Red Hat Fuse 7.2

Designing APIs with Apicurito

Designing REST APIs for Fuse on OpenShift applications

Last Updated: 2020-03-27

Red Hat Fuse 7.2 Designing APIs with Apicurito

Designing REST APIs for Fuse on OpenShift applications

Abstract

Guide to using the Apicurito web-based REST API designer

Table of Contents

PREFACE

Red Hat Fuse on OpenShift provides Apicurito, a web-based API designer, that you can use to design REST APIs that comply with the OpenAPI 2.0 specification, a vendor-neutral and portable open description format for API services. Apicurito is a “light” version of the Apicurio Studio open source project (https://www.apicur.io/). This means that your Apicurito design sessions are stateless and you must save your API definition as a JSON file at the end of each session.

You can also use Apicurito to generate a preliminary Fuse project based on a REST API definition. In your Fuse development environment, you can then complete the project’s Camel routes and build the project. Finally, you can deploy the resulting REST service on Fuse on OpenShift.

Here is an overview of how you can use Apicurito to incorporate REST APIs in your Fuse on OpenShift application solution:

1. Add Apicurito as a service to your OpenShift project.

2. In the Apicurito web-based designer:

    - Create an API definition with Apicurito. Save the REST API definition as a JSON file to your local file system. You can save your API definition at any point during your editing session, even if the API definition is not complete.

    - Upload an API definition to Apicurito.

    - Generate a Fuse Camel project based on the current REST API definition. Apicurito provides a downloadable zip file that contains a complete Maven project.

3. In your Fuse development environment, complete the skeleton implementation provided by the generated Fuse project.

4. Build and deploy the Fuse application to OpenShift.

5. (Optional) Integrate the Fuse application with Red Hat 3scale API Management, using the 3scale service discovery capability to find and configure your Fuse application.

The following chapters provide details on how to use Apicurito:

- Chapter 1, *Add Apicurito as a service to your OpenShift project*

- Chapter 2, *Design and develop an API definition with Apicurito*

- Chapter 3, *Implement, build, and deploy a Fuse application based on a REST API*

For information about the 3scale service discovery capability, see the Chapter 4, *Discover an API service in 3scale* chapter.

CHAPTER 1. ADD APICURITO AS A SERVICE TO YOUR OPENSHIFT PROJECT

You can add Apicurito as a service to your OpenShift project from the OpenShift service catalog. You can access this instance at a URL external to the OpenShift environment.

Prerequisites

- Obtain the hostname that will allow you to access Apicurito by following the guidelines recommended by your OpenShift system administrator.
- Verify that the Fuse on OpenShift images and templates, including **apicurito-ui** and **fuse-apicurito-generator**, are installed on your OpenShift cluster, by running the following command in a command window:

  ```
  oc get is -n openshift
  ```

  If the images and templates are not pre-installed, or if the provided versions are out of date, install (or update) the Fuse on OpenShift images and templates as described in the Fuse on OpenShift Guide.

Procedure

To add Apicurito as a service to your OpenShift project:

1. In a command window, log in to the OpenShift server:

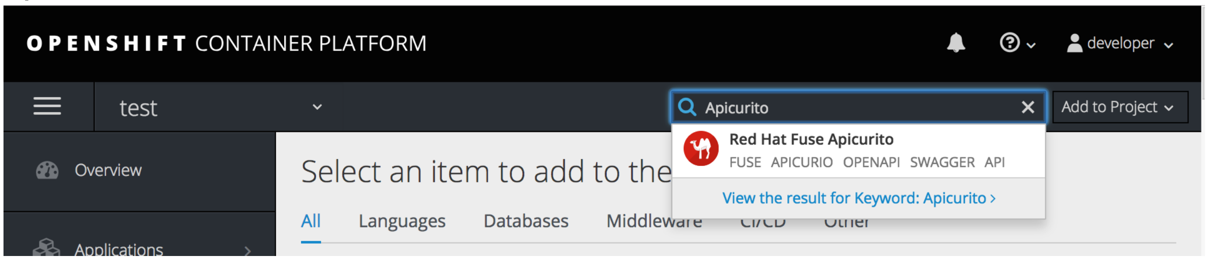
   ```
   oc login -u developer -p developer
   ```

2. Create a new project namespace. For example, the following command creates a new project named **test**:

   ```
   oc new-project test
   ```

3. In your web browser, open the OpenShift console and log in with your credentials (for example, username **developer** and password **developer**).

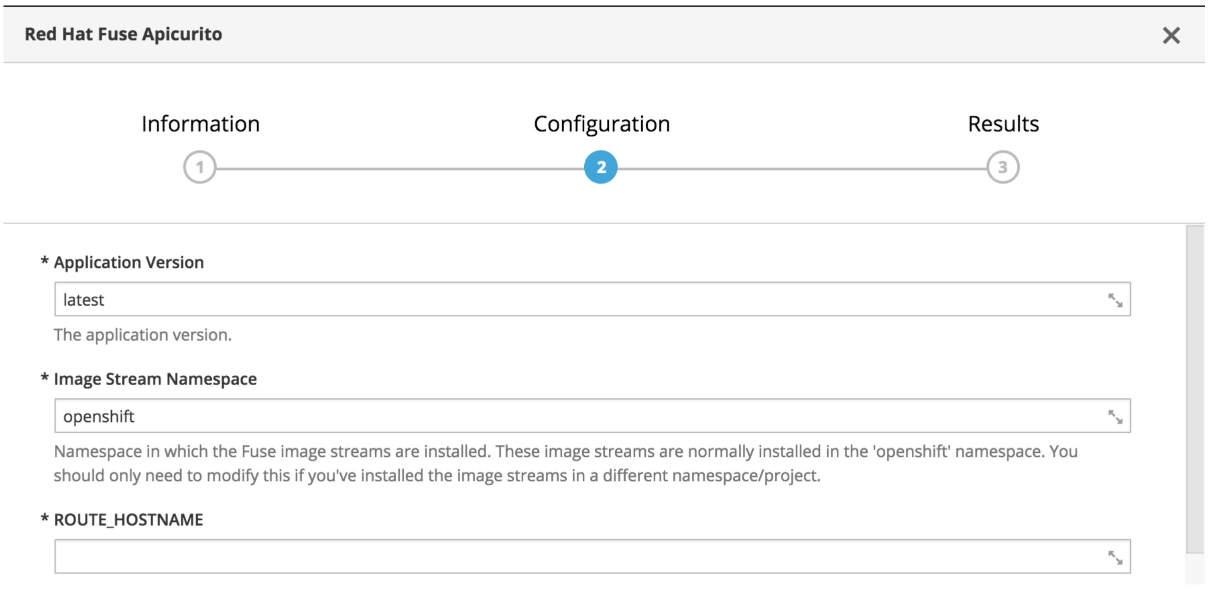
4. Click **Catalog**. In the Catalog search field, type **Apicurito** and then select **Red Hat Fuse Apicurito**.

   
   OPENSHIFT CONTAINER PLATFORM
developer
test
Apicurito
Add to Project
Red Hat Fuse Apicurito
FUSE APICURIO OPENAPI SWAGGER API
View the result for Keyword: Apicurito ›
Overview
Applications
Select an item to add to the
All
Languages
Databases
Middleware
CI/CD
Other

   The **Information** step of the Red Hat Fuse Apicurito wizard opens.

5. Click **Next**.
   The **Configuration** step of the Red Hat Fuse Apicurito wizard opens.

6. In the **Image Stream Namespace** field, type **openshift**.



* Application Version

latest

The application version.

* Image Stream Namespace

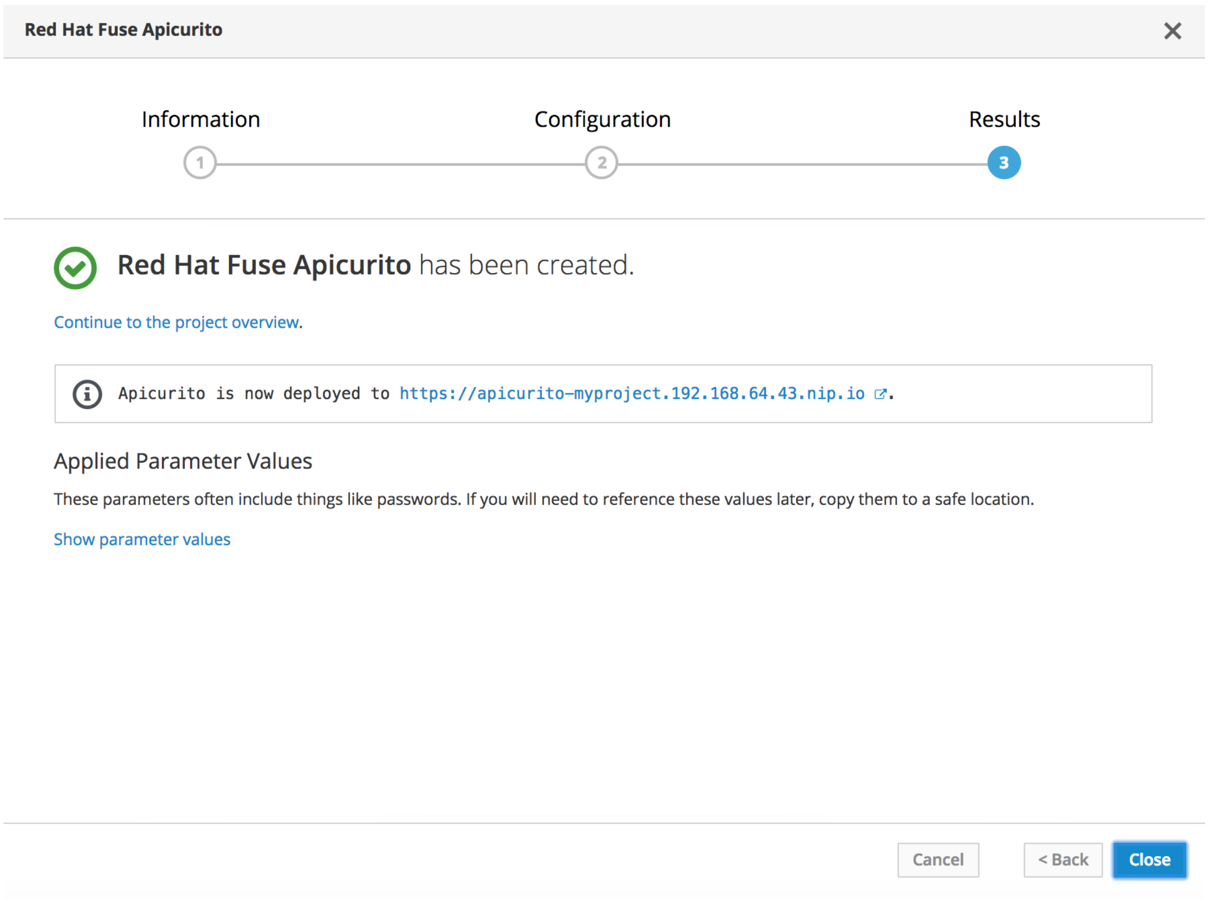
openshift

Namespace in which the Fuse image streams are installed. These image streams are normally installed in the 'openshift' namespace. You should only need to modify this if you've installed the image streams in a different namespace/project.

* ROUTE_HOSTNAME

7. In the **ROUTE_HOSTNAME** field, type the external hostname that allows you to access the Apicurito instance, for example **apicurito-myproject.192.168.64.43.nip.io**.

8. Accept the default values for the rest of the settings in the **Configuration** step and click **Create**.
   The **Results** step of the template wizard opens.



Red Hat Fuse Apicurito has been created.

Continue to the project overview.

```
Apicurito is now deployed to https://apicurito-myproject.192.168.64.43.nip.io.
```

Applied Parameter Values

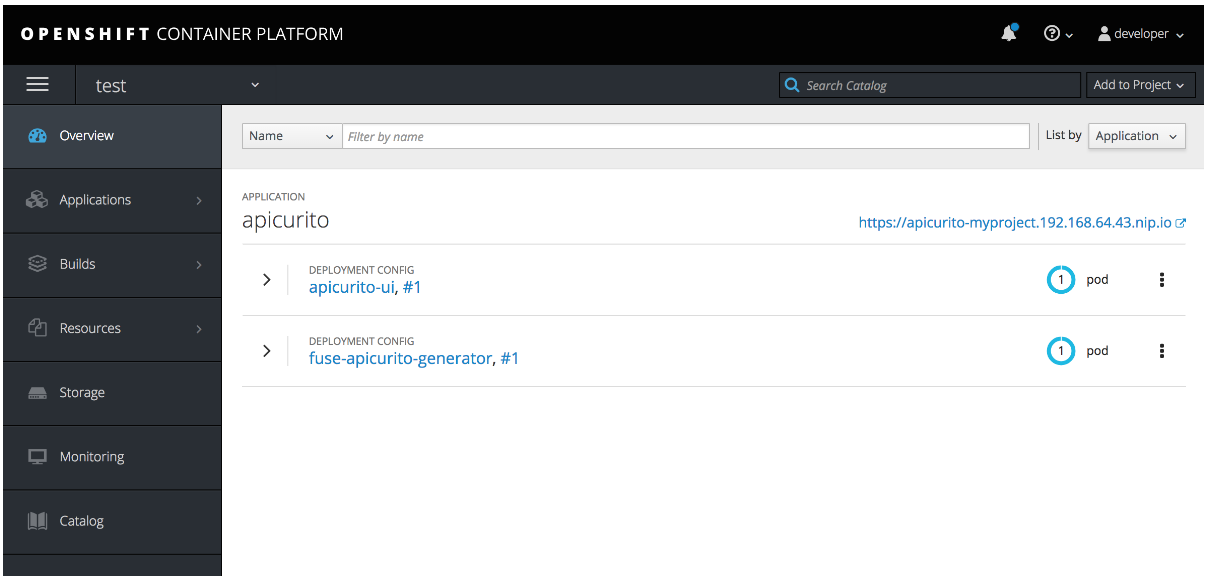
These parameters often include things like passwords. If you will need to reference these values later, copy them to a safe location.

Show parameter values

9. Click **Close**.

10. In the OpenShift web console, in the **My Projects** pane, select the project, for example select **test**.
    The project's **Overview** tab opens, showing the **apicurito-ui** application.

11. Click the arrow to the left of the **apicurito-ui** deployment to expand and view the deployment details:

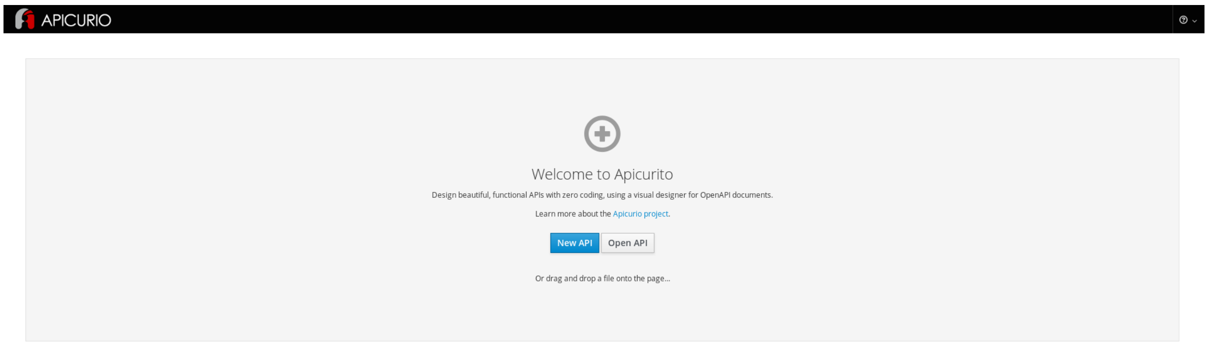


APPLICATION
apicurito
https://apicurito-myproject.192.168.64.43.nip.io

DEPLOYMENT CONFIG
apicurito-ui, #1
1 pod

DEPLOYMENT CONFIG
fuse-apicurito-generator, #1
1 pod

12. Click the link for the Apicurito instance, for example **https://apicurito-myproject.192.168.64.43.nip.io**.
Apicurito opens in a new web browser window or tab:



Welcome to Apicurito
Design beautiful, functional APIs with zero coding, using a visual designer for OpenAPI documents.
Learn more about the Apicurio project.
Or drag and drop a file onto the page...

NOTE

If you cannot open the Apicurito instance, you might need to edit your computer's **/etc/hosts** file to add the ROUTE_HOSTNAME using the following syntax, where **$OPENSHIFT_IP_ADDR** is the IP address for the OpenShift server and **apicurito.my-minishift.apicurio.io** is the ROUTE_HOSTNAME that you specified in step 7.

```
$OPENSHIFT_IP_ADDR apicurito.my-minishift.apicurio.io
```

CHAPTER 2. DESIGN AND DEVELOP AN API DEFINITION WITH APICURITO

You can use Apicurito to design and develop a REST API definition that complies with the OpenAPI 2.0 specification.

Prerequisites

- You created an OpenShift project.
- You added the Apicurito service to your OpenShift project.

2.1. CREATING A REST API DEFINITION

The following steps describe how to create an example REST API definition for managing address data (including information such as street, city, state, zipcode, and so on). To implement the address example, you create two paths - one for addresses and one for a specific address. You then define operations to get a list of all addresses, add an address, update an address, get the details of an address, and delete an address.

NOTE

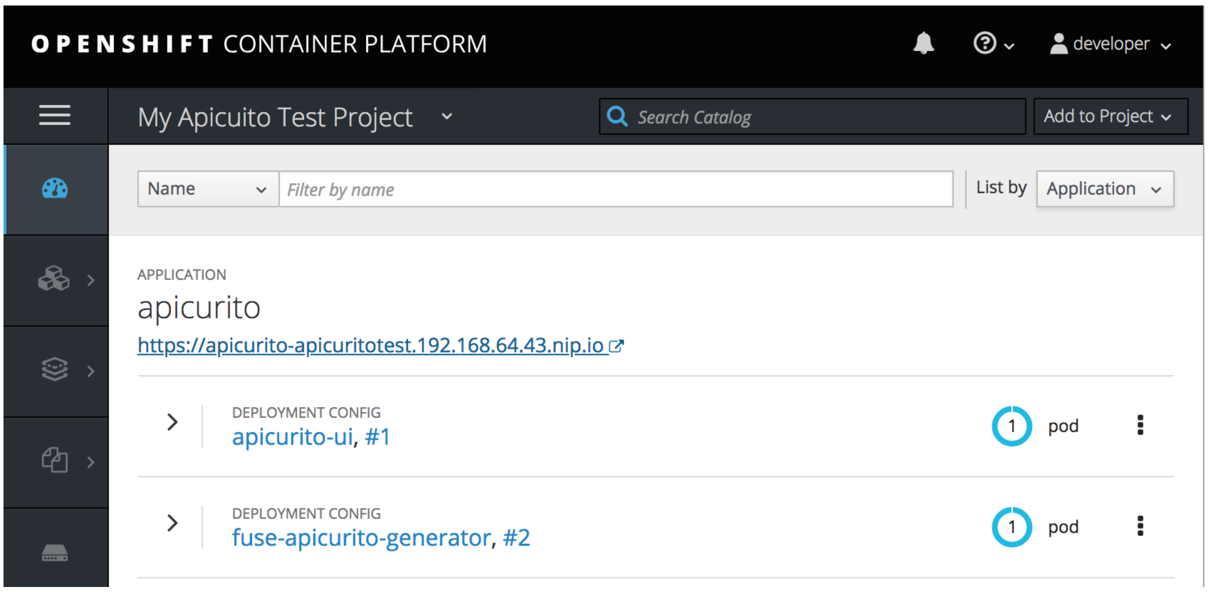
Apicurito is stateless, meaning that it does not save your work between OpenShift sessions. You need to save the API to your local file system between sessions.

Prerequisites

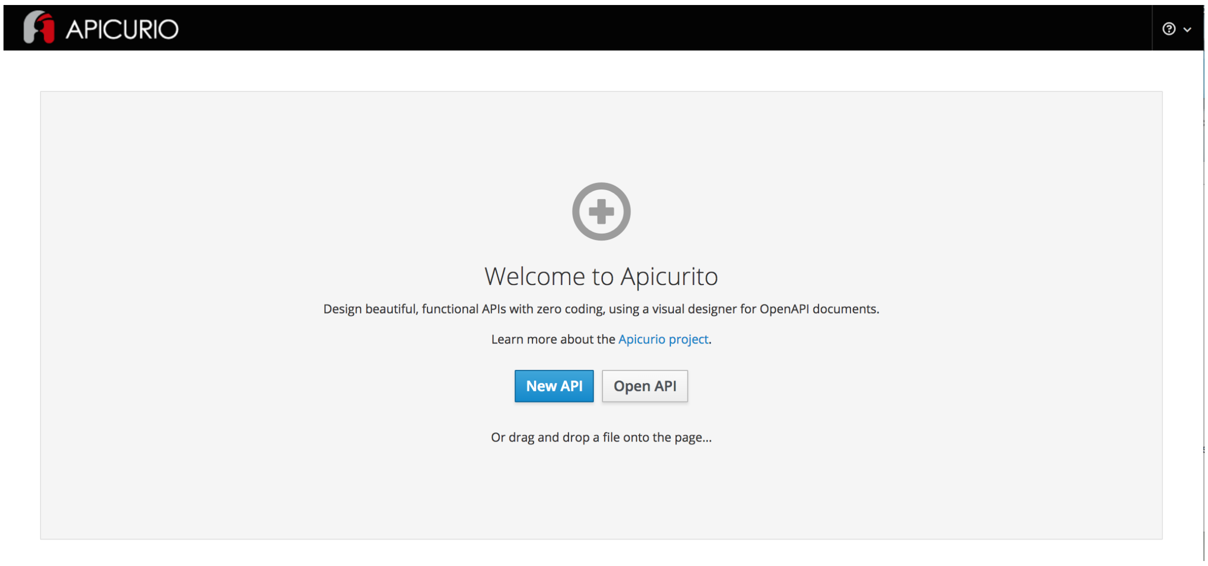
- You created an OpenShift project.
- You added the Apicurito service to your OpenShift project.
- You know the endpoints for the API that you want to create. For the Address example, there are two endpoints: **addresses** and **addresses{addressId}**.

Procedure

1. Log in to your OpenShift web console and then open the project that contains Apicurito.
2. In the list of applications, click the URL for Apicurito, for example https://apicurito-myproject.192.168.64.43.nip.io



A new browser window or tab opens for Apicurito:



Design beautiful, functional APIs with zero coding, using a visual designer for OpenAPI documents.

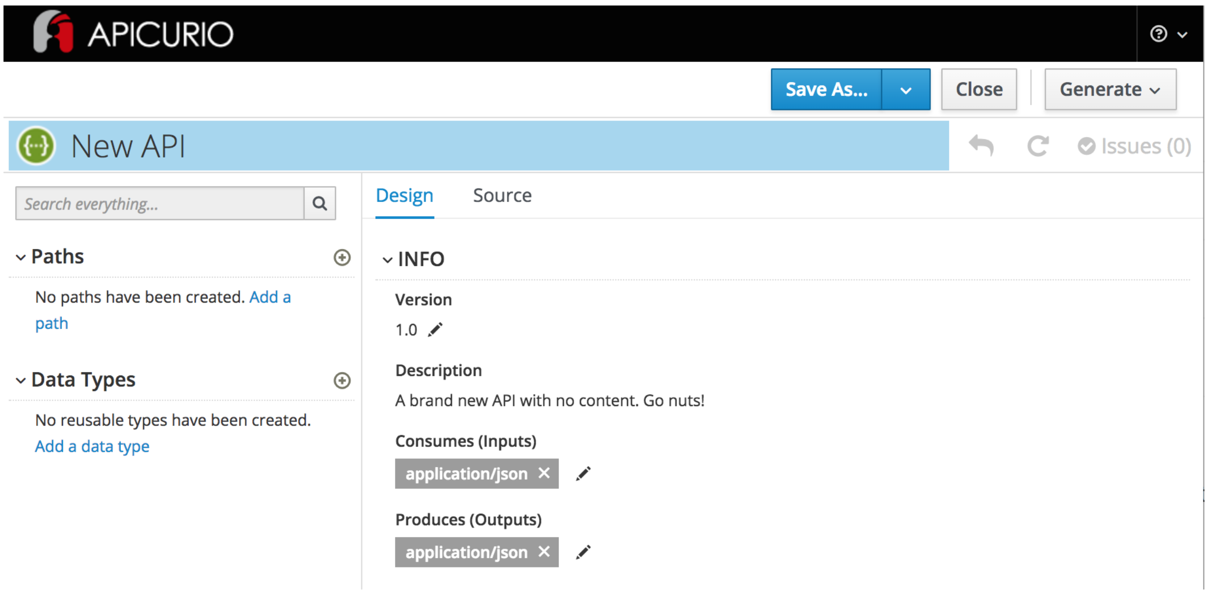
Learn more about the Apicurio project.

Or drag and drop a file onto the page...

NOTE

Because Apicurito is a “light” version of the Apicurio Studio open source project, "Apicurio" shows in the Apicurio interface.

3. Click **New API**.
   A new API page opens.



New API

Paths

No paths have been created. Add a path

Data Types

No reusable types have been created. Add a data type

INFO

Version

1.0

Description

A brand new API with no content. Go nuts!

Consumes (Inputs)

application/json

Produces (Outputs)

application/json

By default, the API name is **New API**, the version is **1.0**, and the input and output types are **application/json**.

4. To change the API name:

   a. Hover the cursor over the name and then click the edit icon (✎) that appears.

   b. Edit the name. For example, type **AddressBook API**.

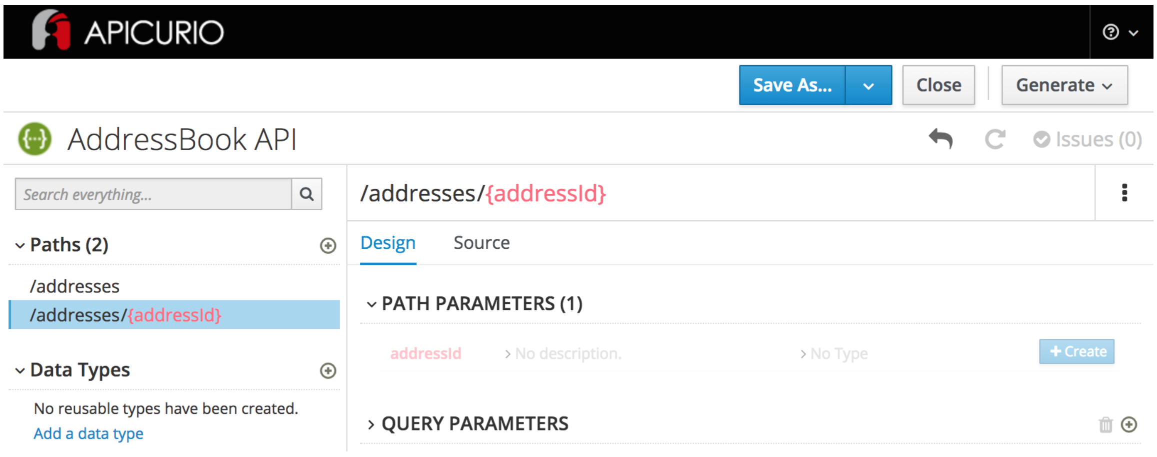
   c. Click **Save**.

5. Optionally:

   - Add your contact information (name, email address, and URL).
   - Select a license.
   - Define tags.
   - Select a security scheme.
   - Specify security requirements.

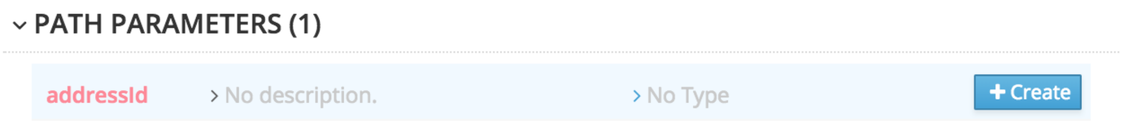
6. Define a relative path to each individual endpoint of the API. The field name must begin with a slash (/).
   For the AddressBook API example, create two paths:

   - A path for addresses: **/addresses**
   - A path for a specific address by ID: **/addresses/{addressId}**



7. Specify the type of any path parameters.
   For the **addressID** parameter example:

   a. In the **Paths** list, click /**addresses**/**{addressId}**.
      The **addressId** parameter appears in the **PATH PARAMETERS** section.



   b. Click **Create**, and then click **Edit**.

   c. For the type, select **string**.

   d. Accept **string** for the format, and then click **OK**.

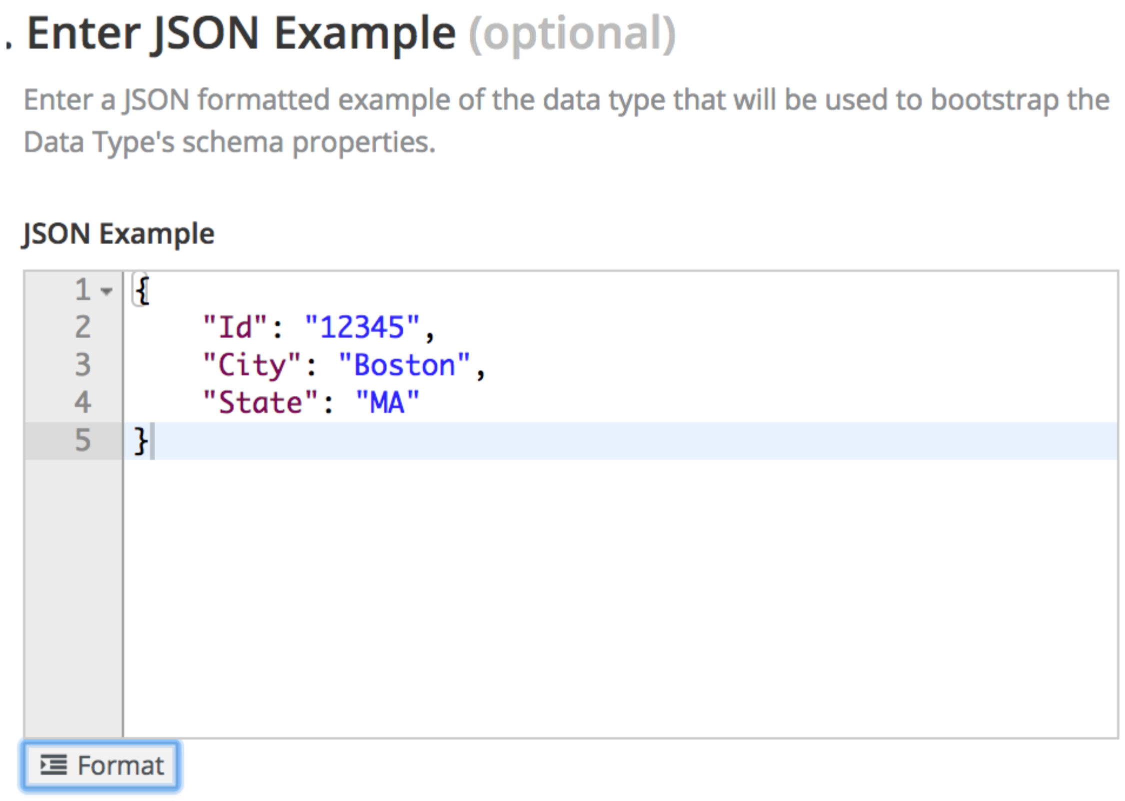
8. In the **Data Types** section, define reusable types for the API.

   a. Click **Add a data type**

   b. In the **Add Data Type** dialog, type a name, for example **Address**.

   c. Optionally, you can provide an example from which Apicurito creates the data type's schema. You can then edit the generated schema.
      For the AddressBook API example, start with the following JSON example:

```
{
"Id" : "12345",
"City" : "Boston",
"State" : "MA"
}
```


. Enter JSON Example (optional)

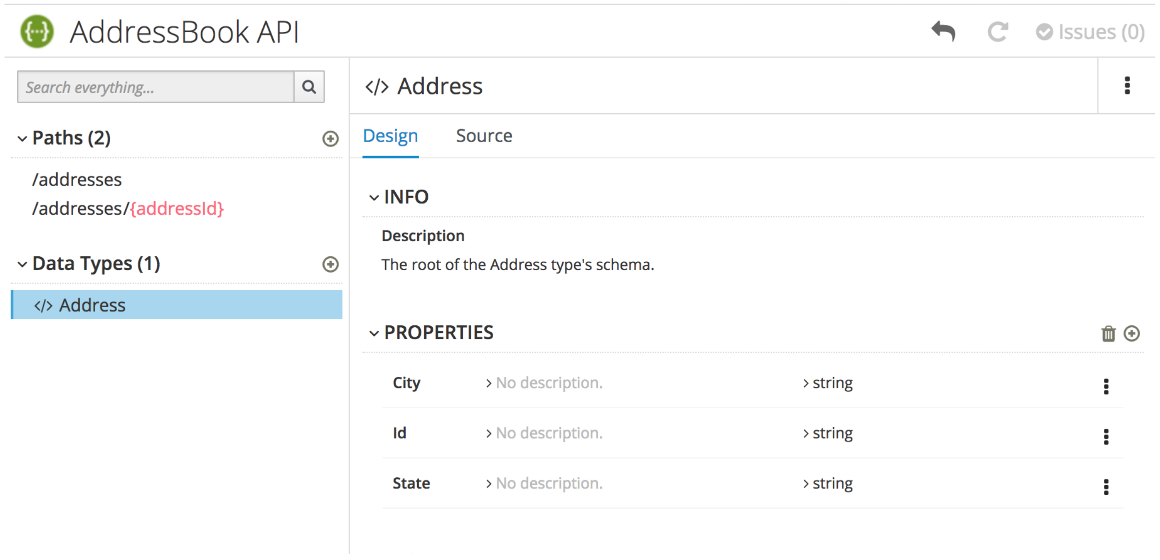
Enter a JSON formatted example of the data type that will be used to bootstrap the Data Type's schema properties.

JSON Example

```
{
    "Id": "12345",
    "City": "Boston",
    "State": "MA"
}
```

d. Optionally, you can choose to create a REST Resource with the data type.

e. Click **Save**. If you provided an example, Apicurito generates a schema from the example:



Description

The root of the Address type's schema.

PROPERTIES

City	No description.	string
Id	No description.	string
State	No description.	string

You can add edit the properties and add new ones, for example street, zipcode, and country.

9. For each path, define operations (GET, PUT, POST, DELETE, OPTIONS, HEAD, and PATCH). For the AddressBook API example, define the following operations:

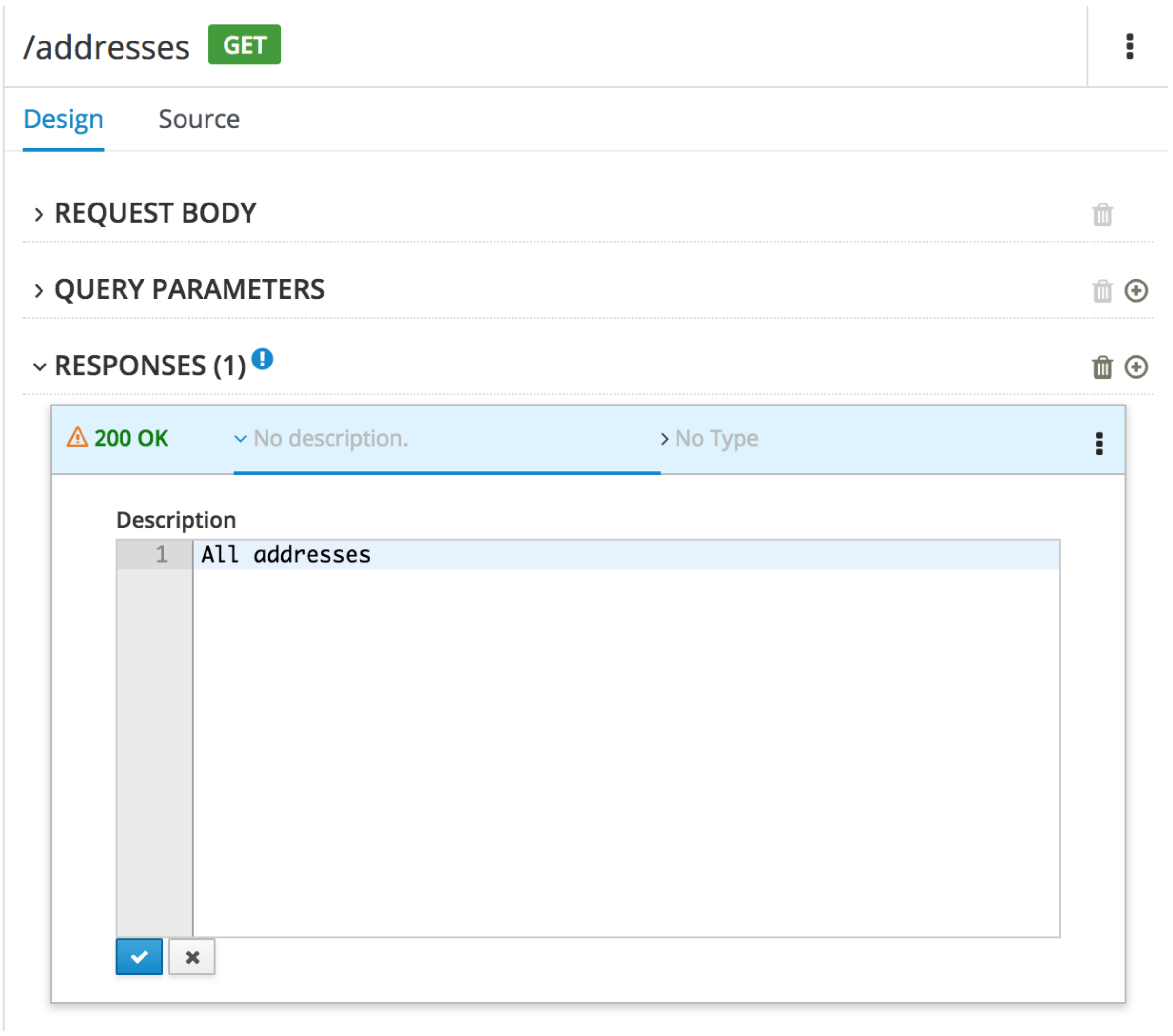
- **/addresses** path:
  - GET - Lists all addresses.
  - POST - Adds an address.
- **/addresses/{addressId}** path:

- GET - Gets an address.
- PUT - Updates an address.
- DELETE - Deletes an address.

10. For each operation define a response.
    For the AddressBook API example:

    - To define a response for the /**addresses** GET operation:
        a. Click the green **GET** button.
        b. Click **Add a response**.
        c. For **Response Status Code**, select **200** and then click **Add**.
    - To define a response for the /**addresses** POST operation:
        a. Click the **POST** button.
        b. For **Request Body Type**, select **Address**.
        c. For **Response**, click **Add a response**.
        d. For **Response Status Code**, select **201 Created** and then click **Add**.

11. For each response, provide a description.
    For example:



```
All addresses
```

12. Resolve any issues, as described in Section 2.2, "Resolving issues in Apicurito" .

13. Click **Save As → Save as JSON**.
    The JSON file is downloaded to your local download folder. The default filename is **openapi-spec**.

Additional resources

- For information about the OpenAPI 2.0 Specification, go to: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md

2.2. RESOLVING ISSUES IN APICURITO

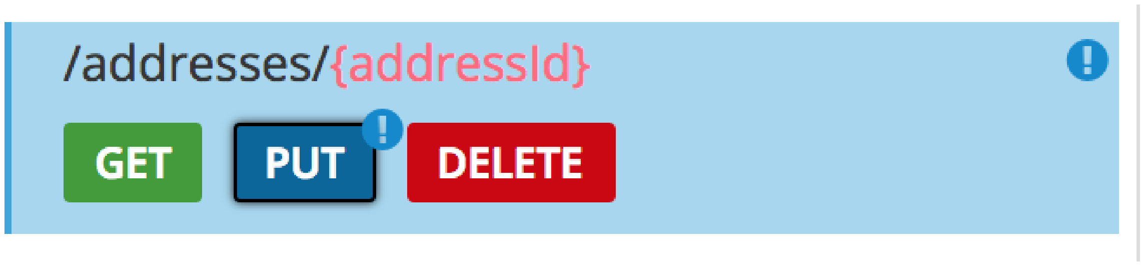
When you create and edit an API, Apicurito identifies issues that you must resolve with an exclamation (!) icon.

Prerequisites

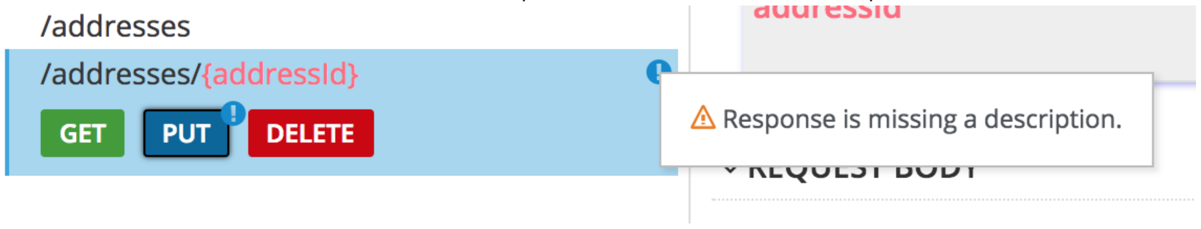
- Open an API in Apicurito.

Procedure

1. Find an issue indicated by an exclamation (!) icon. For example:

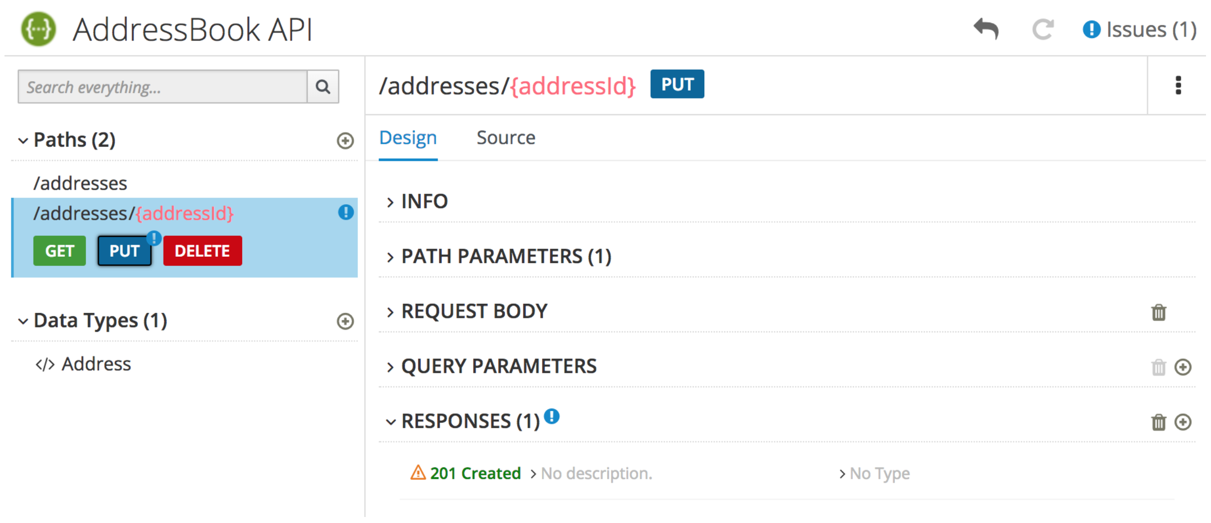


2. Click the exclamation icon to view a description of the issue. For example:

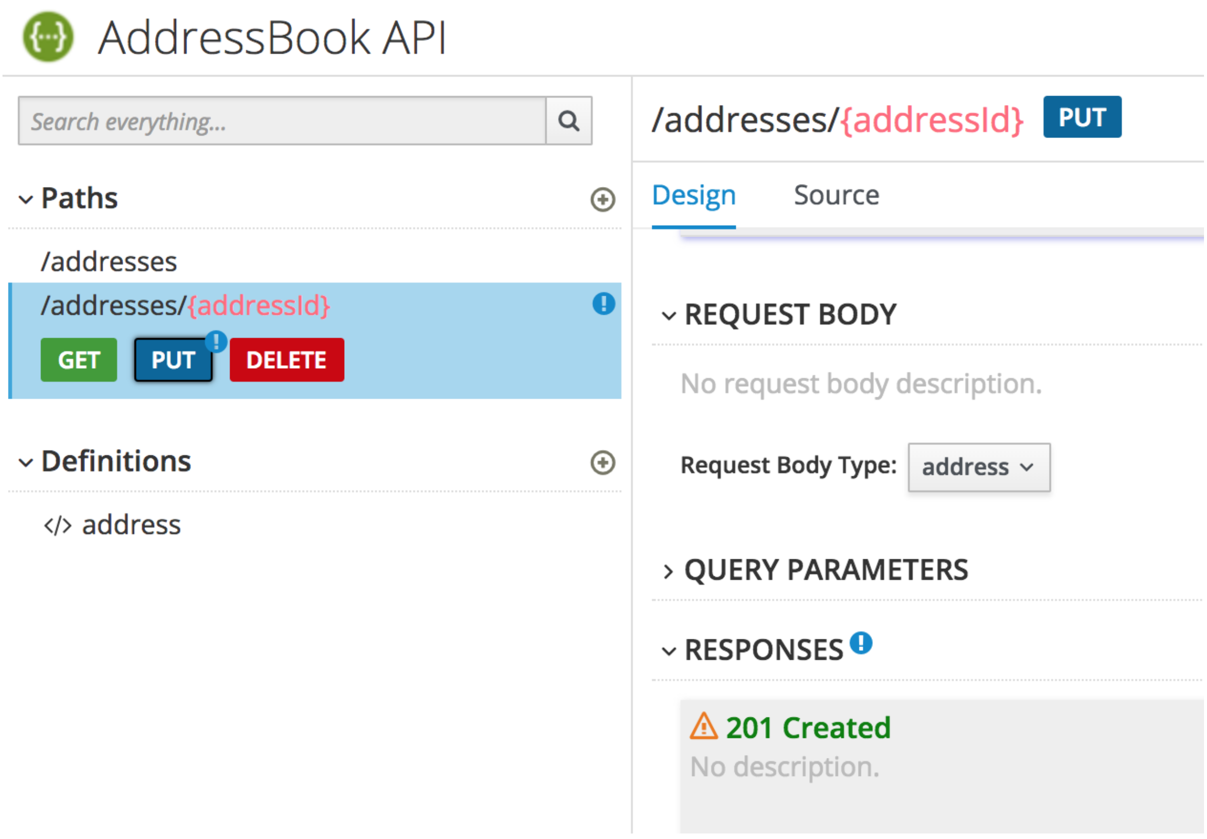


Response is missing a description.

3. Based on the information provided by the issue description, navigate to the location of the issue and fix it.
   For example, open the PUT operation and then add a description for the response.

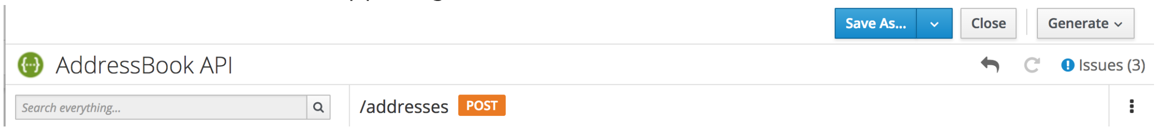


After you add a description, the issue is resolved and the exclamation icon disappears:

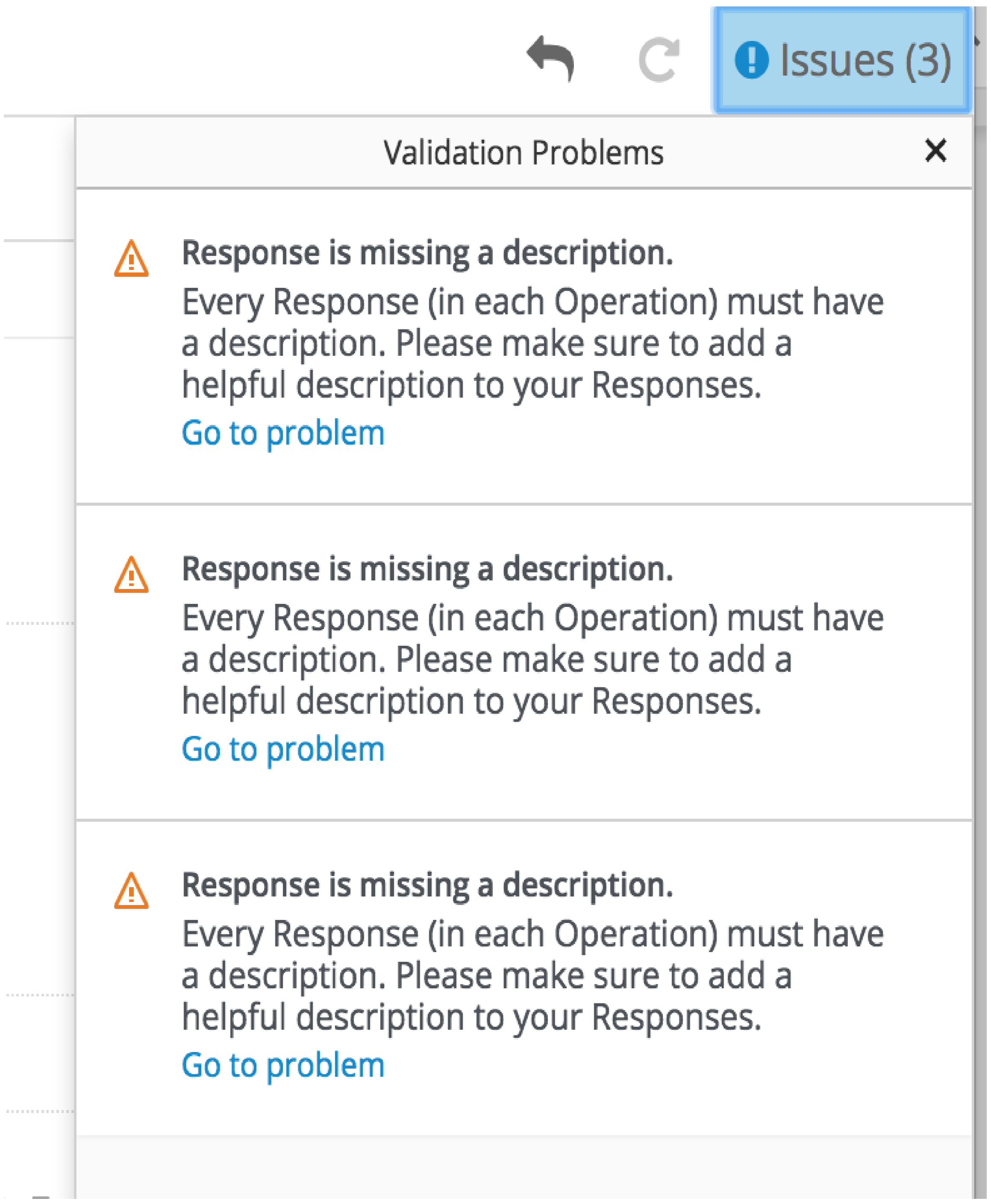


4. For a summary of all issues:

   a. Click the **Issues** link in the upper right corner.



   b. Click **Go to a problem** for a specific issue to go to the location of the issue so that you can resolve it.



Validation Problems

Response is missing a description.
Every Response (in each Operation) must have a description. Please make sure to add a helpful description to your Responses.
Go to problem

Response is missing a description.
Every Response (in each Operation) must have a description. Please make sure to add a helpful description to your Responses.
Go to problem

Response is missing a description.
Every Response (in each Operation) must have a description. Please make sure to add a helpful description to your Responses.
Go to problem

CHAPTER 3. IMPLEMENT, BUILD, AND DEPLOY A FUSE APPLICATION BASED ON A REST API

You can use Red Hat Fuse Apicurito to generate a Camel Fuse project based on a REST API definition. In your Fuse development environment, you can complete the Camel routes and Rest DSL API. Finally, you can build the project and deploy the resulting application to Fuse on OpenShift.

Prerequisites

- You have an existing API definition, which complies with the OpenAPI 2.0 specification. For example, an **openapi-spec.json** file that you created with Apicurito.
- Apicurito is installed and running on your local OpenShift cluster.
- You have an existing OpenShift project with Apicurito added as a service.
- You have installed Maven and Red Hat Fuse.

The following topics describe how to implement, build, and deploy a Fuse application based on a REST API:

- Section 3.1, "Uploading an API definition to Apicurito"
- Section 3.2, "Generating a Fuse Camel project from Apicurito"
- Section 3.3, "Completing the Apicurito-generated Camel project"
- Section 3.4, "Building and deploying a REST service"

3.1. UPLOADING AN API DEFINITION TO APICURITO

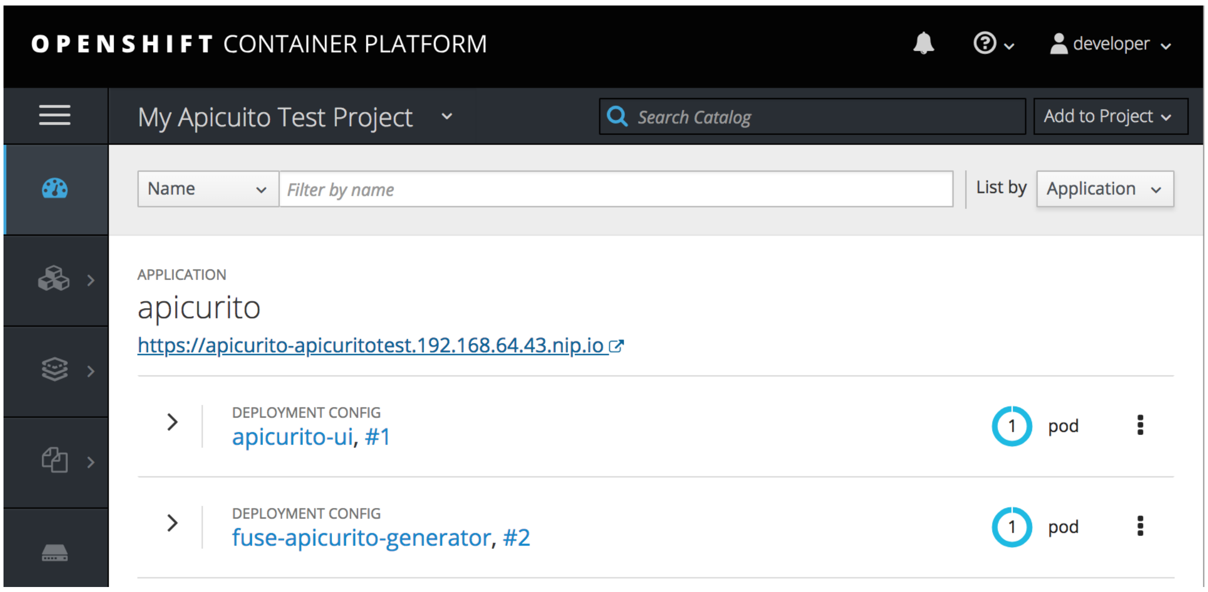
You can upload an existing API definition to Apicurito.

Prerequisites

- You have an existing API definition, which complies with the OpenAPI 2.0 specification. For example, an **openapi.json** file that you created with Apicurito.
- Apicurito is installed and running on your local OpenShift cluster.
- You have an existing OpenShift project with Apicurito added as an application.

Procedure

1. In your OpenShift web console, open the project that contains Apicurito.
2. Open the Apicurito console. In the list of applications for the project, click the URL under apicurito. For example: **https://apicurito-myproject.192.168.64.38.nip.io**

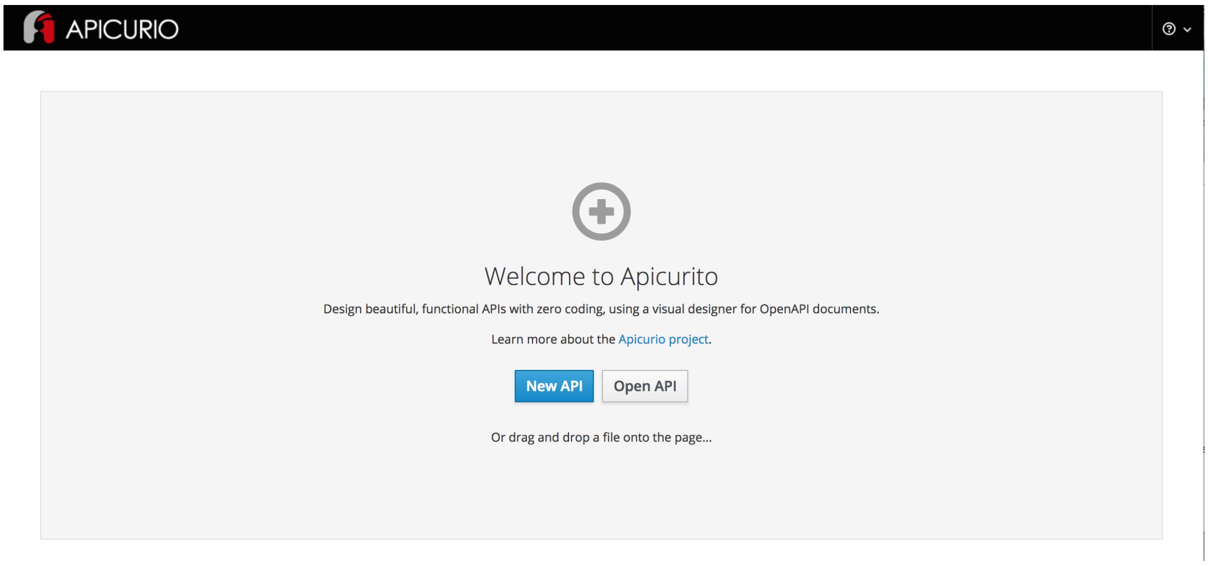


APPLICATION
apicurito
https://apicurito-apicuritotest.192.168.64.43.nip.io

DEPLOYMENT CONFIG
apicurito-ui, #1
1 pod

DEPLOYMENT CONFIG
fuse-apicurito-generator, #2
1 pod

The Apicurito console opens in a separate web browser tab or window:



Welcome to Apicurito

Design beautiful, functional APIs with zero coding, using a visual designer for OpenAPI documents.

Learn more about the Apicurio project.

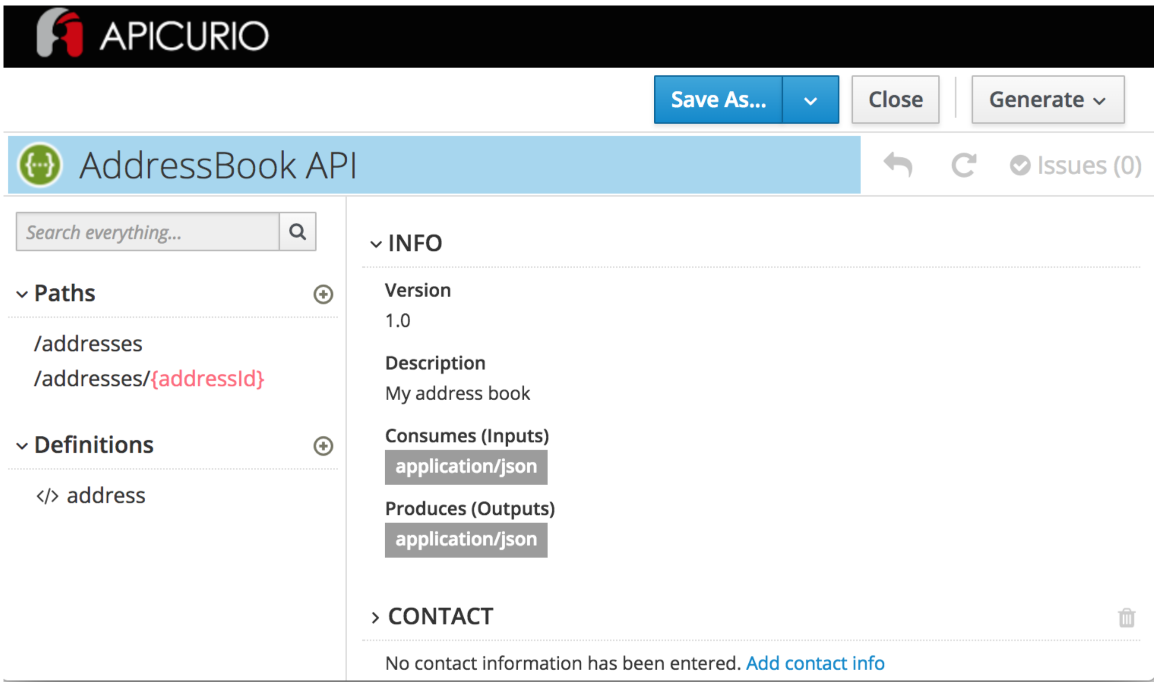
Or drag and drop a file onto the page...

3. Click **Open API**.
   A file manager window opens.

4. In the file manager window:

   a. Navigate to the folder that contains the existing OpenAPI definition file, for example, **openapi.json**.

   b. Select the OpenAPI definition file and then click **Open**.
      The OpenAPI definition opens in the Apicurito console. For example:



3.2. GENERATING A FUSE CAMEL PROJECT FROM APICURITO

You can use Apicurito to generate a Fuse Camel project based on an API definition.

Prerequisites

- Apicurito is installed and running on your local OpenShift cluster.
- You have an existing OpenShift project with Apicurito added as an application.
- You have created or opened an API definition file in the Apicurito console.

Procedure

In the Apicurito console:

1. Click **Generate**.
2. Select **Fuse Camel Project** from the drop-down list.

Apicurito generates a **camel-project.zip** file and downloads it to your local default download folder.

The zip file contains a Fuse Camel project that provides a default skeleton implementation of the API definition using Camel's Rest DSL and includes all resource operations. The project also includes the original OpenAPI definition file that you used to generate the project.

3.3. COMPLETING THE APICURITO-GENERATED CAMEL PROJECT

Apicurito generates a Fuse project that provides a default skeleton implementation of the API definition using Camel's Rest DSL and covering all resource operations. In your Fuse development environment, you complete the project.

Prerequisites

- You have a **camel-project.zip** file generated by Apicurito.

- (Optional) You have installed Red Hat Developer Studio with Fuse Tooling.

Procedure

1. Unzip the Apicurito-generated **camel-project.zip** file to a temporary folder.
2. Open Red Hat Developer Studio.
3. In Developer Studio, select **File → Import**.
4. In the **Import** dialog, select **Maven → Existing Maven Projects**.
5. Open the project's **camel-context.xml** file in the editor view.
6. Click the **REST** tab to edit the Rest DSL components.
   For information on defining REST services, see the "Defining REST services" section of the Apache Camel Development Guide .

   For information on extending JAX-RS endpoints with Swagger support, see the Apache CXF Development Guide.

   For information on using the Fuse Tooling REST editor, see the "Viewing and editing Rest DSL components" section of the Tooling User Guide .
7. In the Design tab, edit the Camel routes.
   For information on editing Camel routes, see the "Editing a routing context in the route editor" section of the Tooling User Guide .

3.4. BUILDING AND DEPLOYING A REST SERVICE

After you complete the Fuse project, you can build and deploy the project in OpenShift.

Prerequisites

- You have a complete, error-free Fuse project that defines a REST service.
- You have installed Java 8 JDK (or later) and Maven 3.3.x (or later).

Procedure

If you have a single-node OpenShift cluster, such as Minishift or the Red Hat Container Development Kit, installed and running, you can deploy your project there.

To deploy this project to a running single-node OpenShift cluster:

1. Log in to your OpenShift cluster:

   ```
   $ oc login -u developer -p developer
   ```

2. Create a new OpenShift project for the project. For example, the following command creates a new project named **test-deploy**.

   ```
   $ oc new-project test-deploy
   ```

3. Change the directory to the folder that contains your Fuse Camel project (for example, **myworkspace/camel-project**) :

   ```
   $ cd myworkspace/camel-project
   ```

4. Build and deploy the project to the OpenShift cluster:

   ```
   $ mvn clean fabric8:deploy -Popenshift
   ```

5. In your browser, open the OpenShift console and navigate to the project (for example, **test-deploy**). Wait until you can see that the pod for the **camel-project** application has started.

6. On the project's **Overview** page, locate the URL for the **camel-project** application. The URL uses this form: **http://camel-project-MY_PROJECT_NAME.OPENSHIFT_IP_ADDR.nip.io**.

7. Click the URL to access the service.

CHAPTER 4. DISCOVER AN API SERVICE IN 3SCALE

Red Hat 3scale API Management is an offering from Red Hat that enables you to regulate access to API services on the public Internet. The functionality of 3scale includes the ability to enforce service-level agreements (SLAs), manage API versions, provide security and authentication services and more. Fuse supports a *service discovery* feature for 3scale, which makes it easy to discover Fuse services from the 3scale Admin Portal UI. Using service discovery, you can scan for Fuse applications running in the same OpenShift cluster and automatically import the associated API definitions into 3scale.

Prerequisites

- A Fuse application that provides an API service is deployed and running in OpenShift.
- The Fuse application is annotated with the requisite annotations to make it discoverable by 3scale.

  NOTE

  Fuse projects that are generated by Apicurito are pre-configured to automatically provide the requisite annotations.

  For Fuse projects that are not generated by Apicurito, you must configure your project as described in Section 4.1, “Adding annotations for Fuse projects that are not generated by Apicurito”.

- The 3scale API Management system is deployed on the *same* OpenShift cluster as the API service that is to be discovered.

For details of the procedure to discover an API service in 3scale, see Using Service Discovery in the Red Hat 3scale API Management *Service Discovery* guide.

Additional resources

- Red Hat 3scale API Management product page .
- Red Hat 3scale API Management documentation .

4.1. ADDING ANNOTATIONS FOR FUSE PROJECTS THAT ARE NOT GENERATED BY APICURITO

In order for 3scale to discover an API service, the Fuse application that provides the API service must include Kubernetes Service Annotations that make it discoverable. These annotations are provided by the Fabric8 service discovery enricher which is part of the Fabric8 Maven Plugin.

For Apache Camel Rest DSL projects, the Fabric8 Maven Plugin runs the Fabric8 service discovery enricher by default.

Fuse projects that are generated by Apicurito are pre-configured to automatically provide the required annotations.

Procedure

For a Fuse Rest DSL project that is not generated by Apicurito, configure the project as follows:

1. Edit the Fuse project's **pom.xml** file to include the **fabric8-maven-plugin** dependency, as shown in this example:

   ```
   <plugin>
         <groupId>org.jboss.redhat-fuse</groupId>
         <artifactId>fabric8-maven-plugin</artifactId>
         <version>${fuse.version}</version>
         <executions>
           <execution>
             <goals>
               <goal>resource</goal>
               <goal>build</goal>
             </goals>
           </execution>
         </executions>
     </plugin>
   ```

   The Fabric8 Maven Plugin runs the Fabric8 service discovery enricher if certain project-level conditions are met (for example, the project must be a Camel Rest DSL project). You do not need to specify the Fabric8 service discovery enricher as a dependency in the **pom.xml** file, unless you want to customize the enricher's behavior (as described in Section 4.2, "Customizing the API service annotation values".)

2. In the Fuse Rest DSL project's **camel-context.xml** file, specify the following attributes in the **restConfiguration** element:

   - **scheme**: The scheme part of the URL where the service is hosted. You can specify "http" or "https".
   - **contextPath**: The path part of the URL where the API service is hosted.
   - **apiContextPath**: The path to the location where the API service description document is hosted. You can specify either a relative path if the document is self-hosted or a full URL if the document is hosted externally.
     The following excerpt from an example **camel-context.xml** file shows annotation attribute values in the **restConfiguration** element:

     ```
     <?xml version="1.0" encoding="UTF-8"?>
     <beans xmlns="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="
         http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
         http://camel.apache.org/schema/spring
     http://camel.apache.org/schema/spring/camel-spring.xsd">

       <camelContext xmlns="http://camel.apache.org/schema/spring">
         <restConfiguration component="servlet" scheme="https"
             contextPath="myapi" apiContextPath="myapi/openapi.json"/>
     ...
     ```

The enricher uses the information provided by these **restConfiguration** element attribute values to create values for the **discovery.3scale.net/scheme**, **discovery.3scale.net/path**, and the **discovery.3scale.net/description-path** annotations, thereby making the project's deployed OpenShift service discoverable by 3scale as described in the Making a service discoverable section of the Red Hat 3scale API Management *Service Discovery* guide.

The enricher adds the following label and annotations to make the service discoverable by 3scale:

- The **discovery.3scale.net** label: By default, the enricher sets this value to “true”. 3scale uses this label when it executes the selector definition to find all services that need discovery.
- The following annotations:
  - **discovery.3scale.net/discovery-version**: (optional) The version of the 3scale discovery process. The enricher sets this value to "v1" by default.
  - **discovery.3scale.net/scheme**: The scheme part of the URL where the service is hosted. The enricher uses the default "http" unless you override it in the **restConfiguration** element’s **scheme** attribute. The other possible value is "https".
  - **discovery.3scale.net/path**: The path part of the URL where the service is hosted. This annotation is omitted when the path is at root, "/". The enricher gets this value from the **restConfiguration** element’s **path** attribute.
  - **discovery.3scale.net/port**: The port of the service. The enricher obtains this value from the Kubernetes service definition, which contains the the port numbers of the services it exposes. If the Kubernetes service definition exposes more than one service, the enricher uses the first port listed.
  - **discovery.3scale.net/description-path**: (optional) The path to the OpenAPI service description document. The enricher gets this value from the **restConfiguration** element’s **contextPath** attribute.

You can customize the behavior of the Fabric8 service discovery enricher, as described in Section 4.2, “Customizing the API service annotation values”.

4.2. CUSTOMIZING THE API SERVICE ANNOTATION VALUES

The Maven Fabric8 Plugin runs the Fabric8 service discovery enricher by default. The enricher adds annotations to the Fuse Rest DSL project’s API service so that the API service is discoverable by 3scale, as described in Using Service Discovery in the Red Hat 3scale API Management *Service Discovery* guide.

The enricher uses default values for some annotations and obtains values for other annotations from the project’s **camel-context.xml** file.

You can override the default values and the values defined in the **camel-context.xml** file by defining values in the Fuse project **pom.xml** file or in a **service.yml** file. (If you define values in both files, the enricher uses the values from the **service.yml** file.) See Section 4.3, “Fabric8 service discovery enricher elements” for a description of the elements that you can specify for the Fabric8 service discovery enricher.

Procedure

To specify annotation values in the Fuse project **pom.xml** file:

1. Open your Fuse project’s **pom.xml** file in an editor of your choice.
2. Locate the **fabric8-maven-plugin** dependency, as shown in this example:

```
<plugin>
      <groupId>org.jboss.redhat-fuse</groupId>
      <artifactId>fabric8-maven-plugin</artifactId>
      <version>${fuse.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>resource</goal>
            <goal>build</goal>
          </goals>
        </execution>
      </executions>
</plugin>
```

3. Add the Fabric8 service discovery enricher as a dependency to the Fabric8-Maven plugin as shown in the following example.

```
<plugin>
  <groupId>org.jboss.redhat-fuse</groupId>
  <artifactId>fabric8-maven-plugin</artifactId>
  <version>${fuse.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>resource</goal>
        <goal>build</goal>
      </goals>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>io.acme</groupId>
      <artifactId>myenricher</artifactId>
      <version>1.0</version>
      <configuration>
        <enricher>
          <config>
            <f8-service-discovery>
              <scheme>https</scheme>
              <path>/api</path>
              <descriptionPath>/api/openapi.json</descriptionPath>
            </f8-service-discovery>
          </config>
        </enricher>
      </configuration>
      </dependency>
  </dependencies>
</plugin>
```

4. Save your changes.

Alternatively, you can use a **src/main/fabric8/service.yml** fragment to override the annotation values, as shown in the following example:

```
kind: Service
name:
metadata:
  labels:
    discovery.3scale.net/discoverable : "true"
  annotations:
    discovery.3scale.net/discovery-version : "v1"
    discovery.3scale.net/scheme : "https"
    discovery.3scale.net/path : "/api"
    discovery.3scale.net/port : "443"
    discovery.3scale.net/description-path : "/api/openapi.json"
spec:
  type: LoadBalancer
```

4.3. FABRIC8 SERVICE DISCOVERY ENRICHER ELEMENTS

The following table describes the elements that you can specify for the Fabric8 service discovery enricher, if you want to override the default values and the values defined in the **camel-context.xml** file.

You can define these values in the Fuse Rest DSL project's **pom.xml** file or in a **src/main/fabric8/service.yml** file. (If you define values in both files, the enricher uses the values from the **service.yml** file.) See Section 4.2, "Customizing the API service annotation values" for examples.

Table 4.1. Fabric8 service discovery enricher elements

Element	Description	Default
springDir	The path to the spring configuration directory that contains the **camel-context.xml** file.	The **/src/main/resources/spring** path which is used to recognize a Camel Rest DSL project.
scheme	The scheme part of the URL where the service is hosted. You can specify "http" or "https".	http
path	The path part of the URL where the API service is hosted.	
port	The port part of the URL where the API service is hosted.	80
descriptionPath	The path to a location where the API service description document is hosted. You can specify either a relative path if the document is self-hosted or a full URL if the document is hosted externally.	
discoveryVersion	The version of the 3scale discovery implementation.	v1

Element	Description	Default
discoverable	The element that sets the **discovery.3scale.net** label to either **true** or **false**. If set to **true**, 3scale will try to discover this service. If set to **false**, 3scale will not try to discover this service. You can use this element as a switch, to temporary turn off 3scale discovery integration by setting it to "false".	If you do not specify a value, the enricher tries to auto-detect whether it can make the service discoverable. If the enricher determines that it cannot make the service discoverable, 3scale will not try to discover this service.