Red Hat build of Apicurio Registry 2.4

Installing and deploying Apicurio Registry on OpenShift

Install, deploy, and configure Apicurio Registry 2.4
Install, deploy, and configure Apicurio Registry 2.4
Abstract

This guide explains how to install and deploy Apicurio Registry on OpenShift with data storage options in AMQ Streams or a PostgreSQL database. This guide also shows how to secure, configure, and manage your Apicurio Registry deployment, and provides configuration reference for Apicurio Registry and the Apicurio Registry Operator.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PREFACE</strong></td>
<td>4</td>
</tr>
<tr>
<td>Making Open Source More Inclusive</td>
<td>4</td>
</tr>
<tr>
<td>Providing Feedback on Red Hat Documentation</td>
<td>4</td>
</tr>
<tr>
<td><strong>CHAPTER 1. APICURIO REGISTRY OPERATOR QUICKSTART</strong></td>
<td>5</td>
</tr>
<tr>
<td>1.1. Quickstart Apicurio Registry Operator Installation</td>
<td>5</td>
</tr>
<tr>
<td>1.2. Quickstart Apicurio Registry Instance Deployment</td>
<td>6</td>
</tr>
<tr>
<td><strong>CHAPTER 2. INSTALLING APICURIO REGISTRY ON OPENSHEET</strong></td>
<td>8</td>
</tr>
<tr>
<td>2.1. Installing Apicurio Registry From the OpenShift OperatorHub</td>
<td>8</td>
</tr>
<tr>
<td><strong>CHAPTER 3. DEPLOYING APICURIO REGISTRY STORAGE IN AMQ STREAMS</strong></td>
<td>10</td>
</tr>
<tr>
<td>3.1. Installing AMQ Streams From the OpenShift OperatorHub</td>
<td>10</td>
</tr>
<tr>
<td>3.2. Configuring Apicurio Registry With Kafka Storage On OpenShift</td>
<td>11</td>
</tr>
<tr>
<td>3.3. Configuring Kafka Storage With TLS Security</td>
<td>13</td>
</tr>
<tr>
<td>3.4. Configuring Kafka Storage With Scram Security</td>
<td>17</td>
</tr>
<tr>
<td>3.5. Configuring OAuth Authentication For Kafka Storage</td>
<td>20</td>
</tr>
<tr>
<td><strong>CHAPTER 4. DEPLOYING APICURIO REGISTRY STORAGE IN A POSTGRESQL DATABASE</strong></td>
<td>22</td>
</tr>
<tr>
<td>4.1. Installing A Postgresql Database From the OpenShift OperatorHub</td>
<td>22</td>
</tr>
<tr>
<td>4.2. Configuring Apicurio Registry With Postgresql Database Storage On OpenShift</td>
<td>23</td>
</tr>
<tr>
<td>4.3. Backing Up Apicurio Registry Postgresql Storage</td>
<td>24</td>
</tr>
<tr>
<td>4.4. Restoring Apicurio Registry Postgresql Storage</td>
<td>25</td>
</tr>
<tr>
<td><strong>CHAPTER 5. SECURING APICURIO REGISTRY DEPLOYMENTS</strong></td>
<td>26</td>
</tr>
<tr>
<td>5.2. Configuring Apicurio Registry Authentication And Authorization With Red Hat Single Sign-On</td>
<td>30</td>
</tr>
<tr>
<td>5.3. Apicurio Registry Authentication And Authorization Configuration Options</td>
<td>33</td>
</tr>
<tr>
<td>Apicurio Registry authentication by using OpenID Connect with Red Hat Single Sign-On</td>
<td>34</td>
</tr>
<tr>
<td>Apicurio Registry authentication by using HTTP basic</td>
<td>34</td>
</tr>
<tr>
<td>Apicurio Registry HTTP basic client credentials cache expiry</td>
<td>35</td>
</tr>
<tr>
<td>Apicurio Registry role-based authorization</td>
<td>35</td>
</tr>
<tr>
<td>Use roles assigned in Red Hat Single Sign-On</td>
<td>35</td>
</tr>
<tr>
<td>Manage roles directly in Apicurio Registry</td>
<td>36</td>
</tr>
<tr>
<td>Apicurio Registry admin-override configuration</td>
<td>37</td>
</tr>
<tr>
<td>Apicurio Registry owner-only authorization</td>
<td>37</td>
</tr>
<tr>
<td>Apicurio Registry authenticated read access</td>
<td>38</td>
</tr>
<tr>
<td>Apicurio Registry anonymous read-only access</td>
<td>38</td>
</tr>
<tr>
<td>5.4. Configuring An HTTPS Connection To Apicurio Registry From Inside The OpenShift Cluster</td>
<td>39</td>
</tr>
<tr>
<td>5.5. Configuring An HTTPS Connection To Apicurio Registry From Outside The OpenShift Cluster</td>
<td>41</td>
</tr>
<tr>
<td><strong>CHAPTER 6. CONFIGURING AND MANAGING APICURIO REGISTRY DEPLOYMENTS</strong></td>
<td>43</td>
</tr>
<tr>
<td>6.1. Configuring Apicurio Registry Health Checks On OpenShift</td>
<td>43</td>
</tr>
<tr>
<td>6.2. Environment Variables For Apicurio Registry Health Checks</td>
<td>44</td>
</tr>
<tr>
<td>Liveness environment variables</td>
<td>44</td>
</tr>
<tr>
<td>Readiness environment variables</td>
<td>45</td>
</tr>
<tr>
<td>6.3. Managing Apicurio Registry Environment Variables</td>
<td>46</td>
</tr>
<tr>
<td>6.4. Configuring Apicurio Registry Deployment Using PodTemplate</td>
<td>47</td>
</tr>
<tr>
<td>6.5. Configuring The Apicurio Registry Web Console</td>
<td>49</td>
</tr>
<tr>
<td>Configuring the web console deployment environment</td>
<td>49</td>
</tr>
</tbody>
</table>
CHAPTER 7. APICURIO REGISTRY OPERATOR CONFIGURATION REFERENCE ............................................. 53
  7.1. APICURIO REGISTRY CUSTOM RESOURCE .......................................................... 53
  7.2. APICURIO REGISTRY CR SPEC ........................................................................... 54
  7.3. APICURIO REGISTRY CR STATUS ....................................................................... 59
  7.4. APICURIO REGISTRY MANAGED RESOURCES .................................................. 61
  7.5. APICURIO REGISTRY OPERATOR LABELS ......................................................... 62

CHAPTER 8. APICURIO REGISTRY CONFIGURATION REFERENCE ................................................... 63
  8.1. APICURIO REGISTRY CONFIGURATION OPTIONS ............................................. 63
    8.1.1. api ..................................................................................................................... 63
    8.1.2. auth .................................................................................................................. 63
    8.1.3. cache ............................................................................................................... 65
    8.1.4. ccompat .......................................................................................................... 65
    8.1.5. download ....................................................................................................... 66
    8.1.6. events ............................................................................................................. 66
    8.1.7. health ............................................................................................................ 66
    8.1.8. import ............................................................................................................ 68
    8.1.9. kafka .............................................................................................................. 68
    8.1.10. limits ........................................................................................................... 68
    8.1.11. log ................................................................................................................ 69
    8.1.12. redirects ...................................................................................................... 70
    8.1.13. rest ................................................................................................................ 70
    8.1.14. store ............................................................................................................. 70
    8.1.15. ui .................................................................................................................... 71

APPENDIX A. USING YOUR SUBSCRIPTION ............................................................................. 73
  Accessing your account .......................................................................................... 73
  Activating a subscription ..................................................................................... 73
  Downloading ZIP and TAR files ............................................................................ 73
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.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION
We appreciate your feedback on our documentation. To provide feedback, highlight text in a document and add comments.

Prerequisites
- You are logged in to the Red Hat Customer Portal.
- In the Red Hat Customer Portal, the document is in the Multi-page HTML viewing format.

Procedure
To provide your feedback, perform the following steps:

1. Click the Feedback button in the top-right corner of the document to see existing feedback.

   NOTE
   The feedback feature is enabled only in the Multi-page HTML format.

2. Highlight the section of the document where you want to provide feedback.

3. Click the Add Feedback pop-up that appears near the highlighted text. A text box appears in the feedback section on the right side of the page.

4. Enter your feedback in the text box and click Submit. A documentation issue is created.

5. To view the issue, click the issue link in the feedback view.
You can quickly install and deploy the Apicurio Registry Operator on the command line, without the Operator Lifecycle Manager, by using a downloaded set of installation files and example CRDs.

**Prerequisites**

- You are logged in to an OpenShift cluster with administrator access.
- You have the OpenShift `oc` command-line client installed. For more details, see the OpenShift CLI documentation.

**Procedure**

1. Browse to Red Hat Software Downloads, select the product version, and download the examples in the Apicurio Registry CRDs `.zip` file.

2. Extract the downloaded CRDs `.zip` file and change to the `apicurio-registry-install-examples` directory.

3. Create an OpenShift project for the Apicurio Registry Operator installation, for example:

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

4. Enter the following command to apply the example CRD in the `install/install.yaml` file:

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

5. Enter `oc get deployment` to check the readiness of the Apicurio Registry Operator. For example, the output should be as follows:

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>apicurio-registry-operator</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>XmYs</td>
</tr>
</tbody>
</table>
1.2. QUICKSTART APICURIO REGISTRY INSTANCE DEPLOYMENT

To create your Apicurio Registry instance deployment, use the SQL database storage option to connect to an existing PostgreSQL database.

Prerequisites

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

Procedure

1. Open the examples/apicurioregistry_sql_cr.yaml file in an editor and view the ApicurioRegistry custom resource (CR):

   **Example CR for SQL storage**

   ```yaml
   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. In the dataSource section, replace the example settings with your database connection details. For example:

   ```yaml
   dataSource:
     url: "jdbc:postgresql://postgresql.apicurio-registry.svc:5432/registry"
     userName: "pgadmin"
     password: "pgpass"
   
   3. Enter the following commands to apply the updated ApicurioRegistry CR in the namespace with the Apicurio Registry Operator, and wait for the Apicurio Registry instance to deploy:

   ```bash
   oc project "$NAMESPACE"
   oc apply -f ./examples/apicurioregistry_sql_cr.yaml
   
   4. Enter `oc get deployment` to check the readiness of the Apicurio Registry instance. For example, the output should be as follows:

   ```bash
   NAME                              READY UP-TO-DATE AVAILABLE AGE
   example-apicurioregistry-sql-deployment 1/1   1          1         XmYs
   
   5. Enter `oc get routes` to get the HOST/PORT URL to launch the Apicurio Registry web console in your browser. For example:
example-apicurioregistry-sql.apicurio-registry.router-default.apps.mycluster.myorg.mycompany.com
CHAPTER 2. INSTALLING APICURIO REGISTRY ON OPENSIFT

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

- Section 2.1, “Installing Apicurio Registry from the OpenShift OperatorHub”

Prerequisites

- Read the introduction in the Red Hat build of Apicurio Registry User Guide.

2.1. INSTALLING APICURIO REGISTRY FROM THE OPENSHIFT OPERATORHUB

You can install the Apicurio 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 Apicurio Registry depending on your environment. The number of instances depends on the number and type of artifacts stored in Apicurio 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 APICURIO REGISTRY STORAGE IN AMQ STREAMS

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

- Section 3.1, "Installing AMQ Streams from the OpenShift OperatorHub"
- Section 3.2, "Configuring Apicurio 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 Apicurio Registry on OpenShift

3.1. INSTALLING AMQ STREAMS FROM THE OPENSHIFT 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 Managing 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-2.4.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 Managing AMQ Streams on OpenShift

### 3.2. CONFIGURING APICURIO REGISTRY WITH KAFKA STORAGE ON OPENSHIFT

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

You can install Apicurio 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 Apicurio Registry. See Chapter 2, *Installing Apicurio 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**.
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 Apicurio 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 Apicurio Registry route to be created on OpenShift.

8. Click **Networking > Route** to access the new route for the Apicurio Registry web console. For example:

```
http://example-apicurioregistry-kafkasql.my-project.my-domain-name.com/
```
To configure the Kafka topic that Apicurio Registry uses to store data, click **Installed Operators > Red Hat Integration - AMQ Streams > Provided APIs > Kafka Topic > kafkasql-journal** **YAML**. For example:

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: kafkasql-journal
labels:
  strimzi.io/cluster: my-cluster
namespace: ...
spec:
  partitions: 3
  replicas: 3
  config:
    cleanup.policy: compact
```

**WARNING**

You must configure the Kafka topic used by Apicurio Registry (named `kafkasql-journal` by default) with a compaction cleanup policy, otherwise a data loss might occur.

Additional resources

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

### 3.3. CONFIGURING KAFKA STORAGE WITH TLS SECURITY

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

**Prerequisites**

- You have installed the Apicurio Registry Operator using the OperatorHub or command line.
- You have installed 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 Apicurio 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 Apicurio Registry storage.

3. Configure the **authorization** and **tls** fields to use TLS authentication for the Kafka cluster, for example:

   ```yaml
   apiVersion: kafka.strimzi.io/v1beta2
   kind: Kafka
   metadata:
     name: my-cluster
     namespace: registry-example-kafkasql-tls
   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 automatically created by Apicurio Registry to store data is **kafkasql-journal**. You can override this behavior or the default topic name by setting environment variables. The default values are as follows:

   - `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:

   ```yaml
   apiVersion: kafka.strimzi.io/v1beta1
   ```
5. Create a **Kafka User** resource to configure authentication and authorization for the Apicurio Registry user. You can specify a user name in the **metadata** section or use the default **my-user**.

```yaml
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: transactionalId
    - operation: All
      resource:
        name: "*
        patternType: literal
        type: group
  type: simple
```

**NOTE**

This simple example assumes admin permissions and creates the Kafka topic automatically. You must configure the **authorization** section specifically for the topics and resources that the Apicurio Registry requires.
The following example shows the minimum configuration required when the Kafka topic is created manually:

```
...  
authorization:
  acls:
    - operations:
      - Read
      - Write
      resource:
        name: kafkasql-journal
        patternType: literal
        type: topic
    - operations:
      - Read
      - Write
      resource:
        name: apicurio-registry-
        patternType: prefix
        type: group
        type: simple
```

6. Click Workloads and then Secrets to find two secrets that AMQ Streams creates for Apicurio 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** - truststore in PKCS12 format
  
  - **ca.password** - truststore password

- **my-user**
  
  - **user.p12** - keystore in PKCS12 format
  
  - **user.password** - keystore password

8. Configure the following example configuration to deploy the Apicurio Registry.

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-tls
spec:
  configuration:
    persistence: "kafkasql"
```
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 Apicurio Registry Operator to use Salted Challenge Response Authentication Mechanism (SCRAM-SHA-512) for the Kafka cluster.

**Prerequisites**

- You have installed the Apicurio Registry Operator using the OperatorHub or command line.
- You have installed 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 Apicurio 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 Apicurio Registry storage.

3. Configure the **authorization** and **tls** fields to use SCRAM-SHA-512 authentication for the Kafka cluster, for example:

```yaml
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
namespace: registry-example-kafkaExampleScram
# 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'
```
The default Kafka topic name automatically created by Apicurio Registry to store data is `kafkasql-journal`. You can override this behavior or the default topic name by setting environment variables. The default values are as follows:

- `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:

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

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

```yaml
apiVersion: kafka.strimzi.io/v1beta1
description: KafkaUser
metadata:
  name: my-user
  labels:
```
NOTE

This simple example assumes admin permissions and creates the Kafka topic automatically. You must configure the authorization section specifically for the topics and resources that the Apicurio Registry requires.

The following example shows the minimum configuration required when the Kafka topic is created manually:

```
strimzi.io/cluster: my-cluster
namespace: registry-example-kafkasql-scram
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
```
6. Click Workloads and then Secrets to find two secrets that AMQ Streams creates for Apicurio 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 Apicurio Registry:

```yaml
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-scram
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, Apicurio Registry supports accessing a Kafka cluster that requires OAuth authentication. To enable this support, you must set some environment variables in your Apicurio Registry deployment.

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

**Prerequisites**

- You must have already configured Kafka-based storage of Apicurio Registry data in AMQ Streams. See Section 3.2, “Configuring Apicurio Registry with Kafka storage on OpenShift”.

**Procedure**

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

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLE_KAFKA_SASL</td>
<td>Enables SASL OAuth authentication for Apicurio Registry storage in Kafka. You must set this variable to true for the other variables to have effect.</td>
<td>false</td>
</tr>
<tr>
<td>CLIENT_ID</td>
<td>The client ID used to authenticate to Kafka.</td>
<td>-</td>
</tr>
<tr>
<td>CLIENT_SECRET</td>
<td>The client secret used to authenticate to Kafka.</td>
<td>-</td>
</tr>
</tbody>
</table>

**Additional resources**

- For an example of how to set Apicurio Registry environment variables on OpenShift, see Section 6.1, “Configuring Apicurio Registry health checks on OpenShift”
CHAPTER 4. DEPLOYING APICURIO REGISTRY STORAGE IN A POSTGRESQL DATABASE

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

- Section 4.1, “Installing a PostgreSQL database from the OpenShift OperatorHub”
- Section 4.2, “Configuring Apicurio Registry with PostgreSQL database storage on OpenShift”
- Section 4.3, “Backing up Apicurio Registry PostgreSQL storage”
- Section 4.4, “Restoring Apicurio Registry PostgreSQL storage”

Prerequisites

- Chapter 2, Installing Apicurio Registry on OpenShift

4.1. INSTALLING A POSTGRESQL DATABASE FROM THE OPENSHIFT 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

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

7. 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 APICURIO REGISTRY WITH POSTGRESQL DATABASE STORAGE ON OPENSHIFT

This section explains how to configure storage for Apicurio Registry on OpenShift using a PostgreSQL database Operator. You can install Apicurio 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 Apicurio Registry. See Chapter 2, Installing Apicurio 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

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 Apicurio Registry and your PostgreSQL Operator are installed. For example, from the Project drop-down, select my-project.

3. Create a PostgreSQL database for your Apicurio Registry storage. For example, click Installed Operators, PostgreSQL Operator by Dev4Ddevs.com, and then Create database.

4. Click YAML and edit the database settings as follows:

   - name: Change the value to registry
   - image: Change the value to centos/postgresql-12-centos7

5. 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
```
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:

   ```yaml
   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 Apicurio Registry route to be created on OpenShift.

10. Click **Networking** > **Route** to access the new route for the Apicurio 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 APICURIO REGISTRY POSTGRESQL STORAGE

When using storage in a PostgreSQL database, you must ensure that the data stored by Apicurio 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 Apicurio Registry content using the REST API.

### 4.4. RESTORING APICURIO REGISTRY POSTGRESQL 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, "Back up Apicurio 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 APICURIO REGISTRY DEPLOYMENTS

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

- Section 5.1, “Securing Apicurio Registry using the Red Hat Single Sign-On Operator”
- Section 5.2, “Configuring Apicurio Registry authentication and authorization with Red Hat Single Sign-On”
- Section 5.3, “Apicurio Registry authentication and authorization configuration options”
- Section 5.4, “Configuring an HTTPS connection to Apicurio Registry from inside the OpenShift cluster”
- Section 5.5, “Configuring an HTTPS connection to Apicurio Registry from outside the OpenShift cluster”

Apicurio 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 Apicurio Registry.

Apicurio Registry provides role-based authentication and authorization for the Apicurio Registry web console and core REST API using Red Hat Single Sign-On. Apicurio 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 Apicurio Registry from inside or outside an OpenShift cluster.

Additional resources

- For details on security configuration for Java client applications, see the following:
  - Apicurio Registry Java client configuration
  - Apicurio Registry serializer/deserializer configuration

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

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

Apicurio Registry supports the following user roles:

**Table 5.1. Apicurio Registry user roles**

<table>
<thead>
<tr>
<th>Name</th>
<th>Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>sr-admin</td>
<td>Full access, no restrictions.</td>
</tr>
<tr>
<td>sr-developer</td>
<td>Create artifacts and configure artifact rules. Cannot modify global rules, perform import/export, or use /admin REST API endpoint.</td>
</tr>
</tbody>
</table>
### 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 Apicurio 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 Apicurio Registry deployment. You can use the default value, for example:

```yaml
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 URL value for later use when deploying Apicurio 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:

```yaml
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:
```

Red Hat build of Apicurio Registry 2.4 Installing and deploying Apicurio Registry on OpenShift
- 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:

   ```yaml
   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:

   ```yaml
   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 Apicurio Registry user, and use it instead of the HTTPS route created by Red Hat Single Sign-On Operator.

7. Click the Apicurio 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>"
        # ^ Required
        # 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 APICURIO REGISTRY AUTHENTICATION AND AUTHORIZATION WITH RED HAT SINGLE SIGN-ON

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

**NOTE**

Alternatively, for details on how to configure these settings automatically, see Section 5.1, “Securing Apicurio Registry using the Red Hat Single Sign-On Operator”.

You can enable authentication for the Apicurio 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 Apicurio Registry web console and core REST API using OpenID Connect so that you only require one set of credentials.

Apicurio Registry provides role-based authorization for default admin, write, and read-only user roles. Apicurio 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. Apicurio 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](#).
- Apicurio 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 Apicurio Registry. By default, Apicurio Registry expects a realm name of `registry`. For details on creating realms, see the [Red Hat Single Sign-On user documentation](#).

2. Create a Red Hat Single Sign-On client for the Apicurio Registry API. By default, Apicurio 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 Apicurio Registry web console. By default, Apicurio 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 Apicurio Registry deployment on OpenShift, set the following Apicurio Registry environment variables to configure authentication using Red Hat Single Sign-On:

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

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_ENABLED</td>
<td>If set to <code>true</code>, the environment variables that follow are required.</td>
<td>String</td>
<td>false</td>
</tr>
</tbody>
</table>
### Environment variable

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KEYCLOAK_URL</strong></td>
<td>The URL of the Red Hat Single Sign-On authentication server to use.</td>
<td>None</td>
<td><strong>KEYCLOAK_REALM</strong></td>
</tr>
<tr>
<td></td>
<td>The Red Hat Single Sign-On realm used for authentication.</td>
<td>String</td>
<td><strong>registry</strong></td>
</tr>
<tr>
<td></td>
<td>The client ID for the Apicurio Registry REST API.</td>
<td>String</td>
<td><strong>registry-api</strong></td>
</tr>
</tbody>
</table>

#### TIP

For an example of setting environment variables on OpenShift, see Section 6.1, "Configuring Apicurio Registry health checks on OpenShift".

5. Set the following option to `true` to enable Apicurio Registry user roles in Red Hat Single Sign-On:

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

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Java system property</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ROLE_BASED_AUTHZ_ENABLED</strong></td>
<td><code>registry.auth.role-based-authorization</code></td>
<td>Boolean</td>
<td><strong>false</strong></td>
</tr>
</tbody>
</table>

6. When Apicurio Registry user roles are enabled, you must assign Apicurio 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**

<table>
<thead>
<tr>
<th>Role</th>
<th>Read artifacts</th>
<th>Write artifacts</th>
<th>Global rules</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sr-admin</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Full access to all create, read, update, and delete operations.</td>
</tr>
<tr>
<td><strong>sr-developer</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Access to create, read, update, and delete operations, except configuring global rules. This role can configure artifact-specific rules.</td>
</tr>
<tr>
<td>Role</td>
<td>Read artifacts</td>
<td>Write artifacts</td>
<td>Global rules</td>
<td>Summary</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>--------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>sr-readonly</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Access to read and search operations only. This role cannot configure any rules.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Java system property</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTRY_AUTH_OBAC_ENABLED</td>
<td>registry.auth.owner-only-authorization</td>
<td>Boolean</td>
<td>false</td>
</tr>
</tbody>
</table>

Additional resources

- For details on configuring non-default user role names, see Section 5.3, “Apicurio Registry authentication and authorization configuration options”.
- For an open source example application and Keycloak realm, see Docker Compose example of Apicurio Registry with Keycloak.
- For details on how to use Red Hat Single Sign-On in a production environment, see the Red Hat Single Sign-On documentation.

### 5.3. APICURIO REGISTRY AUTHENTICATION AND AUTHORIZATION CONFIGURATION OPTIONS

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

Apicurio 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.

**IMPORTANT**

All authentication and authorization options in Apicurio Registry are disabled by default. Before enabling any of these options, you must first set the **AUTH_ENABLED** option to **true**.

This chapter provides details on the following configuration options:
Apicurio Registry authentication by using OpenID Connect with Red Hat Single Sign-On

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

Table 5.6. Configuration for Apicurio Registry authentication with Red Hat Single Sign-On

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_ENABLED</td>
<td>Enables authentication in Apicurio Registry. When set to <strong>true</strong>, the environment variables that follow are required for authentication in Red Hat Single Sign-On.</td>
<td>String</td>
<td>false</td>
</tr>
<tr>
<td>KEYCLOAK_URL</td>
<td>The URL of the Red Hat Single Sign-On authentication server. For example, <strong><a href="http://localhost:8080">http://localhost:8080</a></strong>.</td>
<td>String</td>
<td>-</td>
</tr>
<tr>
<td>KEYCLOAK_REALM</td>
<td>The Red Hat Single Sign-On realm for authentication. For example, <strong>registry</strong>.</td>
<td>String</td>
<td>-</td>
</tr>
<tr>
<td>KEYCLOAK_API_CLIENT_ID</td>
<td>The client ID for the Apicurio Registry REST API.</td>
<td>String</td>
<td>registry-api</td>
</tr>
<tr>
<td>KEYCLOAK_UI_CLIENT_ID</td>
<td>The client ID for the Apicurio Registry web console.</td>
<td>String</td>
<td>apicurio-registry</td>
</tr>
</tbody>
</table>

Apicurio Registry authentication by using HTTP basic

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

Table 5.7. Configuration for Apicurio Registry HTTP basic authentication

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Java system property</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
</table>
You can also configure the HTTP basic client credentials cache expiry time. By default, when using HTTP basic authentication, Apicurio Registry caches JWT tokens, and does not issue a new token when there is no need. You can configure the cache expiry time for JWT tokens, which is set to 10 mins by default.

When using Red Hat Single Sign-On, it is best to set this configuration to your Red Hat Single Sign-On JWT expiry time minus one minute. For example, if you have the expiry time set to 5 mins in Red Hat Single Sign-On, you should set the following configuration option to 4 mins:

Apicurio Registry role-based authorization
You can set the following options to true to enable role-based authorization in Apicurio Registry:

You can then configure role-based authorization to use roles included in the user’s authentication token (for example, granted when authenticating by using Red Hat Single Sign-On), or to use role mappings managed internally by Apicurio 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>
<thead>
<tr>
<th>Environment variable</th>
<th>Java system property</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_ENABLED</td>
<td>registry.auth.enabled</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>CLIENT_CREDENTIALS_BASIC_AUTH_ENABLED</td>
<td>registry.auth.basic-auth-client-credentials.enabled</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>CLIENT_CREDENTIALS_BASIC_CACHE_EXPIRATION</td>
<td>registry.auth.basic-auth-client-credentials.cache-expiration</td>
<td>Integer</td>
<td>10</td>
</tr>
<tr>
<td>AUTH_ENABLED</td>
<td>registry.auth.enabled</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>ROLE_BASED_AUTHZ_ENABLED</td>
<td>registry.auth.role-based-authorization</td>
<td>Boolean</td>
<td>false</td>
</tr>
</tbody>
</table>

Apicurio Registry role-based authorization
You can set the following options to true to enable role-based authorization in Apicurio Registry:

You can then configure role-based authorization to use roles included in the user’s authentication token (for example, granted when authenticating by using Red Hat Single Sign-On), or to use role mappings managed internally by Apicurio 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:
### Environment variable

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

When Apicurio Registry is configured to use roles from Red Hat Single Sign-On, you must assign Apicurio 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 by using the environment variables in Table 5.10, "Configuration for Apicurio Registry role-based authorization by using Red Hat Single Sign-On".

### Table 5.11. Apicurio Registry roles for authentication and authorization

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

### Manage roles directly in Apicurio Registry

To enable using roles managed internally by Apicurio Registry, set the following environment variable:

### Table 5.12. Configuration for Apicurio Registry role-based authorization by using internal role mappings

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROLE_BASED_AUTHZ_SOURCE</td>
<td>When set to <code>application</code>, user roles are managed internally by Apicurio Registry.</td>
<td>String</td>
<td>token</td>
</tr>
</tbody>
</table>
When using internally managed role mappings, users can be assigned a role by using the `/admin/roleMappings` endpoint in the Apicurio 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.

**Apicurio Registry admin-override configuration**

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

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTRY_AUTH_ADMIN_OVVERRIDE_ENABLED</td>
<td>Enables the admin-override feature.</td>
<td>String</td>
<td>false</td>
</tr>
<tr>
<td>REGISTRY_AUTH_ADMIN_OVVERRIDE_FROM</td>
<td>Where to look for admin-override information. Only <strong>token</strong> is currently supported.</td>
<td>String</td>
<td>token</td>
</tr>
<tr>
<td>REGISTRY_AUTH_ADMIN_OVVERRIDE_TYPE</td>
<td>The type of information used to determine if a user is an admin. Values depend on the value of the FROM variable, for example, <strong>role</strong> or <strong>claim</strong> when FROM is <strong>token</strong>.</td>
<td>String</td>
<td>role</td>
</tr>
<tr>
<td>REGISTRY_AUTH_ADMIN_OVVERRIDE_ROLE</td>
<td>The name of the role that indicates a user is an admin.</td>
<td>String</td>
<td>sr-admin</td>
</tr>
<tr>
<td>REGISTRY_AUTH_ADMIN_OVVERRIDE_CLAIM</td>
<td>The name of a JWT token claim to use for determining admin-override.</td>
<td>String</td>
<td>org-admin</td>
</tr>
<tr>
<td>REGISTRY_AUTH_ADMIN_OVVERRIDE_CLAIM_VALUE</td>
<td>The value that the JWT token claim indicated by the CLAIM variable must be for the user to be granted admin-override.</td>
<td>String</td>
<td>true</td>
</tr>
</tbody>
</table>

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).

**Apicurio 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 Apicurio Registry:

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGISTRY_OWNER_OVVERRIDE_ENABLED</td>
<td>Enables the owner-only feature.</td>
</tr>
</tbody>
</table>

CHAPTER 5. SECURING APICURIO REGISTRY DEPLOYMENTS
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.

**Apicurio Registry authenticated read access**

When the authenticated read access option is enabled, Apicurio 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 ensure that the following options are set to true:

**Table 5.15. Configuration for authenticated read access**

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Java system property</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_ENABLED</td>
<td>registry.auth.enabled</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>REGISTRY_AUTH_OBAC_ENABLED</td>
<td>registry.auth.owner-only-authorization</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>REGISTRY_AUTH_OBAC_LIMIT_GROUP_ACCESS</td>
<td>registry.auth.owner-only-authorization.limit-group-access</td>
<td>Boolean</td>
<td>false</td>
</tr>
</tbody>
</table>

For more details, see the section called “Apicurio Registry role-based authorization”.

**Apicurio Registry anonymous read-only access**

In addition to the two main types of authorization (role-based and owner-based authorization), Apicurio 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 options to true:

**Table 5.16. Configuration for anonymous read-only access**

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Java system property</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_ENABLED</td>
<td>registry.auth.enabled</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>REGISTRY_AUTH_AUTHENTICATED_READS_ENABLED</td>
<td>registry.auth.authenticated-read-access.enabled</td>
<td>Boolean</td>
<td>false</td>
</tr>
</tbody>
</table>
### 5.4. CONFIGURING AN HTTPS CONNECTION TO APICURIO REGISTRY FROM INSIDE THE OPENSHIFT CLUSTER

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

#### Prerequisites

- You must have already installed the Apicurio Registry Operator.

#### Procedure

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

   ```bash
   openssl req -newkey rsa:2048 -new -nodes -x509 -days 3650 -keyout tls.key -out tls.crt
   ```

2. Create a new secret to hold the certificate and the private key.
   
   a. In the left navigation menu of the OpenShift web console, click **Workloads > Secrets > Create Key/Value Secret**
b. Use the following values:
   Name: https-cert-secret
   Key 1: tls.key
   Value 1: tls.key (uploaded file)
   Key 2: tls.crt
   Value 2: tls.crt (uploaded file)

or create the secret using the following command:

```
oc create secret generic https-cert-secret --from-file=tls.key --from-file=tls.crt
```

3. Edit the `spec.configuration.security.https` section of the ApicurioRegistry CR for your Apicurio Registry deployment, for example:

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
  security:
    https:
      secretName: https-cert-secret
```

4. Verify that the connection is working:

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

```
oc rsh example-apicurioregistry-deployment-6f788db977-2wzpw
```

   b. Find the cluster IP of the Apicurio 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://172.30.230.78:8443/health
```

**NOTE**

In the Kubernetes secret containing the HTTPS certificate and key, the names `tls.crt` and `tls.key` must be used for the provided values. This is currently not configurable.

**Disabling HTTP**

If you enabled HTTPS using the procedure in this section, you can also disable the default HTTP connection by setting the `spec.security.https.disableHttp` to true. This removes the HTTP port 8080 from the Apicurio Registry pod container, Service, and the NetworkPolicy (if present).

Importantly, Ingress is also removed because the Apicurio Registry Operator currently does not support configuring HTTPS in Ingress. Users must create an Ingress for HTTPS connections manually.

**Additional resources**

- How to enable HTTPS and SSL termination in a Quarkus app
5.5. CONFIGURING AN HTTPS CONNECTION TO APICURIO REGISTRY FROM OUTSIDE THE OPENSIGHT CLUSTER

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

Prerequisites

- You must have already installed the Apicurio 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 Apicurio Registry Operator. For example:

```yaml
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  [...]
labels:
  app: example-apicuriregistry
spec:
  host: example-apicuriregistry-default.apps.example.com
to:
  kind: Service
  name: example-apicuriregistry-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. Alternatively, you can generate a custom self-signed certificate using the following commands:

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

Then create a route using the OpenShift CLI:

```
oc create route edge \
  --service=example-apicurioregistry-service-9whd7 \
  --cert=tls.crt --key=tls.key \
```
--hostname=example-apicuriregistry-default.apps.example.com \
--insecure-policy=Redirect \
-n default
This chapter explains how to configure and manage optional settings for your Apicurio Registry deployment on OpenShift:

- Section 6.1, “Configuring Apicurio Registry health checks on OpenShift”
- Section 6.2, “Environment variables for Apicurio Registry health checks”
- Section 6.3, “Managing Apicurio Registry environment variables”
- Section 6.4, “Configuring Apicurio Registry deployment using PodTemplate”
- Section 6.5, “Configuring the Apicurio Registry web console”
- Section 6.6, “Configuring Apicurio Registry logging”
- Section 6.7, “Configuring Apicurio Registry event sourcing”

## 6.1. CONFIGURING APICURIO REGISTRY HEALTH CHECKS ON OPENSHIFT

You can configure optional environment variables for liveness and readiness probes to monitor the health of the Apicurio 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 Apicurio Registry on OpenShift.
- You must have already installed and configured your chosen Apicurio 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 Apicurio 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 Apicurio Registry health checks”
- OpenShift documentation on monitoring application health

**6.2. ENVIRONMENT VARIABLES FOR APICURIO REGISTRY HEALTH CHECKS**

This section describes the available environment variables for Apicurio Registry health checks on OpenShift. These include liveness and readiness probes to monitor the health of the Apicurio Registry server on OpenShift. For an example procedure, see Section 6.1, “Configuring Apicurio 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 Apicurio Registry liveness probes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIVENESS_ERROR_THRESHOLD</td>
<td>Number of liveness issues or errors that can occur before the liveness probe fails.</td>
<td>Integer</td>
<td>1</td>
</tr>
</tbody>
</table>
**LIVENESS_COUNTER_RESET**
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.

**LIVENESS_STATUS_RESET**
Number of seconds that must elapse without any more errors for the liveness probe to reset to OK status.

**LIVENESS_ERRORS_IGNORED**
Comma-separated list of ignored liveness exceptions.

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

### Readiness environment variables

**Table 6.2. Environment variables for Apicurio Registry readiness probes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>READINESS_ERROR_THRESHOLD</td>
<td>Number of readiness issues or errors that can occur before the readiness probe fails.</td>
<td>Integer</td>
<td>1</td>
</tr>
<tr>
<td>READINESS_COUNTER_RESET</td>
<td>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.</td>
<td>Seconds</td>
<td>60</td>
</tr>
<tr>
<td>READINESS_STATUS_RESET</td>
<td>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.</td>
<td>Seconds</td>
<td>300</td>
</tr>
</tbody>
</table>
**READINESS_TIMEOUT**

Readiness tracks the timeout of two operations:

- How long it takes for storage requests to complete
- How long it takes for HTTP REST API requests to return a response

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>READINESS_TIMEOUT</td>
<td>Readiness tracks the timeout of two operations:</td>
<td>Seconds</td>
<td>5</td>
</tr>
</tbody>
</table>

### Additional resources

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

### 6.3. MANAGING APICURIO REGISTRY ENVIRONMENT VARIABLES

Apicurio Registry Operator manages most common Apicurio Registry configuration, but there are some options that it does not support yet. If a high-level configuration option is not available in the ApicurioRegistry CR, you can use an environment variable to adjust it. You can update these by setting an environment variable directly in the ApicurioRegistry CR, in the `spec.configuration.env` field. These are then forwarded to the Deployment resource of Apicurio Registry.

**Procedure**

You can manage Apicurio Registry environment variables by using the OpenShift web console or CLI.

**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 Apicurio Registry deployment.

3. Click the **YAML** tab and then edit the `spec.configuration.env` section as needed. The following example shows how to set default global content rules:

```yaml
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
  env:
```
OpenShift CLI

1. Select the project where Apicurio Registry is installed.

2. Run `oc get apicurioregistry` to get the list of `ApicurioRegistry` CRs

3. Run `oc edit apicurioregistry` on the CR representing the Apicurio Registry instance that you want to configure.

4. Add or modify the environment variable in the `spec.configuration.env` section.

   The Apicurio Registry Operator might attempt to set an environment variable that is already explicitly specified in the `spec.configuration.env` field. If an environment variable configuration has a conflicting value, the value set by Apicurio Registry Operator takes precedence.

   You can avoid this conflict by either using the high-level configuration for the feature, or only using the explicitly specified environment variables. The following is an example of a conflicting configuration:

   ```yaml
   - name: REGISTRY_RULES_GLOBAL_VALIDITY
     value: FULL # One of: NONE, SYNTAX_ONLY, FULL
   - name: REGISTRY_RULES_GLOBAL_COMPATIBILITY
     value: FULL # One of: NONE, BACKWARD, BACKWARD_TRANSITIVE, 
                 FORWARD, FORWARD_TRANSITIVE, FULL, FULL_TRANSITIVE
   ```

   This configuration results in the Apicurio Registry web console being in read-only mode.

6.4. CONFIGURING APICURIO REGISTRY DEPLOYMENT USING PODTEMPLATE

**IMPORTANT**

This is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production.

These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offers/techpreview.
The **ApicurioRegistry** CRD contains the `spec.deployment.podTemplateSpecPreview` field, which has the same structure as the field `spec.template` in a Kubernetes *Deployment* resource (the `PodTemplateSpec` struct).

With some restrictions, the Apicurio Registry Operator forwards the data from this field to the corresponding field in the Apicurio Registry deployment. This provides greater configuration flexibility, without the need for the Apicurio Registry Operator to natively support each use case.

The following table contains a list of subfields that are not accepted by the Apicurio Registry Operator, and result in a configuration error:

**Table 6.3. Restrictions on the `podTemplateSpecPreview` subfields**

<table>
<thead>
<tr>
<th>podTemplateSpecPreview subfield</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>metadata.annotations</code></td>
<td>alternative exists</td>
<td><code>spec.deployment.metadata.annotations</code></td>
</tr>
<tr>
<td><code>metadata.labels</code></td>
<td>alternative exists</td>
<td><code>spec.deployment.metadata.labels</code></td>
</tr>
<tr>
<td><code>spec.affinity</code></td>
<td>alternative exists</td>
<td><code>spec.deployment.affinity</code></td>
</tr>
<tr>
<td><code>spec.containers[*]</code></td>
<td>warning</td>
<td>To configure the Apicurio Registry container, <strong>name: registry</strong> must be used</td>
</tr>
<tr>
<td><code>spec.containers[name = &quot;registry&quot;].env</code></td>
<td>alternative exists</td>
<td><code>spec.configuration.env</code></td>
</tr>
<tr>
<td><code>spec.containers[name = &quot;registry&quot;].image</code></td>
<td>reserved</td>
<td>-</td>
</tr>
<tr>
<td><code>spec.imagePullSecrets</code></td>
<td>alternative exists</td>
<td><code>spec.deployment.imagePullSecrets</code></td>
</tr>
<tr>
<td><code>spec.tolerations</code></td>
<td>alternative exists</td>
<td><code>spec.deployment.tolerations</code></td>
</tr>
</tbody>
</table>

**WARNING**

If you set a field in `podTemplateSpecPreview`, its value must be valid, as if you configured it in the Apicurio Registry *Deployment* directly. The Apicurio Registry Operator might still modify the values you provided, but it will not fix an invalid value or make sure a default value is present.

**Additional resources**
6.5. CONFIGURING THE APICURIO REGISTRY WEB CONSOLE

You can set optional environment variables to configure the Apicurio Registry web console specifically for your deployment environment or to customize its behavior.

Prerequisites

- You have already installed Apicurio Registry.

Configuring the web console deployment environment

When you access the Apicurio Registry web console in your browser, some initial configuration settings are loaded. The following configuration settings are important:

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

Typically, Apicurio 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:

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

Configuring the web console in read-only mode

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

- Creating an artifact
- Uploading a new artifact version
- Updating artifact metadata
- Deleting an artifact

Procedure

Configure the following environment variable:

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

6.6. CONFIGURING APICURIO REGISTRY LOGGING

You can set Apicurio Registry logging configuration at runtime. Apicurio Registry provides a REST API for configuring logging settings.
You can set Apicurio Registry logging configuration at runtime. Apicurio 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 Apicurio Registry log levels at runtime using the Apicurio Registry /admin REST API.

Prerequisites

- Get the URL to access your Apicurio Registry instance, or get your Apicurio Registry route if you have Apicurio 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`:

   ```bash
   $ 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"}
   
   $ 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"}
   
   $ 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"}
   
6.7. CONFIGURING APICURIO REGISTRY EVENT SOURCING

**IMPORTANT**

This is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production.

These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview.
You can configure Apicurio Registry to send events when changes are made to registry content. For example, Apicurio Registry can trigger events when schema or API artifacts, groups, or content rules are created, updated, deleted, and so on. You can configure Apicurio Registry to send events to your applications and to third-party integrations for these kind of changes.

There are different protocols available for transporting events. The currently implemented protocols are HTTP and Apache Kafka. However, regardless of the protocol, the events are sent by using the CNCF CloudEvents specification. You can configure Apicurio Registry event sourcing by using Java system properties or the equivalent environment variables.

Apicurio Registry event types
All of the event types are defined in `io.apicurio.registry.events.dto.RegistryEventType`. For example, these include the following event types:

- `io.apicurio.registry.artifact-created`
- `io.apicurio.registry.artifact-updated`
- `io.apicurio.registry.artifact-state-changed`
- `io.apicurio.registry.artifact-rule-created`
- `io.apicurio.registry.global-rule-created`
- `io.apicurio.registry.group-created`

**Prerequisites**

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

**Configuring Apicurio Registry event sourcing by using HTTP**

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

**Procedure**

1. When using the HTTP protocol, set your Apicurio 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 Apicurio Registry event sourcing by 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 by using the `KAFKA_BOOTSTRAP_SERVERS` environment variable:

   - `KAFKA_BOOTSTRAP_SERVERS=my-kafka-host:9092`
     Alternatively, you can set the properties for the kafka producer by 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. APICURIO REGISTRY OPERATOR CONFIGURATION REFERENCE

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

- Section 7.1, “Apicurio Registry Custom Resource”
- Section 7.2, “Apicurio Registry CR spec”
- Section 7.3, “Apicurio Registry CR status”
- Section 7.4, “Apicurio Registry managed resources”
- Section 7.5, “Apicurio Registry Operator labels”

7.1. APICURIO REGISTRY CUSTOM RESOURCE

The Apicurio Registry Operator defines an ApicurioRegistry custom resource (CR) that represents a single deployment of Apicurio Registry on OpenShift.

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

Example ApicurioRegistry CR

The following command displays the ApicurioRegistry resource:

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

```yaml
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:
```
By default, the Apicurio Registry Operator watches its own project namespace only. 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.

**7.2. APICURIO 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 might not be required or should not be defined at the same time.

```yaml
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>
  registryLogLevel: <string>
  security:
```

**Additional resources**

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

---

54
The following table describes each configuration option:

**Table 7.1. ApicurioRegistry CR spec configuration options**

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>-</td>
<td>-</td>
<td>Section for configuration of Apicurio Registry application</td>
</tr>
<tr>
<td>configuration/persistence</td>
<td>string</td>
<td>required</td>
<td>Storage backend. One of sql, kafkaSQL</td>
</tr>
<tr>
<td>configuration/sql</td>
<td>-</td>
<td>-</td>
<td>SQL storage backend configuration</td>
</tr>
<tr>
<td>configuration/sql/dataSource</td>
<td>-</td>
<td>-</td>
<td>Database connection configuration for SQL storage backend</td>
</tr>
<tr>
<td>configuration/sql/dataSource/uri</td>
<td>string</td>
<td>required</td>
<td>Database connection URL string</td>
</tr>
<tr>
<td>configuration/sql/dataSource/username</td>
<td>string</td>
<td>required</td>
<td>Database connection user</td>
</tr>
<tr>
<td>configuration/sql/dataSource/password</td>
<td>string</td>
<td>empty</td>
<td>Database connection password</td>
</tr>
<tr>
<td>Configuration option</td>
<td>type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>configuration/kafkasql</td>
<td>-</td>
<td>-</td>
<td>Kafka storage backend configuration</td>
</tr>
<tr>
<td>configuration/kafkasql/bootstrapServers</td>
<td>string</td>
<td>required</td>
<td>Kafka bootstrap server URL, for Streams storage backend</td>
</tr>
<tr>
<td>configuration/kafkasql/security/tls</td>
<td>-</td>
<td>-</td>
<td>Section to configure TLS authentication for Kafka storage backend</td>
</tr>
<tr>
<td>configuration/kafkasql/security/tls/truststoreSecretName</td>
<td>string</td>
<td>required</td>
<td>Name of a secret containing TLS truststore for Kafka</td>
</tr>
<tr>
<td>configuration/kafkasql/security/tls/keystoreSecretName</td>
<td>string</td>
<td>required</td>
<td>Name of a secret containing user TLS keystore</td>
</tr>
<tr>
<td>configuration/kafkasql/security/scram/truststoreSecretName</td>
<td>string</td>
<td>required</td>
<td>Name of a secret containing TLS truststore for Kafka</td>
</tr>
<tr>
<td>configuration/kafkasql/security/scram/user</td>
<td>string</td>
<td>required</td>
<td>SCRAM user name</td>
</tr>
<tr>
<td>configuration/kafkasql/security/scram/passwordSecretName</td>
<td>string</td>
<td>required</td>
<td>Name of a secret containing SCRAM user password</td>
</tr>
<tr>
<td>configuration/kafkasql/security/scram/mechanism</td>
<td>string</td>
<td>SCRAM-SHA-512</td>
<td>SASL mechanism</td>
</tr>
<tr>
<td>configuration/ui</td>
<td>-</td>
<td>-</td>
<td>Apicurio Registry web console settings</td>
</tr>
<tr>
<td>configuration/ui/readOnly</td>
<td>string</td>
<td>false</td>
<td>Set Apicurio Registry web console to read-only mode</td>
</tr>
<tr>
<td>configuration/logLevel</td>
<td>string</td>
<td>INFO</td>
<td>Apicurio Registry log level, for non-Apicurio components and libraries. One of INFO, DEBUG</td>
</tr>
<tr>
<td>Configuration option</td>
<td>type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>configuration/registryLogLevel</td>
<td>string</td>
<td>INFO</td>
<td>Apicurio Registry log level, for Apicurio application components (excludes non-Apicurio components and libraries). One of <em>INFO, DEBUG</em></td>
</tr>
<tr>
<td>configuration/security</td>
<td>-</td>
<td>-</td>
<td>Apicurio Registry web console and REST API security settings</td>
</tr>
<tr>
<td>configuration/security/keycloak</td>
<td>-</td>
<td>-</td>
<td>Web console and REST API security configuration using Red Hat Single Sign-On</td>
</tr>
<tr>
<td>configuration/security/keycloak/url</td>
<td>string</td>
<td>required</td>
<td>Red Hat Single Sign-On URL</td>
</tr>
<tr>
<td>configuration/security/keycloak/realm</td>
<td>string</td>
<td>required</td>
<td>Red Hat Single Sign-On realm</td>
</tr>
<tr>
<td>configuration/security/keycloak/apiClientId</td>
<td>string</td>
<td>registry-client-api</td>
<td>Red Hat Single Sign-On client for REST API</td>
</tr>
<tr>
<td>configuration/security/keycloak/uiClientId</td>
<td>string</td>
<td>registry-client-ui</td>
<td>Red Hat Single Sign-On client for web console</td>
</tr>
<tr>
<td>configuration/security/https</td>
<td>-</td>
<td>-</td>
<td>Configuration for HTTPS. For more details, see <em>Configuring an HTTPS connection to Apicurio Registry from inside the OpenShift cluster.</em></td>
</tr>
<tr>
<td>configuration/security/https/secretName</td>
<td>string</td>
<td>empty</td>
<td>Name of a Kubernetes Secret that contains the HTTPS certificate and key, which must be named <em>tls.crt and tls.key</em>, respectively. Setting this field enables HTTPS, and vice versa.</td>
</tr>
<tr>
<td>configuration/security/https/disableHttp</td>
<td>bool</td>
<td>false</td>
<td>Disable HTTP port and Ingress. HTTPS must be enabled as a prerequisite.</td>
</tr>
<tr>
<td>Configuration option</td>
<td>type</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>configuration/env</strong></td>
<td>k8s.io/api/core/v1 []EnvVar</td>
<td>empty</td>
<td>Configure a list of environment variables to be provided to the Apicurio Registry pod. For more details, see Managing Apicurio Registry environment variables.</td>
</tr>
<tr>
<td><strong>deployment</strong></td>
<td>-</td>
<td>-</td>
<td>Section for Apicurio Registry deployment settings</td>
</tr>
<tr>
<td><strong>deployment/replicas</strong></td>
<td>positive integer</td>
<td>1</td>
<td>Number of Apicurio Registry pods to deploy</td>
</tr>
<tr>
<td><strong>deployment/host</strong></td>
<td>string</td>
<td>auto-generated</td>
<td>Host/URL where the Apicurio Registry console and API are available. If possible, Apicurio 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.</td>
</tr>
<tr>
<td><strong>deployment/affinity</strong></td>
<td>k8s.io/api/core/v1 Affinity</td>
<td>empty</td>
<td>Apicurio Registry deployment affinity configuration</td>
</tr>
<tr>
<td><strong>deployment/tolerations</strong></td>
<td>k8s.io/api/core/v1 []Toleration</td>
<td>empty</td>
<td>Apicurio Registry deployment tolerations configuration</td>
</tr>
<tr>
<td><strong>deployment/imagePullSecrets</strong></td>
<td>k8s.io/api/core/v1 []LocalObjectReference</td>
<td>empty</td>
<td>Configure image pull secrets for Apicurio Registry deployment</td>
</tr>
<tr>
<td><strong>deployment/metadata</strong></td>
<td>-</td>
<td>-</td>
<td>Configure a set of labels or annotations for the Apicurio Registry pod.</td>
</tr>
<tr>
<td><strong>deployment/metadata/labels</strong></td>
<td>map[string]string</td>
<td>empty</td>
<td>Configure a set of labels for Apicurio Registry pod.</td>
</tr>
</tbody>
</table>
### Configuration option

<table>
<thead>
<tr>
<th>Configuration option</th>
<th>type</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment/metadata/annotations</td>
<td>map[string]string</td>
<td>empty</td>
<td>Configure a set of annotations for Apicurio Registry pod</td>
</tr>
<tr>
<td>deployment/managedResources</td>
<td>-</td>
<td>-</td>
<td>Section to configure how the Apicurio Registry Operator manages Kubernetes resources. For more details, see Apicurio Registry managed resources.</td>
</tr>
<tr>
<td>deployment/managedResources/disableIngress</td>
<td>bool</td>
<td>false</td>
<td>If set, the operator will not create and manage an Ingress resource for Apicurio Registry deployment.</td>
</tr>
<tr>
<td>deployment/managedResources/disableNetworkPolicy</td>
<td>bool</td>
<td>false</td>
<td>If set, the operator will not create and manage a NetworkPolicy resource for Apicurio Registry deployment.</td>
</tr>
<tr>
<td>deployment/managedResources/disablePodDisruptionBudget</td>
<td>bool</td>
<td>false</td>
<td>If set, the operator will not create and manage a PodDisruptionBudget resource for Apicurio Registry deployment.</td>
</tr>
<tr>
<td>deployment/podTemplateSpecReview</td>
<td>k8s.io/api/core/v1 PodTemplateSpec</td>
<td>empty</td>
<td>Configure parts of the Apicurio Registry deployment resource. For more details, see Configuring Apicurio Registry deployment using PodTemplate.</td>
</tr>
</tbody>
</table>

**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. APICURIO REGISTRY CR STATUS

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

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

```yaml
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>
<thead>
<tr>
<th>Status field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>-</td>
<td>Section with information about the deployed Apicurio Registry.</td>
</tr>
<tr>
<td>info/host</td>
<td>string</td>
<td>URL where the Apicurio Registry UI and REST API are accessible.</td>
</tr>
<tr>
<td>conditions</td>
<td>-</td>
<td>List of conditions that report the status of the Apicurio Registry, or the Operator with respect to that deployment.</td>
</tr>
<tr>
<td>conditions/type</td>
<td>string</td>
<td>Type of the condition.</td>
</tr>
<tr>
<td>conditions/status</td>
<td>string</td>
<td>Status of the condition, one of True, False, Unknown.</td>
</tr>
<tr>
<td>conditions/reason</td>
<td>string</td>
<td>A programmatic identifier indicating the reason for the condition’s last transition.</td>
</tr>
<tr>
<td>conditions/message</td>
<td>string</td>
<td>A human-readable message indicating details about the transition.</td>
</tr>
<tr>
<td>conditions/lastTransitionTime</td>
<td>string, RFC-3339 timestamp</td>
<td>The last time the condition transitioned from one status to another.</td>
</tr>
<tr>
<td>managedResources</td>
<td>-</td>
<td>List of OpenShift resources managed by Apicurio Registry Operator</td>
</tr>
<tr>
<td>managedResources/kind</td>
<td>string</td>
<td>Resource kind.</td>
</tr>
</tbody>
</table>
The resources managed by the Apicurio Registry Operator when deploying Apicurio Registry are as follows:

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

You can disable the Apicurio Registry Operator from creating and managing some resources, so they can be configured manually. This provides greater flexibility when using features that the Apicurio Registry Operator does not currently support.

If you disable a resource type, its existing instance is deleted. If you enable a resource, the Apicurio Registry Operator attempts to find a resource using the `app` label, for example, `app=example-apicurioregistry`, and starts managing it. Otherwise, the Operator creates a new instance.

You can disable the following resource types in this way:

- **Ingress** (and **Route**)
- **NetworkPolicy**
- **PodDisruptionBudget**

For example:

```yaml
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  deployment:
    managedResources:
      disableIngress: true
      disableNetworkPolicy: true
      disablePodDisruptionBudget: false # Can be omitted
```
7.5. APICURIO REGISTRY OPERATOR LABELS

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

### Table 7.3. Apicurio Registry Operator labels for managed resources

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

### Custom labels and annotations

You can provide custom labels and annotation for the Apicurio Registry pod, using the `spec.deployment.metadata.labels` and `spec.deployment.metadata.annotations` fields, for example:

```yaml
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
  deployment:
    metadata:
      labels:
        example.com/environment: staging
      annotations:
        example.com/owner: my-team
```

### Additional resources

- [Recommended Kubernetes labels for application deployments](#)
This chapter provides reference information on the configuration options that are available for Apicurio Registry.

- Section 8.1, “Apicurio Registry configuration options”

Additional resources

- For details on setting configuration options by using the Core Registry API, see the `/admin/config/properties` endpoint in the Apicurio Registry REST API documentation.
- For details on client configuration options for Kafka serializers and deserializers, see the Red Hat build of Apicurio Registry User Guide.

8.1. APICURIO REGISTRY CONFIGURATION OPTIONS

The following Apicurio Registry configuration options are available for each component category:

8.1.1. api

Table 8.1. api configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.api.errors.include-stack-in-response</td>
<td>boolean</td>
<td>false</td>
<td>2.1.4.Final</td>
<td>Include stack trace in errors responses</td>
</tr>
<tr>
<td>registry.disable.apis</td>
<td>optional&lt;list&lt;string&gt;&gt;</td>
<td></td>
<td>2.0.0.Final</td>
<td>Disable APIs</td>
</tr>
</tbody>
</table>

8.1.2. auth

Table 8.2. auth configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.auth.admin-override.claim</td>
<td>string</td>
<td>org-admin</td>
<td>2.1.0.Final</td>
<td>Auth admin override claim</td>
</tr>
<tr>
<td>registry.auth.admin-override.claim-value</td>
<td>string</td>
<td>true</td>
<td>2.1.0.Final</td>
<td>Auth admin override claim value</td>
</tr>
<tr>
<td>registry.auth.admin-override.enabled</td>
<td>boolean</td>
<td>false</td>
<td>2.1.0.Final</td>
<td>Auth admin override enabled</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Default</td>
<td>Available from</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------</td>
<td>---------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>registry.auth.admin-override.from</td>
<td>string</td>
<td>token</td>
<td>2.1.0.Final</td>
<td>Auth admin override from</td>
</tr>
<tr>
<td>registry.auth.admin-override.role</td>
<td>string</td>
<td>sr-admin</td>
<td>2.1.0.Final</td>
<td>Auth admin override role</td>
</tr>
<tr>
<td>registry.auth.admin-override.type</td>
<td>string</td>
<td>role</td>
<td>2.1.0.Final</td>
<td>Auth admin override type</td>
</tr>
<tr>
<td>registry.auth.anonymous-read-access.enabled</td>
<td>boolean</td>
<td>false</td>
<td>2.1.0.Final</td>
<td>Anonymous read access</td>
</tr>
<tr>
<td>registry.auth.audit.log.prefix</td>
<td>string</td>
<td>audit</td>
<td>2.2.6</td>
<td>Prefix used for application audit logging.</td>
</tr>
<tr>
<td>registry.auth.authenticated-read-access.enabled</td>
<td>boolean</td>
<td>false</td>
<td>2.1.4.Final</td>
<td>Authenticated read access</td>
</tr>
<tr>
<td>registry.auth.basic-auth-client-credentials.cache-expiration</td>
<td>integer</td>
<td>10</td>
<td>2.2.6.Final</td>
<td>Client credentials token expiration time.</td>
</tr>
<tr>
<td>registry.auth.basic-auth-client-credentials.enabled</td>
<td>boolean</td>
<td>false</td>
<td>2.1.0.Final</td>
<td>Enable basic auth client credentials</td>
</tr>
<tr>
<td>registry.auth.client-id</td>
<td>string</td>
<td></td>
<td>2.0.0.Final</td>
<td>Client identifier used by the server for authentication.</td>
</tr>
<tr>
<td>registry.auth.client-secret</td>
<td>optional&lt;string&gt;</td>
<td></td>
<td>2.1.0.Final</td>
<td>Client secret used by the server for authentication.</td>
</tr>
<tr>
<td>registry.auth.enabled</td>
<td>boolean</td>
<td>false</td>
<td>2.0.0.Final</td>
<td>Enable auth</td>
</tr>
<tr>
<td>registry.auth.owner-only-authorization</td>
<td>boolean</td>
<td>false</td>
<td>2.0.0.Final</td>
<td>Artifact owner-only authorization</td>
</tr>
<tr>
<td>registry.auth.owner-only-authorization.limit-group-access</td>
<td>boolean</td>
<td>false</td>
<td>2.1.0.Final</td>
<td>Artifact group owner-only authorization</td>
</tr>
<tr>
<td>registry.auth.role-based-authorization</td>
<td>boolean</td>
<td>false</td>
<td>2.1.0.Final</td>
<td>Enable role based authorization</td>
</tr>
<tr>
<td>registry.auth.role-source</td>
<td>string</td>
<td>token</td>
<td>2.1.0.Final</td>
<td>Auth roles source</td>
</tr>
</tbody>
</table>
## 8.1.3. cache

Table 8.3. cache configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.config.cache.enabled</td>
<td>boolean</td>
<td>true</td>
<td>2.2.2.Final</td>
<td>Registry cache enabled</td>
</tr>
</tbody>
</table>

## 8.1.4. ccompat

Table 8.4. ccompat configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.ccompat.legacy-id-mode.enabled</td>
<td>boolean</td>
<td>false</td>
<td>2.0.2.Final</td>
<td>Legacy ID mode (compatibility API)</td>
</tr>
<tr>
<td>registry.ccompat.max-subjects</td>
<td>integer</td>
<td>1000</td>
<td>2.4.2.Final</td>
<td>Maximum number of Subjects returned (compatibility API)</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Default</td>
<td>Available from</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------</td>
<td>---------</td>
<td>----------------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>registry.ccompat.use-canonical-hash</td>
<td>boolean</td>
<td>false</td>
<td>2.3.0.Final</td>
<td>Canonical hash mode (compatibility API)</td>
</tr>
</tbody>
</table>

### 8.1.5. download

Table 8.5. download configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.download.href.ttl</td>
<td>long</td>
<td>30</td>
<td>2.1.2.Final</td>
<td>Download link expiry</td>
</tr>
</tbody>
</table>

### 8.1.6. events

Table 8.6. events configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.events.ksink</td>
<td>optional&lt;string&gt;</td>
<td></td>
<td>2.0.0.Final</td>
<td>Events Kafka sink enabled</td>
</tr>
</tbody>
</table>

### 8.1.7. health

Table 8.7. health configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.liveness.errors.ignored</td>
<td>optional&lt;list&lt;string&gt;&gt;</td>
<td></td>
<td>1.2.3.Final</td>
<td>Ignored liveness errors</td>
</tr>
<tr>
<td>registry.metrics.PersistenceExceptionLivenessCheck.counterResetWindowDurationSec</td>
<td>integer</td>
<td>60</td>
<td>1.0.2.Final</td>
<td>Counter reset window duration of persistence liveness check</td>
</tr>
<tr>
<td>registry.metrics.PersistenceExceptionLivenessCheck.disableLogging</td>
<td>boolean</td>
<td>false</td>
<td>2.0.0.Final</td>
<td>Disable logging of persistence liveness check</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Default</td>
<td>Available from</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>---------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>registry.metrics.PersistenceExceptionLivenessCheck.errorThreshold</td>
<td>integer</td>
<td>1</td>
<td>1.0.2.Final</td>
<td>Error threshold of persistence liveness check</td>
</tr>
<tr>
<td>registry.metrics.PersistenceExceptionLivenessCheck.statusResetWindowDurationSec</td>
<td>integer</td>
<td>300</td>
<td>1.0.2.Final</td>
<td>Status reset window duration of persistence liveness check</td>
</tr>
<tr>
<td>registry.metrics.PersistenceTimeoutReadinessCheck.counterResetWindowDurationSec</td>
<td>integer</td>
<td>60</td>
<td>1.0.2.Final</td>
<td>Counter reset window duration of persistence readiness check</td>
</tr>
<tr>
<td>registry.metrics.PersistenceTimeoutReadinessCheck.errorThreshold</td>
<td>integer</td>
<td>5</td>
<td>1.0.2.Final</td>
<td>Error threshold of persistence readiness check</td>
</tr>
<tr>
<td>registry.metrics.PersistenceTimeoutReadinessCheck.statusResetWindowDurationSec</td>
<td>integer</td>
<td>300</td>
<td>1.0.2.Final</td>
<td>Status reset window duration of persistence readiness check</td>
</tr>
<tr>
<td>registry.metrics.PersistenceTimeoutReadinessCheck.timeoutSec</td>
<td>integer</td>
<td>15</td>
<td>1.0.2.Final</td>
<td>Timeout of persistence readiness check</td>
</tr>
<tr>
<td>registry.metrics.ResponseErrorLivenessCheck.counterResetWindowDurationSec</td>
<td>integer</td>
<td>60</td>
<td>1.0.2.Final</td>
<td>Counter reset window duration of response liveness check</td>
</tr>
<tr>
<td>registry.metrics.ResponseErrorLivenessCheck.disableLogging</td>
<td>boolean</td>
<td>false</td>
<td>2.0.0.Final</td>
<td>Disable logging of response liveness check</td>
</tr>
<tr>
<td>registry.metrics.ResponseErrorLivenessCheck.errorThreshold</td>
<td>integer</td>
<td>1</td>
<td>1.0.2.Final</td>
<td>Error threshold of response liveness check</td>
</tr>
<tr>
<td>registry.metrics.ResponseErrorLivenessCheck.statusResetWindowDurationSec</td>
<td>integer</td>
<td>300</td>
<td>1.0.2.Final</td>
<td>Status reset window duration of response liveness check</td>
</tr>
<tr>
<td>registry.metrics.ResponseTimeoutReadinessCheck.counterResetWindowDurationSec</td>
<td>integer</td>
<td>60</td>
<td>1.0.2.Final</td>
<td>Counter reset window duration of response readiness check</td>
</tr>
</tbody>
</table>
### 8.1.8. import

Table 8.8. import configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.import.url</td>
<td>optional&lt;url&gt;</td>
<td></td>
<td>2.1.0.Final</td>
<td>The import URL</td>
</tr>
</tbody>
</table>

### 8.1.9. kafka

Table 8.9. kafka configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.events.kafka.topic</td>
<td>optional&lt;string&gt;</td>
<td></td>
<td>2.0.0.Final</td>
<td>Events Kafka topic</td>
</tr>
<tr>
<td>registry.events.kafka.topic.partition</td>
<td>optional&lt;integer&gt;</td>
<td></td>
<td>2.0.0.Final</td>
<td>Events Kafka topic partition</td>
</tr>
</tbody>
</table>

### 8.1.10. limits

Table 8.10. limits configuration options
### Table 8.11. log configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.limits.config.max-artifact-labels</td>
<td>long</td>
<td>-1</td>
<td>2.2.3.Final</td>
<td>Max artifact labels</td>
</tr>
<tr>
<td>registry.limits.config.max-artifact-properties</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max artifact properties</td>
</tr>
<tr>
<td>registry.limits.config.max-artifacts</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max artifacts</td>
</tr>
<tr>
<td>registry.limits.config.max-description-length</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max artifact description length</td>
</tr>
<tr>
<td>registry.limits.config.max-label-size</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max artifact label size</td>
</tr>
<tr>
<td>registry.limits.config.max-name-length</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max artifact name length</td>
</tr>
<tr>
<td>registry.limits.config.max-property-key-size</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max artifact property key size</td>
</tr>
<tr>
<td>registry.limits.config.max-property-value-size</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max artifact property value size</td>
</tr>
<tr>
<td>registry.limits.config.max-requests-per-second</td>
<td>long</td>
<td>-1</td>
<td>2.2.3.Final</td>
<td>Max artifact requests per second</td>
</tr>
<tr>
<td>registry.limits.config.max-schema-size-bytes</td>
<td>long</td>
<td>-1</td>
<td>2.2.3.Final</td>
<td>Max schema size (bytes)</td>
</tr>
<tr>
<td>registry.limits.config.max-total-schemas</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max total schemas</td>
</tr>
<tr>
<td>registry.limits.config.max-versions-per-artifact</td>
<td>long</td>
<td>-1</td>
<td>2.1.0.Final</td>
<td>Max versions per artifacts</td>
</tr>
<tr>
<td>registry.storage.metrics.cache.max-size</td>
<td>long</td>
<td>1000</td>
<td>2.4.1.Final</td>
<td>Storage metrics cache max size.</td>
</tr>
</tbody>
</table>
### Table 8.12. redirects configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.enable-redirects</td>
<td>boolean</td>
<td></td>
<td>2.1.2.Final</td>
<td>Enable redirects</td>
</tr>
<tr>
<td>registry.redirects</td>
<td>map&lt;string, string&gt;</td>
<td></td>
<td>2.1.2.Final</td>
<td>Registry redirects</td>
</tr>
</tbody>
</table>

### Table 8.13. rest configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry.rest.artifact.deletion.enabled</td>
<td>boolean [dynamic]</td>
<td>false</td>
<td>2.4.2-SNAPSHOT</td>
<td>Enables artifact version deletion</td>
</tr>
<tr>
<td>registry.rest.artifact.download.maxSize</td>
<td>int</td>
<td>100000</td>
<td>2.2.6-SNAPSHOT</td>
<td>Max size of the artifact allowed to be downloaded from URL</td>
</tr>
<tr>
<td>registry.rest.artifact.download.skipSSLValidation</td>
<td>boolean</td>
<td>false</td>
<td>2.2.6-SNAPSHOT</td>
<td>Skip SSL validation when downloading artifacts from URL</td>
</tr>
</tbody>
</table>

### Table 8.14. store configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifacts.skip.disabled.latest</td>
<td>boolean</td>
<td>true</td>
<td>2.4.2-SNAPSHOT</td>
<td>Skip artifact versions with DISABLED state when retrieving latest artifact version</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Default</td>
<td>Available from</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
<td>---------</td>
<td>----------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>quarkus.datasource.db-kind</td>
<td>string</td>
<td>postgresql</td>
<td>2.0.0.Final</td>
<td>Datasource Db kind</td>
</tr>
<tr>
<td>quarkus.datasource.jdbc.url</td>
<td>string</td>
<td></td>
<td>2.1.0.Final</td>
<td>Datasource jdbc URL</td>
</tr>
<tr>
<td>registry.sql.init</td>
<td>boolean</td>
<td>true</td>
<td>2.0.0.Final</td>
<td>SQL init</td>
</tr>
</tbody>
</table>

### 8.1.15. ui

Table 8.15. ui configuration options

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Available from</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quarkus.oidc.tenant-enabled</td>
<td>boolean</td>
<td>false</td>
<td>2.0.0.Final</td>
<td>UI OIDC tenant enabled</td>
</tr>
<tr>
<td>registry.ui.config.apiUrl</td>
<td>string</td>
<td></td>
<td>1.3.0.Final</td>
<td>UI APIs URL</td>
</tr>
<tr>
<td>registry.ui.config.auth.oicd.client-id</td>
<td>string</td>
<td>none</td>
<td>2.2.6.Final</td>
<td>UI auth OIDC client ID</td>
</tr>
<tr>
<td>registry.ui.config.auth.oicd.redirect-url</td>
<td>string</td>
<td>none</td>
<td>2.2.6.Final</td>
<td>UI auth OIDC redirect URL</td>
</tr>
<tr>
<td>registry.ui.config.auth.oicd.url</td>
<td>string</td>
<td>none</td>
<td>2.2.6.Final</td>
<td>UI auth OIDC URL</td>
</tr>
<tr>
<td>registry.ui.config.auth.type</td>
<td>string</td>
<td>none</td>
<td>2.2.6.Final</td>
<td>UI auth type</td>
</tr>
<tr>
<td>registry.ui.config.uiCodeGenEnabled</td>
<td>boolean</td>
<td>true</td>
<td>2.4.2.Final</td>
<td>UI codegen enabled</td>
</tr>
<tr>
<td>registry.ui.config.uiContextPath</td>
<td>string</td>
<td>/ui/</td>
<td>2.1.0.Final</td>
<td>UI context path</td>
</tr>
<tr>
<td>registry.ui.features.readOnly</td>
<td>boolean</td>
<td>false</td>
<td>1.2.0.Final</td>
<td>UI read-only mode</td>
</tr>
<tr>
<td>registry.ui.features.settings</td>
<td>boolean</td>
<td>false</td>
<td>2.2.2.Final</td>
<td>UI features settings</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Default</td>
<td>Available from</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>---------</td>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>registry.ui.root</td>
<td>string</td>
<td>2.3.0.Final</td>
<td></td>
<td>Overrides the UI root context (useful when relocating the UI context using an inbound proxy)</td>
</tr>
</tbody>
</table>
APPENDIX A. USING YOUR SUBSCRIPTION

Apicurio 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.
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.
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.
2. Locate the Red Hat Integration entries in the Integration and Automation category.
3. Select the desired Apicurio Registry product. The Software Downloads page opens.
4. Click the Download link for your component.

Revised on 2023-11-07 12:56:02 UTC