Guides Keycloak Server

Version 20.0, 2024-01-03

Welcome to the Keycloak serve guide.

Configuring Keycloak

This guide explains the configuration methods for Keycloak and how to start and apply the preferred configuration. It includes configuration guidelines for optimizing Keycloak for faster startup and low memory footprint.

Configuration Sources for Keycloak

Keycloak loads the configuration from four different configuration sources:

- command-line parameters
- environment variables
- user-created `.conf` file
- `keycloak.conf` file located in the `conf` directory.

Configuration sources have a descending ordinal: command-line parameters take precedence over environment variables. Environment variables take precedence over options set by using a specific configuration file. Options from a specific config file take precedence over options defined in `conf/keycloak.conf`. When the same configuration key is found in multiple configuration sources, the applied value is taken from the configuration source with the highest ordinal.

Example: Configuring the db-url-host parameter.

Source	Format
CLI	--db-url=cliValue
Environment Variable	KC_DB_URL=envVarValue
Configuration file	db-url=confFileValue

In the example above, the `db-url` value is set in all three configuration sources. The actual value that is used at startup would be the `cliValue`. If `--db-url=cliValue` is not used, the used value would be `KC_DB_URL=envVarValue`, and last but not least the `db-url=confFileValue` would be used when no environment variable with the same Key is present. When this value is specified in a user defined configuration file and in `conf/keycloak.conf`, the value from the user defined configuration file takes precedence.

Configuration Format

The configuration follows a "unified-per-source" format, that is easily translatable from one configuration source to another:

Command-line parameter format

Values for the command-line are following the `--<key-with-dashes>=<value>` format. For some values, there’s also a `-<abbreviation>=value` shorthand.

Environment variable format

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

Configuration file format

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

You can easily translate a Key/Value pair from one configuration source to the other.

You will find the relevant configuration options for a specific guide in all three formats on the table at the bottom of each guide. You can find all available options at the All configuration guide.

The configuration source and the corresponding format you should use is use-case specific.

Example - Configure `db-url-host` on different configuration sources:

The following example shows how the configuration for the db url host looks for all 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
```

Using environment variables for configuration values

It is possible to 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}
```

To specify a fallback value in case the environment variable can not be resolved, use a `:`:

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

Configuring the server using 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 invoking the following command:

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

Using the command-line help

Keycloak is packed with a CLI that helps you to configure Keycloak. To find out about the available configuration, invoke the following command:

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

Alternatively, you can find all server options at the All configuration guide.

Using raw Quarkus properties

In most cases, the available configuration options should suffice to configure the server. However, you might need to use properties directly from the underlying Quarkus framework to enable a specific behavior or capability that is missing in the keycloak configuration.

If possible, avoid using properties directly from Quarkus. These are considered unsupported by Keycloak. If your need is essential, consider opening an enhancement request first and help us to improve Keycloak’s configuration to fit your needs.

If that’s not possible, you can configure the server using raw Quarkus properties:

- Create a `quarkus.properties` file in the `conf` directory and define any property you need.

For a complete list of Quarkus properties, see the Quarkus documentation. Be aware that Keycloak uses a subset of quarkus extensions, so not all properties will be available.

When a quarkus property is a runtime property (no lock icon shown in the quarkus guide), it is also handled as runtime property for Keycloak. When a quarkus property is a build time property, you have to invoke a `build` for the property to be applied. See the sections below for further information around the build command.

Note that some quarkus properties are mapped by the Keycloak configuration, for example `quarkus.http.port` and similar properties that are needed to configure Keycloak. If the property is used by Keycloak, defining the same underlying property key in `quarkus.properties` will have no effect, as the keycloak configuration value takes precedence over the quarkus property value.

Starting Keycloak

Keycloak can be started in two operating modes, `development mode` and `production mode`. Both modes offer a different set of defaults for the environment they are intended to be used.

Starting Keycloak in development mode

The development mode is targeted for people trying out Keycloak the first time and get it up and running quickly. It also offers convenient defaults for developers, for example to develop a new Keycloak theme.

The development mode is started by invoking the following command:

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

Defaults

The 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- and Template-caching is disabled

Starting Keycloak in production mode

The production mode is targeted for deployments of Keycloak into production environments and follows a "secure by default" principle.

The production mode is started by invoking the following command:

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

Without further configuration, this command will not start Keycloak and show you an error instead. This is done on purpose, because Keycloak follows a "secure by default" principle in this mode and expects to have a hostname setup and a HTTPS/TLS setup available when started in production mode.

Defaults

The 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

Make sure to follow the steps outlined in the Configuring Keycloak for production guide before deploying Keycloak to production environments.

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

Setup of the initial admin user

The initial admin user can be added manually using the web frontend. It needs to be accessed using a local connection (localhost) or using environment variables:

To add the initial admin user using environment variables, set `KEYCLOAK_ADMIN=<username>` for the initial admin username and `KEYCLOAK_ADMIN_PASSWORD=<password>` for the initial admin password. 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 UI 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. Keycloak ignores the values and starts up correctly.

Optimize the Keycloak startup

It is highly recommended to optimize Keycloak for better startup times and memory consumption before deploying into production environments. This section shows you how to apply a set of optimizations for Keycloak to get the best performance and runtime behavior possible.

Create an optimized Keycloak build

By default, when the `start` or `start-dev` commands are used, Keycloak runs a `build` command under the covers for convenience reasons. This `build` command performs a set of optimizations to achieve an optimized startup- and runtime-behavior. The build process can take some time, usually a few seconds. Especially when running Keycloak in containerized environments like Kubernetes or OpenShift, startup time is important. So in order to avoid the time that gets lost when running a `build` as part of Keycloaks first startup, it is possible and recommended to invoke a `build` explicitly before starting up, for example as a separate step in a CI/CD pipeline.

First step: Run a build explicitly

To run a `build`, invoke the following command:

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

As you may notice, the command above shows `build options` that should be invoked. Keycloak distinguishes between **build options**, that are usable when invoking the `build` command, and **configuration options**, that are usable when starting up the server.

For a non-optimized startup of Keycloak, this distinction has no effect, but when a build is invoked beforehand, there's only a subset of Options available to the build command. The reason is, that build options get persisted into Keycloaks classpath, so configuration for e.g. credentials like `db-password` must not get persisted for security reasons.

Build options are marked in the All configuration guide with a tool icon. Find available build options either by looking at the All configuration page with build options selected or by invoking the following command:

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

Example: Run the `build` command to set the database to PostgreSQL before startup:

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

Second step: Start Keycloak using `--optimized`

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

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

The `--optimized` parameter tells Keycloak to assume a pre-built, already optimized Keycloak image is used. As a result, Keycloak avoids checking for and running a build directly at startup to save the time to walk through this process.

You can invoke all configuration options at startup - these are all options in the All configuration guide that are **not** marked with a tool icon.

If a build option is found at startup with an equal value to the value used when invoking the `build`, it gets silently ignored when using the `--optimized` flag. If it has a different value than the value used when a build was invoked, a warning is shown in the logs and the previously built value is used. In order for this value to take effect, you have to run a new `build` before starting.

The following example shows how to create an optimized build, then start Keycloak using the --optimized parameter:

Create an optimized build

Set build option for the postgresql database vendor using the build command

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

Set the runtime configuration options to keycloak.conf

Set configuration options for postgres inside `conf/keycloak.conf`

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

Start the server with the optimized parameter

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

Most optimizations to startup and runtime behavior can be achieved by using the `build` command. By using the `keycloak.conf` file as a source for configuration options, Keycloak avoids some steps at startup that are needed when invoking the configuration using the command line, for example initialising the CLI itself. As a result, the server starts up even faster.

Underlying concepts

This section gives an overview around the underlying concepts Keycloak uses, especially when it comes to optimizing the startup.

Keycloak uses the Quarkus framework and it’s re-augmentation/mutable-jar approach under the covers. This process is started when a `build` is invoked.

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 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

Configuring Keycloak for production

A 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 guide describes the general areas of configuration required for a production ready 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 guide apply to all environments, whether it is containerized, on-premise, GitOps, or Ansible.

TLS for secure communication

Keycloak continually exchanges sensitive data, which means that all communication to and from 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 Keycloak, see the Configuring TLS and Configuring outgoing HTTP requests guides.

The hostname for Keycloak

In a production environment, Keycloak instances usually run in a private network, but 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 the Configuring the hostname guide.

Reverse proxy in a distributed environment

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

For details on configuring proxy communication modes in Keycloak, see the Using a reverse proxy guide. That guide also recommends which paths should be hidden from public access and which paths should be exposed so that Keycloak can secure your applications.

Production grade database

The database used by Keycloak is crucial for the overall performance, availability, reliability and integrity of Keycloak. For details on how to configure a supported database, see the Configuring the database guide.

Support for Keycloak in a cluster

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

Keycloak runs on top of JGroups and Infinispan, which provide a reliable, high-availability stack for a clustered scenario. When deployed to a cluster, the embedded Infinispan server communication should be secured. You secure this communication either by enabling authentication and encryption or by isolating the network used for cluster communication.

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

Configure Keycloak Server with IPv6 or IPv4

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, Keycloak is configured to prefer IPv4 addresses. In order to run with IPv6 addresses, you need to specify `java.net.preferIPv4Stack=false` (the JVM default) and `java.net.preferIPv6Addresses=true`. The latter ensures that any hostname to IP address conversions always return IPv6 address variants.

These system properties are conveniently set by the `JAVA_OPTS_APPEND` environment variable. For example, to change the IP stack preference from its default of IPv4 to IPv6, set an environment variable as follows:

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

All configuration

Cache

	Type	Default	
cache Defines the cache mechanism for high-availability. By default, a 'ispn' cache is used to create a cluster between multiple server nodes. A 'local' cache disables clustering and is intended for development and testing purposes. **CLI:** `--cache` **Env:** `KC_CACHE`	ispn, local	ispn	🛠
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-stack Define the default stack to use for cluster communication and node discovery. This option only takes effect if 'cache' is set to 'ispn'. Default: udp. **CLI:** `--cache-stack` **Env:** `KC_CACHE_STACK`	tcp, udp, kubernetes, ec2, azure, google		🛠

Storage (Experimental)

	Type	Default	
storage Experimental: Sets the default storage mechanism for all areas. **CLI:** `--storage` **Env:** `KC_STORAGE`	jpa, chm, hotrod		🛠
storage-area-action-token Experimental: Sets a storage mechanism for action tokens. **CLI:** `--storage-area-action-token` **Env:** `KC_STORAGE_AREA_ACTION_TOKEN`	jpa, chm, hotrod		🛠
storage-area-auth-session Experimental: Sets a storage mechanism for authentication sessions. **CLI:** `--storage-area-auth-session` **Env:** `KC_STORAGE_AREA_AUTH_SESSION`	jpa, chm, hotrod		🛠
storage-area-authorization Experimental: Sets a storage mechanism for authorizations. **CLI:** `--storage-area-authorization` **Env:** `KC_STORAGE_AREA_AUTHORIZATION`	jpa, chm, hotrod		🛠
storage-area-client Experimental: Sets a storage mechanism for clients. **CLI:** `--storage-area-client` **Env:** `KC_STORAGE_AREA_CLIENT`	jpa, chm, hotrod		🛠

	Type	Default	
storage-area-client-scope Experimental: Sets a storage mechanism for client scopes. **CLI:** `--storage-area-client-scope` **Env:** `KC_STORAGE_AREA_CLIENT_SCOPE`	jpa, chm, hotrod		🛠
storage-area-event-admin Experimental: Sets a storage mechanism for admin events. **CLI:** `--storage-area-event-admin` **Env:** `KC_STORAGE_AREA_EVENT_ADMIN`	jpa, chm, hotrod		🛠
storage-area-event-auth Experimental: Sets a storage mechanism for authentication and authorization events. **CLI:** `--storage-area-event-auth` **Env:** `KC_STORAGE_AREA_EVENT_AUTH`	jpa, chm, hotrod		🛠
storage-area-group Experimental: Sets a storage mechanism for groups. **CLI:** `--storage-area-group` **Env:** `KC_STORAGE_AREA_GROUP`	jpa, chm, hotrod		🛠
storage-area-login-failure Experimental: Sets a storage mechanism for login failures. **CLI:** `--storage-area-login-failure` **Env:** `KC_STORAGE_AREA_LOGIN_FAILURE`	jpa, chm, hotrod		🛠

	Type	Default	
storage-area-realm Experimental: Sets a storage mechanism for realms. **CLI:** `--storage-area-realm` **Env:** `KC_STORAGE_AREA_REALM`	jpa, chm, hotrod		🛠
storage-area-role Experimental: Sets a storage mechanism for roles. **CLI:** `--storage-area-role` **Env:** `KC_STORAGE_AREA_ROLE`	jpa, chm, hotrod		🛠
storage-area-single-use-object Experimental: Sets a storage mechanism for single use objects. **CLI:** `--storage-area-single-use-object` **Env:** `KC_STORAGE_AREA_SINGLE_USE_OBJECT`	jpa, chm, hotrod		🛠
storage-area-user Experimental: Sets a storage mechanism for users. **CLI:** `--storage-area-user` **Env:** `KC_STORAGE_AREA_USER`	jpa, chm, hotrod		🛠
storage-area-user-session Experimental: Sets a storage mechanism for user and client sessions. **CLI:** `--storage-area-user-session` **Env:** `KC_STORAGE_AREA_USER_SESSION`	jpa, chm, hotrod		🛠

	Type	Default	
storage-deployment-state-version-seed Experimental: Secret that serves as a seed to mask the version number of Keycloak in URLs. Need to be identical across all servers in the cluster. Will default to a random number generated when starting the server which is secure but will lead to problems when a loadbalancer without sticky sessions is used or nodes are restarted. **CLI:** `--storage-deployment-state-version-seed` **Env:** `KC_STORAGE_DEPLOYMENT_STATE_VERSION_SEED`			
storage-hotrod-host Experimental: Sets the host of the Infinispan server. **CLI:** `--storage-hotrod-host` **Env:** `KC_STORAGE_HOTROD_HOST`			
storage-hotrod-password Experimental: Sets the password of the Infinispan user. **CLI:** `--storage-hotrod-password` **Env:** `KC_STORAGE_HOTROD_PASSWORD`			
storage-hotrod-port Experimental: Sets the port of the Infinispan server. **CLI:** `--storage-hotrod-port` **Env:** `KC_STORAGE_HOTROD_PORT`			
storage-hotrod-username Experimental: Sets the username of the Infinispan user. **CLI:** `--storage-hotrod-username` **Env:** `KC_STORAGE_HOTROD_USERNAME`			

Database

	Type	Default	
db The database vendor. **CLI:** `--db` **Env:** `KC_DB`	dev-file, dev-mem, mariadb, mssql, mysql, oracle, postgres	dev-file	🛠
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	
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`			

	Type	Default	
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`			

	Type	Default	
db-url-properties Sets the properties of the default JDBC URL of the chosen vendor. 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`			

Transaction

	Type	Default	
transaction-xa-enabled If set to false, Keycloak uses a non-XA datasource in case the database does not support XA transactions. **CLI:** `--transaction-xa-enabled` **Env:** `KC_TRANSACTION_XA_ENABLED`	true, false	true	🛠

Feature

	Type	Default	
features Enables a set of one or more features. **CLI:** `--features` **Env:** `KC_FEATURES`	authorization, account2, account-api, admin-fine-grained-authz, admin-api, admin, admin2, docker, impersonation, openshift-integration, scripts, token-exchange, web-authn, client-policies, ciba, map-storage, par, declarative-user-profile, dynamic-scopes, client-secret-rotation, step-up-authentication, recovery-codes, update-email, preview		🛠

	Type	Default	
features-disabled Disables a set of one or more features. **CLI:** `--features-disabled` **Env:** `KC_FEATURES_DISABLED`	authorization, account2, account-api, admin-fine-grained-authz, admin-api, admin, admin2, docker, impersonation, openshift-integration, scripts, token-exchange, web-authn, client-policies, ciba, map-storage, par, declarative-user-profile, dynamic-scopes, client-secret-rotation, step-up-authentication, recovery-codes, update-email, preview		🛠

Hostname

	Type	Default	
hostname Hostname for the Keycloak server. **CLI:** `--hostname` **Env:** `KC_HOSTNAME`			

	Type	Default	
hostname-admin The hostname for accessing the administration console. Use this option if you are exposing the administration console using a hostname other than the value set to the 'hostname' option. **CLI:** `--hostname-admin` **Env:** `KC_HOSTNAME_ADMIN`			
hostname-admin-url Set the base URL for accessing the administration console, including scheme, host, port and path **CLI:** `--hostname-admin-url` **Env:** `KC_HOSTNAME_ADMIN_URL`			
hostname-path This should be set if proxy uses a different context-path for Keycloak. **CLI:** `--hostname-path` **Env:** `KC_HOSTNAME_PATH`			
hostname-port The port used by the proxy when exposing the hostname. Set this option if the proxy uses a port other than the default HTTP and HTTPS ports. **CLI:** `--hostname-port` **Env:** `KC_HOSTNAME_PORT`		-1	
hostname-strict Disables dynamically resolving the hostname from request headers. Should always be set to true in production, unless proxy verifies the Host header. **CLI:** `--hostname-strict` **Env:** `KC_HOSTNAME_STRICT`	true, false	true	

	Type	Default	
hostname-strict-backchannel By default backchannel URLs are dynamically resolved from request headers to allow internal and external applications. If all applications use the public URL this option should be enabled. **CLI:** `--hostname-strict-backchannel` **Env:** `KC_HOSTNAME_STRICT_BACKCHANNEL`	true, false	false	
hostname-url Set the base URL for frontend URLs, including scheme, host, port and path. **CLI:** `--hostname-url` **Env:** `KC_HOSTNAME_URL`			

HTTP/TLS

	Type	Default	
http-enabled Enables the HTTP listener. **CLI:** `--http-enabled` **Env:** `KC_HTTP_ENABLED`	true, false	false	
http-host The used HTTP Host. **CLI:** `--http-host` **Env:** `KC_HTTP_HOST`		0.0.0.0	
http-port The used HTTP port. **CLI:** `--http-port` **Env:** `KC_HTTP_PORT`		8080	

	Type	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`		/	🛠
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-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, request, required	none	

	Type	Default	
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	
https-key-store-type The type of the key store file. If not given, the type is automatically detected based on the file name. **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	
https-protocols The list of protocols to explicitly enable. **CLI:** `--https-protocols` **Env:** `KC_HTTPS_PROTOCOLS`		TLSv1.3	

	Type	Default	
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`			
https-trust-store-type The type of the trust store file. If not given, the type is automatically detected based on the file name. **CLI:** `--https-trust-store-type` **Env:** `KC_HTTPS_TRUST_STORE_TYPE`			

Health

	Type	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	false	🛠

Metrics

	Type	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	false	🛠

Proxy

	Type	Default	
proxy The proxy address forwarding mode if the server is behind a reverse proxy. **CLI:** `--proxy` **Env:** `KC_PROXY`	none, edge, reencrypt, passthrough	none	

Vault

	Type	Default	
vault Enables a vault provider. **CLI:** `--vault` **Env:** `KC_VAULT`	file, hashicorp		🛠
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`			

Logging

	Type	**Default**	
log Enable one or more log handlers in a comma-separated list. **CLI:** `--log` **Env:** `KC_LOG`	console, file, gelf	console	
log-console-color Enable or disable colors when logging to console. **CLI:** `--log-console-color` **Env:** `KC_LOG_CONSOLE_COLOR`	true, false	false	
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`		%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n	
log-console-output Set the log output to JSON or default (plain) unstructured logging. **CLI:** `--log-console-output` **Env:** `KC_LOG_CONSOLE_OUTPUT`	default, json	default	
log-file Set the log file path and filename. **CLI:** `--log-file` **Env:** `KC_LOG_FILE`		data/log/keycloak.log	

	Type	Default	
log-file-format Set a format specific to file log entries. **CLI:** `--log-file-format` **Env:** `KC_LOG_FILE_FORMAT`		%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n	
log-file-output Set the log output to JSON or default (plain) unstructured logging. **CLI:** `--log-file-output` **Env:** `KC_LOG_FILE_OUTPUT`	default, json	default	
log-gelf-facility The facility (name of the process) that sends the message. **CLI:** `--log-gelf-facility` **Env:** `KC_LOG_GELF_FACILITY`		keycloak	
log-gelf-host Hostname of the Logstash or Graylog Host. By default UDP is used, prefix the host with 'tcp:' to switch to TCP. Example: 'tcp:localhost' **CLI:** `--log-gelf-host` **Env:** `KC_LOG_GELF_HOST`		localhost	
log-gelf-include-location Include source code location. **CLI:** `--log-gelf-include-location` **Env:** `KC_LOG_GELF_INCLUDE_LOCATION`	true, false	true	

	Type	Default	
log-gelf-include-message-parameters Include message parameters from the log event. **CLI:** `--log-gelf-include-message-parameters` **Env:** `KC_LOG_GELF_INCLUDE_MESSAGE_PARAMETERS`	true, false	true	
log-gelf-include-stack-trace If set to true, occuring stack traces are included in the 'StackTrace' field in the GELF output. **CLI:** `--log-gelf-include-stack-trace` **Env:** `KC_LOG_GELF_INCLUDE_STACK_TRACE`	true, false	true	
log-gelf-level The log level specifying which message levels will be logged by the GELF logger. Message levels lower than this value will be discarded. **CLI:** `--log-gelf-level` **Env:** `KC_LOG_GELF_LEVEL`		INFO	
log-gelf-max-message-size Maximum message size (in bytes). If the message size is exceeded, GELF will submit the message in multiple chunks. **CLI:** `--log-gelf-max-message-size` **Env:** `KC_LOG_GELF_MAX_MESSAGE_SIZE`		8192	
log-gelf-port The port the Logstash or Graylog Host is called on. **CLI:** `--log-gelf-port` **Env:** `KC_LOG_GELF_PORT`		12201	

	Type	Default	
log-gelf-timestamp-format Set the format for the GELF timestamp field. Uses Java SimpleDateFormat pattern. **CLI:** `--log-gelf-timestamp-format` **Env:** `KC_LOG_GELF_TIMESTAMP_FORMAT`		yyyy-MM-dd HH:mm:ss,SSS	
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	

Enabling and disabling features

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 Keycloak.

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
```

Disabling features

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

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

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

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

You can disable all default features by entering this command:

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

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

Supported features

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

account-api	Account Management REST API
account2	New Account Management Console
admin-api	Admin API
admin2	New Admin Console
authorization	Authorization Service
ciba	OpenID Connect Client Initiated Backchannel Authentication (CIBA)
client-policies	Client configuration policies
impersonation	Ability for admins to impersonate users
par	OAuth 2.0 Pushed Authorization Requests (PAR)
step-up-authentication	Step-up Authentication
web-authn	W3C Web Authentication (WebAuthn)

Disabled by default

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

docker	Docker Registry protocol

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
declarative-user-profile	Configure user profiles using a declarative style
openshift-integration	Extension to enable securing OpenShift
recovery-codes	Recovery codes
scripts	Write custom authenticators using JavaScript
token-exchange	Token Exchange Service
update-email	Update Email Action

Deprecated features

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

admin	Legacy Admin Console

Relevant options

	Type	Default	
features Enables a set of one or more features. **CLI:** `--features` **Env:** `KC_FEATURES`	authorization, account2, account-api, admin-fine-grained-authz, admin-api, admin, admin2, docker, impersonation, openshift-integration, scripts, token-exchange, web-authn, client-policies, ciba, map-storage, par, declarative-user-profile, dynamic-scopes, client-secret-rotation, step-up-authentication, recovery-codes, update-email, preview		🛠

	Type	Default	
features-disabled Disables a set of one or more features. **CLI:** `--features-disabled` **Env:** `KC_FEATURES_DISABLED`	authorization, account2, account-api, admin-fine-grained-authz, admin-api, admin, admin2, docker, impersonation, openshift-integration, scripts, token-exchange, web-authn, client-policies, ciba, map-storage, par, declarative-user-profile, dynamic-scopes, client-secret-rotation, step-up-authentication, recovery-codes, update-email, preview		🛠

Running Keycloak in a container

Keycloak handles containerized environments such as Kubernetes or OpenShift as first-class citizens. This guide describes how to optimize and run the Keycloak container image to provide the best experience running a Keycloak container.

Creating a customized and optimized container image

The default Keycloak container image ships ready to be configured and optimized.

For the best start up of your 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.

Building your optimized Keycloak docker image

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

Dockerfile:

```
FROM quay.io/keycloak/keycloak:latest 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 quay.io/keycloak/keycloak:latest
COPY --from=builder /opt/keycloak/ /opt/keycloak/

# change these values to point to a running postgres instance
ENV KC_DB_URL=<DBURL>
ENV KC_DB_USERNAME=<DBUSERNAME>
ENV KC_DB_PASSWORD=<DBPASSWORD>
ENV KC_HOSTNAME=localhost
ENTRYPOINT ["/opt/keycloak/bin/kc.sh"]
```

The build process includes multiple stages:

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

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

```
# A example build step that downloads a JAR file from a URL and adds it to the
providers directory
RUN curl -sL <MY_PROVIDER_JAR_URL> -o /opt/keycloak/providers/myprovider.jar
```

Building the docker image

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

```
podman|docker build . -t mykeycloak
```

Starting the optimized Keycloak docker image

To start the image, run:

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

Keycloak starts in production mode, using only secured HTTPS communication, and is available on `https://localhost:8443`.

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

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

Exposing the container to a different port

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

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

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

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

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

Trying Keycloak in development mode

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

```
podman|docker run --name mykeycloak -p 8080:8080 \
        -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
        quay.io/keycloak/keycloak:latest \
        start-dev
```

Invoking this command starts the Keycloak server in development mode.

This mode should be strictly avoided in production environments because it has insecure defaults. For more information about running Keycloak in production, take a look at the Configuring Keycloak for production guide.

Running a standard 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 Keycloak image by just running the `start` command. For example:

```
podman|docker run --name mykeycloak -p 8080:8080 \
        -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
        quay.io/keycloak/keycloak:latest \
        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 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.

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.

Provide initial admin credentials when running in a container

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

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

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

Importing A Realm On Startup

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

```
podman|docker run --name keycloak_unoptimized -p 8080:8080 \
        -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
        -v /path/to/realm/data:/opt/keycloak/data/import
        quay.io/keycloak/keycloak:latest \
        start-dev --import-realm
```

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

Relevant options

	Type	**Default**	
db The database vendor. **CLI:** `--db` **Env:** `KC_DB`	dev-file, dev-mem, mariadb, mssql, mysql, oracle, postgres	dev-file	🛠

	Type	Default	
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`			

	Type	Default	
features Enables a set of one or more features. **CLI:** `--features` **Env:** `KC_FEATURES`	authorization, account2, account-api, admin-fine-grained-authz, admin-api, admin, admin2, docker, impersonation, openshift-integration, scripts, token-exchange, web-authn, client-policies, ciba, map-storage, par, declarative-user-profile, dynamic-scopes, client-secret-rotation, step-up-authentication, recovery-codes, update-email, preview		🛠
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	false	🛠
hostname Hostname for the Keycloak server. **CLI:** `--hostname` **Env:** `KC_HOSTNAME`			

	Type	**Default**	
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	
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	false	🛠

All provider configuration

authentication-sessions

infinispan

	Type	Default
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`	int	300

map

	Type	Default
spi-authentication-sessions-map-auth-sessions-limit The maximum number of concurrent authentication sessions per RootAuthenticationSession. **CLI:** `--spi-authentication-sessions-map-auth-sessions-limit` **Env:** `KC_SPI_AUTHENTICATION_SESSIONS_MAP_AUTH_SESSIONS_LIMIT`	int	300

ciba-auth-channel

ciba-http-auth-channel

	Type	Default
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`	string	none

connections-http-client

default

	Type	Default
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`	string	-1
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`	string	none

	Type	Default
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`	string	none
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`	int	none
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`	long	-1
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`	boolean	true

	Type	Default
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`	boolean	false
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`	long	-1
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`	long	900000
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`	int	64

	Type	**Default**
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`	string	none
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`	boolean	true
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`	long	5000

connections-jpa

legacy

	Type	Default
spi-connections-jpa-legacy-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-legacy-initialize-empty` **Env:** `KC_SPI_CONNECTIONS_JPA_LEGACY_INITIALIZE_EMPTY`	boolean	true
spi-connections-jpa-legacy-migration-export Path for where to write manual database initialization/migration file. **CLI:** `--spi-connections-jpa-legacy-migration-export` **Env:** `KC_SPI_CONNECTIONS_JPA_LEGACY_MIGRATION_EXPORT`	string	none
spi-connections-jpa-legacy-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-legacy-migration-strategy` **Env:** `KC_SPI_CONNECTIONS_JPA_LEGACY_MIGRATION_STRATEGY`	update, manual, validate	update

dblock

jpa

	Type	Default
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`	int	none

events-listener

email

	Type	Default
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_account_linking, client_initiated_account_linking_error, client_login, client_login_error, client_register, client_register_error, client_update, client_update_error, code_to_token, code_to_token_error, custom_required_action, custom_required_action_error, delete_account, delete_account_error, execute_action_token, execute_action_token_error, execute_actions, execute_actions_error, federated_identity_link, federated_identity_link_error, grant_consent, grant_consent_error, identity_provider_first_login, identity_provider_first_login_error, identity_provider_link_account,	none

	Type	Default
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, grant_consent, grant_consent_error, identity_provider_first_login, identity_provider_first_login_error, identity_provider_link_account,	All events

jboss-logging

	Type	Default
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	warn
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, error, fatal, info, trace, warn	debug

resource-encoding

gzip

	Type	Default
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`	string	image/png image/jpeg

sticky-session-encoder

infinispan

	Type	Default
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`	boolean	true

truststore

file

	Type	Default
spi-truststore-file-file 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`	string	none
spi-truststore-file-hostname-verification-policy The hostname verification policy. **CLI:** `--spi-truststore-file-hostname-verification-policy` **Env:** `KC_SPI_TRUSTSTORE_FILE_HOSTNAME_VERIFICATION_POLICY`	any, wildcard, strict	wildcard
spi-truststore-file-password The trust store password. **CLI:** `--spi-truststore-file-password` **Env:** `KC_SPI_TRUSTSTORE_FILE_PASSWORD`	string	none

well-known

openid-configuration

	Type	Default
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`	boolean	true
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`	string	none

Configuring TLS

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

Configuring TLS in Keycloak

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.

Providing certificates in PEM format

When you use a pair of matching certificate and private key files in PEM format, you configure 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
```

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

Providing a Java Keystore

When no keystore file is explicitly configured, but `http-enabled` is set to false, 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
```

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.

Configuring TLS protocols

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

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

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

Switching the HTTPS port

Keycloak listens for HTTPS traffic on port `8443`. To change this port, use the following command:

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

Using a truststore

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

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

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

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

Note that this trust store is targeted for authenticating clients where Keycloak is acting as a server. For configuring a trust store where Keycloak is acting as a client to external services through TLS, please consider looking at the Configuring a Truststore guide.

Setting the truststore password

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

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

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

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 the Vault Guide / Production deployment guide.

Enabling mutual TLS

Authentication using mTLS is disabled by default. To enable mTLS certificate handling when Keycloak is the server and needs to validate certificates from requests made to Keycloaks endpoints, put the appropriate certificates in Keycloaks 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 Keycloak to always ask for certificates and fail if no certificate is provided in a request. By setting the value to `request`, Keycloak will also accept requests without a certificate and only validate the correctness of a certificate if it exists.

Be aware that this is the basic certificate configuration for mTLS use cases where Keycloak acts as server. When Keycloak acts as client instead, e.g. when 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 the Configuring outgoing HTTP requests guide.

Relevant options

	Type	Default	
http-enabled Enables the HTTP listener. **CLI:** `--http-enabled` **Env:** `KC_HTTP_ENABLED`	true, false	false	
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`			

	Type	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, request, required	none	
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	
https-key-store-type The type of the key store file. If not given, the type is automatically detected based on the file name. **CLI:** `--https-key-store-type` **Env:** `KC_HTTPS_KEY_STORE_TYPE`			

	Type	Default	
https-port The used HTTPS port. **CLI:** `--https-port` **Env:** `KC_HTTPS_PORT`		8443	
https-protocols The list of protocols to explicitly enable. **CLI:** `--https-protocols` **Env:** `KC_HTTPS_PROTOCOLS`		TLSv1.3	
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`			
https-trust-store-type The type of the trust store file. If not given, the type is automatically detected based on the file name. **CLI:** `--https-trust-store-type` **Env:** `KC_HTTPS_TRUST_STORE_TYPE`			

Configuring a Truststore

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

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

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

Configuring the Keycloak Truststore

You can add your truststore configuration by entering this command:

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

The following are possible configuration options for this setting:

file

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

password

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

hostname-verification-policy

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

- `ANY` means that the hostname is not verified.
- `WILDCARD` allows wildcards in subdomain names, such as *.foo.com.
- When using `STRICT`, the Common Name (CN) must match the hostname exactly.

Example of a truststore configuration

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

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

Configuring distributed caches

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.

Enable distributed caching

When you start Keycloak in production mode, by using the `start` command, caching is enabled and all Keycloak nodes in your network are discovered.

By default, caches are using a `UDP` transport stack so that nodes are discovered using IP multicast transport based on UDP. For most production environments, there are better discovery alternatives to UDP available. 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 guide.

To explicitly enable distributed infinispan caching, enter this command:

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

When you start Keycloak in development mode, by using the `start-dev` command, 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.

Configuring caches

Keycloak provides a cache configuration file with sensible defaults located at `conf/cache-ispn.xml`.

The cache configuration is a regular Infinispan configuration file.

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

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

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

Cache types and defaults

Local caches

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

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

Authentication sessions

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

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

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

User sessions

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

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

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

- sessions
- clientSessions

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

As an OpenID Connect Provider, the server is also capable of authenticating users and issuing offline tokens. Similarly to regular user and client sessions, when an offline token is issued by the server upon successful authentication, the server also creates a user and client sessions. However, due to the nature of offline tokens, offline sessions are handled differently as they are long-lived and should survive a complete cluster shutdown. Because of that, they are also persisted to the database.

The following caches are used to store offline sessions:

- offlineSessions
- offlineClientSessions

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

Password brute force detection

The `loginFailures` distributed cache is used to track data about failed login attempts. This cache is needed for the Brute Force Protection feature to work in a multi-node 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.

Configuring caches for availability

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

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

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

Specify your own cache configuration file

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

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

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

Transport stacks

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

- tcp
- udp
- kubernetes
- ec2
- azure
- google

To apply a specific cache stack, enter this command:

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

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

Available transport stacks

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

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

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

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

Additional transport stacks

The following table shows transport stacks that are supported by Keycloak, but need some extra steps to work. Note that *none* of these stacks are Kubernetes / OpenShift stacks, so no need exists to enable the "google" stack if you want to run Keycloak on top of the Google Kubernetes engine. In that case, use the `kubernetes` stack. Instead, when you have a distributed cache setup running on

AWS EC2 instances, you would need to set the stack to `ec2`, because ec2 does not support a default discovery mechanism such as `UDP`.

Stack name	Transport protocol	Discovery
ec2	TCP	NATIVE_S3_PING
google	TCP	GOOGLE_PING2
azure	TCP	AZURE_PING

Cloud vendor specific stacks have additional dependencies for Keycloak. For more information and links to repositories with these dependencies, see the Infinispan documentation.

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

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

Custom transport stacks

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

For more details, see Using inline JGroups stacks.

defining a custom transport stack

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

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

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

Securing cache communication

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

Relevant options

	Type	Default	
cache Defines the cache mechanism for high-availability. By default, a 'ispn' cache is used to create a cluster between multiple server nodes. A 'local' cache disables clustering and is intended for development and testing purposes. **CLI:** `--cache` **Env:** `KC_CACHE`	ispn, local	ispn	🛠
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-stack Define the default stack to use for cluster communication and node discovery. This option only takes effect if 'cache' is set to 'ispn'. Default: udp. **CLI:** `--cache-stack` **Env:** `KC_CACHE_STACK`	tcp, udp, kubernetes, ec2, azure, google		🛠

Configuring logging

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

- root
  - console (*default*)
  - file
  - GELF

Logging configuration

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

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	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).

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`.

Configuring category-specific log levels

You can set different log levels for specific areas in 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.

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

Console log handler

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

Configuring the console log format

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

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

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

The format string supports the symbols in the following table:

Symbol	Summary	Description
%%	%	Renders a simple % character.
%c	Category	Renders the log category name.
%d{xxx}	Date	Renders a date with the given date format string.String syntax defined by `java.text.SimpleDateFormat`
%e	Exception	Renders a thrown exception.
%h	Hostname	Renders the simple host name.
%H	Qualified host name	Renders the fully qualified hostname, which may be the same as the simple host name, depending on the OS configuration.
%i	Process ID	Renders the current process PID.
%m	Full Message	Renders the log message and an exception, if thrown.
%n	Newline	Renders the platform-specific line separator string.
%N	Process name	Renders the name of the current process.
%p	Level	Renders the log level of the message.
%r	Relative time	Render the time in milliseconds since the start of the application log.

%s	Simple message	Renders only the log message without exception trace.
%t	Thread name	Renders the thread name.
%t{id}	Thread ID	Render the thread ID.
%z{<zone name>}	Timezone	Set the time zone of log output to <zone name>.
%L	Line number	Render the line number of the log message.

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-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]`.

Configuring JSON or plain console logging

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

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

Example Log Message

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

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

To use unstructured logging, enter the following command:

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

Example Log Message:

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

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>
```

File logging

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

Enable file logging

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

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

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

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, 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>
   ```

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>
```

Please see the Configuring the console log format section in this guide for more information and a table of the available pattern configuration.

Centralized logging using GELF

Keycloak can send logs to a centralized log management system such as the following:

- Graylog
- Logstash, inside the Elasticsearch, Logstash, Kibana (ELK) logging stack
- Fluentd, inside the Elasticsearch, Fluentd, Kibana (EFK) logging stack

Keycloak uses the Quarkus Logging GELF extension to support these environments.

Enabling the GELF handler

To enable logging using GELF, add it to the list of activated log handlers.

Example:

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

Configuring the GELF handler

To configure the Host and Port of your centralized logging system, enter the following command and substitute the values with your specific values: .Host and port of the GELF server:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-host=myhost --log-gelf-port=12345
```

When the GELF handler is enabled, the host is using `localhost` as host value and UDP for communication. To use TCP instead of UDP, prefix the host value with `tcp:`. The Default port is `12201`.

Include or exclude Stacktraces

Keycloak includes the complete Stacktrace inside the `StackTrace` field. To exclude this field, enter the following command:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-include-stack-trace=false
```

Configure the timestamp format

You can change the format of the `timestamp` field. For example, you can include the date and time down to seconds by entering the following command:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-timestamp-format="'yyyy-MM-dd
HH:mm:ss'"
```

Alternatively, you could use the config file to avoid escaping:

```
log-gelf-timestamp-format=yyyy-MM-dd HH:mm:ss
```

The default timestamp format is `yyyy-MM-dd HH:mm:ss,SSS`. You can use the available SimpleDateFormat patterns to define an appropriate timestamp.

Configure the facility

The `facility` field is an indicator of the process or program that is the source of log messages. The default value is `keycloak`. To set this field to your preferred identifier, enter the following command:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-facility=MyKeycloak
```

To use the CLI to configure Keycloak and use whitespaces for `facility`, enter the following command:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-facility="'my keycloak'"
```

Alternatively, you could use the config file to avoid escaping:

```
log-gelf-facility=my keycloak
```

Configure the default message size

To change the default message size of 8kb (8192 bytes) of GELF log messages for Keycloak, enter the following command:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-max-message-size=16384
```

The maximum size of one GELF log message is set in Bytes. The preceding example increases the size to 16kb. When messages exceed the maximum size, GELF submits the message in multiple chunks.

Configure sending of message parameters

Keycloak includes message parameters of the occurred log event. These fields appear in the output as `MessageParam0`, `MessageParam1`, and so on, depending on the parameter length. To switch off this behavior, enter the following command:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-include-message-parameters=false
```

Configure sending of source code location

Keycloak includes the `SourceClassName`, `SourceMethodName` and `SourceSimpleClassName` fields in the GELF log messages. These fields provide detail on the location of an exception that occurred. To stop sending these fields, enter the following command:

```
bin/kc.[sh|bat] start --log=console,gelf --log-gelf-include-location=false
```

Example: Send logs to Graylog

The following example shows how to send Keycloak logs to the Graylog centralized logging stack. This example assumes you have a container tool such as docker installed to start the `compose.yml`.

Starting the Graylog stack

The composed stack consists of:

- Graylog
- ElasticSearch
- MongoDB

```
version: '3.8'

services:
  elasticsearch:
    image: docker.io/elastic/elasticsearch:7.10.2
    ports:
      - "9200:9200"
    environment:
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
      discovery.type: "single-node"
    networks:
      - graylog

  mongo:
    image: mongo:4.4
    networks:
      - graylog

  graylog:
    image: graylog/graylog:4.3.3
    ports:
      - "9000:9000"
      - "12201:12201/udp"
      - "1514:1514"
    environment:
      GRAYLOG_HTTP_EXTERNAL_URI: "http://127.0.0.1:9000/"
      # CHANGE ME (must be at least 16 characters)!
      GRAYLOG_PASSWORD_SECRET: "forpasswordencryption"
      # Password: admin
      GRAYLOG_ROOT_PASSWORD_SHA2:
"8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918"
    networks:
      - graylog
    depends_on:
      - elasticsearch
      - mongo

networks:
  graylog:
    driver: bridge
```

Copy and save the example locally into a `compose.yml` file and enter this command:

```
docker compose up -d
```

After a few seconds, the Stack is ready to serve requests.

Creating a Graylog UDP Input

Once the stack is running, you need to create a UDP Input Graylog listens to. You can create it from the Graylog web UI (System → Input → Select GELF UDP) available at http://localhost:9000 or using the API:

This `curl` example creates a new GELF UDP Input using the API and the default Graylog login credentials (admin/admin).

```
curl -H "Content-Type: application/json" -H "Authorization: Basic YWRtaW46YWRtaW4=" -H
"X-Requested-By: curl" -X POST -v -d \
'{"title":"udp
input","configuration":{"recv_buffer_size":262144,"bind_address":"0.0.0.0","port":1220
1,"decompress_size_limit":8388608},"type":"org.graylog2.inputs.gelf.udp.GELFUDPInput",
"global":true}' \
http://localhost:9000/api/system/inputs
```

If the stack is still in the bootstrap phase, you receive a response containing `* Empty reply from server`. A successful response includes `HTTP/1.1 201 Created` to indicate that the UDP input is created.

Configure Keycloak to send logs using GELF

Keycloak needs to be configured to send logs using GELF. The appropriate configuration can be seen in the following keycloak.conf example. The example includes the `log-gelf-host` and `log-gelf-`

port values. These are optional values that are included for illustration purposes; default values exist.

Keycloak GELF Configuration

```
log=console,gelf
log-gelf-host=localhost
log-gelf-port=12201
```

Graylog: See the results

1. Open your web browser, go to http://localhost:9000.
2. Log in to the Graylog web UI using the administrator credentials (admin/admin).
3. Go to Streams, All Messages.
4. Start updating the stream by pressing the Play button in the upper right corner.
5. Start Keycloak using `start` or `start-dev` and your GELF config.

After a few seconds, Keycloak messages appear in the Graylog dashboard.

Example Setup using the ELK Stack

The following example shows how to send Keycloak logs to the ELK centralized logging stack. It assumes you have a container tool such as docker installed to start the `compose.yml`.

Enable the logstash GELF plugin and create a pipeline

Logstash uses an input plugin that understands and parses the GELF format. To activate this plugin when you are starting the ELK stack later on, create a directory `pipelines` and a file `gelf.conf` located in this directory. Then create an empty `compose.yml` in the parent directory.

File Structure:

```
/ELK
  - compose.yml
  - pipelines/
    - gelf.conf
```

Add the following contents to `pipelines/gelf.conf` and save it:

```
input {
  gelf {
    port => 12201
  }
}
output {
  stdout {}
  elasticsearch {
    hosts => ["http://elasticsearch:9200"]
  }
}
```

This file activates and configures the logstash GELF plugin and points it to the right elasticsearch instance.

Starting the ELK stack

The composed stack consists of:

- ElasticSearch
- Logstash
- Kibana

Copy the following content to your `compose.yml` file:

```
# Launch Elasticsearch
version: '3.8'

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch-oss:6.8.2
    ports:
      - "9200:9200"
      - "9300:9300"
    environment:
      ES_JAVA_OPTS: "-Xms512m -Xmx512m"
    networks:
      - elk

  logstash:
    image: docker.elastic.co/logstash/logstash-oss:6.8.2
    volumes:
      - source: ./pipelines #the source dir gelf.conf resides
        target: /usr/share/logstash/pipeline
        type: bind
    ports:
      - "12201:12201/udp"
      - "5000:5000"
      - "9600:9600"
    networks:
      - elk
    depends_on:
      - elasticsearch

  kibana:
    image: docker.elastic.co/kibana/kibana-oss:6.8.2
    ports:
      - "5601:5601"
    networks:
      - elk
    depends_on:
      - elasticsearch

networks:
  elk:
    driver: bridge
```

Start the stack by entering the following command:

```
docker compose up -d
```

After a few seconds the Stack should be ready to serve requests.

Configuring Keycloak to send logs using GELF

Keycloak needs to be configured to send logs using GELF. The appropriate configuration can be seen in the following keycloak.conf example. This example includes the `log-gelf-host` and `log-gelf-port` values. These are optional values, which are included for illustration purposes; default values exist.

Keycloak Gelf Configuration

```
log=console,gelf
log-gelf-host=localhost
log-gelf-port=12201
```

With this configuration applied, start keycloak using `start-dev` or `start`.

Kibana: See the results

Open http://localhost:5601 to reach the Kibana dashboard. The exact configuration of a good monitoring dashboard is out of scope for this guide. To find out if logs sent by Keycloak are delivered to Kibana, open the Dev Tools and execute the default `match_all` query. The logs should appear in the result field.

Configure a different log level for the GELF logger

To keep log storage costs and verbosity low, it is often wanted to only store a subset of the verbose application logs inside a centralized log management system. To configure Keycloak to use a different log level for the logs you want to ingest, use the following configuration:

```
log=console,gelf
log-gelf-level=<desired-log-level>
...
```

Example

To only see occurred log levels of warn and above in your centralized logging stack, but still see INFO level logs on the applications console logs, use the following configuration:

```
log=console,gelf
log-level=INFO
log-gelf-level=warn
...
```

Looking at your ingested logs, you will see only messages of level warn or above will appear.

Keep in mind that `--log-level` is setting the leading log level, so for example when you invoke the following command:

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

nothing below the error level will be sent to your logging stack. That means that even GELF in this example will receive only error level log messages.

Configure additional key values

Currently, the Keycloak configuration does not support partly dynamic configuration keys, as they are used in quarkus properties. For example, they are used when defining `quarkus.log.handler.gelf.additional-field.<my-name>.value`.

To add user-defined fields, you can provide these fields through a quarkus.properties file. Refer to the Configuring Keycloak guide and the *Using unsupported server options* section.

Relevant options

	Type	**Default**	
log-console-color Enable or disable colors when logging to console. **CLI:** `--log-console-color` **Env:** `KC_LOG_CONSOLE_COLOR`	true, false	false	

	Type	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`		%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n	
log-console-output Set the log output to JSON or default (plain) unstructured logging. **CLI:** `--log-console-output` **Env:** `KC_LOG_CONSOLE_OUTPUT`	default, json	default	
log-file Set the log file path and filename. **CLI:** `--log-file` **Env:** `KC_LOG_FILE`		data/log/keycloak.log	
log-file-format Set a format specific to file log entries. **CLI:** `--log-file-format` **Env:** `KC_LOG_FILE_FORMAT`		%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n	
log-file-output Set the log output to JSON or default (plain) unstructured logging. **CLI:** `--log-file-output` **Env:** `KC_LOG_FILE_OUTPUT`	default, json	default	

	Type	Default	
log-gelf-facility The facility (name of the process) that sends the message. **CLI:** `--log-gelf-facility` **Env:** `KC_LOG_GELF_FACILITY`		keycloak	
log-gelf-host Hostname of the Logstash or Graylog Host. By default UDP is used, prefix the host with 'tcp:' to switch to TCP. Example: 'tcp:localhost' **CLI:** `--log-gelf-host` **Env:** `KC_LOG_GELF_HOST`		localhost	
log-gelf-include-location Include source code location. **CLI:** `--log-gelf-include-location` **Env:** `KC_LOG_GELF_INCLUDE_LOCATION`	true, false	true	
log-gelf-include-message-parameters Include message parameters from the log event. **CLI:** `--log-gelf-include-message-parameters` **Env:** `KC_LOG_GELF_INCLUDE_MESSAGE_PARAMETERS`	true, false	true	
log-gelf-include-stack-trace If set to true, occuring stack traces are included in the 'StackTrace' field in the GELF output. **CLI:** `--log-gelf-include-stack-trace` **Env:** `KC_LOG_GELF_INCLUDE_STACK_TRACE`	true, false	true	

	Type	Default	
log-gelf-level The log level specifying which message levels will be logged by the GELF logger. Message levels lower than this value will be discarded. **CLI:** `--log-gelf-level` **Env:** `KC_LOG_GELF_LEVEL`		INFO	
log-gelf-max-message-size Maximum message size (in bytes). If the message size is exceeded, GELF will submit the message in multiple chunks. **CLI:** `--log-gelf-max-message-size` **Env:** `KC_LOG_GELF_MAX_MESSAGE_SIZE`		8192	
log-gelf-port The port the Logstash or Graylog Host is called on. **CLI:** `--log-gelf-port` **Env:** `KC_LOG_GELF_PORT`		12201	
log-gelf-timestamp-format Set the format for the GELF timestamp field. Uses Java SimpleDateFormat pattern. **CLI:** `--log-gelf-timestamp-format` **Env:** `KC_LOG_GELF_TIMESTAMP_FORMAT`		yyyy-MM-dd HH:mm:ss,SSS	
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	

Configuring outgoing HTTP requests

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

Client Configuration Command

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

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

The following are the command options:

establish-connection-timeout-millis

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

socket-timeout-millis

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

connection-pool-size

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

max-pooled-per-route

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

connection-ttl-millis

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

max-connection-idle-time-millis

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

disable-cookies

Enable or disable caching of cookies. Default: true.

client-keystore

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

client-keystore-password

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

client-key-password

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

proxy-mappings

Specify proxy configurations for outgoing HTTP requests. For more details, see 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.

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. 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.

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 Keycloak. A proxy-mapping consists of a regex-based hostname pattern and a proxy-uri, using the format `hostname-pattern;proxy-uri`.

For example, consider the following regex:

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

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

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

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

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

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

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

Example of regular expressions for proxy-mapping:

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

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

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

In this example, the following occurs:

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

Outgoing HTTPS request truststore

Please take a look at the Configuring a Truststore guide about how to configure a Keycloak Truststore so that Keycloak is able to perform outgoing requests using TLS.

Configuring providers

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

After reading this guide, 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.

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
```

Setting a provider configuration option

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

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

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

Configuring a default provider

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

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

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

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

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

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`.

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.

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.

References

- Configuring Keycloak
- Server Developer Documentation

Configuring the database

This guide explains how to configure the Keycloak server to store data in a relational database.

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	Tested Version
mariadb	10
mssql	2016
mysql	8
oracle	12c
postgres	10

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.

Configuring a database

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

1. Build a server image for your database. For example, for a PostgreSQL database, enter this command:

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

2. Start the server and set the options for the database host and credentials by entering this command:

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

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

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

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-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.

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.

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`.

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

Unicode support for a Microsoft SQL Server database

Unicode characters are supported only for the special fields for a Microsoft SQL Server database. The JDBC driver and database require no special settings.

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`

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';
   ```

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
```

Using Database Vendors without XA transaction support

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

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

Keycloak automatically chooses the appropriate JDBC driver for your vendor.

Setting JPA provider configuration option for migrationStrategy

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

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

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

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

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

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

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

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

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

Relevant options

	Type	Default	
db The database vendor. **CLI:** `--db` **Env:** `KC_DB`	dev-file, dev-mem, mariadb, mssql, mysql, oracle, postgres	dev-file	🛠
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	
db-pool-min-size The minimal size of the connection pool. **CLI:** `--db-pool-min-size` **Env:** `KC_DB_POOL_MIN_SIZE`			

	Type	Default	
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`			
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`			

	Type	**Default**	
db-url-properties Sets the properties of the default JDBC URL of the chosen vendor. 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`			
transaction-xa-enabled If set to false, Keycloak uses a non-XA datasource in case the database does not support XA transactions. **CLI:** `--transaction-xa-enabled` **Env:** `KC_TRANSACTION_XA_ENABLED`	true, false	true	🛠

Configuring the hostname

Server Endpoints

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

- Frontend
- Backend
- Administration Console

The base URL for each group has an important impact on how tokens are issued and validated, on how links are created for actions that require the user to be redirected to 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`.

Frontend

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

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

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

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

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

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

- `https` scheme is used unless you set `hostname-strict-https=false`
- Use the standard HTTP ports (e.g.: `80` and `443`) if a `proxy` is set or use the port you set to the `hostname-port` option

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

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

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

Backend

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

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

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

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

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

Administration Console

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

By default, the URLs for the administration console are also based on the incoming request. However, you can set a specific host or base URL if you want to restrict access to the administration console using a specific URL. Similarly to how you set the frontend URLs, you can use the `hostname-admin` and `hostname-admin-url` options to achieve that.

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

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

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

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

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

To reduce attack surface, the administration endpoints for Keycloak and the Admin Console should not be publicly accessible. Therefore, you can secure them by using a reverse proxy. For more information about which paths to expose using a reverse proxy, see the Using a reverse proxy Guide.

Example Scenarios

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

Note that the `start` command requires setting up TLS. The corresponding options are not shown for example purposes. For more details, see Configuring TLS guide.

Exposing the server behind a TLS termination proxy

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

Configuration:

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

Exposing the server without a proxy

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

Keycloak configuration:

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

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

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

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

Keycloak configuration:

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

Exposing the server using a port other than the default ports

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

Keycloak configuration:

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

Relevant options

	Type	Default	
hostname Hostname for the Keycloak server. **CLI:** `--hostname` **Env:** `KC_HOSTNAME`			
hostname-admin The hostname for accessing the administration console. Use this option if you are exposing the administration console using a hostname other than the value set to the 'hostname' option. **CLI:** `--hostname-admin` **Env:** `KC_HOSTNAME_ADMIN`			
hostname-admin-url Set the base URL for accessing the administration console, including scheme, host, port and path **CLI:** `--hostname-admin-url` **Env:** `KC_HOSTNAME_ADMIN_URL`			
hostname-path This should be set if proxy uses a different context-path for Keycloak. **CLI:** `--hostname-path` **Env:** `KC_HOSTNAME_PATH`			

	Type	Default	
hostname-port The port used by the proxy when exposing the hostname. Set this option if the proxy uses a port other than the default HTTP and HTTPS ports. **CLI:** `--hostname-port` **Env:** `KC_HOSTNAME_PORT`		-1	
hostname-strict Disables dynamically resolving the hostname from request headers. Should always be set to true in production, unless proxy verifies the Host header. **CLI:** `--hostname-strict` **Env:** `KC_HOSTNAME_STRICT`	true, false	true	
hostname-strict-backchannel By default backchannel URLs are dynamically resolved from request headers to allow internal and external applications. If all applications use the public URL this option should be enabled. **CLI:** `--hostname-strict-backchannel` **Env:** `KC_HOSTNAME_STRICT_BACKCHANNEL`	true, false	false	
hostname-url Set the base URL for frontend URLs, including scheme, host, port and path. **CLI:** `--hostname-url` **Env:** `KC_HOSTNAME_URL`			

	Type	Default	
proxy The proxy address forwarding mode if the server is behind a reverse proxy. **CLI:** `--proxy` **Env:** `KC_PROXY`	none, edge, reencrypt, passthrough	none	

Enabling Keycloak Health checks

Keycloak has built in support for health checks. This guide describes how to enable and use the Keycloak health checks.

Keycloak Health checks

Keycloak exposed health endpoints are three:

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

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

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

Enabling the health checks

Is possible to enable the health checks using the build time option `health-enabled`:

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

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

Available Checks

The table below shows the available checks.

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

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

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

Relevant options

	Type	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	false	🛠

Importing and Exporting Realms

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

Exporting a Realm to a Directory

To export a realm, you can use the `export` command. Your 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.

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
```

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.

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
```

Importing a Realm from a Directory

To import a realm, you can use the `import` command. Your 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.

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>
```

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. Each file in this directory should contain a single realm configuration. Only regular files using the `.json` extension are read from this directory, sub-directories are ignored.

> For the published containers, the import directory is `/opt/keycloak/data/import`

If a realm already exists in the server, the import operation is skipped.

Using Environment Variables within the Realm Configuration Files

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

Realm configuration using placeholders

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

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

Using Kubernetes secrets

Keycloak supports a file-based vault implementation for Kubernetes/OpenShift secrets. Mount Kubernetes secrets into the Keycloak Container, and the data fields will be available in the mounted folder with a flat-file structure.

Available integrations

You can use Kubernetes/OpenShift secrets for the following purposes:

- 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

Enabling the vault

Enable the file based vault by building Keycloak using the following build option:

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

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
```

Realm-specific secret files

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

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

Using underscores in the Name

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

Example

- Realm Name: `sso_realm`
- Desired Name: `ldap_credential`

* Resulting file Name:

```
sso__realm_ldap__credential
```

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

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.

Relevant options

	Type	Default	
vault Enables a vault provider. **CLI:** `--vault` **Env:** `KC_VAULT`	file, hashicorp		🛠
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`			

Using a reverse proxy

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

Proxy modes

The following proxy modes are available:

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

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

passthrough
: Enables communication through HTTP or HTTPS between the proxy and Keycloak. This mode is suitable for deployments where the reverse proxy is not terminating TLS. The proxy instead is forwarding requests to the Keycloak server so that secure connections between the server and clients are based on the keys and certificates used by the Keycloak server.

Configure the proxy mode in Keycloak

To select the proxy mode, enter this command:

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

Configure the reverse proxy

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

When you have a reverse proxy or a load balancer in front of Keycloak, this might not be the case, so please make sure your reverse proxy is configured correctly by performing these actions:

- Set the X-Forwarded-For, X-Forwarded-Proto, and X-Forwarded-Host HTTP headers.

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

Take extra precautions to ensure that the X-Forwarded-For header is set by your reverse proxy. If this header is incorrectly configured, rogue clients can set this header and trick 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.

Trust the proxy to set hostname

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

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

Exposing the administration console

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

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

Exposed path recommendations

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

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.
/js/	-	Yes (see note below)	Access to keycloak.js needed for "internal" clients, e.g. the account console
/welcome/	-	No	No need exists to expose the welcome page after initial installation.

Keycloak Path	Reverse Proxy Path	Exposed	Reason
/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 Keycloak path.
/robots.txt	/robots.txt	Yes	Search engine rules
/metrics	-	No	Exposed metrics lead to an unnecessary attack vector.
/health	-	No	Exposed health checks lead to an unnecessary attack vector.

Note:

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

We assume you run Keycloak on the root path `/` on your reverse proxy/gateway’s public API. If not, prefix the path with your desired one.

Enabling client certificate lookup

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

The server supports some of the most 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 keycloak and verifying it in keycloak.

Configuring the NGINX provider

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

If you are using this provider, please take a look at the Configuring a Truststore guide about how to configure a Keycloak Truststore.

Relevant options

	Type	Default	
proxy The proxy address forwarding mode if the server is behind a reverse proxy. **CLI:** `--proxy` **Env:** `KC_PROXY`	none, edge, reencrypt, passthrough	none	