

Oracle

*NoSQL Database
Administrator's Guide*

12c Release 1
Library Version 12.1.3.5



Legal Notice

Copyright © 2011, 2012, 2013, 2014, 2015 Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Published 12/3/2015

Table of Contents

Preface	ix
Conventions Used in This Book	ix
1. Installing Oracle NoSQL Database	1
Installation Prerequisites	1
Installation	2
Installation Configuration	3
Configuring the Firewall	7
2. Plans	9
Using Plans	9
Feedback While a Plan is Running	9
Plan States	10
Reviewing Plans	11
Plan Ownership	11
3. Configuring the KVStore	12
Configuration Overview	12
Start the Administration CLI	12
The plan Commands	13
Configure and Start a Set of Storage Nodes	13
Name your KVStore	13
Create a Zone	14
Create an Administration Process on a Specific Host	15
Create a Storage Node Pool	16
Create the Remainder of your Storage Nodes	17
Create and Deploy Replication Nodes	17
Configuring with Multiple Zones	18
Using a Script to Configure the Store	25
Smoke Testing the System	26
Troubleshooting	27
Where to Find Error Information	28
Service States	29
Useful Commands	30
4. Determining Your Store's Configuration	31
Steps for Changing the Store's Topology	32
Make the Topology Candidate	32
Transform the Topology Candidate	33
Increase Data Distribution	33
Increase Replication Factor	35
Balance a Non-Compliant Topology	35
View the Topology Candidate	36
Validate the Topology Candidate	36
Preview the Topology Candidate	37
Deploy the Topology Candidate	37
Verify the Store's Current Topology	39
5. Upgrading an Existing Oracle NoSQL Database Deployment	41
Preparing to Upgrade	41
General Upgrade Notes	41

Upgrade from Release 2.0 to Release 3.0	43
Using a Script to Upgrade to Release 3	48
Upgrade from NoSQL DB Release 1.0 to NoSQL DB Release 2.0	49
6. Administrative Procedures	51
Backing Up the Store	51
Taking a Snapshot	51
Snapshot Management	52
Recovering the Store	53
Using the Load Program	54
Load Program and Metadata	55
Restoring Directly from a Snapshot	57
Managing Avro Schema	58
Adding Schema	58
Changing Schema	58
Disabling and Enabling Schema	59
Showing Schema	59
Increasing the capacity of a Storage Node	59
Replacing a Failed Storage Node	61
Replacing a Failed Disk	64
Repairing a Failed Zone by Replacing Hardware	66
Disabling Storage Node Agent Hosted Services	66
Verifying the Store	67
Monitoring the Store	71
Events	72
Other Events	72
Setting Store Parameters	73
Changing Parameters	73
Setting Store Wide Policy Parameters	74
Admin Parameters	75
Storage Node Parameters	76
Replication Node Parameters	78
Security Parameters	79
Admin Restart	80
Replication Node Restart	81
Removing an Oracle NoSQL Database Deployment	82
Fixing Incorrect Storage Node HA Port Ranges	82
7. Failover and Switchover Operations	84
Repairing a Failed Zone	84
Performing a failover	85
Performing a switchover	88
8. Standardized Monitoring Interfaces	92
Simple Network Management Protocol (SNMP) and Java Management Extensions (JMX)	92
Enabling Monitoring	92
In the Bootfile	92
By Changing Storage Node Parameters	93
Displaying the NoSQL DB MBeans	94
9. Integrating Oracle Enterprise Manager (OEM) with Oracle NoSQL Database	95
Importing and Deploying the EM Plug-in	95

Adding NoSQL Database Targets	96
Components of a NoSQL Store	100
Store Targets	100
Store Page	100
Storage Node Page	103
Shard Page	103
Replication Node Page	104
A. KVStore Command Reference	107
CLI Commands and Subcommands	107
aggregate	108
aggregate kv	108
aggregate table	110
await-consistent	111
change-policy	112
configure	112
connect	112
connect admin	112
connect store	113
delete	113
delete kv	113
delete table	113
ddl	114
ddl add-schema	114
ddl enable-schema	114
ddl disable-schema	114
execute	114
exit	115
get	115
get kv	116
get table	118
help	119
hidden	119
history	120
load	120
logtail	121
page	121
ping	121
plan	121
plan add-index	123
plan add-table	123
plan cancel	125
plan change-parameters	125
plan change-storagedir	126
plan change-user	126
plan create-user	126
plan deploy-admin	127
plan deploy-datacenter	127
plan deploy-sn	127
plan deploy-topology	127

plan deploy-zone	127
plan drop-user	128
plan evolve-table	128
plan execute	129
plan failover	129
plan grant	130
plan interrupt	130
plan migrate-sn	130
plan remove-admin	130
plan remove-index	131
plan remove-datacenter	131
plan remove-sn	131
plan remove-table	131
plan remove-zone	132
plan repair-topology	132
plan revoke	132
plan start-service	132
plan stop-service	132
plan wait	133
pool	133
pool create	133
pool join	133
pool remove	133
put	134
put kv	134
put table	135
repair-admin-quorum	136
show	136
show admins	137
show datacenters	137
show events	137
show faults	138
show indexes	138
show parameters	138
show perf	138
show plans	138
show pools	139
show schemas	139
show snapshots	139
show tables	139
show topology	139
show upgrade-order	140
show users	140
show versions	140
show zones	140
snapshot	140
snapshot create	141
snapshot remove	141
table	141

table size	141
timer	145
topology	145
topology change-repfactor	146
topology change-zone-type	146
topology clone	146
topology create	146
topology delete	147
topology list	147
topology preview	147
topology rebalance	147
topology redistribute	147
topology validate	147
topology view	148
verbose	148
verify	148
verify configuration	148
verify prerequisite	148
verify upgrade	149
Utility commands	149
makebootconfig	149
start	153
stop	154
restart	154
runadmin	154
load	155
ping	156
version	157
generateconfig	157
help	157
B. Initial Capacity Planning	158
Shard Capacity	158
Application Characteristics	159
Replication Factor	159
Average Key Size	159
Average Value Size	159
Read and Write Operation Percentages	159
Hardware Characteristics	160
Shard Storage and Throughput Capacities	160
Shard Storage Capacity	160
Shard I/O Throughput capacity	160
Memory and Network Configuration	161
Machine Physical Memory	161
Sizing Advice	162
Determine JE Cache Size	162
Machine Network Throughput	164
Estimate total Shards and Machines	165
Number of Partitions	166
C. Tuning	167

Turn off the swap	167
Linux Page Cache Tuning	167
OS User Limits	168
File Descriptor Limits	169
Process and Thread Limits	169
Linux Network Configuration Settings	169
Server Socket Backlog	170
Isolating HA Network Traffic	170
Receive Packet Steering	170
Managing the Off-heap Cache	171
Configuring the Off-heap Cache	172
D. Solid State Drives (SSDs)	176
Trim requirements	176
Enabling Trim	176
E. Diagnostics Utility	177
Setting up the tool	177
Packaging information and files	178
Verifying Storage Node configuration	180
Key Distribution Statistics Utility	181
F. Third Party Licenses	183

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.

Chapter 1. Installing Oracle NoSQL Database

This chapter describes the installation process for Oracle NoSQL Database in a multi-host environment. If you are planning a large system for production use, please read [Initial Capacity Planning \(page 158\)](#) to estimate the number of storage nodes on which you will need to install the software. For simple uses when you already know the storage nodes you intend to use, simply follow the instructions below and Oracle NoSQL Database will make the best use of the storage nodes you provide.

Installation Prerequisites

Make sure that you have Java SE 7 or later installed on all of the hosts that you are going to use for the Oracle NoSQL Database installation. The command:

```
java -version
```

can be used to verify this.

Note

Oracle NoSQL Database is compatible with Java SE 7 and later, and has been tested and certified against Oracle Java SE 7. It is recommended that you upgrade to the latest Java releases to take advantage of the latest bug fixes and performance improvements. The release notes included in the Oracle NoSQL Database download specify the exact Java versions that have been used for certification.

Make sure that the `jps` utility is working. Installing the JDK allows the `jps` tools to be available for use by the Storage Node Agent (SNA) in order to optimally manage Oracle NoSQL Database processes. The `jps` tools also allow you to diagnose any issues that may turn up. While Oracle NoSQL Database can continue to operate in the absence of the `jps` tools, it diminishes its ability to manage its processes.

If the JDK and its tools have been correctly installed, the output from `jps` should list at least one Java process (the `jps` process itself). Use the following command to verify this:

```
% jps
16216 Jps
```

Note

You must run the commands listed above as the OS user that will run the Oracle NoSQL Database SNA processes.

Only Linux and Solaris 10 are officially supported platforms for Oracle NoSQL Database. It may be that platforms other than Linux or Solaris 10 could work for your deployment. However, Oracle does not test Oracle NoSQL Database on platforms other than Linux and Solaris 10, and so makes no claims as to the suitability of other platforms for Oracle NoSQL Database deployments.

In addition, it is preferable that virtual machines not be used for any of the Oracle NoSQL Database nodes. This is because the usage of virtual machines makes it difficult to

characterize Oracle NoSQL Database performance. For best results, run the Oracle NoSQL Database nodes natively (that is, without VMs) on Linux or Solaris 10 platforms.

You do not necessarily need root access on each node for the installation process.

Finally, **make sure** that some sort of reliable clock synchronization is running on each of the machines. Generally, a synchronization delta of less than half a second is required. Network Time Protocol (ntp) is sufficient for this purpose.

Installation

The following procedures describe how to install Oracle NoSQL Database:

1. Pick a directory where the Oracle NoSQL Database package files (libraries, Javadoc, scripts, and so forth) should reside. It is easiest if that directory has the same path on all nodes in the installation. You should use different directories for the Oracle NoSQL Database package files (referred to as KVHOME in this document) and the Oracle NoSQL Database data (referred to as KVROOT). Both the KVHOME and KVROOT directories should be local to the node (that is, not on a Network File System).

Note

To make future software upgrades easier, adopt a convention for KVHOME that includes the release number. Always use a KVHOME location such as `/var/kv/kv-M.N.O`, where M.N.O are the release.major.minor numbers. This can be easily achieved by simply unzip/untaring the distribution into a common directory (`/var/kv` in this example).

2. Extract the contents of the Oracle NoSQL Database package (`kv-M.N.O.zip` or `kv-M.N.O.tar.gz`) to create the KVHOME directory (i.e. KVHOME is the `kv-M.N.O/` directory created by extracting the package). If KVHOME resides on a network shared directory (not recommended) then you only need to unpack it on one machine. If KVHOME is local to each machine, then you should unpack the package on each node.
3. Verify the installation by issuing the following command on one of the nodes:

```
java -Xmx256m -Xms256m -jar KVHOME/lib/kvclient.jar
```

You should see some output that looks like this:

```
11gR2.M.N.O (....)
```

where M.N.O is the package version number.

Note

Oracle NoSQL Database is a distributed system and the runtime needs to be installed on every node in the cluster. While the entire contents of the Oracle NoSQL Database package do not need to be installed on every node, the contents of the lib and doc directories must be present. How this distribution is done is beyond the scope of this manual.

Installation Configuration

Before you configure Oracle NoSQL Database, you should determine the following parameters for each Storage Node in the store:

- *root*

Where the KVROOT directory should reside. There should be enough disk space on each node to hold the data to be stored in your Oracle NoSQL Database 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 best if the KVROOT is the same local directory path on each node (but not a shared or NFS mounted directory). The examples in this book assume that the KVROOT directory already exists.

- *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*. The examples in this book use port 5000.

- *admin*

The port on which the Oracle NoSQL Database web-based Admin Console is contacted. This port only needs to be free on the node which runs the administration process. The examples in this book use port 5001. You can also change this port number by modifying the value of the `adminHttpPort` parameter. For more information, see [Changing Parameters \(page 73\)](#).

Note

Setting the value of `-admin` or the `adminHttpPort` parameter to 0 disables the web interface.

The administration process can be replicated across multiple nodes, and so the port needs to be available on all the machines where it runs. In this way, if the administration process fails on one machine, it can continue to use the http web service on a different machine. You can actually use a different port for each node that runs an administration process, but for the sake of simplicity we recommend you be consistent.

- *runadmin*

Forces an Admin to be included in the boot configuration even if `-admin 0` was specified.

If `-admin 0` is specified in conjunction with `-runadmin`, the admin web interface will not be started.

If `-runadmin` is not specified, the admin is included in the boot configuration only if an admin port value greater than 0 is specified.

- *harange*

A range of free ports which the Replication Nodes use to communicate among themselves. These ports must be sequential and there must be at least as many as there are Replication Nodes running on each Storage Node in your store. The Storage Node Agent manages this allotment of ports, reserves one for an Admin service, and uses the rest to allocate one per Replication Node. The port range is specified as "startPort,endPort". "5010,5020" is used by the examples in this book.

- *servicerange*

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." The value varies with the capacity of the Storage Node. For more information about the servicePortRange, see [Storage Node Parameters \(page 76\)](#).

- *store-security*

Specifies if security will be used or not. In the examples in this book, no security is used.

If `-store-security none` is specified, no security will be used.

If `-store-security configure` is specified, security will be used and the security configuration utility will be invoked as part of the `makebootconfig` process.

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.

For more information on configuring Oracle NoSQL Database securely, see the Oracle NoSQL Database Security Guide.

- *capacity*

The total number of Replication Nodes a Storage Node can support. Capacity is an optional parameter. Capacity can be set to values greater than 1 when the Storage Node has sufficient disk, cpu, memory and network bandwidth to support multiple Replication Nodes.

Please keep the following configuration considerations in mind for nodes with capacity greater than one:

1. It is best if the Storage Node is configured with a capacity equal to the number of disks available on the machine. Such a configuration permits the placement of each Replication Node on its own disk and ensures that the Replication Nodes on the Storage Node are not competing for I/O resources. The location of this directory on its disk can be specified via the *storagedir* parameter.

For example:

```
> java -jar KVHOME/lib/kvstore.jar makebootconfig \  
-root /opt/ondb/var/kvroot \  

```

```
-port 5000 \  
-admin 5001 \  
-host node10  
-harange 5010,5025 \  
-store-security none \  
-capacity 3 \  
-storagedir /disk1/ondb/data \  
-storagedir /disk2/ondb/data \  
-storagedir /disk3/ondb/data \
```

where capacity=3 equals the number of disks (disk1, disk2, disk3) located on the same Storage Node (node10).

2. Increase the *harange* parameter to account for the additional ports required by the Replication Nodes.
3. Increase the *servicerange* parameter to account for the additional ports required by the Replication Nodes.

The value defaults to the number of *storagedir* parameters if they were specified. Otherwise the value defaults to "1". "1" is used as the capacity by the examples in this book.

- *storagedir*

A path to the directory that will contain the environment associated with a Replication Node. For capacity values greater than one, multiple *storagedir* parameters must be specified, one for each Replication Node that will be hosted on the Storage Node. It is best if each directory path resolves to a separate disk. This is typically accomplished via suitable entries in */etc/fstab* that attach the file system on a disk to an appropriate location in the overall directory hierarchy. Placing each environment on a distinct disk ensures that the Replication Nodes are not competing for I/O resources. It also isolates the impact of a disk failure to a single environment.

In the absence of explicit directory arguments the environments are located under the *KVROOT* directory.

- *num_cpus*

The total number of processors on the machine available to the Replication Nodes. It is used to coordinate the use of processors across Replication Nodes. If the value is 0, the system will attempt to query the Storage Node to determine the number of processors on the machine. This value defaults to "0". "0" numCPUs is used by the examples in this book.

- *memory_mb*

The total number of megabytes of memory that is available in the machine. It is used to guide the specification of the Replication Node's heap and cache sizes. This calculation becomes more critical if a Storage Node hosts multiple Replication Nodes, and must allocate memory between these processes. If the value is 0, the store will attempt to determine the amount of memory on the machine, but that value is only available when the JVM used is the Oracle Hotspot JVM. The default value is "0". "0" is used by the examples in this book.

- *force*

Optionally specified to force generating the boot configuration files even if boot config verification finds any invalid parameters.

Once you have determined this information, configure the installation:

1. Create the initial "boot config" configuration file using the `makebootconfig` utility. You should do this on each Oracle NoSQL Database node. You only need to specify the `-admin` option (the Admin Console port) on the node which hosts the initial Oracle NoSQL Database administration processes. (At a later point in this installation procedure, you deploy additional administration processes.)

Note

Boot config verification is integrated, which checks and validates all the parameters of `makebootconfig` against the environment before generating the boot configuration files. Optionally use the `-force` flag to bypass verification and generate the boot configuration files.

To create the "boot config" file, issue the following commands:

```
> mkdir -p KVR00T      (if it does not already exist)
> java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar makebootconfig -root KVR00T \
                                           -port 5000 \
                                           -admin 5001 \
                                           -host <hostname> \
                                           -hrange 5010,5020 \
                                           -store-security none \
                                           -capacity 1 \
                                           -num_cpus 0 \
                                           -memory_mb 0
```

2. Start the Oracle NoSQL Database Storage Node Agent (SNA) on each of the Oracle NoSQL Database nodes. The SNA manages the Oracle NoSQL Database processes on each node. It also owns and manages the registry port, which is the main way to communicate with Oracle NoSQL Database processes on that node. You can use the `start` utility for this:

```
nohup java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar start -root KVR00T&
```

Note

If the Replication Node or Admin Service crashes, the SNA ensures that the processes restart.

3. Verify that the Oracle NoSQL Database processes are running using the `jps -m` command:

```
> jps -m
29400 ManagedService -root /tmp -class Admin -service
```



```
BootstrapAdmin.13250 -config config.xml
29394 StorageNodeAgentImpl -root /tmp -config config.xml
```

4. Ensure that the Oracle NoSQL Database client library can contact the Oracle NoSQL Database Storage Node Agent (SNA) by using the ping command:

```
java -jar kvstore.tmp/kvstore.jar ping -host node01 -port 5000
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 message is not an error, but instead it is telling you that only the SN process is running on the local host. Once Oracle NoSQL Database is fully configured, the ping option has more to say. If the SNA cannot be contacted, you see this instead:

```
Unable to connect to the storage node agent at host node01,
port 5000, which may not be running; nested exception is:

java.rmi.ConnectException: Connection refused to host: node01;
nested exception is:
java.net.ConnectException: Connection refused
Can't find store topology:
Could not contact any RepNode at: [node01:5000]
```

If the Storage Nodes do not start up, you can look through the adminboot and snaboot logs in the KVROOT directory in order to identify the problem. You can also use the -host option to check an SNA on a remote host:

```
> java -Xmx256m -Xms256m
-jar KVHOME/lib/kvstore.jar ping -port 5000 -host node02
SNA at hostname: node02, registry port: 5000 is not registered. No
further information is available
Can't find store topology: Could not contact any RepNode at: [node02:5000]
```

Assuming the Storage Nodes have all started successfully, you can configure the KVStore. This is described in the next chapter.

Note

For best results, you should configure your nodes such that the SNA starts automatically when your node boots up. How this is done is a function of how your operating system is designed, and so is beyond the scope of this manual. See your operating system documentation for information on automatic application launch at bootup.

Configuring the Firewall

To make sure your firewall works with Oracle NoSQL Database, you should set the ports specified by the `servicerange` parameter of the `makebootconfig` command. This parameter is used to constrain a store to a limited set of ports, usually for security or data center policy reasons. By default the services use anonymous ports.

The format of the value string is "startPort,endPort." The value varies with the capacity of the Storage Node.

For more information about the servicePortRange and determining its sizing, see [Storage Node Parameters \(page 76\)](#).

Chapter 2. Plans

This chapter describes plans, which are useful to perform the configuration of your store. If you are installing a store for the first time, you can skip ahead to the next chapter [Configuring the KVStore \(page 12\)](#).

You configure Oracle NoSQL Database with administrative commands called *plans*. A plan is made up of multiple operations. Plans may modify state managed by the Admin service, and may issue requests to kvstore components such as Storage Nodes and Replication Nodes. Some plans are simple state-changing operations, while others may be long-running operations that affect every node in the store over time.

For example, you use a plan to create a Zone or a Storage Node or to reconfigure the parameters on a Replication Node.

Using Plans

You create and execute plans using the `plan` command in the administrative command line interface. By default, the command line prompt will return immediately, and the plan will execute asynchronously, in the background. You can check the progress of the plan using the `show plan id` command.

If you use the optional `-wait` flag for the `plan` command, the plan will run synchronously, and the command line prompt will only return when the plan has completed. The `plan wait` command can be used for the same purpose, and also lets you specify a time period. The `-wait` flag and the `plan wait` command are particularly useful when issuing plans from scripts, because scripts often expect that each command is finished before the next one is issued.

You can also create, but defer execution of the plan by using the optional `-noexecute` flag. If `-noexecute` is specified, the plan can be run later using the `plan execute -id <id>` command.

Feedback While a Plan is Running

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. Note that the `-verbose` optional plan flag can be used to get more detail.
- The Admin Console's Topology tab refreshes as Oracle NoSQL Database services are created and brought online.
- You can issue the `verify` command using the Topology tab or the CLI as plans are executing. The `verify` plan provides service status information as services come up.

Note

The Topology tab and `verify` command are really only of interest for topology-related plans. For example, if the user is modifying parameters, the changes may not be visible via the topology tab or `verify` command.

- You can follow the store-wide log using the Admin Console's Logs tab, or by using the CLI's `logtail` command.

Plan States

Plans can be in these states:

1. APPROVED

The plan has been created, but is not yet running.

2. RUNNING

The plan is currently executing.

3. SUCCEEDED

The plan has completed successfully.

4. INTERRUPTED

A RUNNING plan has been manually interrupted, using the `interrupt` command in the CLI.

5. INTERRUPT REQUESTED

A plan has been manually interrupted, but is still processing the interrupt request. A plan may have to cleanup or reverse steps taken during plan execution to be sure that the store remains in a consistent state.

6. ERROR

A RUNNING plan has encountered a problem, and has ended without successfully completing.

7. CANCELED

An INTERRUPTED or ERROR plan has been terminated using the CLI. To cancel a plan using the CLI, use the `cancel` command.

Plans in INTERRUPTED, INTERRUPT REQUESTED or ERROR state can be retried using the `plan execute` command. Retrying may be an appropriate approach when the underlying problem was transient or has been rectified. Plans that are retried simply re-execute the same steps. Each step is idempotent, and can be safely repeated.

Note that Storage Nodes and Replication Nodes may encounter errors which are detected by the Admin Console and are displayed in an error dialog before the plan has processed the information. Because of that, the user may learn of the error while the Admin service still considers the plan to be RUNNING and active. The plan eventually sees the error and transitions to an ERROR state.

Reviewing Plans

You can find out what state a plan is in using the `show plans` command in the CLI. Use the `show plan -id <plan number>` command to see more details on that plan. Alternatively, you can see the state of your plans in the Plan History section in the Admin Console. Click on the plan number in order to see more details on that plan.

You can review the execution history of a plan by using the CLI `show plan` command. (How to use the CLI is described in detail in [Configuring the KVStore \(page 12\)](#).)

This example shows the output of the `show plan` command. The plan name, attempt number, started and ended date, status, and the steps, or tasks that make up the plan are displayed. In this case, the plan was executed once. The plan completed successfully.

```
kv-> show plan
1 Deploy KVLite          SUCCEEDED
2 Deploy Storage Node    SUCCEEDED
3 Deploy Admin Service   SUCCEEDED
4 Deploy KVStore         SUCCEEDED
kv-> show plan -id 3
Plan Deploy Admin Service (3)
Owner: null
State:          SUCCEEDED
Attempt number: 1
Started:        2012-11-22 22:05:31 UTC
Ended:          2012-11-22 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 owner of a plan can see and operate it. If a plan is created in an earlier version of Oracle NoSQL Database or in a non secure store, the owner is null.

Note

The SYSOPER privilege allows a role to perform cancel, execute, interrupt, and wait on any plan.

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

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

Chapter 3. Configuring the KVStore

Once you have installed Oracle NoSQL Database on each of the nodes that you could use in your store (see [Installing Oracle NoSQL Database \(page 1\)](#)), you must configure the store. To do this, you use the command line administration interface. In this chapter, we describe the command line tool.

To configure your store, you create and then execute *plans*. Plans describe a series of operations that Oracle NoSQL Database should perform for you. You do not need to know what those internal operations are in detail. Instead, you just need to know how to use and execute the plans.

Configuration Overview

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

1. [Configure and Start a Set of Storage Nodes \(page 13\)](#)
2. [Name your KVStore \(page 13\)](#)
3. [Create a Zone \(page 14\)](#)
4. [Create an Administration Process on a Specific Host \(page 15\)](#)
5. [Create a Storage Node Pool \(page 16\)](#)
6. [Create the Remainder of your Storage Nodes \(page 17\)](#)
7. [Create and Deploy Replication Nodes \(page 17\)](#)

You perform all of these activities using the Oracle NoSQL Database command line interface (CLI). The remainder of this chapter shows you how to perform these activities. Examples are provided that show you which commands to use, and how. For a complete listing of all the commands available to you in the CLI, see [CLI Commands and Subcommands \(page 107\)](#).

Start the Administration CLI

To perform store configuration, you use the `runadmin` utility, which provides a command line interface (CLI). The `runadmin` utility can be used for a number of purposes. In this chapter, we want to use it to administer the nodes in our store, so we have to tell `runadmin` what node and registry port it can use to connect to the store.

In this book, we have been using 5000 as the registry port. For this example, we use the string `node01` to represent the network name of the node to which `runadmin` connects.

Note

You should think about the name of the node to which the `runadmin` connects. The node used for initial configuration of the store, during store creation, cannot be changed.

The most important thing about this node is that it must have the Storage Node Agent running on it. All your nodes should have an SNA running on them at this point. If not, you need to go follow the instructions in [Installing Oracle NoSQL Database \(page 1\)](#) before proceeding with the steps provided in this chapter.

Beyond that, be aware that if this is the very first node you have ever connected to the store using the CLI, then it becomes the node on which the master copy of the administration database resides. If you happen to care about which node serves that function, then make sure you use that node at this time.

To start runadmin for administration purposes:

```
> java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar runadmin \  
-port 5000 -host node01
```

Note that once you have started the CLI, you can use its help command in order to discover all the administration commands available to you.

Also note that the configuration steps described in this chapter can be collected into a script file, and then that file can be passed to the utility using its -script command line option. See [Using a Script to Configure the Store \(page 25\)](#) for more information.

The plan Commands

Some of the steps described in this chapter make heavy use of the CLI's plan command. This command identifies a configuration action that you want to perform on the store. You can either run that action immediately or you can create a series of plans with the -noexecute flag and then execute them later by using the plan execute command.

You can list all available plans by using the plan command without arguments.

For a high-level description of plans, see [Plans \(page 9\)](#).

Configure and Start a Set of Storage Nodes

You should already have configured and started a set of Storage Nodes to host the KVStore cluster. If not, you need to follow the instructions in [Installing Oracle NoSQL Database \(page 1\)](#) before proceeding with this step.

Name your KVStore

When you start the command line interface, the kv-> prompt appears. Once you see this, you can name your KVStore by using the configure -name command. The only information this command needs is the name of the KVStore that you want to configure.

Note that the name of your store is essentially 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 '.'.

For example:

```
kv-> configure -name mystore
Store configured: mystore
```

Note

The store name must be unique across all instances of NoSQL Database that will be monitored by a given instance of Enterprise Manager. For more information, see [Store Targets \(page 100\)](#).

Create a Zone

Once you have started the command line interface and configured a store name, you can create at least one Zone. It is possible, and even desirable, to create more than one Zone. Multiple Zones are used to improve the availability of your store. These instructions describe the installation using a single Zone. For a complete walk-through of a store deployment with multiple Zones, see [Configuring with Multiple Zones \(page 18\)](#).

Note

Once you have added Storage Nodes to a Zone, you cannot remove the Zone from your store.

When you execute the `plan deploy-zone` command, the CLI returns the plan number and whatever additional information it has about plan status. This command takes the following arguments:

- *zone name*

A string to identify the name of the Zone.

- *replication factor*

A number specifying the *Zone Replication Factor*.

For more information on Primary and Secondary Replication Factors, see [Configuring with Multiple Zones \(page 18\)](#).

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

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

```
kv-> show plans
```


1 Deploy Zone (1)	SUCCEEDED
-------------------	-----------

Create an Administration Process on a Specific Host

Every KVStore has an administration database. You must deploy the Storage Node to which the command line interface is currently connecting to, in this case, "node01", and then deploy an Administration process on that same node, in order to proceed to configure this database. Use the `deploy-sn` and `deploy-admin` commands to complete this step.

The `deploy-admin` command will create an Administration process which is the same type as the zone the Storage Node is in. If the zone is primary, the admin will be a primary admin. If the zone is secondary, the admin will be a secondary admin. Secondary admins are needed to support failover, where a failed admin will be converted to an offline secondary in order to reestablish quorum using existing admins, and a secondary admin converted to a primary to take over for failed primaries. For more information on quorum, see the Oracle NoSQL Database Concepts manual.

To support failover, you must make sure that any zones that will be used to continue the operation of the store after a failure contain at least one admin node.

Note

A deployed admin must be the same type (PRIMARY/SECONDARY) as the zone they are in. Also, the number of admins deployed in a zone should be equal to the zone's Replication Factor.

Note that `deploy-sn` requires you to provide a Zone ID. You can get this ID by using the `show topology` command:

```
kv-> show topology
store=mystore numPartitions=0 sequence=1
zn: id=zn1 name=Boston repFactor=3 type=PRIMARY
```

The Zone ID is "zn1" in the above output.

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

```
kv-> plan deploy-sn -zn zn1 -host node01 -port 5000 -wait
Executed plan 2, waiting for completion...
Plan 2 ended successfully
```

Having done that, create the administration process on the node that you just deployed. You do this using the `deploy-admin` command. This command requires the Storage Node ID (which you can obtain using the `show topology` command), the administration port number and an optional plan name. You defined the administration port number during the installation process. This book is using 5001 as an example.

```
kv-> plan deploy-admin -sn sn1 -port 5001 -wait
Executed plan 3, waiting for completion...
Plan 3 ended successfully
```

Note

At this point you have a single administration process deployed in your store. This is enough to proceed with store configuration. However, to increase your store's reliability, you should deploy multiple administration processes, each running on a different storage node. In this way, you are able to continue to administer your store even if one Storage Node goes down, taking an administration process with it. It also means that you can continue to monitor your store, even if you lose a node running an administration process.

Oracle strongly recommends that you deploy three administration processes for a production store. The additional administration processes do not consume many resources.

Before you can deploy any more administration processes, you must first deploy the rest of your Storage Nodes. This is described in the following sections.

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 store. A Storage Node pool is used for resource distribution when creating or modifying a 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.

Note

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.

Remember that we already have a Storage Node created. We did that when we created the Administration process. Therefore, after we add the pool, we can immediately join that first SN 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
kv-> show topology
store=mystore numPartitions=0 sequence=2
```

```
zn: id=zn1 name=Boston repFactor=3 type=PRIMARY
sn=[sn1] zn:[id=zn1 name=Boston] node1:5000 capacity=1 RUNNING
kv-> pool join -name BostonPool -sn sn1
Added Storage Node(s) [sn1] to pool BostonPool
```

Create the Remainder of your Storage Nodes

Having created your Storage Node Pool, you can create the remainder of your Storage Nodes. Storage Nodes host the various Oracle NoSQL Database processes for each of the nodes in the store. Consequently, you must do this for each node that you use in your store. Use the `deploy-sn` command in the same way as you did in [Create an Administration Process on a Specific Host \(page 15\)](#). As you deploy each Storage Node, join it to your Storage Node Pool as described in the previous section.

Hint: Storage Node IDs increase by one as you add each Storage Node. Therefore, you do not have to keep looking up the IDs with `show topology`. If the Storage Node that you created last had an ID of 10, then the next Storage Node that you create has an ID of 11.

```
kv-> plan deploy-sn -zn zn1 -host node02 -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 node03 -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->
....
```

Continue this process until you have created Storage Nodes on every node in your store.

Note

Having deployed all your Storage Nodes, you can now deploy additional administration processes using the `deploy-admin` plan. See [Create an Administration Process on a Specific Host \(page 15\)](#) for details.

Create and Deploy Replication Nodes

The final step in your configuration process is to create Replication Nodes on every node in your store. You do this using the `topology create` and `plan deploy-topology` commands in its place. 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 storage nodes specified by 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 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 \(page 166\)](#).

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

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

```
kv-> topology create -name topo -pool BostonPool -partitions 300
Created: topo
kv-> plan deploy-topology -name topo -wait
Executed plan 6, waiting for completion...
Plan 6 ended successfully
```

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

```
kv-> show plans
1 Deploy Zone (1)          SUCCEEDED
2 Deploy Storage Node (2)  SUCCEEDED
3 Deploy Admin Service (3) SUCCEEDED
4 Deploy Storage Node (4)  SUCCEEDED
5 Deploy Storage Node (5)  SUCCEEDED
6 Deploy-RepNodes          SUCCEEDED
```

Having done that, you can exit the command line interface.

```
kv-> exit
```

Configuring with Multiple Zones

Optimal use of available physical facilities is achieved by deploying your store across multiple Zones. This provides fault isolation and availability for your data if a single zone fails. Each Zone has a copy of your complete store, including a copy of all the shards. With this configuration, reads are always possible, so 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 so long as quorum can be maintained. For more information on quorum, see the Oracle NoSQL Database Concepts manual.

You can specify a different replication factor to each Zone. A replication factor can then be 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 \(page 159\)](#).

- *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*

Represents for all zones in the store, the total number of replicas across the entire store.

Zones located nearby have the benefit of avoiding bottlenecks due to throughput limitations, as well as reducing latency during elections and commits.

Note

Zones come in two types. *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.

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, write performance may be reduced if primary zones are connected through a limited throughput and/or 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.

1. For a new store, create the initial "boot config" configuration files using the `makebootconfig` utility:

```
java -jar kv/lib/kvstore.jar makebootconfig \  
-root Data/virtualroot/datacenter1/KVR00T \  
-host localhost \  
-port 5000 \  
-admin 5001 \  
-harange 5010,5020 \  
-capacity 1 \  
-store-security none \  

```

```
java -jar kv/lib/kvstore.jar makebootconfig \  
-root Data/virtualroot/datacenter2/KVR00T \  
-host localhost \  
-port 6000 \  
-admin 6001 \  
-harange 6010,6020 \  
-capacity 1 \  
-store-security none \  

```

```
java -jar kv/lib/kvstore.jar makebootconfig \  
-root Data/virtualroot/datacenter3/KVR00T \  
-host localhost \  
-port 7000 \  
-admin 7001 \  
-harange 7010,7020 \  
-capacity 1 \  
-store-security none \  

```

```
java -jar kv/lib/kvstore.jar makebootconfig \  
-root Data/virtualroot/datacenter4/KVR00T \  
-host localhost \  
-port 8000 \  
-admin 8001 \  
-harange 8010,8020 \  
-capacity 1 \  
-store-security none \  

```

```
java -jar kv/lib/kvstore.jar makebootconfig \  
-root Data/virtualroot/datacenter5/KVR00T \  
-host localhost \  
-port 9000 \  
-admin 9001 \  
-harange 9010,9020 \  
-capacity 1 \  

```

```
-store-security none \  
  
java -jar kv/lib/kvstore.jar makebootconfig \  
-root Data/virtualroot/datacenter6/KVROOT \  
-host localhost \  
-port 10000 \  
-admin 10001 \  
-harange 10010,10020 \  
-capacity 1 \  
-store-security none \
```

2. Using each of the configuration files, start all of the Storage Node Agents:

```
> nohup java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar \  
start -root Data/virtualroot/datacenter1/KVROOT &  
> [1] 12019  
  
> nohup java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar \  
start -root -root Data/virtualroot/datacenter2/KVROOT &  
> [2] 12020  
  
> nohup java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar \  
start -root -root Data/virtualroot/datacenter3/KVROOT &  
> [3] 12021  
  
> nohup java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar \  
start -root -root Data/virtualroot/datacenter4/KVROOT &  
> [4] 12022  
  
> nohup java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar \  
start -root -root Data/virtualroot/datacenter5/KVROOT &  
> [5] 12023  
  
> nohup java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar \  
start -root -root Data/virtualroot/datacenter6/KVROOT &  
> [6] 12024
```

3. Start the CLI:

```
> java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar runadmin -host \  
localhost -port 5010  
kv->
```

4. Name your store:

```
kv-> configure -name MetroArea
Store configured: MetroArea
```

5. Deploy the first Storage Node with administration process in the Manhattan Zone:

```
kv-> plan deploy-zone -name Manhattan -rf 1 -wait
Executed plan 1, waiting for completion...
Plan 1 ended successfully
kv-> plan deploy-sn -zn 1 -host localhost -port 5000 -wait
Executed plan 2, waiting for completion...
Plan 2 ended successfully
kv-> plan deploy-admin -sn sn1 -port 5001 -wait
Executed plan 3, waiting for completion...
Plan 3 ended successfully
kv-> pool create -name SNs
kv-> pool join -name SNs -sn sn1
Added Storage Node(s) [sn1] to pool SNs
```

6. Deploy a second Storage Node in Manhattan Zone:

```
kv-> plan deploy-sn -znname Manhattan -host localhost \
-port 6000 -wait
kv-> Executed plan 4, waiting for completion...
Plan 4 ended successfully
kv-> pool join -name SNs -sn sn2
Added Storage Node(s) [sn2] to pool SNs
```

7. Deploy the first Storage Node with administration process in the Jersey City Zone:

```
kv-> plan deploy-zone -name JerseyCity -rf 1 -wait
Executed plan 5, waiting for completion...
Plan 5 ended successfully
kv-> plan deploy-sn -znname JerseyCity -host localhost \
-port 7000 -wait
Executed plan 6, waiting for completion...
Plan 6 ended successfully
kv-> plan deploy-admin -sn sn3 -port 7001 -wait
Executed plan 7, waiting for completion...
Plan 7 ended successfully
kv-> pool join -name SNs -sn sn3
Added Storage Node(s) [sn3] to pool SNs
```

8. Deploy a second Storage Node in Jersey City Zone:

```
kv-> plan deploy-sn -znname JerseyCity -host localhost \
-port 8000 -wait
kv-> Executed plan 8, waiting for completion...
Plan 8 ended successfully
kv-> pool join -name SNs -sn sn4
Added Storage Node(s) [sn4] to pool SNs
```

9. Deploy the first Storage Node with administration process in the Queens Zone:


```
kv-> plan deploy-zone -name Queens -rf 1 -wait
Executed plan 9, waiting for completion...
Plan 9 ended successfully
kv-> plan deploy-sn -znname Queens -host localhost -port 9000 -wait
Executed plan 10, waiting for completion...
Plan 10 ended successfully
kv-> plan deploy-admin -sn sn5 -port 9001 -wait
Executed plan 11, waiting for completion...
Plan 11 ended successfully
kv-> pool join -name SNs -sn sn5
Added Storage Node(s) [sn5] to pool SNs
```

10. Deploy a second Storage Node in Queens Zone:

```
kv-> plan deploy-sn -znname Queens -host localhost \
-port 10000 -wait
kv-> Executed plan 12, waiting for completion...
Plan 12 ended successfully
kv-> pool join -name SNs -sn sn6
Added Storage Node(s) [sn6] to pool SNs
```

11. Create and deploy a topology:

```
kv-> topology create -name Topo1 -pool SNs -partitions 100
Created: Topo1
kv-> plan deploy-topology -name Topo1 -wait
kv-> Executed plan 13, waiting for completion...
Plan 13 ended successfully
```

12. Check service status with the show topology command:

```
kv-> show topology
store=MetroArea numPartitions=100 sequence=117
zn: id=zn1 name=Manhattan repFactor=1 type=PRIMARY
zn: id=zn2 name=JerseyCity repFactor=1 type=PRIMARY
zn: id=zn3 name=Queens repFactor=1 type=PRIMARY

sn=[sn1] zn=[id=zn1 name=Manhattan] node01:5000 capacity=1 RUNNING
[rg1-rn2] RUNNING
No performance info available
sn=[sn2] zn=[id=zn1 name=Manhattan] node02:6000 capacity=1 RUNNING
[rg2-rn2] RUNNING
No performance info available
sn=[sn3] zn=[id=zn2 name=JerseyCity] node03:7000 capacity=1 RUNNING
[rg1-rn3] RUNNING
No performance info available
sn=[sn4] zn=[id=zn2 name=JerseyCity] node04:8000 capacity=1 RUNNING
[rg2-rn3] RUNNING
No performance info available
sn=[sn5] zn=[id=zn3 name=Queens] node05:9000 capacity=1 RUNNING
[rg1-rn1] RUNNING
```

```

    No performance info available
sn=[sn6] zn=[id=zn3 name=Queens] node06:10000 capacity=1 RUNNING
  [rg2-rn1] RUNNING
    No performance info available

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

```

13. Verify that each shard has a replica in every zone:

```

kv-> verify configuration
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2015-06-16 00:01:05 UTC   Version: 12.1.3.4.0
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]
    RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY]
    RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=Queens id=zn3 type=PRIMARY]
    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]   Status: RUNNING
    Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC   Build id: db9ec397225c
Verify:      Admin [admin1]           Status: RUNNING,MASTER
Verify:      Rep Node [rg1-rn2]       Status: RUNNING,REPLICA
    sequenceNumber:127 haPort:5011 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on node02:6000
    Zone: [name=Manhattan id=zn1 type=PRIMARY]   Status: RUNNING
    Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC   Build id: db9ec397225c
Verify:      Rep Node [rg2-rn2]       Status: RUNNING,REPLICA
    sequenceNumber:127 haPort:6010 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:7000
    Zone: [name=JerseyCity id=zn2 type=PRIMARY]   Status: RUNNING
    Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC   Build id: db9ec397225c

```

```

Verify:      Admin [admin2]          Status: RUNNING,REPLICA
Verify:      Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
            sequenceNumber:127 haPort:7011 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:8000
            Zone: [name=JerseyCity id=zn2 type=PRIMARY]      Status: RUNNING
            Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Rep Node [rg2-rn3]      Status: RUNNING,REPLICA
            sequenceNumber:127 haPort:8010 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:9000
            Zone: [name=Queens id=zn3 type=PRIMARY]      Status: RUNNING
            Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Admin [admin3]          Status: RUNNING,REPLICA
Verify:      Rep Node [rg1-rn1]      Status: RUNNING,MASTER
            sequenceNumber:127 haPort:9011
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:10000
            Zone: [name=Queens id=zn3 type=PRIMARY]      Status: RUNNING
            Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Rep Node [rg2-rn1]      Status: RUNNING,MASTER
            sequenceNumber:127 haPort:10010

Verification complete, no violations.

```

In the previous example there are three zones (zn1 = Manhattan, zn2 = JerseyCity, zn3=Queens) with six Replication Nodes (two masters and four replicas) in this cluster. 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.

Using a Script to Configure the Store

Up to this point, we have shown how to configure a 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 them 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:

```

> java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar runadmin -port 5000 -host node01 \
load -file scrpt.txt

```

Using directly the load -file command:

```
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, scrpt.txt, would then contain content like this:

```

### Begin Script ###
configure -name mystore
plan deploy-zone -name "Boston" -rf 3 -wait
plan deploy-sn -zn zn1 -host node01 -port 5000 -wait
plan deploy-admin -sn sn1 -port 5001 -wait
pool create -name BostonPool
pool join -name BostonPool -sn sn1
plan deploy-sn -zn zn1 -host node02 -port 6000 -wait
pool join -name BostonPool -sn sn2
plan deploy-sn -zn zn1 -host node03 -port 7000 -wait
pool join -name BostonPool -sn sn3
topology create -name topo -pool BostonPool -partitions 300
plan deploy-topology -name topo -wait
exit
### End Script ###

```

Smoke Testing the System

There are several things you can do to ensure that your KVStore is up and fully functional.

1. Run the ping command.

```

> java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar ping -port 5000 -host node01
Pinging components of store mystore based upon topology sequence #316
300 partitions and 3 storage nodes
Time: 2015-06-11 02:54:35 UTC   Version: 12.1.3.4.0
Shard Status: healthy:3 writable-degraded:0 read-only:0 offline:0
Admin Status: healthy
Zone [name=Boston id=zn1 type=PRIMARY]
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: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
  Admin [admin1]                               Status: RUNNING,MASTER
  Rep Node [rg1-rn1]                           Status: RUNNING,REPLICA
    sequenceNumber:231 haPort:5011 delayMillis:1 catchupTimeSecs:0
  Rep Node [rg2-rn1]                           Status: RUNNING,REPLICA
    sequenceNumber:231 haPort:5012 delayMillis:1 catchupTimeSecs:0
  Rep Node [rg3-rn1]                           Status: RUNNING,MASTER
    sequenceNumber:227 haPort:5013
Storage Node [sn2] on node02:6000
  Zone: [name=Boston id=zn1 type=PRIMARY]      Status: RUNNING
  Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
  Rep Node [rg1-rn2]                           Status: RUNNING,MASTER
    sequenceNumber:231 haPort:6010
  Rep Node [rg2-rn2]                           Status: RUNNING,REPLICA
    sequenceNumber:231 haPort:6011 delayMillis:1 catchupTimeSecs:0
  Rep Node [rg3-rn2]                           Status: RUNNING,REPLICA
    sequenceNumber:227 haPort:6012 delayMillis:1 catchupTimeSecs:0

```

```
Storage Node [sn3] on node03:7000
Zone: [name=Boston id=zn1 type=PRIMARY]    Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
sequenceNumber:231 haPort:7010 delayMillis:1 catchupTimeSecs:0
Rep Node [rg2-rn3]      Status: RUNNING,MASTER
sequenceNumber:231 haPort:7011
Rep Node [rg3-rn3]      Status: RUNNING,REPLICA
sequenceNumber:227 haPort:7012 delayMillis:1 catchupTimeSecs:0
```

2. Run the simple "hello world" example. Go to the KVHOME directory and compile the example:

```
javac -cp lib/kvclient.jar:examples examples/hello/*.java
```

Then run the example (from any directory):

```
java -Xmx256m -Xms256m \
-cp KVHOME/lib/kvclient.jar:KVHOME/examples \
hello.HelloBigDataWorld \
-host <hostname> -port <hostport> -store <kvstore name>
```

This should write the following line to stdout:

```
Hello Big Data World!
```

3. Look through the Javadoc. You can access it from the documentation index page, which can be found at KVHOME/doc/index.html.

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

1. Stop the node using:

```
java -Xmx256m -Xms256m \
-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 \(page 3\)](#).

Troubleshooting

Typical errors when bringing up a 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. In that case be sure to remove all partial store data and configuration and kill any remnant processes. Processes associated with a store as reported by "jps -m" are one of these:

```
StorageNodeAgentImpl
ManagedService
```

If you kill the StorageNodeAgentImpl it should also kill its managed processes.

You can use the monitoring tab in the Admin Console to look at various log files.

There are detailed log files available in `KVR00T/storename/log` as well as logs of the bootstrap process in `KVR00T/*.log`. The bootstrap logs are most useful in diagnosing initial startup problems. The logs in `storename/log` appear once the store has been configured. The logs on the host chosen for the admin process are the most detailed and include a store-wide consolidated log file: `KVR00T/storename/log/storename_*.log`

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

```
2012-10-25 14:28:26.982 UTC INFO [admin1] Initializing Admin for store:
kvstore
```

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 Admin's Topology tab, in the CLI's `show events` command, and when you use the `ping` command.

In addition to log files, these 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 cluster. In addition to contacting the 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 were the case then it might explain why a Replication Node cannot come up. `Verify configuration` also checks on Admins.

Additionally, in order to catch configuration errors early, you can use the diagnostics tool when troubleshooting your KVStore. Also, you can use this tool to package important information and files to be able to send them to Oracle Support, for example. For more information, see [Diagnostics Utility \(page 177\)](#)

Where to Find Error Information

As your 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, or in the Admin's Plan History tab.

Other problems may occur asynchronously. You can learn about unexpected failures, service downtime, and performance issues through the Admin's critical events display in the Logs tab, or 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 Admin's Logs tab, by using the CLI's `logtail` command, or by directly viewing the `<storename>_N.log` file in the `<KVHOME>/<storename>/log` directory. It is also possible to download the store-wide log file using the Admin's Logs tab.

Service States

Oracle NoSQL Database uses three different types of services, all of which should be running correctly in order for your store to be in a healthy state. The three service types are the Admin, Storage Nodes, and Replication 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 Topology tab in the Admin Console
- The `show topology` command in the Administration CLI.
- Using the `ping` command.

The status values can be one of the following:

- **STARTING**

The service is coming up.

- **RUNNING**

The service is running normally.

- **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.

- **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.

- **STOPPED**

The service was stopped intentionally and cleanly.

- **ERROR_RESTARTING**

The service is in an error state. Oracle NoSQL Database attempts to restart the service.

- **ERROR_NO_RESTART**

The service is in an error state and is not automatically restarted. Administrative intervention is required.

- **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.

Note

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.

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.

Note that the Admin's Topology tab only shows abnormal service statuses. A service that is RUNNING does not display its status in that tab.

Useful Commands

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

- `java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar ping -host <host> -port <registryport>`

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.

- `jps -m`

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

In addition you can use the administration console to investigate the state of the KVStore. Point your browser to the administration port chosen on the administration host.

Chapter 4. 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 capacity value. 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 obey 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 from 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.

The initial configuration, or topology of the store is set when the store is created. Over time, it may be necessary to change the topology of the store. 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 that can execute write operations.

You change the store's configuration by changing the number or capacity of Storage Nodes available, or the replication factor of a zone. To change from one configuration to another, you either create a new initial topology, or you clone an existing topology and modify it into your target topology. You then deploy this target topology.

Note

The deployment of the target topology is potentially a long-running operation and the time required scales with the amount of data that must be moved. During the deployment, the system updates the topology at each step. Because of that, the store passes through intermediate topologies which were not explicitly created by the user.

This chapter discusses how configuration, or topological changes are made in a store.

Note

Configuration changes should not be made while a snapshot is being taken and vice versa. When making configuration changes it is safest to first create a snapshot as a

backup and then make the changes. For additional information on creating snapshots, see [Taking a Snapshot \(page 51\)](#).

Steps for Changing the Store's Topology

When you change your topology, you should go through these steps:

1. [Make the Topology Candidate \(page 32\)](#)
2. [Transform the Topology Candidate \(page 33\)](#)
3. [View the Topology Candidate \(page 36\)](#)
4. [Validate the Topology Candidate \(page 36\)](#)
5. [Preview the Topology Candidate \(page 37\)](#)
6. [Deploy the Topology Candidate \(page 37\)](#)
7. [Verify the Store's Current Topology \(page 39\)](#)

Creating a new topology may be an iterative process. You may want to try different options to see what may be best before the changes are deployed. In the end, 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 decide if they are appropriate.

The possible transformations include redistributing data, increasing replication factor, and rebalancing. These are described in [Transform the Topology Candidate \(page 33\)](#).

The following sections walk you through the process of changing the configuration for your store using the Administration Command Line Interface.

Make the Topology Candidate

To create the first topology candidate for an initial deployment, before any Replication Nodes exist, you use the `topology create` command. The `topology create` command takes a topology name, a pool name and the number of partitions as arguments.

Note

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.

For example:

```
kv-> topology create -name firstTopo -pool BostonPool -partitions 300  
Created: firstTopo
```

This initial topology candidate can be deployed, without any further transformations, using the `plan deploy-topology` command.

After the store is deployed, topology candidates are created with the topology clone command. A clone's source 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
Created CloneTopo
```

Also, there is a variant of the topology clone command that takes the following arguments:

- -current

If specified, use the current, deployed topology as a source.

- -name <to topology>

The name of the clone.

For example:

```
kv-> topology clone -current -name ClonedTopo
Created ClonedTopo
```

Transform the Topology Candidate

After the initial deployment, the store is changed by deploying a topology candidate that differs from the topology currently in effect. This target topology is generated by transforming a topology candidate using the topology redistribute, rebalance, or change-repfactor 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 the store.

Increase Data Distribution

You can increase data distribution in order to enhance write throughput by using the topology redistribute command. The redistribute command only works if new Storage Nodes are added to permit the creation of new shards. Partitions are distributed across the new shards, resulting in more Replication Nodes to service write operations.

The following example demonstrates adding a set of Storage Nodes and redistributing the data to those nodes. In this example four nodes are added because the zone's replication factor is four and the new partition requires 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 uses added capacity to create new shards and to migrate partitions to those shards. The command fails if the number of new shards is not greater than the current number of shards.

Note

You should not issue redistribute commands against a mixed shard store. 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. New Replication Nodes are created for each shard and are assigned to Storage Nodes following the topology rules described earlier. It may be necessary to move existing Replication Nodes to different Storage Nodes to best use available resources while still complying with the topology rules.
2. Partitions are distributed evenly among all shards. Partitions that are in shards that are over populated will move to the shards with the least number of partitions.

3. You do not specify which partitions are moved.

Increase Replication Factor

You can increase the replication factor and create more copies of the data to improve read throughput and availability by using the `topology change-repfactor` command. 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, if there are a large number of shards, this command may require a significant number of new Storage Nodes to be successful.

For additional information on how to identify your primary replication factor and its implications, see [Replication Factor \(page 159\)](#).

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:

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 \(page 31\)](#). Changes to the physical characteristics of the store can make the current topology of the store violate those rules. For example, after performance tuning, you may want to decrease the capacity of a Storage Node. If that node was already hosting the maximum permissible number of Replication Nodes, reducing the capacity will put the store out of compliance with the capacity rules.

You can balance a non-compliant configuration by using the `topology rebalance` command. This command requires a topology candidate name and a Storage Node pool name.

The following example examines the topology candidate named `repTopo` for any violations to the topology rules. If no improvements are needed as a result of this examination, the

topology candidate is unchanged. However, if improvements are needed, then the topology rebalance command will move or create Replication Nodes, using the Storage Nodes in the BostonPool pool, in order to correct any violations. The command does not under any circumstances create additional shards.

```
kv-> topology rebalance -name repTopo -pool BostonPool
Rebalanced: repTopo
```

If there are an insufficient number of Storage Nodes, the topology rebalance command may not be able to correct all violations. In that case, the command makes as much progress as possible, and warns of remaining issues.

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
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 \(page 31\)](#). 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

With a satisfactory topology candidate, you can use the admin service to generate and execute a plan which migrates the store to the new topology.

You can deploy the topology candidate by using the `plan deploy-topology` command. This command 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 refuses to deploy a topology candidate if it introduces new violations of the topology rules. This behavior can be overridden by using the `-force` optional plan flag on that command.

For example:

```
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 obeys the topology rules described in [Determining Your Store's Configuration \(page 31\)](#).

You should examine the new topology and decide if it is satisfactory, and if not apply more transformations, or start over with different parameters.

For example:

```

kv-> verify configuration
Verify: starting verification of store mystore based upon
      topology sequence #470
      300 partitions and 8 storage nodes

```

```

Time: 2015-03-04 17:34:23 UTC   Version: 12.1.3.2.15
See localhost:KVR00T/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]
      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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
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]      Status: RUNNING
      Ver: 12cR1.3.2.15 2015-03-04 06:35:02 UTC   Build id: 8e70b50c0b0e
Verify:      Rep Node [rg2-rn4]   Status: RUNNING,REPLICA ...
Verification complete, no violations.

```

Chapter 5. Upgrading an Existing Oracle NoSQL Database Deployment

This section describes how to upgrade the software of your Oracle NoSQL Database deployment.

Installing new software requires that each node be restarted. Depending on the configuration of your store, it may be possible to upgrade it while the store continues to remain online and available to clients, which is useful in cases where downtime is not desirable. An offline upgrade can also be performed instead if desired. The offline upgrade involves shutting down all nodes, installing new software on all of them, and then restarting them. That may be simpler and is certainly permitted, but it does mean that the store is not available for the duration of the upgrade.

Online upgrade can succeed if the store's replication factor is greater than 2, and the upgrade is not being performed on a release 2.0 store whose topology contains only a single zone.

For stores with a replication factor greater than two, the shards can maintain their majorities and continue reading and writing data on behalf of clients while their components are restarted, one at a time. If the replication factor is 2 or 1, then the majorities cannot be maintained across the restart of a single node, and each shard will become unavailable for a short time.

Preparing to Upgrade

Before beginning the upgrade process, you should take a backup of the store by creating a snapshot. See [Taking a Snapshot \(page 51\)](#).

In Oracle NoSQL Database, configuration changes and other administrative activities involve plans. In NoSQL DB release 3.0, the execution or restarting of plans that were created in NoSQL DB release 1.0 is not supported. Any plans that are not in a completed state should be canceled before the upgrade begins. For information about plans, see [Plans \(page 9\)](#).

Note

During the upgrade process, you should not create any plans until all services in the store have been upgraded.

Application programs that use the kvstore client library should be upgraded to the new software version as soon as possible after the service components have been upgraded. New clients can also be used with an old store, so it is possible to test upgraded clients before upgrading the store services.

General Upgrade Notes

This section contains upgrade information that is generally true for all versions of Oracle NoSQL Database. Upgrade instructions and notes for specific releases are given in sections following this one.

When Oracle NoSQL Database is first installed, it is placed in a KVHOME directory, which may be per-machine, or optionally be shared by multiple Storage Nodes (for example, using NFS). Here, we call this existing KVHOME location, OLD_KVHOME.

Note

It is useful for installations to adopt a convention for KVHOME that includes the release number. That is, always use a KVHOME location such as /var/kv/kv-M.N.O, where M.N.O are the release.major.minor numbers. This can be easily achieved by simply unzip/untarring the distribution into a common directory (/var/kv in this example).

Installing new software requires that each node be restarted. Oracle NoSQL Database is a replicated system, so to avoid excessive failover events it is recommended that any node that is running as a MASTER be restarted after all those marked REPLICA. This command tells you which nodes are MASTER and REPLICA:

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar ping -host <hostname> -port <port>
```

To make the process more debuggable, when upgrading a node and while the Storage Node is stopped, you should move the existing log files under KVROOT and KVROOT/<storename>/log to any other directory.

Use the host and registry port for any active node in the store. For example, in the following example, rg1-rn1 and rg2-rn1 are running as MASTER and should be restarted last:

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar ping -port 5000 -host node01
Pinging components of store mystore based upon topology sequence #315
300 partitions and 6 storage nodes
Time: 2015-06-11 04:49:17 UTC   Version: 12.1.3.4.0
Shard Status: healthy:2 writable-degraded:0 read-only:0 offline:0
Admin Status: healthy
Zone [name=Boston id=zn1 type=PRIMARY]
RN Status: online:6 offline:0 maxDelayMillis:1 maxCatchupTimeSecs:0
Storage Node [sn1] on node01:5000
Zone: [name=Boston id=zn1 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
  Admin [admin1]      Status: RUNNING,MASTER
  Rep Node [rg1-rn1]   Status: RUNNING,MASTER
    sequenceNumber:327 haPort:5011
Storage Node [sn2] on node02:6000
Zone: [name=Boston id=zn1 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
  Rep Node [rg1-rn2]   Status: RUNNING,REPLICA
    sequenceNumber:327 haPort:6010 delayMillis:1 catchupTimeSecs:0
Storage Node [sn3] on node03:7000
Zone: [name=Boston id=zn1 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
  Rep Node [rg1-rn3]   Status: RUNNING,REPLICA
```

```

sequenceNumber:327 haPort:7010 delayMillis:1 catchupTimeSecs:0
Storage Node [sn4] on node04:8000
Zone: [name=Boston id=zn1 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
Rep Node [rg2-rn1]      Status: RUNNING,MASTER
sequenceNumber:327 haPort:8010
Storage Node [sn5] on node05:9000
Zone: [name=Boston id=zn1 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
Rep Node [rg2-rn2]      Status: RUNNING,REPLICA
sequenceNumber:327 haPort:9010 delayMillis:1 catchupTimeSecs:0
Storage Node [sn6] on node06:10000
Zone: [name=Boston id=zn1 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC  Build id: db9ec397225c
Rep Node [rg2-rn3]      Status: RUNNING,REPLICA
sequenceNumber:327 haPort:10010 delayMillis:1 catchupTimeSecs:0

```

When upgrading your store, place the updated software in a new KVHOME directory on a Storage Node running the admin service. The new KVHOME directory is referred to here as NEW_KVHOME. If the KVHOME and NEW_KVHOME directories are shared by multiple Storage Nodes (for example, using NFS), it is necessary to maintain both locations while the upgrade is going on. The original KVHOME directory is no longer needed once the upgrade is complete. In this case, the start up procedure on each node needs to be modified to refer to the value of NEW_KVHOME in order to have it use the new software.

Note

In cases where each node has its own copy of the software installation, then it is possible to replace the installation in place and not modify the value of KVHOME.

Upgrade from Release 2.0 to Release 3.0

Upgrading a store from release 2 to release 3 can be accomplished one Storage Node at a time because a mix of release 2 and 3 Storage Nodes are permitted to run simultaneously in the same store. This allows you to strategically upgrade Storage Nodes in the most efficient manner.

Note

Upgrading a 1.0 store directly to release 3 is not supported. You must upgrade your store from 1.0 to 2.0 before upgrading to release 3. For instructions on how to upgrade your 1.0 store, see [Upgrade from NoSQL DB Release 1.0 to NoSQL DB Release 2.0 \(page 49\)](#).

Note

If your store contains more than a handful of Storage Nodes, you may want to perform your upgrade using a script. See [Using a Script to Upgrade to Release 3 \(page 48\)](#) for more information.

To avoid potential problems, new CLI commands are available to identify when nodes can be upgraded at the same time. These commands are described in the following procedure.

To upgrade your store, start by installing the release 3 software on a Storage Node that is running an admin service. The new CLI commands require an updated admin service in order to function.

Do the following:

1. On a Storage Node running a release 2 admin service:
 - a. Place the updated software in a new KVHOME directory on a Storage Node running the admin service. The new KVHOME directory is referred to here as NEW_KVHOME. If nodes share this directory using NFS, this only needs to be done once for each shared directory.
 - b. Stop the Storage Node using the release 2 CLI. When you do this, this shuts down the admin service on that Storage Node.

If you have configured the node to automatically start the Storage Node Agent on reboot using /etc/init.d, Upstart, or some other mechanism first modify that script to point to NEW_KVHOME.

Once you have modified that script, shutdown the Storage Node:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar stop -root <kvroot>
```

- c. Restart the Storage Node using the release 3 code:

```
nohup java -Xmx256m -Xms256m \  
-jar NEW_KVHOME/lib/kvstore.jar start -root <kvroot>&
```

(If the system is configured to automatically restart the Storage Node Agent, this step may not be necessary.)

- d. Use the CLI to connect to the Storage Node which is now running the release 3 code:

```
java -Xmx256m -Xms256m \  
-jar NEW_KVHOME/lib/kvstore.jar runadmin -port 5000 -host node1
```

- e. Verify that all the Storage Nodes in the store are running the proper software level required to upgrade to release 3. Note that any patch release level of 2.0 or 2.1 meets the minimum software level requirements.

```
kv-> verify prerequisite  
Verify: starting verification of mystore based upon topology  
sequence #315  
300 partitions and 6 storage nodes. Version: 12.1.3.0.1 Time:  
2014-01-07 08:19:15 UTC  
See node1:<KVR00T>/mystore/log/mystore_{0..N}.log for progress  
messages  
Verify prerequisite: Storage Node [sn3] on node3:5000
```

```
Zone: [name=Boston id=zn1 type=PRIMARY] Status: RUNNING
Ver: 12cR1.2.1.54 2013-11-11 12:09:35 UTC Build id: 921c25300b5e

...

Verification complete, no violations.
```

Note that only a partial sample of the verification command's output is shown here. The important part is the last line, which shows no violations.

The most likely reason for a violation is if you are (accidentally) attempting a release level downgrade. For example, it is illegal to downgrade from a higher minor release to a lower minor release. Possibly this is occurring simply because you are running the CLI using a package at a minor release level that is lower than the release level at other nodes in the store.

Note

It is legal to downgrade from a higher *patch* level to a lower patch level. So, for example downgrading from 2.1.4 to 2.1.3 would be legal, while downgrading from 2.1.3 to 2.0.39 would not be legal.

Also, a violation will occur if you attempt to upgrade 1.0 nodes directly to release 3. When upgrading a 1.0 store, you must first upgrade to 2.0, and then upgrade to release 3. For more information on upgrading a 1.0 store, see [Upgrade from NoSQL DB Release 1.0 to NoSQL DB Release 2.0 \(page 49\)](#).

In any case, if the `verify prerequisite` command shows violations, resolve the situation before you attempt to upgrade the identified nodes.

- f. Obtain an ordered list of the nodes to upgrade.

```
kv-> show upgrade-order
sn3 sn4
sn2 sn5
sn6
```

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, if desired. 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.

The Storage Nodes combined together on a single line can be upgraded together. Therefore, for this output, you would upgrade sn3 and sn4. Then upgrade sn2 and sn5. And, finally, upgrade sn6.

Note that you must completely upgrade a group of nodes before continuing to the next group. That is, upgrade sn3 and sn4 before you proceed to upgrading sn2, sn5, or sn6.

2. For each of the Storage Nodes in the first group of Storage Nodes to upgrade (sn3 and sn4, in this example):

- a. Place the release 3 software in a new KVHOME directory. The new KVHOME directory is referred to here as NEW_KVHOME. If nodes share this directory using NFS, this only needs to be done once for each shared directory.
- b. Stop the Storage Node using the release 2 utility.

If you have configured the node to automatically start the Storage Node Agent on reboot using /etc/init.d, Upstart, or some other mechanism first modify that script to point to NEW_KVHOME.

Once you have modified that script, shutdown the Storage Node using the old code:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar stop -root <kvroot>
```

- c. Restart the Storage Node using the new code:

```
nohup java -Xmx256m -Xms256m \  
-jar NEW_KVHOME/lib/kvstore.jar start -root <kvroot>&
```

(If the system is configured to automatically restart the Storage Node Agent, this step may not be necessary.)

3. Verify the upgrade before upgrading your next set of nodes. This command shows which nodes have been successfully upgraded, and which nodes still need to be upgraded:

```
kv-> verify upgrade  
Verify: starting verification of mystore based upon topology  
sequence #315  
300 partitions and 6 storage nodes. Version: 12.1.3.0.1 Time: ....  
See node1:<KVROOT>/mystore/log/mystore_{0..N}.log for progress  
messages  
Verify upgrade: Storage Node [sn3] on node3:5000  
Zone: [name=Boston id=zn1 type=PRIMARY] Status: RUNNING  
Ver: 12cR1.3.0.1 2013-12-18 06:35:02 UTC Build id: 8e70b50c0b0e  
  
...  
  
Verify: sn2: Node needs to be upgraded from 12.1.2.1.54 to  
version 12.1.3.0.0 or newer  
  
...  
  
Verification complete, 0 violations, 3 nodes found.
```



```
Verification note: [sn2] Node needs to be upgraded from
12.1.2.1.54 to version 12.1.3.0.0 or newer
Verification note: [sn5] Node needs to be upgraded from
12.1.2.1.54 to version 12.1.3.0.0 or newer
Verification note: [sn6] Node needs to be upgraded from
12.1.2.1.54 to version 12.1.3.0.0 or newer
```

For brevity and space, we only show part of the output generated by the verify upgrade command. Those nodes which have been upgraded are identified with a verification message that includes the current software version number:

```
Verify upgrade: Storage Node [sn3] on node3:5000
Zone: [name=Boston id=zn1 type=PRIMARY]
Status: RUNNING
Ver: 12cR1.3.0.1 2013-12-18 06:35:02 UTC Build id: 8e70b50c0b0e
```

Those nodes which still need to be upgraded are identified in two different ways. First, the verification message for the node indicates that an upgrade is still necessary:

```
Verify: sn2: Node needs to be upgraded from 12.1.2.1.54 to
version 12.1.3.0.0 or newer
```

Second, the very end of the verification output identifies all the nodes that still need to be upgraded:

```
Verification complete, 0 violations, 3 notes found.
Verification note: [sn2] Node needs to be upgraded from
12.1.2.1.54 to version 12.1.3.0.0 or newer
Verification note: [sn5] Node needs to be upgraded from
12.1.2.1.54 to version 12.1.3.0.0 or newer
Verification note: [sn6] Node needs to be upgraded from
12.1.2.1.54 to version 12.1.3.0.0 or newer
```

Note

If the verification shows nodes you thought were upgraded as being still in need of an upgrade, you *must* resolve that problem before upgrading the other nodes in your store. As a kind of a sanity check, you can verify just those nodes you just finished upgrading:

```
kv-> verify upgrade -sn sn3 -sn sn4
Verify: starting verification of mystore based upon topology
sequence #315
...
Verification complete, no violations.
```

4. You can continue upgrading groups of Storage Nodes, as identified by the show upgrade-order command. Follow the procedure outlined above. Stop the release 2 Storage Node using the release 2 stop command, then restart the Storage Node using the release 3 start command. Continue doing this until all Storage Nodes have been 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
Calculating upgrade order, target version: 12.1.3.0.1,
prerequisite: 11.2.2.0.23
sn2 sn5
sn6
```

5. When you are all done upgrading your Storage Nodes, the `verify upgrade` command will show no verification notes at the end of its output:

```
kv-> verify upgrade
Verify: starting verification of mystore based upon topology
sequence #315
...
Verification complete, no violations.
```

Using a Script to Upgrade to Release 3

For any deployments with more than a handful of Storage Nodes, the manual upgrade procedure described above becomes problematic. In that case, you should probably upgrade your store using a script.

An example script (bash shell script) is available for you to examine in the release 3 distribution. It can be found here:

```
<KVHOME>/examples/upgrade/onlineUpgrade
```

This script has the same upgrade restrictions as was described earlier in this section: it will only upgrade a release 2 installation to release 3, and your store must have a replication factor of at least 3 in order for your store to be available during the upgrade process.

The provided script is an example only. It must be modified in order for it to properly function for your installation.

Note that the script does not perform any software provisioning for you. This means you are responsible for placing the release 3 package on your host machines in whatever location you are using for your installation software. That said, the script communicates with your host machines using `ssh`, so you could potentially enhance the script to provision your machines using `scp`.

Because the script uses `ssh`, in order for it to function you must configure your machines to allow automatic login (that is, login over `ssh` without a password). `ssh` supports public/private key authentication, so this is generally a secure way to operate.

For information on how to configure `ssh` in this way, see http://www.linuxproblem.org/art_9.html. For information on how to install and configure `ssh` and the `ssh` server, see your operating system's documentation.

Upgrade from NoSQL DB Release 1.0 to NoSQL DB Release 2.0

NoSQL DB release 2.0 changes the internal protocols by which the components of a store communicate with one other. Limited cross-version compatibility has been implemented. Components (including applications that use the embedded client library) running NoSQL DB release 1.0 and components running NoSQL DB release 2.0 can communicate with one another, so that a store can remain running and available to clients during the upgrade. However, operations that change the configuration of the store should not be attempted while the upgrade is in progress.

Similarly, NoSQL DB release 2.0 changes the persistent meta-data that is kept by the components of the store. When a component, such as the Admin or the Replication Node, first starts on the new software, it will automatically convert its meta-data to the new format. Therefore it is not possible to downgrade to the older NoSQL DB release without restoring the store from a previous backup.

Note

Online upgrade is not supported for a store with a NoSQL DB release 1.0 topology that contains multiple zones. If your store has such a topology, please contact technical support before attempting to upgrade.

Note

The NoSQL DB release 1.0 administrative CLI program is not compatible with a NoSQL DB release 2.0 Admin service, nor vice-versa. Make sure that you are using the compatible library when running the CLI. It may be necessary to clear the cache of your web browser when you first connect to the NoSQL DB release 2. Admin Console.

To upgrade your release 1.0 installation to release 2.0, perform the following steps for each node (machine) in your store:

1. Place the 2.0 software in a new KVHOME directory – referred to here as NEW_KVHOME. If nodes share this directory using NFS, this only needs to be done once for each shared directory.
2. If you have configured the node to automatically start the Storage Node Agent on reboot using /etc/init.d, Upstart, or some other mechanism for example, using the command:

```
nohup java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar start -root <kvroot> ...&
```

First modify that script to point to NEW_KVHOME.

3. For each KVROOT (usually, once per node):
 - a. Stop the Storage Node using the old code:

```
java -Xmx256m -Xms256m \  
-jar OLD_KVHOME/lib/kvstore.jar stop -root <kvroot> \  
[-config <configfile>]
```

- b. Restart the Storage Node using the new code:

```
nohup java -Xmx256m -Xms256m \  
-jar NEW_KVHOME/lib/kvstore.jar start -root <kvroot> \  
[-config <configfile>] &
```

If the system is configured to automatically restart the Storage Node Agent, this step may not be necessary.

4. Make sure that any administrative scripts or other files that reference OLD_KVHOME have been changed.

Once you are done, OLD_KVHOME can be removed.

Chapter 6. Administrative Procedures

This chapter contains procedures that may be generally useful to the Oracle NoSQL Database administrator.

Note

Oracle NoSQL Database Storage Nodes and Admins make use of an embedded database (Oracle Berkeley DB, Java Edition). You should never directly manipulate the files maintained by this database. In general it is a bad idea to move, delete or modify the files and directories located under KVROOT unless you are asked to do so by Oracle Customer Support. But in particular, *never* move or delete any file ending with a `jdb` suffix. These will all be found in an `env` directory somewhere under KVROOT.

Backing Up the Store

To back up the KVStore, you take snapshots of nodes in the store and copy the resulting snapshots to a safe location. Note that the distributed nature and scale of Oracle NoSQL Database makes it unlikely that a single machine can hold the backup for the entire store. These instructions do not address where and how snapshots are stored.

Taking a Snapshot

A snapshot provides consistency across all records within the same shard, but not across partitions in independent shards. The underlying snapshot operations are performed in parallel to the extent possible in order to minimize any potential inconsistencies.

To take a snapshot from the admin CLI, use the `snapshot create` command:

```
kv-> snapshot create -name <snapshot name>
```

Using this command, you can create or remove a named snapshot. (The name of the snapshot is provided using the `<name>` parameter.) You can also remove all snapshots currently stored in the store.

For example, to create and remove a snapshot:

```
kv-> snapshot create -name Thursday
Created snapshot named 110915-153514-Thursday on all 3 nodes
kv-> snapshot remove -name 110915-153514-Thursday
Removed snapshot 110915-153514-Thursday
```

You can also remove all snapshots currently stored in the store:

```
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
```

Note

Snapshots should not be taken while any configuration (topological) changes are being made, because the snapshot might be inconsistent and not usable. At the time of the snapshot, use ping and then save the information that identifies masters for later use during a load or restore. For more information, see [Snapshot Management \(page 52\)](#).

Snapshot Management

When you run a snapshot, data is collected from every Replication Node in the system, including both masters and replicas. If the operation does not succeed for at least one of the nodes in a shard, it fails.

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.

At the time of the snapshot, you can identify which nodes are currently running as the master using the ping command. There is a master for each shard in the store and they are identified by the keyword: MASTER. For example, in the following example, replication node rg1-rn1, running on Storage Node sn1, is the current master:

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar ping -port 5000 -host node01
Pinging components of store mystore based upon topology sequence #316
300 partitions and 3 storage nodes
Time: 2015-06-11 02:54:35 UTC   Version: 12.1.3.4.0
Shard Status: healthy:3 writable-degraded:0 read-only:0 offline:0
Admin Status: healthy
Zone [name=Boston id=zn1 type=PRIMARY]
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: 12cR1.3.4.0 2015-05-29 12:09:04 UTC   Build id: db9ec397225c
    Admin [admin1]           Status: RUNNING,MASTER
    Rep Node [rg1-rn1]       Status: RUNNING,REPLICA
      sequenceNumber:231 haPort:5011 delayMillis:1 catchupTimeSecs:0
    Rep Node [rg2-rn1]       Status: RUNNING,REPLICA
      sequenceNumber:231 haPort:5012 delayMillis:1 catchupTimeSecs:0
    Rep Node [rg3-rn1]       Status: RUNNING,MASTER
      sequenceNumber:227 haPort:5013
Storage Node [sn2] on node02:6000
  Zone: [name=Boston id=zn1 type=PRIMARY]   Status: RUNNING
  Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC   Build id: db9ec397225c
    Rep Node [rg1-rn2]       Status: RUNNING,MASTER
      sequenceNumber:231 haPort:6010
    Rep Node [rg2-rn2]       Status: RUNNING,REPLICA
      sequenceNumber:231 haPort:6011 delayMillis:1 catchupTimeSecs:0
    Rep Node [rg3-rn2]       Status: RUNNING,REPLICA
```

```

sequenceNumber:227 haPort:6012 delayMillis:1 catchupTimeSecs:0
Storage Node [sn3] on node03:7000
Zone: [name=Boston id=zn1 type=PRIMARY] Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Rep Node [rg1-rn3] Status: RUNNING,REPLICA
sequenceNumber:231 haPort:7010 delayMillis:1 catchupTimeSecs:0
Rep Node [rg2-rn3] Status: RUNNING,MASTER
sequenceNumber:231 haPort:7011
Rep Node [rg3-rn3] Status: RUNNING,REPLICA
sequenceNumber:227 haPort:7012 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.

Note

Snapshots include the admin database. Depending on how the store might need to be restored, the admin database may or may not be useful.

Snapshot data for the local Storage Node is stored in a directory inside of the KVR00T directory. For each Storage Node in the store, you have a directory named:

```
KVR00T/<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, and they all are important. For example:

```

> ls /var/kvroot/mystore/sn1/rg1-rn1/snapshots/110915-153828-later
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.

Recovering the Store

There are two ways to recover your store from a previously created snapshot. The first mechanism allows you to use a backup to create a store with any desired topology. The second method requires you to restore the store using the *exact same* topology as was in use when the snapshot was taken.

Note

If you had to replace a failed Storage Node, that qualifies as a topology change. In that case, you must use the Load program to restore your store.

For information on how to replace a failed Storage Node, see [Replacing a Failed Storage Node \(page 61\)](#).

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.

By using this tool, you can restore the store to any topology, not just the one that was in use when the snapshot was created.

This mechanism works by iterating through all records in a snapshot, putting each record into the target store as it proceeds through the snapshot. It should be used only to restore to a new, empty store. Do not use this with an existing store because it 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 because this can cause your store to be restored in an inconsistent state.

This mechanism can only go at the speed of insertion of the new store. Because you probably have multiple shards in your store, you should be restoring your store from data taken from each shard. To do this, run the Load program in parallel, with each instance operating on data captured from different replication nodes.

The program's usage is:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar load [-verbose]  
-source <backupDir> -host <hostname> -port <port>  
-store <storeName> -username <user> -security <security-file-path>  
[-load-admin] [-force] [-status <pathToFile>]
```

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 \(page 55\)](#).

- `-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.
- `-status <pathToFile>` is an optional parameter that causes the status of the load operation to be saved in the named location on the local machine.
- `-security <security-file-path>` identifies the security file used to specify properties for login.
- `-source <backupDir>` identifies the on-disk location where the snapshot data is stored.
- `-store <storeName>` identifies the name of the store.
- `-username <user>` identifies the name of the user to login to the secured store.

For example, suppose there is a snapshot in `/var/backups/snapshots/110915-153828-later`, and there is a new store named "NewStore" on host "NewHost" using registry port 12345. Run the Load program on the host that has the `/var/backups/snapshots` directory:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar load \  
-source /var/backups/snapshots/110915-153828-later -store NewStore \  
-host NewHost -port 12345
```

Note

If the load fails part way through the restore, it can start where it left off by using the status file. The granularity of the status file is per-partition in this NoSQL DB release. If a status file is not used and there is a failure, the load needs to start over from the beginning. The target store does not need to be re-created if this happens, existing records are skipped.

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 configure security yet, even though the target store will eventually have security information. Also, do not make the store accessible to applications yet.

Create the new store:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar makebootconfig \  
-root KVR00T \  
-host NewHost -port 8000 -admin 8001 \  
-harange 8010,8020 \  
-capacity 1 \  
-store-security none
```

Start the new store:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar start \  
-root KVR00T&
```

Configure the new store:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar runadmin \  
-port 8000 -host NewHost  
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 `datacenter1/kvroot/newstore/sn1/admin1/env`.

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 -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar load \  
-source datacenter1/kvroot/newstore/sn1/admin1/env/ \  
-store NewStore -host NewHost -port 8000 -load-admin
```

Note

This command can be run more than once if something goes wrong, as long as the store is not accessible to applications.

4. 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 -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar load \
-source var/backups/snapshots/140827-144141-back -store NewStore \
-host NewHost -port 8000
```

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.

5. Configure security if the store is to be secure. For more information on configuring Oracle NoSQL Database securely, see the Oracle NoSQL Database Security Guide.
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 described in the previous section, but it can be used only to restore to the *exact same* topology as was used 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 the snapshot was taken.

You must perform this procedure for each Storage Node in your store, and for each service running on each Storage Node.

1. Put the to-be-recovered snapshot data in the recovery directory for the service corresponding to the snapshot data. For example, if you are recovering Storage Node sn1, service rg1-rn1 in store mystore, then log in to the node where that service is running and:

```
> mkdir KVROOT/mystore/sn1/rg1-sn1/recovery
> mv /var/kvroot/mystore/sn1/rg1-rn1/snapshots/110915-153828-later \
KVROOT/mystore/sn1/rg1-sn1/recovery/110915-153828-later
```

Do this for each service running on the Storage Node. Production systems should have only one resource running on a given Storage Node, but it is possible to deploy, for example, multiple replication nodes on a single Storage Node. A Storage Node can also have an administration process running on it, and this also needs to be restored.

2. Having done this, restart the Storage Node

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar stop -root /var/kvroot \
> nohup java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar start -root /var/kvroot&
```

On startup, the Storage Node notices the recovery directory, and moves that directory to the resource's environment directory and use it.

Note

Remember that this procedure recovers the store to the time of the snapshot. If your store was active since the time of the snapshot, then all data modifications made since the time of the last snapshot are lost.

Managing Avro Schema

Avro is a data format that can be used by values in your store's records. Whether a record's value uses the Avro data format is determined by your development team. However, the usage of the Avro data format is strongly recommended, so chances are good that your store uses Avro.

When store records use the Avro data format, your development team must define schema for their usage of that format. This schema is provided in flat-text files in JSON format, and must then be added to the store using the CLI. Schema can also be enabled and disabled, and multiple versions of the schema can exist at the same time. The ability to support multiple versions of the schema is required in order to support the ability to change (or evolve) schema.

Adding Schema

Avro schema is defined in a flat-text file, and then added to the store using the command line interface. For example, suppose you have schema defined in a file called `my_schema.avsc`. Then (assuming your store is running) you start your command line interface and add the schema like this:

```
> java -Xmx256m -Xms256m \  
-jar <kvhome>/lib/kvstore.jar runadmin -port <port> -host <host>  
kv-> ddl add-schema -file my_schema.avsc
```

Note that when adding schema to the store, some error checking is performed to ensure that the schema is correctly formed. Errors are problems that must be addressed before the schema can be added to the store. Warnings are problems that should be addressed, but are not so serious that the CLI refuses to add the schema. However, to add schema with Warnings, you must use the `-force` switch.

If you see any Errors or Warnings when you add schema to your store, you should discuss the problem with your development team so as to decide what to do about it.

Changing Schema

To change (evolve) existing schema, use the `-evolve` flag:

```
kv-> ddl add-schema -file my_schema.avsc -evolve
```

Note that when changing schema in the store, some error checking is performed to ensure that schema evolution can be performed correctly. This error checking consists of comparing the new schema to all currently enabled versions of that schema.

This error checking can result in either Errors or Warnings. Errors are fatal problems that must be addressed before the modified schema can be added to the store. Errors represent

situations where data written with an old version of the schema cannot be read by clients using a new version of the schema.

Warnings are problems that can be avoided using a two-phase upgrade process. In a two-phase upgrade, all clients begin using the schema only for reading in phase I (the old schema is still used for writing), and then use the new schema for both reading and writing in phase II. Phase II may not be begun until phase I is complete; that is, no client may use the new schema for writing until all clients are using it for reading.

If you see any Errors or Warnings when you attempt to evolve schema in your store, you should discuss the problem with your development team so as to decide what to do about it.

Disabling and Enabling Schema

You cannot delete schema, but you can disable it:

```
kv-> ddl disable-schema -name avro.MyInfo.1
```

To enable schema that has been disabled:

```
kv-> ddl enable-schema -name avro.MyInfo.1
```

Showing Schema

To see all the schemas currently enabled in your store:

```
kv-> show schemas
```

To see all schemas, including those which are currently disabled:

```
kv-> show schemas -disabled
```

Increasing the capacity of a Storage Node

You can increase the capacity of a Storage Node by adding additional hard disks to the machine. Such a configuration permits the placement of each Replication Node on its own disk and ensures that the Replication Nodes on the SN are not competing for I/O resources. The location of this directory on its disk can be specified via the `storagedir` parameter.

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 -Xmx&javamxvalue; -Xms&javamxvalue; \  
-jar KVHOME/lib/kvstore.jar makebootconfig \  
-root KVR00T \  
-host node20 -port 5000 -admin 5001 \  
-hrange 5010,5030 \  
-capacity 1 \  
-memory_mb 200 \  
-store-security none \  
-storagedir /disk1/ondb/data
```

Start the new store:

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar start \  
-root KVR00T&
```

Configure the new store:

```
java -Xmx&javamxvalue; -Xms&javamxvalue; \  
-jar KVHOME/lib/kvstore.jar runadmin \  
-port 5000 -host node20  
kv-> configure -name kvstore  
Store configured: kvstore
```

2. Create a zone. Then create an administration process on a specific host:

```
plan deploy-zone -name Houston -rf 1 -wait  
Executed plan 1, waiting for completion...  
Plan 1 ended successfully  
plan deploy-sn -znname "Houston" -port 5000 -wait -host node20  
Executed plan 2, waiting for completion...  
Plan 2 ended successfully
```

```
plan deploy-admin -sn sn1 -port 5001 -wait  
Executed plan 3, waiting for completion...  
Plan 3 ended successfully
```

3. Create a topology, preview it, and then deploy it:

```
topology create -name 1x1 -pool AllStorageNodes -partitions 120  
Created: 1x1
```

```
topology preview -name 1x1  
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
```

```
plan deploy-topology -name 1x1 -wait  
Executed plan 4, waiting for completion...  
Plan 4 ended successfully
```

4. Suppose two more disk drives were added to the Storage Node, mounted as disk2 and disk3. Add the storage directories using the plan `change-storagedir` command.

```
kv-> plan change-storagedir -sn sn1 -storagedir /disk2/ondb/data \  
-add -wait  
Executed plan 5, waiting for completion...  
Plan 5 ended successfully
```

```
kv-> plan change-storagedir -sn sn1 -storagedir /disk3/ondb/data \  
-add -wait  
Executed plan 6, waiting for completion...  
Plan 6 ended successfully
```

5. 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  
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.

6. Redistribute your topology to expand the cluster in order to use the new capacity (3) of the Storage Node.

```
kv-> topology clone -current -name 1x3  
Created 1x3
```

```
kv-> topology redistribute -name 1x3 -pool AllStorageNodes  
Redistributed: 1x3
```

```
kv-> topology preview -name 1x3  
Topology transformation from current deployed topology to 1x3:  
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 1x3 -wait  
Executed plan 8, waiting for completion...  
Plan 8 ended successfully
```

Replacing a Failed Storage Node

If a Storage Node has failed, or is in the process of failing, you can replace the Storage Node. 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.

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. Do this on the hardware where your new Storage Node runs. You only need to specify the `-admin` option (the Admin Console's port) if the hardware hosts the Oracle NoSQL Database administration processes.

To create the "boot config" file, issue the following commands:

```
> mkdir -p KVRROOT      (if it doesn't already exist)
> java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar makebootconfig -root KVRROOT \
                                           -port 5000 \
                                           -admin 5001 \
                                           -host <hostname> \
                                           -harange 5010,5020 \
                                           -store-security none
```

3. Start the Oracle NoSQL Database software on the new node.

```
> nohup java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar start -root KVRROOT&
```

4. Deploy the new Storage Node to the new node. You use an existing administrative process to do this, either using the CLI or the Admin Console. To do this using the CLI:

```
> java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar runadmin \
-port <port> -host <host>
kv-> plan deploy-sn -zn <id> -host <host> -port <port> -wait
kv->
```

5. 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, sn26
kv->
```

6. 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 -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar stop -root KVRROOT
```


7. Migrate the services from one Storage Node to another. If the old node hosted an admin service, the `-admin-port` argument is required. The syntax for this plan is:

```
plan migrate-sn -from <old SN ID> -to <new SN ID> \
-admin-port <admin port>
```

Assuming that you are migrating from Storage Node 25 to 26 on port 5000, you would use:

```
kv-> plan migrate-sn -from sn25 -to sn26 -admin-port 5000
```

8. 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 sn25 -wait
```

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 \(page 54\)](#) for more information.

To replace a failed Storage Node by using an identical node (node uses the same host name, internet address, and port as the failed host):

1. Prerequisite information:
 - a. A running Admin process on a known host, with a known registry port.
 - b. The ID of the Storage Node to replace (e.g. "sn1").
 - c. Before starting the new Storage Node, the SN to be replaced must be taken down. This can be done administratively or via failure.

Note

It is recommended that the KVROOT is empty and that you do a full network recovery of data before proceeding.

The instructions below assume that the KVROOT is empty and has no valid data. When the new Storage Node Agent begins it starts the services it hosts, which recovers their data from other hosts. This recovery may take some time, depending on the size of the shards involved and it happens in the background.

2. Create the configuration using the `generateconfig` command:

The `generateconfig`'s usage is:

```
> java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar generateconfig \  
-host <hostname> -port <port> -sn <StorageNodeId> -target <zipfile>
```

For example:

```
> java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar generateconfig -host adminhost \  
-port 13230 -sn sn1 -target /tmp/sn1.config.zip
```

The command above creates the target "/tmp/sn1.config.zip" which is a zip file with the required configuration to re-create that Storage Node. The top-level directory in the zip file is the store's KVROOT.

3. Restore the Storage Node configuration on the target host:
 - a. Copy the zip file to the target host.
 - b. Unzip the archive into your KVROOT directory. That is, if KVROOT is /opt/kvroot, then do the following:

```
> cd/opt  
> unzip <path-to-sn1.config.zip>
```

4. Restart the Storage Node on the new host.

```
> java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar start -root KVROOT
```

Replacing a Failed Disk

If a disk has failed, or is in the process of failing, you can replace the disk. Disk replacement procedures are necessary to keep the store running. This section walks you through the steps of replacing 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 other two disks.

```
> java -jar KVHOME/lib/kvstore.jar makebootconfig \  
-root /opt/ondb/var/kvroot \  
-port 5000 \  
-admin 5001 \  
-host node09  
-h.arange 5010,5020 \  
-num_cpus 0 \  
-memory_mb 0 \  
-store-security none \  
-capacity 2 \  
-storagedir /disk1/ondb/data \  
-storagedir /disk2/ondb/data
```

With a boot configuration such as that shown above, the directory structure that is created and populated on each machine would then be:

- Machine 1 (SN1) -	- Machine 2 (SN2) -	- Machine 3 (SN3) -
/opt/ondb/var/kvroot	/opt/ondb/var/kvroot	/opt/ondb/var/kvroot
log files	log files	log files
/store-name	/store-name	/store-name
/log	/log	/log
/sn1	/sn2	/sn3
config.xml	config.xml	config.xml
/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

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.

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

It can take a considerable amount of time for the disk to recover all of its data; depending on the amount of data that previously resided on the disk before failure. It is also important to note that the system may encounter additional network traffic and load while the new disk is being repopulated.

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 \(page 85\)](#).

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 \(page 61\)](#). 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.

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 -Xmx256m -Xms256m \  
-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

Verification is a tool you can use to:

- Perform general troubleshooting of the store.

Verification inspects all components of the store. It also checks whether all store services are available, and for those services it checks whether there are 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 progress of the plan, or as an aid to diagnose a plan that is in an ERROR state. For example, if you can verify a Deploy Store plan while it is running against many Storage Nodes. You can watch the verify report at each iteration to see that more and more nodes have created and have come online.

For more information on how to manage plans, see [Plans \(page 9\)](#).

- Provide additional information about a plan that is in an ERROR state.

You run store verification using the `verify` command in the CLI. It requires no parameters, and by default it runs in verbose mode. For example:

```
kv-> verify configuration
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2015-06-16 00:01:05 UTC   Version: 12.1.3.4.0
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]
      RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY]
      RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=Queens id=zn3 type=PRIMARY]
      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]      Status: RUNNING
      Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Admin [admin1]      Status: RUNNING,MASTER
Verify:      Rep Node [rg1-rn2]      Status: RUNNING,REPLICA
      sequenceNumber:127 haPort:5011 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on node02:6000
      Zone: [name=Manhattan id=zn1 type=PRIMARY]      Status: RUNNING
      Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Rep Node [rg2-rn2]      Status: RUNNING,REPLICA
      sequenceNumber:127 haPort:6010 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on node03:7000
      Zone: [name=JerseyCity id=zn2 type=PRIMARY]      Status: RUNNING
      Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Admin [admin2]      Status: RUNNING,REPLICA
Verify:      Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
      sequenceNumber:127 haPort:7011 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:8000
      Zone: [name=JerseyCity id=zn2 type=PRIMARY]      Status: RUNNING
      Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Rep Node [rg2-rn3]      Status: RUNNING,REPLICA
      sequenceNumber:127 haPort:8010 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:9000
      Zone: [name=Queens id=zn3 type=PRIMARY]      Status: RUNNING
      Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Admin [admin3]      Status: RUNNING,REPLICA
Verify:      Rep Node [rg1-rn1]      Status: RUNNING,MASTER
      sequenceNumber:127 haPort:9011
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:10000
      Zone: [name=Queens id=zn3 type=PRIMARY]      Status: RUNNING
      Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Rep Node [rg2-rn1]      Status: RUNNING,MASTER
      sequenceNumber:127 haPort:10010

Verification complete, no violations.

```

A silent mode is available which shows only problems.

```

kv-> verify configuration -silent
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2015-06-16 04:00:02 UTC Version: 12.1.3.4.0
See node01:Data/virtualroot/datacenter1/kvroot/MetroArea/

```

```
log/MetroArea_{0..N}.log for progress messages
Verification complete, no violations.
```

Problems with the store are clearly reported. For example, if a Storage Node is unavailable, then in silent mode that problem is displayed in the following way:

```
kv-> verify configuration -silent
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2015-06-16 04:09:18 UTC Version: 12.1.3.4.0
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
```

In verbose mode, the above problem is shown in the following way:

```
kv-> verify configuration
Verify: starting verification of store MetroArea based upon
topology sequence #117
100 partitions and 6 storage nodes
Time: 2015-06-16 04:17:31 UTC Version: 12.1.3.4.0
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]
RN Status: online:1 offline: 1 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY]
RN Status: online:2 offline: 0 maxDelayMillis:1 maxCatchupTimeSecs:0
Verify: Zone [name=Queens id=zn3 type=PRIMARY]
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] Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify: Admin [admin1] Status: RUNNING,MASTER
Verify: Rep Node [rg1-rn2] Status: RUNNING,REPLICA
```

```

sequenceNumber:127 haPort:5011 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] 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]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Admin [admin2]      Status: RUNNING,REPLICA
Verify:      Rep Node [rg1-rn3]      Status: RUNNING,REPLICA
sequenceNumber:127 haPort:7011 delayMillis:1 catchupTimeSecs:0
Verify: == checking storage node sn4 ==
Verify: Storage Node [sn4] on node04:8000
Zone: [name=JerseyCity id=zn2 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Rep Node [rg2-rn3]      Status: RUNNING,REPLICA
sequenceNumber:127 haPort:8010 delayMillis:0 catchupTimeSecs:0
Verify: == checking storage node sn5 ==
Verify: Storage Node [sn5] on node05:9000
Zone: [name=Queens id=zn3 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Admin [admin3]      Status: RUNNING,REPLICA
Verify:      Rep Node [rg1-rn1]      Status: RUNNING,MASTER
sequenceNumber:127 haPort:9011
Verify: == checking storage node sn6 ==
Verify: Storage Node [sn6] on node06:10000
Zone: [name=Queens id=zn3 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.0 2015-05-29 12:09:04 UTC Build id: db9ec397225c
Verify:      Rep Node [rg2-rn1]      Status: RUNNING,MASTER
sequenceNumber:127 haPort:10010

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
```

Note

The verify output is only displayed in the shell when the command is finished. You can tail or grep the Oracle NoSQL Database log file in order to get a sense of how the verify is progressing. Look for the string Verify. For example:

```
grep Verify /KVRT1/mystore/log/mystore_0.log
```

Monitoring the Store

Information about the performance and availability of your store can be obtained both from a server side and client side perspective:

- Your Oracle NoSQL Database applications can obtain performance statistics using the `oracle.kv.KVStore.getStats()` class. This provides a client side view of the complete round trip performance for Oracle NoSQL Database operations.
- Oracle NoSQL Database automatically captures Replication Node performance statistics into a log file that can easily be imported and analyzed with spreadsheet software. Statistics are tracked, logged and written at a user specified interval to a CSV file (`je.stat.csv`) in the Environment directory. The logging occurs per-Environment when the Environment is opened in read/write mode.

Configuration parameters control the size and number of rotating log files used (similar to java logging, see [java.util.logging.FileHandler](#)). For a rotating set of files, as each file reaches a given size limit, it is closed, rotated out, and a new file is opened. Successively older files are named by adding "0", "1", "2", etc. into the file name. The format is `je.stat[version number].csv`

- The Oracle NoSQL Database administrative service collects and aggregates status information, alerts, and performance statistics components that are generated in the store. This provides a detailed, server side view of behavior and performance of the Oracle NoSQL Database server.
- Each Oracle NoSQL Database Storage Node maintains detailed logs of trace information from the services that are housed on that node. The administrative service presents an aggregated, store-wide view of these component logs, but the logs are nevertheless available on each Storage Node in the event that the administrative service is somehow not available, or if it is more convenient to examine the individual logs.
- Oracle NoSQL Database allows Java Management Extensions (JMX) or Simple Network Management Protocol (SNMP) agents to be optionally available for monitoring. The

SNMP and JMX interfaces allow you to poll the Storage Nodes for information about the storage node and about any replication nodes that are hosted on the Storage Node. See [Standardized Monitoring Interfaces \(page 92\)](#) for more information.

In addition to the logging mechanisms noted above, you can also view the current health of the store using the Admin Console. This information is viewable on the Topology pane. It shows you what services are currently unavailable. Problematic services are highlighted in red. Two lines at the top of the pane summarize the number of available and unavailable services.

Finally, you can monitor the status of the store by verifying it from within the CLI. See [Verifying the Store \(page 67\)](#) for more information. You can also use the CLI to examine events.

Events

Events are special messages that inform you of the state of your system. As events are generated, they are routed through the monitoring system so that you can see them. There are four types of events that the store reports:

1. State Change events are issued when a service starts up or shuts down.
2. Performance events report statistics about the performance of various services.
3. Log events are records produced by the various system components to provide trace information about debugging. These records are produced by the standard `java.util.logging` package.
4. Plan Change events record the progress of plans as they execute, are interrupted, fail or are canceled.

Note that some events are considered critical. These events are recorded in the administration service's database, and can be retrieved and viewed using the CLI or the Admin Console.

Other Events

Plan Change events cannot be directly viewed through Oracle NoSQL Database's administrative interfaces. However, State Change events, Performance events, and Log events are recorded using the EventRecorder facility internal to the Admin. Only events that are considered "critical" are recorded, and the criteria for being designated as such vary with the type of the event. All state change events are considered critical, but only SEVERE log events are. Performance events are considered critical if the reported performance is below a certain threshold.

All such events can be viewed in the CLI using the `show events` and `show event` commands.

Use the CLI `show events` command with no arguments to see all the unexpired events in the database. You can bound the range of events that are displayed using the `-from` and `-to` arguments. You can filter events by type or id as well, using either the `-type` or the `-id` arguments respectively.

For example, this is a fragment of the output from the `show events` command:

```
kv-> show events
idarpdfbS STAT 2015-08-13 22:18:39.287 UTC sn1 RUNNING sev1
idarpeg0S STAT 2015-08-13 22:18:40.608 UTC sn2 RUNNING sev1
idarphmuS STAT 2015-08-13 22:18:44.742 UTC rg1-rn1 RUNNING sev1
idarpjLLS STAT 2015-08-13 22:18:47.289 UTC rg1-rn2 RUNNING sev1
idartfcuS STAT 2015-08-13 22:21:48.414 UTC rg1-rn2 UNREACHABLE sev2
                                     (reported by admin1)
```

This shows four service state change events (sev1) and one UNREACHABLE (sev2) log event. The tags at the beginning of each line are individual event record identifiers. If you want to see detailed information for a particular event, you can use the "show event" command, which takes as its argument an event record identifier:

```
kv-> show event -id idartfcuS
idartfcuS STAT 2015-08-13 22:21:48.414 UTC rg1-rn2 UNREACHABLE sev2
                                     (reported by admin1)
```

and so on, for a complete stack trace.

Events expire from the system after a set period, which defaults to thirty days.

The Sev1 flag is associated with the following service status change events: STARTING, WAITING_FOR_DEPLOY, RUNNING, STOPPING and STOPPED. Sev2 is associated with the ERROR_RESTARTING, ERROR_NO_RESTART and UNREACHABLE service status change events.

Setting Store Parameters

The three Oracle NoSQL Database service types; Admin, Storage Node and Replication Node; have configuration parameters, some of which can be tweaked after the service is deployed. To see the parameter values that can be changed, you use the following command in the CLI:

```
show parameters -service <id>
```

This command allows you to display service parameters and state for the specified service. The service may be a Replication Node, a Storage Node, or Admin service, as identified by any valid string. You can use the `-policy` optional 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"
```

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

The following parameters can be set for the Admin service:

- `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.

- `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 zero, then there is no limit.

- `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>`

This is an omnibus string that is added to the command line when the Admin process is started. It is intended for setting Java VM properties, such as `-Xmx` and `-Xms` to control the heap size. If the string is not a valid sequence of tokens for the JVM command line, the Admin process fails to start.

- `adminHttpPort=<Integer>`

Sets the port on which the Oracle NoSQL Database web-based Admin Console is contacted. Examples in this book use port 5001. If the value is 0, the web interface is disabled.

Storage Node Parameters

The following parameters can be set for Storage Nodes:

- `serviceLogFileCount=<Integer>`

Sets the number of log files that are kept, for this Storage Node and for all Replication Nodes hosted on this Storage Node. This value defaults to "20". It is used to control the amount of disk space devoted to logging history.

- `serviceLogFileLimit=<Integer>`

Limits the size of log files. After reaching this limit, the logging subsystem switches to a new log file. This setting applies to this Storage Node and to all Replication Nodes hosted on this Storage Node. This value defaults to "2,000,000" bytes. The limit specifies an approximate maximum amount to write (in bytes) to any one file. If the value is zero, then there is no limit.

- `haPortRange=<String>`

Defines the range of port numbers available for assigning to Replication Nodes that are hosted on this Storage Node. A port is allocated automatically from this range when a Replication Node is deployed. The format of the value string is "lowport,highport".

- `haHostname=<String>`

Sets the name of the network interface used by the HA subsystem. A valid string for a hostname can be a DNS name or an IP address.

- `capacity=<Integer>`

Sets the number of Replication Nodes that can be hosted on this Storage Node. This value is used to inform decisions about where to place new Replication Nodes. Capacity can be set to values greater than 1 when the Storage Node has sufficient disk, CPU, and memory to support multiple Replication Nodes. Default value: "1".

- `memoryMB=<Integer>`

Sets the amount of memory known to be available on this Storage Node, in megabytes. Defaults to "0", which means "unknown." and is determined automatically by the store as the total amount of RAM available on the machine.

You should not need to change this parameter under normal circumstances. Set it to a value less than the amount of RAM, if the machine has other applications running on it (not a recommended configuration) in order to reserve some memory for the other applications.

- `numCPUs=<Integer>`

Sets the number of CPUs known to be available on this Storage Node. Default value: 1.

- `rnHeapPercent=<Integer>`

Sets the percentage of a Storage Node's memory reserved for heap, for all RN processes hosted on this SN. Default value: 85.

- `rnHeapMaxMB=<Integer>`

Sets a hard limit for the maximum size of the Replication Node's Java VM heap. Default value is 0, which means the VM-specific limit is used. (Currently, this is roughly 32 GB.)

Do not set this value to greater than 32 GB. Doing so can adversely impact your Replication Node's performance.

Normally the Replication Node's heap size is determined by the amount of memory available to the Storage Node, and the `rnHeapPercent` parameter value. The value set for `rnHeapMaxMB` indicates an upper limit beyond which the heap will not grow regardless of the value set for `rnHeapPercent` or the amount of memory available to the Replication Node.

- `systemPercent=<Integer>`

Sets the percentage of the Storage Node's memory which will be used for operating system purposes. This memory is used by the operating system for purposes such as the file system cache, page mapping tables, file system handles, thread stacks, and so forth.

If this parameter is set to a value less than 100, then the off-heap cache might be used if there is memory left over after allowing for the heap and system use. See [Managing the Off-heap Cache \(page 171\)](#) for details.

Default value is 100.

- `mgmtClass=<String>`

The name of the class that provides the Management Agent implementation. See [Standardized Monitoring Interfaces \(page 92\)](#) for more information. The port cannot be a privileged port number (<1024).

- `mgmtPollPort=<Integer>`

Sets the port on which the SNMP agent listens.

- `mgmtTrapHost=<String>`

Sets the host to which SNMP notifications are sent.

- `mgmtTrapPort=<Integer>`

Sets the port to which SNMP notifications are sent.

- `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 should be large enough to accommodate the Storage Node and all the Replication Nodes (as defined by the capacity parameter) hosted on the machines. The number of ports required depends on whether the system is configured for security. For a non-secure system, each Storage Node consumes three ports (including the Registry Service) and each Replication Node consumes three ports in the range. An Admin, if configured consumes 2 ports. On a secure system, one additional port is required for each Storage Node, Replication Node and Admin. As a general rule, it is good practice to specify a range that is significantly larger than the minimum to allow for increases in Storage Node capacity or network problems that may render ports temporarily unavailable.

For a non-secure system, as a rough rule of thumb, you can use the following formula to arrive at an estimate for the port range size number:

```
3 (Storage Nodes, adding one for safety) +  
3 * capacity (the number of replication nodes) \  
+ 2 (added only if the Storage Node is hosting an admin process as well)
```

For example, if a Storage Node has capacity 1 and is hosting an admin process, the range size must be at least 8. You may want to increase the range size beyond this minimum (in increments of 3). Doing so allows for safety and expansion of the Storage Node without requiring future changes to this parameter. If capacity were 2, the range size must be greater than or equal to 11. It is best to find a range on the machine that is contiguous. If other services on the host use ports in the range, those will be skipped and the range must be larger.

If you are deploying a secure Oracle NoSQL Database then you can use the following formula to arrive at an estimate for the port range size number:

```
4 (Storage Nodes, adding one for safety) +  
4 * capacity (the number of replication nodes) \  
+ 3 (added only if the Storage Node is hosting an admin process as well)
```

where an additional port was added for each Storage Node, Replication Node or the Admin (if configured).

For more information on configuring Oracle NoSQL Database securely, see the Oracle NoSQL Database Security Guide.

Replication Node Parameters

The following parameters can be set for Replication Nodes:

- `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.

- `maxTrackedLatency=<Long TimeUnit>`

The highest latency that is included in the calculation of latency percentiles.

- `configProperties=<String>`

Contains property settings for the underlying BDB JE subsystem. Its format is `property=value;property=value....`

- `javaMiscParams=<String>`

A string that is added to the command line when the Replication Node process is started. It is intended for setting Java VM properties, such as `-Xmx` and `-Xms` to control the heap size. If the string is not a valid sequence of tokens for the JVM command line, the Admin process fails to start.

- `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.

- `statsInterval=<Long TimeUnit>`

Sets the collection period for latency statistics at this Replication Node. This value defaults to "60" seconds. Values like average interval latencies and throughput are averaged over this period of time.

- `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.

- `latencyCeiling=<Integer>`

If the Replication Node's average latency exceeds this number of milliseconds, it is considered an "alertable" event. Such an event produces a popup in the Admin Console, and it is stored in the Admin's database as a critical event. If SNMP or JMX monitoring is enabled, the event also causes an appropriate notification to be sent.

- `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.

- `rnCachePercent=<Integer>`

The portion of an RN's memory set aside for the JE environment cache.

Security Parameters

The following parameters can be implicitly and uniformly set across all Storage Nodes, Replication Nodes and Admins:

For more information on top-level and transport security parameters see the Oracle NoSQL Database Security Guide.

- `sessionTimeout=<Long TimeUnit>`

Specifies the length of time for which a login session is valid, unless extended. The default value is 24 hours.

- `sessionExtendAllowed=<Boolean>`

Indicates whether session extensions should be granted. Default value is true.

- `accountErrorLockoutThresholdInterval=<Long TimeUnit>`

Specifies the time period over which login error counts are tracked for account lockout monitoring. The default value is 10 minutes.

- `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.

- `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.

Admin Restart

Changes to the following Oracle NoSQL Database parameters will result in a Admin restart by the Storage Node Agent:

Admin parameters:

- `adminLogFileCount`
- `adminLogFileLimit`
- `configProperties`
- `javaMiscParams`
- `loggingConfigProps`
- `adminHttpPort`

For example:

```
kv-> plan change-parameters -all-admins -params adminLogFileCount=10
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
Plan Change Admin Params (14)
Owner: null
State:                                INTERRUPTED
Attempt number:                        1
Started:                              2013-08-26 20:12:06 UTC
Ended:                                2013-08-26 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
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
Plan Change Admin Params (14)
State:                                SUCCEEDED
Attempt number:                        1
Started:                              2013-08-26 20:20:18 UTC
Ended:                                2013-08-26 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

Changes to the following Oracle NoSQL Database parameters will result in a Replication Node restart by the Storage Node Agent:

Storage Node parameters:

- `serviceLogFileCount`
- `serviceLogFileLimit`
- `servicePortRange`

Replication Node parameters:

- configProperties
- javaMiscParams
- loggingConfigProps

Removing an Oracle NoSQL Database Deployment

There are no scripts or tools available to completely remove a 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 -Xmx256m -Xms256m \  
-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
```

Once you have performed this procedure on every machine comprising your store, you have completely removed the Oracle NoSQL Database deployment from your hardware.

Fixing Incorrect Storage Node HA Port Ranges

When you initially configured your installation, you defined a range of ports to be used by the nodes when communication among themselves. (You did this in [Installation Configuration \(page 3\)](#).) This range of ports is called the *HA port range*, where *HA* is short hand for "replication."

If you have specified invalid values for the HA Port Range, you are unable to deploy a Replication Node (RN) or a secondary Administration process (Admin) on any misconfigured Storage Node. You discover the problem when you first attempt to deploy a store or a Admin Replica on a faulty Storage Node. You see these indications that the Replication Node did not come up on this Storage Node:

- The Admin displays an error dialog warning that the Replication Node is in the ERROR_RESTARTING state. The Topology tab also shows this state in red, and after a number of retries, it indicates that the Replication Node is in ERROR_NO_RESTART.
- The plan goes into ERROR state, and its detailed history – available by clicking on the plan in the Admin's Plan History tab, or through the CLI's `show plan <planID>` command – shows 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 or CLI shows an alert that contains the same error information from the plan history.
- An examination of the specified .log file or the store-wide log displayed in the Admin's Log tab shows a 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
```

The misconfiguration can be addressed with the following steps. Some steps must be executed on the physical node which hosts the Oracle NoSQL Database Storage Node, while others can be done from any node which can access the Admin or CLI.

1. Using the Admin or CLI, cancel the deploy-store or deploy-admin plan which ran afoul of the misconfiguration.
2. On the Storage Node, kill the existing, misconfigured StorageNodeAgentImpl process and all its ManagedProcesses. You can distinguish them 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, re-create the Storage Node bootstrap configuration file in the KVROOT directory. For directions on how to do this, see [Installation Configuration \(page 3\)](#).
5. On the Storage Node, restart the Storage Node using the following command:

```
java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar restart
```
6. Using the CLI, re-deploy the storage node using the deploy-sn plan.

You can now create and execute a deploy-store or deploy-admin plan, using the same parameters as the initial attempt which uncovered your misconfigured Storage Node.

Chapter 7. 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 the Oracle NoSQL Database Concepts manual.

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.

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 \(page 66\)](#).

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.

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 primary zones, "Manhattan" and "JerseyCity", each deployed in its own physical data center.

Note

For simplicity, this example uses a store with a replication factor of two. 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.

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 -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar \  
runadmin -host jersey1 -port 6000
```

2. Use the `verify configuration` command to confirm the failures:

```
kv-> verify configuration  
Connected to Admin in read-only mode  
Verify: starting verification of store mystore based upon  
topology sequence #207  
200 partitions and 2 storage nodes.  
Time: 2015-05-29 19:41:14 UTC   Version: 12.1.3.4.1  
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  
  
Verify: Admin Status: read-only  
Verify: Zone [name=Manhattan id=zn1 type=PRIMARY]  
RN Status: online:0 offline:1  
Verify: Zone [name=JerseyCity id=zn2 type=PRIMARY]  
RN Status: online:1 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] 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: Storage Node [sn2] on jersey1:6000
Zone: [name=JerseyCity id=zn2 type=PRIMARY]      Status: RUNNING
Ver: 12cR1.3.4.1 2015-05-28 08:27:41 UTC Build id: 287a5a28cea4
Verify:      Admin [admin2]
Status: RUNNING,MASTER (non-authoritative)
Verify:      Rep Node [rg1-rn2]
Status: RUNNING,MASTER (non-authoritative)
sequenceNumber:217 haPort:6003
Verification complete, 4 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
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
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

```

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 -zname JerseyCity
Connected to admin in read-only mode
Repaired admin quorum using admins: [admin2]
```

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 -zname JerseyCity -type primary
      -zname Manhattan -type offline-secondary -wait
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
Pinging components of store mystore based upon topology sequence #208
200 partitions and 2 storage nodes
Time: 2015-07-26 05:00:34 UTC   Version: 12.1.3.4.1
Shard Status: healthy:0 writable-degraded:1 read-only:0 offline:0
Admin Status: writable-degraded
Zone [name=Manhattan id=zn1 type=SECONDARY]
RN Status: online:0 offline:1
```

```

Zone [name=JerseyCity id=zn2 type=PRIMARY]
RN Status: online:1 offline:0
Storage Node [sn1] on nyc1:5000
Zone: [name=Manhattan id=zn1 type=SECONDARY] UNREACHABLE
      Admin [admin1]           Status: UNREACHABLE
      Rep Node [rg1-rn1]       Status: UNREACHABLE
Storage Node [sn2] on jersey1:6000
Zone: [name=JerseyCity id=zn2 type=PRIMARY]
Status: RUNNING
Ver: 12cR1.3.4.1 2015-07-13 10:16:21 UTC Build id: 508d38507fff
      Admin [admin2]           Status: RUNNING,MASTER
      Rep Node [rg1-rn2]
      Status: RUNNING,MASTER sequenceNumber:427 haPort:6011

```

6. 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 -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 -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar runadmin \

```

```
-host jersey1 -port 5000
kv-> plan repair-topology -wait
Executed plan 10, waiting for completion...
Plan 10 ended successfully
```

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
Pinging components of store mystore based upon topology sequence #208
200 partitions and 2 storage nodes
Time: 2015-07-17 17:10:19 UTC   Version: 12.1.3.4.1
Shard Status: healthy:1 writable-degraded:0 read-only:0 offline:0
Admin Status: healthy
Zone [name=Manhattan id=zn1 type=SECONDARY]
RN Status: online:1 offline:0 maxDelayMillis:120000
                                                    maxCatchupTimeSecs:1800

Zone [name=JerseyCity id=zn2 type=PRIMARY]
RN Status: online:1 offline:0
Storage Node [sn1] on nyc1:5000
Zone: [name=Manhattan id=zn1 type=SECONDARY]
Status: RUNNING
Ver: 12cR1.3.4.1 2015-07-13 10:16:21 UTC   Build id: 508d38507fff
      Admin [admin1]           Status: RUNNING,REPLICA
      Rep Node [rg1-rn1]       Status: RUNNING,REPLICA
      sequenceNumber:434 haPort:5011 delayMillis:0 catchupTimeSecs:0
Storage Node [sn2] on jersey1:6000
Zone: [name=JerseyCity id=zn2 type=PRIMARY]
Status: RUNNING
Ver: 12cR1.3.4.1 2015-07-13 10:16:21 UTC   Build id: 508d38507fff
      Admin [admin2]           Status: RUNNING,MASTER
      Rep Node [rg1-rn2]
      Status: RUNNING,MASTER sequenceNumber:434 haPort:6011
```

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-consistent` 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.

```
kv-> topology clone -current -name newTopo
kv-> topology change-zone-type -name newTopo
      -znname Manhattan -type primary
Changed zone type of zn1 to PRIMARY in newTopo
kv-> plan deploy-topology -name newTopo -wait
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
Pinging components of store mystore based upon topology sequence #208
200 partitions and 2 storage nodes
Time: 2015-07-17 17:10:19 UTC   Version: 12.1.3.4.1
Shard Status: healthy:1 writable-degraded:0 read-only:0 offline:0
Admin Status: healthy
Zone [name=Manhattan id=zn1 type=PRIMARY]
RN Status: online:1 offline:0 maxDelayMillis:120000
                                                    maxCatchupTimeSecs:1800
Zone [name=JerseyCity id=zn2 type=PRIMARY]
RN Status: online:1 offline:0
Storage Node [sn1] on nyc1:5000
Zone: [name=Manhattan id=zn1 type=PRIMARY]
Status: RUNNING
Ver: 12cR1.3.4.1 2015-07-13 10:16:21 UTC   Build id: 508d38507fff
```

```
Admin [admin1]           Status: RUNNING,REPLICA
Rep Node [rg1-rn1]       Status: RUNNING,REPLICA
sequenceNumber:434 haPort:5011 delayMillis:0 catchupTimeSecs:0
Storage Node [sn2] on jersey1:6000
Zone: [name=JerseyCity id=zn2 type=PRIMARY]
Status: RUNNING
Ver: 12cR1.3.4.1 2015-07-13 10:16:21 UTC Build id: 508d38507fff
Admin [admin2]           Status: RUNNING,MASTER
Rep Node [rg1-rn2]
Status: RUNNING,MASTER sequenceNumber:434 haPort:6011
```

Chapter 8. Standardized Monitoring Interfaces

Oracle NoSQL Database allows Java Management Extensions (JMX) or Simple Network Management Protocol (SNMP) agents to be optionally available for monitoring, in addition to the native monitoring provided by the Admin CLI and the Admin Console. These agents provide interfaces on each storage node that allow management clients to poll them for information about the status, performance metrics, and operational parameters of the storage node and its managed services, including replication nodes and admin instances.

Both these management agents can also be configured to push notifications about changes in the status of any of the services, and for violations of preset performance limits.

The JMX interface can be enabled in either the Community Edition or the Enterprise Edition. To use SNMP, however, you must have the Enterprise Edition.

The JMX service exposes MBeans for the three types of components. These MBeans are the java interfaces `StorageNodeMBean`, `RepNodeMBean`, and `AdminMBean` in the package `oracle.kv.impl.mgmt.jmx`. For more information about the status reported for each component, see the [javadoc](#) for these interfaces.

The same information that is reported via JMX can also be reported through SNMP. In this case, the information is organized according to the Management Information Base (MIB) named `OracleNosqlMIB`, which is included with the Enterprise Edition, in the file `lib/nosql.mib`.

Simple Network Management Protocol (SNMP) and Java Management Extensions (JMX)

Both the SNMP and JMX agents in NoSQL Database are read-only interfaces and allow you to poll the storage nodes for information about the storage node and about any replication nodes or admins that are hosted on the storage node. The available information includes service status (such as, `RUNNING`, `STOPPED` etc.), operational parameters, and performance metrics.

SNMP and JMX traps/notifications are also delivered for particular events. Notifications are sent for every service status state change; and for violations of performance limits.

Enabling Monitoring

Monitoring can be enabled on a per-storage node basis in two different ways:

In the Bootfile

You can specify that you want to enable JMX or SNMP in the storage node's boot configuration file. Usually, these files are created by using the `makebootconfig` utility, which has the following options to control these features:

- `[-mgmt {snmp|jmx|none} -pollport <snmp poll port>]`
- `-traphost <snmp trap/notification hostname>]`

- `-trapport <snmp trap/notification port>]`

Note

When you specify `-mgmt snmp`, you must also specify `-pollport`. The SNMP agent listens for connections from SNMP management clients on this port. You may also optionally specify `-traphost` and `-trapport` to indicate the destination address for notifications. This would be the hostname and port number of an SNMP management service that is configured to receive notifications at that address.

Note

When you specify `-mgmt jmx`, you do not have to specify `-pollport`. A storage node's JMX agent uses the RMI registry at the same port number as is used for all other RMI services managed by the storage node. (This port number is specified as the `-port` argument to `makebootconfig`.)

By Changing Storage Node Parameters

You can still enable JMX or SNMP after a store is deployed, by changing the storage node parameters `"mgmtClass"`, `"mgmtPollPort"`, `"mgmtTrapHost"`, and `"mgmtTrapPort"`. Similar to configuring via `makebootconfig`, the `"mgmtPollPort"`, `"mgmtClass"`, `"mgmtTrapHost"`, and `"mgmtTrapPort"` are used only for SNMP; and the parameter `"mgmtPollPort"` must be set when enabling SNMP.

The value of the `"mgmtClass"` parameter may be one of the following class names:

- To enable JMX:

```
oracle.kv.impl.mgmt.jmx.JmxAgent
```

- To enable SNMP:

```
oracle.kv.impl.mgmt.snmp.SnmpAgent
```

- To enable neither JMX nor SNMP:

```
oracle.kv.impl.mgmt.NoOpAgent
```

For example, you could issue the following command in the Admin CLI to enable SNMP on a storage node:

```
plan change-parameters -service sn1 -wait -params \  
mgmtClass=oracle.kv.impl.mgmt.snmp.SnmpAgent \  
mgmtPollPort=5002 mgmtTrapHost=192.168.26.42 \  
mgmtTrapPort=32767
```

Note

Only a single implementation of the management agent may be enabled at a particular time. If you enable SNMP on a storage node where JMX is already enabled; the JMX agent shuts down, and the SNMP agent takes its place.

Displaying the NoSQL DB MBeans

To view the NoSQL Database JMX Mbeans in a monitoring tool such as JConsole, connect using the hostname and registry port for each Storage Node that you would like to view.

For example, in this case you would specify localhost:5000 to the JConsole Remote Process connection box in the New Connection tab.

Chapter 9. Integrating Oracle Enterprise Manager (OEM) with Oracle NoSQL Database

The monitoring of a store in Oracle NoSQL Database (Oracle NoSQL Database) can be done through its native command-line interface (CLI). The native web-based Admin Console also does monitoring, and the monitoring data is available through Simple Network Management Protocol (SNMP) and Java Management Extensions (JMX) interfaces - allowing customers to build their own monitoring solutions. For more information, see [Standardized Monitoring Interfaces](#) (page 92).

In this current release, the integration of Oracle's Enterprise Manager (OEM) with Oracle NoSQL Database (Oracle NoSQL Database) provides a graphical management interface tool to discover and monitor a deployed store.

The integration of Oracle NoSQL Database with OEM primarily takes the form of an EM plug-in. The plug-in allows monitoring through Enterprise Manager of NoSQL Database store components, their availability, performance metrics, and operational parameters. The current 12.1.0.9.0 version of the plug-in is compatible with *Oracle Enterprise Manager Cloud Control 12c Release 3 (Version 12.1.0.3.0)*.

Note

For a Storage Node Agent (SNA) to be discovered and monitored, it must be configured for JMX. JMX is not enabled by default. You can tell whether JMX is enabled on a deployed SNA issuing the `show parameters` command and checking the reported value of the `mgmtClass` parameter. If the value is not `oracle.kv.impl.mgmt.jmx.JmxAgent`, then you need to issue the `change-parameters plan` command to enable JMX.

For example:

```
plan change-parameters -service sn1 -wait \  
-params mgmtClass=oracle.kv.impl.mgmt.jmx.JmxAgent
```

For more information, see [Standardized Monitoring Interfaces](#) (page 92).

Also, the EM agent process must have read permission on the contents of \$KVRROOT.

Importing and Deploying the EM Plug-in

Follow the steps below to import and deploy the EM plug-in:

1. Import the file (.opar) into the Enterprise Manager before deploying it. The plug-in is delivered to the user as a file inside the release package: `lib/12.1.0.9.0_oracle.nosql.snab_2000_0.opar`

For more information, see *Importing Plug-In Archives* section (15.4.4.4) of the *Oracle Enterprise Manager Cloud Control Administrator's Guide*, which you can find here: http://docs.oracle.com/cd/E24628_01/doc.121/e24473/plugin_mgr.htm#autold27

2. Copy the .opar file to the host where Oracle Management Service (OMS) is running, and run the command:

Example:

```
$emcli import_update  
-file=/home/guy/12.1.0.9.0_oracle.nosql.snab_2000_0.opar -omslocal
```

3. Deploy the plug-in to the Oracle Management Service (OMS). You can deploy multiple plug-ins to an OMS instance in graphical interface or command line interface. For more information, see *Deploying Plug-Ins to Oracle Management Service* section (15.4.5) of the *Oracle Enterprise Manager Cloud Control Administrator's Guide*, which you can find here: http://docs.oracle.com/cd/E24628_01/doc.121/e24473/plugin_mngr.htm#autold28

CLI Example:

```
$emcli deploy_plugin_on_server -plugin  
=oracle.nosql.snab:12.1.0.9.0 -sys_password=sysmanpass
```

4. Deploy the plug-in to the EM Agents where NoSQL Database components are running. For more information, see step 4, *Deploying Plug-Ins on Oracle Management Agent* section (15.4.7) of the *Oracle Enterprise Manager Cloud Control Administrator's Guide*, which you can find here: http://docs.oracle.com/cd/E24628_01/doc.121/e24473/plugin_mngr.htm#autold33

CLI Example:

```
$emcli deploy_plugin_on_agent -agent_names=agent1.somewhere.com:3872;  
agent2.somewhere.com:3872 -plugin=oracle.nosql.snab:12.1.0.9.0
```

Note

The plugin components are now installed and can be configured.

Adding NoSQL Database Targets

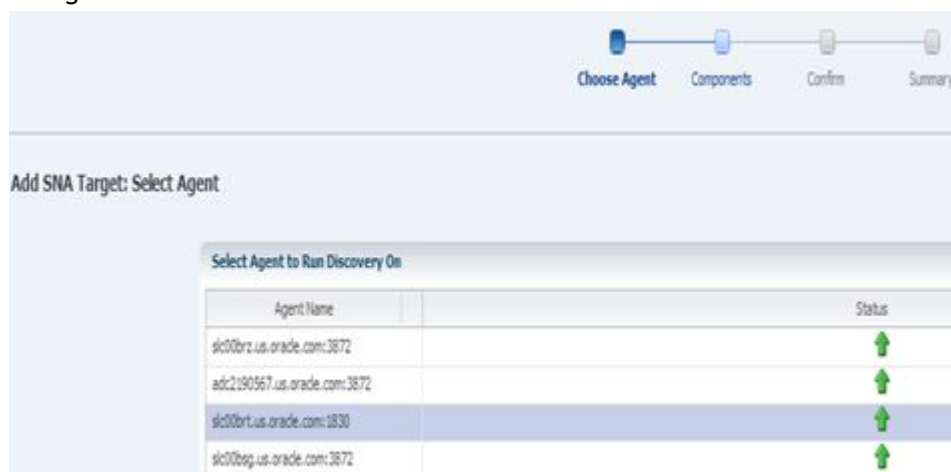
Run the plug-in's discovery program on each host where a Storage Node Agent (SNA) is running, to add the components of a store as monitoring targets.

Follow the steps below to add NoSQL Database targets:

1. Select "Add Targets" from the "Setup" menu, then choose "Add Targets Manually".
2. Select "Add Targets Using Guided Process" on the "Add Targets Manually" page.
3. Select "Discover NoSQL SNA System Targets" in the "Target Types" drop-down list.



4. Click "Add Using Guided Process". This brings up the NoSQL Database Discovery program.
5. Select an agent on which you can run the discovery program. "Choose Agent" (Select Agent to Run Discovery On) in the first page of the program displays a list of all available EM agents.




6. Click "Next". This takes you to the "Components" (Manage NoSQL Database Targets: Select Store Components) page. This shows all the NoSQL Database components that were found on the agent's host. To be found, an SNA must be running when the discovery program runs. The SNA's configuration reports the SNA's managed components, such as Replication Nodes and Admins. For each component, two attributes are listed:
 - Discovered
 - Currently Managed

Each attribute can have a value of "Yes" or "No". For each component found, one of two actions is available:

- add target
- remove target

The action is enabled by means of a check box. The recommended action for a component is shown by the state of its check box.

- If the box is checked, then the action is recommended. The user can override the recommended action by checking or un-checking the box.
- If a component has Discovered = YES, it means that an instance of that component was found on the host.
- If a component has Currently Managed = YES, it means that the component is already configured for monitoring by EM.
- If a component is Discovered and not Currently Managed, then it is a candidate for being added as a target. For such components, the available action is "add target", which is the recommended action.
- If a component is Discovered and Currently Managed, it means that the component has already been discovered and added as a monitoring target. For such components, the available action is "remove target", but the recommended action is to do nothing, because the discovery report is in sync with EM's configuration.
- If a component is Currently Managed and not Discovered, it means that EM is configured to monitor a component that was unexpectedly not found on the agent's host. This could be so because the component no longer resides on the host; or it could reflect a temporary unavailability of the Storage Node Agent. For such components, the recommended action is "remove target".

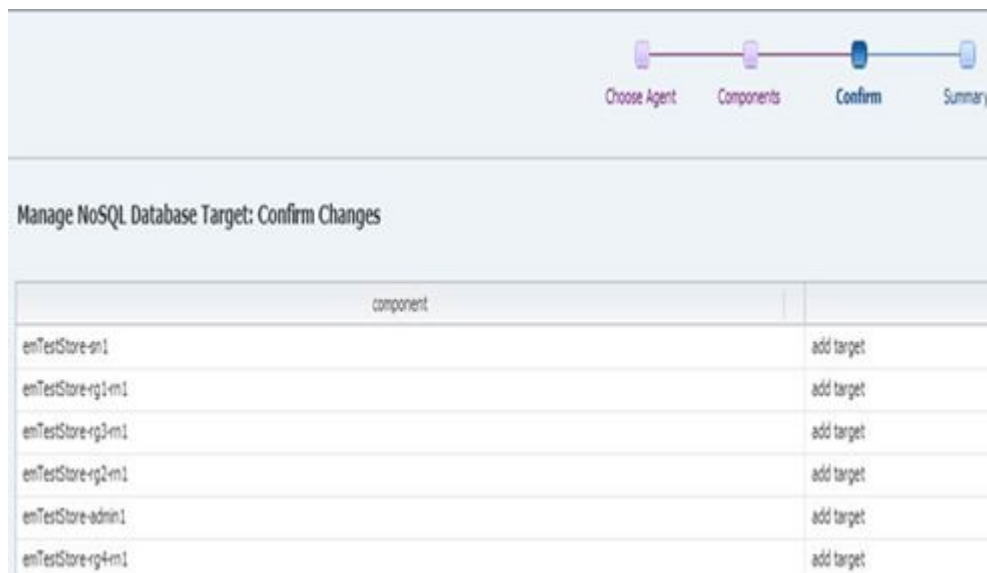


ID	Managed Component	Type	Currently Managed	Discovered	Action
enTestStore-en1	enTestStore-rg1-en1	noad_ima	NO	YES	<input checked="" type="checkbox"/> add target
	enTestStore-rg2-en1	reprobe	NO	YES	<input checked="" type="checkbox"/> add target
	enTestStore-rg3-en1	reprobe	NO	YES	<input checked="" type="checkbox"/> add target
	enTestStore-rg4-en1	reprobe	NO	YES	<input checked="" type="checkbox"/> add target
	enTestStore-admin1	noad_admin	NO	YES	<input checked="" type="checkbox"/> add target
	enTestStore-gk-en1	reprobe	NO	YES	<input checked="" type="checkbox"/> add target
enTestStore	enTestStore-rg1	noad_store	YES	YES	<input type="checkbox"/> remove target
	enTestStore-rg2	noad_ghard	YES	YES	<input type="checkbox"/> remove target
	enTestStore-rg3	noad_ghard	YES	YES	<input type="checkbox"/> remove target
	enTestStore-rg4	noad_ghard	YES	YES	<input type="checkbox"/> remove target

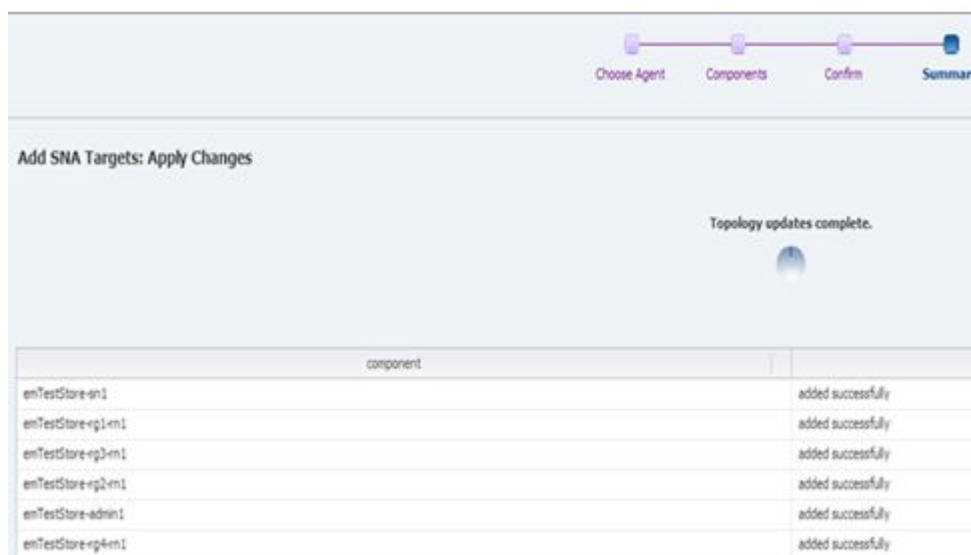
Note

In most cases, the default recommended action is the correct action, and no checkboxes need to be altered.

7. Click "Next" once the desired configuration is set up on the "Components" page. This takes you to the "Confirm" (Manage NoSQL Database Targets: Confirm Changes) page, which shows a list of all the chosen actions.



8. Click "Next" to go to the "Summary" (Add SNA Targets: Apply Changes) page. This shows a report of success or failure of each action.



9. At this point, you may exit Discovery, or you may click on "Choose Agent", near the top of the page, to return to the first page of the program, to re-start and run discovery on a new agent. Once all of the components of a store have been discovered and added, EM's model of the store's topology is complete.

Components of a NoSQL Store

Components of a NoSQL Database Store include the Store itself, Storage Node Agents, Replication Nodes, Admins, and Shards. Of these, Stores and Shards are abstract components that do not correspond to a particular service running on a host. Shards are implied by the existence of Replication Nodes that implement them, and a Store is implied by the existence of the components that belong to it. These components are discovered when components that imply their existence are discovered.

For example, the first time discovery is run on an agent where components belonging to a Store are running, the Store itself will appear as a Discovered and not Managed component to be added. After the Store is added, subsequent runs of discovery on other agents where the existence of the Store is implied will show that the Store is Discovered and Managed, with a recommended action to do nothing. Newly discovered components belonging to the Store will be added as Members of the Store.

Likewise, Shards may appear to be discovered on multiple hosts, but a single Shard need be added only once.

Store Targets

The name of the Store target will be the name of the Store as configured in the NoSQL Database CLI's "configure" command. For more information, see [configure \(page 112\)](#). This name must be unique across all instances of NoSQL Database that will be monitored by a given instance of EM.

Member components of the store have target names made up of their component IDs appended to the store name. For example, in a store named myStore, a Storage Node Agent with an id of "sn1" will have the target name "myStore-sn1", a Replication Node with an id of "rg1-rn1" will have the target name "myStore-rg1-rn1", and so forth. The undeployed StorageNodes will be "UNREGISTERED-hostname-port", for example, "UNREGISTERED-example1.oracle.com-5050". Once the components of a store have been added, you can find the page representing the store by searching for the store name in the "Search Target Name" box in the upper right part of EM's home page. You can also find it via Targets->All Targets or Targets->Systems.

Store Page

Clicking on the Store's name in any of the lists will take you to the Store's target page.

- The page has two large graphs showing the:
 - Average Latency Averaged over all Replication Nodes in the Store
 - Total Throughput for all Replication Nodes



- In the lower right is a list of “Undeployed Storage Node” agents.
- In the lower middle is a list of “Incidents and Problems” related to the store.
- On the left side of the page is the “Store Navigation” panel. This panel presents the topology of the store in three different ways:

- Types

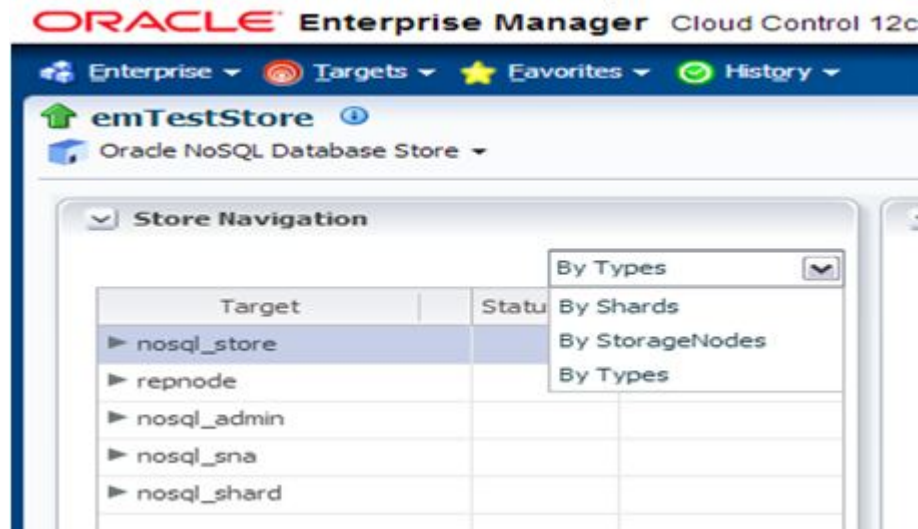
“By Types” groups targets by their target types; so all Replication Nodes are listed together, all Storage nodes are together, and so forth.

- StorageNodes

“By StorageNodes” shows the hierarchy of Store->Storage Node->Replication Node. Each Replication Node is managed by a Storage Node Agent, and always resides on the same host. A Storage Node Agent may manage more than one Replication Node, and this is reflected in the tree layout of the navigation panel.

- Shard

“By Shards” shows the hierarchy of Store->Shard->Replication Node.



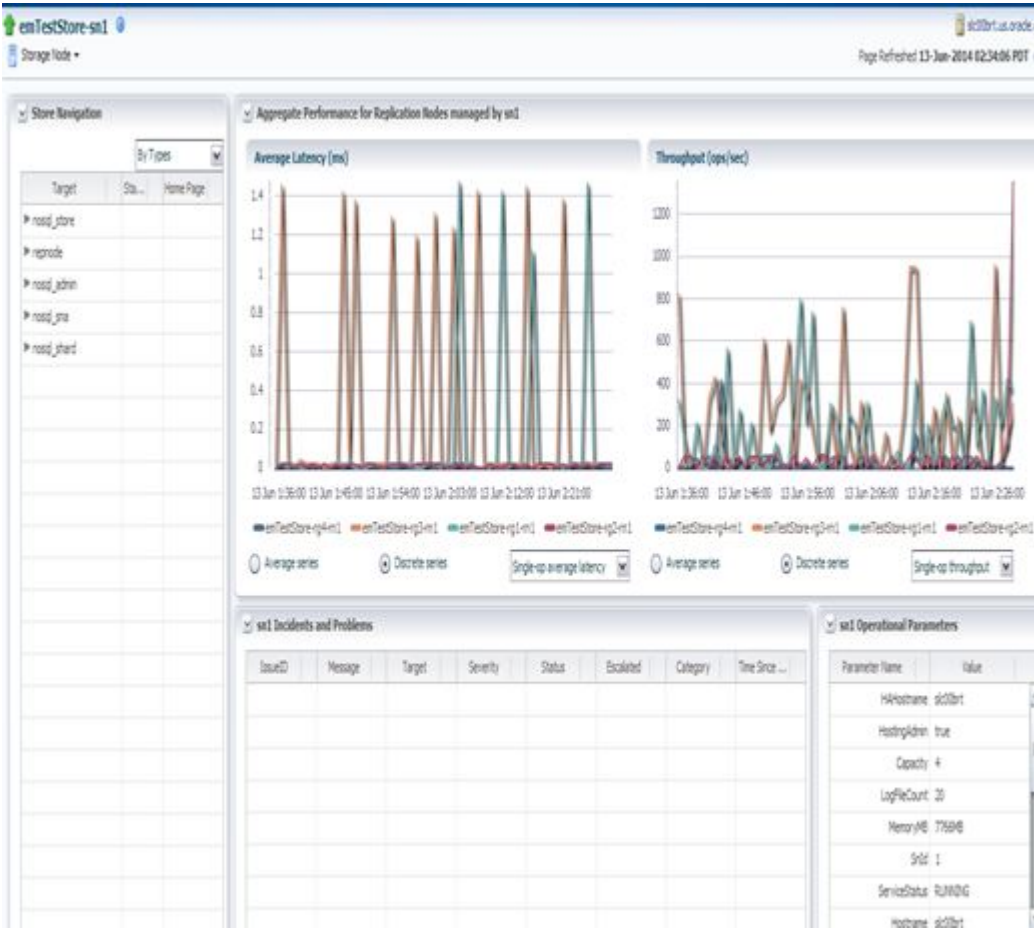
- Each component in the navigation panel has a status "up" or "down", or "unknown" and a link to the target page (labeled "Home Page") for that component. The status can be "unknown" if the targets have yet to be reported for the first time, or if OMS cannot contact the EM Agent.
- The "Store" page, (under menu item Members->Topology) shows the layout of the store as a graph, which is an overview of the "Configuration Topology".



Storage Node Page

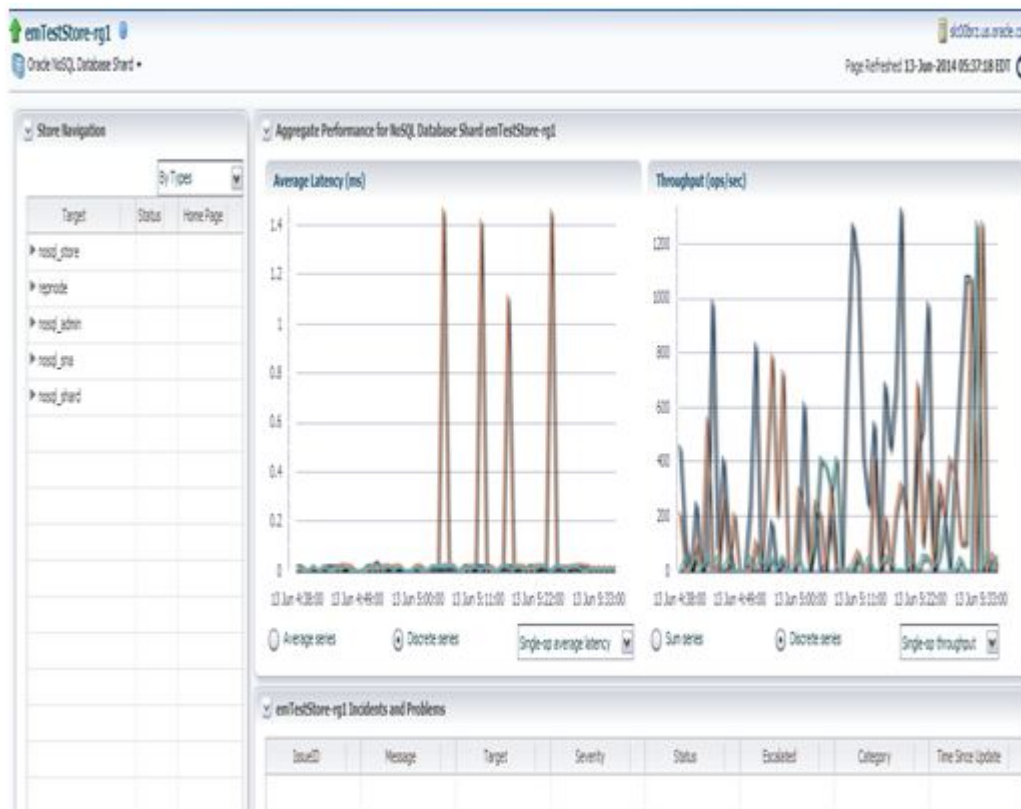
On the “Storage Node” page, you see the same graphs of “Average Latency” and “Throughput”. Here, the graphs show aggregate information for the Replication Nodes that are managed by the Storage Node. The graphs display either discrete series, with one line per Replication Node; or it can combine the series into a single line. The choice is made using the radio buttons at the bottom of the graph.

This page also shows the operational parameters for the Storage Node Agent.



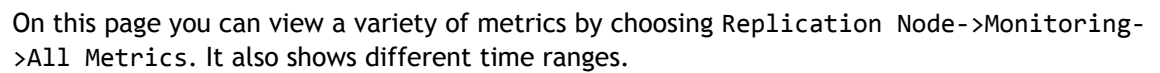
Shard Page

The “Shard” page is similar to the “Storage Node” page. It shows metrics collected from multiple Replication Nodes. These Replication Nodes are the ones that serve as the replication group for the Shard.



Replication Node Page

The “Replication Node” page shows metrics for a single replication node. This page also shows the Replication Node's operational parameters.



Replication Node

emTestStore-rg1-m1 > All Metrics

All Metrics

Search []

MultiCumulativeEnd

ReplNodeID	Last Collected Value	Current Severity	Alert Triggered	Last Collection Timestamp
rg1-m1	Wed Jun 18 00:16:07 PDT 2014	Not Applicable		18-Jun-2014 00:16:16 PDT

ReplNodeID : rg1-m1

Metric Value History

Collection Timestamp	Value
17-Jun-2014 00:26:47 PDT	Tue Jun 17 00:24:48 PDT 2014
17-Jun-2014 00:27:47 PDT	Tue Jun 17 00:25:48 PDT 2014
17-Jun-2014 00:28:47 PDT	Tue Jun 17 00:26:48 PDT 2014
17-Jun-2014 00:29:47 PDT	Tue Jun 17 00:27:48 PDT 2014
17-Jun-2014 00:30:47 PDT	Tue Jun 17 00:28:48 PDT 2014
17-Jun-2014 00:31:47 PDT	Tue Jun 17 00:29:48 PDT 2014
17-Jun-2014 00:32:47 PDT	Tue Jun 17 00:30:48 PDT 2014
17-Jun-2014 00:33:47 PDT	Tue Jun 17 00:31:48 PDT 2014
17-Jun-2014 00:34:47 PDT	Tue Jun 17 00:32:48 PDT 2014
17-Jun-2014 00:35:47 PDT	Tue Jun 17 00:33:48 PDT 2014

From here you can examine and graph any of the numerous metric values that are collected from the Replication Node.

Appendix A. KVStore Command Reference

You can access KVStore commands either through the Command Line Interface (CLI) or through "java -Xmx256m -Xms256m -jar <kvhome>/lib/kvstore.jar <command>". The following sections will describe both the CLI commands and the utility commands accessed through "java -jar".

CLI Commands and Subcommands

The Command Line Interface (CLI) is run interactively or used to run single commands. The general usage to start the CLI is:

```
java -Xmx<javamxvalue> -Xms<javamxvalue> \
-jar KVHOME/lib/kvstore.jar runadmin
-host <hostname> -port <port> [single command and arguments]
```

If you want to run a script file, you can use the "load" command on the command line:

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar runadmin
-host <hostname> -port <port> 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.

This appendix contains the following information on the commands and subcommands:

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.

The aggregate subcommands are:

- [aggregate kv \(page 108\)](#)
- [aggregate table \(page 110\)](#)

aggregate kv

```
aggregate kv [-count] [-sum <field[,field,..]>] [-avg <field[,field,..]>]
              [-key <key>] [-schema <name>]
              [-start <prefixString>] [-end <prefixString>]
```

Performs simple data aggregation operations using the specified key.

where:

- -count

Returns the count of matching records.

- -sum

Returns the sum of the values of matching fields. All records with a specified schema with the named field are matched. Unmatched records are ignored.

- -avg

Returns the average of the values of matching fields. All records with a specified schema with the named field are matched. Unmatched records are ignored.

- -key

Specifies the key (prefix) to use.

- -schema

Specifies the Avro schema to use.

- -start and -end flags

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. -start and -end arguments are inclusive.

For example, a simple count of all records in the store:

```
kv-> aggregate kv -count
count: 33508
```

Sum and average operate on specific field names in matching records which means that only Avro records containing the named fields are used. Sum and average only operate on numeric fields of Avro types INT, LONG, FLOAT, and DOUBLE.

For example, with the following schema:

```
{
  "type" : "record",
  "name" : "Cookie",
  "fields" : [ {
    "name" : "id",
    "type" : "string",
    "default" : ""
  }, {
    "name" : "frequency",
    "type" : "int",
    "default" : 0
  }, {
    "name" : "lastVisit",
    "type" : "string",
    "default" : ""
  }, {
    "name" : "segments",
    "type" : {
      "type" : "array",
      "items" : "string"
    },
    "default" : [ ]
  } ]
}
```

An example of sum on a field named frequency:

```
kv-> aggregate kv -sum frequency -key /visits/charitable_donors/date
sum(frequency): 2068
```

An example of average on a field named frequency:

```
kv -> aggregate kv -avg frequency -key /visits/charitable_donors/date
avg(frequency): 2.494571773220748
```

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:

- -table
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 flag, 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.

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.

The `-zn` and `-znname` options restrict the zones whose replicas need to satisfy the requested consistency requirements. If these options are not specified, all replicas must meet the consistency requirements.

The `-replica-delay-threshold` option specifies the maximum number of milliseconds that a replica may be behind the master and be considered caught up. The default is 1000 milliseconds (1 second).

change-policy

```
change-policy [-dry-run] -params [name=value]*
```

Modifies store-wide policy parameters that apply to not yet deployed services. The parameters to change follow the `-params` flag and are separated by spaces.

Parameter values with embedded spaces must be quoted, for example, `name="value with spaces"`. If `-dry-run` is specified, the new parameters are returned without changing them.

For more information on setting policy parameters, see [Setting Store Wide Policy Parameters \(page 74\)](#).

configure

```
configure -name <storename>
```

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 `'.'`.

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 \(page 112\)](#)
- [connect store \(page 113\)](#)

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.

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:

- -timeout
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.

delete

Encapsulates commands that delete key/value pairs from store or rows from table. The subcommands are as follows:

- [delete kv \(page 113\)](#)
- [delete table \(page 113\)](#)

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

```
delete table -name <name>
[-field <name> -value <value>]*
```

```
[ -field <name> [ -start <value> ] [ -end <value> ] ]
[ -ancestor <name> ] * [ -child <name> ] *
[ -json <string> ] [ -delete-all ]
```

Deletes one or multiple rows from the named table. The table name is a dot-separated name with the format `tableName[.childTableName]*`.

- `-field` and `-value` pairs are used to specify the field values of the primary key, or you can use an empty key to delete all rows from the table.
- The `-field` flag, along with its `-start` and `-end` flags, can be used for restricting the sub range for deletion associated with the parent key.
- `-ancestor` and `-child` flags are used to delete rows from specific ancestor and/or descendant tables as well as the target table.
- `-json` indicates that the key field values are in JSON format.
- `-delete-all` is used to delete all rows in a table.

ddl

Encapsulates operations that manipulate schemas in the store. The subcommands are as follows:

- [ddl add-schema \(page 114\)](#)
- [ddl enable-schema \(page 114\)](#)
- [ddl disable-schema \(page 114\)](#)

For details on managing schema in the store, see [Managing Avro Schema \(page 58\)](#).

ddl add-schema

```
ddl add-schema <-file <file> | -string <schema string>>
[ -evolve ] [ -force ]
```

Adds a new schema or changes (evolves) an existing schema with the same name. Use the `-evolve` flag to indicate that the schema is changing. Use the `-force` flag to add the schema in spite of evolution warnings.

ddl enable-schema

```
ddl enable-schema -name <name>.<ID>
```

Enables an existing, previously disabled schema.

ddl disable-schema

```
ddl disable-schema -name <name>.<ID>
```

Disables an existing schema.

execute

```
execute <statement>
```

Oracle NoSQL Database provides a Data Definition Language (DDL) that you use to form table and index statements. Use the `execute` command to run the specified statement synchronously. The statement must be enclosed in single or double quotes. Before using this command, you need to connect to a store first.

For example:

```
kv-> execute 'CREATE TABLE users (id INTEGER, name STRING,
pets ARRAY(STRING), primary key (id))'
Statement completed successfully
```

```
kv-> execute 'DESCRIBE AS JSON TABLE users'
{
  "type" : "table",
  "name" : "users",
  "comment" : null,
  "shardKey" : [ "id" ],
  "primaryKey" : [ "id" ],
  "fields" : [ {
    "name" : "id",
    "type" : "INTEGER",
    "nullable" : true,
    "default" : null
  }, {
    "name" : "name",
    "type" : "STRING",
    "nullable" : true,
    "default" : null
  }, {
    "name" : "pets",
    "type" : "ARRAY",
    "collection" : {
      "type" : "STRING"
    },
    "nullable" : true,
    "default" : null
  } ]
}
```

For more information on using Data Definition Language (DDL) to perform table operations see [Getting Started with NoSQL Database Table API](#).

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 \(page 116\)](#)
- [get table \(page 118\)](#)

get kv

```
get kv [-key <keyString>] [-json] [-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@gmail.com",
  "phone"  : "408 555 5555"
}
```

- **-json**

Should be specified if the record is JSON.

- **-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 \(page 118\)](#)

get table

```
get table -name <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]
```

Performs a get operation to retrieve row(s) from a named table. The table name is a dot-separated name with the format `tableName[.childTableName]*`.

- `-field` and `-value` pairs are used to specify the field values of the primary key or index key if using 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
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.
```

Use -include-deprecated to show deprecated commands.

For example:

```
kv-> help show -include-deprecated
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, suppose the following commands are collected 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@metalcop.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 ###
```

Then, the script can be run by using the load command in the data command line interface:

Note

A schema must be loaded to the store before this script can successfully run. For more information on adding schema, see "Adding Schema" section in Oracle NoSQL Database Getting Started.

```
> java -Xmx256m -Xms256m \
  -jar KVHOME/lib/kvstore.jar runadmin -host node01 -port 5000 \
  -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.
```

The following schema was previously added to the store:

```
{  
  "type": "record",  
  "name": "ContactInfo",  
  "namespace": "example",  
  "fields": [  
    {"name": "phone", "type": "string", "default": ""},  
    {"name": "email", "type": "string", "default": ""},  
    {"name": "city", "type": "string", "default": ""}  
  ]  
}
```

For more information on using the load command, see [Using a Script to Configure the Store \(page 25\)](#).

logtail

Monitors the store-wide log file until interrupted by an "enter" key press.

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]
```

Pings the runtime components of a store. Components available from the Topology are contacted, as well as Admin services.

where:

- -json

Displays output in JSON format.

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.

The subcommands are as follows:

- [plan add-index \(page 123\)](#)
- [plan add-table \(page 123\)](#)
- [plan cancel \(page 125\)](#)
- [plan change-parameters \(page 125\)](#)
- [plan change-storagedir \(page 126\)](#)
- [plan change-user \(page 126\)](#)
- [plan create-user \(page 126\)](#)
- [plan deploy-admin \(page 127\)](#)
- [plan deploy-datacenter \(page 127\)](#)
- [plan deploy-sn \(page 127\)](#)
- [plan deploy-topology \(page 127\)](#)
- [plan deploy-zone \(page 127\)](#)
- [plan drop-user \(page 128\)](#)
- [plan evolve-table \(page 128\)](#)
- [plan grant \(page 130\)](#)
- [plan execute \(page 129\)](#)
- [plan failover \(page 129\)](#)
- [plan interrupt \(page 130\)](#)
- [plan migrate-sn \(page 130\)](#)

- [plan remove-admin \(page 130\)](#)
- [plan remove-datacenter \(page 131\)](#)
- [plan remove-index \(page 131\)](#)
- [plan remove-sn \(page 131\)](#)
- [plan remove-table \(page 131\)](#)
- [plan remove-zone \(page 132\)](#)
- [plan repair-topology \(page 132\)](#)
- [plan revoke \(page 132\)](#)
- [plan start-service \(page 132\)](#)
- [plan stop-service \(page 132\)](#)
- [plan wait \(page 133\)](#)

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. The table name is a dot-separated name with the format `tableName[childTableName]*`.

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 |
add-schema |
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 the [Introducing Oracle NoSQL Database Tables and Indexes](#).

plan cancel

```
plan cancel -id <plan id> | -last
```

Cancels a plan that is not running. A running plan must be interrupted before it can be canceled.

Use the `-last` option to reference the most recently created plan.

plan change-parameters

```
plan change-parameters -security | -service <id> | -all-rns [-zn <id> | -
znname <name>] | -all-admins [-zn <id> | -znname <name>] [-dry-run]
[-plan-name <name>] [-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.

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 \(page 138\)](#).

For more information on changing parameters in the store, see [Setting Store Parameters \(page 73\)](#).

plan change-storagedir

```
plan change-storagedir -sn <id> -storagedir <path> -add | -remove  
[-plan-name <name>] [-wait] [-noexecute]  
[-force]
```

Adds or removes a storage directory on a Storage Node, for storing a Replication Node.

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 *Oracle NoSQL Database 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 *Oracle NoSQL Database Security Guide*.

plan deploy-admin

```
plan deploy-admin -sn <id> -port <http port> [-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. Its graphical interface listens on the specified port.

For more information on deploying an admin, see [Create an Administration Process on a Specific Host \(page 15\)](#).

plan deploy-datacenter

Deprecated. See [plan deploy-zone \(page 127\)](#) instead.

plan deploy-sn

```
plan deploy-sn -zn <id> | -znname <name> -host <host> -port <port>
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

Deploys the Storage Node at the specified host and port into the specified zone.

For more information on deploying your Storage Nodes, see [Create the Remainder of your Storage Nodes \(page 17\)](#).

plan deploy-topology

```
plan deploy-topology -name <topology name> [-plan-name <name>]
[-wait] [-noexecute] [-force]
```

Deploys the specified topology to the store. This operation can take a while, depending on the size and state of the store.

For more information on deploying a satisfactory topology candidate, see [Deploy the Topology Candidate \(page 37\)](#).

plan deploy-zone

```
plan deploy-zone -name <zone name>
-rf <replication factor>
[-type [primary | secondary]]
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

Deploys the specified zone to the store and creates a primary zone if `-type` is not specified.

For more information on creating a zone, see [Create a Zone \(page 14\)](#).

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 *Oracle NoSQL Database Administrator's Guide*.

plan evolve-table

```
plan evolve-table -name <name>
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

Evolve a table in the store. The table name is a dot-separated 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 \(page 123\)](#).

plan execute

```

plan execute -id <id> | -last
[-plan-name <name>] [-wait] [-noexecute] [-force]

```

Executes a created but not yet executed plan. The plan must have been previously created using the `-noexecute` flag.

Use the `-last` option to reference the most recently created plan.

plan failover

```

plan failover { {-zn <zone-id>|-znname <zone-name>}
-type {primary|offline-secondary} }...
[-plan-name <name>] [-wait] [-noexecute] [-force]

```

Changes zone types to failover to a changed set of primary zones following a failure of primary zones that has resulted in a loss of quorum.

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 (primary/offline-secondary) for the associated zone.

Zones whose new type is primary will be zones that are taking over from failed zones as new primary zones in order to reestablish quorum. For these zones, a quorum of the storage nodes in each shard in the zone needs to 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. None of the command's zone type changes will be performed if these requirements are not met when the command starts.

This command cannot be performed when other plans are in progress for the store. You should cancel or interrupt the plans, before executing this plan.

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 Oracle NoSQL Database Security Guide.

- -user <user_name>

Specifies the user who the role will be granted from.

This command is deprecated. For more information see Grant Role or Privilege in the *Oracle NoSQL Database Administrator's Guide*.

plan interrupt

```
plan interrupt -id <plan id> | -last
```

Interrupts a running plan. An interrupted plan can only be re-executed or canceled. Use -last to reference the most recently created plan.

plan migrate-sn

```
plan migrate-sn -from <id> -to <id> [-admin-port <admin port>]
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

Migrates the services from one Storage Node to another. The old node must not be running.

The -admin-port option is required if the old node hosted an admin service.

Before executing the plan migrate-sn command, you can stop any running old Storage Node by using -java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar stop -root KVR00T.

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-index

```
plan remove-index -name <name> -table <name>
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

Removes an index from a table. The table name is a dot-separated name with the format `tableName[childTableName]*`.

plan remove-datacenter

```
plan remove-datacenter
```

This command is deprecated. See [plan remove-zone \(page 132\)](#) instead.

plan remove-sn

```
plan remove-sn -sn <id> [-plan-name <name>]
[-wait] [-noexecute] [-force]
```

Removes the specified Storage Node from the topology.

This command is useful when removing unused, old Storage Nodes from the store. To do this, see [Replacing a Failed Storage Node \(page 61\)](#).

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.
```

For more information, see [Introducing Oracle NoSQL Database Tables and Indexes](#).

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] [-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.

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 *Oracle NoSQL Database Security Guide*.

- `-user <user_name>`

Specifies the user who the role will be revoked from.

This command is deprecated. For more information see [Revoke Role or Privilege](#) in the *Oracle NoSQL Database Administrator's Guide*.

plan start-service

```
plan start-service {-service <id> | -all-rns [-zn <id> | -znname <name>]} |
-zn <id> | -znname <name>}
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

Starts the specified service(s).

plan stop-service

```
plan stop-service {-service <id> | -all-rns [-zn <id> | -znname <name>]} |
```

```
-zn <id> | -znname <name>}  
[-plan-name <name>] [-wait] [-noexecute] [-force]
```

Stops the specified service(s).

Use this command to stop any affected services 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.

This command is useful during disk replacement process. Use the `plan stop-service` command to stop the affected service prior removing the failed disk. For more information, see [Replacing a Failed Disk \(page 64\)](#).

plan wait

```
plan wait -id <id> | -last [-seconds <timeout in seconds>]
```

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.

pool

Encapsulates commands that manipulates Storage Node pools, which are used for resource allocations. The subcommands are as follows:

- [pool create \(page 133\)](#)
- [pool join \(page 133\)](#)
- [pool remove \(page 133\)](#)

pool create

```
pool create -name <name>
```

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 \(page 16\)](#).

pool join

```
pool join -name <name> [-service] <snX>*
```

Adds Storage Nodes to an existing Storage Node pool.

pool remove

```
pool remove -name <name>
```

Removes a Storage Node pool.

put

Encapsulates commands that put key/value pairs to the store or put rows to a table. The subcommands are as follows:

- [put kv \(page 134\)](#)
- [put table \(page 135\)](#)

put kv

```
put kv -key <keyString> -value <valueString> [-file]
[-hex | -json <schemaName>] [-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 -key /Smith/Bob/-/email -value
"{\"id\": 1, \"email\": \"bob.smith@gmail.com\"}" -json schema.EmailInfo
```

For example, a key containing only a major key path:

```
kv-> put -key /Smith/Bob -value "{\"name\": \"bob.smith\", \"age\": 20, \"phone\": \"408 555 5555\", \"email\": \"bob.smith@gmail.com\"}" -json schema.UserInfo
```

- **-value <valueString>**

If neither **-json** or **-file** is specified, the <valueString> is treated as a raw bytes array.

For example:

```
kv-> put -key /Smith/Bob/-/phonenummer -value "408 555 5555"
```

Note

The mapping of the raw arrays to data structures (serialization and deserialization) is left entirely to the application. This is not the recommended approach. Instead, you should use Avro even for very simple values.

If used with **-json** to specify a Json string, the <valueString> should be encapsulated in quotation marks, and its internal field name and value with string type should also be encapsulated by string quote characters.

For example:


```
kv-> put -key /Smith/John/-/email -value
"{\"id\": 1, \"email\": \"john.smith@gmail.com\"}" -json schema.EmailInfo
```

- **-file**

Indicates that the value is obtained from a file. The file to use is identified by the value parameter.

For example:

```
kv-> put -key /Smith/Bob -value ./smith-bob-info.txt
-file -json schema.UserInfo
```

- **-hex**

Indicates that the value is a BinHex encoded byte value with base64 encoding.

- **-json<schemaName>**

Indicates that the value is a JSON string. Can be specified along with **-file**.

- **-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

```
put table -name <name> [if-absent | -if-present ]
[-json <string>] [-file <file>] [-exact] [-update]
```

Put a row into the named table. The table name is a dot-separated name with the format table[.childTableName]*.

where:

- **-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**

Can be used to load JSON strings from a file.

- -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>}...
```

Repairs admin quorum by reducing membership of the admin group to the admins in the specified zones or the specific admins listed. This command should be used when attempting to recover from a failure that has resulted in a loss of admin quorum. This command can result in data loss.

After obtaining a working admin by calling this command, you should use 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

This command can be repeated if a temporary network or component failure causes the 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 \(page 137\)](#)
- [show datacenters \(page 137\)](#)
- [show events \(page 137\)](#)
- [show faults \(page 138\)](#)
- [show indexes \(page 138\)](#)
- [show parameters \(page 138\)](#)

- [show perf \(page 138\)](#)
- [show plans \(page 138\)](#)
- [show pools \(page 139\)](#)
- [show schemas \(page 139\)](#)
- [show snapshots \(page 139\)](#)
- [show tables \(page 139\)](#)
- [show topology \(page 139\)](#)
- [show upgrade-order \(page 140\)](#)
- [show users \(page 140\)](#)
- [show versions \(page 140\)](#)
- [show zones \(page 140\)](#)

show admins

```
show admins
```

Displays basic information about Admin services.

show datacenters

```
show datacenters
```

Deprecated. See [show zones \(page 140\)](#) instead.

show events

```
show events [-id <id>] | [-from <date>] [-to <date> ]  
[-type <stat | log | perf>]
```

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 \(page 72\)](#).

show faults

```
show faults [-last] [-command <command index>]
```

Displays faulting commands. By default all available faulting commands are displayed. Individual fault details can be displayed using the `-last` and `-command` flags.

show indexes

```
show indexes [-table <name>] [-name <name>]
```

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 \(page 123\)](#).

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 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 \(page 11\)](#).

show pools

```
show pools
```

Lists the Storage Node pools.

show schemas

```
show schemas [-disabled] | [-name <name>]
```

Displays schema details of the named schema or a list of schemas registered with the store.

Use the `-name` option to specify the name of the schema you want to check if it is currently enabled in the store.

Use the `-disabled` option to see all schemas, including those which are currently disabled.

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 tables

```
show tables -name <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 \(page 123\)](#) and [plan evolve-table \(page 128\)](#)

show topology

```
show topology [-zn] [-rn] [-sn] [-store] [-status] [-perf]
```

Displays the current, deployed topology. By default it shows the entire topology. The optional flags restrict the display to one or more of Zones, Replication Nodes, Storage Nodes and store name, or specify service status or performance.

With this command you can obtain the ID of the zone to which Storage Nodes can be deployed to.

show upgrade-order

```
show upgrade-order
```

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.

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
```

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
```

show zones

```
show zones [-zn <id>] | -znname <name>
```

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 or secondary zone.

snapshot

Encapsulates commands that create and delete snapshots, which are used for backup and restore. The subcommands are as follows:

- [snapshot create \(page 141\)](#)
- [snapshot remove \(page 141\)](#)

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 \(page 51\)](#).

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 \(page 54\)](#) or [Restoring Directly from a Snapshot \(page 57\)](#).

table

Deprecated with exception of table size. See [execute \(page 114\)](#) instead. For more information, see [table size \(page 141\)](#)

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.

It 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:

Create table user (id integer, address string, zip_code string) and idx1 on user (zip_code)

```
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
Table	1,024,690	Internal nodes only
	1,024,690	Internal nodes and record versions
	1,024,690	Internal nodes and leaf nodes
idx1	413,728	Internal nodes only
	413,728	Internal nodes and record versions
	413,728	Internal nodes and leaf nodes
Total	1,438,418	Internal nodes only
	1,438,418	Internal nodes and record versions
	1,438,418	Internal nodes and leaf nodes

For more information, see the DbCacheSize javadoc: http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/util/DbCacheSize.html

Note

The cache size are calculated as follows:

- Cache size of table

```
java -jar KVHOME/lib/je.jar DbCacheSize -records
10000 key 8 -data 47 -outputproperties -replicated
<JE properties...>
```

The parameters are as follows:

- 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 as follows:

- 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: http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/util/DbCacheSize.html

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: http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/util/DbCacheSize.html

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 \(page 127\)](#). The subcommands are as follows:

- [topology change-repfactor \(page 146\)](#)
- [topology change-zone-type \(page 146\)](#)
- [topology clone \(page 146\)](#)
- [topology create \(page 146\)](#)
- [topology delete \(page 147\)](#)
- [topology list \(page 147\)](#)
- [topology preview \(page 147\)](#)
- [topology rebalance \(page 147\)](#)
- [topology redistribute \(page 147\)](#)

- [topology validate \(page 147\)](#)
- [topology view \(page 148\)](#)

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 not be decreased at this time.

For more information on modifying the replication factor, see [Increase Replication Factor \(page 35\)](#).

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 create

```
topology create -name <candidate name> -pool <pool name>  
-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.

For more information on creating the first topology candidate, see [Make the Topology Candidate \(page 32\)](#).

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>]
```

Modifies the named topology to create a balanced topology. If the optional -zn flag is used, only Storage Nodes from the specified zone are used for the operation.

For more information on balancing a non-compliant topology, see [Balance a Non-Compliant Topology \(page 35\)](#).

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 \(page 33\)](#).

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 \(page 36\)](#).

topology view

```
topology view -name <name>
```

Displays details of the specified topology.

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 that check various parameters of the store. The subcommands are as follows:

- [verify configuration \(page 148\)](#)
- [verify prerequisite \(page 148\)](#)
- [verify upgrade \(page 149\)](#)

verify configuration

```
verify configuration [-silent] [-json]
```

Verifies the store configuration by iterating the components and checking their state against that expected in the Admin database. This call may take a while on a large store.

The `-json` option specifies that output should be displayed in JSON format.

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. This option has no effect when the `-json` option is specified.

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.

Utility commands

You can access utility commands through "java -jar".

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar <command>
```

This appendix contains the following information on the commands:

makebootconfig

```
java -Xmx256m -Xms256m  
-jar KVHOME/lib/kvstore.jar makebootconfig [-verbose]  
-root <rootDirectory> -host <hostname> -harange <startPort,endPort>  
-store-security [none | configure | enable]  
-port <port> [-admin <adminPort>] [-runadmin] [-config <configFile>]  
[-storagedir <directory path>] [-capacity <n_rep_nodes>]  
[-num_cpus <ncpus>] [-memory_mb <memory_mb>]  
[-servicerange <startPort,endPort>]  
[-hahost <haHostname>]  
[-secdir <security dir>] [-pwmgr {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 {snmp|jmx|none}] [-pollport <snmp poll port>]
[-traphost <snmp trap/notification hostname>]
[-trapport <snmp trap/notification port>]
[-dns-cachettl <time in sec>]
[-force]

```

where:

- `-admin <adminPort>` The port on which the web-based Admin Console is contacted. It only needs to be free on the node which runs the admin process. The value defaults to "0", which means that there will be no admin on this node.
- `-capacity <n_rep_nodes>` The total number of Replication Nodes a Storage Node can support. The value defaults to "1".
- `-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 the *Oracle NoSQL Database 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.
- -hahostname <hostname> Can be used to specify a separate network interface for store replication traffic. This defaults to the hostname specified using the -host flag.

- `-harange <startPort,endPort>` A range of free ports which the Replication Nodes use to communicate among themselves. These ports should be sequential and there must be at least as many as the specified capacity for this node.
- `-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.
- `-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 that is available in the machine. If the value is 0, the store will attempt 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".
- `-num_cpus <ncpus>` The total number of processors on the machine available to the Replication Nodes. If the value is 0, the system will attempt to query the Storage Node to determine the number of processors on the machine. This value defaults to "0".
- `-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, etc.

where `-pwdmgr` can have 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 KVRTX 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." The value varies with the capacity of the Storage Node.
- `-storagedir <path>` Specify a path to the directory to be used for a Replication Node. This flag may be used more than once in the command to specify multiple storage directories, but the number should not exceed the capacity for the node. If no storage directory is specified, Replication Nodes use a directory under the root directory.

The use of the `-storagedir` argument must be coordinated with the use of the `capacity` argument. For example, if your Storage Node hosts four disks, you would specify a capacity of four and have four `-storagedir` arguments.

- `-store-security [none | configure | enable]` Specifies if security will be used or not. If `-store-security none` is specified, no security will be used. If `-store-security configure` is specified, security will be used and the security configuration utility will be invoked as part of the `makebootconfig` process. 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.

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 must not already exist. You only need to specify the `admin` option (the Admin Console port) on the node which hosts the initial Oracle NoSQL Database administration processes. To create the initial "boot config" file used to configure the installation see [Installation Configuration \(page 3\)](#).

OracleNoSQL Database allows Java Management Extensions (JMX) or Simple Network Management Protocol (SNMP) agents to be optionally available for monitoring, in addition to the native monitoring provided by the Admin CLI and the Admin Console. In order to enable JMX or SNMP in the storage node's boot configuration file, you can use the `-mgmt`, `-pollport`, `-traphost` and `-trapport` options. See [Standardized Monitoring Interfaces \(page 92\)](#) for more information.

start

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar start
[-disable-services] [-verbose]
-root <rootDirectory> [-config <bootstrapFileName>]
```

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 \(page 66\)](#)

stop

```
java -Xmx256m -Xms256m \
-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 \(page 66\)](#)

restart

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar restart
[-disable-services] [-verbose]
-root <rootDirectory> [-config <bootstrapFileName>]
```

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 \(page 66\)](#)

runadmin

```
java -Xmx<javamxvalue> -Xms<javamxvalue> \
-jar KVHOME/lib/kvstore.jar runadmin
-host <hostname> -port <port> [-store <storeName> ]
[-admin-host <adminHost> -admin-port <adminPort> ]
[-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>]
[single command and arguments]
```

Runs a utility which provides a command line interface (CLI). It is used to perform store configuration.

Use the `timeout`, `consistency` and `durability` flags to override the default connect configuration.

where:

- **-timeout**
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.

load

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar load [-verbose]  
-source <backupDir> -host <hostname> -port <port>  
-store <storeName> -username <user> -security <security-file-path>  
[-load-admin] [-force] [-status <pathToFile>]
```

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.

- **-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 <backupDir>** identifies the on-disk location where the snapshot data is stored.
- **-status <pathToFile>** is an optional parameter that causes the status of the load operation to be saved in the named location on the local machine.
- **-store <storeName>** identifies the name of the store.
- **-username <user>** identifies the name of the user to login to the secured store.

Program used to restore a store from a previously created snapshot. By using this tool, you can restore the store to any topology, not just the one that was in use when the snapshot was

created. Load should be used only to restore to a new, empty store. Do not use this with an existing store because it only writes records if they do not already exist. See [Using the Load Program \(page 54\)](#) for more information.

ping

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar ping [-verbose] [-json]  
-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. By using the ping command, you can ensure that the Oracle NoSQL Database client library can contact the Oracle NoSQL Database Storage Agent(SNA). You can use the -host option to check an SNA on a remote host.

Specify the -helper-hosts flag as an alternative to the existing -host and -port flags. If multiple helper hosts are provided, the ping utility will have 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. The new -helper-hosts flag can be specified as a comma-separated list of hosts and ports. For example:

```
java -jar kvstore.jar ping -helper-hosts hst1:5000,hst2:5100
```

Ping generates an exit code which acts as a quick summary of the results. The exit code is returned both as a process exit code, if ping is called as a standalone utility, and as part of the JSON output. The exit code is meant to direct the caller to perform the appropriate follow-on action. The values, defined in the Ping.ExitCode enum, are:

- 0 (EXIT_OK)

All services in the store could be located and are in a known, good state (for example, RUNNING).

- 1 (EXIT_OPERATIONAL)

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.

- 2 (EXIT_NO_ADMIN_QUORUM)

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.

- 3 (EXIT_NO_SHARD_QUORUM)

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.

- 100 (EXIT_USAGE)

A usage error.

- 101 (EXIT_TOPOLOGY_FAILURE)

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 host and port or -helper-hosts list are not part of a store).

- 102 (EXIT_UNEXPECTED)

The utility has experienced an unexpected error.

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.

If the -json flag is specified, then output will be displayed in JSON format.

version

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar version
```

Prints version.

generateconfig

```
java -Xmx256m -Xms256m \  
-jar KVHOME/lib/kvstore.jar generateconfig [-verbose]  
-host <hostname> -port <port> -sn <StorageNodeId> -target <zipfile>
```

Generates configuration files for the specified storage node. The generateconfig command creates the target zipfile which contains the required configuration to re-create the storage node. The top-level directory in the zipfile is the store's root directory.

help

```
java -Xmx256m -Xms256m \  
-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.

Appendix B. Initial Capacity Planning

To deploy a store, the user 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 upon the 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 disk throughput. The amount of physical memory required by each machine and its network throughput capacity is also estimated as part of this step.
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.

There is an accompanying spreadsheet that should be used in the planning process. You can find the spreadsheet in your Oracle NoSQL Database distribution here: <KVHOME>/doc/misc/InitialCapacityPlanning.xls.

The sections in this appendix correspond to named sections in the spreadsheet. Column A of the spreadsheet lists cell names that are associated with the values in column B. Cell names in red represent values that must be provided as input. Column C describes the value or the computation associated with the value in column B. The sections: [Application Characteristics \(page 159\)](#), [Hardware Characteristics \(page 160\)](#), and [Machine Physical Memory \(page 161\)](#) contain required inputs. Green cell names denote optional inputs; the default values supplied in the spreadsheet should be adequate for most estimates. All other cells are computed by the spreadsheet using the formulas described below.

After filling in the required inputs, the cell *StoreMachines* value will tell you how many Storage Nodes should be available in the storage node pool. The *StorePartitions* value will tell you how many partitions should be specified when creating the store.

Please keep in mind that the computations below yield estimates. The underlying model used as a basis for the estimation makes simplifying assumptions since it's hard to come up with a simple single underlying model that will work well under a wide range of application requirements. So these estimates should only be used as the basis for an initial starting point and refined under 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. 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 Factors* 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 \(page 162\)](#) 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 \(page 76\)](#).
- 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 \(page 161\)](#) 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 \(page 165\)](#) 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 \(page 3\)](#) and [Storage Node Parameters \(page 76\)](#) 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 record versions 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 -Xmx256m -Xms256m \
-d64 -XX:+UseCompressedOops -jar je.jar DbCacheSize \
-key 12 -data 1000 -records 100000000 -replicated \
-je.rep.preserveRecordVersion true
```

=== Environment Cache Overhead ===

3,163,085 minimum bytes

To account for JE daemon operation and record locks,
a significantly larger amount is needed in practice.

=== Database Cache Size ===

Number of Bytes	Description
3,558,319,072	Internal nodes only
4,322,364,352	Internal nodes and record versions
108,969,601,408	Internal nodes and leaf nodes

Please make note of the following jvm arguments (they have a special meaning when supplied to DbCacheSize):

1. The -d64 argument is used to ensure a 64 bit jvm environment that is capable of supporting heap size greater than 4GB.
2. The -XX:+UseCompressedOops is used to ensure use of more efficient 32 bit pointers in the 62 bit environment thus permitting better utilization of the JE cache.
3. The -replicated is used to ensure use of a ReplicatedEnvironment.
4. The -je.rep.preserveRecordVersion allows using record versions in operations such as "put if version", "delete if version", etc. This feature performs best when the cache is sized large enough to hold the record versions.

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.

Because -je.rep.preserveRecordVersion is specified, the number you want is in the Database Cache Size section of the output, under the "Number of Bytes" column in the "Internal nodes and record versions" row. The output indicates that a cache size of 4.4 GB is sufficient to hold all the internal nodes and record versions 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 FS cache or the disk.

Note

If `-je.rep.preserveRecordVersion` is not specified, the number you want is in the Database Cache Size section of the output, under the "Number of Bytes" column in the "Internal nodes only" row. The output indicates that a cache size of 3.6GB is sufficient to hold all the internal nodes.

For more information on using the `DbCacheSize` utility, see this Javadoc page: http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/util/DbCacheSize.html. 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 \(page 160\)](#), 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:

$$((\text{OpsPerShardPerSec} * \text{ReadOpsPercent}) / \text{RF}) * (\text{AvgKeySize} + \text{ReadRequestOverheadBytes}) * \text{DisksPerMachine}$$

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:

$$((\text{OpsPerShardPerSec} * \text{ReadOpsPercent}) / \text{RF}) * (\text{AvgKeySize} + \text{AvgValueSize}) * \text{DisksPerMachine}$$

2. The bytes sent out as replication traffic by the masters on the machine expressed as:

$$(\text{OpsPerShardPerSec} * (1 - \text{ReadOpsPercent})) * (\text{AvgKeySize} + \text{AvgValueSize}) * (\text{RF} - 1) * \text{MastersOnMachine}$$

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 [Transform the Topology Candidate \(page 33\)](#).
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:

$$\text{MaxKVPairs} / \text{MaxKVPairsPerShard}$$

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
```

Appendix C. 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 subsystems 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 to the `makebootconfig` command to specify the interface to be used by HA as in the example below:

```
java -Xmx256m -Xms256m \
-jar kvstore.jar makebootconfig -root /disk1/kvroot \
-host sn10.example.com -port 5000 -h.arange 5010,5020 \
-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 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. http://docs.oracle.com/cd/E37670_01/E37355/html/ol_about_uek.html
2. <http://lwn.net/Articles/361440/>

Managing the Off-heap Cache

Each Replication Node in the store divides the memory available to it between memory containing store objects, memory used by the host operation system (which includes the file system cache), and the in-memory cache used to contain frequently accessed store data. The amount of memory used by the cache is configurable using the `rnHeapPercent` parameter. This is expressed as a percentage of the total memory available to the Replication Node (which is, in turn, determined by the amount of memory made available to the Java VM). By default, 70% of the memory available to the Replication Node is reserved for the in-memory cache.

However, this can and probably will fill up. When that happens, objects are evicted from the cache based on a Least-Recently-Used (LRU) algorithm which guarantees that the most commonly accessed objects will remain in the cache.

It is possible to configure your Replication Node so that the in-memory cache is very large, and therefore is less likely to overflow. However, several drawbacks occur when the cache becomes too large. The most important of these is that Java Garbage Collection performance can decrease, which will have an adverse impact on your Replication Node's overall performance. *Therefore, it is in general a good idea to keep your heap size below 32 GB.* Doing so limits your cache size to about 27.2 GB.

Note

Be aware that Oracle NoSQL Database limits your maximum heap size to a default value of 32 GB so as to prevent accidentally creating an excessively large cache. This value is managed using the `rnHeapMaxMB`, which is described in greater detail later in this section.

When your Replication Node's heap is sized lower, it is possible to configure the system such that any memory unused by the heap will be used for the operating system and file system cache. A large file system cache has performance benefits, but it also has drawbacks:

1. There is a significant redundancy between the main cache and the file system cache because all data and Btree information that is logged (written) by the Replication Node appears in the file system and may also appear in the main cache.
2. It is not possible for dirty Btree information to be placed in the file system cache without logging it. This logging may be otherwise unnecessary, and the logging creates additional work for internal cleaner threads.

To avoid the problems caused by excessively large in-memory and file system caches, your Replication Node is by default configured to use an *off-heap cache*. The off-heap cache is used to hold record data and Btree nodes when these are evicted from the "main cache" because it overflows. When the off-heap cache overflows, eviction occurs according to the same LRU algorithm used by the main cache.

Note

You should engage in performance testing before modifying the caches in use by your production store.

Configuring the Off-heap Cache

You do not directly control the amount of memory available to the off-heap cache. Instead, you make memory available to it mainly by limiting the amount of memory available to the operating system. You might also need to control the size of the Java heap, which in turn controls the size of the in-memory cache. Any Replication Node memory that is left over once the heap and operating system requirements are met is used for the off-heap cache. If no memory is left over, the off-heap cache is not used. This is the only way to turn off the off-heap cache.

The parameters you use to (indirectly) configure the off-heap cache are:

1. `systemPercent`

This defines the percentage of memory available to the Storage Node that is available for operating system use *after* the heap requirement is subtracted. By default, this value is 10%. Configuring this value to a number that is greater than 10% but less than 100% might leave room for the off-heap cache (depending on the memory on your Storage Node and the value chosen for this parameter). If room is available for the off-heap cache, it will be turned on.

For most production systems, if you are using the off-heap cache, then the default value of 10% for this parameter should be sufficient.

2. rnHeapMaxMB

This is the maximum amount of memory available for the Java heap. (The heap is where the in-memory cache is contained.) The size of your heap will be the lesser of this number, or the size arrived at using the rnHeapPercent parameter value:

$$\text{total SN memory} * \text{rnHeapPercent} = \text{Heap Size}$$

rnHeapPercent is by default 85%. So if your Storage Node has 32GB memory, then the heap size will be $32 * 0.85 = 27.2$ GB. However, if you set your rnHeapMaxMB to 25,600 (25*1024), then your heap size will be 25GB instead.

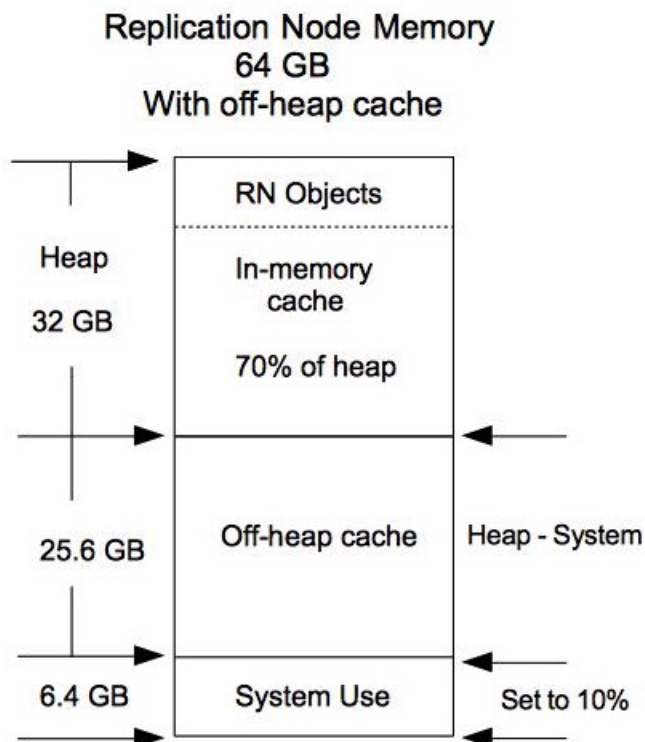
Remember that the heap size is not the same as the in-memory cache size. The in-memory cache size is expressed as a percentage of the heap size. By default, this is 70% (configurable using the rnCachePercent parameter). So if your heap size is 27.2 GB, then your in-memory cache size ends up being 19.04 GB ($27.2 * 0.7$).

Note

Regardless of the values provided to the parameters described here, and the actual memory available for the heap, your heap size is limited to a maximum size of 32 GB.

For example, if your Storage Node has 64GB of memory, and just one Replication Node, then by default:

- The heap size is 32GB. ($64 * .85 = 54.4$ which is greater than the 32GB maximum.)
- The in-memory cache size is 22.4 GB ($32 * 0.7$).
- A system memory size of 6.4 GB. ($64 * .1$)
- An off-heap cache size of 25.6 GB. This is the amount of memory left over after the heap and system requirements have been met.

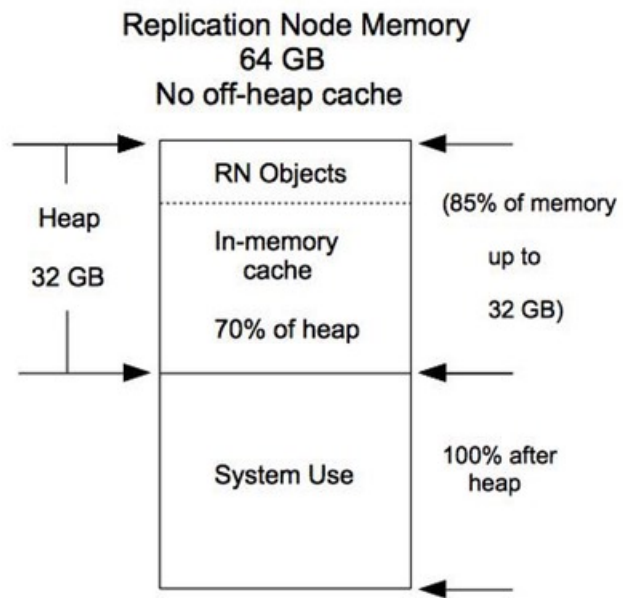


If you want to eliminate the off-heap cache, then set system memory to 100%. Do this for each Storage Node in your store. Be aware that doing this causes your Storage Node to restart:

```
kv-> change-policy -params systemPercent=100 change-parameters \
-svc service sn1 -wait -params systemPercent=100
Executed plan 5, waiting for completion...
Plan 5 ended successfully
kv->
```

This yields:

- A heap size of 32 GB and in-memory cache size of 22.4 GB. Nothing has changed from the default configuration.
- The system memory is 32 GB. System memory is 100% of what is left over when the heap is taken away. 64 GB (total available) - 32 GB (heap size) = 32 GB for the operating system and file system cache.



Appendix D. 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.

Appendix E. 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 -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:

- -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.

- -logfiles

Specified to gather log files of KVStore and pack them up into a compressed file. The log files are all files in the root directory of SNAs(excluding database files jdb files). 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.

- -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.

- -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.

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.

Key Distribution Statistics Utility

If enabled, your KVStore automatically gathers key distribution statistics.

To enable statistics gathering and create the four tables needed to store the key distribution data, perform the following steps:

1. Set the `rnStatisticsEnabled` parameter to true to enable statistics gathering:

```
plan change-parameters -wait -all-rns  
-params "rnStatisticsEnabled=true"
```

2. Run the `CreateStatisticTables` script located in the `lib` directory of the release package to create the four tables needed to store the key distribution data. The file, `CreateStatisticTables.kvs`, would then contain content like this:

```
#Create table "TableStatsPartition".  
execute  
"create table if not exists TableStatsPartition(  
  tableName string,  
  partitionId integer,  
  shardId integer,  
  count long,  
  avgKeySize integer,  
  primary key (tableName, partitionId)  
)";  
  
#Create table "TableStatsIndex".  
execute  
"create table if not exists TableStatsIndex(  
  tableName string,  
  indexName string,  
  shardId integer,  
  count long,  
  avgKeySize integer,  
  primary key (tableName, indexName, shardId)  
)";  
  
#Create table "PartitionStatsLease".  
execute  
"create table if not exists PartitionStatsLease(  
  partitionId integer,  
  leasingRN string,  
  leaseExpiry string,  
  lastUpdated string,  
  primary key (partitionId)  
)";
```

```
#Create table "IndexStatsLease".
execute
"create table if not exists IndexStatsLease(
tableName string,
indexName string,
shardId integer,
leasingRN string,
leaseExpiry string,
lastUpdated string,
primary key (tableName, indexName, shardId)
)";
```

TableStatsPartition stores the distribution of primary keys and table TableStatsIndex stores the distribution of index keys. All four tables are forbidden to modify, delete and insert.

You can query any of the four tables to get key distribution data or review the gathering process.

To modify the behavior of the utility, use the following parameters:

- rnStatisticsEnabled

Used to enable or disable the statistics gathering. The default value is true, which enables statistics gathering.

- rnStatisticsGatherInterval

Specifies the interval time between updates of table level statistics. The default value is 24 hours.

- rnStatisticsLeaseDuration

Specifies the duration of a lease. A Replication Node must hold a lease associated with a partition or a shard-index before it can update the associated statistics. The default value is 10 minutes.

- rnStatisticsLowActivePeriod

Specifies the period for which activity must be below the threshold defined by rnStatisticsRequestThreshold before statistics gathering activity is initiated. The default value is 2 minutes.

- rnStatisticsRequestThreshold

Specifies the threshold number of requests used to initiate statistics gathering. Statistics gathering is initiated if the sampled number of requests is below this threshold over the period defined by rnStatisticsLowActivePeriod. The default value is 2 requests.

Appendix F. Third Party Licenses

All of the third party licenses used by Oracle NoSQL Database are described in the LICENSE.txt file, which you can find in your KVHOME directory.