Oracle

NoSQL Database
Security Guide

12c Release 1
Library Version 12.1.3.4
# Table of Contents

Preface ....................................................................................................... v  
Conventions Used in This Book ................................................................. v  
1. Introducing Oracle NoSQL Database Security ........................................... 1  
2. Security Configuration ........................................................................... 2  
   Security Configuration Overview ............................................................. 2  
   Configuring Security with Makebootconfig ............................................ 4  
   Configuring Security with Securityconfig .............................................. 5  
      Creating the security configuration ....................................................... 5  
      Adding the security configuration ....................................................... 6  
      Removing the security configuration .................................................. 6  
      Merging truststore configuration ....................................................... 7  
3. Performing a Secure Oracle NoSQL Database Installation ...................... 9  
   Single Node Secure Deployment ............................................................. 9  
      Adding Security to a New Installation ............................................... 9  
      Adding Security to an Existing Installation ......................................... 11  
   Multiple Node Secure Deployment ......................................................... 14  
      Adding Security to a New Installation .............................................. 14  
      Adding Security to an Existing Installation ......................................... 18  
4. External Password Storage ...................................................................... 21  
   Oracle Wallet ......................................................................................... 21  
   Password store file ............................................................................... 21  
5. Security.xml parameters .......................................................................... 23  
   Top-level parameters ............................................................................. 23  
   Transport parameters ............................................................................ 24  
6. Encryption .............................................................................................. 26  
   SSL model .............................................................................................. 26  
   SSL communication properties .............................................................. 27  
7. Configuring Authentication ...................................................................... 29  
   User Management ................................................................................... 29  
      User Creation ................................................................................... 29  
      User Modification ............................................................................. 30  
      User Removal ................................................................................... 31  
      User Status ....................................................................................... 31  
      User Login ....................................................................................... 32  
      Password Management ...................................................................... 32  
   Sessions .................................................................................................. 33  
8. Configuring Authorization ....................................................................... 35  
   Privileges ............................................................................................... 35  
      System Privileges ............................................................................. 35  
      Object Privileges ............................................................................. 36  
      Table Ownership .............................................................................. 37  
      Privilege Hierarchy .......................................................................... 38  
   Roles .................................................................................................... 38  
      System Built-in Roles ...................................................................... 38  
      User-Defined Roles ......................................................................... 39  
   Managing Roles, Privileges and Users ................................................... 40
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role Creation</td>
<td>41</td>
</tr>
<tr>
<td>Role Removal</td>
<td>41</td>
</tr>
<tr>
<td>Role Status</td>
<td>41</td>
</tr>
<tr>
<td>Grant Roles or Privileges</td>
<td>42</td>
</tr>
<tr>
<td>Revoke Roles or Privileges</td>
<td>43</td>
</tr>
<tr>
<td>9. Security Policies</td>
<td>45</td>
</tr>
<tr>
<td>Security Policy Modifications</td>
<td>45</td>
</tr>
<tr>
<td>10. Audit Logging</td>
<td>47</td>
</tr>
<tr>
<td>Security Log Messages</td>
<td>47</td>
</tr>
<tr>
<td>11. Keeping Oracle NoSQL Database Secure</td>
<td>48</td>
</tr>
<tr>
<td>Guidelines for Securing the Configuration</td>
<td>48</td>
</tr>
<tr>
<td>Guidelines for Deploying Secure Applications</td>
<td>48</td>
</tr>
<tr>
<td>Guidelines for Securing the SSL protocol</td>
<td>49</td>
</tr>
<tr>
<td>Guidelines for using JMX securely</td>
<td>49</td>
</tr>
<tr>
<td>Guidelines for Updating Keystore Passwords</td>
<td>49</td>
</tr>
<tr>
<td>Guidelines for Updating the SSL key/certificate</td>
<td>50</td>
</tr>
<tr>
<td>Guidelines for Operating System Security</td>
<td>51</td>
</tr>
<tr>
<td>A. SSL keystore generation</td>
<td>52</td>
</tr>
<tr>
<td>B. KVStore Required Privileges</td>
<td>54</td>
</tr>
<tr>
<td>Privileges for Accessing CLI Commands</td>
<td>54</td>
</tr>
<tr>
<td>Privileges for DDL Commands</td>
<td>57</td>
</tr>
<tr>
<td>Privileges for Accessing KVStore APIs</td>
<td>59</td>
</tr>
<tr>
<td>Privileges for Accessing KVStore TableAPIs</td>
<td>60</td>
</tr>
<tr>
<td>Privileges for Accessing KvLargeObject APIs</td>
<td>61</td>
</tr>
<tr>
<td>C. Third Party Licenses</td>
<td>62</td>
</tr>
</tbody>
</table>
Preface

This document describes how you can configure security for Oracle NoSQL Database using the default database features.

This book is aimed at the systems administrator responsible for the security of 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. Introducing Oracle NoSQL Database Security

Oracle NoSQL Database can be configured securely. In a secure configuration, network communications between NoSQL clients, utilities, and NoSQL server components are encrypted using SSL/TLS, and all processes must authenticate themselves to the components to which they connect.

There are two levels of security to be aware of. These are network security, which provides an outer layer of protection at the network level, and user authentication/authorization. Network security is configured at the file system level typically during the installation process, while user authentication/authorization is managed through NoSQL utilities.

You can use the default Oracle NoSQL Database features to configure security in the following areas for your Oracle NoSQL Database installation:

- **Security Configuration Utility.** Allows you to configure and add security to a new or existing Oracle NoSQL Database installation.

- **Authentication methods.** Oracle NoSQL Database provides password authentication for users and systems.

- **Encryption.** You can encrypt data on the network to prevent unauthorized access to that data.

- **External Password Storage.** Oracle NoSQL Database provides two types of external password storage methods that you can manipulate (one type for CE deployments).

- **Security Policies.** Oracle NoSQL Database allows you to set up behaviors in order to ensure a secure environment.

- **Role-based authorization.** Oracle NoSQL Database provides predefined system roles, privileges, and user-defined roles to users. You can set desired privileges to users by role-granting.

In addition, Keeping Oracle NoSQL Database Secure (page 48) provides guidelines that you should follow when securing your Oracle NoSQL Database installation.
Chapter 2. Security Configuration

This chapter describes how to use either the makebootconfig or securityconfig tool to perform the security configuration of your store. If you are installing a store with security for the first time, you can skip ahead to the next chapter Performing a Secure Oracle NoSQL Database Installation (page 9).

Note

For simpler use cases (lab environments) it is possible to perform a basic installation of your store by explicitly opting out of security on the command line. If you do this, your store loses all the security features described in this book. For more information see Configuring Security with Makebootconfig (page 4).

Security Configuration Overview

To set up security, you need to create an initial security configuration. To do this, run securityconfig before, after or as part of the makebootconfig process before starting the SNA on an initial node. You should not create a security configuration at each node. Instead, you should distribute the initial security configuration across all the Storage Nodes in your store. If the stores do not share a common security configuration they will be unable to communicate with one another.

Note

The makebootconfig utility embeds the functionality of securityconfig tool.

This tool creates a set of security files based on the standard configuration. It is possible to perform the same tasks manually, and advanced security configuration might require manual setup, but using this tool helps to ensure a consistent setup. For more information on the manual setup, see SSL keystore generation (page 52).

Note

It is possible to modify the security configuration after it is created in order to use a non-standard configuration. It is recommended that you use a standard configuration.

Those security files are generated, by default, within a directory named “security”. In a secure configuration, the bootstrap configuration file for a Storage Node includes a reference to that directory, which must be within the KVROOT directory for the Storage Node. The security directory contains:

<table>
<thead>
<tr>
<th>security/security.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>security/store.keys</td>
</tr>
<tr>
<td>security/store.trust</td>
</tr>
<tr>
<td>security/store.passwd (CE or EE installations)</td>
</tr>
<tr>
<td>security/store.wallet (EE installations only)</td>
</tr>
<tr>
<td>security/store.wallet/cwallet.sso (EE installations only)</td>
</tr>
</tbody>
</table>
security/client.security
security/client.trust

where:

• security.xml
  A configuration file that tells the Oracle NoSQL Database server how to apply security.

• store.keys
  A Java keystore file containing one or more SSL/TLS key pairs. This keystore is protected by a keystore password, which is recorded in an accompanying password store. The password store may be either an Oracle Wallet or a FileStore. The password is stored under the alias "keystore" in the password store. This file should be accessible only by the Oracle NoSQL Database server processes, and not to Oracle NoSQL Database clients.

• store.trust
  A Java truststore file, which is a keystore file that contains only public certificates, and no private keys.

• store.passwd (CE or EE installations)
  A password file that acts as the password store for a Community Edition (CE) installation. It contains secret information that should be known only to the server processes. Make sure the password file is readable and writable only by the Oracle NoSQL Database server. The file should not be copied to client machines.

  For Enterprise Edition (EE) installations, Oracle Wallet usage is preferred over the password file option.

• store.wallet (EE installations only)
  An Oracle Wallet directory that acts as the password store for an Enterprise Edition (EE) installation. It contains secret information that should be known only to the server processes. Make sure the directory and its contents are readable and writable only by the NoSQL DB server. The file should not be copied to client machines.

• cwallet.sso (EE installations only)
  The wallet password storage file.

• client.security
  A security configuration file that captures the communication transport properties for connecting clients to KVStore.

  The generated client.security file should be copied to and used by Oracle NoSQL Database clients when connecting to the KVStore.

• client.trust
A truststore file used by clients is generated.

The generated client.trust file should be copied to and used by Oracle NoSQL Database clients when connecting to the KVStore.

**Note**

In a multi-host store environment, the security directory and all files contained in it should be copied to each server that will host a Storage Node.

**Configuring Security with Makebootconfig**

Use the `makebootconfig` command with the required `-store-security` option to set up the basic store configuration with security:

```bash
java -jar KVHOME/lib/kvstore.jar makebootconfig
-root <kvroot> -port <port>
-admin <adminport> -host <hostname> -harange <harange>
-store-security configure -capacity <capacity>
```

where `-store-security` has the following options:

- **-store-security none**

  No security will be used. If a directory named “security” exists, a warning message will be displayed. When you opt out of security, you lose all the security features in your store; you are not able to set password authentication for users and systems, encrypt your data to prevent unauthorized access, etc.

- **-store-security configure**

  Security will be used and the `security configuration` utility will be invoked as part of the `makebootconfig` process. If the security directory already exists, an error message is displayed, otherwise the directory will be created.

  For script-based configuration you can use the `-kspwd<password>` option to allow tools to specify the keystore password on the command line. If it is not specified, the user is prompted to enter the password.

  Use the `-pwdmgr` option to select a password manager implementation. Its usage is introduced later in this section.

- **-store-security enable**

  Security will be used. You will need to configure security either by utilizing the `security configuration` utility or by copying a previously created configuration from another system.

  For more information on configuring security with `makebootconfig`, see Adding Security to a New Installation (page 9).
Configuring Security with Securityconfig

You can also run the securityconfig tool before or after the makebootconfig process by using the following command:

```bash
java -jar KVHOME/lib/kvstore.jar securityconfig
```

For more information on creating, adding, removing or merging the security configuration using securityconfig, see the following sections.

Creating the security configuration

You can use the `config create` command to create the security configuration:

```bash
config create
-root <secroot> [ -secdir <security dir> ]
[-pwdmgr { pwdfile | wallet }]
-param [client:|ha:|internal:|]<param>=<value>
```

where:

- `-root <secroot>`

  Specifies the directory in which the security configuration will be created. It is not required that this directory be a full KVROOT, but the directory must exist.

- `-secdir <security dir>`

  Specifies the name of the directory within the KVROOT that will hold the security configuration. This must be specified as a name relative to the specified secroot. If not specified, the default value is “security”.

- `-pwdmgr { pwdfile | wallet }`

  Indicates the password manager mechanism used to hold passwords that are needed for accessing keystores, etc.

  where `-pwdmgr` can have the following options:

  - `-pwdmgr pwdfile`

    Indicates that the password store is a read-protected clear-text password file. This is the only available option for Oracle NoSQL Database CE deployments. You can specify an alternate implementation. For more information on `pwdfile` manipulation, see Password store file (page 21)

  - `-pwdmgr wallet`

    Specifies Oracle Wallet as the password storage mechanism. This option is only available in the Oracle NoSQL Database EE version. For more information on Oracle wallet manipulation, see Oracle Wallet (page 21)
-param [client:|ha:|internal:|]<param>=<value>

A repeatable argument that allows configuration defaults to be overridden. The value may be either a simple parameter, such as "truststore", or a qualified parameter such as "client:serverKeyAlias". If specified in qualified form, the qualifier (for example, "client") names a transport within the security configuration, and the assignment is specific to that transport. If in simple form, it applies to either the securityParams structure or to all transports within the file, depending on the type of parameter.

For more information on configuring security with securityconfig, see Adding Security to an Existing Installation (page 11).

Adding the security configuration

You can use the config add-security command to add the security configuration you created earlier:

```
config add-security
-root <kvroot> [-secdir <security dir>]
[-config <config.xml>]
```

**Note**

When running this command, the securityconfig tool will verify the existence of the referenced files and will update the specified bootstrap configuration file to refer to the security configuration. This process is normally done with the KVStore instance stopped, and must be performed on each Storage Node of the store.

where:

- **-root <kvroot>**

  A KVStore root directory must be provided as an argument.

- **-secdir <security dir>**

  Specifies the name of the directory within the KVROOT that holds the security configuration. This must be specified as a name relative to the KVROOT. If not specified, the default value is "security".

- **-config <config.xml>**

  Specifies the bootstrap configuration file that is to be updated. This must be specified as a name relative to the KVROOT. If not specified, the default value is "config.xml".

Removing the security configuration

If you want to disable security for some reason in an existing installation, you can use the config remove-security command:

```
config remove-security
-root <kvroot> [-config <config.xml>]
```
**Note**

When running this command, the `securityconfig` tool will update the specified bootstrap configuration file to refer to the security configuration. This process is normally done with the KVStore instance stopped, and must be performed on each Storage Node of the store.

where:

- -root <kvroot>

  A KVStore root directory must be provided as an argument.

- -config <config.xml>

  Specifies the bootstrap configuration file that is to be updated. This must be specified as a name relative to the KVROOT. If not specified, the default value is "config.xml".

---

### Merging truststore configuration

If you want to merge truststore entries from one security configuration into another security configuration use the `config merge-trust` command. This command is helpful when performing security maintenance, particularly when you need to update the SSL key/certificate. For more information, see Guidelines for Updating the SSL key/certificate (page 50)

```
config merge-trust
  -root <secroot> [-secdir <security dir>]
  -source-root <secroot> [-source-secdir <security dir>]
```

**Note**

When running this command, the `securityconfig` tool will verify the existence of the referenced files and will combine trust entries from the source security configuration into the primary security configuration.

where:

- -root <secroot>

  Specifies the directory that contains the security configuration that will be updated. It is not required that this directory be a full KVROOT, but the directory must exist and contain an existing security configuration.

- -secdir <security dir>

  Specifies the name of the directory within the `secroot` that holds the security configuration. This must be specified as a name relative to the `secroot`. If not specified, the default value is "security".

- -source-root <secroot>

  Specifies the directory that contains the source security configuration.
Specifies the directory that contains the security configuration that will provide new trust information. It is not required that this directory be a full KVROOT, but the directory must exist and must contain an existing security configuration.

- `source-secdir <security dir>`

Specifies the name of the security directory within the source secroot that will provide new trust information. If not specified, the default value is "security".
Chapter 3. Performing a Secure Oracle NoSQL Database Installation

It is possible to add security to a new or an existing Oracle NoSQL Database installation.

To add security to a new or an existing Oracle NoSQL Database single host deployment, see the next section. For multiple node deployments, see Multiple Node Secure Deployment (page 14).

Single Node Secure Deployment

The following examples describe how to add security to a new or an existing Oracle NoSQL Database single host deployment.

Adding Security to a New Installation

To install Oracle NoSQL Database securely:

1. Run the makebootconfig utility with the required -store-security option to set up the basic store configuration with security:

   ```
   java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar makebootconfig \
   -root KVROOT -port 5000 \
   -admin 5001 -host node01 -harange 5010,5020 \
   -store-security configure -pwdmgr pwdfile -capacity 1
   ```

2. In this example, -store-security configure is used, so the security configuration utility is run as part of the makebootconfig process and you are prompted for a password to use for your keystore file:

   ```
   Enter a password for the Java KeyStore:
   ```

3. Enter a password for your store and then reenter it for verification. In this case, the password file is used, and the securityconfig tool will automatically generate the following security related files:

   ```
   Enter a password for the Java KeyStore: ***********
   Re-enter the KeyStore password for verification: ***********
   Created files:
   security/client.trust
   security/client.security
   security/store.keys
   security/store.trust
   security/store.passwd
   security/security.xml
   ```

   Note

   In a multi-host store environment, the security directory and all files contained in it should be copied to each server that will host a Storage Node.
4. Start the Storage Node Agent (SNA):

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

When a newly created store with a secure configuration is first started, there are no user definitions available against which to authenticate access. In order to reduce risk of unauthorized access, an admin will only allow you to connect to it from the host on which it is running. This security measure is not a complete safeguard against unauthorized access. It is important that you do not provide local access to machines running KVStore. In addition, you should perform steps 5, 6 and 7 soon after this step in order to minimize the time period in which the admin might be accessible without full authentication. For more information on maintaining a secure store see Guidelines for Securing the Configuration (page 48).

5. Start runadmin in security mode on the KVStore server host (node01). To do this, use the following command:

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

Logged in admin as anonymous

6. Use the configure -name command to specify the name of the KVStore that you want to configure:

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

7. Create an admin user. In this case, user root is defined:

```
kv-> plan create-user -name root -admin -wait
Enter the new password: ********
Re-enter the new password: ********
Executed plan 6, waiting for completion...
Plan 6 ended successfully
```

For more information on user creation and administration, see User Management (page 29).

8. Create a new password file to store the credentials needed to allow clients to login as the admin user (root):

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig \
pwdfile create -file KVROOT/security/login.passwd
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig pwdfile secret \
-file KVROOT/security/login.passwd -set -alias root
Enter the secret value to store: ********
Re-enter the secret value for verification: ********
Secret created
```
OK

**Note**

The password must match the one set for the admin in the previous step.

For more information on user creation and administration, see [User Management](page 29).

9. At this point, it is possible to connect to the store as the root user. To login, you can use either the `-username <user>` `runadmin` argument or specify the "oracle.kv.auth.username" property in the security file.

In this example, a security file (mylogin.txt) is used. To login, use the following command:

```sh
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar runadmin -port 5000 \
-host localhost -security mylogin
```

Logged in admin as root

The file `mylogin.txt` should be a copy of the `client.security` file with additional properties settings for authentication. The file would then contain content like this:

```ini
oracle.kv.auth.username=root
oracle.kv.auth.pwdfile.file=KVROOT/security/login.passwd
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=KVROOT/security/client.trust
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

For more information, see [User Login](page 32).

**Adding Security to an Existing Installation**

To add security to an existing Oracle NoSQL Database installation:

1. Shut down the KVStore instance:

   ```sh
   java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar stop \
   -root KVROOT
   ```

2. Run the `securityconfig` utility to set up the basic store configuration with security:

   ```sh
   java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar securityconfig
   ```

3. Use the `config create` command with the `-pwdmgr` option to specify the mechanism used to hold passwords that is needed for accessing the stores. In this case, Oracle Wallet is used. Oracle Wallet is only available in the Oracle NoSQL Database EE version. CE deployments should use the `pwdfile` option instead.

   ```sh
   config create -pwdmgr wallet -root KVROOT
   ```
Enter a password for the Java KeyStore:

4. Enter a password for your store and then reenter it for verification. The configuration tool will automatically generate some security related files:

Enter a password for the Java KeyStore: ***********
Re-enter the KeyStore password for verification: ***********
Created files:
- security/security.xml
- security/store.keys
- security/store.trust
- security/store.wallet/cwallet.sso
- security/client.security
- security/client.trust

**Note**

In a multi-host store environment, the security directory and all files contained in it should be copied to each server that will host a Storage Node.

5. Use the `config add-security` command to add the security configuration you just created:

```
security-> config add-security -root KVROOT
-secdir security -config config.xml
Configuration updated.
```

**Note**

When running this command, the `securityconfig` tool will verify the existence of the referenced files and will update the specified bootstrap configuration file to refer to the security configuration. This process is normally done with the KVStore instance stopped, and must be performed on each Storage Node of the store.

6. Start the Storage Node Agent (SNA):

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

7. Start runadmin in security mode on the KVStore server host (node01). To do this, use the following command:

```
java -Xmx256m -Xms256m
-jar KVHOME/lib/kvstore.jar \
runadmin -port 5000 -host node01 \
-security KVROOT/security/client.security
Logged in admin as anonymous.
```

This command sets SSL as a connection method and names a copy of the generated truststore file (client.security). For more information on SSL properties, see SSL communication properties (page 27).
8. Create an admin user. In this case, user root is defined:

```
kv-> plan create-user -name root -admin -wait
Enter the new password: ********
Re-enter the new password: ********
Executed plan 8, waiting for completion...
Plan 8 ended successfully
```

For more information on user creation and administration, see User Management (page 29).

9. Create a new wallet file to store the credentials needed to allow clients to login as the admin user (root):

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig \
wallet create -dir KVROOT/security/login.wallet
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig wallet secret \
-wallietcreate -dir KVROOT/security/login.wallet -set -alias root
Enter the secret value to store: ********
Re-enter the secret value for verification: ********
Secret created
```

**Note**

The password must match the one set for the admin in the previous step.

For more information on user creation and administration, see User Management (page 29).

10. At this point, it is possible to connect to the store as the root user. To login, you can use either the -username <user> runadmin argument or specify the "oracle.kv.auth.username" property in the security file.

In this example, the oracle.kv.security property is used. To login use the following command:

```
java -Xmx256m -Xms256m \n-Doracle.kv.security=mylogin \n-jar KVHOME/lib/kvstore.jar runadmin -port 5000 -host localhost
```

Logged in admin as root

The file mylogin.txt should be a copy of the client.security file with additional properties settings for authentication. The file would then contain content like this:

```
oracle.kv.auth.username=root
oracle.kv.auth.wallet.dir=KVROOT/security/login.wallet
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=KVROOT/security/client.trust
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1
```
Multiple Node Secure Deployment

The following examples describe how to add security to a new or to an existing Oracle NoSQL Database multiple host deployment.

Adding Security to a New Installation

To install an Oracle NoSQL Database three node, capacity=3 (3x3) secure deployment:

1. Run the makebootconfig utility with the required -store-security option to set up the basic store configuration with security:

   
   ```
   java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar makebootconfig \
   -root KVROOT -port 5000 \
   -admin 5001 -host node01 -harange 5010,5020 \
   -store-security configure -pwdmgr wallet -capacity 3
   ```

2. In this example, -store-security configure is used, so the security configuration utility is run as part of the makebootconfig process and you are prompted for a password to use for your keystore file:

   ```
   Enter a password for the Java KeyStore:
   ```

3. Enter a password for your store and then reenter it for verification. For example, using wallet, the securityconfig tool will automatically generate the following security related files:

   ```
   Enter a password for the Java KeyStore: ***********
   Re-enter the KeyStore password for verification: ***********
   Created files:
   security/security.xml
   security/store.keys
   security/store.trust
   security/store.wallet/cwallet.sso
   security/client.security
   security/client.trust
   ```

4. In a multi-host store environment, the security directory and all files contained in it should be copied from the first node to each server that will host a Storage Node, to setup internal cluster authentication. For example, the following commands assume that the different nodes are visible and accessible on the current node (node01):

   ```
   cp -R node01/KVROOT/security node02/KVROOT/
   cp -R node01/KVROOT/security node03/KVROOT/
   ```
Note

You may need to use a remote copying command, like scp, to do the copying if the files for the different nodes are not visible on the current node.

5. Enable security on the other two nodes using the `-store-security enable` command:

```
java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar makebootconfig -root KVROOT -host node02 -port 6000 -admin 6001 -harange 6010,6020 -capacity 3 -store-security enable -pwdmgr wallet
java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar makebootconfig -root KVROOT -host node03 -port 7000 -admin 7001 -harange 7010,7020 -capacity 3 -store-security enable -pwdmgr wallet
```

6. Start the Storage Node Agent (SNA) on each node:

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

7. Start runadmin in security mode on the KVStore server host (node01). To do this, use the following command:

```
java -Xmx256m -Xms256m -jar KVHOME/lib/kvstore.jar runadmin -port 5000 -host node01 -security KVROOT/security/client.security
```

Logged in admin as anonymous

8. Use the `configure -name` command to specify the name of the KVStore that you want to configure:

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

9. Create an admin user. In this case, user root is defined:

```
kv-> plan create-user -name root -admin -wait
```
Enter the new password: ********
Re-enter the new password: ********
Executed plan 6, waiting for completion...
Plan 6 ended successfully

For more information on user creation and administration, see User Management (page 29).

10. Create the wallet to enable client credentials for the admin user (root):

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig \
wallet create -dir KVROOT/security/login.wallet
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig wallet secret \
-dir KVROOT/security/login.wallet -set -alias root
Enter the secret value to store: ********
Re-enter the secret value for verification: ********
Secret created
OK
```

**Note**

The password must match the one set for the admin in the previous step.

11. At this point, it is possible to connect to the store as the root user. To login, you can use either the `-username <user>` runadmin argument or specify the "oracle.kv.auth.username" property in the security file.

In this example, a security file (adminlogin.txt) is used. To login, use the following command:

```
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar runadmin -port 5000 \
-host localhost -security adminlogin
Logged in admin as root
```

The file adminlogin.txt should be a copy of the client.security file with additional properties settings for authentication. The file would then contain content like this:

```
oracle.kv.auth.username=root
oracle.kv.auth.wallet.dir=KVROOT/security/login.wallet
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=KVROOT/security/client.trust
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

For more information, see User Login (page 32).

12. Once logged in as admin, you can create some users:

```
kv-> plan create-user -name user1 -wait
```
10/1/2015 Oracle NoSQL Database Security Guide Page 17

Enter the new password: ********
Re-enter the new password: ********
Executed plan 7, waiting for completion...
Plan 7 ended successfully

kv-> plan create-user -name user2 -wait
Enter the new password: ********
Re-enter the new password: ********
Executed plan 8, waiting for completion...
Plan 8 ended successfully

13. Create the wallet to enable client credentials for each user. Typically you will reuse this wallet for all your regular users:

```
java -Xmx256m -Xms256m
  -jar KVHOME/lib/kvstore.jar securityconfig \\n  wallet create -dir KVROOT/security/users.wallet
java -Xmx256m -Xms256m
  -jar KVHOME/lib/kvstore.jar securityconfig wallet secret \\n  -dir KVROOT/security/users.wallet -set -alias user1
Enter the secret value to store: ********
Re-enter the secret value for verification: ********
Secret created OK
```

14. At this point, it is possible to connect to the store as a user. To login, you can use either the -username <user> runadmin argument or specify the "oracle.kv.auth.username" property in the security file.

In this example, a security file (userlogin.txt) is used. To login, use the following command:

```
java -Xmx256m -Xms256m
  -jar KVHOME/lib/kvstore.jar runadmin -port 5000 \\n  -host localhost -security userlogin
Logged in admin as user1
```

The file userlogin.txt should be a copy of the client.security file with additional properties settings for authentication. The file would then contain content like this:

Note

Each password must match the one set for each user in the previous step. This wallet is independent from the admin one. It is possible to store admin/user passwords using the same wallet.
For more information, see *User Login (page 32).*

**Adding Security to an Existing Installation**

To add security to an existing three node, capacity=3 (3x3) Oracle NoSQL Database installation:

1. Shut down the KVStore instance on each node:
   ```
   java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar stop \
   -root KVROOT
   ```

2. Run the `securityconfig` utility to set up the basic store configuration with security:
   ```
   java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar securityconfig
   ```

3. Use the `config create` command with the `-pwdmgr` option to specify the mechanism used to hold passwords that is needed for accessing the stores. In this case, Oracle Wallet is used:
   ```
   config create -pwdmgr wallet -root KVROOT
   ```
   Enter a password for the Java KeyStore:
   ```
   ***********
   ```
   Re-enter the KeyStore password for verification: ***********
   
   Created files:
   ```
   security/security.xml
   security/store.keys
   security/store.trust
   security/store.wallet/cwallet.sso
   security/client.security
   security/client.trust
   ```

4. Enter a password for your store and then reenter it for verification. The configuration tool will automatically generate some security related files:
   ```
   Enter a password for the Java KeyStore: ***********
   ```
   Re-enter the KeyStore password for verification: ***********
   
   Created files:
   ```
   security/security.xml
   security/store.keys
   security/store.trust
   security/store.wallet/cwallet.sso
   security/client.security
   security/client.trust
   ```

5. In a multi-host store environment, the security directory and all files contained in it should be copied from the first node to each server that will host a Storage Node, to setup internal cluster authentication. For example, the following commands assume that the different nodes are visible and accessible on the current node (node01):
   ```
   cp -R node01/KVROOT/security node02/KVROOT/
   cp -R node01/KVROOT/security node03/KVROOT/
   ```
Note

You may need to use a remote copying command, like scp, to do the copying if the files for the different nodes are not visible on the current node.

6. Use the config add-security command on each node to add the security configuration you just created:

```shell
security-> config add-security -root KVROOT -secdir security
```

Note

When running this command, the securityconfig tool will verify the existence of the referenced files and will update the specified bootstrap configuration file to refer to the security configuration. This process is normally done with the KVStore instance stopped, and must be performed on each Storage Node of the store.

7. Start the Storage Node Agent (SNA) on each node:

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

8. Start runadmin in security mode on the KVStore server host (node01). To do this, use the following command:

```shell
java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar \
   runadmin -port 5000 -host node01 \
   -security KVROOT/security/client.security
```

This command sets SSL as a connection method and names a copy of the generated truststore file (client.security). For more information on SSL properties, see SSL communication properties (page 27).

9. Create an admin user. In this case, user root is defined:

```shell
kv-> plan create-user -name root -admin -wait
Enter the new password: ********
Re-enter the new password: ********
Executed plan 8, waiting for completion...
Plan 8 ended successfully
```

For more information on user creation and administration, see User Management (page 29).

10. Create the wallet to enable client credentials for the admin user (root):

```shell
java -Xmx256m -Xms256m \
   -jar KVHOME/lib/kvstore.jar securityconfig \
   wallet create -dir KVROOT/security/login.wallet
java -Xmx256m -Xms256m \
```
Performing a Secure Oracle NoSQL Database Installation

- jar KVHOME/lib/kvstore.jar securityconfig wallet secret \
- dir KVROOT/security/login.wallet -set -alias root

Enter the secret value to store: ********
Re-enter the secret value for verification: ********
Secret created
OK

Note

The password must match the one set for the admin in the previous step.

11. At this point, it is possible to connect to the store as the root user. To login, you can use either the `-username <user>` runadmin argument or specify the "oracle.kv.auth.username" property in the security file.

In this example, the oracle.kv.security property is used. To login use the following command:

```
java -Xmx256m -Xms256m \
-Doracle.kv.security=adminlogin \
-jar KVHOME/lib/kvstore.jar runadmin -port 5000 -host localhost
Logged in admin as root >
```

The file adminlogin.txt should be a copy of the client.security file with additional properties settings for authentication. The file would then contain content like this:

```
oracle.kv.auth.username=root
oracle.kv.auth.wallet.dir=KVROOT/security/login.wallet
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=KVROOT/security/client.trust
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1
oracle.kv.ssl.hostnameVerifier=dnmatch(CN=NoSQL)
```

For more information, see User Login (page 32).
Chapter 4. External Password Storage

Depending on the type of store deployment, there are two ways passwords can be externally stored. For Enterprise Edition (EE) deployments, Oracle Wallet is used. For Community Edition (CE) deployments, a simple read protected clear-text password file is used.

In the most basic mode of operation, external passwords are used only by the server to track the keystore password. User passwords, which are stored securely within the database, can also be supplied during client authentication.

When a password store is used as a component of a login file, the alias that is used for either password store type should be the username to which the password applies. For example, for a user named root, the password should be stored under the alias root.

When a password store is used as part of the server, the alias keystore is used. The user password store should be a completely different file than the one in the security directory located under KVROOT.

Oracle Wallet

The following commands provide functionality to manipulate Oracle wallet stores within the securityconfig tool. These commands are available in EE only. For more information on the securityconfig tool, see Configuring Security with Securityconfig (page 5).

To create a new auto-login wallet, run the wallet create command:

```bash
wallet create
-dir <wallet directory>
```

Auto-login wallets store passwords in an obfuscated state. Access to the wallet is secured against reading by unauthorized users using the OS-level login.

To manipulate secrets (passwords), which are associated with a name (alias), run the wallet secret command:

```bash
wallet secret
-dir <wallet directory>
{-set | -delete} -alias <alias>
```

If the -set option is specified, the user is prompted for a new password for the specified alias and required to verify the new secret.

If the -delete option is specified, the secret is deleted from the store.

Special considerations should be taken if Oracle wallet is used and you are deploying your Oracle NoSQL Database. For more information, see Guidelines for Deploying Secure Applications (page 48).

Password store file

The following commands are used to create and manipulate CE password store files within the securityconfig tool. CE password store files managed through this interface are never
password protected. For more information on the securityconfig tool, see Configuring Security with Securityconfig (page 5).

To create a new password store file, run the pwdfile create command:

```
pwdfile create
-file <password store file>
```

To manipulate secrets (passwords), which are associated with a name (alias), run the pwdfile secret command:

```
pwdfile secret
-file <password store file>
{-set | -delete} -alias <alias>
```

If the user specifies the -set option, the user is prompted for a new password for the specified alias and required to verify the new password.

If the -delete option is specified, the alias is deleted from the store.
Chapter 5. Security.xml parameters

This chapter describes the parameters that can be set to the security.xml configuration file. This file is generated by makebootconfig or securityconfig and tells the Oracle NoSQL Database server how to apply security.

The security.xml file specifies parameters that primarily control network communications. It contains top-level parameters, plus nested transport parameters. A transport is a grouping of parameter settings that are specific to a particular type of network connection.

Note

A subset of all the configuration options listed below related to SSL can be specified through Java system properties, security file properties, or through the KVStoreConfig API. For more information, see SSL communication properties (page 27).

Top-level parameters

The following top-level parameters can be set to the security.xml file:

- internalAuth
  Specifies how internal systems authenticate. This parameter must be set to SSL.

- keystore
  Identifies the keystore file within the security directory. This parameter is normally set to store.keys.

- keystoreType
  Identifies the type of keystore that the keystore property references. If not set, the Java default keystore type is assumed.

- securityEnabled
  To enable security this parameter must be set to true.

- certMode
  Specifies the key/certificate management model in use. This must be set to “shared”.

- truststore
  Identifies the truststore file within the security directory. This is normally set to store.trust.

- keystoreType
  Identifies the type of keystore that the truststore property references. If not set, the Java default keystore type is assumed.
• walletDir
  Identifies a directory within the security directory that contains a wallet password store, which in turn holds the password for the keystore.

• passwordFile
  Identifies a file within the security directory that contains a file password store, which in turn holds the password for the keystore.

**Transport parameters**

There are three standard transport types:

• ha
  Controls the communications between the data replication layer.

• client
  Controls most RMI communication.

• internal
  Controls the SSL internal authentication mechanism.

The following parameters can be set and associated to a transport type:

• transportType
  This parameter should be set to SSL.

• serverKeyAlias
  The keystore alias that identifies the keypair used by the server end of a connection.

• clientKeyAlias
  The keystore alias that identifies the keypair used by the client end of a connection.

• clientAuthRequired
  Should always be true for ha and internal transports and should be false for client transports.

• clientIdentityAllowed
  When clientAuthRequired is true, this specifies what client identification check should be applied. This should be set to dnmatch(XXX) where XXX is the Distinguished name from the client certificate.

• serverIdentityAllowed
This specifies what server verification should be performed. This should normally be set to `dnmatch(XXX)` where XXX is the Distinguished name from the server certificate.

- **allowCipherSuites**
  
  This is a comma-delimited list of SSL/TLS cipher suites that should be considered for use. For valid options, see the Java JSSE documentation corresponding to your JDK version. If not specified, the JDK default set of cipher suites is allowed.

- **allowProtocols**
  
  This is a comma-delimited list of SSL/TLS protocols that should be considered for use. For valid options, see the Java JSSE documentation corresponding to your JDK version. If not specified, the JDK default set of protocols is used.

- **clientAllowCipherSuites**
  
  See allowCipherSuites for a description of the format. This parameter sets the cipher suite requirements only for the initiating side of a connection. If set, it overrides any setting of allowCipherSuites for the connection initiator.

- **clientAllowProtocols**
  
  See allowProtocols for a description of the format. This parameter sets the protocol requirements only for the initiating side of a connection. If set, it overrides any setting of allowProtocols for the connection initiator.
Chapter 6. Encryption

Encryption of network data provides data privacy so that unauthorized parties are not able to view plain text data as it passes over the network.

Oracle NoSQL Database uses SSL-based encryption to encrypt network traffic between applications and the server, command line-utilities and the server, as well as between server components.

Note

JMX access requires the use of SSL. The web Admin interface does not operate over SSL.

SSL model

Oracle NoSQL Database uses a simple SSL key management strategy. A single, shared, RSA key is used to protect communication. In this shared key model, you must be sure that there is a master copy of the security directory and that it gets copied to each server. You should not run makebootconfig with the `-store-security configure` option on all servers. Most servers should have the `-store-security enable` option specified in their makebootconfig command.

The shared key has an associated self-signed certificate with a Subject Distinguished Name that is not server-specific. The automatically-created certificates are generated with the Distinguished Name: `CN=NoSQL`.

Each server component listens on SSL interfaces and presents the shared certificate to clients and other servers that connect to it, as proof of its authenticity. Each client and server component uses a Java truststore containing a copy of the shared certificate to validate the certificate presented by servers.

When accessing a NoSQL instance that is secured using SSL/TLS, you must specify at least the following information:

1. You must specify that the client will connect using SSL. This is done by setting the security property `oracle.kv.transport` to “ssl”.
2. You must specify the Java truststore file that is used to validate the server certificate. This is done by setting the security property `oracle.kv.ssl.trustStore`.

For example, to start runadmin in security mode use the following command:

```
java -Doracle.kv.security=mylogin.txt \ -jar KVHOME/lib/kvstore.jar runadmin
```

where the file mylogin.txt should be a copy of the client.security file with additional properties settings for authentication. The file would then contain content like this:

```
oracle.kv.auth.username=root
oracle.kv.auth.wallet.dir=login.wallet
```
oracle.kv.transport=ssl
oracle.kv.ssl.trustStore=client.trust
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)

**Note**

If you fail to correctly specify the oracle.kv.transport property or the truststore, the client will fail to connect to the server.

**SSL communication properties**

Assuming that the NoSQL server is secured by SSL, client connections from Oracle NoSQL Database administrative clients will need to connect over SSL as well. This can be achieved by providing security properties for the connection.

For Oracle-provided command line tools, a security file must be specified. The security configuration process automatically generates a basic security file (client.security) that can be used to connect to the store. You may wish to make a copy of this and modify it to include additional configuration properties.

The minimal configuration needed to connect to a secure store includes setting the following properties:

- **oracle.kv.transport=ssl**
  
  Directs KVStore clients and utilities to connect to the KVStore RMI registry via SSL.

- **oracle.kv.ssl.trustStore=<path-to-ssl-truststore>**
  
  Names a copy of the truststore file generated by makebootconfig or securityconfig to enable validation of the KVStore server SSL certificate.

  **Note**

  You can use SSL to communicate an application with other SSL servers without using truststore-based certification validation.

In addition to the two properties listed above, the following properties are also supported for control of SSL communications:

- **oracle.kv.ssl.ciphersuites**
  
  Specifies a comma-separated list of SSL cipher suites that should be allowed in communication with the server.

- **oracle.kv.ssl.protocols**
  
  Specifies a comma-separated list of SSL protocols that should be allowed in communication with the server.
• oracle.kv.ssl.trustStoreType

  Specifies the type of truststore being used. If not specified, the default type for the Java runtime is used.

  **Note**

  Applications may also set these security properties through API methods on KVStoreConfig.
Chapter 7. Configuring Authentication

Authentication means verifying the identity of someone (a user, server, or other entity) who wants to use data, resources, or applications. Validating that identity establishes a trust relationship for further interactions. Authentication also enables accountability by making it possible to link access and actions to specific identities.

Within a secure Oracle NoSQL Database, access to the database and internal APIs is generally limited to authenticated users. When a secure Oracle NoSQL Database is first started, there are no users defined, and login to the administrative interface is allowed without authentication. However, no data access operations can be performed without user authentication.

User Management

Users can be created, modified or removed in the Oracle NoSQL Database through the admin CLI. The administrative commands of manipulating users are exposed in SQL format via DDL API. Information about a specific user account as well as a summary listing of registered users can also be displayed. For more information, see the next sections describing each user management operation.

User Creation

To create a user, use the following command:

```
CREATE USER user_name IDENTIFIED BY password
[ACCOUNT LOCK|UNLOCK] [ADMIN] [PASSWORD EXPIRE]
[PASSWORD LIFETIME duration]
```

where:

- **user_name**
  The name of the user

- **IDENTIFIED Clause**
  Indicate how ONDB authenticates the users (currently by password).

- **BY password**
  Create an ONDB local user and indicates that the user must specify password to log on to the ONDB.

**Note**

The password must be quoted by double-quotation marks, for example, "123@abc".

- **ACCOUNT Clause**
  Specify ACCOUNT LOCK to lock the user's account and disable access.
• **ADMIN Clause**

Specify ADMIN to make the user granted with "sysadmin" role by default.

• **PASSWORD EXPIRE**

Specify PASSWORD EXPIRE if you want the user's password to expire immediately on first login. This setting forces the user or the user having sysadmin role to change the password before the user can log in to the database.

• **PASSWORD LIFETIME duration**

Specify the duration that current password can be used for authentication.

```
duration: [0-9]+ unit
unit: S | M | H | SECONDS | MINUTES | HOURS | DAYS
```

A simple example to create an admin user Kate with password of "123":

```
Kv-> execute 'CREATE USER Kate IDENTIFIED BY \"123\" ADMIN'
```

**Note**

To create the first user in an Oracle NoSQL Database instance, you need to log in to the Admin CLI as an anonymous user and use the "plan create-user" command. To do this, see, Performing a Secure Oracle NoSQL Database Installation (page 9).

### User Modification

To alter a user, use the following command:

```
ALTER USER user_name [IDENTIFIED BY password
[RETAIN CURRENT PASSWORD]] [CLEAR RETAINED PASSWORD] [PASSWORD EXPIRE]
[PASSWORD LIFETIME duration] [ACCOUNT UNLOCK|LOCK]
```

where:

• **user_name**

Name of user to alter.

• **IDENTIFIED Clause**

Specify BY password to specify a new password for the user.

• **RETAIN CURRENT PASSWORD**

Used with BY password clause. If specified, causes the current password defined for the user to be remembered as a valid alternate password for a limited duration (24 hours by default), or until the password is explicitly cleared. Only one alternate password may be retained at a time. This option allows a password to be changed while an application is still running without affecting its operation.

• **CLEAR RETAINED PASSWORD Clause**
Erases the current alternate retained password.

- PASSWORD EXPIRE
  Causes the user's password to expire immediately, then the user or the user having sysadmin role must change the password before attempting to log in to the database following the expiration.

- PASSWORD LIFETIME duration
  Specify the duration that current password can be used for authentication.
  
<table>
<thead>
<tr>
<th>duration: [0-9]+ unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit: S</td>
</tr>
</tbody>
</table>

- ACCOUNT Clause
  Specify ACCOUNT LOCK to lock the user's account and disable access.

**User Removal**

```
DROP USER user_name
```

Use the DROP USER user_name command to remove the specified user account (users cannot remove themselves):

where, user_name is the name of user to drop.

For example:
```
kv->execute 'DROP USER Kate'
```

**User Status**

```
SHOW [AS JSON] USERS| USER user_name
```

Use the SHOW USERS command to list all existing users in the system.

where, user_name is the name of the registered user.

For example, we have created 3 users in the system:
```
kv-> execute 'SHOW USERS'
user: id=u1 name=Ken
user: id=u2 name=Kate
user: id=u3 name=Alice
```

**Note**

These IDs are an internal mechanism for ensuring uniqueness, which will be increased sequentially.

If a user name is specified, view the detailed information of a user using the SHOW USER user_name command. For example:
```
kv->execute 'SHOW USER Kate'
```
User Login

You can use either the `-username <user>` or the `-security <path to security file>` runadmin argument to login to the admin CLI:

- `-username <user>`
  Specifies the username to log in as. This option is used in conjunction with security properties like `oracle.kv.transport`.

- `-security <path-to-security-file>`
  Specifies the security file that contains property settings for the login. Relative filename references within the security file are interpreted relative to the location of the security properties file. For example, if a security properties file contains the setting `oracle.kv.ssl.truststore=client.trust` then, the `client.trust` file should be in the same directory as the security properties file. If the file is named with an absolute path then it can be anywhere in the file system.

The following properties can be set in the file in addition to any of the SSL communication properties documented in the previous chapter:

- `oracle.kv.auth.username`
- `oracle.kv.auth.wallet.dir`
- `oracle.kv.auth.pwdfile.file`

where the `oracle.kv.auth.wallet.dir` and `oracle.kv.auth.pwdfile.file` properties in this file indicate the location of an EE wallet directory or CE password store file, respectively.

**Note**

The `oracle.kv.security` Java system property can be used as an alternative mechanism for providing a security file path. Setting this system property is equivalent to adding the `-security` option to the command line. This property is supported by all tools as well as by the KVStore client library.

Password Management

The user can configure the lifetime of users’ passwords, or make them expire immediately. When a password expires, the user needs to renew it to log in Oracle NoSQL Database successfully. The two ways to manage passwords from expiring are as follows:

- **Explicit Expiration**
  It makes current password expire immediately as well as retained password if it exists. For this user, the password must be changed before attempting to log in the database.

  For example:
```
kv->execute 'CREATE USER John IDENTIFIED BY "123" PASSWORD EXPIRE'
```

- **Password Lifetime Configuration**

  If a user logs into the database with John’s account, the user must input the new password for John.

  ```
  Logged in admin as John
  The password of John has expired, it is required to change the password.
  Enter the new password:
  Re-enter the new password:
  ```

  Password lifetime limits the duration that current password can be used for authentication.

  **Note**

  This configuration only works for current password but not the retained one.

  For example:
  ```
  kv->execute 'ALTER USER John PASSWORD LIFETIME 15 days'
  ```

  In the example above, the current password for the user John will expire after 15 days. After expiration, if the user attempts to log into the database, the user will be prompted with a notification to change the password.

  A retained password is used to allow a password to be changed while an application is still running without affecting its operation. It is only remembered by the system for a limited duration (24 hours) and there is no way to specify individual duration for each user. For retained password, only explicit expiration is supported using the following command:
  ```
  kv->execute 'ALTER USER John CLEAR RETAINED PASSWORD'
  ```

**Sessions**

When a user successfully logs in, it receives an identifier for a login session that allows a single login operation to be shared across Storage Nodes. That session has an initial lifetime associated with it, after which the session is no longer valid.

The server notifies the user with an error once the session is no longer valid. The application then needs to re-authenticate.

**Note**

The KVStoreFactory API provides a reauthentication handler, which allows the reauthentication to be completed transparently, except for the delay in reauthentication processing.

If allowed, the Oracle NoSQL Database client will transparently attempt to extend session lifetime. For best results, your application should include logic to deal with reauthentication, as operational issues could prevent it from succeeding initially. In this way, you can avoid the use of extended logic in your application to reacquire a valid session state.
You can configure the behavior regarding session management to meet the needs of the application and environment. To do this, you can modify the following parameters using the `plan change-parameters` command: `sessionTimeout`, `sessionExtendAllowed` and `loginCacheTimeout`. For more information, see Security Policy Modifications (page 45).
Chapter 8. Configuring Authorization

Oracle NoSQL Database provides role-based authorization which enables the user to assign kvstore roles to user accounts to define accessible data and allow database administrative operations for each user account.

Users can acquire desired privileges by role-granting. The user-defined role feature allows the user to create new roles using kvstore built-in privileges, and add new privilege groups to users by assigning newly-defined roles to users. You can grant users multiple roles.

For more information, see:

- Privileges (page 35)
- Roles (page 38)
- Managing Roles, Privileges and Users (page 40)

Privileges

A privilege is an approval to perform an operation on one or more Oracle NoSQL Database objects. In Oracle NoSQL Database, all privileges fall into the two general categories:

- System privileges
  This gives a user the ability to perform a particular action, or to perform an action on any data objects of a particular type.

- Object privileges
  This gives a user the ability to perform a particular action on a specific object, such as a table.

System Privileges

Oracle NoSQL Database provides the following system privileges, covering both data access and administrative operations:

- SYSDBA
  Can perform Oracle NoSQL Database management, including table create/drop/evolve, index create/drop and Avro schema create/evolve.

- SYSVIEW
  Can view/show system information, configuration and metadata.

- DBVIEW
  Can query data object information. The object is defined as a resource in Oracle NoSQL Database, subject to access control. At present, you can have this privilege to query the table, index and Avro schema's information.
• USRVIEW
  Can query users' own information, like their own user information, the status of commands they issued.

• SYSOPER
  Can perform Oracle NoSQL Database system configuration, topology management, user privilege/role management, diagnostic and maintenance operations. Allows a role to perform cancel, execute, interrupt, and wait on any plan.

• READ_ANY
  Can get/iterate keys and values in the entire store, including any tables.

• WRITE_ANY
  Can put/delete values in the entire store, including any tables.

• CREATE_ANY_TABLE
  Can create any table in the store.

• DROP_ANY_TABLE
  Can drop any table from the store.

• EVOLVE_ANY_TABLE
  Can evolve any table in the store.

• CREATE_ANY_INDEX
  Can create any index on any table in the store.

• DROP_ANY_INDEX
  Can drop any index from any table in the store.

• READ_ANY_TABLE
  Can read from any table in the store.

• DELETE_ANY_TABLE
  Can delete data from any table in the store.

• INSERT_ANY_TABLE
  Can insert and update data in any table in the store.

Object Privileges

The object privileges defined in Oracle NoSQL Database are:
• **READ_TABLE**
  Can read from a specific table in the.

• **DELETE_TABLE**
  Can delete data from a specific table in the store.

• **INSERT_TABLE**
  Can insert and update data to a specific table in the store.

• **EVOLVE_TABLE**
  Can evolve a specific table.

• **CREATE_INDEX**
  Can create indexes on a specific table.

• **DROP_INDEX**
  Can drop indexes from a specific table.

For more information on the privileges required by the user to access specific KVStore APIs as well as CLI commands, see [KVStore Required Privileges](page 54).

**Table Ownership**

When you are using a secure store, tables are owned by the user that created them. A table's owner has by default full privileges to the table. That is, the owner has all the table object privileges.

**Note**

For tables created in a non-secured store, or tables created prior to the 3.3 release, the table's owner is null.

Once a table is created, its owner cannot be changed. If a table is dropped and then recreated, all previously granted table privileges must be granted again.

Parent and child tables are required to have the same owner. However, table privileges are not automatically granted to the table's children. For example, if **READ_TABLE** is granted to table myTable, then that privilege is not automatically granted to any of that table's children. To grant **READ_TABLE** to the child tables, you must individually grant the privilege to each child table in turn.

A table's owner can grant or revoke all table privileges to or from other roles. To do this, use the **GRANT** DDL statement. (See [Grant Roles or Privileges](page 42) for details.) To make a user other than the owner be able to read/insert/delete a specific table, two conditions must be met:
1. The user has the read/insert/delete privilege for the table in question; and
2. The user has the same privilege, or read privilege, for all parent tables of that table.

For example, for table `myTable` and its child `myTable.child1`, a non-owner user can only insert data to `myTable.child1` when she has insert privilege (or better) on `myTable.child1`, and read and/or insert privilege on `myTable`.

**Privilege Hierarchy**

In Oracle NoSQL Database, there is a relationship between parts of existing privileges, called 'implications'. Implication means that a privilege may be a superset of some other privileges.

For example, Privilege A implies (=>) B means that privilege A has all the permissions defined in privilege B.

The following illustration depicts all implication relationship among Oracle NoSQL Database privileges:

![Privilege Hierarchy Diagram]

**Note**

All implications are transitive, that is, if A=>B and B=>C, then A=>C.

**Roles**

In Oracle NoSQL Database a role is a set of privileges that defines the authority and responsibility of users assigned to the role. Oracle NoSQL Database provides a set of system built-in roles. Users can create new roles to group together privileges or other roles.

**System Built-in Roles**

The following system roles are predefined:

- **readonly**
  
  Contains the READ ANY privilege. Users with this role can read all data in the KVStore.

- **writeonly**
  
  Contains the WRITE ANY privilege. Users with this role can write to the entire KVStore.
• **readwrite**
  
  Contains both the READ_ANY and WRITE_ANY privileges. Users with this role can both read and write the entire KVStore.

• **dbadmin**
  
  Contains the SYSDBA privilege. Users with this role can execute data definition operations, including table, index and Avro schema administration.

• **sysadmin**
  
  Contains the SYSDBA, SYSVIEW and SYSOPER privileges. Users with this role can execute the same operations as dbadmin, and have the ability of executing all Oracle NoSQL Database management tasks. A user created with the -admin option is granted with the sysadmin role besides the default public role.

• **public**
  
  Contains the USRVIEW and DBVIEW privileges. A default role for all Oracle NoSQL Database users, which cannot be revoked. Users with this role can login to database, view and change their own user information, as well as check and operate the plans owned by them. Users with this role can also obtain a read-only view of the data object information, for example, table names, indices, and others.

### User-Defined Roles

Oracle NoSQL Database allows the user to create new roles using kvstore built-in privileges, and add new privilege groups to users by assigning defined roles to the users. To perform role and privilege granting and revocation operations, the user must have a role having SYSOPER privilege, for example, the sysadmin role.

To manage user-defined roles, use the following commands:

```plaintext
kv-> execute 'CREATE ROLE role_name'

kv-> execute 'DROP ROLE role_name'
```

**Note**

The names of user-defined roles are case-insensitive, and are not the same as any existing privilege names or names of system built-in roles. Also, a reserved keyword cannot be used as a role name. For a list of reserved keywords, see Name Constraints in the Table Data Definition Language Overview appendix of the Oracle NoSQL Database Getting Started with the Table API guide.

The following example shows how to create user-defined roles and grant/revoke them to/from the users:

Create 2 users using the following commands:

```plaintext
kv-> execute 'CREATE USER Ken IDENTIFIED BY "123"'
```
Now, for example, create 2 roles - 'manager' with the 'write_any' privilege and 'employee' with 'read_any' privilege:

- `CREATE ROLE manager`
- `GRANT WRITE_ANY TO manager`
- `CREATE ROLE employee`
- `GRANT READ_ANY TO employee`

Role 'employee' is granted to 'manager' (sub-role of manager). Then grant role manager to user Kate, who will have both 'manager' and 'employee' role. In this way, she can read and write any data to the store.

For example:

- `GRANT employee TO ROLE manager`
- `GRANT manager TO USER Kate`

Use the following command to see the user's role status:

- `SHOW USER Kate`
  
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>enabled</th>
<th>type</th>
<th>retain-passwd</th>
<th>granted-role</th>
</tr>
</thead>
<tbody>
<tr>
<td>u2</td>
<td>Kate</td>
<td>true</td>
<td>LOCAL</td>
<td>inactive</td>
<td>[public, manager]</td>
</tr>
</tbody>
</table>

Once the user drops a role, this role and its sub-roles will be revoked automatically from any users and user-defined roles having this role. However, all of its sub-roles will not be removed from the Oracle NoSQL Database.

For example:

- `DROP ROLE manager`
- `SHOW USER Kate`
  
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>enabled</th>
<th>type</th>
<th>retain-passwd</th>
<th>granted-role</th>
</tr>
</thead>
<tbody>
<tr>
<td>u2</td>
<td>Kate</td>
<td>true</td>
<td>LOCAL</td>
<td>inactive</td>
<td>[public]</td>
</tr>
</tbody>
</table>

Now, the `SHOW ROLES` command will list the roles in the system without the 'manager' role.

If the administrator decides to drop the 'manager' role, the role manager is revoked from the user Kate automatically as well as the role 'employee'. In the above example, Kate cannot perform any read or write operations.

**Note**

Circular granting of roles is not allowed. For example, role 'manager' cannot be granted to role 'employee' if role 'employee' has previously been granted to role 'manager'.

**Managing Roles, Privileges and Users**

Oracle NoSQL Database provides a set of declarative language of security operations, including the commands allowing to create, drop, show, grant or revoke roles to or from users, and
grant or revoke privileges to or from roles. All these statements can be executed via the “execute” command in Admin CLI, or the API of KVStore.execute() or KVStore.executeSync().

**Role Creation**

```
CREATE ROLE role_name
```

Where, `role_name` is the name of the role, which is case insensitive.

For example,
```
kv-> execute 'CREATE ROLE administrator'
Statement completed successfully
```

**Role Removal**

```
DROP ROLE role_name
```

Where, `role_name` is the name of the role, which is case insensitive.

For example,
```
kv-> execute 'DROP ROLE administrator'
Statement completed successfully
```

**Role Status**

```
SHOW [AS JSON] ROLES | ROLE role_name
```

Where, `role_name` is the name of the role.

List all available role names by running 'SHOW ROLES', or view the detailed information of a role if the role name is specified.

For example,
```
kv->execute 'SHOW ROLES'
role:name=dbadmin
role:name=public
role:name=readonly
role:name=readwrite
role:name=sysadmin
role:name=writeonly
```

The detailed information of a role can be viewed by specifying the role name:
```
kv->execute 'SHOW ROLE dbadmin'
name=dbadmin assignable=true readonly=true
granted-privileges=[SYSDBA, DBVIEW]
```

**Note**

Assignable indicates whether this role can be explicitly granted to or revoked from a user.
Object privileges will appear in the form of PRIVILEGE(obj). For example, privilege of READ_TABLE on table 'emp' will appear as:

```
kv->execute 'CREATE ROLE emtablereader'
kv->execute 'GRANT READ_TABLE ON emp TO emtablereader'
kv->execute 'SHOW ROLE emtablereader'
name=emtablereader assignable=true readonly=false
granted-privileges=[READ_TABLE(emp)]
```

**Grant Roles or Privileges**

```
GRANT { grant_roles | grant_system_privileges
| grant_object_privileges }
grant_roles ::= role [, role]... TO { USER user | ROLE role }
grant_system_privileges ::= 
{system_privilege | ALL PRIVILEGES}
[,]{system_privilege | ALL PRIVILEGES}]
TO role
grant_object_privileges ::= 
{object_privileges| ALL [PRIVILEGES]}
[,]{object_privileges| ALL [PRIVILEGES]}]
ON object TO role
```

where:

- **role**
  The role that is granted.

- **user**
  The user to which the privileges are granted.

- **system_privileges**
  The system privileges that are granted.

- **object_privileges**
  The object privileges that are granted.

- **object**
  The object on which the privilege is granted. Currently only table privileges are supported.

- **ALL PRIVILEGES**
  Grants all of the system privileges. This is a shortcut for specifying all system privileges.

- **ALL [PRIVILEGES]**
  Grants all object privileges defined for the object. The keyword PRIVILEGES is provided for semantic clarity and is optional.
For example, to grant a role 'manager' to another role 'employee':

```
kv-> execute 'GRANT EMPLOYEE TO ROLE manager'
kv-> execute 'GRANT MANAGER TO USER Kate'
```

Statement completed successfully

If you repeat the command:

```
kv-> execute 'GRANT employee to ROLE manager'
```

You will receive an error of "Could not recursively grant role employee to role manager" because this would lead to a cyclic definition of role manager.

The user can now add new privileges to their defined role. For example:

```
kv-> execute 'GRANT READ_ANY TO Kate'
```

For example, to grant read permission on table T1 to Kate:

```
kv-> execute 'GRANT READ_TABLE on T1 TO Kate'
```

See also notes on granting table privileges in Table Ownership (page 37).

**Revoke Roles or Privileges**

```
REVOKE { revoke_roles | revoke_system_privileges
| revoke_object_privileges
| revoke_roles ::= role [, role]... FROM { USER user | ROLE role }
| revoke_system_privileges ::= { system_privilege | ALL PRIVILEGES }
| [, {system_privilege | ALL PRIVILEGES}]... FROM role
| revoke_object_privileges ::= { object_privileges| ALL [PRIVILEGES] }
| [, { object_privileges | ALL [PRIVILEGES] }]... ON object FROM role

where:

• role
  The role to revoke.

• user
  The user from which the privileges are revoked.

• system_privileges
  The system privileges to revoke.

• object_privileges
  The object privileges to revoke.
```
• **object**

The table from which the privileges are revoked. Currently, the only objects supported are tables.

• **ALL PRIVILEGES**

Revokes all of the system privileges that have been granted to the revokee.

• **ALL [PRIVILEGES]**

Revokes all object privileges defined on the object from the revokee. The keyword PRIVILEGES is provided for semantic clarity and is optional.

For example, to revoke role 'employee' from role 'manager':

```
kv-> execute 'REVOKE employee FROM ROLE manager'
Statement completed successfully
```

To revoke the role 'manager' from user 'Kate':

```
kv-> execute 'REVOKE manager FROM USER Kate'
Statement completed successfully
```
Chapter 9. Security Policies

The following default policies in Oracle NoSQL Database may be used to tailor system behavior to meet your security requirements:

- Login sessions have a limited duration of validity. After that duration has passed, the session needs re-authentication.
- Session login errors are tracked at the component level. Access to an account for a single client host is temporarily disabled if too many failed logins occur at that component within a configurable time duration.

Note
Both of these behaviors can be customized by modifying the values of their respective security parameters. For more information, see the following section.

Security Policy Modifications

You can use the `plan change-parameters` command in order to change a security policy in the system:

```
plan change-parameters -security <id>...
```

Security parameters are applied implicitly and uniformly across all SNs, RNs and Admins.

The following security parameters can be set:

- `sessionTimeout=<Long TimeUnit>`
  Specifies the length of time for which a login session is valid, unless extended. The default value is 24 hours.
- `sessionExtendAllowed=<Boolean>`
  Indicates whether session extensions should be granted. Default value is true.
- `accountErrorLockoutThresholdInterval=<Long TimeUnit>`
  Specifies the time period over which login error counts are tracked for account lockout monitoring. The default value is 10 minutes.
- `accountErrorLockoutThresholdCount=<Integer>`
  Number of invalid login attempts for a user account from a particular host address over the tracking period needed to trigger an automatic account lockout for a host. The default value is 10 attempts.
- `accountErrorLockoutTimeout=<Long TimeUnit>`
  Time duration for which an account will be locked out once a lockout has been triggered. The default value is 30 minutes.
• loginCacheTimeout=<Long TimeUnit>

Time duration for which KVStore components cache login information locally to avoid the need to query other servers for login validation on every request. The default value is 5 minutes.
Chapter 10. Audit Logging

Oracle NoSQL Database monitors and records security sensitive activities. These log messages are available through the SN-local log files and the store-wide logging view. High risky security activities are also visible by using the show events command.

Security Log Messages

For ease of grepping and analysis, the auditing log message uses KVAuditInfo as a prefix. For example:

```
# General audit logging:
<Timestamp>: KVAuditInfo[user: <user_name>,
clienthost: <client_host>, operation:
<operation_description>, status: <SUCCESS/FORBIDDEN>,
reason: <failure_reason>]
```

```
# General audit logging:
# Particular logging for successful execution of plan:
<Timestamp>: KVAuditInfo[<plan_name>, owned by <plan_owner>,
executed by <plan_executor> from <client_host>,
state=<end state of plan execution>]
```

To distinguish security related messages from standard log messages, the following two security related logging levels are introduced:

- **SEC_WARNING**

  Logs unauthenticated login, unauthorized read/write data access and unauthorized execution of CLI commands. Unauthenticated login does not log the reasons of failure.

- **SEC_INFO**

  Logs the success of a user login and the successful execution of plans that require dbadmin or sysadmin role related privileges.
Chapter 11. Keeping Oracle NoSQL Database Secure

This chapter provides a set of guidelines to keep your Oracle NoSQL Database secure. To maximize the security features offered by Oracle NoSQL Database, it is imperative that the database itself be well protected.

Security guidelines provide advice about how to securely configure Oracle NoSQL Database by recommending security practices for operational database deployments.

Guidelines for Securing the Configuration

Follow these guidelines to keep the security configuration secure:

- The initial security configuration should be generated on a host that is not intended for KVStore operational use, using the securityconfig create config command.

- Storage Nodes should be deployed by running makebootconfig with the -store-security enable argument. The configured security directory from the reference host should be copied to the new Storage Node KVROOT using a secure copy mechanism prior to starting the store.

- The security configuration should be kept in a protected location for future use.

- Updates to the security configuration should be performed on the configuration host and copied to the operational Storage Node hosts using a secure copy mechanism.

- After the first user is configured but before allowing applications to use the store, you may wish to restart all SNA processes on hosts running Admin processes and then use the Admin CLI show users command to ensure that there is only the single user definition that is expected. This step validates that no other user creation occurred during the period when administrative login was not required.

Guidelines for Deploying Secure Applications

Follow these guidelines when deploying your Oracle NoSQL Database and if the properties include oracle.kv.auth.wallet.dir in order to use Oracle wallet to hold a user password:

- Include the kvstore-ee.jar file in the application classpath.

- The kvstore-ee.jar, oraclepki.jar, osdt_cert.jar, osdt_core.jar files should all be made available on the application machine.

  Note
  kvstore-ee references the other files, so they do not need to be included in the classpath explicitly.
Guidelines for Securing the SSL protocol

Follow these guidelines to keep the SSL protocol secure:

• When configuring SSL communication for your store, you should consider both performance and security.

• For a more secure store you should opt for higher security where possible.

• The Oracle JDK 7 supports TLSv1.2 as an SSL protocol level, whereas JDK 6 provides only TLSv1 as its highest protocol level.

• If you are currently using JDK 6, it is strongly recommended that you upgrade to JDK 7.

Guidelines for using JMX securely

Follow these guidelines to securely use your Java Management Extensions (JMX) agent:

• If you enable JMX for a secure store, your JMX monitoring application must access the store using SSL.

• You should consult the configuration details for the JMX product you wish to use. In this case, you can use jconsole with a secure store by running the following command:

  jconsole -J-Djavax.net.ssl.trustStore=/home/nosql/client.trust 
  node01:5000

where node01 is the registry host to be monitored and 5000 is the registry port configured for the Storage Node.

Guidelines for Updating Keystore Passwords

Follow these steps to update the keystore passwords:

1. In the security directory on the configuration host run the keytool command. The keytool prompts for the current password and then for a new password to set.

   keytool -storepasswd -keystore store.keys

2. If using a Password File store, skip ahead to the next step. To update the keystore password for wallets, use the following command:

   java -jar KVHOME/lib/kvstore.jar securityconfig \ 
   wallet secret -directory store.wallet -set -alias keystore

   Securityconfig will prompt for the new password. The new password should match the new one provided earlier to the keytool command.

3. If using Password File stores instead of wallets, use the following command to update the keystore password:

   java -jar KVHOME/lib/kvstore.jar securityconfig \
pwdfile secret -file store.pwd -set -alias keystore

Securityconfig will prompt for the new password. The new password should match the new one provided earlier to the keytool command.

4. Copy the updated store.keys file and either store.pwd or the contents of store.wallet to the security directory on each host and restart the Storage Node using the following commands:

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

**Guidelines for Updating the SSL key/certificate**

Follow these steps to update the SSL key/certificate:

1. On the configuration host, run securityconfig to create a new configuration in a directory in parallel to the standard configuration directory.

2. On the configuration host, merge the truststore entries by using the `config merge-trust` command:

```bash
java -jar KVHOME/lib/kvstore.jar securityconfig \ config merge-trust -root <standard config dir> \ -source-root <new config dir>
```

3. In the security directory on the configuration host run the keytool command. The keytool prompts for the current password and then for a new password to set.

```bash
keytool -storepasswd -keystore store.keys
```

Securityconfig will prompt for the new password. The new password should match the new one provided earlier to the keytool command.

4. If using a Password File store, skip ahead to the next step. To update the keystore password for wallets, use the following command:

```bash
java -jar KVHOME/lib/kvstore.jar securityconfig \ wallet secret -directory store.wallet -set -alias keystore
```

Securityconfig will prompt for the new password. The new password should match the new one provided earlier to the keytool command.

5. If using Password File stores instead of wallets, use the following command to update the keystore password:

```bash
java -jar KVHOME/lib/kvstore.jar securityconfig \ pwdfile secret -file store.pwd -set -alias keystore
```

Securityconfig will prompt for the new password. The new password should match the new one provided earlier to the keytool command.
6. Copy the updated store.keys file and either store.pwd or the contents of store.wallet to the security directory on each host and restart the Storage Node using the following commands:

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

Guidelines for Operating System Security

Follow these guidelines regarding operating system security:

- There should be a single user identity that runs the KVStore software.
- The KVStore user should be in its own group, independent of other users.
- JE log files, audit log files, and password stores should have mode 0600 on Linux/UNIX platforms with equivalent settings for Windows systems. The simplest way to achieve this on Linux/UNIX is to set an umask of 0077.
- Security configuration files must be write-protected.
- The KVROOT directory and the security directory must be protected from modification by other users. On UNIX/Linux this should include having the sticky bit (01000) set in order to prevent renaming and deletion of files/directories.
- Access to the systems that are running KVStore should be limited in order to avoid the risk of tampering.

Note

Access protections do not guard against users who have sufficiently elevated access rights (for example, the UNIX root user).
Appendix A. SSL keystore generation

The keystores (store.keys and store.trust) that are automatically generated by makebootconfig or securityconfig can also be manually created using the following keytool commands:

To generate the keypair, use the keytool -genkeypair command:

```
keytool -genkeypair \
-keystore store.keys \
-storepass <passwd> \
-keypass <passwd> \
-alias shared \n-dname "CN=NoSQL" \n-keyAlg RSA \n-keysize 1024 \n-validity 365
```

To export the keypair, use the keytool -export command:

```
keytool -export \
-file <temp file> \
-keystore store.keys \
-storepass <passwd> \
-alias shared
```

To import the keypair, use the keytool -import command:

```
keytool -import \
-file <temp file> \
-keystore store.keys \
-storepass <passwd> \
-noprompt
```

You can also use the keytool commands described above to manually generate other keystore and truststore keys and substitute them for the ones that Oracle NoSQL Database generates, provided you adhere to the following rules:

- The store.keys file should have a key pair with the alias "shared".
- The store.keys store password (-storepass) must match the key password (-keypass)
- If a subject distinguished name other than CN=NoSQL is chosen for the self-signed certificate, then you must specify the following options to the makebootconfig or securityconfig command:

```
-param "ha:serverIdentityAllowed=dnmatch(SOMEDN)"
-param "ha:clientIdentityAllowed=dnmatch(SOMEDN)"
```
-param "internal:serverIdentityAllowed=dnmatch(SOMEDN)"
-param "internal:clientIdentityAllowed=dnmatch(SOMEDN)"
-param "client:serverIdentityAllowed=dnmatch(SOMEDN)"

where SOMEDN is the distinguished name (-dname) chosen.

- The store password for store.trust should match the store password for store.keys.
Appendix B. KVStore Required Privileges

This section lists the user required privileges to access specific KVStore APIs as well as CLI commands.

Privileges for Accessing CLI Commands

READ_ANY:
- get kv

READ_ANY_TABLE or READ_TABLE (on $table_name):
- get table -name table_name

WRITE_ANY:
- delete kv
- put kv

INSERT_ANY_TABLE or INSERT_TABLE (on $table_name):
- put table -name table_name

DELETE_ANY_TABLE or DELETE_TABLE (on $table_name):
- delete table -name table_name

SYSDBA:
- ddl
- plan add-index
- plan add-table
- plan evolve-table
- plan remove-index
- plan remove-table

CREATE_ANY_TABLE:
- plan add-table

DROP_ANY_TABLE:
• plan remove-table

EVLOLVE_ANY_TABLE or EVOLVE_TABLE (on $table_name):

• plan evolve-table -name table_name

CREATE_ANY_INDEX or CREATE_INDEX (on $table_name):

• plan add-index -table table_name

DROP_ANY_INDEX or DROP_INDEX (on $table_name):

• plan remove-index -table table_name

SYSVIEW:

• await-consistency

• logtail

• ping

• show admins

• show events

• show topology

• show upgrade-order

• show users (all users)

• show zones

• verify

• show parameters

• show perf

• show plans (plans created by all users)

• show pools

• show snapshots

DBVIEW:

• show indexes

• show schemas

• show tables
USRVIEW:

• show users (for self)
• show plans (plans created by self)
• plan change-user (for self)

DBVIEW and READ_ANY:

• aggregate

SYSOPER:

• change-policy
• configure
• plan change-parameters
• plan change-storagedir
• plan change-user (for all users)
• plan deploy-admin
• plan deploy-datacenter
• plan deploy-sn
• plan deploy-topology
• plan deploy-zone
• plan drop-user
• plan failover
• plan grant
• plan migrate-sn
• plan remove-admin
• plan remove-sn
• plan remove-zone
• plan repair-topology
• plan revoke
• plan start-service
• plan stop-service
• pool (all sub-commands)
• repair-admin-quorum
• snapshot (all sub-commands)
• topology (all sub-commands)

No privilege is required for the following commands:
• connect
• exit
• help
• hidden
• history
• verbose
• show faults
• table (all sub-commands)

Privilege required depends on the command being timed:
• time

Privilege required depends on the commands contained in the script file:
• load

Privilege required depends on the privilege needed for the plan being referred to:
• plan cancel
• plan execute
• plan interrupt
• plan wait

**Privileges for DDL Commands**

SYSDBA:
• CREATE TABLE
• CREATE INDEX
• DROP INDEX
• DROP TABLE
• ALTER TABLE

CREATE_ANY_TABLE:
• CREATE TABLE

DROP_ANY_TABLE:
• DROP TABLE

EVOLVE_ANY_TABLE or EVOLVE_TABLE (on $table_name):
• ALTER TABLE table_name

CREATE_ANY_INDEX or CREATE_INDEX (on $table_name):
• CREATE INDEX ON table_name

DROP_ANY_INDEX or DROP_INDEX (on $table_name):
• DROP INDEX ON table_name

SYSOPER:
• CREATE USER
• CREATE ROLE
• DROP USER
• DROP ROLE
• ALTER USER
• GRANT
• REVOKE

DBVIEW:
• SHOW TABLE
• SHOW INDEX
• DESCRIBE TABLE
• DESCRIBE INDEX

SYSVIEW:
Privileges for Accessing KVStore APIs

Privilege(s) required: READ_ANY, or READ_TABLE/READ_ANY_TABLE if accessing key-values are in tables.

• get
• multiGet
• multiGetIterator
• multiGetKeys
• multiGetKeysIterator

Note

For multi-XYZ and storeXYZiterator APIs, the parentKey may be null for scanning the whole store. In this case, if the user has no required roles, an empty result set will be returned rather than throwing the UnauthorizedException.

• storeIterator
• storeKeysIterator

Privilege(s) required: WRITE_ANY, or DELETE_TABLE/DELETE_ANY_TABLE if accessing key-values are in tables:

• delete
• deleteIfVersion
• multiDelete

Privilege(s) required: WRITE_ANY, or INSERT_TABLE/INSERT_ANY_TABLE if accessing key-values are in tables:

• put
• putIfAbsent
• putIfPresent
• putIfVersion
Privilege(s) required: DBVIEW
• getAvroCatalog

Privilege(s) required: None:
• getOperationFactory
• getStats

Privilege(s) required: Union of all required roles of each single operation in the operation list:
• execute

Privilege required depends on the privilege needed for the statement being executed:
• execute(String statement)
• executeSync(String statement)

**Privileges for Accessing KVStore TableAPIs**

Privileges(s) required: READ_TABLE/READ_ANY_TABLE:
• get
• multiGet
• multiGetKeys
• tableIterator
• tableKeysIterator

Privilege(s) required: DELETE_TABLE/DELETE_ANY_TABLE:
• delete
• deleteIfVersion
• multiDelete

Privilege(s) required: INSERT_TABLE/INSERT_ANY_TABLE:
• put
• putIfAbsent
• putIfPresent
• putIfVersion

Privilege(s) required: USRVIEW:
• getTable
• getTables

Privilege(s) required: None:

• getTableOperationFactory

Privilege(s) required: Union of all required roles of each single operation in the operation list:

• execute

Privileges for Accessing KvLargeObject APIs

Privilege(s) required: READ_ANY:

• getLOB

Privilege(s) required: READ_ANY and WRITE_ANY:

• appendLOB
• deleteLOB
• putLOB
• putLOBIfAbsent
• putLOBIfPresent
Appendix C. Third Party Licenses

All of the third party licenses used by Oracle NoSQL Database are described in the LICENSE.txt file, which you can find in your KVHOME directory.