

Oracle

*NoSQL Database
Security Guide*

12c Release 1
Library Version DRAFT



Legal Notice

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

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

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

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

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

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

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

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

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

Published 3/27/2014

Table of Contents

Preface	iv
Conventions Used in This Book	iv
1. Introducing Oracle NoSQL Database Security	1
2. Security Configuration	2
Security Configuration Overview	2
Configuring Security with Makebootconfig	4
Configuring Security with Securityconfig	5
Creating the security configuration	5
Adding the security configuration	6
Removing the security configuration	6
Merging truststore configuration	7
3. Performing a Secure Oracle NoSQL Database Installation	9
Adding Security to a New Installation	9
Adding Security to an Existing Installation	10
4. External Password Storage	13
Oracle Wallet	13
Password store file	13
5. Security.xml parameters	15
Top-level parameters	15
Transport parameters	16
6. Encryption	18
SSL model	18
SSL communication properties	19
7. Configuring Authentication	21
User management	21
User creation	21
User modification	22
User removal	23
User status	23
User login	23
Sessions	24
8. Security Policies	25
Security Policy Modifications	25
9. Keeping Oracle NoSQL Database Secure	27
Guidelines for Securing the Configuration	27
Guidelines for Deploying Secure Applications	27
Guidelines for Securing the SSL protocol	27
Guidelines for using JMX securely	28
Guidelines for Updating Keystore Passwords	28
Guidelines for Updating the SSL key/certificate	29
Guidelines for Operating System Security	30
A. SSL keystore generation	31
B. Third Party Licenses	33

Preface

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

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

Conventions Used in This Book

The following typographical conventions are used within this manual:

Information that you are to type literally is presented in monospaced font.

Variable or non-literal text is presented in *italics*. For example: "Go to your *KVHOME* directory."

Note

Finally, notes of special interest are represented using a note block such as this.

Chapter 1. Introducing Oracle NoSQL Database Security

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

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

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

- **Security Configuration Utility.** Allows you to configure and add security to a new or to an existing Oracle NoSQL Database installation.
- **Authentication methods.** Oracle NoSQL Database provides password authentication for users and systems.
- **Encryption.** You can encrypt data on the network to prevent unauthorized access to that data.
- **External Password Storage.** Oracle NoSQL Database provides two types of external password storage methods that you can manipulate (one type for CE deployments).
- **Security Policies.** Oracle NoSQL Database allows you to set up behaviors in order to ensure a secure environment.

In addition, [Keeping Oracle NoSQL Database Secure \(page 27\)](#) provides guidelines that you should follow when you secure your Oracle NoSQL Database installation.

Chapter 2. Security Configuration

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

Note

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

Security Configuration Overview

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

Note

The `makebootconfig` utility embeds the functionality of `securityconfig` tool.

This tool creates a set of security files based on the standard configuration. It is possible to perform the same tasks manually, and advanced security configuration might require manual setup, but using this tool helps to ensure a consistent setup. For more information on the manual setup, see [SSL keystore generation \(page 31\)](#).

Note

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

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

```
security/security.xml
security/store.keys
security/store.trust
security/store.passwd (CE or EE installations)
security/store.wallet (EE installations only)
security/store.wallet/cwallet.sso (EE installations only)
```

```
security/client.security
security/client.trust
```

where:

- `security.xml`

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

- `store.keys`

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

- `store.trust`

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

- `store.passwd` (CE or EE installations)

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

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

- `store.wallet` (EE installations only)

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

- `cwallet.sso` (EE installations only)

The wallet password storage file.

- `client.security`

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

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

- `client.trust`

A truststore file used by clients is generated.

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

Note

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

Configuring Security with Makebootconfig

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

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

where -store-security can have the following options:

- -store-security none

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

- -store-security configure

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

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

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

- -store-security enable

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

For more information on configuring security with makebootconfig, see [Adding Security to a New Installation \(page 9\)](#).

Configuring Security with Securityconfig

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

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

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

Creating the security configuration

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

```
config create
-root <secroot> [ -secdir <security dir> ]
[ -pwmgr { pwdfile | wallet } ]
[ -param <param=value> ]
```

where:

- -root <secroot>

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

- -secdir <security dir>

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

- -pwmgr [pwdfile | wallet]

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

where -pwmgr can have the following options:

- -pwmgr pwdfile

Indicates that the password store is a read-protected clear-text password file. This is the only available option for Oracle NoSQL Database CE deployments. You can specify an alternate implementation. For more information on pwdfile manipulation, see [Password store file \(page 13\)](#)

- -pwmgr wallet

Specifies Oracle Wallet as the password storage mechanism. This option is only available in the Oracle NoSQL Database EE version. For more information on Oracle wallet manipulation, see [Oracle Wallet \(page 13\)](#)

- `-param <param=value>`

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

For more information on configuring security with securityconfig, see [Adding Security to an Existing Installation \(page 10\)](#).

Adding the security configuration

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

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

Note

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

where:

- `-root <kvroot>`

A KVStore root directory must be provided as an argument.

- `-secdir <security dir>`

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

- `-config <config.xml>`

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

Removing the security configuration

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

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

Note

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

where:

- `-root <kvroot>`

A KVStore root directory must be provided as an argument.

- `-config <config.xml>`

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

Merging truststore configuration

If you want to merge truststore entries from one security configuration into another security configuration use the `config merge-trust` command. This command is helpful when performing security maintenance, particularly when you need to update the SSL key/certificate. For more information, see [Guidelines for Updating the SSL key/certificate \(page 29\)](#)

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

Note

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

where:

- `-root <secroot>`

Specifies the directory that contains the security configuration that will be updated. It is not required that this directory be a full KVRTXOT, 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".

Chapter 3. Performing a Secure Oracle NoSQL Database Installation

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

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

Adding Security to a New Installation

To install Oracle NoSQL Database securely:

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

```
java -jar KVHOME/lib/kvstore.jar makebootconfig \  
-root KVRROOT -port 5000 \  
-admin 5001 -host node01 -horange 5010,5020 \  
-store-security configure -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:

```
Creating SSL keystore  
Enter a password for your keystore:
```

3. Enter a password for your store and then reenter it for confirmation. For example, in a Community Edition (CE) installation, the `securityconfig` tool will automatically generate the following security related files:

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

4. Start the Storage Node Agent (SNA):

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

When a newly created store with a secure configuration is first started, there are no user definitions available against which to authenticate access. In order to reduce risk of unauthorized access, an admin will only allow you to connect to it from the host on which it is running. This security measure is not a complete safeguard against unauthorized access. It is important that you do not provide local access to machines running KVStore. In addition, you should perform steps 5, 6 and 7 soon after this step in order to minimize

the time period in which the admin might be accessible without full authentication. For more information on maintaining a secure store see [Guidelines for Securing the Configuration \(page 27\)](#).

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

```
java -jar KVHOME/lib/kvstore.jar \  
runadmin -port 5000 -host node01 \  
-store-security KVR00T/security/client.security
```

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

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

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

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

For more information on user creation and administration, see [User management \(page 21\)](#).

8. At this point, it is possible to connect to the store as the root user. To login, you can use either the -username <user> or the -security <path to security file> runadmin argument.

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

```
java -jar KVHOME/lib/kvstore.jar runadmin -security mylogin.txt
```

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=login.pwd  
oracle.kv.transport=ssl  
oracle.kv.ssl.trustStore=client.trust  
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1  
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

For more information, see [User login \(page 23\)](#).

Adding Security to an Existing Installation

To add security to an existing Oracle NoSQL Database installation:

1. Shut down the KVStore instance:

```
java -jar KVHOME/lib/kvstore.jar stop \  
-root KVRROOT
```

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

```
java -jar /home/kvuser/kvstore/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 access to the stores. In this case, Oracle Wallet is used:

```
config create -pwdmgr wallet -root KVRROOT  
Creating SSL keystore  
Enter a password for your keystore:
```

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

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

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

```
config add-security \  
-root KVRROOT -secdir security \  
-config config.xml
```

6. Start the Storage Node Agent (SNA):

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

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

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

This command sets SSL as a connection method and names a copy of the generated truststore file (client.security). For more information on SSL properties, see [SSL communication properties \(page 19\)](#).

8. Create a user. In this case, user root is defined:

```
kv-> plan create-user -name root -admin -wait
```



```
Enter the new password: *****
Re-enter the new password: *****
Executed plan 8, waiting for completion...
Plan 8 ended successfully
```

For more information on user creation and administration, see [User management \(page 21\)](#).

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>` or the `-security <path to security file>` `runadmin` argument described earlier in this section.

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

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

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

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

For more information, see [User login \(page 23\)](#).

Chapter 4. External Password Storage

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

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

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

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

Oracle Wallet

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

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

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

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

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

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

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

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

Special considerations should be taken if Oracle wallet is used to hold a user password and you are deploying your Oracle NoSQL Database. For more information, see [Guidelines for Deploying Secure Applications \(page 27\)](#).

Password store file

The following commands are used to create and manipulate CE password store files within the securityconfig tool. CE password store files managed through this interface are never

password protected. For more information on the securityconfig tool, see [Configuring Security with Securityconfig \(page 5\)](#).

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

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

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

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

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

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

Chapter 5. Security.xml parameters

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

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

Note

A subset of all the configuration options listed below related to SSL can be specified through Java system properties, security file properties, or through the KVStoreConfig API. For more information, see [SSL communication properties \(page 19\)](#).

Top-level parameters

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

- **internalAuth**
Specifies how internal systems authenticate. This parameter must be set to SSL.
- **keystore**
Identifies the keystore file within the security directory. This parameter is normally set to store.keys.
- **keystoreType**
Identifies the type of keystore that the keystore property references. If not set, the Java default keystore type is assumed.
- **securityEnabled**
To enable security this parameter must be set to true.
- **certMode**
Specifies the key/certificate management model in use. This must be set to "shared".
- **truststore**
Identifies the truststore file within the security directory. This is normally set to store.trust.
- **keystoreType**
Identifies the type of keystore that the truststore property references. If not set, the Java default keystore type is assumed.

- walletDir

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

- passwordFile

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

Transport parameters

There are three standard transport types:

- ha

Controls the communications between the data replication layer.

- client

Controls most RMI communication.

- internal

Controls the SSL internal authentication mechanism.

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

- transportType

This parameter should be set to SSL.

- serverKeyAlias

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

- clientKeyAlias

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

- clientAuthRequired

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

- clientIdentityAllowed

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

- serverIdentityAllowed

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

- `allowCipherSuites`

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

- `allowProtocols`

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

- `clientAllowCipherSuites`

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

- `clientAllowProtocols`

See `allowProtocols` for a description of the format. This parameter sets the protocol requirements only for the initiating side of a connection. If set, it overrides any setting of `allowProtocols` for the connection initiator.

Chapter 6. Encryption

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

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

Note

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

SSL model

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

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

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

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

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

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

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

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

```
oracle.kv.auth.username=root  
oracle.kv.auth.wallet.dir=login.wallet
```

```
oracle.kv.transport=ssl  
oracle.kv.ssl.trustStore=client.trust  
oracle.kv.ssl.protocols=TLSv1.2,TLSv1.1,TLSv1  
oracle.kv.ssl.hostnameVerifier=dnmatch(CN\=NoSQL)
```

Note

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

SSL communication properties

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

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

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

- `oracle.kv.transport=ssl`

Directs KVStore clients and utilities to connect to the KVStore RMI registry via SSL.

- `oracle.kv.ssl.trustStore=<path-to-ssl-truststore>`

Names a copy of the truststore file generated by `makebootconfig` or `securityconfig` to enable validation of the KVStore server SSL certificate.

Note

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

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

- `oracle.kv.ssl.ciphersuites`

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

- `oracle.kv.ssl.protocols`

Specifies a comma-separated list of SSL protocols that should be allowed in communication with the server.

- `oracle.kv.ssl.trustStoreType`

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

Note

Applications may also set these security properties through API methods on `KVStoreConfig`.

Chapter 7. Configuring Authentication

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

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

User management

Users can be created, modified or removed in the Oracle NoSQL Database through the admin CLI. Information about a specific user account as well as a summary listing of registered users can also be displayed. For more information, see the next sections describing each user management operation.

User creation

Once you connect to a deployed admin, users can be added. The first user created must have admin rights (permission to perform user creation/modification operations). Use the plan create-user command to create a new user:

```
plan create-user -name <user-name>
[ -password <password> ]
[ -admin ] [-disable] [ -wait ]
```

where:

- -name

Specifies the name of the new user to create. The user name must be non-zero in length and must be composed of characters in a restricted character set composed of letters, digits, and underscore.

- -password

Specifies the initial password for the new user. Users are encouraged to provide the password in response to a console prompt rather than specifying it on the command line. The use of this argument may be necessary for scripted configuration.

- -admin

Assigns the created user complete system access rights. The first user created must have admin rights and once that first user is created, there must always be at least one admin user defined and enabled.

- -disable

Creates the user in disabled state. The initial admin user may not be created in a disabled state. The default state for a newly created user is enabled.

Create an admin user using the plan `create-user` command with the `-admin` option. You are prompted to set the password if it is not provided as an argument:

```
kv-> plan create-user -name root -admin -wait
Enter the new password:
Re-enter the new password:
Executed plan 7, waiting for completion...
Plan 7 ended successfully
```

User modification

Use the plan `change-user` command to modify the specified user account (users without the `-admin` option may also change their own password):

```
plan change-user -name <user-name>
[ -disable | -enable ]
[ -set-password [ -password <password> ]
[ -retain-current-password ]
[ -clear-retained-password ] ]
```

where:

- `-name`

Specifies the name of the user to modify. The user name must be non-zero in length and must be composed of characters in a restricted character set composed of letters, digits, and underscore.

- `-disable`

Disables the specified account. Disabling an account prevents a user from logging into the account. It also will cause currently logged-in users to become logged out. The logout of users is not immediate, but may be delayed by the login cache timeout period, which is specified through the `loginCacheTimeout` parameter.

- `-enable`

Enables the specified account, assuming that it was previously disabled

- `-set-password`

Changes the password for the specified account (valid only for internal password authentication).

- `-password`

Specifies the new password for the new user. Users are encouraged to provide the password in response to a console prompt rather than specifying it on the command line. The use of this argument may be necessary for scripted configuration.

- `-retain-current-password`.

For use only in conjunction with `-password`. If specified, causes the current password defined for the user to be remembered as a valid alternate password for a limited duration, 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 via the CLI while an application is still running without affecting its operation.

- `-clear-retained-password`.

Erases the current alternate retained password.

Note

If `-set-password`, `-clear-retained-password` and `-retain-current-password` are all specified in the same command, the current retained password is erased before considering whether a password is currently retained.

User removal

Use the `plan drop-user` command to remove the specified user account (users cannot remove themselves):

```
plan drop-user -name <user-name>
```

User status

Use the `show user` command to display information about the specified user account:

```
show user -name <user-name>
```

Use the `show users` command to display a summary listing of registered users:

```
show users
```

User login

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

- `-username <user>`

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

- `-security <path-to-security-file>`

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

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

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

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

Note

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

Sessions

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

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

Note

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

If allowed, the Oracle NoSQL Database client will transparently attempt to extend session lifetime. For best results, your application should include logic to deal with reauthentication, as operational issues could prevent it from succeeding initially. In this way, you can avoid the use of extended logic in your application to reacquire a valid session state.

You can configure the behavior regarding session management to meet the needs of the application and environment. To do this, you can modify the following parameters using the `plan change-parameters` command: `sessionTimeout`, `sessionExtendAllowed` and `loginCacheTimeout`. For more information, see [Security Policy Modifications \(page 25\)](#)

Chapter 8. Security Policies

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

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

Note

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

Security Policy Modifications

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

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

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

The following security parameters can be set:

- `sessionTimeout=<Long TimeUnit>`

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

- `sessionExtendAllowed=<Boolean>`

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

- `accountErrorLockoutThresholdInterval=<Long TimeUnit>`

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

- `accountErrorLockoutThresholdCount=<Integer>`

Number of invalid login attempts for a user account from a particular host address over the tracking period needed to trigger an automatic account lockout for a host. The default value is 10 attempts.

- `accountErrorLockoutTimeout=<Long TimeUnit>`

Time duration for which an account will be locked out once a lockout has been triggered. The default value is 30 minutes.

- loginCacheTimeout=<Long TimeUnit>

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

Chapter 9. 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 configure Oracle NoSQL Database to be secure 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.
- SNs 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 SN hosts using a secure copy mechanism.
- After the first user is configured but before allowing applications to use the store, you may wish to restart all SNA processes on hosts running Admin processes and then use the Admin CLI `show users` command to ensure that there is only the single user definition that is expected. This step validates that no other user creation occurred during the period when administrative login was not required.

Guidelines for Deploying Secure Applications

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

- The `kvstore-ee.jar` file needs to be included in the application classpath.
- The `kvstore-ee.jar`, `oraclepki.jar`, `osdt_cert.jar`, `osdt_core.jar` files should all be made available on the application machine.

Note

`kvstore-ee` references the other files, so they do not need to be included in the classpath explicitly.

Guidelines for Securing the SSL protocol

Follow these guidelines to keep the SSL protocol secure:

- When configuring SSL communication for your store, you should consider both performance and security.
- For a more secure store you should opt for higher security where possible.
- The Oracle JDK 7 support TLSv1.2 as an SSL protocol level, whereas JDK 6 provides only TLSv1 as its highest protocol level.
- If you are currently using JDK 6, it is strongly recommended that you upgrade to JDK 7.

Guidelines for using JMX securely

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

- If you enable JMX for a secure store, your JMX monitoring application must access the store using SSL.
- You should consult the configuration details for the JMX product you wish to use. In this case, you can use jconsole with a secure store by running the following command:

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

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

Guidelines for Updating Keystore Passwords

Follow these steps to update the keystore passwords:

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

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

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

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

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

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

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

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

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

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

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

Guidelines for Updating the SSL key/certificate

Follow these steps to update the SSL key/certificate:

1. On the configuration host, run securityconfig to create a new configuration in a directory in parallel to the standard configuration directory.
2. On the configuration host, merge the truststore entries by using the config merge-trust command:

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

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

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

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

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

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

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

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

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

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

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

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

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

Guidelines for Operating System Security

Follow these guidelines regarding operating system security:

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

Note

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

Appendix A. SSL keystore generation

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

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

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

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

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

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

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

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

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

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

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

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

- The store password for store.trust should match the store password for store.keys.

Appendix B. Third Party Licenses

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