Red Hat Integration 2019-12

Getting Started with Service Registry

Getting Started with Service Registry

Last Updated: 2020-02-21

Red Hat Integration 2019-12 Getting Started with Service Registry

Getting Started with Service Registry

Abstract

This guide introduces Service Registry and explains how to install it in your storage environment.

Table of Contents

CHAPTER 1. INTRODUCTION TO SERVICE REGISTRY

This topic provides an overview of Service Registry features and explains the optional rules used to govern registry content:

- Section 1.1, “Service Registry overview”
- Section 1.2, “Rules for registry content”

IMPORTANT

Service Registry 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.

1.1. SERVICE REGISTRY OVERVIEW

Service Registry is a datastore for standard event schemas and API designs. Service Registry enables enables you to decouple the structure of your data from your applications and to share and manage your data structure using a REST interface. For example, client applications can dynamically push/pull the latest schema updates to/from the registry at runtime without needing to redeploy.

Service Registry enables you to create optional rules to govern how registry content can evolve over time. For example, the available registry rule types include content validation and version compatibility. Service Registry also provides full Apache Kafka schema registry support, client serializer/deserializer (SerDe) to validate Kafka messages at runtime, and compatibility with existing Confluent Schema Registry clients.

Service Registry is built on the Apicurio Registry open source community project. For details, see https://github.com/apicurio/apicurio-registry.

1.1.1. Supported artifact types

Service Registry supports the following artifact types:

Type	Description
AVRO	Apache Avro schema
PROTOBUF	Google protocol buffers schema
JSON	JSON Schema
OPENAPI	OpenAPI specification
ASYNCAPI	AsyncAPI specification

1.1.2. Storage implementations

Service Registry supports the following artifact storage implementations:

- Red Hat AMQ Streams 1.3

1.1.3. Available distributions

Service Registry distributions are available as follows:

Distribution	Location
Container image	Red Hat Container Catalog
Maven repository	Software Downloads for Fuse v7.5.0
Full Maven repository (with all dependencies)	Software Downloads for Fuse v7.5.0
Source code	Software Downloads for Fuse v7.5.0

Both Maven repositories include a Kafka client serializer/deserializer implementation, which can be used by Kafka client developers to integrate with Service Registry. This implementation includes custom **Serde** Java classes, which enable Kafka client applications to push/pull their schemas from Service Registry at runtime.

NOTE

You must have a subscription for Red Hat Fuse and be logged into the Red Hat Customer Portal to access the available Service Registry distributions.

1.1.4. Registry REST API

The Service Registry REST API enables client applications to manage the artifacts in the registry. It provides create, read, update, and delete operations for the following:

- Schema and API artifacts
- Artifact versions
- Artifact metadata
- Global rules
- Artifact rules

For detailed information, see the Apicurio Registry REST API documentation .

Compatibility with Confluent Schema Registry

The Service Registry REST API is compatible with the Confluent Schema Registry REST API. This means that applications using Confluent client libraries can use Service Registry instead as a drop-in replacement. For more details, see Replacing Confluent Schema Registry with Red Hat Integration Service Registry.

1.1.5. Registry demo

Apicurio Registry provides an open source demo of Apache Avro serialization/deserialization based on storage in Apache Kafka Streams. This demo shows how the serializer/deserializer gets the Avro schema from the registry at runtime and then uses it to serialize and deserialize Kafka messages. For more details, see https://github.com/Apicurio/apicurio-registry-demo.

For another demo of Avro serialization/deserialization, this time with storage in an Apache Kafka cluster based on Strimzi, see the Red Hat Developer article on Getting Started with Red Hat Integration Service Registry.

1.2. RULES FOR REGISTRY CONTENT

To govern content evolution, you can configure optional rules for artifacts added to the registry, as a post-installation step. All rules configured for an artifact must pass before a new artifact version can be uploaded to the registry. The goal of these rules is to prevent invalid content from being added to the registry. For example, content can be invalid for the following reasons:

- Invalid syntax for a given artifact type (for example, **AVRO** or **PROTOBUF**)
- Valid syntax, but semantics violate company standards
- New content includes breaking changes to the current artifact version

1.2.1. When rules are applied

Rules are applied only when content is added to the registry. This includes the following REST operations:

- Creating an artifact
- Updating an artifact
- Creating an artifact version

If a rule is violated, Service Registry returns an HTTP error. The response body includes the violated rule and a message showing what went wrong.

NOTE

If no rules are configured for an artifact, the set of currently configured global rules are applied.

1.2.2. How rules work

Each rule is simply a name and some optional configuration. The registry storage maintains the list of rules for each artifact and the list of global rules. Each rule in the list consists of a name and a set of configuration properties, which are specific to the rule implementation. For example, a validation rule might use a **Map<String,String>**, or a compatibility rule might use a single property of **BACKWARD** for compatibility with existing versions.

A rule is provided with the content of the current version of the artifact (if one exists) and the new version of the artifact being added. The rule implementation returns true or false depending on whether the artifact passes the rule. If not, the registry reports the reason why in an HTTP error response. Some rules might not use the previous version of the content. For example, compatibility rules use previous versions, but syntax or semantic validity rules do not.

1.2.3. Supported rule types

You can specify the following rule types to govern content evolution in the registry:

Type	Description
VALIDITY	Validates data before adding it to the registry. Includes the following values: • **FULL**: The validation is both syntax and semantic. • **SYNTAX_ONLY**: The validation is syntax only.
COMPATIBILITY	Ensures that newly added artifacts are compatible with previously added versions. Includes the following values: • **FULL**: The new artifact is forward and backward compatible with the most recently added artifact. • **FULL_TRANSITIVE**: The new artifact is forward and backward compatible with all previously added artifacts. • **BACKWARD**: Clients using the new artifact can read data written using the most recently added artifact. • **BACKWARD_TRANSITIVE**: Clients using the new artifact can read data written using all previously added artifacts. • **FORWARD**: Clients using the most recently added artifact can read data written using the new artifact. • **FORWARD_TRANSITIVE**: Clients using all previously added artifacts can read data written using the new artifact. • **NONE**: All backward and forward compatibility checks are disabled.

CHAPTER 2. INSTALLING SERVICE REGISTRY

This topic explains how to install and run the Service Registry container image with the following storage options:

- Section 2.1, "Installing Service Registry with AMQ Streams storage on OpenShift"

Prerequisites

- Section 1.1, "Service Registry overview"

2.1. INSTALLING SERVICE REGISTRY WITH AMQ STREAMS STORAGE ON OPENSHIFT

This topic explains how to install and run Service Registry with storage in Red Hat AMQ Streams on OpenShift from a container image. This storage option is suitable for production environments.

The following versions are supported:

- Red Hat AMQ Streams 1.3

Prerequisites

- You must have an OpenShift cluster with cluster administrator access. OpenShift 3.11 and 4.x are supported.
- You must have already installed AMQ Streams on your cluster. For more details, see Getting Started with AMQ Streams on OpenShift.
- Ensure that you can access the Service Registry image in the Container Catalog :
  - Create a service account and pull secret for the image. For details, see Container Service Accounts.
  - Download the pull secret and submit it to your OpenShift cluster. For example:

    ```
    $ oc create -f 11223344_service-registry-secret.yaml --namespace=my-project
    ```

Procedure

1. Get the Service Registry OpenShift template.
2. Enter the following command to get the name of the Kafka bootstrap service running on your OpenShift cluster:

   ```
   $ oc get services | grep .*kafka-bootstrap
   ```

3. Create a new OpenShift application and specify the following parameters:
   - **service-registry-template.yml**: The OpenShift template file for Service Registry.
   - **KAFKA_BOOTSTRAP_SERVERS**: The name of the Kafka bootstrap service on your cluster, followed by the Kafka broker port (for example, **my-cluster-kafka-bootstrap:9092**).

- **REGISTRY_ROUTE**: The name of the route that will expose Service Registry, which is based on your cluster environment (for example, **my-cluster-service-registry-myproject.example.com**).
  For example:

  ```
  $ oc new-app service-registry-template.yml -p KAFKA_BOOTSTRAP_SERVERS=my-cluster-kafka-bootstrap:9092 -p REGISTRY_ROUTE=my-cluster-service-registry-myproject.example.com
  ```

  You should see output such as the following:

  ```
  Deploying template "myproject/service-registry" for "service-registry-template.yml" to project myproject

       service-registry
       ---------
       Congratulations on deploying Service Registry into OpenShift!

       All components have been deployed and configured.

       * With parameters:
          * Registry Route Name=my-cluster-service-registry-myproject.example.com
          * Registry Max Memory Limit=1300Mi
          * Registry Memory Requests=600Mi
          * Registry Max CPU Limit=1
          * Registry CPU Requests=100m
          * Kafka Bootstrap Servers=my-cluster-kafka-bootstrap:9092

  --> Creating resources ...
      imagestream.image.openshift.io "registry" created
      service "service-registry" created
      deploymentconfig.apps.openshift.io "service-registry" created
      route.route.openshift.io "service-registry" created
  --> Success
      Access your application via route 'my-cluster-service-registry-myproject.example.com'
      Run 'oc status' to view your app.
  ```

4. Send a test request using the Service Registry REST API. For example, enter the following **curl** command to create a simple Avro schema artifact for a share price application in the registry:

   ```
   $ curl -X POST -H "Content-type: application/json; artifactType=AVRO" -H "X-Registry-ArtifactId: prices-value" --data '{"type":"record","name":"price","namespace":"com.redhat","fields":[{"name":"symbol","type":"string"},{"name":"price","type":"string"}]}' my-cluster-service-registry-myproject.example.com/artifacts
   ```

5. Verify that the response includes the expected JSON body to confirm that the Avro schema artifact was created in the registry. For example:

   ```
   {"createdOn":1578310374517,"modifiedOn":1578310374517,"id":"prices-value","version":1,"type":"AVRO","globalId":8}
   ```

Additional resources

- For more REST API sample requests, see the Registry REST API documentation .
- For more details on registry demo applications:
    - https://github.com/Apicurio/apicurio-registry-demo
    - Getting Started with Red Hat Integration Service Registry
- For more details on AMQ Streams:
    - Using AMQ Streams on OpenShift
    - How to run AMQ Streams on Minishift
- For more details on OpenShift templates, see the OpenShift user documention.

APPENDIX A. USING YOUR SUBSCRIPTION

Fuse 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 Fuse** entries in the **Integration and Automation** category.
3. Select the desired Fuse product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

Registering your system for packages

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

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

To learn more see How to Register and Subscribe a System to the Red Hat Customer Portal .