# Guides Keycloak Operator

Version 20.0, 2024-01-03

Welcome to the Keycloak operator guide.

# Keycloak Operator Installation

In this guide we will show how to install the Keycloak Operator in your Kubernetes or OpenShift cluster.

## OLM Installation

The recommended way to install the Keycloak Operator in Kubernetes environments is to use the Operator Lifecycle Manager (OLM).

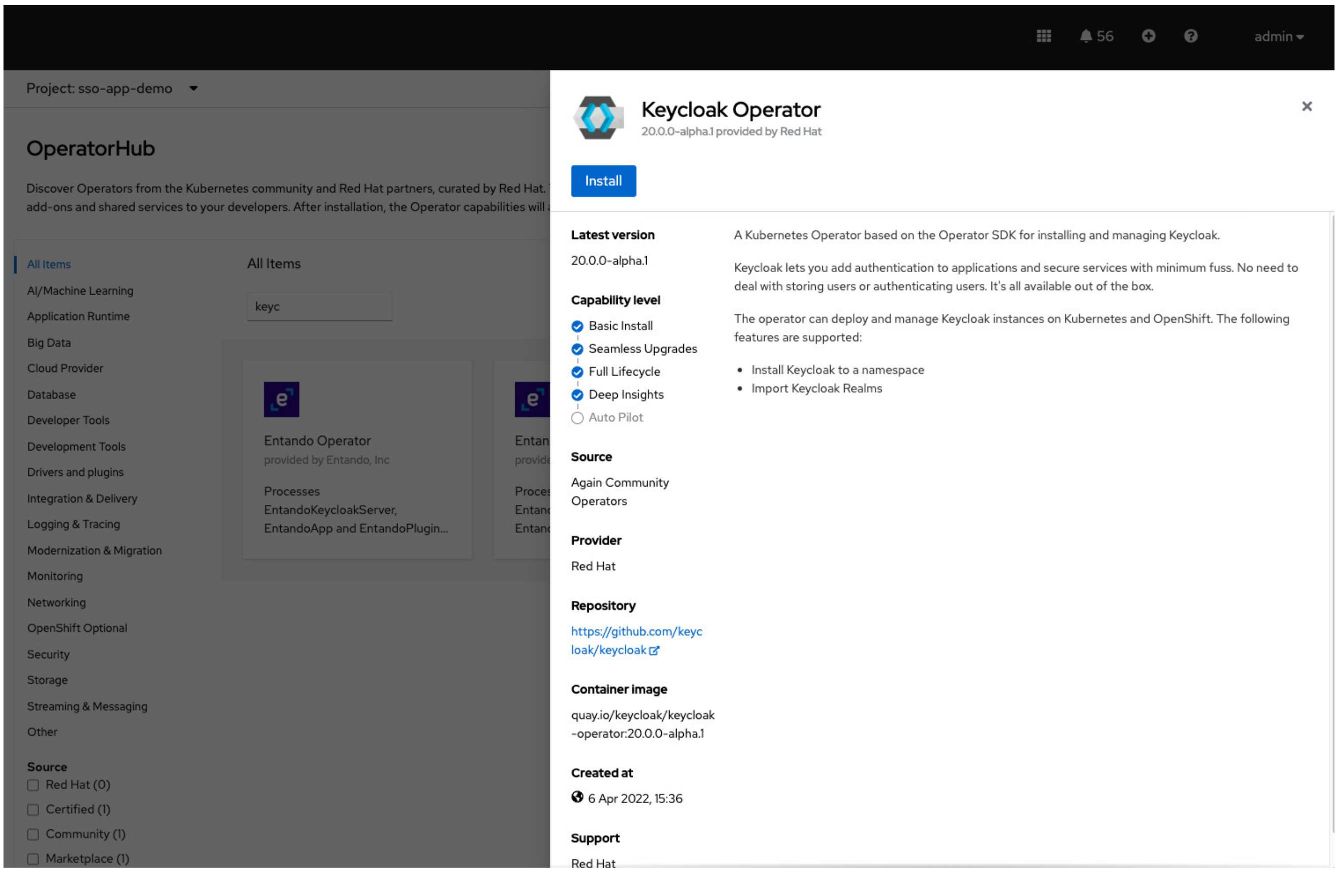
### Prerequisites

Make sure OLM is installed in your environment. For Guidance on how to install OLM, follow this guide.

The Keycloak Operator OLM package can be installed from the OLM catalog. For general instructions on how to install operators using OLM, follow the instructions on the OLM page. In the default Catalog, the Keycloak Operator is named `keycloak-operator`. Make sure to use the `candidate` channel to find the operator.

### OpenShift UI

On OpenShift, use the built-in OLM UI to install the Keycloak Operator. Navigate to `Home`, `Operators`, `OperatorHub` using the menu on the left side of the OpenShift Console. Search for "keycloak" on the search input box:



Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. add-ons and shared services to your developers. After installation, the Operator capabilities will

Entando Operator
provided by Entando, Inc
Processes EntandoKeycloakServer, EntandoApp and EntandoPlugin...

Keycloak Operator
20.0.0-alpha.1 provided by Red Hat

Latest version
20.0.0-alpha.1

Capability level
- ✓ Basic Install
- ✓ Seamless Upgrades
- ✓ Full Lifecycle
- ✓ Deep Insights
- ○ Auto Pilot

Source
Again Community Operators

Provider
Red Hat

Repository
https://github.com/keycloak/keycloak

Container image
quay.io/keycloak/keycloak-operator:20.0.0-alpha.1

Created at
6 Apr 2022, 15:36

Support
Red Hat

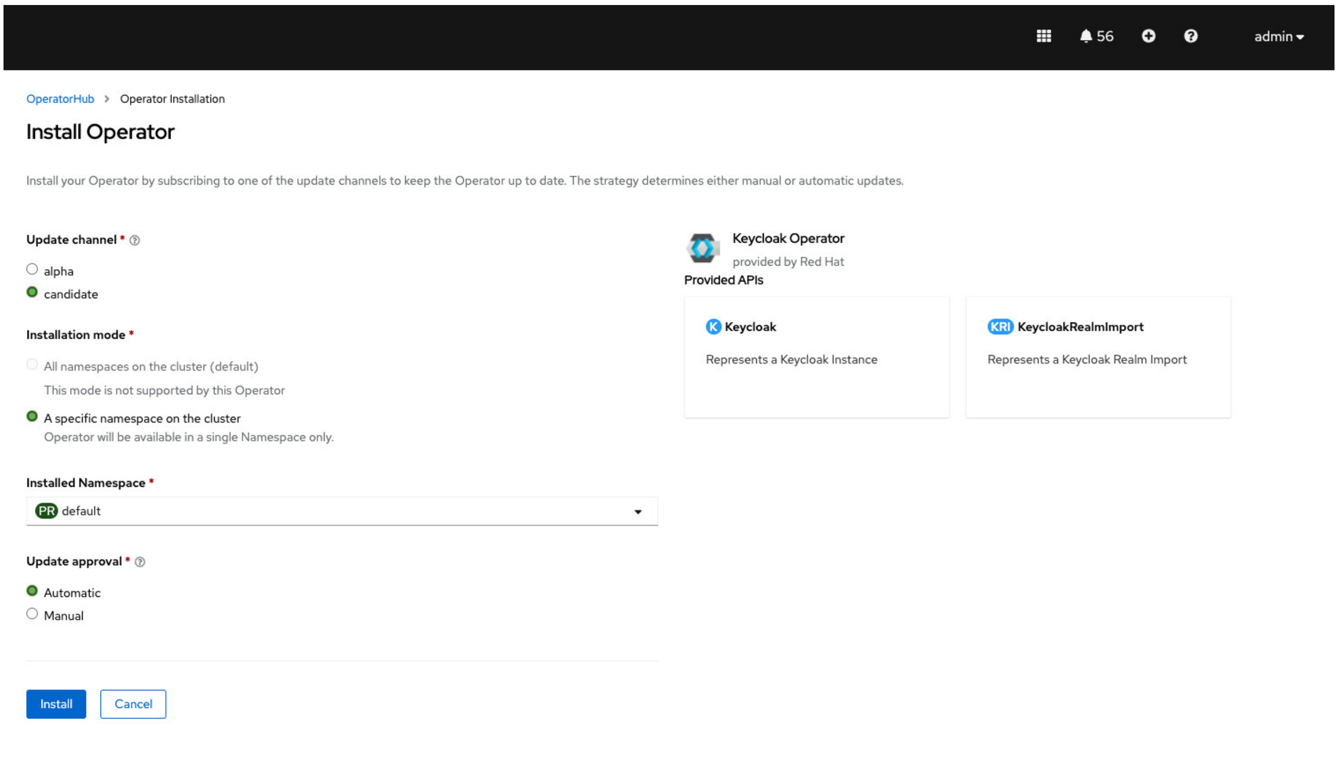
A Kubernetes Operator based on the Operator SDK for installing and managing Keycloak.

Keycloak lets you add authentication to applications and secure services with minimum fuss. No need to deal with storing users or authenticating users. It's all available out of the box.

The operator can deploy and manage Keycloak instances on Kubernetes and OpenShift. The following features are supported:

- Install Keycloak to a namespace
- Import Keycloak Realms

Select the Keycloak Operator from the list of results. After that, follow the instructions on the screen. Make sure you are installing from the `candidate` channel:



Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operator up to date. The strategy determines either manual or automatic updates.

Update channel *
alpha [✓ candidate]

Installation mode *
All namespaces on the cluster (default)
This mode is not supported by this Operator
[✓ A specific namespace on the cluster]
Operator will be available in a single Namespace only.

Installed Namespace *
PR default

Update approval *
[✓ Automatic] Manual

Keycloak Operator
provided by Red Hat
Provided APIs
Keycloak — Represents a Keycloak Instance
KeycloakRealmImport — Represents a Keycloak Realm Import

## Vanilla Kubernetes Installation

To install the operator on a vanilla Kubernetes cluster, you first need to install its CRDs by running the following commands:

```
kubectl apply -f https://raw.githubusercontent.com/keycloak/keycloak-k8s-resources/20.0.0/kubernetes/keycloaks.k8s.keycloak.org-v1.yml
kubectl apply -f https://raw.githubusercontent.com/keycloak/keycloak-k8s-resources/20.0.0/kubernetes/keycloakrealmimports.k8s.keycloak.org-v1.yml
```

After successful CRD installation, install the Keycloak Operator deployment by running the following command:

```
kubectl apply -f https://raw.githubusercontent.com/keycloak/keycloak-k8s-resources/20.0.0/kubernetes/kubernetes.yml
```

Currently the operator watches only the namespace where the operator is installed.

# Basic Keycloak Deployment

In this guide we will show how to have a basic Keycloak Deployment on Kubernetes or OpenShift using the Operator. We assume that the Operator is correctly installed and running in the cluster namespace.

## Pre-requisites

- Database
- Hostname
- TLS Certificate and associated keys

### Database

A database should be available and accessible from the cluster namespace where you want to install Keycloak. Please refer to Configuring the database for the list of supported databases. The Keycloak Operator does not manage the database and you need to provision it yourself, we suggest to verify your cloud provider offering or use a database Operator such as Crunchy.

For development purposes you can use an ephemeral Postgres pod installation. You can provision it using the following commands:

```
cat <<EOF >> example-postgres.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgresql-db
spec:
  serviceName: postgresql-db-service
  selector:
    matchLabels:
      app: postgresql-db
  replicas: 1
  template:
    metadata:
      labels:
        app: postgresql-db
    spec:
      containers:
        - name: postgresql-db
          image: postgres:latest
          env:
            - name: POSTGRES_PASSWORD
              value: testpassword
            - name: PGDATA
              value: /data/pgdata
            - name: POSTGRES_DB
              value: keycloak
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-db
spec:
  selector:
    app: postgresql-db
  type: LoadBalancer
  ports:
  - port: 5432
    targetPort: 5432
EOF
kubectl apply -f example-postgres.yaml
```

### Hostname

To have a production ready installation you need to provide the hostname that will be used to contact Keycloak. Please refer to Configuring the hostname for the available configurations.

For development purposes we will use from now on `test.keycloak.org`.

### TLS Certificate and key

Please refer to Certification Authority of choice to obtain the certificate and the key.

For development purposes you can use this command to obtain a self-signed certificate:

```
openssl req -subj '/CN=test.keycloak.org/O=Test Keycloak./C=US' -newkey rsa:2048
-nodes -keyout key.pem -x509 -days 365 -out certificate.pem
```

and you should install it in the cluster namespace as a Secret by running:

```
kubectl create secret tls example-tls-secret --cert certificate.pem --key key.pem
```

## Deploying Keycloak

To deploy Keycloak you have to create a Custom Resource (CR from now on) shaped after the Keycloak Custom Resource Definition (CRD).

We suggest you to first store the Database credentials in a separate Secret, you can do it for example by running:

```
kubectl create secret generic keycloak-db-secret \
  --from-literal=username=[your_database_username] \
  --from-literal=password=[your_database_password]
```

The Keycloak CRD allow you to customize several fields but, for a simple deployment you can use the following example:

```
cat <<EOF >> example-kc.yaml
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
  serverConfiguration:
    - name: db
      value: postgres
    - name: db-url-host
      value: postgres-db
    - name: db-username
      secret:
        name: keycloak-db-secret
        key: username
    - name: db-password
      secret:
        name: keycloak-db-secret
        key: password
  hostname: test.keycloak.org
  tlsSecret: example-tls-secret
EOF
kubectl apply -f example-kc.yaml
```

And you can check that the Keycloak instance has been provisioned in the cluster by looking at the status of the created CR:

```
kubectl get keycloaks/example-kc -o go-template='{{range
.status.conditions}}CONDITION: {{.type}}{{"\n"}}  STATUS: {{.status}}{{"\n"}}
MESSAGE: {{.message}}{{"\n"}}{{end}}'
```

When the Deployment is ready the output will look like the following:

```
CONDITION: Ready
  STATUS: true
  MESSAGE:
CONDITION: HasErrors
  STATUS: false
  MESSAGE:
CONDITION: RollingUpdate
  STATUS: false
  MESSAGE:
```

## Accessing the Keycloak Deployment

The Keycloak deployment is, by default, exposed through a basic nginx ingress and it will be accessible through the provided hostname. If the default ingress doesn't fit your use-case you can disable it by setting `disableDefaultIngress: true`:

```
cat <<EOF >> example-kc.yaml
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
    ...
    disableDefaultIngress: true
EOF
kubectl apply -f example-kc.yaml
```

And you can provide an alternative ingress resource pointing to the service `<keycloak-cr-name>-service`.

For debugging and development purposes we suggest you to directly connect to the Keycloak service using a port forward:

```
kubectl port-forward service/example-kc-service 8443:8443
```

### Accessing the Admin Console

When deploying Keycloak, the operator generates an arbitrary initial admin `username` and `password` and stores those credentials as a Kubernetes basic-auth Secret in the same namespace as the CR.

*Warning:*

Change the default admin credentials and enable MFA in Keycloak before going to production.

To fetch the initial admin credentials you have to read and decode a Kubernetes Secret. The Secret name is derived from the Keycloak CR name plus the fixed suffix `-initial-admin`. To get the username and password for the `example-kc` CR use the following command:

```
kubectl get secret example-kc-initial-admin -o jsonpath='{.data.username}' | base64
--decode
kubectl get secret example-kc-initial-admin -o jsonpath='{.data.password}' | base64
--decode
```

You can use those credentials to access the Admin Console or the Admin REST API.

# Keycloak Realm Import

The Keycloak Operator ships with the ability to automatically perform a realm import for the Keycloak Deployment.

*Note:*

If a Realm with the same name already exists in Keycloak it will not be overwritten.

*Note:*

The Realm Import CR only supports creation of new realms and doesn't update or delete those.
Changes to the realm performed directly on Keycloak are not synced back in the CR.

## Writing Realm Import CR

A Realm Import Custom Resource (CR) looks like follows:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: KeycloakRealmImport
metadata:
  name: my-realm-kc
spec:
  keycloakCRName: <name of the keycloak CR>
  realm:
    ...
```

This CR should be created in the same namespace as the Keycloak Deployment CR, defined in the field `keycloakCRName`. The `realm` field accepts a full RealmRepresentation.

The recommended way to obtain a `RealmRepresentation` is leveraging the export functionality Importing and Exporting Realms

- export the Realm to a single file
- convert the json to yaml
- copy-paste the obtained yaml as body for the `realm` key (make sure the indentation is correct)

## Applying the Realm Import CR

Use `kubectl` to create the CR in the correct cluster namespace:

```
cat <<EOF >> example-realm-import.yaml
apiVersion: k8s.keycloak.org/v2alpha1
kind: KeycloakRealmImport
```

```
metadata:
  name: my-realm-kc
spec:
  keycloakCRName: <name of the keycloak CR>
  realm:
    id: example-realm
    realm: example-realm
    displayName: ExampleRealm
    enabled: true
EOF
kubectl apply -f example-realm-import.yaml
```

You can check the status of the running import by using the following command:

```
kubectl get keycloakrealmimports/my-realm-kc -o go-template='{{range
.status.conditions}}CONDITION: {{.type}}{{"\n"}}  STATUS: {{.status}}{{"\n"}}
MESSAGE: {{.message}}{{"\n"}}{{end}}'
```

When the import has successfully completed, the output will look like the example below:

```
CONDITION: Done
  STATUS: true
  MESSAGE:
CONDITION: Started
  STATUS: false
  MESSAGE:
CONDITION: HasErrors
  STATUS: false
  MESSAGE:
```

# Advanced configuration

In this guide, you'll learn how to configure your Keycloak deployment using advanced concepts and options provided by Custom Resources (CR).

## Server Configuration details

The `serverConfiguration` field of the Keycloak CR allows to pass to Keycloak any available configuration in the form of key-value pairs. For all the available configuration options, refer to All Configuration.

The values can be expressed as plain text strings or Kubernetes Secret references. e.g:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  serverConfiguration:
    - name: db
      value: postgres # plain text value
    - name: db-url-host
      value: postgres-db # plain text value
    - name: db-username
      secret: # Secret reference
        name: keycloak-db-secret # name of the Secret
        key: username # name of the Key in the Secret
    - name: db-password
      secret: # secret reference
        name: keycloak-db-secret # name of the Secret
        key: password # name of the Key in the Secret
```

## Secret References

A Secret Reference can be either a value in `serverConfiguration` or the `tlsSecret`.

When specifying a Secret Reference, you have to make sure that a Secret containing the referenced keys is present in the same namespace as the CR referencing it. Along with the Keycloak Server Deployment, the operator adds special labels to the referenced Secrets in order to watch for changes.

When a referenced Secret is modified, the operator automatically performs a rolling restart of the Keycloak Deployment to pick up the changes.

## Unsupported features

The `unsupported` field of the CR contains highly experimental configuration options that are not completely tested and supported.

### Pod Template

Pod Template is a raw API representation that is used for the Kubernetes Deployment Template. This field is intended to be used as a temporary workaround if there is no officially supported field at the top level of the CR to cover your use-case. Please consider opening an issue on GitHub to help us make the experience better.

The operator will merge the fields of the provided template with the values generated by the operator for the specific Deployment. Using this feature, you have access to a high level of customizations, but there are no guarantees that the Deployment will work as expected.

As an example you can inject labels, annotations, or even volumes and volume mounts:

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

## Disabling required CR fields

By default, the Keycloak operator is designed to provide you with the best production-ready Deployment of Keycloak with security in mind. Although, for development purposes, you can still disable key security features.

Specifically, you can disable the required fields with a special value `INSECURE-DISABLE`:

```
apiVersion: k8s.keycloak.org/v2alpha1
```

```
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  hostname: INSECURE-DISABLE
  tlsSecret: INSECURE-DISABLE
```

# Using custom Keycloak images

## Keycloak custom image with Operator

The Keycloak Custom Resource (CR) give you the possibility to specify a custom container image for the Keycloak server.

*Note:*

To ensure full compatibility of Operator and Operand, make sure that the version of Keycloak release used in the custom image is aligned with the version of the operator.

### Best Practice

When using the default Keycloak image the server will perform a costly re-augmentation every time a Pod starts. This can be avoided by providing a Custom Image where the augmentation already happened during the build time of the image.

A custom image additionally allows to specify Keycloak's "build-time" configurations and extensions during the build of the container.

For instructions on how to build such image refer to Running Keycloak in containers.

### Providing Custom Keycloak image

To provide a custom image you have to define the `image` field in the Keycloak CR, for example:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
  image: quay.io/my-company/my-keycloak:latest
  hostname: example.com
  tlsSecret: example-tls-secret
```

*Note:*

Using custom images, every build time configuration passed through the `serverConfiguration` key will be ignored.