



Red Hat build of Keycloak 22.0

Server Guide

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide consists of information for administrators to configure Red Hat build of Keycloak 22.0.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	7
CHAPTER 1. CONFIGURING RED HAT BUILD OF KEYCLOAK	8
1.1. CONFIGURING SOURCES FOR RED HAT BUILD OF KEYCLOAK	8
1.1.1. Example: Configuring the db-url-host parameter	8
1.2. FORMATS FOR CONFIGURATION	8
1.2.1. Example - Alternative formats based on configuration source	9
1.2.2. Formats for command-line parameters	9
1.2.3. Formats for environment variables	9
1.2.4. Format to include a specific configuration file	10
1.2.5. Setting sensitive options using a Java KeyStore file	10
1.2.6. Format for raw Quarkus properties	10
1.3. STARTING RED HAT BUILD OF KEYCLOAK	11
1.3.1. Starting Red Hat build of Keycloak in development mode	11
1.3.2. Starting Red Hat build of Keycloak in production mode	11
1.4. CREATING THE INITIAL ADMIN USER	12
1.5. OPTIMIZE THE RED HAT BUILD OF KEYCLOAK STARTUP	12
1.5.1. Creating an optimized Red Hat build of Keycloak build	12
1.5.1.1. First step: Run a build explicitly	12
1.5.1.2. Second step: Start Red Hat build of Keycloak using --optimized	13
1.6. UNDERLYING CONCEPTS	14
CHAPTER 2. CONFIGURING RED HAT BUILD OF KEYCLOAK FOR PRODUCTION	15
2.1. TLS FOR SECURE COMMUNICATION	15
2.2. THE HOSTNAME FOR RED HAT BUILD OF KEYCLOAK	15
2.3. REVERSE PROXY IN A DISTRIBUTED ENVIRONMENT	15
2.4. PRODUCTION GRADE DATABASE	15
2.5. SUPPORT FOR RED HAT BUILD OF KEYCLOAK IN A CLUSTER	15
2.6. CONFIGURE RED HAT BUILD OF KEYCLOAK SERVER WITH IPV4 OR IPV6	16
CHAPTER 3. RUNNING RED HAT BUILD OF KEYCLOAK IN A CONTAINER	17
3.1. CREATING A CUSTOMIZED AND OPTIMIZED CONTAINER IMAGE	17
3.1.1. Writing your optimized Red Hat build of Keycloak Dockerfile	17
3.1.2. Installing additional RPM packages	18
3.1.3. Building the container image	19
3.1.4. Starting the optimized Red Hat build of Keycloak container image	19
3.2. EXPOSING THE CONTAINER TO A DIFFERENT PORT	19
3.3. TRYING RED HAT BUILD OF KEYCLOAK IN DEVELOPMENT MODE	20
3.4. RUNNING A STANDARD RED HAT BUILD OF KEYCLOAK CONTAINER	20
3.5. PROVIDE INITIAL ADMIN CREDENTIALS WHEN RUNNING IN A CONTAINER	21
3.6. IMPORTING A REALM ON STARTUP	21
3.7. RELEVANT OPTIONS	21
CHAPTER 4. CONFIGURING TLS	24
4.1. CONFIGURING TLS IN RED HAT BUILD OF KEYCLOAK	24
4.1.1. Providing certificates in PEM format	24
4.1.2. Providing a Java Keystore	24
4.1.2.1. Setting the Keystore password	24
4.2. CONFIGURING TLS PROTOCOLS	24
4.3. SWITCHING THE HTTPS PORT	25
4.4. USING A TRUSTSTORE	25
4.4.1. Setting the truststore password	25

4.5. SECURING CREDENTIALS	25
4.6. ENABLING MUTUAL TLS	25
4.7. RELEVANT OPTIONS	26
CHAPTER 5. CONFIGURING THE HOSTNAME	29
5.1. SERVER ENDPOINTS	29
5.1.1. Frontend	29
5.1.2. Backend	30
5.1.3. Administration Console	30
5.2. EXAMPLE SCENARIOS	31
5.2.1. Exposing the server behind a TLS termination proxy	31
5.2.2. Exposing the server without a proxy	31
5.2.3. Forcing backend endpoints to use the same URL the server is exposed	31
5.2.4. Exposing the server using a port other than the default ports	31
5.2.5. Exposing Red Hat build of Keycloak behind a TLS reencrypt proxy using different ports	31
5.3. TROUBLESHOOTING	32
5.4. RELEVANT OPTIONS	32
CHAPTER 6. USING A REVERSE PROXY	35
6.1. PROXY MODES	35
6.2. CONFIGURE THE PROXY MODE IN RED HAT BUILD OF KEYCLOAK	35
6.3. CONFIGURE THE REVERSE PROXY	35
6.4. DIFFERENT CONTEXT-PATH ON REVERSE PROXY	36
6.5. TRUST THE PROXY TO SET HOSTNAME	36
6.6. ENABLE STICKY SESSIONS	36
6.6.1. Exposing the administration console	37
6.6.2. Exposed path recommendations	37
6.6.3. Enabling client certificate lookup	38
6.6.3.1. Configuring the NGINX provider	39
6.7. RELEVANT OPTIONS	40
CHAPTER 7. CONFIGURING THE DATABASE	41
7.1. SUPPORTED DATABASES	41
7.2. INSTALLING A DATABASE DRIVER	41
7.2.1. Installing the Oracle Database driver	41
7.2.2. Installing the Microsoft SQL Server driver	42
7.3. CONFIGURING A DATABASE	43
7.4. OVERRIDING DEFAULT CONNECTION SETTINGS	43
7.5. OVERRIDING THE DEFAULT JDBC DRIVER	43
7.6. CONFIGURING UNICODE SUPPORT FOR THE DATABASE	44
7.6.1. Configuring Unicode support for an Oracle database	44
7.6.2. Unicode support for a Microsoft SQL Server database	45
7.6.3. Configuring Unicode support for a MySQL database	45
7.6.4. Configuring Unicode support for a PostgreSQL database	45
7.7. CHANGING DATABASE LOCKING TIMEOUT IN A CLUSTER CONFIGURATION	45
7.8. USING DATABASE VENDORS WITHOUT XA TRANSACTION SUPPORT	46
7.9. SETTING JPA PROVIDER CONFIGURATION OPTION FOR MIGRATIONSTRATEGY	46
7.10. RELEVANT OPTIONS	46
CHAPTER 8. CONFIGURING DISTRIBUTED CACHES	50
8.1. ENABLE DISTRIBUTED CACHING	50
8.2. CONFIGURING CACHES	50
8.2.1. Cache types and defaults	51
8.2.2. Configuring caches for availability	53

8.2.3. Specify your own cache configuration file	53
8.3. TRANSPORT STACKS	53
8.3.1. Available transport stacks	54
8.3.2. Additional transport stacks	54
8.3.3. Custom transport stacks	55
8.4. SECURING CACHE COMMUNICATION	55
8.5. EXPOSING METRICS FROM CACHES	55
8.6. RELEVANT OPTIONS	56
CHAPTER 9. CONFIGURING OUTGOING HTTP REQUESTS	57
9.1. CLIENT CONFIGURATION COMMAND	57
9.2. PROXY MAPPINGS FOR OUTGOING HTTP REQUESTS	58
9.3. PROXY MAPPINGS USING REGULAR EXPRESSIONS	58
9.4. CONFIGURING TRUSTED CERTIFICATES FOR TLS CONNECTIONS	59
CHAPTER 10. CONFIGURING TRUSTED CERTIFICATES FOR OUTGOING REQUESTS	60
10.1. CONFIGURING THE RED HAT BUILD OF KEYCLOAK TRUSTSTORE	60
10.1.1. Example of a truststore configuration	60
CHAPTER 11. ENABLING AND DISABLING FEATURES	62
11.1. ENABLING FEATURES	62
11.2. DISABLING FEATURES	62
11.3. SUPPORTED FEATURES	62
11.3.1. Disabled by default	63
11.4. PREVIEW FEATURES	63
11.5. DEPRECATED FEATURES	64
11.6. RELEVANT OPTIONS	64
CHAPTER 12. CONFIGURING PROVIDERS	66
12.1. CONFIGURATION OPTION FORMAT	66
12.2. SETTING A PROVIDER CONFIGURATION OPTION	66
12.3. CONFIGURING A DEFAULT PROVIDER	66
12.4. ENABLING AND DISABLING A PROVIDER	67
12.5. INSTALLING AND UNINSTALLING A PROVIDER	67
12.6. USING THIRD-PARTY DEPENDENCIES	67
12.7. REFERENCES	67
CHAPTER 13. CONFIGURING LOGGING	68
13.1. LOGGING CONFIGURATION	68
13.1.1. Log levels	68
13.1.2. Configuring the root log level	68
13.1.3. Configuring category-specific log levels	69
13.2. ENABLING LOG HANDLERS	69
13.3. CONSOLE LOG HANDLER	69
13.3.1. Configuring the console log format	69
13.3.2. Setting the logging format	71
13.3.3. Configuring JSON or plain console logging	71
13.3.4. Colors	72
13.4. FILE LOGGING	72
13.4.1. Enable file logging	72
13.4.2. Configuring the location and name of the log file	72
13.4.3. Configuring the file handler format	72
13.5. RELEVANT OPTIONS	73
CHAPTER 14. FIPS 140-2 SUPPORT	75

14.1. BOUNCYCASTLE LIBRARY	75
14.1.1. BouncyCastle FIPS bits	75
14.2. GENERATING KEYSTORE	75
14.2.1. PKCS12 keystore	75
14.2.2. BCFKS keystore	76
14.3. RUNNING THE SERVER.	76
14.4. STRICT MODE	77
14.4.1. Cryptography restrictions in strict mode	77
14.5. OTHER RESTRICTIONS	78
14.6. RUN THE CLI ON THE FIPS HOST	79
14.7. RED HAT BUILD OF KEYCLOAK SERVER IN FIPS MODE IN THE CONTAINER	79
14.8. MIGRATION FROM NON-FIPS ENVIRONMENT	80
14.9. RED HAT BUILD OF KEYCLOAK FIPS MODE ON THE NON-FIPS SYSTEM	80
CHAPTER 15. ENABLING RED HAT BUILD OF KEYCLOAK HEALTH CHECKS	82
15.1. RED HAT BUILD OF KEYCLOAK HEALTH CHECKS	82
15.2. ENABLING THE HEALTH CHECKS	82
15.3. USING THE HEALTH CHECKS	82
15.3.1. curl	82
15.3.2. Kubernetes	83
15.3.3. HEALTHCHECK	83
15.4. AVAILABLE CHECKS	83
15.5. RELEVANT OPTIONS	83
CHAPTER 16. ENABLING RED HAT BUILD OF KEYCLOAK METRICS	84
16.1. ENABLING METRICS	84
16.2. QUERYING METRICS	84
16.3. AVAILABLE METRICS	85
16.4. RELEVANT OPTIONS	85
CHAPTER 17. IMPORTING AND EXPORTING REALMS	86
17.1. PROVIDING OPTIONS FOR DATABASE CONNECTION PARAMETERS	86
17.2. EXPORTING A REALM TO A DIRECTORY	86
17.2.1. Configuring how users are exported	86
17.3. EXPORTING A REALM TO A FILE	87
17.4. EXPORTING A SPECIFIC REALM	87
17.5. IMPORTING A REALM FROM A DIRECTORY	87
17.6. IMPORTING A REALM FROM A FILE	87
17.7. IMPORTING A REALM DURING STARTUP	88
17.7.1. Using Environment Variables within the Realm Configuration Files	88
CHAPTER 18. USING A VAULT	89
18.1. AVAILABLE INTEGRATIONS	89
18.2. ENABLING A VAULT	89
18.3. CONFIGURING THE FILE-BASED VAULT	89
18.3.1. Setting the base directory to lookup secrets	89
18.3.2. Realm-specific secret files	89
18.3.3. Using underscores in the Name	89
18.4. CONFIGURING THE JAVA KEYSTORE-BASED VAULT	90
18.5. EXAMPLE: USE AN LDAP BIND CREDENTIAL SECRET IN THE ADMIN CONSOLE	90
18.6. RELEVANT OPTIONS	91
CHAPTER 19. ALL CONFIGURATION	92
19.1. CACHE	92

19.2. DATABASE	92
19.3. TRANSACTION	95
19.4. FEATURE	95
19.5. HOSTNAME	96
19.6. HTTP/TLS	98
19.7. HEALTH	100
19.8. CONFIG	101
19.9. METRICS	101
19.10. PROXY	102
19.11. VAULT	102
19.12. LOGGING	103
19.13. SECURITY	104
19.14. EXPORT	104
19.15. IMPORT	105
CHAPTER 20. ALL PROVIDER CONFIGURATION	106
20.1. AUTHENTICATION-SESSIONS	106
20.1.1. infinispan	106
20.1.2. map	106
20.2. BRUTE-FORCE-PROTECTOR	106
20.2.1. default-brute-force-detector	106
20.3. CIBA-AUTH-CHANNEL	107
20.3.1. ciba-http-auth-channel	107
20.4. CONNECTIONS-HTTP-CLIENT	107
20.4.1. default	107
20.5. CONNECTIONS-JPA	109
20.5.1. legacy	110
20.6. DBLOCK	110
20.6.1. jpa	110
20.7. EVENTS-LISTENER	110
20.7.1. email	110
20.7.2. jboss-logging	116
20.8. EXPORT	117
20.8.1. dir	117
20.8.2. single-file	117
20.9. GLOBAL-LOCK	118
20.9.1. map	118
20.10. IMPORT	118
20.10.1. dir	118
20.10.2. single-file	118
20.11. MAP-STORAGE	119
20.11.1. hotrod	119
20.11.2. jpa	119
20.12. RESOURCE-ENCODING	120
20.12.1. gzip	120
20.13. STICKY-SESSION-ENCODER	120
20.13.1. infinispan	120
20.14. TRUSTSTORE	120
20.14.1. file	120
20.15. USER-PROFILE	121
20.15.1. declarative-user-profile	121
20.16. WELL-KNOWN	122
20.16.1. openid-configuration	122

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

CHAPTER 1. CONFIGURING RED HAT BUILD OF KEYCLOAK

This chapter explains the configuration methods for Red Hat build of Keycloak and how to start and apply the preferred configuration. It includes configuration guidelines for optimizing Red Hat build of Keycloak for faster startup and low memory footprint.

1.1. CONFIGURING SOURCES FOR RED HAT BUILD OF KEYCLOAK

Red Hat build of Keycloak loads the configuration from five sources, which are listed here in order of application.

1. Command-line parameters
2. Environment variables
3. Options defined in a user-created configuration file.
4. Options defined in the **conf/keycloak.conf** file.
5. Sensitive options defined in a user-created Java KeyStore file.

When an option is set in more than one source, the one that comes first in the list determines the value for that option. For example, the value for an option set by a command-line parameter has a higher priority than an environment variable for the same option.

1.1.1. Example: Configuring the db-url-host parameter

The following example shows how the **db-url** value is set in four configuration sources:

Source	Format
Command line parameters	--db-url=cliValue
Environment variable	KC_DB_URL=envVarValue
Configuration file	db-url=confFileValue
Java KeyStore file	kc.db-url=keystoreValue

Based on the priority of application, the value that is used at startup is **cliValue**, because the command line is the highest priority.

If **--db-url=cliValue** had not been used, the applied value would be **KC_DB_URL=envVarValue**. If the value were not applied by either the command line or an environment variable, **db-url=confFileValue** would be used. However, if this value had been set in a user defined configuration file, that value would take precedence over the **conf/keycloak.conf** file. If none of the previous values were applied, the value **kc.db-url=confFileValue** would be used due to the lowest priority among the available configuration sources.

1.2. FORMATS FOR CONFIGURATION

The configuration uses a *unified-per-source* format, which simplifies translation of a key/value pair from one configuration source to another.

Command-line parameter format

Values for the command-line use the `--<key-with-dashes>=<value>` format. For some values, an `-<abbreviation>=<value>` shorthand also exists.

Environment variable format

Values for environment variables use the uppercased `KC_<key_with_underscores>=<value>` format.

Configuration file format

Values that go into the configuration file use the `<key-with-dashes>=<value>` format.

KeyStore configuration file format

Values that go into the KeyStore configuration file use the `kc.<key-with-dashes>` format. `<value>` is then a password stored in the KeyStore.

At the end of each configuration chapter, look for the *Relevant options* heading, which defines the applicable configuration formats. For all configuration options, see [All configuration](#). Choose the configuration source and format that applies to your use case.

1.2.1. Example - Alternative formats based on configuration source

The following example shows the configuration format for **db-url-host** for three configuration sources:

command-line parameter

```
bin/kc.[sh|bat] start --db-url-host=mykeycloakdb
```

environment variable

```
export KC_DB_URL_HOST=mykeycloakdb
```

conf/keycloak.conf

```
db-url-host=mykeycloakdb
```

1.2.2. Formats for command-line parameters

Red Hat build of Keycloak is packed with many command line parameters for configuration. To see the available configuration formats, enter the following command:

```
bin/kc.[sh|bat] start --help
```

Alternatively, see [All configuration](#) for all server options.

1.2.3. Formats for environment variables

You can use placeholders to resolve an environment specific value from environment variables inside the **keycloak.conf** file by using the `${ENV_VAR}` syntax:

```
db-url-host=${MY_DB_HOST}
```

In case the environment variable cannot be resolved, you can specify a fallback value. Use a `:` (colon) as shown here before **mydb**:

```
db-url-host=${MY_DB_HOST:mydb}
```

1.2.4. Format to include a specific configuration file

By default, the server always fetches configuration options from the **conf/keycloak.conf** file. For a new installation, this file holds only commented settings as an idea of what you want to set when running in production.

You can also specify an explicit configuration file location using the **[-cf|--config-file]** option by entering the following command:

```
bin/kc.[sh|bat] --config-file=/path/to/myconfig.conf start
```

1.2.5. Setting sensitive options using a Java KeyStore file

Thanks to KeyStore Configuration Source you can directly load properties from a Java KeyStore using the **[-config-keystore]** and **[-config-keystore-password]** options. Optionally, you can specify the KeyStore type using the **[-config-keystore-type]** option. By default, the KeyStore type is **PKCS12**.

The secrets in a KeyStore need to be stored using the **PBE** (password-based encryption) key algorithm, where a key is derived from a KeyStore password. You can generate such a KeyStore using the following **keytool** command:

```
keytool -importpass -alias kc.db-password -keystore keystore.p12 -storepass keystorepass -storetype PKCS12 -v
```

After executing the command, you will be prompted to **Enter the password to be stored** which represents a value of the **kc.db-password** property above.

When the KeyStore is created, you can start the server using the following parameters:

```
bin/kc.[sh|bat] start --config-keystore=/path/to/keystore.p12 --config-keystore-password=storepass --config-keystore-type=PKCS12
```

1.2.6. Format for raw Quarkus properties

In most cases, the available configuration options should suffice to configure the server. However, for a specific behavior or capability that is missing in the Red Hat build of Keycloak configuration, you can use properties from the underlying Quarkus framework.

If possible, avoid using properties directly from Quarkus, because they are unsupported by Red Hat build of Keycloak. If your need is essential, consider opening an [enhancement request](#) first. This approach helps us improve the configuration of Red Hat build of Keycloak to fit your needs.

If an enhancement request is not possible, you can configure the server using raw Quarkus properties:

1. Create a **quarkus.properties** file in the **conf** directory.
2. Define the required properties in that file.

You can use only a [subset](#) of the Quarkus extensions that are defined in the [Quarkus documentation](#). Also, note these differences for Quarkus properties:

- A lock icon for a Quarkus property in the [Quarkus documentation](#) indicates a build time property. You run the **build** command to apply this property. For details about the build command, see the subsequent sections on optimizing Red Hat build of Keycloak.
- No lock icon for a property in the Quarkus guide indicates a runtime property for Quarkus and Red Hat build of Keycloak.

3. Use the **[-cf|--config-file]** command line parameter to include that file.

Similarly, you can also store Quarkus properties in a Java KeyStore.

Note that some Quarkus properties are already mapped in the Red Hat build of Keycloak configuration, such as **quarkus.http.port** and similar essential properties. If the property is used by Keycloak, defining that property key in **quarkus.properties** has no effect. The Keycloak configuration value takes precedence over the Quarkus property value.

1.3. STARTING RED HAT BUILD OF KEYCLOAK

You can start Red Hat build of Keycloak in **development mode** or **production mode**. Each mode offers different defaults for the intended environment.

1.3.1. Starting Red Hat build of Keycloak in development mode

Use development mode to try out Red Hat build of Keycloak for the first time to get it up and running quickly. This mode offers convenient defaults for developers, such as for developing a new Red Hat build of Keycloak theme.

To start in development mode, enter the following command:

```
bin/kc.[sh|bat] start-dev
```

Defaults

Development mode sets the following default configuration:

- HTTP is enabled
- Strict hostname resolution is disabled
- Cache is set to local (No distributed cache mechanism used for high availability)
- Theme-caching and template-caching is disabled

1.3.2. Starting Red Hat build of Keycloak in production mode

Use production mode for deployments of Red Hat build of Keycloak in production environments. This mode follows a *secure by default* principle.

To start in production mode, enter the following command:

```
bin/kc.[sh|bat] start
```

Without further configuration, this command will not start Red Hat build of Keycloak and show you an error instead. This response is done on purpose, because Red Hat build of Keycloak follows a *secure by default* principle. Production mode expects a hostname to be set up and an HTTPS/TLS setup to be available when started.

Defaults

Production mode sets the following defaults:

- HTTP is disabled as transport layer security (HTTPS) is essential
- Hostname configuration is expected
- HTTPS/TLS configuration is expected

Before deploying Red Hat build of Keycloak in a production environment, make sure to follow the steps outlined in [Configuring Red Hat build of Keycloak for production](#).

By default, example configuration options for the production mode are commented out in the default **conf/keycloak.conf** file. These options give you an idea about the main configuration to consider when running Red Hat build of Keycloak in production.

1.4. CREATING THE INITIAL ADMIN USER

You can create the initial admin user by using the web frontend, which you access using a local connection (localhost). You can instead create this user by using environment variables. Set **KEYCLOAK_ADMIN=<username>** for the initial admin username and **KEYCLOAK_ADMIN_PASSWORD=<password>** for the initial admin password.

Red Hat build of Keycloak parses these values at first startup to create an initial user with administrative rights. Once the first user with administrative rights exists, you can use the Admin Console or the command line tool **kcadm.[sh|bat]** to create additional users.

If the initial administrator already exists and the environment variables are still present at startup, an error message stating the failed creation of the initial administrator is shown in the logs. Red Hat build of Keycloak ignores the values and starts up correctly.

1.5. OPTIMIZE THE RED HAT BUILD OF KEYCLOAK STARTUP

We recommend optimizing Red Hat build of Keycloak to provide faster startup and better memory consumption before deploying Red Hat build of Keycloak in a production environment. This section describes how to apply Red Hat build of Keycloak optimizations for the best performance and runtime behavior.

1.5.1. Creating an optimized Red Hat build of Keycloak build

By default, when you use the **start** or **start-dev** command, Red Hat build of Keycloak runs a **build** command under the covers for convenience reasons.

This **build** command performs a set of optimizations for the startup and runtime behavior. The build process can take a few seconds. Especially when running Red Hat build of Keycloak in containerized environments such as Kubernetes or OpenShift, startup time is important. To avoid losing that time, run a **build** explicitly before starting up, such as a separate step in a CI/CD pipeline.

1.5.1.1. First step: Run a build explicitly

To run a **build**, enter the following command:

```
bin/kc.[sh|bat] build <build-options>
```

This command shows **build options** that you enter. Red Hat build of Keycloak distinguishes between **build options**, that are usable when running the **build** command, and **configuration options**, that are usable when starting up the server.

For a non-optimized startup of Red Hat build of Keycloak, this distinction has no effect. However, if you run a build before the startup, only a subset of options is available to the build command. The restriction is due to the build options getting persisted into an optimized Red Hat build of Keycloak image. For example, configuration for credentials such as **db-password** (which is a configuration option) must not get persisted for security reasons.



WARNING

All build options are persisted in a plain text. Do not store any sensitive data as the build options. This applies across all the available configuration sources, including the KeyStore Config Source. Hence, we also do not recommend to store any build options in a Java keystore. Also, when it comes to the configuration options, we recommend to use the KeyStore Config Source primarily for storing sensitive data. For non-sensitive data you can use the remaining configuration sources.

Build options are marked in [All configuration](#) with a tool icon. To find available build options, enter the following command:

```
bin/kc.[sh|bat] build --help
```

Example: Run a build to set the database to PostgreSQL before startup

```
bin/kc.[sh|bat] build --db=postgres
```

1.5.1.2. Second step: Start Red Hat build of Keycloak using **--optimized**

After a successful build, you can start Red Hat build of Keycloak and turn off the default startup behavior by entering the following command:

```
bin/kc.[sh|bat] start --optimized <configuration-options>
```

The **--optimized** parameter tells Red Hat build of Keycloak to assume a pre-built, already optimized Red Hat build of Keycloak image is used. As a result, Red Hat build of Keycloak avoids checking for and running a build directly at startup, which saves time.

You can enter all configuration options at startup; these options are the ones in [All configuration](#) that are **not** marked with a tool icon.

- If a build option is found at startup with a value that is equal to the value used when entering the **build**, that option gets silently ignored when you use the **--optimized** parameter.

- If that option has a different value than the value used when a build was entered, a warning appears in the logs and the previously built value is used. For this value to take effect, run a new **build** before starting.

Create an optimized build

The following example shows the creation of an optimized build followed by the use of the **--optimized** parameter when starting Red Hat build of Keycloak.

1. Set the build option for the PostgreSQL database vendor using the build command

```
bin/kc.[sh|bat] build --db=postgres
```

2. Set the runtime configuration options for postgres in the **conf/keycloak.conf** file.

```
db-url-host=keycloak-postgres
db-username=keycloak
db-password=change_me
hostname=mykeycloak.acme.com
https-certificate-file
```

3. Start the server with the optimized parameter

```
bin/kc.[sh|bat] start --optimized
```

You can achieve most optimizations to startup and runtime behavior by using the **build** command. Also, by using the **keycloak.conf** file as a configuration source, you avoid some steps at startup that would otherwise require command line parameters, such as initializing the CLI itself. As a result, the server starts up even faster.

1.6. UNDERLYING CONCEPTS

This section gives an overview of the underlying concepts Red Hat build of Keycloak uses, especially when it comes to optimizing the startup.

Red Hat build of Keycloak uses the Quarkus framework and a re-augmentation/mutable-jar approach under the covers. This process is started when a **build** command is run.

The following are some optimizations performed by the **build** command:

- A new closed-world assumption about installed providers is created, meaning that no need exists to re-create the registry and initialize the factories at every Red Hat build of Keycloak startup.
- Configuration files are pre-parsed to reduce I/O when starting the server.
- Database specific resources are configured and prepared to run against a certain database vendor.
- By persisting build options into the server image, the server does not perform any additional step to interpret configuration options and (re)configure itself.

You can read more at the specific [Quarkus guide](#)

CHAPTER 2. CONFIGURING RED HAT BUILD OF KEYCLOAK FOR PRODUCTION

A Red Hat build of Keycloak production environment provides secure authentication and authorization for deployments that range from on-premise deployments that support a few thousand users to deployments that serve millions of users.

This chapter describes the general areas of configuration required for a production ready Red Hat build of Keycloak environment. This information focuses on the general concepts instead of the actual implementation, which depends on your environment. The key aspects covered in this chapter apply to all environments, whether it is containerized, on-premise, GitOps, or Ansible.

2.1. TLS FOR SECURE COMMUNICATION

Red Hat build of Keycloak continually exchanges sensitive data, which means that all communication to and from Red Hat build of Keycloak requires a secure communication channel. To prevent several attack vectors, you enable HTTP over TLS, or HTTPS, for that channel.

To configure secure communication channels for Red Hat build of Keycloak, see [Configuring TLS](#) and [Configuring outgoing HTTP requests](#).

2.2. THE HOSTNAME FOR RED HAT BUILD OF KEYCLOAK

In a production environment, Red Hat build of Keycloak instances usually run in a private network, but Red Hat build of Keycloak needs to expose certain public facing endpoints to communicate with the applications to be secured.

For details on the endpoint categories and instructions on how to configure the public hostname for them, see [Configuring the hostname](#).

2.3. REVERSE PROXY IN A DISTRIBUTED ENVIRONMENT

Apart from [Configuring the hostname](#), production environments usually include a reverse proxy / load balancer component. It separates and unifies access to the network used by your company or organization. For a Red Hat build of Keycloak production environment, this component is recommended.

For details on configuring proxy communication modes in Red Hat build of Keycloak, see [Using a reverse proxy](#). That chapter also recommends which paths should be hidden from public access and which paths should be exposed so that Red Hat build of Keycloak can secure your applications.

2.4. PRODUCTION GRADE DATABASE

The database used by Red Hat build of Keycloak is crucial for the overall performance, availability, reliability and integrity of Red Hat build of Keycloak. For details on how to configure a supported database, see [Configuring the database](#).

2.5. SUPPORT FOR RED HAT BUILD OF KEYCLOAK IN A CLUSTER

To ensure that users can continue to log in when a Red Hat build of Keycloak instance goes down, a typical production environment contains two or more Red Hat build of Keycloak instances.

Red Hat build of Keycloak runs on top of JGroups and Infinispan, which provide a reliable, high-availability stack for a clustered scenario. When deployed to a cluster, the embedded Infinispan server

communication should be secured. You secure this communication either by enabling authentication and encryption or by isolating the network used for cluster communication.

To find out more about using multiple nodes, the different caches and an appropriate stack for your environment, see [Configuring distributed caches](#).

2.6. CONFIGURE RED HAT BUILD OF KEYCLOAK SERVER WITH IPV4 OR IPV6

The system properties **java.net.preferIPv4Stack** and **java.net.preferIPv6Addresses** are used to configure the JVM for use with IPv4 or IPv6 addresses.

By default, Red Hat build of Keycloak is accessible via IPv4 and IPv6 addresses at the same time. In order to run only with IPv4 addresses, you need to specify the property **java.net.preferIPv4Stack=true**. The latter ensures that any hostname to IP address conversions always return IPv4 address variants.

These system properties are conveniently set by the **JAVA_OPTS_APPEND** environment variable. For example, to change the IP stack preference to IPv4, set an environment variable as follows:

```
export JAVA_OPTS_APPEND="-Djava.net.preferIPv4Stack=true"
```

CHAPTER 3. RUNNING RED HAT BUILD OF KEYCLOAK IN A CONTAINER

This chapter describes how to optimize and run the Red Hat build of Keycloak container image to provide the best experience running a Red Hat build of Keycloak container.



WARNING

This chapter applies only for building an image that you run in a OpenShift environment. Only an OpenShift environment is supported for this image. It is not supported if you run it in other Kubernetes distributions.

3.1. CREATING A CUSTOMIZED AND OPTIMIZED CONTAINER IMAGE

The default Red Hat build of Keycloak container image ships ready to be configured and optimized.

For the best start up of your Red Hat build of Keycloak container, build an image by running the **build** step during the container build. This step will save time in every subsequent start phase of the container image.

3.1.1. Writing your optimized Red Hat build of Keycloak Dockerfile

The following **Dockerfile** creates a pre-configured Red Hat build of Keycloak image that enables the health and metrics endpoints, enables the token exchange feature, and uses a PostgreSQL database.

Dockerfile:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:22 as builder

# Enable health and metrics support
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true

# Configure a database vendor
ENV KC_DB=postgres

WORKDIR /opt/keycloak
# for demonstration purposes only, please make sure to use proper certificates in production instead
RUN keytool -genkeypair -storepass password -storetype PKCS12 -keyalg RSA -keysize 2048 -
dname "CN=server" -alias server -ext "SAN:c=DNS:localhost,IP:127.0.0.1" -keystore
conf/server.keystore
RUN /opt/keycloak/bin/kc.sh build

FROM registry.redhat.io/rhbk/keycloak-rhel9:22
COPY --from=builder /opt/keycloak/ /opt/keycloak/

# change these values to point to a running postgres instance
ENV KC_DB=postgres
ENV KC_DB_URL=<DBURL>
```

```
ENV KC_DB_USERNAME=<DBUSERNAME>
ENV KC_DB_PASSWORD=<DBPASSWORD>
ENV KC_HOSTNAME=localhost
ENTRYPOINT ["/opt/keycloak/bin/kc.sh"]
```

The build process includes multiple stages:

- Run the **build** command to set server build options to create an optimized image.
- The files generated by the **build** stage are copied into a new image.
- In the final image, additional configuration options for the hostname and database are set so that you don't need to set them again when running the container.
- In the entrypoint, the **kc.sh** enables access to all the distribution sub-commands.

To install custom providers, you just need to define a step to include the JAR file(s) into the **/opt/keycloak/providers** directory:

```
# A example build step that downloads a JAR file from a URL and adds it to the providers directory
ADD --chown=keycloak:keycloak <MY_PROVIDER_JAR_URL>
/opt/keycloak/providers/myprovider.jar
```

3.1.2. Installing additional RPM packages

If you try to install new software in a stage **FROM registry.redhat.io/rhbk/keycloak-rhel9**, you will notice that **microdnf**, **dnf**, and even **rpm** are not installed. Also, very few packages are available, only enough for a **bash** shell, and to run Keycloak itself. This is due to security hardening measures, which reduce the attack surface of the Keycloak container.

First, consider if your use case can be implemented in a different way, and so avoid installing new RPMs into the final container:

- A **RUN curl** instruction in your Dockerfile can be replaced with **ADD**, since that instruction natively supports remote URLs.
- Some common CLI tools can be replaced by creative use of the Linux filesystem. For example, **ip addr show tap0** becomes **cat /sys/class/net/tap0/address**
- Tasks that need RPMs can be moved to a former stage of an image build, and the results copied across instead.

Here is an example. Running **update-ca-trust** in a former build stage, then copying the result forward:

```
FROM registry.access.redhat.com/ubi9 AS ubi-micro-build
COPY mycertificate.crt /etc/pki/ca-trust/source/anchors/mycertificate.crt
RUN update-ca-trust

FROM registry.redhat.io/rhbk/keycloak-rhel9
COPY --from=ubi-micro-build /etc/pki /etc/pki
```

It is possible to install new RPMs if absolutely required, following this two-stage pattern established by ubi-micro:

```
FROM registry.access.redhat.com/ubi9 AS ubi-micro-build
```

```

RUN mkdir -p /mnt/rootfs
RUN dnf install --installroot /mnt/rootfs <package names go here> --releasever 9 --setopt
install_weak_deps=false --nodocs -y && \
  dnf --installroot /mnt/rootfs clean all && \
  rpm --root /mnt/rootfs -e --nodeps setup

FROM registry.redhat.io/rhbk/keycloak-rhel9
COPY --from=ubi-micro-build /mnt/rootfs /

```

This approach uses a chroot, **/mnt/rootfs**, so that only the packages you specify and their dependencies are installed, and so can be easily copied into the second stage without guesswork.



WARNING

Some packages have a large tree of dependencies. By installing new RPMs you may unintentionally increase the container's attack surface. Check the list of installed packages carefully.

3.1.3. Building the container image

To build the actual container image, run the following command from the directory containing your Dockerfile:

```
podman build . -t mykeycloak
```

3.1.4. Starting the optimized Red Hat build of Keycloak container image

To start the image, run:

```

podman run --name mykeycloak -p 8443:8443 \
  -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
  mykeycloak \
  start --optimized

```

Red Hat build of Keycloak starts in production mode, using only secured HTTPS communication, and is available on <https://localhost:8443>.

Health check endpoints are available at <https://localhost:8443/health>, <https://localhost:8443/health/ready> and <https://localhost:8443/health/live>.

Opening up <https://localhost:8443/metrics> leads to a page containing operational metrics that could be used by your monitoring solution.

3.2. EXPOSING THE CONTAINER TO A DIFFERENT PORT

By default, the server is listening for **http** and **https** requests using the ports **8080** and **8443**, respectively.

If you want to expose the container using a different port, you need to set the **hostname-port** accordingly:

1. Exposing the container using a port other than the default ports

```
podman run --name mykeycloak -p 3000:8443 \  
-e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \  
mykeycloak \  
start --optimized --hostname-port=3000
```

By setting the **hostname-port** option you can now access the server at <https://localhost:3000>.

3.3. TRYING RED HAT BUILD OF KEYCLOAK IN DEVELOPMENT MODE

The easiest way to try Red Hat build of Keycloak from a container for development or testing purposes is to use the Development mode. You use the **start-dev** command:

```
podman run --name mykeycloak -p 8080:8080 \  
-e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \  
registry.redhat.io/rhbk/keycloak-rhel9:22 \  
start-dev
```

Invoking this command starts the Red Hat build of Keycloak server in development mode.

This mode should be strictly avoided in production environments because it has insecure defaults. For more information about running Red Hat build of Keycloak in production, see [Configuring Red Hat build of Keycloak for production](#).

3.4. RUNNING A STANDARD RED HAT BUILD OF KEYCLOAK CONTAINER

In keeping with concepts such as immutable infrastructure, containers need to be re-provisioned routinely. In these environments, you need containers that start fast, therefore you need to create an optimized image as described in the preceding section. However, if your environment has different requirements, you can run a standard Red Hat build of Keycloak image by just running the **start** command. For example:

```
podman run --name mykeycloak -p 8080:8080 \  
-e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \  
registry.redhat.io/rhbk/keycloak-rhel9:22 \  
start \  
--db=postgres --features=token-exchange \  
--db-url=<JDBC-URL> --db-username=<DB-USER> --db-password=<DB-PASSWORD> \  
--https-key-store-file=<file> --https-key-store-password=<password>
```

Running this command starts a Red Hat build of Keycloak server that detects and applies the build options first. In the example, the line **--db=postgres --features=token-exchange** sets the database vendor to PostgreSQL and enables the token exchange feature.

Red Hat build of Keycloak then starts up and applies the configuration for the specific environment. This approach significantly increases startup time and creates an image that is mutable, which is not the best practice.

3.5. PROVIDE INITIAL ADMIN CREDENTIALS WHEN RUNNING IN A CONTAINER

Red Hat build of Keycloak only allows to create the initial admin user from a local network connection. This is not the case when running in a container, so you have to provide the following environment variables when you run the image:

```
# setting the admin username
-e KEYCLOAK_ADMIN=<admin-user-name>

# setting the initial password
-e KEYCLOAK_ADMIN_PASSWORD=change_me
```

3.6. IMPORTING A REALM ON STARTUP

The Red Hat build of Keycloak containers have a directory `/opt/keycloak/data/import`. If you put one or more import files in that directory via a volume mount or other means and add the startup argument `--import-realm`, the Keycloak container will import that data on startup! This may only make sense to do in Dev mode.

```
podman run --name keycloak_unoptimized -p 8080:8080 \
  -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
  -v /path/to/realm/data:/opt/keycloak/data/import \
  registry.redhat.io/rhbk/keycloak-rhel9:22 \
  start-dev --import-realm
```

Feel free to join the open [GitHub Discussion](#) around enhancements of the admin bootstrapping process.

3.7. RELEVANT OPTIONS

	Value
<p>db</p> <p>The database vendor.</p> <p>CLI: <code>--db</code> Env: <code>KC_DB</code></p>	<p>dev-file (default), dev-mem, mariadb, mssql, mysql, oracle, postgres</p>
<p>db-password</p> <p>The password of the database user.</p> <p>CLI: <code>--db-password</code> Env: <code>KC_DB_PASSWORD</code></p>	

	Value
<p>db-url</p> <p>The full database JDBC URL.</p> <p>If not provided, a default URL is set based on the selected database vendor. For instance, if using postgres, the default JDBC URL would be jdbc:postgresql://localhost/keycloak.</p> <p>CLI: --db-url Env: KC_DB_URL</p>	
<p>db-username</p> <p>The username of the database user.</p> <p>CLI: --db-username Env: KC_DB_USERNAME</p>	
<p>features ■</p> <p>Enables a set of one or more features.</p> <p>CLI: --features Env: KC_FEATURES</p>	<p>account-api, account2, account3, admin-api, admin-fine-grained-authz, admin2, authorization, ciba, client-policies, client-secret-rotation, declarative-user-profile, docker, dynamic-scopes, fips, impersonation, js-adapter, kerberos, linkedin-oauth, map-storage, multi-site, par, preview, recovery-codes, scripts, step-up-authentication, token-exchange, update-email, web-authn</p>
<p>health-enabled ■</p> <p>If the server should expose health check endpoints.</p> <p>If enabled, health checks are available at the /health, /health/ready and /health/live endpoints.</p> <p>CLI: --health-enabled Env: KC_HEALTH_ENABLED</p>	<p>true, false (default)</p>

	Value
<p>hostname</p> <p>Hostname for the Keycloak server.</p> <p>CLI: --hostname Env: KC_HOSTNAME</p>	
<p>https-key-store-file</p> <p>The key store which holds the certificate information instead of specifying separate files.</p> <p>CLI: --https-key-store-file Env: KC_HTTPS_KEY_STORE_FILE</p>	
<p>https-key-store-password</p> <p>The password of the key store file.</p> <p>CLI: --https-key-store-password Env: KC_HTTPS_KEY_STORE_PASSWORD</p>	password (default)
<p>metrics-enabled <input type="checkbox"/></p> <p>If the server should expose metrics.</p> <p>If enabled, metrics are available at the /metrics endpoint.</p> <p>CLI: --metrics-enabled Env: KC_METRICS_ENABLED</p>	true, false (default)

CHAPTER 4. CONFIGURING TLS

Transport Layer Security (short: TLS) is crucial to exchange data over a secured channel. For production environments, you should never expose Red Hat build of Keycloak endpoints through HTTP, as sensitive data is at the core of what Red Hat build of Keycloak exchanges with other applications. In this chapter, you will learn how to configure Red Hat build of Keycloak to use HTTPS/TLS.

4.1. CONFIGURING TLS IN RED HAT BUILD OF KEYCLOAK

Red Hat build of Keycloak can be configured to load the required certificate infrastructure using files in PEM format or from a Java Keystore. When both alternatives are configured, the PEM files takes precedence over the Java Keystores.

4.1.1. Providing certificates in PEM format

When you use a pair of matching certificate and private key files in PEM format, you configure Red Hat build of Keycloak to use them by running the following command:

```
bin/kc.[sh|bat] start --https-certificate-file=/path/to/certfile.pem --https-certificate-key-file=/path/to/keyfile.pem
```

Red Hat build of Keycloak creates a keystore out of these files in memory and uses this keystore afterwards.

4.1.2. Providing a Java Keystore

When no keystore file is explicitly configured, but **http-enabled** is set to false, Red Hat build of Keycloak looks for a **conf/server.keystore** file.

As an alternative, you can use an existing keystore by running the following command:

```
bin/kc.[sh|bat] start --https-key-store-file=/path/to/existing-keystore-file
```

4.1.2.1. Setting the Keystore password

You can set a secure password for your keystore using the **https-key-store-password** option:

```
bin/kc.[sh|bat] start --https-key-store-password=<value>
```

If no password is set, the default password **password** is used.

4.2. CONFIGURING TLS PROTOCOLS

By default, Red Hat build of Keycloak does not enable deprecated TLS protocols. If your client supports only deprecated protocols, consider upgrading the client. However, as a temporary work-around, you can enable deprecated protocols by running the following command:

```
bin/kc.[sh|bat] start --https-protocols=<protocol>[,<protocol>]
```

To also allow TLSv1.2, use a command such as the following: **kc.sh start --https-protocols=TLSv1.3,TLSv1.2**.

4.3. SWITCHING THE HTTPS PORT

Red Hat build of Keycloak listens for HTTPS traffic on port **8443**. To change this port, use the following command:

```
bin/kc.[sh|bat] start --https-port=<port>
```

4.4. USING A TRUSTSTORE

In order to properly validate client certificates and enable certain authentication methods like two-way TLS or mTLS, you can set a trust store with all the certificates (and certificate chain) the server should be trusting. There are number of capabilities that rely on this trust store to properly authenticate clients using certificates such as:

- Mutual-TLS Client Authentication
- End-User X.509 Browser Authentication

You can configure the location of this truststore by running the following command:

```
bin/kc.[sh|bat] start --https-trust-store-file=/path/to/file
```



NOTE

This trust store is targeted for authenticating clients where Red Hat build of Keycloak is acting as a server. For configuring a trust store where Red Hat build of Keycloak is acting as a client to external services through TLS, see [Configuring trusted certificates for outgoing requests](#).

4.4.1. Setting the truststore password

You can set a secure password for your truststore using the **https-trust-store-password** option:

```
bin/kc.[sh|bat] start --https-trust-store-password=<value>
```

If no password is set, the default password **password** is used.

4.5. SECURING CREDENTIALS

Avoid setting a password in plaintext by using the CLI or adding it to **conf/keycloak.conf** file. Instead use good practices such as using a vault / mounted secret. For more detail, see [Using a vault](#) and [Configuring Red Hat build of Keycloak for production](#).

4.6. ENABLING MUTUAL TLS

Authentication using mTLS is disabled by default. To enable mTLS certificate handling when Red Hat build of Keycloak is the server and needs to validate certificates from requests made to Red Hat build of Keycloak endpoints, put the appropriate certificates in Red Hat build of Keycloak truststore and use the following command to enable mTLS:

```
bin/kc.[sh|bat] start --https-client-auth=<none|request|required>
```

Using the value **required** sets up Red Hat build of Keycloak to always ask for certificates and fail if no certificate is provided in a request. By setting the value to **request**, Red Hat build of Keycloak will also accept requests without a certificate and only validate the correctness of a certificate if it exists.

Be aware that this is the basic certificate configuration for mTLS use cases where Red Hat build of Keycloak acts as server. When Red Hat build of Keycloak acts as client instead, e.g. when Red Hat build of Keycloak tries to get a token from a token endpoint of a brokered identity provider that is secured by mTLS, you need to set up the HttpClient to provide the right certificates in the keystore for the outgoing request. To configure mTLS in these scenarios, see [Configuring outgoing HTTP requests](#).

4.7. RELEVANT OPTIONS

	Value
<p>http-enabled</p> <p>Enables the HTTP listener.</p> <p>CLI: --http-enabled Env: KC_HTTP_ENABLED</p>	true, false (default)
<p>https-certificate-file</p> <p>The file path to a server certificate or certificate chain in PEM format.</p> <p>CLI: --https-certificate-file Env: KC_HTTPS_CERTIFICATE_FILE</p>	
<p>https-certificate-key-file</p> <p>The file path to a private key in PEM format.</p> <p>CLI: --https-certificate-key-file Env: KC_HTTPS_CERTIFICATE_KEY_FILE</p>	
<p>https-cipher-suites</p> <p>The cipher suites to use.</p> <p>If none is given, a reasonable default is selected.</p> <p>CLI: --https-cipher-suites Env: KC_HTTPS_CIPHER_SUITES</p>	
<p>https-client-auth</p> <p>Configures the server to require/request client authentication.</p> <p>CLI: --https-client-auth Env: KC_HTTPS_CLIENT_AUTH</p>	none (default), request, required

	Value
<p>https-key-store-file</p> <p>The key store which holds the certificate information instead of specifying separate files.</p> <p>CLI: --https-key-store-file Env: KC_HTTPS_KEY_STORE_FILE</p>	
<p>https-key-store-password</p> <p>The password of the key store file.</p> <p>CLI: --https-key-store-password Env: KC_HTTPS_KEY_STORE_PASSWORD</p>	password (default)
<p>https-key-store-type</p> <p>The type of the key store file.</p> <p>If not given, the type is automatically detected based on the file name. If fips-mode is set to strict and no value is set, it defaults to BCFKS.</p> <p>CLI: --https-key-store-type Env: KC_HTTPS_KEY_STORE_TYPE</p>	
<p>https-port</p> <p>The used HTTPS port.</p> <p>CLI: --https-port Env: KC_HTTPS_PORT</p>	8443 (default)
<p>https-protocols</p> <p>The list of protocols to explicitly enable.</p> <p>CLI: --https-protocols Env: KC_HTTPS_PROTOCOLS</p>	TLSv1.3,TLSv1.2 (default)
<p>https-trust-store-file</p> <p>The trust store which holds the certificate information of the certificates to trust.</p> <p>CLI: --https-trust-store-file Env: KC_HTTPS_TRUST_STORE_FILE</p>	
<p>https-trust-store-password</p> <p>The password of the trust store file.</p> <p>CLI: --https-trust-store-password Env: KC_HTTPS_TRUST_STORE_PASSWORD</p>	

Value	
https-trust-store-type The type of the trust store file. If not given, the type is automatically detected based on the file name. If fips-mode is set to strict and no value is set, it defaults to BCFKS . CLI: --https-trust-store-type Env: KC_HTTPS_TRUST_STORE_TYPE	

CHAPTER 5. CONFIGURING THE HOSTNAME

5.1. SERVER ENDPOINTS

Red Hat build of Keycloak exposes different endpoints to talk with applications as well as to allow accessing the administration console. These endpoints can be categorized into three main groups:

- Frontend
- Backend
- Administration Console

The base URL for each group has an important impact on how tokens are issued and validated, on how links are created for actions that require the user to be redirected to Red Hat build of Keycloak (for example, when resetting password through email links), and, most importantly, how applications will discover these endpoints when fetching the OpenID Connect Discovery Document from **realms/{realm-name}/.well-known/openid-configuration**.

5.1.1. Frontend

The frontend endpoints are those accessible through a public domain and usually related to authentication/authorization flows that happen through the front-channel. For instance, when an SPA wants to authenticate their users it redirects them to the **authorization_endpoint** so that users can authenticate using their browsers through the front-channel.

By default, when the hostname settings are not set, the base URL for these endpoints is based on the incoming request so that the HTTP scheme, host, port, and path, are the same from the request. The default behavior also has a direct impact on how the server is going to issue tokens given that the issuer is also based on the URL set to the frontend endpoints. If the hostname settings are not set, the token issuer will also be based on the incoming request and also lack consistency if the client is requesting tokens using different URLs.

When deploying to production you usually want a consistent URL for the frontend endpoints and the token issuer regardless of how the request is constructed. In order to achieve this consistency, you can set either the **hostname** or the **hostname-url** options.

Most of the time, it should be enough to set the **hostname** option in order to change only the **host** of the frontend URLs:

```
bin/kc.[sh|bat] start --hostname=<host>
```

When using the **hostname** option the server is going to resolve the HTTP scheme, port, and path, automatically so that:

- **https** scheme is used unless you set **hostname-strict-https=false**
- if the **proxy** option is set, the proxy will use the default ports (i.e.: 80 and 443). If the proxy uses a different port, it needs to be specified via the **hostname-port** configuration option

However, if you want to set not only the host but also a scheme, port, and path, you can set the **hostname-url** option:

```
bin/kc.[sh|bat] start --hostname-url=<scheme>://<host>:<port>/<path>
```

This option gives you more flexibility as you can set the different parts of the URL from a single option. Note that the **hostname** and **hostname-url** are mutually exclusive.



NOTE

By **hostname** and **proxy** configuration options you affect only the static resources URLs, redirect URLs, OIDC well-known endpoints, etc. In order to change, where/on which port the server actually listens on, you need to use the **http/tls** configuration options (e.g. **http-host**, **https-port**, etc.). For more details, see [Configuring TLS](#) and [All configuration](#).

5.1.2. Backend

The backend endpoints are those accessible through a public domain or through a private network. They are used for a direct communication between the server and clients without any intermediary but plain HTTP requests. For instance, after the user is authenticated an SPA wants to exchange the **code** sent by the server with a set of tokens by sending a token request to **token_endpoint**.

By default, the URLs for backend endpoints are also based on the incoming request. To override this behavior, set the **hostname-strict-backchannel** configuration option by entering this command:

```
bin/kc.[sh|bat] start --hostname=<value> --hostname-strict-backchannel=true
```

By setting the **hostname-strict-backchannel** option, the URLs for the backend endpoints are going to be exactly the same as the frontend endpoints.

When all applications connected to Red Hat build of Keycloak communicate through the public URL, set **hostname-strict-backchannel** to **true**. Otherwise, leave this parameter as **false** to allow client-server communication through a private network.

5.1.3. Administration Console

The server exposes the administration console and static resources using a specific URL.

By default, the URLs for the administration console are also based on the incoming request. However, you can set a specific host or base URL if you want to restrict access to the administration console using a specific URL. Similarly to how you set the frontend URLs, you can use the **hostname-admin** and **hostname-admin-url** options to achieve that. Note that if HTTPS is enabled (**http-enabled** configuration option is set to false, which is the default setting for the production mode), the Red Hat build of Keycloak server automatically assumes you want to use HTTPS URLs. The admin console then tries to contact Red Hat build of Keycloak over HTTPS and HTTPS URLs are also used for its configured redirect/web origin URLs. It is not recommended for production, but you can use HTTP URL as **hostname-admin-url** to override this behaviour.

Most of the time, it should be enough to set the **hostname-admin** option in order to change only the **host** of the administration console URLs:

```
bin/kc.[sh|bat] start --hostname-admin=<host>
```

However, if you want to set not only the host but also a scheme, port, and path, you can set the **hostname-admin-url** option:

```
bin/kc.[sh|bat] start --hostname-admin-url=<scheme>://<host>:<port>/<path>
```

Note that the **hostname-admin** and **hostname-admin-url** are mutually exclusive.

To reduce attack surface, the administration endpoints for Red Hat build of Keycloak and the Admin Console should not be publicly accessible. Therefore, you can secure them by using a reverse proxy. For more information about which paths to expose using a reverse proxy, see [Using a reverse proxy](#).

5.2. EXAMPLE SCENARIOS

The following are more example scenarios and the corresponding commands for setting up a hostname.

Note that the **start** command requires setting up TLS. The corresponding options are not shown for example purposes. For more details, see [Configuring TLS](#).

5.2.1. Exposing the server behind a TLS termination proxy

In this example, the server is running behind a TLS termination proxy and publicly available from <https://mykeycloak>.

Configuration:

```
bin/kc.[sh|bat] start --hostname=mykeycloak --proxy=edge
```

5.2.2. Exposing the server without a proxy

In this example, the server is running without a proxy and exposed using a URL using HTTPS.

Red Hat build of Keycloak configuration:

```
bin/kc.[sh|bat] start --hostname-url=https://mykeycloak
```

It is highly recommended using a TLS termination proxy in front of the server for security and availability reasons. For more details, see [Using a reverse proxy](#).

5.2.3. Forcing backend endpoints to use the same URL the server is exposed

In this example, backend endpoints are exposed using the same URL used by the server so that clients always fetch the same URL regardless of the origin of the request.

Red Hat build of Keycloak configuration:

```
bin/kc.[sh|bat] start --hostname=mykeycloak --hostname-strict-backchannel=true
```

5.2.4. Exposing the server using a port other than the default ports

In this example, the server is accessible using a port other than the default ports.

Red Hat build of Keycloak configuration:

```
bin/kc.[sh|bat] start --hostname-url=https://mykeycloak:8989
```

5.2.5. Exposing Red Hat build of Keycloak behind a TLS reencrypt proxy using different ports

In this example, the server is running behind a proxy and both the server and the proxy are using their own certificates, so the communication between Red Hat build of Keycloak and the proxy is encrypted. Because we want the proxy to use its own certificate, the proxy mode **reencrypt** will be used. We need to keep in mind that the proxy configuration options (as well as hostname configuration options) are not changing the ports on which the server actually is listening on (it changes only the ports of static resources like JavaScript and CSS links, OIDC well-known endpoints, redirect URLs, etc.). Therefore, we need to use HTTP configuration options to change the Red Hat build of Keycloak server to internally listen on a different port, e.g. 8543. The proxy will be listening on the port 8443 (the port visible while accessing the console via a browser). The example hostname **my-keycloak.org** will be used for the server and similarly the admin console will be accessible via the **admin.my-keycloak.org** subdomain.

Red Hat build of Keycloak configuration:

```
bin/kc.[sh|bat] start --proxy=reencrypt --https-port=8543 --hostname-url=https://my-keycloak.org:8443
--hostname-admin-url=https://admin.my-keycloak.org:8443
```

Note: there is currently no difference between the **passthrough** and **reencrypt** modes. For now, this is meant for future-proof configuration compatibility. The only difference is that when the **edge** proxy mode is used, HTTP is implicitly enabled (again as mentioned above, this does not affect the server behaviour).



WARNING

Usage any of the proxy modes makes Red Hat build of Keycloak rely on Forwarded and X-Forwarded-* headers. Misconfiguration may leave Red Hat build of Keycloak exposed to security issues. For more details, see [Using a reverse proxy](#).

5.3. TROUBLESHOOTING

To troubleshoot the hostname configuration, you can use a dedicated debug tool which can be enabled as:

Red Hat build of Keycloak configuration:

```
bin/kc.[sh|bat] start --hostname=mykeycloak --hostname-debug=true
```

Then after Red Hat build of Keycloak started properly, open your browser and go to:

<http://mykeycloak:8080/realms/<your-realm>/hostname-debug>

5.4. RELEVANT OPTIONS

Table 5.1. By default, this endpoint is disabled (**-hostname-debug=false**)

	Value
<p>hostname</p> <p>Hostname for the Keycloak server.</p> <p>CLI: --hostname Env: KC_HOSTNAME</p>	
<p>hostname-admin</p> <p>The hostname for accessing the administration console.</p> <p>Use this option if you are exposing the administration console using a hostname other than the value set to the hostname option.</p> <p>CLI: --hostname-admin Env: KC_HOSTNAME_ADMIN</p>	
<p>hostname-admin-url</p> <p>Set the base URL for accessing the administration console, including scheme, host, port and path</p> <p>CLI: --hostname-admin-url Env: KC_HOSTNAME_ADMIN_URL</p>	
<p>hostname-debug</p> <p>Toggle the hostname debug page that is accessible at <code>/realms/master/hostname-debug</code></p> <p>CLI: --hostname-debug Env: KC_HOSTNAME_DEBUG</p>	true, false (default)
<p>hostname-path</p> <p>This should be set if proxy uses a different context-path for Keycloak.</p> <p>CLI: --hostname-path Env: KC_HOSTNAME_PATH</p>	
<p>hostname-port</p> <p>The port used by the proxy when exposing the hostname.</p> <p>Set this option if the proxy uses a port other than the default HTTP and HTTPS ports.</p> <p>CLI: --hostname-port Env: KC_HOSTNAME_PORT</p>	-1 (default)

	Value
<p>hostname-strict</p> <p>Disables dynamically resolving the hostname from request headers.</p> <p>Should always be set to true in production, unless proxy verifies the Host header.</p> <p>CLI: --hostname-strict Env: KC_HOSTNAME_STRICT</p>	<p>true (default), false</p>
<p>hostname-strict-backchannel</p> <p>By default backchannel URLs are dynamically resolved from request headers to allow internal and external applications.</p> <p>If all applications use the public URL this option should be enabled.</p> <p>CLI: --hostname-strict-backchannel Env: KC_HOSTNAME_STRICT_BACKCHANNEL</p>	<p>true, false (default)</p>
<p>hostname-url</p> <p>Set the base URL for frontend URLs, including scheme, host, port and path.</p> <p>CLI: --hostname-url Env: KC_HOSTNAME_URL</p>	
<p>proxy</p> <p>The proxy address forwarding mode if the server is behind a reverse proxy.</p> <p>CLI: --proxy Env: KC_PROXY</p>	<p>none (default), edge, reencrypt, passthrough</p>

CHAPTER 6. USING A REVERSE PROXY

Distributed environments frequently require the use of a reverse proxy. For Red Hat build of Keycloak, your choice of proxy modes depends on the TLS termination in your environment.

6.1. PROXY MODES

The following proxy modes are available:

edge

Enables communication through HTTP between the proxy and Red Hat build of Keycloak. This mode is suitable for deployments with a highly secure internal network where the reverse proxy keeps a secure connection (HTTP over TLS) with clients while communicating with Red Hat build of Keycloak using HTTP.

reencrypt

Requires communication through HTTPS between the proxy and Red Hat build of Keycloak. This mode is suitable for deployments where internal communication between the reverse proxy and Red Hat build of Keycloak should also be protected. Different keys and certificates are used on the reverse proxy as well as on Red Hat build of Keycloak.

passthrough

The proxy forwards the HTTPS connection to Red Hat build of Keycloak without terminating TLS. The secure connections between the server and clients are based on the keys and certificates used by the Red Hat build of Keycloak server.

6.2. CONFIGURE THE PROXY MODE IN RED HAT BUILD OF KEYCLOAK

To select the proxy mode, enter this command:

```
bin/kc.[sh|bat] start --proxy <mode>
```

6.3. CONFIGURE THE REVERSE PROXY

Some Red Hat build of Keycloak features rely on the assumption that the remote address of the HTTP request connecting to Red Hat build of Keycloak is the real IP address of the clients machine.

When in **edge** or **reencrypt** proxy mode, Red Hat build of Keycloak will parse the following headers and expects the reverse proxy to set them:

- **Forwarded** as per [RFC7239](#)
- Non-standard **X-Forwarded**
- Non-standard **X-Forwarded-***, such as **X-Forwarded-For**, **X-Forwarded-Proto**, **X-Forwarded-Host**, and **X-Forwarded-Port**

To set these headers, consult the documentation for your reverse proxy.

Take extra precautions to ensure that the client address is properly set by your reverse proxy via the **Forwarded** or **X-Forwarded-For** headers. If this header is incorrectly configured, rogue clients can set this header and trick Red Hat build of Keycloak into thinking the client is connected from a different IP address than the actual address. This precaution can be more critical if you do any deny or allow listing of IP addresses.

6.4. DIFFERENT CONTEXT-PATH ON REVERSE PROXY

Red Hat build of Keycloak assumes it is exposed through the reverse proxy under the same context path as Red Hat build of Keycloak is configured for. By default Red Hat build of Keycloak is exposed through the root (`/`), which means it expects to be exposed through the reverse proxy on `/` as well. You can use **hostname-path** or **hostname-url** in these cases, for example using `--hostname-path=/auth` if Red Hat build of Keycloak is exposed through the reverse proxy on `/auth`.

Alternatively you can also change the context path of Red Hat build of Keycloak itself to match the context path for the reverse proxy using the **http-relative-path** option, which will change the context-path of Red Hat build of Keycloak itself to match the context path used by the reverse proxy.

6.5. TRUST THE PROXY TO SET HOSTNAME

By default, Red Hat build of Keycloak needs to know under which hostname it will be called. If your reverse proxy is configured to check for the correct hostname, you can set Red Hat build of Keycloak to accept any hostname.

```
bin/kc.[sh|bat] start --proxy <mode> --hostname-strict=false
```

6.6. ENABLE STICKY SESSIONS

Typical cluster deployment consists of the load balancer (reverse proxy) and 2 or more Red Hat build of Keycloak servers on private network. For performance purposes, it may be useful if load balancer forwards all requests related to particular browser session to the same Red Hat build of Keycloak backend node.

The reason is, that Red Hat build of Keycloak is using Infinispan distributed cache under the covers for save data related to current authentication session and user session. The Infinispan distributed caches are configured with two owners by default. That means that particular session is primarily stored on two cluster nodes and the other nodes need to lookup the session remotely if they want to access it.

For example if authentication session with ID 123 is saved in the Infinispan cache on node1, and then node2 needs to lookup this session, it needs to send the request to node1 over the network to return the particular session entity.

It is beneficial if particular session entity is always available locally, which can be done with the help of sticky sessions. The workflow in the cluster environment with the public frontend load balancer and two backend Red Hat build of Keycloak nodes can be like this:

- User sends initial request to see the Red Hat build of Keycloak login screen
- This request is served by the frontend load balancer, which forwards it to some random node (eg. node1). Strictly said, the node doesn't need to be random, but can be chosen according to some other criterias (client IP address etc). It all depends on the implementation and configuration of underlying load balancer (reverse proxy).
- Red Hat build of Keycloak creates authentication session with random ID (eg. 123) and saves it to the Infinispan cache.
- Infinispan distributed cache assigns the primary owner of the session based on the hash of session ID. See Infinispan documentation for more details around this. Let's assume that Infinispan assigned node2 to be the owner of this session.

- Red Hat build of Keycloak creates the cookie `AUTH_SESSION_ID` with the format like `<session-id>.<owner-node-id>` . In our example case, it will be `123.node2` .
- Response is returned to the user with the Red Hat build of Keycloak login screen and the `AUTH_SESSION_ID` cookie in the browser

From this point, it is beneficial if load balancer forwards all the next requests to the `node2` as this is the node, who is owner of the authentication session with ID 123 and hence Infinispan can lookup this session locally. After authentication is finished, the authentication session is converted to user session, which will be also saved on `node2` because it has same ID 123 .

The sticky session is not mandatory for the cluster setup, however it is good for performance for the reasons mentioned above. You need to configure your loadbalancer to sticky over the `AUTH_SESSION_ID` cookie. How exactly do this is dependent on your loadbalancer.

If your proxy supports session affinity without processing cookies from backend nodes, you should set the **`spi-sticky-session-encoder-infinispan-should-attach-route`** option to **`false`** in order to avoid attaching the node to cookies and just rely on the reverse proxy capabilities.

```
bin/kc.[sh|bat] start --spi-sticky-session-encoder-infinispan-should-attach-route=false
```

By default, the **`spi-sticky-session-encoder-infinispan-should-attach-route`** option value is **`true`** so that the node name is attached to cookies to indicate to the reverse proxy the node that subsequent requests should be sent to.

6.6.1. Exposing the administration console

By default, the administration console URLs are created solely based on the requests to resolve the proper scheme, host name, and port. For instance, if you are using the **`edge`** proxy mode and your proxy is misconfigured, backend requests from your TLS termination proxy are going to use plain HTTP and potentially cause the administration console from being accessible because URLs are going to be created using the **`http`** scheme and the proxy does not support plain HTTP.

In order to proper expose the administration console, you should make sure that your proxy is setting the **`X-Forwarded-*`** headers herein mentioned in order to create URLs using the scheme, host name, and port, being exposed by your proxy.

6.6.2. Exposed path recommendations

When using a reverse proxy, Red Hat build of Keycloak only requires certain paths need to be exposed. The following table shows the recommended paths to expose.

Red Hat build of Keycloak Path	Reverse Proxy Path	Exposed	Reason
/	-	No	When exposing all paths, admin paths are exposed unnecessarily.
/admin/	-	No	Exposed admin paths lead to an unnecessary attack vector.

Red Hat build of Keycloak Path	Reverse Proxy Path	Exposed	Reason
/js/	-	Yes (see note below)	Access to keycloak.js needed for "internal" clients, e.g. the account console
/welcome/	-	No	No need exists to expose the welcome page after initial installation.
/realms/	/realms/	Yes	This path is needed to work correctly, for example, for OIDC endpoints.
/resources/	/resources/	Yes	This path is needed to serve assets correctly. It may be served from a CDN instead of the Red Hat build of Keycloak path.
/robots.txt	/robots.txt	Yes	Search engine rules
/metrics	-	No	Exposed metrics lead to an unnecessary attack vector.
/health	-	No	Exposed health checks lead to an unnecessary attack vector.



NOTE

As it's true that the **js** path is needed for internal clients like the account console, it's good practice to use **keycloak.js** from a JavaScript package manager like npm or yarn for your external clients.

We assume you run Red Hat build of Keycloak on the root path / on your reverse proxy/gateway's public API. If not, prefix the path with your desired one.

6.6.3. Enabling client certificate lookup

When the proxy is configured as a TLS termination proxy the client certificate information can be forwarded to the server through specific HTTP request headers and then used to authenticate clients. You are able to configure how the server is going to retrieve client certificate information depending on

the proxy you are using.

The server supports some of the most common TLS termination proxies such as:

Proxy	Provider
Apache HTTP Server	apache
HAProxy	haproxy
NGINX	nginx

To configure how client certificates are retrieved from the requests you need to:

Enable the corresponding proxy provider

```
bin/kc.[sh|bat] build --spi-x509cert-lookup-provider=<provider>
```

Configure the HTTP headers

```
bin/kc.[sh|bat] start --spi-x509cert-lookup-<provider>-ssl-client-cert=SSL_CLIENT_CERT --spi-x509cert-lookup-<provider>-ssl-cert-chain-prefix=CERT_CHAIN --spi-x509cert-lookup-<provider>-certificate-chain-length=10
```

When configuring the HTTP headers, you need to make sure the values you are using correspond to the name of the headers forwarded by the proxy with the client certificate information.

The available options for configuring a provider are:

Option	Description
ssl-client-cert	The name of the header holding the client certificate
ssl-cert-chain-prefix	The prefix of the headers holding additional certificates in the chain and used to retrieve individual certificates accordingly to the length of the chain. For instance, a value CERT_CHAIN will tell the server to load additional certificates from headers CERT_CHAIN_0 to CERT_CHAIN_9 if certificate-chain-length is set to 10 .
certificate-chain-length	The maximum length of the certificate chain.
trust-proxy-verification	Enable trusting NGINX proxy certificate verification, instead of forwarding the certificate to Red Hat build of Keycloak and verifying it in Red Hat build of Keycloak.

6.6.3.1. Configuring the NGINX provider

The NGINX SSL/TLS module does not expose the client certificate chain. Red Hat build of Keycloak's NGINX certificate lookup provider rebuilds it by using the Red Hat build of Keycloak truststore.

If you are using this provider, see [Configuring trusted certificates for outgoing requests](#) for how to configure a Red Hat build of Keycloak Truststore.

6.7. RELEVANT OPTIONS

	Value
<p>hostname-path</p> <p>This should be set if proxy uses a different context-path for Keycloak.</p> <p>CLI: --hostname-path Env: KC_HOSTNAME_PATH</p>	
<p>hostname-url</p> <p>Set the base URL for frontend URLs, including scheme, host, port and path.</p> <p>CLI: --hostname-url Env: KC_HOSTNAME_URL</p>	
<p>http-relative-path ■</p> <p>Set the path relative to / for serving resources.</p> <p>The path must start with a /.</p> <p>CLI: --http-relative-path Env: KC_HTTP_RELATIVE_PATH</p>	/ (default)
<p>proxy</p> <p>The proxy address forwarding mode if the server is behind a reverse proxy.</p> <p>CLI: --proxy Env: KC_PROXY</p>	none (default), edge , reencrypt , passthrough

CHAPTER 7. CONFIGURING THE DATABASE

This chapter explains how to configure the Red Hat build of Keycloak server to store data in a relational database.

7.1. SUPPORTED DATABASES

The server has built-in support for different databases. You can query the available databases by viewing the expected values for the **db** configuration option. The following table lists the supported databases and their tested versions.

Database	Option value	Tested Version
MariaDB Server	mariadb	10.11
Microsoft SQL Server	mssql	2022
MySQL	mysql	8.0
Oracle Database	oracle	19.3
PostgreSQL	postgres	16

By default, the server uses the **dev-file** database. This is the default database that the server will use to persist data and only exists for development use-cases. The **dev-file** database is not suitable for production use-cases, and must be replaced before deploying to production.

7.2. INSTALLING A DATABASE DRIVER

Database drivers are shipped as part of Red Hat build of Keycloak except for the Oracle Database and Microsoft SQL Server drivers which need to be installed separately.

Install the necessary driver if you want to connect to one of these databases or skip this section if you want to connect to a different database for which the database driver is already included.

7.2.1. Installing the Oracle Database driver

To install the Oracle Database driver for Red Hat build of Keycloak:

1. Download the **ojdbc11** and **orai18n** JAR files from one of the following sources:
 - a. **Zipped JDBC driver and Companion Jars** version 23.2.0.0 from the [Oracle driver download page](#).
 - b. Maven Central via **ojdbc11** and **orai18n**.
 - c. Installation media recommended by the database vendor for the specific database in use.
2. When running the unzipped distribution: Place the **ojdbc11** and **orai18n** JAR files in Red Hat build of Keycloak's **providers** folder

- When running containers: Build a custom Red Hat build of Keycloak image and add the JARs in the **providers** folder. When building a custom image for the Keycloak Operator, those images need to be optimized images with all build-time options of Keycloak set.

A minimal Dockerfile to build an image which can be used with the Red Hat build of Keycloak Operator and includes Oracle Database JDBC drivers downloaded from Maven Central looks like the following:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:22
ADD --chown=keycloak:keycloak
https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc11/23.2.0.0/ojdbc11-23.2.0.0.jar /opt/keycloak/providers/ojdbc11.jar
ADD --chown=keycloak:keycloak
https://repo1.maven.org/maven2/com/oracle/database/nls/orai18n/23.2.0.0/orai18n-23.2.0.0.jar /opt/keycloak/providers/orai18n.jar
# Setting the build parameter for the database:
ENV KC_DB=oracle
# Add all other build parameters needed, for example enable health and metrics:
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true
# To be able to use the image with the {project_name} Operator, it needs to be optimized,
which requires {project_name}'s build step:
RUN /opt/keycloak/bin/kc.sh build
```

See the [Running Red Hat build of Keycloak in a container](#) chapter for details on how to build optimized images.

Then continue configuring the database as described in the next section.

7.2.2. Installing the Microsoft SQL Server driver

To install the Microsoft SQL Server driver for Red Hat build of Keycloak:

- Download the **mssql-jdbc** JAR file from one of the following sources:
 - Download a version from the [Microsoft JDBC Driver for SQL Server](#) page.
 - Maven Central via [mssql-jdbc](#).
 - Installation media recommended by the database vendor for the specific database in use.
- When running the unzipped distribution: Place the **mssql-jdbc** in Red Hat build of Keycloak's **providers** folder
- When running containers: Build a custom Red Hat build of Keycloak image and add the JARs in the **providers** folder. When building a custom image for the Red Hat build of Keycloak Operator, those images need to be optimized images with all build-time options of Red Hat build of Keycloak set.

A minimal Dockerfile to build an image which can be used with the Red Hat build of Keycloak Operator and includes Microsoft SQL Server JDBC drivers downloaded from Maven Central looks like the following:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:22
ADD --chown=keycloak:keycloak
https://repo1.maven.org/maven2/com/microsoft/sqlserver/mssql-jdbc/12.2.0.jre11/mssql-jdbc-12.2.0.jre11.jar /opt/keycloak/providers/mssql-jdbc.jar
# Setting the build parameter for the database:
```

```
ENV KC_DB=mssql
# Add all other build parameters needed, for example enable health and metrics:
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true
# To be able to use the image with the {project_name} Operator, it needs to be optimized,
which requires {project_name}'s build step:
RUN /opt/keycloak/bin/kc.sh build
```

See the [Running Red Hat build of Keycloak in a container](#) chapter for details on how to build optimized images.

Then continue configuring the database as described in the next section.

7.3. CONFIGURING A DATABASE

For each supported database, the server provides some opinionated defaults to simplify database configuration. You complete the configuration by providing some key settings such as the database host and credentials.

1. Start the server and set the basic options to configure a database

```
bin/kc.[sh|bat] start --db postgres --db-url-host mypostgres --db-username myuser --db-
password change_me
```

This command includes the minimum settings needed to connect to the database.

The default schema is **keycloak**, but you can change it by using the **db-schema** configuration option.



WARNING

Do NOT use the **--optimized** flag for the **start** command if you want to use a particular DB (except the H2). Executing the build phase before starting the server instance is necessary. You can achieve it either by starting the instance without the **-optimized** flag, or by executing the **build** command before the optimized start. For more information, see [Configuring Red Hat build of Keycloak](#).

7.4. OVERRIDING DEFAULT CONNECTION SETTINGS

The server uses JDBC as the underlying technology to communicate with the database. If the default connection settings are insufficient, you can specify a JDBC URL using the **db-url** configuration option.

The following is a sample command for a PostgreSQL database.

```
bin/kc.[sh|bat] start --db postgres --db-url jdbc:postgresql://mypostgres/mydatabase
```

Be aware that you need to escape characters when invoking commands containing special shell characters such as `;` using the CLI, so you might want to set it in the configuration file instead.

7.5. OVERRIDING THE DEFAULT JDBC DRIVER

The server uses a default JDBC driver accordingly to the database you chose.

To set a different driver you can set the **db-driver** with the fully qualified class name of the JDBC driver:

```
bin/kc.[sh|bat] start --db postgres --db-driver=my.Driver
```

Regardless of the driver you set, the default driver is always available at runtime.

Only set this property if you really need to. For instance, when leveraging the capabilities from a JDBC Driver Wrapper for a specific cloud database service.

7.6. CONFIGURING UNICODE SUPPORT FOR THE DATABASE

Unicode support for all fields depends on whether the database allows VARCHAR and CHAR fields to use the Unicode character set.

- If these fields can be set, Unicode is likely to work, usually at the expense of field length.
- If the database only supports Unicode in the NVARCHAR and NCHAR fields, Unicode support for all text fields is unlikely to work because the server schema uses VARCHAR and CHAR fields extensively.

The database schema provides support for Unicode strings only for the following special fields:

- **Realms:** display name, HTML display name, localization texts (keys and values)
- **Federation Providers:** display name
- **Users:** username, given name, last name, attribute names and values
- **Groups:** name, attribute names and values
- **Roles:** name
- Descriptions of objects

Otherwise, characters are limited to those contained in database encoding, which is often 8-bit. However, for some database systems, you can enable UTF-8 encoding of Unicode characters and use the full Unicode character set in all text fields. For a given database, this choice might result in a shorter maximum string length than the maximum string length supported by 8-bit encodings.

7.6.1. Configuring Unicode support for an Oracle database

Unicode characters are supported in an Oracle database if the database was created with Unicode support in the VARCHAR and CHAR fields. For example, you configured AL32UTF8 as the database character set. In this case, the JDBC driver requires no special settings.

If the database was not created with Unicode support, you need to configure the JDBC driver to support Unicode characters in the special fields. You configure two properties. Note that you can configure these properties as system properties or as connection properties.

1. Set **oracle.jdbc.defaultNChar** to **true**.
2. Optionally, set **oracle.jdbc.convertNcharLiterals** to **true**.

**NOTE**

For details on these properties and any performance implications, see the Oracle JDBC driver configuration documentation.

7.6.2. Unicode support for a Microsoft SQL Server database

Unicode characters are supported only for the special fields for a Microsoft SQL Server database. The database requires no special settings.

The **sendStringParametersAsUnicode** property of JDBC driver should be set to **false** to significantly improve performance. Without this parameter, the Microsoft SQL Server might be unable to use indexes.

7.6.3. Configuring Unicode support for a MySQL database

Unicode characters are supported in a MySQL database if the database was created with Unicode support in the VARCHAR and CHAR fields when using the CREATE DATABASE command.

Note that the utf8mb4 character set is not supported due to different storage requirements for the utf8 character set. See MySQL documentation for details. In that situation, the length restriction on non-special fields does not apply because columns are created to accommodate the number of characters, not bytes. If the database default character set does not allow Unicode storage, only the special fields allow storing Unicode values.

1. Start MySQL Server.
2. Under JDBC driver settings, locate the **JDBC connection settings**.
3. Add this connection property: **characterEncoding=UTF-8**

7.6.4. Configuring Unicode support for a PostgreSQL database

Unicode is supported for a PostgreSQL database when the database character set is UTF8. Unicode characters can be used in any field with no reduction of field length for non-special fields. The JDBC driver requires no special settings. The character set is determined when the PostgreSQL database is created.

1. Check the default character set for a PostgreSQL cluster by entering the following SQL command.

```
show server_encoding;
```

2. If the default character set is not UTF 8, create the database with the UTF8 as the default character set using a command such as:

```
create database keycloak with encoding 'UTF8';
```

7.7. CHANGING DATABASE LOCKING TIMEOUT IN A CLUSTER CONFIGURATION

Because cluster nodes can boot concurrently, they take extra time for database actions. For example, a booting server instance may perform some database migration, importing, or first time initializations. A database lock prevents start actions from conflicting with each other when cluster nodes boot up

concurrently.

The maximum timeout for this lock is 900 seconds. If a node waits on this lock for more than the timeout, the boot fails. The need to change the default value is unlikely, but you can change it by entering this command:

```
bin/kc.[sh|bat] start --spi-dblock-jpa-lock-wait-timeout 900
```

7.8. USING DATABASE VENDORS WITHOUT XA TRANSACTION SUPPORT

Red Hat build of Keycloak uses XA transactions and the appropriate database drivers by default. Certain vendors, such as Azure SQL and MariaDB Galera, do not support or rely on the XA transaction mechanism. To use Keycloak without XA transaction support using the appropriate JDBC driver, enter the following command:

```
bin/kc.[sh|bat] build --db=<vendor> --transaction-xa-enabled=false
```

Red Hat build of Keycloak automatically chooses the appropriate JDBC driver for your vendor.

7.9. SETTING JPA PROVIDER CONFIGURATION OPTION FOR MIGRATIONSTRATEGY

To setup the JPA migrationStrategy (manual/update/validate) you should setup JPA provider as follows:

Setting the migration-strategy for the quarkus provider of the connections-jpa SPI

```
bin/kc.[sh|bat] start --spi-connections-jpa-legacy-migration-strategy=manual
```

If you want to get a SQL file for DB initialization, too, you have to add this additional SPI initializeEmpty (true/false):

Setting the initialize-empty for the quarkus provider of the connections-jpa SPI

```
bin/kc.[sh|bat] start --spi-connections-jpa-legacy-initialize-empty=false
```

In the same way the migrationExport to point to a specific file and location:

Setting the migration-export for the quarkus provider of the connections-jpa SPI

```
bin/kc.[sh|bat] start --spi-connections-jpa-legacy-migration-export=<path>/<file.sql>
```

7.10. RELEVANT OPTIONS

	Value
<p>db</p> <p>The database vendor.</p> <p>CLI: --db Env: KC_DB</p>	<p>dev-file (default), dev-mem, mariadb, mssql, mysql, oracle, postgres</p>
<p>db-driver</p> <p>The fully qualified class name of the JDBC driver.</p> <p>If not set, a default driver is set accordingly to the chosen database.</p> <p>CLI: --db-driver Env: KC_DB_DRIVER</p>	
<p>db-password</p> <p>The password of the database user.</p> <p>CLI: --db-password Env: KC_DB_PASSWORD</p>	
<p>db-pool-initial-size</p> <p>The initial size of the connection pool.</p> <p>CLI: --db-pool-initial-size Env: KC_DB_POOL_INITIAL_SIZE</p>	
<p>db-pool-max-size</p> <p>The maximum size of the connection pool.</p> <p>CLI: --db-pool-max-size Env: KC_DB_POOL_MAX_SIZE</p>	<p>100 (default)</p>
<p>db-pool-min-size</p> <p>The minimal size of the connection pool.</p> <p>CLI: --db-pool-min-size Env: KC_DB_POOL_MIN_SIZE</p>	
<p>db-schema</p> <p>The database schema to be used.</p> <p>CLI: --db-schema Env: KC_DB_SCHEMA</p>	

	Value
<p>db-url</p> <p>The full database JDBC URL.</p> <p>If not provided, a default URL is set based on the selected database vendor. For instance, if using postgres, the default JDBC URL would be jdbc:postgresql://localhost/keycloak.</p> <p>CLI: --db-url Env: KC_DB_URL</p>	
<p>db-url-database</p> <p>Sets the database name of the default JDBC URL of the chosen vendor.</p> <p>If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-database Env: KC_DB_URL_DATABASE</p>	
<p>db-url-host</p> <p>Sets the hostname of the default JDBC URL of the chosen vendor.</p> <p>If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-host Env: KC_DB_URL_HOST</p>	
<p>db-url-port</p> <p>Sets the port of the default JDBC URL of the chosen vendor.</p> <p>If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-port Env: KC_DB_URL_PORT</p>	
<p>db-url-properties</p> <p>Sets the properties of the default JDBC URL of the chosen vendor.</p> <p>Make sure to set the properties accordingly to the format expected by the database vendor, as well as appending the right character at the beginning of this property value. If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-properties Env: KC_DB_URL_PROPERTIES</p>	

Value	
db-username The username of the database user. CLI: --db-username Env: KC_DB_USERNAME	
transaction-xa-enabled ■ If set to false, Keycloak uses a non-XA datasource in case the database does not support XA transactions. CLI: --transaction-xa-enabled Env: KC_TRANSACTION_XA_ENABLED	true (default), false

CHAPTER 8. CONFIGURING DISTRIBUTED CACHES

Red Hat build of Keycloak is designed for high availability and multi-node clustered setups. The current distributed cache implementation is built on top of [Infinispan](#), a high-performance, distributable in-memory data grid.

8.1. ENABLE DISTRIBUTED CACHING

When you start Red Hat build of Keycloak in production mode, by using the **start** command, caching is enabled and all Red Hat build of Keycloak nodes in your network are discovered.

By default, caches are using a **UDP** transport stack so that nodes are discovered using IP multicast transport based on UDP. For most production environments, there are better discovery alternatives to UDP available. Red Hat build of Keycloak allows you to either choose from a set of pre-defined default transport stacks, or to define your own custom stack, as you will see later in this chapter.

To explicitly enable distributed infinispan caching, enter this command:

```
bin/kc.[sh|bat] build --cache=ispn
```

When you start Red Hat build of Keycloak in development mode, by using the **start-dev** command, Red Hat build of Keycloak uses only local caches and distributed caches are completely disabled by implicitly setting the **--cache=local** option. The **local** cache mode is intended only for development and testing purposes.

8.2. CONFIGURING CACHES

Red Hat build of Keycloak provides a cache configuration file with sensible defaults located at **conf/cache-ispn.xml**.

The cache configuration is a regular [Infinispan configuration file](#).

The following table gives an overview of the specific caches Red Hat build of Keycloak uses. You configure these caches in **conf/cache-ispn.xml**:

Cache name	Cache Type	Description
realms	Local	Cache persisted realm data
users	Local	Cache persisted user data
authorization	Local	Cache persisted authorization data
keys	Local	Cache external public keys
work	Replicated	Propagate invalidation messages across nodes

authenticationSessions	Distributed	Caches authentication sessions, created/destroyed/expired during the authentication process
sessions	Distributed	Caches user sessions, created upon successful authentication and destroyed during logout, token revocation, or due to expiration
clientSessions	Distributed	Caches client sessions, created upon successful authentication to a specific client and destroyed during logout, token revocation, or due to expiration
offlineSessions	Distributed	Caches offline user sessions, created upon successful authentication and destroyed during logout, token revocation, or due to expiration
offlineClientSessions	Distributed	Caches client sessions, created upon successful authentication to a specific client and destroyed during logout, token revocation, or due to expiration
loginFailures	Distributed	keep track of failed logins, fraud detection
actionTokens	Distributed	Caches action Tokens

8.2.1. Cache types and defaults

Local caches

Red Hat build of Keycloak caches persistent data locally to avoid unnecessary round-trips to the database.

The following data is kept local to each node in the cluster using local caches:

- **realms** and related data like clients, roles, and groups.
- **users** and related data like granted roles and group memberships.
- **authorization** and related data like resources, permissions, and policies.
- **keys**

Local caches for realms, users, and authorization are configured to hold up to 10,000 entries per default. The local key cache can hold up to 1,000 entries per default and defaults to expire every one hour. Therefore, keys are forced to be periodically downloaded from external clients or identity providers.

In order to achieve an optimal runtime and avoid additional round-trips to the database you should consider looking at the configuration for each cache to make sure the maximum number of entries is aligned with the size of your database. More entries you can cache, less often the server needs to fetch data from the database. You should evaluate the trade-offs between memory utilization and performance.

Invalidation of local caches

Local caching improves performance, but adds a challenge in multi-node setups.

When one Red Hat build of Keycloak node updates data in the shared database, all other nodes need to be aware of it, so they invalidate that data from their caches.

The **work** cache is a replicated cache and used for sending these invalidation messages. The entries/messages in this cache are very short-lived, and you should not expect this cache growing in size over time.

Authentication sessions

Authentication sessions are created whenever a user tries to authenticate. They are automatically destroyed once the authentication process completes or due to reaching their expiration time.

The **authenticationSessions** distributed cache is used to store authentication sessions and any other data associated with it during the authentication process.

By relying on a distributable cache, authentication sessions are available to any node in the cluster so that users can be redirected to any node without losing their authentication state. However, production-ready deployments should always consider session affinity and favor redirecting users to the node where their sessions were initially created. By doing that, you are going to avoid unnecessary state transfer between nodes and improve CPU, memory, and network utilization.

User sessions

Once the user is authenticated, a user session is created. The user session tracks your active users and their state so that they can seamlessly authenticate to any application without being asked for their credentials again. For each application, the user authenticates with a client session is created too, so that the server can track the applications the user is authenticated with and their state on a per-application basis.

User and client sessions are automatically destroyed whenever the user performs a logout, the client performs a token revocation, or due to reaching their expiration time.

The following caches are used to store both user and client sessions:

- sessions
- clientSessions

By relying on a distributable cache, user and client sessions are available to any node in the cluster so that users can be redirected to any node without losing their state. However, production-ready deployments should always consider session affinity and favor redirecting users to the node where their sessions were initially created. By doing that, you are going to avoid unnecessary state transfer between nodes and improve CPU, memory, and network utilization.

As an OpenID Connect Provider, the server is also capable of authenticating users and issuing offline tokens. Similarly to regular user and client sessions, when an offline token is issued by the server upon successful authentication, the server also creates a user and client sessions. However, due to the nature

of offline tokens, offline sessions are handled differently as they are long-lived and should survive a complete cluster shutdown. Because of that, they are also persisted to the database.

The following caches are used to store offline sessions:

- `offlineSessions`
- `offlineClientSessions`

Upon a cluster restart, offline sessions are lazily loaded from the database and kept in a shared cache using the two caches above.

Password brute force detection

The **loginFailures** distributed cache is used to track data about failed login attempts. This cache is needed for the Brute Force Protection feature to work in a multi-node Red Hat build of Keycloak setup.

Action tokens

Action tokens are used for scenarios when a user needs to confirm an action asynchronously, for example in the emails sent by the forgot password flow. The **actionTokens** distributed cache is used to track metadata about action tokens.

8.2.2. Configuring caches for availability

Distributed caches replicate cache entries on a subset of nodes in a cluster and assigns entries to fixed owner nodes.

Each distributed cache has two owners per default, which means that two nodes have a copy of the specific cache entries. Non-owner nodes query the owners of a specific cache to obtain data. When both owner nodes are offline, all data is lost. This situation usually leads to users being logged out at the next request and having to log in again.

The default number of owners is enough to survive 1 node (owner) failure in a cluster setup with at least three nodes. You are free to change the number of owners accordingly to better fit into your availability requirements. To change the number of owners, open **conf/cache-ispn.xml** and change the value for **owners=<value>** for the distributed caches to your desired value.

8.2.3. Specify your own cache configuration file

To specify your own cache configuration file, enter this command:

```
bin/kc.[sh|bat] build --cache-config-file=my-cache-file.xml
```

The configuration file is relative to the **conf/** directory.

8.3. TRANSPORT STACKS

Transport stacks ensure that distributed cache nodes in a cluster communicate in a reliable fashion. Red Hat build of Keycloak supports a wide range of transport stacks:

- `tcp`
- `udp`
- `kubernetes`

- ec2
- azure
- google

To apply a specific cache stack, enter this command:

```
bin/kc.[sh|bat] build --cache-stack=<stack>
```

The default stack is set to **UDP** when distributed caches are enabled.

8.3.1. Available transport stacks

The following table shows transport stacks that are available without any further configuration than using the **--cache-stack** build option:

Stack name	Transport protocol	Discovery
tcp	TCP	MPING (uses UDP multicast).
udp	UDP	UDP multicast

The following table shows transport stacks that are available using the **--cache-stack** build option and a minimum configuration:

Stack name	Transport protocol	Discovery
kubernetes	TCP	DNS_PING (requires -Djgroups.dns.query=<headless-service-FQDN> to be added to JAVA_OPTS or JAVA_OPTS_APPEND environment variable).

8.3.2. Additional transport stacks

The following table shows transport stacks that are supported by Red Hat build of Keycloak, but need some extra steps to work. Note that *none* of these stacks are Kubernetes / OpenShift stacks, so no need exists to enable the "google" stack if you want to run Red Hat build of Keycloak on top of the Google Kubernetes engine. In that case, use the **kubernetes** stack. Instead, when you have a distributed cache setup running on AWS EC2 instances, you would need to set the stack to **ec2**, because ec2 does not support a default discovery mechanism such as **UDP**.

Stack name	Transport protocol	Discovery
ec2	TCP	NATIVE_S3_PING
google	TCP	GOOGLE_PING2

azure	TCP	AZURE_PING
-------	-----	------------

Cloud vendor specific stacks have additional dependencies for Red Hat build of Keycloak. For more information and links to repositories with these dependencies, see the [Infinispan documentation](#).

To provide the dependencies to Red Hat build of Keycloak, put the respective JAR in the **providers** directory and **build** Keycloak by entering this command:

```
bin/kc.[sh|bat] build --cache-stack=<ec2|google|azure>
```

8.3.3. Custom transport stacks

If none of the available transport stacks are enough for your deployment, you are able to change your cache configuration file and define your own transport stack.

For more details, see [Using inline JGroups stacks](#).

defining a custom transport stack

```
<jgroups>
  <stack name="my-encrypt-udp" extends="udp">
    <SSL_KEY_EXCHANGE keystore_name="server.jks"
      keystore_password="password"
      stack.combine="INSERT_AFTER"
      stack.position="VERIFY_SUSPECT2"/>
    <ASYM_ENCRYPT asym_keylength="2048"
      asym_algorithm="RSA"
      change_key_on_coord_leave = "false"
      change_key_on_leave = "false"
      use_external_key_exchange = "true"
      stack.combine="INSERT_BEFORE"
      stack.position="pbcast.NAKACK2"/>
  </stack>
</jgroups>

<cache-container name="keycloak">
  <transport lock-timeout="60000" stack="my-encrypt-udp"/>
  ...
</cache-container>
```

By default, the value set to the **cache-stack** option has precedence over the transport stack you define in the cache configuration file. If you are defining a custom stack, make sure the **cache-stack** option is not used for the custom changes to take effect.

8.4. SECURING CACHE COMMUNICATION

The current Infinispan cache implementation should be secured by various security measures such as RBAC, ACLs, and Transport stack encryption. For more information about securing cache communication, see the [Infinispan security guide](#).

8.5. EXPOSING METRICS FROM CACHES

By default, metrics from caches are not automatically exposed when the metrics are enabled. For more details about how to enable metrics, see [Enabling Red Hat build of Keycloak Metrics](#) .

To enable global metrics for all caches within the **cache-container**, you need to change your cache configuration file (e.g.: **conf/cache-ispn.xml**) to enable **statistics** at the **cache-container** level as follows:

enabling metrics for all caches

```
<cache-container name="keycloak" statistics="true">
  ...
</cache-container>
```

Similarly, you can enable metrics individually for each cache by enabling **statistics** as follows:

enabling metrics for a specific cache

```
<local-cache name="realms" statistics="true">
  ...
</local-cache>
```

8.6. RELEVANT OPTIONS

	Value
<p>cache ■</p> <p>Defines the cache mechanism for high-availability.</p> <p>By default in production mode, a ispn cache is used to create a cluster between multiple server nodes. By default in development mode, a local cache disables clustering and is intended for development and testing purposes.</p> <p>CLI: --cache Env: KC_CACHE</p>	<p>ispn (default), local</p>
<p>cache-config-file ■</p> <p>Defines the file from which cache configuration should be loaded from.</p> <p>The configuration file is relative to the conf/ directory.</p> <p>CLI: --cache-config-file Env: KC_CACHE_CONFIG_FILE</p>	
<p>cache-stack ■</p> <p>Define the default stack to use for cluster communication and node discovery.</p> <p>This option only takes effect if cache is set to ispn. Default: udp.</p> <p>CLI: --cache-stack Env: KC_CACHE_STACK</p>	<p>tcp, udp, kubernetes, ec2, azure, google</p>

CHAPTER 9. CONFIGURING OUTGOING HTTP REQUESTS

Red Hat build of Keycloak often needs to make requests to the applications and services that it secures. Red Hat build of Keycloak manages these outgoing connections using an HTTP client. This chapter shows how to configure the client, connection pool, proxy environment settings, timeouts, and more.

9.1. CLIENT CONFIGURATION COMMAND

The HTTP client that Red Hat build of Keycloak uses for outgoing communication is highly configurable. To configure the Red Hat build of Keycloak outgoing HTTP client, enter this command:

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-<configurationoption>=<value>
```

The following are the command options:

establish-connection-timeout-millis

Maximum time in milliseconds until establishing a connection times out. Default: Not set.

socket-timeout-millis

Maximum time of inactivity between two data packets until a socket connection times out, in milliseconds. Default: 5000ms

connection-pool-size

Size of the connection pool for outgoing connections. Default: 128.

max-pooled-per-route

How many connections can be pooled per host. Default: 64.

connection-ttl-millis

Maximum connection time to live in milliseconds. Default: Not set.

max-connection-idle-time-millis

Maximum time an idle connection stays in the connection pool, in milliseconds. Idle connections will be removed from the pool by a background cleaner thread. Set this option to -1 to disable this check. Default: 900000.

disable-cookies

Enable or disable caching of cookies. Default: true.

client-keystore

File path to a Java keystore file. This keystore contains client certificates for two-way SSL.

client-keystore-password

Password for the client keystore. REQUIRED, when **client-keystore** is set.

client-key-password

Password for the private key of the client. REQUIRED, when client-keystore is set.

proxy-mappings

Specify proxy configurations for outgoing HTTP requests. For more details, see [Section 9.2, "Proxy mappings for outgoing HTTP requests"](#).

disable-trust-manager

If an outgoing request requires HTTPS and this configuration option is set to true, you do not have to specify a truststore. This setting should be used only during development and **never in production** because it will disable verification of SSL certificates. Default: false.

9.2. PROXY MAPPINGS FOR OUTGOING HTTP REQUESTS

To configure outgoing requests to use a proxy, you can use the following standard proxy environment variables to configure the proxy mappings: **HTTP_PROXY**, **HTTPS_PROXY**, and **NO_PROXY**.

- The **HTTP_PROXY** and **HTTPS_PROXY** variables represent the proxy server that is used for outgoing HTTP requests. Red Hat build of Keycloak does not differentiate between the two variables. If you define both variables, **HTTPS_PROXY** takes precedence regardless of the actual scheme that the proxy server uses.
- The **NO_PROXY** variable defines a comma separated list of hostnames that should not use the proxy. For each hostname that you specify, all its subdomains are also excluded from using proxy.

The environment variables can be lowercase or uppercase. Lowercase takes precedence. For example, if you define both **HTTP_PROXY** and **http_proxy**, **http_proxy** is used.

Example of proxy mappings and environment variables

```
HTTPS_PROXY=https://www-proxy.acme.com:8080
NO_PROXY=google.com,login.facebook.com
```

In this example, the following results occur:

- All outgoing requests use the proxy <https://www-proxy.acme.com:8080> except for requests to google.com or any subdomain of google.com, such as auth.google.com.
- login.facebook.com and all its subdomains do not use the defined proxy, but groups.facebook.com uses the proxy because it is not a subdomain of login.facebook.com.

9.3. PROXY MAPPINGS USING REGULAR EXPRESSIONS

An alternative to using environment variables for proxy mappings is to configure a comma-delimited list of proxy-mappings for outgoing requests sent by Red Hat build of Keycloak. A proxy-mapping consists of a regex-based hostname pattern and a proxy-uri, using the format **hostname-pattern;proxy-uri**.

For example, consider the following regex:

```
.*\.(google|googleapis)\.com
```

You apply a regex-based hostname pattern by entering this command:

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-proxy-mappings=".*\.\.
(google|googleapis)\.com;http://www-proxy.acme.com:8080"
```

To determine the proxy for the outgoing HTTP request, the following occurs:

- The target hostname is matched against all configured hostname patterns.
- The proxy-uri of the first matching pattern is used.
- If no configured pattern matches the hostname, no proxy is used.

When your proxy server requires authentication, include the credentials of the proxy user in the format **username:password@**. For example:

```
.*\.(google|googleapis)\.com;http://proxyuser:password@www-proxy.acme.com:8080
```

Example of regular expressions for proxy-mapping:

```
# All requests to Google APIs use http://www-proxy.acme.com:8080 as proxy
.*\.(google|googleapis)\.com;http://www-proxy.acme.com:8080

# All requests to internal systems use no proxy
.*\.acme\.com;NO_PROXY

# All other requests use http://fallback:8080 as proxy
.*;http://fallback:8080
```

In this example, the following occurs:

- The special value NO_PROXY for the proxy-uri is used, which means that no proxy is used for hosts matching the associated hostname pattern.
- A catch-all pattern ends the proxy-mappings, providing a default proxy for all outgoing requests.

9.4. CONFIGURING TRUSTED CERTIFICATES FOR TLS CONNECTIONS

See [Configuring trusted certificates for outgoing requests](#) for how to configure a Red Hat build of Keycloak Truststore so that Red Hat build of Keycloak is able to perform outgoing requests using TLS.

CHAPTER 10. CONFIGURING TRUSTED CERTIFICATES FOR OUTGOING REQUESTS

When Red Hat build of Keycloak communicates with external services through TLS, it has to validate the remote server's certificate in order to ensure it is connecting to a trusted server. This is necessary in order to prevent man-in-the-middle attacks. The certificates of these remote server's or the CA that signed these certificates must be put in a truststore. This truststore is managed by the Keycloak server.

The truststore is used when connecting securely to identity brokers, LDAP identity providers, when sending emails, and for backchannel communication with client applications. It is also useful when you want to change the policy on how host names are verified and trusted by the server.

By default, a truststore provider is not configured, and any TLS/HTTPS connections fall back to standard Java Truststore configuration. If there is no trust established, then these outgoing requests will fail.

10.1. CONFIGURING THE RED HAT BUILD OF KEYCLOAK TRUSTSTORE

You can add your truststore configuration by entering this command:

```
bin/kc.[sh|bat] start --spi-truststore-file-file=myTrustStore.jks --spi-truststore-file-password=password --spi-truststore-file-hostname-verification-policy=ANY
```

The following are possible configuration options for this setting:

file

The path to a Java keystore file. TLS requests need a way to verify the host of the server to which they are talking. This is what the truststore does. The keystore contains one or more trusted host certificates or certificate authorities. This truststore file should only contain public certificates of your secured hosts. This is *REQUIRED* if any of these properties are defined.

password

Password of the keystore. This option is *REQUIRED* if any of these properties are defined.

hostname-verification-policy

For HTTPS requests, this option verifies the hostname of the server's certificate. Default: **WILDCARD**

- **ANY** means that the hostname is not verified.
- **WILDCARD** allows wildcards in subdomain names, such as *.foo.com.
- When using **STRICT**, the Common Name (CN) must match the hostname exactly. Please note that this setting does not apply to LDAP secure connections, which require strict hostname checking.

type

The type of truststore, such as **jks**, **pkcs12** or **bcfks**. If not provided, the type would be detected based on the truststore file extension or platform default type.

10.1.1. Example of a truststore configuration

The following is an example configuration for a truststore that allows you to create trustful connections to all **mycompany.org** domains and its subdomains:

```
bin/kc.[sh|bat] start --spi-truststore-file-file=path/to/truststore.jks --spi-truststore-file-  
password=change_me --spi-truststore-file-hostname-verification-policy=WILDCARD
```

CHAPTER 11. ENABLING AND DISABLING FEATURES

Red Hat build of Keycloak has packed some functionality in features, including some disabled features, such as Technology Preview and deprecated features. Other features are enabled by default, but you can disable them if they do not apply to your use of Red Hat build of Keycloak.

11.1. ENABLING FEATURES

Some supported features, and all preview features, are disabled by default. To enable a feature, enter this command:

```
bin/kc.[sh|bat] build --features="<name>[,<name>]"
```

For example, to enable **docker** and **token-exchange**, enter this command:

```
bin/kc.[sh|bat] build --features="docker,token-exchange"
```

To enable all preview features, enter this command:

```
bin/kc.[sh|bat] build --features="preview"
```

11.2. DISABLING FEATURES

To disable a feature that is enabled by default, enter this command:

```
bin/kc.[sh|bat] build --features-disabled="<name>[,<name>]"
```

For example to disable **impersonation**, enter this command:

```
bin/kc.[sh|bat] build --features-disabled="impersonation"
```

You can disable all default features by entering this command:

```
bin/kc.[sh|bat] build --features-disabled="default"
```

This command can be used in combination with **features** to explicitly set what features should be available. If a feature is added both to the **features-disabled** list and the **features** list, it will be enabled.

11.3. SUPPORTED FEATURES

The following list contains supported features that are enabled by default, and can be disabled if not needed.

account-api

Account Management REST API

account2

Account Management Console version 2

admin-api

Admin API

admin2

New Admin Console

authorization

Authorization Service

ciba

OpenID Connect Client Initiated Backchannel Authentication (CIBA)

client-policies

Client configuration policies

impersonation

Ability for admins to impersonate users

js-adapter

Host keycloak.js and keycloak-authz.js through the Keycloak server

kerberos

Kerberos

par

OAuth 2.0 Pushed Authorization Requests (PAR)

step-up-authentication

Step-up Authentication

web-authn

W3C Web Authentication (WebAuthn)

11.3.1. Disabled by default

The following list contains supported features that are disabled by default, and can be enabled if needed.

docker

Docker Registry protocol

fips

FIPS 140-2 mode

11.4. PREVIEW FEATURES

Preview features are disabled by default and are not recommended for use in production. These features may change or be removed at a future release.

account3

Account Management Console version 3

admin-fine-grained-authz

Fine-Grained Admin Permissions

client-secret-rotation

Client Secret Rotation

declarative-user-profile

Configure user profiles using a declarative style

multi-site

Multi-site support

recovery-codes

Recovery codes

scripts

Write custom authenticators using JavaScript

token-exchange

Token Exchange Service

update-email

Update Email Action

11.5. DEPRECATED FEATURES

The following list contains deprecated features that will be removed in a future release. These features are disabled by default.

linkedin-oauth

LinkedIn Social Identity Provider based on OAuth

11.6. RELEVANT OPTIONS

	Value
<p>features ■</p> <p>Enables a set of one or more features.</p> <p>CLI: --features Env: KC_FEATURES</p>	<p>account-api, account2, account3, admin-api, admin-fine-grained-authz, admin2, authorization, ciba, client-policies, client-secret-rotation, declarative-user-profile, docker, dynamic-scopes, fips, impersonation, js-adapter, kerberos, linkedin-oauth, map-storage, multi-site, par, preview, recovery-codes, scripts, step-up-authentication, token-exchange, update-email, web-authn</p>

	Value
<p>features-disabled ■</p> <p>Disables a set of one or more features.</p> <p>CLI: --features-disabled Env: KC_FEATURES_DISABLED</p>	<p>account-api, account2, account3, admin-api, admin-fine-grained-authz, admin2, authorization, ciba, client-policies, client-secret-rotation, declarative-user-profile, docker, dynamic-scopes, fips, impersonation, js-adapter, kerberos, linkedin-oauth, map-storage, multi-site, par, preview, recovery-codes, scripts, step-up-authentication, token-exchange, update-email, web-authn</p>

CHAPTER 12. CONFIGURING PROVIDERS

The server is built with extensibility in mind and for that it provides a number of Service Provider Interfaces or SPIs, each one responsible for providing a specific capability to the server. In this chapter, you are going to understand the core concepts around the configuration of SPIs and their respective providers.

After reading this chapter, you should be able to use the concepts and the steps herein explained to install, uninstall, enable, disable, and configure any provider, including those you have implemented to extend the server capabilities in order to better fulfill your requirements.

12.1. CONFIGURATION OPTION FORMAT

Providers can be configured by using a specific configuration format. The format consists of:

```
spi-<spi-id>-<provider-id>-<property>=<value>
```

The **<spi-id>** is the name of the SPI you want to configure.

The **<provider-id>** is the id of the provider you want to configure. This is the id set to the corresponding provider factory implementation.

The **<property>** is the actual name of the property you want to set for a given provider.

All those names (for spi, provider, and property) should be in lower case and if the name is in camel-case such as **myKeycloakProvider**, it should include dashes (-) before upper-case letters as follows: **my-keycloak-provider**.

Taking the **HttpClientSpi** SPI as an example, the name of the SPI is **connectionsHttpClient** and one of the provider implementations available is named **default**. In order to set the **connectionPoolSize** property you would use a configuration option as follows:

```
spi-connections-http-client-default-connection-pool-size=10
```

12.2. SETTING A PROVIDER CONFIGURATION OPTION

Provider configuration options are provided when starting the server, as shown in the following example:

Setting the connection-pool-size for the default provider of the connections-http-client SPI

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-connection-pool-size=10
```

12.3. CONFIGURING A DEFAULT PROVIDER

Depending on the SPI, multiple provider implementations can co-exist but only one of them is going to be used at runtime. For these SPIs, a default provider is the primary implementation that is going to be active and used at runtime.

To configure a provider as the default you should run the **build** command as follows:

Marking the mycustomprovider provider as the default provider for the email-template SPI

```
bin/kc.[sh|bat] build --spi-email-template-provider=mycustomprovider
```

In the example above, we are using the **provider** property to set the id of the provider we want to mark as the default.

12.4. ENABLING AND DISABLING A PROVIDER

To enable or disable a provider you should run the **build** command as follows:

Enabling a provider

```
bin/kc.[sh|bat] build --spi-email-template-mycustomprovider-enabled=true
```

To disable a provider, use the same command and set the **enabled** property to **false**.

12.5. INSTALLING AND UNINSTALLING A PROVIDER

Custom providers should be packaged in a Java Archive (JAR) file and copied to the **providers** directory of the distribution. After that, you must run the **build** command in order to update the server's provider registry with the implementations from the JAR file.

This step is needed in order to optimize the server runtime so that all providers are known ahead-of-time rather than discovered only when starting the server or at runtime.

To uninstall a provider, you should remove the JAR file from the **providers** directory and run the **build** command again.

12.6. USING THIRD-PARTY DEPENDENCIES

When implementing a provider you might need to use some third-party dependency that is not available from the server distribution.

In this case, you should copy any additional dependency to the **providers** directory and run the **build** command. Once you do that, the server is going to make these additional dependencies available at runtime for any provider that depends on them.

12.7. REFERENCES

- [Configuring Red Hat build of Keycloak](#)
- [Server Developer Documentation](#)

CHAPTER 13. CONFIGURING LOGGING

Red Hat build of Keycloak uses the JBoss Logging framework. The following is a high-level overview for the available log handlers:

- root
 - console (*default*)
 - file

13.1. LOGGING CONFIGURATION

Logging is done on a per-category basis in Red Hat build of Keycloak. You can configure logging for the root log level or for more specific categories such as **org.hibernate** or **org.keycloak**. This chapter describes how to configure logging.

13.1.1. Log levels

The following table defines the available log levels.

Level	Description
FATAL	Critical failures with complete inability to serve any kind of request.
ERROR	A significant error or problem leading to the inability to process requests.
WARN	A non-critical error or problem that might not require immediate correction.
INFO	Red Hat build of Keycloak lifecycle events or important information. Low frequency.
DEBUG	More detailed information for debugging purposes, such as database logs. Higher frequency.
TRACE	Most detailed debugging information. Very high frequency.
ALL	Special level for all log messages.
OFF	Special level to turn logging off entirely (not recommended).

13.1.2. Configuring the root log level

When no log level configuration exists for a more specific category logger, the enclosing category is used instead. When there is no enclosing category, the root logger level is used.

To set the root log level, enter the following command:

```
bin/kc.[sh|bat] start --log-level=<root-level>
```

Use these guidelines for this command:

- For **<root-level>**, supply a level defined in the preceding table.
- The log level is case-insensitive. For example, you could either use **DEBUG** or **debug**.
- If you were to accidentally set the log level twice, the last occurrence in the list becomes the log level. For example, if you included the syntax **--log-level="info,...,DEBUG,..."**, the root logger would be **DEBUG**.

13.1.3. Configuring category-specific log levels

You can set different log levels for specific areas in Red Hat build of Keycloak. Use this command to provide a comma-separated list of categories for which you want a different log level:

```
bin/kc.[sh|bat] start --log-level="<root-level>,<org.category1>:<org.category1-level>"
```

A configuration that applies to a category also applies to its sub-categories unless you include a more specific matching sub-category.

Example

```
bin/kc.[sh|bat] start --log-level="INFO,org.hibernate:debug,org.hibernate.hql.internal.ast:info"
```

This example sets the following log levels:

- Root log level for all loggers is set to INFO.
- The hibernate log level in general is set to debug.
- To keep SQL abstract syntax trees from creating verbose log output, the specific subcategory **org.hibernate.hql.internal.ast** is set to info. As a result, the SQL abstract syntax trees are omitted instead of appearing at the **debug** level.

13.2. ENABLING LOG HANDLERS

To enable log handlers, enter the following command:

```
bin/kc.[sh|bat] start --log="<handler1>,<handler2>"
```

The available handlers are **console** and **file**. The more specific handler configuration mentioned below will only take effect when the handler is added to this comma-separated list.

13.3. CONSOLE LOG HANDLER

The console log handler is enabled by default, providing unstructured log messages for the console.

13.3.1. Configuring the console log format

Red Hat build of Keycloak uses a pattern-based logging formatter that generates human-readable text logs by default.

The logging format template for these lines can be applied at the root level. The default format template is:

- `%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n`

The format string supports the symbols in the following table:

Symbol	Summary	Description
%%	%	Renders a simple % character.
%c	Category	Renders the log category name.
%d{xxx}	Date	Renders a date with the given date format string.String syntax defined by java.text.SimpleDateFormat
%e	Exception	Renders a thrown exception.
%h	Hostname	Renders the simple host name.
%H	Qualified host name	Renders the fully qualified hostname, which may be the same as the simple host name, depending on the OS configuration.
%i	Process ID	Renders the current process PID.
%m	Full Message	Renders the log message and an exception, if thrown.
%n	Newline	Renders the platform-specific line separator string.
%N	Process name	Renders the name of the current process.
%p	Level	Renders the log level of the message.
%r	Relative time	Render the time in milliseconds since the start of the application log.
%s	Simple message	Renders only the log message without exception trace.

%t	Thread name	Renders the thread name.
%t{id}	Thread ID	Render the thread ID.
%z{<zone name>}	Timezone	Set the time zone of log output to <zone name>.
%L	Line number	Render the line number of the log message.

13.3.2. Setting the logging format

To set the logging format for a logged line, perform these steps:

1. Build your desired format template using the preceding table.
2. Enter the following command:

```
bin/kc.[sh|bat] start --log-console-format="<format>"
```

Note that you need to escape characters when invoking commands containing special shell characters such as `;` using the CLI. Therefore, consider setting it in the configuration file instead.

Example: Abbreviate the fully qualified category name

```
bin/kc.[sh|bat] start --log-console-format="%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t)
%s%e%n"
```

This example abbreviates the category name to three characters by setting `[%c{3.}]` in the template instead of the default `[%c]`.

13.3.3. Configuring JSON or plain console logging

By default, the console log handler logs plain unstructured data to the console. To use structured JSON log output instead, enter the following command:

```
bin/kc.[sh|bat] start --log-console-output=json
```

Example Log Message

```
{"timestamp":"2022-02-
25T10:31:32.452+01:00","sequence":8442,"loggerClassName":"org.jboss.logging.Logger","loggerName":"io.quarkus","level":"INFO","message":"Keycloak 18.0.0-SNAPSHOT on JVM (powered by
Quarkus 2.7.2.Final) started in 3.253s. Listening on:
http://0.0.0.0:8080","threadName":"main","threadId":1,"mdc":{},"ndc":"","hostName":"host-
name","processName":"QuarkusEntryPoint","processId":36946}
```

When using JSON output, colors are disabled and the format settings set by **--log-console-format** will not apply.

To use unstructured logging, enter the following command:

```
bin/kc.[sh|bat] start --log-console-output=default
```

Example Log Message:

```
2022-03-02 10:36:50,603 INFO [io.quarkus] (main) Keycloak 18.0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.2.Final) started in 3.615s. Listening on: http://0.0.0.0:8080
```

13.3.4. Colors

Colored console log output for unstructured logs is disabled by default. Colors may improve readability, but they can cause problems when shipping logs to external log aggregation systems. To enable or disable color-coded console log output, enter following command:

```
bin/kc.[sh|bat] start --log-console-color=<false|true>
```

13.4. FILE LOGGING

As an alternative to logging to the console, you can use unstructured logging to a file.

13.4.1. Enable file logging

Logging to a file is disabled by default. To enable it, enter the following command:

```
bin/kc.[sh|bat] start --log="console,file"
```

A log file named **keycloak.log** is created inside the **data/log** directory of your Keycloak installation.

13.4.2. Configuring the location and name of the log file

To change where the log file is created and the file name, perform these steps:

1. Create a writable directory to store the log file.
If the directory is not writable, Red Hat build of Keycloak will start correctly, but it will issue an error and no log file will be created.
2. Enter this command:

```
bin/kc.[sh|bat] start --log="console,file" --log-file=<path-to>/<your-file.log>
```

13.4.3. Configuring the file handler format

To configure a different logging format for the file log handler, enter the following command:

```
bin/kc.[sh|bat] start --log-file-format="<pattern>"
```

See [Section 13.3.1, “Configuring the console log format”](#) for more information and a table of the available pattern configuration.

13.5. RELEVANT OPTIONS

	Value
<p>log-console-color</p> <p>Enable or disable colors when logging to console.</p> <p>CLI: --log-console-color Env: KC_LOG_CONSOLE_COLOR</p>	<p>true, false (default)</p>
<p>log-console-format</p> <p>The format of unstructured console log entries.</p> <p>If the format has spaces in it, escape the value using "<format>".</p> <p>CLI: --log-console-format Env: KC_LOG_CONSOLE_FORMAT</p>	<p>%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n (default)</p>
<p>log-console-output</p> <p>Set the log output to JSON or default (plain) unstructured logging.</p> <p>CLI: --log-console-output Env: KC_LOG_CONSOLE_OUTPUT</p>	<p>default (default), json</p>
<p>log-file</p> <p>Set the log file path and filename.</p> <p>CLI: --log-file Env: KC_LOG_FILE</p>	<p>data/log/keycloak.log (default)</p>
<p>log-file-format</p> <p>Set a format specific to file log entries.</p> <p>CLI: --log-file-format Env: KC_LOG_FILE_FORMAT</p>	<p>%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n (default)</p>
<p>log-file-output</p> <p>Set the log output to JSON or default (plain) unstructured logging.</p> <p>CLI: --log-file-output Env: KC_LOG_FILE_OUTPUT</p>	<p>default (default), json</p>

		Value
log-level		info (default)
<p>The log level of the root category or a comma-separated list of individual categories and their levels.</p> <p>For the root category, you don't need to specify a category.</p>		
CLI: --log-level		
Env: KC_LOG_LEVEL		

CHAPTER 14. FIPS 140-2 SUPPORT

The Federal Information Processing Standard Publication 140-2, (FIPS 140-2), is a U.S. government computer security standard used to approve cryptographic modules. Red Hat build of Keycloak supports to run in FIPS 140-2 compliant mode. In this case, Red Hat build of Keycloak will use only FIPS approved cryptography algorithms for its functionality.

To run in FIPS 140-2, Red Hat build of Keycloak should run on a FIPS 140-2 enabled system. This requirement usually assumes RHEL or Fedora where FIPS was enabled during installation. See [RHEL documentation](#) for the details. When the system is in FIPS mode, it makes sure that the underlying OpenJDK is in FIPS mode as well and would use only [FIPS enabled security providers](#).

To check that the system is in FIPS mode, you can check it with the following command from the command line:

```
fips-mode-setup --check
```

If the system is not in FIPS mode, you can enable it with the following command, however it is recommended that system is in FIPS mode since the installation rather than later enabled this way:

```
fips-mode-setup --enable
```

14.1. BOUNCYCASTLE LIBRARY

Red Hat build of Keycloak internally uses the BouncyCastle library for many cryptography utilities. However, the default flavor of the BouncyCastle library that shipped with Red Hat build of Keycloak is not FIPS compliant, but, BouncyCastle also provides a FIPS validated version of its library. The FIPS validated BouncyCastle library cannot be shipped with Red Hat build of Keycloak due to license constraints and Red Hat build of Keycloak cannot provide official support of it. Therefore, to run in FIPS compliant mode, you need to download BouncyCastle-FIPS bits and add them to the Red Hat build of Keycloak distribution. When Red Hat build of Keycloak executes in fips-mode, it will use the BCFIPS bits instead of the default BouncyCastle bits, which achieves FIPS compliance.

14.1.1. BouncyCastle FIPS bits

BouncyCastle FIPS can be downloaded from the [BouncyCastle official page](#). Then you can add them to the directory **KEYCLOAK_HOME/providers** of your distribution. Make sure to use proper versions compatible with BouncyCastle Keycloak dependencies. The supported BCFIPS bits needed are:

- **bc-fips-1.0.2.3.jar**
- **bctls-fips-1.0.16.jar**
- **bcpkix-fips-1.0.7.jar**

14.2. GENERATING KEYSTORE

You can create either **pkcs12** or **bcfks** keystore to be used for the Red Hat build of Keycloak server SSL.

14.2.1. PKCS12 keystore

The **p12** (or **pkcs12**) keystore (and/or truststore) works well in BCFIPS non-approved mode.

PKCS12 keystore can be generated with OpenJDK 17 Java on RHEL 9 in the standard way. For instance this command can be used to generate such keystore:

```
keytool -genkeypair -sigalg SHA512withRSA -keyalg RSA -storepass passwordpassword \
  -keystore $KEYCLOAK_HOME/conf/server.keystore \
  -alias localhost \
  -dname CN=localhost -keypass passwordpassword
```

When system is in FIPS mode, the default **java.security** file is changed in order to use FIPS enabled security providers, so no additional configuration is needed. Additionally, in PKCS12 keystore you can store PBE (password-based encryption) keys simply via the keytool command, which makes it ideal for using it with Red Hat build of Keycloak KeyStore Vault and/or to store configuration properties in the KeyStore Config Source. For more details, see the [Configuring Red Hat build of Keycloak](#) and the [Using a vault](#).

14.2.2. BCFKS keystore

BCFKS keystore generation requires the use of the BouncyCastle FIPS libraries and a custom security file.

You can start with create a helper file, such as **/tmp/kc.keystore-create.java.security**. The content of the file needs only to have the following property:

```
securerandom.strongAlgorithms=PKCS11:SunPKCS11-NSS-FIPS
```

Next enter a command such as the following to generate the keystore:

```
keytool -keystore $KEYCLOAK_HOME/conf/server.keystore \
  -storetype bcfks \
  -providertype BCFIPS \
  -providerclass org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider \
  -provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider \
  -providerpath $KEYCLOAK_HOME/providers/bc-fips-*.jar \
  -alias localhost \
  -genkeypair -sigalg SHA512withRSA -keyalg RSA -storepass passwordpassword \
  -dname CN=localhost -keypass passwordpassword \
  -J-Djava.security.properties=/tmp/kc.keystore-create.java.security
```



WARNING

Using self-signed certificates is for demonstration purposes only, so replace these certificates with proper certificates when you move to a production environment.

Similar options are needed when you are doing any other manipulation with keystore/truststore of **bcfks** type.

14.3. RUNNING THE SERVER.

To run the server with BCFIPS in non-approved mode, enter the following command

```
bin/kc.[sh|bat] start --features=fips --hostname=localhost --https-key-store-
password=passwordpassword --log-
level=INFO,org.keycloak.common.crypto:TRACE,org.keycloak.crypto:TRACE
```



NOTE

In non-approved mode, the default keystore type (as well as default truststore type) is PKCS12. Hence if you generated BCFKS keystore as described above, it is also needed to use command **--https-key-store-type=bcfks**. Similar might be needed for the truststore as well if you want to use it.



NOTE

You can disable logging in production if everything works as expected.

14.4. STRICT MODE

There is the **fips-mode** option, which is automatically set to **non-strict** when the **fips** feature is enabled. This means to run BCFIPS in the "non-approved mode". The more secure alternative is to use **--features=fips --fips-mode=strict** in which case BouncyCastle FIPS will use "approved mode". Using that option results in stricter security requirements on cryptography and security algorithms.



NOTE

In strict mode, the default keystore type (as well as default truststore type) is BCFKS. If you want to use different keystore type it is needed to use option **--https-key-store-type** with appropriate type. Similar might be needed for the truststore as well if you want to use it.

When starting server, you can check that the startup log contains **KC** provider with the note about **Approved Mode** such as the following:

```
KC(BCFIPS version 1.000203 Approved Mode, FIPS-JVM: enabled) version 1.0 - class
org.keycloak.crypto.fips.KeycloakFipsSecurityProvider,
```

14.4.1. Cryptography restrictions in strict mode

- As mentioned in the previous section, strict mode may not work with **pkcs12** keystore. It is needed to use another keystore (like **bcfks**) as mentioned earlier. Also **jks** and **pkcs12** keystores are not supported in Red Hat build of Keycloak when using strict mode. Some examples are importing or generating a keystore of an OIDC or SAML client in the Admin Console or for a **java-keystore** provider in the realm keys.
- User passwords must be 14 characters or longer. Red Hat build of Keycloak uses PBKDF2 based password encoding by default. BCFIPS approved mode requires passwords to be at least 112 bits (effectively 14 characters) with PBKDF2 algorithm. If you want to allow a shorter password, set the property **max-padding-length** of provider **pbkdf2-sha256** of SPI **password-hashing** to value 14 to provide additional padding when verifying a hash created by this algorithm. This setting is also backwards compatible with previously stored passwords. For example, if the user's

database is in a non-FIPS environment and you have shorter passwords and you want to verify them now with Red Hat build of Keycloak using BCFIPS in approved mode, the passwords should work. So effectively, you can use an option such as the following when starting the server:

```
--spi-password-hashing-pbkdf2-sha256-max-padding-length=14
```



NOTE

Using the option above does not break FIPS compliance. However, note that longer passwords are good practice anyway. For example, passwords auto-generated by modern browsers match this requirement as they are longer than 14 characters.

- RSA keys of 1024 bits do not work (2048 is the minimum). This applies for keys used by the Red Hat build of Keycloak realm itself (Realm keys from the **Keys** tab in the admin console), but also client keys and IDP keys
- HMAC SHA-XXX keys must be at least 112 bits (or 14 characters long). For example if you use OIDC clients with the client authentication **Signed Jwt with Client Secret** (or **client-secret-jwt** in the OIDC notation), then your client secrets should be at least 14 characters long. Note that for good security, it is recommended to use client secrets generated by Red Hat build of Keycloak server, which always match this requirement.

14.5. OTHER RESTRICTIONS

To have SAML working, make sure that a **XMLDSig** security provider is available in your security providers. To have Kerberos working, make sure that a **SunJGSS** security provider is available. In FIPS enabled RHEL 9 in OpenJDK 17.0.6, these security providers are not present in the **java.security**, which means that they effectively cannot work.

To have SAML working, you can manually add the provider into **JAVA_HOME/conf/security/java.security** into the list fips providers. For example, add the line such as the following:

```
fips.provider.7=XMLDSig
```

Adding this security provider should work well. In fact, it is FIPS compliant and likely will be added by default in the future OpenJDK 17 micro version. Details are in the [bugzilla](#).



NOTE

It is recommended to look at **JAVA_HOME/conf/security/java.security** and check all configured providers here and make sure that the number matches. In other words, **fips.provider.7** assumes that there are already 6 providers configured with prefix like **fips.provider.N** in this file.

If prefer not to edit your **java.security** file inside java itself, you can create a custom java security file (for example named **kc.java.security**) and add only the single property above for adding XMLDSig provider into that file. Then start your Red Hat build of Keycloak server with this property file attached:

```
-Djava.security.properties=/location/to/your/file/kc.java.security
```

For Kerberos/SPNEGO, the security provider **SunJGSS** is not yet fully FIPS compliant. Hence it is not recommended to add it to your list of security providers if you want to be FIPS compliant. The

KERBEROS feature is disabled by default in Red Hat build of Keycloak when it is executed on FIPS platform and when security provider is not available. Details are in the [bugzilla](#).

14.6. RUN THE CLI ON THE FIPS HOST

If you want to run Client Registration CLI (**kcreg.sh|bat** script) or Admin CLI (**kcadm.sh|bat** script), the CLI must also use the BouncyCastle FIPS dependencies instead of plain BouncyCastle dependencies. To achieve this, you may copy the jars to the CLI library folder and that is enough. CLI tool will automatically use BCFIPS dependencies instead of plain BC when it detects that corresponding BCFIPS jars are present (see above for the versions used). For example, use command such as the following before running the CLI:

```
cp $KEYCLOAK_HOME/providers/bc-fips-*.jar $KEYCLOAK_HOME/bin/client/lib/
cp $KEYCLOAK_HOME/providers/bctls-fips-*.jar $KEYCLOAK_HOME/bin/client/lib/
```



NOTE

When trying to use BCFKS truststore/keystore with CLI, you may see issues due this truststore is not the default java keystore type. It can be good to specify it as default in java security properties. For example run this command on unix based systems before doing any operation with kcadm|kcreg clients:

```
echo "keystore.type=bcfks
fips.keystore.type=bcfks" > /tmp/kcadm.java.security
export KC_OPTS="-Djava.security.properties=/tmp/kcadm.java.security"
```

14.7. RED HAT BUILD OF KEYCLOAK SERVER IN FIPS MODE IN THE CONTAINER

When you want Red Hat build of Keycloak in FIPS mode to be executed inside a container, your "host" must be using FIPS mode as well. The container will then "inherit" FIPS mode from the parent host. See [this section](#) in the RHEL documentation for the details.

The Red Hat build of Keycloak container image will be just automatically in fips mode when executed from the host in FIPS mode. However, make sure that the Red Hat build of Keycloak container also uses BCFIPS jars (instead of BC jars) and proper options when started.

Regarding this, it is best to build your own container image as described in the [Running Red Hat build of Keycloak in a container](#) and tweak it to use BCFIPS etc.

For example in the current directory, you can create sub-directory **files** and add:

- BC FIPS jar files as described above
- Custom keystore file - named for example **keycloak-fips.keystore.bcfks**
- Security file **kc.java.security** with added provider for SAML

Then create **Dockerfile** in the current directory similar to this:

Dockerfile:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:22 as builder
```

```
ADD files /tmp/files/
```

```
WORKDIR /opt/keycloak
```

```
RUN cp /tmp/files/*.jar /opt/keycloak/providers/
```

```
RUN cp /tmp/files/keycloak-fips.keystore.* /opt/keycloak/conf/server.keystore
```

```
RUN cp /tmp/files/kc.java.security /opt/keycloak/conf/
```

```
RUN /opt/keycloak/bin/kc.sh build --features=fips --fips-mode=strict
```

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:22
```

```
COPY --from=builder /opt/keycloak/ /opt/keycloak/
```

```
ENTRYPOINT ["/opt/keycloak/bin/kc.sh"]
```

Then build FIPS an optimized docker image and start it as described in the [Running Red Hat build of Keycloak in a container](#). These steps require that you use arguments as described above when starting the image.

14.8. MIGRATION FROM NON-FIPS ENVIRONMENT

If you previously used Red Hat build of Keycloak in the non-fips environment, it is possible to migrate it to FIPS environment including its data. However, restrictions and considerations exist as mentioned in previous sections. To highlight some of them:

- Make sure all the Red Hat build of Keycloak functionality relying on keystores uses only supported keystore types. This differs based on whether strict or non-strict mode is used.
- Kerberos authentication may not work. If your authentication flow uses **Kerberos** authenticator, this authenticator will be automatically switched to **DISABLED** when migrated to FIPS environment. It is recommended to remove any **Kerberos** user storage providers from your realm and disable **Kerberos** related functionality in LDAP providers before switching to FIPS environment.

In addition to the preceding requirements, be sure to doublecheck this before switching to FIPS strict mode:

- Make sure that all the Red Hat build of Keycloak functionality relying on keys (for example, realm or client keys) use RSA keys of at least 2048 bits
- Make sure that clients relying on **Signed JWT with Client Secret** use at least 14 characters long secrets (ideally generated secrets)
- Password length restriction as described earlier. In case your users have shorter passwords, be sure to start the server with the max padding length set to 14 of PBKDF2 provider as mentioned earlier. If you prefer to avoid this option, you can for instance ask all your users to reset their password (for example by the **Forget password** link) during the first authentication in the new environment.

14.9. RED HAT BUILD OF KEYCLOAK FIPS MODE ON THE NON-FIPS SYSTEM

Red Hat build of Keycloak is supported and tested on a FIPS enabled RHEL 8 system and **ubi8** image. It is supported with RHEL 9 (and **ubi9** image) as well. Running on the non-RHEL compatible platform or on the non-FIPS enabled platform, the FIPS compliance cannot be strictly guaranteed and cannot be

officially supported.

If you are still restricted to run Red Hat build of Keycloak on such a system, you can at least update your security providers configured in **java.security** file. This update does not mean FIPS compliance, but at least the setup is closer to it. It can be done by providing a custom security file with only an overridden list of security providers as described earlier. For a list of recommended providers, see the [OpenJDK 17 documentation](#).

You can check the Red Hat build of Keycloak server log at startup to see if the correct security providers are used. TRACE logging should be enabled for crypto-related Keycloak packages as described in the Keycloak startup command earlier.

CHAPTER 15. ENABLING RED HAT BUILD OF KEYCLOAK HEALTH CHECKS

Red Hat build of Keycloak has built in support for health checks. This chapter describes how to enable and use the Keycloak health checks.

15.1. RED HAT BUILD OF KEYCLOAK HEALTH CHECKS

Red Hat build of Keycloak exposed health endpoints are three:

- `/health`
- `/health/live`
- `/health/ready`

The result is returned in json format and it looks as follows:

```
{
  "status": "UP",
  "checks": []
}
```

15.2. ENABLING THE HEALTH CHECKS

It is possible to enable the health checks using the build time option **health-enabled**:

```
bin/kc.[sh|bat] build --health-enabled=true
```

By default, no check is returned from the health endpoints.

15.3. USING THE HEALTH CHECKS

It is recommended that the health endpoints be monitored by external HTTP requests. Due to security measures that remove **curl** and other packages from the Red Hat build of Keycloak container image, local command-based monitoring will not function easily.

If you are not using Red Hat build of Keycloak in a container, use whatever you want to access the health check endpoints.

15.3.1. curl

You may use a simple HTTP HEAD request to determine the **live** or **ready** state of Red Hat build of Keycloak. **curl** is a good HTTP client for this purpose.

If Red Hat build of Keycloak is deployed in a container, you must run this command from outside it due to the previously mentioned security measures. For example:

```
curl --head -fsS http://localhost:8080/health/ready
```

If the command returns with status 0, then Red Hat build of Keycloak is **live** or **ready**, depending on which endpoint you called. Otherwise there is a problem.

15.3.2. Kubernetes

Define a [HTTP Probe](#) so that Kubernetes may externally monitor the health endpoints. Do not use a liveness command.

15.3.3. HEALTHCHECK

The Dockerfile image **HEALTHCHECK** instruction defines a command that will be periodically executed inside the container as it runs. The Red Hat build of Keycloak container does not have any CLI HTTP clients installed. Consider installing **curl** as an additional RPM, as detailed by the [Running Red Hat build of Keycloak in a container](#) chapter. Note that your container may be less secure because of this.

15.4. AVAILABLE CHECKS

The table below shows the available checks.

Check	Description	Requires Metrics
Database	Returns the status of the database connection pool.	Yes

For some checks, you'll need to also enable metrics as indicated by the **Requires Metrics** column. To enable metrics use the **metrics-enabled** option as follows:

```
bin/kc.[sh|bat] build --health-enabled=true --metrics-enabled=true
```

15.5. RELEVANT OPTIONS

	Value
health-enabled  If the server should expose health check endpoints. If enabled, health checks are available at the /health , /health/ready and /health/live endpoints. CLI: --health-enabled Env: KC_HEALTH_ENABLED	true, false (default)

CHAPTER 16. ENABLING RED HAT BUILD OF KEYCLOAK METRICS

Red Hat build of Keycloak has built in support for metrics. This chapter describes how to enable and configure server metrics.

16.1. ENABLING METRICS

It is possible to enable metrics using the build time option **metrics-enabled**:

```
bin/kc.[sh|bat] start --metrics-enabled=true
```

16.2. QUERYING METRICS

Red Hat build of Keycloak exposes metrics at the following endpoint:

- **/metrics**

The response from the endpoint uses a **application/openmetrics-text** content type and it is based on the Prometheus (OpenMetrics) text format. The snippet bellow is an example of a response:

```
# HELP base_gc_total Displays the total number of collections that have occurred. This attribute lists
-1 if the collection count is undefined for this collector.
# TYPE base_gc_total counter
base_gc_total{name="G1 Young Generation",} 14.0
# HELP jvm_memory_usage_after_gc_percent The percentage of long-lived heap pool used after the
last GC event, in the range [0..1]
# TYPE jvm_memory_usage_after_gc_percent gauge
jvm_memory_usage_after_gc_percent{area="heap",pool="long-lived",} 0.0
# HELP jvm_threads_peak_threads The peak live thread count since the Java virtual machine
started or peak was reset
# TYPE jvm_threads_peak_threads gauge
jvm_threads_peak_threads 113.0
# HELP agroal_active_count Number of active connections. These connections are in use and not
available to be acquired.
# TYPE agroal_active_count gauge
agroal_active_count{datasource="default",} 0.0
# HELP base_memory_maxHeap_bytes Displays the maximum amount of memory, in bytes, that
can be used for memory management.
# TYPE base_memory_maxHeap_bytes gauge
base_memory_maxHeap_bytes 1.6781410304E10
# HELP process_start_time_seconds Start time of the process since unix epoch.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.675188449054E9
# HELP system_load_average_1m The sum of the number of runnable entities queued to available
processors and the number of runnable entities running on the available processors averaged over a
period of time
# TYPE system_load_average_1m gauge
system_load_average_1m 4.005859375

...
```

16.3. AVAILABLE METRICS

The table below summarizes the available metrics groups:

Metric	Description
System	A set of system-level metrics related to CPU and memory usage.
JVM	A set of metrics from the Java Virtual Machine (JVM) related to GC, and heap.
Database	A set of metrics from the database connection pool, if using a database.
Cache	A set of metrics from Infinispan caches. See Configuring distributed caches for more details.

16.4. RELEVANT OPTIONS

	Value
<p>metrics-enabled </p> <p>If the server should expose metrics.</p> <p>If enabled, metrics are available at the /metrics endpoint.</p> <p>CLI: --metrics-enabled Env: KC_METRICS_ENABLED</p>	<p>true, false (default)</p>

CHAPTER 17. IMPORTING AND EXPORTING REALMS

In this chapter, you are going to understand the different approaches for importing and exporting realms using JSON files.



NOTE

Exporting and importing into single files can produce large files, so if your database contains more than 500 users, export to a directory and not a single file. Using a directory performs better as the directory provider uses a separate transaction for each "page" (a file of users). The default count of users per file and per transaction is fifty. Increasing this to a larger number leads to an exponentially increasing execution time.

17.1. PROVIDING OPTIONS FOR DATABASE CONNECTION PARAMETERS

When using the **export** and the **import** commands below, Red Hat build of Keycloak needs to know how to connect to the database where the information about realms, clients, users and other entities is stored. As described in [Configuring Red Hat build of Keycloak](#) that information can be provided as command line parameters, environment variables or a configuration file. Use the **--help** command line option for each command to see the available options.

Some of the configuration options are build time configuration options. As default, Red Hat build of Keycloak will re-build automatically for the **export** and **import** commands if it detects a change of a build time parameter.

If you have built an optimized version of Red Hat build of Keycloak with the **build** command as outlined in [Configuring Red Hat build of Keycloak](#), use the command line option **--optimized** to have Keycloak skip the build check for a faster startup time. When doing this, remove the build time options from the command line and keep only the runtime options.

17.2. EXPORTING A REALM TO A DIRECTORY

To export a realm, you can use the **export** command. Your Red Hat build of Keycloak server instance must not be started when invoking this command.

```
bin/kc.[sh|bat] export --help
```

To export a realm to a directory, you can use the **--dir <dir>** option.

```
bin/kc.[sh|bat] export --dir <dir>
```

When exporting realms to a directory, the server is going to create separate files for each realm being exported.

17.2.1. Configuring how users are exported

You are also able to configure how users are going to be exported by setting the **--users <strategy>** option. The values available for this option are:

- **different_files**: Users export into different json files, depending on the maximum number of users per file set by **--users-per-file**. This is the default value.

- **skip**: Skips exporting users.
- **realm_file**: Users will be exported to the same file as the realm settings. For a realm named "foo", this would be "foo-realm.json" with realm data and users.
- **same_file**: All users are exported to one explicit file. So you will get two json files for a realm, one with realm data and one with users.

If you are exporting users using the **different_files** strategy, you can set how many users per file you want by setting the **--users-per-file** option. The default value is **50**.

```
bin/kc.[sh|bat] export --dir <dir> --users different_files --users-per-file 100
```

17.3. EXPORTING A REALM TO A FILE

To export a realm to a file, you can use the **--file <file>** option.

```
bin/kc.[sh|bat] export --file <file>
```

When exporting realms to a file, the server is going to use the same file to store the configuration for all the realms being exported.

17.4. EXPORTING A SPECIFIC REALM

If you do not specify a specific realm to export, all realms are exported. To export a single realm, you can use the **--realm** option as follows:

```
bin/kc.[sh|bat] export [--dir|--file] <path> --realm my-realm
```

17.5. IMPORTING A REALM FROM A DIRECTORY

To import a realm, you can use the **import** command. Your Red Hat build of Keycloak server instance must not be started when invoking this command.

```
bin/kc.[sh|bat] import --help
```

After exporting a realm to a directory, you can use the **--dir <dir>** option to import the realm back to the server as follows:

```
bin/kc.[sh|bat] import --dir <dir>
```

When importing realms using the **import** command, you are able to set if existing realms should be skipped, or if they should be overridden with the new configuration. For that, you can set the **--override** option as follows:

```
bin/kc.[sh|bat] import --dir <dir> --override false
```

By default, the **--override** option is set to **true** so that realms are always overridden with the new configuration.

17.6. IMPORTING A REALM FROM A FILE

To import a realm previously exported in a single file, you can use the `--file <file>` option as follows:

```
bin/kc.[sh|bat] import --file <file>
```

17.7. IMPORTING A REALM DURING STARTUP

You are also able to import realms when the server is starting by using the `--import-realm` option.

```
bin/kc.[sh|bat] start --import-realm
```

When you set the `--import-realm` option, the server is going to try to import any realm configuration file from the `data/import` directory. Only regular files using the `.json` extension are read from this directory, sub-directories are ignored.



NOTE

For the Red Hat build of Keycloak containers, the import directory is `/opt/keycloak/data/import`

If a realm already exists in the server, the import operation is skipped. The main reason behind this behavior is to avoid re-creating realms and potentially loose state between server restarts.

To re-create realms you should explicitly run the `import` command prior to starting the server.

Importing the `master` realm is not supported because as it is a very sensitive operation.

17.7.1. Using Environment Variables within the Realm Configuration Files

When importing a realm at startup, you are able to use placeholders to resolve values from environment variables for any realm configuration.

Realm configuration using placeholders

```
{
  "realm": "${MY_REALM_NAME}",
  "enabled": true,
  ...
}
```

In the example above, the value set to the `MY_REALM_NAME` environment variable is going to be used to set the `realm` property.

CHAPTER 18. USING A VAULT

Red Hat build of Keycloak provides two out-of-the-box implementations of the Vault SPI: a plain-text file-based vault and Java KeyStore-based vault.

The file-based vault implementation is especially useful for Kubernetes/OpenShift secrets. You can mount Kubernetes secrets into the Red Hat build of Keycloak Container, and the data fields will be available in the mounted folder with a flat-file structure.

The Java KeyStore-based vault implementation is useful for storing secrets in bare metal installations. You can use the KeyStore vault, which is encrypted using a password.

18.1. AVAILABLE INTEGRATIONS

Secrets stored in the vaults can be used at the following places of the Administration Console:

- Obtain the SMTP Mail server Password
- Obtain the LDAP Bind Credential when using LDAP-based User Federation
- Obtain the OIDC identity providers Client Secret when integrating external identity providers

18.2. ENABLING A VAULT

For enabling the file-based vault you need to build Red Hat build of Keycloak first using the following build option:

```
bin/kc.[sh|bat] build --vault=file
```

Analogically, for the Java KeyStore-based you need to specify the following build option:

```
bin/kc.[sh|bat] build --vault=keystore
```

18.3. CONFIGURING THE FILE-BASED VAULT

18.3.1. Setting the base directory to lookup secrets

Kubernetes/OpenShift secrets are basically mounted files. To configure a directory where these files should be mounted, enter this command:

```
bin/kc.[sh|bat] start --vault-dir=/my/path
```

18.3.2. Realm-specific secret files

Kubernetes/OpenShift Secrets are used on a per-realm basis in Red Hat build of Keycloak, which requires a naming convention for the file in place:

```
#{vault.<realmname>_<secretname>}
```

18.3.3. Using underscores in the Name

To process the secret correctly, you double all underscores in the <realmname> or the <secretname>, separated by a single underscore.

Example

- Realm Name: **sso_realm**
- Desired Name: **ldap_credential**
- Resulting file Name:

```
sso__realm__ldap__credential
```

Note the doubled underscores between *sso* and *realm* and also between *ldap* and *credential*.

18.4. CONFIGURING THE JAVA KEYSTORE-BASED VAULT

In order to use the Java KeyStore-based vault, you need to create a KeyStore file first. You can use the following command for doing so:

```
keytool -importpass -alias <realm-name>_<alias> -keystore keystore.p12 -storepass  
keystorepassword
```

and then enter a value you want to store in the vault. Note that the format of the **-alias** parameter depends on the key resolver used. The default key resolver is **REALM_UNDERSCORE_KEY**.

This by default results to storing the value in a form of generic PBEKey (password based encryption) within SecretKeyEntry.

You can then start Red Hat build of Keycloak using the following runtime options:

```
bin/kc.[sh|bat] start --vault-file=/path/to/keystore.p12 --vault-pass=<value> --vault-type=<value>
```

Note that the **--vault-type** parameter is optional and defaults to **PKCS12**.

Secrets stored in the vault can then be accessed in a realm via the following placeholder (assuming using the **REALM_UNDERSCORE_KEY** key resolver): **\${vault.realm-name_alias}**.

18.5. EXAMPLE: USE AN LDAP BIND CREDENTIAL SECRET IN THE ADMIN CONSOLE

Example setup

- A realm named **secrettest**
- A desired Name **ldapBc** for the bind Credential
- Resulting file name: **secrettest_ldapBc**

Usage in Admin Console

You can then use this secret from the Admin Console by using **\${vault.ldapBc}** as the value for the **Bind Credential** when configuring your LDAP User federation.

18.6. RELEVANT OPTIONS

	Value
<p>vault ■</p> <p>Enables a vault provider.</p> <p>CLI: --vault Env: KC_VAULT</p>	file, keystore
<p>vault-dir</p> <p>If set, secrets can be obtained by reading the content of files within the given directory.</p> <p>CLI: --vault-dir Env: KC_VAULT_DIR</p>	
<p>vault-file</p> <p>Path to the keystore file.</p> <p>CLI: --vault-file Env: KC_VAULT_FILE</p>	
<p>vault-pass</p> <p>Password for the vault keystore.</p> <p>CLI: --vault-pass Env: KC_VAULT_PASS</p>	
<p>vault-type</p> <p>Specifies the type of the keystore file.</p> <p>CLI: --vault-type Env: KC_VAULT_TYPE</p>	PKCS12 (default)

CHAPTER 19. ALL CONFIGURATION

19.1. CACHE

	Value
<p>cache ■</p> <p>Defines the cache mechanism for high-availability.</p> <p>By default in production mode, a ispn cache is used to create a cluster between multiple server nodes. By default in development mode, a local cache disables clustering and is intended for development and testing purposes.</p> <p>CLI: --cache Env: KC_CACHE</p>	<p>ispn (default), local</p>
<p>cache-config-file ■</p> <p>Defines the file from which cache configuration should be loaded from.</p> <p>The configuration file is relative to the conf/ directory.</p> <p>CLI: --cache-config-file Env: KC_CACHE_CONFIG_FILE</p>	
<p>cache-stack ■</p> <p>Define the default stack to use for cluster communication and node discovery.</p> <p>This option only takes effect if cache is set to ispn. Default: udp.</p> <p>CLI: --cache-stack Env: KC_CACHE_STACK</p>	<p>tcp, udp, kubernetes, ec2, azure, google</p>

19.2. DATABASE

	Value
<p>db ■</p> <p>The database vendor.</p> <p>CLI: --db Env: KC_DB</p>	<p>dev-file (default), dev-mem, mariadb, mssql, mysql, oracle, postgres</p>

	Value
<p>db-driver</p> <p>The fully qualified class name of the JDBC driver.</p> <p>If not set, a default driver is set accordingly to the chosen database.</p> <p>CLI: --db-driver Env: KC_DB_DRIVER</p>	
<p>db-password</p> <p>The password of the database user.</p> <p>CLI: --db-password Env: KC_DB_PASSWORD</p>	
<p>db-pool-initial-size</p> <p>The initial size of the connection pool.</p> <p>CLI: --db-pool-initial-size Env: KC_DB_POOL_INITIAL_SIZE</p>	
<p>db-pool-max-size</p> <p>The maximum size of the connection pool.</p> <p>CLI: --db-pool-max-size Env: KC_DB_POOL_MAX_SIZE</p>	100 (default)
<p>db-pool-min-size</p> <p>The minimal size of the connection pool.</p> <p>CLI: --db-pool-min-size Env: KC_DB_POOL_MIN_SIZE</p>	
<p>db-schema</p> <p>The database schema to be used.</p> <p>CLI: --db-schema Env: KC_DB_SCHEMA</p>	

	Value
<p>db-url</p> <p>The full database JDBC URL.</p> <p>If not provided, a default URL is set based on the selected database vendor. For instance, if using postgres, the default JDBC URL would be jdbc:postgresql://localhost/keycloak.</p> <p>CLI: --db-url Env: KC_DB_URL</p>	
<p>db-url-database</p> <p>Sets the database name of the default JDBC URL of the chosen vendor.</p> <p>If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-database Env: KC_DB_URL_DATABASE</p>	
<p>db-url-host</p> <p>Sets the hostname of the default JDBC URL of the chosen vendor.</p> <p>If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-host Env: KC_DB_URL_HOST</p>	
<p>db-url-port</p> <p>Sets the port of the default JDBC URL of the chosen vendor.</p> <p>If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-port Env: KC_DB_URL_PORT</p>	
<p>db-url-properties</p> <p>Sets the properties of the default JDBC URL of the chosen vendor.</p> <p>Make sure to set the properties accordingly to the format expected by the database vendor, as well as appending the right character at the beginning of this property value. If the db-url option is set, this option is ignored.</p> <p>CLI: --db-url-properties Env: KC_DB_URL_PROPERTIES</p>	

	Value
<p>db-username</p> <p>The username of the database user.</p> <p>CLI: --db-username Env: KC_DB_USERNAME</p>	

19.3. TRANSACTION

	Value
<p>transaction-xa-enabled ■</p> <p>If set to false, Keycloak uses a non-XA datasource in case the database does not support XA transactions.</p> <p>CLI: --transaction-xa-enabled Env: KC_TRANSACTION_XA_ENABLED</p>	<p>true (default), false</p>

19.4. FEATURE

	Value
<p>features ■</p> <p>Enables a set of one or more features.</p> <p>CLI: --features Env: KC_FEATURES</p>	<p>account-api, account2, account3, admin-api, admin-fine-grained-authz, admin2, authorization, ciba, client-policies, client-secret-rotation, declarative-user-profile, docker, dynamic-scopes, fips, impersonation, js-adapter, kerberos, linkedin-oauth, map-storage, multi-site, par, preview, recovery-codes, scripts, step-up-authentication, token-exchange, update-email, web-authn</p>

	Value
<p>features-disabled ■</p> <p>Disables a set of one or more features.</p> <p>CLI: --features-disabled Env: KC_FEATURES_DISABLED</p>	<p>account-api, account2, account3, admin-api, admin-fine-grained-authz, admin2, authorization, ciba, client-policies, client-secret-rotation, declarative-user-profile, docker, dynamic-scopes, fips, impersonation, js-adapter, kerberos, linkedin-oauth, map-storage, multi-site, par, preview, recovery-codes, scripts, step-up-authentication, token-exchange, update-email, web-authn</p>

19.5. HOSTNAME

	Value
<p>hostname</p> <p>Hostname for the Keycloak server.</p> <p>CLI: --hostname Env: KC_HOSTNAME</p>	
<p>hostname-admin</p> <p>The hostname for accessing the administration console.</p> <p>Use this option if you are exposing the administration console using a hostname other than the value set to the hostname option.</p> <p>CLI: --hostname-admin Env: KC_HOSTNAME_ADMIN</p>	

	Value
<p>hostname-admin-url</p> <p>Set the base URL for accessing the administration console, including scheme, host, port and path</p> <p>CLI: --hostname-admin-url Env: KC_HOSTNAME_ADMIN_URL</p>	
<p>hostname-debug</p> <p>Toggle the hostname debug page that is accessible at <code>/realms/master/hostname-debug</code></p> <p>CLI: --hostname-debug Env: KC_HOSTNAME_DEBUG</p>	true, false (default)
<p>hostname-path</p> <p>This should be set if proxy uses a different context-path for Keycloak.</p> <p>CLI: --hostname-path Env: KC_HOSTNAME_PATH</p>	
<p>hostname-port</p> <p>The port used by the proxy when exposing the hostname.</p> <p>Set this option if the proxy uses a port other than the default HTTP and HTTPS ports.</p> <p>CLI: --hostname-port Env: KC_HOSTNAME_PORT</p>	-1 (default)
<p>hostname-strict</p> <p>Disables dynamically resolving the hostname from request headers.</p> <p>Should always be set to true in production, unless proxy verifies the Host header.</p> <p>CLI: --hostname-strict Env: KC_HOSTNAME_STRICT</p>	true (default), false
<p>hostname-strict-backchannel</p> <p>By default backchannel URLs are dynamically resolved from request headers to allow internal and external applications.</p> <p>If all applications use the public URL this option should be enabled.</p> <p>CLI: --hostname-strict-backchannel Env: KC_HOSTNAME_STRICT_BACKCHANNEL</p>	true, false (default)

	Value
<p>hostname-url</p> <p>Set the base URL for frontend URLs, including scheme, host, port and path.</p> <p>CLI: --hostname-url Env: KC_HOSTNAME_URL</p>	

19.6. HTTP/TLS

	Value
<p>http-enabled</p> <p>Enables the HTTP listener.</p> <p>CLI: --http-enabled Env: KC_HTTP_ENABLED</p>	true, false (default)
<p>http-host</p> <p>The used HTTP Host.</p> <p>CLI: --http-host Env: KC_HTTP_HOST</p>	0.0.0.0 (default)
<p>http-port</p> <p>The used HTTP port.</p> <p>CLI: --http-port Env: KC_HTTP_PORT</p>	8080 (default)
<p>http-relative-path ■</p> <p>Set the path relative to / for serving resources.</p> <p>The path must start with a /.</p> <p>CLI: --http-relative-path Env: KC_HTTP_RELATIVE_PATH</p>	/ (default)
<p>https-certificate-file</p> <p>The file path to a server certificate or certificate chain in PEM format.</p> <p>CLI: --https-certificate-file Env: KC_HTTPS_CERTIFICATE_FILE</p>	

	Value
<p>https-certificate-key-file</p> <p>The file path to a private key in PEM format.</p> <p>CLI: --https-certificate-key-file Env: KC_HTTPS_CERTIFICATE_KEY_FILE</p>	
<p>https-cipher-suites</p> <p>The cipher suites to use.</p> <p>If none is given, a reasonable default is selected.</p> <p>CLI: --https-cipher-suites Env: KC_HTTPS_CIPHER_SUITES</p>	
<p>https-client-auth</p> <p>Configures the server to require/request client authentication.</p> <p>CLI: --https-client-auth Env: KC_HTTPS_CLIENT_AUTH</p>	<p>none (default), request, required</p>
<p>https-key-store-file</p> <p>The key store which holds the certificate information instead of specifying separate files.</p> <p>CLI: --https-key-store-file Env: KC_HTTPS_KEY_STORE_FILE</p>	
<p>https-key-store-password</p> <p>The password of the key store file.</p> <p>CLI: --https-key-store-password Env: KC_HTTPS_KEY_STORE_PASSWORD</p>	<p>password (default)</p>
<p>https-key-store-type</p> <p>The type of the key store file.</p> <p>If not given, the type is automatically detected based on the file name. If fips-mode is set to strict and no value is set, it defaults to BCFKS.</p> <p>CLI: --https-key-store-type Env: KC_HTTPS_KEY_STORE_TYPE</p>	

	Value
<p>https-port</p> <p>The used HTTPS port.</p> <p>CLI: --https-port Env: KC_HTTPS_PORT</p>	8443 (default)
<p>https-protocols</p> <p>The list of protocols to explicitly enable.</p> <p>CLI: --https-protocols Env: KC_HTTPS_PROTOCOLS</p>	TLSv1.3,TLSv1.2 (default)
<p>https-trust-store-file</p> <p>The trust store which holds the certificate information of the certificates to trust.</p> <p>CLI: --https-trust-store-file Env: KC_HTTPS_TRUST_STORE_FILE</p>	
<p>https-trust-store-password</p> <p>The password of the trust store file.</p> <p>CLI: --https-trust-store-password Env: KC_HTTPS_TRUST_STORE_PASSWORD</p>	
<p>https-trust-store-type</p> <p>The type of the trust store file.</p> <p>If not given, the type is automatically detected based on the file name. If fips-mode is set to strict and no value is set, it defaults to BCFKS.</p> <p>CLI: --https-trust-store-type Env: KC_HTTPS_TRUST_STORE_TYPE</p>	

19.7. HEALTH

	Value
<p>health-enabled </p> <p>If the server should expose health check endpoints.</p> <p>If enabled, health checks are available at the /health, /health/ready and /health/live endpoints.</p> <p>CLI: --health-enabled Env: KC_HEALTH_ENABLED</p>	true, false (default)

19.8. CONFIG

	Value
<p>config-keystore</p> <p>Specifies a path to the KeyStore Configuration Source.</p> <p>CLI: --config-keystore Env: KC_CONFIG_KEYSTORE</p>	
<p>config-keystore-password</p> <p>Specifies a password to the KeyStore Configuration Source.</p> <p>CLI: --config-keystore-password Env: KC_CONFIG_KEYSTORE_PASSWORD</p>	
<p>config-keystore-type</p> <p>Specifies a type of the KeyStore Configuration Source.</p> <p>CLI: --config-keystore-type Env: KC_CONFIG_KEYSTORE_TYPE</p>	PKCS12 (default)

19.9. METRICS

	Value
<p>metrics-enabled </p> <p>If the server should expose metrics.</p> <p>If enabled, metrics are available at the /metrics endpoint.</p> <p>CLI: --metrics-enabled Env: KC_METRICS_ENABLED</p>	true, false (default)

19.10. PROXY

	Value
<p>proxy</p> <p>The proxy address forwarding mode if the server is behind a reverse proxy.</p> <p>CLI: --proxy Env: KC_PROXY</p>	<p>none (default), edge, reencrypt, passthrough</p>

19.11. VAULT

	Value
<p>vault </p> <p>Enables a vault provider.</p> <p>CLI: --vault Env: KC_VAULT</p>	<p>file, keystore</p>
<p>vault-dir</p> <p>If set, secrets can be obtained by reading the content of files within the given directory.</p> <p>CLI: --vault-dir Env: KC_VAULT_DIR</p>	
<p>vault-file</p> <p>Path to the keystore file.</p> <p>CLI: --vault-file Env: KC_VAULT_FILE</p>	
<p>vault-pass</p> <p>Password for the vault keystore.</p> <p>CLI: --vault-pass Env: KC_VAULT_PASS</p>	
<p>vault-type</p> <p>Specifies the type of the keystore file.</p> <p>CLI: --vault-type Env: KC_VAULT_TYPE</p>	<p>PKCS12 (default)</p>

19.12. LOGGING

	Value
<p>log</p> <p>Enable one or more log handlers in a comma-separated list.</p> <p>CLI: --log Env: KC_LOG</p>	<p>console (default), file</p>
<p>log-console-color</p> <p>Enable or disable colors when logging to console.</p> <p>CLI: --log-console-color Env: KC_LOG_CONSOLE_COLOR</p>	<p>true, false (default)</p>
<p>log-console-format</p> <p>The format of unstructured console log entries.</p> <p>If the format has spaces in it, escape the value using "<format>".</p> <p>CLI: --log-console-format Env: KC_LOG_CONSOLE_FORMAT</p>	<p>%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n (default)</p>
<p>log-console-output</p> <p>Set the log output to JSON or default (plain) unstructured logging.</p> <p>CLI: --log-console-output Env: KC_LOG_CONSOLE_OUTPUT</p>	<p>default (default), json</p>
<p>log-file</p> <p>Set the log file path and filename.</p> <p>CLI: --log-file Env: KC_LOG_FILE</p>	<p>data/log/keycloak.log (default)</p>
<p>log-file-format</p> <p>Set a format specific to file log entries.</p> <p>CLI: --log-file-format Env: KC_LOG_FILE_FORMAT</p>	<p>%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n (default)</p>
<p>log-file-output</p> <p>Set the log output to JSON or default (plain) unstructured logging.</p> <p>CLI: --log-file-output Env: KC_LOG_FILE_OUTPUT</p>	<p>default (default), json</p>

	Value
<p>log-level</p> <p>The log level of the root category or a comma-separated list of individual categories and their levels.</p> <p>For the root category, you don't need to specify a category.</p> <p>CLI: --log-level Env: KC_LOG_LEVEL</p>	info (default)

19.13. SECURITY

	Value
<p>fips-mode ■</p> <p>Sets the FIPS mode.</p> <p>If non-strict is set, FIPS is enabled but on non-approved mode. For full FIPS compliance, set strict to run on approved mode. This option defaults to disabled when fips feature is disabled, which is by default. This option defaults to non-strict when fips feature is enabled.</p> <p>CLI: --fips-mode Env: KC_FIPS_MODE</p>	non-strict, strict

19.14. EXPORT

	Value
<p>dir</p> <p>Set the path to a directory where files will be created with the exported data.</p> <p>CLI: --dir Env: KC_DIR</p>	
<p>realm</p> <p>Set the name of the realm to export.</p> <p>If not set, all realms are going to be exported.</p> <p>CLI: --realm Env: KC_REALM</p>	

	Value
<p>users</p> <p>Set how users should be exported.</p> <p>CLI: --users Env: KC_USERS</p>	<p>skip, realm_file, same_file, different_files (default)</p>
<p>users-per-file</p> <p>Set the number of users per file.</p> <p>It is used only if users is set to different_files. Increasing this number leads to exponentially increasing export times.</p> <p>CLI: --users-per-file Env: KC_USERS_PER_FILE</p>	<p>50 (default)</p>

19.15. IMPORT

	Value
<p>file</p> <p>Set the path to a file that will be read.</p> <p>CLI: --file Env: KC_FILE</p>	
<p>override</p> <p>Set if existing data should be overwritten.</p> <p>If set to false, data will be ignored.</p> <p>CLI: --override Env: KC_OVERRIDE</p>	<p>true (default), false</p>

CHAPTER 20. ALL PROVIDER CONFIGURATION

20.1. AUTHENTICATION-SESSIONS

20.1.1. infinispan

	Value
<p>spi-authentication-sessions-infinispan-auth-sessions-limit</p> <p>The maximum number of concurrent authentication sessions per RootAuthenticationSession.</p> <p>CLI: --spi-authentication-sessions-infinispan-auth-sessions-limit Env: KC_SPI_AUTHENTICATION_SESSIONS_INFINISPAN_AUTH_SESSIONS_LIMIT</p>	<p>300 (default) or any int</p>

20.1.2. map

	Value
<p>spi-authentication-sessions-map-auth-sessions-limit</p> <p>The maximum number of concurrent authentication sessions per RootAuthenticationSession.</p> <p>CLI: --spi-authentication-sessions-map-auth-sessions-limit Env: KC_SPI_AUTHENTICATION_SESSIONS_MAP_AUTH_SESSIONS_LIMIT</p>	<p>300 (default) or any int</p>

20.2. BRUTE-FORCE-PROTECTOR

20.2.1. default-brute-force-detector

	Value
<p>spi-brute-force-protector-default-brute-force-detector-allow-concurrent-requests</p> <p>If concurrent logins are allowed by the brute force protection.</p> <p>CLI: --spi-brute-force-protector-default-brute-force-detector-allow-concurrent-requests Env: KC_SPI_BRUTE_FORCE_PROTECTOR_DEFAULT_BRUTE_FORCE_DETECTOR_ALLOW_CONCURRENT_REQUESTS</p>	<p>true, false (default)</p>

20.3. CIBA-AUTH-CHANNEL

20.3.1. ciba-http-auth-channel

	Value
<p>spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri</p> <p>The HTTP(S) URI of the authentication channel.</p> <p>CLI: --spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri Env: KC_SPI_CIBA_AUTH_CHANNEL_CIBA_HTTP_AUTH_CHANNEL_HTTP_AUTHENTICATION_CHANNEL_URI</p>	any string

20.4. CONNECTIONS-HTTP-CLIENT

20.4.1. default

	Value
<p>spi-connections-http-client-default-client-key-password</p> <p>The key password.</p> <p>CLI: --spi-connections-http-client-default-client-key-password Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEY_PASSWORD</p>	-1 (default) or any string
<p>spi-connections-http-client-default-client-keystore</p> <p>The file path of the key store from where the key material is going to be read from to set-up TLS connections.</p> <p>CLI: --spi-connections-http-client-default-client-keystore Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE</p>	any string
<p>spi-connections-http-client-default-client-keystore-password</p> <p>The key store password.</p> <p>CLI: --spi-connections-http-client-default-client-keystore-password Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE_PASSWORD</p>	any string

	Value
<p>spi-connections-http-client-default-connection-pool-size</p> <p>Assigns maximum total connection value.</p> <p>CLI: --spi-connections-http-client-default-connection-pool-size Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_POOL_SIZE</p>	any int
<p>spi-connections-http-client-default-connection-ttl-millis</p> <p>Sets maximum time, in milliseconds, to live for persistent connections.</p> <p>CLI: --spi-connections-http-client-default-connection-ttl-millis Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_TTL_MILLIS</p>	-1 (default) or any long
<p>spi-connections-http-client-default-disable-cookies</p> <p>Disables state (cookie) management.</p> <p>CLI: --spi-connections-http-client-default-disable-cookies Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_COOKIES</p>	true (default), false
<p>spi-connections-http-client-default-disable-trust-manager</p> <p>Disable trust management and hostname verification.</p> <p>NOTE this is a security hole, so only set this option if you cannot or do not want to verify the identity of the host you are communicating with.</p> <p>CLI: --spi-connections-http-client-default-disable-trust-manager Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_TRUST_MANAGER</p>	true , false (default)
<p>spi-connections-http-client-default-establish-connection-timeout-millis</p> <p>When trying to make an initial socket connection, what is the timeout?</p> <p>CLI: --spi-connections-http-client-default-establish-connection-timeout-millis Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_ESTABLISH_CONNECTION_TIMEOUT_MILLIS</p>	-1 (default) or any long

	Value
<p>spi-connections-http-client-default-max-connection-idle-time-millis</p> <p>Sets the time, in milliseconds, for evicting idle connections from the pool.</p> <p>CLI: --spi-connections-http-client-default-max-connection-idle-time-millis Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_CONNECTION_IDLE_TIME_MILLIS</p>	<p>900000 (default) or any long</p>
<p>spi-connections-http-client-default-max-pooled-per-route</p> <p>Assigns maximum connection per route value.</p> <p>CLI: --spi-connections-http-client-default-max-pooled-per-route Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_POOLED_PER_ROUTE</p>	<p>64 (default) or any int</p>
<p>spi-connections-http-client-default-proxy-mappings</p> <p>Denotes the combination of a regex based hostname pattern and a proxy-uri in the form of <code>hostnamePattern;proxyUri</code>.</p> <p>CLI: --spi-connections-http-client-default-proxy-mappings Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_PROXY_MAPPINGS</p>	<p>any string</p>
<p>spi-connections-http-client-default-reuse-connections</p> <p>If connections should be reused.</p> <p>CLI: --spi-connections-http-client-default-reuse-connections Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_REUSE_CONNECTIONS</p>	<p>true (default), false</p>
<p>spi-connections-http-client-default-socket-timeout-millis</p> <p>Socket inactivity timeout.</p> <p>CLI: --spi-connections-http-client-default-socket-timeout-millis Env: KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_SOCKET_TIMEOUT_MILLIS</p>	<p>5000 (default) or any long</p>

20.5. CONNECTIONS-JPA

20.5.1. legacy

	Value
<p>spi-connections-jpa-legacy-initialize-empty</p> <p>Initialize database if empty.</p> <p>If set to false the database has to be manually initialized. If you want to manually initialize the database set migrationStrategy to manual which will create a file with SQL commands to initialize the database.</p> <p>CLI: --spi-connections-jpa-legacy-initialize-empty Env: KC_SPI_CONNECTIONS_JPA_LEGACY_INITIALIZE_EMPTY</p>	<p>true (default), false</p>
<p>spi-connections-jpa-legacy-migration-export</p> <p>Path for where to write manual database initialization/migration file.</p> <p>CLI: --spi-connections-jpa-legacy-migration-export Env: KC_SPI_CONNECTIONS_JPA_LEGACY_MIGRATION_EXPORT</p>	<p>any string</p>
<p>spi-connections-jpa-legacy-migration-strategy</p> <p>Strategy to use to migrate database.</p> <p>Valid values are update, manual and validate. Update will automatically migrate the database schema. Manual will export the required changes to a file with SQL commands that you can manually execute on the database. Validate will simply check if the database is up-to-date.</p> <p>CLI: --spi-connections-jpa-legacy-migration-strategy Env: KC_SPI_CONNECTIONS_JPA_LEGACY_MIGRATION_STRATEGY</p>	<p>update (default), manual, validate</p>

20.6. DBLOCK

20.6.1. jpa

	Value
<p>spi-dblock-jpa-lock-wait-timeout</p> <p>The maximum time to wait when waiting to release a database lock.</p> <p>CLI: --spi-dblock-jpa-lock-wait-timeout Env: KC_SPI_DBLOCK_JPA_LOCK_WAIT_TIMEOUT</p>	<p>any int</p>

20.7. EVENTS-LISTENER

20.7.1. email

	Value
<p>spi-events-listener-email-exclude-events</p> <p>A comma-separated list of events that should not be sent via email to the user's account.</p> <p>CLI: --spi-events-listener-email-exclude-events Env: KC_SPI_EVENTS_LISTENER_EMAIL_EXCLUDE_EVENTS</p>	<p>authreqid_to_token, authreqid_to_token_error, client_delete, client_delete_error, client_info, client_info_error, client_initiated_account_linking, client_initiated_account_linking_error, client_login, client_login_error, client_register, client_register_error, client_update, client_update_error, code_to_token, code_to_token_error, ', custom_required_action, custom_required_action_error, delete_account, delete_account_error, execute_action_token, execute_action_token_error, execute_actions, execute_actions_error, federated_identity_link, federated_identity_link_error, grant_consent, grant_consent_error, identity_provider_first_login, identity_provider_first_login_error, identity_provider_link_account, identity_provider_link_account_error, identity_provider_login, identity_provider_login_error, identity_provider_post_login,</p>

	identity_provider_pos Value
	identity_provider_pos_login_error, identity_provider_response, identity_provider_response_error, identity_provider_retrieve_token, identity_provider_retrieve_token_error, impersonate, impersonate_error, introspect_token, introspect_token_error, invalid_signature, invalid_signature_error, login, login_error, logout, logout_error, oauth2_device_auth, oauth2_device_auth_error, oauth2_device_code_to_token, oauth2_device_code_to_token_error, oauth2_device_verify_user_code, oauth2_device_verify_user_code_error, permission_token, permission_token_error, pushed_authorization_request, pushed_authorization_request_error, refresh_token, refresh_token_error, register, register_error, register_node, register_node_error, remove_federated_identity, remove_federated_identity_error, remove_totp, remove_totp_error, reset_password, reset_password_error, restart_authentication, restart_authentication_error, revoke_grant,

	Value
	revoke_grant_error, send_identity_provi der_link, send_identity_provi der_link_error, send_reset_passwor d, send_reset_passwor d_error, send_verify_email, send_verify_email_e rror, token_exchange, token_exchange_err or, unregister_node, unregister_node_err or, update_consent, update_consent_err or, update_email, update_email_error, update_password, update_password_er ror, update_profile, update_profile_error, update_totp, update_totp_error, user_info_request, user_info_request_e rror, validate_access_tok en, validate_access_tok en_error, verify_email, verify_email_error, verify_profile, verify_profile_error
<p>spi-events-listener-email-include-events</p> <p>A comma-separated list of events that should be sent via email to the user's account.</p> <p>CLI: --spi-events-listener-email-include-events Env: KC_SPI_EVENTS_LISTENER_EMAIL_INCLUDE_EVENTS</p>	authreqid_to_token, authreqid_to_token_ error, client_delete, client_delete_error, client_info, client_info_error, client_initiated_accou nt_linking, client_initiated_accou nt_linking_error, client_login, client_login_error, client_register, client_register_error, client_update, client_update_error, code_to_token, code_to_token_error

	Value
	custom_required_ac tion, custom_required_ac tion_error, delete_account, delete_account_erro r, execute_action_toke n, execute_action_toke n_error, execute_actions, execute_actions_err or, federated_identity_li nk, federated_identity_li nk_error, grant_consent, grant_consent_error, identity_provider_fir st_login, identity_provider_fir st_login_error, identity_provider_lin k_account, identity_provider_lin k_account_error, identity_provider_lo gin, identity_provider_lo gin_error, identity_provider_po st_login, identity_provider_po st_login_error, identity_provider_re sponse, identity_provider_re sponse_error, identity_provider_ret rieve_token, identity_provider_ret rieve_token_error, impersonate, impersonate_error, introspect_token, introspect_token_err or, invalid_signature, invalid_signature_er ror, login, login_error, logout, logout_error, oauth2_device_auth, oauth2_device_auth _error,

	oauth2_device_code Value
	oauth2_device_code _to_token, oauth2_device_code _to_token_error, oauth2_device_verif y_user_code, oauth2_device_verif y_user_code_error, permission_token, permission_token_e rror, pushed_authORIZATIO n_request, pushed_authORIZATIO n_request_error, refresh_token, refresh_token_error, register, register_error, register_node, register_node_error, remove_federated_i dentity, remove_federated_i dentity_error, remove_totp, remove_totp_error, reset_password, reset_password_err or, restart_authenticatio n, restart_authenticatio n_error, revoke_grant, revoke_grant_error, send_identity_provi der_link, send_identity_provi der_link_error, send_reset_passwor d, send_reset_passwor d_error, send_verify_email, send_verify_email_e rror, token_exchange, token_exchange_err or, unregister_node, unregister_node_err or, update_consent, update_consent_err or, update_email, update_email_error, update_password,

	Value
	update_password_error, update_profile_error, update_profile_error, update_totp, update_totp_error, user_info_request, user_info_request_error, validate_access_token, validate_access_token_error, verify_email, verify_email_error, verify_profile, verify_profile_error

20.7.2. jboss-logging

	Value
<p>spi-events-listener-jboss-logging-error-level</p> <p>The log level for error messages.</p> <p>CLI: --spi-events-listener-jboss-logging-error-level Env: KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_ERROR_LEVEL</p>	debug, error, fatal, info, trace, warn (default)
<p>spi-events-listener-jboss-logging-quotes</p> <p>The quotes to use for values, it should be one character like " or '.</p> <p>Use "none" if quotes are not needed.</p> <p>CLI: --spi-events-listener-jboss-logging-quotes Env: KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_QUOTES</p>	" (default) or any string
<p>spi-events-listener-jboss-logging-sanitize</p> <p>If true the log messages are sanitized to avoid line breaks.</p> <p>If false messages are not sanitized.</p> <p>CLI: --spi-events-listener-jboss-logging-sanitize Env: KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_SANITIZE</p>	true (default), false
<p>spi-events-listener-jboss-logging-success-level</p> <p>The log level for success messages.</p> <p>CLI: --spi-events-listener-jboss-logging-success-level Env: KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_SUCCESS_LEVEL</p>	debug (default), error, fatal, info, trace, warn

20.8. EXPORT

20.8.1. dir

	Value
<p>spi-export-dir-dir</p> <p>Directory to export to</p> <p>CLI: --spi-export-dir-dir Env: KC_SPI_EXPORT_DIR_DIR</p>	any string
<p>spi-export-dir-realm-name</p> <p>Realm to export</p> <p>CLI: --spi-export-dir-realm-name Env: KC_SPI_EXPORT_DIR_REALM_NAME</p>	any string
<p>spi-export-dir-users-export-strategy</p> <p>Users export strategy</p> <p>CLI: --spi-export-dir-users-export-strategy Env: KC_SPI_EXPORT_DIR_USERS_EXPORT_STRATEGY</p>	DIFFERENT_FILES (default) or any string
<p>spi-export-dir-users-per-file</p> <p>Users per exported file</p> <p>CLI: --spi-export-dir-users-per-file Env: KC_SPI_EXPORT_DIR_USERS_PER_FILE</p>	50 (default) or any int

20.8.2. single-file

	Value
<p>spi-export-single-file-file</p> <p>File to export to</p> <p>CLI: --spi-export-single-file-file Env: KC_SPI_EXPORT_SINGLE_FILE_FILE</p>	any string
<p>spi-export-single-file-realm-name</p> <p>Realm to export</p> <p>CLI: --spi-export-single-file-realm-name Env: KC_SPI_EXPORT_SINGLE_FILE_REALM_NAME</p>	any string

20.9. GLOBAL-LOCK

20.9.1. map

	Value
<p>spi-global-lock-map-default-timeout-milliseconds</p> <p>Default timeout when waiting for a lock</p> <p>CLI: --spi-global-lock-map-default-timeout-milliseconds Env: KC_SPI_GLOBAL_LOCK_MAP_DEFAULT_TIMEOUT_MILLISECONDS</p>	<p>5000 (default) or any int</p>

20.10. IMPORT

20.10.1. dir

	Value
<p>spi-import-dir-dir</p> <p>Directory to import from</p> <p>CLI: --spi-import-dir-dir Env: KC_SPI_IMPORT_DIR_DIR</p>	<p>any string</p>
<p>spi-import-dir-realm-name</p> <p>Realm to export</p> <p>CLI: --spi-import-dir-realm-name Env: KC_SPI_IMPORT_DIR_REALM_NAME</p>	<p>any string</p>
<p>spi-import-dir-strategy</p> <p>Strategy for import: IGNORE_EXISTING, OVERWRITE_EXISTING</p> <p>CLI: --spi-import-dir-strategy Env: KC_SPI_IMPORT_DIR_STRATEGY</p>	<p>any string</p>

20.10.2. single-file

	Value
<p>spi-import-single-file-file</p> <p>File to import from</p> <p>CLI: --spi-import-single-file-file Env: KC_SPI_IMPORT_SINGLE_FILE_FILE</p>	any string
<p>spi-import-single-file-realm-name</p> <p>Realm to export</p> <p>CLI: --spi-import-single-file-realm-name Env: KC_SPI_IMPORT_SINGLE_FILE_REALM_NAME</p>	any string
<p>spi-import-single-file-strategy</p> <p>Strategy for import: IGNORE_EXISTING, OVERWRITE_EXISTING</p> <p>CLI: --spi-import-single-file-strategy Env: KC_SPI_IMPORT_SINGLE_FILE_STRATEGY</p>	any string

20.11. MAP-STORAGE

20.11.1. hotrod

	Value
<p>spi-map-storage-hotrod-lock-timeout</p> <p>The maximum time to wait in milliseconds when waiting for acquiring a pessimistic read lock.</p> <p>If set to negative there is no timeout configured.</p> <p>CLI: --spi-map-storage-hotrod-lock-timeout Env: KC_SPI_MAP_STORAGE_HOTROD_LOCK_TIMEOUT</p>	10000 (default) or any long

20.11.2. jpa

	Value
<p>spi-map-storage-jpa-lock-timeout</p> <p>The maximum time to wait in milliseconds when waiting for acquiring a pessimistic read lock.</p> <p>If set to negative there is no timeout configured.</p> <p>CLI: --spi-map-storage-jpa-lock-timeout Env: KC_SPI_MAP_STORAGE_JPA_LOCK_TIMEOUT</p>	<p>10000 (default) or any long</p>

20.12. RESOURCE-ENCODING

20.12.1. gzip

	Value
<p>spi-resource-encoding-gzip-excluded-content-types</p> <p>A space separated list of content-types to exclude from encoding.</p> <p>CLI: --spi-resource-encoding-gzip-excluded-content-types Env: KC_SPI_RESOURCE_ENCODING_GZIP_EXCLUDED_CONTENT_TYPES</p>	<p>image/png image/jpeg (default) or any string</p>

20.13. STICKY-SESSION-ENCODER

20.13.1. infinispn

	Value
<p>spi-sticky-session-encoder-infinispn-should-attach-route</p> <p>If the route should be attached to cookies to reflect the node that owns a particular session.</p> <p>CLI: --spi-sticky-session-encoder-infinispn-should-attach-route Env: KC_SPI_STICKY_SESSION_ENCODER_INFISPAN_SHOULD_ATTACH_ROUTE</p>	<p>true (default), false</p>

20.14. TRUSTSTORE

20.14.1. file

	Value
<p>spi-truststore-file-file</p> <p>The file path of the trust store from where the certificates are going to be read from to validate TLS connections.</p> <p>CLI: --spi-truststore-file-file Env: KC_SPI_TRUSTSTORE_FILE_FILE</p>	any string
<p>spi-truststore-file-hostname-verification-policy</p> <p>The hostname verification policy.</p> <p>CLI: --spi-truststore-file-hostname-verification-policy Env: KC_SPI_TRUSTSTORE_FILE_HOSTNAME_VERIFICATION_POLICY</p>	any, wildcard (default), strict
<p>spi-truststore-file-password</p> <p>The trust store password.</p> <p>CLI: --spi-truststore-file-password Env: KC_SPI_TRUSTSTORE_FILE_PASSWORD</p>	any string
<p>spi-truststore-file-type</p> <p>Type of the truststore.</p> <p>If not provided, the type would be detected based on the truststore file extension or platform default type.</p> <p>CLI: --spi-truststore-file-type Env: KC_SPI_TRUSTSTORE_FILE_TYPE</p>	any string

20.15. USER-PROFILE

20.15.1. declarative-user-profile

	Value
<p>spi-user-profile-declarative-user-profile-admin-read-only-attributes</p> <p>Array of regular expressions to identify fields that should be treated read-only so administrators can't change them.</p> <p>CLI: --spi-user-profile-declarative-user-profile-admin-read-only-attributes Env: KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_ADMIN_READ_ONLY_ATTRIBUTES</p>	any MultivaluedString

	Value
<p>spi-user-profile-declarative-user-profile-max-email-local-part-length</p> <p>To set user profile max email local part length</p> <p>CLI: --spi-user-profile-declarative-user-profile-max-email-local-part-length Env: KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_MAX_EMAIL_LOCAL_PART_LENGTH</p>	any String
<p>spi-user-profile-declarative-user-profile-read-only-attributes</p> <p>Array of regular expressions to identify fields that should be treated read-only so users can't change them.</p> <p>CLI: --spi-user-profile-declarative-user-profile-read-only-attributes Env: KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_READ_ONLY_ATTRIBUTES</p>	any MultivaluedString

20.16. WELL-KNOWN

20.16.1. openid-configuration

	Value
<p>spi-well-known-openid-configuration-include-client-scopes</p> <p>If client scopes should be used to calculate the list of supported scopes.</p> <p>CLI: --spi-well-known-openid-configuration-include-client-scopes Env: KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_INCLUDE_CLIENT_SCOPES</p>	true (default), false
<p>spi-well-known-openid-configuration-openid-configuration-override</p> <p>The file path from where the metadata should be loaded from.</p> <p>You can use an absolute file path or, if the file is in the server classpath, use the classpath: prefix to load the file from the classpath.</p> <p>CLI: --spi-well-known-openid-configuration-openid-configuration-override Env: KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_OPENID_CONFIGURATION_OVERRIDE</p>	any string

