Red Hat build of Apicurio Registry 2.5

Installing and deploying Apicurio Registry on OpenShift

Install, deploy, and configure Apicurio Registry 2.5

Last Updated: 2024-02-22

Red Hat build of Apicurio Registry 2.5 Installing and deploying Apicurio Registry on OpenShift

Install, deploy, and configure Apicurio Registry 2.5

Legal Notice

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

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 propose improvements, open a Jira issue and describe your suggested changes. Provide as much detail as possible to enable us to address your request quickly.

Prerequisite

- You have a Red Hat Customer Portal account. This account enables you to log in to the Red Hat Jira Software instance.
  If you do not have an account, you will be prompted to create one.

Procedure

1. Click the following link: Create issue.
2. In the **Summary** text box, enter a brief description of the issue.
3. In the **Description** text box, provide the following information:
   - The URL of the page where you found the issue.
   - A detailed description of the issue.
     You can leave the information in any other fields at their default values.
4. Click **Create** to submit the Jira issue to the documentation team.

Thank you for taking the time to provide feedback.

CHAPTER 1. APICURIO REGISTRY OPERATOR QUICKSTART

You can quickly install the Apicurio Registry Operator on the command line by using Custom Resource Definitions (CRDs).

The quickstart example deploys your Apicurio Registry instance with storage in an SQL database:

- Section 1.1, “Quickstart Apicurio Registry Operator installation”
- Section 1.2, “Quickstart Apicurio Registry instance deployment”

NOTE

The recommended installation option for production environments is the OpenShift OperatorHub. The recommended storage option is an SQL database for performance, stability, and data management.

1.1. QUICKSTART APICURIO REGISTRY OPERATOR INSTALLATION

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:

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

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

```
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:

```
NAME                        READY   UP-TO-DATE  AVAILABLE   AGE
apicurio-registry-operator  1/1  1        1       XmYs
```

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

   ```
   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:

   ```
   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:

   ```
   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:

   ```
   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 OPENSHIFT

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-2.6.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**.

WARNING

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

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

9. 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:

```
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

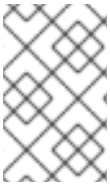
- For more details on creating Kafka clusters and topics using AMQ Streams, see Deploying and Managing AMQ Streams on OpenShift.

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:

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

The default Kafka topic name 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:

```
apiVersion: kafka.strimzi.io/v1beta1
```

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

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

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

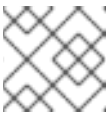
NOTE

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

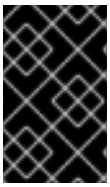
   NOTE

   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"
```

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



IMPORTANT

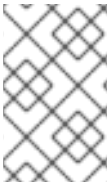
You must use a different **bootstrapServers** address than in the plain insecure use case. The address must support TLS connections and is found in the specified **Kafka** resource under the **type: tls** field.

3.4. CONFIGURING KAFKA STORAGE WITH SCRAM SECURITY

You can configure the AMQ Streams Operator and 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:

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: registry-example-kafkasql-scram
  # Change or remove the explicit namespace
spec:
  kafka:
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      log.message.format.version: '2.7'
```

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

The default Kafka topic name 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:

```
apiVersion: kafka.strimzi.io/v1beta1
kind: 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**.

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

```
    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
```

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** - 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:

```
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 to 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:

Environment variable	Description	Default value
ENABLE_KAFKA_SASL	Enables SASL OAuth authentication for Apicurio Registry storage in Kafka. You must set this variable to **true** for the other variables to have effect.	**false**
CLIENT_ID	The client ID used to authenticate to Kafka.	**-**
CLIENT_SECRET	The client secret used to authenticate to Kafka.	**-**
OAUTH_TOKEN_ENDPOINT_URI	The URL of the OAuth identity server.	**http://localhost:8090**

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

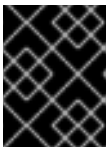
If you do not already have a PostgreSQL database Operator installed, you can install a PostgreSQL Operator on your OpenShift cluster from the OperatorHub. The OperatorHub is available from the OpenShift Container Platform web console and provides an interface for cluster administrators to discover and 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
```

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

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

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

9. Click **Create** and wait for the 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 datbase using **pg_dump**. See Section 4.3, “Backing 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

Apicurio Registry provides authentication and authorization by using Red Hat Single Sign-On based on OpenID Connect (OIDC) and 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 also provides authentcation and authorization by using Microsoft Azure Active Directory based on OpenID Connect (OIDC) and the OAuth Authorization Code Flow. You can configure the required settings manually in Azure AD and Apicurio Registry.

In addition to role-based authorization options with Red Hat Single Sign-On or Azure AD, 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.

This chapter explains how to configure the following security options 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, “Configuring Apicurio Registry authentication and authorization with Microsoft Azure Active Directory”
- Section 5.4, “Apicurio Registry authentication and authorization configuration options”
- Section 5.5, “Configuring an HTTPS connection to Apicurio Registry from inside the OpenShift cluster”
- Section 5.6, “Configuring an HTTPS connection to Apicurio Registry from outside the 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

Name	Capabilities
sr-admin	Full access, no restrictions.
sr-developer	Create artifacts and configure artifact rules. Cannot modify global rules, perform import/export, or use **/admin** REST API endpoint.
sr-readonly	View and search only. Cannot modify artifacts or rules, perform import/export, or use **/admin** REST API endpoint.

NOTE

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

Prerequisites

- You must have already installed the 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:

```
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:

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

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

IMPORTANT

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

6. If your cluster does not have a valid HTTPS certificate configured, you can create the following HTTP **Service** and **Ingress** resources as a temporary workaround:

   a. Click **Networking** and then **Services**, and click **Create Service** using the following example:

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

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

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: keycloak-http
```

```
    labels:
      app: keycloak
  spec:
    rules:
      - host: KEYCLOAK_HTTP_HOST
        http:
          paths:
            - path: /
              pathType: ImplementationSpecific
              backend:
                service:
                  name: keycloak-http
                  port:
                    number: 8080
```

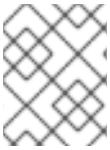
Modify the **host** value to create a route accessible for the Apicurio Registry user, and use it instead of the HTTPS route created by Red Hat Single Sign-On Operator.

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 and 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” .

The Apicurio Registry web console and core REST API support authentication in Red Hat Single Sign-On based on OAuth and 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 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 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 with Red Hat Single Sign-On

Environment variable	Description	Type	Default
AUTH_ENABLED	Enables authentication for Apicurio Registry. When set to **true**, the environment variables that follow are required for authentication using Red Hat Single Sign-On.	String	**false**
KEYCLOAK_URL	The URL of the Red Hat Single Sign-On authentication server. For example, **http://localhost:8080**.	String	-
KEYCLOAK_REALM	The Red Hat Single Sign-On realm for authentication. For example, **registry.**	String	-
KEYCLOAK_API_CLIENT_ID	The client ID for the Apicurio Registry REST API.	String	**registry-api**
KEYCLOAK_UI_CLIENT_ID	The client ID for the Apicurio Registry web console.	String	**apicurio-registry**

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

Environment variable	Java system property	Type	Default value
ROLE_BASED_AUTHZ_ENABLED	**registry.auth.role-based-authorization**	Boolean	**false**

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

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

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

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

Environment variable	Java system property	Type	Default value
REGISTRY_AUTH_OBAC_ENABLED	**registry.auth.owner-only-authorization**	Boolean	**false**

Additional resources

- For details on configuring non-default user role names, see Section 5.4, "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. CONFIGURING APICURIO REGISTRY AUTHENTICATION AND AUTHORIZATION WITH MICROSOFT AZURE ACTIVE DIRECTORY

This section explains how to manually configure authentication and authorization options for Apicurio Registry and Microsoft Azure Active Directory (Azure AD).

The Apicurio Registry web console and core REST API support authentication in Azure AD based on OpenID Connect (OIDC) and the OAuth Authorization Code Flow. Apicurio Registry provides role-based authorization for default admin, write, and read-only user roles. Apicurio Registry authentication and authorization settings are disabled by default.

To secure Apicurio Registry with Azure AD, you require a valid directory in Azure AD with specific configuration. This involves registering the Apicurio Registry application in the Azure AD portal with recommended settings and configuring environment variables in Apicurio Registry.

Prerequisites

- Azure AD is installed and running. For more details, see the Microsoft Azure AD user documentation.
- Apicurio Registry is installed and running.

Procedure

1. Log in to the Azure AD portal using your email address or GitHub account.
2. In the navigation menu, select **Manage > App registrations > New registration**, and complete the following settings:
   - **Name**: Enter your application name. For example: **apicurio-registry-example**
   - **Supported account types**: Click **Accounts in any organizational directory**
   - **Redirect URI**: Select **Single-page application** from the list, and enter your Apicurio Registry web console application host. For example: **https://test-registry.com/ui/**

   IMPORTANT

   You must register your Apicurio Registry application host as a **Redirect URI**. When logging in, users are redirected from Apicurio Registry to Azure AD for authentication, and you want to send them back to your application afterwards. Azure AD does not allow any redirect URLs that are not registered.
3. Click **Register**. You can view your app registration details by selecting **Manage > App registrations** > **apicurio-registry-example**.
4. Select **Manage > Authentication** and ensure that the application is configured with your redirect URLs and tokens as follows:
   - **Redirect URIs**: For example: **https://test-registry.com/ui/**
   - **Implicit grant and hybrid flows**: Click **ID tokens (used for implicit and hybrid flows)**
5. Select **Azure AD > Admin > App registrations > Your app > Application (client) ID**. For example: **123456a7-b8c9-012d-e3f4-5fg67h8i901**
6. Select **Azure AD > Admin > App registrations > Your app > Directory (tenant) ID**. For example: **https://login.microsoftonline.com/1a2bc34d-567e-89f1-g0hi-1j2kl3m4no56/v2.0**
7. In Apicurio Registry, configure the following environment variables with your Azure AD settings:

Table 5.6. Configuration for Azure AD settings in Apicurio Registry

Environment variable	Description	Setting
KEYCLOAK_API_CLIENT_ID	The client application ID for the Apicurio Registry REST API	Your Azure AD Application (client) ID obtained in step 5. For example: **123456a7-b8c9-012d-e3f4-5fg67h8i901**

Environment variable	Description	Setting
REGISTRY_OIDC_UI_CLIENT_ID	The client application ID for the Apicurio Registry web console.	Your Azure AD Application (client) ID obtained in step 5. For example: **123456a7-b8c9-012d-e3f4-5fg67h8i901**
REGISTRY_AUTH_URL_CONFIGURED	The URL for authentication in Azure AD.	Your Azure AD Application (tenant) ID obtained in step 6. For example: **https://login.microsoftonline.com/1a2bc34d-567e-89f1-g0hi-1j2kl3m4no56/v2.0**.

8. In Apicurio Registry, configure the following environment variables for Apicurio Registry-specific settings:

Table 5.7. Configuration for Apicurio Registry-specific settings

Environment variable	Description	Setting
REGISTRY_AUTH_ENABLED	Enables authentication for Apicurio Registry.	**true**
REGISTRY_UI_AUTH_TYPE	The Apicurio Registry authentication type.	**oidc**
CORS_ALLOWED_ORIGINS	The host for your Apicurio Registry deployment for cross-origin resource sharing (CORS).	For example: **https://test-registry.com**
REGISTRY_OIDC_UI_REDIRECT_URL	The host for your Apicurio Registry web console.	For example: **https://test-registry.com/ui**
ROLE_BASED_AUTHZ_ENABLED	Enables role-based authorization in Apicurio Registry.	**true**
QUARKUS_OIDC_ROLES_ROLE_CLAIM_PATH	The name of the claim in which Azure AD stores roles.	**roles**

NOTE

When you enable roles in Apicurio Registry, you must also create the same roles in Azure AD as application roles. The default roles expected by Apicurio Registry are **sr-admin**, **sr-developer**, and **sr-readonly**.

Additional resources

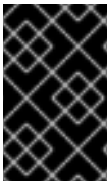
- For details on configuring non-default user role names, see Section 5.4, "Apicurio Registry authentication and authorization configuration options".
- For more details on using Azure AD, see the Microsoft Azure AD user documentation.

5.4. APICURIO REGISTRY AUTHENTICATION AND AUTHORIZATION CONFIGURATION OPTIONS

Apicurio Registry provides authentication options for OpenID Connect with Red Hat Single Sign-On and 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
- Apicurio Registry authentication by using HTTP basic
- Apicurio Registry role-based authorization
- Apicurio Registry owner-only authorization
- Apicurio Registry authenticated read access
- Apicurio Registry anonymous read-only access

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

Environment variable	Description	Type	Default
AUTH_ENABLED	Enables authentication for Apicurio Registry. When set to **true**, the environment variables that follow are required for authentication using Red Hat Single Sign-On.	String	**false**

Environment variable	Description	Type	Default
KEYCLOAK_URL	The URL of the Red Hat Single Sign-On authentication server. For example, **http://localhost:8080**.	String	-
KEYCLOAK_REALM	The Red Hat Single Sign-On realm for authentication. For example, **registry.**	String	-
KEYCLOAK_API_CLIENT_ID	The client ID for the Apicurio Registry REST API.	String	**registry-api**
KEYCLOAK_UI_CLIENT_ID	The client ID for the Apicurio Registry web console.	String	**apicurio-registry**

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 5.9. Configuration for Apicurio Registry HTTP basic authentication

Environment variable	Java system property	Type	Default value
AUTH_ENABLED	**registry.auth.enabled**	Boolean	**false**
CLIENT_CREDENTIALS_BASIC_AUTH_ENABLED	**registry.auth.basic-auth-client-credentials.enabled**	Boolean	**false**

Apicurio Registry HTTP basic client credentials cache expiry

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:

Table 5.10. Configuration for HTTP basic client credentials cache expiry

Environment variable	Java system property	Type	Default value
CLIENT_CREDENTIALS_BASIC_CACHE_EXPIRATION	**registry.auth.basic-auth-client-credentials.cache-expiration**	Integer	**10**

Apicurio Registry role-based authorization

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

Table 5.11. Configuration for Apicurio Registry role-based authorization

Environment variable	Java system property	Type	Default value
AUTH_ENABLED	**registry.auth.enabled**	Boolean	**false**
ROLE_BASED_AUTHZ_ENABLED	**registry.auth.role-based-authorization**	Boolean	**false**

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 5.12. Configuration for Apicurio Registry role-based authorization by using Red Hat Single Sign-On

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

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 5.12, "Configuration for Apicurio Registry role-based authorization by using Red Hat Single Sign-On".

Table 5.13. Apicurio Registry roles for authentication and authorization

Role name	Read artifacts	Write artifacts	Global rules	Description
sr-admin	Yes	Yes	Yes	Full access to all create, read, update, and delete operations.

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

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

Table 5.14. Configuration for Apicurio Registry role-based authorization by using internal role mappings

Environment variable	Description	Type	Default
ROLE_BASED_AUTHZ_SOURCE	When set to **application**, user roles are managed internally by Apicurio Registry.	String	**token**

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 5.15. Configuration for Apicurio Registry admin-override

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

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

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 5.16. Configuration for owner-only authorization

Environment variable	Java system property	Type	Default value
AUTH_ENABLED	**registry.auth.enabled**	Boolean	**false**
REGISTRY_AUTH_OBAC_ENABLED	**registry.auth.owner-only-authorization**	Boolean	**false**
REGISTRY_AUTH_OBAC_LIMIT_GROUP_ACCESS	**registry.auth.owner-only-authorization.limit-group-access**	Boolean	**false**

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 5.17. Configuration for authenticated read access

Environment variable	Java system property	Type	Default value
AUTH_ENABLED	**registry.auth.enabled**	Boolean	**false**
REGISTRY_AUTH_AUTHENTICATED_READS_ENABLED	**registry.auth.authenticated-read-access.enabled**	Boolean	**false**

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 5.18. Configuration for anonymous read-only access

Environment variable	Java system property	Type	Default value
AUTH_ENABLED	**registry.auth.enabled**	Boolean	**false**
REGISTRY_AUTH_ANONYMOUS_READ_ACCESS_ENABLED	**registry.auth.anonymous-read-access.enabled**	Boolean	**false**

Additional resources

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

5.5. 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.

WARNING

This kind of connection is not directly available outside of the cluster. Routing is based on hostname, which is encoded in the case of an HTTPS connection. Therefore, edge termination or other configuration is still needed. See Section 5.6, "Configuring an HTTPS connection to Apicurio Registry from outside 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.

```
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
```

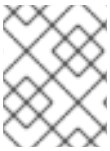
4. Verify that the connection is working:

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

5.6. CONFIGURING AN HTTPS CONNECTION TO APICURIO REGISTRY FROM OUTSIDE THE OPENSHIFT 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:

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  [...]
  labels:
    app: example-apicurioregistry
    [...]
spec:
```

```
  host: example-apicurioregistry-default.apps.example.com
  to:
    kind: Service
    name: example-apicurioregistry-service-9whd7
    weight: 100
  port:
    targetPort: 8080
  tls:
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
  wildcardPolicy: None
```

NOTE

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

If you do not specify a certificate, OpenShift will use a default. 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-apicurioregistry-default.apps.example.com \
  --insecure-policy=Redirect \
  -n default
```

CHAPTER 6. CONFIGURING AND MANAGING APICURIO REGISTRY DEPLOYMENTS

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

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.

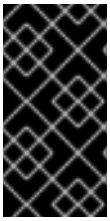
2. Click **Installed Operators** > **Red Hat Integration - Service Registry Operator**
3. On the **ApicurioRegistry** tab, click the Operator custom resource for your deployment, for example, **example-apicurioregistry**.
4. In the main overview page, find the **Deployment Name** section and the corresponding **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

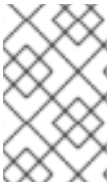
The following environment variables are provided for reference only. The default values are designed for most cases and should only be changed if required by your environment. Any changes to the defaults depend on your hardware, network, and amount of data stored. These values should be kept as low as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

Liveness environment variables

Table 6.1. Environment variables for Apicurio Registry liveness probes

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

Name	Description	Type	Default
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	Seconds	**60**
LIVENESS_STATUS_RESET	Number of seconds that must elapse without any more errors for the liveness probe to reset to OK status.	Seconds	**300**
LIVENESS_ERRORS_IGNORED	Comma-separated list of ignored liveness exceptions.	String	**io.grpc.StatusRuntimeException,org.apache.kafka.streams.errors.InvalidStateStoreException**



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

Name	Description	Type	Default
READINESS_ERROR_THRESHOLD	Number of readiness issues or errors that can occur before the readiness probe fails.	Integer	**1**
READINESS_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.	Seconds	**60**
READINESS_STATUS_RESET	Number of seconds that must elapse without any more errors for the liveness probe to reset to OK status. In this case, this means how long the Pod stays not ready, until it returns to normal operation.	Seconds	**300**

Name	Description	Type	Default
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.	Seconds	**5**

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:

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

```
    - 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
```

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:

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
spec:
  configuration:
    # ...
    ui:
      readOnly: true
    env:
      - name: REGISTRY_UI_FEATURES_READONLY
        value: false
```

   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/offerings/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

podTemplateSpecPreview subfield	Status	Details
metadata.annotations	alternative exists	**spec.deployment.metadata.annotations**
metadata.labels	alternative exists	**spec.deployment.metadata.labels**
spec.affinity	alternative exists	**spec.deployment.affinity**
spec.containers[*]	warning	To configure the Apicurio Registry container, **name: registry** must be used
spec.containers[name = "registry"].env	alternative exists	**spec.configuration.env**
spec.containers[name = "registry"].image	reserved	-
spec.imagePullSecrets	alternative exists	**spec.deployment.imagePullSecrets**
spec.tolerations	alternative exists	**spec.deployment.tolerations**

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

- Kubernetes documentation on Pod templates

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:

- **REGISTRY_UI_CONFIG_APIURL**: Specifies the URL for the core Apicurio Registry server REST API. For example, **https://registry.my-domain.com/apis/registry**
- **REGISTRY_UI_CONFIG_UIURL**: Specifies the URL for the Apicurio Registry web console client. For example, **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:

- **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 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**:

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

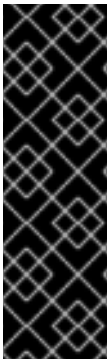
2. Use this **curl** command to change the log level for the logger **io.apicurio.registry.storage** to **DEBUG**:

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

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

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

6.7. 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:

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
```

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

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

IMPORTANT

By default, the 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.

Additional resources

- Extending the Kubernetes API with Custom Resource Definitions

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.

```
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:
```

```
      keycloak:
        url: <string>
        realm: <string>
        apiClientId: <string>
        uiClientId: <string>
      https:
        disableHttp: <bool>
        secretName: <string>
    env: <k8s.io/api/core/v1 []EnvVar>
  deployment:
    replicas: <int32>
    host: <string>
    affinity: <k8s.io/api/core/v1 Affinity>
    tolerations: <k8s.io/api/core/v1 []Toleration>
    imagePullSecrets: <k8s.io/api/core/v1 []LocalObjectReference>
    metadata:
      annotations: <map[string]string>
      labels: <map[string]string>
    managedResources:
      disableIngress: <bool>
      disableNetworkPolicy: <bool>
  disablePodDisruptionBudget: <bool>
    podTemplateSpecPreview: <k8s.io/api/core/v1 PodTemplateSpec>
```

The following table describes each configuration option:

Table 7.1. ApicurioRegistry CR spec configuration options

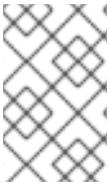
Configuration option	type	Default value	Description
configuration	-	-	Section for configuration of Apicurio Registry application
configuration/persistence	string	*required*	Storage backend. One of **sql**, **kafkasql**
configuration/sql	-	-	SQL storage backend configuration
configuration/sql/dataSource	-	-	Database connection configuration for SQL storage backend
configuration/sql/dataSource/url	string	*required*	Database connection URL string
configuration/sql/dataSource/userName	string	*required*	Database connection user
configuration/sql/dataSource/password	string	*empty*	Database connection password

Configuration option	type	Default value	Description
configuration/kafkasql	-	-	Kafka storage backend configuration
configuration/kafkasql/bootstrapServers	string	*required*	Kafka bootstrap server URL, for Streams storage backend
configuration/kafkasql/security/tls	-	-	Section to configure TLS authentication for Kafka storage backend
configuration/kafkasql/security/tls/truststoreSecretName	string	*required*	Name of a secret containing TLS truststore for Kafka
configuration/kafkasql/security/tls/keystoreSecretName	string	*required*	Name of a secret containing user TLS keystore
configuration/kafkasql/security/scram/truststoreSecretName	string	*required*	Name of a secret containing TLS truststore for Kafka
configuration/kafkasql/security/scram/user	string	*required*	SCRAM user name
configuration/kafkasql/security/scram/passwordSecretName	string	*required*	Name of a secret containing SCRAM user password
configuration/kafkasql/security/scram/mechanism	string	**SCRAM-SHA-512**	SASL mechanism
configuration/ui	-	-	Apicurio Registry web console settings
configuration/ui/readOnly	string	**false**	Set Apicurio Registry web console to read-only mode
configuration/logLevel	string	**INFO**	Apicurio Registry log level, for non-Apicurio components and libraries. One of **INFO**, **DEBUG**

Configuration option	type	Default value	Description
configuration/registryLogLevel	string	**INFO**	Apicurio Registry log level, for Apicurio application components (excludes non-Apicurio components and libraries). One of **INFO**, **DEBUG**
configuration/security	-	-	Apicurio Registry web console and REST API security settings
configuration/security/keycloak	-	-	Web console and REST API security configuration using Red Hat Single Sign-On
configuration/security/keycloak/url	string	*required*	Red Hat Single Sign-On URL
configuration/security/keycloak/realm	string	*required*	Red Hat Single Sign-On realm
configuration/security/keycloak/apiClientId	string	**registry-client-api**	Red Hat Single Sign-On client for REST API
configuration/security/keycloak/uiClientId	string	**registry-client-ui**	Red Hat Single Sign-On client for web console
configuration/security/https	-	-	Configuration for HTTPS. For more details, see Configuring an HTTPS connection to Apicurio Registry from inside the OpenShift cluster.
configuration/security/https/secretName	string	*empty*	Name of a Kubernetes Secret that contains the HTTPS certificate and key, which must be named **tls.crt** and **tls.key**, respectively. Setting this field enables HTTPS, and vice versa.
configuration/security/https/disableHttp	bool	**false**	Disable HTTP port and Ingress. HTTPS must be enabled as a prerequisite.

Configuration option	type	Default value	Description
configuration/env	k8s.io/api/core/v1 []EnvVar	*empty*	Configure a list of environment variables to be provided to the Apicurio Registry pod. For more details, see Managing Apicurio Registry environment variables.
deployment	-	-	Section for Apicurio Registry deployment settings
deployment/replicas	positive integer	**1**	Number of Apicurio Registry pods to deploy
deployment/host	string	*auto-generated*	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.
deployment/affinity	k8s.io/api/core/v1 Affinity	*empty*	Apicurio Registry deployment affinity configuration
deployment/tolerations	k8s.io/api/core/v1 []Toleration	*empty*	Apicurio Registry deployment tolerations configuration
deployment/imagePullSecrets	k8s.io/api/core/v1 []LocalObjectReference	*empty*	Configure image pull secrets for Apicurio Registry deployment
deployment/metadata	-	-	Configure a set of labels or annotations for the Apicurio Registry pod.
deployment/metadata/labels	map[string]string	*empty*	Configure a set of labels for Apicurio Registry pod

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



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:

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

Table 7.2. ApicurioRegistry CR status fields

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

Status field	Type	Description
managedResources/namespace	string	Resource namespace.
managedResources/name	string	Resource name.

7.4. APICURIO REGISTRY MANAGED RESOURCES

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:

```
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

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

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:

```
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

CHAPTER 8. APICURIO REGISTRY CONFIGURATION REFERENCE

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

Name	Type	Default	Available from	Description
registry.api.errors.include-stack-in-response	**boolean**	**false**	**2.1.4.Final**	Include stack trace in errors responses
registry.disable.apis	**optional<list<string>>**		**2.0.0.Final**	Disable APIs

8.1.2. auth

Table 8.2. auth configuration options

Name	Type	Default	Available from	Description
registry.auth.admin-override.claim	**string**	**org-admin**	**2.1.0.Final**	Auth admin override claim
registry.auth.admin-override.claim-value	**string**	**true**	**2.1.0.Final**	Auth admin override claim value
registry.auth.admin-override.enabled	**boolean**	**false**	**2.1.0.Final**	Auth admin override enabled

Name	Type	Default	Available from	Description
registry.auth.admin-override.from	**string**	**token**	**2.1.0.Final**	Auth admin override from
registry.auth.admin-override.role	**string**	**sr-admin**	**2.1.0.Final**	Auth admin override role
registry.auth.admin-override.type	**string**	**role**	**2.1.0.Final**	Auth admin override type
registry.auth.anonymous-read-access.enabled	**boolean [dynamic]**	**false**	**2.1.0.Final**	Anonymous read access
registry.auth.audit.log.prefix	**string**	**audit**	**2.2.6**	Prefix used for application audit logging.
registry.auth.authenticated-read-access.enabled	**boolean [dynamic]**	**false**	**2.1.4.Final**	Authenticated read access
registry.auth.basic-auth-client-credentials.cache-expiration	**integer**	**10**	**2.2.6.Final**	Client credentials token expiration time.
registry.auth.basic-auth-client-credentials.enabled	**boolean [dynamic]**	**false**	**2.1.0.Final**	Enable basic auth client credentials
registry.auth.basic-auth.scope	**optional<string>**		**2.5.0.Final**	Client credentials scope.
registry.auth.client-id	**string**		**2.0.0.Final**	Client identifier used by the server for authentication.
registry.auth.client-secret	**optional<string>**		**2.1.0.Final**	Client secret used by the server for authentication.
registry.auth.enabled	**boolean**	**false**	**2.0.0.Final**	Enable auth
registry.auth.owner-only-authorization	**boolean [dynamic]**	**false**	**2.0.0.Final**	Artifact owner-only authorization
registry.auth.owner-only-authorization.limit-group-access	**boolean [dynamic]**	**false**	**2.1.0.Final**	Artifact group owner-only authorization
registry.auth.role-based-authorization	**boolean**	**false**	**2.1.0.Final**	Enable role based authorization

Name	Type	Default	Available from	Description
registry.auth.role-source	**string**	**token**	**2.1.0.Final**	Auth roles source
registry.auth.role-source.header.name	**string**		**2.4.3.Final**	Header authorization name
registry.auth.roles.admin	**string**	**sr-admin**	**2.0.0.Final**	Auth roles admin
registry.auth.roles.developer	**string**	**sr-developer**	**2.1.0.Final**	Auth roles developer
registry.auth.roles.readonly	**string**	**sr-readonly**	**2.1.0.Final**	Auth roles readonly
registry.auth.tenant-owner-is-admin.enabled	**boolean**	**true**	**2.1.0.Final**	Auth tenant owner admin enabled
registry.auth.token.endpoint	**string**		**2.1.0.Final**	Authentication server url.

8.1.3. cache

Table 8.3. cache configuration options

Name	Type	Default	Available from	Description
registry.config.cache.enabled	**boolean**	**true**	**2.2.2.Final**	Registry cache enabled

8.1.4. ccompat

Table 8.4. ccompat configuration options

Name	Type	Default	Available from	Description
registry.ccompat.legacy-id-mode.enabled	**boolean [dynamic]**	**false**	**2.0.2.Final**	Legacy ID mode (compatibility API)

Name	Type	Default	Available from	Description
registry.ccompat.max-subjects	**integer [dynamic]**	**1000**	**2.4.2.Final**	Maximum number of Subjects returned (compatibility API)
registry.ccompat.use-canonical-hash	**boolean [dynamic]**	**false**	**2.3.0.Final**	Canonical hash mode (compatibility API)

8.1.5. download

Table 8.5. download configuration options

Name	Type	Default	Available from	Description
registry.download.href.ttl	**long [dynamic]**	**30**	**2.1.2.Final**	Download link expiry

8.1.6. events

Table 8.6. events configuration options

Name	Type	Default	Available from	Description
registry.events.ksink	**optional<string>**		**2.0.0.Final**	Events Kafka sink enabled

8.1.7. health

Table 8.7. health configuration options

Name	Type	Default	Available from	Description
registry.liveness.errors.ignored	**optional<list<string>>**		**1.2.3.Final**	Ignored liveness errors
registry.metrics.PersistenceExceptionLivenessCheck.counterResetWindowDurationSec	**integer**	**60**	**1.0.2.Final**	Counter reset window duration of persistence liveness check
registry.metrics.PersistenceExceptionLivenessCheck.disableLogging	**boolean**	**false**	**2.0.0.Final**	Disable logging of persistence liveness check

Name	Type	Default	Available from	Description
registry.metrics.PersistenceExceptionLivenessCheck.errorThreshold	**integer**	**1**	**1.0.2.Final**	Error threshold of persistence liveness check
registry.metrics.PersistenceExceptionLivenessCheck.statusResetWindowDurationSec	**integer**	**300**	**1.0.2.Final**	Status reset window duration of persistence liveness check
registry.metrics.PersistenceTimeoutReadinessCheck.counterResetWindowDurationSec	**integer**	**60**	**1.0.2.Final**	Counter reset window duration of persistence readiness check
registry.metrics.PersistenceTimeoutReadinessCheck.errorThreshold	**integer**	**5**	**1.0.2.Final**	Error threshold of persistence readiness check
registry.metrics.PersistenceTimeoutReadinessCheck.statusResetWindowDurationSec	**integer**	**300**	**1.0.2.Final**	Status reset window duration of persistence readiness check
registry.metrics.PersistenceTimeoutReadinessCheck.timeoutSec	**integer**	**15**	**1.0.2.Final**	Timeout of persistence readiness check
registry.metrics.ResponseErrorLivenessCheck.counterResetWindowDurationSec	**integer**	**60**	**1.0.2.Final**	Counter reset window duration of response liveness check
registry.metrics.ResponseErrorLivenessCheck.disableLogging	**boolean**	**false**	**2.0.0.Final**	Disable logging of response liveness check
registry.metrics.ResponseErrorLivenessCheck.errorThreshold	**integer**	**1**	**1.0.2.Final**	Error threshold of response liveness check
registry.metrics.ResponseErrorLivenessCheck.statusResetWindowDurationSec	**integer**	**300**	**1.0.2.Final**	Status reset window duration of response liveness check
registry.metrics.ResponseTimeoutReadinessCheck.counterResetWindowDurationSec	**instance<integer>**	**60**	**1.0.2.Final**	Counter reset window duration of response readiness check

Name	Type	Default	Available from	Description
registry.metrics.ResponseTimeoutReadinessCheck.errorThreshold	**instance<integer>**	**1**	**1.0.2.Final**	Error threshold of response readiness check
registry.metrics.ResponseTimeoutReadinessCheck.statusResetWindowDurationSec	**instance<integer>**	**300**	**1.0.2.Final**	Status reset window duration of response readiness check
registry.metrics.ResponseTimeoutReadinessCheck.timeoutSec	**instance<integer>**	**10**	**1.0.2.Final**	Timeout of response readiness check
registry.storage.metrics.cache.check-period	**long**	**30000**	**2.1.0.Final**	Storage metrics cache check period

8.1.8. import

Table 8.8. import configuration options

Name	Type	Default	Available from	Description
registry.import.url	**optional<url>**		**2.1.0.Final**	The import URL

8.1.9. kafka

Table 8.9. kafka configuration options

Name	Type	Default	Available from	Description
registry.events.kafka.topic	**optional<string>**		**2.0.0.Final**	Events Kafka topic
registry.events.kafka.topic-partition	**optional<integer>**		**2.0.0.Final**	Events Kafka topic partition

8.1.10. limits

Table 8.10. limits configuration options

Name	Type	Default	Available from	Description
registry.limits.config.max-artifact-labels	**long**	**-1**	**2.2.3.Final**	Max artifact labels
registry.limits.config.max-artifact-properties	**long**	**-1**	**2.1.0.Final**	Max artifact properties
registry.limits.config.max-artifacts	**long**	**-1**	**2.1.0.Final**	Max artifacts
registry.limits.config.max-description-length	**long**	**-1**	**2.1.0.Final**	Max artifact description length
registry.limits.config.max-label-size	**long**	**-1**	**2.1.0.Final**	Max artifact label size
registry.limits.config.max-name-length	**long**	**-1**	**2.1.0.Final**	Max artifact name length
registry.limits.config.max-property-key-size	**long**	**-1**	**2.1.0.Final**	Max artifact property key size
registry.limits.config.max-property-value-size	**long**	**-1**	**2.1.0.Final**	Max artifact property value size
registry.limits.config.max-requests-per-second	**long**	**-1**	**2.2.3.Final**	Max artifact requests per second
registry.limits.config.max-schema-size-bytes	**long**	**-1**	**2.2.3.Final**	Max schema size (bytes)
registry.limits.config.max-total-schemas	**long**	**-1**	**2.1.0.Final**	Max total schemas
registry.limits.config.max-versions-per-artifact	**long**	**-1**	**2.1.0.Final**	Max versions per artifacts
registry.storage.metrics.cache.max-size	**long**	**1000**	**2.4.1.Final**	Storage metrics cache max size.

8.1.11. log

Table 8.11. log configuration options

Name	Type	Default	Available from	Description
quarkus.log.level	**string**		**2.0.0.Final**	Log level

8.1.12. redirects

Table 8.12. redirects configuration options

Name	Type	Default	Available from	Description
registry.enable-redirects	**boolean**		**2.1.2.Final**	Enable redirects
registry.redirects	**map<string, string>**		**2.1.2.Final**	Registry redirects
registry.url.override.host	**optional<string>**		**2.5.0.Final**	Override the hostname used for generating externally-accessible URLs. The host and port overrides are useful when deploying Registry with HTTPS passthrough Ingress or Route. In cases like these, the request URL (and port) that is then re-used for redirection does not belong to actual external URL used by the client, because the request is proxied. The redirection then fails because the target URL is not reachable.
registry.url.override.port	**optional<integer>**		**2.5.0.Final**	Override the port used for generating externally-accessible URLs.

8.1.13. rest

Table 8.13. rest configuration options

Name	Type	Default	Available from	Description
registry.rest.artifact.deletion.enabled	**boolean [dynamic]**	**false**	**2.4.2-SNAPSHOT**	Enables artifact version deletion
registry.rest.artifact.download.maxSize	**int**	**1000000**	**2.2.6-SNAPSHOT**	Max size of the artifact allowed to be downloaded from URL
registry.rest.artifact.download.skipSSLValidation	**boolean**	**false**	**2.2.6-SNAPSHOT**	Skip SSL validation when downloading artifacts from URL

8.1.14. store

Table 8.14. store configuration options

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

8.1.15. ui

Table 8.15. ui configuration options

Name	Type	Default	Available from	Description
quarkus.oidc.tenant-enabled	**boolean**	**false**	**2.0.0.Final**	UI OIDC tenant enabled
registry.ui.config.apiUrl	**string**		**1.3.0.Final**	UI APIs URL

Name	Type	Default	Available from	Description
registry.ui.config.auth.oidc.client-id	**string**	**none**	**2.2.6.Final**	UI auth OIDC client ID
registry.ui.config.auth.oidc.redirect-url	**string**	**none**	**2.2.6.Final**	UI auth OIDC redirect URL
registry.ui.config.auth.oidc.url	**string**	**none**	**2.2.6.Final**	UI auth OIDC URL
registry.ui.config.auth.type	**string**	**none**	**2.2.6.Final**	UI auth type
registry.ui.config.uiCodegenEnabled	**boolean**	**true**	**2.4.2.Final**	UI codegen enabled
registry.ui.config.uiContextPath	**string**	**/ui/**	**2.1.0.Final**	UI context path
registry.ui.features.readOnly	**boolean [dynamic]**	**false**	**1.2.0.Final**	UI read-only mode
registry.ui.features.settings	**boolean**	**false**	**2.2.2.Final**	UI features settings
registry.ui.root	**string**		**2.3.0.Final**	Overrides the UI root context (useful when relocating the UI context using an inbound proxy)

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 2024-02-22 17:15:05 UTC