Oracle® NoSQL Database

Security Guide

Release 12.2.4.5
E85375-01
February 2018

Oracle NoSQL Database Security Guide, Release 12.2.4.5

E85375-01

Contents

Preface

Conventions Used in This Book vii

1 Introducing Oracle NoSQL Database Security

2 Security Configuration

Security Configuration Overview 2-1
Configuring Security with Makebootconfig 2-3
Configuring Security with Securityconfig 2-4
- Creating the security configuration 2-4
- Adding the security configuration 2-6
- Verifying the security configuration 2-7
- Updating the security configuration 2-8
- Showing the security configuration 2-8
- Removing the security configuration 2-9
- Merging truststore configuration 2-10

3 Performing a Secure Oracle NoSQL Database Installation

Single Node Secure Deployment 3-1
- Adding Security to a New Installation 3-1
- Adding Security to an Existing Installation 3-3

Multiple Node Secure Deployment 3-5
- Adding Security to a New Installation 3-5
- Adding Security to an Existing Installation 3-9

4 Kerberos Authentication Service

Installation Prerequisites 4-1
Kerberos Principal 4-1
Keytabs 4-2

Kadmin and kadmin.local 4-2
Kerberos Security Properties 4-2
Setting Security Properties in a security login file 4-3
Setting Security Properties through KVStoreConfig 4-4
Using Security Properties to Log In 4-4
Using credential cache 4-5
Using a keytab 4-6
JAAS programming framework integration 4-6
Performing a Secure Oracle NoSQL Database Installation with Kerberos 4-8
Adding Kerberos to a New Installation 4-8
Adding Kerberos to an Existing Secure Installation 4-12
Using Oracle NoSQL Database with Kerberos and Microsoft Active Directory (AD) 4-14

5 External Password Storage

Oracle Wallet 5-1
Password store file 5-2

6 Security.xml Parameters

Top-level parameters 6-1
Transport parameters 6-2

7 Encryption

SSL model 7-1
SSL communication properties 7-2

8 Configuring Authentication

User Management 8-1
User Creation 8-1
User Modification 8-2
User Removal 8-3
User Status 8-3
User Login 8-4
Password Management 8-5
Sessions 8-6

9 Configuring Authorization

Privileges 9-1

System Privileges 9-1
Object Privileges 9-2
Table Ownership 9-3
Privilege Hierarchy 9-3
Roles 9-4
System Built-in Roles 9-4
User-Defined Roles 9-5
Managing Roles, Privileges and Users 9-6
Role Creation 9-6
Role Removal 9-6
Role Status 9-7
Grant Roles or Privileges 9-7
Revoke Roles or Privileges 9-8

10 Security Policies

Security Policy Modifications 10-1

11 Audit Logging

Security Log Messages 11-1

12 Keeping Oracle NoSQL Database Secure

Guidelines for Securing the Configuration 12-1
Guidelines for Deploying Secure Applications 12-1
Guidelines for Securing the SSL protocol 12-2
Guidelines for using JMX securely 12-2
Guidelines for Updating Keystore Passwords 12-2
Guidelines for Updating Kerberos Passwords 12-3
Guidelines for Updating the SSL key/certificate 12-5
Guidelines for Configuring External Certificates for a new Installation 12-6
Guidelines for Configuring External Certificates for an Existing Default Secure Installation 12-8
Guidelines for Updating the External Certificates 12-10
Guidelines for Operating System Security 12-12

A Password Complexity Policies

B SSL keystore generation

C Java KeyStore Preparation

Import Key Pair to Java Keystore C-2

D KVStore Required Privileges

Privileges for Accessing CLI Commands D-1
Privileges for DDL Commands D-4
Privileges for Accessing KVStore APIs D-5
Privileges for Accessing KVStore TableAPIs D-6
Privileges for Accessing KvLargeObject APIs D-6

E Configuring the Kerberos Administrative Utility

F Manually Registering Oracle NoSQL Database Service Principal

G Third Party Licenses

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.

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 following Oracle NoSQL Database features to configure security for your Oracle NoSQL Database installation:

- `Security Configuration Utility`. Allows you to configure and add security to a new or to an existing Oracle NoSQL Database installation.
- `Authentication methods`. Oracle NoSQL Database provides password authentication for users and systems. The EE version of Oracle NoSQL Database also supports Kerberos authentication.
- `Encryption`. Data is encrypted 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 provides guidelines that you should follow when securing your Oracle NoSQL Database installation.

> **Note:**
>
> Full Text Search (see Introduction in the *Oracle NoSQL Database Full Text Search* guide) and a secure Oracle NoSQL Database store are disjoint, that is, if Oracle NoSQL Database is configured as a secure store, Full Text Search should be disabled. On the other hand, if Full Text Search is enabled (that is, an external Elasticsearch cluster is registered) in a nonsecure store, users cannot reconfigure the nonsecure store to a secure store, unless Full Text Search is disabled before reconfiguration.
>
> For more information, see Security Configuration in the *Oracle NoSQL Database Full Text Search* guide.

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.

> **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.

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.

Th `securityconfig` 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.

> **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:

```
security/security.xml
security/store.keys
security/store.trust
security/store.passwd  (CE or EE installations)
security/store.wallet  (EE installations only)
security/store.wallet/cwallet.sso (EE installations only)
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 Oracle NoSQL Database. 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 `-store-security` option to set up the basic store configuration with security:

```
java -jar KVHOME/lib/kvstore.jar makebootconfig
-root <kvroot> -port <port>
-host <hostname> -harange <harange>
-store-security configure -capacity <capacity>
[-secdir <security dir>]
[-pwdmgr {pwdfile | wallet | <class-name>}]
[-kspwd <password>]
[-external-auth {kerberos}]
[-krb-conf <kerberos configuration>]
[-kadmin-path <kadmin utility path>]
[-instance-name <database instance name>]
[-admin-principal <kerberos admin principal name>]
[-kadmin-keytab <keytab file>]
[-kadmin-ccache <credential cache file>]
[-princ-conf-param <param=value>]*
[-security-param <param=value>]*
[-noadmin]
```

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.

  Use the `-external-auth` option to specify Kerberos as an external authentication service. This option is only available in the Oracle NoSQL Database EE version. If

information for the Kerberos admin interface (e.g. *password*) is needed and no keytab or credential cache has been specified on the command line, an interactive version of securityconfig `config create` utility will run.

Using the `-external-auth` flag allows Oracle NoSQL Database to generate the security files needed for Kerberos authentication, based on a standard configuration. Although not recommended, it is possible to use a non-standard configuration. To do this, see Manually Registering Oracle NoSQL Database Service Principal

- `-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.

> **Note:**
>
> The `-store-security` command is optional. Even if the user does not specify `-store-security`, it would be enabled by default.

For more information on configuring security with makebootconfig, see Adding Security to a New Installation.

Configuring Security with Securityconfig

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

```
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:

```
config create
-root <secroot> [ -secdir <security dir> ]
[-pwdmgr { pwdfile | wallet <class-name>} ]
[-kspwd <password> ]
[-external-auth {kerberos}]
  [-krb-conf <kerberos configuration>]
  [-kadmin-path <kadmin utility path>]
  [-instance-name <database instance name>]
  [-admin-principal <kerberos admin principal name>]
  [-kadmin-keytab <keytab file>]
  [-kadmin-ccache <credential cache file>]
  [-princ-conf-param <param=value>]*
[-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.

- `-external-auth {kerberos}` Specifies Kerberos as an external authentication service. This option is only available in the Oracle NoSQL Database EE version. If no keytab or credential cache has been specified on the command line, an interactive version of the `securityconfig` utility will run.

  Using this flag allows Oracle NoSQL Database to generate the security files needed for Kerberos authentication, based on a standard configuration. Although not recommended, it is possible to use a non-standard configuration. To do this, see Manually Registering Oracle NoSQL Database Service Principal

  This flag is only permitted when the value of the `-store-security` flag is specified as configure or enable.

  To remove Kerberos authentication from a running store, set the value of the `userExternalAuth` security.xml parameter to `NONE`.

  where `-external-auth` can have the following flags:

  – `-admin-principal <kerberos admin principal name>`

    Specifies the principal used to login to the Kerberos admin interface. This is required while using kadmin keytab or password to connect to the admin interface.

  – `-kadmin-ccache <credential cache file>`

    Specifies the complete path name to the Kerberos credentials cache file that should contain a service ticket for the kadmin/ADMINHOST. `ADMINHOST` is the fully-qualified hostname of the admin server or kadmin/admin service.

    If not specified, the user is prompted to enter the password for principal while logging to the Kerberos admin interface. This flag cannot be specified in conjunction with the `-kadmin-keytab` flag.

  – `-kadmin-keytab <keytab file>`

    Specifies the location of a Kerberos keytab file that stores Kerberos admin user principals and encrypted keys. The security configuration tool will use the specified keytab file to login to the Kerberos admin interface.

    The default location of the keytab file is specified by the Kerberos configuration file. If the keytab is not specified there, then the system looks for the file `user.home/krb5.keytab`.

    You need to specify the `-admin-principal` flag when using keytab to login to the Kerberos admin, otherwise the correct admin principal will not be recognized. This flag cannot be specified in conjunction with the `-kadmin-ccache` flag.

  – `-kadmin-path <kadmin utility path>`

    Indicates the absolute path of the Kerberos kadmin utility. The default value is `/usr/kerberos/sbin/kadmin`.

  – `-krb-conf <kerberos configuration>`

    Specifies the location of the Kerberos configuration file that contains the default realm and KDC information. If not specified, the default value is `/etc/krb5.conf`.

  – `-princ-conf-param <param=value>*`

A repeatable argument that allows configuration defaults to be overridden.

Use the `krbPrincValidity` parameter to specify the expiration date of the Oracle NoSQL Database Kerberos service principal.

Use the `krbPrincPwdExpire` parameter to specify the password expiration date of the Oracle NoSQL Database Kerberos service principal.

Use the `krbKeysalt` parameter to specify the keysalt list used to generate the keytab file.

- `-secdir <security dir>`

  Specifies the name of the directory within 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

  – `-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

- `-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 Performing a Secure Oracle NoSQL Database Installation.

For more information on configuring Kerberos with securityconfig, see Kerberos Authentication Service.

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".

When using Kerberos as an external authentication service, you can use the `config add-kerberos` command to add the security configuration you created earlier:

```
config add-kerberos -root <secroot> [-secdir <security dir>]
[-krb-conf <Kerberos configuration>]
[-kadmin-path <kadmin utility path>]
[-instance-name <database instance name>]
[-admin-principal <kerberos admin principal name>]
[-kadmin-keytab <keytab file> ]
[-kadmin-ccache <credential cache file>]
[-princ-conf-param <param=value>]*
[-param <param=value>]*
```

Verifying the security configuration

You can use the `config verify` command to verify the consistency and correctness of a security configuration:

```
config verify -secdir <security dir>
```

where:

- `-secdir <securitydir>`

  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".

For example:

```
security-> config verify -secdir security
Security configuration verification passed.
```

Updating the security configuration

You can use the `config update` command to update the security parameters of a security configuration:

```
config update -secdir <security dir> [-param <param=value>]*
```

where:

- `-secdir <securitydir>`

  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".

- `-param <param=value*>`

  List of security parameters to update.

For example:

```
security-> config update -secdir security -param clientAuthRequired=false
Security parameters updated.
```

Showing the security configuration

You can use the `config show` command to print out all security configuration information.

```
config show -secdir <security dir>
```

where:

For example:

```
security-> config show -secdir security
Security parameters:
certMode=shared
internalAuth=ssl
keystore=store.keys
keystorePasswordAlias=keystore
passwordClass=oracle.kv.impl.security.filestore.FileStoreManager
passwordFile=store.passwd
securityEnabled=true
truststore=store.trust

internal Transport parameters:
clientAllowProtocols=TLSv1.2,TLSv1.1,TLSv1
clientAuthRequired=true
clientIdentityAllowed=dnmatch(CN=NoSQL)
clientKeyAlias=shared
serverIdentityAllowed=dnmatch(CN=NoSQL)
serverKeyAlias=shared
transportType=ssl

client Transport parameters:
clientAllowProtocols=TLSv1.2,TLSv1.1,TLSv1
serverIdentityAllowed=dnmatch(CN=NoSQL)
serverKeyAlias=shared
transportType=ssl
```

```
ha Transport parameters:
allowProtocols=TLSv1.2,TLSv1.1,TLSv1
clientAuthRequired=true
clientIdentityAllowed=dnmatch(CN=NoSQL)
serverIdentityAllowed=dnmatch(CN=NoSQL)
serverKeyAlias=shared
transportType=ssl

Keystore:
security/store.keys

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

shared, Jun 1, 2016, PrivateKeyEntry,
Certificate fingerprint (SHA1): A6:54:9C:42:13:66:DC:E9:A8:62:DB:
A8:87:FD:DE:23:F7:AD:11:FB

Keystore:
security/store.trust

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

mykey, Jun 1, 2016, trustedCertEntry,
Certificate fingerprint (SHA1):A6:54:9C:42:13:66:DC:E9:A8:62:DB:
A8:87:FD:DE:23:F7:AD:11:FB
```

- `-secdir <securitydir>`

  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".

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".

For example:

```
security-> config remove-security -secdir security
Configuration updated.
```

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

```
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 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".

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.

If you want to use Kerberos as an external authentication service, you should instead complete the steps under Performing a Secure Oracle NoSQL Database Installation with Kerberos.

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 `-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 \
   -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.

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. The password should comply with the security policies described in Password Complexity Policies. 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.

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.

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:

   ```
   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:

   ```
   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.

Adding Security to an Existing Installation

To add security to an existing Oracle NoSQL Database installation:

1. Shut down the KVStore instance:

   ```
   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. Oracle Wallet is only available in the Oracle NoSQL Database EE version. CE deployments should use the `pwdfile` option instead.

   ```
   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.

8. Create an admin user. The password should comply with the security policies described in Password Complexity Policies. 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.

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 \
-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.

For more information on user creation and administration, see User Management.

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 \
-Doracle.kv.security=mylogin \
-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
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

    For more information, see User Login.

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 `-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 \
-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 \
   -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 \
   -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. The password should comply with the security policies described in Password Complexity Policies. 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.

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.

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

```
kv-> plan create-user -name user1 -wait
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 \
wallet create -dir KVROOT/security/users.wallet
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig wallet secret \
-dir KVROOT/security/users.wallet -set -alias user1
Enter the secret value to store: ********
Re-enter the secret value for verification: ********
Secret created
OK
java -Xmx256m -Xms256m \
-jar KVHOME/lib/kvstore.jar securityconfig wallet secret \
-dir KVROOT/security/users.wallet -set -alias user2
Enter the secret value to store: ********
Re-enter the secret value for verification: ********
Secret created OK
```

> **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.

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 \
-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:

```
oracle.kv.auth.username=user1
oracle.kv.auth.wallet.dir=KVROOT/security/users.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.

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:
   ```

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:

```
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:

```
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:

```
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.

9. Create an admin user. The password should comply with the security policies described in Password Complexity Policies. 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.

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, 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.

4
Kerberos Authentication Service

Existing or new installations of Oracle NoSQL Database can be configured to use Kerberos as an external authentication service. Kerberos is an industry standard authentication protocol for large client/server systems.

Setting up and configuring a Kerberos deployment is beyond the scope of this chapter. This chapter assumes that you have a running Key Distribution Center (KDC) and realm setup.

This chapter first describes some Kerberos concepts and then shows you how to configure existing or new installations of Oracle NoSQL Database to use Kerberos as an external authentication service.

Installation Prerequisites

Make sure that you have Kerberos V5 installed. Oracle NoSQL Database is compatible and tested with MIT Kerberos V5.

If your Kerberos installation/keytab is configured to use a strong encryption type — for example, AES with 256-bit keys — the JCE Unlimited Strength Jurisdiction Policy Files must be obtained and installed in the JDK/JRE. Be aware that these files might already exist in your installation. If so, they must be updated.

Kerberos Principal

A Kerberos Principal represents a unique identity in a Kerberos system to which Kerberos can assign tickets to access Kerberos-aware services. A service principal should be created for each Storage Node. Oracle NoSQL Database service principals follow this naming format: <service_name>/instance@REALM.

where:

- `service_name`

  Is a case-sensitive string that represents the Oracle NoSQL Database service. The default value is `oraclenosql`.

  All Oracle NoSQL Database service principals should use the same service name across different Storage Nodes.

- `instance`

  Represents the service principal instance name. It is recommended to use the fully qualified domain name (FQDN) of the Storage Node where Oracle NoSQL Database is running.

  If `instance` is not specified, the default principal will be created as oraclenosql@REALM.

- `REALM`

Represents the Kerberos realm name where the database service is registered. It must be specified in UPPERCASE and is typically the DNS domain name.

If no realm is given, the service principal is assumed to belong to the default realm, as configured in the Kerberos configuration file.

Keytabs

A keytab is a file containing pairs of Kerberos principals and an encrypted copy of that principal's key.

Keytabs are used to authenticate a principal on a host to Kerberos.

> **Note:**
>
> Because having access to the keytab file for a principal allows one to act as that principal, access to the keytab files should be tightly secured.

Kadmin and kadmin.local

`Kadmin` and `kadmin.local` are command-line interfaces to the Kerberos administration system.

In general, both interfaces provide the same functionality. When creating Kerberos principals and keytabs, you can use `kadmin.local` or `kadmin` depending on your access and account.

For more information, see the MIT Kerberos documentation.

Kerberos Security Properties

To set up the Kerberos security properties, you can set them in a login file or through the `KVStoreConfig` class.

The minimal configuration needed to set up Kerberos includes the following properties:

- `oracle.kv.auth.username`

  Specifies the Kerberos user name in Oracle NoSQL Database. It must match the principal name in KDC and match the Kerberos user account name created in the database. The client will use the value of this option to create the credential which is used in the client-server authentication. If the short name of principal is specified in this field, you must also specify `oracle.kv.auth.kerberos.realm`.

  If KerberosCredentials is not used, this field has to be specified in the login file or security properties field of KVStoreConfig.

- `oracle.kv.auth.kerberos.services`

  Specifies the Kerberos principals for services associated with each helper host. Setting this property is required if, as recommended, each host uses a different principal that includes its own principal name. All principals should specify the same service and realm. If this property is not set, the client will use `oraclenosql` as the principal name for services on all helper hosts.

Each entry should specify the helper host name followed by the Kerberos service name, and optionally an instance name and realm name. The entries are separated by commas, ignoring spaces. If any entry does not specify a realm, each entry will use the default realm specified in Kerberos configuration file. If any entry specifies a realm name, then all entries must specify the same one. The syntax is:

```
host:service[:instance[@realm]][, host:service[:instance[@realm]]]*
```

For example:

```
host37:nosql/host37@EXAMPLE.COM,
host53:nosql/host53@EXAMPLE.COM
```

- `oracle.kv.auth.kerberos.keytab`

  The default location of the keytab file is specified by the Kerberos configuration file. If the keytab is not specified there, then the system looks for the file `user.home/krb5.keytab`.

- `oracle.kv.auth.kerberos.realm`

  Specifies the Kerberos realm for the user principal if using a short name to specify the client login principal.

- `oracle.kv.auth.kerberos.ccache`

  Specifies the path of the Kerberos ticket cache. This field is optional. The default ticket cache is "/tmp/krbcc_<uid>". If the credential cache is not found, the system will look for the file user.home/krb5cc_user.name. If you want to use your own ticket cache, set this field to the path of the ticket cache.

- `oracle.kv.auth.kerberos.mutualAuth`

  Specifies whether the client should use mutual authentication. If this value is set to true, the client will authenticate the server's identity in the login results.

  The default value is `false`, so mutual authentication is disabled.

Setting Security Properties in a security login file

To set the properties in a security file, specify the location of the login file by setting the `oracle.kv.security` Java system property. For example:

```
java -Doracle.kv.security=kerberoslogin.txt HelloWorld
```

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

```
oracle.kv.auth.username=krbuser@EXAMPLE.COM
oracle.kv.auth.external.mechanism=kerberos
oracle.kv.auth.kerberos.keytab=/kerberos/krb5.keytab
oracle.kv.auth.kerberos.services=
                node01:oraclenosql/node01.example.com@EXAMPLE.COM
oracle.kv.auth.kerberos.mutualAuth=false
```

You can specify the location of the Kerberos configuration file by specifying the `java.security.krb5.conf` Java system property. For example:

```
java -Djava.security.krb5.conf=/kerberos/krb5.conf \
-Doracle.kv.security=kerberoslogin.txt HelloWorld
```

You can also set the default realm using `java.security.krb5.realm`. To set the default KDC, use `java.security.krb5.kdc`.

> **Note:**
>
> Set the Java system properties for both the realm and the KDC or neither of them. These properties override the default realm and KDC values specified in the `krb5.conf` file.

Setting Security Properties through KVStoreConfig

You can also set security properties using `KVStoreConfig`. For example:

```
Properties securityProps = new Properties();
securityProps.setProperty("oracle.kv.auth.username",
                                          "krbuser@EXAMPLE.COM");
securityProps.setProperty("oracle.kv.auth.external.mechanism",
                                          "kerberos");
securityProps.setProperty("oracle.kv.auth.kerberos.keytab",
                                           "/kerberos/krb5.keytab");
securityProps.setProperty("oracle.kv.auth.kerberos.services",
                "node01:oraclenosql/node01.example.com@EXAMPLE.COM");
securityProps.setProperty("oracle.kv.auth.kerberos.ccache",
                                             "/kerberos/krbcc_501");
securityProps.setProperty("oracle.kv.auth.kerberos.mutualAuth",
                                             "false");

KVStoreConfig kvConfig = new KVStoreConfig("mystore", "node01:5000");
kvConfig.setSecurityProperties(securityProps);
```

Using Security Properties to Log In

To log in to Oracle NoSQL Database using security properties, you can use credential cache, a keytab file or the principal password.

> **Note:**
>
> When connecting through the Admin CLI, if credential cache or keytabs login attempts fail, Oracle NoSQL Database prompts for the principal's password.

Using Credential Cache

To login to Oracle NoSQL Database using credential cache:

1. Run the `kinit` Kerberos tool to save the credential in the credential cache.

   For example, to authenticate the client principal `krbuser@EXAMPLE.COM` to KDC:

   ```
   kinit krbuser@EXAMPLE.COM
   Password for krbuser@EXAMPLE.COM: ********
   ```

   The granted ticket-granting ticket (TGT) will be saved in the default credential cache for later authentication.

2. You can also generate a separate cache. To do this run the following command:

```
kinit krbuser@EXAMPLE.COM -c krbcc_krbuser
```

3. Perform the login by specifying `oracle.kv.auth.kerberos.ccache` in a security login file or through `KVStoreConfig`. In this case, a security login file is used:

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

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

```
oracle.kv.auth.kerberos.ccache=/kerberos/krbcc_krbuser
oracle.kv.auth.username = krbuser@EXAMPLE.COM
oracle.kv.auth.external.mechanism=kerberos
oracle.kv.auth.kerberos.services=
                 node01:oraclenosql/node01.example.com@EXAMPLE.COM
oracle.kv.auth.kerberos.mutualAuth=false
```

In this case, Oracle NoSQL Database reads the credential cache and logins to Kerberos without needing a password.

Using credential cache

To login to Oracle NoSQL Database using credential cache:

1. Run the `kinit` Kerberos tool to save the credential in the credential cache.

   For example, to authenticate the client principal krbuser@EXAMPLE.COM to KDC:

```
kinit krbuser@EXAMPLE.COM
Password for krbuser@EXAMPLE.COM: ********
```

The granted ticket-granting ticket (TGT) will be saved in the default credential cache for later authentication.

2. You can also generate a separate cache. To do this run the following command:

```
kinit krbuser@EXAMPLE.COM -c krbcc_krbuser
```

3. Perform the login by specifying `oracle.kv.auth.kerberos.ccache` in a security login file or through `KVStoreConfig`. In this case, a security login file is used:

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

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

```
oracle.kv.auth.kerberos.ccache=/kerberos/krbcc_krbuser
oracle.kv.auth.username = krbuser@EXAMPLE.COM
oracle.kv.auth.external.mechanism=kerberos
oracle.kv.auth.kerberos.services=
                 node01:oraclenosql/node01.example.com@EXAMPLE.COM
oracle.kv.auth.kerberos.mutualAuth=false
```

In this case, Oracle NoSQL Database reads the credential cache and logins to Kerberos without needing a password.

Using a keytab

To login to Oracle NoSQL Database using a keytab:

1. Run the `kinit` Kerberos tool to extract the keytab:

   ```
   kadmin.local:  ktadd -k /tmp/mykeytab krbuser@EXAMPLE.COM
   Entry for principal krbuser@EXAMPLE.COM added to
   keytab WRFILE:/tmp/mykeytab.
   ```

2. Copy the keytab file to any client machine that will use the krbuser@EXAMPLE.COM principal to login automatically to Oracle NoSQL Database.

3. Set the Kerberos security properties, including the keytab file location, on each client by specifying them in a security file or through the `KVStoreConfig` class.

   In this example, a security file (login) is used. To login, specify the keytab location by using `oracle.kv.auth.kerberos.keytab`. You must also specify the username using `oracle.kv.auth.username`. For example, the login file would then contain content like this:

   ```
   oracle.kv.auth.kerberos.keytab = /kerberos/mykeytab
   oracle.kv.auth.username = krbuser@EXAMPLE.COM
   oracle.kv.auth.external.mechanism=kerberos
   oracle.kv.auth.kerberos.services=
                   node01:oraclenosql/node01.example.com@EXAMPLE.COM
   oracle.kv.auth.kerberos.mutualAuth=false
   ```

   In this case, Oracle NoSQL Database reads the keytab and logins to Kerberos without needing a password.

   For more information on Kerberos security properties, see Kerberos Security Properties.

JAAS programming framework integration

Oracle NoSQL Database allows client applications to integrate with programs using the Java Authentication and Authorization Service (JAAS) programming framework.

Use the `oracle.kv.jaas.login.conf.entryName` security property to specify the JAAS login configuration.

> **Note:**
>
> If a JAAS login configuration file is set, you cannot specify keytab or credential cache in security properties.

A login configuration file would then contain content like this:

```
oraclenosql {
   com.sun.security.auth.module.Krb5LoginModule required
   useKeyTab=true
   keyTab=test.keytab
```

```
   storeKey=true
   principal=krbuser
   doNotPrompt=false;
};
```

where `oraclenosql` is the value for `oracle.kv.jaas.login.conf.entryName`. This configuration file can be used for Kerberos login.

In the following example, assume the client application has already obtained the Kerberos credentials for user `krbuser` before it tries to connect to Oracle NoSQL Database. You do not have to specify security properties in the login file. You can specify the credentials using the `Subject.doAs` method:

```
final LoginContext lc =
    new LoginContext("oraclenosql", new TextCallbackHandler());

// Attempt authentication
lc.login();

// Get the authenticated Subject
final Subject subj = lc.getSubject();

// Specify configuration
final KVStoreConfig kvConfig =
    new KVStoreConfig("mystore", "nosql1:5000");

// Set security properties SSL needed
final Properties securityProps = new Properties();
securityProps.setProperty(KVSecurityConstants.TRANSPORT_PROPERTY,
                          KVSecurityConstants.SSL_TRANSPORT_NAME);
securityProps.setProperty(
                 KVSecurityConstants.SSL_TRUSTSTORE_FILE_PROPERTY,
                 trustStore);
kvConfig.setSecurityProperties(securityProps);

// Set Kerberos properties
final Properties krbProperties = new Properties();

// Set service principal associated with helper host
krbProperties.setProperty(KVSecurityConstants.AUTH_KRB_SERVICES_PROPERTY,
                          hostName + ":" + servicePrincipal);

// Set default realm name, because the short name
// for user principal is used.
krbProperties.setProperty(KVSecurityConstants.AUTH_KRB_REALM_PROPERTY,
                          "EXAMPLE.COM");

// Specify Kerberos principal
final KerberosCredentials krbCreds =
    new KerberosCredentials("krbuser", krbProperties);

// Get store using credentials in subject
KVStore kvstore = Subject.doAs(
    subj, new PrivilegedExceptionAction<KVStore>() {
        @Override
        public KVStore run() throws Exception {
            return KVStoreFactory.getStore(kvConfig, krbCreds, null);
        }
    });
```

In this case, a KerberosCredentials instance is used to set the security properties needed to retrieve the credentials of the specified user principal from KDC.

Performing a Secure Oracle NoSQL Database Installation with Kerberos

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

At a high-level, to configure a Oracle NoSQL Database installation to use Kerberos, you first need to register Oracle NoSQL Database as a service principal in KDC and extract corresponding keytab files on each database server node. Then, to allow client login, a user principal must be added in KDC and a mapped user account with the same name of principal needs to be created in the database. Finally, login can be performed through the CLI or the kvclient driver.

Adding Kerberos to a New Installation

To install Oracle NoSQL Database with Kerberos authentication:

> **Note:**
>
> The following example assumes you have configured an admin/admin principal on the KDC and that you distributed its keytab (kadm5.keytab) to the Oracle NoSQL Database Storage Nodes. For more information, see Configuring the Kerberos Administrative Utility.

1. Run the `makebootconfig` utility with the `-store-security configure` and `-external-auth kerberos` flags to set up the basic store configuration with Kerberos security:

   ```
   java -jar KVHOME/lib/kvstore.jar makebootconfig \
   -root KVROOT -port 5000 \
   -host node01 -harange 5010,5020 \
   -capacity 3 \
   -store-security configure \
   -external-auth kerberos \
   -instance-name node01.example.com \
   -kadmin-keytab /kerberos/kadm5.keytab \
   -admin-principal admin/admin
   ```

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, Oracle Wallet is used. Oracle Wallet and Kerberos support are only available in the Oracle NoSQL Database EE version.

   ```
   Enter a password for the Java KeyStore: ***********
   Re-enter the KeyStore password for verification: ***********
   ```

4. In this case, `-kadmin-keytab` points to the admin/admin keytab file you distributed earlier. Once authenticated, the configuration tool will automatically generate some security related files:

```
Login Kerberos admin via
keytab /kerberos/kadm5.keytab
Adding principal oraclenosql/node01.example.com@EXAMPLE.COM
Authenticating as principal admin/admin with
keytab /kerberos/kadm5.keytab
Extracting keytab KVROOT/security/store.keytab
Created files:
security/security.xml
security/store.keys
security/store.trust
security/store.wallet/cwallet.sso
security/store.keytab
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 Secure Copy Protocol (SCP), to do the copying if the files for the different nodes are not visible on the current node.

6. Run makebootconfig on the other two nodes:

   - Add Kerberos and create their individual service principal and keytab:

```
java -jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 6000 \
-host node02 -harange 6010,6020 \
-capacity 3 \
-store-security configure \
-external-auth kerberos \
-instance-name node02.example.com \
-kadmin-keytab /kerberos/kadm5.keytab \
-admin-principal admin/admin

java -jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 7000 \
-host node03 -harange 7010,7020 \
-capacity 3 \
-store-security configure \
-external-auth kerberos \
-instance-name node03.example.com \
-kadmin-keytab /kerberos/kadm5.keytab \
-admin-principal admin/admin
```

> **Note:**
>
> The service principal name of node2 and node3 are using the same service name "oraclenosql", but different instance names. Their keytab files are different, which contains the key for principal "oraclenosql/node2.example.com" and "oraclenosql/node3.example.com" respectively.

- To enable Kerberos authentication if the store is using the same service principal on every node:

```
java -jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 6000 \
-host node02 -harange 6010,6020 \
-capacity 3 \
-store-security enable

java -jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 6000 \
-host node03 -harange 6010,6020 \
-capacity 3 \
-store-security enable
```

> **Note:**
>
> The service principal created in step one is "oraclenosql/node01.example.com". The instance name can be replaced with any more general one like "nosql". In above example, node02 and node03 are all using the same service principal and keytab file without creating new one individually.

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

```
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 the following steps 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.

8. 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
```

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

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

10. Register the user principal on the KDC. To do this, use `kadmin` or `kadmin.local`:

```
kadmin.local: addprinc  krbuser@EXAMPLE.COM
Enter password for principal: "krbuser@EXAMPLE.COM": ***********
Re-enter password for principal: "krbuser@EXAMPLE.COM": **********
```

11. After user principal is registered on KDC, create the user in Oracle NoSQL Database. The username needs to match the full principal name in the KDC (includes realm name). In this case, user `krbuser` is defined:

```
kv-> execute 'CREATE USER "krbuser@EXAMPLE.COM" IDENTIFIED EXTERNALLY'
```

    For more information on user creation and administration, see User Management.

12. At this point, it is possible to connect to the store as the krbuser. To login, you can use credential cache, a keytab file or enter the principal password.

    In this example, a keytab file is used. To do this, first extract the keytab of principal krbuser@EXAMPLE.COM on the KDC host by using `kadmin.local`.

```
kadmin.local:  ktadd -k /tmp/mykeytab  krbuser@EXAMPLE.COM
Entry for principal krbuser@EXAMPLE.COM added to
keytab WRFILE:/tmp/mykeytab.
```

13. Copy the keytab file to client machines that will use the krbuser@EXAMPLE.COM principal to login automatically to Oracle NoSQL Database.

14. Set the Kerberos security properties, including the keytab file location, on each client by specifying them in a security file or through the `KVStoreConfig` class.

    In this example, a security file (mylogin.txt) is used. To login, specify the file location by using the `oracle.kv.security property`. For example:

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

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

```
oracle.kv.auth.kerberos.keytab = kerberos/mykeytab
oracle.kv.auth.username = krbuser@EXAMPLE.COM
oracle.kv.auth.external.mechanism=kerberos
oracle.kv.auth.kerberos.services=
                node01:oraclenosql/node01.example.com@EXAMPLE.COM
oracle.kv.auth.kerberos.mutualAuth=false
```

    In this case, Oracle NoSQL Database reads the keytab and logins to Kerberos without needing a password.

    For more information on Kerberos security properties, see Kerberos Security Properties.

Adding Kerberos to an Existing Secure Installation

To add Kerberos to an existing Oracle NoSQL Database secure installation:

> **Note:**
>
> The following example assumes you have configured an admin/admin principal on the KDC and that you distributed its keytab (kadm5.keytab) to the Oracle NoSQL Database Storage Nodes. For more information, see Configuring the Kerberos Administrative Utility.

> **Note:**
>
> If your Kerberos installation/keytab will be configured to use a strong encryption type — for example, AES with 256-bit keys — the JCE Unlimited Strength Jurisdiction Policy Files must be obtained and installed in the JDK/JRE. Be aware that these files might already exist in your installation. If so, they must be updated.

1. Shut down the KVStore instance:

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

2. Use the `config add-kerberos` command to add Kerberos authentication:

```
java -jar KVHOME/lib/kvstore.jar securityconfig
config add-kerberos -root KVROOT \
-secdir security \
-admin-principal admin/admin

Adding principal oraclenosql@EXAMPLE.COM
Password for admin/admin: *******
Created files:
   security/store.keytab
Updated Kerberos configuration
```

> **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.

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

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

4. 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.

5. Register the user principal on the KDC. To do this, use `kadmin` or `kadmin.local`:

```
kadmin.local: addprinc  krbuser@EXAMPLE.COM
Enter password for principal: "krbuser@EXAMPLE.COM": ***********
Re-enter password for principal: "krbuser@EXAMPLE.COM": **********
```

6. After user principal is registered on KDC, create the user in Oracle NoSQL Database. The username needs to match the full principal name in the KDC (includes realm name). In this case, user `krbuser` is defined:

```
kv-> execute 'CREATE USER "krbuser@EXAMPLE.COM" IDENTIFIED EXTERNALLY'
```

   For more information on user creation and administration, see User Management.

7. At this point, it is possible to connect to the store as the krbuser. To login, you can use credential cache, a keytab file or enter the principal password.

   In this example, a keytab file is used. To do this, first extract the keytab of principal krbuser@EXAMPLE.COM on the KDC host by using `kadmin.local`.

```
kadmin.local:  ktadd -k /tmp/mykeytab  krbuser@EXAMPLE.COM
Entry for principal krbuser@EXAMPLE.COM added to
keytab WRFILE:/tmp/mykeytab.
```

8. Copy the keytab file to any client machine that will use the krbuser@EXAMPLE.COM principal to login automatically to Oracle NoSQL Database.

9. Set the Kerberos security properties, including the keytab file location, on each client by specifying them in a security file or through the `KVStoreConfig` class.

   In this example, a security file (mylogin.txt) is used. To login, specify the file location by using the `oracle.kv.security property`. For example:

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

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

```
oracle.kv.auth.kerberos.keytab = kerberos/mykeytab
oracle.kv.auth.username = krbuser@EXAMPLE.COM
oracle.kv.auth.external.mechanism=kerberos
oracle.kv.auth.kerberos.services=
                  node01:oraclenosql/node01.example.com@EXAMPLE.COM
oracle.kv.auth.kerberos.mutualAuth=false
```

In this case, Oracle NoSQL Database reads the keytab and logins to Kerberos without needing a password.

For more information on Kerberos security properties, see Kerberos Security Properties.

Using Oracle NoSQL Database with Kerberos and Microsoft Active Directory (AD)

To use Oracle NoSQL Database with Kerberos and Microsoft Active Directory:

1. Update Kerberos Configuration krb5.conf with AD.

   The *Microsoft Guide* (see here) details how to update the Kerberos configuration file on a Unix host in step 3: *Edit the file (/etc/krb5.conf) to refer to the Windows 2000 domain controller as the Kerberos KDC*. After changing the Kerberos configuration file, run kinit using a user account in AD to verify that the configuration is correct.

   For example, suppose you have user account `krbuser08` on domain `TEST08` of AD, and the KDC realm name is `TEST08.LOCAL`:

   ```
   $ kinit krbuser08@TEST08.LOCAL
   Password for krbuser08@TEST08.LOCAL
   ```

   After you provide the password, the command should return without error. An error indicates there are probably configuration issues. If the `kinit` command ran successfully, then run `klist` to check that the ticket cache contains the TGT of `krbuser08`.

   ```
   $ klist
   Ticket cache: FILE:/tmp/krb5cc_500
   Default principal: krbuser08@TEST08.LOCAL

   Valid starting     Expires            Service principal
   08/12/16 11:45:03  08/12/16 21:45:11 krbtgt/TEST08.LOCAL@TEST08.LOCAL
          renew until 08/19/16 11:45:03
   ```

   The `klist` shows the tickets in your ticket cache. Perform this step to check if the ticket-granting ticket has been properly obtained using the principal `krbuser08` described by "Default Principal." The "Service Principal" describes each ticket, the ticket-granting ticket has the primary `krbtgt`, and the instance name is the KDC realm name. Also check if the lifetime indicated by "Valid Starting" and "Expires" is correct.

2. Create service instance account and generate keytab on AD.

   The *Microsoft Guide* (see https://technet.microsoft.com/en-us/library/bb742433.aspx#EEAA) details how to support a service running on a Unix system when using Active Directory. Follow the steps in this document to generate the service principal and keytab file for Oracle NoSQL Database. Note that you do not need to perform step 3 in the Microsoft Guide to merge keytab files if you plan to use same keytab file on every host.

   For example, you can set the instance name to `nosql` and use this keytab on every node.

   - Use the Active Directory Management tool to create a user account named `oraclenosql`.

In the user creation interface, you can choose which Kerberos encryption type this account can support. The user account may use Data Encryption Standard (DES) encryption as default. To enable other encryption types for this account, you need to manually configure in the "Properties" interface, or by using `ktpass` utility. Note that you need to disable the "User must change password at next logon" setting.

- Use ktpass tool on Windows Server to set up an identity mapping.

```
c:\ktpass -princ oraclenosql/nosql@TEST08.LOCAL
-mapuser oraclenosql -pass "*"-cypto DES-CBC-MD5 -ptype
KRB5_NT_PRINCIPAL -out c:\store.keytab
```

You may need to add `allow_weak_crypto = true` to the `krb5.conf` file on the Unix host, as well as `default_tkt_enctypes` and `default_tgs_enctypes`, if you use the DES decryption type. The default name of the keytab for Oracle NoSQL Database is `store.keytab` and the default service name of the service principal is `oraclenosql`.

- Copy the keytab file to your Unix hosts used by Oracle NoSQL Database.

  Typically, you can use Secure Copy Protocol (scp) or PuTTY Secure Copy (PSCP) to transfer this file securely, or upload this file to an FTP server shared by Windows Server and Unix hosts. After creating the service principal and keytab, run kinit tests on your Unix hosts (described next) to confirm that they are configured properly.

3. Test if the user account can acquire service tickets for the service principal, and if the service keytab is generated correctly by running kinit:

   - Test if the user account can acquire service tickets for service principal `oraclenosql`.

```
$ kinit -S oraclenosql/nosql@TEST08.LOCAL krbuser08@TEST08.LOCAL
Password for krbuser08@TEST08.LOCAL:
$ klist
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: krbuser08@TEST08.LOCAL

Valid starting     Expires            Service principal
08/12/16 11:50:55  08/12/16 21:51:00  oraclenosql/nosql@TEST08.LOCAL
       renew until 08/19/16 11:50:55
```

   If the ticket cache does not contains a service ticket for `oraclenosql/nosql`, or if any errors are reported in the first command, then check if the account was created properly.

   - Test if the service keytab was generated correctly by running kinit `oraclenosql`.

```
$ kinit -k -t store.keytab oraclenosql/nosql@TEST08.LOCAL
$ klist
Ticket cache: FILE:/tmp/krb5cc_500
Default principal: oraclenosql/nosql@TEST08.LOCAL

Valid starting     Expires            Service principal
08/12/16 11:51:44  08/12/16 21:51:45 krbtgt/TEST08.LOCAL@TEST08.LOCAL
       renew until 08/19/16 11:51:44
```

   As with the previous tests, any errors need to be fixed before attempting to configure Oracle NoSQL Database. Some versions of the kinit utility may need to explicitly specify `default_tkt_enctypes` and `default_tgs_enctypes` with the

encryption type you configured for the service account `oraclenosql` in Active Directory, otherwise kinit cannot successfully obtain tickets from AD.

4. Begin to configure Oracle NoSQL Database.

   Oracle NoSQL Database utilizes the Unix `kadmin` tool to help users create service principal and generate keytab file. However, AD does not have remote admin utility support, so it is necessary to skip this step in AD Kerberos environment.

   For Oracle NoSQL Database releases prior to 4.2, you must specify `none` as the value for both the `-kadmin-path` and `-admin-principal` `makebootconfig` command line options.

```
java -jar $KVHOME/lib/kvstore.jar makebootconfig -root kvroot
-port 5000 \
-host node01.example.com -harange 5010,5020 \
-store-security configure -kspwd password \
-external-auth kerberos \
-kadmin-path none \
-admin-principal none \
-instance-name nosql
Adding principal oraclenosql/nosql
IO error encountered: Cannot run program "none": error=13,
Permission denied
Created files
   KVROOT/security/client.security
   KVROOT/security/client.trust
   KVROOT/security/security.xml
   KVROOT/security/store.wallet/cwallet.sso
   KVROOT/security/store.keys
   KVROOT/security/store.trust
```

   The IO error can be ignored in this example, because we did not specify a correct `kadmin` path.

   For Oracle NoSQL Database 4.2 and later releases, you only need to specify `none` as the value for the `-kadmin-path` flag:

```
java -jar $KVHOME/lib/kvstore.jar makebootconfig -root kvroot
-port 5000 \
-host node01.example.com -harange 5010,5020 \
-store-security configure -kspwd password \
-external-auth kerberos \
-kadmin-path none \
-instance-name nosql
```

   The kadmin path was specified as `NONE`, so this example is not creating a keytab for the database server. The keytab must be generated and copied to the security configuration directory manually.

```
Created files
   KVROOT/security/client.security
   KVROOT/security/client.trust
   KVROOT/security/security.xml
   KVROOT/security/store.wallet/cwallet.sso
   KVROOT/security/store.keys
   KVROOT/security/store.trust
```

   After the security directory is created, it is worth checking that the Kerberos parameters are configured as expected.

   Check security.xml in `kvroot/security` and look for the following parameters:

- krbInstanceName
- krbRealmName

For Oracle NoSQL Database 4.2 and later releases, you can use the `securityconfig` tool to view the parameters:

```
java -jar KVHOME/lib/kvstore.jar securityconfig
config show -secdir kvroot/security
...
krbInstanceName=nosql
krbRealmName=TEST08.LOCAL
```

5. Manage service principals in a multi-node environment.

   - In a multi-node environment, if you want to use a single service principal `oraclenosql/nosql` for all nodes, you can simply copy the contents of the first security directory to the other nodes. 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/
     ```

     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.

     Run makebootconfig on the other two nodes to enable Kerberos authentication.

     ```
     java -jar KVHOME/lib/kvstore.jar makebootconfig \
     -root KVROOT -port 5000 \
     -host node02 -harange 5010,5020 \
     -store-security enable

     java -jar KVHOME/lib/kvstore.jar makebootconfig \
     -root KVROOT -port 5000 \
     -host node03 -harange 5010,5020 \
     -store-security enable
     ```

     > **Note:**
     >
     > The service principal for node02 and node03 will be configured as `oraclenosql/nosql@TEST08.LOCAL`. Also they will use the same keytab file generated in step two.

   - To set up individual service principals for each node, run step two to create a service account on AD and generate a new keytab for each node. For example, each node uses host name as instance name of service principal and their corresponding keytab files.

     ```
     oracelnosql/node01@TEST08.LOCAL
     oracelnosql/node02@TEST08.LOCAL
     oracelnosql/node03@TEST08.LOCAL
     ```

     Copy the security directory created on node01 to other nodes. For example, the following commands assume that the different nodes are accessible using ssh from the current node (host01):

```
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 copy the files for the different nodes if they are not visible on the current node.

Replace keytab files of node2 and node3 generated in step two with the one in their security configuration directory. For example:

```
cp store.keytab node02/KVROOT/security
cp store.keytab node03/KVROOT/security
```

> **Note:**
>
> The name of all of the keytab files generated in step two is `store.keytab` by default. Make sure that you have given each node the proper keytab file. Use the `klist` tool to check keytab file on each node to make sure they contain the correct key of service principal for the node.

Run the `securityconfig` tool on node02 and node03 to modify instance name of security configuration:

```
security -> config update -secdir KVROOT/security \
-param krbInstanceName=node02

security -> config update -secdir KVROOT/security \
-param krbInstanceName=node03
```

Run makebootconfig on the other two nodes to enable Kerberos authentication.

```
java -jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 5000 \
-host node02 -harange 5010,5020 \
-store-security enable

java -jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 5000 \
-host node03 -harange 5010,5020 \
-store-security enable
```

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

```
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. 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, perform the following steps 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.

7. Start `runadmin` in security mode on the KVStore server host (node01). To do this:

```
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, and then complete store deployment. For more information, see the *Oracle NoSQL Database Admininstrator's Guide*:

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

9. Create a user account on Microsoft Active Directory. In this example, `krbuser` is created on Active Directory.

10. Create mapping user in Oracle NoSQL Database. The username needs to match the full principal name in the KDC (includes realm name). In this case, user krbuser is defined:

```
kv-> execute 'CREATE USER "krbuser@TEST08.LOCAL"
IDENTIFIED EXTERNALLY'
```

    For more information on user creation and administration, see User Management.

11. At this point, it is possible to connect to the store as the krbuser. To login, you can use credential cache, a keytab file or enter the principal password.

12. Set the Kerberos security properties, including the keytab file location, on each client by specifying them in a security file or through the `KVStoreConfig` class.

    In this example, a security file (mylogin.txt) is used. To login, specify the file location by using the `oracle.kv.security` property. For example:

```
java -Xmx256m -Xms256m \
-Doracle.kv.security=mylogin.txt \
-jar KVHOME/lib/kvstore.jar runadmin -port 5000 -host localhost
krbuser@TEST08.LOCAL's kerberos password:
Logged in admin as krbuser@TEST08.LOCAL
kv->
```

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

```
oracle.kv.auth.username = krbuser@TEST08.LOCAL
oracle.kv.auth.external.mechanism=kerberos
oracle.kv.auth.kerberos.services=node01:oraclenosql/nosql@TEST08.LOCAL
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)
```

In this example, the store nodes are using the single service principal `oraclenosql/nosql`. Without specifying keytab or credential cache, Admin CLI prompts for principal password.

For more information on Kerberos security properties, see Kerberos Security Properties.

5
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 the 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.

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

```
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:

```
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.

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 though this interface are never password protected. For more information on the `securityconfig` tool, see Configuring Security with Securityconfig.

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.

6
Security.xml Parameters

This chapter describes the parameters that can be set in 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.

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.

- krbServiceName

  Specifies the service name of the Oracle NoSQL Database Kerberos service principal.

- krbInstanceName

  Specifies the service principal instance name.

- krbServiceKeytab

  Specifies the keytab file name in the security directory that contains the KVStore server service principal and encrypted copy of principal's key.

- krbConf

  Specifies the location of the Kerberos configuration file that contains the default realm and KDC information. If not specified, the default value is `/etc/krb5.conf`.

- krbRealmName

  Specifies the realm name of service principal. If not specified, this value is acquired from the Kerberos configuration file.

- userExternalAuth

  Specifies and enables the external mechanism used for authentication. Kerberos is supported. Set the value to `KERBEROS` to enable Kerberos authentication. To remove Kerberos authentication from a running store, set the value to `NONE`.

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.

7 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.

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.

8
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 .

All user passwords should follow the password security policies. For more information see Password Complexity Policies.

User Creation

To create a user, use the following command:

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

where:

- `user_name`

  The name of the user. If specifying a Kerberos user, the name must match the fully qualified principal name created in KDC.

  Kerberos users need to have different names from existing users, as you cannot change the authentication type of an existing user.

- `IDENTIFIED Clause`

  Indicate how Oracle NoSQL Database authenticates the users (currently by password). Kerberos users can be identified by using the IDENTIFIED EXTERNALLY clause instead.

- `BY password`

  Create an Oracle NoSQL Database local user and indicates that the user must specify password to log on to the Oracle NoSQL Database.

> **Note:**
>
> The password must be quoted by double-quotation marks, for example, "*password*".

- `IDENTIFIED EXTERNALLY`

  Indicate that Oracle NoSQL Database will use an external mechanism to authenticate the user.

  Kerberos is supported as an external authentication service.

- `ACCOUNT Clause`

  Specify ACCOUNT LOCK to lock the user's account and disable access. Specify ACCOUNT UNLOCK to enable the user.

- `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 "*password*":

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

  Note that specifying 0 time unit for PASSWORD LIFETIME will make the password as "never expired".

> **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.

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. If specifying a Kerberos user, you can only alter the ACCOUNT clause options.

- `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.

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

  Note that specifying 0 time unit for PASSWORD LIFETIME will make the password as "never expired".

- `ACCOUNT Clause`

  Specify ACCOUNT LOCK to lock the user's account and disable access. Specify ACCOUNT UNLOCK to enable the user.

If you are updating the password of an existing user, the new password should comply with the password security policies. For more information see Password Complexity Policies.

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'
id=u2 name=kate enabled=true auth-type=LOCAL retain-passwd=inactive
granted-role=[public]
```

If a Kerberos user is specified, then the auth-type field shows EXTERNAL:

```
kv-> execute 'SHOW USER krbuser@EXAMPLE.COM'
user: id=u4 name=krbuser@EXAMPLE.COM enabled=true auth-type=EXTERNAL
retain-passwd=inactive granted-roles=[readwrite, public, sysadmin]
```

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. All user passwords should follow the password security policies. For more information see Password Complexity Policies.

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 \"password\" 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

9
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
- Roles
- Managing Roles, Privileges and Users

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.

- `DELETE_TABLE`

  Can delete data from a specific table.

- `INSERT_TABLE`

  Can insert and update data to a specific table.

- `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.

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 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.

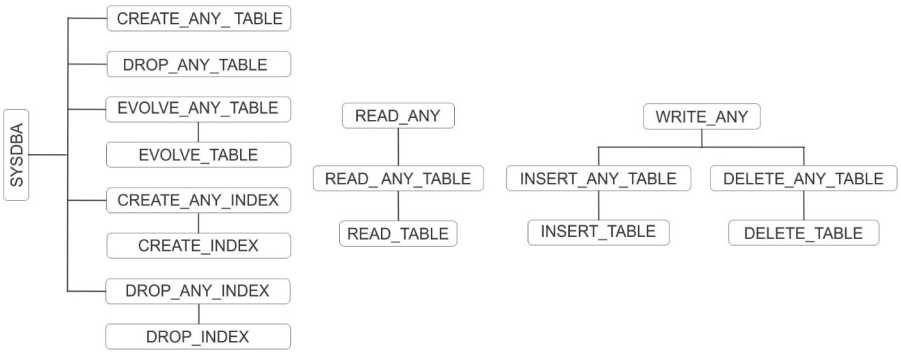
For example, for table `myTable` and its child `myTable.child1`, a non-owner user can only insert data to `myTable.child1` when she has insert privilege (or better) on `myTable.child1`, and read and/or insert privilege on `myTable`.

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:



> **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:

```
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:

```
kv-> execute 'CREATE USER Ken IDENTIFIED BY \"password\" '

kv-> execute 'CREATE USER Kate IDENTIFIED BY \"password\" '
```

Now, for example, create 2 roles – 'manager' with the 'write_any' privilege and 'employee' with 'read_any' privilege:

```
kv->execute 'CREATE ROLE manager'
kv->execute 'GRANT WRITE_ANY TO manager'
kv->execute 'CREATE ROLE employee'
kv->execute '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:

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

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

```
kv->execute 'SHOW USER Kate'
id=u2 name=Kate enabled=true type=LOCAL retain-passwd=inactive
granted-role=[public, manager]
```

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:

```
kv->execute 'DROP ROLE manager'
kv->execute 'SHOW USER Kate'
id=u2 name=Kate enabled=true type=LOCAL retain-passwd=inactive
granted-role=[public]
```

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 emptablereader'
kv->execute 'GRANT READ_TABLE ON emp TO emptablereader'
kv->execute 'SHOW ROLE emptablereader'
name=emptablereader 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.

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
```

10
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.

11
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.

12
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.

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. You can provide the new passwords through the keytool interactive prompt or using arguments. For example, to set the new key and store passwords for store.keys as well as the new store password for store.trust using the keytool prompt:

   > **Note:**
   >
   > The 3 new passwords must be equal, otherwise the store cannot be successfully restarted.

   ```
   keytool -keypasswd -keystore store.keys -alias shared
   Enter keystore password:
   New key password for <shared>:
   Re-enter new key password for <shared>:

   keytool -storepasswd -keystore store.keys
   Enter keystore password:
   New keystore password:
   Re-enter new keystore password:
   ```

```
keytool -storepasswd -keystore store.trust
Enter keystore password:
New keystore password:
Re-enter new keystore password:
```

You could also run the keytool command and set the new passwords using arguments instead. For example:

```
keytool -keypasswd -keystore store.keys \
-alias shared -keypass <old_pwd> -new <new_pwd> -storepass <old_pwd>

keytool -storepasswd -keystore store.keys \
-storepass <old_pwd> -new <new_pwd>

keytool -storepasswd -keystore store.trust \
-storepass <old_pwd> -new <new_pwd>
```

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, store.trust 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 Updating Kerberos Passwords

The password of Kerberos principal should be periodically changed. To do this, you can either manually specify it by using `kadmin.local` or automatically randomize principal keys by using the `config renew-keytab` command of the `securityconfig` tool.

The syntax for this command is:

```
config renew-keytab -root <secroot> [-secdir <security dir>]
[-keysalt <enc:salt[,enc:salt,..]>]
[-kadmin-path <kadmin utility path>]
[-instance-name <database instance name>]
[-admin-principal <kerberos admin principal name>]
[-kadmin-keytab <keytab file> ]
[-kadmin-ccache <credential cache file>]
```

where:

- `-keysalt`

  Sets the list of encryption types and salt types to be used for any new keys created. The default value is `des3-cbc-sha1:normal,aes128-cts-hmac-sha1-96:normal,arcfour-hmac:normal`.

- `-kadmin-path`

  Indicates the absolute path of Kerberos kadmin utility. The default value is `/usr/kerberos/sbin/kadmin`.

- `-instance-name`

  Specifies the service principal name. The default value is the fully qualified domain name (FQDN) of the Storage Node where Oracle NoSQL Database is running.

- `-admin-principal`

  Specifies the principal used to login to the Kerberos admin interface. This is required while using kadmin keytab or password to connect to the admin interface.

- `-kadmin-keytab`

  Specifies the location of a Kerberos keytab file that stores Kerberos admin user principals and encrypted keys. The security configuration tool will use the specified keytab file to login to the Kerberos admin interface.

  You need to specify the `-admin-principal` flag when using keytab to login to the Kerberos admin, otherwise the correct admin principal will not be recognized. This flag cannot be specified in conjunction with the `-kadmin-ccache` flag.

- `-kadmin-ccache`

  Specifies the complete path name to the Kerberos credentials cache file that should contain a service ticket for the kadmin/ADMINHOST. `ADMINHOST` is the fully-qualified hostname of the admin server or kadmin/admin service.

  If not specified, the user is prompted to enter the password for principal while logging to the Kerberos admin interface. This flag cannot be specified in conjunction with the `-kadmin-keytab` flag.

To manually update the Kerberos principal password instead, you should follow these steps:

1. Use kadmin.local utility to change the service principal password:

   ```
   kadmin.local: cpw nosql/myhost
   Enter password for principal nosql/myhost@EXAMPLE.COM
   Re-enter password for principal nosql/myhost@EXAMPLE.COM
   ```

2. Regenerate the keytab file for Oracle NoSQL Database service principal.

   ```
   kadmin.local: ktadd –norandkey –k new.keytab
   ```

3. Copy the new keytab file for Oracle NoSQL Database service principal to each Storage Node. For example:

   ```
   scp new.keytab kvuser@mystore:KVROOT/security/store.keytab
   ...
   ```

4. Validate the keytab file by comparing the key version number (kvno):

   ```
   kadmin.local:getprinc nosql/myhost@EXAMPLE.COM
   Principal: nosql/myhost@EXAMPLE.COM
   Expiration date: [never]
   Last password change: Thu Jun 04 03:16:38 UTC 2015
   ```

```
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 0 days 00:00:00
Last modified: Thu Jun 04 03:16:38 UTC 2015
(root/admin@ORACLE.EXAMPLE.COM)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 4
Key: vno 12, aes256-cts-hmac-sha1-96
Key: vno 12, aes128-cts-hmac-sha1-96
Key: vno 12, des3-cbc-sha1
Key: vno 12, arcfour-hmac
MKey: vno 1
Attributes:
Policy: [none]
Kadmin.local: quit
# klist –k new.keytab
KVNO Principal
---- ------------------------
12 nosql/myhost@EXAMPLE.COM
12 nosql/myhost@EXAMPLE.COM
12 nosql/myhost@EXAMPLE.COM
12 nosql/myhost@EXAMPLE.COM
```

Client side user principals require similar password rotation. Keytab or credential cache used to login to the database should be renewed. If `kinit` tool is used to create a credential cache, you should run `kdestroy` to clear cached tickets and re-run kinit to generate a new credential cache.

For example:

```
# kdestroy –c /tmp/krb5ccache
# kinit –c /tmp/krb5ccache
```

Guidelines for Updating the SSL key/certificate

Follow these steps to update the SSL key/certificate. The store can remain online during the update process:

1. On the configuration host, run securityconfig to create a new configuration in a directory in parallel to the standard configuration directory. The new configuration needs to specify the same keystore password as your standard configuration. If no password is specified with the `-kspwd` option, then the utility will prompt you to set a password.
2. On the configuration host, merge the truststore entries by using the `config merge-trust` command:

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

> **Note:**
>
> In a multiple Storage Node deployment, you need to copy the new configuration to each host's new configuration directory and run merge-trust on each host like in the example above.

3. Copy the updated client.trust file to the security directory on each host so that clients can use it to access the store.
4. Check that all Replication Nodes are online and then restart each Storage Node one by one using the following commands:

```
java -jar <KVHOME>/lib/kvstore.jar stop -root KVROOT

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

5. Copy the updated store.keys file to the security directory on each host. Then, check that all Replication Nodes are online and restart each Storage Node one by one using the following commands:

```
java -jar <KVHOME>/lib/kvstore.jar stop -root KVROOT

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

6. For all Storage Nodes, remove the obsolete certificate "mykey" in store.trust. Also, rename the new certificate "mykey_2" to "mykey" using the following command:

```
keytool -delete -keystore KVROOT/security/store.trust \
-alias mykey

keytool -changealias -keystore \
KVROOT/security/store.trust -alias mykey_2 -destalias mykey
```

7. Verify that the new certificate is the only one used using the following command:

```
keytool -list -keystore KVROOT/security/store.trust
```

Guidelines for Configuring External Certificates for a new Installation

Follow these steps to configure a new store to use external certificates:

> **Note:**
>
> This procedure assumes you already have a Java keystore and truststore setup. For more information see Java KeyStore Preparation.

1. Collect the distinguished name from the verbose information of the external certificate. In this example, it is the value of the owner field.

```
keytool -list -v -keystore store.keys alias shared
Certificate chain length: 3
Certificate[1]:
Owner: CN=myhost, OU=TeamA, O=MyCompany, L=Unknown, ST=California,
C=US
```

```
Issuer: CN=intermediate CA, OU=CA, O=MyCompany, ST=California,
C=US
```

2. Prepare `dnmatch` expression using a distinguished name. Oracle NoSQL Database verifies identities of server and client while establishing SSL connection between the server components. The verification is performed by checking if principal names on each side match the specified `dnmatch` expressions, which uses regular expressions as specified by `java.util.regex.Pattern`. The principal names represent the identities, which are specified by the subject name attribute of the certificate, represented as a distinguished name in RFC 1779 format, using the exact order, capitalization, and spaces of the attribute value. RFC 1779 defines well-known attributes for distinguished names, including CN, L, ST O, OU, C and STREET. If the distinguished name of the external certificate contains non-standard attributes, for example, EMAILADDRESS, then the expression used for `dnmatch` must replace these attribute names with an OID that is valid in RFC 1779 form, or use special constructs of regular expression to skip checking these attributes. The format for a `dnmatch` expression is:

   dnmatch*(regular expression)*

   In above example, the `dnmatch` expression is:

   ```
   dnmatch(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
   ST=California, C=US)
   ```

   If you are using a wild card to match a certificate with a non-standard distinguished name attribute, the `dnmatch` expression needs to match the attribute name in its OID format properly. For example, if the distinguished name is:

   ```
   EMAILADDRESS=person@example.com, CN=myhost, OU=TeamA, O=MyCompany,
   L=Unknown, ST=California, C=US
   ```

   Then wild card should represent the entire EMAILADDRESS attribute name:

   ```
   dnmatch(.*=person@example.com, CN=myhost, OU=TeamA, O=MyCompany,
   L=Unknown, ST=California, C=US)
   ```

3. Run makebootconfig to setup the secure store. Also specify the keystore password and dnmatch expressions in the security parameters. The keystore password "*password*" must use the same password as the Java Keystore of the external certificates. See:

   ```
   {{{
   java -Xmx256m -Xms256m -jar <KVHOME>/lib/kvstore.jar makebootconfig \
   -root KVROOT -host node01 -port 5000 -harange 5010,5020 -admin 5001 \
   -store-security configure \
   -pwdmgr wallet -kspwd password \
   -security-param client:serverIdentityAllowed="dnmatch
   (CN=myhost, OU=TeamA, O=MyCompany, L=Unknown, ST=California, C=US)" \
   -security-param internal:serverIdentityAllowed="dnmatch
   (CN=myhost, OU=TeamA, O=MyCompany, L=Unknown, ST=California, C=US)" \
   -security-param internal:clientIdentityAllowed="dnmatch
   (CN=myhost, OU=TeamA, O=MyCompany, L=Unknown, ST=California, C=US)" \
   -security-param ha:serverIdentityAllowed="dnmatch
   (CN=myhost, OU=TeamA, O=MyCompany, L=Unknown, ST=California, C=US)" \
   -security-param ha:clientIdentityAllowed="dnmatch
   (CN=myhost, OU=TeamA, O=MyCompany, L=Unknown, ST=California, C=US)"
   }}}
   ```

4. The makebootconfig command automatically generates keystore, server, and client truststore files using self-signed certificates. To use external certificates

instead, you need to replace the keystore and truststore files with your own on each server that will host a Storage Node. For example:

```
copy store.keys store.trust client.trust KVROOT/security/
```

5. Use the securityconfig tool to verify installation. For example:

```
security-> config verify -secdir KVROOT/security
Security configuration verification passed.
```

> **Note:**
>
> For previous releases (prior 4.1), you need to verify the configuration manually. The distinguished name of the certificate must match the content inside of dnmatch in security.xml. Also, the user-generated keystore password must be the same as the one stored in the wallet (store.wallet) or the password file (store.pwd). Finally, the truststore (store.trust) must contain the CA certificates and the one used for Oracle NoSQL Database.

6. Finally, deliver the client.trust or import the CA certificates into the client truststore.

Guidelines for Configuring External Certificates for an Existing Default Secure Installation

Follow these steps to install external certificates in an existing secure NoSQL database installation that uses a default security configuration and a self-signed certificate:

> **Note:**
>
> This procedure assumes you already have a Java keystore and truststore setup. For more information see Java KeyStore Preparation.

1. Create a new security configuration that uses external certificates:

```
security-> config create -root NEW_KVROOT \
-pwdmgr wallet -kspwd password \
-param "client:serverIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "internal:serverIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "internal:clientIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "ha:serverIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "ha:clientIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)"
```

> **Note:**
>
> NEW_KVROOT should be a temporary directory that only holds the generated security files.

2. Replace the keystore and truststore files with your own on each server that will host a Storage Node. For example:

```
copy store.keys store.trust client.trust NEW_KVROOT/security/
```

3. It is easier to install an external certificate if the existing store does not needs to be kept accessible during the certificate installation. To do this, you only need to copy the entire new security security configuration to each Storage Node and then restart all of the Storage Nodes.

```
copy -r NEW_KVROOT/security KVROOT
java -jar <KVHOME>/lib/kvstore.jar stop -root KVROOT
java -jar <KVHOME>/lib/kvstore.jar start -root KVROOT&
```

4. If the existing store need to be kept accessible during the credential changes instead, then you should create an interim truststore and modify the security parameters having dnmatch field. On the configuration host, merge the truststore entries by using the config merge-trust command, and also import the root and intermediate certificate:

```
java -jar <KVHOME>/lib/kvstore.jar
securityconfig \
config merge-trust -root KVROOT -source-root NEW_KVROOT
keytool -import -keystore KVROOT/security/store.trust -file
ca.cert.pem -alias root
keytool -import -keystore KVROOT/security/store.trust -file
intermediate.cert.pem -alias intermediate
copy KVROOT/security/store.trust KVROOT/security/client.trust
```

> **Note:**
>
> In a multiple Storage Node deployment, you need to copy the new configuration to each host's new configuration directory and run merge-trust on each host like in the example above.

5. Copy the updated client.trust file to the security directory on each host so that clients can use it to access the store.

6. To keep the store accessible during the process, change the dnmatch value in the security configuration to be a compatible one. The values specified in the various dnmatch(xxx) expressions are a regular expression, as specified by java.util.regex.Pattern. The compatible "dnmatch" value should be in the form of old certificate DN | new certificate DN. In this case, the CN=NoSQL| represents the DN of the original self-signed certificate.

```
security-> config update \
-secdir KVROOT/security \
-param "client:serverIdentityAllowed=dnmatch
(CN=NoSQL|CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "internal:serverIdentityAllowed=dnmatch
(CN=NoSQL|CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
```

```
ST=California, C=US)" \
-param "internal:clientIdentityAllowed=dnmatch
(CN=NoSQL|CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "ha:serverIdentityAllowed=dnmatch
(CN=NoSQL|CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "ha:clientIdentityAllowed=dnmatch
(CN=NoSQL|CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)"
```

If clients set the login property `oracle.kv.ssl.hostnameVerifier`, change the value of the dnmatch field. For example:

```
oracle.kv.ssl.trustStore=client.trust
oracle.kv.transport=ssl
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL|CN\=myhost,
OU\=TeamA, O\=MyCompany, L\=Unknown, ST\=California, C\=US)
```

7. Check that all Replication Nodes are online and then restart each Storage Node one by one using the following commands:

```
java -jar <KVHOME>/lib/kvstore.jar stop -root KVROOT

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

8. Copy the updated store.keys file to the security directory on each host. Then, check that all Replication Nodes are online and restart each Storage Node one by one using the following commands:

```
java -jar <KVHOME>/lib/kvstore.jar stop -root KVROOT

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

9. For all Storage Nodes, remove the obsolete certificate `mykey` in store.trust. Also, rename the new certificate `mykey_2` to `mykey` using the following command:

```
keytool -delete -keystore KVROOT/security/store.trust \
-alias mykey

keytool -changealias -keystore \
KVROOT/security/store.trust -alias mykey_2 -destalias mykey
```

Guidelines for Updating the External Certificates

Follow these steps to update the external certificates for a secure installation that is already using external certificates.

> **Note:**
>
> This procedure assumes you already have a Java keystore and truststore setup having the updated external certificates. For more information see Java KeyStore Preparation.

1. Create a new security configuration that uses external certificates.

```
security-> config create -root NEW_KVROOT \
-pwdmgr wallet -kspwd password \
-param "client:serverIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "internal:serverIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "internal:clientIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "ha:serverIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)" \
-param "ha:clientIdentityAllowed=dnmatch
(CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
ST=California, C=US)"
```

2. Replace the keystore and server truststores with your own:

```
copy store.keys store.trust NEW_KVROOT/security/
```

3. On the configuration host, merge the truststore entries with the `NEW_KVROOT` directory. Check that all Replication Nodes are online and then restart each Storage Node one by one using the following commands. If the updated external certificate uses a different distinguished name, update the dnmatch value in the security configuration to a compatible one using the procedures found in Guidelines for Configuring External Certificates for an Existing Default Secure Installation.

```
java -jar <KVHOME>/lib/kvstore.jar securityconfig \
config merge-trust -root KVROOT \
-source-root <NEW_KVROOT>

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

> **Note:**
>
> You do not need to update the client truststore if the new certificates are signed by the same Certificate Authority (CA).

4. Copy the updated store.keys file to the security directory on each host. Then, check that all Replication Nodes are online and restart each Storage Node one by one using the following commands:

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

5. For all Storage Nodes, remove the obsolete certificate `mykey` in store.trust. Also, rename the new certificate `mykey_2` to `mykey` using the following command:

```
keytool -delete -keystore KVROOT/security/store.trust \
-alias mykey

keytool -changealias -keystore \
KVROOT/security/store.trust -alias mykey_2 -destalias mykey
```

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).

A
Password Complexity Policies

A set of default rules should be followed when creating or updating a user password in order to enhance security. Password complexity policies do not apply to the SSL keystore password.

Any user that has the SYSOPER privilege can customize the global password policies and control the password complexity when creating or updating the passwords for users. Oracle NoSQL Database checks if the new passwords are sufficiently complex to prevent attackers to break into the system.

When using the `CREATE USER` and `ALTER USER` commands, Oracle NoSQL Database will check if the passwords set comply with the password complexity policies. Otherwise, a message will be shown with all the violating policies. For example:

```
kv-> exec "create user test identified by \"password\""
Error handling command
exec "create user test identified by \"password\"":
Error: User error in query: CreateUser failed for:
Password must have at least 9 characters
```

You can enable or disable the password complexity policy like this:

```
kv-> change-policy -params passwordComplexityCheck=true
```

Then, you can change the password complexity policies by using the `change-policy` command. For example:

```
kv-> change-policy -params
passwordMinLength=20 passwordMaxLength=50 passwordMinUpper=3
passwordMinLower=3 passwordMinDigit=3 passwordMinSpecial=3
```

The following password security parameters can be set:

Parameter Name	Value Range and Type	Description
passwordAllowedSpecial	Sub set or full set of #$%&'()* +,-./:; <=>?@[]^_`{|} (string)~	Lists the allowed special characters.
passwordComplexityCheck	[true|false] (boolean)	Whether to enable the password complexity checking. The default value is true.
passwordMaxLength	1 - 2048 (integer)	The maximum length of a password. The default value is 256.
passwordMinDigit	0 - 2048 (integer)	The minimum required number of numeric digits. The default value is 2.
passwordMinLength	1 - 2048 (integer)	The Minimum length of a password. The default value is 9.

Parameter Name	Value Range and Type	Description
passwordMinLower	0 - 2048 (integer)	The minimum required number of lower case letters. The default value is 2.
passwordMinSpecial	0 - 2048 (integer)	The minimum required number of special characters. The default value is 2.
passwordMinUpper	0 - 2048 (integer)	The minimum required number of upper case letters. The default value is 2.
passwordNotStoreName	[true|false] (boolean)	If true, password should not be the same as current store name, nor is it the store name spelled backwards or with the numbers 1–100 appended. The default value is true.
passwordNotUserName	[true|false] (boolean)	If true, password should not be the same as current user name, nor is it the user name spelled backwards or with the numbers 1-100 appended. The default value is true.
passwordProhibited	list of strings separated by comma (string)	Simple list of words that are not allowed to be used as a password. The default reserved words are: oracle,password,user,nosql.
passwordRemember	0 - 256 (integer)	The maximum number of passwords to be remembered that are not allowed to be reused when setting a new password. The default value is 3.

Most of the special characters in the standard US keyboard are allowed to be used in a password with exception of `" (double quote) and \ (back slash).`

If you want to allow certain special characters use the `passwordAllowedSpecial` parameter. For example:

```
kv-> change-policy -params passwordAllowedSpecial="@# $"
```

If you want to enforce the password complexity for existing users, then you need to set the existing user's password to expired, like this:

1. Review the existing users in the system:

   ```
   kv-> exec "show users"
   user:id=u1 name=root
   user:id=u3 name=user1
   user:id=u4 name=user2
   user:id=u5 name=user3
   ```

2. Set the new password complexity policies:

   ```
   kv-> change-policy -params
   passwordComplexityCheck=true passwordMinLength=9
   ```

```
passwordMinUpper=2 passwordMinLower=2
passwordMinSpecial=2 passwordMinDigit=2
```

3. Finally, change the existing user's password life time to be expired:

```
kv-> exec "alter user user1 password expire"
Statement completed successfully
kv-> exec "alter user user2 password expire"
Statement completed successfully
kv-> exec "alter user user3 password expire"
Statement completed successfully
```

In this case, user 1, 2, and 3 will need to re-new their password according to the new policy. For example, when trying to login with user 1, the system will prompt to change the password:

```
java -jar kvstore.jar runadmin -host localhost
-port 5000 -security login_file
user1's password:
The password of user1 has expired, it is required to
change the password.
Enter the new password:
Re-enter the new password:
```

If the new password violates any password complexity policies, an exception with a violation message will be thrown. For example:

```
user1's password:
The password of user1 has expired, it is required to
change the password.
Enter the new password: password
Re-enter the new password: password
Exception in thread "main" oracle.kv.AuthenticationFailureException:
Renew password failed:
Password must have at least 9 characters
Password must contain at least 2 upper case letters
Password must contain at least 2 lower case letters
...
```

B
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` (Java built-in key and certificate management tool) commands:

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

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

C

Java KeyStore Preparation

The following example demonstrates how to use `keytool` to prepare keystore and truststore with external certificate. If you want to import an existing private/public key pair generated by an external tool instead, see Import Key Pair to Java Keystore.

1. Generate a keypair and store it into store.keys

```
keytool -genkeypair -keystore store.keys \
-alias shared -keyAlg RSA -keySize 2048 \
-validity 365 -dname <xxx> \
-storepass <sslPwd>
...
Enter key password for <shared>
(RETURN if same as keystore password):
```

> **Note:**
>
> `Store.keys` is the default name of Oracle NoSQL Database keystore and `shared` is the default alias of the Oracle NoSQL Database certificate You can customize the name by specifying a security parameter in the makebootconfig command or the securityconfig utility. Additionally, you can specify the algorithm, size and validity of key.

2. Generate a certificate request and send to CA.

```
keytool -certreq -keystore store.keys -alias
shared -file myhost.csr
Enter keystore password:
```

3. A public trusted CA usually signs the certificate after receiving your csr file. A pem file is generated (myhost.cert.pem).

4. Import certificates that are part of a certificate chain in order. If there are multiple intermediate certificates, they also need to be imported in order.

```
keytool -import -file ca.cert.pem
-keystore store.keys -alias root
keytool -import -file intermediate.cert.pem -keystore store.keys
-alias intermediate
# After importing the root and intermediate certificates,
# install the signed certificate for this server. The alias name
# must be specified.
keytool -import -file myhost.cert.pem -keystore store.keys
-alias shared
Certificate reply was installed in keystore
```

5. Verify the installation by checking the certificate content in `store.keys`:

```
keytool -list -v -keystore store.keys -alias shared
Certificate chain length: 3
Certificate[1]:
Owner: CN=myhost, OU=TeamA, O=MyCompany, L=Unknown,
```

```
ST=California, C=US
Issuer: CN=intermediate CA, OU=CA, O=MyCompany,
ST=California, C=US
```

The certificate chain length should match the number of certificates in the chain that were imported, in this case, three.

6. Build server truststore (store.trust). The server truststore must contain the signed certificate as well as the root and intermediate certificate. Note that the server and client truststores need to use the same password as that of the keystore.

```
keytool -export -file store.tmp
-keystore store.keys -alias shared
keytool -import -keystore store.trust -file store.tmp
keytool -import -keystore store.trust -file ca.cert.pem
-alias root
keytool -import -keystore store.trust -file intermediate.cert.pem
-alias intermediate
```

7. Create client truststore (client.trust). In this case, import the root and intermediate certificates into the client truststore.

```
keytool -import -keystore client.trust
-file ca.cert.pem -alias root
keytool -import -keystore client.trust -file intermediate.cert.pem
-alias intermediate
```

Import Key Pair to Java Keystore

This section describes how to import an existing private/public key pair into Java keystore. This is useful if you have your own tools for generating a CA signed key pair. The procedure assumes you already have the root and intermediate certificates as well as the private key and its signed certificate.

To import an existing key pair:

1. Build the certificate chain and convert the private key and certificate files into a PKCS12 file.

```
cat myhost.pem intermediate.pem root.pem > import.pem
openssl pkcs12 -export -in import.pem -inkey myhost.key.pem
-name shared > server.p12
```

2. Import the PKCS12 file into Java keystore:

```
keytool -importkeystore -srckeystore server.p12
-destkeystore store.keys -srcstoretype pkcs12 -alias shared
```

3. Finally, to complete the preparation of the Java keystore, perform the procedures for creating the server and client truststore described in the previous section.

D

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

EVOLVE_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:

- SHOW USERS
- SHOW ROLES

USRVIEW:

- SHOW USERS (for self only)

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
- tableIKeysIterator

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

E

Configuring the Kerberos Administrative Utility

Before using `kadmin`, you first need to configure permissions on the KDC. Kerberos uses an Access Control List (ACL) file to determine which principals have administrative access to the Kerberos database and their level of access.

The default location of the Kerberos ACL file is <LOCALSTATEDIR>/krb5kdc/kadm5.acl, where `LOCALSTATEDIR` is the directory prefix where the KDC databases are located. This location can be modified by the `acl_file` variable in `kdc.conf`.

Lines containing ACL entries have this format:

```
principal  permissions  [target_principal  [restrictions] ]
```

> **Note:**
>
> Line order in the ACL file is important. The first matching entry will control access for an actor principal on a target principal.

To configure `kadmin`, perform the following steps:

1. Create an access control list file and put the Kerberos principal of at least one of the administrators into it. For example:

   ```
   */admin@EXAMPLE.COM  *
   ```

   In this case, any principal in the `EXAMPLE.COM` realm with an `admin` instance has all administrative privileges on the KDC.

   For example, `joe/admin@EXAMPLE.com` has all privileges over the realm's Kerberos database.

2. Create the first principal before accessing the KDC remotely:

   ```
   kadmin.local: addprinc -randkey admin/admin
   kadmin.local: ktadd –k kadm5.keytab admin/admin
   ```

> **Note:**
>
> To enable passwordless autentication, copy `kadm5.keytab` to any client machine.

Kadmin can also be used to perform security maintenance. For more information, see Guidelines for Updating Kerberos Passwords.

F

Manually Registering Oracle NoSQL Database Service Principal

The securityconfig tool allows you to create service principals and generate keytabs assuming that each Storage Node is able to access the Kerberos admin interface remotely. Although this is the typical configuration most Kerberos deployments have, you may want to use a non-standard configuration. You can manage service principals by using only kadmin.local or ktutil utility on the KDC host.

To register Oracle NoSQL Database service principal by using `kadmin.local`:

1. Register the service principal:

   ```
   kadmin.local: addprinc -randkey nosql/abc.example.com
   ```

2. Extract the keytab file using the `ktadd` command:

   ```
   kadmin.local: ktadd –norandkey –k keytab nosql/abc.example.com
   ```

3. Verify the entries of the generated keytab using the `klist` tool:

   ```
   klist –k –e /tmp/keytab
   Keytab name: FILE:keytab
   KVNO Principal
   ---- -------------------------------------------------
   12   nosql/abc.example.com@EXAMPLE.COM
                            (AES-128 CTS mode with 96-bit SHA-1 HMAC)
   12   nosql/abc.example.com@EXAMPLE.COM
                            (AES-256 CTS mode with 96-bit SHA-1 HMAC)
   ```

4. Copy the keytab of Oracle NoSQL Database server principal to each Storage Node. The default location is under kvroot/security. You need to create the security directory.

5. Run makebootconfig or securityconfig utility to complete the rest of the Kerberos security configuration.

To register Oracle NoSQL Database service principal by using `ktutil` utility:

1. Add principal entries:

   ```
   ktutil: add_entry –password –p \
   nosql/abc.example.com –k 1 –e aes128-cts-hmac-sha1-96
   Password for nosql/abc.example.com@EXAMPLE.COM:
   ktutil:add_entry –password –p nosql/abc.example.com \
   –k 1 –e aes256-cts-hmac-sha1-96
   Password for nosql/abc.example.com@EXAMPLE.COM
   ```

2. Write the current keylist into the keytab file:

   ```
   Ktutil: write_kt keytab
   ```

3. Verify the entries of the generated keytab using the `klist` tool:

   ```
   klist –k –e /tmp/keytab
   Keytab name: FILE:keytab
   KVNO Principal
   ```

```
---- ---------------------------------------------
12   nosql/abc.example.com@EXAMPLE.COM
                          (AES-128 CTS mode with 96-bit SHA-1 HMAC)
12   nosql/abc.example.com@EXAMPLE.COM
                          (AES-256 CTS mode with 96-bit SHA-1 HMAC)
```

4. Copy the keytab of Oracle NoSQL Database server principal to each Storage Node. The default location is under kvroot/security. You need to create the security directory.
5. Run makebootconfig or securityconfig utility to complete the rest of the Kerberos security configuration.

G

Third Party Licenses

All of the third party licenses used by Oracle NoSQL Database are described in the License document.