



## Red Hat Integration 2022.q3

# Installing and deploying Service Registry on OpenShift

Installing and deploying Service Registry 2.3



# Red Hat Integration 2022.q3 Installing and deploying Service Registry on OpenShift

---

Installing and deploying Service Registry 2.3

## Legal Notice

Copyright © 2022 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

This guide explains how to install and deploy Service Registry on OpenShift with registry data storage options in AMQ Streams or PostgreSQL database. This guide also shows how to secure, configure, and manage a Service Registry, and provides reference information about the Service Registry Operator.

## Table of Contents

<b>PREFACE</b> .....	<b>4</b>
MAKING OPEN SOURCE MORE INCLUSIVE .....	4
<b>CHAPTER 1. SERVICE REGISTRY OPERATOR QUICKSTART</b> .....	<b>5</b>
1.1. QUICKSTART SERVICE REGISTRY OPERATOR INSTALLATION .....	5
1.2. QUICKSTART SERVICE REGISTRY DEPLOYMENT .....	5
<b>CHAPTER 2. INSTALLING SERVICE REGISTRY ON OPENSIFT</b> .....	<b>7</b>
2.1. INSTALLING SERVICE REGISTRY FROM THE OPENSIFT OPERATORHUB .....	7
<b>CHAPTER 3. DEPLOYING SERVICE REGISTRY STORAGE IN AMQ STREAMS</b> .....	<b>9</b>
3.1. INSTALLING AMQ STREAMS FROM THE OPENSIFT OPERATORHUB .....	9
3.2. CONFIGURING SERVICE REGISTRY WITH KAFKA STORAGE ON OPENSIFT .....	10
3.3. CONFIGURING KAFKA STORAGE WITH TLS SECURITY .....	12
3.4. CONFIGURING KAFKA STORAGE WITH SCRAM SECURITY .....	15
3.5. CONFIGURING OAUTH AUTHENTICATION FOR KAFKA STORAGE .....	18
<b>CHAPTER 4. DEPLOYING SERVICE REGISTRY STORAGE IN A POSTGRESQL DATABASE</b> .....	<b>20</b>
4.1. INSTALLING A POSTGRESQL DATABASE FROM THE OPENSIFT OPERATORHUB .....	20
4.2. CONFIGURING SERVICE REGISTRY WITH POSTGRESQL DATABASE STORAGE ON OPENSIFT .....	21
4.3. BACKING UP SERVICE REGISTRY POSTGRESQL STORAGE .....	22
4.4. RESTORING SERVICE REGISTRY POSTGRESQL STORAGE .....	23
<b>CHAPTER 5. SECURING A SERVICE REGISTRY DEPLOYMENT</b> .....	<b>24</b>
5.1. SECURING SERVICE REGISTRY USING THE RED HAT SINGLE SIGN-ON OPERATOR .....	24
5.2. CONFIGURING SERVICE REGISTRY AUTHENTICATION AND AUTHORIZATION WITH RED HAT SINGLE SIGN-ON .....	28
5.3. SERVICE REGISTRY AUTHENTICATION AND AUTHORIZATION CONFIGURATION OPTIONS .....	31
Service Registry authentication using OpenID Connect with Red Hat Single Sign-On .....	32
Service Registry authentication using HTTP basic .....	32
Service Registry role-based authorization .....	33
Use roles assigned in Red Hat Single Sign-On .....	33
Manage roles directly in Service Registry .....	34
Service Registry admin-override configuration .....	34
Service Registry owner-only authorization .....	35
Service Registry authenticated read access .....	35
Service Registry anonymous read-only access .....	36
5.4. CONFIGURING AN HTTPS CONNECTION TO SERVICE REGISTRY FROM INSIDE THE OPENSIFT CLUSTER .....	36
5.5. CONFIGURING AN HTTPS CONNECTION TO SERVICE REGISTRY FROM OUTSIDE THE OPENSIFT CLUSTER .....	39
<b>CHAPTER 6. CONFIGURING AND MANAGING A SERVICE REGISTRY DEPLOYMENT</b> .....	<b>41</b>
6.1. CONFIGURING SERVICE REGISTRY HEALTH CHECKS ON OPENSIFT .....	41
6.2. ENVIRONMENT VARIABLES FOR SERVICE REGISTRY HEALTH CHECKS .....	42
Liveness environment variables .....	42
Readiness environment variables .....	43
6.3. MANAGING SERVICE REGISTRY ENVIRONMENT VARIABLES .....	44
6.4. CONFIGURING THE SERVICE REGISTRY WEB CONSOLE .....	45
Configuring the web console deployment environment .....	45
Configuring the web console in read-only mode .....	45
6.5. CONFIGURING SERVICE REGISTRY LOGGING .....	46
6.6. CONFIGURING SERVICE REGISTRY EVENT SOURCING .....	46

Configuring Service Registry event sourcing using HTTP	47
Configuring Service Registry event sourcing using Apache Kafka	47
<b>CHAPTER 7. SERVICE REGISTRY OPERATOR CONFIGURATION REFERENCE .....</b>	<b>49</b>
7.1. SERVICE REGISTRY CUSTOM RESOURCE	49
7.2. SERVICE REGISTRY CR SPEC	50
7.3. SERVICE REGISTRY CR STATUS	53
7.4. SERVICE REGISTRY MANAGED RESOURCES	55
7.5. SERVICE REGISTRY OPERATOR LABELS	55
<b>APPENDIX A. USING YOUR SUBSCRIPTION .....</b>	<b>56</b>
Accessing your account	56
Activating a subscription	56
Downloading ZIP and TAR files	56
Registering your system for packages	56



## PREFACE

### MAKING OPEN SOURCE MORE INCLUSIVE

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



# CHAPTER 1. SERVICE REGISTRY OPERATOR QUICKSTART

This chapter explains how to quickly install Service Registry Operator on the command line.

This quickstart example deploys Service Registry using the SQL database storage option:

- [Section 1.1, "Quickstart Service Registry Operator installation"](#)
- [Section 1.2, "Quickstart Service Registry deployment"](#)



## NOTE

The recommended installation option for production environments is using the OpenShift OperatorHub. The recommended storage option is SQL or Kafka.

## 1.1. QUICKSTART SERVICE REGISTRY OPERATOR INSTALLATION

You can quickly deploy the Service Registry Operator on the command line, without the Operator Lifecycle Manager, by using a downloaded set of installation files and examples.

### Prerequisites

- You must go to [Red Hat Software Downloads](#), select the product version, and download the examples in the Service Registry CRDs **.zip** file.

### Procedure

1. Create a project for the installation, for example, **apicurio-registry**:

```
NAMESPACE="apicurio-registry"
oc new-project "$NAMESPACE"
```

2. Apply the file located in the **install/** folder:

```
cat install/install.yaml | sed "s/apicurio-registry-operator-namespace/$NAMESPACE/g" | oc
apply -f -
```

## 1.2. QUICKSTART SERVICE REGISTRY DEPLOYMENT

To create a new Service Registry deployment, use the SQL database storage option. This requires external PostgreSQL storage to be configured as a prerequisite.

### Prerequisites

- Ensure that the Service Registry Operator is already installed.
- You have a PostgreSQL database reachable from your OpenShift cluster.

### Procedure

1. Create an **ApicurioRegistry** custom resource (CR), with your database connection configured, for example:

## Example CR for SQL storage

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-sql
spec:
  configuration:
    persistence: "sql"
    sql:
      dataSource:
        url: "jdbc:postgresql://<service name>.<namespace>.svc:5432/<database name>"
        userName: "postgres"
        password: "<password>" # Optional
```

2. Create the **ApicurioRegistry** CR in the same namespace that the Operator is deployed

```
oc project "$NAMESPACE"
oc apply -f ./examples/apicurioregistry_sql_cr.yaml
```

## CHAPTER 2. INSTALLING SERVICE REGISTRY ON OPENSIFT

This chapter explains how to install Service Registry on OpenShift Container Platform:

- [Section 2.1, “Installing Service Registry from the OpenShift OperatorHub”](#)

### Prerequisites

- Read the introduction in the [Service Registry User Guide](#)

## 2.1. INSTALLING SERVICE REGISTRY FROM THE OPENSIFT OPERATORHUB

You can install the Service Registry Operator on your OpenShift cluster from the OperatorHub. The OperatorHub is available from the OpenShift Container Platform web console and provides an interface for cluster administrators to discover and install Operators. For more details, see [Understanding OperatorHub](#).



### NOTE

You can install more than one instance of Service Registry depending on your environment. The number of instances depends on the number and type of artifacts stored in Service Registry and on your chosen storage option.

### Prerequisites

- You must have cluster administrator access to an OpenShift cluster.

### Procedure

1. In the OpenShift Container Platform web console, log in using an account with cluster administrator privileges.
2. Create a new OpenShift project:
  - a. In the left navigation menu, click **Home**, **Project**, and then **Create Project**.
  - b. Enter a project name, for example, **my-project**, and click **Create**.
3. In the left navigation menu, click **Operators** and then **OperatorHub**.
4. In the **Filter by keyword** text box, enter **registry** to find the **Red Hat Integration - Service Registry Operator**.
5. Read the information about the Operator, and click **Install** to display the Operator subscription page.
6. Select your subscription settings, for example:
  - **Update Channel** Select one of the following:
    - **2.x**: Includes all minor and patch updates, such as 2.3.0 and 2.0.3. For example, an installation on 2.0.x will upgrade to 2.3.x.

- **2.0.x:** Includes patch updates only, such as 2.0.1 and 2.0.2. For example, an installation on 2.0.x will ignore 2.3.x.
  - **Installation Mode:** Select one of the following:
    - **All namespaces on the cluster (default)**
    - **A specific namespace on the cluster** and then **my-project**
  - **Approval Strategy:** Select **Automatic** or **Manual**
7. Click **Install**, and wait a few moments until the Operator is ready for use.

#### Additional resources

- [Adding Operators to an OpenShift cluster](#)
- [Apicurio Registry Operator community in GitHub](#)

## CHAPTER 3. DEPLOYING SERVICE REGISTRY STORAGE IN AMQ STREAMS

This chapter explains how to install and configure Service Registry data storage in AMQ Streams.

- [Section 3.1, “Installing AMQ Streams from the OpenShift OperatorHub”](#)
- [Section 3.2, “Configuring Service Registry with Kafka storage on OpenShift”](#)
- [Section 3.3, “Configuring Kafka storage with TLS security”](#)
- [Section 3.4, “Configuring Kafka storage with SCRAM security”](#)
- [Section 3.5, “Configuring OAuth authentication for Kafka storage”](#)

### Prerequisites

- [Chapter 2, \*Installing Service Registry on OpenShift\*](#)

## 3.1. INSTALLING AMQ STREAMS FROM THE OPENSIFT OPERATORHUB

If you do not already have AMQ Streams installed, you can install the AMQ Streams Operator on your OpenShift cluster from the OperatorHub. The OperatorHub is available from the OpenShift Container Platform web console and provides an interface for cluster administrators to discover and install Operators. For more details, see [Understanding OperatorHub](#).

### Prerequisites

- You must have cluster administrator access to an OpenShift cluster
- See [Deploying and Upgrading AMQ Streams on OpenShift](#) for detailed information on installing AMQ Streams. This section shows a simple example of installing using the OpenShift OperatorHub.

### Procedure

1. In the OpenShift Container Platform web console, log in using an account with cluster administrator privileges.
2. Change to the OpenShift project in which you want to install AMQ Streams. For example, from the **Project** drop-down, select **my-project**.
3. In the left navigation menu, click **Operators** and then **OperatorHub**.
4. In the **Filter by keyword** text box, enter **AMQ Streams** to find the **Red Hat Integration - AMQ Streams** Operator.
5. Read the information about the Operator, and click **Install** to display the Operator subscription page.
6. Select your subscription settings, for example:
  - **Update Channel** and then **amq-streams-2.1.x**

- **Installation Mode:** Select one of the following:
    - All namespaces on the cluster (default)
    - A specific namespace on the cluster > my-project
  - **Approval Strategy:** Select **Automatic** or **Manual**
7. Click **Install**, and wait a few moments until the Operator is ready for use.

#### Additional resources

- [Adding Operators to an OpenShift cluster](#)
- [Deploying and Upgrading AMQ Streams on OpenShift](#)

## 3.2. CONFIGURING SERVICE REGISTRY WITH KAFKA STORAGE ON OPENSIFT

This section explains how to configure Kafka-based storage for Service Registry using AMQ Streams on OpenShift. The **kafkasql** storage option uses Kafka storage with in-memory H2 database. This storage option is suitable for production environments when **persistent** storage is configured for the Kafka cluster on OpenShift.

You can install Service Registry in an existing Kafka cluster or create a new Kafka cluster, depending on your environment.

#### Prerequisites

- You must have an OpenShift cluster with cluster administrator access.
- You must have already installed Service Registry. See [Chapter 2, \*Installing Service Registry on OpenShift\*](#).
- You must have already installed AMQ Streams. See [Section 3.1, "Installing AMQ Streams from the OpenShift OperatorHub"](#).

#### Procedure

1. In the OpenShift Container Platform web console, log in using an account with cluster administrator privileges.
2. If you do not already have a Kafka cluster configured, create a new Kafka cluster using AMQ Streams. For example, in the OpenShift OperatorHub:
  - a. Click **Installed Operators** and then **Red Hat Integration - AMQ Streams**
  - b. Under **Provided APIs** and then **Kafka**, click **Create Instance** to create a new Kafka cluster.
  - c. Edit the custom resource definition as appropriate, and click **Create**.

**WARNING**

The default example creates a cluster with 3 Zookeeper nodes and 3 Kafka nodes with **ephemeral** storage. This temporary storage is suitable for development and testing only, and not for production. For more details, see [Deploying and Upgrading AMQ Streams on OpenShift](#).

3. After the cluster is ready, click **Provided APIs > Kafka > my-cluster > YAML**.
4. In the **status** block, make a copy of the **bootstrapServers** value, which you will use later to deploy Service Registry. For example:

```
status:
  ...
  conditions:
  ...
  listeners:
    - addresses:
      - host: my-cluster-kafka-bootstrap.my-project.svc
        port: 9092
      bootstrapServers: 'my-cluster-kafka-bootstrap.my-project.svc:9092'
      type: plain
  ...
```

5. Click **Installed Operators > Red Hat Integration - Service Registry > ApicurioRegistry > Create ApicurioRegistry**.
6. Paste in the following custom resource definition, but use your **bootstrapServers** value that you copied earlier:

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql
spec:
  configuration:
    persistence: 'kafkasql'
    kafkasql:
      bootstrapServers: 'my-cluster-kafka-bootstrap.my-project.svc:9092'
```

7. Click **Create** and wait for the Service Registry route to be created on OpenShift.
8. Click **Networking > Route** to access the new route for the Service Registry web console. For example:

```
http://example-apicurioregistry-kafkasql.my-project.my-domain-name/
```

**Additional resources**

- For more details on creating Kafka clusters and topics using AMQ Streams, see [Deploying and Upgrading AMQ Streams on OpenShift](#).

### 3.3. CONFIGURING KAFKA STORAGE WITH TLS SECURITY

You can configure the AMQ Streams Operator and Service Registry Operator to use an encrypted Transport Layer Security (TLS) connection.

#### Prerequisites

- You must install the Service Registry Operator using the OperatorHub or command line.
- You must install the AMQ Streams Operator or have Kafka accessible from your OpenShift cluster.



#### NOTE

This section assumes that the AMQ Streams Operator is available, however you can use any Kafka deployment. In that case, you must manually create the Openshift secrets that the Service Registry Operator expects.

#### Procedure

1. In the OpenShift web console, click **Installed Operators**, select the **AMQ Streams Operator** details, and then the **Kafka** tab.
2. Click **Create Kafka** to provision a new Kafka cluster for Service Registry storage.
3. Configure the **authorization** and **tls** fields to use TLS authentication for the Kafka cluster, for example:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: registry-example-kafkasql-tls
  # Change or remove the explicit namespace
spec:
  kafka:
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      log.message.format.version: '2.7'
      inter.broker.protocol.version: '2.7'
    version: 2.7.0
  storage:
    type: ephemeral
  replicas: 3
  listeners:
    - name: tls
      port: 9093
      type: internal
      tls: true
    authentication:
```



```

    type: tls
  authorization:
    type: simple
  entityOperator:
    topicOperator: {}
    userOperator: {}
  zookeeper:
    storage:
      type: ephemeral
    replicas: 3

```

The default Kafka topic name that Service Registry uses to store data is **kafkasql-journal**. This topic is created automatically by Service Registry. You can override this behavior or the default topic name by setting the appropriate environment variables (default values):

- **REGISTRY\_KAFKASQL\_TOPIC\_AUTO\_CREATE=true**
- **REGISTRY\_KAFKASQL\_TOPIC=kafkasql-journal**

If you decide not to create the Kafka topic manually, skip the next step.

4. Click the **Kafka Topic** tab, and then **Create Kafka Topic** to create the **kafkasql-journal** topic:

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-tls
spec:
  partitions: 2
  replicas: 1
  config:
    cleanup.policy: compact

```

5. Create a **Kafka User** resource to configure authentication and authorization for the Service Registry user. You can specify a user name in the **metadata** section or use the default **my-user**.

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-tls
spec:
  authentication:
    type: tls
  authorization:
    acls:
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: topic

```

```

- operation: All
  resource:
    name: '*'
    patternType: literal
    type: cluster
- operation: All
  resource:
    name: '*'
    patternType: literal
    type: transactionalld
- operation: All
  resource:
    name: '*'
    patternType: literal
    type: group
type: simple

```

**NOTE**

You must configure the authorization specifically for the topics and resources that the Service Registry requires. This is a simple permissive example.

- Click **Workloads** and then **Secrets** to find two secrets that AMQ Streams creates for Service Registry to connect to the Kafka cluster:

- **my-cluster-cluster-ca-cert** - contains the PKCS12 truststore for the Kafka cluster
- **my-user** - contains the user's keystore

**NOTE**

The name of the secret can vary based on your cluster or user name.

- If you create the secrets manually, they must contain the following key-value pairs:

- **my-cluster-ca-cert**
  - **ca.p12** - truststore in PKCS12 format
  - **ca.password** - truststore password
- **my-user**
  - **user.p12** - keystore in PKCS12 format
  - **user.password** - keystore password

- Configure the following example configuration to deploy the Service Registry.

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkaql
spec:
  configuration:

```

```

persistence: "kafkasql"
kafkasql:
  bootstrapServers: "my-cluster-kafka-bootstrap.registry-example-kafkasql-tls.svc:9093"
  security:
    tls:
      keystoreSecretName: my-user
      truststoreSecretName: my-cluster-cluster-ca-cert

```



### IMPORTANT

You must use a different **bootstrapServers** address than in the plain insecure use case. The address must support TLS connections and is found in the specified **Kafka** resource under the **type: tls** field.

## 3.4. CONFIGURING KAFKA STORAGE WITH SCRAM SECURITY

You can configure the AMQ Streams Operator and Service Registry Operator to use Salted Challenge Response Authentication Mechanism (SCRAM-SHA-512) for the Kafka cluster.

### Prerequisites

- You must install the Service Registry Operator using the OperatorHub or command line.
- You must install the AMQ Streams Operator or have Kafka accessible from your OpenShift cluster.



### NOTE

This section assumes that AMQ Streams Operator is available, however you can use any Kafka deployment. In that case, you must manually create the Openshift secrets that the Service Registry Operator expects.

### Procedure

1. In the OpenShift web console, click **Installed Operators**, select the **AMQ Streams Operator** details, and then the **Kafka** tab.
2. Click **Create Kafka** to provision a new Kafka cluster for Service Registry storage.
3. Configure the **authorization** and **tls** fields to use SCRAM-SHA-512 authentication for the Kafka cluster, for example:

```

apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: registry-example-kafkasql-scrum
  # Change or remove the explicit namespace
spec:
  kafka:
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2

```

```

log.message.format.version: '2.7'
inter.broker.protocol.version: '2.7'
version: 2.7.0
storage:
  type: ephemeral
replicas: 3
listeners:
  - name: tls
    port: 9093
    type: internal
    tls: true
    authentication:
      type: scram-sha-512
authorization:
  type: simple
entityOperator:
  topicOperator: {}
  userOperator: {}
zookeeper:
  storage:
    type: ephemeral
  replicas: 3

```

The default Kafka topic name that Service Registry uses to store data is **kafkasql-journal**. This topic is created automatically by Service Registry. You can override this behavior or the default topic name by setting the appropriate environment variables (default values):

- **REGISTRY\_KAFKASQL\_TOPIC\_AUTO\_CREATE=true**
- **REGISTRY\_KAFKASQL\_TOPIC=kafkasql-journal**

If you decide not to create the Kafka topic manually, skip the next step.

4. Click the **Kafka Topic** tab, and then **Create Kafka Topic** to create the **kafkasql-journal** topic:

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-scam
spec:
  partitions: 2
  replicas: 1
  config:
    cleanup.policy: compact

```

5. Create a **Kafka User** resource to configure SCRAM authentication and authorization for the Service Registry user. You can specify a user name in the **metadata** section or use the default **my-user**.

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user

```

```

labels:
  strimzi.io/cluster: my-cluster
namespace: registry-example-kafkasql-scrum
spec:
  authentication:
    type: scram-sha-512
  authorization:
    acls:
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: topic
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: cluster
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: transactionalId
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: group
    type: simple

```

**NOTE**

You must configure the authorization specifically for the topics and resources that the Service Registry requires. This is a simple permissive example.

6. Click **Workloads** and then **Secrets** to find two secrets that AMQ Streams creates for Service Registry to connect to the Kafka cluster:

- **my-cluster-cluster-ca-cert** - contains the PKCS12 truststore for the Kafka cluster
- **my-user** - contains the user's keystore

**NOTE**

The name of the secret can vary based on your cluster or user name.

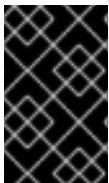
7. If you create the secrets manually, they must contain the following key-value pairs:

- **my-cluster-ca-cert**
  - **ca.p12** - the truststore in PKCS12 format
  - **ca.password** - truststore password
- **my-user**

- **password** - user password

8. Configure the following example settings to deploy the Service Registry:

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql
spec:
  configuration:
    persistence: "kafkasql"
    kafkasql:
      bootstrapServers: "my-cluster-kafka-bootstrap.registry-example-kafkasql-
scram.svc:9093"
    security:
      scram:
        truststoreSecretName: my-cluster-cluster-ca-cert
        user: my-user
        passwordSecretName: my-user
```



### IMPORTANT

You must use a different **bootstrapServers** address than in the plain insecure use case. The address must support TLS connections, and is found in the specified **Kafka** resource under the **type: tls** field.

## 3.5. CONFIGURING OAUTH AUTHENTICATION FOR KAFKA STORAGE

When using Kafka-based storage in AMQ Streams, Service Registry supports accessing a Kafka cluster that requires OAuth authentication. To enable this support, you must set some environment variables in your Service Registry deployment.

When you set these environment variables, the Kafka producer and consumer applications in Service Registry will use this configuration to authenticate to the Kafka cluster over OAuth.

### Prerequisites

- You must have already configured Kafka-based storage of Service Registry data in AMQ Streams. See [Section 3.2, "Configuring Service Registry with Kafka storage on OpenShift"](#).

### Procedure

- Set the following environment variables in your Service Registry deployment:

Environment variable	Description	Default value
<b>ENABLE_KAFKA_SASL</b>	Enables SASL OAuth authentication for Service Registry storage in Kafka. You must set this variable to <b>true</b> for the other variables to have effect.	<b>false</b>

Environment variable	Description	Default value
<b>CLIENT_ID</b>	The client ID used to authenticate to Kafka.	-
<b>CLIENT_SECRET</b>	The client secret used to authenticate to Kafka.	-
<b>OAuth_TOKEN_ENDPOINT_URI</b>	The URL of the OAuth identity server.	<b>http://localhost:8090</b>

### Additional resources

- For an example of how to set Service Registry environment variables on OpenShift, see [Section 6.1, “Configuring Service Registry health checks on OpenShift”](#)

## CHAPTER 4. DEPLOYING SERVICE REGISTRY STORAGE IN A POSTGRESQL DATABASE

This chapter explains how to install, configure, and manage Service Registry data storage in a PostgreSQL database.

- [Section 4.1, “Installing a PostgreSQL database from the OpenShift OperatorHub”](#)
- [Section 4.2, “Configuring Service Registry with PostgreSQL database storage on OpenShift”](#)
- [Section 4.3, “Backing up Service Registry PostgreSQL storage”](#)
- [Section 4.4, “Restoring Service Registry PostgreSQL storage”](#)

### Prerequisites

- [Chapter 2, \*Installing Service Registry on OpenShift\*](#)

### 4.1. INSTALLING A POSTGRESQL DATABASE FROM THE OPENSIFT OPERATORHUB

If you do not already have a PostgreSQL database Operator installed, you can install a PostgreSQL Operator on your OpenShift cluster from the OperatorHub. The OperatorHub is available from the OpenShift Container Platform web console and provides an interface for cluster administrators to discover and install Operators. For more details, see [Understanding OperatorHub](#).

### Prerequisites

- You must have cluster administrator access to an OpenShift cluster.

### Procedure

1. In the OpenShift Container Platform web console, log in using an account with cluster administrator privileges.
2. Change to the OpenShift project in which you want to install the PostgreSQL Operator. For example, from the **Project** drop-down, select **my-project**.
3. In the left navigation menu, click **Operators** and then **OperatorHub**.
4. In the **Filter by keyword** text box, enter **PostgreSQL** to find an Operator suitable for your environment, for example, **Crunchy PostgreSQL for OpenShift** or **PostgreSQL Operator by Dev4Ddevs.com**.
5. Read the information about the Operator, and click **Install** to display the Operator subscription page.
6. Select your subscription settings, for example:
  - **Update Channel:** **stable**
  - **Installation Mode:** **A specific namespace on the cluster** and then **my-project**
  - **Approval Strategy:** Select **Automatic** or **Manual**



- Click **Install**, and wait a few moments until the Operator is ready for use.



### IMPORTANT

You must read the documentation from your chosen **PostgreSQL** Operator for details on how to create and manage your database.

#### Additional resources

- [Adding Operators to an OpenShift cluster](#)
- [Crunchy PostgreSQL Operator QuickStart](#)

## 4.2. CONFIGURING SERVICE REGISTRY WITH POSTGRESQL DATABASE STORAGE ON OPENSIFT

This section explains how to configure storage for Service Registry on OpenShift using a PostgreSQL database Operator. You can install Service Registry in an existing database or create a new database, depending on your environment. This section shows a simple example using the PostgreSQL Operator by Dev4Ddevs.com.

#### Prerequisites

- You must have an OpenShift cluster with cluster administrator access.
- You must have already installed Service Registry. See [Chapter 2, Installing Service Registry on OpenShift](#).
- You must have already installed a PostgreSQL Operator on OpenShift. For example, see [Section 4.1, “Installing a PostgreSQL database from the OpenShift OperatorHub”](#).

#### Procedure

- In the OpenShift Container Platform web console, log in using an account with cluster administrator privileges.
- Change to the OpenShift project in which Service Registry and your PostgreSQL Operator are installed. For example, from the **Project** drop-down, select **my-project**.
- Create a PostgreSQL database for your Service Registry storage. For example, click **Installed Operators, PostgreSQL Operator by Dev4Ddevs.com**, and then **Create database**.
- Click **YAML** and edit the database settings as follows:
  - **name**: Change the value to **registry**
  - **image**: Change the value to **centos/postgresql-12-centos7**
- Edit any other database settings as needed depending on your environment, for example:

```
apiVersion: postgresql.dev4devs.com/v1alpha1
kind: Database
metadata:
  name: registry
  namespace: my-project
```

```
spec:
  databaseCpu: 30m
  databaseCpuLimit: 60m
  databaseMemoryLimit: 512Mi
  databaseMemoryRequest: 128Mi
  databaseName: example
  databaseNameKeyEnvVar: POSTGRESQL_DATABASE
  databasePassword: postgres
  databasePasswordKeyEnvVar: POSTGRESQL_PASSWORD
  databaseStorageRequest: 1Gi
  databaseUser: postgres
  databaseUserKeyEnvVar: POSTGRESQL_USER
  image: centos/postgresql-12-centos7
  size: 1
```

6. Click **Create**, and wait until the database is created.
7. Click **Installed Operators** > **Red Hat Integration - Service Registry** > **ApicurioRegistry** > **Create ApicurioRegistry**.
8. Paste in the following custom resource definition, and edit the values for the database **url** and credentials to match your environment:

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-sql
spec:
  configuration:
    persistence: 'sql'
  sql:
    dataSource:
      url: 'jdbc:postgresql://<service name>.<namespace>.svc:5432/<database name>'
      # e.g. url: 'jdbc:postgresql://acid-minimal-cluster.my-project.svc:5432/registry'
      userName: 'postgres'
      password: '<password>' # Optional
```

9. Click **Create** and wait for the Service Registry route to be created on OpenShift.
10. Click **Networking** > **Route** to access the new route for the Service Registry web console. For example:

```
http://example-apicurioregistry-sql.my-project.my-domain-name.com/
```

#### Additional resources

- [Crunchy PostgreSQL Operator QuickStart](#)
- [Apicurio Registry Operator QuickStart](#)

### 4.3. BACKING UP SERVICE REGISTRY POSTGRESQL STORAGE

When using storage in a PostgreSQL database, you must ensure that the data stored by Service Registry is backed up regularly.

**SQL Dump** is a simple procedure that works with any PostgreSQL installation. This uses the `pg_dump` utility to generate a file with SQL commands that you can use to recreate the database in the same state that it was in at the time of the dump.

**pg\_dump** is a regular PostgreSQL client application, which you can execute from any remote host that has access to the database. Like any other client, the operations that can perform are limited to the user permissions.

### Procedure

- Use the **pg\_dump** command to redirect the output to a file:

```
$ pg_dump dbname > dumpfile
```

You can specify the database server that **pg\_dump** connects to using the **-h host** and **-p port** options.

- You can reduce large dump files using a compression tool, such as `gzip`, for example:

```
$ pg_dump dbname | gzip > filename.gz
```

### Additional resources

- For details on client authentication, see the [PostgreSQL documentation](#).
- For details on importing and exporting registry content, see [Managing Service Registry content using the REST API](#).

## 4.4. RESTORING SERVICE REGISTRY POSTGRES SQL STORAGE

You can restore SQL Dump files created by **pg\_dump** using the **psql** utility.

### Prerequisites

- You must have already backed up your PostgreSQL database using **pg\_dump**. See [Section 4.3, “Backing up Service Registry PostgreSQL storage”](#).
- All users who own objects or have permissions on objects in the dumped database must already exist.

### Procedure

1. Enter the following command to create the database:

```
$ createdb -T template0 dbname
```

2. Enter the following command to restore the SQL dump

```
$ psql dbname < dumpfile
```

3. Run [ANALYZE](#) on each database so the query optimizer has useful statistics.

## CHAPTER 5. SECURING A SERVICE REGISTRY DEPLOYMENT

This chapter explains how to configure security settings for your Service Registry deployment on OpenShift:

- [Section 5.1, "Securing Service Registry using the Red Hat Single Sign-On Operator"](#)
- [Section 5.2, "Configuring Service Registry authentication and authorization with Red Hat Single Sign-On"](#)
- [Section 5.3, "Service Registry authentication and authorization configuration options"](#)
- [Section 5.4, "Configuring an HTTPS connection to Service Registry from inside the OpenShift cluster"](#)
- [Section 5.5, "Configuring an HTTPS connection to Service Registry from outside the OpenShift cluster"](#)

Service Registry provides authentication and authorization using Red Hat Single Sign-On based on OpenID Connect (OIDC) or HTTP basic. You can configure the required settings automatically using the Red Hat Single Sign-On Operator, or manually configure them in Red Hat Single Sign-On and Service Registry.

Service Registry provides role-based authentication and authorization for the Service Registry web console and core REST API using Red Hat Single Sign-On. Service Registry also provides content-based authorization at the schema or API level, where only the artifact creator has write access. You can also configure an HTTPS connection to Service Registry from inside or outside an OpenShift cluster.

### Additional resources

- For details on security configuration for Java client applications, see the following:
  - [Service Registry Java client configuration](#)
  - [Service Registry serializer/deserializer configuration](#)

## 5.1. SECURING SERVICE REGISTRY USING THE RED HAT SINGLE SIGN-ON OPERATOR

The following procedure shows how to configure a Service Registry REST API and web console to be protected by Red Hat Single Sign-On.

Service Registry supports the following user roles:

**Table 5.1. Service Registry user roles**

Name	Capabilities
<b>sr-admin</b>	Full access, no restrictions.
<b>sr-developer</b>	Create artifacts and configure artifact rules. Cannot modify global rules, perform import/export, or use <b>/admin</b> REST API endpoint.

Name	Capabilities
<b>sr-readonly</b>	View and search only. Cannot modify artifacts or rules, perform import/export, or use <b>/admin</b> REST API endpoint.



## NOTE

There is a related configuration option in the **ApicurioRegistry** CRD that you can use to set the web console to read-only mode. However, this configuration does not affect the REST API.

## Prerequisites

- You must have already installed the Service Registry Operator.
- You must install the Red Hat Single Sign-On Operator or have Red Hat Single Sign-On accessible from your OpenShift cluster.



## IMPORTANT

The example configuration in this procedure is intended for development and testing only. To keep the procedure simple, it does not use HTTPS and other defenses recommended for a production environment. For more details, see the Red Hat Single Sign-On documentation.

## Procedure

1. In the OpenShift web console, click **Installed Operators** and **Red Hat Single Sign-On Operator**, and then the **Keycloak** tab.
2. Click **Create Keycloak** to provision a new Red Hat Single Sign-On instance for securing a Service Registry deployment. You can use the default value, for example:

```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: example-keycloak
labels:
  app: sso
spec:
  instances: 1
externalAccess:
  enabled: True
podDisruptionBudget:
  enabled: True
```

3. Wait until the instance has been created, and click **Networking** and then **Routes** to access the new route for the **keycloak** instance.
4. Click the **Location** URL and copy the displayed **../auth** URL value for later use when deploying Service Registry.

5. Click **Installed Operators** and **Red Hat Single Sign-On Operator**, and click the **Keycloak Realm** tab, and then **Create Keycloak Realm** to create a **registry** example realm:

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  name: registry-keycloakrealm
  labels:
    app: registry
spec:
  instanceSelector:
    matchLabels:
      app: sso
  realm:
    displayName: Registry
    enabled: true
    id: registry
    realm: registry
    sslRequired: none
    roles:
      realm:
        - name: sr-admin
        - name: sr-developer
        - name: sr-readonly
  clients:
    - clientId: registry-client-ui
      implicitFlowEnabled: true
      redirectUris:
        - '*'
      standardFlowEnabled: true
      webOrigins:
        - '*'
      publicClient: true
    - clientId: registry-client-api
      implicitFlowEnabled: true
      redirectUris:
        - '*'
      standardFlowEnabled: true
      webOrigins:
        - '*'
      publicClient: true
  users:
    - credentials:
        - temporary: false
          type: password
          value: changeme
      enabled: true
      realmRoles:
        - sr-admin
      username: registry-admin
    - credentials:
        - temporary: false
          type: password
          value: changeme
      enabled: true
      realmRoles:
```

```

- sr-developer
username: registry-developer
- credentials:
  - temporary: false
    type: password
    value: changeme
enabled: true
realmRoles:
  - sr-readonly
username: registry-user

```



## IMPORTANT

You must customize this **KeycloakRealm** resource with values suitable for your environment if you are deploying to production. You can also create and manage realms using the Red Hat Single Sign-On web console.

6. If your cluster does not have a valid HTTPS certificate configured, you can create the following HTTP **Service** and **Ingress** resources as a temporary workaround:
  - a. Click **Networking** and then **Services**, and click **Create Service** using the following example:

```

apiVersion: v1
kind: Service
metadata:
  name: keycloak-http
  labels:
    app: keycloak
spec:
  ports:
    - name: keycloak-http
      protocol: TCP
      port: 8080
      targetPort: 8080
  selector:
    app: keycloak
    component: keycloak
  type: ClusterIP
  sessionAffinity: None
status:
  loadBalancer: {}

```

- b. Click **Networking** and then **Ingresses**, and click **Create Ingress** using the following example::

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: keycloak-http
  labels:
    app: keycloak
spec:
  rules:
    - host: KEYCLOAK_HTTP_HOST
      http:

```

```

paths:
  - path: /
    pathType: ImplementationSpecific
  backend:
    service:
      name: keycloak-http
      port:
        number: 8080

```

Modify the **host** value to create a route accessible for the Service Registry user, and use it instead of the HTTPS route created by Red Hat Single Sign-On Operator.

- Click the **Service Registry Operator**, and on the **ApicurioRegistry** tab, click **Create ApicurioRegistry**, using the following example, but replace your values in the **keycloak** section.

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-keycloak
spec:
  configuration:
    security:
      keycloak:
        url: "http://keycloak-http-<namespace>.apps.<cluster host>/auth"
        # ^ Required
        # Keycloak server URL, must end with `/auth`.
        # Use an HTTP URL in development.
        realm: "registry"
        # apiClientId: "registry-client-api"
        # ^ Optional (default value)
        # uiClientId: "registry-client-ui"
        # ^ Optional (default value)
      persistence: 'kafkasql'
      kafkasql:
        bootstrapServers: '<my-cluster>-kafka-bootstrap.<my-namespace>.svc:9092'

```

## 5.2. CONFIGURING SERVICE REGISTRY AUTHENTICATION AND AUTHORIZATION WITH RED HAT SINGLE SIGN-ON

This section explains how to manually configure authentication and authorization options for Service Registry using Red Hat Single Sign-On.



### NOTE

Alternatively, for details on how to configure these settings automatically, see [Section 5.1, "Securing Service Registry using the Red Hat Single Sign-On Operator"](#).

You can enable authentication for the Service Registry web console and core REST API using Red Hat Single Sign-On based on OAuth using OpenID Connect (OIDC). The same Red Hat Single Sign-On realm and users are federated across the Service Registry web console and core REST API using OpenID Connect so that you only require one set of credentials.

Service Registry provides role-based authorization for default admin, write, and read-only user roles. Service Registry also provides content-based authorization at the schema or API level, where only the



creator of the registry artifact can update or delete it. Service Registry authentication and authorization settings are disabled by default.

## Prerequisites

- Red Hat Single Sign-On is installed and running. For more details, see the [Red Hat Single Sign-On user documentation](#).
- Service Registry is installed and running.

## Procedure

1. In the Red Hat Single Sign-On Admin Console, create a Red Hat Single Sign-On realm for Service Registry. By default, Service Registry expects a realm name of **registry**. For more details on creating realms, see the [Red Hat Single Sign-On user documentation](#).
2. Create a Red Hat Single Sign-On client for the Service Registry API. By default, Service Registry expects the following settings:

- **Client ID:** **registry-api**
- **Client Protocol:** **openid-connect**
- **Access Type:** **bearer-only**  
You can use the defaults for the other client settings.



### NOTE

If you are using Red Hat Single Sign-On service accounts, the client **Access Type** must be **confidential** instead of **bearer-only**.

3. Create a Red Hat Single Sign-On client for the Service Registry web console. By default, Service Registry expects the following settings:

- **Client ID:** **apicurio-registry**
- **Client Protocol:** **openid-connect**
- **Access Type:** **public**
- **Valid Redirect URLs:** **http://my-registry-url:8080/\***
- **Web Origins:** **+**  
You can use the defaults for the other client settings.

4. In your Service Registry deployment on OpenShift, set the following Service Registry environment variables to configure authentication using Red Hat Single Sign-On:

**Table 5.2. Configuration for Service Registry authentication**

Environment variable	Description	Type	Default
<b>AUTH_ENABLED</b>	If set to <b>true</b> , the environment variables that follow are required.	String	<b>false</b>

Environment variable	Description	Type	Default
<b>KEYCLOAK_URL</b>	The URL of the Red Hat Single Sign-On authentication server to use. Must end with <b>/auth</b> .	String	None
<b>KEYCLOAK_REALM</b>	The Red Hat Single Sign-On realm used for authentication.	String	<b>registry</b>
<b>KEYCLOAK_API_CLIENT_ID</b>	The client ID for the Service Registry REST API.	String	<b>registry-api</b>
<b>KEYCLOAK_UI_CLIENT_ID</b>	The client ID for the Service Registry web console.	String	<b>apicurio-registry</b>

**TIP**

For an example of setting environment variables on OpenShift, see [Section 6.1, “Configuring Service Registry health checks on OpenShift”](#).

- Set the following option to **true** to enable Service Registry user roles in Red Hat Single Sign-On:

**Table 5.3. Configuration for Service Registry role-based authorization**

Environment variable	Java system property	Type	Default value
<b>ROLE_BASED_AUTHZ_ENABLED</b>	<b>registry.auth.role-based-authorization</b>	Boolean	<b>false</b>

- When Service Registry user roles are enabled, you must assign Service Registry users to at least one of the following default user roles in your Red Hat Single Sign-On realm:

**Table 5.4. Default user roles for registry authentication and authorization**

Role	Read artifacts	Write artifacts	Global rules	Summary
<b>sr-admin</b>	Yes	Yes	Yes	Full access to all create, read, update, and delete operations.

Role	Read artifacts	Write artifacts	Global rules	Summary
<b>sr-developer</b>	Yes	Yes	No	Access to create, read, update, and delete operations, except configuring global rules. This role can configure artifact rules.
<b>sr-readonly</b>	Yes	No	No	Access to read and search operations only. This role cannot configure any rules.

- Set the following to **true** to enable owner-only authorization for updates to schema and API artifacts in Service Registry:

**Table 5.5. Configuration for owner-only authorization**

Environment variable	Java system property	Type	Default value
<b>REGISTRY_AUTH_OBAC_ENABLED</b>	<b>registry.auth.owner-only-authorization</b>	Boolean	<b>false</b>

#### Additional resources

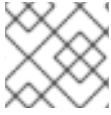
- For details on configuring non-default user role names, see [Section 5.3, “Service Registry authentication and authorization configuration options”](#)
- For an open source example application and Keycloak realm, see [Docker Compose-based example of using Keycloak with Apicurio Registry](#)
- For details on how to use Red Hat Single Sign-On in a production environment, see [Red Hat Single Sign-On documentation](#)
- For details on custom security configuration, the see [Quarkus Open ID Connect documentation](#)

## 5.3. SERVICE REGISTRY AUTHENTICATION AND AUTHORIZATION CONFIGURATION OPTIONS

Service Registry provides authentication options for OpenID Connect with Red Hat Single Sign-On or HTTP basic authentication.

Service Registry provides authorization options for role-based and content-based approaches:

- Role-based authorization for default admin, write, and read-only user roles.
- Content-based authorization for schema or API artifacts, where only the owner of the artifacts or artifact group can update or delete artifacts.

**NOTE**

Service Registry authentication and authorization options are disabled by default.

This chapter provides details on the following configuration options:

- [Service Registry authentication using OpenID Connect with Red Hat Single Sign-On](#)
- [Service Registry authentication using HTTP basic](#)
- [Service Registry role-based authorization](#)
- [Service Registry owner-only authorization](#)
- [Service Registry authenticated read access](#)
- [Service Registry anonymous read-only access](#)

### Service Registry authentication using OpenID Connect with Red Hat Single Sign-On

You can set the following environment variables to configure authentication for the Service Registry web console and API using Red Hat Single Sign-On:

**Table 5.6. Configuration for Service Registry authentication options**

Environment variable	Description	Type	Default
<b>AUTH_ENABLED</b>	Enables or disables authentication in Service Registry. When set to <b>true</b> , the environment variables that follow are required.	String	<b>false</b>
<b>KEYCLOAK_URL</b>	The URL of the Red Hat Single Sign-On authentication server to use. Must end with <b>/auth</b> .	String	-
<b>KEYCLOAK_REALM</b>	The Red Hat Single Sign-On realm used for authentication.	String	-
<b>KEYCLOAK_API_CLIENT_ID</b>	The client ID for the Service Registry REST API.	String	<b>registry-api</b>
<b>KEYCLOAK_UI_CLIENT_ID</b>	The client ID for the Service Registry web console.	String	<b>apicurio-registry</b>

### Service Registry authentication using HTTP basic

**NOTE**

By default, Service Registry supports authentication using OpenID Connect. Users (or API clients) must obtain an access token to make authenticated calls to the Service Registry REST API. However, because some tools do not support OpenID Connect, you can also configure Service Registry to support HTTP basic authentication by setting the following configuration option to **true**.

Table 5.7. Configuration for Service Registry HTTP basic authentication

Environment variable	Java system property	Type	Default value
<b>CLIENT_CREDENTIALS_BASIC_AUTH_ENABLED</b>	<b>registry.auth.basic-auth-client-credentials.enabled</b>	Boolean	<b>false</b>

### Service Registry role-based authorization

You can set the following option to **true** to enable role-based authorization in Service Registry:

Table 5.8. Configuration for Service Registry role-based authorization

Environment variable	Java system property	Type	Default value
<b>ROLE_BASED_AUTHZ_ENABLED</b>	<b>registry.auth.role-based-authorization</b>	Boolean	<b>false</b>

You can then configure role-based authorization to use roles included in the user's authentication token (for example, granted when authenticating using Red Hat Single Sign-On), or to use role mappings managed internally by Service Registry.

### Use roles assigned in Red Hat Single Sign-On

To enable using roles assigned by Red Hat Single Sign-On, set the following environment variables:

Table 5.9. Configuration for Service Registry role-based authorization using Red Hat Single Sign-On

Environment variable	Description	Type	Default
<b>ROLE_BASED_AUTHZ_SOURCE</b>	When set to <b>token</b> , user roles are taken from the authentication token.	String	<b>token</b>
<b>REGISTRY_AUTH_ROLES_ADMIN</b>	The name of the role that indicates a user is an admin.	String	<b>sr-admin</b>
<b>REGISTRY_AUTH_ROLES_DEVELOPER</b>	The name of the role that indicates a user is a developer.	String	<b>sr-developer</b>
<b>REGISTRY_AUTH_ROLES_READONLY</b>	The name of the role that indicates a user has read-only access.	String	<b>sr-readonly</b>

When Service Registry is configured to use roles from Red Hat Single Sign-On, you must assign Service Registry users to at least one of the following user roles in Red Hat Single Sign-On. However, you can configure different user role names using the environment variables in [Table 5.9, "Configuration for Service Registry role-based authorization using Red Hat Single Sign-On"](#).

Table 5.10. Service Registry roles for authentication and authorization

Role name	Read artifacts	Write artifacts	Global rules	Description
<b>sr-admin</b>	Yes	Yes	Yes	Full access to all create, read, update, and delete operations.
<b>sr-developer</b>	Yes	Yes	No	Access to create, read, update, and delete operations, except configuring global rules and import/export. This role can configure artifact rules only.
<b>sr-readonly</b>	Yes	No	No	Access to read and search operations only. This role cannot configure any rules.

### Manage roles directly in Service Registry

To enable using roles managed internally by Service Registry, set the following environment variables:

**Table 5.11. Configuration for Service Registry role-based authorization using internal role mappings**

Environment variable	Description	Type	Default
<b>ROLE_BASED_AUTHZ_SOURCE</b>	When set to <b>application</b> , user roles are managed internally by Service Registry.	String	<b>token</b>

When using internally managed role mappings, users can be assigned a role using the `/admin/roleMappings` endpoint in the Service Registry REST API. For more details, see [Apicurio Registry REST API documentation](#).

Users can be granted exactly one role: **ADMIN**, **DEVELOPER**, or **READ\_ONLY**. Only users with admin privileges can grant access to other users.

### Service Registry admin-override configuration

Because there are no default admin users in Service Registry, it is usually helpful to configure another way for users to be identified as admins. You can configure this admin-override feature using the following environment variables:

**Table 5.12. Configuration for Service Registry admin-override**

Environment variable	Description	Type	Default
<b>REGISTRY_AUTH_ADMIN_OVERRIDE_ENABLED</b>	Enables the admin-override feature.	String	<b>false</b>
<b>REGISTRY_AUTH_ADMIN_OVERRIDE_FROM</b>	Where to look for admin-override information. Only <b>token</b> is currently supported.	String	<b>token</b>

Environment variable	Description	Type	Default
<b>REGISTRY_AUTH_ADMIN_OVERRIDE_TYPE</b>	The type of information used to determine if a user is an admin. Values depend on the value of the FROM variable, for example, <b>role</b> or <b>claim</b> when FROM is <b>token</b> .	String	<b>role</b>
<b>REGISTRY_AUTH_ADMIN_OVERRIDE_ROLE</b>	The name of the role that indicates a user is an admin.	String	<b>sr-admin</b>
<b>REGISTRY_AUTH_ADMIN_OVERRIDE_CLAIM</b>	The name of a JWT token claim to use for determining admin-override.	String	<b>org-admin</b>
<b>REGISTRY_AUTH_ADMIN_OVERRIDE_CLAIM_VALUE</b>	The value that the JWT token claim indicated by the CLAIM variable must be for the user to be granted admin-override.	String	<b>true</b>

For example, you can use this admin-override feature to assign the **sr-admin** role to a single user in Red Hat Single Sign-On, which grants that user the admin role. That user can then use the `/admin/roleMappings` REST API (or associated UI) to grant roles to additional users (including additional admins).

### Service Registry owner-only authorization

You can set the following options to **true** to enable owner-only authorization for updates to artifacts or artifact groups in Service Registry:

Table 5.13. Configuration for owner-only authorization

Environment variable	Java system property	Type	Default value
<b>REGISTRY_AUTH_OBAC_ENABLED</b>	<b>registry.auth.owner-only-authorization</b>	Boolean	<b>false</b>
<b>REGISTRY_AUTH_OBAC_LIMIT_GROUP_ACCESS</b>	<b>registry.auth.owner-only-authorization.limit-group-access</b>	Boolean	<b>false</b>

When owner-only authorization is enabled, only the user who created an artifact can modify or delete that artifact.

When owner-only authorization and group owner-only authorization are both enabled, only the user who created an artifact group has write access to that artifact group, for example, to add or remove artifacts in that group.

### Service Registry authenticated read access

When the authenticated read access option is enabled, Service Registry grants at least read-only access to requests from any authenticated user in the same organization, regardless of their user role.

To enable authenticated read access, you must first enable role-based authorization, and then set the following option to **true**:

**Table 5.14. Configuration for authenticated read access**

Environment variable	Java system property	Type	Default value
<b>REGISTRY_AUTH_AUTHENTICATED_READS_ENABLED</b>	<b>registry.auth.authenticated-read-access.enabled</b>	Boolean	<b>false</b>

For more details, see [the section called "Service Registry role-based authorization"](#) .

### Service Registry anonymous read-only access

In addition to the two main types of authorization (role-based and owner-based authorization), Service Registry supports an anonymous read-only access option.

To allow anonymous users, such as REST API calls with no authentication credentials, to make read-only calls to the REST API, set the following option to **true**:

**Table 5.15. Configuration for anonymous read-only access**

Environment variable	Java system property	Type	Default value
<b>REGISTRY_AUTH_ANONYMOUS_READ_ACCESS_ENABLED</b>	<b>registry.auth.anonymous-read-access.enabled</b>	Boolean	<b>false</b>

### Additional resources

- For an example of how to set environment variables in your Service Registry deployment on OpenShift, see [Section 6.1, "Configuring Service Registry health checks on OpenShift"](#)
- For details on configuring custom authentication for Service Registry, the see [Quarkus Open ID Connect documentation](#)

## 5.4. CONFIGURING AN HTTPS CONNECTION TO SERVICE REGISTRY FROM INSIDE THE OPENSIFT CLUSTER

The following procedure shows how to configure Service Registry deployment to expose a port for HTTPS connections from inside the OpenShift cluster.



**WARNING**

This kind of connection is not directly available outside of the cluster. Routing is based on hostname, which is encoded in the case of an HTTPS connection. Therefore, edge termination or other configuration is still needed. See [Section 5.5, “Configuring an HTTPS connection to Service Registry from outside the OpenShift cluster”](#).

**Prerequisites**

- You must have already installed the Service Registry Operator.

**Procedure**

1. Generate a **keystore** with a self-signed certificate. You can skip this step if you are using your own certificates.

```
keytool -genkey -trustcacerts -keyalg RSA -keystore registry-keystore.jks -storepass password
```

2. Create a new secret to hold the keystore and keystore password.
  - a. In the left navigation menu of the OpenShift web console, click **Workloads > Secrets > Create Key/Value Secret**
  - b. Use the following values:
    - Name: **registry-keystore**
    - Key 1: **keystore.jks**
    - Value 1: *registry-keystore.jks* (uploaded file)
    - Key 2: **password**
    - Value 2: *password*

**NOTE**

If you encounter a **java.io.IOException: Invalid keystore format**, the upload of the binary file did not work properly. As an alternative, encode the file as a base64 string using **cat registry-keystore.jks | base64 -w0 > data.txt** and edit the **Secret** resource as yaml to manually add the encoded file.

3. Edit the **Deployment** resource of the Service Registry instance. You can find the correct name in a status field of the Service Registry Operator.
  - a. Add the keystore secret as a volume:

```
template:
  spec:
    volumes:
      - name: registry-keystore-secret-volume
        secret:
          secretName: registry-keystore
```

- b. Add a volume mount:

```
volumeMounts:
  - name: registry-keystore-secret-volume
    mountPath: /etc/registry-keystore
    readOnly: true
```

- c. Add **JAVA\_OPTIONS** and **KEYSTORE\_PASSWORD** environment variables:

```
- name: KEYSTORE_PASSWORD
  valueFrom:
    secretKeyRef:
      name: registry-keystore
      key: password
- name: JAVA_OPTIONS
  value: >-
    -Dquarkus.http.ssl.certificate.key-store-file=/etc/registry-keystore/keystore.jks
    -Dquarkus.http.ssl.certificate.key-store-file-type=jks
    -Dquarkus.http.ssl.certificate.key-store-password=${KEYSTORE_PASSWORD}
```



#### NOTE

Order is important when using string interpolation.

- d. Enable the HTTPS port:

```
ports:
  - containerPort: 8080
    protocol: TCP
  - containerPort: 8443
    protocol: TCP
```

4. Edit the **Service** resource of the Service Registry instance. You can find the correct name in a status field of the Service Registry Operator.

```
ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
  - name: https
    protocol: TCP
    port: 8443
    targetPort: 8443
```

5. Verify that the connection is working:

- a. Connect into a pod on the cluster using SSH (you can use the Service Registry pod):

```
oc rsh -n default example-apicuriregistry-deployment-vx28s-4-lmtqb
```

- b. Find the cluster IP of the Service Registry pod from the **Service** resource (see the **Location** column in the web console). Afterwards, execute a test request (we are using self-signed certificate, so an insecure flag is required):

```
curl -k https://172.30.209.198:8443/health
[...]
```

## 5.5. CONFIGURING AN HTTPS CONNECTION TO SERVICE REGISTRY FROM OUTSIDE THE OPENSIFT CLUSTER

The following procedure shows how to configure Service Registry deployment to expose an HTTPS edge-terminated route for connections from outside the OpenShift cluster.

### Prerequisites

- You must have already installed the Service Registry Operator.
- Read the [OpenShift documentation for creating secured routes](#).

### Procedure

1. Add a second **Route** in addition to the HTTP route created by the Service Registry Operator. See the following example:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  [...]
labels:
  app: example-apicurioregistry
  [...]
spec:
  host: example-apicurioregistry-default.apps.example.com
  to:
    kind: Service
    name: example-apicurioregistry-service-9whd7
    weight: 100
  port:
    targetPort: 8080
  tls:
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
    wildcardPolicy: None
```



### NOTE

Make sure the **`insecureEdgeTerminationPolicy: Redirect`** configuration property is set.

If you do not specify a certificate, OpenShift will use a default. You can alternatively generate a custom self-signed certificate using the following commands:

```
openssl genrsa 2048 > host.key &&  
openssl req -new -x509 -nodes -sha256 -days 365 -key host.key -out host.cert
```

and then create a route using the OpenShift CLI:

```
oc create route edge \  
  --service=example-apicurioregistry-service-9whd7 \  
  --cert=host.cert --key=host.key \  
  --hostname=example-apicurioregistry-default.apps.example.com \  
  --insecure-policy=Redirect \  
  -n default
```

## CHAPTER 6. CONFIGURING AND MANAGING A SERVICE REGISTRY DEPLOYMENT

This chapter explains how to configure and manage optional settings for your Service Registry deployment on OpenShift:

- [Section 6.1, “Configuring Service Registry health checks on OpenShift”](#)
- [Section 6.2, “Environment variables for Service Registry health checks”](#)
- [Section 6.3, “Managing Service Registry environment variables”](#)
- [Section 6.4, “Configuring the Service Registry web console”](#)
- [Section 6.5, “Configuring Service Registry logging”](#)
- [Section 6.6, “Configuring Service Registry event sourcing”](#)

### 6.1. CONFIGURING SERVICE REGISTRY HEALTH CHECKS ON OPENSHIFT

You can configure optional environment variables for liveness and readiness probes to monitor the health of the Service Registry server on OpenShift:

- *Liveness probes* test if the application can make progress. If the application cannot make progress, OpenShift automatically restarts the failing Pod.
- *Readiness probes* test if the application is ready to process requests. If the application is not ready, it can become overwhelmed by requests, and OpenShift stops sending requests for the time that the probe fails. If other Pods are OK, they continue to receive requests.



#### IMPORTANT

The default values of the liveness and readiness environment variables are designed for most cases and should only be changed if required by your environment. Any changes to the defaults depend on your hardware, network, and amount of data stored. These values should be kept as low as possible to avoid unnecessary overhead.

#### Prerequisites

- You must have an OpenShift cluster with cluster administrator access.
- You must have already installed Service Registry on OpenShift.
- You must have already installed and configured your chosen Service Registry storage in AMQ Streams or PostgreSQL.

#### Procedure

1. In the OpenShift Container Platform web console, log in using an account with cluster administrator privileges.
2. Click **Installed Operators** > **Red Hat Integration - Service Registry Operator**

3. On the **ApicurioRegistry** tab, click the Operator custom resource for your deployment, for example, **example-apicurioregistry**.
4. In the main overview page, find the **Deployment Name** section and the corresponding **DeploymentConfig** name for your Service Registry deployment, for example, **example-apicurioregistry**.
5. In the left navigation menu, click **Workloads > Deployment Configs**, and select your **DeploymentConfig** name.
6. Click the **Environment** tab, and enter your environment variables in the **Single values env** section, for example:
  - **NAME: LIVENESS\_STATUS\_RESET**
  - **VALUE: 350**
7. Click **Save** at the bottom.  
Alternatively, you can perform these steps using the OpenShift **oc** command. For more details, see the [OpenShift CLI documentation](#).

#### Additional resources

- [Section 6.2, “Environment variables for Service Registry health checks”](#)
- [OpenShift documentation on monitoring application health](#)

## 6.2. ENVIRONMENT VARIABLES FOR SERVICE REGISTRY HEALTH CHECKS

This section describes the available environment variables for Service Registry health checks on OpenShift. These include liveness and readiness probes to monitor the health of the Service Registry server on OpenShift. For an example procedure, see [Section 6.1, “Configuring Service Registry health checks on OpenShift”](#).



### IMPORTANT

The following environment variables are provided for reference only. The default values are designed for most cases and should only be changed if required by your environment. Any changes to the defaults depend on your hardware, network, and amount of data stored. These values should be kept as low as possible to avoid unnecessary overhead.

#### Liveness environment variables

Table 6.1. Environment variables for Service Registry liveness probes

Name	Description	Type	Default
<b>LIVENESS_ERROR_THRESHOLD</b>	Number of liveness issues or errors that can occur before the liveness probe fails.	Integer	<b>1</b>

Name	Description	Type	Default
<b>LIVENESS_COUNTER_RESET</b>	Period in which the threshold number of errors must occur. For example, if this value is 60 and the threshold is 1, the check fails after two errors occur in 1 minute	Seconds	<b>60</b>
<b>LIVENESS_STATUS_RESET</b>	Number of seconds that must elapse without any more errors for the liveness probe to reset to OK status.	Seconds	<b>300</b>
<b>LIVENESS_ERRORS_IGNORED</b>	Comma-separated list of ignored liveness exceptions.	String	<b>io.grpc.StatusRuntimeException,org.apache.kafka.streams.errors.InvalidStateStoreException</b>

**NOTE**

Because OpenShift automatically restarts a Pod that fails a liveness check, the liveness settings, unlike readiness settings, do not directly affect behavior of Service Registry on OpenShift.

**Readiness environment variables**

Table 6.2. Environment variables for Service Registry readiness probes

Name	Description	Type	Default
<b>READINESS_ERROR_THRESHOLD</b>	Number of readiness issues or errors that can occur before the readiness probe fails.	Integer	<b>1</b>
<b>READINESS_COUNTER_RESET</b>	Period in which the threshold number of errors must occur. For example, if this value is 60 and the threshold is 1, the check fails after two errors occur in 1 minute.	Seconds	<b>60</b>
<b>READINESS_STATUS_RESET</b>	Number of seconds that must elapse without any more errors for the liveness probe to reset to OK status. In this case, this means how long the Pod stays not ready, until it returns to normal operation.	Seconds	<b>300</b>

Name	Description	Type	Default
<b>READINESS_TIMEOUT</b>	<p>Readiness tracks the timeout of two operations:</p> <ul style="list-style-type: none"> <li>• How long it takes for storage requests to complete</li> <li>• How long it takes for HTTP REST API requests to return a response</li> </ul> <p>If these operations take more time than the configured timeout, this is counted as a readiness issue or error. This value controls the timeouts for both operations.</p>	Seconds	<b>5</b>

#### Additional resources

- [Section 6.1, “Configuring Service Registry health checks on OpenShift”](#)
- [OpenShift documentation on monitoring application health](#)

## 6.3. MANAGING SERVICE REGISTRY ENVIRONMENT VARIABLES

Service Registry Operator manages most common Service Registry configuration, but there are some options that you can adjust manually. You can update these by setting an environment variable on the Service Registry **Deployment** resource. If the specific configuration option is not available in the **ApicurioRegistry** CR, you can use an environment variable to adjust it.

#### Procedure

##### OpenShift web console

1. Select the **Installed Operators** tab, and then **Red Hat Integration - Service Registry Operator**.
2. On the **Apicurio Registry** tab, click the **ApicurioRegistry** CR for your Service Registry deployment.
3. On the main overview page, view the **managedResources** section, which contains the name of the **Deployment** managed by the Operator to deploy your Service Registry instance.
4. Find that **Deployment** in the **Workloads > Deployments** in the left menu.
5. Select the **Deployment** with the correct name, and select the **Environment** tab.
6. You can add or modify your environment variable to the **Single values (env)** section.
7. Click **Save** at the bottom.

##### OpenShift CLI



1. Select the project where Service Registry is installed.
2. Run **oc get apicuriregistry** to get the list of **ApicurioRegistry** CRs
3. Run **oc describe** on the CR representing the Service Registry instance that you want to configure.
4. View the **managedResource** in the **status** section.
5. Find that **Deployment** and enter **oc edit**.
6. Add or modify the environment variable in the **spec.template.spec.containers[0].env** section.

## 6.4. CONFIGURING THE SERVICE REGISTRY WEB CONSOLE

You can configure the Service Registry web console specifically for your deployment environment or to customize its behavior. This section provides details on how to configure optional environment variables for the Service Registry web console.

### Prerequisites

- You have already installed Service Registry.

### Configuring the web console deployment environment

When a user navigates their browser to the Service Registry web console, some initial configuration settings are loaded. Two important configuration properties are:

- URL for core Service Registry server REST API
- URL for Service Registry web console client

Typically, Service Registry automatically detects and generates these settings, but there are some deployment environments where this automatic detection can fail. If this happens, you can configure environment variables to explicitly set these URLs for your environment.

### Procedure

Configure the following environment variables to override the default URLs:

- **REGISTRY\_UI\_CONFIG\_APIURL**: Specifies the URL for the core Service Registry server REST API. For example, **https://registry.my-domain.com/apis/registry**
- **REGISTRY\_UI\_CONFIG\_UIURL**: Specifies the URL for the Service Registry web console client. For example, **https://registry.my-domain.com/ui**

### Configuring the web console in read-only mode

You can configure the Service Registry web console in read-only mode as an optional feature. This mode disables all features in the Service Registry web console that allow users to make changes to registered artifacts. For example, this includes the following:

- Creating an artifact
- Uploading a new version of an artifact
- Updating an artifact's metadata

- Deleting an artifact

## Procedure

Configure the following environment variable:

- **REGISTRY\_UI\_FEATURES\_READONLY**: Set to **true** to enable read-only mode. Defaults to **false**.

## 6.5. CONFIGURING SERVICE REGISTRY LOGGING

You can set Service Registry logging configuration at runtime. Service Registry provides a REST endpoint to set the log level for specific loggers for finer grained logging. This section explains how to view and set Service Registry log levels at runtime using the Service Registry **/admin** REST API.

### Prerequisites

- Get the URL to access your Service Registry instance, or get your Service Registry route if you have Service Registry deployed on OpenShift. This simple example uses a URL of **localhost:8080**.

### Procedure

1. Use this **curl** command to obtain the current log level for the logger **io.apicurio.registry.storage**:

```
$ curl -i localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"INFO"}
```

2. Use this **curl** command to change the log level for the logger **io.apicurio.registry.storage** to **DEBUG**:

```
$ curl -X PUT -i -H "Content-Type: application/json" --data '{"level":"DEBUG"}'
localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"DEBUG"}
```

3. Use this **curl** command to revert the log level for the logger **io.apicurio.registry.storage** to its default value:

```
$ curl -X DELETE -i localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"INFO"}
```

## 6.6. CONFIGURING SERVICE REGISTRY EVENT SOURCING

You can configure Service Registry to send events when changes are made to the registry. For example,

Service Registry can trigger events when schema and API artifacts are created, updated, deleted, and so on. You can configure Service Registry to send events to your applications and to third-party integrations in this way.

There are different protocols available for transporting the events. The currently implemented protocols are HTTP and Apache Kafka. However, regardless of the protocol, the events are sent using the CNCF CloudEvents specification.

All of the event types are defined in **io.apicurio.registry.events.dto.RegistryEventType**. For example, the event types include:

- **io.apicurio.registry.artifact-created**
- **io.apicurio.registry.artifact-updated**
- **io.apicurio.registry.artifact-rule-created**
- **io.apicurio.registry.global-rule-created**

You can configure cloud events in Service Registry using Java system properties or equivalent environment variables.

### Prerequisites

- You must have an application that you want to send Service Registry cloud events to. For example, this can be a custom application or a third-party application.

### Configuring Service Registry event sourcing using HTTP

The example in this section shows a custom application running at **http://my-app-host:8888/events**.

#### Procedure

1. When using the HTTP protocol, set your Service Registry configuration to send events to a your application as follows:
  - **registry.events.sink.my-custom-consumer=http://my-app-host:8888/events**
2. If required, you can configure multiple event consumers as follows:
  - **registry.events.sink.my-custom-consumer=http://my-app-host:8888/events**
  - **registry.events.sink.other-consumer=http://my-consumer.com/events**

### Configuring Service Registry event sourcing using Apache Kafka

The example in this section shows a Kafka topic named **my-registry-events** running on **my-kafka-host:9092**.

#### Procedure

1. When using the Kafka protocol, set your Kafka topic as follows:
  - **registry.events.kafka.topic=my-registry-events**
2. You can set the configuration for the Kafka producer using the **KAFKA\_BOOTSTRAP\_SERVERS** environment variable:
  - **KAFKA\_BOOTSTRAP\_SERVERS=my-kafka-host:9092**

Alternatively, you can set the properties for the kafka producer using the **registry.events.kafka.config** prefix, for example:  
**registry.events.kafka.config.bootstrap.servers=my-kafka-host:9092**

3. If required, you can also set the Kafka topic partition to use to produce events:

- **registry.events.kafka.topic-partition=1**

#### Additional resources

- For more details, see the [CNCF CloudEvents specification](#)

# CHAPTER 7. SERVICE REGISTRY OPERATOR CONFIGURATION REFERENCE

This chapter provides detailed information on the custom resource used to configure the Service Registry Operator to deploy Service Registry:

- [Section 7.1, "Service Registry Custom Resource"](#)
- [Section 7.2, "Service Registry CR spec"](#)
- [Section 7.3, "Service Registry CR status"](#)
- [Section 7.4, "Service Registry managed resources"](#)
- [Section 7.5, "Service Registry Operator labels"](#)

## 7.1. SERVICE REGISTRY CUSTOM RESOURCE

The Service Registry Operator defines an **ApicurioRegistry** [custom resource \(CR\)](#) that represents a single deployment of Service Registry on OpenShift.

These resource objects are created and maintained by users to instruct the Service Registry Operator how to deploy and configure Service Registry.

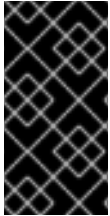
### Example ApicurioRegistry CR

The following command displays the **ApicurioRegistry** resource:

```
oc get apicurioregistry
oc edit apicurioregistry example-apicurioregistry

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
  namespace: demo-kafka
  # ...
spec:
  configuration:
    persistence: kafkasql
    kafkasql:
      bootstrapServers: 'my-cluster-kafka-bootstrap.demo-kafka.svc:9092'
  deployment:
    host: >-
      example-apicurioregistry.demo-kafka.example.com
status:
  conditions:
  - lastTransitionTime: "2021-05-03T10:47:11Z"
    message: ""
    reason: Reconciled
    status: "True"
    type: Ready
  info:
    host: example-apicurioregistry.demo-kafka.example.com
  managedResources:
```

- kind: Deployment  
name: example-apicurioregistry-deployment  
namespace: demo-kafka
- kind: Service  
name: example-apicurioregistry-service  
namespace: demo-kafka
- kind: Ingress  
name: example-apicurioregistry-ingress  
namespace: demo-kafka



## IMPORTANT

By default, the Service Registry Operator only watches its own project namespace. Therefore you must create the **ApicurioRegistry** CR in the same namespace, if you are deploying the operator manually. You can modify this behavior by updating **WATCH\_NAMESPACE** environment variable in the Operator **Deployment** resource.

### Additional resources

- [Extending the Kubernetes API with Custom Resource Definitions](#)

## 7.2. SERVICE REGISTRY CR SPEC

The **spec** is the part of the **ApicurioRegistry** CR that is used to provide the desired state or configuration for the Operator to achieve.

### ApicurioRegistry CR spec contents

The following example block contains the full tree of possible **spec** configuration options. Some fields may not be required or should not be defined at the same time.

```
spec:
  configuration:
    persistence: <string>
    sql:
      dataSource:
        url: <string>
        userName: <string>
        password: <string>
      kafkasql:
        bootstrapServers: <string>
    security:
      tls:
        truststoreSecretName: <string>
        keystoreSecretName: <string>
      scram:
        mechanism: <string>
        truststoreSecretName: <string>
        user: <string>
        passwordSecretName: <string>
    ui:
      readOnly: <string>
      logLevel: <string>
    security:
      keycloak:
```

```

url: <string>
realm: <string>
apiClientId: <string>
uiClientId: <string>
deployment:
  replicas: <int32>
  host: <string>
  affinity: <k8s.io/api/core/v1 Affinity struct>
  tolerations: <k8s.io/api/core/v1 []Toleration slice>

```

The following table describes each configuration option:

Table 7.1. ApicurioRegistry CR spec configuration options

Configuration option	type	Default value	Description
<b>configuration</b>	-	-	Section for configuration of Service Registry application
<b>configuration/persistence</b>	string	<i>required</i>	Storage backend. One of <b>sql, kafkasql</b>
<b>configuration/sql</b>	-	-	SQL storage backend configuration
<b>configuration/sql/dataSource</b>	-	-	Database connection configuration for SQL storage backend
<b>configuration/sql/dataSource/ur l</b>	string	<i>required</i>	Database connection URL string
<b>configuration/sql/dataSource/us erName</b>	string	<i>required</i>	Database connection user
<b>configuration/sql/dataSource/pa ssword</b>	string	<i>empty</i>	Database connection password
<b>configuration/kafkasql</b>	-	-	Kafka storage backend configuration
<b>configuration/kafkasql/bootstra pServers</b>	string	<i>required</i>	Kafka bootstrap server URL, for Streams storage backend
<b>configuration/kafkasql/security/ tls</b>	-	-	Section to configure TLS authentication for Kafka storage backend

Configuration option	type	Default value	Description
<b>configuration/kafkasql/security/tls/truststoreSecretName</b>	string	<i>required</i>	Name of a secret containing TLS truststore for Kafka
<b>configuration/kafkasql/security/tls/keystoreSecretName</b>	string	<i>required</i>	Name of a secret containing user TLS keystore
<b>configuration/kafkasql/security/scram/truststoreSecretName</b>	string	<i>required</i>	Name of a secret containing TLS truststore for Kafka
<b>configuration/kafkasql/security/scram/user</b>	string	<i>required</i>	SCRAM user name
<b>configuration/kafkasql/security/scram/passwordSecretName</b>	string	<i>required</i>	Name of a secret containing SCRAM user password
<b>configuration/kafkasql/security/scram/mechanism</b>	string	<b>SCRAM-SHA-512</b>	SASL mechanism
<b>configuration/ui</b>	-	-	Service Registry web console settings
<b>configuration/ui/readOnly</b>	string	<b>false</b>	Set Service Registry web console to read-only mode
<b>configuration/logLevel</b>	string	<b>INFO</b>	Service Registry log level. One of <b>INFO, DEBUG</b>
<b>configuration/security</b>	-	-	Service Registry web console and REST API security settings
<b>configuration/security/keycloak</b>	-	-	Web console and REST API security configuration using Keycloak
<b>configuration/security/keycloak/url</b>	string	<i>required</i>	Keycloak URL, must end with <b>/auth</b>
<b>configuration/security/keycloak/realm</b>	string	<i>required</i>	Keycloak realm



Configuration option	type	Default value	Description
<b>configuration/security/keycloak/apiClientId</b>	string	<b>registry-client-api</b>	Keycloak client for REST API
<b>configuration/security/keycloak/uiClientId</b>	string	<b>registry-client-ui</b>	Keycloak client for web console
<b>deployment</b>	-	-	Section for Service Registry deployment settings
<b>deployment/replicas</b>	positive integer	<b>1</b>	Number of Service Registry pods to deploy
<b>deployment/host</b>	string	<i>auto-generated</i>	Host/URL where the Service Registry console and API are available. If possible, Service Registry Operator attempts to determine the correct value based on the settings of your cluster router. The value is auto-generated only once, so user can override it afterwards.
<b>deployment/affinity</b>	k8s.io/api/core/v1 Affinity struct	<i>empty</i>	Service Registry deployment affinity configuration
<b>deployment/tolerations</b>	k8s.io/api/core/v1 []Toleration slice	<i>empty</i>	Service Registry deployment tolerations configuration



#### NOTE

If an option is marked as *required*, it might be conditional on other configuration options being enabled. Empty values might be accepted, but the Operator does not perform the specified action.

### 7.3. SERVICE REGISTRY CR STATUS

The **status** is the section of the CR managed by the Service Registry Operator that contains a description of the current deployment and application state.

#### ApicurioRegistry CR status contents

The **status** section contains the following fields:

```

status:
  info:
    host: <string>
  conditions: <list of:>
  - type: <string>
    status: <string, one of: True, False, Unknown>
    reason: <string>
    message: <string>
    lastTransitionTime: <string, RFC-3339 timestamp>
  managedResources: <list of:>
  - kind: <string>
    namespace: <string>
    name: <string>

```

Table 7.2. ApicurioRegistry CR status fields

Status field	Type	Description
<b>info</b>	-	Section with information about the deployed Service Registry.
<b>info/host</b>	string	URL where the Service Registry UI and REST API are accessible.
<b>conditions</b>	-	List of conditions that report the status of the Service Registry, or the Operator with respect to that deployment.
<b>conditions/type</b>	string	Type of the condition.
<b>conditions/status</b>	string	Status of the condition, one of <b>True</b> , <b>False</b> , <b>Unknown</b> .
<b>conditions/reason</b>	string	A programmatic identifier indicating the reason for the condition's last transition.
<b>conditions/message</b>	string	A human readable message indicating details about the transition.
<b>conditions/lastTransitionTime</b>	string	The last time the condition transitioned from one status to another.
<b>managedResources</b>	-	List of OpenShift resources managed by Service Registry Operator
<b>managedResources/kind</b>	string	Resource kind.
<b>managedResources/namespace</b>	string	Resource namespace.

Status field	Type	Description
<b>managedResources/name</b>	string	Resource name.

## 7.4. SERVICE REGISTRY MANAGED RESOURCES

The resources managed by the Service Registry Operator when deploying Service Registry are as follows:

- **Deployment**
- **Service**
- **Ingress** (and **Route**)
- **PodDisruptionBudget**

## 7.5. SERVICE REGISTRY OPERATOR LABELS

Resources managed by the Service Registry Operator are usually labeled as follows:

Table 7.3. Service Registry Operator labels for managed resources

Label	Description
<b>app</b>	Name of the Service Registry deployment that the resource belongs to, based on the name of the specified <b>ApicurioRegistry</b> CR.
<b>apicur.io/type</b>	Type of the deployment: <b>apicurio-registry</b> or <b>operator</b>
<b>apicur.io/name</b>	Name of the deployment: same value as <b>app</b> or <b>apicurio-registry-operator</b>
<b>apicur.io/version</b>	Version of the Service Registry or the Service Registry Operator
<b>app.kubernetes.io/*</b>	A set of recommended Kubernetes labels for application deployments.
<b>com.company</b> and <b>rht.*</b>	Metering labels for Red Hat products.

### Additional resources

- [Recommended Kubernetes labels for application deployments](#)

## APPENDIX A. USING YOUR SUBSCRIPTION

Service Registry is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

### Accessing your account

1. Go to [access.redhat.com](https://access.redhat.com).
2. If you do not already have an account, create one.
3. Log in to your account.

### Activating a subscription

1. Go to [access.redhat.com](https://access.redhat.com).
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

### Downloading ZIP and TAR files

To access ZIP or TAR files, use the customer portal to find the relevant files for download. If you are using RPM packages, this step is not required.

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at [access.redhat.com/downloads](https://access.redhat.com/downloads).
2. Locate the **Red Hat Integration** entries in the **Integration and Automation** category.
3. Select the desired Service Registry product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

### Registering your system for packages

To install RPM packages on Red Hat Enterprise Linux, your system must be registered. If you are using ZIP or TAR files, this step is not required.

1. Go to [access.redhat.com](https://access.redhat.com).
2. Navigate to **Registration Assistant**.
3. Select your OS version and continue to the next page.
4. Use the listed command in your system terminal to complete the registration.

To learn more see [How to Register and Subscribe a System to the Red Hat Customer Portal](#) .