Red Hat build of Keycloak 24.0

Migration Guide

Last Updated: 2025-03-31

Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

This guide consists of the migration guide for Red Hat build of Keycloak.

Table of Contents

CHAPTER 1. MIGRATING RED HAT SINGLE SIGN-ON 7.6 TO RED HAT BUILD OF KEYCLOAK

The purpose of this guide is to document the steps that are required to successfully migrate Red Hat Single Sign-On 7.6 to Red Hat build of Keycloak 24.0. The instructions address migration of the following elements:

- Red Hat Single Sign-On 7.6 server
- Operator deployments on OpenShift
- Template deployments on OpenShift
- Applications secured by Red Hat Single Sign-On 7.6
- Custom providers
- Custom themes

This guide also includes guidelines for migrating upstream Keycloak to Red Hat build of Keycloak 24.0. Before you start the migration, you might consider installing a new instance of Red Hat build of Keycloak to become familiar with the changes for this release. See the Red Hat build of Keycloak Getting Started Guide.

CHAPTER 2. MIGRATING A RED HAT SINGLE SIGN-ON 7.6 SERVER

This section provides instructions for migrating a standalone server deployed from the ZIP distribution. Red Hat build of Keycloak 24.0 is built with Quarkus, which replaces the Red Hat JBoss Enterprise Application Platform (JBoss EAP) that was used by Red Hat Single Sign-On 7.6.

The main changes to the server are the following:

- A revamped configuration experience, which is streamlined and supports great flexibility in setting configuration options.
- An RPM distribution of the server is no longer available

2.1. PREREQUISITES

- The previous instance of Red Hat Single Sign-On 7.6 was shut down so that it does not use the same database instance that will be used by Red Hat build of Keycloak .
- You backed up the database.
- OpenJDK17 is installed.
- You reviewed the Release Notes.

2.2. MIGRATION PROCESS OVERVIEW

The following sections provide instructions for these migration steps:

1. Download Red Hat build of Keycloak .
2. Migrate the configuration.
3. Migrate the database.
4. Start the Red Hat build of Keycloak server.

2.3. DOWNLOADING RED HAT BUILD OF KEYCLOAK

The Red Hat build of Keycloak server download ZIP file contains the scripts and binaries to run the Red Hat build of Keycloak server.

1. Download the Red Hat build of Keycloak server distribution file from the Red Hat customer portal.
2. Unpack the ZIP file using the unzip command.

2.4. MIGRATING THE CONFIGURATION

A new unified way to configure the Red Hat build of Keycloak server is through configuration options. The Red Hat Single Sign-On 7.6 configuration mechanism, such as standalone.xml, jboss-cli, and so on, no longer applies.

Each option can be defined through the following configuration sources:

- CLI parameters
- Environment variables
- Configuration file
- Java KeyStore file

If the same configuration option is specified through different configuration sources, the first source in the list above is used.

All configuration options are available in all the different configuration sources, where the main difference is the format of the key. For example, here are four ways to configure the database hostname:

Source	Format
CLI parameters	--db-url-host cliValue
Environment variables	KC_DB_URL_HOST=envVarValue
Configuration file	db-url-host=confFileValue
Java KeyStore file	kc.db-url-host=keystoreValue

The **kc.sh --help** command as well as the Red Hat build of Keycloak documentation provides a complete list of all available configuration options, where options are grouped by categories such as cache, database, and so on. Also, separate chapters exist for each area to configure, such as the chapter for Configuring the database.

Additional resources

- Configuring Keycloak

2.4.1. Migrating the database configuration

In contrast to Red Hat Single Sign-On 7.6, Red Hat build of Keycloak has built-in support for the supported databases removing the need to manually install and configure the database drivers. The exception is Oracle and Microsoft SQL Server, which still require manual installation of the drivers.

In terms of configuration, consider the datasource subsystem from your existing Red Hat Single Sign-On 7.6 installation and map those configurations to the options available from the Database configuration category in Red Hat build of Keycloak . For example, a previous configuration appears as follows:

```
<datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-name="KeycloakDS"
enabled="true" use-java-context="true" statistics-enabled="true">
    <connection-url>jdbc:postgresql://mypostgres:5432/mydb?
currentSchema=myschema</connection-url>
    <driver>postgresql</driver>
    <pool>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>50</max-pool-size>
```

```
    </pool>
    <security>
        <user-name>myuser</user-name>
        <password>myuser</password>
    </security>
</datasource>
```

In Red Hat build of Keycloak , the equivalent configuration using CLI parameters would be:

```
kc.sh start
  --db postgres
  --db-url-host mypostgres
  --db-url-port 5432
  --db-url-database mydb
  --db-schema myschema
  --db-pool-min-size 5 --db-pool-max-size 50
  --db-username myser --db-password myuser
```

NOTE

Consider storing database credentials in a secure KeyStore configuration source.

Additional resources

- Configuring the database, which also includes instructions for installing the Oracle and Microsoft SQL Server JDBC drivers
- Setting sensitive options using a Java KeyStore file , which provides instructions for how to securely store database credentials.

2.4.2. Migrating HTTP and TLS configuration

HTTP is disabled and TLS configuration is required by default, whenever the production mode (represented by the **start** option) is used.

You can enable HTTP with the **--http-enabled=true** configuration option, but it is not recommended unless the Red Hat build of Keycloak server is within a fully isolated network, and no risk exists of internal or external attackers being able to observe networking traffic.

A Red Hat build of Keycloak instance has a different context root (URL path) as it uses the root of the server while Red Hat Single Sign-On 7.6 by default appends /**auth**. To mimic the old behavior, the **--http-relative-path=/auth** configuration option can be used. The default ports remain the same, but they can also be changed by the **--http-port** and **--https-port** options.

Two ways exist to configure TLS, either through files in the PEM format or with a Java Keystore. For example, a previous configuration by Java Keystore appears as follows:

```
<tls>
 <key-stores>
          <key-store name="applicationKS">
              <credential-reference
  clear-text="password"/>
  <implementation type="JKS"/>
              <file
```

```
    path="/path/to/application.keystore”/>
            </key-store>
      </key-stores>
      <key-managers>
        <key-manager name="applicationKM"
         key-store="applicationKS">
              <credential-reference
          clear-text="password"/>
        </key-manager>
      </key-managers>
      <server-ssl-contexts>
         <server-ssl-context name="applicationSSC"
      key-manager="applicationKM"/>
      </server-ssl-contexts>
</tls>
```

In Red Hat build of Keycloak , the equivalent configuration using CLI parameters would be as follows:

```
kc.sh start
  --https-key-store-file /path/to/application.keystore
  --https-key-store-password password
```

In Red Hat build of Keycloak , you can configure TLS by providing certificates in PEM format as follows:

```
kc.sh start
  --https-certificate-file /path/to/certfile.pem
  --https-certificate-key-file /path/to/keyfile.pem
```

Additional resources

- Configuring TLS

2.4.3. Migrating clustering and cache configuration

Red Hat Single Sign-On 7.6 provided distinct operating modes for running the server as standalone, standalone clustered, and domain clustered. These modes differed in the start script and configuration files. Red Hat build of Keycloak offers a simplified solution with a single start script: **kc.sh**.

To run the server as standalone or standalone clustered, use the **kc.sh** script:

Red Hat build of Keycloak	Red Hat Single Sign-On 7.6
./kc.sh start --cache=local	./standalone.sh
./kc.sh start [--cache=ispn]	./standalone.sh --server-config=standalone-ha.xml

The default values for the **--cache** parameter is start mode aware:

- **local** - when the start-dev command is executed
- **ispn** - when the start command is executed

In Red Hat Single Sign-On 7.6, clustering and cache configuration was done through the Infinispan subsystem, while in Red Hat build of Keycloak the majority of the configuration is done through a separate Infinispan configuration file. For example, a previous configuration of Infinispan appears as follows:

```
<subsystem xmlns="urn:jboss:domain:infinispan:13.0">
<cache-container name="keycloak" marshaller="JBOSS" modules="org.keycloak.keycloak-model-infinispan">
        <local-cache name="realms">
            <heap-memory size="10000"/>
        </local-cache>
        <local-cache name="users">
            <heap-memory size="10000"/>
        </local-cache>
        <local-cache name="sessions"/>
        <local-cache name="authenticationSessions"/>
        <local-cache name="offlineSessions"/>
    ...
</cache-container>
</subsystem>
```

In Red Hat build of Keycloak , the default Infinispan configuration file is located in the **conf/cache-ispn.xml** file. You can provide your own Infinispan configuration file and specify it using the CLI parameter as follows:

```
kc.sh start --cache-config-file my-cache-file.xml
```

NOTE

Domain clustered mode is not supported with Red Hat build of Keycloak .

Transport stacks

No migration is needed for default and custom JGroups transport stacks in the Red Hat build of Keycloak.

The only improvement is the possibility to override the stack defined in the cache configuration file by providing a CLI option cache-stack, which takes precedence. Consider a part of the Infinispan configuration file **my-cache-file.xml**, specified above, with the custom JGroups transport stack as follows:

You can notice the transport stack for the keycloak cache container is set to tcp, but it can be overridden using the CLI option as follows:

```
<jgroups>
 <stack name=”my-encrypt-udp” extends=”udp”>
  …
 </stack>
</jgroups>

<cache-container name=”keycloak”>
 <transport stack=”tcp”/>
 …
</cache-container>
```

```
kc.sh start
  --cache-config-file my-cache-file.xml
  --cache-stack my-encrypt-udp
```

After executing the above commands, the **my-encrypt-udp** transport stack is used.

Additional resources

- Configuring distributed caches

2.4.4. Migrating hostname and proxy configuration

In Red Hat build of Keycloak, you are now obligated to configure the Hostname SPI in order to set how front and back end URLs are going to be created by the server when redirecting users or communicating with their clients.

For example, consider if you have a configuration similar as the follows in your Red Hat Single Sign-On 7.6 installation:

```
<spi name="hostname">
    <default-provider>default</default-provider>
    <provider name="default" enabled="true">
        <properties>
          <property name="frontendUrl" value="myFrontendUrl"/>
          <property name="forceBackendUrlToFrontendUrl" value="true"/>
        </properties>
    </provider>
</spi>
```

You can translate it to the following configuration options in Red Hat build of Keycloak:

```
kc.sh start
  --hostname-url myFrontendUrl
  --hostname-strict-backchannel true
```

The **hostname-url** configuration option allows you to set the base URL where the cluster is exposed to the public from an ingress layer running in front of your cluster. You can also set the URL for administration resources by setting the **hostname-admin-url** configuration option.

Red Hat build of Keycloak allows you to configure which reverse proxy headers should be reflected. You can use either the **Forwarded** header or the set of **X-Forwarded-*** headers. For example:

```
kc.sh start --proxy-headers xforwarded
```

NOTE

The hostname and proxy configuration are used for determining the resource URLs (redirect URIs, CSS and JavaScript links, OIDC well-known endpoints, and so on) and not for actively blocking incoming requests. None of the hostname/proxy options change the actual binding addresses or ports that the Red Hat build of Keycloak server listens on - this is a responsibility of the HTTP/TLS options. In Red Hat Single Sign-On 7.6, setting a hostname was highly recommended, but not enforced. In Red Hat build of Keycloak , when the start command is used this is now required, unless explicitly disabled with the **--hostname-strict=false** option.

Additional resources

- Using a reverse proxy
- Configuring the hostname

2.4.5. Migrating truststore configuration

The truststore is used for external TLS communication, for example HTTPS requests and LDAP servers. To use a truststore, you import the remote server's or CA's certificate into the trustore. Then, you can start the Red Hat build of Keycloak server specifying the system truststore.

For example, a previous configuration appears as follows:

```
<spi name="truststore">
    <provider name="file" enabled="true">
        <properties>
            <property name="file" value="path/to/myTrustStore.jks"/>
            <property name="password" value="password"/>
            <property name="hostname-verification-policy" value="WILDCARD"/>
        </properties>
    </provider>
</spi>
```

Red Hat build of Keycloak supports truststores in the following formats: PEM files, or PKCS12 files with the extensions .p12 and .pfx. For the PKCS12 files, the certs must be unencrypted, which means that no password is expected. The JKS truststores need to be converted. Since the Java **keytool** enforces a minimum password length of 6 characters, **openssl** can be used as an alternative to create an unencrypted PKCS12 truststore.

1. Create a PKCS12 truststore with a temporary password using keytool by importing the contents of an old JKS truststore.

```
keytool -importkeystore -srckeystore path/to/myTrustStore.jks \
  -destkeystore path/to/myTrustStore.p12 \
  -srcstoretype jks \
  -deststoretype pkcs12 \
  -srcstorepass password \
  -deststorepass temp-password
```

2. Export the contents of the new PKCS12 truststore using **openssl**.

```
openssl pkcs12 -in path/to/myTrustStore.p12 \
  -out path/to/myTrustStore.pem \
  -nodes -passin pass:temp-password
```

3. Reimport the contents into a new unencrypted PKCS12 truststore using **openssl**.

```
openssl pkcs12 -export -in path/to/myTrustStore.pem \
  -out path/to/myUnencryptedTrustStore.p12 \
  -nokeys -passout pass:
```

In this example, the resulting CLI parameter would be as follows:

```
kc.sh start
  --truststore-paths path/to/myUnencryptedTrustStore.p12
  --tls-hostname-verifier WILDCARD
```

Additional resources

- Configuring trusted certificates for outgoing requests

2.4.6. Migrating vault configuration

The Keystore Vault is an implementation of the Vault SPI and it is useful for storing secrets in bare metal installations. This vault is a replacement of the Elytron Credential Store in Red Hat Single Sign-On 7.6..

```
<spi name="vault">
    <provider name="elytron-cs-keystore" enabled="true">
      <properties>
          <property name="location" value="path/to/keystore.p12"/>
           <property name="secret" value="password"/>
    </properties>
  </provider>
</spi>
```

In Red Hat build of Keycloak, the equivalent configuration using CLI parameters would be:

```
kc.sh start
  --vault keystore
  --vault-file /path/to/keystore.p12
  --vault-pass password
```

Secrets stored in the vault can be then accessed at multiple places within the Admin Console. When it comes to the migration from the existing Elytron vault to the new Java KeyStore-based vault, no realm configuration changes are required. If a newly created Java keystore contains the same secrets, your existing realm configuration should work.

Given that you use the default **REALM_UNDERSCORE_KEY** key resolver, the secret can be accessed by **${vault.realm-name_alias}** (for example, in your LDAP User federation configuration) the same way as before.

Additional resources

- Using a vault.

2.4.7. Migrating JVM settings

The approach for JVM settings in Red Hat build of Keycloak is similar to the Red Hat Single Sign-On 7.6 approach. You still need to set particular environment variables, however, the **/bin** folder contains no configuration files, such as **standalone.conf**.

Red Hat build of Keycloak provides various default JVM arguments, which proved to be suitable for the majority of deployments as it provides good throughput and efficiency in memory allocation and CPU overhead. Also, other default JVM arguments ensure a smooth run of the Red Hat build of Keycloak instance, so use caution when you change the arguments for your use case.

To change JVM arguments or GC settings, you set particular environment variables, which are specified as Java options. For a complete override of these settings, you specify the **JAVA_OPTS** environment variable.

When only an append of a particular Java property is required, you specify the **JAVA_OPTS_APPEND** environment variable. When no **JAVA_OPTS** environment variable is specified, the default Java properties are used and can be found inside the **./kc.sh** script.

For instance, you can specify a particular Java option as follows:

```
export JAVA_OPTS_APPEND=-XX:+HeapDumpOnOutOfMemoryError
kc.sh start
```

2.4.8. Migrating SPI provider configuration

Configuration for SPI providers is available through the new configuration system. This is the old format:

```
<spi name="<spi-id>">
    <provider name="<provider-id>" enabled="true">
        <properties>
            <property name="<property>" value="<value>"/>
        </properties>
    </provider>
</spi>
```

This is the new format:

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

Source	Format
CLI	./kc.sh start --spi-connections-http-client-default-connection-pool-size 10
Environment Variable	KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_POOL_SIZE=10
Configuration file	spi-connections-http-client-default-connection-pool-size=10

Source	Format
Java Keystore file	kc.spi-connections-http-client-default-connection-pool-size=10

Additional resources

- All Provider Config.

2.4.9. Troubleshooting the configuration

Use these commands for troubleshooting:

- **kc.sh show-config** - shows you the configuration sources from which particular properties are loaded and what their values are. You can check whether a property and its value is propagated correctly.
- **kc.sh --verbose start** - prints out the whole error stack trace, when there is an error.

2.5. MIGRATING THE DATABASE

Red Hat build of Keycloak can automatically migrate the database schema, or you can choose to do it manually. By default the database is automatically migrated when you start the new installation for the first time.

2.5.1. Automatic relational database migration

To perform an automatic migration, start the server connected to the desired database. If the database schema has changed for the new version of the server, it will be migrated.

2.5.2. Manual relational database migration

To enable manual upgrading of the database schema, set the **migration-strategy** property value to *manual* for the default connections-jpa provider:

```
kc.sh start --spi-connections-jpa-legacy-migration-strategy manual
```

When you start the server with this configuration, it checks if the database needs to be migrated. The required changes are written to the **bin/keycloak-database-update.sql** SQL file that you can review and manually run against the database.

To change the path and name of the exported SQL file, set the **migration-export** property for the default **connections-jpa** provider:

```
kc.sh start
  --spi-connections-jpa-legacy-migration-export <path>/<file.sql>
```

For further details on how to apply this file to the database, see the documentation for your relational database. After the changes have been written to the file, the server exits.

2.6. STARTING THE RED HAT BUILD OF KEYCLOAK SERVER

The difference in starting the distribution of Red Hat Single Sign-On 7.6 and Red Hat build of Keycloak is in the executed script. These scripts live in the /**bin** folder of the server distribution.

2.6.1. Starting the server in development mode

To try out Red Hat build of Keycloak without worrying about supplying any properties, you can start the distribution in the development mode as described in the table below. However, note that this mode is strictly for development and should not be used in production.

Red Hat build of Keycloak	Red Hat Single Sign-On 7.6
./kc.sh start-dev	./standalone.sh

WARNING

The development mode should NOT be used in production.

2.6.2. Starting the server in production mode

Red Hat build of Keycloak has a dedicated start mode for production: **./kc.sh start**. The difference from running with **start-dev** is different default configuration values. It automatically uses a strict and by-default secured configuration setup. In the production mode, HTTP is disabled, and explicit TLS and hostname configuration is required.

Additional resources

- Configuring Keycloak for production
- Optimize the Keycloak startup

CHAPTER 3. MIGRATING OPERATOR DEPLOYMENTS ON OPENSHIFT

To adapt to the revamped server configuration, the Red Hat build of Keycloak Operator was completely recreated. The Operator provides full integration with Red Hat build of Keycloak, but it is not backward compatible with the Red Hat Single Sign-On 7.6 Operator. Using the new Operator requires creating a new Red Hat build of Keycloak deployment. For full details, see the Operator Guide.

3.1. PREREQUISITES

- The previous instance of Red Hat Single Sign-On 7.6 was shut down so that it does not use the same database instance that will be used by Red Hat build of Keycloak .
- In case the unsupported embedded database (that is managed by the Red Hat Single Sign-On 7.6 Operator)) was used, it has been converted to an external database that is provisioned by the user.
- Database backup was created.
- You reviewed the Release Notes.

3.2. MIGRATION PROCESS

1. Install Red Hat build of Keycloak Operator to the namespace.
2. Create new CRs and related Secrets. Manually migrate your Red Hat Single Sign-On 7.6 configuration to your new Keycloak CR.
3. If custom providers were used, migrate them and create a custom Red Hat build of Keycloak container image to include them.
4. If custom themes were used, migrate them and create a custom Red Hat build of Keycloak container image to include them.

3.3. MIGRATING KEYCLOAK CR

Keycloak CR now supports all server configuration options. All relevant options are available as first class citizen fields directly under the spec of the CR. All options in the CR follow the same naming conventions as the server options making the experience between bare metal and Operator deployments seamless.

Additionally, you can define any options that are missing from the CR in the **additionalOptions** field such as SPI providers configuration. Another option is to use **podTemplate**, a Technology Preview field, to modify the raw Kubernetes deployment pod template in case a supported alternative does not exist as a first class citizen field in the CR.

The following shows an example Keycloak CR to deploy Red Hat build of Keycloak through the Operator:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
```

```
db:
  vendor: postgres
  host: postgres-db
  usernameSecret:
    name: keycloak-db-secret
    key: username
  passwordSecret:
    name: keycloak-db-secret
    key: password
http:
  tlsSecret: example-tls-secret
hostname:
  hostname: test.keycloak.org
additionalOptions:
  - name: spi-connections-http-client-default-connection-pool-size
    value: 20
```

Notice the resemblance with CLI configuration:

```
./kc.sh start --db=postgres --db-url-host=postgres-db --db-username=user --db-password=pass --
https-certificate-file=mycertfile --https-certificate-key-file=myprivatekey --hostname=test.keycloak.org
--spi-connections-http-client-default-connection-pool-size=20
```

Additional resources

- Basic Keycloak deployment
- Advanced configuration

3.3.1. Migrating database configuration

Red Hat build of Keycloak can use the same database instance as was previously used by Red Hat Single Sign-On 7.6. The database schema will be migrated automatically the first time Red Hat build of Keycloak connects to it.

WARNING

Migrating the embedded database managed by Red Hat Single Sign-On 7.6 Operator is not supported.

In the Red Hat Single Sign-On 7.6 Operator, the external database connection was configured using a Secret, for example:

```
apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-secret
  namespace: keycloak
  labels:
```

```
    app: sso
stringData:
  POSTGRES_DATABASE:  kc-db-name
  POSTGRES_EXTERNAL_ADDRESS: my-postgres-hostname
  POSTGRES_EXTERNAL_PORT: 5432
  POSTGRES_USERNAME: user
  POSTGRES_PASSWORD: pass
type: Opaque
```

In Red Hat build of Keycloak , the database is configured directly in the Keycloak CR with credentials referenced as Secrets, for example:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  db:
    vendor: postgres
    host: my-postgres-hostname
    port: 5432
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  ...

apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-secret
stringData:
  username: "user"
  password: "pass"
type: Opaque
```

3.3.1.1. Supported database vendors

Red Hat Single Sign-On 7.6 Operator supported only PostgreSQL databases, but the Red Hat build of Keycloak Operator supports all database vendors that are supported by the server.

3.3.2. Migrating TLS configuration

Red Hat Single Sign-On 7.6 Operator by default configured the server to use the TLS Secret generated by OpenShift CA. Red Hat build of Keycloak Operator does not make any assumptions around TLS to meet production best practices and requires users to provide their own TLS certificate and key pair, for example:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
```

```
spec:
  http:
    tlsSecret: example-tls-secret
  ...
```

The expected format of the secret referred to in tlsSecret should use the standard Kubernetes TLS Secret (**kubernetes.io/tls**) type.

The Red Hat Single Sign-On 7.6 Operator used the reencrypt TLS termination strategy by default on Route. Red Hat build of Keycloak Operator uses the passthrough strategy by default. Additionally, the Red Hat Single Sign-On 7.6 Operator supported configuring TLS termination. Red Hat build of Keycloak Operator does not support TLS termination in the current release.

If the default Operator-managed Route does not meet desired TLS configuration, a custom Route needs to be created by the user and the default one disabled as:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ingress:
    enabled: false
  ...
```

3.3.3. Using a custom image for extensions

To reflect best practices and support immutable containers, the Red Hat build of Keycloak Operator no longer supports specifying extensions in the Keycloak CR. In order to deploy an extension, an optimized custom image must be built. Keycloak CR now includes a dedicated field for specifying Red Hat build of Keycloak images, for example:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  image: quay.io/my-company/my-keycloak:latest
  ...
```

NOTE

When specifying a custom image, the Operator assumes it is already optimized and does not perform the costly optimization at each server start.

Additional resources

- Using custom Keycloak images in the Operator
- Creating a customized and optimized container image

3.3.4. Upgrade strategy option removed

The Red Hat Single Sign-On 7.6 Operator supported recreate and rolling strategies when performing a

server upgrade. This approach was not practical. It was up to the user to choose if the Red Hat Single Sign-On 7.6 Operator should scale down the deployment before performing an upgrade and database migration. It was not clear to the users when the rolling strategy could be safely used.

Therefore, this option was removed in the Red Hat build of Keycloak Operator and it always implicitly performs the recreate strategy, which scales down the whole deployment before creating Pods with the new server container image to ensure only a single server version accesses the database.

3.3.5. Health endpoint exposed by default

The Red Hat build of Keycloak configures the server to expose a simple health endpoint by default that is used by OpenShift probes. The endpoint does not expose any security sensitive data about deployment but it is accessible without any authentication. As an alternative, *<your-server-context-root>*/**health** endpoint can be blocked on a custom Route.

For example,

1. Create Keycloak configured for TLS edge termination.
   Make sure to omit the **tlsSecret** field:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  proxy:
   Headers:xforwarded
  hostname:
    hostname: example.com
  ...
```

2. Create a blocking Route to prohibit access to the health endpoint:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: example-kc-block-health
  annotations:
    haproxy.router.openshift.io/rewrite-target: /404
spec:
  host: example.com
  path: /health
  to:
    kind: Service
    name: example-kc-service
  port:
    targetPort: http
  tls:
    termination: edge
```

NOTE

Path-based Routes require TLS termination to be configured for either edge or reencrypt. By default, the Operator uses passthrough.

3.3.6. Migrating advanced deployment options using Pod templates

The Red Hat Single Sign-On 7.6 Operator exposed multiple low-level fields for deployment configuration, such as volumes. Red Hat build of Keycloak Operator is more opinionated and does not expose most of these fields. However, it is still possible to configure any desired deployment fields specified as the **podTemplate**, for example:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  unsupported:
    podTemplate:
      metadata:
        labels:
          foo: "bar"
      spec:
        containers:
          - volumeMounts:
              - name: test-volume
                mountPath: /mnt/test
        volumes:
          - name: test-volume
            secret:
              secretName: test-secret
  ...
```

NOTE

The **spec.unsupported.podTemplate** field offers only limited support as it exposes low-level configuration where full functionality has not been tested under all conditions. Whenever possible, use the fully supported first class citizen fields in the top level of the CR spec.

For example, instead of **spec.unsupported.podTemplate.spec.imagePullSecrets**, use directly **spec.imagePullSecrets**.

3.3.7. Connecting to an external instance is no longer supported

The Red Hat Single Sign-On 7.6 Operator supported connecting to an external instance of Red Hat Single Sign-On 7.6. For example, creating clients within an existing realm through Client CRs is no longer supported in the Red Hat build of Keycloak Operator.

3.3.8. Migrating Horizontal Pod Autoscaler enabled deployments

To use a Horizontal Pod Autoscaler (HPA) with Red Hat Single Sign-On 7.6, it was necessary to set the **disableReplicasSyncing: true** field in the Keycloak CR and scale the server StatefulSet. This is no longer necessary as the Keycloak CR in Red Hat build of Keycloak Operator can be scaled directly by an HPA.

3.4. MIGRATING THE KEYCLOAK REALM CR

The Realm CR was replaced by the Realm Import CR, which offers similar functionality and has a similar schema. The Realm Import CR offers only Realm bootstrapping and as such no longer supports Realm deletion. It also does not support updates, similarly to the previous Realm CR.

Full Realm representation is now included in the Realm Import CR, in comparison to the previous Realm CR that offered only a few selected fields.

Example of Red Hat Single Sign-On 7.6 Realm CR:

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  name: example-keycloakrealm
spec:
  instanceSelector:
    matchLabels:
      app: sso
  realm:
    id: "basic"
    realm: "basic"
    enabled: True
    displayName: "Basic Realm"
```

Example of corresponding Red Hat build of Keycloak Realm Import CR:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: KeycloakRealmImport
metadata:
  name: example-keycloakrealm
spec:
  keycloakCRName: example-kc
  realm:
    id: "basic"
    realm: "basic"
    enabled: True
    displayName: "Basic Realm"
```

Additional resources

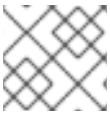
- Realm Import

3.5. REMOVED CRS

The Client and User CRs were removed from Red Hat build of Keycloak Operator. The lack of these CRs can be partially mitigated by the new Realm Import CR. Adding support for Client CRs is on the road-map for a future Red Hat build of Keycloak release, while User CRs are not currently a planned feature.

CHAPTER 4. MIGRATING TEMPLATES DEPLOYMENTS ON OPENSHIFT

OpenShift templates were deprecated and removed from the Red Hat build of Keycloak container images. Using the Operator is the recommended alternative for deploying Red Hat build of Keycloak on OpenShift.



NOTE

OpenShift 3.x is no longer supported.

You will generally need to create a Keycloak CR (of the Red Hat build of Keycloak Operator) that references an externally managed database. The PostgreSQL database with this template is managed by a DeploymentConfig. You initially retain the **application_name-postgresql** DeploymentConfig that was created by the template. The PostgreSQL database instance created by the DeploymentConfig will be usable by the Red Hat build of Keycloak Operator.

This guide does not include directions for migrating from this instance to a self-managed database, either by an operator or your cloud provider.

The Red Hat build of Keycloak Operator does not manage a database and it is required to have a database provisioned and managed separately.

4.1. MIGRATING DEPLOYMENTS WITH THE INTERNAL H2 DATABASE

The following are the affected templates:

- sso76-ocp3-https
- sso76-ocp4-https
- sso76-ocp3-x509-https
- sso76-ocp4-x509-https

These templates rely upon the devel database and are not supported for production use.

4.2. MIGRATING DEPLOYMENTS WITH EPHEMERAL POSTGRESQL DATABASE

The following are the affected templates:

- sso76-ocp3-postgresql
- sso76-ocp4-postgresql

This template creates a PostgreSQL database without persistent storage, which is only recommended for development purposes.

4.3. MIGRATING DEPLOYMENTS WITH PERSISTENT POSTGRESQL DATABASE

The following are the affected templates:

- sso76-ocp3-postgresql-persistent
- sso76-ocp4-postgresql-persistent
- sso76-ocp3-x509-postgresql-persistent
- sso76-ocp4-x509-postgresql-persistent

4.3.1. Prerequisites

- The previous instance of Red Hat Single Sign-On 7.6 was shut down so that it does not use the same database instance that will be used by Red Hat build of Keycloak .
- Database backup was created.
- You reviewed the Release Notes.

4.4. MIGRATION PROCESS

1. Install Red Hat build of Keycloak Operator to the namespace.
2. Create new CRs and related Secrets.
   Manually migrate your template based Red Hat Single Sign-On 7.6 configuration to your new Red Hat build of Keycloak CR. See the following examples for suggested mappings between Template parameters and Keycloak CR fields.

The following examples compare a Red Hat build of Keycloak Operator CR to the DeploymentConfig that was previously created by a Red Hat Single Sign-On 7.6 Template.

Operator CR for Red Hat build of Keycloak

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: rhbk
spec:
  instances: 1
  db:
    vendor: postgres
    host: postgres-db
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  http:
    tlsSecret: sso-x509-https-secret
```

DeploymentConfig for Red Hat Single Sign-On 7.6

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: rhsso
```

```
spec:
  replicas: 1
  template:
    spec:
      volumes:
        - name: sso-x509-https-volume
          secret:
            secretName: sso-x509-https-secret
            defaultMode: 420
      containers:
          volumeMounts:
            - name: sso-x509-https-volume
              readOnly: true
          env:
            - name: DB_SERVICE_PREFIX_MAPPING
              value: postgres-db=DB
            - name: DB_USERNAME
              value: username
            - name: DB_PASSWORD
              value: password
```

The following tables refer to fields of Keycloak CR by a JSON path notation. For example, **.spec** refers to the **spec** field. Note that **spec.unsupported** is a Technology Preview field. It is more an indication that eventually that functionality will be achievable by other CR fields. Parameters marked in **bold** are supported by both the passthrough and reencrypt templates.

4.4.1. General Parameter Migration

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
APPLICATION_NAME	.metadata.name
IMAGE_STREAM_NAMESPACE	N/A - the image is controlled by the operator or you main use spec.image to specify a custom image
SSO_ADMIN_USERNAME	No direct setting, defaults to admin
SSO_ADMIN_PASSWORD	N/A - created by the operator during the initial reconciliation
MEMORY_LIMIT	**.spec.unsupported.podTemplate.spec.containers[0].resources.limits['memory']**
SSO_SERVICE_PASSWORD, SSO_SERVICE_USERNAME	No longer used.
SSO_TRUSTSTORE, SSO_TRUSTSTORE_PASSWORD, SSO_TRUSTSTORE_SECRET	**.spec.truststores** Notice that truststores must not be password protected.

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
SSO_REALM	Not needed if you are reusing the existing database. An alternative is the RealmImport CR.

4.4.2. Database Deployment Parameter Migration

POSTGRESQL_IMAGE_STREAM_TAG, **POSTGRESQL_MAX_CONNECTIONS**, **VOLUME_CAPACITY** and **POSTGRESQL_SHARED_BUFFERS** will need to be migrated to whatever replacement you have chosen creating the database deployment.

4.4.3. Database Connection Parameter Migration

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
DB_VENDOR	**.spec.db.vendor** - will need to be set to PostgreSQL if PostgreSQL is still being used
DB_DATABASE	**.spec.db.database**
DB_MIN_POOL_SIZE	**.spec.db.poolMinSize**
DB_MAX_POOL_SIZE	**.spec.db.maxPoolSize**
DB_TX_ISOLATION	may be set by the **spec.db.url** if it is supported by the driver or as a general setting on the target database
DB_USERNAME	**.spec.db.usernameSecret**
DB_PASSWORD	**.spec.db.passwordSecret**
DB_JNDI	No longer applicable

4.4.4. Networking Parameter Migration

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
HOSTNAME_HTTP	**.spec.hostname.hostname** - with **.spec.http.httpEnabled=true.** Since the Red Hat build of Keycloak operator will only create a single Ingress/Route, for this to create an http route **.spec.http.tlsSecret** needs to be left unspecified
HOSTNAME_HTTPS	**.spec.hostname.hostname** - with **.spec.http.tlsSecret** specified.

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
SSO_HOSTNAME	**.spec.hostname.hostname**
HTTPS_SECRET	**.spec.http.tlsSecret** - see the other HTTPS parameters below
HTTPS_KEYSTORE HTTPS_KEYSTORE_TYPE HTTPS_NAME HTTPS_PASSWORD	No longer applicable. The secret referenced by **.spec.http.tlsSecret** should be of type **kubernetes.io/tls** with **tls.crt** and **tls.key** entries
X509_CA_BUNDLE	**.spec.truststores**

Note that the Red Hat build of Keycloak Operator does not currently support a way to configure the TLS termination. By default, the passthrough strategy is used. Therefore, the proxy option is not yet exposed as a first-class citizen option field, because it does not matter whether the passthrough or reencrypt strategy is used. However, if you need this option, you can replace the default Ingress Operator certificate and manually configure a Route in order to trust Red Hat build of Keycloak's certificate.

The default behavior of the Red Hat build of Keycloak Operator can be then overridden by:

```
additionalOptions:
  name: proxy
  value: reencrypt
```

4.4.5. JGroups Parameter Migration

JGROUPS_ENCRYPT_SECRET, JGROUPS_ENCRYPT_KEYSTORE, JGROUPS_ENCRYPT_NAME, JGROUPS_ENCRYPT_PASSWORD, and **JGROUPS_CLUSTER_PASSWORD** have no first-class representation in the Keycloak CR. Securing cache communication is still possible using the cache configuration file.

Additional resources

- Configuring distributed cache

CHAPTER 5. MIGRATING APPLICATIONS SECURED BY RED HAT SINGLE SIGN-ON 7.6

Red Hat build of Keycloak introduces key changes to how applications are using some of the Red Hat Single Sign-On 7.6 Client Adapters.

In addition to no longer releasing some client adapters, Red Hat build of Keycloak also introduces fixes and improvements that impact how client applications use OpenID Connect and SAML protocols.

In this chapter, you will find the instructions to address these changes and migrate your application to integrate with Red Hat build of Keycloak .

5.1. MIGRATING OPENID CONNECT CLIENTS

The following Java Client OpenID Connect Adapters are no longer released starting with this release of Red Hat build of Keycloak

- Red Hat JBoss Enterprise Application Platform 6.x
- Red Hat JBoss Enterprise Application Platform 7.x
- Spring Boot
- Red Hat Fuse

Compared to when these adapters were first released, OpenID Connect is now widely available across the Java Ecosystem. Also, much better interoperability and support is achieved by using the capabilities available from the technology stack, such as your application server or framework.

These adapters have reached their end of life and are only available from Red Hat Single Sign-On 7.6. It is highly recommended to look for alternatives to keep your applications updated with the latest updates from OAuth2 and OpenID connect protocols.

5.1.1. Key changes in OpenID Connect protocol and client settings

5.1.1.1. Access Type client option no longer available

When you create or update an OpenID Connect client, **Access Type** is no longer available. However, you can use other methods to achieve this capability.

- To achieve the **Bearer Only** capability, create a client with no authentication flow. In the **Capability config** section of the client details, make sure that no flow is selected. The client cannot obtain any tokens from Keycloak, which is equivalent to using the **Bearer Only** access type.
- To achieve the **Public** capability, make sure that client authentication is disabled for this client and at least one flow is enabled.
- To achieve **Confidential** capability, make sure that **Client Authentication** is enabled for the client and at least one flow is enabled.

The boolean flags **bearerOnly** and **publicClient** still exist on the client JSON object. They can be used when creating or updating a client by the admin REST API or when importing this client by partial import or realm import. However, these options are not directly available in the Admin Console v2.

5.1.1.2. Changes in validating schemes for valid redirect URIs

If an application client is using non http(s) custom schemes, the validation now requires that a valid redirect pattern explicitly allows that scheme. Example patterns for allowing custom scheme are custom:/test, custom:/test/* or custom:. For security reasons, a general pattern such as * no longer covers them.

5.1.1.3. Support for the client_id parameter in OpenID Connect Logout Endpoint

Support for the **client_id** parameter, which is based on the OIDC RP-Initiated Logout 1.0 specification. This capability is useful to detect what client should be used for Post Logout Redirect URI verification in case that **id_token_hint** parameter cannot be used. The logout confirmation screen still needs to be displayed to the user when only the **client_id** parameter is used without parameter **id_token_hint**, so clients are encouraged to use **id_token_hint** parameter if they do not want the logout confirmation screen to be displayed to the user.

5.1.2. Valid Post Logout Redirect URIs

The **Valid Post Logout Redirect URIs** configuration option is added to the OIDC client and is aligned with the OIDC specification. You can use a different set of redirect URIs for redirection after login and logout. The value **+** used for **Valid Post Logout Redirect URIs** means that the logout uses the same set of redirect URIs as specified by the option of **Valid Redirect URIs**. This change also matches the default behavior when migrating from a previous version due to backwards compatibility.

5.1.2.1. UserInfo Endpoint Changes

5.1.2.1.1. Error response changes

The UserInfo endpoint is now returning error responses fully compliant with RFC 6750 (The OAuth 2.0 Authorization Framework: Bearer Token Usage). Error code and description (if available) are provided as **WWW-Authenticate** challenge attributes rather than JSON object fields.

The responses will be the following, depending on the error condition:

- In case no access token is provided:

```
401 Unauthorized
WWW-Authenticate: Bearer realm="myrealm"
```

- In case several methods are used simultaneously to provide an access token (for example, Authorization header + POST access_token parameter), or POST parameters are duplicated:

```
400 Bad Request
WWW-Authenticate: Bearer realm="myrealm", error="invalid_request", error_description="..."
```

- In case an access token is missing **openid** scope:

```
403 Forbidden
WWW-Authenticate: Bearer realm="myrealm", error="insufficient_scope",
error_description="Missing openid scope"
```

- In case of inability to resolve cryptographic keys for UserInfo response signing/encryption:

```
500 Internal Server Error
```

- In case of a token validation error, a **401 Unauthorized** is returned in combination with the **invalid_token** error code. This error includes user and client related checks and actually captures all the remaining error cases:

```
401 Unauthorized
WWW-Authenticate: Bearer realm="myrealm", error="invalid_token", error_description="..."
```

5.1.2.1.2. Other Changes to the UserInfo endpoint

It is now required for access tokens to have the **openid** scope, which is stipulated by UserInfo being a feature specific to OpenID Connect and not OAuth 2.0. If the **openid** scope is missing from the token, the request will be denied as **403 Forbidden**. See the preceding section.

UserInfo now checks the user status, and returns the **invalid_token** response if the user is disabled.

5.1.2.1.3. Change of the default Client ID mapper of Service Account Client.

Default **Client ID** mapper of **Service Account Client** has been changed. **Token Claim Name** field value has been changed from **clientId** to **client_id**. **client_id** claim is compliant with OAuth2 specifications:

- JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens
- OAuth 2.0 Token Introspection
- OAuth 2.0 Token Exchange

clientId userSession note still exists.

5.1.2.1.4. Added iss parameter to OAuth 2.0/OpenID Connect Authentication Response

RFC 9207 OAuth 2.0 Authorization Server Issuer Identification specification adds the parameter **iss** in the OAuth 2.0/OpenID Connect Authentication Response for realizing secure authorization responses.

In past releases, we did not have this parameter, but now Red Hat build of Keycloak adds this parameter by default, as required by the specification. However, some OpenID Connect / OAuth2 adapters, and especially older Red Hat build of Keycloak adapters, may have issues with this new parameter. For example, the parameter will be always present in the browser URL after successful authentication to the client application.

In these cases, it may be useful to disable adding the **iss** parameter to the authentication response. This can be done for the particular client in the Admin Console, in client details in the section with **OpenID Connect Compatibility Modes**. You can enable **Exclude Issuer From Authentication Response** to prevent adding the **iss** parameter to the authentication response.

5.2. MIGRATING RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM APPLICATIONS

5.2.1. Red Hat JBoss Enterprise Application Platform 8.x

Your applications no longer need any additional dependency to integrate with Red Hat build of Keycloak or any other OpenID Provider.

Instead, you can leverage the OpenID Connect support from the JBoss EAP native OpenID Connect Client. For more information, take a look at OpenID Connect in JBoss EAP.

The JBoss EAP native adapter relies on a configuration schema very similar to the Red Hat build of Keycloak Adapter JSON Configuration. For instance, a deployment using a **keycloak.json** configuration file can be mapped to the following configuration in JBoss EAP:

```
{
  "realm": "quickstart",
  "auth-server-url": "http://localhost:8180",
  "ssl-required": "external",
  "resource": "jakarta-servlet-authz-client",
  "credentials": {
    "secret": "secret"
  }
}
```

For examples about integrating Jakarta-based applications using the JBoss EAP native adapter with Red Hat build of Keycloak, see the following examples at the Red Hat build of Keycloak Quickstart Repository:

- JAX-RS Resource Server
- Servlet Application

It is strongly recommended to migrate to JBoss EAP native OpenID Connect client as it is the best candidate for Jakarta applications deployed to JBoss EAP 8 and newer.

5.2.2. Red Hat JBoss Enterprise Application Platform 7.x

As Red Hat JBoss Enterprise Application Platform 7.x is close to ending full support, Red Hat build of Keycloak will not provide support for it. For existing applications deployed to Red Hat JBoss Enterprise Application Platform 7.x adapters with maintenance support are available through Red Hat Single Sign-On 7.6.

Red Hat Single Sign-On 7.6 adapters are supported to be used in combination with the Red Hat build of Keycloak 24.0 server.

5.2.3. Red Hat JBoss Enterprise Application Platform 6.x

As Red Hat JBoss Enterprise Application PlatformJBoss EAP 6.x has reached end of maintenance support, going forward neither Red Hat Single Sign-On 7.6 or Red Hat build of Keycloak will provide support for it.

5.3. MIGRATING SPRING BOOT APPLICATIONS

The Spring Framework ecosystem is evolving fast and you should have a much better experience by leveraging the OpenID Connect support already available there.

Your applications no longer need any additional dependency to integrate with Red Hat build of Keycloak or any other OpenID Provider but rely on the comprehensive OAuth2/OpenID Connect support from Spring Security. For more information, see OAuth2/OpenID Connect support from Spring Security .

In terms of capabilities, it provides a standard-based OpenID Connect client implementation. An example of a capability that you might want to review, if not already using the standard protocols, is

Logout. Red Hat build of Keycloak provides full support for standard-based logout protocols from the OpenID Connect ecosystem.

For examples of how to integrate Spring Security applications with Red Hat build of Keycloak, see the Quickstart Repository.

If migrating from the Red Hat build of Keycloak Client Adapter for Spring Boot is not an option, you still have access to the adapter from Red Hat Single Sign-On 7.6, which is now in maintenance only support.

Red Hat Single Sign-On 7.6 adapters are supported to be used in combination with the Red Hat build of Keycloak 24.0 server.

5.4. MIGRATING RED HAT FUSE APPLICATIONS

As Red Hat Fuse has reached the end of full support, Red Hat build of Keycloak 24.0 will not provide any support for it. Red Hat Fuse adapters are still available with maintenance support through Red Hat Single Sign-On 7.6.

Red Hat Single Sign-On 7.6 adapters are supported to be used in combination with the Red Hat build of Keycloak 24.0 server.

5.5. MIGRATING APPLICATIONS USING THE AUTHORIZATION SERVICES POLICY ENFORCER

To support integration with the Red Hat build of Keycloak Authorization Services, the policy enforcer is available separately from the Java Client Adapters.

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-policy-enforcer</artifactId>
    <version>${Red Hat build of Keycloak .version}</version>
</dependency>
```

By decoupling it from the Java Client Adapters, it is possible now to integrate Red Hat build of Keycloak to any Java technology that provides built-in support for OAuth2 or OpenID Connect. The Red Hat build of Keycloak Policy Enforcer provides built-in support for the following types of applications:

- Servlet Application Using Fine-grained Authorization
- Spring Boot REST Service Protected Using Red Hat build of Keycloak Authorization Services

For integration of the Red Hat build of Keycloak Policy Enforcer with different types of applications, consider the following examples:

- Servlet Application Using Fine-grained Authorization
- Spring Boot REST Service Protected Using Keycloak Authorization Services

If migrating from the Red Hat Single Sign-On 7.6 Java Adapter you are using is not an option, you still have access to the adapter from Red Hat Single Sign-On 7.6, which is now in maintenance support.

Red Hat Single Sign-On 7.6 adapters are supported to be used in combination with the Red Hat build of Keycloak 24.0 server.

Additional resources

- Policy enforcers

5.6. MIGRATING SINGLE PAGE APPLICATIONS (SPA) USING THE RED HAT BUILD OF KEYCLOAK JS ADAPTER

To migrate applications secured with the Red Hat Single Sign-On 7.6 adapter, upgrade to Red Hat build of Keycloak 24.0, which provides a more recent version of the adapter. Depending on how it is used, there are some minor changes needed, which are described below.

5.6.1. Legacy Promise API removed

With this release, the legacy Promise API methods from the Red Hat build of Keycloak JS adapter is removed. This means that calling **.success()** and **.error()** on promises returned from the adapter is no longer possible.

5.6.2. Required to be instantiated with the new operator

In a previous release, deprecation warnings were logged when the Red Hat build of Keycloak JS adapter is constructed without the new operator. Starting with this release, doing so will throw an exception instead. This change is to align with the expected behavior of JavaScript classes, which will allow further refactoring of the adapter in the future.

To migrate applications secured with the Red Hat Single Sign-On 7.6 adapter, upgrade to Red Hat build of Keycloak 24.0, which provides a more recent version of the adapter.

5.7. MIGRATING SAML APPLICATIONS

5.7.1. Migrating Red Hat JBoss Enterprise Application Platform applications

5.7.1.1. Red Hat JBoss Enterprise Application Platform 8.x

Red Hat build of Keycloak 24.0 includes client adapters for Red Hat JBoss Enterprise Application Platform 8.x, including support for Jakarta EE.

5.7.1.2. Red Hat JBoss Enterprise Application Platform 7.x

As Red Hat JBoss Enterprise Application Platform 7.x is close to ending full support, Red Hat build of Keycloak will not provide support for it. For existing applications deployed to Red Hat JBoss Enterprise Application Platform 7.x adapters with maintenance support are available through Red Hat Single Sign-On 7.6.

Red Hat Single Sign-On 7.6 adapters are supported to be used in combination with the Red Hat build of Keycloak 24.0 server.

5.7.1.3. Red Hat JBoss Enterprise Application Platform 6.x

As Red Hat JBoss Enterprise Application PlatformJBoss EAP 6.x has reached end of maintenance support, going forward neither Red Hat Single Sign-On 7.6 or Red Hat build of Keycloak will provide support for it..

5.7.2. Key changes in SAML protocol and client settings

5.7.2.1. SAML SP metadata changes

Prior to this release, SAML SP metadata contained the same key for both signing and encryption use. Starting with this version of Keycloak, we include only encryption intended realm keys for encryption use in SP metadata. For each encryption key descriptor we also specify the algorithm that it is supposed to be used with. The following table shows the supported XML-Enc algorithms with the mapping to Red Hat build of Keycloak realm keys.

XML-Enc algorithm	Realm key algorithm
rsa-oaep-mgf1p	RSA-OAEP
rsa-1_5	RSA1_5

Additional resources

- Keycloak Upgrading Guide

5.7.2.2. Deprecated RSA_SHA1 and DSA_SHA1 algorithms for SAML

Algorithms **RSA_SHA1** and **DSA_SHA1**, which can be configured as **Signature algorithms** on SAML adapters, clients and identity providers are deprecated. We recommend to use safer alternatives based on **SHA256** or **SHA512**. Also, verifying signatures on signed SAML documents or assertions with these algorithms do not work on Java 17 or higher. If you use this algorithm and the other party consuming your SAML documents is running on Java 17 or higher, verifying signatures will not work.

The possible workaround is to remove algorithms such as the following:

- **http://www.w3.org/2000/09/xmldsig#rsa-sha1** or **http://www.w3.org/2000/09/xmldsig#dsa-sha1** from the list
- "disallowed algorithms" configured on property **jdk.xml.dsig.secureValidationPolicy** in the file **$JAVA_HOME/conf/security/java.security**

CHAPTER 6. MIGRATING CUSTOM PROVIDERS

Similarly to the Red Hat Single Sign-On 7.6, custom providers are deployed to the Red Hat build of Keycloak by copying them to a deployment directory. In the Red Hat build of Keycloak, copy your providers to the **providers** directory instead of **standalone/deployments**, which no longer exists. Additional dependencies should also be copied to the **providers** directory.

Red Hat build of Keycloak does not use a separate classpath for custom providers, so you may need to be more careful with additional dependencies that you include. In addition, the **EAR** and **WAR** packaging formats, and **jboss-deployment-structure.xml** files, are no longer supported.

While Red Hat Single Sign-On 7.6 automatically discovered custom providers, and even supported the ability to hot-deploy custom providers while Keycloak is running, this behavior is no longer supported. Also, after you make a change to the providers or dependencies in the **providers** directory, you have to do a build or restart the server with the auto build feature.

Depending on what APIs your providers use you may also need to make some changes to the providers. See the following sections for details.

6.1. TRANSITION FROM JAVA EE TO JAKARTA EE

Keycloak migrated its codebase from Java EE (Enterprise Edition) to Jakarta EE, which brought various changes. We have upgraded all Jakarta EE specifications in order to support Jakarta EE 10, such as:

- Jakarta Persistence 3.1
- Jakarta RESTful Web Services 3.1
- Jakarta Mail API 2.1
- Jakarta Servlet 6.0
- Jakarta Activation 2.1

Jakarta EE 10 provides a modernized, simplified, lightweight approach to building cloud-native Java applications. The main changes provided within this initiative are changing the namespace from **javax.*** to **jakarta.***. This change does not apply for **javax.*** packages provided directly in the JDK, such as **javax.security**, **javax.net**, **javax.crypto**, etc.

In addition, Jakarta EE APIs like session/stateless beans are no longer supported.

6.2. REMOVED THIRD PARTY DEPENDENCIES

Some dependencies were removed in Red Hat build of Keycloak including

- **openshift-rest-client**
- **okio-jvm**
- **okhttp**
- **commons-lang**
- **commons-compress**
- **jboss-dmr**

- **kotlin-stdlib**

Also, since Red Hat build of Keycloak is no longer based on EAP, most of the EAP dependencies were removed. This change means that if you use any of these libraries as dependencies of your own providers deployed to the Red Hat build of Keycloak, you may also need to copy those JAR files explicitly to the Keycloak distribution **providers** directory.

6.3. CONTEXT AND DEPENDENCY INJECTION ARE NO LONGER ENABLED FOR JAX-RS RESOURCES

To provide a better runtime and leverage as much as possible the underlying stack, all injection points for contextual data using the **javax.ws.rs.core.Context** annotation were removed. The expected improvement in performance involves no longer creating proxies instances multiple times during the request lifecycle, and drastically reducing the amount of reflection code at runtime.

If you need access to the current request and response objects, you can now obtain their instances directly from the **KeycloakSession**:

```
@Context
org.jboss.resteasy.spi.HttpRequest request;
@Context
org.jboss.resteasy.spi.HttpResponse response;
```

was replaced by:

```
KeycloakSession session = // obtain the session, which is usually available when creating a custom provider from a factory
KeycloakContext context = session.getContext();

HttpRequest request = context.getHttpRequest();
HttpResponse response = context.getHttpResponse();
```

Additional contextual data can be obtained from the runtime through the **KeycloakContext** instance:

```
KeycloakSession session = // obtain the session
KeycloakContext context = session.getContext();
MyContextualObject myContextualObject = context.getContextObject(MyContextualObject.class);
```

6.4. DEPRECATED METHODS FROM DATA PROVIDERS AND MODELS

Some previously deprecated methods are now removed in Red Hat build of Keycloak:

- **RealmModel#searchForGroupByNameStream(String, Integer, Integer)**
- **UserProvider#getUsersStream(RealmModel, boolean)**
- **UserSessionPersisterProvider#loadUserSessions(int, int, boolean, int, String)**
- Interfaces added for Streamification work, such as **RoleMapperModel.Streams** and similar
- **KeycloakModelUtils#getClientScopeMappings**
- Deprecated methods from **KeycloakSession**

- **UserQueryProvider#getUsersStream** methods

Also, these other changes were made:

- Some methods from **UserSessionProvider** were moved to **UserLoginFailureProvider**.
- **Streams** interfaces in federated storage provider classes were deprecated.
- Streamification - interfaces now contain only Stream-based methods.
  For example in **GroupProvider** interface

```
@Deprecated
List<GroupModel> getGroups(RealmModel realm);
```

  was replaced by

```
Stream<GroupModel> getGroupsStream(RealmModel realm);
```

- Consistent parameter ordering - methods now have strict parameter ordering where **RealmModel** is always the first parameter.
  For example in **UserLookupProvider** interface:

```
@Deprecated
UserModel getUserById(String id, RealmModel realm);
```

  was replaced by

```
UserModel getUserById(RealmModel realm, String id)
```

6.4.1. List of changed interfaces

(**o.k.** stands for **org.keycloak.** package)

- **server-spi** module
  - **o.k.credential.CredentialInputUpdater**
  - **o.k.credential.UserCredentialStore**
  - **o.k.models.ClientProvider**
  - **o.k.models.ClientSessionContext**
  - **o.k.models.GroupModel**
  - **o.k.models.GroupProvider**
  - **o.k.models.KeyManager**
  - **o.k.models.KeycloakSessionFactory**
  - **o.k.models.ProtocolMapperContainerModel**
  - **o.k.models.RealmModel**

- **o.k.models.RealmProvider**
- **o.k.models.RoleContainerModel**
- **o.k.models.RoleMapperModel**
- **o.k.models.RoleModel**
- **o.k.models.RoleProvider**
- **o.k.models.ScopeContainerModel**
- **o.k.models.UserCredentialManager**
- **o.k.models.UserModel**
- **o.k.models.UserProvider**
- **o.k.models.UserSessionProvider**
- **o.k.models.utils.RoleUtils**
- **o.k.sessions.AuthenticationSessionProvider**
- **o.k.storage.client.ClientLookupProvider**
- **o.k.storage.group.GroupLookupProvider**
- **o.k.storage.user.UserLookupProvider**
- **o.k.storage.user.UserQueryProvider**

- **server-spi-private** module
  - **o.k.events.EventQuery**
  - **o.k.events.admin.AdminEventQuery**
  - **o.k.keys.KeyProvider**

6.4.2. Refactorings in the storage layer

Red Hat build of Keycloak undergoes a large refactoring to simplify the API usage, which impacts existing code. Some of these changes require updates to existing code. The following sections provide more detail.

6.4.2.1. Changes in the module structure

Several public APIs around storage functionality in **KeycloakSession** have been consolidated, and some have been moved, deprecated, or removed. Three new modules have been introduced, and data-oriented code from **server-spi**, **server-spi-private**, and **services** modules have been moved there:

org.keycloak:keycloak-model-legacy

Contains all public facing APIs from the legacy store, such as the User Storage API.

org.keycloak:keycloak-model-legacy-private

Contains private implementations that relate to user storage management, such as storage ***Manager** classes.

org.keycloak:keycloak-model-legacy-services

Contains all REST endpoints that directly operate on the legacy store.

If you are using for example in your custom user storage provider implementation the classes which have been moved to the new modules, you need to update your dependencies to include the new modules listed above.

6.4.2.2. Changes in KeycloakSession

KeycloakSession has been simplified. Several methods have been removed in **KeycloakSession**.

KeycloakSession session contained several methods for obtaining a provider for a particular object type, such as for a **UserProvider** there are **users()**, **userLocalStorage()**, **userCache()**, **userStorageManager()**, and **userFederatedStorage()**. This situation may be confusing for the developer who has to understand the exact meaning of each method.

For those reasons, only the **users()** method is kept in **KeycloakSession**, and should replace all other calls listed above. The rest of the methods have been removed. The same pattern of depreciation applies to methods of other object areas, such as **clients()** or **groups()**. All methods ending in ***StorageManager()** and ***LocalStorage()** have been removed. The next section describes how to migrate those calls to the new API or use the legacy API.

6.4.3. Migrating existing providers

The existing providers need no migration if they do not call a removed method, which should be the case for most providers.

If the provider uses removed methods, but does not rely on local versus non-local storage, changing a call from the now removed **userLocalStorage()** to the method **users()** is the best option. Be aware that the semantics change here as the new method involves a cache if that has been enabled in the local setup.

Before migration: accessing a removed API doesn't compile

```
session.userLocalStorage();
```

After migration: accessing the new API when caller does not depend on the legacy storage API

```
session.users();
```

In the rare case when a custom provider needs to distinguish between the mode of a particular provider, access to the deprecated objects is provided by using the **LegacyStoreManagers** data store provider. This might be the case if the provider accesses the local storage directly or wants to skip the cache. This option will be available only if the legacy modules are part of the deployment.

Before migration: accessing a removed API

```
session.userLocalStorage();
```

After migration: accessing the new functionality via the LegacyStoreManagers API

```
((LegacyDatastoreProvider) session.getProvider(DatastoreProvider.class)).userLocalStorage();
```

Some user storage related APIs have been wrapped in **org.keycloak.storage.UserStorageUtil** for convenience.

6.4.4. Changes to RealmModel

The methods **getUserStorageProviders**, **getUserStorageProvidersStream**, **getClientStorageProviders**, **getClientStorageProvidersStream**, **getRoleStorageProviders** and **getRoleStorageProvidersStream** have been removed. Code which depends on these methods should cast the instance as follows:

Before migration: code will not compile due to the changed API

```
realm.getClientStorageProvidersStream()...;
```

After migration: cast the instance to the legacy interface

```
((LegacyRealmModel) realm).getClientStorageProvidersStream()...;
```

Similarly, code that used to implement the interface **RealmModel** and wants to provide these methods should implement the new interface **LegacyRealmModel**. This interface is a sub-interface of **RealmModel** and includes the old methods:

Before migration: code implements the old interface

```
public class MyClass extends RealmModel {
    /* might not compile due to @Override annotations for methods no longer present
       in the interface RealmModel. / / ... */
}
```

After migration: code implements the new interface

```
public class MyClass extends LegacyRealmModel {
    /* ... */
}
```

6.4.5. Interface UserCache moved to the legacy module

As the caching status of objects will be transparent to services, the interface **UserCache** has been moved to the module **keycloak-model-legacy**.

Code that depends on the legacy implementation should access the **UserCache** directly.

Before migration: code will not compile[source,java,subs="+quotes"]

```
session**.userCache()**.evict(realm, user);
```

After migration: use the API directly

```
UserStorageUitl.userCache(session);
```

To trigger the invalidation of a realm, instead of using the **UserCache** API, consider triggering an event:

Before migration: code uses cache API[source,java,subs="+quotes"]

```
UserCache cache = session.getProvider(UserCache.class);
if (cache != null) cache.evict(realm)();
```

After migration: use the invalidation API

```
session.invalidate(InvalidationHandler.ObjectType.REALM, realm.getId());
```

6.4.6. Credential management for users

Credentials for users were previously managed using **session.userCredentialManager().*method*(realm, user, ...)**. The new way is to leverage **user.credentialManager().*method*(...)**. This form gets the credential functionality closer to the API of users, and does not rely on prior knowledge of the user credential's location in regard to realm and storage.

The old APIs have been removed.

Before migration: accessing a removed API

```
session.userCredentialManager().createCredential(realm, user, credentialModel)
```

After migration: accessing the new API

```
user.credentialManager().createStoredCredential(credentialModel)
```

For a custom **UserStorageProvider**, there is a new method **credentialManager()** that needs to be implemented when returning a **UserModel**. Those must return an instance of the **LegacyUserCredentialManager**:

Before migration: code will not compile due to the new method credentialManager() required by UserModel

```
public class MyUserStorageProvider implements UserLookupProvider, ... {
   /* ... */
   protected UserModel createAdapter(RealmModel realm, String username) {
      return new AbstractUserAdapter(session, realm, model) {
         @Override
         public String getUsername() {
            return username;
         }
      };
   }
}
```

After migration: implementation of the API UserModel.credentialManager() for the legacy store.

```
public class MyUserStorageProvider implements UserLookupProvider, ... {
```

```
    /* ... */
    protected UserModel createAdapter(RealmModel realm, String username) {
        return new AbstractUserAdapter(session, realm, model) {
            @Override
            public String getUsername() {
                return username;
            }

            @Override
            public SubjectCredentialManager credentialManager() {
                return new LegacyUserCredentialManager(session, realm, this);
            }
        };
    }
}
```

CHAPTER 7. MIGRATING CUSTOM THEMES

7.1. NEW ADMIN CONSOLE

The new Admin Console (keycloak.v2) is built using React. The old Admin Console (keycloak) was built with AngularJS 1.x, which reached end-of-life a while ago. Thus, there is no migration path from the old console or any theme that extends it. The base theme Admin Console is also not supported for the same reason.

7.2. NEW ACCOUNT CONSOLE

The new Account Console (keycloak.v2) is built using React, providing a better user experience. The old Account Console (keycloak) was built with basic server-side templating. Thus, there is no migration path from the old console or any theme that extends it.

7.3. MIGRATING LOGIN THEMES

Themes are used to configure the look and feel of login pages and the Account Console.

When creating or updating custom themes, especially when overriding templates, it may be useful to use the built-in templates as a reference. These templates are in **${KC_HOME}/lib/lib/main/org.keycloak.keycloak-themes-${KC_VERSION}.jar**, which can be opened using any standard ZIP archive tool.

When running the server in development mode using **start-dev**, themes are not cached so that you can easily work on them without a need to restart the server when making changes.

To install custom themes, you can choose from packaging your theme files as a JAR file and deploy it to the **${KC_HOME}/providers** directory, or copy files directly to the **${KC_HOME}/themes** directory. In both cases, see the Server Developer Guide for more details about the file and directory structure expected by the server.

CHAPTER 8. MIGRATING UPSTREAM KEYCLOAK TO RED HAT BUILD OF KEYCLOAK 24.0

Starting with version 22, minimal differences exist between Red Hat build of Keycloak and upstream Keycloak. The following differences exist:

- For upstream Keycloak, the distribution artifacts are on keycloak.org; for Red Hat build of Keycloak, the distribution artifacts are on the Red Hat customer portal.
- Oracle and MSSQL database drivers are bundled with upstream Keycloak, but not bundled with Red Hat build of Keycloak. See Configuring the database for detailed steps on how to install those drivers.
- The GELF log handler is not available in Red Hat build of Keycloak.

The migration process depends on the version of Keycloak to be migrated and the type of Keycloak installation. See the following sections for details.

8.1. MATCHING KEYCLOAK VERSION

The migration process depends on the version of Keycloak to be migrated.

- If your Keycloak project version matches the Red Hat build of Keycloak version, migrate Keycloak by using the Red Hat build of Keycloak artifacts on the Red Hat customer portal.
- If your Keycloak project version is an older version, use the Keycloak Upgrading Guide to upgrade Keycloak to match the Red Hat build of Keycloak version. Then, migrate Keycloak using the artifacts on the Red Hat customer portal.
- If your Keycloak project version is greater than the Red Hat build of Keycloak version, you cannot migrate to Red Hat build of Keycloak. Instead, create a new deployment of Red Hat build of Keycloak or wait for a future Red Hat build of Keycloak release.

8.2. MIGRATION BASED ON TYPE OF KEYCLOAK INSTALLATION

Once you have a matching version of Keycloak, migrate Keycloak based on the type of installation.

- If you installed Keycloak from a ZIP distribution, migrate Keycloak by using the artifacts on the Red Hat customer portal.
- If you deployed the Keycloak Operator, uninstall it and install the Red Hat build of Keycloak Operator by using the Operator guide. The CRs are compatible between upstream Keycloak and Red Hat build of Keycloak.
- If you created a custom server container image, rebuild it by using the Red Hat build of Keycloak image. See Running Keycloak in a Container .

CHAPTER 9. OTHER NOTABLE CHANGES

9.1. JAVASCRIPT ENGINE AVAILABLE BY DEFAULT ON THE CLASSPATH

In the previous version, when Keycloak was used on Java 17 with Javascript providers (Script authenticator, Javascript authorization policy or Script protocol mappers for OIDC and SAML clients), it was needed to copy the javascript engine to the distribution. This is no longer needed as Nashorn javascript engine is available in Red Hat build of Keycloak server by default. When you deploy script providers, it is recommended to not copy Nashorn's script engine and its dependencies into the Red Hat build of Keycloak distribution.

9.2. RENAMED KEYCLOAK ADMIN CLIENT ARTIFACTS

After the upgrade to Jakarta EE, artifacts for Keycloak Admin clients were renamed to more descriptive names with consideration for long-term maintainability. However, two separate Keycloak Admin clients still exist: one with Jakarta EE and the other with Java EE support.

The **org.keycloak:keycloak-admin-client-jakarta** artifact is no longer released. The default one for the Keycloak Admin client with Jakarta EE support is **org.keycloak:keycloak-admin-client** (since version 24.0.0).

The new artifact with Java EE support is **org.keycloak:keycloak-admin-client-jee**.

9.2.1. Jakarta EE support

The new artifact with Java EE support is **org.keycloak:keycloak-admin-client-jee**. Jakarta EE support

Before migration:

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client-jakarta</artifactId>
    <version>18.0.0.redhat-00001</version>
</dependency>
```

After migration:

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
    <version>22.0.0.redhat-00001</version>
</dependency>
```

9.2.2. Java EE support

Before migration:

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
```

```
    <version>18.0.0.redhat-00001</version>
</dependency>
```

After migration:

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client-jee</artifactId>
    <version>22.0.0.redhat-00001</version>
</dependency>
```

9.3. NEVER EXPIRES OPTION REMOVED FROM CLIENT ADVANCED SETTINGS COMBOS

The option **Never expires** is now removed from all the combos of the **Advanced Settings** client tab. This option was misleading because the different lifespans or idle timeouts were never infinite, but limited by the general user session or realm values. Therefore, this option is removed in favor of the other two remaining options: **Inherits from the realm settings** (the client uses general realm timeouts) and **Expires in** (the value is overridden for the client). Internally the **Never expires** was represented by **-1**. Now that value is shown with a warning in the Admin Console and cannot be set directly by the administrator.

9.4. NEW EMAIL RULES AND LIMITS VALIDATION

Red Hat build of Keycloak has new rules on email creation to allow ASCII characters during the email creation. Also, a new limit of 64 characters on exists on local email part (before the @). So, a new parameter **--spi-user-profile-declarative-user-profile-max-email-local-part-length** is added to set max email local part length taking backwards compatibility into consideration. The default value is 64.

```
kc.sh start --spi-user-profile-declarative-user-profile-max-email-local-part-length=100
```