Oracle

NoSQL Database
Administrator's Guide

11g Release 2

Library Version 11.2.2.0

ORACLE®

Legal Notice

Table of Contents

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. Introduction to Oracle NoSQL Database

Welcome to Oracle NoSQL Database (Oracle NoSQL Database). Oracle NoSQL Database provides multi-terabyte distributed key/value pair storage that offers scalable throughput and performance. That is, it services network requests to store and retrieve data which is organized into key-value pairs. Oracle NoSQL Database services these types of data requests with a latency, throughput, and data consistency that is predictable based on how the store is configured.

Oracle NoSQL Database offers full Create, Read, Update and Delete (CRUD) operations with adjustable durability guarantees. Oracle NoSQL Database is designed to be highly available, with excellent throughput and latency, while requiring minimal administrative interaction.

Oracle NoSQL Database provides performance scalability. If you require better performance, you use more hardware. If your performance requirements are not very steep, you can purchase and manage fewer hardware resources.

Oracle NoSQL Database is meant for any application that requires network-accessible key-value data with user-definable read/write performance levels. The typical application is a web application which is servicing requests across the traditional three-tier architecture: web server, application server, and back-end database. In this configuration, Oracle NoSQL Database is meant to be installed behind the application server, causing it to either take the place of the back-end database, or work alongside it. To make use of Oracle NoSQL Database, code must be written (using Java or C) that runs on the application server.

An application makes use of Oracle NoSQL Database by performing network requests against Oracle NoSQL Database's key-value store, which is referred to as the KVStore. The requests are made using the Oracle NoSQL Database Driver, which is linked into your application as a Java library (.jar file), and then accessed using a series of Java APIs.

The usage of these APIs is introduced in the *Oracle NoSQL Database Getting Started Guide*.

The KVStore

The KVStore is a collection of Storage Nodes which host a set of Replication Nodes. Data is spread across the Replication Nodes. Given a traditional three-tier web architecture, the KVStore either takes the place of your back-end database, or runs alongside it.

The store contains multiple Storage Nodes. A *Storage Node* is a physical (or virtual) machine with its own local storage. The machine is intended to be commodity hardware. It should be, but is not required to be, identical to all other Storage Nodes within the store.

The following illustration depicts the typical architecture used by an application that makes use of Oracle NoSQL Database:

Load Balancers and web servers
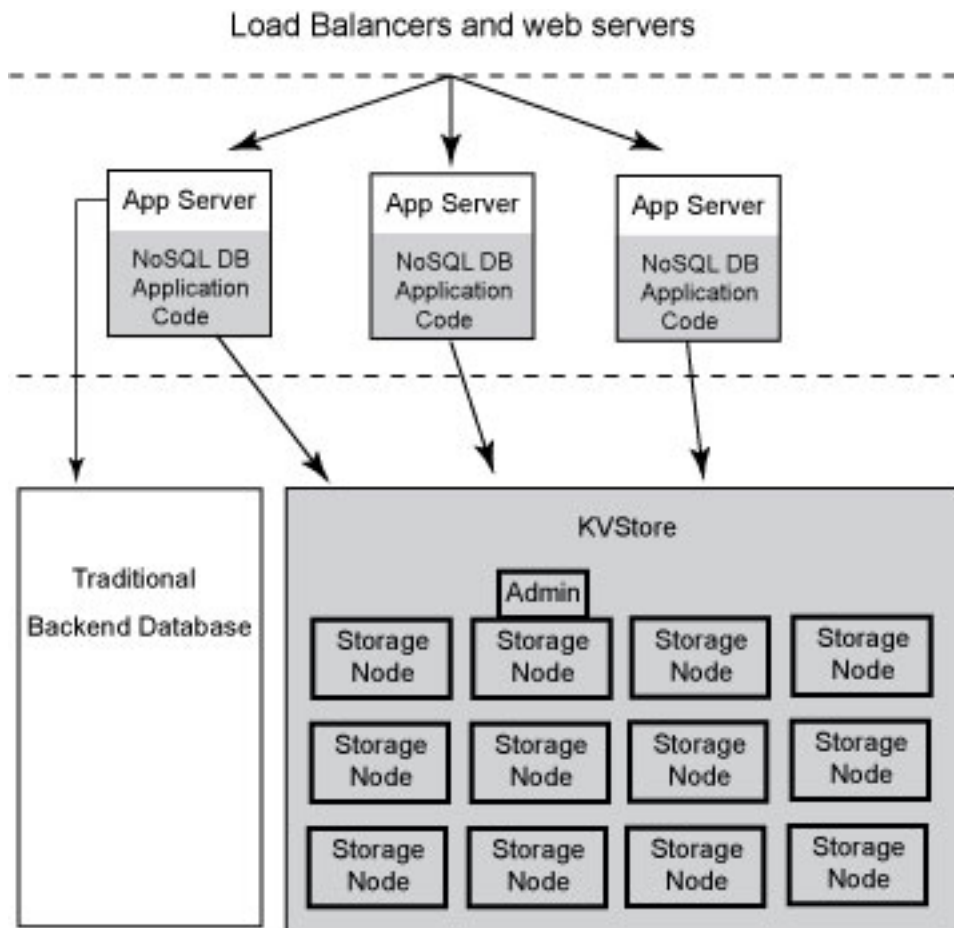
Every Storage Node hosts one or more Replication Nodes, which in turn contain one or more partitions. (For information on the best way to balance the number of Storage Nodes and Replication Nodes, see Balance a Non-Compliant Topology (page 38).) Also, each Storage Node contains monitoring software that ensures the Replication Nodes which it hosts are running and are otherwise healthy.

Replication Nodes and Shards

At a very high level, a *Replication Node* can be thought of as a single database which contains key-value pairs.

Replication Nodes are organized into *shards*. A shard contains a single Replication Node which is responsible for performing database writes, and which copies those writes to the other Replication Nodes in the shard. This is called the *master* node. All other Replication Nodes in the shard are used to service read-only operations. These are called the *replicas*. Although there can be only one master node at any given time, any of the members of the shard are capable of becoming a master node. In other words, each shard uses a single master/multiple replica strategy to improve read throughput and availability.

The following illustration shows how the KVStore is divided up into shards:
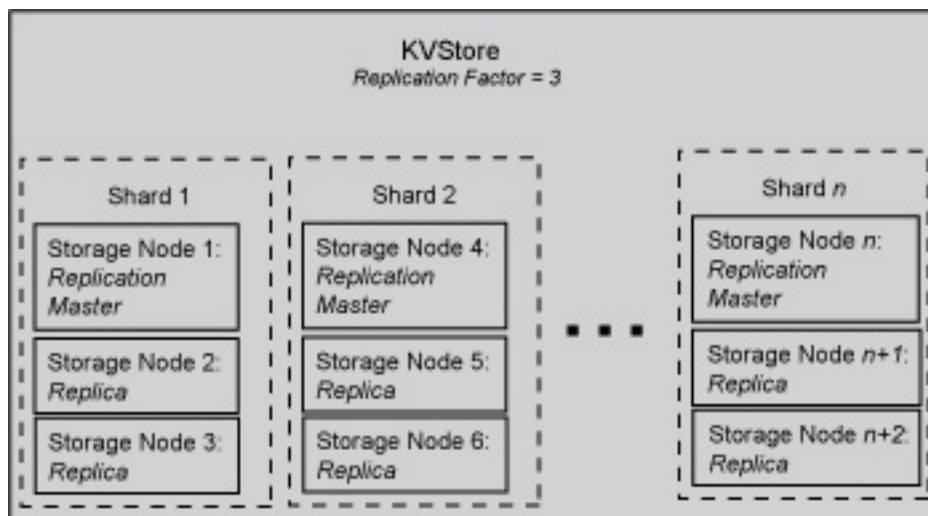
Note that if the machine hosting the master should fail in any way, then the master automatically fails over to one of the other nodes in the shard. (That is, one of the replica nodes is automatically promoted to master.)

Production KVStores should contain multiple shards. At installation time you provide information that allows Oracle NoSQL Database to automatically decide how many shards the store should contain. The more shards that your store contains, the better your write performance is because the store contains more nodes that are responsible for servicing write requests.

Replication Factor

The number of nodes belonging to a shard is called its *Replication Factor*. The larger a shard's Replication Factor, the faster its read throughput (because there are more machines to service the read requests) but the slower its write performance (because there are more machines to which writes must be copied). You set the Replication Factor for the store, and then Oracle NoSQL Database makes sure the appropriate number of Replication Nodes are created for each shard that your store contains.

For additional information on how to identify your replication factor and its implications, see .

Partitions

Each shard contains one or more *partitions*. Key-value pairs in the store are organized according to the key. Keys, in turn, are assigned to a partition. Once a key is placed in a partition, it cannot be moved to a different partition. Oracle NoSQL Database automatically assigns keys evenly across all the available partitions.

As part of your planning activities, you must decide how many partitions your store should have. Note that this is not configurable after the store has been installed.

It is possible to expand and change the number of Storage Nodes in use by the store. When this happens, the store can be reconfigured to take advantage of the new resources by adding

new shards. When this happens, partitions are balanced between new and old shards by redistributing partitions from one shard to another. For this reason, it is desirable to have enough partitions so as to allow fine-grained reconfiguration of the store. Note that there is a minimal performance cost for having a large number of partitions. As a rough rule of thumb, there should be at least 10 to 20 partitions per shard. Since the number of partitions cannot be changed after the initial deployment, you should consider the maximum future size of the store when specifying the number of partitions.

Topologies

A *topology* is the collection of storage nodes, replication nodes and administration services that make up an NoSQL DB store. A deployed store has one topology that describes its state at a given time.

Topologies can be changed to achieve different performance characteristics, or in reaction to changes in the number or characteristics of the Storage Nodes. Changing and deploying a topology is an iterative process. For information on how to use the command line interface to create, transform, view, validate and preview a topology, see topology (page 73).

Access and Security

Access to the KVStore and its data is performed in two different ways. Routine access to the data is performed using Java APIs that the application developer uses to allow his application to interact with the Oracle NoSQL Database Driver, which communicates with the store's Storage Nodes in order to perform whatever data access the application developer requires. The Java APIs that the application developer uses are introduced later in this manual.

In addition, administrative access to the store is performed using a command line interface or a browser-based graphical user interface. System administrators use these interfaces to perform the few administrative actions that are required by Oracle NoSQL Database. You can also monitor the store using these interfaces.

Note

Oracle NoSQL Database is intended to be installed in a secure location where physical and network access to the store is restricted to trusted users. For this reason, at this time Oracle NoSQL Database's security model is designed to prevent accidental access to the data. It is *not* designed to prevent malicious access or denial-of-service attacks.

The Administration Command Line Interface

The Administration command line interface (CLI) is the primary tool used to manage your store. It is used to configure, deploy, and change store components. It can also be used to verify the system, check service status, check for critical events and browse the store-wide log file. Alternatively, you can use a browser-based graphical user interface to do read-only monitoring. (Described in the next section.)

The command line interface is accessed using the following command: `java -jar KVHOME/lib/kvstore.jar runadmin`.

For a complete listing of all the commands available to you in the CLI, see Command Line Interface (CLI) Command Reference (page 63).

The Admin Console

Oracle NoSQL Database provides an HTML-based graphical user interface that you can use to monitor your store. It is called the *Admin Console*. To access it, you point your browser to a machine and port where your administration process is running. In the examples used later in this book, we use port 5001 for this purpose.

The Admin Console offers the following main functional areas:

- Topology. Use the Topology screen to see all the nodes that have been installed for your store. This screen also shows you at a glance the health of the nodes in your store.
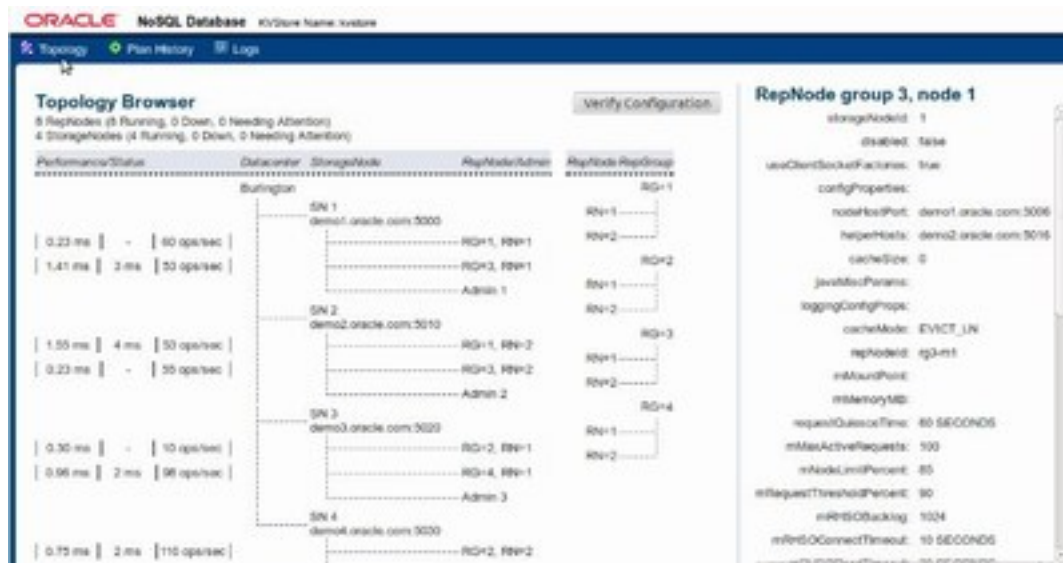
- Plan & History. This screen offers you the ability to view the last twenty plans that have been executed.

- Logs. This screen shows you the contents of the store's log files. You can also download the contents of the log files from this screen.

Chapter 2. Planning Your Installation

To successfully deploy a KVStore requires analyzing the workload you place on the store, and determining how many hardware resources are required to support that workload. Once you have performed this analysis, you can then determine how you should deploy the KVStore across those resources.

The overall process for planning the installation of your store involves these steps:

- Gather the store size and throughput requirements

- Determine the store's configuration. This involves identifying the total number of nodes your store requires, the number of partitions your store uses, the number of shards, and the Replication Factor in use by your store.

- Determine the cache size that you should use for your nodes.

Once you have performed each of the above steps, you should test your installation under a simulated load, refining the configuration as is necessary, before placing your store into a production environment.

The following sections more fully describe these steps.

Identify Store Size and Throughput Requirements

Before you can plan your store's installation, you must have some understanding of the store's contents, as well as the performance characteristics that your application requires from the store.

- The number and size of the keys and data items that are placed in the store.

- Roughly the maximum number of put and get operations that are performed per unit of time.

- The maximum permissible latency for each store operation.

These topics are discussed in the following sections.

Estimating the Record Size

Your KVStore contains some number of key-value pairs. The number and size of the key-value pairs contained by your store determine how much disk storage your store requires. It also defines how large an in-memory cache is required for each physical machine used to support the store.

The key portion of each key-value comprises some combination of *major* and *minor* key components. Taken together, these look something like a path to a file in a file system. Like any file system path, keys can be very short or very long. Records that use a large number of long key components obviously require more storage resources than do records with a small number of short key components.

Similarly, the amount of data associated with each key (that is, the *value* portion of each key-value pair) also affects how much storage capacity your store requires.

Finally, the number of records to be placed in your store also drives your storage capacity.

Ultimately, prior to an actual production deployment, there is only one way for you to estimate your store's storage requirements: ask the people who are designing and building the application that the store is meant to support. Schema design is an important part of designing an Oracle NoSQL Database application, so your engineering team should be able to describe the size of the keys as well as the size of the data items in use by the store. They should also have an idea of how many key-value pairs the store contains, and they should be able to advise you on how much disk storage you need for each node based on how they designed their keys and values, as well as how many partitions you want to use.

Estimating the Workload

In order to determine how to deploy your store, you must determine how many operations per second your store is expected to support. Estimate:

- How many read operations your store must handle per second.

- How many updates per second your store must support. This estimate must include all possible variants of put operations to existing keys.

- How many record creations per second your store must support. This estimate must include all possible variants of put operations on new keys.

- How many record deletions per second your store must support. This estimate must include all possible variants of delete operations.

If your application uses the multi-key operations (`KVStore.execute()`, `multiGet()`, or `multiDelete()`), then approximate the key-value pairs actually involved in each such multi-key operation to arrive at the necessary throughput numbers.

Ultimately, the throughput requirements you identify must be well matched to the I/O capacity available with the disk storage system in use by your nodes, as well as the amount of memory available at each node.

It may be necessary for you to consult with your engineering team and/or the business plan driving the development and deployment of your Oracle NoSQL Database application in order to obtain these estimates.

Estimate the Store's Permissible Average Latency

Latency is the measure of the time it takes your store to perform any given operation. You need to determine the average permissible latency for all possible store operations: reads, creates, updates, and deletes. The average latency for each of these is determined primarily by:

- How long it takes your disk I/O system to perform reads and writes.

- How much memory is available to the node (the more memory you have, the more data you can cache in memory, thereby avoiding expensive disk I/O).

- Your application's data access patterns (the more your store's operations cluster on records, the more efficient the store is at servicing store operations from the in-memory cache).

Note that if your read latency requirements are less than 10ms, then the typical hard disk available on the market today is not sufficient on its own. To achieve latencies of less than 10ms, you must make sure there is enough physical memory on each node so that an appropriate fraction of your read requests can be serviced from the in-memory cache. How much physical memory your nodes require is affected in part by how well your read requests cluster on records. The more your read requests tend to access the same records, the smaller your cache needs to be.

Also, version-based write operations may require disk access to read the version number. The KVStore caches version numbers whenever possible to minimize this source of disk reads. Nevertheless, if your version-based write operations do not cluster well, then you may require a larger in-memory cache in order to achieve your latency requirements.

Determine the Store's Configuration

Now that you have some idea of your store's storage and performance requirements, you can decide how you should configure the store. To do this, you must decide:

- How many shards you should use.

- How many replication partitions you should use.

- What your Replication Factor should be.

- Finally, how many nodes you should use in your store.

The following sections cover these topics in greater detail.

Identify the Target Number of Shards

The KVStore contains one or more shards. Each shard contains a single node that is responsible for servicing write requests, plus one or more nodes that are responsible for servicing read requests.

The more shards your store contains, the better your store is at servicing write requests. Therefore, if your Oracle NoSQL Database application requires high throughput on data writes (that is, record creations, updates, and deletions) then you want to configure your store with more shards.

Shards contain one or more partitions (described in the next section), and key-value pairs are spread evenly across these partitions. This means that the more shards your store contains, the less disk space your store requires on a per-node basis.

For example, suppose you know your store contains roughly n records, each of which represents a total of m bytes of data, for a total of $n * m$ bytes of data to be managed by

your store. If you have three shards, then each Storage Node must have enough disk space to contain *(n * m) / 3* bytes of data.

It might help you to use the following formula to arrive at a rough initial estimate of the number of shards that you need:

```
RG = (((((avg key size * 2) + avg value size) * max kv pairs) * 2) +
      (avg key size * max kv pairs) / 100 ) /
      (node storage capacity)
```

Note that the final factor of two in the first line of the equation is based upon a KVStore tuning control called the *cleaner utilization*. Here, we assume you leave the cleaner utilization at 50%.

As an example, a store sized to hold a maximum of 1 billion key value pairs, having an average key size of 10 bytes and an average value size of 1K, with 1TB (10^{12}) of storage available at each node would require two shards:

```
(((((10*2)+1000) * (10^9)) * 2) + ((10 * (10^9))/100) / 10^12 = 2 RGs
```

Remember that this formula only provides a rough estimate. Other factors such as I/O throughput and cache sizes need to be considered in order to arrive at a better approximation. Whatever number you arrive at here, you should thoroughly test it in a pre-production environment, and then make any necessary adjustments. (This is true of any estimate you make when planning your Oracle NoSQL Database installation.)

Identify the Number of Partitions

Every shard in your store must contain at least one partition, but you should configure your store so that it contains many partitions. The records in the KVStore are spread evenly across the KVStore partitions, and as a consequence they are also spread evenly across your shards. You identify the total number of partitions that your store should contain when you initially create your store. This number is static and cannot be changed over your store's lifetime.

Make sure the number of partitions you select is more than the largest number of shards you ever expect your store to 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 that you select 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 gives you plenty of room for growing your store. It is not unreasonable to select a partition number that is 100 times the maximum number of shards that you ever expect to use with your store.

Identify your Replication Factor

The KVStore contains one or more shards. Each shard contains a single node that is responsible for servicing write requests (the master), plus one or more nodes that are responsible for servicing read requests (the replicas).

The store's Replication Factor simply describes how many nodes (master + replicas) each shard contains. A Replication Factor of 3 gives you shards with one master plus two replicas. (Of course, if you lose or shut down a node that is hosting a master, then the master fails over to one of the other nodes in the shard, giving you a shard with one master and one replica. But this should be an unusual, and temporary, condition for your shards.)

The bigger your Replication Factor, the more responsive your store can be at servicing read requests because there are more nodes per shard available to service those requests. However, a larger Replication Factor reduces the number of shards your store can have, assuming a static number of Storage Nodes.

A large Replication Factor can also slow down your store's write performance, because each shard has more nodes to which updates must be transferred.

In general, we recommend a Replication Factor of 3, unless your performance testing suggests some other number works better for your particular workload. Also, do not select a Replication Factor of 2 because doing so means that even a single failure results in too few sites to elect a new master.

Identify the Total Number of Nodes

You can estimate the total number of Storage Nodes needed for your store by multiplying the number of shards you require times your Replication Factor. This number should suffice, unless you discover that your hard disks are unable to deliver enough IOPs to meet your throughput requirements. In that case, you might need to increase your Replication Factor, or increase your total number of shards.

If you underestimate the number of Storage Nodes, remember that it is possible to dynamically increase the number of Storage Nodes in use by the store. To use the command line interface to expand your store, see Transform the Topology Candidate (page 36).

Whatever estimates you arrive at, make sure to *thoroughly* test your configuration before deploying your store into a production environment.

Determining the Per-Node Cache Size

Sizing your in-memory cache correctly is an important part of meeting your store's performance goals. Disk I/O is an expensive operation from a performance point of view; the more operations you can service from cache, the better your store's performance is going to be.

There are several disk cache strategies that you can use, each of which is appropriate for different workloads. However, Oracle NoSQL Database was designed for applications that cannot place all their data in memory, so this release of the product describes a caching strategy that is appropriate for that class of workload.

Before continuing, it is worth noting that there are two caches that we are concerned with:

- JE cache size. 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 you most need to think about, because it is the one that you have the most control over.

- 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 your 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 your database's INs, and leave the rest of your node's memory available for system overhead (negligible) and the FS cache.

You cannot control whether INs or LNs are being served out of the FS cache, so sizing the JE cache to be large enough for your INs is simply sizing advice. 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 this strategy to be truly effective, your 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.

Assuming a reasonable amount of clustering in your data access patterns, your disk subsystem should be capable of delivering roughly the following throughput if you size your cache as described here:

```
((readOps/Sec + createOps/Sec + updateOps/Sec + deleteOps/Sec) *
 (1-cache hit fraction))/nReplicationNodes => throughput in IOPs/sec
```

The above rough calculation assumes that each create, update, and delete operation results in a random I/O operation. Due to the log structured nature of the underlying storage system, this is not typically the case and application-level write operations result in batched sequential synchronous write operations. So the above rough calculation may overstate the IOPs requirements, but it does provide a good conservative number for estimation purposes.

For example, if a KVStore with two shards and a replication factor of 3 (for a total of six replication nodes) needs to deliver an aggregate 2000 ops/sec (summing all read, create,

update and delete operations), and a 50% cache hit ratio is expected, then the I/O subsystem on each replication node should be able to deliver:

```
((2000 ops/sec) * (1 - 0.5)) / 6 nodes = 166 IOPs/sec
```

This is roughly in the range of what a single spindle disk subsystem can provide. For higher throughput, a multi-spindle I/O subsystem may be more appropriate. Another option is to increase the number of shards and therefore the number of replication nodes and therefore disks, thus spreading out the I/O load.

Arriving at Sizing Numbers

In order to identify an appropriate JE cache size for your Big Data application, use the com.sleepycat.je.util.DbCacheSize utility. This utility requires you to provide the number of records and the size of your keys. You can also optionally provide other information, such as your 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 Minimum, internal nodes only 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 -d64 -XX:+UseCompressedOops -jar je.jar DbCacheSize \
-key 12 -data 1000 -records 100000000

=== Environment Cache Overhead ===

3,156,253 minimum bytes

To account for JE daemon operation and record locks,
a significantly larger amount is needed in practice.

=== Database Cache Size ===

 Minimum Bytes     Maximum Bytes     Description
 ---------------   ---------------   -----------
   2,888,145,968     3,469,963,312   Internal nodes only
 107,499,427,952   108,081,245,296   Internal nodes and leaf nodes

=== Internal Node Usage by Btree Level ===

 Minimum Bytes     Maximum Bytes        Nodes     Level
 ---------------   ---------------   ----------   -----
   2,849,439,456     3,424,720,608    1,123,596     1
      38,275,968        44,739,456       12,624     2
         427,512           499,704          141     3
           3,032             3,544            1     4
```

The numbers you want are in the Database Cache Size section of the output. In the Minimum Bytes column, there are two numbers: One for internal nodes only, and one for

---

internal nodes plus leaf nodes. What this means is that the absolutely minimum cache size you should use for a dataset of this size is 2.9 GB. However, that stores only your internal database structure; the cache is not large enough to hold any data.

The second number in the output represents the minimum cache size required to hold your entire database, including all data. At 107.5 GB, it is highly unlikely that you have machines with that much RAM. Which means that you now have to make some decisions about your data. Namely, you have to decide how large your *working set* is. Your working set is the data that your application accesses so frequently that it is worth placing it in the in-memory cache. How large your working set has to be is determined by the nature of your application. Hopefully your working set is small enough to fit into the amount of RAM available to your node machines, as this provides you the best read throughput by avoiding a lot of disk I/O.

```
java -d64 -XX:+UseCompressedOops -jar je.jar DbCacheSize \
-key 12 -data 1000 -records 10000000


=== Environment Cache Overhead ===

3,156,253 minimum bytes

To account for JE daemon operation and record locks,
a significantly larger amount is needed in practice.

=== Database Cache Size ===

 Minimum Bytes    Maximum Bytes    Description
---------------  ---------------  -----------
    288,816,824      346,998,968  Internal nodes only
 10,749,982,264   10,808,164,408  Internal nodes and leaf nodes

=== Internal Node Usage by Btree Level ===

 Minimum Bytes    Maximum Bytes        Nodes    Level
---------------  ---------------  ----------  -----
    284,944,960      342,473,280     112,360  1
      3,826,384        4,472,528       1,262  2
         42,448           49,616          14  3
          3,032            3,544           1  4
```

Not surprisingly, our cache sizes are now approximately 10% of what they were for our entire data set size (because we decided that our working set is about 10% of our entire data set size). That is, our working set can be placed in a cache that is about 10.8 GB in size. This should be easily possible for modern commodity hardware.

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 a targeted cache size value, you need to find out how big your Java heap must be in order to support it. To do this, use the `KVS Node Heap Shaping and Sizing` spreadsheet. Plug the number you obtained from `DbCacheSize` into cell 8B of the spreadsheet. Cell 29B then shows you how large to make the Java heap size.

Your file system cache is whatever memory is left over on your node after you subtract system overhead and the Java heap size.

You can find the `KVS Node Heap Shaping and Sizing` spreadsheet in your Oracle NoSQL Database distribution here: `<KVHOME>/doc/misc/MemoryConfigPlanning.xls`

Chapter 3. Plans

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 Data Center 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 take 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 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 23).)

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
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
```

Chapter 4. Installing Oracle NoSQL Database

This chapter describes the installation process for Oracle NoSQL Database in a multi-host environment. Before proceeding with the installation, please read Planning Your Installation (page 7).

Installation Prerequisites

Make sure that you have Java SE 6 (JDK 1.6.0 u25) 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.

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. 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 -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:

1.   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. 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.

2.   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.

3.   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.

Note that 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. Note that 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.

4.   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 port range is specified as "startPort,endPort". "5010,5020" is used by the examples in this book.

5. A second range of free ports that may be used by a Storage Node or a Replication Node when exporting RMI based services. Specifying this range is optional, and by default any available port may be used when exporting Storage or Replication Node services. The format of the value string is "startPort,endPort". This parameter is useful when there is a firewall between the clients and the nodes that comprise the store and the firewall is being used to restrict access to specific ports. See the section on Setting Store Parameters for more information about the servicePortRange.

6. 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, and memory to support multiple Replication Nodes. This value defaults to "1". "1" is used as capacity by the examples in this book.

7. 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.

8. 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.

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.)

    To create the "boot config" file, issue the following commands:
    ```
    > mkdir -p KVROOT      (if it does not already exist)
    > java -jar KVHOME/lib/kvstore.jar makebootconfig -root KVROOT \
                                         -port 5000 \
                                         -admin 5001 \
                                         -host <hostname> \
                                         -harange 5010,5020 \
                                         -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. You can use the `start` utility for this:

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

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 KVHOME/lib/kvstore.jar ping -port 5000 -host node01
```

If SNA is running, you see the following output:

```
SNA at hostname: node01, registry port: 5000 is not registered.
No further information is available
```

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:

```
Could not connect to registry at node01:5000
Connection refused to host: node01; nested exception is:
java.net.ConnectException: Connection refused
```

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 -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
```

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.

Chapter 5. 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 19), 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 24)

2. Name your KVStore (page 24)

3. Create a Data Center (page 25)

4. Create an Administration Process on a Specific Host (page 25)

5. Create a Storage Node Pool (page 26)

6. Create the Remainder of your Storage Nodes (page 27)

7. Create and Deploy Replication Nodes (page 27)

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 Command Line Interface (CLI) Command Reference (page 63).

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 19) 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 -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 (page 28) 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 16).

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 19) 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
```

Create a Data Center

Once you have started the command line interface and configured a store name, you can create a Data Center. When you execute the `plan deploy-datacenter` command, the CLI returns the plan number and whatever additional information it has about plan status. This command takes the following arguments:

Note

A store can currently have only one Data Center.

- *datacenter name*

  A string to identify the name of the Data Center.

- *replication factor*

  A number specifying the replication factor.

For additional information on how to identify your replication factor and its implications, see Identify your Replication Factor (page 10).

When you execute the `plan deploy-datacenter` 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-datacenter -name "Boston" -rf 3 -wait
Executed plan 1, waiting for completion...
Plan 1 ended successfully
kv->
```

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

```
kv-> show plans
1 Deploy DC          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.

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

```
kv-> show topology
dc=[dc1] name=Boston
kv->
```

The Data Center ID is "dc1" in the above output.

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

```
kv-> plan deploy-sn -dc dc1 -host node01 -port 5000 -wait
Executed plan 2, waiting for completion...
Plan 2 ended successfully
kv->
```

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
kv->
```

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 must create a Storage Node Pool. This pool is used to contain all the SNs 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.

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
dc=[dc1] name=Boston
 sn=[sn1]  dc=dc1 node1:5000 status=UNREPORTED
kv-> pool join -name BostonPool -sn sn1
Added Storage Node sn1 to pool BostonPool
kv->
```

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 25). 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 -dc dc1 -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 sn2 to pool BostonPool
kv->  plan deploy-sn -dc dc1 -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 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 25) 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 fails if there is not enough capacity in the pool to justify the number of the replication nodes needed to create the topology.

  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 additional information on how to identify the total number of partitions, see Identify the Number of Partitions (page 10).

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
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 DataCenter <1>        SUCCEEDED
2 Deploy Storage Node <2>      SUCCEEDED
3 Deploy Admin Service         SUCCEEDED
4 Deploy Storage Node <4>      SUCCEEDED
5 Deploy Storage Node <5>      SUCCEEDED
6 Deploy Topo <6>              SUCCEEDED
```

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

```
kv-> exit
```

Using a Script

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 -jar KVHOME/lib/kvstore.jar runadmin -port 5000 -host node01 \
load -file scrpt.txt
```

```
kv->
```

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-datacenter -name "Boston" -rf 3 -wait
plan deploy-sn -dc dc1 -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 -dc dc1 -host node02 -port 5000 -wait
pool join -name BostonPool -sn sn2
plan deploy-sn -dc dc1 -host node03 -port 5000 -wait
pool join -name BostonPool -sn sn3
topology create -name topo -pool BostonPool -partitions 300
plan deploy-topology -name topo -wait
exit
### End Script ###
```

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 -jar KVHOME/lib/kvstore.jar ping -port 5000 -host node01
Pinging components of store mystore based upon topology sequence #107
mystore comprises 300 partitions on 3 Storage Nodes
Datacenter:Boston [dc1]
Storage Node [sn1] on node01:5000   Datacenter: Boston [dc1] Status:
RUNNING Ver: 11gR2.1.0.28
        Rep Node [rg1-rn1] Status:RUNNING,MASTER at sequence number:31
haPort:5011
Storage Node [sn2] on node02:5000   Datacenter: Boston [dc1] Status:
RUNNING Ver: 11gR2.1.0.28
        Rep Node [rg1-rn2] Status:RUNNING,REPLICA at sequence number:31
haPort:5011
Storage Node [sn3] on node03:5000   Datacenter: Boston [dc1] Status:
RUNNING Ver: 11gR2.1.0.28
        Rep Node [rg1-rn3] Status:RUNNING,REPLICA at sequence number:31
haPort:5011
```

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 -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 -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 20).

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 `KVROOT/storename/log` as well as logs of the bootstrap process in `KVROOT/*.log`. The bootstrap logs are most useful in diagnosing initial startup problems. The logs in `storename/log` appear once the 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: `KVROOT/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.

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.

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 -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 6. 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 factor of the store's datacenter. Topology layouts are also governed by a set of rules that maximize the availability of the store.

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 the number or capacity of Storage Nodes available, or the replication factor of a datacenter. 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 41).

Steps for Changing the Store's Topology

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

1. Make the Topology Candidate (page 35)

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 36)](#).

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.

For example:

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

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 NewTopo -name CloneTopo
```

Also, there is a variant of the topology create 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
```

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.

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 datacenter must have one or more Replication Nodes from each shard.

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 data center's replication factor is four and the new partition requires four nodes to satisfy the replication requirements:

```
kv-> plan deploy-sn -dc dc1 -host node05 -port 5008 -wait
```

```
Executed plan 7, waiting for completion...
Plan 7 ended successfully
kv-> plan deploy-sn -dc dc1 -host node06 -port 5010 -wait
Executed plan 8, waiting for completion...
Plan 8 ended successfully
kv-> plan deploy-sn -dc dc1 -host node07 -port 5012 -wait
Executed plan 9, waiting for completion...
Plan 9 ended successfully
kv-> plan deploy-sn -dc dc1 -host node08 -port 5014 -wait
Executed plan 10, waiting for completion...
Plan 10 ended successfully
kv-> pool join -name BostonPool -sn sn5
kv-> pool join -name BostonPool -sn sn6
kv-> pool join -name BostonPool -sn sn7
kv-> pool join -name BostonPool -sn sn8
kv-> topology redistribute -name NewTopo -pool BostonPool
```

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

Redistribute commands should not issued 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 datacenter 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 replication factor and its implications, see Identify your Replication Factor (page 10).

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. She then clones the existing topology and transforms it to use a new replication factor of 4.

```
kv-> plan deploy-sn -dc dc1 -host node09 -port 5016 -wait
Executed plan 11, waiting for completion...
Plan 11 ended successfully
kv-> pool join -name BostonPool -sn sn9
kv-> topology clone -current -name NewTopo
kv-> topology change-repfactor -name NewTopo -pool BostonPool -rf 4 -dc dc1
kv-> plan deploy-topology -name NewTopo -wait
Executed plan 12, waiting for completion...
Plan 12 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 Transform the Topology Candidate (page 36). 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 NewTopo 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 NewTopo -pool BostonPool
```

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.

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 Transform the Topology Candidate (page 36). 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.

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.

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.

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 Transform the Topology Candidate (page 36).

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

Chapter 7. Administrative Procedures

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

Backing Up the Store

To back up the KVStore, you take snapshots of nodes in the store and optionally 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

To create a backup, you take a snapshot of the store. A snapshot provides consistency across all records within the same partition, 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
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
kv-> snapshot create -name later
Created snapshot named 110915-153710-later
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.

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.

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 -jar KVHOME/lib/kvstore.jar ping -port 5000 -host node01
Pinging components of store mystore based upon topology sequence #107
mystore comprises 300 partitions on 3 Storage Nodes
Datacenter:Boston [dc1]
Storage Node [sn1] on node01:5000   Datacenter: Boston [dc1] Status:
RUNNING Ver: 11gR2.1.0.28
        Rep Node [rg1-rn1] Status:RUNNING,MASTER at sequence number:31
haPort:5011
Storage Node [sn2] on node02:5000   Datacenter: Boston [dc1] Status:
RUNNING Ver: 11gR2.1.0.28
        Rep Node [rg1-rn2] Status:RUNNING,REPLICA at sequence number:31
haPort:5011
Storage Node [sn3] on node03:5000   Datacenter: Boston [dc1] Status:
RUNNING Ver: 11gR2.1.0.28
        Rep Node [rg1-rn3] Status:RUNNING,REPLICA at sequence number:31
haPort:5011
```

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 KVROOT directory. For each Storage Node in the store, you have a directory named:

```
KVROOT/<store>/<SN>/<resource>/snapshots/<snapshot_name>/files
```

where:

- <store> is the name of the store.

- <SN> is the name of the storage node

- <resource> is the name of the resource running on the storage node. Typically this is the name of a replication node.

- <snapshot_name> is the name of the snapshot.

Snapshot data consists of a number of files, 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
```

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 46).

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.)

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 -jar KVHOME/lib/kvstore.jar load -source <backupDir>
    -store <storeName> -host <hostname> -port <port>
    [-status <pathToFile>][-verbose]
```

where:

- `-source <backupDir>` identifies the on-disk location where the snapshot data is stored.

- `-store <storeName>` identifies the name of the store.

- `-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.

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 -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 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.

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 host names, IP addresses and ports 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 -jar KVHOME/lib/kvstore.jar stop -root /var/kvroot
> nohup java -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 -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 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
```

Replacing a Failed Storage Node

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

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 KVROOT      (if it doesn't already exist)
> java -jar KVHOME/lib/kvstore.jar makebootconfig -root KVROOT \
                                                  -port 5000 \
                                                  -admin 5001 \
                                                  -host <hostname> \
                                                  -harange 5010,5020
```

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

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

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 -jar KVHOME/lib/kvstore.jar runadmin  \
    -port <port> -host <host>
    kv-> plan deploy-sn -dc <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. Migrates 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
    ```

7. 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 -jar KVHOME/lib/kvstore.jar stop -root KVROOT
    ```

8. The old storage node is shown in the topology and is reported as UNREACHABLE. If you are not using the old Storage Node in your NoSQL Database deployment, you should remove it from the store using the remove-storage node plan.

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 43) 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").

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

    For example:
    ```
    > java -jar KVHOME/lib/kvstore.jar generateconfig -host adminhost \
     -port 13230 -sn sn1 -target /tmp/sn1.config
    ```

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

3. Restore the Storage Node configuration on the target host:

    a.   Copy the zipfile 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 -jar KVHOME/lib/kvstore.jar start -root KVROOT
```

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 16).

- 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:

```
> java -jar KVHOME/lib/kvstore.jar runadmin -port <registry port> \
-host <hostname>
kv-> verify
Verify: starting for mystore based upon topology sequence #1008, 1000
partitions and 3 Storage Nodes, version:11gR2.1.1.0
See <nodeHostname>:/KVRT1/mystore/log/mystore_{0..N}.log for progress
messages

Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on <nodeHostname>:5000
Datacenter: baskin [dc1]    Status: RUNNING   Ver: 11gR2.1.1.0
Verify:      Admin [admin1]        Status: RUNNING
Verify:      Rep Node [rg1-rn1]  Status: RUNNING,REPLICA at
sequence number:2,025 haPort:5011
Verify: == checking storage node sn2 ==
Verify: Storage Node [sn2] on <nodeHostname>:5100
Datacenter: baskin [dc1]    Status: RUNNING   Ver: 11gR2.1.1.0
Verify:      Rep Node [rg1-rn2]  Status: RUNNING,REPLICA at sequence
number:2,025 haPort:5110
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on <nodeHostname>:5200
Datacenter: baskin [dc1]    Status: RUNNING   Ver: 11gR2.1.1.0
Verify:      Rep Node [rg1-rn3]  Status: RUNNING,MASTER at sequence
```

```
number:2,025 haPort:5210

Verification complete, no problems.
```

A silent mode is available which shows only problems.

```
kv-> verify -silent
Verify: starting for mystore based upon topology sequence #1008, 1000
partitions and 3 Storage Nodes, version:11gR2.1.1.0
See <nodeHostname>:/KVRT1/mystore/log/mystore_{0..N}.log for progress
messages

Verification complete, no problems.
```

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 -silent
Verify: starting for mystore based upon topology sequence #1008, 1000
partitions and 3 Storage Nodes, version:11gR2.1.1.0
See <nodeHostname>:/KVRT1/mystore/log/mystore_{0..N}.log for progress
messages

Verification complete, 2 problems found.
Verify: sn2:    ping() failed for sn2 : Connection refused to host:
<nodeHostname>; nested exception is:
        java.net.ConnectException: Connection refused
Verify: rg1-rn2:    ping() failed for rg1-rn2 : Connection refused to
host: <nodeHostname>; nested exception is:
        java.net.ConnectException: Connection refused
```

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

```
kv-> verify
Verify: starting for mystore based upon topology sequence #1008, 1000
partitions and 3 Storage Nodes, version:11gR2.1.1.0
See <nodeHostname>:/KVRT1/mystore/log/mystore_{0..N}.log for progress
messages

Verify: == checking storage node sn1 ==
Verify: Storage Node [sn1] on <nodeHostname>:5000
Datacenter: baskin [dc1]    Status: RUNNING   Ver: 11gR2.1.1.0
Verify:     Admin [admin1]         Status: RUNNING
Verify:     Rep Node [rg1-rn1]  Status: RUNNING,REPLICA at sequence
number:2,025 haPort:5011
Verify: == checking storage node sn2 ==
Verify: sn2: ping() failed for sn2 : Connection refused to host:
<nodeHostname>; nested exception is:
        java.net.ConnectException: Connection refused
Verify: Storage Node [sn2] on <nodeHostname>:5100
Datacenter: baskin [dc1] UNREACHABLE
Verify: rg1-rn2: ping() failed for rg1-rn2 : Connection refused to
```

```
host: <nodeHostname>; nested exception is:
        java.net.ConnectException: Connection refused
Verify:      Rep Node [rg1-rn2]  Status: UNREACHABLE
Verify: == checking storage node sn3 ==
Verify: Storage Node [sn3] on <nodeHostname>:5200
Datacenter: baskin [dc1]     Status: RUNNING    Ver: 11gR2.1.1.0
Verify:      Rep Node [rg1-rn3]  Status: RUNNING,MASTER at sequence
number:2,025 haPort:5210

Verification complete, 2 problems found.
Verify: sn2:     ping() failed for sn2 : Connection refused to host:
<nodeHostname>; nested exception is:
        java.net.ConnectException: Connection refused
Verify: rg1-rn2:    ping() failed for rg1-rn2 : Connection refused to
host: <nodeHostname>; 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.

- 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 61) 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 49) 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:

```
gt0hgvkiS STAT 09-25-11 16:30:54:162 EDT rg2-rn3 RUNNING sev1
gt0hgvkjS STAT 09-25-11 16:30:41:703 EDT rg1-rn1 RUNNING sev1
gt0hgvkkS STAT 09-25-11 16:30:51:540 EDT rg2-rn2 RUNNING sev1
gt0hicphL LOG  09-25-11 16:32:03:29 EDT SEVERE[admin1] Task StopAdmin
```

```
failed: StopAdmin [INTERRUPTED] start=09-25-11 16:32:03 end=09-25-11
16:32:03 Plan has been interrupted.: null: java.lang.InterruptedException
```

This shows three state change events and one severe 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 gt0hicphL
gt0hicphL LOG  09-25-11 16:32:03:29 EDT SEVERE[admin1] Task StopAdmin
failed: StopAdmin [INTERRUPTED] start=09-25-11 16:32:03 end=09-25-11
16:32:03 Plan has been interrupted.: null: java.lang.InterruptedException
            at java.util.concurrent.locks.AbstractQueuedSynchronizer.
doAcquireSharedNanos(AbstractQueuedSynchronizer.java:1024)
            at java.util.concurrent.locks.AbstractQueuedSynchronizer.
tryAcquireSharedNanos(AbstractQueuedSynchronizer.java:1303)
    ....
```

and so on, for a complete stack trace.

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

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.

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.

- `adminLogFileLimit=<Integer>`

Limits the size of log files. After reaching this limit, the logging subsystem switches to a new log file.

- `collectorPollPeriod=<Long TimeUnit>`

Sets the Monitor subsystem's delay for polling the various services for status updates.

- `loggingConfigProps=<String>`

Property settings for the Logging subsystem in the Admin process. Its format is `property=value;property=value....`

- `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.

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.

- `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.

- `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.

- 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. This number is used to inform the allocation of resources equitably among Replication Nodes when capacity > 1. Defaults to 0, which means "unknown."

- 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.

- mgmtClass=<String>

  The name of the class that provides the Management Agent implementation. See Standardized Monitoring Interfaces (page 61) for more information.

- 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 that may be used by a Storage Node or a Replication Node when exporting RMI based services. By default any available port may be used when exporting Storage or Replication Node services. The format of the value string is "lowport,highport".

  This parameter is useful when there is a firewall between the clients and the nodes that comprise the store and the firewall is being used to restrict access to specific ports. The range should be large enough to accommodate the Storage Node as well as all the Replication Nodes (as defined by the capacity parameter) hosted on the machines. Each Storage Node consumes two ports and each Replication Node three ports in the range. In addition, the Registry Service, hosted by the Storage Node, consumes a port if it is defined to fall within this range. 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.

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.

- 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.

- loggingConfigProps=<String>

  Contains property settings for the Logging subsystem. The format of this string is like that of configProperties, above.

- statsInterval=<Long TimeUnit>

  Sets the collection period for latency statistics at this Replication Node.

- cacheSize=<Long>

  Sets the cache size in the underlying BDB JE subsystem.

- 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.

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 -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.

Updating an Existing Oracle NoSQL Database Deployment

This section describes how to update your Oracle NoSQL Database deployment should Oracle release a software update.

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 (that is, via 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. 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).

Note

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

Use the host and registry port for any active node in the store.

To upgrade to new software, for each node (machine) in the system do the following steps. These steps should be done serially to minimize possible performance impact:

1. Place the updated 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 `nohup java -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 -jar OLD_KVHOME/lib/kvstore.jar stop -root <kvroot> \
      [-config <configfile>]
      ```

   b. Restart the Storage Node using the new code:

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

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.

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 20).) 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 SN. You discover the problem when you first attempt to deploy a store or a Admin Replica on a faulty SN. You see these indications that the RN 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 20).

5. On the Storage Node, restart the storage node using the java –jar KVHOME/lib/ kvstore.jar restart command.

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 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.

Appendix A. Command Line Interface (CLI) Command Reference

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

Usage:

```
java -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:

Usage:

```
java -jar KVHOME/lib/kvstore.jar runadmin
-host <hostname> -port <port> load -file <path-to-script>
```

If none of the optional arguments are passes, 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.

- -verbose

  Enables verbose output for the command.

Commands and Subcommands

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

configure

Usage:

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

Configures a new store. This call must be made before any other administration can be performed.

---

connect

Usage:

```
connect -host <hostname> -port <registry port>
```

Connects to the specified host and registry port to perform administrative functions. An Admin service must be active on the target host. This command can be used if the initial connection information is lost or incorrect.

ddl

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

For details on managing schema in the store, see .

ddl add-schema

Usage:

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

Use this subcommand to add a new schema or change (<evolve>) 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 the evolution warnings.

ddl enable-schema

Usage:

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

Use this subcommand to enable an existing, previously disabled schema.

ddl disable-schema

Usage:

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

Use this subcommand to disable an existing schema.

exit

Usage:

```
exit | quit
```

Exits the interactive command shell.

help

Usage:

```
help [command [sub-command]]
```

Prints help messages. With no arguments the top-level shell commands are listed. With additional commands and subcommands, additional detail is provided.

hidden

Toggles visibility and setting of parameters that are normally hidden. Use these parameters only if advised to do so by Oracle Support.

history

Usage:

```
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

Usage:

```
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.

logtail

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

ping

Displays the status of runtime components of the store, excluding administrative services.

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 change-mountpoint

Usage:

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

Adds or removes a mount point on a Storage Node.

plan change-parameters

Usage:

```
plan change-parameters -service <id> | -all-rns | -all-admins
                       [-dry-run] [-plan-name <name>] [-wait]
                       [-noexecute] [-force] -params [name=value]
```

Changes parameters for the specified service or all service instances of the same type. The -service flag allows a single instance to be affected. The -all-* flags can be used to change all instance 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". If -dry-run is specified, the new parameters are returned without changing them.

plan deploy-admin

Usage:

```
plan deploy-admin -sn <id> -port <http port> [-plan-name <name>]
                  [-wait] [-noexecute] [-force]
```

Deploys an Admin to the specified Storage Node. Its graphical interface listens on the specified port.

plan deploy-datacenter

Usage:

```
plan deploy-datacenter -name <datacenter name>
                       -rf <replication factor> [-plan-name <name>]
                       [-wait] [-noexecute] [-force]
```

Deploys the specified datacenter to the store.

> ### Note
>
> A store can currently have only one Data Center.

plan deploy-sn

Usage:

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

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

plan execute

Usage:

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

Executes a created but not yet executed plan. The plan must have been previously created using the -noexecute flag . Use -last to reference the most recently created plan.

plan interrupt

Usage:

```
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 cancel

Usage:

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

plan migrate-sn

Usage:

```
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. If the old node hosted an admin service, the -admin-port argument is required.

plan remove-admin

Usage:

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

Removes the specified Admin replica.

You cannot remove the sole Admin if only one Admin instance is configured. Also, if you try to remove an Admin when the removal would result in there being fewer than three deployed Admins, the command will fail unless you give the –force flag.

If you try to remove the Admin that is currently the master, mastership will transfer to another Admin. The plan will be interrupted , and subsequently can be re-executed on the new master Admin, by using the `plan execute` command.

plan remove-sn

Usage:

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

Removes the specified Storage Node from the topology.

plan start-service

Usage:

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

Starts the specified service(s).

plan stop-service

Usage:

```
plan stop-service -service <id> | -all-rns
```

Stops the specified service(s).

plan deploy-topology

Usage:

```
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.

plan wait

Usage:

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

Waits for the specified plan to complete. If the optional timeout is specified, wait that long, otherwise wait indefinitely. Use -last to reference the most recently created plan.

change-policy

Usage:

```
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.

pool

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

- pool create (page 70)

- pool remove (page 70)

- pool join (page 70)

pool create

Usage:

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

Creates a new Storage Node pool to be used for resource distribution when creating or modifying a store.

pool remove

Usage:

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

Removes a Storage Node pool.

pool join

Usage:

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

Adds Storage Nodes to an existing storage node pool.

show

Encapsulates commands that display the state of the store and its components. The subcommands are as follows:

- show parameters (page 71)

- show admins (page 71)

- show events (page 71)

- show faults (page 71)

- show perf (page 71)

- show plans (page 72)

- show pools (page 72)

- show schemas (page 72)

- show snapshots (page 72)

- show topology (page 72)

show parameters

Usage:

```
show parameters -policy | -service <name>
```

Displays service parameters and state for the specified service. The service may be a RepNode, StorageNode, 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.

show admins

Usage:

```
show admins
```

Displays basic information about Admin services.

show events

Usage:

```
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 must be investigated. Events marked SEVERE must always 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

show faults

Usage:

```
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 perf

Usage:

```
show perf
```

Displays recent performance information for each RepNode.

show plans

Usage:

```
show plans [-id <id> | -last]
```

Shows details of the specified plan or list all plans that have been created along with their corresponding plan IDs and status. Use -last to show details of the most recent created plan.

show pools

Usage:

```
show pools
```

Lists Storage Node pools.

show schemas

Usage:

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

Displays schema details of the named schema or a list of schemas registered with the store. The -disabled flag enables listing of disabled schemas.

show snapshots

Usage:

```
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 topology

Usage:

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

Displays the store topology. By default it shows the entire topology. The optional flags restrict the display to one or more of Datacenters, RepNodes, StorageNodes and Storename, or specify service status or performance.

snapshots

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

- snapshot create (page 73)

snapshot create

Usage:

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

Creates a new snapshot using the specified name as the prefix.

snapshot remove

Usage:

```
snapshot remove -name <name> | -all
```

Removes the named snapshot. If -all is specified, remove all snapshots.

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.The subcommands are as follows:

topology change-repfactor

Usage:

```
topology change-repfactor -name <name>  -pool <pool name>
                            -dc <id> | -dcname <name>
```

```
                          -rf <replication factor>
```

Modifies the topology to change the replication factor of the specified datacenter to a new value. The replication factor may not be decreased at this time.

topology clone

Usage:

```
 topology clone -from <from topology> -name <to topology>
```

or

```
 topology clone -current -name <to topology>
```

Clones an existing topology to a new candidate topology to be used for topology change operations.

topology create

Usage:

```
 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.

topology delete

Usage:

```
 topology delete -name <name>
```

Deletes a topology.

topology list

Usage:

```
 topology list
```

Lists explaining the topologies.

topology move-repnode

Usage:

```
 topology move-repnode -name <name> -rn <id>
```

Modifies the topology to move the specified RepNode to an available Storage Node chosen by the system.

topology preview

Usage:

```
topology preview -name <name> [-start <from topology>]
```

Previews the changes that would be made for the named topology relative to a starting topology. If -start is not specified, the current topology is used. This command should be used before deploying a new topology.

topology rebalance

Usage:

```
topology rebalance -name <name> -pool <pool name>
                      [-dc <id> | -dcname <name>]
```

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

topology redistribute

Usage:

```
topology redistribute -name <name> -pool <pool name>
```

Modifies the named topology to redistribute resources to more efficiently use those available.

topology validate

Usage:

```
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 may be potential issues, but may be expected.

topology view

Usage:

```
topology view -name <name>
```

Views details of the specified topology.

verbose

Used to toggle the global verbosity setting. This property can also be set per-command basis using the -verbose flag.

verify

Usage:

```
verify [-silent]
```

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.