Red Hat build of Keycloak 26.2

Server Configuration Guide

Last Updated: 2026-02-10

Abstract

This guide consists of information for administrators to configure Red Hat build of Keycloak 26.2.

Table of Contents

CHAPTER 1. CONFIGURING RED HAT BUILD OF KEYCLOAK

Configure and start 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 four sources, which are listed here in order of application.

1. Command-line parameters
2. Environment variables
3. Options defined in the **conf/keycloak.conf** file, or in a user-created configuration file.
4. 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. If none of the previous values were applied, the value **kc.db-url=keystoreValue** 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. Note that these formats apply to spi options as well.

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
```

Setting that option makes Red Hat build of Keycloak read configuration from the specified file instead of **conf/keycloak.conf**.

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=keystorepass
--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.

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 Red Hat build of Keycloak, defining that property key in **quarkus.properties** has no effect. The Red Hat build of Keycloak configuration value takes precedence over the Quarkus property value.

1.2.7. Using special characters in values

Red Hat build of Keycloak depends upon Quarkus and MicroProfile for processing configuration values. Be aware that value expressions are supported. For example, **${some_key}** evaluates to the value of **some_key**.

To disable expression evaluation, the \ character functions as an escape character. In particular, it must be used to escape the usage of **$** characters when they appear to define an expression or are repeated. For example, if you want the configuration value **my$$password**, use **my\$\$password** instead. Note that the \ character requires additional escaping or quoting when using most unix shells, or when it appears in properties files. For example, bash single quotes preserve the single backslash **--db-password='my\$\$password'**. Also, with bash double quotes, you need an additional backslash **--db-password="my\\$\\$password"**. Similarly in a properties file, backslash characters must also be escaped: **kc.db-password=my\\$\\$password**

Windows-specific considerations

When specifying Windows file paths in configuration values, backslashes must also be escaped. For example, if you want to specify the path **C:\path\to\file**, you should write it as **C:\\path\\to\\file**. Alternatively, you can use forward slashes which don't need escaping: **C:/path/to/file**.

When using PowerShell and your values contain special characters like commas, use single quotes around double quotes:

```
.\kc.bat start --log-level='"INFO,org.hibernate:debug"'
```

PowerShell handles quotes differently. It interprets the quoted string before passing it to the **kc.bat** script, removing the outer quote characters. Therefore, an extra layer of quotes is needed to preserve the value structure. Otherwise, the comma would be interpreted as a delimiter. In Windows CMD, you can use double quotes directly.

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 **KC_BOOTSTRAP_ADMIN_USERNAME=*<username>*** for the initial admin username and **KC_BOOTSTRAP_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. USING SYSTEM VARIABLES IN THE REALM CONFIGURATION

Some of the realm capabilities allow administrators to reference system variables such as environment variables and system properties when configuring the realm and its components.

By default, Red Hat build of Keycloak disallow using system variables but only those explicitly specified through the **spi-admin-allowed-system-variables** configuration option. This option allows you to specify a comma-separated list of keys that will eventually resolve to values from system variables with the same key.

1. Start the server and expose a set of system variables to the server runtime

```
bin/kc.[sh|bat] start --spi-admin-allowed-system-variables=FOO,BAR
```

In future releases, this capability will be removed in favor of preventing any usage of system variables in the realm configuration.

1.7. 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

Prepare Red Hat build of Keycloak for use in 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.

To secure the cache communication for Red Hat build of Keycloak, see Configuring distributed caches.

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 (v2).

2.2.1. Exposing the Red Hat build of Keycloak Administration APIs and UI on a different hostname

It is considered a best practice to expose the Red Hat build of Keycloak Administration REST API and Console on a different hostname or context-path than the one used for the public frontend URLs that are used e.g. by login flows. This separation ensures that the Administration interfaces are not exposed to the public internet, which reduces the attack surface.

WARNING

Access to REST APIs needs to be blocked on the reverse proxy level, if they are not intended to be publicly exposed.

For details, see Configuring the hostname (v2).

2.3. REVERSE PROXY IN A DISTRIBUTED ENVIRONMENT

Apart from Configuring the hostname (v2), 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 Configuring 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. LIMIT THE NUMBER OF QUEUED REQUESTS

A production environment should protect itself from an overload situation, so that it responds to as many valid requests as possible, and to continue regular operations once the situation returns to normal again. One way of doing this is rejecting additional requests once a certain threshold is reached.

Load shedding should be implemented on all levels, including the load balancers in your environment. In addition to that, there is a feature in Red Hat build of Keycloak to limit the number of requests that can't be processed right away and need to be queued. By default, there is no limit set. Set the option **http-max-queued-requests** to limit the number of queued requests to a given threshold matching your environment. Any request that exceeds this limit would return with an immediate **503 Server not Available** response.

2.5. 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.

2.6. RUNNING 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. In the default setup, communication between the nodes is encrypted using TLS.

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

2.6.1. Configure Firewall ports

A set of network ports must be open to allow a healthy network communication between Red Hat build of Keycloak servers. See Configuring distributed caches. It describes what ports need to be open and their usage.

2.7. 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"
```

To set up the server for IPv6 only, set an environment variable as follows for the distributed caches to form a cluster:

```
export JAVA_OPTS_APPEND="-Djava.net.preferIPv4Stack=false -
Djava.net.preferIPv6Addresses=true"
```

See Configuring distributed caches for more details.

CHAPTER 3. BOOTSTRAPPING AND RECOVERING AN ADMIN ACCOUNT

Bootstrap Red Hat build of Keycloak and recover access by creating a temporary admin account.

3.1. A TEMPORARY ADMIN ACCOUNT

A user or service admin account created using one of the methods described below is **temporary**. This means the account should exist only for the duration necessary to perform operations needed to gain permanent and more secure admin access. After that, the account needs to be removed manually. Various UI/UX elements, such as the Administration Console warning banner, labels, and log messages, will indicate to a Red Hat build of Keycloak administrator that the account is temporary.

3.2. BOOTSTRAPPING A TEMPORARY ADMIN ACCOUNT AT RED HAT BUILD OF KEYCLOAK STARTUP

Red Hat build of Keycloak **start** and **start-dev** commands support options for bootstrapping both temporary admin users and admin service accounts. These options are standard configuration options, so they can be specified in any of the configuration sources such as environment variables or CLI parameters. For instance, the following examples demonstrate how to use the **start** and **start-dev** commands with CLI parameters to bootstrap a temporary admin user and an admin service account, respectively:

```
bin/kc.[sh|bat] start --bootstrap-admin-username tmpadm --bootstrap-admin-password pass
```

```
bin/kc.[sh|bat] start-dev --bootstrap-admin-client-id tmpadm --bootstrap-admin-client-secret secret
```

The username or client ID values can be omitted; see the Section 3.5, “Default values” section below for more information.

The purpose of these options is solely for bootstrapping temporary admin accounts. These accounts will be created only during the initial start of the Red Hat build of Keycloak server when the master realm doesn’t exist yet. The accounts are always created in the master realm. For recovering lost admin access, use the dedicated command described in the sections below.

3.3. BOOTSTRAPPING AN ADMIN USER OR SERVICE ACCOUNT USING THE DEDICATED COMMAND

The **bootstrap-admin** command can be executed even before the first-ever start of Red Hat build of Keycloak. Bear in mind that all the Red Hat build of Keycloak nodes need to be stopped prior to using this command. Its execution will trigger the creation of the initial master realm, and as a result, the startup options to bootstrap the admin user and service account will be ignored later when the server is started for the first time.

Additionally, it is strongly recommended to use the dedicated command with the same options that the Red Hat build of Keycloak server is started with (e.g., **db** options).

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 Red Hat build of 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.

NOTE

if you do not use **--optimized** keep in mind that an **bootstrap-admin** command will implicitly create or update an optimized image for you - if you are running the command from the same machine as a server instance, this may impact the next start of your server.

3.3.1. Create an admin user

To create a temporary admin user, execute the following command:

```
bin/kc.[sh|bat] bootstrap-admin user
```

If no other parameters are specified and/or no corresponding environment variables are set, the user is prompted to enter the required information. The username value can be omitted to use the default values. For more information, see the Section 3.5, “Default values” and Section 3.7, “Environment variables” sections below.

Alternatively, the parameters can be directly specified in the command:

```
bin/kc.[sh|bat] bootstrap-admin user --username tmpadm --password:env PASS_VAR
```

This command creates a temporary admin user with the username **tmpadm** and the password retrieved from the environment variable.

3.3.2. Create a service account

In automated scenarios, a temporary admin service account can be a more suitable alternative to a temporary admin user.

To create a temporary admin service account, execute the following command:

```
bin/kc.[sh|bat] bootstrap-admin service
```

Similarly, if no corresponding environment variables or additional parameters are set, the user will be prompted to enter the required information. The client ID value can be omitted to use the default values. For more information, see the Section 3.5, “Default values” and Section 3.7, “Environment variables” sections below.

Alternatively, the parameters can be directly specified in the command:

```
bin/kc.[sh|bat] bootstrap-admin service --client-id tmpclient --client-secret:env=SECRET_VAR
```

This command creates a temporary admin service account with the client ID **tmpclient** and the secret retrieved from the environment variable.

3.4. REGAINING ACCESS TO THE REALM WITH AN INCREASED SECURITY

Passwordless, OTP, or other advanced authentication methods can be enforced for a realm with lost admin access. In such a case, the admin service account needs to be created to recover lost admin access to the realm. After the service account is created, authentication against the Red Hat build of Keycloak instance is required to perform all necessary operations:

```
bin/kcadm.[sh|bat] config credentials --server http://localhost:8080 --realm master --client
<service_account_client_name> --secret <service_account_secret>
```

Next, retrieve the **credentialId**. For this example, the OTP credential is the relevant one. Use the following command to get an array of **CredentialRepresentation** objects and find the one with **type** set to **otp**:

```
bin/kcadm.[sh|bat] get users/{userId}/credentials -r {realm-name}
```

Finally, the retrieved ID can be used to remove the advanced authentication method (in our case, OTP):

```
bin/kcadm.[sh|bat] delete users/{userId}/credentials/{credentialId} -r {realm-name}
```

3.5. DEFAULT VALUES

For both the startup and dedicated command scenarios, the username and client ID are optional and default to **temp-admin** for both the user and service account, respectively.

3.6. DISABLE THE PARAMETERS PROMPT

To disable the prompt for the parameters, the **--no-prompt** parameter can be used. For example:

```
bin/kc.[sh|bat] bootstrap-admin user --username tmpadm --no-prompt
```

If no corresponding environment variable is set, the command will fail with an error message indicating that the required password parameter is missing.

The **--no-prompt** parameter can be useful if the username or client ID should be omitted. For example:

```
bin/kc.[sh|bat] bootstrap-admin user --password:env PASS_VAR --no-prompt
```

This creates a temporary admin user with the default username without prompting for confirmation. For more information, see the Section 3.5, “Default values” section above.

3.7. ENVIRONMENT VARIABLES

For the **bootstrap-admin user** command, both username and password can be optionally set as environment variables:

```
bin/kc.[sh|bat] bootstrap-admin user --username:env <YourUsernameEnv> --password:env
<YourPassEnv>
```

For the **bootstrap-admin service** command, the client ID is optional and defaults to **temp-admin**, while the client secret is required to be set as an environment variable:

```
bin/kc.[sh|bat] bootstrap-admin service --client-id:env <YourClientIdEnv> --client-secret:env
<YourSecretEnv>
```

CHAPTER 4. DIRECTORY STRUCTURE

Understand the purpose of the directories under the installation root.

4.1. INSTALLATION LOCATIONS

If you are installing from a zip file then by default there will be an install root directory of **rhbk-26.2.13**, which can be created anywhere you choose on your filesystem.

/opt/keycloak is the root install location for the server in all containerized usage shown for Red Hat build of Keycloak.

NOTE

In the rest of the documentation, relative paths are understood to be relative to the install root - for example, **conf/file.xml** means **<install root>/conf/file.xml**

4.2. DIRECTORY STRUCTURE

Under the Red Hat build of Keycloak install root there exists a number of folders:

- **bin/** - contains all the shell scripts for the server, including **kc.sh|bat**, **kcadm.sh|bat**, and **kcreg.sh|bat**
  - **client/** - used internally
- **conf/** - directory used for configuration files, including **keycloak.conf** - see Configuring Red Hat build of Keycloak. Many options for specifying a configuration file expect paths relative to this directory.
  - **truststores/** - default path used by the **truststore-paths** option - see Configuring trusted certificates
- **data/** - directory for the server to store runtime information, such as transaction logs
  - **logs/** - default directory for file logging - see Configuring logging
- **lib/** - used internally
- **providers/** - directory for user provided dependencies - see Configuring providers for extending the server and Configuring the database for an example of adding a JDBC driver.
- **themes/** - directory for customizations to the Admin Console - see Developing Themes

CHAPTER 5. RUNNING RED HAT BUILD OF KEYCLOAK IN A CONTAINER

Run Red Hat build of Keycloak from a container image.

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

5.1. 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.

5.1.1. Writing your optimized Red Hat build of Keycloak Containerfile

The following **Containerfile** 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.

Containerfile:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:26.2 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-rhel9:26.2
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. This step must be placed before the line that **RUNs** the **build** command, as below:

```
# A example build step that downloads a JAR file from a URL and adds it to the providers directory
FROM registry.redhat.io/rhbk/keycloak-rhel9:26.2 as builder

...

# Add the provider JAR file to the providers directory
ADD --chown=keycloak:keycloak --chmod=644 <MY_PROVIDER_JAR_URL>
/opt/keycloak/providers/myprovider.jar

...

# Context: RUN the build command
RUN /opt/keycloak/bin/kc.sh build
```

5.1.2. 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 Red Hat build of Keycloak itself. This is due to security hardening measures, which reduce the attack surface of the Red Hat build of 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 Containerfile 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.

5.1.3. Custom ENTRYPOINT shell scripts

If you use a custom entry point script, start Red Hat build of Keycloak with **exec** so it can receive termination signals that are essential for a graceful shutdown.

Correct approach for an ENTRYPOINT shell script

```
#!/bin/bash

# (add your custom logic here)

# Run the 'exec' command as the last step of the script.
# As it replaces the current shell process, no additional shell commands will run after the 'exec'
command.
exec /opt/keycloak/bin/kc.sh start "$@"
```

WARNING

Without **exec**, the shell script remains PID 1 in the container and blocks signals like **SIGTERM** from reaching Red Hat build of Keycloak. This prevents a graceful shutdown and can lead to cache inconsistencies or data loss.

5.1.4. Building the container image

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

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

NOTE

Podman can be used only for creating or customizing images. Podman is not supported for running Red Hat build of Keycloak in production environments.

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

To start the image, run:

```
podman run --name mykeycloak -p 8443:8443 -p 9000:9000 \
        -e KC_BOOTSTRAP_ADMIN_USERNAME=admin -e
KC_BOOTSTRAP_ADMIN_PASSWORD=change_me \
        mykeycloak \
        start --optimized --hostname=localhost
```

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:9000/health**, **https://localhost:9000/health/ready** and **https://localhost:9000/health/live**.

Opening up **https://localhost:9000/metrics** leads to a page containing operational metrics that could be used by your monitoring solution.

5.1.6. Known issues with Docker

- If a **RUN dnf install** command seems to be taking an excessive amount of time, then likely your Docker systemd service has the file limit setting **LimitNOFILE** configured incorrectly. Either update the service configuration to use a better value, such as 1024000, or directly use **ulimit** in the RUN command:

```
...
RUN ulimit -n 1024000 && dnf install --installroot ...
...
```

- If you are including provider JARs and your container fails a **start --optimized** with a notification that a provider JAR has changed, this is due to Docker truncating or otherwise modifying file modification timestamps from what the **build** command recorded to what is seen at runtime. In this case you will need to force the image to use a known timestamp of your choosing with a **touch** command prior to running a **build**:

```
...
# ADD or copy one or more provider jars
ADD --chown=keycloak:keycloak --chmod=644 some-jar.jar /opt/keycloak/providers/
...
RUN touch -m --date=@1743465600 /opt/keycloak/providers/*
RUN /opt/keycloak/bin/kc.sh build
...
```

5.2. 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** accordingly:

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

```
podman run --name mykeycloak -p 3000:8443 \
        -e KC_BOOTSTRAP_ADMIN_USERNAME=admin -e
KC_BOOTSTRAP_ADMIN_PASSWORD=change_me \
        mykeycloak \
        start --optimized --hostname=https://localhost:3000
```

By setting the **hostname** option to a full url you can now access the server at **https://localhost:3000**.

5.3. 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 KC_BOOTSTRAP_ADMIN_USERNAME=admin -e
KC_BOOTSTRAP_ADMIN_PASSWORD=change_me \
        registry.redhat.io/rhbk/keycloak-rhel9:26.2 \
        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.

5.4. 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 KC_BOOTSTRAP_ADMIN_USERNAME=admin -e
KC_BOOTSTRAP_ADMIN_PASSWORD=change_me \
        registry.redhat.io/rhbk/keycloak-rhel9:26.2 \
        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.

5.5. 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 KC_BOOTSTRAP_ADMIN_USERNAME=<admin-user-name>

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

5.6. 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 Red Hat build of 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 KC_BOOTSTRAP_ADMIN_USERNAME=admin -e
KC_BOOTSTRAP_ADMIN_PASSWORD=change_me \
        -v /path/to/realm/data:/opt/keycloak/data/import \
        registry.redhat.io/rhbk/keycloak-rhel9:26.2 \
        start-dev --import-realm
```

Feel free to join the open GitHub Discussion around enhancements of the admin bootstrapping process.

5.7. SPECIFYING DIFFERENT MEMORY SETTINGS

The Red Hat build of Keycloak container, instead of specifying hardcoded values for the initial and maximum heap size, uses relative values to the total memory of a container. This behavior is achieved by JVM options **-XX:MaxRAMPercentage=70**, and **-XX:InitialRAMPercentage=50**.

The **-XX:MaxRAMPercentage** option represents the maximum heap size as 70% of the total container memory. The **-XX:InitialRAMPercentage** option represents the initial heap size as 50% of the total container memory. These values were chosen based on a deeper analysis of Red Hat build of Keycloak memory management.

As the heap size is dynamically calculated based on the total container memory, you should **always set the memory limit** for the container. Previously, the maximum heap size was set to 512 MB, and in order to approach similar values, you should set the memory limit to at least 750 MB. For smaller production-ready deployments, the recommended memory limit is 2 GB.

The JVM options related to the heap might be overridden by setting the environment variable **JAVA_OPTS_KC_HEAP**. You can find the default values of the **JAVA_OPTS_KC_HEAP** in the source code of the **kc.sh**, or **kc.bat** script.

For example, you can specify the environment variable and memory limit as follows:

```
podman run --name mykeycloak -p 8080:8080 -m 1g \
        -e KC_BOOTSTRAP_ADMIN_USERNAME=admin -e
KC_BOOTSTRAP_ADMIN_PASSWORD=change_me \
        -e JAVA_OPTS_KC_HEAP="-XX:MaxHeapFreeRatio=30 -XX:MaxRAMPercentage=65" \
        registry.redhat.io/rhbk/keycloak-rhel9:26.2 \
        start-dev
```

WARNING

If the memory limit is not set, the memory consumption rapidly increases as the heap size can grow up to 70% of the total container memory. Once the JVM allocates the memory, it is returned to the OS reluctantly with the current Red Hat build of Keycloak GC settings.

5.8. RELEVANT OPTIONS

	Value
db ▮ The database vendor. In production mode the default value of **dev-file** is deprecated, you should explicitly specify the db instead. CLI: **--db** Env: **KC_DB**	**dev-file** (default), **dev-mem**, **mariadb**, **mssql**, **mysql**, **oracle**, **postgres**

	Value
db-password The password of the database user. CLI: **--db-password** Env: **KC_DB_PASSWORD**	
db-url The full database JDBC URL. 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**. CLI: **--db-url** Env: **KC_DB_URL**	
db-username The username of the database user. CLI: **--db-username** Env: **KC_DB_USERNAME**	

	Value
features ▮ Enables a set of one or more features. CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account[:v3]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1,v2]**, **admin[:v2]**, **authorization[:v1]**, **cache-embedded-remote-store[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **clusterless[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v2]**, **impersonation[:v1]**, **ipa-tuura-federation[:v1]**, **kerberos[:v1]**, **login[:v2,v1]**, **multi-site[:v1]**, **oid4vc-vci[:v1]**, **opentelemetry[:v1]**, **organization[:v1]**, **par[:v1]**, **passkeys[:v1]**, **persistent-user-sessions[:v1]**, **preview**, **quick-theme[:v1]**, **recovery-codes[:v1]**, **rolling-updates[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange-standard[:v2]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **user-event-metrics[:v1]**, **web-authn[:v1]**

	Value
hostname Address at which is the server exposed. Can be a full URL, or just a hostname. When only hostname is provided, scheme, port and context path are resolved from the request. CLI: **--hostname** Env: **KC_HOSTNAME** Available only when hostname:v2 feature is enabled	
https-key-store-file The key store which holds the certificate information instead of specifying separate files. CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password The password of the key store file. CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**password** (default)
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)
metrics-enabled ▮ If the server should expose metrics. If enabled, metrics are available at the**/metrics** endpoint. CLI: **--metrics-enabled** Env: **KC_METRICS_ENABLED**	**true**, **false** (default)

CHAPTER 6. CONFIGURING TLS

Configure Red Hat build of Keycloak's https certificates for ingoing and outgoing requests.

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.

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.

6.1. 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.

6.2. PROVIDING A 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
```

Recognized file extensions for a keystore:

- **.p12**, **.pkcs12**, and **.pfx** for a pkcs12 file
- **.jks**, and **.keystore** for a jks file
- **.key**, **.crt**, and **.pem** for a pem file

If your keystore does not have an extension matching its file type, you will also need to set the **https-key-store-type** option.

6.2.1. 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.

6.2.1.1. 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 .

6.3. 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>]
```

For example to only enable TLSv1.3, use a command such as the following: **kc.sh start --https-protocols=TLSv1.3**.

6.4. 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>
```

6.5. CERTIFICATE AND KEY RELOADING

By default Red Hat build of Keycloak will reload the certificates, keys, and keystores specified in **https-*** options every hour. For environments where your server keys may need frequent rotation, this allows that to happen without a server restart. You may override the default via the **https-certificates-reload-period** option. Interval on which to reload key store, trust store, and certificate files referenced by **https-*** options. The value may be a java.time.Duration value, an integer number of seconds, or an integer followed by one of the time units [**ms**, **h**, **m**, **s**, **d**]. Must be greater than 30 seconds. Use **-1** to disable.

6.6. RELEVANT OPTIONS

	Value
http-enabled Enables the HTTP listener. Enabled by default in development mode. Typically not enabled in production unless the server is fronted by a TLS termination proxy. CLI: **--http-enabled** Env: **KC_HTTP_ENABLED**	**true**, **false** (default)

	Value
https-certificate-file The file path to a server certificate or certificate chain in PEM format. CLI: **--https-certificate-file** Env: **KC_HTTPS_CERTIFICATE_FILE**	
https-certificate-key-file The file path to a private key in PEM format. CLI: **--https-certificate-key-file** Env: **KC_HTTPS_CERTIFICATE_KEY_FILE**	
https-certificates-reload-period Interval on which to reload key store, trust store, and certificate files referenced by https-* options. May be a java.time.Duration value, an integer number of seconds, or an integer followed by one of [ms, h, m, s, d]. Must be greater than 30 seconds. Use -1 to disable. CLI: **--https-certificates-reload-period** Env: **KC_HTTPS_CERTIFICATES_RELOAD_PERIOD**	**1h** (default)
https-cipher-suites The cipher suites to use. If none is given, a reasonable default is selected. CLI: **--https-cipher-suites** Env: **KC_HTTPS_CIPHER_SUITES**	
https-key-store-file The key store which holds the certificate information instead of specifying separate files. CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password The password of the key store file. CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**password** (default)

	Value
https-key-store-type The type of the key store file. If not given, the type is automatically detected based on the file extension. If **fips-mode** is set to **strict** and no value is set, it defaults to **BCFKS**. CLI: **--https-key-store-type** Env: **KC_HTTPS_KEY_STORE_TYPE**	
https-port The used HTTPS port. CLI: **--https-port** Env: **KC_HTTPS_PORT**	**8443** (default)
https-protocols The list of protocols to explicitly enable. If a value is not supported by the JRE / security configuration, it will be silently ignored. CLI: **--https-protocols** Env: **KC_HTTPS_PROTOCOLS**	**TLSv1.3**, **TLSv1.2**, or any

6.6.1. Management server

	Value
https-management-certificate-file The file path to a server certificate or certificate chain in PEM format for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificate-file** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATE_FILE**	

	Value
https-management-certificate-key-file The file path to a private key in PEM format for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificate-key-file** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATE_KEY_FILE**	
https-management-certificates-reload-period Interval on which to reload key store, trust store, and certificate files referenced by https-management-* options for the management server. May be a java.time.Duration value, an integer number of seconds, or an integer followed by one of [ms, h, m, s, d]. Must be greater than 30 seconds. Use -1 to disable. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificates-reload-period** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATES_RELOAD_PERIOD**	**1h** (default)
https-management-key-store-file The key store which holds the certificate information instead of specifying separate files for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-key-store-file** Env: **KC_HTTPS_MANAGEMENT_KEY_STORE_FILE**	
https-management-key-store-password The password of the key store file for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-key-store-password** Env: **KC_HTTPS_MANAGEMENT_KEY_STORE_PASSWORD**	**password** (default)

CHAPTER 7. CONFIGURING THE HOSTNAME (V2)

Configure the frontend and backchannel endpoints exposed by Red Hat build of Keycloak.

7.1. THE IMPORTANCE OF SETTING THE HOSTNAME OPTION

By default, Red Hat build of Keycloak mandates the configuration of the **hostname** option and does not dynamically resolve URLs. This is a security measure.

Red Hat build of Keycloak freely discloses its own URLs, for instance through the OIDC Discovery endpoint, or as part of the password reset link in an email. If the hostname was dynamically interpreted from a hostname header, it could provide a potential attacker with an opportunity to manipulate a URL in the email, redirect a user to the attacker's fake domain, and steal sensitive data such as action tokens, passwords, etc.

By explicitly setting the **hostname** option, we avoid a situation where tokens could be issued by a fraudulent issuer. The server can be started with an explicit hostname using the following command:

```
bin/kc.[sh|bat] start --hostname my.keycloak.org
```

NOTE

The examples start the Red Hat build of Keycloak instance in production mode, which requires a public certificate and private key in order to secure communications. For more information, refer to the Configuring Red Hat build of Keycloak for production .

7.2. DEFINING SPECIFIC PARTS OF THE HOSTNAME OPTION

As demonstrated in the previous example, the scheme and port are not explicitly required. In such cases, Red Hat build of Keycloak automatically handles these aspects. For instance, the server would be accessible at **https://my.keycloak.org:8443** in the given example. However, a reverse proxy will typically expose Red Hat build of Keycloak at the default ports, e.g. **443**. In that case it's desirable to specify the full URL in the **hostname** option rather than keeping the parts of the URL dynamic. The server can then be started with:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org
```

Similarly, your reverse proxy might expose Red Hat build of Keycloak at a different context path. It is possible to configure Red Hat build of Keycloak to reflect that via the **hostname** and **hostname-admin** options. See the following example:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org:123/auth
```

7.3. UTILIZING AN INTERNAL URL FOR COMMUNICATION AMONG CLIENTS

Red Hat build of Keycloak has the capability to offer a separate URL for backchannel requests, enabling internal communication while maintaining the use of a public URL for frontchannel requests. Moreover, the backchannel is dynamically resolved based on incoming headers. Consider the following example:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org --hostname-backchannel-dynamic true
```

In this manner, your applications, referred to as clients, can connect with Red Hat build of Keycloak through your local network, while the server remains publicly accessible at **https://my.keycloak.org**.

7.4. USING EDGE TLS TERMINATION

As you can observe, the HTTPS protocol is the default choice, adhering to Red Hat build of Keycloak's commitment to security best practices. However, Red Hat build of Keycloak also provides the flexibility for users to opt for HTTP if necessary. This can be achieved simply by specifying the HTTP listener, consult the Configuring TLS for details. With an edge TLS-termination proxy you can start the server as follows:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org --http-enabled true
```

The result of this configuration is that you can continue to access Red Hat build of Keycloak at **https://my.keycloak.org** via HTTPS, while the proxy interacts with the instance using HTTP and port **8080**.

7.5. USING A REVERSE PROXY

When a proxy is forwarding http or reencrypted TLS requests, the **proxy-headers** option should be set. Depending on the hostname settings, some or all of the URL, may be dynamically determined.

WARNING

If either **forwarded** or **xforwarded** is selected, make sure your reverse proxy properly sets and overwrites the **Forwarded** or **X-Forwarded-*** headers respectively. To set these headers, consult the documentation for your reverse proxy. Misconfiguration will leave Red Hat build of Keycloak exposed to security vulnerabilities.

7.5.1. Fully dynamic URLs.

For example if your reverse proxy correctly sets the Forwarded header, and you don't want to hardcode the hostname, Red Hat build of Keycloak can accommodate this. You simply need to initiate the server as follows:

```
bin/kc.[sh|bat] start --hostname-strict false --proxy-headers forwarded
```

With this configuration, the server respects the value set by the Forwarded header. This also implies that all endpoints are dynamically resolved.

7.5.2. Partially dynamic URLs

The **proxy-headers** option can be also used to resolve the URL partially dynamically when the **hostname** option is not specified as a full URL. For example:

```
bin/kc.[sh|bat] start --hostname my.keycloak.org --proxy-headers xforwarded
```

In this case, scheme, and port are resolved dynamically from X-Forwarded-* headers, while hostname is statically defined as **my.keycloak.org**.

7.5.3. Fixed URLs

The **proxy-headers** is still relevant even when the **hostname** is set to a full URL as the headers are used to determine the origin of the request. For example:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org --proxy-headers xforwarded
```

In this case, while nothing is dynamically resolved from the X-Forwarded-* headers, the X-Forwarded-* headers are used to determine the correct origin of the request.

7.6. EXPOSING THE ADMINISTRATION CONSOLE ON A SEPARATE HOSTNAME

If you wish to expose the Admin Console on a different host, you can do so with the following command:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org --hostname-admin
https://admin.my.keycloak.org:8443
```

This allows you to access Red Hat build of Keycloak at **https://my.keycloak.org** and the Admin Console at **https://admin.my.keycloak.org:8443**, while the backend continues to use **https://my.keycloak.org**.

NOTE

Keep in mind that hostname and proxy options do not change the ports on which the server listens. Instead it changes only the ports of static resources like JavaScript and CSS links, OIDC well-known endpoints, redirect URIs, etc. that will be used in front of the proxy. You need to use HTTP configuration options to change the actual ports the server is listening on. Refer to the All configuration for details.

WARNING

Using the **hostname-admin** option does not prevent accessing the Administration REST API endpoints via the frontend URL specified by the **hostname** option. If you want to restrict access to the Administration REST API, you need to do it on the reverse proxy level. Administration Console implicitly accesses the API using the URL as specified by the **hostname-admin** option.

7.7. BACKGROUND - SERVER ENDPOINTS

Red Hat build of Keycloak exposes several endpoints, each with a different purpose. They are typically used for communication among applications or for managing the server. We recognize 3 main endpoint groups:

- Frontend

- Backend
- Administration

If you want to work with either of these endpoints, you need to set the base URL. The base URL consists of a several parts:

- a scheme (e.g. https protocol)
- a hostname (e.g. example.keycloak.org)
- a port (e.g. 8443)
- a path (e.g. /auth)

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**.

7.7.1. Frontend

Users and applications use the frontend URL to access Red Hat build of Keycloak through a front channel. The front channel is a publicly accessible communication channel. For example browser-based flows (accessing the login page, clicking on the link to reset a password or binding the tokens) can be considered as frontchannel requests.

In order to make Red Hat build of Keycloak accessible via the frontend URL, you need to set the **hostname** option:

```
bin/kc.[sh|bat] start --hostname my.keycloak.org
```

7.7.2. Backend

The backend endpoints are those accessible through a public domain or through a private network. They’re related to direct backend communication between Red Hat build of Keycloak and a client (an application secured by Red Hat build of Keycloak). Such communication might be over a local network, avoiding a reverse proxy. Examples of the endpoints that belong to this group are the authorization endpoint, token and token introspection endpoint, userinfo endpoint, JWKS URI endpoint, etc.

The default value of **hostname-backchannel-dynamic** option is **false**, which means that the backchannel URLs are same as the frontchannel URLs. Dynamic resolution of backchannel URLs from incoming request headers can be enabled by setting the following options:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org --hostname-backchannel-dynamic true
```

Note that **hostname** option must be set to a URL. For more information, refer to the Section 7.9, “Validations” section below.

7.7.3. Administration

Similarly to the base frontend URL, you can also set the base URL for resources and endpoints of the administration console. The server exposes the administration console and static resources using a specific URL. This URL is used for redirect URLs, loading resources (CSS, JS), Administration REST API etc. It can be done by setting the **hostname-admin** option:

```
bin/kc.[sh|bat] start --hostname https://my.keycloak.org --hostname-admin
https://admin.my.keycloak.org:8443
```

Again, the **hostname** option must be set to a URL. For more information, refer to the Section 7.9, “Validations” section below.

7.8. SOURCES FOR RESOLVING THE URL

As indicated in the previous sections, URLs can be resolved in several ways: they can be dynamically generated, hardcoded, or a combination of both:

- Dynamic from an incoming request:
  - Host header, scheme, server port, context path
  - Proxy-set headers: **Forwarded** and **X-Forwarded-***
- Hardcoded:
  - Server-wide config (e.g **hostname**, **hostname-admin**, etc.)
  - Realm configuration for frontend URL

7.9. VALIDATIONS

- **hostname** URL and **hostname-admin** URL are verified that full URL is used, incl. scheme and hostname. Port is validated only if present, otherwise default port for given protocol is assumed (80 or 443).
- In production profile (**kc.sh|bat start**), either **--hostname** or **--hostname-strict false** must be explicitly configured.
  - This does not apply for dev profile (**kc.sh|bat start-dev**) where **--hostname-strict false** is the default value.
- If **--hostname** is not configured:
  - **hostname-backchannel-dynamic** must be set to false.
  - **hostname-strict** must be set to false.
- If **hostname-admin** is configured, **hostname** must be set to a URL (not just hostname). Otherwise Red Hat build of Keycloak would not know what is the correct frontend URL (incl. port etc.) when accessing the Admin Console.
- If **hostname-backchannel-dynamic** is set to true, **hostname** must be set to a URL (not just hostname). Otherwise Red Hat build of Keycloak would not know what is the correct frontend URL (incl. port etc.) when being access via the dynamically resolved bachchannel.

Additionally if hostname is configured, then hostname-strict is ignored.

7.10. 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
```

After Red Hat build of Keycloak starts properly, open your browser and go to: **http://mykeycloak:8080/realms/<your-realm>/hostname-debug**

7.11. RELEVANT OPTIONS

Table 7.1. By default, this endpoint is disabled (--hostname-debug=false)

	Value
hostname Address at which is the server exposed. Can be a full URL, or just a hostname. When only hostname is provided, scheme, port and context path are resolved from the request. CLI: **--hostname** Env: **KC_HOSTNAME** Available only when hostname:v2 feature is enabled	
hostname-admin Address for accessing the administration console. Use this option if you are exposing the administration console using a reverse proxy on a different address than specified in the **hostname** option. CLI: **--hostname-admin** Env: **KC_HOSTNAME_ADMIN** Available only when hostname:v2 feature is enabled	
hostname-backchannel-dynamic Enables dynamic resolving of backchannel URLs, including hostname, scheme, port and context path. Set to true if your application accesses Keycloak via a private network. If set to true, **hostname** option needs to be specified as a full URL. CLI: **--hostname-backchannel-dynamic** Env: **KC_HOSTNAME_BACKCHANNEL_DYNAMIC** Available only when hostname:v2 feature is enabled	**true**, **false** (default)

	Value
hostname-debug Toggles the hostname debug page that is accessible at /realms/master/hostname-debug. CLI: **--hostname-debug** Env: **KC_HOSTNAME_DEBUG** Available only when hostname:v2 feature is enabled	**true**, **false** (default)
hostname-strict Disables dynamically resolving the hostname from request headers. Should always be set to true in production, unless your reverse proxy overwrites the Host header. If enabled, the **hostname** option needs to be specified. CLI: **--hostname-strict** Env: **KC_HOSTNAME_STRICT** Available only when hostname:v2 feature is enabled	**true** (default), **false**

CHAPTER 8. CONFIGURING A REVERSE PROXY

Configure Red Hat build of Keycloak with a reverse proxy, API gateway, or load balancer.

Distributed environments frequently require the use of a reverse proxy. Red Hat build of Keycloak offers several options to securely integrate with such environments.

8.1. PORT TO BE PROXIED

Red Hat build of Keycloak runs on the following ports by default:

- **8443** (**8080** when you enable HTTP explicitly by **--http-enabled=true**)
- **9000**

The port **8443** (or **8080** if HTTP is enabled) is used for the Admin UI, Account Console, SAML and OIDC endpoints and the Admin REST API as described in the Configuring the hostname (v2) chapter.

The port **9000** is used for management, which includes endpoints for health checks and metrics as described in the Configuring the Management Interface chapter.

You only need to proxy port **8443** (or **8080**) even when you use different host names for frontend/backend and administration as described at Configuring Red Hat build of Keycloak for production. You should not proxy port **9000** as health checks and metrics use those ports directly, and you do not want to expose this information to external callers.

8.2. CONFIGURE THE REVERSE PROXY HEADERS

Red Hat build of Keycloak will parse the reverse proxy headers based on the **proxy-headers** option which accepts several values:

- By default if the option is not specified, no reverse proxy headers are parsed. This should be used when no proxy is in use or with https passthrough.
- **forwarded** enables parsing of the **Forwarded** header as per RFC7239.
- **xforwarded** enables parsing of non-standard **X-Forwarded-*** headers, such as **X-Forwarded-For**, **X-Forwarded-Proto**, **X-Forwarded-Host**, and **X-Forwarded-Port**.

NOTE

If you are using a reverse proxy for anything other than https passthrough and do not set the **proxy-headers** option, then by default you will see 403 Forbidden responses to requests via the proxy that perform origin checking.

For example:

```
bin/kc.[sh|bat] start --proxy-headers forwarded
```

WARNING

If either **forwarded** or **xforwarded** is selected, make sure your reverse proxy properly sets and overwrites the **Forwarded** or **X-Forwarded-*** headers respectively. To set these headers, consult the documentation for your reverse proxy. Do not use **forwarded** or **xforwarded** with https passthrough. Misconfiguration will leave Red Hat build of Keycloak exposed to security vulnerabilities.

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.

NOTE

When using the **xforwarded** setting, the **X-Forwarded-Port** takes precedence over any port included in the **X-Forwarded-Host**.

NOTE

If the TLS connection is terminated at the reverse proxy (edge termination), enabling HTTP through the **http-enabled** setting is required.

8.3. 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 a full URL for the **hostname** option in these cases, for example using **--hostname=https://my.keycloak.org/auth** if Red Hat build of Keycloak is exposed through the reverse proxy on **/auth**.

For more details on exposing Red Hat build of Keycloak on different hostname or context-path incl. Administration REST API and Console, see Configuring the hostname (v2).

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.

8.4. 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 limited number of owners. That means that session related data are stored only in some cluster nodes and the other nodes need to lookup the data 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 criteria (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 stick over the AUTH_SESSION_ID cookie. The appropriate procedure to make this change depends 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.

8.5. EXPOSED PATH RECOMMENDATIONS

When using a reverse proxy, Red Hat build of Keycloak only requires certain paths 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.
/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.
/metrics	-	No	Exposed metrics lead to an unnecessary attack vector.
/health	-	No	Exposed health checks lead to an unnecessary attack vector.

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.

8.6. TRUSTED PROXIES

To ensure that proxy headers are used only from proxies you trust, set the **proxy-trusted-addresses** option to a comma separated list of IP addresses (IPv4 or IPv6) or Classless Inter-Domain Routing (CIDR) notations.

For example:

```
bin/kc.[sh|bat] start --proxy-headers forwarded --proxy-trusted-addresses=192.168.0.32,127.0.0.0/8
```

8.7. PROXY PROTOCOL

The **proxy-protocol-enabled** option controls whether the server should use the HA PROXY protocol when serving requests from behind a proxy. When set to true, the remote address returned will be the one from the actual connecting client. The value cannot be **true** when using the **proxy-headers** option.

This is useful when running behind a compatible https passthrough proxy because the request headers cannot be manipulated.

For example:

```
bin/kc.[sh|bat] start --proxy-protocol-enabled true
```

8.8. 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.

WARNING

Client certificate lookup via a proxy header for X.509 authentication is considered security-sensitive. If misconfigured, a forged client certificate header can be used for authentication. **Extra precautions need to be taken to ensure that the client certificate information can be trusted when passed via a proxy header.**

- Double check your use case needs reencrypt or edge TLS termination which implies using a proxy header for client certificate lookup. TLS passthrough is recommended as a more secure option when X.509 authentication is desired as it does not require passing the certificate via a proxy header. Client certificate lookup from a proxy header is applicable only to reencrypt and edge TLS termination.
- If passthrough is not an option, implement the following security measures:
    - Configure your network so that Red Hat build of Keycloak is isolated and can accept connections only from the proxy.
    - Make sure that the proxy overwrites the header that is configured in **spi-x509cert-lookup-<provider>-ssl-client-cert** option.
    - Pay extra attention to the **spi-x509cert-lookup-<provider>-trust-proxy-verification** setting. Make sure you enable it only if you can trust your proxy to verify the client certificate. Setting **spi-x509cert-lookup-<provider>-trust-proxy-verification=true** without the proxy verifying the client certificate chain will expose Red Hat build of Keycloak to security vulnerability when a forged client certificate can be used for authentication.

The server supports some of the most commons 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.

Option	Description
cert-is-url-encoded	Whether the forwarded certificate is url-encoded or not. In NGINX, this corresponds to the **$ssl_client_cert** and **$ssl_client_escaped_cert** variables. This can also be used for the Traefik PassTlsClientCert middleware, as it sends the client certficate unencoded.

8.8.1. 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 how to configure a Red Hat build of Keycloak Truststore.

8.9. RELEVANT OPTIONS

	Value
hostname Address at which is the server exposed. Can be a full URL, or just a hostname. When only hostname is provided, scheme, port and context path are resolved from the request. CLI: **--hostname** Env: **KC_HOSTNAME** Available only when hostname:v2 feature is enabled	
hostname-admin Address for accessing the administration console. Use this option if you are exposing the administration console using a reverse proxy on a different address than specified in the **hostname** option. CLI: **--hostname-admin** Env: **KC_HOSTNAME_ADMIN** Available only when hostname:v2 feature is enabled	
http-relative-path ▮ Set the path relative to / for serving resources. The path must start with a /. CLI: **--http-relative-path** Env: **KC_HTTP_RELATIVE_PATH**	/ (default)

	Value
proxy-headers The proxy headers that should be accepted by the server. Misconfiguration might leave the server exposed to security vulnerabilities. Takes precedence over the deprecated proxy option. CLI: **--proxy-headers** Env: **KC_PROXY_HEADERS**	**forwarded**, **xforwarded**
proxy-protocol-enabled Whether the server should use the HA PROXY protocol when serving requests from behind a proxy. When set to true, the remote address returned will be the one from the actual connecting client. Cannot be enabled when the **proxy-headers** is used. CLI: **--proxy-protocol-enabled** Env: **KC_PROXY_PROTOCOL_ENABLED**	**true**, **false** (default)
proxy-trusted-addresses A comma separated list of trusted proxy addresses. If set, then proxy headers from other addresses will be ignored. By default all addresses are trusted. A trusted proxy address is specified as an IP address (IPv4 or IPv6) or Classless Inter-Domain Routing (CIDR) notation. Available only when proxy-headers is set. CLI: **--proxy-trusted-addresses** Env: **KC_PROXY_TRUSTED_ADDRESSES**	

CHAPTER 9. CONFIGURING THE DATABASE

Configure a relational database for Red Hat build of Keycloak to store user, client, and realm data.

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

9.1. 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**	11.4
Microsoft SQL Server	**mssql**	2022
MySQL	**mysql**	8.4
Oracle Database	**oracle**	23.5
PostgreSQL	**postgres**	17
Amazon Aurora PostgreSQL	**postgres**	16.8

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.

9.2. 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.

Install the necessary missing driver manually 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.

9.2.1. Installing the Oracle Database driver

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

1. Download the **ojdbc17** and **orai18n** JAR files from one of the following sources:
    a. **Zipped JDBC driver and Companion Jars**version 23.6.0.24.10 from the Oracle driver download page.
    b. Maven Central via **ojdbc17** and **orai18n**.

c. Installation media recommended by the database vendor for the specific database in use.

2. When running the unzipped distribution: Place the **ojdbc17** and **orai18n** JAR files in Red Hat build of Keycloak's **providers** folder

3. 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 Operator, those images need to be optimized images with all build-time options of Red Hat build of Keycloak set.
   A minimal Containerfile 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-rhel9:26.2
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc17/23.6.0.24.10/ojdbc17-23.
6.0.24.10.jar /opt/keycloak/providers/ojdbc17.jar
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/oracle/database/nls/orai18n/23.6.0.24.10/orai18n-
23.6.0.24.10.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 Red Hat build of Keycloak Operator, it needs to be
optimized, which requires Red Hat build of Keycloak'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.

9.2.2. Installing the Microsoft SQL Server driver

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

1. Download the **mssql-jdbc** JAR file from one of the following sources:
   a. Download a version from the Microsoft JDBC Driver for SQL Server page .
   b. Maven Central via **mssql-jdbc**.
   c. Installation media recommended by the database vendor for the specific database in use.

2. When running the unzipped distribution: Place the **mssql-jdbc** in Red Hat build of Keycloak's **providers** folder

3. 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 Containerfile 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-rhel9:26.2
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/microsoft/sqlserver/mssql-jdbc/12.8.2.jre11/mssql-
jdbc-12.8.2.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 Red Hat build of Keycloak Operator, it needs to be
optimized, which requires Red Hat build of Keycloak'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.

9.3. 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.

The configuration can be set during a **build** command OR a **start** command:

1. Using a **build** command followed by an optimized **start** command (recommended)
   First, the minimum settings needed to connect to the database can be specified in **conf/keycloak.conf**:

   ```
   # The database vendor.
   db=postgres

   # The username of the database user.
   db-username=keycloak

   # The password of the database user.
   db-password=change_me

   # Sets the hostname of the default JDBC URL of the chosen vendor
   db-url-host=keycloak-postgres
   ```

   Then, the following commands create a new and optimized server image based on the configuration options and start the server.

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

2. Using **only a start** command (without **--optimized**)

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

WARNING

The examples above include the minimum settings needed to connect to the database but it exposes the database password and is not recommended. Use the **conf/keycloak.conf** as shown above, environment variables, or keystore for at least the password.

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

It is also possible to configure the database when Importing and exporting realms or Bootstrapping and recovering an admin account:

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

For more information, see Configuring Red Hat build of Keycloak .

9.4. 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.

9.5. 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.

9.6. 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.

9.6.1. 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.

9.6.2. 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.

9.6.3. 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**

9.6.4. 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';
   ```

9.7. PREPARING FOR AMAZON AURORA POSTGRESQL

When using Amazon Aurora PostgreSQL, the Amazon Web Services JDBC Driver offers additional features like transfer of database connections when a writer instance changes in a Multi-AZ setup. This driver is not part of the distribution and needs to be installed before it can be used.

To install this driver, apply the following steps:

1. When running the unzipped distribution: Download the JAR file from the Amazon Web Services JDBC Driver releases page and place it in Red Hat build of Keycloak's **providers** folder.
2. When running containers: Build a custom Red Hat build of Keycloak image and add the JAR in the **providers** folder.
   A minimal Containerfile to build an image which can be used with the Red Hat build of Keycloak Operator looks like the following:

   ```
   FROM registry.redhat.io/rhbk/keycloak-rhel9:26.2
   ADD --chmod=0666 https://github.com/awslabs/aws-advanced-jdbc-
   wrapper/releases/download/2.3.1/aws-advanced-jdbc-wrapper-2.3.1.jar
   /opt/keycloak/providers/aws-advanced-jdbc-wrapper.jar
   ```

See the Running Red Hat build of Keycloak in a container chapter for details on how to build optimized images, and the Using custom Red Hat build of Keycloak images chapter on how to run optimized and non-optimized images with the Red Hat build of Keycloak Operator.

3. Configure Red Hat build of Keycloak to run with the following parameters:

db-url

Insert **aws-wrapper** to the regular PostgreSQL JDBC URL resulting in a URL like **jdbc:aws-wrapper:postgresql://...**.

db-driver

Set to **software.amazon.jdbc.Driver** to use the AWS JDBC wrapper.

9.8. PREPARING FOR MYSQL SERVER

Beginning with MySQL 8.0.30, MySQL supports generated invisible primary keys for any InnoDB table that is created without an explicit primary key (more information here). If this feature is enabled, the database schema initialization and also migrations will fail with the error message **Multiple primary key defined (1068)**. You then need to disable it by setting the parameter **sql_generate_invisible_primary_key** to **OFF** in your MySQL server configuration before installing or upgrading Red Hat build of Keycloak.

9.9. 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
```

9.10. USING DATABASE VENDORS WITH XA TRANSACTION SUPPORT

Red Hat build of Keycloak uses non-XA transactions and the appropriate database drivers by default.

If you wish to use the XA transaction support offered by your driver, enter the following command:

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

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

NOTE

Certain vendors, such as Azure SQL and MariaDB Galera, do not support or rely on the XA transaction mechanism.

XA recovery defaults to enabled and will use the file system location **KEYCLOAK_HOME/data/transaction-logs** to store transaction logs.

NOTE

Enabling XA transactions in a containerized environment does not fully support XA recovery unless stable storage is available at that path.

9.11. 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-quarkus-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-quarkus-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-quarkus-migration-export=<path>/<file.sql>
```

For more information, check the Migrating the database documentation.

9.12. RELEVANT OPTIONS

	Value
db The database vendor. In production mode the default value of **dev-file** is deprecated, you should explicitly specify the db instead. CLI: **--db** Env: **KC_DB**	**dev-file** (default), **dev-mem**, **mariadb**, **mssql**, **mysql**, **oracle**, **postgres**

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

	Value
db-url The full database JDBC URL. 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**. CLI: **--db-url** Env: **KC_DB_URL**	
db-url-database Sets the database name of the default JDBC URL of the chosen vendor. If the **db-url** option is set, this option is ignored. CLI: **--db-url-database** Env: **KC_DB_URL_DATABASE**	
db-url-host Sets the hostname of the default JDBC URL of the chosen vendor. If the **db-url** option is set, this option is ignored. CLI: **--db-url-host** Env: **KC_DB_URL_HOST**	
db-url-port Sets the port of the default JDBC URL of the chosen vendor. If the **db-url** option is set, this option is ignored. CLI: **--db-url-port** Env: **KC_DB_URL_PORT**	
db-url-properties Sets the properties of the default JDBC URL of the chosen vendor. 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. CLI: **--db-url-properties** Env: **KC_DB_URL_PROPERTIES**	

	Value
db-username The username of the database user. CLI: **--db-username** Env: **KC_DB_USERNAME**	
transaction-xa-enabled ▮ If set to true, XA datasources will be used. CLI: **--transaction-xa-enabled** Env: **KC_TRANSACTION_XA_ENABLED**	**true**, **false** (default)

CHAPTER 10. CONFIGURING DISTRIBUTED CACHES

Configure the caching layer to cluster multiple Red Hat build of Keycloak instances and to increase performance.

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.

10.1. 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 use the **jdbc-ping** stack which is based upon a TCP transport and uses the configured database to track nodes joining the cluster. 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] start --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.

10.2. 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_configuring_docs}[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
crl	Local	Cache for X.509 authenticator CRLs

Cache name	Cache Type	Description
work	Replicated	Propagate invalidation messages across nodes
authenticationSessions	Distributed	Caches authentication sessions, created/destroyed/expired during the authentication process
sessions	Distributed	Cache persisted user session data
clientSessions	Distributed	Cache persisted client session data
offlineSessions	Distributed	Cache persisted offline user session data
offlineClientSessions	Distributed	Cache persisted offline client session data
loginFailures	Distributed	keep track of failed logins, fraud detection
actionTokens	Distributed	Caches action Tokens

10.2.1. 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, 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 session data are stored in the database by default and loaded on-demand to the following caches:

- **sessions**
- **clientSessions**

By relying on a distributable cache, cached user and client sessions are available to any node in the cluster so that users can be redirected to any node without the need to load session data from the database. 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.

These in-memory caches for user sessions and client sessions are limited to, by default, 10000 entries per node which reduces the overall memory usage of Red Hat build of Keycloak for larger installations. The internal caches will run with only a single owner for each cache entry.

Offline user sessions

As an OpenID Connect Provider, the server is capable of authenticating users and issuing offline tokens. When issuing an offline token after successful authentication, the server creates an offline user session and offline client session.

The following caches are used to store offline sessions:

- offlineSessions
- offlineClientSessions

Like the user and client sessions caches, the offline user and client session caches are limited to 10000 entries per node by default. Items which are evicted from the memory will be loaded on-demand from the database when needed.

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.

10.2.2. Volatile user sessions

By default, regular user sessions are stored in the database and loaded on-demand to the cache. It is possible to configure Red Hat build of Keycloak to store regular user sessions in the cache only and minimize calls to the database.

Since all the sessions in this setup are stored in-memory, there are two side effects related to this:

- Losing sessions when all Red Hat build of Keycloak nodes restart.
- Increased memory consumption.

When using volatile user sessions, the cache is the source of truth for user and client sessions. Red Hat build of Keycloak automatically adjusts the number of entries that can be stored in memory, and increases the number of copies to prevent data loss.

WARNING

It is not recommended to use volatile user sessions when using offline sessions extensively due to potentially high memory usage. For volatile sessions, the time offline sessions are cached in memory can be limited with the SPI options **spi-user-sessions—infinispan—offline-client-session-cache-entry-lifespan-override** and **spi-user-sessions—infinispan—offline-session-cache-entry-lifespan-override**.

Follow these steps to enable this setup:

1. Disable **persistent-user-sessions** feature using the following command:

```
bin/kc.sh start --features-disabled=persistent-user-sessions ...
```

NOTE

Disabling **persistent-user-sessions** is not possible when **multi-site** feature is enabled.

10.2.3. Configuring cache maximum size

In order to reduce memory usage, it's possible to place an upper bound on the number of entries which are stored in a given cache. To specify an upper bound of on a cache, you must provide the following command line argument **--cache-embedded-${CACHE_NAME}-max-count=**, with **${CACHE_NAME}** replaced with the name of the cache you would like to apply the upper bound to. For example, to apply an upper-bound of **1000** to the **offlineSessions** cache you would configure **--cache-embedded-offline-sessions-max-count=1000**. An upper bound can not be defined on the following caches: **actionToken**, **authenticationSessions**, **loginFailures**, **work**.

Setting a maximum cache size for **sessions**, **clientSessions**, **offlineSessions** and **offlineClientSessions** is not supported when volatile sessions are enabled.

10.2.4. Specify your own cache configuration file

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

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

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

10.2.5. CLI options for remote server

For configuration of Red Hat build of Keycloak server for high availability and multi-node clustered setup there was introduced following CLI options **cache-remote-host**, **cache-remote-port**, **cache-remote-username** and **cache-remote-password** simplifying configuration within the XML file. Once any of the declared CLI parameters are present, it is expected there is no configuration related to remote store present in the XML file.

10.2.5.1. Connecting to an insecure Infinispan server

WARNING

Disabling security is not recommended in production!

In a development or test environment, it is easier to start an unsecured Infinispan server. For these use case, the CLI options **cache-remote-tls-enabled** disables the encryption (TLS) between Red Hat build of Keycloak and Data Grid. Red Hat build of Keycloak will fail to start if the Data Grid server is configured to accept only encrypted connections.

The CLI options **cache-remote-username** and **cache-remote-password** are optional and, if not set, Red Hat build of Keycloak will connect to the Data Grid server without presenting any credentials. If the Data Grid server has authentication enabled, Red Hat build of Keycloak will fail to start.

10.3. TRANSPORT STACKS

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

- **jdbc-ping**
- **kubernetes**
- **jdbc-ping-udp** (deprecated)
- **tcp** (deprecated)
- **udp** (deprecated)
- **ec2** (deprecated)
- **azure** (deprecated)
- **google** (deprecated)

To apply a specific cache stack, enter this command:

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

The default stack is set to **jdbc-ping** when distributed caches are enabled, which is backwards compatible with the defaults in the 26.x release stream of Red Hat build of Keycloak.

10.3.1. Available transport stacks

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

Stack name	Transport protocol	Discovery
jdbc-ping	TCP	Database registry using the JGroups **JDBC_PING2** protocol.
jdbc-ping-udp (deprecated)	UDP	Database registry using the JGroups **JDBC_PING2** protocol.

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

Stack name	Transport protocol	Discovery
kubernetes	TCP	DNS resolution using the JGroups **DNS_PING** protocol. It requires to set **jgroups.dns.query** to the headless service FQDN.

Stack name	Transport protocol	Discovery
tcp (deprecated)	TCP	IP multicast using the JGroups **MPING** protocol. See below on how to configure a unique **jgroups.mcast_addr** or **jgroups.mcast_port** for each cluster.
udp (deprecated)	UDP	IP multicast using the JGroups **PING** protocol. See below on how to configure a unique **jgroups.mcast_addr** or **jgroups.mcast_port** for each cluster.

When using the **tcp**, **udp** or **jdbc-ping-udp** stack, each cluster must use a different multicast address and/or port so that their nodes form distinct clusters. By default, Red Hat build of Keycloak uses **239.6.7.8** as multicast address for **jgroups.mcast_addr** and **46655** for the multicast port **jgroups.mcast_port**.

NOTE

Use **-D<property>=<value>** to pass the properties via the **JAVA_OPTS_APPEND** environment variable or in the CLI command.

Additional Stacks

It is recommended to use one of the stacks available above. Additional stacks are provided by Infinispan, but it is outside the scope of this guide how to configure them. Please refer to Setting up Infinispan cluster transport and Customizing JGroups stacks for further documentation.

10.4. SECURING TRANSPORT STACKS

Encryption using TLS is enabled by default for TCP-based transport stacks, which is also the default configuration. No additional CLI options or modifications of the cache XML are required as long as you are using a TCP-based transport stack.

NOTE

If you are using a transport stack based on **UDP** or **TCP_NIO2**, proceed as follows to configure the encryption of the transport stack:

1. Set the option **cache-embedded-mtls-enabled** to **false**.
2. Follow the documentation in JGroups Encryption documentation and Encrypting cluster transport.

With TLS enabled, Red Hat build of Keycloak auto-generates a self-signed RSA 2048 bit certificate to secure the connection and uses TLS 1.3 to secure the communication. The keys and the certificate are stored in the database so they are available to all nodes. By default, the certificate is valid for 60 days and is rotated at runtime every 30 days. Use the option **cache-embedded-mtls-rotation-interval-days** to change this.

10.4.1. Running inside a service mesh

When using a service mesh like Istio, you might need to allow a direct mTLS communication between the Red Hat build of Keycloak Pods to allow for the mutual authentication to work. Otherwise, you might see error messages like **JGRP000006: failed accepting connection from peer SSLSocket** that indicate that a wrong certificate was presented, and the cluster will not form correctly.

You then have the option to allow direct mTLS communication between the Red Hat build of Keycloak Pods, or rely on the service mesh transport security to encrypt the communication and to authenticate peers.

To allow direct mTLS communication for Red Hat build of Keycloak when using Istio:

- Apply the following configuration to allow direct communication.

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: infinispan-allow-nomtls
spec:
  selector:
    matchLabels:
      app: keycloak (1)
  portLevelMtls:
    "7800": (2)
      mode: PERMISSIVE
```

(1) Update the labels to match your Red Hat build of Keycloak deployment.

(2) Port 7800 is the default. Adjust it if you change the data transmission port.

As an alternative, to disable the mTLS communication, and rely on the service mesh to encrypt the traffic:

- Set the option **cache-embedded-mtls-enabled** to **false**.
- Configure your service mesh to authorize only traffic from other Red Hat build of Keycloak Pods for the data transmission port (default: 7800).

10.4.2. Providing your own keys and certificates

Although not recommended for standard setups, if it is essential in a specific setup, you can configure the keystore with the certificate for the transport stack manually. **cache-embedded-mtls-key-store-file** sets the path to the keystore, and **cache-embedded-mtls-key-store-password** sets the password to decrypt it. The truststore contains the valid certificates to accept connection from, and it can be configured with **cache-embedded-mtls-trust-store-file** (path to the truststore), and **cache-embedded-mtls-trust-store-password** (password to decrypt it). To restrict unauthorized access, always use a self-signed certificate for each Red Hat build of Keycloak deployment.

10.5. NETWORK PORTS

To ensure a healthy Red Hat build of Keycloak clustering, some network ports need to be open. The table below shows the TCP ports that need to be open for the **jdbc-ping** stack, and a description of the traffic that goes through it.

Port	Property	Description
7800	**jgroups.bind.port**	Unicast data transmission.
57800	**jgroups.fd.port-offset**	Failure detection by protocol **FD_SOCK2**. It listens to the abrupt closing of a socket to suspect a Red Hat build of Keycloak server failure. The **jgroups.fd.port-offset** property defines the offset from the **jgroups.bind.port**.

NOTE

Use **-D<property>=<value>** to modify the ports above in your **JAVA_OPTS_APPEND** environment variable or in your CLI command.

10.6. NETWORK BIND ADDRESS

To ensure a healthy Red Hat build of Keycloak clustering, the network port must be bound on an interface that is accessible from all other nodes of the cluster.

By default, it picks a site local (non-routable) IP address, for example, from the 192.168.0.0/16 or 10.0.0.0/8 address range.

To override the address, set the **jgroups.bind.address** property.

NOTE

Use **-Djgroups.bind.address=<IP>** to modify the bind address in your **JAVA_OPTS_APPEND** environment variable or in your CLI command.

To set up for IPv6 only and have Red Hat build of Keycloak pick the bind address automatically, use the following settings:

```
export JAVA_OPTS_APPEND="-Djava.net.preferIPv4Stack=false -
Djava.net.preferIPv6Addresses=true"
```

10.7. RUNNING INSTANCES ON DIFFERENT NETWORKS

If you run Red Hat build of Keycloak instances on different networks, for example behind firewalls or in containers, the different instances will not be able to reach each other by their local IP address. In such a case, set up a port forwarding rule (sometimes called "virtual server") to their local IP address.

When using port forwarding, use the following properties so each node correctly advertises its external address to the other nodes:

Property	Description
jgroups.external_port	Port that other instances in the Red Hat build of Keycloak cluster should use to contact this node.
jgroups.external_addr	IP address that other instances in the Red Hat build of Keycloak should use to contact this node.

NOTE

Use **-D<property>=<value>** set this in your **JAVA_OPTS_APPEND** environment variable or in your CLI command.

10.8. EXPOSING METRICS FROM CACHES

Metrics from caches are automatically exposed when the metrics are enabled.

To enable histograms for the cache metrics, set **cache-metrics-histograms-enabled** to **true**. While these metrics provide more insights into the latency distribution, collecting them might have a performance impact, so you should be cautious to activate them in an already saturated system.

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

For more details about how to enable metrics, see Gaining insights with metrics.

10.9. RELEVANT OPTIONS

	Value
cache Defines the cache mechanism for high-availability. 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. CLI: **--cache** Env: **KC_CACHE**	**ispn** (default), **local**

	Value
cache-config-file Defines the file from which cache configuration should be loaded from. The configuration file is relative to the **conf/** directory. CLI: **--cache-config-file** Env: **KC_CACHE_CONFIG_FILE**	
cache-metrics-histograms-enabled Enable histograms for metrics for the embedded caches. CLI: **--cache-metrics-histograms-enabled** Env: **KC_CACHE_METRICS_HISTOGRAMS_ENABLED** Available only when metrics are enabled	**true**, **false** (default)
cache-stack Define the default stack to use for cluster communication and node discovery. Defaults to **jdbc-ping** if not set. CLI: **--cache-stack** Env: **KC_CACHE_STACK** Available only when 'cache' type is set to 'ispn' Use 'jdbc-ping' instead by leaving it unset **Deprecated values: azure, ec2, google, tcp, udp, jdbc-ping-udp**	**jdbc-ping**, **kubernetes**, **jdbc-ping-udp** (deprecated), **tcp** (deprecated), **udp** (deprecated), **ec2** (deprecated), **azure** (deprecated), **google** (deprecated), or any

10.9.1. Embedded Cache

	Value
cache-embedded-authorization-max-count The maximum number of entries that can be stored in-memory by the authorization cache. CLI: **--cache-embedded-authorization-max-count** Env: **KC_CACHE_EMBEDDED_AUTHORIZATION_MAX_COUNT**	

	Value
cache-embedded-client-sessions-max-count The maximum number of entries that can be stored in-memory by the clientSessions cache. CLI: **--cache-embedded-client-sessions-max-count** Env: **KC_CACHE_EMBEDDED_CLIENT_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	
cache-embedded-crl-max-count The maximum number of entries that can be stored in-memory by the crl cache. CLI: **--cache-embedded-crl-max-count** Env: **KC_CACHE_EMBEDDED_CRL_MAX_COUNT**	
cache-embedded-keys-max-count The maximum number of entries that can be stored in-memory by the keys cache. CLI: **--cache-embedded-keys-max-count** Env: **KC_CACHE_EMBEDDED_KEYS_MAX_COUNT**	
cache-embedded-mtls-enabled Encrypts the network communication between Keycloak servers. If no additional parameters about a keystore and truststore are provided, ephemeral key pairs and certificates are created and rotated automatically, which is recommended for standard setups. CLI: **--cache-embedded-mtls-enabled** Env: **KC_CACHE_EMBEDDED_MTLS_ENABLED** Available only when a TCP based cache-stack is used	**true** (default), **false**
cache-embedded-mtls-key-store-file The Keystore file path. The Keystore must contain the certificate to use by the TLS protocol. By default, it looks up **cache-mtls-keystore.p12** under conf/ directory. CLI: **--cache-embedded-mtls-key-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_FILE** Available only when property 'cache-embedded-mtls-enabled' is enabled	

	Value
cache-embedded-mtls-key-store-password The password to access the Keystore. CLI: **--cache-embedded-mtls-key-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_PASSWORD** Available only when property 'cache-embedded-mtls-enabled' is enabled	
cache-embedded-mtls-rotation-interval-days Rotation period in days of automatic JGroups MTLS certificates. CLI: **--cache-embedded-mtls-rotation-interval-days** Env: **KC_CACHE_EMBEDDED_MTLS_ROTATION_INTERVAL_DAYS** Available only when property 'cache-embedded-mtls-enabled' is enabled	**30** (default)
cache-embedded-mtls-trust-store-file The Truststore file path. It should contain the trusted certificates or the Certificate Authority that signed the certificates. By default, it lookup **cache-mtls-truststore.p12** under conf/ directory. CLI: **--cache-embedded-mtls-trust-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_FILE** Available only when property 'cache-embedded-mtls-enabled' is enabled	
cache-embedded-mtls-trust-store-password The password to access the Truststore. CLI: **--cache-embedded-mtls-trust-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_PASSWORD** Available only when property 'cache-embedded-mtls-enabled' is enabled	
cache-embedded-offline-client-sessions-max-count The maximum number of entries that can be stored in-memory by the offlineClientSessions cache. CLI: **--cache-embedded-offline-client-sessions-max-count** Env: **KC_CACHE_EMBEDDED_OFFLINE_CLIENT_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	

	Value
cache-embedded-offline-sessions-max-count The maximum number of entries that can be stored in-memory by the offlineSessions cache. CLI: **--cache-embedded-offline-sessions-max-count** Env: **KC_CACHE_EMBEDDED_OFFLINE_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	
cache-embedded-realms-max-count The maximum number of entries that can be stored in-memory by the realms cache. CLI: **--cache-embedded-realms-max-count** Env: **KC_CACHE_EMBEDDED_REALMS_MAX_COUNT**	
cache-embedded-sessions-max-count The maximum number of entries that can be stored in-memory by the sessions cache. CLI: **--cache-embedded-sessions-max-count** Env: **KC_CACHE_EMBEDDED_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	
cache-embedded-users-max-count The maximum number of entries that can be stored in-memory by the users cache. CLI: **--cache-embedded-users-max-count** Env: **KC_CACHE_EMBEDDED_USERS_MAX_COUNT**	

10.9.2. Remote Cache

	Value
cache-remote-host The hostname of the external Infinispan cluster. Available only when feature **multi-site**, **clusterless** or **cache-embedded-remote-store** is set. CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	

	Value
cache-remote-password The password for the authentication to the external Infinispan cluster. It is optional if connecting to an unsecure external Infinispan cluster. If the option is specified, **cache-remote-username** is required as well. CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD** Available only when remote host is set	
cache-remote-port The port of the external Infinispan cluster. CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT** Available only when remote host is set	**11222** (default)
cache-remote-tls-enabled Enable TLS support to communicate with a secured remote Infinispan server. Recommended to be enabled in production. CLI: **--cache-remote-tls-enabled** Env: **KC_CACHE_REMOTE_TLS_ENABLED** Available only when remote host is set	**true** (default), **false**
cache-remote-username The username for the authentication to the external Infinispan cluster. It is optional if connecting to an unsecure external Infinispan cluster. If the option is specified, **cache-remote-password** is required as well. CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME** Available only when remote host is set	

CHAPTER 11. CONFIGURING OUTGOING HTTP REQUESTS

Configure the client used for 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.

11.1. CONFIGURING TRUSTED CERTIFICATES FOR TLS CONNECTIONS

See Configuring trusted certificates 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.

11.2. 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 mTLS.

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 11.3, "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.

11.3. 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.

11.4. 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'
```

The backslash character \ is escaped again because micro-profile config is used to parse the array of mappings.

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.

11.5. RELEVANT OPTIONS

	Value
truststore-paths List of pkcs12 (p12, pfx, or pkcs12 file extensions), PEM files, or directories containing those files that will be used as a system truststore. CLI: **--truststore-paths** Env: **KC_TRUSTSTORE_PATHS**	

CHAPTER 12. CONFIGURING TRUSTED CERTIFICATES

Configure the Red Hat build of Keycloak Truststore to communicate through TLS.

When Red Hat build of Keycloak communicates with external services or has an incoming connection through TLS, it has to validate the remote 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 clients or servers, or the CA that signed these certificates, must be put in a truststore. This truststore is then configured for use by Red Hat build of Keycloak.

12.1. CONFIGURING THE SYSTEM TRUSTSTORE

The existing Java default truststore certs will always be trusted. If you need additional certificates, which will be the case if you have self-signed or internal certificate authorities that are not recognized by the JRE, they can be included in the **conf/truststores** directory or subdirectories. The certs may be in PEM files, or PKCS12 files with extension **.p12**, **.pfx**, or **.pkcs12**. If in PKCS12, the certs must be unencrypted - meaning no password is expected.

If you need an alternative path, use the **--truststore-paths** option to specify additional files or directories where PEM or PKCS12 files are located. Paths are relative to where you launched Red Hat build of Keycloak, so absolute paths are recommended instead. If a directory is specified, it will be recursively scanned for truststore files.

After all applicable certs are included, the truststore will be used as the system default truststore via the **javax.net.ssl** properties, and as the default for internal usage within Red Hat build of Keycloak.

For example:

```
bin/kc.[sh|bat] start --truststore-paths=/opt/truststore/myTrustStore.pfx,/opt/other-
truststore/myOtherTrustStore.pem
```

It is still possible to directly set your own **javax.net.ssl** truststore System properties, but it's recommended to use the **--truststore-paths** instead.

12.2. HOSTNAME VERIFICATION POLICY

You may refine how hostnames are verified by TLS connections with the **tls-hostname-verifier** property.

- **DEFAULT** (the default) allows wildcards in subdomain names (e.g. *.foo.com) to match names with the same number of levels (e.g. a.foo.com, but not a.b.foo.com) - with rules and exclusions for public suffixes based upon https://publicsuffix.org/list/
- **ANY** means that the hostname is not verified - this mode should not be used in production.
- **WILDCARD** (deprecated) allows wildcards in subdomain names (e.g. *.foo.com) to match anything, including multiple levels (e.g. a.b.foo.com). Use DEFAULT instead.
- **STRICT** (deprecated) allows wildcards in subdomain names (e.g. *.foo.com) to match names with the same number of levels (e.g. a.foo.com, but not a.b.foo.com) - with some limited exclusions. Use DEFAULT instead.
  Please note that this setting does not apply to LDAP secure connections, which require strict hostname checking.

12.3. RELEVANT OPTIONS

	Value
tls-hostname-verifier The TLS hostname verification policy for out-going HTTPS and SMTP requests. ANY should not be used in production. CLI: **--tls-hostname-verifier** Env: **KC_TLS_HOSTNAME_VERIFIER** STRICT and WILDCARD have been deprecated, use DEFAULT instead. Deprecated values: **STRICT**, **WILDCARD**	**ANY**, **WILDCARD** (deprecated), **STRICT** (deprecated), **DEFAULT** (default)
truststore-paths List of pkcs12 (p12, pfx, or pkcs12 file extensions), PEM files, or directories containing those files that will be used as a system truststore. CLI: **--truststore-paths** Env: **KC_TRUSTSTORE_PATHS**	

CHAPTER 13. CONFIGURING TRUSTED CERTIFICATES FOR MTLS

Configure Mutual TLS to verify clients that are connecting to Red Hat build of Keycloak.

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 and X.509 Authentication.

13.1. ENABLING MTLS

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 a 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.

WARNING

The mTLS configuration and the truststore is shared by all Realms. It is not possible to configure different truststores for different Realms.

NOTE

Management interface properties are inherited from the main HTTP server, including mTLS settings. It means when mTLS is set, it is also enabled for the management interface. To override the behavior, use the **https-management-client-auth** property.

13.2. USING A DEDICATED TRUSTSTORE FOR MTLS

By default, Red Hat build of Keycloak uses the System Truststore to validate certificates. See Configuring trusted certificates for details.

If you need to use a dedicated truststore for mTLS, 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 --https-trust-store-password=<value>
```

Recognized file extensions for a truststore:

- **.p12**, **.pkcs12**, and **.pfx** for a pkcs12 file

- **.jks**, and **.truststore** for a jks file
- **.ca**, **.crt**, and **.pem** for a pem file

If your truststore does not have an extension matching its file type, you will also need to set the **https-key-store-type** option.

13.3. ADDITIONAL RESOURCES

13.3.1. Using mTLS for outgoing HTTP requests

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.

13.3.2. Configuring X.509 Authentication

For more information on how to configure X.509 Authentication, see X.509 Client Certificate User Authentication section.

13.4. RELEVANT OPTIONS

	Value
https-client-auth ▮ Configures the server to require/request client authentication. CLI: **--https-client-auth** Env: **KC_HTTPS_CLIENT_AUTH**	**none** (default), **request**, **required**
https-trust-store-file The trust store which holds the certificate information of the certificates to trust. CLI: **--https-trust-store-file** Env: **KC_HTTPS_TRUST_STORE_FILE**	
https-trust-store-password The password of the trust store file. CLI: **--https-trust-store-password** Env: **KC_HTTPS_TRUST_STORE_PASSWORD**	

	Value
https-trust-store-type The type of the trust store file. If not given, the type is automatically detected based on the file extension. 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**	
https-management-client-auth ▮ Configures the management interface to require/request client authentication. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-client-auth** Env: **KC_HTTPS_MANAGEMENT_CLIENT_AUTH**	**none** (default), **request**, **required**

CHAPTER 14. ENABLING AND DISABLING FEATURES

Configure Red Hat build of Keycloak to use optional 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.

14.1. 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"
```

Enabled feature may be versioned, or unversioned. If you use a versioned feature name, e.g. feature:v1, that exact feature version will be enabled as long as it still exists in the runtime. If you instead use an unversioned name, e.g. just feature, the selection of the particular supported feature version may change from release to release according to the following precedence:

1. The highest default supported version
2. The highest non-default supported version
3. The highest deprecated version
4. The highest preview version
5. The highest experimental version

14.2. 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"
```

It is not allowed to have a feature in both the **features-disabled** list and the **features** list.

When a feature is disabled all versions of that feature are disabled.

14.3. 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

account-v3
: Account Console version 3

admin-api
: Admin API

admin-fine-grained-authz-v2
: Fine-Grained Admin Permissions version 2

admin-v2
: New Admin Console

authorization
: Authorization Service

ciba
: OpenID Connect Client Initiated Backchannel Authentication (CIBA)

client-policies
: Client configuration policies

device-flow
: OAuth 2.0 Device Authorization Grant

hostname-v2
: Hostname Options V2

impersonation
: Ability for admins to impersonate users

kerberos
: Kerberos

login-v2
: New Login Theme

opentelemetry
: OpenTelemetry Tracing

organization
: Organization support within realms

par
: OAuth 2.0 Pushed Authorization Requests (PAR)

persistent-user-sessions
: Persistent online user sessions across restarts and upgrades

rolling-updates-v1
: Rolling Updates

step-up-authentication
: Step-up Authentication

token-exchange-standard-v2
: Standard Token Exchange version 2

user-event-metrics
: Collect metrics based on user events

web-authn
: W3C Web Authentication (WebAuthn)

14.3.1. 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

multi-site
: Multi-site support

14.4. 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.

admin-fine-grained-authz
: Fine-Grained Admin Permissions

client-secret-rotation
: Client Secret Rotation

dpop
: OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer

passkeys
: Passkeys

recovery-codes
: Recovery codes

scripts
: Write custom authenticators using JavaScript

token-exchange
: Token Exchange Service

update-email
: Update Email Action

14.5. DEPRECATED FEATURES

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

login-v1

Legacy Login Theme

14.6. RELEVANT OPTIONS

	Value

	Value
features ▮ Enables a set of one or more features. CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account[:v3]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1,v2]**, **admin[:v2]**, **authorization[:v1]**, **cache-embedded-remote-store[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **clusterless[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v2]**, **impersonation[:v1]**, **ipa-tuura-federation[:v1]**, **kerberos[:v1]**, **login[:v2,v1]**, **multi-site[:v1]**, **oid4vc-vci[:v1]**, **opentelemetry[:v1]**, **organization[:v1]**, **par[:v1]**, **passkeys[:v1]**, **persistent-user-sessions[:v1]**, **preview**, **quick-theme[:v1]**, **recovery-codes[:v1]**, **rolling-updates[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange-standard[:v2]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **user-event-metrics[:v1]**, **web-authn[:v1]**

	Value
features-disabled ▮ Disables a set of one or more features. CLI: **--features-disabled** Env: **KC_FEATURES_DISABLED**	**account**, **account-api**, **admin**, **admin-api**, **admin-fine-grained-authz**, **authorization**, **cache-embedded-remote-store**, **ciba**, **client-policies**, **client-secret-rotation**, **client-types**, **clusterless**, **declarative-ui**, **device-flow**, **docker**, **dpop**, **dynamic-scopes**, **fips**, **impersonation**, **ipa-tuura-federation**, **kerberos**, **login**, **multi-site**, **oid4vc-vci**, **opentelemetry**, **organization**, **par**, **passkeys**, **persistent-user-sessions**, **preview**, **quick-theme**, **recovery-codes**, **rolling-updates**, **scripts**, **step-up-authentication**, **token-exchange**, **token-exchange-standard**, **transient-users**, **update-email**, **user-event-metrics**, **web-authn**

CHAPTER 15. CONFIGURING PROVIDERS

Configure providers for Red Hat build of Keycloak.

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.

15.1. 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
```

15.2. SETTING A PROVIDER CONFIGURATION OPTION

Provider configuration options are provided when starting the server. See all support configuration sources and formats for options in Configuring Red Hat build of Keycloak . For example via a command line option:

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
```

15.3. CONFIGURING A SINGLE PROVIDER FOR AN SPI

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 specific provider is the primary implementation that is going to be active and used at runtime.

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

Marking the **mycustomprovider** provider as the single provider for the **email-template** SPI

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

15.4. CONFIGURING A DEFAULT PROVIDER FOR AN SPI

Depending on the SPI, multiple provider implementations can co-exist and one is used by default. For these SPIs, a specific provider is the default implementation that is going to selected unless a specific provider is requested.

The following logic is used to determine the default provider:

1. The explicitly configured default provider
2. The provider with the highest order (providers with order ⇐ 0 are ignored)
3. The provider with the id set to **default**

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

Marking the **mycustomhash** provider as the default provider for the **password-hashing** SPI

```
bin/kc.[sh|bat] build --spi-password-hashing-provider-default=mycustomprovider
```

15.5. 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**.

15.6. 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.

15.7. 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.

15.8. REFERENCES

- Configuring Red Hat build of Keycloak
- Server Developer Documentation

CHAPTER 16. CONFIGURING LOGGING

Configure logging for Red Hat build of Keycloak.

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

- **console**
- **file**
- **syslog**

16.1. 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**. It is also possible to tailor log levels for each particular log handler.

This chapter describes how to configure logging.

16.1.1. 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).

16.1.2. 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**.

16.1.3. 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.

16.1.3.1. Configuring levels as individual options

When configuring category-specific log levels, you can also set the log levels as individual **log-level-<category>** options instead of using the **log-level** option for that. This is useful when you want to set the log levels for selected categories without overwriting the previously set **log-level** option.

Example

If you start the server as:

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

you can then set an environmental variable **KC_LOG_LEVEL_ORG_KEYCLOAK=trace** to change the log level for the **org.keycloak** category.

The **log-level-<category>** options take precedence over **log-level**. This allows you to override what was set in the **log-level** option. For instance if you set **KC_LOG_LEVEL_ORG_HIBERNATE=trace** for the CLI example above, the **org.hibernate** category will use the **trace** level instead of **debug**.

Bear in mind that when using the environmental variables, the category name must be in uppercase and the dots must be replaced with underscores. When using other config sources, the category name must be specified "as is", for example:

```
bin/kc.[sh|bat] start --log-level="INFO,org.hibernate:debug" --log-level-org.keycloak=trace
```

16.2. ENABLING LOG HANDLERS

To enable log handlers, enter the following command:

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

The available handlers are:

- **console**
- **file**
- **syslog**

The more specific handler configuration mentioned below will only take effect when the handler is added to this comma-separated list.

16.2.1. Specify log level for each handler

The **log-level** property specifies the global root log level and levels for selected categories. However, a more fine-grained approach for log levels is necessary to comply with the modern application requirements.

To set log levels for particular handlers, properties in format **log-<handler>-level** (where **<handler>** is available log handler) were introduced.

It means properties for log level settings look like this:

- **log-console-level** - Console log handler
- **log-file-level** - File log handler
- **log-syslog-level** - Syslog log handler

NOTE

The **log-<handler>-level** properties are available only when the particular log handlers are enabled. More information in log handlers settings below.

Only log levels specified in Section 16.1.1, "Log levels" section are accepted, and **must be in lowercase**. There is no support for specifying particular categories for log handlers yet.

16.2.1.1. General principle

It is necessary to understand that setting the log levels for each particular handler **does not override the root level** specified in the **log-level** property. Log handlers respect the root log level, which represents the maximal verbosity for the whole logging system. It means individual log handlers can be configured to be less verbose than the root logger, but not more.

Specifically, when an arbitrary log level is defined for the handler, it does not mean the log records with the log level will be present in the output. In that case, the root **log-level** must also be assessed. Log handler levels provide the **restriction for the root log level**, and the default log level for log handlers is **all** - without any restriction.

16.2.1.2. Examples

Example: debug for file handler, but info for console handler:

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

The root log level is set to **debug**, so every log handler inherits the value - so does the file log handler. To hide **debug** records in the console, we need to set the minimal (least severe) level to **info** for the console handler.

Example: warn for all handlers, but debug for file handler:

```
bin/kc.[sh|bat] start --log=console,file,syslog --log-level=debug --log-console-level=warn --log-syslog-
level=warn
```

The root level must be set to the most verbose required level (**debug** in this case), and other log handlers must be amended accordingly.

Example: info for all handlers, but debug+org.keycloak.events:trace for Syslog handler:

```
bin/kc.[sh|bat] start --log=console,file,syslog --log-level=debug,org.keycloak.events:trace, --log-
syslog-level=trace --log-console-level=info --log-file-level=info
```

In order to see the **org.keycloak.events:trace**, the **trace** level must be set for the Syslog handler.

16.2.2. Use different JSON format for log handlers

Every log handler provides the ability to have structured log output in JSON format. It can be enabled by properties in the format **log-<handler>-output=json** (where **<handler>** is a log handler).

If you need a different format of the produced JSON, you can leverage the following JSON output formats:

- **default** (default)
- **ecs**

The **ecs** value refers to the ECS (Elastic Common Schema).

ECS is an open-source, community-driven specification that defines a common set of fields to be used with Elastic solutions. The ECS specification is being converged with OpenTelemetry Semantic Conventions with the goal of creating a single standard maintained by OpenTelemetry.

In order to change the JSON output format, properties in the format **log-<handler>-json-format** (where **<handler>** is a log handler) were introduced:

- **log-console-json-format** - Console log handler
- **log-file-json-format** - File log handler
- **log-syslog-json-format** - Syslog log handler

16.2.2.1. Example

If you want to have JSON logs in **ECS** (Elastic Common Schema) format for the console log handler, you can enter the following command:

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

Example Log Message

```
{"@timestamp":"2025-02-
03T14:53:22.539484211+01:00","event.sequence":9608,"log.logger":"io.quarkus","log.level":"INFO","
message":"Keycloak 999.0.0-SNAPSHOT on JVM (powered by Quarkus 3.17.8) started in 4.615s.
Listening on: http://0.0.0.0:8080","process.thread.name":"main","process.thread.id":1,"mdc":
{},"ndc":"","host.hostname":"host-name","process.name":"/usr/lib/jvm/jdk-
21.0.3+9/bin/java","process.pid":77561,"data_stream.type":"logs","ecs.version":"1.12.2","service.envir
onment":"prod","service.name":"Keycloak","service.version":"999.0.0-SNAPSHOT"}
```

16.3. CONSOLE LOG HANDLER

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

16.3.1. 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.

Symbol	Summary	Description
%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.

16.3.2. 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]**.

16.3.3. 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":"2025-02-
03T14:52:20.290353085+01:00","sequence":9605,"loggerClassName":"org.jboss.logging.Logger","log
gerName":"io.quarkus","level":"INFO","message":"Keycloak 999.0.0-SNAPSHOT on JVM (powered by
Quarkus 3.17.8) started in 4.440s. Listening on:
http://0.0.0.0:8080","threadName":"main","threadId":1,"mdc":{},"ndc":"","hostName":"host-
name","processName":"/usr/lib/jvm/jdk-21.0.3+9/bin/java","processId":76944}
```

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

```
2025-02-03 14:53:56,653 INFO  [io.quarkus] (main) Keycloak 999.0.0-SNAPSHOT on JVM (powered
by Quarkus 3.17.8) started in 4.795s. Listening on: http://0.0.0.0:8080
```

16.3.4. 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>
```

16.3.5. Configuring the console log level

Log level for console log handler can be specified by **--log-console-level** property as follows:

```
bin/kc.[sh|bat] start --log-console-level=warn
```

For more information, see the section Section 16.2.1, “Specify log level for each handler” above.

16.4. FILE LOGGING

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

16.4.1. 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 Red Hat build of Keycloak installation.

16.4.2. 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>
   ```

16.4.3. 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 16.3.1, “Configuring the console log format” for more information and a table of the available pattern configuration.

16.4.4. Configuring the file log level

Log level for file log handler can be specified by **--log-file-level** property as follows:

```
bin/kc.[sh|bat] start --log-file-level=warn
```

For more information, see the section Section 16.2.1, “Specify log level for each handler” above.

16.5. CENTRALIZED LOGGING USING SYSLOG

Red Hat build of Keycloak provides the ability to send logs to a remote Syslog server. It utilizes the protocol defined in RFC 5424.

16.5.1. Enable the Syslog handler

To enable logging using Syslog, add it to the list of activated log handlers as follows:

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

16.5.2. Configuring the Syslog Application Name

To set a different application name, add the **--log-syslog-app-name** option as follows:

```
bin/kc.[sh|bat] start --log="console,syslog" --log-syslog-app-name=kc-p-itadmins
```

If not set, the application name defaults to **keycloak**.

16.5.3. Configuring the Syslog endpoint

To configure the endpoint(*host:port*) of your centralized logging system, enter the following command and substitute the values with your specific values:

```
bin/kc.[sh|bat] start --log="console,syslog" --log-syslog-endpoint=myhost:12345
```

When the Syslog handler is enabled, the host is using **localhost** as host value. The Default port is **514**.

16.5.4. Configuring the Syslog log level

Log level for Syslog log handler can be specified by **--log-syslog-level** property as follows:

```
bin/kc.[sh|bat] start --log-syslog-level=warn
```

For more information, see the section Section 16.2.1, “Specify log level for each handler” above.

16.5.5. Configuring the Syslog protocol

Syslog uses TCP as the default protocol for communication. To use UDP instead of TCP, add the **--log-syslog-protocol** option as follows:

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

The available protocols are: **tpc**, **udp**, and **ssl-tcp**.

16.5.6. Configuring the Syslog log 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-syslog-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-syslog-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]**.

16.5.7. Configuring the Syslog type

Syslog uses different message formats based on particular RFC specifications. To change the Syslog type with a different message format, use the **--log-syslog-type** option as follows:

```
bin/kc.[sh|bat] start --log-syslog-type=rfc3164
```

Possible values for the **--log-syslog-type** option are:

- **rfc5424** (default)
- **rfc3164**

The preferred Syslog type is RFC 5424, which obsoletes RFC 3164, known as BSD Syslog protocol.

16.5.8. Configuring the Syslog maximum message length

To set the maximum length of the message allowed to be sent (in bytes), use the **--log-syslog-max-length** option as follows:

```
bin/kc.[sh|bat] start --log-syslog-max-length=1536
```

The length can be specified in memory size format with the appropriate suffix, like **1k** or **1K**. The length includes the header and the message.

If the length is not explicitly set, the default values are set based on the **--log-syslog-type** option as follows:

- **2048B** - for RFC 5424
- **1024B** - for RFC 3164

16.5.9. Configuring the Syslog structured output

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

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

Example Log Message

```
2024-04-05T12:32:20.616+02:00 host keycloak 2788276 io.quarkus - {"timestamp":"2024-04-05T12:32:20.616208533+02:00","sequence":9948,"loggerClassName":"org.jboss.logging.Logger","loggerName":"io.quarkus","level":"INFO","message":"Profile prod activated.
","threadName":"main","threadId":1,"mdc":
{},"ndc":"","hostName":"host","processName":"QuarkusEntryPoint","processId":2788276}
```

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

To use unstructured logging, enter the following command:

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

Example Log Message

```
2024-04-05T12:31:38.473+02:00 host keycloak 2787568 io.quarkus - 2024-04-05 12:31:38,473 INFO
[io.quarkus] (main) Profile prod activated.
```

As you can see, the timestamp is present twice, so you can amend it correspondingly via the **--log-syslog-format** property.

16.6. RELEVANT OPTIONS

	Value
log Enable one or more log handlers in a comma-separated list. CLI: **--log** Env: **KC_LOG**	**console**, **file**, **syslog**
log-level The log level of the root category or a comma-separated list of individual categories and their levels. For the root category, you don’t need to specify a category. CLI: **--log-level** Env: **KC_LOG_LEVEL**	**[info]** (default)

16.6.1. Console

	Value
log-console-color Enable or disable colors when logging to console. CLI: **--log-console-color** Env: **KC_LOG_CONSOLE_COLOR** Available only when Console log handler is activated	**true**, **false** (default)
log-console-format The format of unstructured console log entries. If the format has spaces in it, escape the value using "<format>". CLI: **--log-console-format** Env: **KC_LOG_CONSOLE_FORMAT** Available only when Console log handler is activated	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (default)
log-console-include-trace Include tracing information in the console log. If the **log-console-format** option is specified, this option has no effect. CLI: **--log-console-include-trace** Env: **KC_LOG_CONSOLE_INCLUDE_TRACE** Available only when Console log handler and Tracing is activated	**true** (default), **false**
log-console-json-format Set the format of the produced JSON. CLI: **--log-console-json-format** Env: **KC_LOG_CONSOLE_JSON_FORMAT** Available only when Console log handler is activated and output is set to 'json'	**default** (default), **ecs**
log-console-level Set the log level for the console handler. It specifies the most verbose log level for logs shown in the output. It respects levels specified in the **log-level** option, which represents the maximal verbosity for the whole logging system. For more information, check the Logging guide. CLI: **--log-console-level** Env: **KC_LOG_CONSOLE_LEVEL** Available only when Console log handler is activated	**off**, **fatal**, **error**, **warn**, **info**, **debug**, **trace**, **all** (default)

	Value
log-console-output Set the log output to JSON or default (plain) unstructured logging. CLI: **--log-console-output** Env: **KC_LOG_CONSOLE_OUTPUT** Available only when Console log handler is activated	**default** (default), **json**

16.6.2. File

	Value
log-file Set the log file path and filename. CLI: **--log-file** Env: **KC_LOG_FILE** Available only when File log handler is activated	**data/log/keycloak.log** (default)
log-file-format Set a format specific to file log entries. CLI: **--log-file-format** Env: **KC_LOG_FILE_FORMAT** Available only when File log handler is activated	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (default)
log-file-include-trace Include tracing information in the file log. If the **log-file-format** option is specified, this option has no effect. CLI: **--log-file-include-trace** Env: **KC_LOG_FILE_INCLUDE_TRACE** Available only when File log handler and Tracing is activated	**true** (default), **false**
log-file-json-format Set the format of the produced JSON. CLI: **--log-file-json-format** Env: **KC_LOG_FILE_JSON_FORMAT** Available only when File log handler is activated and output is set to 'json'	**default** (default), **ecs**

	Value
log-file-level Set the log level for the file handler. It specifies the most verbose log level for logs shown in the output. It respects levels specified in the **log-level** option, which represents the maximal verbosity for the whole logging system. For more information, check the Logging guide. CLI: **--log-file-level** Env: **KC_LOG_FILE_LEVEL** Available only when File log handler is activated	**off**, **fatal**, **error**, **warn**, **info**, **debug**, **trace**, **all** (default)
log-file-output Set the log output to JSON or default (plain) unstructured logging. CLI: **--log-file-output** Env: **KC_LOG_FILE_OUTPUT** Available only when File log handler is activated	**default** (default), **json**

16.6.3. Syslog

	Value
log-syslog-app-name Set the app name used when formatting the message in RFC5424 format. CLI: **--log-syslog-app-name** Env: **KC_LOG_SYSLOG_APP_NAME** Available only when Syslog is activated	**keycloak** (default)
log-syslog-endpoint Set the IP address and port of the Syslog server. CLI: **--log-syslog-endpoint** Env: **KC_LOG_SYSLOG_ENDPOINT** Available only when Syslog is activated	**localhost:514** (default)
log-syslog-format Set a format specific to Syslog entries. CLI: **--log-syslog-format** Env: **KC_LOG_SYSLOG_FORMAT** Available only when Syslog is activated	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (default)

	Value
log-syslog-include-trace Include tracing information in the Syslog. If the **log-syslog-format** option is specified, this option has no effect. CLI: **--log-syslog-include-trace** Env: **KC_LOG_SYSLOG_INCLUDE_TRACE** Available only when Syslog handler and Tracing is activated	**true** (default), **false**
log-syslog-json-format Set the format of the produced JSON. CLI: **--log-syslog-json-format** Env: **KC_LOG_SYSLOG_JSON_FORMAT** Available only when Syslog is activated and output is set to 'json'	**default** (default), **ecs**
log-syslog-level Set the log level for the Syslog handler. It specifies the most verbose log level for logs shown in the output. It respects levels specified in the **log-level** option, which represents the maximal verbosity for the whole logging system. For more information, check the Logging guide. CLI: **--log-syslog-level** Env: **KC_LOG_SYSLOG_LEVEL** Available only when Syslog is activated	**off**, **fatal**, **error**, **warn**, **info**, **debug**, **trace**, **all** (default)
log-syslog-max-length Set the maximum length, in bytes, of the message allowed to be sent. The length includes the header and the message. If not set, the default value is 2048 when **log-syslog-type** is rfc5424 (default) and 1024 when **log-syslog-type** is rfc3164. CLI: **--log-syslog-max-length** Env: **KC_LOG_SYSLOG_MAX_LENGTH** Available only when Syslog is activated	
log-syslog-output Set the Syslog output to JSON or default (plain) unstructured logging. CLI: **--log-syslog-output** Env: **KC_LOG_SYSLOG_OUTPUT** Available only when Syslog is activated	**default** (default), **json**

	Value
log-syslog-protocol Set the protocol used to connect to the Syslog server. CLI: **--log-syslog-protocol** Env: **KC_LOG_SYSLOG_PROTOCOL** Available only when Syslog is activated	**tcp** (default), **udp**, **ssl-tcp**
log-syslog-type Set the Syslog type used to format the sent message. CLI: **--log-syslog-type** Env: **KC_LOG_SYSLOG_TYPE** Available only when Syslog is activated	**rfc5424** (default), **rfc3164**

CHAPTER 17. FIPS 140-2 SUPPORT

Configure Red Hat build of Keycloak server for FIPS compliance.

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 running 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 subsequently enabling it as follows:

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

17.1. BOUNCYCASTLE LIBRARY

Red Hat build of Keycloak internally uses the BouncyCastle library for many cryptography utilities. Please note that the default version of the BouncyCastle library that shipped with Red Hat build of Keycloak is not FIPS compliant; however, BouncyCastle also provides a FIPS validated version of its library. The FIPS validated BouncyCastle library is not shipped with Red Hat build of Keycloak as 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.

17.1.1. 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 Red Hat build of Keycloak dependencies. The supported BCFIPS bits needed are:

- bc-fips version 2.0.0.
- bctls-fips version 2.0.19.
- bcpkix-fips version 2.0.7.
- bcutil-fips version 2.0.3.

17.2. GENERATING KEYSTORE

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

17.2.1. PKCS12 keystore

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

PKCS12 keystore can be generated with OpenJDK 21 Java on RHEL 9 in the standard way. For instance, the following command can be used to generate the keystore:

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

The **pkcs12** keystores in FIPS mode **do not** manage secret (symmetric) keys. This limitation is imposed by the **BCFIPS** provider which does not allow this type of keys inside the **pkcs12** keystore type.

When the 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 the PKCS12 keystore, you can store PBE (password-based encryption) keys simply by using 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.

17.2.2. BCFKS keystore

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

You can start by creating 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 \
  -providername 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.

17.3. 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 a BCFKS keystore as described above, it is also required to use the command **--https-key-store-type=bcfks**. A similar command 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.

17.4. 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 a different keystore type it is required to use the option **--https-key-store-type** with appropriate type. A similar command might be needed for the truststore as well if you want to use it.

When starting the server, you can include TRACE level in the startup command. For example:

```
--log-level=INFO,org.keycloak.common.crypto.CryptoIntegration:TRACE
```

By using TRACE level, you can check that the startup log contains **KC** provider with the note about **Approved Mode** such as the following:

```
KC(BCFIPS version 2.0 Approved Mode, FIPS-JVM: enabled) version 1.0 - class
org.keycloak.crypto.fips.KeycloakFipsSecurityProvider,
```

17.4.1. Cryptography restrictions in strict mode

- As mentioned in the previous section, strict mode may not work with **pkcs12** keystore. It is required 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-sha512** of SPI **password-hashing** to 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-sha512-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. If you want to omit the option for max-padding-length, you can set the password policy to your realms to have passwords at least 14 characters long.

NOTE

When you are migrating from Red Hat build of Keycloak older than 24, or if you explicitly set the password policy to override the default hashing algorithm, it is possible that some of your users use an older algorithm like **pbkdf2-sha256**. In this case, consider adding the **--spi-password-hashing-pbkdf2-sha256-max-padding-length=14** option to ensure that users having their passwords hashed with the older **pbkdf2-sha256** can log in because their passwords may be shorter 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 the Red Hat build of Keycloak server, which always fulfils this requirement.

- The bc-fips version 1.0.2.4 deals with the end of the transition period for PKCS 1.5 RSA encryption. Therefore JSON Web Encryption (JWE) with algorithm **RSA1_5** is not allowed in strict mode by default (BC provides the system property **-**

Dorg.bouncycastle.rsa.allow_pkcs15_enc=true as backward compatibility option for the moment). **RSA-OAEP** and **RSA-OAEP-256** are still available as before.

17.5. 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 21, the **XMLDSig** security provider may be already enabled in the **java.security** by default and the same applies with latest OpenJDK 17. But with older OpenJDK 17, it may not be enabled by default, which means that SAML 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 in case that it is not already available in your FIPS security providers:

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

Adding this security provider should work well. In fact, it is FIPS compliant and is already added by default in the OpenJDK 21 and newer versions of OpenJDK 17. 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 you 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.

The algorithm **EdDSA** cannot be used in FIPS mode. Although the current **BCFIPS** provider supports **Ed25519** and **Ed448** curves, the resulting keys do not implement the standard JDK interfaces to manage them (**EdECKey**, **EdECPublicKey**, **EdECPrivateKey**,…), and Red Hat build of Keycloak cannot use them for signatures.

17.6. 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/
cp $KEYCLOAK_HOME/providers/bcutil-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"
```

17.7. RED HAT BUILD OF KEYCLOAK SERVER IN FIPS MODE IN CONTAINERS

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 automatically be 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 (Not needed with OpenJDK 21 or newer OpenJDK 17)

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

Containerfile:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:26.2 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-rhel9:26.2
COPY --from=builder /opt/keycloak/ /opt/keycloak/

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

Then build FIPS as 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.

17.8. MIGRATION FROM NON-FIPS ENVIRONMENT

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

- Starting with Red Hat build of Keycloak 25, the default algorithm for password hashing is **argon2**. However, this algorithm is not supported for FIPS 140-2. This means that if your users hashed their password with **argon2**, they will not be able to login after switch to the FIPS environment. If you plan to migrate to the FIPS environment, consider setting the Password policy for your realm from the beginning (before any users are created) and override the default algorithm for example to **pbkdf2-sha512**, which is FIPS compliant. This strategy helps to make the migration to the FIPS environment to be smooth. Otherwise, if your users are already on **argon2** passwords, simply ask users to reset the password after migrating to the FIPS environment. For instance, ask users to use "Forget password" or send the email for reset-password to all users.

- 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 **Forgot password** link) during the first authentication in the new environment.

17.9. 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 running 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 amount to 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 21 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 Red Hat build of Keycloak packages as described in the Keycloak startup command earlier.

CHAPTER 18. CONFIGURING THE MANAGEMENT INTERFACE

Configure Red Hat build of Keycloak's management interface for endpoints such as metrics and health checks.

The management interface allows accessing management endpoints via a different HTTP server than the primary one. It provides the possibility to hide endpoints like **/metrics** or **/health** from the outside world and, therefore, hardens the security. The most significant advantage might be seen in Kubernetes environments as the specific management port might not be exposed.

18.1. MANAGEMENT INTERFACE CONFIGURATION

The management interface is turned on when something is exposed on it. Management endpoints such as **/metrics** and **/health** are exposed on the default management port **9000** when metrics and health are enabled. The management interface provides a set of options and is fully configurable.

NOTE

If management interface properties are not explicitly set, their values are automatically inherited from the default HTTP server.

18.1.1. Port

In order to change the port for the management interface, you can use the Red Hat build of Keycloak option **http-management-port**.

18.1.2. Relative path

You can change the relative path of the management interface, as the prefix path for the management endpoints can be different. You can achieve it via the Red Hat build of Keycloak option **http-management-relative-path**.

For instance, if you set the CLI option **--http-management-relative-path=/management**, the metrics, and health endpoints will be accessed on the **/management/metrics** and **/management/health** paths.

User is automatically **redirected** to the path where Red Hat build of Keycloak is hosted when the relative path is specified. It means when the relative path is set to **/management**, and the user access **localhost:9000/**, the page is redirected to **localhost:9000/management**.

NOTE

If you do not explicitly set the value for it, the value from the **http-relative-path** property is used. For instance, if you set the CLI option **--http-relative-path=/auth**, these endpoints are accessible on the **/auth/metrics** and **/auth/health** paths.

18.1.3. TLS support

When the TLS is set for the default Red Hat build of Keycloak server, the management interface will be accessible through HTTPS as well. The management interface can run only either on HTTP or HTTPS, not both as for the main server.

Specific Red Hat build of Keycloak management interface options with the prefix **https-management-*** were provided for setting different TLS parameters for the management HTTP server. Their function is similar to their counterparts for the main HTTP server, for details see Configuring TLS. When these options are not explicitly set, the TLS parameters are inherited from the default HTTP server.

18.1.4. Disable Management interface

The management interface is automatically turned off when nothing is exposed on it. Currently, only health checks and metrics are exposed on the management interface regardless. If you want to disable exposing them on the management interface, set the Red Hat build of Keycloak property **legacy-observability-interface** to **true**.

> **WARNING**
>
> Exposing health and metrics endpoints on the default server is not recommended for security reasons, and you should always use the management interface. Beware, the **legacy-observability-interface** option is deprecated and will be removed in future releases. It only allows you to give more time for the migration.

18.2. RELEVANT OPTIONS

	Value
http-management-port Port of the management interface. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--http-management-port** Env: **KC_HTTP_MANAGEMENT_PORT**	**9000** (default)
http-management-relative-path ▮ Set the path relative to / for serving resources from management interface. The path must start with a /. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--http-management-relative-path** Env: **KC_HTTP_MANAGEMENT_RELATIVE_PATH**	/ (default)

	Value
https-management-certificate-file The file path to a server certificate or certificate chain in PEM format for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificate-file** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATE_FILE**	
https-management-certificate-key-file The file path to a private key in PEM format for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificate-key-file** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATE_KEY_FILE**	
https-management-certificates-reload-period Interval on which to reload key store, trust store, and certificate files referenced by https-management-* options for the management server. May be a java.time.Duration value, an integer number of seconds, or an integer followed by one of [ms, h, m, s, d]. Must be greater than 30 seconds. Use -1 to disable. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificates-reload-period** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATES_RELOAD_PERIOD**	**1h** (default)
https-management-client-auth ▮ Configures the management interface to require/request client authentication. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-client-auth** Env: **KC_HTTPS_MANAGEMENT_CLIENT_AUTH**	**none** (default), **request**, **required**

	Value
https-management-key-store-file The key store which holds the certificate information instead of specifying separate files for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-key-store-file** Env: **KC_HTTPS_MANAGEMENT_KEY_STORE_FILE**	
https-management-key-store-password The password of the key store file for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-key-store-password** Env: **KC_HTTPS_MANAGEMENT_KEY_STORE_PASSWORD**	**password** (default)
legacy-observability-interface ▮ If metrics/health endpoints should be exposed on the main HTTP server (not recommended). If set to true, the management interface is disabled. CLI: **--legacy-observability-interface** Env: **KC_LEGACY_OBSERVABILITY_INTERFACE** DEPRECATED.	**true**, **false** (default)

CHAPTER 19. IMPORTING AND EXPORTING REALMS

Import and export realms as JSON files.

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.

All Red Hat build of Keycloak nodes need to be stopped prior to using **kc.[sh|bat] import | export** commands. This ensures that the resulting operations will have no consistency issues with concurrent requests. It also ensures that running an import or export command from the same machine as a server instance will not result in port or other conflicts.

19.1. 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 Red Hat build of 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.

NOTE

if you do not use **--optimized** keep in mind that an **import** or **export** command will implicitly create or update an optimized image for you - if you are running the command from the same machine as a server instance, this may impact the next start of your server.

19.2. 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.

19.2.1. 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
```

19.3. 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.

19.4. 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
```

19.5. IMPORT FILE NAMING CONVENTIONS

When you export a realm specific file name conventions are used, which must also be used for importing from a directory or import at startup. The realm file to be imported must be named <realm name>-realm.json. Regular and federated user files associated with a realm must be named <realm-name>-

users-<file number>.json and <realm-name>-federated-users-<file number>.json. Failure to use this convention will result in errors or user files not being imported.

19.6. 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.

19.7. 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>
```

19.8. USING ENVIRONMENT VARIABLES WITHIN THE REALM CONFIGURATION FILES

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.

NOTE

there are currently no restrictions on what environment variables may be referenced. When environment variables are used to convey sensitive information, take care to ensure placeholders references do not inappropriately expose sensitive environment variable values.

19.9. 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 lose state between server restarts.

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

19.10. IMPORTING AND EXPORTING BY USING THE ADMIN CONSOLE

You can also import and export a realm using the Admin Console. This functionality is different from the other CLI options described in previous sections because the Admin Console offers only the capability to *partially* export a realm. In this case, the current realm settings, along with some resources like clients, roles, and groups, can be exported. The users for that realm *cannot* be exported using this method.

NOTE

When using the Admin Console export, the realm and the selected resources are always exported to a file named **realm-export.json**. Also, all sensitive values like passwords and client secrets will be masked with ***** symbols.

To export a realm using the Admin Console, perform these steps:

1. Select a realm.
2. Click **Realm settings** in the menu.
3. Point to the **Action** menu in the top right corner of the realm settings screen, and select **Partial export**.
   A list of resources appears along with the realm configuration.
4. Select the resources you want to export.
5. Click **Export**.

NOTE

Realms exported from the Admin Console are not suitable for backups or data transfer between servers. Only CLI exports are suitable for backups or data transfer between servers.

WARNING

If the realm contains many groups, roles, and clients, the operation may cause the server to be unresponsive to user requests for a while. Use this feature with caution, especially on a production system.

In a similar way, you can import a previously exported realm. Perform these steps:

1. Click **Realm settings** in the menu.
2. Point to the **Action** menu in the top right corner of the realm settings screen, and select **Partial import**.
   A prompt appears where you can select the file you want to import. Based on this file, you see the resources you can import along with the realm settings.
3. Click **Import**.

You can also control what Red Hat build of Keycloak should do if the imported resource already exists. These options exist:

Fail import

Abort the import.

Skip

Skip the duplicate resources without aborting the process

Overwrite

Replace the existing resources with the ones being imported.

NOTE

The Admin Console partial import can also import files created by the CLI **export** command. In other words, full exports created by the CLI can be imported by using the Admin Console. If the file contains users, those users will also be available for importing into the current realm.

CHAPTER 20. USING A VAULT

Configure and use a vault in Red Hat build of Keycloak.

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.

20.1. 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

20.2. 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
```

20.3. CONFIGURING THE FILE-BASED VAULT

20.3.1. 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
```

20.3.2. 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>}
```

20.4. 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}**.

20.5. USING UNDERSCORES IN THE SECRET NAMES

To process the secret correctly, you double all underscores in the <secretname>. When **REALM_UNDERSCORE_KEY** key resolver is used, underscores in <realmname> are also doubled and <secretname> and <realmname> is 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*.

To learn more about key resolvers, see Key resolvers section in the Server Administration guide .

20.6. 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.

20.7. RELEVANT OPTIONS

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

CHAPTER 21. ALL CONFIGURATION

Review build options and configuration for Red Hat build of Keycloak.

21.1. CACHE

	Value
cache Defines the cache mechanism for high-availability. 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. CLI: **--cache** Env: **KC_CACHE**	**ispn** (default), **local**
cache-config-file Defines the file from which cache configuration should be loaded from. The configuration file is relative to the **conf/** directory. CLI: **--cache-config-file** Env: **KC_CACHE_CONFIG_FILE**	
cache-embedded-authorization-max-count The maximum number of entries that can be stored in-memory by the authorization cache. CLI: **--cache-embedded-authorization-max-count** Env: **KC_CACHE_EMBEDDED_AUTHORIZATION_MAX_COUNT**	
cache-embedded-client-sessions-max-count The maximum number of entries that can be stored in-memory by the clientSessions cache. CLI: **--cache-embedded-client-sessions-max-count** Env: **KC_CACHE_EMBEDDED_CLIENT_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	
cache-embedded-crl-max-count The maximum number of entries that can be stored in-memory by the crl cache. CLI: **--cache-embedded-crl-max-count** Env: **KC_CACHE_EMBEDDED_CRL_MAX_COUNT**	

	Value
cache-embedded-keys-max-count The maximum number of entries that can be stored in-memory by the keys cache. CLI: **--cache-embedded-keys-max-count** Env: **KC_CACHE_EMBEDDED_KEYS_MAX_COUNT**	
cache-embedded-mtls-enabled Encrypts the network communication between Keycloak servers. If no additional parameters about a keystore and truststore are provided, ephemeral key pairs and certificates are created and rotated automatically, which is recommended for standard setups. CLI: **--cache-embedded-mtls-enabled** Env: **KC_CACHE_EMBEDDED_MTLS_ENABLED** Available only when a TCP based cache-stack is used	**true** (default), **false**
cache-embedded-mtls-key-store-file The Keystore file path. The Keystore must contain the certificate to use by the TLS protocol. By default, it looks up **cache-mtls-keystore.p12** under conf/ directory. CLI: **--cache-embedded-mtls-key-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_FILE** Available only when property 'cache-embedded-mtls-enabled' is enabled	
cache-embedded-mtls-key-store-password The password to access the Keystore. CLI: **--cache-embedded-mtls-key-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_PASSWORD** Available only when property 'cache-embedded-mtls-enabled' is enabled	
cache-embedded-mtls-rotation-interval-days Rotation period in days of automatic JGroups MTLS certificates. CLI: **--cache-embedded-mtls-rotation-interval-days** Env: **KC_CACHE_EMBEDDED_MTLS_ROTATION_INTERVAL_DAYS** Available only when property 'cache-embedded-mtls-enabled' is enabled	**30** (default)

	Value
cache-embedded-mtls-trust-store-file The Truststore file path. It should contain the trusted certificates or the Certificate Authority that signed the certificates. By default, it lookup **cache-mtls-truststore.p12** under conf/ directory. CLI: **--cache-embedded-mtls-trust-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_FILE** Available only when property 'cache-embedded-mtls-enabled' is enabled	
cache-embedded-mtls-trust-store-password The password to access the Truststore. CLI: **--cache-embedded-mtls-trust-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_PASSWORD** Available only when property 'cache-embedded-mtls-enabled' is enabled	
cache-embedded-offline-client-sessions-max-count The maximum number of entries that can be stored in-memory by the offlineClientSessions cache. CLI: **--cache-embedded-offline-client-sessions-max-count** Env: **KC_CACHE_EMBEDDED_OFFLINE_CLIENT_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	
cache-embedded-offline-sessions-max-count The maximum number of entries that can be stored in-memory by the offlineSessions cache. CLI: **--cache-embedded-offline-sessions-max-count** Env: **KC_CACHE_EMBEDDED_OFFLINE_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	
cache-embedded-realms-max-count The maximum number of entries that can be stored in-memory by the realms cache. CLI: **--cache-embedded-realms-max-count** Env: **KC_CACHE_EMBEDDED_REALMS_MAX_COUNT**	

	Value
cache-embedded-sessions-max-count The maximum number of entries that can be stored in-memory by the sessions cache. CLI: **--cache-embedded-sessions-max-count** Env: **KC_CACHE_EMBEDDED_SESSIONS_MAX_COUNT** Available only when embedded Infinispan clusters configured	
cache-embedded-users-max-count The maximum number of entries that can be stored in-memory by the users cache. CLI: **--cache-embedded-users-max-count** Env: **KC_CACHE_EMBEDDED_USERS_MAX_COUNT**	
cache-metrics-histograms-enabled Enable histograms for metrics for the embedded caches. CLI: **--cache-metrics-histograms-enabled** Env: **KC_CACHE_METRICS_HISTOGRAMS_ENABLED** Available only when metrics are enabled	**true**, **false** (default)
cache-remote-host The hostname of the external Infinispan cluster. Available only when feature **multi-site**, **clusterless** or **cache-embedded-remote-store** is set. CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	
cache-remote-password The password for the authentication to the external Infinispan cluster. It is optional if connecting to an unsecure external Infinispan cluster. If the option is specified, **cache-remote-username** is required as well. CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD** Available only when remote host is set	

	Value
cache-remote-port The port of the external Infinispan cluster. CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT** Available only when remote host is set	**11222** (default)
cache-remote-tls-enabled Enable TLS support to communicate with a secured remote Infinispan server. Recommended to be enabled in production. CLI: **--cache-remote-tls-enabled** Env: **KC_CACHE_REMOTE_TLS_ENABLED** Available only when remote host is set	**true** (default), **false**
cache-remote-username The username for the authentication to the external Infinispan cluster. It is optional if connecting to an unsecure external Infinispan cluster. If the option is specified, **cache-remote-password** is required as well. CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME** Available only when remote host is set	
cache-stack Define the default stack to use for cluster communication and node discovery. Defaults to **jdbc-ping** if not set. CLI: **--cache-stack** Env: **KC_CACHE_STACK** Available only when 'cache' type is set to 'ispn' Use 'jdbc-ping' instead by leaving it unset **Deprecated values: azure, ec2, google, tcp, udp, jdbc-ping-udp**	**jdbc-ping**, **kubernetes**, **jdbc-ping-udp** (deprecated), **tcp** (deprecated), **udp** (deprecated), **ec2** (deprecated), **azure** (deprecated), **google** (deprecated), or any

21.2. CONFIG

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

21.3. DATABASE

	Value
db ▮ The database vendor. In production mode the default value of **dev-file** is deprecated, you should explicitly specify the db instead. CLI: **--db** Env: **KC_DB**	**dev-file** (default), **dev-mem**, **mariadb**, **mssql**, **mysql**, **oracle**, **postgres**
db-driver ▮ The fully qualified class name of the JDBC driver. If not set, a default driver is set accordingly to the chosen database. CLI: **--db-driver** Env: **KC_DB_DRIVER**	
db-password The password of the database user. CLI: **--db-password** Env: **KC_DB_PASSWORD**	

	Value
db-pool-initial-size The initial size of the connection pool. CLI: **--db-pool-initial-size** Env: **KC_DB_POOL_INITIAL_SIZE**	
db-pool-max-size The maximum size of the connection pool. CLI: **--db-pool-max-size** Env: **KC_DB_POOL_MAX_SIZE**	**100** (default)
db-pool-min-size The minimal size of the connection pool. CLI: **--db-pool-min-size** Env: **KC_DB_POOL_MIN_SIZE**	
db-schema The database schema to be used. CLI: **--db-schema** Env: **KC_DB_SCHEMA**	
db-url The full database JDBC URL. 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**. CLI: **--db-url** Env: **KC_DB_URL**	
db-url-database Sets the database name of the default JDBC URL of the chosen vendor. If the **db-url** option is set, this option is ignored. CLI: **--db-url-database** Env: **KC_DB_URL_DATABASE**	

	Value
db-url-host Sets the hostname of the default JDBC URL of the chosen vendor. If the **db-url** option is set, this option is ignored. CLI: **--db-url-host** Env: **KC_DB_URL_HOST**	
db-url-port Sets the port of the default JDBC URL of the chosen vendor. If the **db-url** option is set, this option is ignored. CLI: **--db-url-port** Env: **KC_DB_URL_PORT**	
db-url-properties Sets the properties of the default JDBC URL of the chosen vendor. 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. CLI: **--db-url-properties** Env: **KC_DB_URL_PROPERTIES**	
db-username The username of the database user. CLI: **--db-username** Env: **KC_DB_USERNAME**	

21.4. TRANSACTION

	Value
transaction-xa-enabled ▮ If set to true, XA datasources will be used. CLI: **--transaction-xa-enabled** Env: **KC_TRANSACTION_XA_ENABLED**	**true**, **false** (default)

21.5. FEATURE

	Value
features ▮ Enables a set of one or more features. CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account[:v3]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1,v2]**, **admin[:v2]**, **authorization[:v1]**, **cache-embedded-remote-store[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **clusterless[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v2]**, **impersonation[:v1]**, **ipa-tuura-federation[:v1]**, **kerberos[:v1]**, **login[:v2,v1]**, **multi-site[:v1]**, **oid4vc-vci[:v1]**, **opentelemetry[:v1]**, **organization[:v1]**, **par[:v1]**, **passkeys[:v1]**, **persistent-user-sessions[:v1]**, **preview**, **quick-theme[:v1]**, **recovery-codes[:v1]**, **rolling-updates[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange-standard[:v2]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **user-event-metrics[:v1]**, **web-authn[:v1]**

	Value
features-disabled ▮ Disables a set of one or more features. CLI: **--features-disabled** Env: **KC_FEATURES_DISABLED**	**account**, **account-api**, **admin**, **admin-api**, **admin-fine-grained-authz**, **authorization**, **cache-embedded-remote-store**, **ciba**, **client-policies**, **client-secret-rotation**, **client-types**, **clusterless**, **declarative-ui**, **device-flow**, **docker**, **dpop**, **dynamic-scopes**, **fips**, **impersonation**, **ipa-tuura-federation**, **kerberos**, **login**, **multi-site**, **oid4vc-vci**, **opentelemetry**, **organization**, **par**, **passkeys**, **persistent-user-sessions**, **preview**, **quick-theme**, **recovery-codes**, **rolling-updates**, **scripts**, **step-up-authentication**, **token-exchange**, **token-exchange-standard**, **transient-users**, **update-email**, **user-event-metrics**, **web-authn**

21.6. HOSTNAME V2

	Value
hostname Address at which is the server exposed. Can be a full URL, or just a hostname. When only hostname is provided, scheme, port and context path are resolved from the request. CLI: **--hostname** Env: **KC_HOSTNAME** Available only when hostname:v2 feature is enabled	
hostname-admin Address for accessing the administration console. Use this option if you are exposing the administration console using a reverse proxy on a different address than specified in the **hostname** option. CLI: **--hostname-admin** Env: **KC_HOSTNAME_ADMIN** Available only when hostname:v2 feature is enabled	
hostname-backchannel-dynamic Enables dynamic resolving of backchannel URLs, including hostname, scheme, port and context path. Set to true if your application accesses Keycloak via a private network. If set to true, **hostname** option needs to be specified as a full URL. CLI: **--hostname-backchannel-dynamic** Env: **KC_HOSTNAME_BACKCHANNEL_DYNAMIC** Available only when hostname:v2 feature is enabled	**true**, **false** (default)
hostname-debug Toggles the hostname debug page that is accessible at /realms/master/hostname-debug. CLI: **--hostname-debug** Env: **KC_HOSTNAME_DEBUG** Available only when hostname:v2 feature is enabled	**true**, **false** (default)

	Value
hostname-strict Disables dynamically resolving the hostname from request headers. Should always be set to true in production, unless your reverse proxy overwrites the Host header. If enabled, the **hostname** option needs to be specified. CLI: **--hostname-strict** Env: **KC_HOSTNAME_STRICT** Available only when hostname:v2 feature is enabled	**true** (default), **false**

21.7. HTTP(S)

	Value
http-enabled Enables the HTTP listener. Enabled by default in development mode. Typically not enabled in production unless the server is fronted by a TLS termination proxy. CLI: **--http-enabled** Env: **KC_HTTP_ENABLED**	**true**, **false** (default)
http-host The HTTP Host. CLI: **--http-host** Env: **KC_HTTP_HOST**	**0.0.0.0** (default)
http-max-queued-requests Maximum number of queued HTTP requests. Use this to shed load in an overload situation. Excess requests will return a "503 Server not Available" response. CLI: **--http-max-queued-requests** Env: **KC_HTTP_MAX_QUEUED_REQUESTS**	

	Value
http-metrics-histograms-enabled Enables a histogram with default buckets for the duration of HTTP server requests. CLI: **--http-metrics-histograms-enabled** Env: **KC_HTTP_METRICS_HISTOGRAMS_ENABLED** Available only when metrics are enabled	**true**, **false** (default)
http-metrics-slos Service level objectives for HTTP server requests. Use this instead of the default histogram, or use it in combination to add additional buckets. Specify a list of comma-separated values defined in milliseconds. Example with buckets from 5ms to 10s: 5,10,25,50,250,500,1000,2500,5000,10000 CLI: **--http-metrics-slos** Env: **KC_HTTP_METRICS_SLOS** Available only when metrics are enabled	
http-pool-max-threads The maximum number of threads. If this is not specified then it will be automatically sized to the greater of 4 * the number of available processors and 50. For example if there are 4 processors the max threads will be 50. If there are 48 processors it will be 192. CLI: **--http-pool-max-threads** Env: **KC_HTTP_POOL_MAX_THREADS**	
http-port The used HTTP port. CLI: **--http-port** Env: **KC_HTTP_PORT**	**8080** (default)
http-relative-path ▮ Set the path relative to / for serving resources. The path must start with a /. CLI: **--http-relative-path** Env: **KC_HTTP_RELATIVE_PATH**	/ (default)

	Value
https-certificate-file The file path to a server certificate or certificate chain in PEM format. CLI: **--https-certificate-file** Env: **KC_HTTPS_CERTIFICATE_FILE**	
https-certificate-key-file The file path to a private key in PEM format. CLI: **--https-certificate-key-file** Env: **KC_HTTPS_CERTIFICATE_KEY_FILE**	
https-certificates-reload-period Interval on which to reload key store, trust store, and certificate files referenced by https-* options. May be a java.time.Duration value, an integer number of seconds, or an integer followed by one of [ms, h, m, s, d]. Must be greater than 30 seconds. Use -1 to disable. CLI: **--https-certificates-reload-period** Env: **KC_HTTPS_CERTIFICATES_RELOAD_PERIOD**	**1h** (default)
https-cipher-suites The cipher suites to use. If none is given, a reasonable default is selected. CLI: **--https-cipher-suites** Env: **KC_HTTPS_CIPHER_SUITES**	
https-client-auth ▮ Configures the server to require/request client authentication. CLI: **--https-client-auth** Env: **KC_HTTPS_CLIENT_AUTH**	**none** (default), **request**, **required**
https-key-store-file The key store which holds the certificate information instead of specifying separate files. CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	

	Value
https-key-store-password The password of the key store file. CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**password** (default)
https-key-store-type The type of the key store file. If not given, the type is automatically detected based on the file extension. If **fips-mode** is set to **strict** and no value is set, it defaults to **BCFKS**. CLI: **--https-key-store-type** Env: **KC_HTTPS_KEY_STORE_TYPE**	
https-port The used HTTPS port. CLI: **--https-port** Env: **KC_HTTPS_PORT**	**8443** (default)
https-protocols The list of protocols to explicitly enable. If a value is not supported by the JRE / security configuration, it will be silently ignored. CLI: **--https-protocols** Env: **KC_HTTPS_PROTOCOLS**	**TLSv1.3**, **TLSv1.2**, or any
https-trust-store-file The trust store which holds the certificate information of the certificates to trust. CLI: **--https-trust-store-file** Env: **KC_HTTPS_TRUST_STORE_FILE**	
https-trust-store-password The password of the trust store file. CLI: **--https-trust-store-password** Env: **KC_HTTPS_TRUST_STORE_PASSWORD**	

	Value
https-trust-store-type The type of the trust store file. If not given, the type is automatically detected based on the file extension. 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**	

21.8. HEALTH

	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)

21.9. MANAGEMENT

	Value
http-management-port Port of the management interface. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--http-management-port** Env: **KC_HTTP_MANAGEMENT_PORT**	**9000** (default)
http-management-relative-path ▮ Set the path relative to / for serving resources from management interface. The path must start with a /. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--http-management-relative-path** Env: **KC_HTTP_MANAGEMENT_RELATIVE_PATH**	/ (default)

	Value
https-management-certificate-file The file path to a server certificate or certificate chain in PEM format for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificate-file** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATE_FILE**	
https-management-certificate-key-file The file path to a private key in PEM format for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificate-key-file** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATE_KEY_FILE**	
https-management-certificates-reload-period Interval on which to reload key store, trust store, and certificate files referenced by https-management-* options for the management server. May be a java.time.Duration value, an integer number of seconds, or an integer followed by one of [ms, h, m, s, d]. Must be greater than 30 seconds. Use -1 to disable. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-certificates-reload-period** Env: **KC_HTTPS_MANAGEMENT_CERTIFICATES_RELOAD_PERIOD**	**1h** (default)
https-management-client-auth ▮ Configures the management interface to require/request client authentication. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-client-auth** Env: **KC_HTTPS_MANAGEMENT_CLIENT_AUTH**	**none** (default), **request**, **required**

	Value
https-management-key-store-file The key store which holds the certificate information instead of specifying separate files for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-key-store-file** Env: **KC_HTTPS_MANAGEMENT_KEY_STORE_FILE**	
https-management-key-store-password The password of the key store file for the management server. If not given, the value is inherited from HTTP options. Relevant only when something is exposed on the management interface - see the guide for details. CLI: **--https-management-key-store-password** Env: **KC_HTTPS_MANAGEMENT_KEY_STORE_PASSWORD**	**password** (default)
legacy-observability-interface ▮ If metrics/health endpoints should be exposed on the main HTTP server (not recommended). If set to true, the management interface is disabled. CLI: **--legacy-observability-interface** Env: **KC_LEGACY_OBSERVABILITY_INTERFACE** DEPRECATED.	**true**, **false** (default)

21.10. METRICS

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

21.11. PROXY

	Value
proxy-headers The proxy headers that should be accepted by the server. Misconfiguration might leave the server exposed to security vulnerabilities. Takes precedence over the deprecated proxy option. CLI: **--proxy-headers** Env: **KC_PROXY_HEADERS**	**forwarded**, **xforwarded**
proxy-protocol-enabled Whether the server should use the HA PROXY protocol when serving requests from behind a proxy. When set to true, the remote address returned will be the one from the actual connecting client. Cannot be enabled when the **proxy-headers** is used. CLI: **--proxy-protocol-enabled** Env: **KC_PROXY_PROTOCOL_ENABLED**	**true**, **false** (default)
proxy-trusted-addresses A comma separated list of trusted proxy addresses. If set, then proxy headers from other addresses will be ignored. By default all addresses are trusted. A trusted proxy address is specified as an IP address (IPv4 or IPv6) or Classless Inter-Domain Routing (CIDR) notation. Available only when proxy-headers is set. CLI: **--proxy-trusted-addresses** Env: **KC_PROXY_TRUSTED_ADDRESSES**	

21.12. VAULT

	Value
vault ▮ Enables a vault provider. CLI: **--vault** Env: **KC_VAULT**	**file**, **keystore**

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

21.13. LOGGING

	Value
log Enable one or more log handlers in a comma-separated list. CLI: **--log** Env: **KC_LOG**	**console**, **file**, **syslog**
log-console-color Enable or disable colors when logging to console. CLI: **--log-console-color** Env: **KC_LOG_CONSOLE_COLOR** Available only when Console log handler is activated	**true**, **false** (default)

	Value
log-console-format The format of unstructured console log entries. If the format has spaces in it, escape the value using "<format>". CLI: **--log-console-format** Env: **KC_LOG_CONSOLE_FORMAT** Available only when Console log handler is activated	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (default)
log-console-include-trace Include tracing information in the console log. If the **log-console-format** option is specified, this option has no effect. CLI: **--log-console-include-trace** Env: **KC_LOG_CONSOLE_INCLUDE_TRACE** Available only when Console log handler and Tracing is activated	**true** (default), **false**
log-console-json-format Set the format of the produced JSON. CLI: **--log-console-json-format** Env: **KC_LOG_CONSOLE_JSON_FORMAT** Available only when Console log handler is activated and output is set to 'json'	**default** (default), **ecs**
log-console-level Set the log level for the console handler. It specifies the most verbose log level for logs shown in the output. It respects levels specified in the **log-level** option, which represents the maximal verbosity for the whole logging system. For more information, check the Logging guide. CLI: **--log-console-level** Env: **KC_LOG_CONSOLE_LEVEL** Available only when Console log handler is activated	**off**, **fatal**, **error**, **warn**, **info**, **debug**, **trace**, **all** (default)
log-console-output Set the log output to JSON or default (plain) unstructured logging. CLI: **--log-console-output** Env: **KC_LOG_CONSOLE_OUTPUT** Available only when Console log handler is activated	**default** (default), **json**

	Value
log-file Set the log file path and filename. CLI: **--log-file** Env: **KC_LOG_FILE** Available only when File log handler is activated	**data/log/keycloak.log** (default)
log-file-format Set a format specific to file log entries. CLI: **--log-file-format** Env: **KC_LOG_FILE_FORMAT** Available only when File log handler is activated	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (default)
log-file-include-trace Include tracing information in the file log. If the **log-file-format** option is specified, this option has no effect. CLI: **--log-file-include-trace** Env: **KC_LOG_FILE_INCLUDE_TRACE** Available only when File log handler and Tracing is activated	**true** (default), **false**
log-file-json-format Set the format of the produced JSON. CLI: **--log-file-json-format** Env: **KC_LOG_FILE_JSON_FORMAT** Available only when File log handler is activated and output is set to 'json'	**default** (default), **ecs**
log-file-level Set the log level for the file handler. It specifies the most verbose log level for logs shown in the output. It respects levels specified in the **log-level** option, which represents the maximal verbosity for the whole logging system. For more information, check the Logging guide. CLI: **--log-file-level** Env: **KC_LOG_FILE_LEVEL** Available only when File log handler is activated	**off**, **fatal**, **error**, **warn**, **info**, **debug**, **trace**, **all** (default)

	Value
log-file-output Set the log output to JSON or default (plain) unstructured logging. CLI: **--log-file-output** Env: **KC_LOG_FILE_OUTPUT** Available only when File log handler is activated	**default** (default), **json**
log-level The log level of the root category or a comma-separated list of individual categories and their levels. For the root category, you don’t need to specify a category. CLI: **--log-level** Env: **KC_LOG_LEVEL**	**[info]** (default)
log-syslog-app-name Set the app name used when formatting the message in RFC5424 format. CLI: **--log-syslog-app-name** Env: **KC_LOG_SYSLOG_APP_NAME** Available only when Syslog is activated	**keycloak** (default)
log-syslog-endpoint Set the IP address and port of the Syslog server. CLI: **--log-syslog-endpoint** Env: **KC_LOG_SYSLOG_ENDPOINT** Available only when Syslog is activated	**localhost:514** (default)
log-syslog-format Set a format specific to Syslog entries. CLI: **--log-syslog-format** Env: **KC_LOG_SYSLOG_FORMAT** Available only when Syslog is activated	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (default)

	Value
log-syslog-include-trace Include tracing information in the Syslog. If the **log-syslog-format** option is specified, this option has no effect. CLI: **--log-syslog-include-trace** Env: **KC_LOG_SYSLOG_INCLUDE_TRACE** Available only when Syslog handler and Tracing is activated	**true** (default), **false**
log-syslog-json-format Set the format of the produced JSON. CLI: **--log-syslog-json-format** Env: **KC_LOG_SYSLOG_JSON_FORMAT** Available only when Syslog is activated and output is set to 'json'	**default** (default), **ecs**
log-syslog-level Set the log level for the Syslog handler. It specifies the most verbose log level for logs shown in the output. It respects levels specified in the **log-level** option, which represents the maximal verbosity for the whole logging system. For more information, check the Logging guide. CLI: **--log-syslog-level** Env: **KC_LOG_SYSLOG_LEVEL** Available only when Syslog is activated	**off**, **fatal**, **error**, **warn**, **info**, **debug**, **trace**, **all** (default)
log-syslog-max-length Set the maximum length, in bytes, of the message allowed to be sent. The length includes the header and the message. If not set, the default value is 2048 when **log-syslog-type** is rfc5424 (default) and 1024 when **log-syslog-type** is rfc3164. CLI: **--log-syslog-max-length** Env: **KC_LOG_SYSLOG_MAX_LENGTH** Available only when Syslog is activated	

	Value
log-syslog-output Set the Syslog output to JSON or default (plain) unstructured logging. CLI: **--log-syslog-output** Env: **KC_LOG_SYSLOG_OUTPUT** Available only when Syslog is activated	**default** (default), **json**
log-syslog-protocol Set the protocol used to connect to the Syslog server. CLI: **--log-syslog-protocol** Env: **KC_LOG_SYSLOG_PROTOCOL** Available only when Syslog is activated	**tcp** (default), **udp**, **ssl-tcp**
log-syslog-type Set the Syslog type used to format the sent message. CLI: **--log-syslog-type** Env: **KC_LOG_SYSLOG_TYPE** Available only when Syslog is activated	**rfc5424** (default), **rfc3164**

21.14. TRACING

	Value
tracing-compression OpenTelemetry compression method used to compress payloads. If unset, compression is disabled. CLI: **--tracing-compression** Env: **KC_TRACING_COMPRESSION** Available only when Tracing is enabled	**gzip**, **none** (default)
tracing-enabled ▮ Enables the OpenTelemetry tracing. CLI: **--tracing-enabled** Env: **KC_TRACING_ENABLED** Available only when 'opentelemetry' feature is enabled	**true**, **false** (default)

	Value
tracing-endpoint OpenTelemetry endpoint to connect to. CLI: **--tracing-endpoint** Env: **KC_TRACING_ENDPOINT** Available only when Tracing is enabled	**http://localhost:4317** (default)
tracing-jdbc-enabled ▮ Enables the OpenTelemetry JDBC tracing. CLI: **--tracing-jdbc-enabled** Env: **KC_TRACING_JDBC_ENABLED** Available only when Tracing is enabled	**true** (default), **false**
tracing-protocol OpenTelemetry protocol used for the telemetry data. CLI: **--tracing-protocol** Env: **KC_TRACING_PROTOCOL** Available only when Tracing is enabled	**grpc** (default), **http/protobuf**
tracing-resource-attributes OpenTelemetry resource attributes present in the exported trace to characterize the telemetry producer. Values in format **key1=val1,key2=val2**. For more information, check the Tracing guide. CLI: **--tracing-resource-attributes** Env: **KC_TRACING_RESOURCE_ATTRIBUTES** Available only when Tracing is enabled	
tracing-sampler-ratio OpenTelemetry sampler ratio. Probability that a span will be sampled. Expected double value in interval [0,1]. CLI: **--tracing-sampler-ratio** Env: **KC_TRACING_SAMPLER_RATIO** Available only when Tracing is enabled	**1.0** (default)

	Value
tracing-sampler-type ▮ OpenTelemetry sampler to use for tracing. CLI: **--tracing-sampler-type** Env: **KC_TRACING_SAMPLER_TYPE** Available only when Tracing is enabled	**always_on**, **always_off**, **traceidratio** (default), **parentbased_always_on**, **parentbased_always_off**, **parentbased_traceidratio**
tracing-service-name OpenTelemetry service name. Takes precedence over **service.name** defined in the **tracing-resource-attributes** property. CLI: **--tracing-service-name** Env: **KC_TRACING_SERVICE_NAME** Available only when Tracing is enabled	**keycloak** (default)

21.15. EVENTS

	Value
event-metrics-user-enabled ▮ Create metrics based on user events. CLI: **--event-metrics-user-enabled** Env: **KC_EVENT_METRICS_USER_ENABLED** Available only when metrics are enabled and feature user-event-metrics is enabled	**true**, **false** (default)
event-metrics-user-events Comma-separated list of events to be collected for user event metrics. This option can be used to reduce the number of metrics created as by default all user events create a metric. CLI: **--event-metrics-user-events** Env: **KC_EVENT_METRICS_USER_EVENTS** Available only when user event metrics are enabled	**authreqid_to_token**, **client_delete**, **client_info**, **client_initiated_account_linking**, **client_login**, **client_register**, **client_update**, **code_to_token**, **custom_required_action**, **delete_account**, **execute_action_token**, **execute_actions**,

	Value
Use **remove_credential** instead of **remove_totp**, and **update_credential** instead of **update_totp** and **update_password**. Deprecated values: **remove_totp**, **update_totp**, **update_password**	**federated_identity_link**, **federated_identity_override_link**, **grant_consent**, **identity_provider_first_login**, **identity_provider_link_account**, **identity_provider_login**, **identity_provider_post_login**, **identity_provider_response**, **identity_provider_retrieve_token**, **impersonate**, **introspect_token**, **invalid_signature**, **invite_org**, **login**, **logout**, **oauth2_device_auth**, **oauth2_device_code_to_token**, **oauth2_device_verify_user_code**, **oauth2_extension_grant**, **permission_token**, **pushed_authorization_request**, **refresh_token**, **register**, **register_node**, **remove_credential**, **remove_federated_identity**, **remove_totp** (deprecated), **reset_password**, **restart_authentication**, **revoke_grant**, **send_identity_provider_link**, **send_reset_password**, **send_verify_email**, **token_exchange**, **unregister_node**, **update_consent**, **update_credential**, **update_email**, **update_password** (deprecated),

	Value
	update_profile, **update_totp** (deprecated), **user_disabled_by_permanent_lockout**, **user_disabled_by_temporary_lockout**, **user_info_request**, **verify_email**, **verify_profile**
event-metrics-user-tags Comma-separated list of tags to be collected for user event metrics. By default only **realm** is enabled to avoid a high metrics cardinality. CLI: **--event-metrics-user-tags** Env: **KC_EVENT_METRICS_USER_TAGS** Available only when user event metrics are enabled	**realm**, **idp**, **clientId**

21.16. TRUSTSTORE

	Value
tls-hostname-verifier The TLS hostname verification policy for out-going HTTPS and SMTP requests. ANY should not be used in production. CLI: **--tls-hostname-verifier** Env: **KC_TLS_HOSTNAME_VERIFIER** STRICT and WILDCARD have been deprecated, use DEFAULT instead. Deprecated values: **STRICT**, **WILDCARD**	**ANY**, **WILDCARD** (deprecated), **STRICT** (deprecated), **DEFAULT** (default)
truststore-paths List of pkcs12 (p12, pfx, or pkcs12 file extensions), PEM files, or directories containing those files that will be used as a system truststore. CLI: **--truststore-paths** Env: **KC_TRUSTSTORE_PATHS**	

21.17. SECURITY

	Value
fips-mode ▮ Sets the FIPS mode. 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. CLI: **--fips-mode** Env: **KC_FIPS_MODE**	**non-strict**, **strict**

21.18. EXPORT

	Value
dir Set the path to a directory where files will be created with the exported data. CLI: **--dir** Env: **KC_DIR**	
file Set the path to a file that will be created with the exported data. To export more than 500 users, export to a directory with different files instead. CLI: **--file** Env: **KC_FILE**	
realm Set the name of the realm to export. If not set, all realms are going to be exported. CLI: **--realm** Env: **KC_REALM**	
users Set how users should be exported. CLI: **--users** Env: **KC_USERS**	**skip**, **realm_file**, **same_file**, **different_files** (default)

	Value
users-per-file Set the number of users per file. It is used only if **users** is set to **different_files**. Increasing this number leads to exponentially increasing export times. CLI: **--users-per-file** Env: **KC_USERS_PER_FILE**	**50** (default)

21.19. IMPORT

	Value
dir Set the path to a directory where files will be read from. CLI: **--dir** Env: **KC_DIR**	
file Set the path to a file that will be read. CLI: **--file** Env: **KC_FILE**	
override Set if existing data should be overwritten. If set to false, data will be ignored. CLI: **--override** Env: **KC_OVERRIDE**	**true** (default), **false**

21.20. BOOTSTRAP ADMIN

	Value
bootstrap-admin-client-id Client id for the temporary bootstrap admin service account. Used only when the master realm is created. Available only when bootstrap admin client secret is set. CLI: **--bootstrap-admin-client-id** Env: **KC_BOOTSTRAP_ADMIN_CLIENT_ID**	**temp-admin** (default)
bootstrap-admin-client-secret Client secret for the temporary bootstrap admin service account. Used only when the master realm is created. Use a non-CLI configuration option for this option if possible. CLI: **--bootstrap-admin-client-secret** Env: **KC_BOOTSTRAP_ADMIN_CLIENT_SECRET**	
bootstrap-admin-password Temporary bootstrap admin password. Used only when the master realm is created. Use a non-CLI configuration option for this option if possible. CLI: **--bootstrap-admin-password** Env: **KC_BOOTSTRAP_ADMIN_PASSWORD**	
bootstrap-admin-username Temporary bootstrap admin username. Used only when the master realm is created. Available only when bootstrap admin password is set. CLI: **--bootstrap-admin-username** Env: **KC_BOOTSTRAP_ADMIN_USERNAME**	**temp-admin** (default)

CHAPTER 22. ALL PROVIDER CONFIGURATION

Review provider configuration options.

22.1. AUTHENTICATION-SESSIONS

22.1.1. infinispan

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

22.1.2. remote

	Value
spi-authentication-sessions-remote-auth-sessions-limit The maximum number of concurrent authentication sessions per RootAuthenticationSession. CLI: **--spi-authentication-sessions-remote-auth-sessions-limit** Env: **KC_SPI_AUTHENTICATION_SESSIONS_REMOTE_AUTH_SESSIONS_LIMIT**	**300** (default) or any **int**
spi-authentication-sessions-remote-max-retries The maximum number of retries if an error occurs. A value of zero or less disable any retries. CLI: **--spi-authentication-sessions-remote-max-retries** Env: **KC_SPI_AUTHENTICATION_SESSIONS_REMOTE_MAX_RETRIES**	**10** (default) or any **int**

	Value
spi-authentication-sessions-remote-retry-base-time The base back-off time in milliseconds. CLI: **--spi-authentication-sessions-remote-retry-base-time** Env: **KC_SPI_AUTHENTICATION_SESSIONS_REMOTE_RETRY_BASE_TIME**	**10** (default) or any **int**

22.2. BRUTE-FORCE-PROTECTOR

22.2.1. default-brute-force-detector

	Value
spi-brute-force-protector-default-brute-force-detector-allow-concurrent-requests If concurrent logins are allowed by the brute force protection. 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**	**true**, **false** (default)

22.3. CIBA-AUTH-CHANNEL

22.3.1. ciba-http-auth-channel

	Value
spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri The HTTP(S) URI of the authentication channel. 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**	any **string**

22.4. CONNECTIONS-HTTP-CLIENT

22.4.1. default

	Value
spi-connections-http-client-default-client-key-password The key password. CLI: **--spi-connections-http-client-default-client-key-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEY_PASSWORD**	**-1** (default) or any **string**
spi-connections-http-client-default-client-keystore The file path of the key store from where the key material is going to be read from to set-up TLS connections. CLI: **--spi-connections-http-client-default-client-keystore** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE**	any **string**
spi-connections-http-client-default-client-keystore-password The key store password. CLI: **--spi-connections-http-client-default-client-keystore-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE_PASSWORD**	any **string**
spi-connections-http-client-default-connection-pool-size Assigns maximum total connection value. CLI: **--spi-connections-http-client-default-connection-pool-size** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_POOL_SIZE**	any **int**
spi-connections-http-client-default-connection-ttl-millis Sets maximum time, in milliseconds, to live for persistent connections. CLI: **--spi-connections-http-client-default-connection-ttl-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_TTL_MILLIS**	**-1** (default) or any **long**

	Value
spi-connections-http-client-default-disable-cookies Disables state (cookie) management. CLI: **--spi-connections-http-client-default-disable-cookies** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_COOKIES**	**true** (default), **false**
spi-connections-http-client-default-disable-trust-manager Disable trust management and hostname verification. 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. CLI: **--spi-connections-http-client-default-disable-trust-manager** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_TRUST_MANAGER**	**true**, **false** (default)
spi-connections-http-client-default-establish-connection-timeout-millis When trying to make an initial socket connection, what is the timeout? CLI: **--spi-connections-http-client-default-establish-connection-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_ESTABLISH_CONNECTION_TIMEOUT_MILLIS**	**-1** (default) or any **long**
spi-connections-http-client-default-max-connection-idle-time-millis Sets the time, in milliseconds, for evicting idle connections from the pool. CLI: **--spi-connections-http-client-default-max-connection-idle-time-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_CONNECTION_IDLE_TIME_MILLIS**	**900000** (default) or any **long**

	Value
spi-connections-http-client-default-max-consumed-response-size Maximum size of a response consumed by the client (to prevent denial of service) CLI: **--spi-connections-http-client-default-max-consumed-response-size** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_CONSUMED_RESPONSE_SIZE**	**10000000** (default) or any **long**
spi-connections-http-client-default-max-pooled-per-route Assigns maximum connection per route value. CLI: **--spi-connections-http-client-default-max-pooled-per-route** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_POOLED_PER_ROUTE**	**64** (default) or any **int**
spi-connections-http-client-default-proxy-mappings Denotes the combination of a regex based hostname pattern and a proxy-uri in the form of hostnamePattern;proxyUri. CLI: **--spi-connections-http-client-default-proxy-mappings** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_PROXY_MAPPINGS**	any **string**
spi-connections-http-client-default-reuse-connections If connections should be reused. CLI: **--spi-connections-http-client-default-reuse-connections** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_REUSE_CONNECTIONS**	**true** (default), **false**
spi-connections-http-client-default-socket-timeout-millis Socket inactivity timeout. CLI: **--spi-connections-http-client-default-socket-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_SOCKET_TIMEOUT_MILLIS**	**5000** (default) or any **long**

22.4.2. opentelemetry

	Value
spi-connections-http-client-opentelemetry-client-key-password The key password. CLI: **--spi-connections-http-client-opentelemetry-client-key-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_CLIENT_KEY_PASSWORD**	**-1** (default) or any **string**
spi-connections-http-client-opentelemetry-client-keystore The file path of the key store from where the key material is going to be read from to set-up TLS connections. CLI: **--spi-connections-http-client-opentelemetry-client-keystore** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_CLIENT_KEYSTORE**	any **string**
spi-connections-http-client-opentelemetry-client-keystore-password The key store password. CLI: **--spi-connections-http-client-opentelemetry-client-keystore-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_CLIENT_KEYSTORE_PASSWORD**	any **string**
spi-connections-http-client-opentelemetry-connection-pool-size Assigns maximum total connection value. CLI: **--spi-connections-http-client-opentelemetry-connection-pool-size** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_CONNECTION_POOL_SIZE**	any **int**
spi-connections-http-client-opentelemetry-connection-ttl-millis Sets maximum time, in milliseconds, to live for persistent connections. CLI: **--spi-connections-http-client-opentelemetry-connection-ttl-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_CONNECTION_TTL_MILLIS**	**-1** (default) or any **long**

	Value
spi-connections-http-client-opentelemetry-disable-cookies Disables state (cookie) management. CLI: **--spi-connections-http-client-opentelemetry-disable-cookies** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_DISABLE_COOKIES**	**true** (default), **false**
spi-connections-http-client-opentelemetry-disable-trust-manager Disable trust management and hostname verification. 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. CLI: **--spi-connections-http-client-opentelemetry-disable-trust-manager** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_DISABLE_TRUST_MANAGER**	**true**, **false** (default)
spi-connections-http-client-opentelemetry-establish-connection-timeout-millis When trying to make an initial socket connection, what is the timeout? CLI: **--spi-connections-http-client-opentelemetry-establish-connection-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_ESTABLISH_CONNECTION_TIMEOUT_MILLIS**	**-1** (default) or any **long**
spi-connections-http-client-opentelemetry-max-connection-idle-time-millis Sets the time, in milliseconds, for evicting idle connections from the pool. CLI: **--spi-connections-http-client-opentelemetry-max-connection-idle-time-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_MAX_CONNECTION_IDLE_TIME_MILLIS**	**900000** (default) or any **long**

	Value
spi-connections-http-client-opentelemetry-max-consumed-response-size Maximum size of a response consumed by the client (to prevent denial of service) CLI: **--spi-connections-http-client-opentelemetry-max-consumed-response-size** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_MAX_CONSUMED_RESPONSE_SIZE**	**10000000** (default) or any **long**
spi-connections-http-client-opentelemetry-max-pooled-per-route Assigns maximum connection per route value. CLI: **--spi-connections-http-client-opentelemetry-max-pooled-per-route** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_MAX_POOLED_PER_ROUTE**	**64** (default) or any **int**
spi-connections-http-client-opentelemetry-proxy-mappings Denotes the combination of a regex based hostname pattern and a proxy-uri in the form of hostnamePattern;proxyUri. CLI: **--spi-connections-http-client-opentelemetry-proxy-mappings** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_PROXY_MAPPINGS**	any **string**
spi-connections-http-client-opentelemetry-reuse-connections If connections should be reused. CLI: **--spi-connections-http-client-opentelemetry-reuse-connections** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_REUSE_CONNECTIONS**	**true** (default), **false**
spi-connections-http-client-opentelemetry-socket-timeout-millis Socket inactivity timeout. CLI: **--spi-connections-http-client-opentelemetry-socket-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_OPENTELEMETRY_SOCKET_TIMEOUT_MILLIS**	**5000** (default) or any **long**

22.5. CONNECTIONS-INFINISPAN

22.5.1. quarkus

	Value
spi-connections-infinispan-quarkus-site-name Site name for multi-site deployments CLI: **--spi-connections-infinispan-quarkus-site-name** Env: **KC_SPI_CONNECTIONS_INFINISPAN_QUARKUS_SITE_NAME**	any **string**

22.6. CONNECTIONS-JPA

22.6.1. quarkus

	Value
spi-connections-jpa-quarkus-initialize-empty Initialize database if empty. 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. CLI: **--spi-connections-jpa-quarkus-initialize-empty** Env: **KC_SPI_CONNECTIONS_JPA_QUARKUS_INITIALIZE_EMPTY**	**true** (default), **false**
spi-connections-jpa-quarkus-migration-export Path for where to write manual database initialization/migration file. CLI: **--spi-connections-jpa-quarkus-migration-export** Env: **KC_SPI_CONNECTIONS_JPA_QUARKUS_MIGRATION_EXPORT**	any **string**
spi-connections-jpa-quarkus-migration-strategy Strategy to use to migrate database. 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. CLI: **--spi-connections-jpa-quarkus-migration-strategy** Env: **KC_SPI_CONNECTIONS_JPA_QUARKUS_MIGRATION_STRATEGY**	**update** (default), **manual**, **validate**

22.7. CREDENTIAL

22.7.1. keycloak-password

	Value
spi-credential-keycloak-password-validations-counter-tags Comma-separated list of tags to be used when publishing password validation counter metric. CLI: **--spi-credential-keycloak-password-validations-counter-tags** Env: **KC_SPI_CREDENTIAL_KEYCLOAK_PASSWORD_VALIDATIONS_COUNTER_TAGS**	**realm**, **algorithm**, **hashing_strength**, **outcome**

22.8. CRL-STORAGE

22.8.1. infinispan

	Value
spi-crl-storage-infinispan-cache-time Interval in seconds that the CRL is cached. The next update time of the CRL is always a minimum if present. Zero or a negative value means CRL is cached until the next update time specified in the CRL (or infinite if the CRL does not contain the next update). CLI: **--spi-crl-storage-infinispan-cache-time** Env: **KC_SPI_CRL_STORAGE_INFINISPAN_CACHE_TIME**	**-1** (default) or any **int**
spi-crl-storage-infinispan-min-time-between-requests Minimum interval in seconds between two requests to retrieve the CRL. The CRL is not updated from the URL again until this minimum time has passed since the previous refresh. In theory this option is never used if the CRL is refreshed correctly in the next update time. The interval should be a positive number. Default 10 seconds. CLI: **--spi-crl-storage-infinispan-min-time-between-requests** Env: **KC_SPI_CRL_STORAGE_INFINISPAN_MIN_TIME_BETWEEN_REQUESTS**	**10** (default) or any **int**

22.9. DATASTORE

22.9.1. legacy

	Value
spi-datastore-legacy-allow-migrate-existing-database-to-snapshot By default, it is not allowed to run the snapshot/development server against the database, which was previously migrated to some officially released server version. As an attempt of doing this indicates that you are trying to run development server against production database, which can result in a loss or corruption of data, and also does not allow upgrading. If it is really intended, you can use this option, which will allow to use nightly/development server against production database when explicitly switch to true. This option is recommended just in the development environments and should be never used in the production! CLI: **--spi-datastore-legacy-allow-migrate-existing-database-to-snapshot** Env: **KC_SPI_DATASTORE_LEGACY_ALLOW_MIGRATE_EXISTING_DATABASE_TO_SNAPSHOT**	**true**, **false** (default)

22.10. DBLOCK

22.10.1. jpa

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

22.11. EVENTS-LISTENER

22.11.1. email

	Value
spi-events-listener-email-exclude-events A comma-separated list of events that should not be sent via email to the user's account. CLI: **--spi-events-listener-email-exclude-events** Env: **KC_SPI_EVENTS_LISTENER_EMAIL_EXCLUDE_EVENTS**	**authreqid_to_token**, **authreqid_to_token_error**, **client_delete**, **client_delete_error**, **client_info**, **client_info_error**, **client_initiated_acco**

	Value
	unt_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**, **federated_identity_override_link**, **federated_identity_override_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**, **identity_provider_post_login_error**, **identity_provider_re**

	Value
	sponse, 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, invite_org, invite_org_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, oauth2_extension_grant, oauth2_extension_grant_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_credential, remove_credential_error, remove_federated_identity, remove_federated_identity_error, remove_totp,

	Value
	remove_totp_error, reset_password, reset_password_error, restart_authentication, restart_authentication_error, revoke_grant, revoke_grant_error, send_identity_provider_link, send_identity_provider_link_error, send_reset_password, send_reset_password_error, send_verify_email, send_verify_email_error, token_exchange, token_exchange_error, unregister_node, unregister_node_error, update_consent, update_consent_error, update_credential, update_credential_error, update_email, update_email_error, update_password, update_password_error, update_profile, update_profile_error, update_totp, update_totp_error, user_disabled_by_permanent_lockout, user_disabled_by_permanent_lockout_error, user_disabled_by_temporary_lockout, user_disabled_by_temporary_lockout_error, user_info_request, user_info_request_error, validate_access_tok

	Value
	en, validate_access_token_error, verify_email, verify_email_error, verify_profile, verify_profile_error
spi-events-listener-email-include-events A comma-separated list of events that should be sent via email to the user’s account. CLI: **--spi-events-listener-email-include-events** Env: **KC_SPI_EVENTS_LISTENER_EMAIL_INCLUDE_EVENTS**	**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**, **federated_identity_override_link**, **federated_identity_override_link_error**, **grant_consent**, **grant_consent_error**, **identity_provider_first_login**,

	Value
	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**, **identity_provider_post_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**, **invite_org**, **invite_org_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**, **oauth2_extension_grant**, **oauth2_extension_grant_error**, **permission_token**, **permission_token_error**, **pushed_authorization_request**,

	Value
	pushed_authorization_request_error, **refresh_token**, **refresh_token_error**, **register**, **register_error**, **register_node**, **register_node_error**, **remove_credential**, **remove_credential_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**, **revoke_grant_error**, **send_identity_provider_link**, **send_identity_provider_link_error**, **send_reset_password**, **send_reset_password_error**, **send_verify_email**, **send_verify_email_error**, **token_exchange**, **token_exchange_error**, **unregister_node**, **unregister_node_error**, **update_consent**, **update_consent_error**, **update_credential**, **update_credential_error**, **update_email**, **update_email_error**, **update_password**, **update_password_error**, **update_profile**, **update_profile_error**, **update_totp**,

	Value
	update_totp_error, **user_disabled_by_permanent_lockout**, **user_disabled_by_permanent_lockout_error**, **user_disabled_by_temporary_lockout**, **user_disabled_by_temporary_lockout_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**

22.11.2. jboss-logging

	Value
spi-events-listener-jboss-logging-error-level The log level for error messages. CLI: **--spi-events-listener-jboss-logging-error-level** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_ERROR_LEVEL**	**debug**, **error**, **fatal**, **info**, **trace**, **warn** (default)
spi-events-listener-jboss-logging-include-representation When "true" the "representation" field with the JSON admin object is also added to the message. The realm should be also configured to include representation for the admin events. CLI: **--spi-events-listener-jboss-logging-include-representation** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_INCLUDE_REPRESENTATION**	**true**, **false** (default)

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

22.12. EXPORT

22.12.1. dir

	Value
spi-export-dir-dir Directory to export to CLI: **--spi-export-dir-dir** Env: **KC_SPI_EXPORT_DIR_DIR**	any **string**
spi-export-dir-realm-name Realm to export CLI: **--spi-export-dir-realm-name** Env: **KC_SPI_EXPORT_DIR_REALM_NAME**	any **string**

	Value
spi-export-dir-users-export-strategy Users export strategy CLI: **--spi-export-dir-users-export-strategy** Env: **KC_SPI_EXPORT_DIR_USERS_EXPORT_STRATEGY**	**DIFFERENT_FILES** (default) or any **string**
spi-export-dir-users-per-file Users per exported file CLI: **--spi-export-dir-users-per-file** Env: **KC_SPI_EXPORT_DIR_USERS_PER_FILE**	**50** (default) or any **int**

22.12.2. single-file

	Value
spi-export-single-file-file File to export to CLI: **--spi-export-single-file-file** Env: **KC_SPI_EXPORT_SINGLE_FILE_FILE**	any **string**
spi-export-single-file-realm-name Realm to export CLI: **--spi-export-single-file-realm-name** Env: **KC_SPI_EXPORT_SINGLE_FILE_REALM_NAME**	any **string**

22.13. GROUP

22.13.1. jpa

	Value
spi-group-jpa-escape-slashes-in-group-path If true slashes/ in group names are escaped with the character~ when converted to paths. CLI: **--spi-group-jpa-escape-slashes-in-group-path** Env: **KC_SPI_GROUP_JPA_ESCAPE_SLASHES_IN_GROUP_PATH**	**true**, **false** (default)
spi-group-jpa-searchable-attributes The list of attributes separated by comma that are allowed in client attribute searches. CLI: **--spi-group-jpa-searchable-attributes** Env: **KC_SPI_GROUP_JPA_SEARCHABLE_ATTRIBUTES**	any **string**

22.14. IMPORT

22.14.1. dir

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

22.14.2. single-file

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

22.15. LOAD-BALANCER-CHECK

22.15.1. remote

	Value
spi-load-balancer-check-remote-poll-interval The Remote caches poll interval, in milliseconds, for connection availability CLI: **--spi-load-balancer-check-remote-poll-interval** Env: **KC_SPI_LOAD_BALANCER_CHECK_REMOTE_POLL_INTERVAL**	**5000** (default) or any **int**

22.16. LOGIN-PROTOCOL

22.16.1. openid-connect

	Value
spi-login-protocol-openid-connect-add-req-params-fail-fast Whether the fail-fast strategy should be enforced in case if the limit for some standard OIDC parameter or additional OIDC parameter is not met for the parameters sent to the OIDC authentication request. If false, then all additional request parameters to not meet the configuration are silently ignored. If true, an exception will be raised and OIDC authentication request will not be allowed. CLI: **--spi-login-protocol-openid-connect-add-req-params-fail-fast** Env: **KC_SPI_LOGIN_PROTOCOL_OPENID_CONNECT_ADD_REQ_PARAMS_FAIL_FAST**	**true**, **false** (default)
spi-login-protocol-openid-connect-add-req-params-max-number Maximum number of additional request parameters sent to the OIDC authentication request. As 'additional request parameter' is meant some custom parameter not directly treated as standard OIDC/OAuth2 protocol parameter. Additional parameters might be useful for example to add custom claims to the OIDC token (in case that also particular protocol mappers are configured). CLI: **--spi-login-protocol-openid-connect-add-req-params-max-number** Env: **KC_SPI_LOGIN_PROTOCOL_OPENID_CONNECT_ADD_REQ_PARAMS_MAX_NUMBER**	**5** (default) or any **int**
spi-login-protocol-openid-connect-add-req-params-max-overall-size Maximum size of all additional request parameters values together. See **add-req-params-max-number** for more details about additional request parameters CLI: **--spi-login-protocol-openid-connect-add-req-params-max-overall-size** Env: **KC_SPI_LOGIN_PROTOCOL_OPENID_CONNECT_ADD_REQ_PARAMS_MAX_OVERALL_SIZE**	**2147483647** (default) or any **int**

	Value
spi-login-protocol-openid-connect-add-req-params-max-size Maximum size of single additional request parameter value See**add-req-params-max-number** for more details about additional request parameters CLI: **--spi-login-protocol-openid-connect-add-req-params-max-size** Env: **KC_SPI_LOGIN_PROTOCOL_OPENID_CONNECT_ADD_REQ_PARAMS_MAX_SIZE**	**2000** (default) or any **int**
spi-login-protocol-openid-connect-req-params-default-max-size Maximum default length of the standard OIDC parameter sent to the OIDC authentication request. This applies to most of the standard parameters like for example**state**, **nonce** etc. The exception is **login_hint** parameter, which has maximum length of 255 characters. CLI: **--spi-login-protocol-openid-connect-req-params-default-max-size** Env: **KC_SPI_LOGIN_PROTOCOL_OPENID_CONNECT_REQ_PARAMS_DEFAULT_MAX_SIZE**	**4000** (default) or any **int**
spi-login-protocol-openid-connect-req-params-max-size—login_hint Maximum length of the standard OIDC authentication request parameter overriden for the specified parameter. Useful if some standard OIDC parameter should have different limit than**req-params-default-max-size**. It is needed to add the name of the parameter after this prefix into the configuration. In this example, the **login_hint** parameter is used, but this format is supported for any known standard OIDC/OAuth2 parameter. CLI: **--spi-login-protocol-openid-connect-req-params-max-size—login_hint** Env: **KC_SPI_LOGIN_PROTOCOL_OPENID_CONNECT_REQ_PARAMS_MAX_SIZE__LOGIN_HINT**	any **int**

22.17. LOGIN-FAILURE

22.17.1. remote

	Value
spi-login-failure-remote-max-retries The maximum number of retries if an error occurs. A value of zero or less disable any retries. CLI: **--spi-login-failure-remote-max-retries** Env: **KC_SPI_LOGIN_FAILURE_REMOTE_MAX_RETRIES**	**10** (default) or any **int**
spi-login-failure-remote-retry-base-time The base back-off time in milliseconds. CLI: **--spi-login-failure-remote-retry-base-time** Env: **KC_SPI_LOGIN_FAILURE_REMOTE_RETRY_BASE_TIME**	**10** (default) or any **int**

22.18. PASSWORD-HASHING

22.18.1. argon2

	Value
spi-password-hashing-argon2-cpu-cores Maximum parallel CPU cores to use for hashing CLI: **--spi-password-hashing-argon2-cpu-cores** Env: **KC_SPI_PASSWORD_HASHING_ARGON2_CPU_CORES**	any **int**
spi-password-hashing-argon2-hash-length Hash length CLI: **--spi-password-hashing-argon2-hash-length** Env: **KC_SPI_PASSWORD_HASHING_ARGON2_HASH_LENGTH**	**32** (default) or any **int**
spi-password-hashing-argon2-iterations Iterations CLI: **--spi-password-hashing-argon2-iterations** Env: **KC_SPI_PASSWORD_HASHING_ARGON2_ITERATIONS**	**5** (default) or any **int**

	Value
spi-password-hashing-argon2-memory Memory size (KB) CLI: **--spi-password-hashing-argon2-memory** Env: **KC_SPI_PASSWORD_HASHING_ARGON2_MEMORY**	**7168** (default) or any **int**
spi-password-hashing-argon2-parallelism Parallelism CLI: **--spi-password-hashing-argon2-parallelism** Env: **KC_SPI_PASSWORD_HASHING_ARGON2_PARALLELISM**	**1** (default) or any **int**
spi-password-hashing-argon2-type Type CLI: **--spi-password-hashing-argon2-type** Env: **KC_SPI_PASSWORD_HASHING_ARGON2_TYPE**	**id** (default), **d**, **i**
spi-password-hashing-argon2-version Version CLI: **--spi-password-hashing-argon2-version** Env: **KC_SPI_PASSWORD_HASHING_ARGON2_VERSION**	**1.3** (default), **1.0**

22.19. PUBLIC-KEY-STORAGE

22.19.1. infinispan

	Value
spi-public-key-storage-infinispan-max-cache-time Maximum interval in seconds that keys are cached when they are retrieved via all keys methods. When all keys for the entry are retrieved there is no way to detect if a key is missing (different to the case when the key is retrieved via ID for example). In that situation this option forces a refresh from time to time. Default 24 hours. CLI: **--spi-public-key-storage-infinispan-max-cache-time** Env: **KC_SPI_PUBLIC_KEY_STORAGE_INFINISPAN_MAX_CACHE_TIME**	**86400** (default) or any **int**

	Value
spi-public-key-storage-infinispan-min-time-between-requests Minimum interval in seconds between two requests to retrieve the new public keys. The server will always try to download new public keys when a single key is requested and not found. However it will avoid the download if the previous refresh was done less than 10 seconds ago (by default). This behavior is used to avoid DoS attacks against the external keys endpoint. CLI: **--spi-public-key-storage-infinispan-min-time-between-requests** Env: **KC_SPI_PUBLIC_KEY_STORAGE_INFINISPAN_MIN_TIME_BETWEEN_REQUESTS**	**10** (default) or any **int**

22.20. REQUIRED-ACTION

22.20.1. UPDATE_PASSWORD

	Value
spi-required-action-UPDATE_PASSWORD-max_auth_age Configures the duration in seconds this action can be used after the last authentication before the user is required to re-authenticate. This parameter is used just in the context of AIA when the kc_action parameter is available in the request, which is for instance when user himself updates his password in the account console. When the 'Maximum Authentication Age' password policy is used in the realm, it’s value has precedence over the value configured here. CLI: **--spi-required-action-UPDATE_PASSWORD-max_auth_age** Env: **KC_SPI_REQUIRED_ACTION_UPDATE_PASSWORD_MAX_AUTH_AGE**	**300** (default) or any **String**

22.21. RESOURCE-ENCODING

22.21.1. gzip

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

22.22. SECURITY-PROFILE

22.22.1. default

	Value
spi-security-profile-default-name Name for the security configuration file to use. File **name**.json is searched in classapth and **conf** installation folder. CLI: **--spi-security-profile-default-name** Env: **KC_SPI_SECURITY_PROFILE_DEFAULT_NAME**	any **string**

22.23. SINGLE-USE-OBJECT

22.23.1. infinispan

	Value
spi-single-use-object-infinispan-persist-revoked-tokens If revoked tokens are stored persistently across restarts CLI: **--spi-single-use-object-infinispan-persist-revoked-tokens** Env: **KC_SPI_SINGLE_USE_OBJECT_INFINISPAN_PERSIST_REVOKED_TOKENS**	**true** (default), **false**

22.23.2. remote

	Value
spi-single-use-object-remote-persist-revoked-tokens If revoked tokens are stored persistently across restarts CLI: **--spi-single-use-object-remote-persist-revoked-tokens** Env: **KC_SPI_SINGLE_USE_OBJECT_REMOTE_PERSIST_REVOKED_TOKENS**	**true** (default), **false**

22.24. STICKY-SESSION-ENCODER

22.24.1. infinispan

	Value
spi-sticky-session-encoder-infinispan-should-attach-route If the route should be attached to cookies to reflect the node that owns a particular session. CLI: **--spi-sticky-session-encoder-infinispan-should-attach-route** Env: **KC_SPI_STICKY_SESSION_ENCODER_INFINISPAN_SHOULD_ATTACH_ROUTE**	**true** (default), **false**

22.24.2. remote

	Value
spi-sticky-session-encoder-remote-should-attach-route If the route should be attached to cookies to reflect the node that owns a particular session. CLI: **--spi-sticky-session-encoder-remote-should-attach-route** Env: **KC_SPI_STICKY_SESSION_ENCODER_REMOTE_SHOULD_ATTACH_ROUTE**	**true** (default), **false**

22.25. STORAGE

22.25.1. ldap

	Value
spi-storage-ldap-secure-referral Allow only secure LDAP referrals (deprecated) CLI: **--spi-storage-ldap-secure-referral** Env: **KC_SPI_STORAGE_LDAP_SECURE_REFERRAL**	**true** (default), **false**

22.26. TRUSTSTORE

22.26.1. file

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

22.27. USER-PROFILE

22.27.1. declarative-user-profile

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

22.28. USER-SESSIONS

22.28.1. infinispan

	Value
spi-user-sessions-infinispan-max-batch-size Maximum size of a batch size (only applicable to persistent sessions CLI: **--spi-user-sessions-infinispan-max-batch-size** Env: **KC_SPI_USER_SESSIONS_INFINISPAN_MAX_BATCH_SIZE**	**4** (default) or any **int**

	Value
spi-user-sessions-infinispan-offline-client-session-cache-entry-lifespan-override Override how long offline client sessions should be kept in memory in seconds CLI: **--spi-user-sessions-infinispan-offline-client-session-cache-entry-lifespan-override** Env: **KC_SPI_USER_SESSIONS_INFINISPAN_OFFLINE_CLIENT_SESSION_CACHE_ENTRY_LIFESPAN_OVERRIDE**	any **int**
spi-user-sessions-infinispan-offline-session-cache-entry-lifespan-override Override how long offline user sessions should be kept in memory in seconds CLI: **--spi-user-sessions-infinispan-offline-session-cache-entry-lifespan-override** Env: **KC_SPI_USER_SESSIONS_INFINISPAN_OFFLINE_SESSION_CACHE_ENTRY_LIFESPAN_OVERRIDE**	any **int**
spi-user-sessions-infinispan-use-caches Enable or disable caches. Enabled by default unless the external feature to use only external remote caches is used CLI: **--spi-user-sessions-infinispan-use-caches** Env: **KC_SPI_USER_SESSIONS_INFINISPAN_USE_CACHES**	**true**, **false**

22.28.2. remote

	Value
spi-user-sessions-remote-batch-size Batch size when streaming session from the remote cache CLI: **--spi-user-sessions-remote-batch-size** Env: **KC_SPI_USER_SESSIONS_REMOTE_BATCH_SIZE**	**1024** (default) or any **int**

	Value
spi-user-sessions-remote-max-retries The maximum number of retries if an error occurs. A value of zero or less disable any retries. CLI: **--spi-user-sessions-remote-max-retries** Env: **KC_SPI_USER_SESSIONS_REMOTE_MAX_RETRIES**	**10** (default) or any **int**
spi-user-sessions-remote-retry-base-time The base back-off time in milliseconds. CLI: **--spi-user-sessions-remote-retry-base-time** Env: **KC_SPI_USER_SESSIONS_REMOTE_RETRY_BASE_TIME**	**10** (default) or any **int**

22.29. WELL-KNOWN

22.29.1. oauth-authorization-server

	Value
spi-well-known-oauth-authorization-server-include-client-scopes If client scopes should be used to calculate the list of supported scopes. CLI: **--spi-well-known-oauth-authorization-server-include-client-scopes** Env: **KC_SPI_WELL_KNOWN_OAUTH_AUTHORIZATION_SERVER_INCLUDE_CLIENT_SCOPES**	**true** (default), **false**
spi-well-known-oauth-authorization-server-openid-configuration-override The file path from where the metadata should be loaded from. 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. CLI: **--spi-well-known-oauth-authorization-server-openid-configuration-override** Env: **KC_SPI_WELL_KNOWN_OAUTH_AUTHORIZATION_SERVER_OPENID_CONFIGURATION_OVERRIDE**	any **string**

22.29.2. openid-configuration

	Value
spi-well-known-openid-configuration-include-client-scopes If client scopes should be used to calculate the list of supported scopes. CLI: **--spi-well-known-openid-configuration-include-client-scopes** Env: **KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_INCLUDE_CLIENT_SCOPES**	**true** (default), **false**
spi-well-known-openid-configuration-openid-configuration-override The file path from where the metadata should be loaded from. 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. CLI: **--spi-well-known-openid-configuration-openid-configuration-override** Env: **KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_OPENID_CONFIGURATION_OVERRIDE**	any **string**

CHAPTER 23. CHECKING IF ROLLING UPDATES ARE POSSIBLE

Execute the update compatibility command to check if Red Hat build of Keycloak supports a rolling update for a change in your deployment.

Use the update compatibility command to determine if you can update your deployment with a rolling update strategy when enabling or disabling features or changing the Red Hat build of Keycloak version, configurations or providers and themes. The outcome shows whether a rolling update is possible or if a recreate update is required.

In its current version, it shows that a rolling update is possible when the Red Hat build of Keycloak version is the same for the old and the new version. Future versions of Red Hat build of Keycloak might change that behavior to use additional information from the configuration, the image and the version to determine if a rolling update is possible.

This is fully scriptable, so your update procedure can use that information to perform a rolling or recreate strategy depending on the change performed. It is also GitOps friendly, as it allows storing the metadata of the previous configuration in a file. Use this file in a CI/CD pipeline with the new configuration to determine if a rolling update is possible or if a recreate update is needed.

If you are using the Red Hat build of Keycloak Operator, continue to the Avoiding downtime with rolling updates chapter and the **Auto** strategy for more information.

23.1. SUPPORTED UPDATE STRATEGIES

Rolling Update

In this guide, a rolling update is an update that can be performed with zero downtime for your deployment, which consists of at least two nodes. Update your Red Hat build of Keycloak one by one; shut down one of your old deployment nodes and start a new deployment node. Wait until the new node's start-up probe returns success before proceeding to the next Red Hat build of Keycloak node. See chapter Tracking instance status with health checks for details on how to enable and use the start-up probe.

Recreate Update

A recreate update is not compatible with zero-downtime and requires downtime to be applied. Shut down all nodes of the cluster running the old version before starting the nodes with the new version.

23.2. DETERMINING THE UPDATE STRATEGY FOR AN UPDATED CONFIGURATION

To determine if a rolling update is possible, run the update compatibility command:

1. Generate the required metadata with the old configuration.
2. Check the metadata with the new configuration to determine the update strategy.

WARNING

This command currently offers only a limited functionality. At the moment, it takes into consideration only the version of Red Hat build of Keycloak and the embedded Infinispan to determine if a rolling update is possible. If those are unchanged, it reports that a rolling update is possible.

The current version does not yet verify configuration changes and assumes all configuration changes are eligible for a rolling update. The same applies to changes to custom extensions and themes.

A good use case when to use this is, for example, when you want to do a rolling update when you change the Red Hat build of Keycloak theme or your custom extensions, and only want run recreate update when the version of Red Hat build of Keycloak changes which does not yet allow a rolling update.

While consumers of these commands should know the limitations that exist today, they should not rely on the internal behavior or the structure of the metadata file. Instead, they should rely only on the exit code of the **check** command to benefit from future enhancements on the internal logic to determine when a rolling update is possible.

23.2.1. Generating the Metadata

To generate the metadata, execute the following command using the same Red Hat build of Keycloak version and configuration options:

Generate and save the metadata from the current deployment.

```
bin/kc.[sh|bat] update-compatibility metadata --file=/path/to/file.json
```

This command accepts all options used by the **start** command. The command displays the metadata, in JSON format, in the console for debugging purposes. The **--file** parameter allows you to save the metadata to a file. Use this file with the subsequent **check** command.

WARNING

Ensure that all configuration options, whether set via environment variables or CLI arguments, are included when running the above command.

Omitting any configuration options results in incomplete metadata, and could lead to a wrong reported result in the next step.

23.2.2. Checking the Metadata

This command checks the metadata generated by the previous command and compares it with the current configuration and Red Hat build of Keycloak version. If you are upgrading to a new Red Hat build of Keycloak version, this command must be executed with the new version.

Check the metadata from a previous deployment.

```
bin/kc.[sh|bat] update-compatibility check --file=/path/to/file.json
```

WARNING

- Ensure that all configuration options, whether set via environment variables or CLI arguments, are included when running this command.
- Verify that the correct Red Hat build of Keycloak version is used.

Failure to meet these requirements results in an incorrect outcome.

The command prints the result to the console. For example, if a rolling update is possible, it displays:

Rolling Update possible message

```
[OK] Rolling Update is available.
```

If no rolling update is possible, the command provides details about the incompatibility:

Rolling Update not possible message

```
[keycloak] Rolling Update is not available. 'keycloak.version' is incompatible: 26.2.0 -> 26.2.1 (1)
```

(1) In this example, the Keycloak version **26.2.0** is not compatible with version **26.2.1** and a rolling update is not possible.

Command exit code

Use the command’s exit code to determine the update type in your automation pipeline:

Exit Code	Description
0	Rolling Update is possible.
1	Unexpected error occurred (such as the metadata file is missing or corrupted).
2	Invalid CLI option.

Exit Code	Description
3	Rolling Update is not possible. The deployment must be shut down before applying the new configuration.
4	Rolling Update is not possible. The feature **rolling-updates** is disabled.

23.3. FURTHER READING

The Red Hat build of Keycloak Operator uses the functionality described above to determine if a rolling update is possible. See the Avoiding downtime with rolling updates chapter and the **Auto** strategy for more information.