Oracle® NoSQL Database

Security Guide

Release 26.1
E85375-37
July 2026

Oracle NoSQL Database Security Guide, Release 26.1

E85375-37

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

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

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

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

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

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Contents

Preface

Conventions Used in This Book i

1 Introducing Oracle NoSQL Database Security

2 Security Configuration

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

3 Performing a Secure Oracle NoSQL Database Installation

Single Node Secure Deployment 1
- Adding Security to a New Installation 1
- Adding Security to an Existing Installation 5

Multiple Node Secure Deployment 7
- Adding Security to a New Installation 7
- Adding Security to an Existing Installation 11

4 Kerberos Authentication Service

Installation Prerequisites 1
Kerberos Principal 1
Keytabs 2
Kadmin and kadmin.local 2

Copyright © 2011, 2026, Oracle and/or its affiliates.

Kerberos Security Properties 2
Setting Security Properties in a security login file 3
Setting Security Properties through KVStoreConfig 4
Using Security Properties to Log In 4
Using credential cache 5
Using a keytab 6
JAAS programming framework integration 6
Performing a Secure Oracle NoSQL Database Installation with Kerberos 8
Adding Kerberos to a New Installation 8
Adding Kerberos to an Existing Secure Installation 12
Using Oracle NoSQL Database with Kerberos and Microsoft Active Directory (AD) 15

5 External Password Storage

Oracle Wallet 1
Password store file 3

6 Security.xml Parameters

Top-level parameters 1
Transport parameters 2

7 Encryption

SSL model 1
SSL communication properties 2
Disk Encryption in a Linux Environment 3

8 Configuring Authentication

User Management 1
User Creation 1
User Modification 3
User Removal 3
User Status 4
User Login 4
Sessions 4

9 Configuring Authorization

Privileges 1
System Privileges 1
Object Privileges 2

Copyright © 2011, 2026, Oracle and/or its affiliates.

Table Ownership 4
Privilege Hierarchy 4
Roles 5
System Built-in Roles 5
User-Defined Roles 6
Managing Roles, Privileges and Users 8
Role Creation 8
Role Removal 8
Role Status 8
Grant Roles or Privileges 9
Revoke Roles or Privileges 11
Granting Authorization Access to Namespaces 11

10 Security Policies

Security Policy Modifications 1

11 Audit Logging

Security Log Messages 1

12 Keeping Oracle NoSQL Database Secure

Guidelines for Securing the Configuration 1
Guideline for Securing Store Topology 1
Guidelines for Deploying Secure Applications 2
Guidelines for Securing the SSL protocol 2
Guidelines for Disabling TLSv1.1 and TLSv1 Protocols 2
Guidelines for enabling TLSV1.3 protocol 4
Guidelines for using JMX securely 6
Guidelines for using PKCS12 Java KeyStore 7
Default Security Configuration 7
Updating KeyStore Type of an Existing Security Configuration 8
Updating SSL Keys and Certificates 10
Guidelines for Updating Keystore Passwords 10
Guidelines for Updating Kerberos Passwords 12
Guidelines for Updating SSL Keys and Certificates 14
Guidelines for Configuring External Certificates for a new Installation 20
Guidelines for Configuring External Certificates for an Existing Default Secure Installation 22
Guidelines for Updating the External Certificates 25
Guidelines for Operating System Security 28

Copyright © 2011, 2026, Oracle and/or its affiliates.

Guidelines for Resetting Admin Password 28

A Password Complexity Policies

B SSL keystore generation

C Java KeyStore Preparation

Import Key Pair to Java Keystore C-4

D KVStore Required Privileges

Privileges for Accessing CLI Commands D-1
Privileges for DDL Commands D-3
Privileges for Accessing KVStore APIs D-4
Privileges for Accessing KVStore TableAPIs D-5
Privileges for Accessing KvLargeObject APIs D-6
Privileges for Running XRegion Service D-6

E Configuring the Kerberos Administrative Utility

F Manually Registering Oracle NoSQL Database Service Principal

G Generating Certificate and Private Key for the Oracle NoSQL Database Proxy

Guidelines for Generating Self-Signed Certificate and Private Key using OpenSSL G-1
Guidelines for Generating Certificate Chain and Private Key using OpenSSL G-3
Troubleshooting issues with self-signed certificate G-6

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 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. See Security in Full Text Search in the *Integrations Guide*.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 either the `securityconfig` or the `makebootconfig` 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.

The by-product of using one of the tools is they create 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 these tools help 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

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

  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.

- `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 the data store.

- `client.trust`

  A truststore file used by clients is generated.

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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 -Xmx64m -Xms64m
-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 <server key and trust store password>]
[-kstype <key and trust store type>]
[-ctspwd <client.trust 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

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 in single node and multi-node deployments, see Performing a Secure Oracle NoSQL Database Installation.

Configuring Security with Securityconfig

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

```
java -Xmx64m -Xms64m -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 <server key and trust store password>]
[-kstype <key and trust store type>]
[-ctspwd <client.trust 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:

Copyright © 2011, 2026, Oracle and/or its affiliates.

- `-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.

- `-kspwd <server key and trust store password>`

  Specifies the password used to create keystore and truststore needed by NoSQL Database Server.

- `-kstype <key and trust store type>`

  Specifies the store type of keystore and truststore. It must be either JKS or PKCS12.

- `-ctspwd <client.trust password>`

  Specifies the password to create PKCS12 password-protected truststore used by client applications to connect NoSQL Database Server.

- `-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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

  - `-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 in single node and multi-node deployments, see Performing a Secure Oracle NoSQL Database Installation.

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

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 data store instance stopped, and must be performed on each Storage Node of the store.

where:

- `-root $KVROOT`

  A data store 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>
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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> [-kstype <keystore type>] [-ctspwd
<client.trust password>] [-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".

- `-kstype <keystore type>`

  Specify the store type to update. Only PKCS12 is allowed. This command updates the keystore (store.keys) and truststore (store.trust) used by NoSQL Database Server to PKCS12 password-protected store. If the Java used to run this command supports password-less truststore, utilities create the truststore used by client applications (client.trust) as a PKCS12 password-less store. If not, utilities fall back to create a JKS store instead if no password specified using `-ctspwd <client.trust password>`.

- `-ctspwd <client.trust password>`

  When updating JKS keystore and truststore in a security configuration to PKCS12, you can use this flag to specify the password to create PKCS12 password-protected truststore used by client applications (client.trust).

- `-param <param=value*>`

  List of security parameters to update.

For example:

```
security-> config update -secdir security -kstype PKCS12 -param
clientAuthRequired=false
Configuration updated.
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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
clientAuthRequired=true
clientIdentityAllowed=dnmatch(CN=NoSQL)
clientKeyAlias=shared
serverIdentityAllowed=dnmatch(CN=NoSQL)
serverKeyAlias=shared
transportType=ssl

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

ha Transport parameters:
allowProtocols=TLSv1.2
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:
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
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

Copyright © 2011, 2026, Oracle and/or its affiliates.

performing security maintenance, particularly when you need to update the SSL key/certificate. See Guidelines for Updating SSL Keys and Certificates

When running the `config merge-trust` command, the `securityconfig` tool will verify the existence of the referenced files (`client.trust` and `store.trust`) and will combine trust entries from the source security configuration (For example: `/users/user_name/tmp/kvroot/newKey`) into the primary security configuration(`$KVROOT/security`). After running this command, the `client.trust` and `store.trust` files will have two SSL certificate entries.

```
config merge-trust
-root <secroot> [-secdir <security dir>]
-source-root <source secroot> [-source-secdir <source secdir>] [-ctspwd
<client.trust password>]
```

> **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".

- `ctspwd <client.trust password>`

  When merging truststore entries from a security configuration that uses PKCS12 store, utilities create a PKCS12 password-protected truststore used by client applications (client.trust) if password specified using `-ctspwd`.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar makebootconfig \
   -root $KVROOT -port 5000 \
   -host $KVHOST -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
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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):

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to `1`. Setting `MALLOC_ARENA_MAX` to `1` ensures that the memory usage is restricted to the specified heap size.

```
nohup java -Xmx64m -Xms64m \
-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 the data store. 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 data store server host ($KVHOST). To do this, use the following command:

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

6. Use the `configure -name` command to specify the name of the data store that you want to configure:

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

After naming the data store, you can create at least one zone.

```
kv-> plan deploy-zone -name zone_name -rf 1 -type primary -wait
```

Every data store has an administration database. You must deploy the Storage Node first and then deploy an Administration process on the same node to continue configuring the database.

Copyright © 2011, 2026, Oracle and/or its affiliates.

When you deploy the node, provide the zone ID, the node's network name, and its registry port number.

```
kv-> plan deploy-sn -znname zone_name -host hostname -port 5000 -wait
```

Having deployed the node, create the Admin process on the node that you just deployed, using the deploy-admin command. This command requires the Storage Node ID and an optional plan name.

> **Note**
>
> Note: You can obtain the Storage Node ID using the `show topology` command. See show topology for more details.

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

The final step in your configuration process is to create Replication Nodes on every node in your store. You do this using the `topology create` and `plan deploy-topology` commands.

```
kv-> topology create -name storeTopo -pool AllStorageNodes -partitions 150
kv-> plan deploy-topology -name storeTopo -wait
```

Your store is fully installed and configured.

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-> execute 'CREATE USER root IDENTIFIED BY \"password\" ADMIN
Statement completed 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 -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar securityconfig \
pwdfile create -file $KVROOT/security/login.passwd
java -Xmx64m -Xms64m \
-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
```

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

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar runadmin -port 5000 \
   -host localhost -security mylogin.txt
   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
   oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
   ```

   Note that the hostname verifier provides a way for Oracle NoSQL Database clients to specify the name that they expect the Oracle NoSQL Database server hosts to use during SSL handshake (when they attempt to connect server using SSL/TLS).

   For a secure store using the standard configuration, server hosts will be required to authenticate themselves, and clients will use their SSL truststore to confirm that the server authenticates with a trusted identity. The hostname verifier provides the additional assurance that the server host authenticates using the expected identity, not just any trusted identity.

   This additional check is desirable if either the truststore contains multiple certificates or if the certificate it contains is a CA certificate rather than a self-signed or leaf certificate. In both those cases, the truststore can vouch for multiple identities. The host verifier allows the user to specify the specific identity that is expected.

   The only hostname verifier currently supported is the `dnmatch` verifier, which must be specified in the form of `dnmatch(distinguished-name)`, where distinguished name must be the NoSQL DB server certificate's distinguished name. If you are using the default security configuration, then the hostname verifier in the example specifies that the server should authenticate with a certificate whose distinguished name is `CN=NoSQL`. This is the name used in the server certificates that the system generates by default.

   The verification is performed by checking if the distinguished name of server certificate match the specified `dnmatch` expressions, which uses regular expressions as specified by `java.util.regex.Pattern`. The distinguished name specified in `dnmatch` must be 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. 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. See User Login.

Copyright © 2011, 2026, Oracle and/or its affiliates.

Adding Security to an Existing Installation

To add security to an existing Oracle NoSQL Database installation:

1. Shut down the KVStore instance:

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

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

   ```
   java -Xmx64m -Xms64m \
   -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.
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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):

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to `1`. Setting `MALLOC_ARENA_MAX` to `1` ensures that the memory usage is restricted to the specified heap size.

```
nohup java -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
-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-> execute 'CREATE USER root IDENTIFIED BY \"password\" ADMIN
Statement completed 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 -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar securityconfig \
wallet create -dir KVROOT/security/login.wallet
java -Xmx64m -Xms64m \
-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
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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 -Xmx64m -Xms64m \
    -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
    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 -Xmx64m -Xms64m \
    -jar $KVHOME/lib/kvstore.jar makebootconfig \
    -root $KVROOT -port 5000 \
    -host node01 -harange 5010,5020 \
    -store-security configure -pwdmgr wallet -capacity 3
    ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar makebootconfig \
   -root $KVROOT \
   -host node02 \
   -port 6000 \
   -harange 6010,6020 \
   -capacity 3 \
   -store-security enable \
   -pwdmgr wallet

   java -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar makebootconfig \
   -root $KVROOT \
   -host node03 \
   -port 7000 \
   -harange 7010,7020 \
   -capacity 3 \
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
-store-security enable \
-pwdmgr wallet
```

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

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to `1`. Setting `MALLOC_ARENA_MAX` to `1` ensures that the memory usage is restricted to the specified heap size.

```
nohup java -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
-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-> execute 'CREATE USER root IDENTIFIED BY \"password\" ADMIN
Statement completed 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 -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar securityconfig \
wallet create -dir $KVROOT/security/login.wallet
java -Xmx64m -Xms64m \
-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
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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 -Xmx64m -Xms64m \
    -jar $KVHOME/lib/kvstore.jar runadmin -port 5000 \
    -host localhost -security adminlogin.txt
    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
    oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
    ```

    For more information, see User Login.

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

    ```
    sql-> CREATE USER user1 IDENTIFIED BY "<password>"
    Statement completed successfully


    sql-> CREATE USER user2 IDENTIFIED BY "<password>"
    Statement completed successfully
    ```

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

    ```
    java -Xmx64m -Xms64m \
    -jar $KVHOME/lib/kvstore.jar securityconfig \
    wallet create -dir $KVROOT/security/users.wallet
    java -Xmx64m -Xms64m \
    -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
    java -Xmx64m -Xms64m \
    -jar $KVHOME/lib/kvstore.jar securityconfig wallet secret \
    -dir $KVROOT/security/users.wallet -set -alias user2
    Enter the secret value to store: ********
    ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
Re-enter the secret value for verification: ********
Secret created
```

> **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 -Xmx64m -Xms64m \
    -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
    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 -Xmx64m -Xms64m \
   -jar KVHOME/lib/kvstore.jar stop \
   -root KVROOT
   ```

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

   ```
   java -Xmx64m -Xms64m \
   -jar KVHOME/lib/kvstore.jar securityconfig
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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:

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to 1. Setting `MALLOC_ARENA_MAX` to 1 ensures that the memory usage is restricted to the specified heap size.

```
java -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
   -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-> execute 'CREATE USER root IDENTIFIED BY \"password\" ADMIN
   Statement completed 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 -Xmx64m -Xms64m \
    -jar KVHOME/lib/kvstore.jar securityconfig \
    wallet create -dir KVROOT/security/login.wallet
    java -Xmx64m -Xms64m \
    -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
    ```

    > **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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

```
java -Xmx64m -Xms64m \
-Doracle.kv.security=adminlogin.txt \
-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
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

For more information, see User Login.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

Copyright © 2011, 2026, Oracle and/or its affiliates.

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
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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: ********
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
-Doracle.kv.security=mylogin.txt \
-jar $KVHOME/lib/kvstore.jar runadmin -port 5000 -host localhost
Logged in admin as krbuser
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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);
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
// 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 -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar makebootconfig \
   -root $KVROOT -port 5000 \
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
-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 data store 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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

6. Run makebootconfig on the other two nodes:
   - Add Kerberos and create their individual service principal and keytab:

```
java -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 6000 \
-host node02 -harange 6010,6020 \
-capacity 3 \
-store-security enable
```

```
java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar makebootconfig \
-root $KVROOT -port 6000 \
-host node03 -harange 6010,6020 \
-capacity 3 \
-store-security enable
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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:

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to 1. Setting `MALLOC_ARENA_MAX` to 1 ensures that the memory usage is restricted to the specified heap size.

```
nohup java -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
-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": **********
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
    -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:

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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 data store instance:

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

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

```
java -Xmx64m -Xms64m \
-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 data store instance stopped, and must be performed on each Storage Node of the store.

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

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to 1. Setting `MALLOC_ARENA_MAX` to 1 ensures that the memory usage is restricted to the specified heap size.

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

4. Start `runadmin` in security mode on the data store server host (node01). To do this, use the following command:

   ```
   java -Xmx64m -Xms64m \
   -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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
-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
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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
     ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
-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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

```
java -Xmx64m -Xms64m \
-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 -Xmx64m -Xms64m \
-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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

```
java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar makebootconfig \
-root $KVROOT -port 5000 \
-host node02 -harange 5010,5020 \
-store-security enable

java -Xmx64m -Xms64m \
-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
  ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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 -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar makebootconfig \
-root $KVROOT -port 5000 \
-host node02 -harange 5010,5020 \
-store-security enable

java -Xmx64m -Xms64m \
-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:

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to 1. Setting `MALLOC_ARENA_MAX` to 1 ensures that the memory usage is restricted to the specified heap size.

```
nohup java -Xmx64m -Xms64m \
-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 the data store. 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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

```
java -Xmx64m -Xms64m \
-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 data store that you want to configure, and then complete store deployment. For more information, see the *Oracle NoSQL Database Administrator'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 -Xmx64m -Xms64m \
-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
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

An Oracle wallet is a mechanism used to securely store sensitive information such as passwords that are required for authentication and secure communication. It is only available in the Enterprise Edition version of Oracle NoSQL Database.

If you want to use a wallet to store your passwords, you must specify `wallet` as the password management mechanism while configuring the store security using the `securityconfig` utility or `makebootconfig` utility.

The following steps show you how this can be done.

- Using `securityconfig`:
  Run the `securityconfig` utility to set up the basic store configuration with security. For more information on the `securityconfig` utility, see Configuring Security with Securityconfig.

  ```
  java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar securityconfig
  ```

  Use the `config create` command with the `-pwdmgr` option to specify the password management mechanism. In this case, the mechanism is `wallet`.

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

- Using `makebootconfig`:
  Run the `makebootconfig` utility with `-store-security configure` option. For more information on the `makebootconfig` utility, see Configuring Security with Makebootconfig. For example:

  ```
  java -Xmx64m -Xms64m \
  -jar KVHOME/lib/kvstore.jar makebootconfig \
  ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
    -root KVROOT -port 5000 \
    -host node01 -harange 5010,5020 \
    -store-security configure -pwdmgr wallet -capacity 3


    Enter a password for the Java KeyStore:
```

Enter a password for your store and then re-enter it for verification. The configuration tool generates the security related files. It creates a wallet directory `store.wallet` that contains the keystore access password. The keystore access password protects the keys and certificates used for secure communication within the database cluster. It contains secret information that should be known only to the server processes. The file should remain on the server side.

The `client.security` and `client.trust` files should be copied to the client and used when connecting to the data store.

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

You can create and manipulate wallets to store user passwords. User passwords are required to authenticate individual users or clients to the database for data access and operations based on the privileges granted to them.

The following commands show you how this can be done.

To create a new wallet, run the `wallet create` command:

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

For example:

```
security-> wallet create -dir $KVROOT/security/loginwallet
Created
```

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

For example, to manipulate the secret (password) associated with user John, run the `wallet secret` command as follows:

```
security-> wallet secret -dir $KVROOT/security/loginwallet -set -alias John
Enter the secret value to store: <password for user John>
Re-enter the secret value for verification: <password for user John>
Secret created
```

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.

In order to authenticate as a user, you must provide the user name and the wallet directory as security properties while connecting to the data store. To do this, create a copy of the `client.security` file present in the `security` folder that was created earlier. The `client.security` file contains the security properties and configuration details required for clients to securely connect to the data store. In the copied file, include additional properties containing the user name and the wallet directory.

The following steps show you how this can be done.

Make a copy of the `client.security` file present in the `security` folder. Let us call it `user.login`.

```
cp client.security user.login
```

Include the following additional properties in `user.login` file:

```
oracle.kv.auth.username=John
oracle.kv.auth.wallet.dir=$KVROOT/security/loginwallet
```

Now, you can use `user.login` to securely connect to the store as user John:

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar runadmin -port 8000 -host localhost
-security kvroot/security/user.login
Logged in to Admin as John
kv->
```

Password store file

A password store file is a mechanism used to securely store sensitive information such as passwords that are required for authentication and secure communication. This mechanism is available in the Community and Enterprise Edition versions of Oracle NoSQL Database. The password store file is an unencrypted file. It is read-protected to prevent unauthorized access.

If you want to use a password store file to store your passwords, you must specify password file (`pwdfile`) as the password management mechanism while configuring the store security using the `securityconfig` utility or `makebootconfig` utility.

The following steps show you how this can be done.

- Using `securityconfig`:

Copyright © 2011, 2026, Oracle and/or its affiliates.

Run the `securityconfig` utility to set up the basic store configuration with security. For more information on the `securityconfig` utility, see Configuring Security with Securityconfig.

```
java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar securityconfig
```

Use the `config create` command with the `-pwdmgr` option to specify the password management mechanism. In this case, the mechanism is `pwdfile`.

```
security-> config create -pwdmgr pwdfile -root $KVROOT
Enter a password for the Java KeyStore:
```

- Using `makebootconfig`:
  Run the `makebootconfig` utility with `-store-security configure` option. For more information on the `makebootconfig` utility, see Configuring Security with Makebootconfig. For example:

```
java -Xmx64m -Xms64m \
-jar KVHOME/lib/kvstore.jar makebootconfig \
-root KVROOT -port 5000 \
-host node01 -harange 5010,5020 \
-store-security configure -pwdmgr pwdfile -capacity 3


Enter a password for the Java KeyStore:
```

Enter a password for your store and then re-enter it for verification. The configuration tool generates the security related files. The file `store.passwd` is the password store file that contains the keystore access password. It contains secret information that should be known only to the server processes. The file should should remain on the server side.

The `client.security` and `client.trust` files should be copied to the client and used when connecting to the data store.

```
Created files
    $KVROOT/security/store.keys
    $KVROOT/security/store.trust
    $KVROOT/security/client.trust
    $KVROOT/security/client.security
    $KVROOT/security/store.passwd
    $KVROOT/security/security.xml
Created
```

You can create and manipulate password store files to store user passwords. User passwords are required to authenticate individual users or clients to the database for data access and operations based on the privileges granted to them.

The following commands show you how this can be done.

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

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


Copyright © 2011, 2026, Oracle and/or its affiliates.

For example:

```
security-> pwdfile create -file $KVROOT/security/login.pwd
Created
```

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.

For example, to manipulate the secret (password) associated with user John, run the `pwdfile secret` command as follows:

```
security-> pwdfile secret -file $KVROOT/security/login.pwd -set -alias John
Enter the secret value to store: <password for user John>
Re-enter the secret value for verification: <password for user John>
Secret created
```

In order to authenticate as a user, you must provide the user name and the password file as security properties while connecting to the data store. To do this, create a copy of the `client.security` file present in the `security` folder that was created earlier. The `client.security` file contains the security properties and configuration details required for clients to securely connect to the data store. In the copied file, include additional properties containing the user name and the password file.

The following steps show you how this can be done.

Make a copy of the `client.security` file present in the `security` folder. Let us call it `user.login`.

```
cp client.security user.login
```

Include the following additional properties in `user.login` file:

```
oracle.kv.auth.username=John
oracle.kv.auth.pwdfile.file=$KVROOT/security/login.pwd
```

Now, you can use `user.login` to securely connect to the store as user John:

```
java -Xmx64m -Xms64m -jar lib/kvstore.jar runadmin -port 8000 -host localhost
-security kvroot/security/user.login
Logged in to Admin as John
kv->
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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 JKS keystore type is used by default.

- keystoreSigPrivateKeyAlias

  Specifies the keystore alias that identifies the keypair used by replication nodes to create signatures. If not specified, the alias "`shared`" is used.

- truststoreSigPublicKeyAlias

  Specifies the truststore alias that identifies the certificate used by replication nodes to verify signatures. If not specified, the alias "`mykey`" is used.

- 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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

- truststoreType

  Identifies the type of keystore that the truststore property references. If not set, the JKS keystore type is used by default.

- 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

Copyright © 2011, 2026, Oracle and/or its affiliates.

The keystore alias that identifies the keypair used by the store services, including Storage Nodes, Replication Nodes, Admins, and Arbiter Nodes. If not specified, the alias "`shared`" is used.

- clientKeyAlias

  The keystore alias that identifies the keypair used by either a direct connect Java client or a proxy. See Configuring the Proxy for more details. If not specified, the alias "`shared`" is used.

- 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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

7 Encryption

Network data encryption provides data privacy so that unauthorized parties are unable to view plain text data during transmission across 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 -Xmx64m -Xms64m \
   -Doracle.kv.security=mylogin.txt \
   -jar $KVHOME/lib/kvstore.jar runadmin
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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
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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

- `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.

Disk Encryption in a Linux Environment

If you are using the Linux operating system, you can secure your data by configuring disk encryption to encrypt whole disks (including removable media), partitions, software RAID volumes, logical volumes, as well as your NoSQL files.

`dm-crypt` is the Linux kernel's device mapper crypto target which provides transparent disk encryption subsystem in the Linux kernel using the kernel crypto API.

`Cryptsetup` is the command line tool to interface with `dm-crypt` for creating, accessing and managing encrypted devices. The most commonly used encryption is `Cryptsetup` for the Linux Unified Key Setup (LUKS) extension, which stores all of the needed setup information for `dm-crypt` on the disk itself and abstracts partition and key management in an attempt to improve ease of use.

This topic demonstrates how to convert a normal disk to a `dm-crypt` enabled disk and vice versa using the command-line interface.

Assume that you have the following disks in your Linux system. The `df -h` command displays the amount of available disk space for each disk.

```
$df -h
/dev/nvme0n1 2.9T 76G 2.7T 3% /ons/nvme0n1
/dev/nvme1n1 2.9T 76G 2.7T 3% /ons/nvme1n1
...
```

If you nominate disk `/dev/nvme0n1` to store databases, then you should encrypt this disk to secure the data within it.

Normal disk to a dm-crypt enabled disk:

Execute the following commands to convert a normal disk to a `dm-crypt` enabled disk:

1. Unmount the file system on the disk.

   ```
   sudo umount -l /dev/nvme0n1
   ```

2. Generate the key to be used by `luksFormat`.

   ```
   sudo dd if=/dev/urandom of=/home/opc/key0.key bs=1 count=4096
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

3. Initialize a LUKS partition and set the initial key.

```
sudo /usr/sbin/cryptsetup -q -s 512 \
luksFormat /dev/nvme0n1 /home/opc/key0.key
```

4. Open the LUKS partition on disk/device and set up a mapping name.

```
sudo /usr/sbin/cryptsetup --allow-discards \
luksOpen -d /home/opc/key0.key /dev/nvme0n1 dm-nvme0n1
```

5. Create an `ext4` file system on the disk.

```
sudo /sbin/mkfs.ext4 /dev/mapper/dm-nvme0n1
```

6. Set parameters for the `ext4` file system.

```
sudo /usr/sbin/tune2fs -e remount-ro /dev/mapper/dm-nvme0n1
```

7. Mount the file system to a specified directory.

```
sudo mount /dev/mapper/dm-nvme0n1 /ons/nvme0n1
```

dm-crypt enabled disk to normal disk:

If you want to convert the encrypted disk back to its normal state, execute the following steps:

1. Unmount the file system on the disk.

```
sudo umount -l /ons/nvme0n1
```

2. Remove luks mapping.

```
sudo /usr/sbin/cryptsetup luksClose /dev/mapper/dm-nvme0n1
```

3. Create an `ext4` file system on the disk.

```
sudo /sbin/mkfs.ext4 /dev/nvme0n1
```

4. Mount the file system on a specified directory.

```
sudo mount /dev/nvme0n1 /ons/nvme0n1
```

> **Note**
>
> If you convert a normal disk to a `dm-crypt` enabled disk or convert a `dm-crypt` enabled disk to a normal disk, you cannot bring the disk back to its previous state without losing its data. This is because the `mkfs.ext4` command will format the disk. Therefore, all the data stored in the disk will be lost.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

You can create, modify, or remove users in the Oracle NoSQL Database through the SQL CLI, where the commands for manipulating users are exposed in SQL format through DDL API. To start the SQL CLI, see Starting the SQL shell. You can also display information about a specific user account, as well as get a summary list of registered users. 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_time_unit])
  [ACCOUNT LOCK|UNLOCK]
  [ADMIN]
```

where:

- *user_name*

  Assigns a name to identify a user. If you are creating a Kerberos user, the *user_name* must match the fully qualified principal name created in the Key Distribution Center (KDC) at your site. A username is an ID, just as a table name. The formal definition for each ID is as follows:

  ```
  ALPHA (ALPHA | DIGIT | UNDER)* ;
  ```

  Each ID must start with a letter (`a-z`, `A-Z`), followed by other letters, numerical values (`0 - 9`), and underscore (_) characters. There is no ID size limit for the number of characters it contains. An an ID can consist of as many characters as the memory required to

Copyright © 2011, 2026, Oracle and/or its affiliates.

accommodate its length. In practice, most sites have name length recommendations, but they are not checked or enforced by Oracle NoSQL Database.

Kerberos users must have different names from existing users, since you cannot change the authentication type of an existing user.

- `IDENTIFIED BY "password"`

  Indicates that Oracle NoSQL Database authenticates the new user by the password you assign. The new user must log on using that password.

  > **Note**
  >
  > You must specify a user password with quotation marks, for example, "*password*".

- `PASSWORD EXPIRE`

  Specifies that the assigned password has already expired. With this setting the user is forced to change the given password as soon as they initially login. They must enter a password of their choice (which meets any site requirements) before accessing Oracle NoSQL Database.

- `PASSWORD LIFETIME {INT duration_time_unit}`

  Indicates the password duration unit, which is required for using the assigned password. Enter the integer *time_unit* as follows:

  ```
  time_unit : (SECONDS | MINUTES | HOURS | DAYS)
  ```

  Using zero (0) with any time unit specifies that the password never expires. Entering a negative value causes an error. If you do not specify a PASSWORD LIFETIME time_unit, the lifetime from the global configuration is used. The default for this parameter is 180 days.

  Following is a basic example of creating new user Kate, `IDENTIFIED BY` a password you assign to her, represented here as "*password*", with a `PASSWORD LIFETIME` duration specifying the integer unit of time as `30 DAYS`.

  ```
  sql->create USER Kate IDENTIFIED BY "<password>" PASSWORD LIFETIME:30 DAYS;
  ```

- `ACCOUNT {LOCK | UNLOCK}`

  Specify `ACCOUNT LOCK` to lock a user's account to disable access. An Admin can use this option to remove access from a user, but retain the account. Then, as required, reinstate the user account specifying `ACCOUNT UNLOCK`.

- `ADMIN Clause`

  Specify `ADMIN` to grant the user `sysadmin` role automatically.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 "non-expiring".

- `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 [CASCADE]
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

Use the `DROP USER user_name` command to remove the specified user account (users cannot remove themselves), where *`user_name`* is the name of the user to drop.

If the user has existing tables, drop each of the tables first, and then drop the user. Alternatively, use the optional `CASCADE` option, which drops the user tables along with the user.

For example:

```
sql->drop USER Kate cascade;
```

Dropping a user occurs immediately. If another user was accessing tables that the user owned, the tables are no longer available for DML or DDL operations.

User Status

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

For example:

```
sql->show users;
user:KVStoreUser[id=u3 name=Alice]
user:KVStoreUser[id=u2 name=Kate]
user:KVStoreUser[id=u1 name=Ken]
```

> **Note**
>
> The User ID values are incremented sequentially as you add each user. They are an internal mechanism for ensuring each user is unique.

To view detailed information about a specific user:

```
sql-> show user Kate;
KVStoreUser[id=u2 name=Kate] enabled=true auth-type=LOCAL current-passwd-
expiration=2026-05-04 11:20:56 UTC retain-passwd=inactive granted-
roles=[public, readwrite]
```

User Login

You can use either the `-username <user>` or the `-security <path to security file>` argument to login to the SQL CLI. For more details on logging in to the SQL client, see Starting the SQL shell.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 and index create/drop.
- `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 and index information.
- `USRVIEW`

Copyright © 2011, 2026, Oracle and/or its affiliates.

Can query users' own information, like their own user information, the status of commands they issued and have access to the current topology information using Oracle NoSQL Database Java direct driver.

- `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.

- `WRITE_SYSTEM_TABLE`

  Can make modifications to system tables if the necessary read and write privileges are granted for the table. The multi-region agent is the intended user of this privilege. The `WRITE_SYSTEM_TABLE` privilege is needed when you need to use the `Load` program to restore records into a secure data store. See Using the Load Program for more details. Typically, normal users should not modify system tables.

- `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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

- `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.

The object privileges defined in Oracle NoSQL Database for namespaces are:

- `CREATE_ANY_NAMESPACE`

  Can create any namespace. When creating a new namespace the user will also be able to READ_IN_NAMESPACE, INSERT_IN_NAMESPACE, DELETE_IN_NAMESPACE on the respective new namespace.

- `DROP_ANY_NAMESPACE`

  Can drop any namespace.

- `CREATE_TABLE_IN_NAMESPACE`

  Can create tables in a specific namespace.

- `DROP_TABLE_IN_NAMESPACE`

  Can drop tables in a specific namespace.

- `EVOLVE_TABLE_IN_NAMESPACE`

  Can evolve tables in a specific namespace.

- `CREATE_INDEX_IN_NAMESPACE`

  Can create an index in a specific namespace.

- `DROP_INDEX_IN_NAMESPACE`

  Can drop an index in a specific namespace.

- `READ_IN_NAMESPACE`

  Can read items in a specific namespace.

- `INSERT_IN_NAMESPACE`

  Can insert items in a specific namespace.

- `DELETE_IN_NAMESPACE`

  Can delete items in a specific namespace.

- `MODIFY_IN_NAMESPACE`

  Has all the DDL privileges for a specific namespace.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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`.

If you have one or more namespaces in your store, you can grant authorization permissions to a namespace to determine who can access both the namespace and the tables within it. For more details, see Granting Authorization Access to Namespaces.

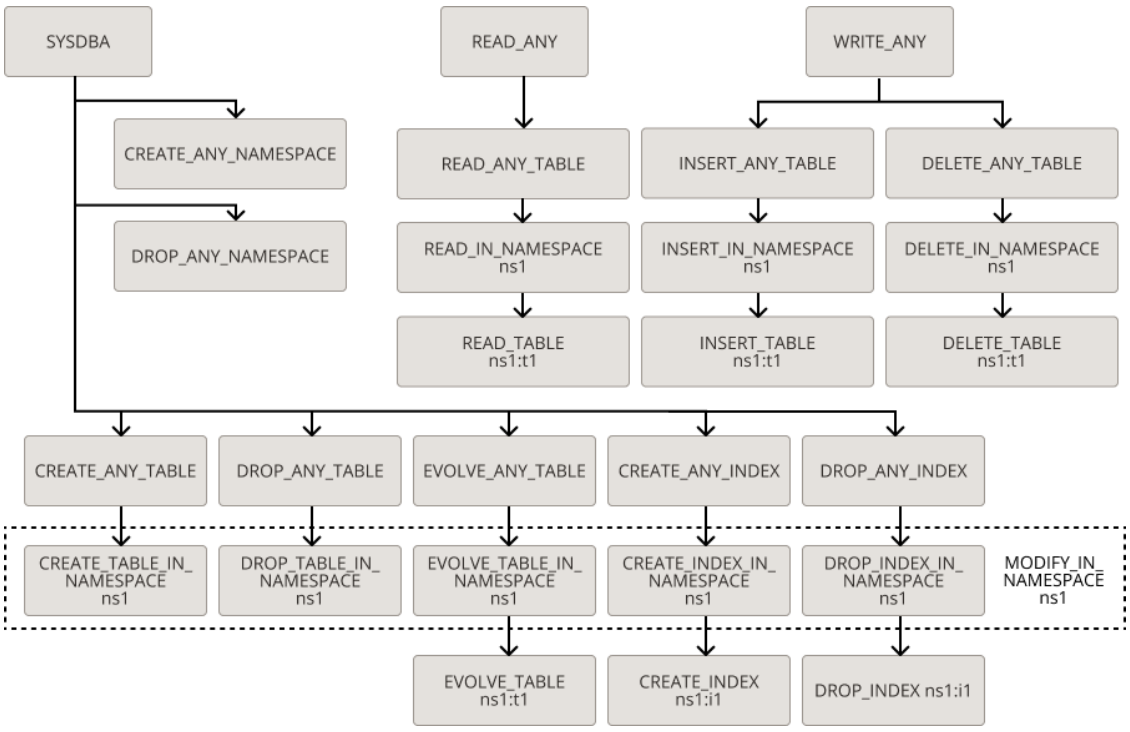
Privilege Hierarchy

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:

Copyright © 2011, 2026, Oracle and/or its affiliates.



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

You can perform read operation on a table in namespace ns1:t1 if the user has any of the following privileges specified:

- READ_ANY
- READ_ANY_TABLE
- READ_IN_NAMESPACE ns1
- READ_TABLE ns1:t1

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

- `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 and index 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.

- `writesystable`

  Contains the WRITE_SYSTEM_TABLE privilege. Users with this role can modify system tables if they have the necessary read and write privileges. The multi-region table agent is the intended user of this role. The `WRITE_SYSTEM_TABLE` privilege is needed when you need to use the `Load` program to restore records into a secure data store. See Using the Load Program for more details. Typically, normal users should not modify system tables.

- `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. Users with this role can view current topology of store using Oracle NoSQL Database Java direct driver.

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 from the SQL CLI. To start SQL CLI, see Starting the SQL shell.

```
sql-> create ROLE role_name;


sql-> 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 Reserved Words in the *SQL Reference Guide*.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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

Create two users with the following commands:

```
sql-> create USER Ken IDENTIFIED BY "<password>";


sql-> create USER Kate IDENTIFIED BY "<password>";
```

> **Note**
>
> Use the following guidelines to define a password :
>
> - Password must have at least 9 characters
> - Password must have at least 2 upper case letters
> - Password must have at least 2 special characters
>
> For more information, see Password Complexity Policies

Now, create two roles – `manager` with the `write_any` privilege and `employee` with the `read_any` privilege:

```
sql->create ROLE manager;
sql->grant WRITE_ANY to manager;
sql->create ROLE employee;
sql->grant READ_ANY to employee;
```

The next example shows granting role `employee` to role `manager` (sub-role of manager), and then grants role `manager` to user `Kate`. User Kate then has both `manager` and `employee` role, with both of their privileges, to `write_any` data to the store, and `read_any` data.

```
sql->GRANT employee TO ROLE manager;
sql->GRANT manager TO USER Kate;
```

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

```
sql->show USER Kate;
KVStoreUser[id=u2 name=Kate] enabled=true auth-type=LOCAL current-passwd-
expiration=2026-05-04 11:23:48 UTC retain-passwd=inactive granted-
roles=[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:

```
sql->drop ROLE manager;
sql->show USER Kate;
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
KVStoreUser[id=u2 name=Kate] enabled=true auth-type=LOCAL current-passwd-
expiration=2026-05-04 11:23:48 UTC retain-passwd=inactive granted-
roles=[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 system revokes the manager role from user Kate automatically, as well as the `employee` role. In the above example, Kate cannot perform any read or write operations.

> **Note**
>
> Granting circular 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 security operations, including commands to create, drop, show, grant or revoke roles to or from users, and to grant or revoke privileges to or from roles. All these statements can be executed through the SQL CLI.

To start SQL CLI, see Starting the SQL shell.

Role Creation

```
CREATE ROLE role_name
```

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

For example,

```
sql->create ROLE administrator;
```

Role Removal

```
DROP ROLE role_name
```

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

For example,

```
sql->drop ROLE administrator;
```

Role Status

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

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

Copyright © 2011, 2026, Oracle and/or its affiliates.

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,

```
sql->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:

```
sql->show AS JSON ROLE dbadmin;
{"name":"dbadmin", "assignable":"true", "readonly":"true","granted-
privileges":["SYSDBA","DBVIEW"],"granted-roles":[]}
```

> **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:

```
sql->create ROLE emptablereader;
sql->grant READ_TABLE ON emp to emptablereader;
sql->show ROLE emptablereader
RoleInstance[ name=emptablereader assignable=true readonly=false granted-
privileges=[READ_TABLE(emptable)]]
```

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`

Copyright © 2011, 2026, Oracle and/or its affiliates.

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, you can grant a role with fewer privileges to one with more privileges, such as `employee` to role `manager`:

```
sql->grant employee to ROLE manager;
```

You can grant the role `manager` to user `Kate` as follows:

```
sql->grant manager to USER Kate;
```

If you try to grant the same role in the other direction, an error occurs because this would lead to a cyclic definition of role manager.

```
sql->grant manager to ROLE employee;
Error handling command grant manager to ROLE employee: Error: User error in
query: GrantRoles failed for: Could not complete grant, circular role grant
detected
```

You can now add new privileges to their defined role. For example, to grant the system privilege `READ_ANY` to `manager`:

```
sql->grant READ_ANY to manager;
```

To grant read permission on table T1 to `manager` :

```
sql->grant READ_TABLE ON T1 to manager;
```

To know more about granting table privileges, see Table Ownership.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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`:

```
sql->revoke employee from ROLE manager;
```

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

```
sql->revoke manager from USER Kate;
```

Granting Authorization Access to Namespaces

You can manage permission for users or roles to access namespaces and tables. These are the applicable permissions given to the developers and other users:

Copyright © 2011, 2026, Oracle and/or its affiliates.

Table 9-1 Namespace Privileges and Permissions

Privilege	Description
CREATE_ANY_NAMESPACE DROP_ANY_NAMESPACE	Grant permission to a role to create or drop any namespace. GRANT CREATE_ANY_NAMESPACE TO <*Role*>; GRANT DROP_ANY_NAMESPACE TO <*Role*>;
CREATE_TABLE_IN_NAMESPACE DROP_TABLE_IN_NAMESPACE EVOLVE_TABLE_IN_NAMESPACE	Grant permission to a user or to a role to create, drop or evolve tables in a specific namespace. You can evolve tables to update table definitions, add or remove fields, or change field properties, such as a default value. You may even add a particular kind of column, like an IDENTITY column, to increment some value automatically. Only tables that already exist in the store are candidates for table evolution. For more details, see Alter Table. GRANT CREATE_TABLE_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>; GRANT DROP_TABLE_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>; GRANT EVOLVE_TABLE_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>*user_role*;
CREATE_INDEX_IN_NAMESPACE DROP_INDEX_IN_NAMESPACE	Grant permission to a user or to a role to create or drop an index in a specific namespace. GRANT CREATE_INDEX_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>; GRANT DROP_INDEX_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>;
READ_IN_NAMESPACE INSERT_IN_NAMESPACE DELETE_IN_NAMESPACE	Grant permission to a user or role to read, insert, or delete items in a specific namespace. GRANT READ_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>; GRANT INSERT_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>; GRANT DELETE_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>;
MODIFY_IN_NAMESPACE	Helper label for granting or revoking permissions to all DDL privileges for a specific namespace to a user or role. GRANT MODIFY_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>; REVOKE MODIFY_IN_NAMESPACE ON NAMESPACE *namespace_name* TO <*User*|*Role*>;

Copyright © 2011, 2026, Oracle and/or its affiliates.

Grant privileges on a namespace

You can grant permissions to a role or a user on a namespace. Following is the syntax for granting permissions on a namespace:

```
GRANT {Namespace-scoped privileges} ON NAMESPACE namespace_name TO <User|Role>
Namespace-scoped privileges ::= namespace_privilege [, namespace_privilege]
```

where,

- namespace_privilege

  The namespace privilege that can be granted to a user or a role. For more information on the applicable privileges, see the *Privilege* column in the Namespace Privileges and Permissions table.

- namespace_name

  The namespace that the user wishes to access.

- <User|Role>

  The name of the KVStore user or the role of a user.

For example, you can grant read access to a user for all the tables in the namespace.

```
GRANT READ_IN_NAMESPACE ON NAMESPACE ns1 TO Kate
```

Here, ns1 is the namespace and Kate is the user.

> **Note**
>
> The label MODIFY_IN_NAMESPACE can be used as a helper for granting or revoking permissions to all DDL privileges for a specific namespace to a user or role.

Revoke privileges on a namespace

You can revoke the permissions from a role or a user on a namespace. Following is the syntax for revoking the permissions on a namespace.

```
REVOKE {Namespace-scoped privileges} ON NAMESPACE namespace_name FROM <User|
Role>
Namespace-scoped privileges ::= namespace_privilege [, namespace_privilege]
```

where,

- namespace_privilege

  The namespace privilege that can be revoked from a user or a role. For more information on the applicable privileges, see the *Privilege* column in the Namespace Privileges and Permissions table.

- namespace_name

  The namespace that the user wishes to access.

- <User|Role>

Copyright © 2011, 2026, Oracle and/or its affiliates.

The name of the KVStore user or the role of a user.

For example, you can revoke the read access from a user for all the tables in the namespace.

```
REVOKE READ_IN_NAMESPACE ON NAMESPACE ns1 FROM Kate
```

Here, ns1 is the namespace and Kate is the user.

> **Note**
>
> The label MODIFY_IN_NAMESPACE can be used as a helper for granting or revoking permissions to all DDL privileges for a specific namespace to a user or role.

The following example shows a set of commands to grant and revoke privileges on a namespace to a user or role:

Example: Namespace Scoped Privileges

```
CREATE NAMESPACE IF NOT EXISTS ns1;
GRANT MODIFY_IN_NAMESPACE ON NAMESPACE ns1 TO <User|Role>;
CREATE TABLE ns1:t (id INTEGER, name STRING, primary key (id));
INSERT INTO ns1:t VALUES (1, 'Smith');
SELECT * FROM ns1:t;
REVOKE CREATE_TABLE_IN_NAMESPACE ON NAMESPACE ns1 FROM <User|Role>;
DROP NAMESPACE ns1 CASCADE;
```

> **Note**
>
> You can save all of the above commands as a **sql** script and execute it in a single command.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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>

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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>]
```

> **Note**
>
> If the log files are compressed, you can use the `gzcat` command to view the contents without uncompressing the zipped files. Use the `zgrep` command to search the compressed log files. You can also uncompress the files into another directory. For more information, see Log File Compression in the *Administrator's Guide.*

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Guideline for Securing Store Topology

All Oracle NoSQL Database users will be granted a default role `public`, which cannot be revoked.

Authenticated applications using Oracle NoSQL Database Java direct driver will keep a memory copy of the current topology of the database store for dispatching requests to the right node in the store. All database users can view the content of the current topology of the database store. A topology in Oracle NoSQL Database has the basic information about store layout including zones, storage nodes, shards, replication nodes, and administrative services, as well as hostnames and registry ports of each storage node. For more information on topologies see topologies in the Concepts Guide.

If there is a security and compliance requirement that the access of topology information must be limited to a certain group of users, applications should access Oracle NoSQL Database through Oracle NoSQL Database Proxy using various Oracle NoSQL Database Drivers instead of Java direct driver. For more information on Oracle NoSQL Database Drivers, see Oracle NoSQL Database Drivers in Developer's Guide. Oracle NoSQL Database Proxy should be deployed as an intermediary between applications and Oracle NoSQL Database store in this

Copyright © 2011, 2026, Oracle and/or its affiliates.

case. For more information on the Oracle NoSQL Database Proxy, see Oracle NoSQL Database Proxy in the Administrator's Guide.

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` file should be made available on the application machine.

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 Disabling TLSv1.1 and TLSv1 Protocols

Update TLS protocol configuration to TLSv1.2 only

NoSQL Database has disabled TLSv1 and TLSv1.1 protocols in the default security configuration, the only protocol enabled is TLSv1.2.

Upgrade Implication:

This change doesn't remove the support of TLSv1.1 and TLSv1.2 but only disable them in the default security configuration. Upgrading to 24.1.11 release with security configuration created by previous release won't have compatibility issue, but it's recommended to disable the TLSv1.1 and TLSv1 in the existing NoSQL Database installation.

Prerequisite:

Before updating the TLS protocol to TLSv1.2 only, you must ensure the existing security configuration has already enabled the TLSv1.2 protocol, otherwise your NoSQL Database server won't be functional during the update.

1. . Check if protocol settings in the security configuration of your NoSQL Database server has enabled TLSv1.2 protocol. Run `securityconfig` utility to verify if protocols have TLSv1.2 included.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig config show -secdir
         $KVROOT/security
   ```

   If protocols in the security configuration don't have TLSv1.2, follow the section "Enable TLSv1.2 protocol" to enable TLSv1.2 first.
2. Check the client application login properties. Verify if the following NoSQL login property has TLSv1.2.

Copyright © 2011, 2026, Oracle and/or its affiliates.

For example:

```
oracle.kv.ssl.protocols="TLSv1.2,TLSv1.1,TLSv1"
```

Add TLSv1.2 and restart the client application if it wasn't specified in this security property.

Enable TLSv1.2 protocol

This is the procedure to enable TLSv1.2 in the NoSQL Database security configuration. It assumes the existing security configuration only has enabled TLSv1.1 and TLSv1.

1. Make two copies of the existing security configuration directory. Keep one as backup, and use the other for updating the protocols.
2. Update the SSL protocols in the copied security configuration directory.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig \
    config update -secdir security \
    -param "allowProtocols=TLSv1.2,TLSv1.1,TLSv1" \
    -param "clientAllowProtocols=TLSv1.2,TLSv1.1,TLSv1"
   ```

3. Verify if protocols in the updated security configuration has TLSv1.2 enabled.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig config show -secdir
         $KVROOT/security
   ```

   Verify if the protocol has TLSv1.2.
4. Copy the updated security directory to each Storage Node, and replace the old security configuration directory. Then, check that all Replication Nodes (RN) are online and restart each Storage Node, one by one, using the following command.

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

5. Start the Admin CLI, and check that all Replication Nodes (RNs) are up using the `ping` command:

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

   Output:

   ```
   Logged in admin as anonymous


   kv-> ping
   ```

Update TLS protocol to TLSv1.2 only

This is the procedure to update the existing security configuration to only enable protocol TLSv1.2. It assumes the TLSv1.2 is already enabled in the security configuration.

1. Update login properties of the client application. Update oracle.kv.ssl.protocols to have TLSv1.2 only (if it exists).

Copyright © 2011, 2026, Oracle and/or its affiliates.

2. Make two copies of existing security configuration directory. Keep one as backup, and use the other one for updating the protocols.
3. Update the SSL protocols in the copied security configuration directory.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig \
   config update -secdir security \
   -param "allowProtocols=TLSv1.2" -param "clientAllowProtocols=TLSv1.2"
   ```

4. Verify if protocols in the updated security configuration has only TLSv1.2.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig config show -secdir
         $KVROOT/security
   ```

   Verify if protocols has TLSv1.2 only.
5. Copy the updated security directory to each server node (Storage Node), and replace the old security configuration directory. Then, check that all Replication Nodes are online and restart each Storage Node, one by one, using the following command:

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

6. Start the Admin CLI, and check that all Replication Nodes (RNs) are up using the `ping` command:

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

   Output:

   ```
   Logged in admin as anonymous


   kv-> ping
   ```

Guidelines for enabling TLSV1.3 protocol

Update TLS protocol configuration to enable TLSv1.3

Oracle NoSQL Database now supports TLSv1.3 protocol. To run NoSQL Database and application with TLSv1.3, you must use JDK11 or later, JDK8 Update 261 (JDK 8u261) or later. Since 21.3 release, NoSQL Database adds TLSv1.3 protocol to the default TLS protocols of security configuration created via `makebootconfig` or `securityconfig` utility. It's recommended to update the TLS protocol of existing security configuration to latest protocol TLSv1.3 since it is the most secure.

Enable TLSv1.3 protocol

This is the procedure to update the existing security configuration to enable protocol TLSv1.3. It assumes the existing security configuration is made by previous NoSQL Database releases, which has TLSv1.2 enabled.

Copyright © 2011, 2026, Oracle and/or its affiliates.

1. Update login properties of client application. Add TLSv1.3 to oracle.kv.ssl.protocols if it exists. Then restart the client application to make the protocol change to take effect.

   ```
   oracle.kv.ssl.protocols="TLSv1.3,TLSv1.2"
   ```

2. Make two copies of existing security configuration directory of the storage node. Keep one as backup, and the other one for updating the protocols.

   > **Note**
   >
   > This step is to update the security configuration of storage node used by NoSQL Database server, as opposed to the client application changes in the previous step.

3. Update the SSL protocols in the copied security configuration directory.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig \
   config update -secdir security \
   -param "allowProtocols=TLSv1.3,TLSv1.2" \
   -param "clientAllowProtocols=TLSv1.3,TLSv1.2"
   ```

4. Verify if protocols in the updated security configuration has TLSv1.3.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig config
   show -secdir $KVROOT/security
   ```

   Verify if the protocol has TLSv1.3.

5. Copy the updated security directory to each Storage Node, and replace the old security configuration directory. Then, check that all Replication Nodes (RN) are online and restart each Storage Node, one by one, using the following command.

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

6. Start the Admin CLI, and check that all Replication Nodes (RNs) are up using the `ping` command:

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

   Output:

   ```
   Logged in admin as anonymous


   kv-> ping
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

Update TLS protocol to TLSv1.3 only

This is the procedure to enable TLSv1.3 only in NoSQL Database security configuration. It assumes the existing security configuration has already TLSv1.3 protocol, if not, follow the last procedure to enable TLSv1.3 first.

After this change, all client application only can establish TLS connections with NoSQL Database using TLSv1.3 protocol. Before this change, you must ensure oracle.kv.ssl.protocols in login properties file of the client applications have TLSv1.3 enabled, otherwise follow the section "Enable TLSv1.3 protocol" to enable TLSv1.3 first.

1. Make two copies of existing security configuration directory. Keep one as backup, and use the other one for updating the protocols.
2. Update the SSL protocols in the copied security configuration directory.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig \
   config update -secdir security \
   -param "allowProtocols=TLSv1.3" \
   -param "clientAllowProtocols=TLSv1.3"
   ```

3. Verify if protocols in the updated security configuration has TLSv1.3 only.

   ```
   java -jar kv/lib/kvstore.jar securityconfig config
   show -secdir KVROOT/security
   ```

   Verify if protocols has TLSv1.3 only.
4. Copy the updated security directory to each Storage Node, and replace the old security configuration directory. Then, check that all Replication Nodes (RN) are online and restart each Storage Node, one by one, using the following command.

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

5. Start the Admin CLI, and check that all Replication Nodes (RNs) are up using the `ping` command:

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

   Output:

   ```
   Logged in admin as anonymous


   kv-> ping
   ```

Guidelines for using JMX securely

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

Copyright © 2011, 2026, Oracle and/or its affiliates.

- 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.
- If you create the client.trust in PKCS12 format and protected by password, you need to specify the password for client.trust by running the jconsole command:

  ```
  jconsole -J-Djavax.net.ssl.trustStore=/home/nosql/client.trust \
  -J-Djavax.net.ssl.trustStorePassword=<client.trust password> node01:5000
  ```

Guidelines for using PKCS12 Java KeyStore

Oracle NoSQL Database supports Java KeyStore in PKCS12 format. From the 22.1 release onward, NoSQL Database switched the KeyStore type of default security configuration created by makebootconfig or securityconfig utility to PKCS12.

> **Note**
>
> Oracle recommends that you switch the KeyStore format of the existing security configuration to PKCS12, which is an industry-standard format.

Default Security Configuration

Starting from release 22.1, in the default security configuration, the database server KeyStore and TrustStore, typically named store.keys and store.trust respectively, are created in PKCS12 format and protected by password specified using `-kspwd`.

The TrustStore, client.trust, that is used by the client application is created in PKCS12 format and password-less, by default, if no password is specified using `-ctspwd`. Additionally, if the Java used to run the configuration utilities does not support password-less PKCS12 store, utilities fall back to create the client.trust in JKS format. The Java version supporting password-less PKCS12 must have security properties keystore.pkcs12.certProtectionAlgorithm and keystore.pkcs12.macAlgorithm available. The minimum JAVA versions required for this feature are JDK 8u301 for Java 8, JDK 11.0.12 for Java 11, and the first release of Java 17.

When creating the configuration, you can create the client.trust in PKCS12 format and protected by the password specified using `-ctspwd`. If you are using the client.trust protected by password, the password must be specified in the login properties file. The following two login properties are supported:

- oracle.kv.ssl.trustStorePassword
- oracle.kv.ssl.trustStorePasswordAlias

The client application can specify the password in the login properties file using the `oracle.kv.ssl.trustStorePassword` property or store the password in External

Copyright © 2011, 2026, Oracle and/or its affiliates.

Password Storage and specify only the alias name using the `oracle.kv.ssl.trustStorePasswordAlias` property.

Example:

```
#Security property settings for communication with KVStore servers using
password
oracle.kv.ssl.trustStore=client.trust
oracle.kv.ssl.trustStoreType=PKCS12
oracle.kv.ssl.trustStorePassword=<client.trust password>
oracle.kv.ssl.protocols=TLSv1.3,TLSv1.2
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
oracle.kv.transport=ssl
```

```
#Security property settings for communication with KVStore servers using
password alias
oracle.kv.ssl.trustStore=client.trust
oracle.kv.ssl.trustStoreType=PKCS12
oracle.kv.ssl.trustStorePasswordAlias=cts
oracle.kv.ssl.protocols=TLSv1.3,TLSv1.2
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
oracle.kv.transport=ssl
oracle.kv.auth.wallet.dir=<wallet_directory>
oracle.kv.auth.username=<user_name>
```

Updating KeyStore Type of an Existing Security Configuration

The security configuration created by releases earlier than NoSQL Database Release 22.1 generates all Java KeyStores in JKS format. You need to perform the following steps to upgrade Java KeyStores to PKCS12 format.

1. Copy the existing security configuration directory from one of the NoSQL Database storage nodes.
2. Run the following command to update the KeyStore to PKCS12 format:

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig config update -secdir
   <security dir> -kstype PKCS12 [-ctspwd <client.trust password>]
   ```

   This command converts the existing KeyStore (store.keys) and TrustStore (store.trust) used by the NoSQL Database server to PKCS12 format and reuses the KeyStore password of stores in the existing configuration. Similar to the configuration creation, this command also creates a new password-less client.trust in PKCS12 format, if no password is specified using `-ctspwd`. If Java doesn't support the password-less PKCS12 store, it falls back to creating a JKS format client.trust.
3. Run the show security configuration command to verify that the store type is updated to PKCS12 format.

   ```
   java -jar $KVHOME/lib/kvstore.jar securityconfig config show -secdir
   security
        Security parameters:
        certMode=shared
        internalAuth=ssl
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
keystore=store.keys
keystorePasswordAlias=keystore
keystoreType=PKCS12
securityEnabled=true
truststore=store.trust
truststoreType=PKCS12
walletDir=store.wallet

internal Transport parameters:
...
Keystore: security/store.keys
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

shared, Feb 11, 2022, PrivateKeyEntry,
Certificate fingerprint (SHA-256): AA:98:B8:C6...

Keystore: security/store.trust
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

mykey, Feb 11, 2022, trustedCertEntry,
Certificate fingerprint (SHA-256): AA:98:B8:C6...
```

4. In the configuration directory, verify that there is a backup of each Java KeyStore named with the suffix '.old'.

```
ls $KVROOT/security
     store.wallet
     store.trust.old
     store.trust
     store.keys.old
     store.keys
     security.xml
     client.trust.old
     client.trust
     client.security
```

5. Verify that the base login properties file, `client.security`, is updated with PKCS12 format.

```
cat security/client.security
     #Security property settings for communication with KVStore servers
     #Fri Feb 11 10:59:39 PST 2022
     oracle.kv.ssl.trustStore=client.trust
     oracle.kv.ssl.trustStoreType=PKCS12
     oracle.kv.ssl.protocols=TLSv1.2
     oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
     oracle.kv.transport=ssl
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

6. Copy the updated security directory to each server node (Storage Node), and replace the old security configuration directory. Then, check that all Replication Nodes are online and restart each Storage Node, one by one, using the following command:

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

7. Start the Admin CLI, and check that all Replication Nodes (RNs) are up using the `ping` command:

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

Output:

```
Logged in admin as anonymous


kv-> ping
```

Updating SSL Keys and Certificates

When updating the SSL keys and certificates with a new security configuration using Guidelines for Updating SSL Keys and Certificates, the merge-trust command automatically converts the merged truststore (store.trust) used by the NoSQL Database server to PKCS12 format and protected by the original KeyStore password of the existing security configuration. It also creates a new password-less client.trust in PKCS12 format if no password is specified using `-ctspwd`. If the Java used to run `merge-trust` command doesn't support the password-less PKCS12 store, it falls back to create a JKS client.trust.

After updating the key and certificate, if you want to keep KeyStore and truststore in JKS format, you need to create the new security configuration to be merged by running the following command:

```
java -jar $KVHOME/lib/kvstore.jar securityconfig config /
create -root /Users/my_name/tmp/kvroot/newKey -kspwd 123456 -kstype JKS
```

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:

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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 -Xmx64m -Xms64m \
   -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 -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar securityconfig \
   pwdfile secret -file store.pwd -set -alias keystore
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to 1. Setting `MALLOC_ARENA_MAX` to 1 ensures that the memory usage is restricted to the specified heap size.

```
java -Xmx64m -Xms64m \
-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`

Copyright © 2011, 2026, Oracle and/or its affiliates.

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
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
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 SSL Keys and Certificates

If the certificate that the server uses is going to expire, or is no longer valid, you may need to replace the SSL key and certificate. This section describes the procedure to complete this task.

These directions describe creating a self-signed certificate, and an associated key, which is the default for Oracle NoSQL Database. Alternatively, you can use an external certificate, as described in Guidelines for Configuring External Certificates for an Existing Default Secure Installation.

You can update SSL Keys and certificates using one of the following two methods:

- Use the `plan update-tls-credentials` command.
- Use the manual process in each of the SNs to create a new SSL key and certificate, merge the truststore entries, copy the `store.keys` file to the security directory, and stop and start each SN sequentially as part of a rolling restart, if needed. Restarting the SNs will now only be needed if the data store has not yet been fully upgraded to Oracle NoSQL Database 24.4 or later.

Use the `plan` command

The `plan update-tls-credentials` command retrieves and installs the credential updates to the set of shared TLS (Transport Layer Security, earlier known as SSL) credentials used by Storage Node Agents (SNA) in the data store. You should use this plan only with data stores where all SNAs share the same credentials, and not for data stores with host-specific credentials.

```
plan update-tls-credentials [-retrieve-only|-install-only] [-force]
```

The plan command can be used in multiple ways depending on your requirements.

Copyright © 2011, 2026, Oracle and/or its affiliates.

- You can manually copy the new credentials and use the `plan update-tls-credentials` command with the `-install-only` flag. This is the best choice which you can use by default.
- If you need to automate the entire process of updating SSL credentials and keys, you could use the `plan update-tls-credentials` command without specifying any options or flags. Then the plan command retrieves the credentials and installs them.
- You could use the `plan update-tls-credentials` command with the `-retrieve-only` flag if you just want to retrieve the credentials and plan to install it later.

See plan update-tls-credentials for more details on how the SSL Keys and certificates are updated automatically using this plan command.

Manual process of updating SSL Keys and certificates

Updating the SSL key/certificate involves several steps:

1. Create a new key/certificate pair on a storage node.
2. Copy the new key/certificate pair to every storage node and merge the new certificate into the existing trust store files: `client.trust` and `store.trust`.
3. Restart each storage node sequentially, if needed. You will need to restart the SNs only if the data store has not yet been fully upgraded to Oracle NoSQL Database 24.4 or later. Once all SNs have been upgraded, they will notice the key and certificate changes without needing a restart.
4. Copy the `client.trust` with the merged entries to each of the clients.
5. Copy the `store.keys` that has the merged entries to each of the storage nodes, and restart each storage node sequentially, a second time. You need this step only for SNs running Oracle NoSQL Database 24.3 or earlier.
6. Remove the old certificate in `store.trust` in all the storage nodes.
7. Verify that only the new certificate is in use.

Complete these steps to update the SSL keys and certificates on a running store. Oracle NoSQL Database can remain operational throughout the entire process.

> **Note**
>
> The Oracle NoSQL Database environment used below is deployed on 3 Storage Nodes with capacity=3 and Replication Factor (RF)=3.

For more information on security configuration files, see Security Configuration.

Create a New SSL Key Certificate

1. In the first Storage Node (SN1), create a temporary directory to store the key.

   ```
   cd /users/user_name/tmp/kvroot/
   mkdir newKey
   ```

   On SN1 run the `securityconfig` utility to create a new key/certificate pair in the new directory, `newKey`. The new configuration needs to specify the same keystore password as

Copyright © 2011, 2026, Oracle and/or its affiliates.

your current configuration. If you do not specify a password with the `-kspwd` option, the utility prompts you to set a password.

```
java -jar $KVHOME/lib/kvstore.jar securityconfig \
config create -root /users/user_name/tmp/kvroot/newKey  -kspwd 123456


cd /users/user_name/tmp/kvroot/newKey
ls -R security
```

Output:

```
./security:client.security
security.xml store.trust temp.cert client.trust store.keys store.wallet
./security/store.wallet:
cwallet.sso
```

2. On SN1, merge the truststore entries using the config merge-trust command, as follows. After running this command, the `client.trust` and `store.trust` files will have two SSL certificate entries.

```
java -jar $KVHOME/lib/kvstore.jar securityconfig \
config merge-trust  -root $KVROOT -source-root \
/users/user_name/tmp/kvroot/newKey
```

Use `keytool -list` to list the entries in the keystore.

```
cd $KVROOT/security
keytool -list -keystore store.trust
Enter keystore password: *********
```

Output:

```
Keystore type:
JKS Keystore provider: SUN
Your keystore contains 2 entries
mykey_2, Feb 6, 2024,trustedCertEntry,
Certificate fingerprint (SHA1):
A3:75:F2:97:25:20:F9:AD:52:61:71:8F:6B:7E:B1:BB:E8:54:D1:7A
mykey, Feb 6, 2024,trustedCertEntry,
Certificate fingerprint SHA1):
89:71:8C:F1:6D:7E:25:D7:AD:C4:7E:23:8C:09:0D:AC:CE:AE:3F:67
```

> **Note**
>
> The `client.trust` file also contains the same two entries as `store.trust` file shown above.

In a multiple Storage Node deployment, you must copy the new configuration (the security directory and its contents) to each Storage Node host's new configuration directory and run `merge-trust` as described on each host.

Copyright © 2011, 2026, Oracle and/or its affiliates.

In SN2 and SN3, do the following:

```
cd /users/user_name/tmp/kvroot/
mkdir newKey
```

Copy the contents of the directory `newKey` from SN1 to the `/users/user_name/tmp/kvroot/newKey` directory in SN2 and SN3.

In SN2 and SN3, merge the truststore entries using the `config merge-trust` command as shown below.

```
java -jar $KVHOME/lib/kvstore.jar securityconfig  \
config merge-trust -root $KVROOT \
-source-root /users/user_name/tmp/kvroot/newKey
```

You can optionally list the entries in the keystore in SN2 and SN3 using `keytool` command (as shown above for SN1).

3. In an Oracle NoSQL Database running with a self-signed certificate, the client-side application will be able to connect to the data store with either the old or the new credentials once they switch to using the merged client truststore. It is a good practice to modify the clients at this point in the update procedure because that will allow the clients to connect to the data store now and throughout the update process. You need to copy the `client.trust` file with the merged entries (2 certificate entries) to each of the clients replacing the existing `client.trust` file used by the client applications. Once the update is complete, clients should switch to using the new truststore, so that the old certificates are cleared.

   ```
   scp <SN1_host>:$KVROOT/security/client.trust \
   <client_host>:<directory_client.trust>/client.trust
   ```

   > **Note**
   >
   > The `client.trust` is used to authenticate client-server communication, and `store.trust` to authenticate server-server communication.

4. Stop and start each Storage Node sequentially, making sure that each SN is completely up before restarting the next SN. Restarting one SN at a time is necessary because the data store must maintain the quorum during the restart of Storage Nodes while the SSL certificate is updated.
   After the restart, the Storage Nodes will load both the new and old certificates from the merged `store.trust`, so that the Storage Nodes can recognize both the new and old SSL credentials. The Storage Nodes that are not yet restarted will continue to use the old SSL credentials from `store.trust` file.

   In each of the Storage Nodes (SN1, SN2 and SN3), perform the following steps:

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


Copyright © 2011, 2026, Oracle and/or its affiliates.

Start the Admin CLI, and check that all Replication Nodes (RNs) are up using the `ping` command:

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

Logged in admin as anonymous

```
kv-> ping
```

> **Note**
>
> You need to restart the SNs only if the data store has not yet been fully upgraded to Oracle NoSQL Database 24.4 or later. Once all SNs have been upgraded, they will notice the key and certificate changes without needing a restart.

5. The `store.keys` file contains the generated private key. The `merge-trust` utility used above merges only the certificates in the `store.trust`, but does not merge the private keys. To make the NoSQL Database use the new SSL private key, the new `store.keys` needs to be copied to the security directory under $KVROOT in every Storage Node as shown below.
   In each of the Storage Nodes (SN1, SN2 and SN3), perform the following steps:

   Copy the `store.keys` file to the security directory, stop the Storage Node and start it again. Stop and start each storage node sequentially as a rolling restart, making sure that each SN is completely up before restarting the next SN. Restarting one SN at a time is necessary because the data store must maintain the quorum during the restart.

   ```
   cp /users/user_name/tmp/kvroot/newKey/security/store.keys $KVROOT/
   security/.
   java -jar $KVHOME/lib/kvstore.jar stop -root $KVROOT
   java -jar $KVHOME/lib/kvstore.jar start -root $KVROOT&
   ```

   > **Note**
   >
   > You need to restart the SNs only if the data store has not yet been fully upgraded to Oracle NoSQL Database 24.4 or later. Once all SNs have been upgraded, they will notice the key and certificate changes without needing a restart.

6. On each Storage Node, remove the obsolete certificate `mykey` in `store.trust`. Then, rename the new certificate `mykey_2` to `mykey`.
   In each of the Storage Nodes (SN1, SN2 and SN3), perform the following steps:

   Remove the old certificate named `mykey`.

   ```
   keytool -delete -keystore $KVROOT/security/store.trust -alias mykey
   Enter keystore password:
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

Rename the newly created certificate, `mykey_2`, to the previous key's name, `mykey`.

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

Only one key now exists, which is the newly generated one, called `mykey`. Verify that the new certificate is the only one listed using the following command:

```
keytool -list -keystore $KVROOT/security/store.trust
Enter keystore password:


Keystore type: JKS
Keystore provider: SUN
Your keystore contains 1 entry
mykey, Feb 6, 2024,trustedCertEntry,
Certificate fingerprint (SHA1):
A3:75:F2:97:25:20:F9:AD:52:61:71:8F:6B:7E:B1:BB:E8:54:D1:7A
```

The SSL keys and certificates have now been updated on the data store.

Additional verification while updating SSL Keys and Certificates

When you update SSL keys and certificates for a data store, these additional checks are performed. These checks are intended to detect potential problems with the new credentials.

- The passwords for the keystore and truststore files must be the same as the ones for the keystore and truststore currently in use by the data store.
- The types of the keystore and truststore files that you specified using the `keystoreType` and `truststoreType` parameters must match the exact types of the files.
- The keystore must have an entry for the alias that you specified using the `keystoreSigPrivateKeyAlias` parameter. The alias identifies the keypair used to create signatures. The certificate corresponding to this key pair must be validated successfully with the certificate in the truststore. In this case, the keystore and truststore need to contain the same certificate although, the truststore can contain more than one certificate (For example, if it is part of a certificate chain).
- The truststore must have an entry for the alias that you specified using the `truststoreSigPublicKeyAlias` parameter. This alias identifies the certificate used to verify signatures. The associated certificate must be validated successfully and match the certificate for the `keystoreSigPrivateKeyAlias` entry in the keystore.
- The keystore must have entries for all of the `serverKeyAlias` parameters that you specified for any transport type. This alias identifies the keypair used by the store services. If you have not specified anything, the alias `shared` is used. The certificate corresponding to this key pair must be validated successfully with the certificate in the truststore.

> **Note**
>
> The truststore can contain multiple certificates (For example, if it is part of a certificate chain). Additionally, the validation of the keystore certificate may also require multiple certificates in the truststore.

Copyright © 2011, 2026, Oracle and/or its affiliates.

- All `serverKeyAlias` entries must satisfy the verification requirements of the associated `clientIdentityAllowed` parameter, if any.
- For transports for which the parameter `clientAuthRequired` is true, the keystore must have entries for all of the `clientKeyAlias` parameters that you specified. The `clientkeyAlias` parameter identifies the keypair used by either a direct connect Java client or a proxy. If you have not specified anything, the alias `shared` is used. The certificate corresponding to this key pair must be validated successfully with the certificate in the truststore.
- All `clientKeyAliasentries` must satisfy the verification requirements of the associated `serverIdentityAllowed` parameter, if any.

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)*

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar makebootconfig \
-root $KVROOT -host $KVHOST -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)"
```

By default the keystore entry is stored under an alias "`shared`" and the truststore entry is stored under an alias "`mykey`". If you are using customized aliases for keystore and truststore, then additional flags need to be specified in the `makebootconfig` command.

For example if your customized keystore alias is "`currentKey`" and the certificate is stored in the truststore under the "`currentCert`" alias, the following additional parameters have to be included in the `makebootconfig` command.

```
-security-param "client:serverKeyAlias=currentKey"
-security-param "ha:serverKeyAlias=currentKey"
-security-param "internal:clientKeyAlias=currentKey"
-security-param "internal:serverKeyAlias=currentKey"
-security-param "keystoreSigPrivateKeyAlias=currentKey"
-security-param "truststoreSigPublicKeyAlias=currentCert"
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

The modified `makebootconfig` command with these additional flags is given below.

```
java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar makebootconfig \
-root $KVROOT -host $KVHOST -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)"
-security-param "client:serverKeyAlias=currentKey" \
-security-param "ha:serverKeyAlias=currentKey" \
-security-param "internal:clientKeyAlias=currentKey" \
-security-param "internal:serverKeyAlias=currentKey" \
-security-param "keystoreSigPrivateKeyAlias=currentKey" \
-security-param "truststoreSigPublicKeyAlias=currentCert"
```

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 older releases (prior 4.1), you needed to verify the configuration manually. In that case, 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:

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **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 configuration to each Storage Node and then restart all of the Storage Nodes.

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to 1. Setting `MALLOC_ARENA_MAX` to 1 ensures that the memory usage is restricted to the specified heap size.

```
copy -r $NEW_KVROOT/security $KVROOT
java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop -root $KVROOT
java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar start -root KVROOT&
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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 -Xmx64m -Xms64m \
-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
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
oracle.kv.ssl.protocols=TLSv1.2
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 -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop -root $KVROOT


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


java -Xmx64m -Xms64m -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

If the external certificate that the data store uses has expired, or is no longer valid, you may need to replace the SSL key and certificate. This section describes the procedure to complete this task.

You can update the external certificates using one of the following two methods:

- Use the `plan update-tls-credentials` command
- Use the manual process in each of the SNs to create a certificate, merge the truststore entries, copy the `store.keys` file to the security directory, and stop and start each Storage Node sequentially as part of a rolling restart, if needed. You need to restart the SNs only if the data store has not yet been fully upgraded to Oracle NoSQL Database 24.4 or later. Once all SNs have been upgraded, they will notice the key and certificate changes without needing a restart.

Use the `plan` command

The `plan update-tls-credentials` command retrieves and installs the credential updates to the set of shared TLS (Transport Layer Security, earlier known as SSL) credentials used by Storage Node Agents (SNA) in the data store. You should use this plan only with data stores

Copyright © 2011, 2026, Oracle and/or its affiliates.

where all SNAs share the same credentials, and not for data stores with host-specific credentials.

```
plan update-tls-credentials  [-retrieve-only|-install-only] [-force]
```

The plan command can be used in multiple ways depending on your requirements.

- You can manually copy the new credentials and use the `plan update-tls-credentials` command with the `-install-only` flag. This is the best choice which you can use by default.
- If you need to automate the entire process of updating SSL credentials and keys, you could use the plan `update-tls-credentials` command without specifying any options or flags. Then the plan command retrieves the credentials and installs them.
- You could use the `plan update-tls-credentials` command with the `-retrieve-only` flag if you just want to retrieve the credentials and plan to install it later.

See plan update-tls-credentials for more details on how the SSL Keys and certificates are updated automatically using this plan command.

Manual process of updating 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)"
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

> **Note**
>
> Before starting the SNA, set the environment variable `MALLOC_ARENA_MAX` to 1. Setting `MALLOC_ARENA_MAX` to 1 ensures that the memory usage is restricted to the specified heap size.

```
java -Xmx64m -Xms64m \
-jar $KVHOME/lib/kvstore.jar securityconfig \
config merge-trust -root $KVROOT \
-source-root $NEW_KVROOT

java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar stop -root $KVROOT
java -Xmx64m -Xms64m -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).
>
> You need to restart the SNs only if the data store has not yet been fully upgraded to Oracle NoSQL Database 24.4 or later. Once all SNs have been upgraded, they will notice the key and certificate changes without needing a restart.

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

> **Note**
>
> You need to restart the SNs only if the data store has not yet been fully upgraded to Oracle NoSQL Database 24.4 or later. Once all SNs have been upgraded, they will notice the key and certificate changes without needing a restart.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 data store 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 the data store 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).

Guidelines for Resetting Admin Password

From Oracle NoSQL Database 24.3 release, Admin Command Line Interface (Admin CLI) supports authentication with the data store SSL credentials.

When you invoke the Admin CLI, you need to provide the security directory in the Storage Node of the data store. Administrators who have access to this security directory can login to the Admin CLI to reset the password when the password has been forgotten.

> **Note**
>
> It is not recommended to run other CLI commands when you login to the Admin CLI with server SSL credentials.

Steps to reset admin password:

Copyright © 2011, 2026, Oracle and/or its affiliates.

1. Login with the store SSL credentials.

   ```
   java -Xmx64m -Xms64m \
   -jar $KVHOME/lib/kvstore.jar runadmin \
   -host $HOSTNAME -port 5000 -store mystore \
   -store-security-dir $KVROOT/security
   ```

   Logged in to Admin and store with store SSL credentials.

2. Reset the admin password using `ALTER USER` command.

   ```
   kv-> execute 'ALTER USER <user_name> IDENTIFIED BY "<new_password>"'
   ```

   Statement completed successfully.

Administrator should now be able to login with the new password.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

Parameter Name	Value Range and Type	Description
passwordHashIterations	5000 - 10000000 (integer)	The number of `PBKDF2-HMAC-SHA256` iterations used when a local password hash is generated. The default value is 600000. Existing password hashes continue to use the iteration count stored with the digest.

Note

You should not need to change `passwordHashIterations` unless local password authentication latency or CPU usage is measured to be unacceptable in your environment. Lowering this value reduces the pass

Copyright © 2011, 2026, Oracle and/or its affiliates.

Parameter Name	Value Range and Type	Description
		word-hash work factor and should be treated as a security trade-off. Very high values can make local password authentication, user creation, and password changes noticeably slower. To apply the current policy value to existing local password hashes, use the ALTER

Copyright © 2011, 2026, Oracle and/or its affiliates.

Parameter Name	Value Range and Type	Description
		USER command to update the corresponding user passwords so that the hashes are recreated.

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
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
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 -Xmx64m -Xms64m \
-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
...
```

> **Note**
>
> After the password is reset, if you're using Oracle Wallet for external password storage, you must recreate the wallet files for all your Oracle NoSQL Database user accounts. See Oracle Wallet.

Copyright © 2011, 2026, Oracle and/or its affiliates.

B
SSL keystore generation

The keystores (store.keys and store.trust) that are automatically generated by `makebootconfig` or `securityconfig` is using a RSA private key with size of 2048 and the associated certificate that has 365 days lifetime. They can also be manually created to have different key algorithm, size, validity or other characteristics, 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`), they must be the same as the password specified in the (`-kspwd`) when the security configuration directory is created via `makebootconfig` or `securityconfig`.

Copyright © 2011, 2026, Oracle and/or its affiliates.

- 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.

After creating the keystores (store.keys and store.trust) with above commands, replace the old ones in the security configuration directory created by `makebootconfig` or `securityconfig` utility.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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 in store.keys

```
keytool -genkeypair -keystore store.keys \
-alias shared -keyAlg RSA -keysize 2048 \
-validity 365  -dname \
"CN=my-nosql-cluster.example.com, \
OU=My Company, O=IT, L=San Francisco, ST=CA, C=US" \
-storepass <passwd> -keypass <passwd>

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.

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

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

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
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
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.

   The client.trust file is a truststore used by the client applications to validate the server's certificate. You can create it either in a password-less PKCS12 format or password protected format, depending on your setup. By default, when using the NoSQL `securityconfig` utility, the truststore is created in PKCS12 format without a password. Note the default behavior for standard Java truststore requires a password.

   a. **Password-less PKCS12 Truststore**
      Starting with Oracle NoSQL Database release 22.1, the `securityconfig` utility creates a password-less `client.trust`, if the Java version supports it.

      The supported Java versions are:

      - JDK 8u301 or higher for Java 8
      - JDK 11.0.12 or higher for Java 11

Copyright © 2011, 2026, Oracle and/or its affiliates.

- Any JDK 17 or higher

```
mkdir tmp_root

java -jar $KVHOME/lib/kvstore.jar securityconfig config create -root
tmp_root -kspwd <store-password>
```

This generates `tmp_root/security/client.trust` without a password.

Import your CA root and intermidiate certificates:

```
keytool -importcert -keystore client.trust -storetype PKCS12 -file
ca.cert.pem -alias root -noprompt

keytool -importcert -keystore client.trust -storetype PKCS12 -file
intermediate.cert.pem -alias intermediate -noprompt
```

Remove the unused default alias:

```
keytool -delete -keystore tmp_root/security/client.trust -alias mykey
```

> **Note**
>
> You may receive an error if you attempt to create a password-less PKCS12 truststore with an unsupported Java version. In this case, either use a password-protected truststore or create a JKS truststore, as below, instead:

```
keytool -importcert -keystore client.trust -storetype JKS -storepass
<password> -file ca.cert.pem -alias root -noprompt

keytool -importcert -keystore client.trust -storetype JKS -storepass
<password> -file intermediate.cert.pem -alias intermediate -noprompt
```

b. Create password-protected truststore

```
keytool -importcert -keystore client.trust -storetype PKCS12 -storepass
<password> -file ca.cert.pem -alias root -noprompt

keytool -importcert -keystore client.trust -storetype PKCS12 -storepass
<password> -file intermediate.cert.pem -alias intermediate -noprompt
```

The client application should set the following the login properties according to the Default Security Configuration.

```
oracle.kv.ssl.trustStorePassword=<password>

oracle.kv.ssl.trustStorePasswordAlias=<aliasName>
```

For more information, see Guidelines for using PKCS12 Java KeyStore.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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):

Copyright © 2011, 2026, Oracle and/or its affiliates.

- plan remove-index –table table_name

SYSVIEW:

- 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 tables

USRVIEW:

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

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

Copyright © 2011, 2026, Oracle and/or its affiliates.

- 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

Copyright © 2011, 2026, Oracle and/or its affiliates.

- 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

Copyright © 2011, 2026, Oracle and/or its affiliates.

- 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

Copyright © 2011, 2026, Oracle and/or its affiliates.

- 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

Privileges for Running XRegion Service

Privilege(s) required:

- WRITE_SYSTEM_TABLE
- READ_ANY_TABLE
- INSERT_ANY_TABLE

Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

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
   ```


Copyright © 2011, 2026, Oracle and/or its affiliates.

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.

Copyright © 2011, 2026, Oracle and/or its affiliates.

G

Generating Certificate and Private Key for the Oracle NoSQL Database Proxy

Topics

- Guidelines for Generating Self-Signed Certificate and Private Key using OpenSSL
- Guidelines for Generating Certificate Chain and Private Key using OpenSSL
- Troubleshooting issues with self-signed certificate

Guidelines for Generating Self-Signed Certificate and Private Key using OpenSSL

Self-signed certificates can be used to securely connect to the Oracle NoSQL Database Proxy. This section provides the steps to generate the self-signed certificate and other required files for a secure connection using OpenSSL.

As a pre-requisite, download and install OpenSSL on the host machine. See OpenSSL.

> **Note**
>
> In the examples below, the OpenSSL command has been used in Oracle Linux Version 8 (OL8). The syntax of `openssl` can be different in other Oracle Linux versions.

Before generating your certificate, list all the different hostnames, domains, sub-domains, and IP addresses that need to be secured. Understanding how the application connects to the Oracle NoSQL Database Proxy, allows you to understand the needs of SSL/TLS certificates. First you need to determine how many hostnames, domains and sub-domains need to be secured. This will help in determining the right SSL/TLS certificate or a certificate mix that is needed to encrypt the traffic between the applications and the Oracle NoSQL Database Proxy.

To generate a self-signed certificate and private key using the OpenSSL, complete the following steps:

1. On the configuration host, navigate to the directory where the certificate file is required to be placed.
2. Use one of the following OpenSSL command to generate the self-signed certificate and private key. When prompted, provide a secure password of your choice for the certificate file.

Copyright © 2011, 2026, Oracle and/or its affiliates.

> **Note**
>
> All prompt password will use `123456` in this example.

```
openssl req -x509 -days 365 -newkey rsa:4096 -keyout key.pem -out
certificate.pem
 -subj "/C=US/ST=CA/L=San/CN=*.example.com/
emailAddress=localhost@oracle.com"
or
openssl req -x509 -days 365 -newkey rsa:4096 -keyout key.pem -out
certificate.pem
 -subj "/C=US/ST=CA/L=San/CN=proxy-nosql.example.com/
emailAddress=localhost@oracle.com"
```

where, `CN` in the `subj` should map to either the NoSQL Database proxy server hostname or the NoSQL Database proxy domain name.

Using CN and optionally SAN while generating a self-signed certificate:

- Common Name (CN) is used to specify the NoSQL Database proxy server hostname or NoSQL Database proxy server domain name.
- When a client tries to connect to the Oracle NoSQL Database Proxy it will get the SSL certificate and compare the NoSQL Database proxy server hostname or NoSQL Database proxy server domain name it wants to connect, with the CN provided in the SSL certificate. If they are exactly the same it will use the SSL certificate to encrypt the connection, otherwise, the connection fails.
- The standard X509 defines that single SSL certificate can only use a single CN. This means an SSL certificate can be used only for a single NoSQL Database proxy server hostname or NoSQL Database proxy server domain name.
- To solve this limitation, Subject Alternative Name(SAN) is created. SAN is used to define multi-name or many CNs in SSL certificates.
- SAN is shown as a separate attribute in SSL Certificates. Here is an example of a SAN.

```
openssl req -x509 -days 365 -newkey rsa:4096 -sha256   \
  -keyout key.pem  -out certificate.pem -extensions san -config \
  <(echo "[req]";
    echo distinguished_name=req;
    echo "[san]";
    echo subjectAltName=DNS:proxy-nosql,IP:10.0.0.9
    ) \
  -subj "/C=US/ST=CA/L=San/CN=proxy-nosql.example.com/
emailAddress=localhost@example.com"
```

3. Convert the private key to `PKCS#8` format. When prompted, provide a secure password of your choice for the encryption.

```
openssl pkcs8 -topk8 \
-inform PEM -outform PEM \
-in key.pem -out key-pkcs8.pem
```

The following files are generated in the directory:

Copyright © 2011, 2026, Oracle and/or its affiliates.

- `key.pem` is the private key.
- `key-pkcs8.pem` is the private key in `PKCS#8` format.
- `certificate.pem` is the SSL certificate file in pem format.

> **Note**
>
> The below conversion should be done if your key is encrypted with the `PKCS#5 v2.0` algorithm. Otherwise, you might encounter `IllegalArgumentException` exception that indicates the file does not contain a valid private key due to the unsupported algorithm. The encryption algorithm can be converted via OpenSSL `pkcs8` utility by specifying `PKCS#5 v1.5` or `PKCS#12` algorithms with `-v1` flag. The following command converts the encryption algorithm of a key to `PBE-SHA1-3DES`.
>
> ```
> openssl pkcs8 -topk8 -in <PKCS#5v2.0_key_file> -out <new_key_file> -v1
> PBE-SHA1-3DES
> ```

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

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

Guidelines for Generating Certificate Chain and Private Key using OpenSSL

Certificate chains can be used to securely connect to the Oracle NoSQL Database Proxy. This section provides the steps to generate certificate chains and other required files for a secure connection using OpenSSL.

A certificate chain is provided by a Certificate Authority (CA). There are many CAs. Each CA has a different registration process to generate a certificate chain. Follow the steps provided by your CA for the process to obtain a certificate chain from them.

As a pre-requisite, download and install OpenSSL on the host machine. See OpenSSL.

> **Note**
>
> In the examples below, the OpenSSL command has been used in Oracle Linux Version 8 (OL8). The syntax of `openssl` can be different in other Oracle Linux versions.

Before generating your certificate, list all the different hostnames, domains, sub-domains, and IP addresses that need to be secured. Understanding how the application connects to the Oracle NoSQL Database Proxy, allows you to understand the needs of SSL/TLS certificates. First you need to determine how many hostnames, domains and sub-domains need to be secured. This will help in determining the right SSL/TLS certificate or a certificate mix that is needed to encrypt the traffic between the applications and the Oracle NoSQL Database Proxy.

Copyright © 2011, 2026, Oracle and/or its affiliates.

To generate a certificate chain and private key using the OpenSSL, complete the following steps:

1. On the configuration host, navigate to the directory where the certificate file is required to be placed.
2. Create a 2048 bit server private key.

   ```
   openssl genrsa -out key.pem 2048
   ```

   The following output is displayed.

   ```
   Generating RSA private key, 2048 bit long modulus
   ..................+++
   ..................+++
   e is 65537 (0x10001)
   ```

3. This step is required only when your server private key is not in PKCS#8 format. Convert the private key to PKCS#8 format. When prompted, provide a secure password of your choice for the encryption.

   ```
   openssl pkcs8 -topk8 \
   -inform PEM -outform PEM \
   -in key.pem -out key-pkcs8.pem
   ```

   The following output is displayed.

   ```
   Enter Encryption Password:
   Verifying - Enter Encryption Password:
   ```

   > **Note**
   >
   > The below conversion should be done if your key is encrypted with the `PKCS#5 v2.0` algorithm. Otherwise, you might encounter `IllegalArgumentException` exception that indicates the file does not contain a valid private key due to the unsupported algorithm. The encryption algorithm can be converted via OpenSSL `pkcs8` utility by specifying `PKCS#5 v1.5` or `PKCS#12` algorithms with `-v1` flag. The following command converts the encryption algorithm of a key to `PBE-SHA1-3DES`.
   >
   > ```
   > openssl pkcs8 -topk8 -in <PKCS#5v2.0_key_file> -out <new_key_file> -
   > v1 PBE-SHA1-3DES
   > ```

4. Create a Certificate Signing Request (CSR).

   ```
   openssl req -new  -key key.pem -out request.csr \
   -subj "/C=US/ST=CA/L=San/CN=proxy-nosql.example.com"
   ```

   where, `CN` in the `subj` should map to either the NoSQL Database proxy server hostname or the NoSQL Database proxy domain name.
   Using CN and optionally SAN while generating a self-signed certificate:

Copyright © 2011, 2026, Oracle and/or its affiliates.

- Common Name (CN) is used to specify the NoSQL Database proxy server hostname or NoSQL Database proxy server domain name.
- When a client tries to connect to the Oracle NoSQL Database Proxy it will get the SSL certificate and compare the NoSQL Database proxy server hostname or NoSQL Database proxy server domain name it wants to connect, with the CN provided in the SSL certificate. If they are exactly the same it will use the SSL certificate to encrypt the connection, otherwise, the connection fails.
- The standard X509 defines that single SSL certificate can only use a single CN. This means an SSL certificate can be used only for a single NoSQL Database proxy server hostname or NoSQL Database proxy server domain name.
- To solve this limitation, Subject Alternative Name(SAN) is created. SAN is used to define multi-name or many CNs in SSL certificates.
- SAN is shown as a separate attribute in SSL Certificates. Here is an example of a SAN.

```
openssl req  -new  \
-key key.pem  -out request.csr -config \
<(echo "[req]";
echo distinguished_name=req;
echo req_extensions=req_ext
echo "[req_ext]";
echo subjectAltName=@alt_names
echo "[alt_names]";
echo DNS.1=proxy-nosql.example.com
echo DNS.2=proxy-nosql
echo DNS.3=localhost
echo IP.1=10.0.0.9
)  \
-subj "/C=US/ST=CA/L=San/CN=proxy-nosql.example.com"
```

5. Send Certificate Signing Request (CSR) data file to CA. CA will use CSR data to issue a SSL certificate.
6. CA returns a signed certificate `certificate.pem`. If it is not yet chained up with CA's certificate (`ca-chain.cert.pem`), you need to manually chain up.

```
cat intermediateCA.crt > ca-chain.cert.pem
cat rootCA.crt > ca-chain.cert.pem
cat ca-chain.cert.pem >> certificate.pem
```

The following files are generated in the directory:

- `key.pem` is the server private key.
- `key-pkcs8.pem` is the server private key in PKCS#8 format.
- `certificate.pem` is the certificate chain file in pem format. It includes the server certificates issued by CA.
- `request.csr` is the server certificate request file.
- `rootCA.crt` is the root certificate provided by the CA.
- `intermediateCA.crt` is the intermediate certificate(s) provided by CA.

Additionally, a `driver.trust` file is also required if you are using the Java driver, and if the `rootCA.crt` is not listed in Java default trust store `JAVA_HOME/jre/lib/security/cacerts`.

Copyright © 2011, 2026, Oracle and/or its affiliates.

This `driver.trust` file is not required for other language drivers. To generate the `driver.trust` file, import the `rootCA.crt` certificate to the Java keystore. When prompted, provide the keystore password.

```
keytool -import -alias example -keystore driver.trust -file rootCA.crt
```

For the Python driver, if your selected CA is not trusted by default, you need to get the `rootCA.crt` or the entire chain of certificates from CA and set the system environment variable accordingly:

Example:

```
REQUESTS_CA_BUNDLE=PATH_OF_CA_FILE/rootCA.crt
```

Troubleshooting issues with self-signed certificate

After you create a self-signed certificate and try to connect to the Oracle NoSQL Database Proxy, sometimes the connection fails.

Table G-1 Connection failure scenarios and fixes

Value of CN	Client Connection URL	Result of the connection
CN=localhost	https://localhost:8089	Connection successful
CN=proxy-nosql.example.com	https://proxy-nosql.example.com:8089	Connection successful
CN=*.example.com	https://proxy-nosql.example.com:8089	Connection successful
CN=proxy-nosql	https://proxy-nosql:808	Connection successful
CN=proxy-nosql.example.com	https://proxy-nosql:8089	Connection fails
CN=proxy-nosql.example.com	https://10.0.0.9:8089	Connection fails
CN=localhost	https://10.0.0.9:8089	Connection fails
CN=localhost	https://proxy-nosql:8089	Connection fails
CN=localhost	https://proxy-nosql.example.com:8089	Connection fails
CN=proxy-nosql	https://proxy-nosql.example.com:8089	Connection fails
CN=*.example.com	https://proxy-nosql.subdomain.example.com:8089	Connection fails

General Guidelines in avoiding connection failure errors:

- When a client tries to connect to the Oracle NoSQL Database Proxy it fetches the SSL certificate and compares the NoSQL Database proxy server hostname or NoSQL Database proxy server domain name. it wants to connect with the Common Name (CN) provided in the SSL certificate. If they are exactly the same it will use the SSL certificate to encrypt the connection, otherwise, the connection fails.
- When the connections fails, determine if you need to change the CN or add a SAN and regenerate the certificate.

Copyright © 2011, 2026, Oracle and/or its affiliates.

Example:

```
 CN=proxy-nosql.example.com
subjectAltName=DNS:proxy-nosql,DNS:localhost,IP:10.0.0.9,DNS:proxy-
alias.example.com
```

Sample error scenarios with solution:

Example 1 : The value of `CN=proxy-nosql.example.com` and the application is connecting using `https://proxy-nosql:8089` and the connection fails.

The reason for connection failure is that the hostnames are not the same. You can implement one of the following solutions:

- The client connection URL can be modified to `https://proxy-nosql.example.com:8089`
- You can change the CN in the certificate as `CN=proxy-nosql`
- You can add proxy-nosql as SAN `subjectAltName=DNS:proxy-nosql`

Example 2 : The value of `CN=*.example.com` and the application is connecting using`https://proxy-nosql.subdomain.example.com:8089` and the connection fails.

The reason for connection failure is that domain names are not the same. You can implement one of the following solutions:

- You can change the CN in the certificate as `CN=*.subdomain.example.com`
- You can add subdomain.example.com SAN `subjectAltName=DNS:*.subdomain.example.com`

Example 3: The value of `CN=proxy-nosql.example.com` and the application is connecting using `https://localhost:8089` and the connection fails.

The reason for connection failure is that the hostnames are not the same. You can implement one of the following solutions:

- The client connection URL can be modified to `https://proxy-nosql.example.com:8089`
- You can change the CN in the certificate as `CN=localhost`
- You can add localhost as SAN `subjectAltName=DNS:localhost`

Example 4: The vale of `CN=proxy-nosql.example.com` and the application is connecting using `https://10.0.0.9:8089` and the connection fails

The reason for connection failure is that you are trying to use an IP to do the connection. You can implement one of the following solutions:

- The client connection URL can be modified to `https://proxy-nosql.example.com:8089`
- You can add the IP as SAN `subjectAltName=IP:10.0.0.9`

Simple commands to test if the connection would be successful :

You can use simple `curl` commands to simulate the connection and validate your configuration.

Use case 1: View the certificate only CN:

```
$ openssl x509 -text -noout -in certificate.pem | grep CN
Issuer: C=US, ST=CA, L=San,CN=proxy-nosql/emailAddress=localhost@oraclevcn.com
```


Copyright © 2011, 2026, Oracle and/or its affiliates.

```
Subject: C=US, ST=CA, L=San, CN=proxy-nosql/
emailAddress=localhost@oraclevcn.com
```

Fail test using curl:

```
$ curl --cacert certificate.pem https://node1-nosql.example.com:8087
curl: (51) Unable to communicate securely with peer: requested domain name
does
not match the server's certificate.
```

Reason: The address in the url node1-nosql.example.com doesn't match with CN=proxy-nosql in the certificate. And there is no SAN.

Success test using curl:

```
$ curl --cacert certificate.pem https://proxy-nosql:8087
```

Reason: The address in the url matches with CN=proxynosql in the certificate

Use case 2: Viewing the certificate CN and SAN

```
$ openssl x509 -text -noout -in
certificate.pem | grep CN
Issuer: CN=*.example.com
Subject: CN=*.example.com
```

```
$ openssl x509 -text -noout -in certificate.pem | grep -e DNS -e IP
  DNS:proxy-nosql,  DNS:*.subdomain.example.com, IP Address:10.0.0.9
```

Success test using curl:

```
$ curl --cacert certificate.pem https://proxy-nosql.subdomain.example.com:8087
```

Reason: Even if the address in the url proxy-nosql.subdomain.example.com doesn't match with CN=*.example.com in the certificate, it matches with the SAN in the certificate `subjectAltName=DNS:*.subdomain.example.com`.

```
$ curl --cacert certificate.pem https://proxy-nosql:8087
```

Reason: Even if the address in the url proxy-nosql doesn't match with CN=*.example.com in the certificate, it matches with the SAN in the certificate `subjectAltName=DNS:proxy-nosql`.

Copyright © 2011, 2026, Oracle and/or its affiliates.