Admin Status: healthy
Verify: Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
   RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
   RN Status: online:2 offline: 0
Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on node01:5000
   Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin1]          Status: RUNNING,MASTER
Verify:         Rep Node [rg1-rn2]      Status: RUNNING,REPLICA
   sequenceNumber:127 haPort:5011 available storage size:14 GB delayMillis:1
catchupTimeSecs:0
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on node02:6000
   Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn2]      Status: RUNNING,REPLICA
   sequenceNumber:127 haPort:6010 available storage size:24 GB delayMillis:1
catchupTimeSecs:0
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:7000
   Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
```

```
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin2]          Status: RUNNING,REPLICA
Verify:         Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
   sequenceNumber:127 haPort:7011 available storage size:22 GB delayMillis:1
catchupTimeSecs:0
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:8000
   Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn3]      Status: RUNNING,REPLICA
   sequenceNumber:127 haPort:8010 available storage size:24 GB delayMillis:1
catchupTimeSecs:0
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:9000
   Zone: [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin3]          Status: RUNNING,REPLICA
Verify:         Rep Node [rg1-rn1]      Status: RUNNING,MASTER
   sequenceNumber:127 haPort:9011 available storage size:18 GB
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:10000
   Zone: [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn1]      Status: RUNNING,MASTER
   sequenceNumber:127 haPort:10010 available storage size:16 GB

Verification complete, no violations.
```

Use the optional `-silent` mode to show only problems or completion.

```
kv-> verify configuration -silent
```

Output:

```
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
See node01:Data/virtualroot/datacenter1/kvroot/MetroArea/
                           log/MetroArea_{0..N}.log for progress messages
Verification complete, no violations.
```

The `verify` command clearly reports any problems with the store. For example, if a Storage Node is unavailable, using `-silent` mode displays that problem as follows:

```
kv-> verify configuration -silent
```

Output:

```
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
See node01:Data/virtualroot/datacenter1/kvroot/MetroArea/
                           log/MetroArea_{0..N}.log for progress messages
Verification complete, 2 violations, 0 notes found.
Verification violation: [rg2-rn2]       ping() failed for rg2-rn2 :
Unable to connect to the storage node agent at host node02, port 6000,
which may not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
        nested exception is:
        java.net.ConnectException: Connection refused
Verification violation: [sn2]   ping() failed for sn2 : Unable to connect
        to the storage node agent at host node02, port 6000,
        which may not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
        nested exception is:
        java.net.ConnectException: Connection refused
```

Using the default mode (verbose), `verify configuration` shows the same problem as follows:

```
kv-> verify configuration
```

Output:

```
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
See node01:Data/virtualroot/datacenter1/kvroot/MetroArea/
                           log/MetroArea_{0..N}.log for progress messages
Verify: Shard Status: healthy:1 writable-degraded:1
                                                read-only:0 offline:0
Verify: Admin Status: healthy
Verify: Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   RN Status: online:1 offline: 1 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
   RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
   RN Status: online:2 offline: 0
Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on node01:5000
   Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   Status: RUNNING
   Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin1]          Status: RUNNING,MASTER
Verify:         Rep Node [rg1-rn2]      Status: RUNNING,REPLICA
```

```
   sequenceNumber:127 haPort:5011 available storage size:18 GB delayMillis:1
catchupTimeSecs:0
Verify: == checking storage node sn2 ==
Verify:         sn2: ping() failed for sn2 :
Unable to connect to the storage node agent at host node02, port 6000,
which may not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
        nested exception is:
        java.net.ConnectException: Connection refused
Verify: Storage Node [sn2] on node02:6000
   Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
   UNREACHABLE
Verify:         rg2-rn2: ping() failed for rg2-rn2 :
Unable to connect to the storage node agent at host node02, port 6000,
which may not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
        nested exception is:
        java.net.ConnectException: Connection refused
Verify:         Rep Node [rg2-rn2]      Status: UNREACHABLE
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:7000
  Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
  Status: RUNNING
  Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin2]          Status: RUNNING,REPLICA
Verify:         Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
  sequenceNumber:127 haPort:7011 available storage size:12 GB delayMillis:1
catchupTimeSecs:0
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:8000
  Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
  Status: RUNNING
  Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn3]      Status: RUNNING,REPLICA
  sequenceNumber:127 haPort:8010 available storage size:11 GB delayMillis:0
catchupTimeSecs:0
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:9000
  Zone: [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
  Status: RUNNING
  Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Admin [admin3]          Status: RUNNING,REPLICA
Verify:         Rep Node [rg1-rn1]      Status: RUNNING,MASTER
  sequenceNumber:127 haPort:9011 available storage size:14 GB
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:10000
  Zone: [name=Queens id=zn3 type=PRIMARY allowArbiters=false
masterAffinity=false]
  Status: RUNNING
  Ver: 24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Verify:         Rep Node [rg2-rn1]      Status: RUNNING,MASTER
  sequenceNumber:127 haPort:10010 available storage size:16 GB
```

```
Verification complete, 2 violations, 0 notes found.
Verification violation: [rg2-rn2]       ping() failed for rg2-rn2 :
Unable to connect to the storage node agent at host node02, port 6000,
which may not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
        nested exception is:
        java.net.ConnectException: Connection refused
Verification violation: [sn2]   ping() failed for sn2 :
Unable to connectto the storage node agent at host node02, port 6000,
which may not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: node02;
        nested exception is:
        java.net.ConnectException: Connection refused
```

Note:

The verify output is only displayed in the shell after the command is complete. Use `tail`, or `grep` the Oracle NoSQL Database log file to get a sense of how the verification is progressing. Look for the string `Verify`. For example:

```
grep Verify /KVRT1/mystore/log/mystore_0.log
```

Violations and Solutions Table

The table below lists some violation or verification notes and their solutions that may arise when running the verify configuration command.

Table 4-1 Violations and Solutions Table

Violation / Verification Note	Solution
The zone needs more Admins to meet the required Replication Factor.	Add more Admins in a particular zone using plan deploy-admin command
There are fewer Replication Nodes than expected by the Replication Factor.	• Add more Storage Nodes using plan deploy-sn command. or • Increase the capacity of existing Storage Nodes using plan change-parameters command.
The zone is empty.	Remove the zone manually using plan remove-zone command.
The zone has excess Admins than required by the Replication Factor.	Remove the Admins that are not needed using plan remove-admin command.
The storage size is not defined for the root directory of a Storage Node.	Add or change the storage directory size using plan change-storagedir command.
The storage size is not defined for the storage directory of a Storage Node.	
The Replication Nodes in a particular shard have significantly different sizes.	

Table 4-1 (Cont.) Violations and Solutions Table

Violation / Verification Note	Solution
More than one Replication Node is present in the root directory of the Storage Node.	Create and specify storage directory size for individual Replication Nodes using plan change-storagedir command.

> **Note:**
>
> The first three points from the table above are violations, and the others are the verification notes.

Erasing Data

The Oracle NoSQL Database has a built-in erasure feature that can be run in the background to erase user data after it has become obsolete (i.e. deleted data, expired data, older versions of updated records). The goal of this background thread is to erase the data with very little impact on application performance. The background thread takes a best effort approach. If a replication node is down, no erasure will occur until it comes back up.

Erasure happens over a user defined erasure time period (its cycle) and it is recommended that the cycle time be long enough for all the erasure work to complete. Any work not completed will be carried over to the next cycle. Data is erased from the tables and corresponding indexes. The Replication Node parameters: "enableErasure", and "erasurePeriod" control the behavior of erasure. For more information on setting these parameters, see Setting Store Wide Policy Parameters.

Setting Store Parameters

The three Oracle NoSQL Database service types, Admin, Storage Node, and Replication Node, have configuration parameters. You can modify some parameters after deploying the service. Use the following Admin CLI command to see the parameter values that you can change:

```
show parameters -service <>
```

You identify an Admin, Storage Node, or Replication service using a valid string. The `show parameters -service` command displays service parameters and state for any of the three services. Use the optional `-policy` flag to show global policy parameters.

Changing Parameters

All of the CLI commands used for creating parameter-changing plans share a similar syntax:

```
plan change-parameters -service <id>...
```

All such commands can have multiple `ParameterName=NewValue` assignment arguments on the same command line. If NewValue contains spaces, then the entire assignment argument must

be quoted within double quote marks. For example, to change the Admin parameter collectorPollPeriod, you would issue the command:

```
kv-> plan change-parameters -all-admins -params \
    "collectorPollPeriod=20 SECONDS">
```

If your configProperties for all Replication Nodes is set to:

```
"configProperties=je.cleaner.minUtilization=40;">
```

And you want to add new settings for configProperties, you would issue the following command:

```
kv-> plan change-parameters -all-rns -params \
     "configProperties=je.cleaner.minUtilization=40;\
     je.env.runVerifier=false;">
```

If for some reason, different Replication Nodes have different configProperties parameter values, then the change-parameters command will need to be tailored for each Replication Node.

The following commands are used to change service parameters:

- `plan change-parameters -service <shardId-nodeId> -params [assignments]`

  This command is used to change the parameters of a single Replication Node, which must be identified using the shard and node numbers. The `shardId-nodeId` identifier must be given as a single argument with one embedded hyphen and no spaces. The `shardId` identifier is represented by `rgX`, where `X` refers to the shard number.

- `plan change-parameters -all-rns -params [assignments]`

  This command is used to change the parameters of all Replication Nodes in a store. No Replication Node identifier is needed in this case.

- `plan change-parameters -service <storageNodeId> -params [assignments]`

  This command is used to change the parameters of a single Storage Node instance. The storageNodeId is a simple integer.

- `plan change-parameters -all-admins -params [assignments]`

  This command is used to change Admin parameters. Because each instance of Admin is part of the same replicated service, all instances of the Admin are changed at the same time, so no Admin identifier is needed in this command.

  If an Admin parameter change requires the restarting of the Admin service, KVAdmin loses its connection to the server. Under normal circumstances, KVAdmin automatically reconnects after a brief pause, when the next command is given. At this point the plan is in the `INTERRUPTED` state, and must be completed manually by issuing the `plan execute` command.

- `plan change-parameters -security <id>`

  This command is used to change security parameters. The parameters are applied implicitly and uniformly across all SNs, RNs and Admins.

In all cases, you can choose to create a plan and execute it; or to create the plan and execute it in separate steps by using the `-noexecute` option of the plan command.

Setting Store Wide Policy Parameters

Most admin, Storage Node, and replication node parameters are assigned to default values when a store is deployed. It can be inconvenient to adjust them after deployment, so Oracle NoSQL Database provides a way to set the defaults that are used during deployment. These defaults are called store-wide Policy parameters.

You can set policy parameters in the CLI by using this command:

```
change-policy -params [name=value]
```

The parameters to change follow the `-params` flag and are separated by spaces. Parameter values with embedded spaces must be separated by spaces. Parameter values with embedded spaces must be quoted. For example: name = "value with spaces". If the optional `dry-run` flag is specified, the new parameters are returned without changing them.

Admin Parameters

You can set the following parameters for the Admin service:

- `collectorPollPeriod=<Long TimeUnit>`

  Sets the Monitor subsystem's delay for polling the various services for status updates. This value defaults to "20" seconds. Units are supplied as a string in the `change-parameters` command, for example: -params collectorPollPeriod="2 MINUTES"

- `loggingConfigProps=<String>`

  Property settings for the Logging subsystem in the Admin process. Its format is `property=value;property=value...`. Standard java.util.logging properties can be set by this parameter.

- `eventExpiryAge=<Long TimeUnit>`

  You can use this parameter to adjust how long the Admin stores critical event history. The default value is "30 DAYS".

- `configProperties=<String>`

  This is an omnibus string of property settings for the underlying BDB JE subsystem. Its format is `property=value;property=value...`.

- `javaMiscParams=<String>` [deprecated]

  This parameter should ONLY be used to set flags for which there are no service parameters.

  This parameter is **deprecated**. You are encouraged not to use it, and instead use the `javaAdminParamsOverride` parameter.

- `javaAdminParamsOverride=<String>`

  This is an omnibus string that is added to the command line when the Admin process is started. This parameter is intended for specifying miscellaneous JVM properties that cannot be specified using other Admin parameters. If the string is not a valid sequence of tokens for the JVM command line, the Admin process fails to start.

  No default value is provided for this parameter.

Changing Admin JVM Memory Parameters

Admin processes can run out of memory. One of the most likely reasons is that the default memory setting was insufficient for the Admin services to represent all of the metadata associated with the store. Metadata includes information about tables, security information about users and roles, and information about incomplete plans. Stores with large amounts of metadata may need to increase the memory setting for Admin services if the activity logs show that Admin services are failing with `OutOfMemoryError`. This topic describes increasing the memory setting of the `javaAdminParamsOverride`.

The system continues to use the old `javaMiscParams` setting to specify the initial JVM memory settings for the admin. The system does not use `javaAdminParamsOverride` since it is reserved for use if you want to override the default settings.

To change the `javaAdminParamsOverride` requires a comprehensive all or nothing change. You cannot change individual parameters within the set. To change any setting, declare them all in the `plan change-parameters` command, described next.

First determine the basic information about all Admin services using the `show admins` command. You get the output as shown below.

```
show admins
```

Output:

```
admin1: Storage Node sn1 storageDir=/home/opc/nosql/kvroot type=PRIMARY
(connected RUNNING,MASTER)
admin2: Storage Node sn2 storageDir=/home/opc/nosql/kvroot type=PRIMARY
(RUNNING,REPLICA)
admin3: Storage Node sn3 storageDir=/home/opc/nosql/kvroot type=PRIMARY
(RUNNING,REPLICA)
```

> **Note:**
>
> The above output is just an example. Here the replication factor RF=3 for the primary nodes. Your output will reflect your topology.

To determine the current settings of `javaAdminParamsOverride` and `configProperties`, enter the Admin CLI `show parameters -service admin` command as follows:

```
kv-> show parameters -service admin1
```

Output:

```
adminId=1
adminLogFileCount=20
adminLogFileLimit=5242880
adminMountPoint=/home/opc/nosql/kvroot
collectEnvStats=true
collectorPollPeriod=20 SECONDS
disabled=false
```

```
eventExpiryAge=30 DAYS
hideUserData=true
javaAdminParamsOverride=
loggingConfigProps=com.sleepycat.je.util.FileHandler.level=OFF
maxEvents=10000
storageNodeId=1
```

In this example, the `javaAdminParamsOverride` parameters that specify the Admin JVM memory shows the default value.

```
javaAdminParamsOverride=
```

> **Note:**
>
> The value may or may not have a default value. The value in your setup could also be different.

To increase Admin JVM memory when Admins are operational, use the `plan change-parameters` command from the Admin CLI, as follows:

```
kv-> plan change-parameters -wait -all-admins  -params \
javaAdminParamsOverride="-Xms2048m -Xmx2048m
```

> **Note:**
>
> It is recommended that you apply the modification to all of the admins by using the `all-admins` option. However, you can modify the admin parameters of every primary admin individually, but only if you have three or more primary admins. When you have only two admins, you must use the `all-admins` parameter; otherwise, the restart needed during this operation will cause the admins to lose the quorum.

Specifying these new values changes the Java heap size from the default values to 2 GB for both as shown below.

```
kv-> show parameters -service admin1
```

Output:

```
adminId=1
adminLogFileCount=20
adminLogFileLimit=5242880
adminMountPoint=/home/opc/nosql/kvroot
collectEnvStats=true
collectorPollPeriod=20 SECONDS
disabled=false
eventExpiryAge=30 DAYS
hideUserData=true
javaAdminParamsOverride=-Xms2048m -Xmx2048m
loggingConfigProps=com.sleepycat.je.util.FileHandler.level=OFF
```

```
maxEvents=10000
storageNodeId=1
```

Make sure that you locate the existing `javaAdminParamsOverride` from the Admin CLI as shown above, and update the individual entries. The `javaAdminParamsOverride` setting must represent all desired flags, not just new ones, so be sure to include any previously existing flag values that you want to retain.

If the Admin loses quorum, then you must use the Admin CLI `repair-admin-quorum` command.

Storage Node Parameters

You can set the following Storage Node parameters:

- `capacity=<Integer>`

  Sets the number of Replication Nodes that this Storage Node can host. This value informs decisions about where to place new Replication Nodes. The default value is `1`. You can set the capacity level to greater than `1` if the Storage Node has sufficient disk, CPU, and memory resources to support multiple Replication Nodes.

  Setting the Storage Node capacity to 0 indicates that the Storage Node can be used to host Arbiter Nodes. The pool of Storage Nodes in a zone configured to host Arbiter Nodes is used for Arbiter Node allocation. See Deploying an Arbiter Node Enabled Topology.

- `jvmOverheadPercent=<Integer>`

  Sets the percentage of Java heap size, for additional memory used by JVM overhead. Default value: 25. In standard memory allocation, 85% of the SN's memory is for Java heap and JVM overhead: 68% for Java heap (`rnHeapPercent`), 25% (`jvmOverheadPercent`) * 68 (`rnHeapPercent`) = 17% for JVM overhead, and 68% + 17% = 85%.

- `memoryMB=<Integer>`

  Sets the amount of memory (in megabytes) available on this Storage Node. The default value is `0`, which indicates that the amount of memory is `unknown`. The store determines the amount of memory automatically as the total amount of RAM available on the machine.

  You should not need to change this parameter. If the machine has other applications running on it, reserve some memory for those applications, and set the `memoryMB` parameter value with a memory allowance for application needs. Having other applications running on a Storage Node is not a recommended configuration.

- `mgmtClass=<String>`

  The name of the class that provides the Management Agent implementation. See Standardized Monitoring Interfaces. The port cannot be a privileged port number (<1024).

- `numCPUs=<Integer>`

  Sets the number of CPUs known to be available on this Storage Node. Default value: 1.

- `rnHeapMaxMB=<Integer>`

  Sets a hard limit for the maximum size of the Replication Node's Java VM heap. The default value is `0`, which means the VM-specific limit is used. The default is roughly 32 GB, which represents the largest heap size that can make use of compressed object references.

  Do not set this value to greater than 32 GB. Doing so can adversely impact your Replication Node's performance.

Settings larger than the maximum size that supports compressed object references will maintain the default limit unless the size is large enough that the heap can reference a larger number of objects given the increased memory requirements for uncompressed object references. Using larger heap sizes is not recommended.

- `rnHeapPercent=<Integer>`

  Sets the percentage of a Storage Node's memory reserved for heap space for all RN processes that the SN hosts. Default value: `68`.

- `rootDirPath=<path>`

  The path to the Storage Node's root directory.

- `rootDirSize=<Long Unit_String>`

  Sets the storage size of the root directory. However, no run-time checks are performed to verify that the actual directory size is greater than or equal to the size you specify. Use this setting for heterogeneous installation environments where some Storage Nodes have more disk capacity than others. Then, use this parameter only for those Storage Nodes that store data in the root directory (not recommended).

  The value that you specify for this parameter must be a long, followed optionally by a unit string. Accepted unit strings are: KB, MB, GB, and TB, corresponding to 1024, 1024^2, 1024^3, 1024^4, respectively. Acceptable strings are case insensitive. Valid delimiters between the long value and the unit string are " ", "-", or "_".

  > **Note:**
  >
  > The `rootDirSize` parameter is intended for backward compatibility with older installations that were created without specifying the `-storagedir` parameter. We strongly recommend not storing data in your root directory. See Managing Storage Directory Sizes. However, if you do specify a `-rootDirPath` parameter, you must also specify `-rootDirSize`. If you are trying to change parameter settings (`plan change-parameters`), and do not specify both parameters, a warning is displayed.

  Do not use the `rootDir` parameter if a Storage Nodes uses some other directory (such as you can specify using `plan change-storagedir`).

The following store-wide parameter settings apply to statistics files and performance files, as well as the service debug logs across all Storage Nodes, Replication Nodes, Admins, and Arbiters. The associated Storage Node Agent must be restarted to reflect any changes in the settings.

- `serviceLogFileCount=<Integer>`

  Sets the number of log files kept by this Storage Node, and for all the Replication Nodes it hosts. This default value is `20`. Limiting the number of log files controls the amount of disk space devoted to logging history. If the value is less than 1, then it is converted to 1.

- `serviceLogFileLimit=<Integer>`

  Limits the size of each log file. After reaching this size, the logging subsystem starts a new log file. This setting applies to the Storage Node and to all Replication Nodes that it hosts. The default value is `2,000,000` bytes. The limit specifies an approximate maximum amount of bytes written to any one file. If the value is lesser than or equal to 0, then there is no limit to the size of the service log files.

- `serviceLogFileCompression=<Boolean>`

  Enables the compression of the log files to store significantly more logging output in the same amount of disk space. By default, the compression is disabled. You can enable log file compression by setting the parameter to `true`.

  When you enable compression, the `adminLogFileLimit` and `serviceLogFileLimit` parameters are auto-adjusted to retain larger log files than the specified size. With the default value of the maximum file count, the actual size limit is approximately five times larger than the specified limit.

  > **Note:**
  >
  > The size of the log files may temporarily exceed the defined limits in certain cases.

- `servicePortRange=<String>`

  Sets the range of ports used for communication among administrative services running on a Storage Node and its managed services. This parameter is optional. By default the services use anonymous ports. The format of the value string is "*`startPort,endPort`*."

  The range needs to be large enough to accommodate the Storage Node, all the Replication Nodes (as defined by the capacity parameter), Admin and Arbiter services hosted on the machine, and JMX, if enabled. The number of ports required also depends on whether the system is configured for security, which is the default. For a non-secure system, the Storage Node consumes 1 port (shared with the port assigned separately for the Registry Service, if it overlaps the service port range), and each Replication Node consumes 1 port in the range. An Admin, if configured, consumes 1 port. Arbiters consume 1 port each. If JMX is enabled, that consumes 1 additional port. On a secure system, two additional ports are required for the Storage Node, and two for the Admin. As a general rule, we recommend that you specify a range significantly larger than the minimum. More available ports allows for increases in Storage Node capacity, or network problems that can render ports temporarily unavailable.

  The ports that you specify in the `servicePortRange` should not overlap with the Admin port or with `haPortRange`. The service port range can include the registry port, so the registry and Storage Node share a port.

  For deploying a secure Oracle NoSQL Database, use the following formula to estimate the port range size number, adding an additional port for each Storage Node, Replication Node or the Admin (if configured):

  ```
   3 (Storage Nodes) +
  capacity (the number of Replication Nodes) +
  Arbiters (the number of Arbiter Nodes) +
  3 (if the Storage Node is hosting an admin) +
  1 (if the Storage node is running JMX)
  ```

  For more information on configuring Oracle NoSQL Database securely, see *Security Guide*.

  For a non-secure system, use the following formula to estimate the port range size number:

  ```
  1 (Storage Node) +
  capacity (the number of Replication Nodes) +
  ```

```
Arbiters (the number of Arbiter Nodes) +
1 (if the Storage Node is hosting an admin) +
1 (if the Storage Node is running JMX)
```

For example, if a Storage Node has capacity 1, is hosting an Admin process, and neither Arbiters nor JMX are in use, the range size must be at least 3. You can increase the range size beyond this minimum, for safety and Storage Node expansion. Then, if you expand the Storage Node, you will not need to make changes to this parameter. If capacity is `2`, the range size must be greater than or equal to 4.

Replication Node Parameters

The following parameters can be set for Replication Nodes:

- `cacheSize=<Long>`

  Sets the cache size in the underlying BDB JE subsystem. The units are bytes. The size is limited by the java heap size, which in turn is limited by the amount of memory available on the machine. You should only ever change this low level parameter under explicit directions from Oracle support.

- `collectEnvStats=<Boolean>`

  If true, then the underlying BDB JE subsystem dumps statistics into the .stat file. This information is useful for tuning JE performance. Oracle Support may request these statistics to aid in tuning or to investigate a problem.

- `configProperties=<String>`

  Contains property settings for the underlying BDB JE subsystem. Its format is `property=value;property=value....`

- `enableErasure=<Boolean>`

  If true, then erasure is enabled for the underlying storage system. Erasure periodically wipes the obsolete data (i.e. delete data, older versions of updated records, expired data) from the storage layer by zeroing out the corresponding records. Erasure can be enabled or disabled without restarting the database. Erasure is enabled by default.

  Default value is true.

- `erasurePeriod=<Long Timeunit>`

  The duration for one complete erasure pass over the entire data set (the cycle time). Erasure is throttled based on this value, to minimize its impact on performance. It is recommended that erasure period be set to less than half of the duration one expects the obsoleted data to stay around. In other words, we recommend two erasure cycles to remove the obsoleted data. For example, if we intend to remove all obsolete data in 30 days, then erasure period can be set to 14 days.

  Default value is "6 DAYS".

- `javaMiscParams=<String>` [deprecated]

  This parameter should ONLY be used to set flags for which there are no service parameters.

  The `javaMiscParams` parameter is **deprecated**. You are encouraged not to use it, and instead use the `javaRnParamsOverride` parameter.

- `javaRnParamsOverride=<String>`

A string that is added to the command line when the Replication Node process is started. This parameters is intended for specifying miscellaneous JVM properties that cannot be specified using other RN parameters. If the string is not a valid sequence of tokens for the JVM command line, the Admin process fails to start.

No default value is provided for this parameter.

It is recommended that to specify the heap sizes for Replication Nodes you use Storage Node's memoryMB and other JVM parameters. For more information about these parameters see, Storage Node Parameters.

- `latencyCeiling=<Integer>`

  If the Replication Node's average latency exceeds this number of milliseconds, it is considered an "alertable" event. If JMX monitoring is enabled, the event also causes an appropriate notification to be sent.

- `loggingConfigProps=<String>`

  Contains property settings for the Logging subsystem. The format of this string is like that of configProperties, above. Standard java.util.logging properties can be set by this parameter.

- `maxTrackedLatency=<Long TimeUnit>`

  The highest latency that is included in the calculation of latency percentiles.

- `rnCachePercent=<Integer>`

  The portion of an RN's memory set aside for the JE environment cache.

- `rnStatisticsEnabled=<Boolean>`

  If true, then the Replication Nodes gather key distribution statistics.

- `rnStatisticsGatherInterval=<Long TimeUnit>`

  The time interval at which Replication Nodes should gather distribution statistics.

- `rnStatisticsTTL=<Long DaysOrHours>`

  Specifies the duration for which the key distribution statistics should be retained in the system tables. The duration specified must be in days or hours. By default, these statistics are retained for 60 days.

- `rnStatisticsIncludeStorageSize=<Boolean>`

  If true, then the information on storage sizes are included when gathering key distribution statistics.

- `throughputFloor=<Integer>`

  Similar to latencyCeiling, throughputFloor sets a lower bound on Replication Node throughput. Lower throughput reports are considered alertable. This value is given in operations per second.

Arbiter Node Parameters

The following parameters can be set for Arbiter Nodes:

`javaAnParamsOverride=<String>`

A string that is added to the command line when the Arbiter Node process is started. This parameters is intended for specifying miscellaneous JVM properties. If the string is not a valid sequence of tokens for the JVM command line, the Admin process fails to start.

No default value is provided for this parameter.

It is recommended that to specify the heap sizes for Arbiter Nodes you use Storage Node's memoryMB and other JVM parameters. For more information about these parameters see, Storage Node Parameters.

Global Parameters

The following store-wide non-security parameters can be implicitly and uniformly set across all Storage Nodes, Replication Nodes and Admins:

- `collectorInterval =<Long TimeUnit>`

  Sets the collection period for latency statistics at each component. This value defaults to 20 seconds. Values like average interval latencies and throughput are averaged over this period of time.

The following store-wide parameters can be set for the debug log files:

- `adminLogFileCount=<Integer>`

  Sets the number of log files that are kept. This value defaults to 20. It is used to control the amount of disk space devoted to logging history. If the value is less than 1, then it is converted to 1.

- `adminLogFileLimit=<Integer>`

  Limits the size of log files. After reaching this limit, the logging subsystem switches to a new log file. This value defaults to 4,000,000 bytes. The limit specifies an approximate maximum amount to write (in bytes) to any one file. If the value is lesser than or equal to 0, then there is no limit to the size of the log files.

Security Parameters

The following store-wide security parameters can be implicitly and uniformly set across all Storage Nodes, Replication Nodes and Admins:

- `accountErrorLockoutThresholdCount=<Integer>`

  Number of invalid login attempts for a user account from a particular host address over the tracking period needed to trigger an automatic account lockout for a host. The default value is 10 attempts.

- `accountErrorLockoutThresholdInterval=<Long TimeUnit>`

  Specifies the time period over which login error counts are tracked for account lockout monitoring. The default value is 10 minutes.

- `accountErrorLockoutTimeout=<Long TimeUnit>`

  Time duration for which an account will be locked out once a lockout has been triggered. The default value is 30 minutes.

- `loginCacheTimeout=<Long TimeUnit>`

  Time duration for which KVStore components cache login information locally to avoid the need to query other servers for login validation on every request. The default value is 5 minutes.

- `sessionExtendAllowed=<Boolean>`

  Indicates whether session extensions should be granted. Default value is true.

- `sessionTimeout=<Long TimeUnit>`

Specifies the length of time for which a login session is valid, unless extended. The default value is 24 hours.

The following password security parameters can be set:

Parameter Name	Value Range and Type	Description
passwordAllowedSpecial	Sub set or full set of #$ %&'()*+,-./:; <=>?@[]^_`{|} (string)~	Lists the allowed special characters.
passwordComplexityCheck	[true|false] (boolean)	Whether to enable the password complexity checking. The default value is true.
passwordMaxLength	1 - 2048 (integer)	The maximum length of a password. The default value is 256.
passwordMinDigit	0 - 2048 (integer)	The minimum required number of numeric digits. The default value is 2.
passwordMinLength	1 - 2048 (integer)	The Minimum length of a password. The default value is 9.
passwordMinLower	0 - 2048 (integer)	The minimum required number of lower case letters. The default value is 2.
passwordMinSpecial	0 - 2048 (integer)	The minimum required number of special characters. The default value is 2.
passwordMinUpper	0 - 2048 (integer)	The minimum required number of upper case letters. The default value is 2.
passwordNotStoreName	[true|false] (boolean)	If true, password should not be the same as current store name, nor is it the store name spelled backwards or with the numbers 1–100 appended. The default value is true.
passwordNotUserName	[true|false] (boolean)	If true, password should not be the same as current user name, nor is it the user name spelled backwards or with the numbers 1-100 appended. The default value is true.
passwordProhibited	list of strings separated by comma (string)	Simple list of words that are not allowed to be used as a password. The default reserved words are: oracle,password,user,nosql.
passwordRemember	0 - 256 (integer)	The maximum number of passwords to be remembered that are not allowed to be reused when setting a new password. The default value is 3.

For more information on top-level, transport, and password security parameters see the *Security Guide*.

Admin Restart

Changes to the following Oracle NoSQL Database parameters will result in a Admin restart by the Storage Node Agent:

Admin parameters:

- adminHttpPort
- adminLogFileCount
- adminLogFileLimit
- configProperties
- javaAdminParamsOverride
- javaMiscParams (**deprecated**)
- loggingConfigProps

For example:

```
kv-> plan change-parameters -all-admins
-params adminLogFileCount=10
```

Output:

```
Started plan 14. Use show plan -id 14 to check status.
        To wait for completion, use plan wait -id 14


kv-> show plan -id 14
```

Output:

```
Plan Change Admin Params (14)
Owner: null
State:                 INTERRUPTED
Attempt number:        1
Started:               2024-04-05 20:12:06 UTC
Ended:                 2024-04-05 20:12:06 UTC
Total tasks:           4
 Successful:           1
 Interrupted:          1
 Not started:          2
Tasks not started
   Task StartAdmin start admin1
   Task WaitForAdminState waits for Admin admin1 to reach RUNNING state


kv-> plan execute -id 14
```

Output:

```
Started plan 14. Use show plan -id 14 to check status.
        To wait for completion, use plan wait -id 14


kv-> show plan -id 14
```

Output:

```
Plan Change Admin Params (14)
State:                  SUCCEEDED
Attempt number:         1
Started:                2024-04-05 20:20:18 UTC
Ended:                  2024-04-05 20:20:18 UTC
Total tasks:            2
 Successful:            2
```

> **Note:**
>
> When you change a parameter that requires an Admin restart using the `plan change-parameters` command, the plan ends in an `INTERRUPTED` state. To transition it to a `SUCCESSFUL` state, re-issue the plan a second time using the `plan execute -id <id>` command.

Replication Node Restart

The Storage Node Agent must be restarted to reflect any changes in the setting of the following parameters.

Storage Node parameters:

- serviceLogFileCount
- serviceLogFileLimit
- serviceLogFileCompression
- servicePortRange

Replication Node parameters:

- configProperties
- javaMiscParams (**deprecated**)
- javaRnParamsOverride
- loggingConfigProps

Removing an Oracle NoSQL Database Deployment

There are no scripts or tools available to completely remove an Oracle NoSQL Database installation from your hardware. However, the procedure is simple. On each node (machine) comprising your store:

1. Shut down the Storage Node:

   ```
   java -Xmx64m -Xms64m \
   -jar KVHOME/lib/kvstore.jar stop -root KVROOT
   ```

   Note that if an Admin process is running on the machine, this command also stops that process.

2. Physically remove the entire contents of KVROOT:

   ```
   > rm -rf KVROOT
   ```

3. Empty the contents of all the storage directories configured for the KVStore. For example, if you configured three storage directories using the `makebootconfig` utility, you must clean up all the three storage directories.

   ```
   cd /disk1
   rm -rf *
   ```

Once you have performed this procedure on every machine comprising your store, you have completely removed the Oracle NoSQL Database deployment from your hardware.

Modifying Storage Node HA Port Ranges

When you initially configured your installation, you defined a range of ports for the nodes to use when communicating between themselves. (You did this in Installation Configuration Parameters.) This range of ports is called the *HA port range*, where *HA* is an acronym for High Availability, and indicates your store's replication factor.

If you inadvertently used invalid values for the HA Port Range, you cannot deploy a Replication Node (RN) or a secondary Administration process (Admin) on any Storage Node. You will discover the problem when you first attempt to deploy a store with a Replication node. Following are indications that the Replication Node did not come up on the Storage Node:

- The Admin logs include an error that the Replication Node is in the `ERROR_RESTARTING` state. After a number of retries, the warning error changes to `ERROR_NO_RESTART`. You can find the Replication Node state in the `ping` command output.
- The plan enters an `ERROR` state. Using the CLI's `show plan <planID>` command to get more history details includes an error message like this:

  ```
  Attempt 1
          state: ERROR
          start time: 10-03-11 22:06:12
          end time: 10-03-11 22:08:12
          DeployOneRepNode of rg1-rn3 on sn3/farley:5200 [RUNNING]
          failed. ....  Failed to attach to RepNodeService for rg1-rn3,
          see log, /KVRT3/<storename>/log/rg1-rn3*.log, on host
          farley for more information.
  ```

- The critical events mechanism, accessible through the Admin CLI `show events` command, includes an alert containing the same error information from the plan history.

- The store's runtime or boot logs for the Storage Node and/or Admin shows a port specific error message, such as:

```
[rg1-rn3] Process exiting
java.lang.IllegalArgumentException: Port number 1 is invalid because
the port must be outside the range of "well known" ports
```

You can address incorrect HA port ranges in a configuration by completing the following steps. Steps that require you to execute them on the physical node hosting the Oracle NoSQL Database Storage Node, begin with the directive *On the Storage Node*. You can execute other steps from any node that can access the Admin CLI.

1. Using the Admin CLI, cancel the `plan deploy-sn` or `plan deploy-admin` command that includes invalid HA Port Range values.
2. On the Storage Node, kill the existing, incorrectly configured `StorageNodeAgentImpl` process and all of its Managed Processes. You can distinguish managed processes from other processes because they have the parameter `-root <KVROOT>`.
3. On the Storage Node, remove all files from the KVROOT directory.
4. On the Storage Node, recreate the storage node bootstrap configuration file in the KVROOT directory. For directions, see Installation Configuration Parameters.
5. On the Storage Node, restart the storage node using this Java command:

```
java -Xmx64m -Xms64m
-jar KVHOME/lib/kvstore.jar restart
```

6. Using the Admin CLI, you can now create and execute a `deploy-sn` or `deploy-admin` plan, using the same parameters as the initial plan, but with the correct HA range.

Modifying Storage Node Service Port Ranges

This section explains how to modify your Storage Node service port ranges after an initial configuration and deployment.

When you initially configure your installation, you specify a range of ports that your Storage Node's Replication Nodes and Admin services use. These ports are collectively called the *service port ranges*. Configuring them at installation time was optional. If you did not configure them, the configuration scripts automatically selected a range of ports for you.

The process of modifying your service port range depends on whether the Storage Node has already been deployed. You can determine whether a Storage Node has been deployed by using the Command Line Interface (CLI) to run the `show topology` command. (See show topology for details). The `show topology` command lists the Storage Node, along with the host and port if it has been deployed.

Storage Node Not Deployed

Use this process to modify your Service Port Ranges if the Storage Node has been configured but not deployed.

Execute the following steps on the Storage Node host:

1. Kill the existing Storage Node process. You can find the ID of this process by using:

```
ps -af | grep -e "kvstore.jar.*start.*<KVROOT>"
```

Kill the process using:

```
kill <storage node id>
```

2. Remove all the files from the <KVROOT> directory.

```
rm -rf <KVROOT>/*
```

3. Recreate the Storage Node bootstrap configuration file with the updated service port ranges, being sure to specify the `-servicerange` parameter. For example:

```
java -Xmx64m -Xms64m \
-jar <KVHOME>/lib/kvstore.jar makebootconfig -root <KVROOT> \
-port <port> -host <host> -harange <harange> \
-servicerange <startPort, endPort>
```

See makebootconfig for details on using this utility.

4. Restart the Storage Node:

```
java -Xmx64m -Xms64m -jar <KVHOME>/lib/kvstore.jar restart
```

You can proceed to deploy the Storage Node using the Admin CLI. It will use the newly specified service port range.

Storage Node Deployed

Use this process to modify your Service Port Ranges if the Storage Node has been deployed.

1. Using the Admin CLI, modify the service port range using the `plan change-parameters` command. Specify `servicePortRange` while you do. For example:

```
plan change-parameters -service <id> \
-params servicePortRange=<startPort,endPort>
```

`servicePortRange` is described in Storage Node Parameters.

2. Restart the Storage Node process and its services. The Replication Nodes and any admin services for the Storage Node can be stopped in an orderly fashion using the CLI. Use the `show topology` command (show topology) to list all the services associated with the Storage Node.

   Stop each of these services using the `plan stop-service` command. See plan stop-service for details on this command. Note that when you stop a service, you must use the services ID, which you can find from the output of the `show topology` command. Keep track of these IDs because you will need them when you restart the Storage Node.

   Repeat until all services for the Storage Node have been stopped.

3. Kill the existing Storage Node process. You can find the ID of this process by going to the Storage Node host and issuing:

```
ps -af | grep -e "kvstore.jar.*start.*<KVROOT>"
```

Kill the process using:

```
kill <storage node id>
```

> **Note:**
>
> Avoid killing all Replication Nodes in your store at the same time, as doing so will result in unexpected errors.

4. Restart the Storage Node by going to the Storage Node host and issuing:

   ```
   java -Xmx64m -Xms64m -jar <KVHOME>/lib/kvstore.jar restart
   ```

5. Restart the Storage Node services by using `plan start-service` for each service on the Storage Node. See plan start-service for details.
6. When the Storage Node is restarted and all its Replication Nodes and any admin services are running, the services will be using the updated service port range. You can check by first locating the process ID of the Storage Node services using this command:

   ```
   ps -af | grep -e "ManageService.*<KVROOT>"
   ```

   and then check the ports the services are listening to by using this command:

   ```
   netstat -tlpn | grep <id>
   ```

   One of the listening ports is the service port and it should be within the new range.

Availability, Failover and Switchover

Topics:

Availability and Failover

Oracle NoSQL Database is a data storage product with enormous scalability and performance benefits. Additionally, Oracle NoSQL Database offers excellent *availability* mechanisms. These

mechanisms are designed to provide your applications access to data contained in the store in the event of localized hardware and network failures.

This document describes the mechanisms Oracle NoSQL Database uses to ensure your data remains available, along with the various failover algorithms that Oracle NoSQL Database employs. In addition, this document describes application design patterns you can use to best make use of Oracle NoSQL Database's availability mechanisms. In some cases, tradeoffs exist between ensuring data is highly available, and achieving optimal performance. This document explores these tradeoffs.

The intended audience for this document includes system architects, engineers, and others who want to understand the concepts and issues surrounding data availability when using Oracle NoSQL Database. In addition, software engineers responsible for writing code that interacts with an Oracle NoSQL Database store should also read this document.

We recommend that you read and get familiar with the following contents before continuing.

- Developers Guide

  This document introduces terms and concepts you need to know before reading this document.
- Durability Guarantees in the *Java Direct Driver Developer's Guide*

  This section includes concepts that lead to issues surrounding write availability.
- Consistency Guarantees in the *Java Direct Driver Developer's Guide*

  This section includes concepts that lead to issues surrounding read availability.

Replication Overview

To ensure data durability and availability, Oracle NoSQL Database uses a single-master replication strategy. Using a single machine to perform write operations, Oracle NoSQL Database then broadcasts those operations to multiple read-only replicas.

The *Concepts Guide* describes a shard as a collection of replication nodes, associated with a single master node and multiple replicas. Your store contains multiple shards, and your data is spread evenly across all of the shards that your store uses.

When you perform a write operation in your store, Oracle NoSQL Database completes the write operation on the master node in use by the shard containing your data. The master node performs this write according to whatever durability guarantees are in place at the time. If you set a strong durability guarantee, the master requires the participation of some or all of the replicas in the shard to complete the write operation.

If the master node of the shard becomes unavailable for any reason, the replica nodes in primary zones hold an election to determine which of the remaining replication nodes should take over as the master node. The replication node with the most up-to-date data wins the election.

The election is decided based on a simple majority vote. This means that a majority of the nodes in the shard in primary zones must be available to participate in the election to select a new master.

Loss of a Read-Only Replica Node

A common fail over case is losing a replica node due to a problem with the machine upon which it is running. This loss can be due to something as common as a hard drive failure.

In this case, the only shard that is affected is the one using the replica. By default, the effect on the shard is reduced read throughput capacity. The shard itself is capable of continuing normal operations. However, losing a single Replication Node reduces its capacity to service read requests by whatever read throughput a single host machine offers your store. Whether you detect this reduction in read throughput capacity depends on how heavy a read load your shard is experiencing. The shard could have a low enough read load that losing the replica results in a minor performance reduction.

Such a small performance reduction assumes that a single host machine contains only one Replication Node. If you configure your store so that multiple Replication Nodes run on a single host, then the loss of throughput capacity increases accordingly. It is likely that the loss of a machine running multiple Replication Nodes will affect the throughput capacity of more than one shard, because it is unlikely that all the Replication Nodes on that machine will belong to the same shard. Again, whether you notice any performance reduction from the loss of the Storage Node depends on how heavy a read load the individual affected shards are experiencing.

In this scenario, with one exception, the shard will continue servicing write requests, and may be able to do so with no changes to its write throughput capacity. The master itself is not affected, so it can continue performing writes and replicating them to the remaining replicas in the shard. There can be reduced write throughput capacity if:

- there is such a heavy read load on the shard that the loss of one replica saturates the remaining replica(s); and
- the master requires an acknowledgement before finishing a write commit.

In this scenario, write performance capacity can be reduced either because the master is continually waiting for the replica to acknowledge commits, or because the master itself is expending resources responding to read requests. In either case, you may see degraded write throughput, but the level of degradation depends on how heavy the read/write load actually is on the shard. Again, it is possible that you will never detect any write throughput reduction, because the write load on the shard is low.

In addition, the loss of a single read-only replica can cause all write operations at that shard to fail with a `DurabilityException` exception. This happens if you are using a durability guarantee that requires acknowledgements from all replicas in the shard in primary zones. In this case, writes at that shard will fail until either that replica is brought back online, or you place a less strict durability guarantee into use.

Using durability guarantees that require acknowledgements from all replicas in primary zones offer you the strongest data durability possible (by making certain that your writes are replicated to every machine in a shard). At the same time, they have the potential to lose write capabilities for an entire shard from a single hardware failure. Consequently, be sure to balance your durability requirements against your availability requirements, and configure your store and related code accordingly.

Loss of a Read/Write Master

If you lose a host machine containing a shard's master, the shard will be incapable of responding to write requests, momentarily. The lack of write request response is so brief that it may not be detected by your client code. Only the shard containing the master is affected by this outage. All other shards continue to perform as normal.

In this case, the shard's replicas in primary zones will quickly notice the master is missing and call for an election. Typically this will occur within a few milliseconds after losing the master.

The replica nodes will conduct an election, and the replica in a primary zone with the most up-to-date set of data will be elected master. To be elected master requires a simple majority vote

from the other machines in the shard hosting nodes in primary zones. Keep in mind that this simple majority requirement has implications if many machines are lost from your store.

Once a new master is elected, the shard will continue operations, reducing its read throughput capacity by one machine. As with the loss of a single replica (see the previous section), all write operations can continue as long as your durability guarantee does not require acknowledgements from all replicas in primary zones.

Your client code will not notice the missing master if the new master is elected and services the write request within the timeout value used for the write operation. However, we recommend that your production code include ways to guard against timeout problems. In the event of a timeout, your code should include a decision policy about what to do next. For example, your policy could:

- Retry the write operation immediately,
- Retry the write operation after a defined wait,
- Abandon the write operation entirely.

Unplanned Network Partitions

A shard can be split into two, non-communicating networks. Such an event can occur when a piece of network hardware, such as a router, fails in some way that divides the shard. The store's response to such an event depends on how the network partition divides the shard's Replication Nodes as in these three cases:

A single Replication Node is isolated from the rest of the shard. If the Replication Node is a read-only replica, the shard continues operating as normal, but without the read throughput capacity caused by the loss of a single machine. See Loss of a Read-Only Replica Node for more details.

A single Replication Node becomes isolated from the rest of the shard. If the Replication Node is a master, the shard handles the event in the same way as if it had lost a master. The shard holds an election to select a new master and then continues operating as normal. See Loss of a Read/Write Master for further information.

The new network partition divides the shard into two or more groups of machines. In this case, there will be at least one *minority node partition.* A minority node partition contains less than a majority of the Replication Nodes in the shard. There could also be a *majority node partition.* A majority node partition has the majority of nodes in the shard —. However, a majority node partition is not a given, especially if the new network partition creates more than two sets of Replication Nodes.

How failover is handled in this scenario depends on whether a majority node partition does exist, and if the master exists in that partition. There are also other issues to consider, such as the durability and consistency policies that were in use at the time the new network partition was created.

Master is in the Majority Node Partition

Suppose the shard is divided into two partitions. Partition A contains a simple majority of the Replication Nodes in primary zones, including the master. Partition B has the remaining nodes.

- Partition A continues to service read and write requests as normal, but with a reduced read throughput from the loss of however many Replication Nodes are in Partition B. A caveat in this situation is what durability policy is in use at the time. If Partition A does not have enough replicas from primary zones to meet the durability policy requirements, it could be

prevented from servicing write requests. If the durability policy requires a simple majority, or less, of replicas, then the shard will be able to service write requests.

- Partition B continues to service read requests as normal, but with increasingly stale data. Depending on the consistency guarantee in place, Partition B might cease to service read requests. If a version-based consistency is in use, then Partition B will probably encounter `ConsistencyException` exceptions soon after the network partition occurs, due to its inability to obtain version tokens from the master. Similarly, if a time-based consistency policy is in use, then `ConsistencyException` exceptions will occur as soon as the replica lags too far behind the master, from which it is no longer receiving write updates. By default, a consistency guarantee is not required to service read requests. So unless you explicitly create and use a consistency policy, Partition B can continue to service read requests through the entire network outage.

  Partition B will attempt to elect a new master, but will be unable to do so because it does not contain the simple majority of Replication Nodes required to hold an election.

Further, if the partition is such that your client code can reach Partition A but not Partition B, then the shard will continue to service read and write requests as normal, but with a reduced read capacity.

However, if the partition is such that your client code can read Partition B but not Partition A, then the shard will be unable to service any write requests. This is because Partition A contains the master, and Partition B does not include enough Replication Nodes to elect a new master.

Master is in the Minority Node Partition

Suppose the shard is divided into two partitions. Partition A contains a simple majority of the Replication Nodes from primary zones, but NOT the master. Partition B has the remaining nodes, including the master.

Assuming both partitions are network accessible by your client code, then:

- Partition A will notice that it no longer has a master. Because Partition A has at least a simple majority of the Replication Nodes in primary zones, it will be able to elect a new master. It will do this quickly, and the shard will continue operations as normal.

  Whether Partition A can service write requests is determined by the durability policy in use. As long as the durability policy requires a simple majority, or less, of replicas, then the shard is able to service write requests.

- Partition B will continue to operate as normal, believing that it has a valid master. However, the only way Partition B can service write requests is if the durability policy in use requires no participation from the shard's replicas. If a majority of nodes in primary zones must acknowledge the write operation, or if all nodes in primary zones must acknowledge the write, then the partitions will be unable to service writes because not enough nodes are available to satisfy the durability policy.

  If durability NONE is in use, then for the period of time that it takes to resolve the network partition, the shard will operate with two masters. When the partition is resolved, the shard will recognize the problem and correct it. Because Partition A held a valid election, writes performed there will be kept. *Any writes performed in Partition B will be discarded.* The old master in Partition B will be demoted to a simple replica, and the replicas in Partition B will all be synced with the new master.

> **Note:**
>
> Because of the potential for loss of data in this scenario, Oracle *strongly* recommends that you do NOT use durability NONE. The only time you should use that durability setting is if you want to absolutely maximize write throughput, and do not care if you lose the data.

Further, if the partition is such that your client code can reach Partition A but not Partition B, then the shard will continue to service read and write requests as normal, but only after an election is held, and then with a reduced read capacity.

However, if the partition is such that your client code can read Partition B but not Partition A, then the shard will be unable to service write requests at all, unless you use the weakest durability policy available. This is because Partition B does not include enough Replication Nodes to satisfy anything other than the weakest available durability policy.

No Majority Node Partition

Suppose the shard is divided into multiple partitions, and no partition contains a majority of the Replication Nodes in the shard. In this case, the shard's partitions can service read requests, so long as the consistency policy in use for the read supports it. If the read requires tight consistency with the master, and the master is not available to ensure the consistency can be met, then the read will fail.

The partition containing the master can service write requests only if you are using the weakest available durability policy, in which no acknowledgements from replicas are required. If acknowledgements are required, then there will not be enough replicas to satisfy the durability policy and no write operations can occur.

Once the network partition is resolved, the shard will elect a new master, synchronize all replicas with it, and continue operations as normal.

Failover and Switchover Operations

Optimal use of available physical datacenters is achieved by deploying your store across multiple zones. This provides fault isolation as each zone has a copy of your complete store, including a copy of all the shards. With this configuration, when a zone fails, write availability is automatically reestablished as long as quorum is maintained.

> **Note:**
>
> To achieve other levels of fault isolation, best practices for data center design should be applied. For example, site location, building selection, floor layout, mechanical design, electrical system design, modularity, etc.

However, if quorum is lost, manual procedures such as failovers can be used instead to recover from zone failures. For more information on quorum, see *Concepts Guide*.

A failover is typically performed when the primary zone fails or has become unreachable and one of the secondary zones is transitioned to take over the primary role. Failover can also be performed to reduce the quorum to the available primary zones. Failover may or may not result in data loss.

Switchovers can be used after performing a failover (to restore the original configuration) or for planned maintenance.

A switchover is typically a role reversal between a primary zone and one of the secondary zones of the store. A switchover can also be performed to convert one or more zones to another type for maintenance purposes. Switchover requires quorum and guarantees no data loss. It is typically done for planned maintenance of the primary system.

In this chapter, we explain how failover and switchover operations are performed.

> **Note:**
>
> Arbiters are not currently supported during failover and switchover operations.

Repairing a Failed Zone

If a zone fails but quorum is maintained, you have the option to repair the failed zone with new hardware by following the procedure described in Repairing a Failed Zone by Replacing Hardware.

Another option is to convert the failed zone to a secondary zone. In some cases, this approach can improve the high availability characteristics of the store by reducing the quorum requirements.

For example, suppose a store consists of two primary zones: zone 1 with a replication factor of three and zone 2, with a replication factor of two. Additionally, suppose zone 2 fails. In this case, quorum is maintained because you would have 3 out of the 5 replicas, but any additional failure would result in a loss of quorum.

Converting zone 2 to a secondary zone would reduce the primary replication factor to 3, meaning that each shard could tolerate an additional failure.

You should determine if switching zone types would actually improve availability. If so, then decide if it is worth doing in the current circumstances.

Need for an Admin node in the secondary zone:

Having admins in a secondary zone is very useful to support failure recovery. For example, if a store has primary and secondary zones, and all of the primary zones are lost, the administrator can use the repair-admin-quorum and plan failover commands to resume operations by converting the secondary zone to a primary zone. But these operations can occur only if an Admin node is available. For this reason, stores with secondary zones should include Admins in the secondary zones.

The recommendation is to deploy the same number of admins as the replication factor for the zone. For example if you have primary and secondary zone with a replication factor of 3, then each zone should be configured with three admins. If a zone failure occurs and no admins remain available, the failover procedures cannot be used. To avoid this situation you need to configure as many admins as the replication factor for the zone.

Performing a Failover

If quorum is maintained, you do not need to do anything because the store is still performing normally.

In situations where a zone fails but quorum is lost, your only option is to perform a failover.

For example, suppose a store consists of two zones, "Manhattan" and "JerseyCity", each deployed in its own physical data center.

> **Note:**
>
> This example uses a store with a replication factor of three. In this case, each zone is also configured with three admins.

Additionally, suppose that the "Manhattan" zone fails, resulting in the failure of all of the associated Storage Nodes and a loss of quorum. In this case, if the host hardware of "Manhattan" was irreparably damaged or the problem will take too long to repair you may choose to initiate a failover.

The following steps walk you through the process of verifying failures, isolating Storage Nodes, and reducing admin quorum to perform a failover operation. This process allows service to be continued in the event of a zone failure.

1. Connect to the store. To do this, connect to an admin running in the JerseyCity zone:

```
java -Xmx64m -Xms64m -jar KVHOME/lib/kvstore.jar \
runadmin -host jersey1 -port 6000 \
-security USER/security/admin.security
```

> **Note:**
>
> This assumes that you must have followed the steps as mentioned in Create users and configure security with remote access .

2. Use the `verify configuration` command to confirm the failures. The output confirms the Storage Node Agents in the Manhattan zone are unavailable.

```
kv-> verify configuration
```

Output:

```
Connected to Admin in read-only mode
Verify: starting verification of store mystore based upon topology
sequence #115
100 partitions and 6 storage nodes
Time: 2024-04-05 07:15:23 UTC   Version: 24.1.11
See jersey1:/kvroot/mystore/log/mystore_{0..N}.log for progress messages
Verify: Shard Status: healthy: 0 writable-degraded: 0 read-only: 1
offline: 0 total: 1
Verify: Admin Status: read-only
Verify: Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false] RN Status: online: 0 read-only: 0 offline: 3
Verify: Zone [name=JerseyCity id=zn2 type=SECONDARY allowArbiters=false
masterAffinity=false]   RN Status: online: 0 read-only: 3 offline: 0
Verify: == checking storage node sn1 ==
Verify:  sn1: ping() failed for sn1 :
Unable to connect to the storage node agent at host nyc1,
```

```
port 5000, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify: Storage Node [sn1] on nyc1:5000
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
  UNREACHABLE
Verify:  admin1: ping() failed for admin1 :
Unable to connect to the storage node agent at host nyc1,
port 5000, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify:  Admin [admin1]      Status: UNREACHABLE
Verify:  rg1-rn1: ping() failed for rg1-rn1 :
Unable to connect to the storage node agent at host nyc1,
port 5000, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify:  Rep Node [rg1-rn1]  Status: UNREACHABLE
Verify: == checking storage node sn2 ==
Verify:  sn2: ping() failed for sn2:
Unable to connect to the storage node agent at host nyc1,
port 5100, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify: Storage Node [sn2] on nyc1:5100
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
  UNREACHABLE
Verify:  admin2: ping() failed for admin2:
Unable to connect to the storage node agent at host nyc1,
port 5100, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify:   Admin [admin2]      Status: UNREACHABLE
Verify:   rg1-rn2: ping() failed for rg1-rn2 :
Unable to connect to the storage node agent at host nyc1,
port 5100, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify: Rep Node [rg1-rn2]  Status: UNREACHABLE
Verify: == checking storage node sn3 ==
Verify:  sn3: ping() failed for sn3:
Unable to connect to the storage node agent at host nyc1,
port 5200, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify: Storage Node [sn3] on nyc1:5200
Zone: [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
```

```
masterAffinity=false]
  UNREACHABLE
Verify:  admin3: ping() failed for admin3:
Unable to connect to the storage node agent at host nyc1,
port 5200, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify:  Admin [admin3]      Status: UNREACHABLE
Verify: rg1-rn3: ping() failed for rg1-rn3 :
Unable to connect to the storage node agent at host nyc1,
port 5200, which may not be running; nested exception is:
    java.rmi.ConnectException: Connection refused to host:
    nyc1; nested exception is:
    java.net.ConnectException: Connection refused
Verify: Rep Node [rg1-rn3]  Status: UNREACHABLE
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on jersey1:6000
Zone: [name=JerseyCity id=zn2 type=SECONDARY allowArbiters=false
masterAffinity=false]
Ver: 24.1.11 2024-04-05 21:24:59 UTC  Build id: 78bbc4cb976b Edition:
Enterprise isMasterBalanced: true   serviceStartTime: 2024-04-05 07:05:44
UTC
Verify:  Admin [admin4]
Status: RUNNING,MASTER (non-authoritative)
Verify:  Rep Node [rg1-rn4]
Status: RUNNING,MASTER (non-authoritative) sequenceNumber:217 haPort:6003
available storage size:12 GB
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on jersey1:6100
Zone: [name=JerseyCity id=zn2 type=SECONDARY allowArbiters=false
masterAffinity=false]
Ver: 24.1.11 2024-04-05 21:24:59 UTC  Build id: 78bbc4cb976b Edition:
Enterprise isMasterBalanced: true   serviceStartTime: 2024-04-05 07:05:44
UTC
Verify:  Admin [admin5]
Status: RUNNING,MASTER (non-authoritative)
Verify:  Rep Node [rg1-rn5]
Status: RUNNING,MASTER (non-authoritative) sequenceNumber:217 haPort:6003
available storage size:12 GB
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on jersey1:6200
Zone: [name=JerseyCity id=zn2 type=SECONDARY allowArbiters=false
masterAffinity=false]
Ver: 24.1.11 2024-04-05 21:24:59 UTC  Build id: 78bbc4cb976b Edition:
Enterprise isMasterBalanced: true   serviceStartTime: 2024-04-05 07:05:44
UTC
Verify:  Admin [admin6]
Status: RUNNING,MASTER (non-authoritative)
Verify:  Rep Node [rg1-rn6]
Status: RUNNING,MASTER (non-authoritative) sequenceNumber:217 haPort:6003
available storage size:12 GB
Verification complete, 9 violations, 0 notes found.
Verification violation: [admin1]        ping() failed for admin1 : Unable
to connect to the storage node agent at host nyc1, port 5000 , which may
not be running; nested exception is:
```

```
        java.rmi.ConnectException: Connection refused to host: nyc1;
nested exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [admin2]        ping() failed for admin2 : Unable
to connect to the storage node agent  at host nyc1, port 5100, which may
not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: nyc1;
nested exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [admin3]        ping() failed for admin3 : Unable
to connect to the storage node  at host nyc1, port 5200, which may not be
running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: nyc1;
nested exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [rg1-rn1]       ping() failed for rg1-rn1 : Unable
to connect to the storage node agent at host nyc1, port 5000, which may
not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: nyc1;
nested exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [rg1-rn2]       ping() failed for rg1-rn2 : Unable
to connect to the storage node agent at host nyc1, port 5100, which may
not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: nyc1;
nested exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [rg1-rn3]       ping() failed for rg1-rn3 : Unable
to connect to the storage node agent at host nyc1, port 5200, which may
not be running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: nyc1;
nested exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [sn1]   ping() failed for sn1 : Unable to connect
to the storage node agent at host nyc1, port 5000, which may not be
running; nested exception is:
        java.rmi.ConnectException: Connection refused to host:nyc1; nested
exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [sn2]   ping() failed for sn2 : Unable to connect
to the storage node agent at host nyc1, port 5100, which may not be
running; nested exception is:
        java.rmi.ConnectException: Connection refused to host: nyc1;
nested exception is:
        java.net.ConnectException: Connection refused (Connection refused)
Verification violation: [sn3]   ping() failed for sn3 : Unable to connect
to the storage node agent at host nyc1, port 5200, which may not be
running; nested exception is:
        java.rmi.ConnectException: Connection refused to host:nyc1; nested
exception is:
        java.net.ConnectException: Connection refused (Connection refused)
```

In this case, the Storage Node Agent at host nyc1 is confirmed unavailable.

3. To prevent a hard rollback and data loss, isolate failed nodes (Manhattan) from the rest of the system. Make sure all failed nodes are prevented from rejoining the store until their configurations have been updated.

   To do this, you can:

   - Disconnect the network physically or use a firewall.
   - Modify the start-up sequence on failed nodes to prevent SNAs from starting.

4. To make changes to the store, you first need to reduce admin quorum. To do this, use the `repair-admin-quorum` command, specifying the available primary zone:

   ```
   kv-> repair-admin-quorum -znname JerseyCity
   ```

   Output:

   ```
   Connected to admin in read-only mode
   Repaired admin quorum using admins: [admin4, admin5, admin6]
   ```

   Now you can perform administrative procedures using the remaining admin service with the temporarily reduced quorum.

5. Use the `plan failover` command to update the configuration of the store with the available zones.

   ```
   kv-> plan failover -znname \
   JerseyCity -type primary \
   -znname Manhattan -type offline-secondary -wait
   ```

   Output:

   ```
   Executing plan 8, waiting for completion...
   Plan 8 ended successfully
   ```

   The `plan failover` command fails if it is executed while other plans are still running. You should cancel or interrupt the plans, before executing this plan.

   For example, suppose the `topology redistribute` is in progress. If you run the `plan failover` command, it will fail. For it to succeed, you need to first cancel or interrupt the `topology redistribute` command.

   To do this, first use the `show plans` command to learn the plan ID of the topology redistribute command. In this case, 9. Then, cancel the `topology redistribute` command using the `plan cancel` command:

   ```
   kv-> plan cancel -id 9
   ```

   After performing the failover, confirm that the zone type of Manhattan has been changed to secondary using the `ping` command.

   ```
   kv-> ping
   ```

Output:

```
Pinging components of store mystore based upon topology sequence #208
100 partitions and 6 storage nodes
Time: 2024-04-05 07:33:51 UTC   Version: 24.1.11
Shard Status: healthy:0 writable-degraded:1 read-only:0 offline:0
Admin Status: writable-degraded
Zone [name=Manhattan id=zn1 type=SECONDARY allowArbiters=false
masterAffinity=false]
RN Status: online:0 offline:3
Zone [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
RN Status: online:3 offline:0
Storage Node [sn1] on nyc1:5000
Zone: [name=Manhattan id=zn1 type=SECONDARY allowArbiters=false
masterAffinity=false]
UNREACHABLE
      Admin [admin1]          Status: UNREACHABLE
      Rep Node [rg1-rn1]      Status: UNREACHABLE
Storage Node [sn2] on nyc1:5100
Zone: [name=Manhattan id=zn1 type=SECONDARY allowArbiters=false
masterAffinity=false]
UNREACHABLE
      Admin [admin2]          Status: UNREACHABLE
      Rep Node [rg1-rn2]      Status: UNREACHABLE
Storage Node [sn3] on nyc1:5200
Zone: [name=Manhattan id=zn1 type=SECONDARY allowArbiters=false
masterAffinity=false]
UNREACHABLE
      Admin [admin3]          Status: UNREACHABLE
      Rep Node [rg1-rn3]      Status: UNREACHABLE
Storage Node [sn4] on jersey1:6000
Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 21:24:59 UTC  Build id: 78bbc4cb976b
      Admin [admin4]          Status: RUNNING,REPLICA
      Rep Node [rg1-rn4]
Status: RUNNING,REPLICA sequenceNumber:427 haPort:6011 available storage
Storage Node [sn5] on jersey1:6100
Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 21:24:59 UTC  Build id: 78bbc4cb976b
      Admin [admin5]          Status: RUNNING,REPLICA
      Rep Node [rg1-rn5]
Status: RUNNING,REPLICA sequenceNumber:427 haPort:6011 available storage
Storage Node [sn6] on jersey1:6200
Zone: [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 21:24:59 UTC  Build id: 78bbc4cb976b
     Admin [admin6]          Status: RUNNING,MASTER
     Rep Node [rg1-rn6]
Status: RUNNING,MASTER sequenceNumber:427 haPort:6011 available storage
```

The failover operation is now complete. Write availability in the store is reestablished using zone 2 as the only available primary zone. Zone 1 is offline. Any data that was not propagated from zone 1 prior to the failure will be lost.

> **Note:**
>
> In this case, the store has only a single working copy of its data, so single node failures in the surviving zone will prevent read and write access, and, if the failure is a permanent one, may produce permanent data loss.

If the problems that led to the failover have been corrected and the original data from the previously failed nodes (Manhattan) is still available, you can return the old nodes to service by performing a switchover. To do this, see the next section.

Performing a Switchover

To continue from the example of the previous section, after performing the failover, you can return the old nodes to service by performing the following switchover procedure:

1. After the failed zones are repaired, restart all the Storage Nodes of the failed zones without starting any services (avoids hard rollback):

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar restart -disable-services \
-root nyc1/KVROOT &
```

> **Note:**
>
> When performing planned maintenance, there is no need to isolate nodes or disable services prior to bringing nodes back online.

2. Reestablish network connectivity or reenable the standard startup sequence of the previously failed zones.
3. Repair the topology so that the topology for the newly restarted Storage Nodes can be updated with changes made by the failover.

```
java -Xmx64m -Xms64m -jar KVHOME/lib/kvstore.jar runadmin \
-host jersey1 -port 5000 \
-security USER/security/admin.security


kv-> plan repair-topology -wait
```

Output:

```
Executed plan 10, waiting for completion...
Plan 10 ended successfully
```

> **Note:**
>
> This assumes that you must have followed the steps as mentioned in Create users and configure security with remote access .

> **Note:**
>
> This command will also restart services on the previously failed nodes.

Use the `verify configuration` command to confirm that there are no configuration problems.

4. Run the ping command. The "maxCatchupTimeSecs" value will be used for the -timeout flag of the `await-consistency` command.

   Use the timeout flag to specify an estimate of how long the switchover will take. For example, if the nodes have been offline for a long time it might take many hours for them to catch up so that they can be converted back to primary nodes.

   ```
   kv-> ping
   ```

   Output:

   ```
   Pinging components of store mystore based upon topology sequence #117
   100 partitions and 6 storage nodes
   Time: 2024-04-05 07:39:18 UTC   Version: 24.1.11
   Shard Status: healthy: 1 writable-degraded: 0 read-only: 0 offline: 0
   total: 1
   Admin Status: healthy
   Zone [name=Manhattan id=zn1 type=SECONDARY allowArbiters=false
   masterAffinity=false]
   RN Status: online: 3 read-only: 0 offline: 0 maxDelayMillis: 3
   maxCatchupTimeSecs: 0
   Zone [name=JerseyCity id=zn2 type=PRIMARY allowArbiters=false
   masterAffinity=false]
   RN Status: online: 3 read-only: 0 offline: 0 maxDelayMillis: 4
   maxCatchupTimeSecs: 0
   Storage Node [sn1] on nyc1: 5000    Zone: name=Manhattan id=zn1
   type=SECONDARY
   allowArbiters=false masterAffinity=false]
   Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
   Build id: 78bbc4cb976b Edition: Enterprise isMasterBalanced: true
   serviceStartTime: 2024-04-05 07:36:01 UTC
   Admin [admin1]  Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
   07:38:14 UTC
   stateChangeTime: 2024-04-05 07:38:14 UTC  availableStorageSize: 2 GB
   Rep Node [rg1-rn1] Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
   5111
   availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
   07:37:14 UTC
   stateChangeTime: 2024-04-05 07:37:20 UTC delayMillis: 0 catchupTimeSecs: 0
   ```

```
Storage Node [sn2] on nyc1: 5100    Zone: [name=Manhattan id=zn1
type=SECONDARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise  isMasterBalanced: true
serviceStartTime: 2024-04-05 07:36:25 UTC
Admin [admin2]  Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:38:34 UTC
stateChangeTime: 2024-04-05 07:38:33 UTC availableStorageSize: 2 GB
Rep Node [rg1-rn2]  Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5211
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
07:37:28 UTC
stateChangeTime: 2024-04-05 07:37:33 UTC delayMillis: 0 catchupTimeSecs: 0
Storage Node [sn3] on nyc1: 5200    Zone: [name=Manhattan id=zn1
type=SECONDARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise isMasterBalanced: true
serviceStartTime: 2024-04-05 07:36:35 UTC
Admin [admin3]  Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:38:56 UTC
stateChangeTime: 2024-04-05 07:38:56 UTC  availableStorageSize: 2 GB
Rep Node [rg1-rn3] Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5311
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
07:37:43 UTC
stateChangeTime: 2024-04-05 07:37:49 UTC delayMillis: 3 catchupTimeSecs: 0
Storage Node [sn4] on jersey1: 6000  Zone: [name=JerseyCity id=zn2
type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise isMasterBalanced: true
serviceStartTime: 2024-04-05 07:05:44 UTC
Admin [admin4]  Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:36:49 UTC
stateChangeTime: 2024-04-05 07:36:47 UTC availableStorageSize: 2 GB
Rep Node [rg1-rn4]  Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5411
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
07:36:36 UTC
stateChangeTime: 2024-04-05 07:36:59 UTC delayMillis: 4 catchupTimeSecs: 0
Storage Node [sn5] on jersey1: 6100    Zone: [name=JerseyCity id=zn2
type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise  isMasterBalanced: true
serviceStartTime: 2024-04-059 07:05:54 UTC
Admin [admin5] Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:36:49 UTC
stateChangeTime: 2024-04-05 07:36:48 UTC  availableStorageSize: 2 GB
Rep Node [rg1-rn5]  Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5511
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
07:36:36 UTC
stateChangeTime: 2024-04-05 07:36:59 UTC delayMillis: 0 catchupTimeSecs: 0
```

```
Storage Node [sn6] on jersey1: 6200    Zone: [name=JerseyCity id=zn2
type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise  isMasterBalanced: true
serviceStartTime: 2024-04-05 07:06:03 UTC
Admin [admin6]  Status: RUNNING,MASTER  serviceStartTime: 2024-04-05
07:36:55 UTC
stateChangeTime: 2024-04-05 07:36:46 UTC  availableStorageSize: 2 GB
Rep Node [rg1-rn6] Status: RUNNING,MASTER sequenceNumber: 2,672 haPort:
5611
availableStorageSize: 273 GB storageType: HD  serviceStartTime: 2024-04-05
07:36:36 UTC
stateChangeTime: 2024-04-05 07:36:57 UTC
```

In this case, 1800 seconds (30 minutes) is the value to be used.

5. Use the `await-consistency` command to specify the wait time (1800 seconds) used for the secondary zones to catch up with their masters.

   The system will only wait five minutes for nodes to catch up when attempting to change a zone's type. If the nodes do not catch up in that amount of time, the plan will fail.

   If the nodes will take more than five minutes to catch up, you should run the `await-consistency` command, specifying a longer wait time using the -timeout flag. In this case, the wait time (1800 seconds) is used:

   ```
   kv-> await-consistent -timeout 1800 -znname Manhattan
   The specified zone is consistent
   ```

   By default, nodes need to have a delay of no more than 1 second to be considered caught up. You can change this value by specifying the -replica-delay-threshold flag. You should do this if network delays prevent the nodes from catching up within 1 second of their masters.

   > **Note:**
   >
   > If you do not want the switchover to wait for the nodes to catch up, you can use the -no-replica-delay threshold flag. In that case, nodes will be converted to primary nodes even if they are behind. You should evaluate whether this risk is worth taking.

6. Perform the switchover to convert the previously failed zone back to a primary zone , and the formerly secondary zone back to its earlier state.

   ```
   kv-> topology clone -current -name newTopo
   ```

   ```
   kv-> topology change-zone-type -name newTopo \
   -znname Manhattan -type primary
   ```

Output:

```
Changed zone type of zn1 to PRIMARY in newTopo
```

```
kv-> topology change-zone-type -name newTopo \
-znname JerseyCity -type secondary
```

Output:

```
Changed zone type of zn2 to SECONDARY in newTop
```

```
kv-> plan deploy-topology -name newTopo -wait
```

Output:

```
Executed plan 11, waiting for completion...
Plan 11 ended successfully
```

Confirm the zone type change of the Manhattan zone to PRIMARY by running the ping command.

```
kv-> ping
```

Output:

```
Pinging components of store mystore based upon topology sequence #117
100 partitions and 6 storage nodes
Time: 2024-04-05 07:39:18 UTC   Version: 24.1.11
Shard Status: healthy: 1 writable-degraded: 0 read-only: 0 offline: 0
total: 1
Admin Status: healthy
Zone [name=Manhattan id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
RN Status: online: 3 read-only: 0 offline: 0 maxDelayMillis: 3
maxCatchupTimeSecs: 0
Zone [name=JerseyCity id=zn2 type=SECONDARY allowArbiters=false
masterAffinity=false]
RN Status: online: 3 read-only: 0 offline: 0 maxDelayMillis: 4
maxCatchupTimeSecs: 0
Storage Node [sn1] on nyc1: 5000    Zone: name=Manhattan id=zn1
type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise isMasterBalanced: true
serviceStartTime: 2024-04-05 07:36:01 UTC
Admin [admin1]  Status: RUNNING,MASTER serviceStartTime: 2024-04-05
07:38:14 UTC
stateChangeTime: 2024-04-05 07:38:14 UTC  availableStorageSize: 2 GB
Rep Node [rg1-rn1]  Status: RUNNING,MASTER sequenceNumber: 2,672 haPort:
5111
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
```

```
07:37:14 UTC
stateChangeTime: 2024-04-05 07:37:20 UTC delayMillis: 0 catchupTimeSecs: 0
Storage Node [sn2] on nyc1: 5100  Zone: [name=Manhattan id=zn1
type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING  Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise  isMasterBalanced: true
serviceStartTime: 2024-04-05 07:36:25 UTC
Admin [admin2]   Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:38:34 UTC
stateChangeTime: 2024-04-05 07:38:33 UTC availableStorageSize: 2 GB
Rep Node [rg1-rn2]  Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5211
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
07:37:28 UTC
stateChangeTime: 2024-04-05 07:37:33 UTC delayMillis: 0 catchupTimeSecs: 0
Storage Node [sn3] on nyc1: 5200    Zone: [name=Manhattan id=zn1
type=PRIMARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise    isMasterBalanced: true
 serviceStartTime: 2024-04-05 07:36:35 UTC
Admin [admin3]        Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:38:56 UTC
stateChangeTime: 2024-04-05 07:38:56 UTC  availableStorageSize: 2 GB
Rep Node [rg1-rn3     Status: RUNNING,REPLICA sequenceNumber: 2,672
haPort: 5311
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
07:37:43 UTC
stateChangeTime: 2024-04-05 07:37:49 UTC delayMillis: 3 catchupTimeSecs: 0
Storage Node [sn4] on jersey1: 6000    Zone: [name=JerseyCity id=zn2
type=SECONDARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise isMasterBalanced: true
serviceStartTime: 2024-04-05 07:05:44 UTC
Admin [admin4]  Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:36:49 UTC
stateChangeTime: 2024-04-05 07:36:47 UTC availableStorageSize: 2 GB
Rep Node [rg1-rn4]  Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5411
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
07:36:36 UTC
stateChangeTime: 2024-04-05 07:36:59 UTC delayMillis: 4 catchupTimeSecs: 0
Storage Node [sn5] on jersey1: 6100 Zone: [name=JerseyCity id=zn2
type=SECONDARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise isMasterBalanced: true
serviceStartTime: 2024-04-05 07:05:54 UTC
Admin [admin5]    Status: RUNNING,REPLICA serviceStartTime: 2024-04-05
07:36:49 UTC
stateChangeTime: 2024-04-05 07:36:48 UTC  availableStorageSize: 2 GB
Rep Node [rg1-rn5]   Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5511
availableStorageSize: 273 GB storageType: HD serviceStartTime: 2024-04-05
```

```
07:36:36 UTC
stateChangeTime: 2024-04-05 07:36:59 UTC delayMillis: 0 catchupTimeSecs: 0
Storage Node [sn6] on jersey1: 6200 Zone: [name=JerseyCity id=zn2
type=SECONDARY
allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 21:24:59 UTC
Build id: 78bbc4cb976b Edition: Enterprise  isMasterBalanced: true
serviceStartTime: 2024-04-05 07:06:03 UTC
Admin [admin6]   Status: RUNNING,REPLICA  serviceStartTime: 2024-04-05
07:36:55 UTC
stateChangeTime: 2024-04-05 07:36:46 UTC  availableStorageSize: 2 GB
Rep Node [rg1-rn6]  Status: RUNNING,REPLICA sequenceNumber: 2,672 haPort:
5611
availableStorageSize: 273 GB storageType: HD  serviceStartTime: 2024-04-05
07:36:36 UTC
stateChangeTime: 2024-04-05 07:36:57 UTC
```

Zone Failover

Zones allow you to spread your data store across various physical installation locations. The different locations can be anything from different physical buildings near each other, to different racks in the same building. The basic goal of spreading your store across locations is to guard against large-scale infrastructure disruptions, such as power outages or major storm damage, by placing the nodes in your store physically as far apart as possible.

Oracle NoSQL Database provides support for two kinds of zones. *Primary* zones contain nodes which can serve as masters or replicas. Zones are created as primary zones by default. *Secondary* zones contain nodes which can serve only as replicas. Secondary zones can be used to make a copy of the data available at a distant location, or to maintain an extra copy of the data to increase redundancy or read capacity.

Both types of zones require high throughput network connections to transmit the replication data required to keep replicas up-to-date. Failing to provide sufficient network capacity will result in nodes in poorly connected zones falling farther and farther behind. Locations connected by low throughput network connections are not suitable for use with zones.

For primary zones, in addition to a high throughput network, the network connections with other primary zones should provide highly reliable and low latency communication. These capabilities make it possible to perform master elections for quick master failovers, and to provide acknowledgments to meet write request timeout requirements. Primary zones are not, therefore, suitable for use with an unreliable or slow wide area network.

For secondary zones, the nodes do not participate in master elections or acknowledgments. For this reason, the system can tolerate reduced reliability or increased latency for connections between secondary and primary zones. The network connections still need to provide sufficient throughput to support replication, and must provide sufficient reliability that temporary interruptions do not interfere with network throughput.

If you deploy your store across multiple zones, then Oracle NoSQL Database tries to physically place at least one Replication Node from each shard in each zone. Whether Oracle NoSQL Database can do this depends on the number of shards in use in your store, the number of zones, the number of Replication Nodes, and the number of physical machines available in each zone. Still, Oracle NoSQL Database makes a best-effort to spread Replication Nodes across available zones. Doing so guards against losing entire shards should the zone become unavailable for any reason.

All of the failover descriptions covered here apply to zones. Failover works across zones in the same way as it does if all nodes are contained within a single zone. Zones offer you the ability for your data to remain available in the event of a large outage. However, read and write capability for any given shard is still gated by whether the remaining zone(s) constitute a majority node partition, and the durability and consistency policies in use for your store activities.

Durability Summary

This document has described how durability guarantees affect a shard's write availability in the event of hardware or network failures. In summary:

- A durability guarantee that requires no acknowledgements from the shard's replicas gives you the best chance that the shard can continue servicing write requests in the event of an outage. However, this durability guarantee can also result in the shard operating with two masters, which leads to data loss once hardware problems are resolved. This is *not* a recommended configuration.
- A durability guarantee requiring a simple majority of primary zone replicas to acknowledge the write operation guards against two masters accidently operating at one time. However, it also means that the shard will be incapable of servicing write requests if more than a majority of the replicas are offline due to a hardware failure.
- A durability guarantee requiring all primary zone replicas to acknowledge the write operation guards against any possibility of data loss. However, it also means that the shard will be unable to service write requests if even one of the replicas is unavailable for any reason.

Consistency Summary

In most cases, replicas can continue to service read requests as long as the underlying hardware remains functional. In its default configuration, there is nothing that stops a replica from doing this, even if it is the only node running after some catastrophic failure.

However, is is possible for a replica to stop servicing read requests following a network failure, if the consistency policy requires either version information, or disallows stale data relative to the master. Whether this happens depends on how your Replication Nodes are exactly partitioned as a result of the failure, and how long it takes to establish a new master. The replica's ability to service read requests is also determined by the consistency policy in use for each request. If the read requires tight consistency with the master, and the master is not available to ensure the consistency can be met, then the read will fail.

5

Reference

The articles in this section contains reference information on various command line tools and utilities.

Terminologies used in Oracle NoSQL Database

Some of the terminologies used in Oracle NoSQL Database

Release: Oracle NoSQL Database is released three times in a year. The release number of the Oracle NoSQL database follows this pattern `release.major.minor` where **release** is the last 2 characters of the year(For example: 23), **major release number** denotes the quarter which is 1,2, or 3 and **minor release number** is the final patch number of the release. For example : `23.1.16`.

Data Store: Oracle NoSQL Database applications read and write data by performing network requests against an Oracle NoSQL Database data store. The data store is a collection of Storage Nodes, each of which hosts one or more replication nodes.

host: The hostname associated with the Storage Node on which the data store is installed. A hostname is a unique term assigned to a Storage Node on a network. It distinguishes one Storage Node from another on a specific network.

port: The TCP/IP port through which the Storage Node connects to the Oracle NoSQL Database. This port must be free on the Storage Node. The default port used is 5000. This port is referred to as the registry port.

KVROOT: A directory that stores the data of your data store and security related information. There should be enough disk space on each Storage Node to hold the data of your data store. It is recommended that you use the same directory path for `$KVROOT` on each of the Storage Nodes in the installation.

Storage Node: A Storage Node is a physical (or virtual) machine with its own local storage, which houses the Replication Node.

Replication Node: Every Storage Node hosts one or more replication nodes(RN) as determined by its capacity. A Storage Node's capacity serves as a rough measure of the hardware resources associated with it (memory, CPUs, and disks). , each of which hosts one or more replication node.

Admin service: An administrative process that runs on a Storage Node and runs various Admin CLI commands.

Managed service: The replication node process that run on a Storage Node.

Storage Node Agent: . The Storage Node Agent manages all the Replication Nodes running on the Storage Node (host)

Shard: A shard is a horizontal partition of data in a database. Your data store's replication nodes are organized into shards. A single shard contains multiple replication nodes.

Majority in a Shard: The number of replication nodes which are available and online in a shard. This will help in ensuring read and write availability in the shard and to elect master when one of the nodes fail.

Master Node and Replica Nodes: There are two types of Replication Nodes, namely, **master** and **replica.** Each shard must contain one master node. The master node performs all database write activities. Each shard can also contain one or more read-only replicas. The master node copies all new write activity data to the replicas. The replicas are then used to service read-only operations.

Quorum: Quorum is the minimum number of primary nodes required in a shard, or in the set of admin nodes, to permit electing a master to support write operations.

Zone: A zone is a physical location that supports high-capacity network connectivity between the Storage Nodes deployed within it.

Master Affinity Zones: Master Affinity is a way for you to indicate which primary zones can host master replication nodes. Master Affinity zones service high demand "write" requests across shards.

Replication factor: The total number of masters and replicas in a shard are equal to the replication factor (RF). You can also think of Replication Factor as the number of copies of your data.

Zone Replication factor: The number of copies, or replicas, maintained in a zone.

Primary Replication factor: The total number of replicas in all primary zones.

Secondary Replication factor: The total number of replicas in all secondary zones.

Store Replication factor: The total number of replicas in all zones across the entire store.

Arbiter Node: An Arbiter Node is a lightweight process that is capable of supporting write availability when the primary replication factor is two and a single replication node becomes unavailable or when two replication nodes are unable to communicate to determine which one of them is the master.

Multi-Region data store: Oracle NoSQL Database supports creating tables in multiple data stores, and still maintain consistent data across these clusters. If you have replicated data across data stores, it is called a multi-region data store.

Xregion Service Agent: In a Multi-Region data store, a Cross-Region Service or XRegion Service is a standalone service running on a separate node. In simple terms, this is also called an agent. The XRegion Service is deployed when you are connecting the local data store with a remote data store to create a multi-region table.

Admin CLI Reference

This appendix describes the following commands:

- help
- hidden
- history
- load
- logtail
- namespace
- page
- ping
- plan
- pool
- put
- repair-admin-quorum
- show
- snapshot
- table
- table-size
- timer
- topology
- verbose
- verify

The Command Line Interface (CLI) is run interactively or used to run single commands. The general usage to start the CLI is:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar runadmin \
-host <hostname> -port <port> [single command and arguments]
-security KVROOT/security/client.security
```

If you want to run a script file, you can use the "load" command on the command line:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar runadmin -host <hostname> -port <port> \
-security \
KVROOT/securtiy/client.security \
load -file <path-to-script>
```

If none of the optional arguments are passed, it starts interactively. If additional arguments are passed they are interpreted as a single command to run, then return. The interactive prompt for the CLI is:

```
"kv-> "
```

Upon successful completion of the command, the CLI's process exit code is zero. If there is an error, the exit code will be non-zero.

The CLI comprises a number of commands, some of which have subcommands. Complex commands are grouped by general function, such as "show" for displaying information or "ddl" for manipulating schema. All commands accept the following flags:

- -help

  Displays online help for the command or subcommand.

- ?

  Synonymous with -help. Displays online help for the command or subcommand.

- -verbose

  Enables verbose output for the command.

CLI commands have the following general format:

1. All commands are structured like this:

   ```
   "kv-> command [sub-command] [arguments]
   ```

2. All arguments are specified using flags which start with "-"
3. Commands and subcommands are case-insensitive and match on partial strings(prefixes) if possible. The arguments, however, are case-sensitive.

Inside a CLI script file, you can use `#` to designate a comment. Also, you can terminate a line with a backslash `\` to continue a command onto the next line.

aggregate

Performs simple data aggregation operations on numeric fields like count, sum, average, keys, start and end. The aggregate command iterates matching keys or rows in the store so, depending on the size of the specified key or row, it may take a very long time to complete.

aggregate table is an aggregate subcommand.

aggregate table

```
aggregate table -name <name>
    [-count] [-sum <field[,field,..]>]
    [-avg <field[,field,..]>]
    [-index <name>]
    [-field <name> -value <value>]*
    [-field <name> [-start <value>] [-end <value>]]
    [-json <string>]
```

Performs simple data aggregation operations on numeric fields of the table.

where:

- `-name`

  Specifies the table for the operation.

- `-count`

  Returns the count of matching records.

- `-sum`

Returns the sum of the values of matching fields.

- `-avg`

  Returns the average of the values of matching fields.

- `-index`

  Specifies the name of the index to use. When an index is used, the fields named must belong to the specified index and the aggregation is performed over rows with matching index entries.

- `-field` and `-value` pairs are used to specify the field values of the primary key to use to match for the aggregation, or you can use an empty key to match the entire table.
- The `-field` flat, along with its `-start` and `-end` flags, can be used for restricting the range used to match rows.
- `-json`

  Specifies the fields and values to use for the aggregation as a JSON input string.

See the example below:

```
# Create a table 'user_test' with an index on user_test(age):
kv-> execute 'CREATE TABLE user_test (id INTEGER,
firstName STRING, lastName STRING, age INTEGER, PRIMARY KEY (id))'
Statement completed successfully

kv-> execute 'CREATE INDEX idx1 on user_test (age)'
Statement completed successfully

# Insert 3 rows:
kv-> put table -name user_test -json
'{"id":1,"firstName":"joe","lastName":"wang","age":21}'
Operation successful, row inserted.
kv-> put table -name user_test -json
'{"id":2,"firstName":"jack","lastName":"zhao","age":32}'
Operation successful, row inserted.
kv-> put table -name user_test -json
'{"id":3,"firstName":"john","lastName":"gu","age":43}'
Operation successful, row inserted.

# Get count(*), sum(age) and avg(age) of rows in table:
kv-> aggregate table -name user_test -count -sum age -avg age
Row count: 3
Sum:
        age(3 values): 96
Average:
        age(3 values): 32.00

# Get count(*), sum(age) and avg(age) of rows where
age >= 30, idx1 is utilized to filter the rows:
kv-> aggregate table -name user_test -count -sum age
-avg age -index idx1 -field age -start 30
Row count: 2
Sum:
        age(2 values): 75
Average:
        age(2 values): 37.50
```

await-consistent

```
await-consistent -timeout <timeout-secs> [-zn <id> | -znname <name> ]...
[-replica-delay-threshold <time-millis>]
```

Waits for up to the specified number of seconds for the replicas in one or more zones, or in the entire store, to catch up with the masters in their associated shards. Prints information about whether consistency was achieved or, if not, details about which nodes failed to become consistent.

where:

- `-timeout`

  Specifies the number of seconds for the replicas to catch up with the masters in their associated shards.

- `-zn <id>`

  Specifies the zone name to restrict the zones whose replicas need to satisfy the requested consistency requirements. If this option is not specified, all replicas must meet the consistency requirements.

- `-znname <name>`

  Specifies the zone name to restrict the zones whose replicas need to satisfy the requested consistency requirements. If this option is not specified, all replicas must meet the consistency requirements.

- `-replica-delay-threshold <time-millis>`

  Specifies the maximum number of milliseconds that a replica may be behind the master and be considered caught up. The default if 1000 milliseconds (1 second).

When performing a switchover, you can use this command to wait for secondary nodes to catch up with their masters, and to obtain information about progress towards reaching consistency.

change-policy

```
change-policy [-dry-run] -params [name=value]*
```

Modifies store-wide policy parameters to services you have not yet deployed. You use this command to change a policy to set new parameters that all new Replication Nodes will use. These policy parameters are the default set of parameters for the new services. All new services will use the parameters specified in the policy when they start. Specify the parameters to change after the `-params` flag, separating each parameter with a space character.

To specify parameter values that include embedded spaces, use quotation marks (") around the value, like this:

```
name="value with spaces"
```

If you use `-dry-run`, the command returns the parameters you specify without changing them.

For more information on setting policy parameters, see Setting Store Wide Policy Parameters.

configure

```
configure -name <storename> -json
```

Configures a new store. This call must be made before any other administration can be performed.

Use the `-name` option to specify the name of the KVStore that you want to configure. The name is used to form a path to records kept in the store. For this reason, you should avoid using characters in the store name that might interfere with its use within a file path. The command line interface does not allow an invalid store name. Valid characters are alphanumeric, '-', '_', and '.'.

```
kv-> configure -name mystore -json{
"operation" : "configure",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "storeName" : "mystore"
    }
}
```

connect

Encapsulates commands that connect to the specified host and registry port to perform administrative functions or connect to the specified store to perform data access functions.

The current store, if any, will be closed before connecting to another store. If there is a failure opening the specified KVStore, the following warning is displayed: "Warning: You are no longer connected to KVStore".

The subcommands are as follows:

- connect admin
- connect store

connect admin

```
connect admin -host <hostname> -port <registry port>
[-username <user>] [-security <security-file-path>]
```

Connects to the specified host and registry port to perform administrative functions. An Admin service must be active on the target host. If the instance is secured, you may need to provide login credentials.

where:

- `-host <hostname>`

  Identifies the host name of a node in your store.

- `-port <registry port>`

The TCP/IP port on which Oracle NoSQL Database should be contacted. This port should be free (unused) on each node. It is sometimes referred to as the registry port.

- `-username <user>`

  Specifies a username to log on as in a secure deployment.

- `-security <security-file-path>`

  In a secured deployment, specifies a path to the security file. If not specified in a secure store, updating the sn-target-list will fail.

connect store

```
connect store [-host <hostname>] [-port <port>]
-name <storename> [-timeout <timeout ms>]
[-consistency <NONE_REQUIRED(default)
                                    | ABSOLUTE| NONE_REQUIRED_NO_MASTER>]
[-durability <COMMIT_SYNC(default)
                                | COMMIT_NO_SYNC | COMMIT_WRITE_NO_SYNC>]
[-username <user>] [-security <security-file-path>]
```

Connects to a KVStore to perform data access functions. If the instance is secured, you may need to provide login credentials.

Use the timeout, consistency and durability flags to override the default connect configuration.

where:

- `-host <hostname>`

  Identifies the host name of a node in your store.

- `-port <port>`

  The TCP/IP port on which Oracle NoSQL Database should be contacted. This port should be free (unused) on each node.

- `-name <storename>`

  Identifies the name of the store.

- `-timeout <timeout ms>`

  Specifies the store request timeout in milliseconds.

- `-consistency`

  Specifies the store request consistency. The default value is NONE_REQUIRED.

- `-durability`

  Specifies the store request durability. The default value is COMMIT_SYNC.

- `-username <user>`

  Specifies a username to log on as in a secure deployment.

- `-security <security-file-path>`

  In a secured deployment, specifies a path to the security file.

delete

Encapsulates commands that delete key/value pairs from store or rows from table. The subcommands are as follows:

- delete kv
- delete table

delete kv

```
delete kv [-key <key>] [-start prefixString] [-end prefixString] [-all]
```

Deletes one or more keys. If `-all` is specified, delete all keys starting at the specified key. If no key is specified, delete all keys in the store. The `-start` and `-end` flags can be used for restricting the range used for deleting.

For example, to delete all keys in the store starting at root:

```
kv -> delete kv -all
301 Keys deleted starting at root
```

delete table

```
kv-> delete table -name <table_name>
 [-field <name> -value <value>]*
 [-field <name> [-start <value>] [-end <value>]]
 [-ancestor <name>]* [-child <child_name>]*
 [-json <string>] [-delete-all]
```

Deletes one or multiple rows from the named table.

- `-name`
  Identifies a table name, which can be any of the following:
  - *`table_name`* – The target table is a top level table created in the default namespace, `sysdefault`. The default namespace (`sysdefault:`) prefix is not required to identify such tables.
  - *`table_name.child_name`* – The target table is the child of a parent table. Identify the child table by preceding it with the parent *`table_name`*, followed by a period (.) separator before *`child_name`*.
  - *`namespace_name:table_name`* – The target table was not created in the default (`sysdefault`) namespace. Identify *`table_name`* by preceding it with its *`namespace_name`*, followed by a colon (:).
  - *`namespace_name:table_name.child_name`* – The target table is the child of a parent table that was created in a namespace. Identify *`child_name`* by preceding it with both *`namespace_name`*: and the parent *`table_name`*, , followed by a period (.) separator.
- `-field` and `-value`

  Pairs specify the field values of the primary key or, use an empty key to delete all rows from the table.

- `-field` , `-start`, and `-end`

  Use these flags to restrict the sub-range for deletion associated with the parent key.

- `-ancestor` and `-child`

  Use to delete rows from a specific ancestor or descendant tables, in addition to the target table.

- `-json`

  Indicates that the key field values are in JSON format.

- `-delete-all`

  Indicates to delete all rows in a table.

execute

```
execute <statement> [-json] [-wait]
```

Oracle NoSQL Database provides a way to run Data Definition Language (DDL) statements used to form table and index statements. Using the `execute` command runs each statement you specify synchronously. You must enclose each DDL statement in single or double quotes. You must connect to a database store before using the execute command.

> **Note:**
>
> All DDL commands from the Admin CLI, including `execute`, are deprecated. Use the SQL for Oracle NoSQL Database Shell to execute this command. For more information, see Appendix A Introduction to the SQL for Oracle NoSQL Database Shell.

For example:

```
kv-> plan execute -id 19 -json -wait
{
    "operation" : "plan deploy-zone -name zn6 -rf 1 -type PRIMARY -no-
arbiters -no-master-affinity",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "id" : 19,
        "name" : "Deploy Zone",
        "isDone" : true,
        "state" : "SUCCEEDED",
        "start" : "2024-04-05 09:35:31 UTC",
        "interrupted" : null,
        "end" : "2024-04-05 09:35:31 UTC",
        "error" : null,
        "executionDetails" : {
            "taskCounts" : {
            "total" : 1,
            "successful" : 1,
            "failed" : 0,
            "interrupted" : 0,
```

```
            "incomplete" : 0,
            "notStarted" : 0
            },
        "finished" : [ {
            "taskNum" : 1,
            "name" : "Plan 19 [Deploy Zone] task [DeployDatacenter zone=zn6]",
            "state" : "SUCCEEDED",
            "start" : "2024-04-05 09:35:31 UTC",
            "end" : "2024-04-05 09:35:31 UTC"
            } ],
        "running" : [ ],
        "pending" : [ ]
        },
        "planId" : 19,
        "zoneName" : "zn6",
        "zoneId" : "zn4",
        "type" : "PRIMARY",
        "rf" : 1,
        "allowArbiters" : false,
        "masterAffinity" : false
        }
    }
```


exit

```
exit | quit
```

Exits the interactive command shell.

get

Encapsulates commands that get key/value pairs from store or get rows from table. The subcommands are as follows:

- get kv
- get table

get kv

```
get kv [-key <keyString>] [-file <output>] [-all] [-keyonly]
[-valueonly] [-start <prefixString>] [-end <prefixString>]
```

Perform a simple get operation using the specified key. The obtained value is printed out if it contains displayable characters, otherwise the bytes array is encoded using Base64 for display purposes. "[Base64]" is appended to indicate this transformation. The arguments for the get command are:

- `-key <keyString>`

  Indicates the full or the prefix key path to use. If `<keyString>` is a full key path, it returns a single value information. The format of this `get` command is: `get -key <keyString>`. If `<keyString>` is a prefix key path, it returns multiple key/value pairs. The format of this `get` command is: `get -key <keyString> -all`. Key can be composed of both major and

minor key paths, or a major key path only. The <keyString> format is: "major-key-path/-/minor-key-path". Additionally, in the case of the prefix key path, a key can be composed of the prefix part of a major key path.

For example, with some sample keys in the KVStore:

```
/group/TC/-/user/bob
/group/TC/-/user/john
/group/TC/-/dep/IT
/group/SZ/-/user/steve
/group/SZ/-/user/diana
```

A get command with a key containing only the prefix part of the major key path results in:

```
kv -> get kv -key /group -all -keyonly
/group/TC/-/user/bob
/group/TC/-/user/john
/group/TC/-/dep/IT
/group/SZ/-/user/steve
/group/SZ/-/user/diana
```

A get command with a key containing a major key path results in:

```
kv -> get kv -key /group/TC -all -keyonly
/group/TC/-/user/bob
/group/TC/-/user/john
/group/TC/-/dep/IT
```

Get commands with a key containing major and minor key paths results in:

```
kv -> get kv -key /group/TC/-/user -all -keyonly
/group/TC/-/user/bob
/group/TC/-/user/john
kv -> get kv -key /group/TC/-/user/bob
{
    "name"  : "bob.smith",
    "age"   : 20,
    "email" : "bob.smith@example.com",
    "phone" : "408 555 5555"
}
```

- `-file <output>`

  Specifies an output file, which is truncated, replacing all existing content with new content.

  In the following example, records from the key `/Smith/Bob` are written to the file `"data.out"`.

  ```
  kv -> get kv -key /Smith/Bob -all -file ./data.out
  ```

  In the following example, contents of the file `"data.out"` are replaced with records from the key `/Wong/Bill`.

  ```
  kv -> get kv -key /Wong/Bill -all -file ./data.out
  ```

- `-all`

  Specified for iteration starting at the specified key. If the key argument is not specified, the entire store will be iterated.

- `-keyonly`

  Specified with `-all` to return only keys.

- `-valueonly`

  Specified with `-all` to return only values.

- `-start <prefixString>` and `-end <prefixString>`

  Restricts the range used for iteration. This is particularly helpful when getting a range of records based on a key component, such as a well-formatted string. Both the `-start` and `-end` arguments are inclusive.

  > **Note:**
  >
  > `-start` and `-end` only work on the key component specified by `-key <keyString>`. The value of `<keyString>` should be composed of simple strings and cannot have multiple key components specified.

  For example, a log where its key structure is:

  ```
  /log/<year>/<month>/-/<day>/<time>
  ```

  puts all log entries for the same day in the same partition, but splits the days across shards. The `time` format is: "hour.minute".

  In this way, you can do a `get` of all log entries in February and March, 2013 by specifying:

  ```
  kv-> get kv -all -keyonly -key /log/2013 -start 02 -end 03
  /log/2013/02/-/01/1.45
  /log/2013/02/-/05/3.15
  /log/2013/02/-/15/10.15
  /log/2013/02/-/20/6.30
  /log/2013/02/-/28/8.10
  /log/2013/03/-/01/11.13
  /log/2013/03/-/15/2.28
  /log/2013/03/-/22/4.52
  /log/2013/03/-/31/11.55
  ```

  You can be more specific to the `get` command by specifying a more complete key path. For example, to display all log entries from April 1st to April 4th:

  ```
  kv-> get kv -all -keyonly -key /log/2013/04 -start 01 -end 04
  /log/2013/04/-/01/1.03
  /log/2013/04/-/01/4.05
  /log/2013/04/-/02/7.22
  /log/2013/04/-/02/9.40
  /log/2013/04/-/03/4.15
  /log/2013/04/-/03/6.30
  ```

```
/log/2013/04/-/03/10.25
/log/2013/04/-/04/4.10
/log/2013/04/-/04/8.35
```

See the subcommand get table

get table

```
kv-> get table -name <table_name> [-index <name>]
    [-field <name> -value <value>]+
    [-field <name> [-start <value>] [-end <value>]]
    [-ancestor <name>]+ [-child <name>]+
    [-json <string>] [-file <output>] [-keyonly]
    [-pretty] [-report-size]
```

Identifies a table name, which can be any of the following:

- `-name`
  Identifies any of the following tables:
  - *`table_name`* – The target table is a top level table created in the default namespace, `sysdefault`. The default namespace (`sysdefault:`) prefix is not required to identify such tables.
  - *`table_name.child_name`* – The target table is the child of a parent table. Identify the child table by preceding it with the parent *`table_name`*, followed by a period (.) separator before *`child_name`*.
  - *`namespace_name`*`:`*`table_name`* – The target table was not created in the default (`sysdefault`) namespace. Identify *`table_name`* by preceding it with its *`namespace_name`*, followed by a colon (:).
  - *`namespace_name`*`:`*`table_name.child_name`* – The target table is the child of a parent table that was created in a namespace. Identify *`child_name`* by preceding it with both *`namespace_name`*`:` and the parent *`table_name`*, , followed by a period (.) separator.

  `-field` and `-value` pairs are used to specify the field values of the primary key or index key if using an index, specified by `-index`, or with an empty key to iterate the entire table.
- `-field` flag, along with its `-start` and `-end` flags, can be used to define a value range for the last field specified.
- `-ancestor` and `-child` flags are used to return results from specific ancestor and/or descendant tables as well as the target table.
- `-json` indicates that the key field values are in JSON format.
- `-file` is used to specify an output file, which is truncated.
- `-keyonly` is used to restrict information to keys only.
- `-pretty` is used for a nicely formatted JSON string with indentation and carriage returns.
- `-report-size` is used to show key and data size information for primary keys, data values, and index keys for matching records. When `-report-size` is specified no data is displayed.

help

```
help [command [sub-command]] [-include-deprecated]
```

Prints help messages. With no arguments the top-level shell commands are listed. With additional commands and sub-commands, additional detail is provided.

```
kv-> help load
Usage: load -file <path to file>
    Load the named file and interpret its contents as a script of
    commands to be executed.  If any command in the script fails
    execution will end.
```

Use `-include-deprecated` to show deprecated commands.

For example:

```
kv-> help show -include-deprecated
Encapsulates commands that display the state of the store and its
components.
Usage: show admins |
            datacenters |
            events |
            faults |
            indexes |
            parameters |
            perf |
            plans |
            pools |
            schemas |
            snapshots |
            tables |
            topology |
            upgrade-order |
            users |
            versions |
            zones
```

hidden

```
hidden [on|off]
```

Toggles visibility and setting of parameters that are normally hidden. Use these parameters only if advised to do so by Oracle Support.

history

```
history [-last <n>] [-from <n>] [-to <n>]
```

Displays command history. By default all history is displayed. Optional flags are used to choose ranges for display.

load

```
load -file <path to file>
```

Loads the named file and interpret its contents as a script of commands to be executed. If any of the commands in the script fail, execution will stop at that point.

For example, users of the Table API can use the load command to define a table and insert data using a single script. Suppose you have a table defined like this:

```
create table IF NOT EXISTS Users (
  id integer,
  firstname string,
  lastname string,
  age integer,
  income integer,
  primary key (id)
);
```

Then sample data for that table can be defined using JSON like this:

```
{
"id":1,
"firstname":"David",
"lastname":"Morrison",
"age":25,
"income":100000
}
{
"id":2,
"firstname":"John",
"lastname":"Anderson",
"age":35,
"income":100000
}
{
"id":3,
"firstname":"John",
"lastname":"Morgan",
"age":38,
"income":200000
}
{
"id":4,
"firstname":"Peter",
"lastname":"Smith",
"age":38,
"income":80000
}
{
"id":5,
```

```
"firstname":"Dana",
"lastname":"Scully",
"age":47,
"income":400000
}
```

Assume that the sample data is contained in a file called `Users.json`. Then you can define the table and load the sample data using a script that looks like this (file name `loadTable.txt`) :

```
### Begin Script ###
execute "create table IF NOT EXISTS Users ( \
  id integer, \
  firstname string, \
  lastname string, \
  age integer, \
  income integer, \
  primary key (id) \
)"

put table -name Users -file users.json
```

Then, the script can be run by using the `load` command:

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar runadmin -host node01 -port 5000 \
-security \
KVROOT/securtiy/client.security \
-store mystore
kv-> load -file ./loadTable.txt
Statement completed successfully
Loaded 5 rows to Users

kv->
```

If you are using the Key/Value API, first you create schema in the store:

```
{
     "type": "record",
     "name": "ContactInfo",
     "namespace": "example",
     "fields": [
         {"name": "phone", "type": "string", "default": ""},
         {"name": "email", "type": "string", "default": ""},
         {"name": "city", "type": "string", "default": ""}
     ]
}
```

Then you can collect the following commands in the script file `load-contacts-5.txt`:

```
### Begin Script ###
put -key /contact/Bob/Walker -value "{\"phone\":\"857-431-9361\", \
\"email\":\"Nunc@Quisque.com\",\"city\":\"Turriff\"}" \
-json example.ContactInfo
```

```
put -key /contact/Craig/Cohen -value "{\"phone\":\"657-486-0535\", \
\"email\":\"sagittis@metalcorp.net\",\"city\":\"Hamoir\"}" \
-json example.ContactInfo
put -key /contact/Lacey/Benjamin -value "{\"phone\":\"556-975-3364\", \
\"email\":\"Duis@laceyassociates.ca\",\"city\":\"Wasseiges\"}" \
-json example.ContactInfo
put -key /contact/Preston/Church -value "{\"phone\":\"436-396-9213\", \
\"email\":\"preston@mauris.ca\",\"city\":\"Helmsdale\"}" \
-json example.ContactInfo
put -key /contact/Evan/Houston -value "{\"phone\":\"028-781-1457\", \
\"email\":\"evan@texfoundation.org\",\"city\":\"Geest-G\"}" \
-json example.ContactInfo
exit
### End Script ###
```

The script can be run by using the `load` command:

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar runadmin -host node01 -port 5000 \
-security \
KVROOT/securtiy/client.security \
-store mystore
kv-> load -file ./load-contacts-5.txt
Operation successful, record inserted.
Operation successful, record inserted.
Operation successful, record inserted.
Operation successful, record inserted.
Operation successful, record inserted.
```

For more information on using the load command, see Create a script to configure the data store.

logtail

Monitors the store-wide log file until interrupted by an "enter" key press.

namespace

```
namespace [namespace_name]
```

Sets *namespace_name* as the default namespace for table operations and queries. For example:

```
kv-> namespace ns1
Namespace is ns1
```

Entering the command without *namespace_name* returns to the default namespace:

```
kv-> namespace
Namespace is sysdefault
```

page

```
page [on|<n>|off]
```

Turns query output paging on or off. If specified, `n` is used as the page height.

If `n` is 0, or "on" is specified, the default page height is used. Setting `n` to "off" turns paging off.

ping

```
ping [-json] [-shard <shardId>]
```

The `ping` and `verify` commands return information about the runtime entities of a data store. The command accesses components and Admin services available from the topology, returning information about the state of various components.

- `-json`

  Displays output in JSON format.

- `-shard <shardId>`

  Displays a subset of status information about the specific shard ID you supply.

Here is a basic example of calling ping from the Admin CLI:

```
kv-> ping
Pinging components of store mystore based upon topology sequence #308
300 partitions and 3 storage nodes
Time: 2024-04-05 20:19:27 UTC   Version: 24.1.11
Shard Status: healthy:1 writable-degraded:0 read-only:0 offline:0 total:1
Admin Status: healthy
Zone [name=1 id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
RN Status: online:3 read-only:0 offline:0
maxDelayMillis:0 maxCatchupTimeSecs:0
Storage Node [sn1] on localhost:13230
Zone: [name=1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 08:17:52 UTC  Build id:
12641466031c Edition: Enterprise
Admin [admin1]        Status: RUNNING,MASTER
Rep Node [rg1-rn1]    Status: RUNNING,MASTER sequenceNumber:633 haPort:13233
available storage size:109 GB
Storage Node [sn2] on localhost:13240
Zone: [name=1 id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 08:17:52 UTC  Build id:
12641466031c Edition: Enterprise
Admin [admin2]          Status: RUNNING,REPLICA
    Rep Node [rg1-rn2]     Status: RUNNING,REPLICA sequenceNumber:633
haPort:13243 available storage size:109 GB delayMillis:0 catchupTimeSecs:0
Storage Node [sn3] on localhost:13250    Zone: [name=1 id=zn1 type=PRIMARY
allowArbiters=false masterAffinity=false]    Status: RUNNING   Ver: 24.1.11
2024-04-05 08:17:52 UTC  Build id: 12641466031c Edition: Enterprise
```

```
    Admin [admin3]      Status: RUNNING,REPLICA
    Rep Node [rg1-rn3]      Status: RUNNING,REPLICA sequenceNumber:633
haPort:13253 available storage size:109 GB delayMillis:0 catchupTimeSecs:0
```

About Shard and Admin Status

After running a `ping` command, you should understand what is most useful (or troubling) about the system health. The most important content is the *Shard Status* entry. The following `ping` output details indicate one shard (`total:1`) that is healthy (`healthy:1`). All of the status types you'd prefer not to see (`writable-degraded`, `read-only`, and `offline` are zero (`0`), indicating nothing has one of those states. Everything is good.

```
Shard Status: healthy:1
writable-degraded:0
read-only:0
offline:0
total:1
```

What exactly does a healthy shard indicate? A healthy shard is one with all of its RNs running. Thus, if all shards in the topology are healthy, then all RNs are running, and no failures exist. Why are RNs so important? Because they are the components that perform read and write data operations.

Checking the Admin nodes status is also useful. In this simple example, only one Admin shard exists, so there is a single result: `Admin Status: healthy`. Other possible states are: `writable-degraded`, `read-only`, or `offline`.

For both RN shards and admins, these are what each result indicates:

Result	Meaning
`healthy`	All nodes are running, and the system is fully operational.
`writable-degraded`	A majority of the nodes are running. All operations are supported, but a minority of the nodes are offline or don't support writes. If you are using RF=3, this state is one step closer to being unable to support all operations. For example, with one node offline, losing another node means quorum will be lost, and the shard becomes read-only. Most people use RF=3, so this is typically what writable-degraded means.
`read-only`	Only a minority of the nodes are running. Read operations are supported, but write operations are not.
`offline`	No nodes are running, so no operations are supported.

About Zone Status

The next information from **ping** is about zones:

```
Zone [name=1 id=zn1
type=PRIMARY
allowArbiters=false
masterAffinity=false]
RN Status: online:3 read-only:0 offline:0
maxDelayMillis:0
maxCatchupTimeSecs:0
```

For stores with multiple zones, this information provides the status of nodes in different locations. For example, if a store was deployed using three zones, with the machines for each zone in a separate building, this information gives a quick summary status for machines in each building. In this simple example, there is only one zone, so that status information is similar to that for the entire store. The `maxDelayMillis` and `maxCatchupTimeSecs` entries provide information about data replication to replicas located in the zone. In our example, both values are zero (0). However, having large numbers for these entries could suggest that there are hardware problems with the machines in the zone, or problems with the network that connects that zone to other zones. Such information would be used only for more detailed debugging.

About Storage Nodes

Next, there is information about the nodes associated with a particular storage node:

```
Storage Node [sn1] on localhost:13230 Zone:
[name=1 id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
Status: RUNNING Ver: 24.1.11 2024-04-05 08:17:52 UTC Build id: 12641466031c
Edition: Enterprise
Admin [admin1] Status: RUNNING,MASTER
Rep Node [rg1-rn1] Status: RUNNING,MASTER
sequenceNumber:633 haPort:13233 available storage size:109 GB
```

The `Status:` entry for the SN can have several possible values:

Status	Description
`STARTING`	The storage node is starting up.
`WAITING_FOR_DEPLOY`	The storage node is running but is waiting to be deployed in a new store.
`RUNNING`	The storage node is running -- this is the usual state.
`STOPPING`	The storage node is in the process of stopping, but is not yet in a `STOPPED` status.
`STOPPED`	The storage node is stopped.
`UNREACHABLE`	The storage node is not reachable, either because the SN service is down, the host machine is offline, or the machine is not reachable over the network.

About RNs and Admins on the Storage Node

The next entries provide status information about RNs and any Admin processes that are running on the storage node. Not all storage nodes have admin nodes. The number of RNs running on the storage node depends on the SN capacity.

```
Admin [admin1] Status: RUNNING,MASTER Rep Node [rg1-rn1]
Status: RUNNING,MASTER sequenceNumber:633 haPort:13233 available storage
size:109 GB
```

The `Status:` entry for both admin nodes and RNs, can have the following values:

Status	Description
`STARTING`	The node is starting up.

Status	Description
RUNNING,MASTER	The node is up and is the master. The master is in contact with a majority of nodes in the shard, and can perform writes requiring acknowledgment. This is the first of two normal states.
RUNNING,REPLICA	The node is up and is a replica. This is the second of two normal states.
RUNNING,MASTER (non-authoritative)	The node is up and is the master, but is not in contact with a majority of nodes in the shard. A non-authoritative master can perform only writes that do not require acknowledgment.
STOPPING	The node is stopping.
UNREACHABLE	The node could not be contacted over the network. The node is either stopped, failed, or there is a problem with the network connection to the machine.
Additional status values that can be appended to the status line to provide more information:	
readonly requests enabled	The node is running in read-only mode because the plan enable-requests command was run to set the node into read-only user operations mode.
requests disabled	The node is running with all user operation requests disabled, because the plan enable-requests command was run to disable all requests on the node. The plan enable-requests command disables requests on a per-shard basis, so it will prevent writes or all operations on all data in the shard.

While not shown in the initial example, the ping and verify commands can display one of the following states for RNs and shards. The table describes their effects and outcomes:

Displayed State	Effects	Outcome
Unknown	Masters go down.	Represents the read-only state of the RNs and shards still running. Currently, we do not support read-only status for any RN.
Non-Authoritative Master	Replica nodes go down.	After Replica nodes are down, remaining RNs and shards are in read-only mode. Currently, we do not support read-only status for any RN.
Out of disk space	Masters and replica nodes go down. Replicas are left in the RUNNING, UNKNOWN state, and the masters are in the Non-Authoritative state.	When masters and replica nodes go down, any remaining RNs and shards are in read-only mode. Currently, we do not support read-only status for any RN.
Write requests disabled	RNs and shard health are in read-only enabled request state.	RNs and shards are unable to accept any user requests, and are marked offline.

Both the `ping` and `verify` commands detect these states. Following is the output of a ping command on a shard (`rg2`), in a normal state, showing how results are returned:

```
kv-> ping -shard rg2
Pinging components of store mystore based upon topology sequence #2376
shard rg2 500 partitions and 3 storage nodes Time: 2024-04-05 07:06:46 UTC
Version: 24.1.11
Shard Status: healthy: Admin Status: healthy Zone [name=shardzone id=zn1
type=PRIMARY
allowArbiters=false masterAffinity=false]
RN Status: online:3 offline:0 maxDelayMillis:0 maxCatchupTimeSecs:0
Storage Node [sn10] on nodeA:5000    Zone: [name=shardzone id=zn1
type=PRIMARY
allowArbiters=false masterAffinity=false] Status: RUNNING   Ver: 24.1.11
2024-04-05 09:33:45 UTC
Build id: a72484b8b33c Edition: Enterprise
        Rep Node [rg2-rn1]
        Status: RUNNING,MASTER sequenceNumber:71,166 haPort:5010
        available storage size:8 GB Storage Node [sn11] on nodeB:5000
        Zone: [name=shardzone id=zn1 type=PRIMARY
        allowArbiters=false masterAffinity=false]  Status: RUNNING   Ver:
24.1.11 2024-04-05 09:33:45 UTC
        Build id: a72484b8b33c Edition: Enterprise
        Rep Node [rg2-rn2]
        Status: RUNNING,REPLICA sequenceNumber:71,166 haPort:5011
        available storage size:4 GB delayMillis:0 catchupTimeSecs:0
Storage Node [sn12] on nodeC:5000    Zone: [name=shardzone id=zn1
type=PRIMARY
allowArbiters=false masterAffinity=false] Status: RUNNING   Ver: 24.1.11
2024-04-05 09:33:45 UTC
Build id: a72484b8b33c Edition: Enterprise
        Rep Node [rg2-rn3]
        Status: RUNNING,REPLICA sequenceNumber:71,166 haPort:5012
        available storage size:6 GB delayMillis:0 catchupTimeSecs:0
```

Following are examples of return information when different states occur.

- Shard status becomes writable-degraded and is read-only:

```
kv-> ping
Pinging components of store concurrent plan store based upon topology
sequence #1082
 1000 partitions and 9 storage nodes
Time: 2024-04-05 05:12:36 UTC Version: 24.1.11
        Shard Status: healthy:2 writable-degraded:12 read-only:4 offline:0
total:18
Admin Status: healthy
Zone [name=dc1 id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
RN Status: online:30 read-only:24 offline:0 maxDelayMillis:0
maxCatchupTimeSecs:0
Storage Node [sn1] on slcao397:5000
Zone: [name=dc1 id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
Status: RUNNING
Ver: 24.1.11 2024-04-05 11:36:43 UTC Build id: 6259xxxxxxxx Edition:
Enterprise
```

- RNs can have the `RUNNING,UNKNOWN` state for more than one reason, including reaching a disk limit, or when the RN is down:

```
Storage Node [sn4] on slcao400:5000 Zone: [name=dc1 id=zn1 type=PRIMARY
allowArbiters=false masterAffinity=false] Status: RUNNING Ver: 24.1.11
2024-04-05 11:36:43 UTC
Build id: 6259xxxxxxxx Edition: Enterprise
        Rep Node [rg7-rn1] Status: RUNNING,UNKNOWN sequenceNumber:173,717,825
haPort:5020
        available storage size:-3 GB delayMillis:? catchupTimeSecs:?
        Rep Node [rg8-rn1] Status: RUNNING,UNKNOWN sequenceNumber:173,555,937
haPort:5021
        available storage size:-3 GB delayMillis:? catchupTimeSecs:?
        Rep Node [rg9-rn1] Status: RUNNING,MASTER sequenceNumber:173,697,007
haPort:5022 available storage size:-3 GB
        Rep Node [rg10-rn1] Status: RUNNING,UNKNOWN
sequenceNumber:173,293,747 haPort:5023
        available storage size:-3 GB delayMillis:? catchupTimeSecs:?
        Rep Node [rg11-rn1] Status: RUNNING,UNKNOWN
sequenceNumber:170,561,758 haPort:5024 available storage size:-3 GB
        delayMillis:? catchupTimeSecs:?
        Rep Node [rg12-rn1] Status: RUNNING,MASTER sequenceNumber:170,410,483
haPort:5025 available storage size:-3 GB
```

- A running out of disk space error results in the master becoming `non-authoritative`:

```
 Storage Node [sn6] on slcao402:5000 Zone: [name=dc1 id=zn1 type=PRIMARY
allowArbiters=false
masterAffinity=false] Status: RUNNING Ver: 24.1.11 2024-04-05 11:36:43 UTC
Build id: 6259xxxxxxxx
Edition: Enterprise
Rep Node [rg7-rn3] Status: RUNNING,MASTER (non-authoritative)
sequenceNumber:173,754,579 haPort:5020 available storage size:45 GB
Rep Node [rg8-rn3] Status: RUNNING,REPLICA sequenceNumber:173,555,937
haPort:5021 available storage size:46 GB
delayMillis:0 catchupTimeSecs:0
Rep Node [rg9-rn3] Status: RUNNING,REPLICA
sequenceNumber:173,697,007 haPort:5022 available storage size:45 GB
delayMillis:0 catchupTimeSecs:0
Rep Node [rg10-rn3] Status: RUNNING,MASTER (non-authoritative)
sequenceNumber:173,293,747 haPort:5023 available storage size:45 GB
Rep Node [rg11-rn3] Status: RUNNING,REPLICA sequenceNumber:170,561,758
haPort:5024 available storage size:45 GB delayMillis:0 catchupTimeSecs:0
Rep Node [rg12-rn3] Status: RUNNING,REPLICA sequenceNumber:170,410,483
haPort:5025
available storage size:46 GB delayMillis:0 catchupTimeSecs:0
```

Finally, here is a basic example of calling `ping -json`:

```
kv-> ping -json
{
  "operation" : "ping",
  "returnCode" : 5000,
  "description" : "No errors found",
```

```
  "returnValue" : {
    "topology" : {
      "storeName" : "OurStore",
      "sequenceNumber" : 104,
      "numPartitions" : 100,
      "numStorageNodes" : 1,
      "time" : 1546801860520,
      "version" : "24.1.11"
    },
    "adminStatus" : "healthy",
    "shardStatus" : {
      "healthy" : 1,
      "writable-degraded" : 0,
      "read-only" : 0,
      "offline" : 0,
      "total" : 1
    },
    "zoneStatus" : [ {
      "resourceId" : "zn1",
      "name" : "OurZone",
      "type" : "PRIMARY",
      "allowArbiters" : false,
      "masterAffinity" : false,
      "rnSummaryStatus" : {
        "online" : 1,
        "offline" : 0,
        "read-only" : 0,
        "hasReplicas" : false
      }
    } ],
    "snStatus" : [ {
      "resourceId" : "sn1",
      "hostname" : "OurHost",
      "registryPort" : 5000,
      "zone" : {
        "resourceId" : "zn1",
        "name" : "OurZone",
        "type" : "PRIMARY",
        "allowArbiters" : false,
        "masterAffinity" : false
      },
      "serviceStatus" : "RUNNING",
      "version" : "24.1.11 2024-04-05 09:21:03 UTC  Build id: fbfbd1541004
Edition: Enterprise",
      "adminStatus" : {
        "resourceId" : "admin1",
        "status" : "RUNNING",
        "state" : "MASTER",
        "authoritativeMaster" : true
      },
      "rnStatus" : [ {
        "resourceId" : "rg1-rn1",
        "status" : "RUNNING",
        "requestsEnabled" : "ALL",
        "state" : "MASTER",
        "authoritativeMaster" : true,
```

```
        "sequenceNumber" : 381,
        "haPort" : 5013,
        "availableStorageSize" : "97 GB"
      } ],
      "anStatus" : [ ]
    } ],
    "exitCode" : 0
  }
}
```

You can also access the `ping` utility through Admin utility tools, available in `kvtool.jar`. For more information see ping.

plan

Encapsulates operations, or jobs that modify store state. All subcommands with the exception of interrupt and wait change persistent state. Plans are asynchronous jobs so they return immediately unless `-wait` is used. Plan status can be checked using `show plans`. The optional arguments for all plans include:

- `-wait`

  Wait for the plan to complete before returning.

- `-plan-name`

  The name for a plan. These are not unique.

- `-noexecute`

  Do not execute the plan. If specified, the plan can be run later using `plan execute`.

- `-force`

  Used to force plan execution and plan retry.

- `-json | -json-v1`

  Displays the plan output as json or json-v1. The `-json` flag can be used to output in the new json format. The `-json-v1` flag can be used to output in the json-v1 format. If you have an existing script that relies on an older version of JSON output, you may want to consider using `-json-v1` flag so that your existing scripts continue to function.

The subcommands are as described below.

- plan add-index
- plan add-table
- plan cancel
- plan change-parameters
- plan change-storagedir
- plan change-user
- plan create-user
- plan deploy-admin
- plan deploy-datacenter
- plan deploy-sn

plan add-index

```
plan add-index -name <name> -table <name> [-field <name>]*
    [-desc <description>]
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Adds an index to a table in the store.

where:

- `-name`

  Specifies the name of the index to add to a table.

- `-table`

Specifies the table name where the index will be added. The table name is a dot-separated name with the format `tableName[.childTableName]*`.

- `-field`

  Specifies the field values of the primary key.

plan add-table

```
plan add-table -name <name>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Adds a new table to the store. The table name is a dot-separated name with the format `tableName[.childTableName]*`.

Before adding a table, first use the `table create` command to create the named table. The following example defines a table (creates a table by name, adds fields and other table metadata).

```
## Enter into table creation mode
table create -name user -desc "A sample user table"
user->
user-> help
Usage: add-array-field |
add-field |
add-map-field |
add-record-field |
cancel |
exit |
primary-key |
remove-field |
set-description |
shard-key |
show
## Now add the fields
user-> help add-field
Usage: add-field -type <type> [-name <field-name> ] [-not-required]
[-nullable] [-default <value>] [-max <value>] [-min <value>]
[-max-exclusive] [-min-exclusive] [-desc <description>]
[-size <size>] [-enum-values <value[,value[,...]]
<type>: INTEGER, LONG, DOUBLE, FLOAT, STRING, BOOLEAN, DATE, BINARY, FIX
ED_BINARY, ENUM
## Adds a field. Ranges are inclusive with the exception of String,
## which will be set to exclusive.
user-> add-field -type Integer -name id
user-> add-field -type String -name firstName
user-> add-field -type String -name lastName
user-> help primary-key
Usage: primary-key -field <field-name> [-field <field-name>]*
## Sets primary key.
user-> primary-key -field id
## Exit table creation mode
user-> exit
## Table User built.
```

Use `table list -create` to see the list of tables that can be added. The following example lists and displays tables that are ready for deployment.

```
kv-> table list
## Tables to be added:
## User -- A sample user table
kv-> table list -name user
## Add table User:
{
  "type" : "table",
  "name" : "User",
  "id" : "User",
  "description" : "A sample user table",
  "shardKey" : [ "id" ],
  "primaryKey" : [ "id" ],
  "fields" : [ {
    "name" : "id",
    "type" : "INTEGER"
  }, {
    "name" : "firstName",
    "type" : "STRING"
  }, {
    "name" : "lastName",
    "type" : "STRING"
  } ]
}
```

The following example adds the table to the store.

```
## Add the table to the store.
kv-> help plan add-table
kv-> plan add-table -name user -wait
Executed plan 5, waiting for completion...
Plan 5 ended successfully
kv-> show tables -name user
{
  "type" : "table",
  "name" : "User",
  "id" : "r",
  "description" : "A sample user table",
  "shardKey" : [ "id" ],
  "primaryKey" : [ "id" ],
  "fields" : [ {
    "name" : "id",
    "type" : "INTEGER"
  }, {
    "name" : "firstName",
    "type" : "STRING"
  }, {
    "name" : "lastName",
    "type" : "STRING"
  } ]
  }
```

For more information and examples on table design, see Table Management in the *SQL Reference Guide*.

plan cancel

```
plan cancel -id <plan id> | -last - json
```

Cancels a plan that is not running. A running plan must be interrupted before it can be canceled.

Use `show plans` to list all plans that have been created along with their corresponding plan IDs and status.

Use the `-last` option to reference the most recently created plan.

```
kv-> plan cancel -id 23 -json
{
"operation" : "plan cancel|interrupt",
"returnCode" : 5000,
"description" : "Plan 23 was canceled",
"returnValue" : null
}
```

plan change-parameters

```
plan change-parameters -security | -service <id> |
    -all-rns [-zn <id> | -znname <name>] | -all-ans [-zn <id> |
    -znname <name>] | -all-admins [-zn <id> | -znname <name>]
    [-dry-run] [-plan-name <name>]
    [-json] [-wait] [-noexecute] [-force] -params [name=value]*
```

Changes parameters for either the specified service, or for all service instances of the same type that are deployed to the specified zone or all zones. This plan changes the parameters for all services that are running at the time of plan creation. If more services are added after plan creation, perhaps via a topology plan, then it is not guaranteed the new services will receive the new parameters.

The `-security` flag allows changing store-wide global security parameters, and should never be used with other flags.

The `-service` flag allows a single instance to be affected; and should never be used with either the `-zn` or `-znname` flag.

The `-all-*` flags can be used to change all instances of the service type. The parameters to change follow the `-params` flag and are separated by spaces. The parameter values with embedded spaces must be quoted; for example, name="value with spaces".

One of the -all-* flags can be combined with the -zn or -znname flag to change all instances of the service type deployed to the specified zone; leaving unchanged, any instances of the specified type deployed to other zones. If one of the -all-* flags is used without also specifying the zone, then the desired parameter change will be applied to all instances of the specified type within the store, regardless of zone.

If `-dry-run` is specified, the new parameters are returned without changing them. Use the command `show parameters` to see what parameters can be modified. For more information, see show parameters.

For more information on changing parameters in the store, see Setting Store Parameters.

If you want to change the parameter for all Replication Nodes, including the new ones that will be created in the future, do the following:

- Run a change-policy command to set the parameter values to be used when creating Replication Nodes in the future.
- Run the command `plan change-parameters -all-rns` to change the parameter values for existing Replication Nodes.

Running the commands in the above order will make sure that both new and existing Replication Nodes will have their parameters changed.

> **Note:**
>
> The `plan change-parameters` updates the store metadata database even if the component is not available. The component's configuration will be made consistent when the KVStore system detects an inconsistency.

```
kv-> plan change-parameters -service rg1-rn2 -json -wait -params
loggingConfigProps="oracle.kv.level=DEBUG"
{
    "operation" : "Change RepNode Params",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "id" : 20,
        "owner" : "root(id:u1)",
        "name" : "Change RepNode Params",
        "isDone" : true,
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 05:31:05 UTC",
        "interrupted" : null,
        "end" : "2017-09-28 05:31:10 UTC",
        "error" : null,
        "executionDetails" : {
            "taskCounts" : {
                "total" : 4,
                "successful" : 4,
                "failed" : 0,
                "interrupted" : 0,
                "incomplete" : 0,
                "notStarted" : 0
            },
        "finished" : [ {
            "taskNum" : 1,
            "name" : "Plan 20 [Change RepNode Params] task [WriteNewParams
rg1-rn2]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:31:05 UTC",
```

```
            "end" : "2017-09-28 05:31:06 UTC"
        }, {
            "taskNum" : 2,
            "name" : "Plan 20 [Change RepNode Params] task [StopNode rg1-
rn2]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:31:06 UTC",
            "end" : "2017-09-28 05:31:07 UTC"
        }, {
            "taskNum" : 3,
            "name" : "Plan 20 [Change RepNode Params] task [StartNode]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:31:07 UTC",
            "end" : "2017-09-28 05:31:07 UTC"
        }, {
            "taskNum" : 4,
            "name" : "Plan 20 [Change RepNode Params] task [WaitForNodeState
rg1-rn2 to reach RUNNING]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:31:07 UTC",
            "end" : "2017-09-28 05:31:10 UTC"
        } ],
        "running" : [ ],
        "pending" : [ ]
        }
    }
}
```

plan change-storagedir

```
plan change-storagedir -sn <id> -storagedir <path> -add | -remove
    [-storagedirsize <size>] [-plan-name <name>] [-json] [-wait] [-noexecute]
    [-force]
```

Adds or removes a storage directory on a Storage Node, for storing a Replication Node.

where:

- `-sn`

  Specifies the Storage Node where the storage directory is added or removed.

- `-storagedir`

  Specifies the path to the storage directory on a Storage Node for storing a Replication Node.

- `-add | -remove`

  Specifies to add (`-add`) the storage dir.

  Specifies to remove (`-remove`) the storage dir.

- `-storagedirsize`

  Specifies the size of the directory specified in `-storagedir`. This parameter is optional; however, it is an error to specify this parameter for some, but not all, storage directories.

Use of this parameter is recommended for heterogeneous installation environments where some hardware has more storage capacity than other hardware. If this parameter is specified for all storage directories, then the store's topology will place more data on the shards that offer more storage space. If this parameter is not used, then data is spread evenly across all shards.

The value specified for this parameter must be a long, optionally followed by a unit string. Accepted unit strings are: KB, MB, GB, and TB, corresponding to 1024, 1024^2, 1024^3, 1024^4 respectively. Acceptable strings are case insensitive. Valid delimiters between the long value and the unit string are " ", "-", or "_".

```
-storagedirsize 200 MB
-storagedirsize 4_tb
-storagedirsize 5000-Mb

kv-> plan change-storagedir -sn sn2 -storagedir /tmp/kvroot -add -json -
wait
{
"operation" : "Change Storage Node Params",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "id" : 21,
    "owner" : "root(id:u1)",
    "name" : "Change Storage Node Params",
    "isDone" : true,
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:33:14 UTC",
    "interrupted" : null,
    "end" : "2017-09-28 05:33:14 UTC",
    "error" : null,
    "executionDetails" : {
        "taskCounts" : {
        "total" : 1,
        "successful" : 1,
        "failed" : 0,
        "interrupted" : 0,
        "incomplete" : 0,
        "notStarted" : 0
    },
    "finished" : [ {
        "taskNum" : 1,
        "name" : "Plan 21 [Change Storage Node Params] task
[WriteNewSNParams sn2]",
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 05:33:14 UTC",
        "end" : "2017-09-28 05:33:14 UTC"
    } ],
"running" : [ ],
"pending" : [ ]
        }
    }
}
```

plan change-user

```
plan change-user -name <user name>
    [-disable | -enable] [-set-password [-password <new password>]
    [-retain-current-password]] [-clear-retained-password]
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Change a user with the specified name in the store. The -retain-current-password argument option causes the current password to be remembered during the -set-password operation as a valid alternate password for configured retention time or until cleared using -clear-retained-password. If a retained password has already been set for the user, setting password again will cause an error to be reported.

This command is deprecated. For more information see User Modification in the *Security Guide*.

plan create-user

```
plan create-user -name <user name>
    [-admin] [-disable] [-password <new password>]
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Create a user with the specified name in the store. The -admin argument indicates that the created user has full administrative privileges.

This command is deprecated. For more information, see User Creation in the *Security Guide*.

plan deploy-admin

```
plan deploy-admin -sn <id> [-plan-name <name>]
    [-wait] [-noexecute] [-force]
```

Deploys an Admin to the specified Storage Node. The admin type (PRIMARY/SECONDARY) is the same type as the zone the Storage Node is in.

For more information on deploying an admin, see Create an Administration Process on a Specific Storage Node.

```
kv-> plan deploy-admin -sn sn1 -json -wait
"operation" : "plan deploy-admin -sn 1",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "id" : 22,
    "owner" : "root(id:u1)",
    "name" : "Deploy Admin Service",
    "isDone" : true,
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:34:26 UTC",
    "interrupted" : null,
    "end" : "2017-09-28 05:34:27 UTC",
    "error" : null,
```

```
    "executionDetails" : {
        "taskCounts" : {
            "total" : 4,
            "successful" : 4,
            "failed" : 0,
            "interrupted" : 0,
            "incomplete" : 0,
            "notStarted" : 0
        },
    "finished" : [ {
            "taskNum" : 1,
            "name" : "Plan 22 [Deploy Admin Service] task [DeployAdmin admin1
on sn1]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:34:26 UTC",
            "end" : "2017-09-28 05:34:27 UTC"
        }, {
            "taskNum" : 2,
            "name" : "Plan 22 [Deploy Admin Service] task [WaitForAdminState
admin1 to reach RUNNING]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:34:27 UTC",
            "end" : "2017-09-28 05:34:27 UTC"
        }, {
            "taskNum" : 3,
            "name" : "Plan 22 [Deploy Admin Service] task
[UpdateAdminHelperHost admin1]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:34:27 UTC",
            "end" : "2017-09-28 05:34:27 UTC"
        }, {
            "taskNum" : 4,
            "name" : "Plan 22 [Deploy Admin Service] task [NewAdminParameters
refresh admin1 parameter state
            without restarting]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 05:34:27 UTC",
            "end" : "2017-09-28 05:34:27 UTC"
        } ],
    "running" : [ ],
    "pending" : [ ]
    },
    "planId" : 22,
    "resourceId" : "admin1",
    "snId" : "sn1"
    }
}
```


plan deploy-datacenter

Deprecated. See plan deploy-zone instead.

plan deploy-sn

```
plan deploy-sn -zn <id> | -znname <name> -host <host> -port <port>
    [-plan-name <name>] [-json] [-wait] [-noexecute] [-force]
```

Deploys the Storage Node at the specified host and port into the specified zone.

where:

- `-sn`

  Specifies the Storage Node to deploy.

- `-zn <id> | -znname <name>`

  Specifies the Zone where the Storage Node is going to be deployed.

- `-host`

  Specifies the host name where the Storage Node is going to be deployed.

- `-port`

  Specifies the port number of the host.

For more information on deploying your Storage Nodes, see Create the Remainder of your Storage Nodes.

```
kv-> plan deploy-sn -zn 1 -json -host localhost -port 10000 -wait
{
"operation" : "plan deploy-sn -zn 1 -host localhost -port 10000",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "id" : 25,
    "owner" : "root(id:u1)",
    "name" : "Deploy Storage Node",
    "isDone" : true,
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:40:50 UTC",
    "interrupted" : null,
    "end" : "2017-09-28 05:40:51 UTC",
    "error" : null,
    "executionDetails" : {
        "taskCounts" : {
            "total" : 1,
            "successful" : 1,
            "failed" : 0,
            "interrupted" : 0,
            "incomplete" : 0,
            "notStarted" : 0
        },
    "finished" : [ {
        "taskNum" : 1,
        "name" : "Plan 25 [Deploy Storage Node] task [DeploySN
sn4(localhost:10000)]",
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 05:40:50 UTC",
```

```
        "end" : "2017-09-28 05:40:51 UTC"
        } ],
    "running" : [ ],
    "pending" : [ ]
    },
    "planId" : 25,
    "resourceId" : "sn4",
    "zoneId" : "zn1",
    "host" : "localhost",
    "port" : 10000
    }
}
```

plan deploy-topology

```
plan deploy-topology -name <topology name> [-plan-name <name>]
    [-json] [-wait] [-noexecute] [-force]
```

Deploys the specified topology to the store. The KVStore size determines how long the command takes to deploy replication and arbiter nodes to become fully functional shard members. The `plan deploy-topology` command does not wait for this command to finish.

After running the `plan deploy-topology` command, use the `verify configuration` command to check the running state of the components in the topology. See Deploy the Topology Candidate.

```
kv-> plan deploy-topology -name MyStoreLayout -json -wait
{
"operation" : "plan deploy-topology -name MyStoreLayout",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "id" : 26,
    "owner" : "root(id:u1)",
    "name" : "Deploy Topo",
    "isDone" : true,
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:56:25 UTC",
    "interrupted" : null,
    "end" : "2017-09-28 05:56:26 UTC",
    "error" : null,
    "executionDetails" : {
    "taskCounts" : {
    "total" : 6,
    "successful" : 6,
    "failed" : 0,
    "interrupted" : 0,
    "incomplete" : 0,
    "notStarted" : 0
    },
"finished" : [ {
    "taskNum" : 1,
    "name" : "Plan 26 [Deploy Topo] task [UpdateDatacenterV2 zone=zn1]",
    "state" : "SUCCEEDED",
```

```
    "start" : "2017-09-28 05:56:25 UTC",
    "end" : "2017-09-28 05:56:25 UTC"
    }, {
    "taskNum" : 2,
    "name" : "Plan 26 [Deploy Topo] task [UpdateDatacenterV2 zone=zn2]",
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:56:25 UTC",
    "end" : "2017-09-28 05:56:25 UTC"
    }, {
    "taskNum" : 3,
    "name" : "Plan 26 [Deploy Topo] task [UpdateDatacenterV2 zone=zn3]",
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:56:25 UTC",
    "end" : "2017-09-28 05:56:25 UTC"
    }, {
    "taskNum" : 4,
    "name" : "Plan 26 [Deploy Topo] task [BroadcastTopo]",
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:56:25 UTC",
    "end" : "2017-09-28 05:56:26 UTC"
    }, {
    "taskNum" : 5,
    "name" : "Plan 26 [Deploy Topo] task [BroadcastMetadata]",
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:56:26 UTC",
    "end" : "2017-09-28 05:56:26 UTC"
    }, {
    "taskNum" : 6,
    "name" : "Plan 26 [Deploy Topo] task [BroadcastTopo]",
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 05:56:26 UTC",
    "end" : "2017-09-28 05:56:26 UTC"
    } ],
    "running" : [ ],
    "pending" : [ ]
    },
    "planId" : 26,
    "topoName" : "MyStoreLayout"
    }
}
```

plan deploy-zone

```
plan deploy-zone -name <zone name>
    -rf <replication factor>
    [-type [primary | secondary]]
    [-arbiters | -no-arbiters]
    [-json ]
    [-master-affinity | -no-master-affinity]
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Deploys the specified zone to the store and creates a primary zone if you do not specify a `-type`.

where:

- `-name`

  Specifies the name of the zone to deploy.

- `-rf`

  Specifies the replication factor of the zone.

- `-type`

  Specifies the type of the zone to deploy. It can be a primary or a secondary zone. If -type is not specified, a primary zone is deployed.

- `-json`

  Formats the command output in JSON.

- `-arbiters | -no-arbiters`

  If you specify `-arbiters`, you can allocate Arbiter Nodes on the Storage Node in the zone. You can specify this flag only on a primary zone.

  Specifying `-no-arbiters` precludes allocating Arbiter Nodes on the Storage Node in the zone.
  The default value is `-no-arbiters`.

- `-master–affinity | -no-master-affinity`

  Specifying `-master-affinity` indicates that this zone can host a master.

  Specifying `-no-master-affinity` indicates that this zone cannot host a master.
  The default value is `-no-master-affinity`.

For more information on creating a zone, see Create a Zone.

```
kv-> plan deploy-zone -name zn6 -rf 1 -json -wait
{
    "operation" : "plan deploy-zone -name zn6 -rf 1 -type PRIMARY -no-
arbiters -no-master-affinity",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "id" : 27,
        "owner" : "root(id:u1)",
        "name" : "Deploy Zone",
        "isDone" : true,
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 05:57:29 UTC",
        "interrupted" : null,
        "end" : "2017-09-28 05:57:29 UTC",
        "error" : null,
        "executionDetails" : {
            "taskCounts" : {
            "total" : 1,
            "successful" : 1,
            "failed" : 0,
            "interrupted" : 0,
            "incomplete" : 0,
            "notStarted" : 0
        },
```

```
    "finished" : [ {
        "taskNum" : 1,
        "name" : "Plan 27 [Deploy Zone] task [DeployDatacenter zone=zn6]",
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 05:57:29 UTC",
        "end" : "2017-09-28 05:57:29 UTC"
        } ],
        "running" : [ ],
        "pending" : [ ]
        },
    "planId" : 27,
    "zoneName" : "zn6",
    "zoneId" : "zn4",
    "type" : "PRIMARY",
    "rf" : 1,
    "allowArbiters" : false,
    "masterAffinity" : false
    }
}
```

plan deregister-es

```
plan deregister-es
```

Deregisters the Elasticsearch cluster from the Oracle NoSQL Database store, using the `deregister-es` plan command. This is allowed only if all full text indexes are first removed using the `plan remove-index` command, see plan remove-index.

For example:

```
kv-> plan deregister-es
Cannot deregister ES because these text indexes exist:
mytestIndex
JokeIndex
```

For more information, see Integration with Elastic Search for Full Text Search in the *Integrations Guide*.

plan drop-user

```
plan drop-user -name <user name>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Drop a user with the specified name in the store. A logged-in user may not drop itself.

This command is deprecated. For more information, see User Removal in the *Security Guide*.

plan enable-requests

This command will change the type of user requests supported by a set of shards or the entire store.

```
plan enable-requests
    -request-type {all|readonly|none}
    {-shards <shardId[,shardId]*> | -store}
    [-plan-name <name>] [-wait]
    [-noexecute] [-force]
    [-json|-json-v1]
```

Limit the type of requests enabled for specific shards or the whole store.

The `-request-type` flag configures the read and write requests. The following request types can be configured by this command.

- **all** means the store or shards can process both read and write requests;
- **readonly** makes the store or shards only respond to read requests;
- **none** means no read or write requests will be processed by the store or shards.

The `-shards` flag specifies the list of shards that should be configured, if you want the configuration to be done on one or more shards. You can get details about the `shardid` by executing the `show topology` command. The `rgXX` portion in the show topology output denotes the `shardid`. See show topology.

The `-store` flag specifies that the configuration to be done on the entire store.

You should specify either the `-shard` flag or the `-store` flag.

Example 5-1 plan enable-requests

For example, If you want to put the shard rg1 in readonly mode, you would specify `rg1` as the `shardid` and `readonly` as the `request-type`.

```
kv-> plan enable-requests
    -request-type readonly -shards rg1
Started plan 25. Use show plan -id 25 to check status.
    To wait for completion, use plan wait -id 25
```

Example 5-2 plan enable-requests

For example, If you want to put the whole store in readonly mode and to get the output in json format, you would specify the `store` attribute, `request-type` attribute as `readonly` and `json` attribute.

```
kv-> plan enable-requests
    -request-type readonly -store -json
{
  "operation" : "plan enable-requests",
  "returnCode" : 5000,
  "description" : "Operation ends successfully",
  "returnValue" : {
    "planId" : 26
```

```
  }
}
```

Example 5-3 plan enable-requests

For example, If you want to put the whole store in readonly mode and to get the output in json v1 format, you would specify the `store` attribute, `request-type` attribute as `readonly` and `json-v1` attribute.

```
kv-> plan enable-requests
    -request-type readonly -store -json-v1
{
  "operation" : "plan enable-requests",
  "return_code" : 5000,
  "description" : "Operation ends successfully",
  "return_value" : {
    "plan_id" : 27
  }
}
```

plan evolve-table

```
plan evolve-table -name <name>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Evolves a table in the store. The table name is a dot-separate with the format `tableName[.childTableName]*`.

Use the `table evolve` command to evolve the named table. The following example evolves a table.

```
## Enter into table evolution mode
kv-> table evolve -name User
kv-> show
{
  "type" : "table",
  "name" : "User",
  "id" : "r",
  "description" : "A sample user table",
  "shardKey" : [ "id" ],
  "primaryKey" : [ "id" ],
  "fields" : [ {
    "name" : "id",
    "type" : "INTEGER"
  }, {
    "name" : "firstName",
    "type" : "STRING"
  }, {
    "name" : "lastName",
    "type" : "STRING"
  } ]
}
## Add a field
kv-> add-field -type String -name address
```

```
## Exit table creation mode
kv-> exit
## Table User built.
kv-> plan evolve-table -name User -wait
## Executed plan 6, waiting for completion...
## Plan 6 ended successfully
kv-> show tables -name User
{
  "type" : "table",
  "name" : "User",
  "id" : "r",
  "description" : "A sample user table",
  "shardKey" : [ "id" ],
  "primaryKey" : [ "id" ],
  "fields" : [ {
    "name" : "id",
    "type" : "INTEGER"
  }, {
    "name" : "firstName",
    "type" : "STRING"
  }, {
    "name" : "lastName",
    "type" : "STRING"
  }, {
    "name" : "address",
    "type" : "STRING"
  } ]
}
```

Use `table list -evolve` to see the list of tables that can be evolved. For more information, see plan add-table .

plan execute

```
plan execute -id <id> | -last
    [-plan-name <name>] [-json] [-wait] [-noexecute] [-force]
```

Executes an existing plan that has not yet been executed. The plan must have been previously created using the `-noexecute` flag .

Use the `-last` option to reference the most recently created plan.

```
kv-> plan execute -id 19 -json -wait
{
    "operation" : "plan deploy-zone -name zn6 -rf 1 -type PRIMARY -no-
arbiters -no-master-affinity",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "id" : 19,
        "name" : "Deploy Zone",
        "isDone" : true,
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 09:35:31 UTC",
```

```
            "interrupted" : null,
            "end" : "2017-09-28 09:35:31 UTC",
            "error" : null,
            "executionDetails" : {
                "taskCounts" : {
                "total" : 1,
                "successful" : 1,
                "failed" : 0,
                "interrupted" : 0,
                "incomplete" : 0,
                "notStarted" : 0
            },
        "finished" : [ {
            "taskNum" : 1,
            "name" : "Plan 19 [Deploy Zone] task [DeployDatacenter zone=zn6]",
            "state" : "SUCCEEDED",
            "start" : "2017-09-28 09:35:31 UTC",
            "end" : "2017-09-28 09:35:31 UTC"
} ],
"running" : [ ],
"pending" : [ ]
},
"planId" : 19,
"zoneName" : "zn6",
"zoneId" : "zn4",
"type" : "PRIMARY",
"rf" : 1,
"allowArbiters" : false,
"masterAffinity" : false
}
}
```


plan failover

```
plan failover  { [-zn <zone-id> | -znname <zone-name>]
    -type [primary | offline-secondary] }...
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

where:

- `-zn <zone-id> | -znname <zone-name>`

  Specifies a zone either by zone ID or by name.

- `-type [primary | offline-secondary]`

  Specifies the new type for the associated zone.

Changes zone types to failover to either Primary or Secondary zones, whenever a primary zone failure results in a loss of quorum. Arbiters will not be created or removed from the topology. This command can introduce violations if a zone that contains Arbiters is specified as secondary-offline. Use the force flag if arbiter violations are introduced.

Zones whose new type is primary are taking over from failed primary zones to reestablish quorum. For these zones, a quorum of storage nodes in each shard in the zone must be available and responding to requests.

Zones whose new type is offline-secondary represent primary zones that are currently offline, resulting in the current loss of quorum. For these zones, all of the storage nodes in the zones must currently be unavailable. No zone type changes can be performed if these requirements are not met when the command starts.

> **Note:**
>
> Arbiter nodes are not currently supported during failover and switchover operations.

To correct any violations after the topology components are repaired, the `plan failover` command executes a `rebalance` command. To successfully deploy the new topology after a `rebalance`, the Storage Nodes hosting topology components must be running. If a Storage Node in a zone that failed over to a Secondary zone that contained an Arbiter, when the SN restarts, the Arbiter rejoins the shard.

You cannot execute this command when other plans are in progress for the data store. Before executing this plan, cancel or interrupt any other plans.

plan grant

```
plan grant [-role <role name>]* -user <user_name>
```

Allows granting roles to users.

where:

- `-role <role name>`

  Specifies the roles that will be granted. The role names should be the system-defined roles (except `public`) listed in the *Security Guide*.

- `-user <user_name>`

  Specifies the user who the role will be granted from.

This command is deprecated. For more information see Grant Roles or Privileges in the *Security Guide*.

plan interrupt

```
plan interrupt -id <plan id> | -last [-json]
```

Interrupts a running plan. An interrupted plan can only be re-executed or canceled. Use -last to reference the most recently created plan.

```
kv-> plan interrupt -id 20 -json
{
"operation" : "plan cancel|interrupt",
"returnCode" : 5000,
"description" : "Plan 20 was interrupted",
"returnValue" : null
}
```

plan migrate-sn

```
plan migrate-sn -from <id> -to <id>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Migrates the services from one Storage Node to another. The old node must not be running.

where:

- `-from`

  Specifies the Storage Node (old) that you are migrating from.

- `-to`

  Specifies the Storage Node (new) that you are migrating to.

For example, assuming that you are migrating from Storage Node 25 to 26, you would use:

```
kv-> plan migrate-sn -from sn25 -to sn26
```

Before executing the `plan migrate-sn` command, you can stop any running old Storage Node by using `-java -Xmx64m -Xms64m -jar KVHOME/lib/kvstore.jar stop -root KVROOT`.

plan network-restore

```
plan network-restore -from <id> -to <id> -retain-logs
    [-plan-name <name>] [-wait] [-noexecute] [-force] [-json|-json-v1]
```

The `plan network-restore` command restores a replication node (RN) with updates that the RN missed after losing networking connectivity. Use this only if the RN cannot be restored through the automatic procedures described here.

When a replication node becomes disconnected for any reason, it misses updates that occur while it was not connected. Oracle NoSQL Database uses two ways to update the recovered RN after it comes back online.

One way occurs within the RN's replication group. When the recovered RN returns, the replication group's master node streams all missed updates from the time the time the RN became disconnected, to the time it resumed operations.

Another way to restore a reconnected RN is over a network connection. Performing a network restore copies a complete set of data log files (`*.jdb`) from a peer, supplying the recovered RN with a comprehensive data set. The content contains many intermediate changes that are not reflected in the current store contents. This is because the data log files (`*.jdb`), which the recipient RN ingests, contain all changes, including any intermediate ones.

Do not confuse the data `*.jdb` log files, which contain data store activities, with the debug log files (`*.log`), which are used for debugging purposes.

If neither of the automatic Oracle NoSQL Database RN repopulation attempts succeed, it can be due to unforeseen circumstances, or a catastrophic situation that destroys data on multiple hosts. In this case, you can execute `plan network-restore` manually from the Admin CLI. However, doing so requires you to specify the RN that will supply the updated data.

You can attempt a network restore using the `plan network-restore` command from the admin CLI:

```
kv-> plan network-restore -help
Usage: plan network-restore -from <id> -to <id> [-retain-logs] \
[-plan-name <name>] [-wait] [-noexecute] [-force] [-json | json-v1]
Network restore a RepNode from another one in their replication group.
```

where:

- `-from` flag – Specifies the Replication Node ID from the same replication group (matching rgX). The `-from` node must be fully up to date, and able to supply the `*.dbd` log files to the destination RN. For example, if the `-to` recipient RN ID is `rg1-rn3`, and the ping output shows that `rg1-rn2` is the master, then that ID (`rg1-rn2`) is a good choice for the `-from` value.
- `-to` flag – Specifies the ID (rgX-rnY) of the recipient RN.
- `-retain-logs` flag – Retains obsolete log files on the lagging replica. The system renames the files, rather than deleting them. It is generally unnecessary to use this flag, unless you suspect that log files are corrupted on the recovering RN.

plan register-es

```
plan register-es -clustername <name> -host <host>
    -port <transport port> [-force]
```

Registers the Elasticsearch cluster with the Oracle NoSQL Database store, using the `register-es` plan command. It is only necessary to register one node of the cluster, as the other nodes in the cluster will be found automatically.

where:

- `-clustername`

  Specifies the name of the Elasticsearch cluster.

- `-host`

  Specifies the host name of a node in the cluster.

- `-port`

  Specifies the transport port of a node in the cluster.

For more information, see Integration with Elastic Search for Full Text Searchin the *Integrations Guide*.

plan remove-admin

```
plan remove-admin -admin <id> | -zn <id> | -znname <name>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Removes the desired Admin instances; either the single specified instance, or all instances deployed to the specified zone.

If you use the -admin flag and there are 3 or fewer Admins running in the store, or if you use the -zn or -znname flag and the removal of all Admins from the specified zone would result in only one or two Admins in the store, then the desired Admins will be removed only if you specify the -force flag.

Also, if you use the -admin flag and there is only one Admin in the store, or if you use the -zn or -znname flag and the removal of all Admins from the specified zone would result in the removal of all Admins from the store, then the desired Admins will not be removed.

plan remove-datacenter

```
plan remove-datacenter
```

This command is deprecated. See plan remove-zone instead.

plan remove-index

```
plan remove-index -name <name> -table <name>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Removes an index from a table.

where:

- -name

  Specifies the name of the index to remove.

- -table

  Specifies the table name to remove the index from. The table name is a dot-separated name with the format tableName[.childTableName]*.

plan remove-sn

```
plan remove-sn -sn <id>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Removes the specified Storage Node from the topology. The Storage Node is automatically stopped before removal.

This command is useful when removing unused, old Storage Nodes from the store. To do this, see Replacing a Failed Storage Node.

If the Storage Node is being removed as part of removing a secondary zone then,

- any replication nodes must first be removed using the topology change-replication-factor and plan deploy-topology commands, and
- any Admin Nodes must first be removed using plan remove-admin command.

plan remove-table

```
plan remove-table -name <name> [-keep-data]
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Removes a table from the store. The named table must exist and must not have any child tables. Indexes on the table are automatically removed. By default data stored in this table is also removed. Table data may be optionally saved by specifying the -keep-data flag. Depending on the indexes and amount of data stored in the table this may be a long-running plan.

The following example removes a table.

```
## Remove a table.
kv-> plan remove-table -name User
## Started plan 7. Use show plan -id 7 to check status.
## To wait for completion, use plan wait -id 7.
kv-> show tables
## No table found.
```

plan remove-zone

```
plan remove-zone -zn <id> | -znname <name>
    [-plan-name <name>] [-wait] [-noexecute] [-force]
```

Removes the specified zone from the store.

Before running this command, all Storage Nodes that belong to the specified zone must first be removed using the `plan remove-sn` command.

plan repair-topology

```
plan repair-topology
    [-plan-name <name>] [-wait] [-json] [-noexecute] [-force]
```

Inspects the store's deployed, current topology for inconsistencies in location metadata that may have arisen from the interruption or cancellation of previous deploy-topology or migrate-sn plans. Where possible, inconsistencies are repaired. This operation can take a while, depending on the size and state of the store.

```
kv-> plan repair-topology -json -wait
{
"operation" : "Repair Topology",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "id" : 25,
    "name" : "Repair Topology",
    "isDone" : true,
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 09:43:06 UTC",
    "interrupted" : null,
    "end" : "2017-09-28 09:43:06 UTC",
    "error" : null,
    "executionDetails" : {
        "taskCounts" : {
        "total" : 1,
        "successful" : 1,
```

```
        "failed" : 0,
        "interrupted" : 0,
        "incomplete" : 0,
        "notStarted" : 0
        },
    "finished" : [ {
        "taskNum" : 1,
        "name" : "Plan 25 [Repair Topology] task [VerifyAndRepair]",
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 09:43:06 UTC",
        "end" : "2017-09-28 09:43:06 UTC"
    } ],
    "running" : [ ],
    "pending" : [ ]
    }
    }
}
```

plan revoke

```
plan revoke [-role <role name>]* -user <user_name>
```

Allows revoking roles to users.

where:

- `-role <role name>`

  Specifies the roles that will be revoked. The role names should be the system-defined roles (except `public`) listed in the *Security Guide*.

- `-user <user_name>`

  Specifies the user who the role will be revoked from.

This command is deprecated. For more information see Revoke Roles or Privileges in the *Security Guide*.

plan start-service

```
plan start-service {-service <id> | -all-rns [-zn <id> |
    -znname <name>] | -all-ans [-zn <id> | -znname <name>] |
    -zn <id> | -znname <name> } [-plan-name <name>]
    [-json] [-wait] [-noexecute] [-force]
```

Starts the specified service(s). The service may be a Replication Node, an Arbiter Node, or Admin service, as identified by any valid string.

For example, to identify a Replication Node, use `-service shardId-nodeId`, where `shardId-nodeId` must be given as a single argument with one embedded hyphen and no spaces. The `shardId` identifier is represented by rgX, where X refers to the shard number.

where:

- `-service`

  Specifies the name of the service to start.

- `-all-rns`

  If specified, starts the services of all Replication Nodes in a store.

- `-all-ans`

  If specified, starts all the Arbiter Nodes in the specified zone.

Note:

This plan cannot be used to start a Storage Node. Further, you cannot restart the Storage Node's services without first starting the Storage Node itself. To start the Storage Node, go to the Storage Node host and enter the following command:

```
nohup java -Xmx64m -Xms64m \
-jar <KVHOME>/lib/kvstore.jar start -root <KVROOT> &

kv-> plan start-service -service rg1-rn3 -json -wait
{
    "operation" : "Start Services",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "id" : 21,
        "name" : "Start Services",
        "isDone" : true,
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 09:50:54 UTC",
        "interrupted" : null,
        "end" : "2017-09-28 09:50:57 UTC",
        "error" : null,"executionDetails" : {
        "taskCounts" : {
            "total" : 2,
            "successful" : 2,
            "failed" : 0,
            "interrupted" : 0,
            "incomplete" : 0,
            "notStarted" : 0
        },
    "finished" : [ {
        "taskNum" : 1,
        "name" : "Plan 21 [Start Services] task [StartNode]",
        "state" : "SUCCEEDED",
        "start" : "2017-09-28 09:50:54 UTC",
        "end" : "2017-09-28 09:50:55 UTC"
        }, {
    "taskNum" : 2,
    "name" : "Plan 21 [Start Services] task [WaitForNodeState rg1-rn3
to reach RUNNING]",
    "state" : "SUCCEEDED",
    "start" : "2017-09-28 09:50:55 UTC",
    "end" : "2017-09-28 09:50:57 UTC"
        } ],
    "running" : [ ],
    "pending" : [ ]
        }
    }
}
```

plan stop-service

```
plan stop-service {-service <id> |
    -all-rns [-zn <id> | -znname <name>] | -all-ans [-zn <id> |
    -znname <name>] | -zn <id> | -znname <name> }
    [-plan-name <name>] [-json] [-wait] [-noexecute] [-force]
```

Stops the specified service(s). The service may be a Replication Node, an Arbiter Node, or Admin service as identified by any valid string.

For example, to identify a Replication Node, use `-service` *`shardId-nodeId`*, where *`shardId-nodeId`* must be a single string with an embedded hyphen (-) and no spaces. The *`shardId`* identifier is represented as *`rgX`*, where *X* represents the shard number.

Other options to specify after `-service` include:

- `-all-rns`

  Stops the services of all Replication Nodes in a store.

- `-all-ans`

  Stops the services of all Arbiter Nodes in the specified zone.

Use this command to stop any affected services so that any attempts by the system to communicate with the services are no longer accepted. Stopping communication to one or more services reduces the amount of error output about a failure you are already aware of.

Whenever you execute the `plan stop-service` command, the system automatically initiates a health check. The health check determines if stopping an indicated service will result in losing quorum. There are no further checks performed, only whether quorum will be lost if you stop the service. To avoid losing quorum, the `plan stop-service` fails to execute if the health check fails, and outputs detailed health check information such as the following:

```
One of the groups is not healthy enough for the operation:
[rg1] Only 1 primary nodes are running such that a simple
majority cannot be formed which requires 2 primary nodes.
The shard is vulnerable and will not be able to elect a new master.
Nodes not running: [rg1-rn1]. Nodes to stop: {rg1=[rg1-rn2]}
...
```

If you cannot stop a service because it will result in lost quorum, you should determine what problem is occurring before trying to stop the service.

If, on the other hand, you understand that stopping a service will result in losing quorum, but such an event is necessary to make some important change, you can force the `plan stop-service` command to execute by appending the `-force` flag.

> **Note:**
>
> If you forcefully stop the Admin service and Admin quorum is lost, you cannot use the `start-service` plan to bring up the Admin services anymore. All plan operations will also fail thereafter.

The `plan stop-service` command is also useful during disk replacement process. Use the command to stop the affected service prior removing the failed disk. For more information, see Replacing a Failed Disk.

> **Note:**
>
> - This plan cannot be used to stop a Storage Node. To stop a Storage Node, first stop all services running on it. Then, find the ID of the Storage Node process by going to the Storage Node host and issuing this command:
>
>   ```
>   ps -af | grep -e "kvstore.jar.*start.*<KVROOT>"
>   ```
>
>   Kill the process using:
>
>   ```
>   kill <storage node id>
>   ```
>
> - Also, because the `plan stop-service -all-rns` command always results in losing quorum, executing `plan stop-service` with this option skips running a health check. Further, you do not need to use the `-force` flag is when using the `-all-rns` option.

plan update-tls-credentials

The `plan update-tls-credentials` command retrieves and installs the credential updates to the set of shared TLS (Transport Layer Security, earlier known as SSL) credentials used by Storage Node Agents (SNA) in the data store. You should use this plan only with data stores where all SNAs share the same credentials, and not for data stores with host-specific credentials.

If you do not specify any flags, the plan command does both the retrieve and install tasks. If you specify either `-retrieve-only` or `-install-only`, then only the corresponding action, either retrieve or install is performed. You cannot specify both flags together. The plan command requires the SYSOPER privilege, which is granted to the sysadmin built-in role by default.

```
plan update-tls-credentials  [-retrieve-only|-install-only] [-force]
```

where:

- `-retrieve-only`: This option instructs the plan command to only perform the retrieve action. For each SNA, the plan looks for a script named `security/updates/retrieve` under the SNA root directory. If the script is present, the SNA will invoke the script and the plan command waits for the script to be completed. The plan checks each SNA for a consistent set of installed credentials and pending updates. The plan also checks that the updates meet additional requirements that are intended to detect potential problems with the new credentials. More details on how this plan command works is given below.
- `-install-only`: This option only installs the updated credentials to the set of shared TLS credentials used by the SNA's in the data store. The store administrator should arrange to retrieve the updated credentials either with an earlier call to this command using the `-retrieve-only` option, or through a separate mechanism. The plan checks each SNA for a consistent set of installed credentials and pending updates. The plan also checks that the

updates meet additional requirements that are intended to detect potential problems with the new credentials. More details on how this plan command works is given below.

- `-force`: If this option is specified, the plan skips checking if the updates meet additional requirements that are intended to detect potential problems with the new credentials.

The first task the plan command does is to retrieve the updated credentials. The retrieve action is performed either when you specify no flags or when you specify the `retrieve-only` flag. For each SNA, the plan looks for an executable file named `security/updates/retrieve` under the SNA root directory (`$KVROOT`). By default, this file will not be present. The system administrator is responsible for creating the retrieve script, which should check for credential changes, retrieve any updated keystore and truststore files, and copy them into the updates directory (`$KVROOT/security/updates`). You should apply these credential changes to the store clients as needed, including the CLI admin clients and MR agents (if applicable), before you apply them to the data store.

The plan command waits for the `retrieve` script to be completed. If the `retrieve` script exits with a non-zero return code, the failure will be logged, along with any output to standard output and standard error, and the plan will fail. The `retrieve` script should finish in a short time because the plan command will only wait for the script to return for the amount of time specified by the `wait -id` parameter. The default value for this parameter is 5 minutes. If the retrieve script runs for more than that amount of time, the script will be killed and the plan will fail.

The plan then checks each SNA for a consistent set of installed credentials and pending updates. This check is performed in all cases regardless of whether the plan is performing both retrieval and installation, just retrieval, or just installation. The plan will fail in one of the following scenarios:

- If there are SNAs with different installed files and none have any updates.
- If any SNAs have updates, and either the updates differ or are not present on all SNAs that require updates.
- If any SNAs have uninstalled updates and the updates do not include both keystore and truststore files.

The plan then checks if the updates meet the additional requirements that are intended to detect potential problems with the new credentials. These checks will be skipped if the `-force` flag is specified. The names of keystore and truststore files that are checked will be the ones you specify using the `keystore` and `truststore` parameters. These are set by default to `store.keys` and `store.trust` respectively. See Additional verification while updating SSL Keys and Certificates for more details. If the updates fail to meet any of the additional verifications, the plan will fail unless the `-force` flag is specified.

If any of the above checks fail, the plan command exits with the error code. Else if you had specified the `-retrieve-only` flag, the plan would be completed successfully after this step. If you had specified no flags, the plan continues to the install step as explained below.

The installation of credentials will not modify the contents of the updates directory even after the updates have been installed. The `retrieve` script, or any manual process for adding updated credentials, should be prepared to handle the case where the update directory contains previous updates. The `retrieve` script can also store additional files in the update directory as needed to check if new credentials are available.

As part of installing new credentials, the plan command will first update the installed truststore on all SNAs, as needed, to include all the certificates in the new truststore, without modifying the value of the `truststoreSigPublicKeyAlias` parameter. Next, it will install the new keystore on all SNAs. Finally, for each SNA, it will install the new truststore.

The plan command will fail in one of the following scenarios:

- Unable to contact every SNA in the data store after repeated attempts. The SNAs need to be available initially to check for credential updates, and at each stage where changes are made to the SNAs. Other services, including Admins, Replication Nodes, and Arbiters, are not required to be online.
- If the set of SNAs in the data store is changed between when the plan is created and when it is executed. This is because the tasks associated with the plan are specific to the SNAs in the store topology.

plan verify-data

```
plan verify-data
    [-verify-log <enable|disable> [-log-read-delay <milliseconds>]]
    [-verify-btree <enable|disable> [-btree-batch-delay <milliseconds>]
        [-index <enable|disable>] [-datarecord <enable|disable>]]
    [-valid-time <time>]
    [-show-corrupt-files <enable|disable>]
    -service <id> | -all-services [-zn <id> | -znname <name>] |
    -all-rns [-zn <id> | -znname <name>] |
    -all-admins [-zn <id> | -znname <name>]
    [-plan-name <name>] [-wait] [-noexecute] [-force] [-json|-json-v1]
```

Verifies and controls certain elements (such as log files and indexes), as presented in this section. Here is a description for each of the `verify-data` parameters and options:

Option	Description
`-verify-log`	Verifies the checksum of each data record in the JE log file of JE. The Berkeley DB Java Edition (JE) is the data storage engine of Oracle NoSQL Database. It is enabled by default.
`-log-read-delay`	Configures the delay time between file reads. The default value is 100 milliseconds.
`-verify-btree`	Verifies that the B-tree of the database in memory contains a valid reference to each data record on disk. You can combine `-verify-btree` with `-datarecord` and `-index`. It is enabled by default.
`-btree-batch-delay`	Configures the delay time, in milliseconds, between batches. The default delay value is 10 milliseconds.
`-datarecord`	Reads and verifies data records from disk, if the data record is not in cache. The `-datarecord` option takes longer than verifying records only in cache, and results in more read I/O. It is disabled by default.
`-index`	Verifies indexes. Using the `-index` option alone verifies only the reference from the index to the primary table, not the reference from the primary table to index. To verify both references from index to primary table, and primary table to index, specify the `-datarecord` and `-index` options. It is enabled by default.

Option	Description
`-valid-time`	Specifies the amount of time for which an existing verification will be considered valid and not be rerun. The format is 'number unit' where the unit can be minutes or seconds. The unit is case insensitive and can be separated from the number by a space, "-" or "_". The default is '10 minutes'.
`-show-corrupt-files`	Specifies whether to show corrupt files, including missing files and reserved files that are referenced. It is disabled by default.
`-service` *id*	Runs verification on the specified service (*id*)
`-all-services [-zn` *id* `| -znname` *name*`]`	Runs verification on all services, both RNs and Admins, in the specified zone, or in all zones if none is specified.
`| -all-rns [-zn` *id* `| -znname` *name*`]`	Runs verification on all RNs in the specified zone, or in all zones if none is specified.
`| -all-admins [-zn` *id* `| -znname` *name*`]`	Runs verification on all Admins in the specified zone, or in all zones if none is specified.
`[-plan-name` *name*`]`	Runs the named plan that you have saved to execute `plan verify-data` and its available options.
`[-wait]`	Runs a plan synchronously, so that the command line prompt returns after the command completes.
`[-noexecute]`	Lets you create a plan but delay its execution. Conversely, use the `plan execute` command to run the plan.
`[-force]`	Runs the plan as you enter it on the CLI, without validating the flags.
`[-json|-json-v1]`	Displays the plan output as json or json-v1.

Executing verify-data

The `plan verify-data` command is available to verify both primary table and secondary indexes. The command lets you verify either a checksum of data records, or the B-tree of the database.

> **Note:**
>
> Since Oracle NoSQL Database uses Oracle Berkeley DB Java Edition (JE) as its underlying storage engine, verifying data using `plan verify-data` depends on several low-level JE features that are neither described here, nor visible. Throughout this section, terms or concepts related to Oracle Berkeley DB Java Edition (JE) are indicated by the term *Berkeley*, indicating their origination. For more information about Oracle Berkeley DB Java Edition, start here: Oracle Berkeley DB Java Edition.

The `plan verify-data` has two parts for verifications:

- Log record integrity on disk
- B-tree integrity

To verify the integrity of log records on disk, `verify-data` accesses and verifies each record's checksum. Since this procedure includes disk reads, it consumes I/O resources and is

relatively time consuming. To reduce the performance effects of verification, you can configure a longer delay time between reading each batch of log files. While increasing the delay time increases operation time overall, it consumes fewer I/O activities. If that choice is preferable for your requirements, use `-btree-batch-delay` to increase the delay between log file integrity checks during peak I/O operations.

When verifying B-tree integrity, the `plan verify-data` process verifies in-memory integrity. The basic verification checks only if the LSN (*Berkeley*) for each data record in primary tables is valid. You can configure the verification to include data records on disk, as well as secondary index integrity.

If you do not enable data record verification, the secondary index verification checks only the reference from secondary index to primary table, but not from primary table to index. Since basic verification checks only in-memory data structures, it is significantly faster and less resource intensive than verification involving disk reads.

plan wait

```
plan wait -id <id> | -last [-seconds <timeout in seconds>] [-json]
```

Waits indefinitely for the specified plan to complete, unless the optional timeout is specified.

Use the `-seconds` option to specify the time to wait for the plan to complete.

The `-last` option references the most recently created plan.

```
kv-> plan wait -id 26 -json
{
    "operation" : "plan wait",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "planId" : 26,
        "state" : "CANCELED"
    }
}
```

pool

Encapsulates commands that manipulates Storage Node pools, which are used for resource allocations. The subcommands are as follows:

- pool clone
- pool create
- pool join
- pool leave
- pool remove

pool clone

```
pool clone -name <name> -from <source pool name> [-json]
```

Clone an existing Storage Node pool to a new Storage Node pool to be used for resource distribution when creating or modifying a store.

For more information on using a cloned Storage Node pool when contracting a topology, see Contracting a Topology.

```
kv-> pool clone -name mypool from snpool -json{
"operation" : "pool clone",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "poolName" : "mypool"
    }
}
```

pool create

```
pool create -name <name> -json
```

Creates a new Storage Node pool to be used for resource distribution when creating or modifying a store.

For more information on creating a Storage Node pool, see Create a Storage Node Pool.

```
kv-> pool create -name newPool -json{
"operation" : "pool create",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "storeName" : "newPool"
    }
}
```

pool join

```
pool join -name <name> [-sn <snX>]* [-json]
```

Adds Storage Nodes to an existing Storage Node pool.

```
kv-> pool join -name newPool -sn sn1 -json{
"operation" : "pool join",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "storeName" : "newPool"
    }
}
```

pool leave

```
pool leave -name <name> [-sn <snX>]* [-json]
```

Remove Storage Nodes from an existing Storage Node pool.

```
kv-> pool leave -name newPool -sn sn1 -json{
"operation" : "pool leave",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "storeName" : "newPool"
    }
}
```

pool remove

```
pool remove -name <name>
```

Removes a Storage Node pool.

```
kv-> pool remove -name newPool -json{
"operation" : "pool remove",
"returnCode" : 5000,
"description" : "Operation ends successfully",
"returnValue" : {
    "storeName" : "newPool"
    }
}
```

put

Encapsulates commands that put key/value pairs to the store or put rows to a table. The subcommands are as follows:

- put kv
- put table

put kv

```
put kv -key <keyString> -value <valueString> [-file]
    [-hex] [-if-absent] [-if-present]
```

Put the specified key/value pair into the store. The following arguments apply to the put command:

- `-key<keyString>`

  Specifies the name of the key to be put into the store. Key can be composed of both major and minor key paths, or a major key path only. The <keyString> format is: "major-key-path/-/minor-key-path".

  For example, a key containing major and minor key paths:

  ```
  kv-> put kv -key /Smith/Bob/-/email -value
  "{\"id\": 1,\"email\":\"bob.smith@example.com\"}"
  ```

For example, a key containing only a major key path:

```
kv-> put kv -key /Smith/Bob -value"{\"name\":
 \"bob.smith\", \"age\": 20, \"phone\":\"408 555 5555\", \"email\":
 \"bob.smith@example.com\"}"
```

- `-value <valueString>`

  If -file is not specified, the <valueString> is treated as a raw bytes array.

  For example:

  ```
  kv-> put kv -key /Smith/Bob/-/phonenumber -value "408 555 5555"
  ```

  > **Note:**
  >
  > The mapping of the raw arrays to data structures (serialization and deserialization) is left entirely to the application.

- `-file`

  Indicates that the value is obtained from a file. The file to use is identified by the value parameter.

  For example:

  ```
  kv-> put kv -key /Smith/Bob -value ./smith-bob-info.txt
   -file
  ```

- `-hex`

  Indicates that the value is a BinHex encoded byte value with base64 encoding.

- `-if-absent`

  Indicates that a key/value pair is `put` only if no value for the given key is present.

- `-if-present`

  Indicates that a key/value pair is `put` only if a value for the given key is present.

put table

```
kv-> put table -name table_name [if-absent | -if-present ]
    [-json string] [-file file_name] [-exact] [-update]
```

Puts one or more rows into the named table.

- `-name`
  Specifies a table name, which can identify different types of tables:
  - *`table_name`* – The table is a top level table created in the default namespace, `sysdefault`. The default `sysdefault:` namespace prefix is not required.
  - *`table_name.child_name`* – The table is a child table. Always precede a *`child_name`* table with its parent *`table_name`*, followed by a period (.) separator.

  - *namespace_name*:*table_name* – The table was created in the namespace you specify. Always precede *table_name* with its *namespace_name*, followed by a colon (:).
  - *namespace_name*:*table_name*.*child_name* – The table is a child table of a parent table created in a namespace. Specify *child_name* by preceding it with both *namespace_name*: and its parent *table_name*, , followed by a period (.) separator.
- `-if-absent`

  Indicates to put a row only if the row does not exist.
- `-if-present`

  Indicates to put a row only if the row already exists.
- `-json`

  Indicates that the value is a JSON string.
- `-file`

  Use to load a file of JSON strings.
- `-exact`

  Indicates that the input JSON string or file must contain values for all columns in the table and cannot contain extraneous fields.
- `-update`

  Can be used to partially update the existing record.

repair-admin-quorum

```
repair-admin-quorum {-zn <id> | -znname <name> | -admin <id>}...
```

Restores Admin quorum after it is lost by reducing membership of the admin group to the admins in the specified zones, or to the specific admins you can list. Use this command when attempting to recover from a failure that has resulted in losing admin quorum. This command can result in data loss.

After obtaining a working admin by using the `repair-admin-quorum` command, call the `plan failover` command to failover to the zones that remain available after a failure, and to update the topology to match the changes made to the admins.

The arguments specify which admins to use as the new set of primary admins, either by specifying all of the admins in one or more zones, or by identifying specific admins. The specified set of admins must not be empty, must contain only currently available admins, and must include all currently available primary admins. It may also include secondary admins, if desired, to increase the admin replication factor or because no primary admins are available.

> **Note:**
>
> You can repeat this command if a temporary network or component failure results in the initial command invocation to fail.

show

Encapsulates commands that display the state of the store and its components or schemas. The subcommands are as follows:

- show admins
- show datacenters
- show events
- show faults
- show indexes
- show mrtable-agent-statistics
- show parameters
- show perf
- show plans
- show pools
- show snapshots
- show regions
- show tables
- show tls-credentials
- show topology
- show upgrade-order
- show users
- show versions
- show zones

show admins

```
show admins [-json]
```

Displays basic information about Admin services.

```
kv-> show admins -json
{
    "operation" : "show admins",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
    "admins" : [ {
        "id" : "admin1",
        "storageNode" : "sn1",
        "type" : "PRIMARY",
        "connected" : true,
        "adminStatus" : "RUNNING",
        "replicationState" : "MASTER",
        "authoritative" : true
```

```
    }, {
        "id" : "admin2",
        "storageNode" : "sn2",
        "type" : "PRIMARY",
        "connected" : false,
        "adminStatus" : "RUNNING",
        "replicationState" : "REPLICA",
        "authoritative" : true
        } ]
    }
}
```

show datacenters

```
show datacenters
```

Deprecated. See show zones instead.

show events

```
show events [-id <id>] | [-from <date>] [-to <date> ]
    [-type <stat | log | perf>] [-json]
```

Displays event details or list of store events. The status events indicate changes in service status.

Log events are noted if they require attention.

Performance events are not usually critical but may merit investigation. Events marked "SEVERE" should be investigated.

The following date/time formats are accepted. They are interpreted in the local time zone.

- **MM-dd-yy HH:mm:ss:SS**
- **MM-dd-yy HH:mm:ss**
- **MM-dd-yy HH:mm**
- **MM-dd-yy**
- **HH:mm:ss:SS**
- **HH:mm:ss**
- **HH:mm**

For more information on events, see Events.

```
kv-> show events -json
{
    "operation" : "show events",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "events" : [ {
            "event" : "j84a16s3S STAT 2017-09-28 09:48:12.819 UTC sn1 RUNNING
sev1"
```

```
        }, {
            "event" : "j84a17j0S STAT 2017-09-28 09:48:13.788 UTC sn2 RUNNING
sev1"
        }, {
            "event" : "j84a19xoS STAT 2017-09-28 09:48:16.908 UTC sn3 RUNNING
sev1"
        }, {
            "event" : "j84a1cznS STAT 2017-09-28 09:48:20.867 UTC rg1-rn1
RUNNING sev1"
        }, {
            "event" : "j84a1f75S STAT 2017-09-28 09:48:23.729 UTC rg1-rn2
RUNNING sev1"
        }, {
            "event" : "j84a1h7xS STAT 2017-09-28 09:48:26.349 UTC rg1-rn3
RUNNING sev1"
        }, {
            "event" : "j84a3i9rS STAT 2017-09-28 09:50:01.023 UTC rg1-rn3
STOPPED sev1 (reported by sn3)"
        }, {
            "event" : "j84a4oquS STAT 2017-09-28 09:50:56.070 UTC rg1-rn3
RUNNING sev1"
        }, {
            "event" : "j84a5hfeS STAT 2017-09-28 09:51:33.242 UTC rg1-rn3
STOPPED sev1 (reported by sn3)"
        }, {
            "event" : "j84aw53tS STAT 2017-09-28 10:12:16.985 UTC sn3
UNREACHABLE sev2 (reported by
            admin1)"
        }, {
            "event" : "j84b585yL LOG 2017-09-28 10:19:20.854 UTC SEVERE
[admin1] Plan 24 [Remove Admin
            Replica] task [DestroyAdmin admin3] of plan 24 ended in state
ERROR with java.rmi.ConnectException:
            Unable to connect to the storage node agent at host localhost,
port 22000, which may not be running;
            nested exception is: "
        }, {
            "event" : "j84b585zL LOG 2017-09-28 10:19:20.854 UTC SEVERE
[admin1] Plan [null] failed. Attempt 1
            [RUNNING] start=2017-09-28 10:19:20 UTC end=2017-09-28 10:19:20
UTC "
        } ]
    }
}
```

show faults

```
show faults [-last] [-command <command index>] [-json]
```

Displays faulting commands. By default all available faulting commands are displayed. Individual fault details can be displayed using the -last and -command flags.

```
kv-> show faults -json
{
```

```
    "operation" : "show faults",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "faultCommands" : [ {
            "faultCommand" : "503 plan remove-admin -admin 3 -json -wait:
class
            oracle.kv.util.shell.ShellException"
            }, {
            "faultCommand" : "526 topology create -name mytopo -pool snpool -
json -partitions 300 -json: class
            java.lang.NullPointerException"
            } ]
        }
}
```

show indexes

```
show indexes [-table <name>] [-name <name>] [-json]
```

Displays index metadata. By default the indexes metadata of all tables are listed.

If a specific table is named, its indexes metadata are displayed. If a specific index of the table is named, its metadata is displayed. For more information, see plan add-index.

Use `SHOW INDEX` statement to indicate the index type (TEXT, SECONDARY) when you enable text-searching capability to Oracle NoSQL Database, in-concert with the tables interface.

For example:

```
kv-> show index
Indexes on table Joke
JokeIndex (category, txt), type: TEXT
```

For more information, see Integration with Elastic Search for Full Text Search in the *Integrations Guide*.

```
kv-> show indexes -json
{
    "operation" : "show indexes",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "tables" : [ {
        "table" : {
            "tableName" : "t1",
        "indexes" : [ {
        "name" : "idx1",
        "fields" : [ "id1", "id2" ],
        "type" : "SECONDARY",
    "description" : null
    }, {
        "name" : "idx2",
        "fields" : [ "id2" ],
```

```
        "type" : "SECONDARY",
        "description" : null
    } ]
},
    "childTable" : [ {
        "tables" : [ ]
        } ]
    } ]
    }
}
```

show mrtable-agent-statistics

```
show mrtable-agent-statistics [-agent <agentID>][-table <tableName>][-json]
```

Shows the latest statistics as of the last one minute for multi-region table agents. With no arguments, this command shows combined statistics over all regions the MR Table spans.

Input Parameters

Optionally, you can enable the following flags with appropriate parameters with this command:

Table 5-1 Input Parameters

Flag	Parameter	Description
- agent	agentID	Limits the statistics to the agent ID specified. You can find the agent ID from the JSON config file created while configuring your agent. See Configure XRegion Service.
- table	tableName	Limits the statistics to the MR Table specified.
- json	-	Returns the complete statistics in JSON format. Even though the statistics are returned in JSON format by default, specifying this flag adds additional information in the output such as operation, return code, and the return code's description.

Output Statistics

The statistics reported by the `show mrtable-agent-statistics` can be categorized as those used to:

- **Monitor streams from other regions**

Table 5-2 Output Statistics 1

Statistic	Description
completeWriteOps	Number of complete write operations per region.

Table 5-2 (Cont.) Output Statistics 1

Statistic	Description
`lastMessageMs`	Timestamp when the agent sees the last message from a remote region, in milliseconds. If this statistic information is not available, -1 is printed as its output value.
`lastModificationMs`	Timestamp of the last operation performed in each remote region, in milliseconds. If this statistic information is not available, -1 is printed as its output value.
`laggingMs (avg, max, min)`	In a multi-region KVStore, each shard in a region pushes all the operations performed on all its tables to the agent's queue. The agent replicates the contents of its queue, in event order, to all other regions. The lagging statistic represents the time difference between an event being pushed into the queue and replicated to the other regions by the agent. If this value is high, it indicates that the queue is getting backed up. A smaller value indicates that the agent is able to keep up with the number of events coming from remote regions. The lagging statistics are reported as average, minimum, and maximum in milliseconds for each remote region. If this statistic information is not available, -1 is printed as its output value.
`latencyMs (avg, max, min)`	In MR tables, the latency statistic indicates the time taken in milliseconds for each operation to travel from its origin (remote) region to the target (local) region. The latency is computed as T2 - T1, where: – T1 is the timestamp when an operation is performed in the remote region, and – T2 is the timestamp when the agent persisted the replicated operation to the local region. For each remote region, the latency statistics are reported as the average, minimum, and maximum latency for all the operations from that region. If this statistic information is not available, -1 is printed as its output value.

- **Check the persistence of remote data**

Table 5-3 Output Statistics 2

Statistic	Description
`puts`	Number of write operations received.
`dels`	Number of delete operations received.
`streamBytes`	Total bytes replicated from a remote region.

Table 5-3 (Cont.) Output Statistics 2

Statistic	Description
`persistStreamBytes`	Reports the total number of bytes that are successfully committed in the local region. This is different from the total bytes replicated from a remote region because in case of any conflicts with operations from other regions, some of the operations may not persist if they fail the built-in conflict resolution rule.
`winPuts`	Number of write operations performed successfully. More specifically, this statistic excludes the writes that failed to win the conflict resolution rule, in case of a conflict with writes in other regions.
`winDels`	Number of delete operations performed successfully. More specifically, this statistic excludes the deletes that failed to win the conflict resolution rule, in case of a conflict with deletes in other regions.
`incompatibleRows`	Number of operations that did not persist because of incompatible table schemas. This can happen when there is a schema mismatch between the origin region and the region that is trying to replicate the row to its local data store.

- **Monitor the interaction between admin and the agent**

Table 5-4 Output Statistics 3

Statistic	Description
`requests`	All the DDL commands executed by the user on an MR table are converted into requests to the agent by the admin. This statistic reports the number of requests posted by the admin.
`responses`	Number of requests processed and responded by the agent.

- **Monitor multi-region tables**
  When you execute the `show mrtable-agent-statistics` command with the `-table` flag, it returns the table level statistics indicating:

  1. **Persistence of remote data in the local region:** This includes the statistics such as `puts`, `dels`, `winPuts`, `winDels`, `streamBytes`, `persistStreamBytes`, and `incompatibleRows` discussed above.
  2. **Progress of table initialization in each remote region:** This is indicated by the `state` attribute under the `Initialization statistics` in the output. The table below lists the different possible values for state and their meaning.

Table 5-5 Table Initialization States

State	Description
NOT_START	MR table initialization has not started, or there is no need to do initialization. For example, if the agent resumes the stream from an existing checkpoint successfully, there is no need to re-initialize the MR table.
IN_PROGRESS	MR table initialization is ongoing, that is, the MR table initialization has started and the data is being replicated from the remote regions.
COMPLETE	MR table initialization is complete and table transfer is done. The agent is streaming from the remote region.
ERROR	MR table initialization cannot complete because of an irrecoverable error. You can view the error severity in the agent log as WARNING or SEVERE. The agent log can be found in the directory specified in the JSON config file. See Configure XRegion Service.
SHUTDOWN	MR table initialization cannot complete because the service is shut down.

3. **Persistence of the table data per remote region:**

Table 5-6 Output Statistics 4

Statistic	Description
transferStartMs	Timestamp of the initiation of table initialization, in milliseconds. If this statistic information is not available, -1 is printed as its output value.
transferCompleteMs	Timestamp of the completion of table initialization, in milliseconds. If this statistic information is not available, -1 is printed as its output value.
elapsedMs	The time elapsed from the start of the table initialization until its completion. elapsedMs = transferCompleteMs - transferStartMs This statistic is reported in milliseconds. Before the transfer completion, it reports *-1* indicating the unavailability of this statistic.
transferBytes	Number of bytes transferred from the remote (origin or source) region to the local (target) region.
transferRows	Number of rows transferred from the remote region to the local region successfully.
expireRows	Number of rows expired before transferring from the remote region. Because of their TTL value, some rows might expire during the replication. Such rows expire by the time they reach the agent. This statistic counts such expired rows.

Table 5-6 (Cont.) Output Statistics 4

Statistic	Description
persistBytes	Reports the total number of bytes that are successfully committed in the local region. This excludes the rows that are not committed in the local region because they failed the built-in conflict resolution rule. In case of row updates, the entire row is counted for this statistic.
persistRows	Reports the total number of rows that are successfully committed in the local region. Similar to the above statistic, the rows that are not committed in the local region because of the built-in conflict resolution rule are excluded for this count.

Example

Below are a few examples of the statistics returned by the show mrtable-agent-statistics command with different input parameters.

> **Note:**
>
> If any of the statistics information is not available, -1 is reported as the value for that statistic parameter in the output.

```
# MR table agent statistics for a specific agent
kv-> show mrtable-agent-statistics -agent 0 -json
{
  "operation": "show mrtable-agent-statistics",
  "returnCode": 5000,
  "description": "Operation ends successfully",
  "returnValue": {
    "XRegionService-1_0": {
      "timestamp": 1592901180001,
      "statistics": {
        "agentId": "XRegionService-1_0",
        "beginMs": 1592901120001,
        "dels": 1024,
        "endMs": 1592901180001,
        "incompatibleRows": 100,
        "intervalMs": 60000,
        "localRegion": "slc1",
        "persistStreamBytes": 524288,
        "puts": 2048,
        "regionStat": {
          "lnd": {
            "completeWriteOps": 10,
            "laggingMs": {
              "avg": 512,
              "max": 998,
              "min": 31
            },
```

```
            "lastMessageMs": 1591594977587,
            "lastModificationMs": 1591594941686,
            "latencyMs": {
              "avg": 20,
              "max": 40,
              "min": 10
            }
          },
          "dub": {
            "completeWriteOps": 20,
            "laggingMs": {
              "avg": 535,
              "max": 1024,
              "min": 45
            },
            "lastMessageMs": 1591594978254,
            "lastModificationMs": 1591594956786,
            "latencyMs": {
              "avg": 30,
              "max": 45,
              "min": 15
            }
          }
        },
        "requests": 12,
        "responses": 12,
        "streamBytes": 1048576,
        "winDels": 1024,
        "winPuts": 2048
      }
    }
  }
}
```

 # MR table agent statistics for a specific MR table

```
kv-> show mrtable-agent-statistics -table users -json
{
  "operation": "show mrtable-agent-statistics",
  "returnCode": 5000,
  "description": "Operation ends successfully",
  "returnValue": {
    "XRegionService-1_0": {
      "tableID": 12,
      "tableName": "users",
      "timestamp": 1592901300001,
      "statistics": {
        "agentId": "XRegionService-1_0",
        "beginMs": 1592901240001,
        "dels": 1000,
        "endMs": 1592901300001,
        "expiredPuts": 200,
        "incompatibleRows": 100,
        "initialization": {
          "lnd": {
            "elapsedMs": 476,
```

```
            "expireRows": 100,
            "persistBytes": 6492160,
            "persistRows": 6340,
            "state": "COMPLETE",
            "transferBytes": 8115200,
            "transferCompleteMs": 1592822625333,
            "transferRows": 7925,
            "transferStartMs": 1592822624857
          },
          "dub": {
            "transferStartMs": 0,
            "transferCompleteMs": 0,
            "elapsedMs": -1,
            "transferRows": 0,
            "persistRows": 0,
            "expireRows": 0,
            "transferBytes": 0,
            "persistBytes": 0,
            "state": "NOT_START"
          }
        },
        "intervalMs": 60000,
        "localRegion": "fra",
        "persistStreamBytes": 104960000,
        "puts": 100000,
        "streamBytes": 115200000,
        "tableId": 12,
        "tableName": "users",
        "winDels": 745,
        "winPuts": 90000
      }
    }
  }
}
```


show parameters

```
show parameters -policy | -service <name>
```

Displays service parameters and state for the specified service. The service may be a Replication Node, Storage Node, or Admin service, as identified by any valid string, for example rg1-rn1, sn1, admin2, etc. Use the -policy flag to show global policy parameters. Use the -security flag to show global security parameters.

```
show parameters -service sn1
```

When you enable text-searching capability to Oracle NoSQL Database, in-concert with the tables interface, the `show parameter` command also provides information on the Elasticsearch cluster name and transport port as values for the parameters searchClusterMembers and searchClusterName.

For more information, see Integration with Elastic Search for Full Text Search in the *Integrations Guide*.

show perf

```
show perf
```

Displays recent performance information for each Replication Node.

show plans

```
show plans [-last] [-id <id>] [-from <date>] [-to <date>][-num <howMany>]
```

Shows details of the specified plan or list all plans that have been created along with their corresponding plan IDs and status.

- The `-last` option shows details of the most recently created plan.
- The `-id <n>` option details the plan with the given id. If -num <n> is also given, list <n> plans, starting with plan #<id>.
- The `-num <n>` option sets the number of plans to the list. The default is 10.
- The `-from <date>` option lists plans after <date>.
- The `-to <date>` option lists plans before <date>.

Combining `-from` with `-to` describes the range between the two dates. Otherwise `-num` applies.

The following date formats are accepted. They are interpreted in the UTC time zone.

- *yyyy-MM-dd HH:mm:ss.SSS*
- *yyyy-MM-dd HH:mm:ss*
- *yyyy-MM-dd HH:mm*
- *yyyy-MM-dd*
- *MM-dd-yyyy HH:mm:ss.SSS*
- *MM-dd-yyyy HH:mm:ss*
- *MM-dd-yyyy HH:mm*
- *MM-dd-yyyy*
- *HH:mm:ss.SSS*
- *HH:mm:ss*
- *HH:mm*

For more information on plan review, see Reviewing Plans.

show pools

```
show pools
```

Lists the Storage Node pools.

show snapshots

```
show snapshots [-sn <id>]
```

Lists snapshots on the specified Storage Node. If no Storage Node is specified, one is chosen from the store. You can use this command to view the existing snapshots.

show regions

```
show regions
```

Displays the list of all the remote regions included in a Multi-Region Oracle NoSQL Database setup.

```
kv-> execute 'show regions'
regions
  DEN
```

show tables

```
show tables -name table_name
```

Displays the table information. Use `-original` flag to show the original table information if you are building a table for evolution. The flag is ignored for building table for addition. For more information, see plan add-table and plan evolve-table

Use `show table -name table_name` statement to list the full text index. This command provides the table structure including the indexes that have been created for that table. For more information, see Creating FTI in the *Integrations Guide*.

show tls-credentials

The `show-tls-credentials` command shows information about the TLS credentials installed, and the updates waiting to be installed, on all SNAs. The command shows the summary information on the overall status of TLS credentials. Additionally, it also provides information about every SNA in the data store.

In the output of this show command, the two entries `installedCredentialsStatus` and `pendingUpdatesStatus` summarize the information about the TLS credentials installed.

`installedCredentialsStatus`: Verifies if the same pair of keystore and truststore files are installed in all SNs. This entry has one of the following values:

- `consistent`: All SNs report having the same pair of keystore and truststore files installed.
- `mixed`: Some SNAs are found to have different pairs of keystore and truststore files installed.
- `maybe-consistent`: All SNAs that responded reported having the same pair of files installed, but not all SNAs could be contacted.

`pendingUpdatesStatus`: Provides information about any pending updates. This entry has one of the following values:

- `none`: No SNAs have any pending updates, or the update files match currently installed files.
- `consistent`: There are pending updates and all installed files will be consistent once the pending updates are installed.
- `mixed`: There are pending updates and some installed files will be different after the pending updates are installed.
- `maybe-consistent`: There are pending updates, all SNAs that responded will have the same files after the updates are installed, but not all SNAs could be contacted.

In the output of this show command, information about SNAs will be stored in the sns entry. The value of the sns entry maps SN names to information about the associated SNA. Each SNA has two entries: `installedCredentials`, and, `pendingUpdates` each with `keystore` and `truststore` entries for all the files present. For each credential file found, the following entries are present.

- `file`: Provides the name of the file relative to the security subdirectory under the SN's root directory (`$KVROOT`)
- `hash`: Represents the SHA-256 hash of the file contents represented as a 64-character hexadecimal value. You can compute an SHA-256 hash of the credentials files and compare the calculated value with this hash value to ensure that the credential file has not been modified after it was copied to the SNAs.
- `modtime`: Represents the modification time of the file in the UTC timezone Note: If there is a problem contacting an SNA, the value for that SNA will have an exception entry whose value is a string describing the problem.

> **Note:**
>
> If there is a problem contacting an SNA, the value for that SNA will have an `exception` entry whose value is a string describing the problem.

Sample JSON Output:

```
{
    "operation": "show tls-credentials",
    "returnCode": 5000,
    "description": "Operation ends successfully",
    "returnValue": {
      "installedCredentialsStatus": "consistent",
      "pendingUpdatesStatus": "consistent",
      "sns": {
        "sn1": {
          "installedCredentials": {
            "keystore": {
              "file": "store.keys",
              "hash":
"b157d9dc61bed1e2642425adb6791dd13e2c0868160b693963a85596130c388d",
              "modTime": "2024-06-07 09:21:53 UTC"
            },
            "truststore": {
              "file": "store.trust",
              "hash":
"c5a4c67820b64d324a1b8ed493041e04ef91697f64d4a605d750b9dbe8708f6b",
```

```
              "modTime": "2024-06-07 09:21:57 UTC"
            }
          },
          "pendingUpdates": {
            "truststore": {
              "file": "updates/store.trust",
              "hash":
"4fe0231942e732d829cf97cb3742c9aa86ae06db8a80939b6fb834382c205a53",
              "modTime": "2024-06-08 10:00:00 UTC"
            }
          }
        },
        "sn2": { ... },
        "sn3": { ... }
      }
    }
}
```

show topology

```
show topology [-zn] [-rn] [-an] [-sn] [-store] [-status] [-json] [-verbose]
```

Displays the current, deployed topology. By default it shows the entire topology, including the number of shards. The first set of optional flags restrict the display to one or more zones, Replication Nodes, Storage Nodes, Arbiter Nodes, store name, or to specify service status. Use –json to display the results in JSON format. If you specify -verbose, then additional information will be displayed, including Replication Node storage directories, storage directory sizes, log directories, and JE HA ports.

You can also obtain the zone ID to which you can deploy Storage Nodes.

```
kv-> show topology
store=mystore  numPartitions=1000 sequence=2376
  zn: id=zn1 name=myzone repFactor=3 type=PRIMARY allowArbiters=false
masterAffinity=false
  sn=[sn1] zn:[id=zn1 name=myzone] nodeA:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
              single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
  sn=[sn2] zn:[id=zn1 name=myzone] nodeB:5000 capacity=1 RUNNING
    [rg1-rn2] RUNNING
              single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
  sn=[sn3] zn:[id=zn1 name=myzone] nodeC:5000 capacity=1 RUNNING
    [rg1-rn3] RUNNING
              single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
  sn=[sn4] zn:[id=zn1 name=myzone] nodeD:5000 capacity=1 RUNNING
    [rg2-rn1] RUNNING
          No performance info available
  sn=[sn5] zn:[id=zn1 name=myzone] nodeE:5000 capacity=1 RUNNING
    [rg2-rn2] RUNNING
              single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
  sn=[sn6] zn:[id=zn1 name=myzone] nodeF:5000 capacity=1 RUNNING
    [rg2-rn3] RUNNING
              single-op avg latency=0.0 ms   multi-op avg latency=0.0 ms
```

```
  numShards=2
  shard=[rg1] num partitions=500
    [rg1-rn1] sn=sn1
    [rg1-rn2] sn=sn2
    [rg1-rn3] sn=sn3
  shard=[rg2] num partitions=500
    [rg2-rn1] sn=sn4
    [rg2-rn2] sn=sn5
    [rg2-rn3] sn=sn6
```

show upgrade-order

```
show upgrade-order [-json]
```

Lists the Storage Nodes which need to be upgraded in an order that prevents disruption to the store's operation.

This command displays one or more Storage Nodes on a line. Multiple Storage Nodes on a line are separated by a space. If multiple Storage Nodes appear on a single line, then those nodes can be safely upgraded at the same time. When multiple nodes are upgraded at the same time, the upgrade must be completed on all nodes before the nodes next on the list can be upgraded.

If at some point you lose track of which group of nodes should be upgraded next, you can always run the show upgrade-order command again.

```
kv-> show upgrade-order -json
{
    "operation" : "show upgrade-order",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
    "singleTextResult" : "Calculating upgrade order, target version:
12.2.4.6.0, prerequisite:
    12.1.3.0.5\nUnable to contact sn3 Unable to connect to the storage node
agent at host localhost, port
    22000, which may not be running; nested exception is:
\n\tjava.rmi.ConnectException: Connection refused
    to host: localhost; nested exception is: \n\tjava.net.ConnectException:
Connection refused (Connection
    refused)\nThere are no nodes that need to be upgraded"
    }
}
```

show users

```
show users -name <name>
```

Lists the names of all users, or displays information about a specific user. If no user is specified, lists the names of all users. If a user is specified using the `-name` option, then lists detailed information about the user.

show versions

```
show versions [-json]
```

Lists the client and server version information.

For example

```
kv-> show versions
Client version: 12.1.3.4.0
Server version: 12.1.3.4.0

kv-> show versions -json
{
    "operation" : "show version",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "clientVersion" : "12.2.4.6.0",
        "serverVersion" : "12.2.4.6.0"
    }
}
```

show zones

```
show zones [-zn <id>] | -znname <name>] [-json]
```

Lists the names of all zones, or display information about a specific zone.

Use the `-zn` or the `-znname` flag to specify the zone that you want to show additional information; including the names of all of the Storage Nodes in the specified zone, and whether that zone is a primary of secondary zone.

```
kv-> show zones -json
{
    "operation" : "show zone",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "zones" : [ {
            "zone" : {
            "id" : "zn1",
            "name" : "1",
            "repfactor" : 1,
            "type" : "PRIMARY",
            "allowArbiters" : false
        }
        }, {
            "zone" : {
            "id" : "zn2",
            "name" : "2",
            "repfactor" : 1,
```

```
                "type" : "PRIMARY",
                "allowArbiters" : false
            }
            }, {
                "zone" : {
                "id" : "zn3",
                "name" : "3",
                "repfactor" : 1,
                "type" : "PRIMARY",
                "allowArbiters" : false
            }
            } ]
        }
}
```

snapshot

Encapsulates commands that create and delete snapshots, which are used for backup and restore. The subcommands are as follows:

- snapshot create
- snapshot remove

snapshot create

```
snapshot create -name <name>
```

Creates a new snapshot using the specified name as the prefix.

Use the `-name` option to specify the name of the snapshot that you want to create.

Snapshots should not be taken while any configuration (topological) changes are being made, because the snapshot might be inconsistent and not usable.

snapshot remove

```
snapshot remove -name <name> | -all
```

Removes the named snapshot. If -all is specified, remove all snapshots.

Use the `-name` option to specify the name of the snapshot that you want to remove.

If the `-all` option is specified, remove all snapshots.

To create a backup of your store using a snapshot see Taking a Snapshot.

To recover your store from a previously created snapshot you can use the load utility or restore directly from a snapshot. For more information, see Using the Load Program or Restoring Directly from a Snapshot.

table

Deprecated with exception of table-size. See execute instead.

table-size

```
table-size -name <name> -json <string>
    [-rows <num> [[-primarykey | -index <name>] -keyprefix <size>]]
```

Calculates key and data sizes for the specified table using the row input, optionally estimating the NoSQL DB cache size required for a specified number of rows of the same format. Running this command on multiple sample rows can help determine the necessary cache size for desired store performance.

- -json specifies a sample row used for the calculation.
- -rows specifies the number of rows to use for the cache size calculation
- Use the -index or -primarykey and -keyprefix to specify the expected commonality of index keys in terms of number of bytes.

This command mainly does the following:

1. Calculates the key and data size based on the input row in JSON format.
2. Estimates the DB Cache size required for a specified number of rows in the same JSON format.

   The output contains both detailed size info for primary key/index and the total size; internally it calls JE's DbCacheSize utility to calculate the cache size required for primary key and indexes with the input parameters:

   ```
   java -jar $KVHOME/dist/lib/je.jar DbCacheSize
   -records <num> -key <size> -data <size> -keyprefix
   <size> -outputproperties -replicated <JE properties...>
   -duplicates]
   ```

   where:

   - -records <num>: The number of rows specified by -row <num>.
   - -key <size>: The size of key get from step 1.
   - -data <size>: The size of data get from step1.
   - -keyprefix <size>: The expected commonality of keys, specified using -primarykey | -index <name> -keyprefix <size>
   - -duplicates: Used only for table index.
   - -<JE properties...>: The JE configuration parameters used in kvstore.

For example:

```
kv-> execute "create table user (id integer, address string,
zip_code string, primary key(id))"
kv-> execute "create index idx1 on user (zip_code)"
```

See the following cases:

1. Calculates the key size and data size based on the input row in JSON.

```
kv-> table-size -name user -json '{"id":1,
"address": "Oracle Building ZPark BeiJing China",
"zip_code":"100000"}'

=== Key and Data Size ===

      Name         Number of Bytes
-----------------  ---------------
Primary Key               8
Data                     47
Index Key of idx1         7
```

2. Calculates the key/data size and the cache size of the table with 10000 rows.

```
kv-> table-size -name user -json '{"id":1,
"address": "Oracle Building ZPark BeiJing China",
"zip_code":"100000"}'
-rows 10000
=== Key and Data Size ===

      Name         Number of Bytes
-----------------  ---------------
Primary Key               8
Data                     47
Index Key of idx1         7

=== Environment Cache Overhead ===

16,798,797 minimum bytes

=== Database Cache Sizes ===

Name   Number of Bytes             Description
-----  ---------------  ---------------------------------
             1,024,690  Internal nodes only
Table        1,024,690  Internal nodes and record versions
             1,024,690  Internal nodes and leaf nodes
-----  ---------------  ---------------------------------
               413,728  Internal nodes only
idx1           413,728  Internal nodes and record versions
               413,728  Internal nodes and leaf nodes
-----  ---------------  ---------------------------------
             1,438,418  Internal nodes only
Total        1,438,418  Internal nodes and record versions
             1,438,418  Internal nodes and leaf nodes
```

For more information, see the DbCacheSize javadoc.

Note:

The cache size is calculated in the following way:

- Cache size of table

```
java -jar KVHOME/lib/je.jar DbCacheSize -records
   10000 key 8 -data 47 -outputproperties -replicated
   <JE properties...>
```

  The parameters are:

  - Record number: 10000
  - Primary key size: 8
  - Data size: 47

- Cache size of table

```
 java -jar KVHOME/lib/je.jar DbCacheSize -records
   10000 -key 7 -data 8 -outputproperties -replicated
   <JE properties...> -duplicates
```

  The parameters are:

  - Record number: 10000
  - Index key size: 7
  - Data size: 8. The primary key size is used here, since the data of secondary index is the primary key.
  - Use -duplicates for index.

- Total size = cache size of table + cache size of idx1.

3. Calculates the cache size with a key prefix size for idx1

```
kv-> table-size -name user -json
'{"id":1, "address":"Oracle Building ZPark BeiJing China",
"zip_code":"100000"}' -rows 10000 -index idx1 -keyprefix 3

=== Key and Data Size ===


      Name         Number of Bytes
-----------------  ---------------
Primary Key                  8
Data                        47
Index Key of idx1            7

=== Environment Cache Overhead ===

16,798,797 minimum bytes

=== Database Cache Sizes ===
```

```
Name   Number of Bytes             Description
-----  ---------------  ---------------------------------
             1,024,690  Internal nodes only
Table        1,024,690  Internal nodes and record versions
             1,024,690  Internal nodes and leaf nodes
-----  ---------------  ---------------------------------
               413,691  Internal nodes only
idx1           413,691  Internal nodes and record versions
               413,691  Internal nodes and leaf nodes
-----  ---------------  ---------------------------------
             1,438,381  Internal nodes only
Total        1,438,381  Internal nodes and record versions
             1,438,381  Internal nodes and leaf nodes
```

For more information, see the DbCacheSize javadoc.

> **Note:**
>
> A key prefix size is provided for idx1, the idx1's cache size is calculated like this:
>
> ```
> java -jar KVHOME/lib/je.jar DbCacheSize -records
> 10000 -key 7 -data 8 -keyprefix 3 -outputproperties
> -replicated <JE properties...> -duplicates
> ```
>
> The above examples show that the cache size of idx1 is 413,691 and is smaller than 413,728 of case 2. For more information about the usage of keyprefix, see JE DbCacheSize document.

timer

```
timer [on|off]
```

Turns the measurement and display of execution time for commands on or off.

topology

Encapsulates commands that manipulate store topologies. Examples are redistribution/ rebalancing of nodes or changing replication factor. Topologies are created and modified using this command. They are then deployed by using the `plan deploy-topology` command. For more information, see plan deploy-topology. The subcommands are as follows:

- topology change-repfactor
- topology change-zone-arbiters
- topology change-zone-type
- topology clone
- topology contract
- topology create
- topology delete

- topology list
- topology preview
- topology rebalance
- topology redistribute
- topology validate
- topology view

topology change-repfactor

```
topology change-repfactor -name <name>  -pool <pool name>
    -zn <id> | -znname <name> -rf <replication factor>
```

Modifies the topology to change the replication factor of the specified zone to a new value. The replication factor may be decreased for secondary zones, but decreasing it for primary zones is not currently supported.

When increasing the replication factor, the command may create Replication Nodes or Arbiter Nodes and may remove Arbiter Nodes only in the zone specified in the command. If the change in replication factor increases the total primary replication factor equal to two and the zone is configured to allow Arbiters, then Arbiters are created in that zone. If the change in replication factor increases the total primary replication factor from two to a number greater than two and if the zone contained Arbiters, then the Arbiters are removed from the zone. If some other zone contained Arbiters, a topology rebalance must be performed to remove the Arbiters from the topology.

For more information on increasing the replication factor, see Increase Replication Factor.

When decreasing the replication factor for a secondary zone, the command will remove the replication nodes from the zone.

If you want to remove a secondary zone, then the replication factor for that secondary zone should be reduced to zero.

After reducing the replication factor to zero, do the following steps to remove the secondary zone:

1. Remove any admins in the zone using `plan remove-admin` command
2. Remove the Storage Nodes in the zone using `plan remove-sn` command
3. Remove the zone using `plan remove-zone` command

topology change-zone-arbiters

```
topology change-zone-arbiters -name <name>
    {-zn <id> | -znname <name>}  {-arbiter | -no-arbiter}
```

Modifies the topology to change the Arbiter Node attribute of the specified zone.

topology change-zone-master-affinity

```
topology change-zone-master-affinity -name <name>
    -zn <{-no-master-affinity | -master-affinity}
```

Modifies the topology of the existing specified zone to `-no-master-affinity`, or to `-master-affinity`. For example:

```
topology change-zone-master-affinity -name new-topo -zn zn1 -no-master-
affinity
```

Use this command after initially deploying a topology (`plan deploy-zone`).

topology change-zone-type

```
topology change-zone-type -name <name>
    {-zn <id> | -znname <name>} -type {primary | secondary}
```

Modifies the topology to change the type of the specified zone to a new type.

If one or more zones have their type changed and the resulting topology is deployed using the plan deploy-topology command, the following rules apply:

- The plan waits for up to five minutes for secondary nodes that are being converted to primary nodes to catch up with their masters.
- The plan will fail, and print details about lagging zones and nodes, if a quorum of secondary nodes in each shard fails to catch up within the required amount of time. This behavior helps to reduce the time that a newly added primary node cannot become a master, and so is not able to contribute to availability.
- Because this command can only be performed successfully if quorum can be maintained, it does not result in data loss.

topology clone

```
topology clone -from <from topology> -name <to topology>
```

or

```
topology clone -current -name <to topology>
```

Clones an existing topology so as to create a new candidate topology to be used for topology change operations.

topology contract

```
topology contract -name <name> -pool <pool name>
```

Modifies the named topology to contract storage nodes. For more information, see Contracting a Topology.

topology create

```
topology create -name <candidate name> -pool <pool name> [-json]
    -partitions <num>
```

Creates a new topology with the specified number of partitions using the specified storage pool.

You should avoid using the dollar sign ('$') character in topology candidate names. The CLI displays a warning when trying to create or clone topologies whose names contain the reserved character.

If the primary replication factor is equal to two, the `topology create` command will allocate Arbiter Nodes on the Storage Nodes in a zone that supports hosting Arbiter Nodes. During topology deployment, an error is issued if there are not enough Storage Nodes for Arbiter Node distribution. A valid Arbiter Node distribution is one in which the Arbiter Node is hosted on a Storage Node that does not contain other members of its Replication Group.

For more information on creating the first topology candidate, see Make the Topology Candidate.

```
kv-> topology create -name mytopo -pool snpool -json -partitions 20
{
    "operation" : "topology create",
    "returnCode" : 5000,
    "description" : "Operation ends successfully",
    "returnValue" : {
        "store" : "mystore",
        "numPartitions" : 20,
        "sequence" : 32,
        "zone" : [ {
        "id" : "zn1",
        "name" : "1",
        "repfactor" : 1,
        "type" : "PRIMARY"
        }, {
    "id" : "zn2",
    "name" : "2",
    "repfactor" : 1,
    "type" : "PRIMARY"
    }, {
    "id" : "zn3",
    "name" : "3",
    "repfactor" : 1,
    "type" : "PRIMARY"
    } ],
    "sns" : [ {
    "id" : "sn1",
    "zone_id" : "zn1",
    "host" : "localhost",
    "port" : 20000,
    "capacity" : 1,
    "rns" : [ "rg1-rn1" ],
    "ans" : [ ]
    }, {
    "id" : "sn2",
    "zone_id" : "zn2",
    "host" : "localhost",
    "port" : 21000,
    "capacity" : 1,
    "rns" : [ "rg1-rn2" ],
    "ans" : [ ]
```

```
    }, {
    "id" : "sn3",
    "zone_id" : "zn3",
    "host" : "localhost",
    "port" : 22000,
    "capacity" : 1,
    "rns" : [ "rg1-rn3" ],
    "ans" : [ ]
    } ],
    "shards" : [ {
        "id" : "rg1",
        "numPartitions" : 20,
        "rns" : [ "rg1-rn1", "rg1-rn2", "rg1-rn3" ],
        "ans" : [ ]
        } ],
    "name" : "mytopo"
    }
}
```

topology delete

```
topology delete -name <name>
```

Deletes a topology.

topology list

```
topology list
```

Lists existing topologies.

topology preview

```
topology preview -name <name> [-start <from topology>]
```

Describes the actions that would be taken to transition from the starting topology to the named, target topology. If -start is not specified, the current topology is used. This command should be used before deploying a new topology.

topology rebalance

```
topology rebalance -name <name> -pool <pool name>
    [-zn <id> | -znname <name>]
```

Purpose of `topoplogy rebalance` command

In a data store, the `topology rebalance` command modifies a non-compliant topology to a compliant topology state, making it a balanced topology. For a compliant topology, you must ensure that topologies follow a specific set of rules. For more information, see Determining Your Store's Configuration.

The arguments for the `topology rebalance` command include:

- `-name`: The name of the topology you want to balance.
- `-pool`: The name of the Storage Node pool.
- `-zn <id> or -zname <name>`: These arguments are optional, and you can either provide the `id` or `name` of the zone.

> **Note:**
>
> If the optional `-zn` or `-zname` flag is used, only Storage Nodes from the specified zone are used for the operation.

If there are any changes made to the physical characteristics of the data store, it could become unbalanced, for example:

- The capacity of the Storage Node is increased or decreased resulting in overcapacity or undercapacity respectively.
- There are changes in the disk sizes of Storage Nodes, Replication Nodes.
- The Arbiter Nodes are present when the topology does not support them.
- The Arbiter Nodes are not present when the topology supports them.
- The Arbiter Nodes are present in a zone with Replication Factor greater than zero when a zone with zero Replication Factor exists.
- There are changes in the shard configurations, such as changes in the number of partitions in a shard due to an elasticity change, and so on.

The command ensures compliance by resolving violations and notes arising from the physical changes. It adds, moves, or removes Arbiter Nodes and Replication Nodes. For example, the `topology rebalance` command:

- Moves the Replication Nodes to match the new capacity of the Storage Nodes.
- Adds Arbiter Nodes if the new topology supports them and the old topology does not.
- Removes Arbiter Nodes if the old topology supported them and the new one does not.
- Moves Arbiter Nodes to a zero Replication Factor zone if they are hosted in a zone with Replication Factor greater than zero.
- Initiates partition migration when there is a change in the shard configurations.

To understand more about the scenarios in which you can use the command, see Usage of `topology rebalance` command.

You can verify the topology of the data store by using the verify command. For more information, see Verifying the Store.

```
verify configuration
```

> **Note:**
>
> The topology rebalance command will not be needed when the topology is fully compliant with no changes in the physical characteristics.

Usage of `topology rebalance` command

When you initially deploy the data store, you establish its configuration, or topology, during setup, ensuring it meets all requirements without needing rebalancing. If there is any change in the physical characteristics of the topology, such as the store's configuration, capacity, and so on, then:

1. Run the `verify configuration` command to view any violations or verification notes.
2. Run the `topology rebalance` command to fix the non-compliant topology.

Following are some of the scenarios where the use of the `topology rebalance` command can fix violations/verification notes and help in the transition of a non-complaint topology to a compliant topology:

- **Replication Node allocation violations:**
  - When you modify a storage node's defined capacity using the command plan change-parameters to a value insufficient to host all its Replication Nodes, it causes an imbalance, resulting in overcapacity.

    ```
    Verification violation: [sn1]  sn1 has 2 repNodes and is over its
    capacity limit of 1
    ```

  - If Storage Nodes have more capacity than the Replication Nodes they host, the data store underutilizes its resources, resulting in undercapacity.

    ```
    Verification note: [sn1]  sn1 has 0 RepNodes and is under its capacity
    limit of 1
    ```

    > **Note:**
    >
    > * The `topology rebalance` command resolves overcapacity and undercapacity by moving the Replication Nodes from overutilized Storage Nodes to underutilized ones. This optimizes the use of all available resources.
    > * If there is no overcapacity issue, the `topology rebalance` command can't actively resolve undercapacity, conversely, if there is no undercapacity issue, it can't resolve overcapacity.

- **Arbiter Node allocation violations:**
  - Considering there is a primary zone with a Replication Factor of two, deploy a new zone with zero Storage Node capacity and Replication Factor. This new zone can only host Arbiter Nodes. In this case, you see a violation due to insufficient Arbiter Nodes. For more information, see Arbiter Nodes in *Concepts Guide*.

    ```
    Verification violation: [rg1]  The shard rg1 does not have an Arbiter.
    ```

    The `topology rebalance` command will resolve the issue by moving the Arbiter Node in the same shard along with the other Replication Nodes.
  - Consider a topology already hosting Arbiter Nodes, and you deploy a new zone containing a Storage node with capacity one and arbiters set to false. Since the

updated topology can't host Arbiter Nodes, you will receive a verification note for excess Arbiter Nodes.

```
Verification note: [rg1]  Shard rg1 has an Arbiter when it should not.
```

The `topology rebalance` command resolves the issue by replacing the Arbiter Node with the added Storage Node in the same shard.

- In the topology, an Arbiter Node is in a zone with other Replication Nodes, while another zone exists with a Replication Factor of zero, known as an Arbiter-only zone. The violation indicates that you must move the Arbiter Node to a zone with a zero Replication Factor.

  ```
  Verification violation: [rg1-an1]  rg1-an1 is hosted in zone zn1; a
  better zone host is zn2
  ```

  For high availability and better performance, the `topology rebalance` command will move the Arbiter Node to a zone with a zero Replication Factor.

- When you allow arbiters in a zone initially and then change the configuration to disallow arbiters, the `verify configuration` command still shows the initial setting with arbiters set to true.

  ```
  Verification violation: [rg1-an1]  sn3 doesn't allow arbiters but hosts
  rg1-an1
  ```

  The `topology rebalance` command will fix the violation by removing the Arbiter Node and updating the topology's configuration.

- When there is a change in the disk configuration of Storage Nodes in a shard, the `topology rebalance` command will initiate a partition migration. This forms a balance in the partitions across the shards proportional to the disk sizes.

  ```
  Verification note: [rg1]  rg1 should have 80 partitions if balanced, but
  has 50
  Verification note: [rg2]  rg2 should have 20 partitions if balanced, but
  has 50
  ```

> **Note:**
>
> The `topology rebalance` command does not migrate partitions if there are changes in the minimum number of Storage Nodes in a shard that is less that is less than half of total. It only performs partition distribution to resolve violations or verification notes without introducing new ones and moves partitions only if the disk space on all Replication Nodes in the shard supports the change.

> **Note:**
>
> During the `topology rebalance`, the data store's performance is lower and the chance of a loss of write quorum higher, see Quorum. Therefore, you should perform the `topology rebalance` operation during the quiet time for the data store.

The `topology rebalance` command resolves most of the violations to transition the non-compliant topology to a compliant topology. However, you must manually resolve some violations or verification notes for optimal results. See, Violations and Solutions Table (Table 4-1).

For more information on balancing a non-compliant topology, see Balance a Non-Compliant Topology.

topology redistribute

```
topology redistribute -name <name> -pool <pool name>
```

Modifies the named topology to redistribute resources to more efficiently use those available.

For more information on redistributing resources to enhance write throughput, see Increase Data Distribution.

topology validate

```
topology validate [-name <name>]
```

Validates the specified topology. If no topology is specified, the current topology is validated. Validation generates violations and notes.

Violations are issues that can cause problems and should be investigated.

Notes are informational and highlight configuration oddities that can be potential issues or may be expected.

For more information, see Validate the Topology Candidate.

topology view

```
topology view -name <name>
```

Displays details of the specified topology. Also displays any available Arbiter Node information.

verbose

```
verbose [on|off]
```

Toggles or sets the global verbosity setting. This property can also be set on a per-command basis using the `-verbose` flag.

verify

Encapsulates commands to check various store parameters. Specify one of the subcommands, optionally with `-silent` or `-json`:

```
verify {configuration | prerequisite | upgrade} [-silent] [-json]
```

- verify configuration
- verify prerequisite
- verify upgrade

Invoking `verify` without a subcommand or flag, the returns a deprecated message:

```
kv-> verify
The command:

        verify [-silent]

is deprecated and has been replaced by:

        verify configuration [-silent]
```

verify configuration

```
verify configuration [-silent] [-json]
```

Verifies the store configuration by iterating over components and checking their state against what the Admin database contains. On a large store, this command can be time consuming.

The `-json` option specifies that the command display all output in JSON format.

The `-silent` option suppresses verbose verification messages as verification proceeds. Using the `-silent` option displays only the initial startup messages and the final verification message. This option has no effect when the `-json` option is specified.

In some situations, the `verify configuration` command can generate *violations* and *notes*. For example, if:

- The disk reaches a limit exception.
- The available storage size is less than 5 GB.
- The shard has no partitions.
- A replication node or a storage node is not running.

verify prerequisite

```
verify prerequisite [-silent] [-sn snX]*
```

Verifies that the storage nodes are at or above the prerequisite software version needed to upgrade to the current version. This call may take a while on a large store.

As part of the verification process, this command displays the components which do not meet the prerequisites or cannot be contacted. It also checks for illegal downgrade situations where the installed software is of a newer minor release than the current version.

When using this command, the current version is the version of the software running the command line interface.

Use the `-sn` option to specify those storage nodes that you want to verify. If no storage nodes are specified, all the nodes in the store are checked.

The `-silent` option suppresses verbose verification messages that are displayed as the verification is proceeding. Instead, only the initial startup messages and the final verification message is displayed.

verify upgrade

```
verify upgrade [-silent] [-sn snX]*
```

Verifies the storage nodes (and their managed components) are at or above the current version. This call may take a while on a large store.

As part of the verification process, this command displays the components which have not yet been upgraded or cannot be contacted.

When using this command, the current version is the version of the software running the command line interface.

Use the `-sn` option to specify those storage nodes that you want to verify. If no storage nodes are specified, all the nodes in the store are checked.

The `-silent` option suppresses verbose verification messages that are displayed as the verification is proceeding. Instead, only the initial startup messages and the final verification message is displayed.

Admin Utility Command Reference

This appendix describes the following Admin utility commands:

- diagnostics
- generateconfig
- help
- kvlite
- load admin metadata
- makebootconfig
- ping
- restart
- runadmin
- securityconfig
- start
- status
- stop
- version
- xrstart
- xrstatus
- xrstop

Oracle NoSQL Database utility commands are stand-alone utilities that do not require the use of the Oracle NoSQL Database Command Line Interface. They are available using one of two

jar files. In some cases, kvstore.jar is used. In others, kvtool.jar is required. Both are packaged with the server libraries.

diagnostics

You can troubleshoot your KVStore using the diagnostics tool. You should first run the `diagnostics setup` command in order to set up the tool. You can then use the `diagnostics collect` command to package important information and files to be able to send them to Oracle Support. You can use the `diagnostics verify` command to verify the configuration of the specified Storage Nodes.

For all details on using the diagnostics tool to troubleshoot your KVStore, see Diagnostics Utility.

generateconfig

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar generateconfig [-verbose]
-host <hostname> -port <port>
 -sn <StorageNodeId> -target <zipfile>
[-username <user >]
[-security <security-file-path>]
 [-secdir <overriden security directory>]
```

This command generates configuration files for any Storage Node identifier (value of "sn" parameter) specified in the command.

Parameter	Required	Default value	Description
`host`	Yes		The host name of the Storage Node for which the config file is generated.

Parameter	Required	Default value	Description
`port`	Yes		The registry port of the Storage Node for which the config file is generated.

Note: The user can use the Admin CLI ping command, to get t

Parameter	Required	Default value	Description
			hereregistryportofanyStorageNode.

Parameter	Required	Default value	Description
sn	Yes		Identifier of the Storage Node.

Note: The user can use the Admin CLI ping command, to get the

Parameter	Required	Default value	Description
			StorageNodeIdentifierofanyStorageNode.
`target`	Yes		Full path of the zip file to be created.
`username`	No		The name of the user to log in to the secured store. This parameter is only required if your store is configured to require authentication.
`security`	No		The client security configuration file. This parameter is only required if your store is secure. A fully qualified path to a file containing security information can be specified.

Parameter	Required	Default value	Description
secdir	No	security	The name of the directory within the KVROOT that will hold the security configuration.

help

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar help <commandName>
```

Prints usage info. With no arguments the top-level shell commands are listed. With a command name, additional detail is provided.

kvlite

KVLite is a simplified version of the Oracle NoSQL Database. It provides a single storage node, single shard store, that is not replicated. It runs in a single process without requiring any administrative interface.

Below are the command line options that you can use with the kvlite utility.

```
java -jar KVHOME/lib/kvstore.jar kvlite
[-root <rootDirectory>]
[-store <storeName>]
[-host <hostname>]
[-port <port>]
[-storagedirsizegb <GB>]
[-noadmin]
[-secure-config <enable|disable>]
[-restore-from-snapshot <name of snapshot>]
[-admin-web-port <admin web service port>]
[-version]
[-harange <startPort, endPort>]
[-servicerange <startPort, endPort>]
[-storagedir <path>]
[-memorymb]
[-printstartupok]
```

You can specify the parameter values the first time you start KVLite. The parameter values are recorded in a config file and are used when KVLite is restarted. The config file is stored in the root directory. To change your initial values, either delete your root directory before starting KVLite, or specify the -root option with a different root directory than you initially used. Following are the descriptions of the command line options:

- `root <path>`: Identifies the path to the root directory. The path provided can be an absolute path or a relative path. The database files are stored in this location. If the database files do not exist, they are created for you.

  Default value: ./kvroot

- `-store <storename>`: Identifies the name of the data store. You should avoid using characters in the data store name that might interfere with its use within a file path. Valid characters are all alphanumeric characters, hyphen, underscore, and period.

  Default value: kvstore

- `-host <hostname>`: Identifies the name of the host on which KVLite is running. If you want to access this instance of KVLite from remote machines, you must supply the actual hostname of the machine. Otherwise, specify `localhost` for this option.

  Default value: <host name of the machine>

- `-port <port>`: Identifies the port on which KVLite is listening for client connections. This is sometimes referred to as the registry port.

  Default value: 5000

- `-storagedirsizegb <GB>`: Identifies the maximum amount of available disk space in GB for a KVLite database. If the disk usage of the KVLite database exceeds a value that is 5GB less than the specified value, KVLite suspends all write operations until you delete some existing records and free up enough space to satisfy the disk storage requirement. If you set `storagedirsizegb` to 0, KVLite opportunistically uses all the available space.

  The system maintains 5 GB of free space for recovery purposes in case the configured disk limit is exceeded.

  Default value: 10

- `-noadmin`: Identifies if the administration service needs to be started. If -noadmin parameter is provided, then the administration service is not started.

  The administration service is the service that runs the various admin CLI commands.

- `secure-config`: Identifies if security needs to be enabled for the store. If enabled, all clients connecting to the store must present security credentials.

  Default value: enable

- `-restore-from-snapshot`:
  Identifies the snapshot from which the store can be restored. You can restore a store directly from a snapshot. For example:

  ```
  -restore-from-snapshot 240424-104506-mySnapshot
  ```

  where, `240424-104506-mySnapshot` represents the directory name of the snapshot to restore. This procedure restores the store to the time you created the snapshot. If your store was active after snapshot creation, all modifications made since the snapshot are lost.

  Default value: null

- `-admin-web-port <admin web service port>`: Identifies the TCP/IP port on which the admin web service is started. If a positive integer value is not specified, then the admin web service does not start up. For more details on the admin web service, see REST API for Administering Oracle NoSQL Database.

  Default value: -1

- `-version`: Provides the KVLite Database version, the date and time it was installed, the build ID and the edition (community/enterprise).

- `-harange <startPort, endPort>`: Identifies the harange (high availability range) ports used by the replication node for communication purposes.

- `-servicerange <startPort,endPort>`: Identifies a range of ports that a storage node uses to communicate with its administrative services and managed services. This parameter is useful when services on a storage node must use specific ports for firewall or other security reasons. By default, the services use anonymous ports.
- `-storagedir <path>`: Identifies the path to the directory that the replication node can use for storage. This should be an absolute path. If you do not specify a storage directory explicitly, the replication node uses a directory under the root directory.
- `-memorymb`: Specifies the total number of megabytes of memory that is available in the machine for the replication node. The system uses the `memory_mb` value to guide the specification of the replication node's heap and cache sizes. If the value is zero, the store attempts to determine the amount of memory on the machine. For best results, do not specify this parameter. KVLite will determine the proper value by default. This parameter should be used sparingly, and only for exceptional situations.

  Default value: 0
- `-printstartupok`: Displays messages that show if the starting up of KVLite processes is successful.

load admin metadata

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar load -store <storeName>
-host <hostname> -port <port> -load-admin
-source <admin-backup-dir> [-force]
[-username <user>] [-security <security-file-path>]
```

Loads the admin metadata from the snapshot to the new store. In this case the `-source` directory must point to the environment directory of the admin node from the snapshot. The store must not be available for use by users at the time of this operation.

where:

- `-load-admin` Specifies that only admin metadata will be loaded into the store.

  > **Note:**
  >
  > This option should not be used on a store unless that store is being restored from scratch. If `-force` is specified in conjunction with `-load-admin`, any existing metadata in the store, including tables and security metadata, will be overwritten. See Using the Load Program for more information.

- `-host <hostname>` Identifies the host name of a node in your store.
- `-port <port>` Identifies the registry port in use by the store's Node.
- `-security <security-file-path>` Identifies the security file used to specify properties for login.
- `-source <admin-backup-dir>` The admin snapshot directory containing the contents of the admin metadata that is to be loaded into the store.
- `-store <storeName>` Identifies the new store which is the target of the load.
- `-username <user>` Identifies the name of the user to login to the secured store.

load store data

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar load [-verbose]
-store <storeName> -host <hostname> -port <port>
-source <shard-backup-dir>[, <shard-backup-dir>]*
[-checkpoint <checkpoint-files-directory>]
[-username <user>] [-security <security-file-path>]
```

Loads data into a store from backup directories. The bulk put API is used by this utility to load data into the target store. To recreate the complete contents of the store, you must specify one directory per shard for each shard associated with the store.

The load utility is highly parallelized. To further boost load performance, you can choose to run multiple concurrent invocations of the load utility on different machines, and assign each invocation a non-overlapping subset of the shard directories, using the `-source` argument. The use of these additional machine resources could significantly decrease overall elapsed load times.

> **Note:**
>
> Creating multiple processes on the same machine is unlikely to be beneficial and could be detrimental, since the two processes are likely to be contending for the same CPU and network resources.

where:

- `-checkpoint <checkpoint-files-directory>` The utility used this directory to checkpoint its progress on a periodic basis. If the load process is interrupted for some reason, the progress checkpoint information is used to skip data that had already been loaded when the load utility is subsequently re-executed with the same arguments. If the `-checkpoint` flag is not specified, progress will not be checkpointed and all the data in the partitions that were already loaded will be reread.
- `-host <hostname>` Identifies the host name of a node in your store.
- `-port <port>` Identifies the registry port in use by the store's node.
- `-security <security-file-path>` Identifies the security file used to specify properties for login.
- `-source <shard-backup-dir>[,<shard-backup-dir>]*` These backup directories typically represent the contents of snapshots created using the snapshot commands described at Taking a Snapshot.
- `-store <storeName>` Identifies the name of the store.
- `-username <user>` Identifies the name of the user to login to the secured store.

makebootconfig

```
java -Xmx64m -Xms64m
-jar $KVHOME/lib/kvstore.jar makebootconfig [-verbose]
```

```
-root <rootDirectory> -host <hostname> -harange <startPort,endPort>
-port <port> [-config <configFile>]
[-store-security <none | configure | enable> ]
[-noadmin]
[-admindir <directory path>]
[-admindirsize <directory size>]
[-storagedir <directory path>]
[-storagedirsize <directory size>]
[-rnlogdir <directory path>]
[-capacity <n_rep_nodes>]
[-storage-type <HD | SSD | NVME | UNKNOWN>]
[-num_cpus <ncpus>][-memory_mb <memory_mb>]
[-servicerange <startPort,endPort>]
[-admin-web-port <admin web service port>]
[-hahost <haHostname>]
[-secdir <security dir>] [-pwdmgr {pwdfile | wallet | <class-name>}]
[-kspwd <password>]
[-external-auth {kerberos}]
  [-krb-conf <kerberos configuration>]
  [-kadmin-path <kadmin utility path>]
  [-instance-name <database instance name>]
  [-admin-principal <kerberos admin principal name>]
  [-kadmin-keytab <keytab file>]
  [-kadmin-ccache <credential cache file>]
  [-princ-conf-param <param=value>]*
[-security-param <param=value>]*
[-mgmt {jmx|none}]
[-dns-cachettl <time in sec>]
      [-force]
```

where:

- `-capacity <n_rep_nodes>` The total number of Replication Nodes a Storage Node can support. The value defaults to "1".

  If capacity is set to 0, then this Storage Node may be used to host Arbiter Nodes.

- `storage-type [HD | SSD | NVME | UNKNOWN]` Specifies the type of disk on which the storage directories reside. You can specify storage type only for a Storage Node and not for replication nodes. You can set one value for this parameter for a Storage Node. The valid values are HD, SSD, NVME and UNKNOWN.

  > **Note:**
  >
  > The parameters `storagedir` and `storagedirsize` are specific to every replication node, whereas storage-type is specific to a Storage Node.

  Storage-type parameter should be used when the data store is unable to determine the type of storage. If the data store makes an incorrect guess, then it can lead to poor performance. Here are two possible situations:

  – **Linux Logical Volumes:** The data store may be unable to detect the disk type when the disk is used as part of a logical volume. This problem may also occur if disks are mounted using non-standard device names. The storage-type should be specified to resolve this.

- **Network Attached Storage (NAS)**: If using NAS, the system can't identify the underlying storage type and it defaults to HD, which can create performance issues. If the destination disk is known (SSD or NVME), you can improve the performance by specifying this in the `storage-type` parameter in the `makebootconfig` utility.

You can use the ping command from the Admin CLI to see which storage-type the data store has detected. Here is an example (using a host with a hard drive):

```
kv-> ping
[...]
    Rep Node [rg1-rn1]
    Status: RUNNING,MASTER sequenceNumber: 51 haPort: 5003
    availableStorageSize: 9 GB
    storageType: HD (default for UNKNOWN)
    serviceStartTime: 2024-04-05 12:55:04 UTC
    stateChangeTime: 2024-04-05 12:55:05 UTC
```

- `-config <configFile>` Only specified if more than one Storage Node Agent process will share the same root directory. This value defaults to `config.xml`.
- `-dns-cachettl <time in sec>` Specifies the number of seconds that Replication Nodes should cache host name to IP address mappings. The default value is -1, which means mappings should be cached indefinitely. A value of 0 means mappings should not be cached. The value of this flag is used to set the `networkaddress.cache.ttl` and `networkaddress.cache.negative.ttl` security properties.
- `-external-auth {kerberos}` Specifies Kerberos as an external authentication service. If no keytab or credential cache has been specified on the command line, an interactive version of the `securityconfig` utility will run.

  This flag is only permitted when the value of the `-store-security` flag is specified as configure or enable.

  To remove Kerberos authentication from a running store, set the value of the `userExternalAuth` security.xml parameter to `NONE`.

  For more information on Kerberos, see Kerberos Authentication Service in the *Security Guide*.

  where `-external-auth` can have the following flags:

  - `-admin-principal <kerberos admin principal name>`

    Specifies the principal used to login to the Kerberos admin interface. This is required while using kadmin keytab or password to connect to the admin interface.

  - `-kadmin-ccache <credential cache file>`

    Specifies the complete path name to the Kerberos credentials cache file that should contain a service ticket for the kadmin/ADMINHOST. `ADMINHOST` is the fully-qualified hostname of the admin server or kadmin/admin service.

    If not specified, the user is prompted to enter the password for principal while logging to the Kerberos admin interface. This flag cannot be specified in conjunction with the `-kadmin-keytab` flag.

  - `-kadmin-keytab <keytab file>`

    Specifies the location of a Kerberos keytab file that stores Kerberos admin user principals and encrypted keys. The security configuration tool will use the specified keytab file to login to the Kerberos admin interface.

The default location of the keytab file is specified by the Kerberos configuration file. If the keytab is not specified there, then the system looks for the file `user.home/krb5.keytab`.

You need to specify the `-admin-principal` flag when using keytab to login to the Kerberos admin, otherwise the correct admin principal will not be recognized. This flag cannot be specified in conjunction with the `-kadmin-ccache` flag.

- `-kadmin-path <kadmin utility path>`

  Indicates the absolute path of the Kerberos kadmin utility. The default value is `/usr/kerberos/sbin/kadmin`.

- `-krb-conf <kerberos configuration>`

  Specifies the location of the Kerberos configuration file that contains the default realm and KDC information. If not specified, the default value is `/etc/krb5.conf`.

- `-princ-conf-param <param=value>*`

  A repeatable argument that allows configuration defaults to be overridden.

  Use the `krbPrincValidity` parameter to specify the expiration date of the Oracle NoSQL Database Kerberos service principal.

  Use the `krbPrincPwdExpire` parameter to specify the password expiration date of the Oracle NoSQL Database Kerberos service principal.

  Use the `krbKeysalt` parameter to specify the list of encryption types and salt types to be used for any new keys created.

- `-force` Optionally specified to force generating the boot configuration files even if boot config verification finds any invalid parameters.
- `-hahost <haHostname>` Can be used to specify a separate network interface for store replication traffic. This defaults to the hostname specified using the -host flag.

  The host name specified here must be resolvable using DNS or the `/etc/hosts` file on any machine running client code that wants to connect to the node.
- `-harange <startPort,endPort>` A range of free ports that the Replication Nodes and Admins use to communicate among themselves. These ports should be sequential. You must assign at least as many ports as the specified capacity for this node, plus an additional port if the node hosts an Admin.
- `-host <hostname>` Identifies a host name associated with the node on which the command is run. This hostname identifies the network interface used for communication with this node.

  The host name specified here must be resolvable using DNS or the `/etc/hosts` file on any machine running client code that wants to connect to the node.
- `-kspwd<password>` For script-based configuration you can use this option to allow tools to specify the keystore password on the command line. If it is not specified, the user is prompted to enter the password.
- `-memory_mb <memory_mb>` The total number of megabytes of memory available in the machine. If the value is 0, the store attempts to determine the amount of memory on the machine, but the value is only available when the JVM used is the Oracle Hotspot JVM. The default value is "0".

  For best results, do not specify this parameter. Oracle NoSQL Database will determine the proper value by default. This parameter should be used sparingly, and only for exceptional situations.

- `-num_cpus <ncpus>` The total number of processors on the machine available to the Replication Nodes. If the value is 0, the system attempts to query the Storage Node to determine the number of processors on the machine. This value defaults to "0".

  For best results, do not specify this parameter. Oracle NoSQL Database will determine the proper value by default. This parameter should be used sparingly, and only for exceptional situations.

- `-port <port>` The TCP/IP port on which Oracle NoSQL Database should be contacted. Sometimes referred to as the registry port. This port must be free on the node on which this command is run.

- `-pwdmgr [ pwdfile | wallet ]`

  Indicates the password manager mechanism used to hold passwords that are needed for access to keystores, and so on.

  where `-pwdmgr` has the following options:

  - `-pwdmgr pwdfile`

    Indicates that the password store is a read-protected clear-text password file. This is the only available option for Oracle NoSQL Database CE deployments. You can specify an alternate implementation.

  - `-pwdmgr wallet`

    Specifies Oracle Wallet as the password storage mechanism. This option is only available in the Oracle NoSQL Database EE version.

- `-root <rootDirectory>` Identifies where the root directory should reside.

- `-secdir <security dir>`

  Specifies the name of the directory within the KVROOT that will hold the security configuration. This must be specified as a name relative to the specified secroot. If not specified, the default value is `security`.

- `-security-param <param=value>*`

  A repeatable argument that allows configuration defaults to be overridden.

  Use the `krbServiceName` parameter to specify the service name of the Oracle NoSQL Database Kerberos service principal.

  Use the `krbServiceKeytab` parameter to specify the keytab file name in security directory of the Oracle NoSQL Database Kerberos service principal.

- `-servicerange <startPort,endPort>` A range of ports that may be used for communication among administrative services running on a Storage Node and its managed services. This parameter is optional and is useful when services on a Storage Node must use specific ports for firewall or other security reasons. By default the services use anonymous ports. The format of the value string is "startPort,endPort."

- `-admin-web-port <admin web service port>` The TCP/IP port on which the admin web service should be started. If not specified, the default port value is `-1`. If a positive integer number is not specified for `-admin-web-port`, then admin web service does not start up along with the admin service. See REST API for Administering Oracle NoSQL Database.

- `-noadmin` Specifies to disable the bootstrap admin service for SNA.

- `-admindir <path>` Specify a path to the directory to be used to store the environment associated with an Admin Node. If no directory is specified, Admin Nodes use a directory under the root directory.

- `-admindirsize <directory size>` Specify the size of the admin storage directory identified by `-admindir`. This parameter is optional. See Managing Admin Directory Size.

  The value specified for this parameter must be a long, followed optionally by a unit string. Accepted unit strings are: KB, MB, and GB, corresponding to 1024, 1024^2, and 1024^3 respectively. Acceptable strings are case insensitive. Valid delimiters between the long value and the unit string are " ", "-", or "_". If you specify the delimiter as " ", your value should be enclosed in double quotes.

  For example:

  ```
  -admindirsize "200 MB"
  -admindirsize 1_gb
  -admindirsize 3000-Mb
  ```

- `-storagedir <path>` Specifies a path to the directory that a Replication Node will use for storage. If your Storage Node will host more than one (1) replication node, specify this argument once for each Replication Node, being sure that the number of arguments does not exceed the Storage Node capacity.

  If you do not specify a storage directory explicitly, Replication Nodes use a directory under the root directory. Be sure to match the number of `-storagedir` arguments to the value of the capacity argument. For example, if your Storage Node hosts four disks, and you are using one disk for each replication node, specify a capacity of four, and have four `-storagedir` arguments, each with a corresponding `-storagedirsize <directory size>` value.

- `-storagedirsize <directory size>` Specifies the size of the directory identified by each `-storagedir` argument. While this parameter is optional, we strongly recommend that you specify its value, since the system takes the `-storagedirsize <directory size>` into consideration when determining store topology. For example, if you have some Storage Nodes each with smaller disk capacity than other store SNs, the system arranges to store less data on those SNs by adjusting partition distribution to shards to match the storage capacity. See Managing Storage Directory Sizes for details.

  Further, it is an error to specify the `-storagedirsize <directory size>` parameter for some named storage directories, but not all.

  Specify the `-storagedirsize <directory size>` value as a long, optionally followed by a unit string. The accepted unit strings are: KB, MB, GB, and TB, corresponding to 1024, 1024^2, 1024^3, 1024^4, respectively. Acceptable strings are case insensitive. Valid delimiter characters between the long value and the unit string are " ", "-", or "_". If you specify the delimiter as " ", your value should be enclosed in double quotes.

  For example:

  ```
  -storagedirsize "200 MB"
  -storagedirsize 4_tb
  -storagedirsize 5000-Mb
  ```

> **Note:**
>
> If you specify the `-storagedir` parameter, but not `-storagedirsize`, `makebootconfig` displays a warning. We strongly recommend specifying both parameters.

- `-rnlogdir <path>` Specify a path to the directory to be used for storing the Replication Node log files. This flag may be used more than once in the command to specify multiple Replication Node log directories, but the number should not exceed the capacity for the node.

  If no directory is specified, by default, the logs are stored under the root directory.

- `-store-security [none | configure | enable]` Specifies if security will be used or not. If `-store-security none` is specified, no security will be in use. If `-store-security configure` is specified, security will be used, and the makebootconfig process invokes the security configuration utility as part processing. If `-store-security enable` is specified, security will be used. You will need to configure security either by utilizing the security configuration utility or by copying a previously created configuration from another system.

  > **Note:**
  >
  > The `-store-security` command is optional. Even if the user does not specify `-store-security`, security is enabled by default. The user must run `securityconfig` utility to create the security folder before starting up the storage node agent.

- `-mgmt {jmx|none}`
  Specifies the type of monitoring to be enabled for the Storage Node . This parameter is optional. The default value is none when monitoring is disabled. Use this parameter to make Java Management Extensions (JMX) agents available for monitoring.

  If you specify `jmx`, JMX interfaces will be used for monitoring the Storage Node and any NoSQL components like Replication Nodes, Admin Node and Storage Node Agent hosted on that Storage Node. JMX agents in Oracle NoSQL Database are read-only interfaces. These interfaces let you poll a Storage Node for information about the Storage Node and about any Replication Nodes or Admins that the Storage Node hosts. The information available from polling includes the service status (RUNNING, STOPPED, UNREACHABLE etc.), operational parameters, and performance metrics. Also, JMX can be used to monitor Arbiter Nodes.

  JMX agents also deliver event traps and notifications for particular events. For example, JMX sends notifications for every service status state change, and any performance limits that the store exceeds. You can get the total number of operation requests using the metric `TotalReq` and the metric `TotalOps` gives the total number of records returned or processed. See Monitoring for Storage Nodes for the definitions of the events available for monitoring .

Creates a configuration file used to start a not-yet-deployed Storage Node to be used in an instance of Oracle NoSQL Database. The file cannot pre-exist. To create the initial "boot config" file used to configure the installation see Installation Configuration Parameters.

You can change parameters after setting them with the `makebootconfig` utility. The commands to use are `change-policy –params` and `plan change-parameters -params`. Changing parameters may require restarting a node. For more information, see CLI Command Reference.

ping

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar ping [-verbose] [-json] [-shard <shardId>]
```

```
-host <hostname> -port <port> or
-helper-hosts  <host:port>[,host:port]*>
-username <user>
-security <security-file-path>
```

Attempts to contact a store to get status of running services. This utility provides both a concise summary of the health of a store, as well as detailed information about the topology of the store. It can signal a red/yellow/green status, to let you know whether the store is in full health, whether the store has experienced some failures but is operational, or whether the store has critical problems. `ping` uses the nodes specified by the `-helper-hosts` or `-host/-port` arguments to locate topology metadata describing the store. Using that topology, `ping` contacts all the RNs, SNs, Arbiters, and Admin services associated with a store. You can also indicate a specific shard to return its status information.

Specify the `-helper-hosts` flag as an alternative to the existing `-host` and `-port` flags. If multiple helper hosts are in use, this utility has multiple nodes it can use to make an initial point of contact with the store, and will have a greater chance of success if some nodes of the store are unavailable.

Specify `-shard <shardId>` to return a subset of information.

Ping Command Line Parameters

The `ping` utility's command line parameters are:

- `-host` identifies the name of a specific host in the store. Use this option to check whether the SNA on that particular host can be contacted.

  If this parameter is specified, then `-port` must also be specified. Further, if the `-host` and `-port` parameters are specified, then the `-helper-hosts` must not be specified.

- `-port` identifies the listening port for a specific host in the store. Use this parameter only if you are also using the `-host` parameter.

- `-helper-hosts` identifies a comma-separated list of one or more host:port pairs in the store. Use this parameter to check the health of the entire store.

  Using the `-helper-hosts` parameter precludes specifying the `-host` and `-port` flags.

  If multiple helper hosts are provided, this utility has multiple nodes it can use to make an initial point of contact with the store, and thus a greater chance of success if some nodes of the store are unavailable. For example:

  ```
  -helper-hosts hst1:5000,hst2:5100, hst3:5100
  ```

- `-username` is the name of the user that you want to ping the store as. This parameter is required if your store is configured to require authentication. This user should have at least `SYSVIEW` access to the store. The built-in `dbadmin` role is sufficient.

- `-security` is the client security configuration file. This parameter is required if your store is configured to require authentication. For information on the parameters contained in this file, see Configuring SSL in the *Java Direct Driver Developer's Guide*. For example:

  ```
  oracle.kv.auth.username=clientUID1
  oracle.kv.auth.pwdfile.file=/home/nosql/login.pwd
  oracle.kv.transport=ssl
  oracle.kv.ssl.trustStore=/home/nosql/client.trust
  ```

If you are using Kerberos, then this file would look something like this:

```
oracle.kv.auth.kerberos.keytab = kerberos/mykeytab
oracle.kv.auth.username = krbuser@EXAMPLE.COM
oracle.kv.auth.external.mechanism=kerberos
oracle.kv.auth.kerberos.services=
node01:oraclenosql/node01.example.com@EXAMPLE.COM
oracle.kv.auth.kerberos.mutualAuth=false
```

- `-verbose` is optional. It causes the `ping` utility to provide additional information about the utility's current actions.
- `-json` causes the `ping` utility to write its output in JSON format.
- `-shard <shardId>` is optional and returns a subset of status information about the specific shard ID you supply. .

  For example:

```
bash-4.1$ java -jar $KVHOME/lib/kvstore.jar ping -host
mynode.mycompany.com
-port 5000 -shard rg2 Pinging components of store mystore based upon
topology
sequence #2376    shard rg2
500 partitions and 3 storage nodes
Time: 2024-04-05 06:57:10 UTC   Version: 24.1.11
Shard Status: healthy
Admin Status: healthy
Zone [name=myshardzone id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
RN Status: online:3 offline:0 maxDelayMillis:0 maxCatchupTimeSecs:0
Storage Node [sn10] on nodeA:5000    Zone: [name=myshardzone id=zn1
type=PRIMARY allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 09:33:45 UTC
Build id: a72484b8b33c Edition: Enterprise
         Rep Node [rg2-rn1]      Status: RUNNING,MASTER
sequenceNumber:71,166
haPort:5010 available storage size:12 GB
Storage Node [sn11] on nodeB:5000    Zone: [name=myshardzone id=zn1
type=PRIMARY allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 09:33:45 UTC
Build id: a72484b8b33c Edition: Enterprise
         Rep Node [rg2-rn2]      Status: RUNNING,REPLICA
sequenceNumber:71,166
haPort:5011 available storage size:14 GB delayMillis:0 catchupTimeSecs:0
Storage Node [sn12] on nodeC:5000    Zone: [name=myshardzone id=zn1
type=PRIMARY allowArbiters=false masterAffinity=false]
Status: RUNNING   Ver: 24.1.11 2024-04-05 09:33:45 UTC
Build id: a72484b8b33c Edition: Enterprise
         Rep Node [rg2-rn3]      Status: RUNNING,REPLICA
sequenceNumber:71,166
haPort:5012 available storage size:24 GB delayMillis:0 catchupTimeSecs:0
```

Ping Exit Codes

The following exit codes can be returned by this utility. Exit codes can be returned both as a process exit code, and as part of the JSON output.

Name	Code	Description
EXIT_OK	0	All services in the store could be located and are in a known, good state (for example, RUNNING).
EXIT_OPERATIONAL	1	One or more services in the store could not be reached, or are in an unknown or not usable state. In this case the store should support all data operations across all shards, as well as all administrative operations, but may be in a state of degraded performance. Some action should be taken to find and fix the problem before part of the store becomes unavailable.
EXIT_NO_ADMIN_QUORUM	2	The Admin Service replication group does not have quorum or is not available at all, and it is not possible to execute administrative operations which modify store configuration. The store supports all normal data operations despite the loss of admin quorum, but this state requires immediate attention to restore full store capabilities.
EXIT_NO_SHARD_QUORUM	3	One or more of the shards does not have quorum and either cannot accept write requests, or is completely unavailable. This state requires immediate attention to restore store capabilities. The exit code takes precedence over `EXIT_NO_ADMIN_QUORUM`, so if this exit code is used, it is possible that the administrative capabilities are also reduced or unavailable.
EXIT_USAGE	100	Illegal ping command usage.
EXIT_TOPOLOGY_FAILURE	101	`ping` was unable to find a topology in order to operate. This could be a store problem, a network problem, or it could be a usage problem with the parameters passed to ping. For example, the specified `-host`/`-port` pair are not part of the store, or none of the hosts specified on `-helper-hosts` can be contacted.

Name	Code	Description
EXIT_UNEXPECTED	102	The utility has experienced an unexpected error.
EXIT_STATUS_UNKNOWN	103	The store is operational but some Replication Nodes are in unknown state.

> **Note:**
>
> Exit codes 1 through 3 may indicate a network connectivity issue that should be checked first before concluding that any services have a problem.

Ping Report Text Output

By default, the `ping` utility reports store health in human readable format. For example:

> **Note:**
>
> Extra line breaks have been added so that the command output fits in the available space.

```
$ java -Xmx64m -Xms64m -jar <KVHOME>/lib/kvstore.jar ping -host nodeA -port
1310
Pinging components of store mystore based upon topology sequence #108
100 partitions and 3 storage nodes
Time: 2024-04-05 10:37:43 UTC Version: 24.1.11
Shard Status: healthy:1 writable-degraded:0 read-only:0 offline:0 total:1
Admin Status: healthy
Zone [name=MyDC id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
RN Status: online:3 read-only:0 offline:0 maxDelayMillis:0
maxCatchupTimeSecs:0
Storage Node [sn1] on nodeA:13100
Zone: [name=MyDC id=zn1 type=PRIMARY allowArbiters=false masterAffinity=false]
Status: RUNNING Ver: 24.1.11 2024-04-05 05:02:01 UTC
Build id: 0ce629097e92 Edition: Enterprise isMasterBalanced:true
        Admin [admin1]          Status: RUNNING,MASTER
        Rep Node [rg1-rn1]      Status: RUNNING,MASTER
        sequenceNumber:227 haPort:13117 available storage size:16 GB storage
type:HD
Storage Node [sn2] on nodeB:13200
    Zone: [name=MyDC id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING Ver: 24.1.11 2024-04-05 05:02:01 UTC
    Build id: 0ce629097e92
        Admin [admin2]          Status: RUNNING,REPLICA
        Rep Node [rg1-rn2]      Status: RUNNING,REPLICA
            sequenceNumber:227 haPort:13217 available storage size:14 GB
storage type:HD delayMillis:0
```

```
            catchupTimeSecs:0
Storage Node [sn3] on nodeC:13300
    Zone: [name=MyDC id=zn1 type=PRIMARY allowArbiters=false
masterAffinity=false]
    Status: RUNNING Ver: 24.1.11 2024-04-05 05:02:01 UTC
    Build id: 0ce629097e92
        Admin [admin3]          Status: RUNNING,REPLICA
        Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
            sequenceNumber:227 haPort:13317 available storage size:24 GB
storage type:HD delayMillis:0
            catchupTimeSecs:0
```

Ping Report JSON Output

When the `-json` command line parameter is specified, this utility provides its report in JSON formatting.

> **Note:**
>
> Extra line breaks have been introduced to allow this output to fit in the available space.

```
bash-3.2$ java -Xmx64m -Xms64m \
-jar dist/lib/kvstore.jar ping -host node01 \
-port 5000 -json
{
  "operation" : "ping",
  "returnCode" : 5000,
  "description" : "No errors found",
  "returnValue" : {
    "topology" : {
      "storeName" : "orcl",
      "sequenceNumber" : 9,
      "numPartitions" : 0,
      "numStorageNodes" : 2,
      "time" : 1539857069504,
      "version" : "24.1.11"
    },
    "adminStatus" : "healthy",
    "shardStatus" : {
      "healthy" : 1,
      "writable-degraded" : 1,
      "read-only" : 0,
      "offline" : 0,
      "total" : 2
    },
    "zoneStatus" : [ {
      "resourceId" : "zn1",
      "name" : "Atlanta",
      "type" : "PRIMARY",
      "allowArbiters" : false,
      "masterAffinity" : false,
```

```
      "rnSummaryStatus" : {
        "online" : 2,
        "offline" : 0,
        "read-only" : 0,
        "hasReplicas" : false
      }
    }, {
      "resourceId" : "zn2",
      "name" : "Boston",
      "type" : "SECONDARY",
      "allowArbiters" : false,
      "masterAffinity" : false,
      "rnSummaryStatus" : {
        "online" : 1,
        "offline" : 0,
        "read-only" : 0,
        "hasReplicas" : true,
        "maxDelayMillis" : 0,
        "maxCatchupTimeSecs" : 0
      }
    } ],
    "snStatus" : [ {
      "resourceId" : "sn1",
      "hostname" : "node01",
      "registryPort" : 5000,
      "zone" : {
        "resourceId" : "zn1",
        "name" : "Atlanta",
        "type" : "PRIMARY",
        "allowArbiters" : false,
        "masterAffinity" : false
      },
      "serviceStatus" : "RUNNING",
      "version" : "24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Edition: Enterprise",
      "adminStatus" : {
        "resourceId" : "admin1",
        "status" : "RUNNING",
        "state" : "MASTER",
        "authoritativeMaster" : true
      },
      "rnStatus" : [ {
        "resourceId" : "rg1-rn1",
        "status" : "RUNNING",
        "requestsEnabled" : "ALL",
        "state" : "MASTER",
        "authoritativeMaster" : true,
        "sequenceNumber" : 23,
        "haPort" : 5002,
        "availableStorageSize" : "3 GB"
      }, {
        "resourceId" : "rg2-rn1",
        "status" : "RUNNING",
        "requestsEnabled" : "ALL",
        "state" : "MASTER",
        "authoritativeMaster" : true,
```

```
        "sequenceNumber" : 23,
        "haPort" : 5003,
        "availableStorageSize" : "3 GB"
      } ],
      "anStatus" : [ ]
    }, {
      "resourceId" : "sn2",
      "hostname" : "node02",
      "registryPort" : 6000,
      "zone" : {
        "resourceId" : "zn2",
        "name" : "Boston",
        "type" : "SECONDARY",
        "allowArbiters" : false,
        "masterAffinity" : false
      },
      "serviceStatus" : "RUNNING",
      "version" : "24.1.11 2024-04-05 09:33:45 UTC  Build id: a72484b8b33c
Edition: Enterprise",
      "adminStatus" : {
        "resourceId" : "admin2",
        "status" : "RUNNING",
        "state" : "REPLICA"
      },
      "rnStatus" : [ {
        "resourceId" : "rg1-rn2",
        "status" : "RUNNING",
        "requestsEnabled" : "ALL",
        "state" : "REPLICA",
        "sequenceNumber" : 23,
        "haPort" : 6003,
        "availableStorageSize" : "3 GB",
        "networkRestoreUnderway" : false,
        "delayMillis" : 0,
        "catchupTimeSecs" : 0,
        "catchupRateMillisPerMinute" : 0
      } ],
      "anStatus" : [ ]
    } ],
    "exitCode" : 0
  }
}
```


restart

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar restart
[-disable-services] [-verbose]
-root <rootDirectory> [-config <bootstrapFileName>]
```

> **Note:**
>
> Before restarting the SNA, set the environment variable `MALLOC_ARENA_MAX` to `1`. Setting `MALLOC_ARENA_MAX` to `1` ensures that the memory usage is restricted to the specified heap size.

Stops and then starts the Oracle NoSQL Database Storage Node Agent and services related to the root directory.

To disable all services associated with a stopped SNA use the `-disable-services` flag. For more information, see Disabling Storage Node Agent Hosted Services.

runadmin

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar runadmin
-host <hostname> -port <port> | -helper-hosts <host:port[,host:port]*>
[-store <storeName> ]
[-username <user>] [-security <security-file-path>]
[-admin-username <adminUser>]
[-admin-security <admin-security-file-path>]
[-timeout <timeout ms>]
[-consistency <NONE_REQUIRED(default) | ABSOLUTE |
               NONE_REQUIRED_NO_MASTER>]
[-durability <COMMIT_SYNC(default) | COMMIT_NO_SYNC |
              COMMIT_WRITE_NO_SYNC>]
[-dns-cachettl <time in sec>]
[-registry-open-timeout <time in ms>]
[-registry-read-timeout <time in ms>]
```

The `runadmin` command starts the Admin command line interface (CLI) utility on the host Storage Node (SN) of your choice. You use the CLI to perform configuration activities for your store.

You can start the CLI on a single host, using the following flags. You can specify any storage node as a single host, including an Admin-only host without any replica nodes:

```
-host <hostname> -port <port>
```

To have more than one host support the Admin command line interface, use the `-helper-hosts` option with two or more hosts:

```
-helper-hosts <host:port[,host:port]*>
```

> **Note:**
>
> The `runadmin -admin-host <adminHost> -admin-port <adminPort>` options are deprecated. Entering either option results in an error. If you are using these options in scripts, replace them with either the `-host` or `-helper-hosts` options (and their port specifications), as noted in the syntax statement.

Use the `-timeout`, `-consistency`, and `-durability` flags to override the connect configuration settings.

where:

- `-timeout`

  Specifies the store request time-out in milliseconds. There is no default.

- `-consistency`

  Indicates the store request consistency. The default value is `NONE_REQUIRED`.

- `-durability`

  Indicates the store request durability. The default value is `COMMIT_SYNC`.

securityconfig

A KVStore can be configured securely. In a secure configuration, network communications between NoSQL clients, utilities, and NoSQL server components are encrypted using SSL/TLS, and all processes must authenticate themselves to the components to which they connect. To set up security when configuring a KVStore, you need to create an initial security configuration. To do this, run securityconfig tool before, after, or as part of the makebootconfig process. You should not create a security configuration at each node. Instead, you should distribute the initial security configuration across all the Storage Nodes in your store. If the stores do not share a common security configuration they will be unable to communicate with one another.

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar securityconfig
```

Various commands used in the securityconfig tool:

- `config create`
- `config add-security`
- `config verify`
- `config update`
- `config merge-trust`
- `config show`
- `config remove-security`

You invoke the `config create` command to create the security configuration.

Use the `config create` command with the `-pwdmgr` option to specify the mechanism used to hold password that is needed for accessing the store. In the example below, Oracle Wallet is used.

```
security-> config create -pwdmgr wallet -root KVROOT
```

Enter a password for your store and then reenter it for verification. The configuration tool will automatically generate some security related files.

For more information on `config create` command, see Creating the security configuration.

Use the `config add-security` command to add the security configuration you just created.

```
security-> config add-security -root KVROOT -secdir security -config
config.xml
```

You can use the `config verify` command to verify the consistency and correctness of the security configuration.

```
security-> config verify -secdir <security dir>
```

You can use the config update command to update the security parameters of a security configuration. You can specify a list of security parameters to update.

```
security-> config update -secdir <security dir> [-param <param=value>]*
```

You can use the config merge-trust command to merge truststore entries from one security configuration into another security configuration. This command is helpful when performing security maintenance, particularly when you need to update the SSL key/certificate. You can specify a list of parameters which includes the directory that contains the security configuration that will be updated (`secroot`) and the directory that contains the security configuration that will provide new trust information ().

```
security-> config merge-trust -root <secroot>
[-secdir <security dir>] -source-root <source secroot>
[-source-secdir <source secdir>] [-ctspwd <client.trust password>]
```

You can use the config show command to print out all security configuration information.

```
security-> config show -secdir <security dir>
```

If you want to disable security for some reason in an existing installation, you can use the config remove-security command.

```
security-> config remove-security -root <kvroot> [-config >config.xml>]
```

For more information on configuring security using securityconfig tool, see Configuring Security with securityconfig.

start

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar start
[-disable-services] [-verbose]
-root <rootDirectory>
[-config <bootstrapFileName>][-restore-from-snapshot]<snapshot-time_snapshot-dir-name> [-update-config {true | false} ]
```

Starts the Oracle NoSQL Database Storage Node Agent (and if configured, store) in the root directory.

To disable all services associated with a stopped SNA use the `-disable-services` flag. For more information, see Disabling Storage Node Agent Hosted Services.

You can optionally start from an existing snapshot, instead of using `-config <bootstrapFileName>`.

To start from a snapshot, use the `-restore-from-snapshot` option, followed by the snapshot directory name with its `snapshot-time` prefix. Specify `-update-config true` to override the existing configuration as part of restoring snapshot data.

status

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar status
-root <rootDirectory> [-config <bootstrapFileName>]
[-verbose] [-disable-services]
```

Attempts to connect to a running Oracle NoSQL Database Storage Node Agent and prints out its status.

For example:

```
java -Xmx64m -Xms64m -jar KVHOME/lib/kvstore.jar  \
status -root KVROOT
SNA Status : RUNNING
```

stop

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar stop
[-disable-services] [-verbose]
-root <rootDirectory> [-config <bootstrapFileName>]
```

Stops the Oracle NoSQL Database Storage Node Agent and services related to the root directory.

To disable all services associated with a stopped SNA use the `-disable-services` flag. For more information, see Disabling Storage Node Agent Hosted Services.

version

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar version
```

Prints version.

xrstart

In a multi-region setup, you must start the XRegion service in each region using the `xrstart` command providing the complete path to the JSON config file. As this service is a long-running process, it is recommended to invoke it as a background process by appending the `-bg` at the end of the command.

> **Note:**
>
> The local KVStore must be started before starting the XRegion Service. If the KVStore in the local region has not started or is not reachable, the XRegion Service will not start.

```
java -Xms256m -Xmx2048m -jar $KVHOME/lib/kvstore.jar xrstart \
-config <complete path to the json.config file> -bg
```

Table 5-7 Parameters used in xrstart command

Parameter	Description
`-config`	Specifies the complete path where the `json.config` file is placed.

Once you run the `xrstart` command, a status.<number of agents>.<agentId>.txt is generated with the process ID of a successfully started agent.

You can view the status of the `xrstart` command execution by reading the contents of `status.<number of agents>.<agentId>.txt`.

```
cat <complete path to the home directory for the XRegion Service>/
status.<number of agents>.<agentId>.txt
```

You can even check the detailed logs in the service log, that is available in the XRegion Service home directory specified in the XRegion Service configuration file (`json.config`) earlier.

xrstatus

In a multi-region setup, you can check the status of the agent using the `xrstatus` command.

```
java -Xms256m -Xmx2048m -jar $KVHOME/lib/kvstore.jar xrstatus -config <path
to the json config file>
```

The parameter config specifies the complete path where the `json.config` file is placed.

xrstop

In a multi-region setup, you can stop any running Xregion service using `xrstop` command. For example, you must stop an Xregion Service if you want to relocate the service to another host machine. Then you must shut it down in the current machine and restart it in the new host machine.

```
java -Xmx1024m -Xms256m -jar $KVHOME/lib/kvstore.jar xrstop \
-config <complete path to the json.config file>
```

The parameter config specifies the complete path where the `json.config` file is placed.

Initial Capacity Planning

To deploy a store, you must specify a replication factor, the desired number of partitions, and the Storage Nodes on which to deploy the store. The following sections describe how to calculate these values based on your application's requirements and the characteristics of the hardware available to host the store.

The resource estimation is a two step process:

1. Determine the storage and I/O throughput capacity of a representative shard, given the characteristics of the application, the disk configuration on each machine, and the disk throughput. As part of this step, you should also estimate the amount of physical memory that each machine requires, and its network throughput capacity.
2. Use the shard level storage and I/O throughput capacities as a basis for extrapolating throughput from one shard to the required number of shards and machines, given the storewide application requirements.

Oracle NoSQL Database distribution includes a spreadsheet for you to use in the capacity planning process. The spreadsheet is located here: `<KVHOME>/doc/misc/InitialCapacityPlanning.xls`.

The spreadsheet has two main sections:

- 1. Shard Capacity
- 2. Store Sizing

The two main sections both have some required parameters for you to complete, as well as parameters with default options.

The next sections in this appendix correspond to named columns in the spreadsheet:

- Column A lists cell names associated with the values in column B.
- Dark purple, bold text labels represent required values for you to provide as input.

- Dark blue, bold text labels indicate default values that you can optionally change. The supplied default values are adequate for most estimates.
- Column C has descriptions of the value or computation associated with the value in column B.
- The first three sections cover Shard Capacity: Application Characteristics, Hardware Characteristics Machine Physical Memory contain required inputs.

The spreadsheet computes all other cells using the following formulas.

- After filling in the required inputs, the *StoreMachines* cell indicates how many Storage Nodes should be available in the Storage Node pool.
- The *StorePartitions* cell indicates how many partitions to specify when creating the store.

The spreadsheet calculations also account for JVM overhead. Keep in mind that these computations yield estimates. The underlying model used as a basis for the estimation makes certain simple assumptions. These assumptions are necessary because it is difficult to provide a simple single underlying model that works well under a wide range of application requirements. Use these estimates only as an initial starting point, and refine them as necessary under a simulated or actual load.

Shard Capacity

To determine the shard capacity, first determine the application and hardware characteristics described in this section. Having determined these characteristics, enter them into the accompanying spreadsheet. The spread sheet will then calculate the capacity of a shard on the basis of the supplied application and hardware characteristics.

Application Characteristics

Replication Factor

In general, a *Primary Replication Factor* of 3 is adequate for most applications and is a good starting point, because 3 replicas allow write availability if a single primary zone fails. It can be refined if performance testing suggests some other number works better for the specific workload. Do not select a *Primary Replication Factor* of 2 because doing so means that even a single failure results in too few sites to elect a new master. This is not the case if you have an Arbiter Node, as a new master can still be elected if the Replication Factor is two and you lose a Replication Node. However, if you have multiple failures before both Replication Nodes are caught up, you may not be able to elect a new master. A *Primary Replication Factor* of 1 is to be avoided in general since Oracle NoSQL Database has just a single copy of the data; if the storage device hosting the data were to fail the data could be lost.

Larger *Primary Replication Factor* provide two benefits:

1. Increased durability to better withstand disk or machine failures.
2. Increased read request throughput, because there are more nodes per shard available to service those requests.

However, the increased durability and read throughput has costs associated with it: more hardware resources to host and serve the additional copies of the data and slower write performance, because each shard has more nodes to which updates must be replicated.

> **Note:**
>
> Only the Primary Replication Factor affects write availability, but both Primary and Secondary Replication Factors, and therefore the Store Replication Factor, have an effect on read availability.

The *Primary Replication Factor* is defined by the cell *RF*.

Average Key Size

Use knowledge of the application's key schema and the relative distributions of the various keys to arrive at an average key length. The length of a key on disk is the number of UTF-8 bytes needed to represent the components of the key, plus the number of components, minus one.

This value is defined by the cell *AvgKeySize*.

Average Value Size

Use knowledge of the application to arrive at an average serialized value size. The value size will vary depending upon the particular serialization format used by the application.

This value is defined by the cell *AvgValueSize*.

Read and Write Operation Percentages

Compute a rough estimate of the relative frequency of store level read and write operations on the basis of the KVS API operations used by the application.

At the most basic level, each KVS get() call results in a store level read operation and each put() operation results in a store level write operation. Each KVS multi key operation (KVStore.execute(), multiGet(), or multiDelete()) can result in multiple store level read/write operations. Again, use application knowledge about the number of keys accessed in these operations to arrive at an estimate.

Express the estimate as a read percentage, that is, the percentage of the total operations on the store that are reads. The rest of the operations are assumed to be write operations.

This value is defined by the cell *ReadOpsPercent*.

Estimate the percentage of read operations that will likely be satisfied from the file system cache. The percentage depends primarily upon the application's data access pattern and the size of the file system cache. Sizing Advice contains a discussion of how this cache is used.

This value is defined by the cell *ReadCacheHitPercent*.

Hardware Characteristics

Determine the following hardware characteristics based on a rough idea of the type of the machines that will be used to host the store:

- The number of disks per machine that will be used for storing KV pairs. This value is defined by the cell *DisksPerMachine*. The number of disks per machine typically determines the Storage Node Capacity as described in Storage Node Parameters.
- The usable storage capacity of each disk. This value is defined by the cell *DiskCapacityGB*.

- The IOPs capacity of each disk. This information is typically available in the disk spec sheet as the number of sustained random IO operations/sec that can be delivered by the disk. This value is defined by the cell *DiskIopsPerSec*.

The following discussion assumes that the system will be configured with one RN per disk.

Shard Storage and Throughput Capacities

There are two types of capacity that are relevant to this discussion: 1) Storage Capacity 2) Throughput Capacity. The following sections describe how these two measures of capacity are calculated. The underlying calculations are done automatically by the attached spreadsheet based upon the application and hardware characteristics supplied earlier.

Shard Storage Capacity

The storage capacity is the maximum number of KV pairs that can be stored in a shard. It is calculated by dividing the storage actually available for live KV pairs (after accounting for the storage set aside as a safety margin and cleaner utilization) by the storage (including a rough estimation of Btree overheads) required by each KV pair.

The KV Storage Capacity is computed by the cell: *MaxKVPairsPerShard*.

Shard I/O Throughput capacity

The throughput capacity is a measure of the read and write ops that can be supported by a single shard. In the calculations below, the logical throughput capacity is derived from the disk IOPs capacity based upon the percentage of logical operations that actually translate into disk IOPs after allowing for cache hits. The Machine Physical Memory section contains more detail about configuring the caches used by Oracle NoSQL Database.

For logical read operations, the shard-wide IOPs is computed as:

```
(ReadOpsPercent * (1 - ReadCacheHitPercent))
```

Note that all percentages are expressed as fractions.

For logical write operations, the shard-wide IOPs is computed as:

```
(((1 - ReadOpsPercent) / WriteOpsBatchSize) * RF)
```

The writeops calculations are very approximate. Write operations make a much smaller contribution to the IOPs load than do the read ops due to the sequential writes used by the log structured storage system. The use of WriteOpsBatchSize is intended to account for the sequential nature of the writes to the underlying JE log structured storage system. The above formula does not work well when there are no reads in the workload, that is, under pure insert or pure update loads. Under pure insert, the writes are limited primarily by acknowledgement latency which is not modeled by the formula. Under pure update loads, both the acknowledgement latency and cleaner performance play an important role.

The sum of the above two numbers represents the percentage of logical operations that actually result in disk operations (the **DiskIopsPercent** cell). The shard's logical throughput can then be computed as:

```
(DiskIopsPerSec * RF)/DiskIopsPercent
```

and is calculated by the cell **OpsPerShardPerSec**.

Memory and Network Configuration

Having established the storage and throughput capacities of a shard, the amount of physical memory and network capacity required by each machine can be determined. Correct configuration of physical memory and network resources is essential for the proper operation of the store. If your primary goal is to determine the total size of the store, skip ahead to Estimate total Shards and Machines but make sure to return to this section later when it is time to finalize the machine level hardware requirements.

> **Note:**
>
> You can also set the memory size available for each Storage Node in your store, either through the `memory_mb` parameter of the `makebootconfig` utility or through the `memorymb` Storage Node parameter. For more information, see Installation Configuration Parameters and Storage Node Parameters respectively.

Machine Physical Memory

The shard storage capacity (computed by the cell *MaxKVPairsPerShard*) and the average key size (defined by the cell *AvgKeySize* cell) can be used to estimate the physical memory requirements of the machine. The physical memory on the machine backs up the caches used by Oracle NoSQL Database.

Sizing the in-memory cache correctly is essential for meeting store's performance goals. Disk I/O is an expensive operation from a performance point of view; the more operations that can be serviced from the cache, the better the store's performance.

Before continuing, it is worth noting that there are two caches that are relevant to this discussion:

1. The JE cache. The underlying storage engine used by Oracle NoSQL Database is Berkeley DB Java Edition (JE). JE provides an in-memory cache. For the most part, this is the cache size that is most important, because it is the one that is simplest to control and configure.
2. The file system (FS) cache. Modern operating systems attempt to improve their I/O subsystem performance by providing a cache, or buffer, that is dedicated to disk I/O. By using the FS cache, read operations can be performed very quickly if the reads can be satisfied by data that is stored there.

Sizing Advice

JE uses a Btree to organize the data that it stores. Btrees provide a tree-like data organization structure that allows for rapid information lookup. These structures consist of interior nodes (INs) and leaf nodes (LNs). INs are used to navigate to data. LNs are where the data is actually stored in the Btree.

Because of the very large data sets that an Oracle NoSQL Database application is expected to use, it is unlikely that you can place even a small fraction of the data into JE's in-memory cache. Therefore, the best strategy is to size the cache such that it is large enough to hold most, if not all, of the database's INs, and leave the rest of the node's memory available for system overhead (negligible) and the FS cache.

Both INs and LNs can take advantage of the FS cache. Because INs and LNs do not have Java object overhead when present in the FS cache (as they would when using the JE cache), they can make more effective use of the FS cache memory than the JE cache memory.

Of course, in order for the FS cache to be truly effective, the data access patterns should not be completely random. Some subset of your key-value pairs must be favored over others in order to achieve a useful cache hit rate. For applications where the access patterns are not random, the high file system cache hit rates on LNs and INs can increase throughput and decrease average read latency. Also, larger file system caches, when properly tuned, can help reduce the number of stalls during sequential writes to the log files, thus decreasing write latency. Large caches also permit more of the writes to be done asynchronously, thus improving throughput.

Determine JE Cache Size

To determine an appropriate JE cache size, use the `com.sleepycat.je.util.DbCacheSize` utility. This utility requires as input the number of records and the size of the application keys. You can also optionally provide other information, such as the expected data size. The utility then provides a short table of information. The number you want is provided in the `Cache Size` column, and in the `Internal nodes and leaf nodes: MAIN cache` row.

For example, to determine the JE cache size for an environment consisting of 100 million records, with an average key size of 12 bytes, and an average value size of 1000 bytes, invoke `DbCacheSize` as follows:

```
java -Xmx64m -Xms64m \
-d64 -XX:+UseCompressedOops -jar je.jar DbCacheSize \
-key 12 -data 1000 -records 100000000 -replicated

      === Environment Cache Overhead ===

      2,536,302 minimum bytes

      To account for JE daemon operation, record locks, HA network
          connections, etc, a larger amount is needed in practice.

      === Database Cache Size ===

       Number of Bytes  Description
       ---------------  -----------
         3,896,520,528  Internal nodes only:
         4,660,565,808  Internal nodes and records version
       110,107,803,216  Internal nodes and leaf nodes
```

Please make note of the following jvm arguments (they have a special meaning when supplied to DbCacheSize):

1. The above example command assumes using Java 11 or later. It is recommended to use Java 17 version. Only 64-bit JVMs are supported by NoSQL DB.
2. The -XX:+UseCompressedOops causes cache sizes to account for CompressedOops mode, which is used by NoSQL DB by default. This mode uses more efficient 32 bit pointers in a 64-bit JVM thus permitting better utilization of the JE cache.
3. The -replicated is used to account for memory usage in a JE ReplicatedEnvironment, which is always used by NoSQL DB.

These arguments when supplied to `Database Cache Size` serve as an indication that the JE application will also be supplied these arguments and `Database Cache Size` adjusts its calculations appropriately. The arguments are used by Oracle NoSQL Database when starting up the Replication Nodes which uses these caches.

The output indicates that a cache size of 3.6 GB is sufficient to hold all the internal nodes representing the Btree in the JE cache. With a JE cache of this size, the IN nodes will be fetched from the JE cache and the LNs will be fetched from the off-heap cache or the disk.

For more information on using the `DbCacheSize` utility, see this Javadoc page. Note that in order to use this utility, you must add the `<KVHOME>/lib/je.jar` file to your Java classpath. <KVHOME> represents the directory where you placed the Oracle NoSQL Database package files.

Having used `DbCacheSize` to obtain the JE cache size, the heap size can be calculated from it. To do this, enter the number obtained from `DbCacheSize` into the cell named *DbCacheSizeMB* making sure to convert the units from bytes to MB. The heap size is computed by the cell *RNHeapMB* as below:

```
(DBCacheSizeMB/RNCachePercent)
```

where *RNCachePercent* is the percentage of the heap that is used for the JE cache. The computed heap size should not exceed 32GB, so that the java VM can use its efficient CompressedOops format to represent the java objects in memory. Heap sizes with values exceeding 32GB will appear with a strikethrough in the *RNHeapMB* cell to emphasize this requirement. If the heap size exceeds 32GB, try to reduce the size of the keys to reduce the JE cache size in turn and bring the overall heap size below 32GB.

The heap size is used as the basis for computing the memory required by the machine as below:

```
(RNHeapMB * DisksPerMachine)/SNRNHeapPercent
```

where *SNRNHeapPercent* is the percentage of the physical memory that is available for use by the RN's hosted on the machine. The result is available in the cell *MachinePhysicalMemoryMB*.

Machine Network Throughput

We need to ensure that the NIC attached to the machine is capable of delivering the application I/O throughput as calculated earlier in Shard I/O Throughput capacity, because otherwise it could prove to be a bottleneck.

The number of bytes received by the machine over the network as a result of write operations initiated by the client is calculated as:

```
(OpsPerShardPerSec * (1 - ReadOpsPercent) *
 (AvgKeySize + AvgValueSize)) * DisksPerMachine
```

and is denoted by *ReceiveBytesPerSec* in the spreadsheet. Note that whether a node is a master or a replica does not matter for the purposes of this calculation; the inbound write bytes come from the client for the master and from the masters for the replicas on the machine.

The number of bytes received by the machine as a result of read requests is computed as:

```
((OpsPerShardPerSec * ReadOpsPercent)/RF) *
 (AvgKeySize + ReadRequestOverheadBytes) * DisksPerMachine
```

where *ReadRequestOverheadBytes* is a fixed constant overhead of 100 bytes.

The bytes sent out by the machine over the network as a result of the read operations has two underlying components:

1. The bytes sent out in direct response to application read requests and can be expressed as:

   ```
   ((OpsPerShardPerSec *  ReadOpsPercent)/RF) *
    (AvgKeySize + AvgValueSize) * DisksPerMachine
   ```

2. The bytes sent out as replication traffic by the masters on the machine expressed as:

   ```
   (OpsPerShardPerSec * (1 - ReadOpsPercent) *
    (AvgKeySize + AvgValueSize) * (RF-1)) * MastersOnMachine
   ```

The sum of the above two values represents the total outbound traffic denoted by SendBytesPerSec in the spreadsheet.

The total inbound and outbound traffic must be comfortably within the NIC's capacity. The spreadsheet calculates the kind of network card, GigE or 10GigE, which is required to support the traffic.

Estimate total Shards and Machines

Having calculated the per shard capacity in terms of storage and throughput, the total number of shards and partitions can be estimated on the basis of the maximum storage and throughput required by the store as a whole using a simple extrapolation. The following inputs must be supplied for this calculation:

1. The maximum number of KV pairs that will stored in the initial store. This value is defined by the cell *MaxKVPairs*. This initial maximum value can be increased subsequently by using the topology transformation commands described in Transforming the Topology Candidate.
2. The maximum read/write mixed operation throughput expressed as operations/sec for the entire store. The percentage of read operations in this mix must be the same as that supplied earlier in the *ReadOpsPercent* cell. This value is defined by the cell *MaxStorewideOpsPerSec*.

The required number of shards is first computed on the basis of storage requirements as below:

```
MaxKVPairs/MaxKVPairsPerShard
```

This value is calculated by the cell *StorageBasedShards*.

The required number of shards is then computed again based upon IO throughput requirements as below:

```
MaxStorewideOpsPerSec/OpsPerShardPerSec
```

This value is calculated by the cell named *OpsBasedShards*.

The maximum of the shards computed on the basis of storage and throughput above is sufficient to satisfy both the total storage and throughput requirements of the application.

The value is calculated by the cell *StoreShards*. To highlight the basis on which the choice was made, the smaller of the two values in *StorageBasedShards* or *OpsBasedShards* has its value crossed out.

Having determined the number of required shards, the number of required machines is calculated as:

```
MAX(RF, (StoreShards*RF)/DisksPerMachine)
```

Number of Partitions

Every shard in the store must contain at least one partition, but it is best to configure the store so that each shard always contains more than one partition. The records in the KVStore are spread evenly across the KVStore partitions, and as a consequence they are also spread evenly across shards. The total number of partitions that the store should contain is determined when the store is initially created. This number is static and cannot be changed over the store's lifetime, so it is an important initial configuration parameter.

The number of partitions must be more than the largest number of shards the store will contain. It is possible to add shards to the store, and when you do, the store is re-balanced by moving partitions between shards (and with them, the data that they contain). Therefore, the total number of partitions is actually a permanent limit on the total number of shards your store is able to contain.

Note that there is some overhead in configuring an excessively large number of partitions. That said, it does no harm to select a partition value that provides plenty of room for growing the store. It is not unreasonable to select a partition number that is 10 times the maximum number of shards.

The number of partitions is calculated by the cell *StorePartitions*.

```
StoreShards * 10
```

Tuning

The default tuning parameters available for the Oracle NoSQL Database software should in general be acceptable for production systems, and so do not require any tuning. However, the underlying operating system will have default values for various kernel parameters which require modification in order to achieve the best possible performance for your store's installation.

This appendix identifies the kernel parameters and other system tuning that you should manage when installing a *production* store. By this, we mean any store whose performance is considered critical. Evaluation systems installed into a lab environment probably do not need this level of tuning unless you are using those systems to measure the store's performance.

> **Note:**
>
> Oracle NoSQL Database is most frequently installed on Linux systems, and so that is what this appendix focuses on.

Turn off the swap

For best performance on a dedicated Oracle NoSQL Database server machine, turn off the swap on the machine. Oracle NoSQL Database processes are careful in their management of the memory they use to ensure that they do not exceed the RAM available on the machine.

The performance gains come from two sources:

1. The I/O overhead due to swap is eliminated. This is especially important if the disk normally used for swap also holds the store's log files used to persist data.
2. Reduces the CPU overhead associated with kswapd.

To turn off the swap, do not mount any swap partitions at boot time. You do this by eliminating all swap related mount entries from `/etc/fstab`. These are all the rows with the entry "swap" in their mount point (column 2) and file system type (column 3) entries.

You can verify that no swap space is being used by running the `free` command. Do this after the `/etc/fstab` has been modified and the machine has been rebooted:

```
-bash-4.1$ free -m
             total       used       free     shared    buffers     cached
Mem:         72695      72493        202          0        289       2390
-/+ buffers/cache:      69813       2882
Swap:            0          0          0
```

The `Swap/total` cell in the above table should read `0`.

Linux Page Cache Tuning

Tune your page cache to permit the OS to write asynchronously to disk whenever possible. This allows background writes, which minimize the latency resulting from serial write operations such as fsync. This also helps with write stalls which occur when the file system cache is full and needs to be flushed to disk to make room for new writes. We have observed significant speedups (15-20%) on insert-intensive benchmarks when these parameters are tuned as described below.

Place the following commands in `/etc/sysctl.conf`. Run

```
sysctl -p
```

to load the new settings so they can take effect without needing to reboot the machine.

```
# Set vm.dirty_background_bytes to 10MB to ensure that
# on a 40MB/sec hard disk a fsync never takes more than 250ms and takes
# just 125ms on average. The value of vm.dirty_background_bytes
# should be increased on faster SSDs or I/O subsytems with higher
# throughput. You should increase this setting by the same proportion
```

```
# as the relative increase in throughput. For example, for a typical SSD
# with a throughput of 160MB/sec, vm.dirty_background_bytes should be set
# to 40MB so fsync takes ~250ms. In this case, the value was increased by
# a factor of 4.
vm.dirty_background_bytes=10485760

# IO calls effectively become synchronous(waiting for the underlying
# device to complete them). This setting helps minimize the
# possibility of a write request stalling in JE while holding the
# write log latch.
vm.dirty_ratio=40

# Ensures that data does not hang around in memory longer than
# necessary. Given JE's append-only style of writing, there is
# typically little benefit from having an intermediate dirty page
# hanging around, because it is never going to be modified. By
# evicting the dirty page earlier, its associated memory is readily
# available for reading or writing new pages, should that become
# necessary.
vm.dirty_expire_centisecs=1000
```

Earlier versions of the Linux kernel may not support `vm.dirty_background_bytes`. On these older kernels you can use `vm.dirty_background_ratio` instead. Pick the ratio that gets you closest to 10MB. On some systems with a lot of memory this may not be possible due to the large granularity associated with this configuration knob. A further impediment is that a ratio of 5 is the effective minimum in some kernels.

```
vm.dirty_background_ratio=5
```

Use `sysctl -a` to verify that the parameters described here are set as expected.

OS User Limits

When running a large Oracle NoSQL Database store, the default OS limits may be insufficient. The following sections list limits that are worth reviewing.

File Descriptor Limits

Use `ulimit -n` to determine the maximum number of files that can be opened by a user. The number of open file descriptors may need to be increased if the defaults are too low. It's worth keeping in mind that each open network connection also consumes a file descriptor. Machines running clients as well as machines running RNs may need to increase this limit for large stores with 100s of nodes.

Add entries like the ones below in `/etc/security/limits.conf` to change the file descriptor limits:

```
$username soft nofile 10240
$username hard nofile 10240
```

where `$username` is the username under which the Oracle NoSQL Database software runs.

Note that machines hosting multiple replication nodes; that is, machines configured with a capacity > 1; will need larger limits than what is identified here.

Process and Thread Limits

Use `ulimit -u` to determine the maximum number of processes (threads are counted as processes under Linux) that the user is allowed to create. Machines running clients as well as machines running RNs may need to increase this limit to accommodate large numbers of concurrent requests.

Add entries like the ones below in `/etc/security/limits.conf` to change the thread limits:

```
$username soft nproc 8192
$username hard nproc 8192
```

where `$username` is the username under which the Oracle NoSQL Database software runs.

Note that machines hosting multiple replication nodes; that is, machines configured with a capacity > 1; will need larger limits than what is identified here.

Linux Network Configuration Settings

Before continuing, it is worth checking that the network interface card is configured as expected during the initial setup of each SN, because it is harder to debug these problems later when such configuration problems show up under load.

Use the following command to determine which network interface is being used to access a particular subnet on each host. This command is particularly useful for machines with multiple NICs:

```
$ ip addr ls to 192.168/16
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    state UP qlen 1000
    inet 192.168.1.19/24 brd 192.168.1.255 scope global eth0
```

Use the following command to get information about the configuration of the NIC:

```
$ ethtool -i eth2
driver: enic
version: 2.1.1.13
firmware-version: 2.0(2g)
bus-info: 0000:0b:00.0
```

Use the following command to get information about the NIC hardware:

```
$ lspci -v | grep "Ethernet controller"
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit
Ethernet Controller (rev 02)
```

Use the following command to get information about the network speed. Note that this command requires sudo:

```
$ sudo ethtool eth0 | grep Speed
    Speed: 1000Mb/s
```

You may want to consider using 10 gigabit Ethernet, or other fast network implementations, to improve performance for large clusters.

Server Socket Backlog

The typical default maximum server socket backlog, typically set at 128, is too small for server style loads. It should be at least 1K for server applications and even a 10K value is not unreasonable for large stores.

Set the `net.core.somaxconn` property in `sysctl.conf` to modify this value.

Isolating HA Network Traffic

If the machine has multiple network interfaces, you can configure Oracle NoSQL Database to isolate HA replication traffic on one interface, while client request traffic uses another interface. Use the `-hahost` parameter of the `makebootconfig` command to specify the interface to be used by HA as in the example below:

```
java -Xmx64m -Xms64m \
-jar kvstore.jar makebootconfig -root /disk1/kvroot \
-host sn10.example.com -port 5000 -harange 5010,5020 \
-admindir /disk2/admin -admindirsize 2 GB
-storagedir /disk2/kv -hahost sn10-ha.example.com
```

In this example, all client requests will use the interface associated with `sn10.example.com`, while HA traffic will use the interface associated with `sn10-ha.example.com`.

Receive Packet Steering

When multiple RNs are located on a machine with a single queue network device, enabling Receive Packet Steering (RPS) can help performance by distributing the CPU load associated with packet processing (soft interrupt handling) across multiple cores. Multi-queue NICs provide such support directly and do not need to have RPS enabled.

Note that this tuning advice is particularly appropriate for customers using Oracle Big Data Appliance.

You can determine whether a NIC is multi-queue by using the following command:

```
sudo ethtool -S eth0
```

A multi-queue NIC will have entries like this:

```
 rx_queue_0_packets: 271623830
     rx_queue_0_bytes: 186279293607
     rx_queue_0_drops: 0
     rx_queue_0_csum_err: 0
     rx_queue_0_alloc_failed: 0
     rx_queue_1_packets: 273350226
     rx_queue_1_bytes: 188068352235
     rx_queue_1_drops: 0
     rx_queue_1_csum_err: 0
     rx_queue_1_alloc_failed: 0
     rx_queue_2_packets: 411500226
     rx_queue_2_bytes: 206830029846
```

```
     rx_queue_2_drops: 0
     rx_queue_2_csum_err: 0
     rx_queue_2_alloc_failed: 0
...
```

For a 32 core Big Data Appliance using Infiniband, use the following configuration to distribute receive packet processing across all 32 cores:

```
echo ffffffff > /sys/class/net/eth0/queues/rx-0/rps_cpus
```

where `ffffffff` is a bit mask selecting all 32 cores.

For more information on RPS please consult:

1. About the Unbreakable Enterprise Kernel
2. Receive packet steering

MTU Size

When using machines connected to networks running at 1000Mb/s or higher speeds, it is recommended that you enable jumbo frames on the machines that are hosting the RNs. HA replication benefits from the use of Jumbo frames such that the feeder (via the HA parameter: feederBatchBuffKb) uses a default batch buffer size of 8K, which is well matched to use a Jumbo frame.

Setting the MTU to 9000 is also helps in improving network performance on KV client machines with high speed networks, especially if the request or response payloads frequently exceed the default MTU size of 1500.

To enable jumbo frames, set the MTU to 9000 on each machine. Also, verify that this MTU is supported on the entire network path between machines hosting the RNs.

For example, to determine the speed of the *ens3* interface, use the following command:

```
# ip link show ens3
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether 00:00:17:01:2c:b6 brd ff:ff:ff:ff:ff:ff
```

If required, change the MTU configuration using the *ip* command:

```
# ip link set ens3 mtu 9000
```

Check AES Intrinsics Settings

While most modern hardware systems enable AES Intrinsics by default, you can check these settings yourself to confirm their use.

An Oracle NoSQL Database installation using the SSL/TLS encryption gets better performance if it can take advantage of hardware acceleration available on the host machine.

Most SSL cipher suites use the AES encryption algorithm, and most modern processors support hardware acceleration for AES. To confirm that a Java installation is taking advantage of AES hardware acceleration, check to see if AES intrinsics are enabled. You can get that information by printing flag values for the Java virtual machine from your terminal using the -

XXPrintFlagsFinal flag, as follows. Then, search for the two boolean flags UseAES, and UseAESIntrinsics. In this example, results show that AES intrinsics are enabled.

```
java -XX:+PrintFlagsFinal -version | grep 'AES\|Intrinsics'
bool UseAES = true {product} {default}
bool UseSSE42Intrinsics = true {ARCH product} {default}
java version "10.0.2" 2018-07-17
Java(TM) SE Runtime Environment 18.3 (build 10.0.2+13)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10.0.2+13, mixed mode)
```

Setting AES Intrinsics

For best performance, enable AES intrinsics on all machines that support them. If not enabled when you run the check just described, you must specify -XX:+UseAES and -XX:+UseAESIntrinsics for every JVM command line that uses SSL, using these flags:

```
java -XX:+UseAES -XX:+UseAESIntrinsics [...]
```

You can add these two flags to the JVM options for RNs by setting the configProperties parameter. See Replication Node Parameters .

Client applications that make calls to the NoSQL API, should specify these system properties on the Java command line for the application.

Viewing Key Distribution Statistics

As you might already know, Oracle NoSQL Database stores the data by distributing the rows across all the partitions by hashing each row's shard key. Based on the activity in your store's tables, Oracle NoSQL Database collects the key distribution data into internally managed system tables. As needed, you can access these statistics by querying these system tables.

As an Oracle NoSQL Database administrator, you may encounter many situations where you need to view the key distribution statistics. To discuss one such use-case, consider a situation where you are not able achieve the expected amount of throughput for your Oracle NoSQL Database in spite of having multiple shards in your cluster. This might happen if the data in your store is not distributed across the shards evenly. In order to confirm if this is the reason behind low throughput, you need a mechanism to understand how the data is distributed across the Oracle NoSQL Database cluster. The Key Distribution Statistics provided by the Oracle NoSQL Database can help you understand the data distribution across multiple partitions and shards in your store.

The two system tables into which the Oracle NoSQL Database collects the key distribution statistics are:

- SYS$TableStatsPartition
- SYS$TableStatsIndex

Oracle NoSQL Database manages and maintains these system tables internally. When you enable security on your store, these system tables are read-only. Regardless of security, the schema for system tables is immutable. The name of system tables is defined with the prefix SYS$. You are not allowed to create any other table name using this reserved prefix.

SYS$TableStatsPartition
This table stores the table key statistics at the partition level. It contains a row for each partition for every table. For example, if you created a store with 100 partitions, this table contains 100

rows for every table in your store. The statistics stored per partition for each table in your store are:

1. The number of rows stored
2. The average size of keys in bytes
3. The size in bytes consumed by the rows

The structure of the `SYS$TableStatsPartition` table is as below:

Column	Data Type	Description
`tableName`	string	Name of the table whose Key Distribution Statistics are being stored.
`partitionId`	integer	Partition ID
`shardId`	integer	Shard ID
`count`	long	Number of rows stored.
`avgKeySize`	integer	The average size of keys in bytes.
`tableSize`	long	The size in bytes consumed by the rows.
`tableSizeWithTombstones`	long	The table storage in bytes including tombstones.

> **Note:**
>
> The tombstone is a small piece of storage when a row in a multi-region table is deleted. This per-row storage overhead is used to keep some metadata of the deleted row, which will be used in conflict resolution when another row is replicated from remote regions. The tombstone will expire in 7 days after it is created and the storage will be released. For tables without tombstones (For example, non-multi-region tables, system tables, etc,) the metric `tableSizeWithTombstones` would be the same as the metric `tableSize` in the system table. The difference between the two metrics is the total storage size of tombstones in the table.

`SYS$TableStatsIndex`

This table stores the index key statistics at the shard level. This table contains a row for each shard for every index. You do not have direct control over the number of shards created in your store, but you can always view the store topology to know how many shards are created in your store. For more information, see show topology.

The statistics stored per shard for each table in your store are:

1. The number of index rows
2. The average size of the index keys in bytes
3. The size in bytes consumed by the index rows

The structure of `SYS$TableStatsIndex` system table is as below:

Column	Data Type	Description
`tableName`	string	Name of the table whose Key Distribution Statistics are being stored.

Column	Data Type	Description
indexName	string	Name of the index
shardId	integer	Shard ID
count	long	Number of index rows stored.
avgKeySize	integer	Average size of index keys in bytes.
indexSize	long	The size in bytes consumed by the index rows.

Gathering the Key Distribution Statistics

The gathering of the key distribution statistics into the system tables is determined by two parameters:

- rnStatisticsEnabled:
  In Oracle NoSQL Database, the Key Distribution Statistics are enabled by default for all newly created stores. You can disable the capturing of these statistics by executing the following command from Admin Command Line Interface (CLI):

  ```
  plan change-parameters -wait -all-rns -params "rnStatisticsEnabled=false"
  ```

- rnStatisticsGatherInterval:
  In Oracle NoSQL Database, the default time interval between two consecutive updates on SYS$TableStatsPartition and SYS$TableStatsIndex is 24 hours. You can change the time interval between the capture of these statistics by modifying the rnStatisticsGatherInterval parameter. The time unit specified must be in days, hours, or minutes.

  For example, to instruct Oracle NoSQL Database to collect the Key Distribution Statistics after every minute, execute the following command from Admin Command Line Interface (CLI):

  ```
  plan change-parameters -wait -all-rns -params
  "rnStatisticsGatherInterval=1 min"
  ```

> **Note:**
>
> Enabling the Key Distribution Statistics does not immediately trigger the collection of statistics. Oracle NoSQL Database initiates the statistics collection at a time based on the collection interval defined by the rnStatisticsGatherInterval parameter.

rnStatisticsGatherInterval

Reading the Key Distribution Statistics

You can query the system tables to get key distribution data or review the gathering process.

In order to get a complete set of statistics for a given table, you must aggregate the per-partition values stored for that table in the SYS$TableStatsPartition system table.

For example, to get the total number of rows in a table named myTable, you must sum the values in the count column for all the rows in the SYS$TableStatsPartition table where tableName = myTable.

Example Query:

```
sql-> select * from SYS$TableStatsPartition where tableName = 'myTable';
```

Result:

```
{"tableName":"myTable","partitionId":8,"shardId":3,"count":0,"avgKeySize":0,"t
ableSize":0}
{"tableName":"myTable","partitionId":9,"shardId":4,"count":0,"avgKeySize":0,"t
ableSize":0}
{"tableName":"myTable","partitionId":1,"shardId":1,"count":0,"avgKeySize":0,"t
ableSize":0}
{"tableName":"myTable","partitionId":4,"shardId":2,"count":0,"avgKeySize":0,"t
ableSize":0}
{"tableName":"myTable","partitionId":7,"shardId":3,"count":50,"avgKeySize":15,
"tableSize":103}
{"tableName":"myTable","partitionId":10,"shardId":4,"count":50,"avgKeySize":15
,"tableSize":103}
{"tableName":"myTable","partitionId":5,"shardId":2,"count":0,"avgKeySize":0,"t
ableSize":0}
{"tableName":"myTable","partitionId":6,"shardId":2,"count":0,"avgKeySize":0,"t
ableSize":0}
{"tableName":"myTable","partitionId":2,"shardId":1,"count":0,"avgKeySize":0,"t
ableSize":0}
{"tableName":"myTable","partitionId":3,"shardId":1,"count":0,"avgKeySize":0,"t
ableSize":0}
```

In the above result, observe that there are 50 keys each in `"partitionId":7,"shardId":3` and `"partitionId":10,"shardId":4` whereas all the other partitions and shards are empty. This shows that the key data is not distributed evenly across all the partitions and shards.

Similarly, you can query the `SYS$TableStatsIndex` system table to read the index key distribution statistics for a given table at the shard level.

For example, to get the total number of index rows in a table named `myTable`, you must sum the values in the count column for all the index rows in the `SYS$TableStatsIndex` table where tableName = myTable.

Example Query:

```
sql-> select * from SYS$TableStatsIndex where tableName = 'myTable';
```

Result:

```
{"tableName":"myTable","indexName":"idx_shard_key","shardId":3,"count":50,"avg
KeySize":1,"indexSize":75}
{"tableName":"myTable","indexName":"idx_shard_key","shardId":4,"count":50,"avg
KeySize":1,"indexSize":75}
{"tableName":"myTable","indexName":"idx_shard_key","shardId":1,"count":0,"avgK
eySize":0,"indexSize":0}
{"tableName":"myTable","indexName":"idx_shard_key","shardId":2,"count":0,"avgK
eySize":0,"indexSize":0}
```

As you can see from the above result, there are 50 index keys each in `"shardId":3` and `"shardId":4` whereas all the other shards are empty. This shows that the index key data is not distributed evenly across all the shards.

Retention of the Key Distribution Statistics
After collecting the key distribution statistics, they are retained in the system tables for a fixed time period. This value is determined by the `rnStatisticsTTL` parameter. By default, these statistics are retained for 60 days. However, you can change this value by executing the change-parameters plan from the Admin CLI. The time unit specified must be in days or hours.

For example, execute the following command from Admin Command Line Interface (CLI) to retain the Key Data Statistics in the system tables for 90 days:

```
plan change-parameters -wait -all-rns -params "rnStatisticsTTL=90 days"
```

Few points to note are:

- Any changes that you make to the `rnStatisticsTTL` parameter will not be applied to the existing rows in the `SYS$TableStatsPartition` and `SYS$TableStatsIndex` tables. They will take effect only after the next gathering scan.
- If you disable the collection of Key Distribution Statistics, all the rows present in the system tables will expire after the current Time to Live (TTL) period.
- If you drop any tables or indexes in your store, their statistics rows present in the system tables will also expire after the TTL period.
- Even if you change the `rnStatisticsTTL` to a value less than `rnStatisticsGatherInterval`, all the existing statistics rows will only expire as the TTL value defined during the last scan.
- `rnStatisticsTTL` can be set to 0 days. However, this is not recommended as it disables automatic removal of the statistics rows.

Examples: Key Distribution Statistics

Key distribution statistics can also be used to provide estimates of other information about tables that may prove useful.

Example 5-4 Key Distribution Statistics

To estimate the number of elements in each table, perform the following query:

```
SELECT tableName,
    sum(count) AS count
FROM SYS$TableStatsPartition
WHERE NOT contains (tableName, "$")
GROUP BY tableName
```

The clause `WHERE NOT CONTAINS (tableName, "$")` filters out system tables by only including tables whose names do not contain the "$" character.

The clause `GROUP BY tableName` is what causes the sums to be computed over all of the partition entries for the same table.

Example 5-5 Key Distribution Statistics

To estimate the average key size for each table, perform the following query:

```
SELECT tableName,
    CASE WHEN sum(count) = 0
        THEN 0
        ELSE sum(avgKeySize*count)/sum(count)
    END AS avgKeySize
FROM SYS$TableStatsPartition
WHERE NOT contains(tableName, "$")
GROUP BY tableName
```

The case clause skips entries whose count is zero, and otherwise weights each entry by the element count, dividing the result by the total count.

Example 5-6 Key Distribution Statistics

To estimate the number of elements in each index, perform the following query:

```
SELECT tableName,
       indexName,
       sum(count) AS count
FROM SYS$TableStatsIndex
WHERE NOT contains(tableName, "$")
GROUP BY tableName, indexName
```

Example 5-7 Size of the tables

The clause WHERE NOT CONTAINS (tableName, "$") filters out system tables by only including tables whose names do not contain the "$" character.

```
SELECT tableName,TableSize,
tableSizeWithTombstones FROM SYS$TableStatsPartition
WHERE NOT contains(tableName,"$");
```

For tables without tombstones (For example, non-multi-region tables, system tables, etc,), the metric `tableSizeWithTombstones` would be the same as the metric tableSize in the system table. The difference between the two metrics is the total storage size of tombstones in the table.

Example 5-8 Determine the size before a table export

You want to export a gigantic table to another place (another disk, kvstore, etc.), You can use `tableSize` to determine the size of the data. You can determine the size of live data without tombstone for that table since export does not copy tombstones.

```
SELECT tableName,TableSize FROM SYS$TableStatsPartition
WHERE NOT contains(tableName,"$");
```

Solid State Drives (SSDs)

If you are planning on using Solid State Drives (SSDs) for your Oracle NoSQL Database deployment, a special consideration should be taken. Because of how SSDs work, I/O latency

can become an issue with SSDs over time. Correct configuration and use of `trim` can help minimize these latency issues.

Trim requirements

In general, for TRIM to be effective, the following requirements must be met:

- The SSD itself must support trim.
- Linux-kernel 2.6.33 or later.
- Filesystem ext4 (ext3 does not support trim).

Enabling Trim

The `trim` support must be explicitly enabled for the ext4 file system. You should mount the file system with trim enabled.

Diagnostics Utility

In order to catch configuration errors early, you can use this tool when troubleshooting your KVStore. Also, you can use this tool to package important information and files to send them to Oracle Support, for example.

The usage for the utility is:

```
> java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar diagnostics {setup | collect} [args]
```

Setting up the tool

You should first run the `diagnostics setup` command in order to setup the tool. This command generates the configuration file `sn-target-list` with the Storage Node target list, which contains the IP/hostname, registry ports, and root directory of SNAs in the remote machines.

The usage of this command is:

```
diagnostics setup {-add |
-list |
-delete |
-clear} [args]
```

where:

- -add

  Adds the specified information of each SNA to the `sn-target-list`. The usage is:

  ```
  setup -add -store <store name>
  -sn <SN name>
  -host <host>
  -rootdir <kvroot directory>
  ```

```
[-sshusername <SSH username>]
[-configdir <directory of configuration>]
```

In the `sn-target-list`, the SNA information has the following format:

```
<store name>|<sn name>|<SSH username@host>|<root directory>
```

For example:

```
mystore|sn3|lroot@localhost|/scratch/tests/kvroot
```

> **Note:**
>
> You can also create and edit the `sn-target-list` manually in your preferred text editor to add or delete any SNA information.

- -list

  Lists and tests the SNAs information of the `sn-target-list`. The usage is:

  ```
  setup -list [-configdir <configuration file directory>]
              [-sshusername <SSH username>]
  ```

  This command checks if:

  – The host name is reachable or not.

  – The root directory exists or not.

- -delete

  Specified to delete the information of the specified SNA from the `sn-target-list`.

  The usage of this command is:

  ```
  diagnostics setup -delete
  [-store <store name>]
  [-sn <SN name>]
  [-host <host>]
  [-rootdir kvroot directory>]
  [-sshusername <SSH username>]
  [-configdir <configuration file directory>]
  ```

- -clear

  Specified to clear all the SNA information in the `sn-target-list`.

  The usage of this command is:

  ```
  diagnostics setup -clear [-configdir <configuration file directory>]
  ```

- -configdir

  Optionally specified to change the default directory where the `sn-target-list` file is saved. If the flag is not specified, the default directory is the working directory.

Packaging Information and Files

After completing the `diagnostics setup`, you can use the `diagnostics collect` tool to package important information and files to be able to send them to Oracle Support, for example.

The usage of this command is:

```
diagnostics collect -logfiles
[-host <host name of a SN in topology>]
[-port <registry port of a SN in topology>]
[-sshusername <SSH username>]
[-username <store username>]
[-security <security-file-path>]
[-configdir <location of Storage Node target file>]
[-savedir <destination directory for log files>]
[-nocompress]
```

where:

- -logfiles

  Specified to gather log files of KVStore and pack them up into a compressed file. These files can be a part of the `KVROOT` directory or the `rnlogdir` directory, depending on what was specified when running the `makebootconfig` file.

  > **Note:**
  >
  > In old servers, `je.[info, config, stat]` files will still be a part of the environment directory.

  Available disk space in all the hosting machines and the client machine is required. If available disk space is not enough, an error message is prompted. Log files are helpful to analyze some sophisticated issues.

- -host

  Specifies the host of a Storage Node. If specified, it detects a running topology in order to update the `sn-target-list` without having to run `diagnostics setup` first. It needs to be specified with -port.

- -port

  Specifies the host of a Storage Node. If specified, it detects a running topology in order to update the `sn-target-list` without having to run `diagnostics setup` first. It needs to be specified with -host.

- -sshusername

  Specifies a SSH username to log on as in a Storage Node.

- -username

  Specifies a username to log on as in a secure deployment.

- -security

In a secured deployment, specifies a path to the security file. If not specified in a secure store, updating the `sn-target-list` will fail.

- -configdir

  Specifies the directory which contains the `sn-target-list`. If the flag is not specified, the default directory is the working directory.

- -savedir

  Optionally used to specify the path of the directory to contain all the log files. If the flag is not specified, the default directory is the working directory.

- -nocompress

  Specifies that log files should be copied directly instead of being compressed. If the log files size is large, copying can take a while. You should use -nocompress if the remote servers do not have an unzip tool or if compress mode encounters errors.

Verifying Storage Node configuration

You can use the `diagnostics verify` tool to verify the configuration of the specified Storage Nodes. You can also check if the configuration of each Storage Node is consistent with other members of the cluster.

The usage of this command is:

```
diagnostics verify { -checkLocal | -checkMulti }
[-host <host name of a SN in topology>]
[-port <registry port of a SN in topology>]
[-sshusername <SSH username>]
[-username <store username>]
[-security <security-file-path>]
[-configdir <location of Storage Node target file>]
```

where:

- -checkLocal

  If specified, verifies the configuration of the specified Storage Nodes.

- -checkMulti

  If specified, verifies that the configuration of each Storage Node is consistent with other members of the cluster.

- -host

  Specifies the host of a Storage Node. If specified, it detects a running topology in order to update the `sn-target-list` without having to run `diagnostics setup` first. It needs to be specified with -port.

- -port

  Specifies the host of a Storage Node. If specified, it detects a running topology in order to update the `sn-target-list` without having to run `diagnostics setup` first. It needs to be specified with -host.

- -sshusername

  Specifies a SSH username to log on as in a Storage Node.

- -username

Specifies a username to log on as in a secure deployment.

- -security

  In a secured deployment, specifies a path to the security file. If not specified in a secure store, updating the `sn-target-list` will fail.

- -configdir

  Specifies the directory which contains the `sn-target-list`. If the flag is not specified, the default directory is the working directory.