



Red Hat Fuse 7.5

Designing APIs

Designing REST APIs for Fuse applications on OpenShift

Red Hat Fuse 7.5 Designing APIs

Designing REST APIs for Fuse applications on OpenShift

Legal Notice

Copyright © 2020 Red Hat, Inc.

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

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

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

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

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

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

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

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

All other trademarks are the property of their respective owners.

Abstract

Guide to using the web-based REST API Designer

Table of Contents

CHAPTER 1. OVERVIEW	3
1.1. ADD API DESIGNER AS A SERVICE TO YOUR OPENSIFT PROJECT	3
CHAPTER 2. DESIGN AND DEVELOP AN API DEFINITION WITH API DESIGNER	5
2.1. CREATING A REST API DEFINITION	5
2.2. RESOLVING ISSUES IN API DESIGNER	9
CHAPTER 3. IMPLEMENT, BUILD, AND DEPLOY A FUSE APPLICATION BASED ON A REST API	12
3.1. UPLOADING AN API DEFINITION TO API DESIGNER	12
3.2. GENERATING A FUSE CAMEL PROJECT FROM API DESIGNER	13
3.3. COMPLETING THE API DESIGNER-GENERATED CAMEL PROJECT	13
3.4. BUILDING AND DEPLOYING A REST SERVICE	14
CHAPTER 4. PREPARE AN API SERVICE FOR 3SCALE DISCOVERY	16
4.1. ADDING ANNOTATIONS FOR FUSE PROJECTS THAT ARE NOT GENERATED BY API DESIGNER	16
4.2. CUSTOMIZING THE API SERVICE ANNOTATION VALUES	18
4.3. FABRIC8 SERVICE DISCOVERY ENRICHER ELEMENTS	20

CHAPTER 1. OVERVIEW

Red Hat Fuse on OpenShift provides API Designer, a web-based API editor, 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. API Designer is a “light” version of the Apicurio Studio open source project (<https://www.apicur.io/>). This means that your API Designer sessions are stateless and you must save your API definition as a JSON file at the end of each session.

You can also use API Designer 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 API Designer to incorporate REST APIs in your Fuse on OpenShift application solution:

1. Add API Designer as a service to your OpenShift project.
2. In API Designer:
 - Create an API definition with API Designer. 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 API Designer.
 - Generate a Fuse Camel project based on the current REST API definition. API Designer 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.

1.1. ADD API DESIGNER AS A SERVICE TO YOUR OPENSHIFT PROJECT

You can add API Designer 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 API Designer by following the guidelines recommended by your OpenShift system administrator.
- Verify that the Fuse on OpenShift images and templates, including **apidesigner-ui** and **fuse-apidesigner-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 API Designer 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 **API Designer** and then select **Red Hat Fuse API Designer**.
The **Information** step of the Red Hat Fuse API Designer wizard opens.
5. Click **Next**.
The **Configuration** step of the Red Hat Fuse API Designer wizard opens.
6. In the **Image Stream Namespace** field, type **openshift**.
7. In the **ROUTE_HOSTNAME** field, type the external hostname that allows you to access the API Designer instance, for example **apidesigner-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.
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 **apidesigner-ui** application.
11. Click the arrow to the left of the **apidesigner-ui** deployment to expand and view the deployment details.
12. Click the link for the API Designer instance, for example **https://apidesigner-myproject.192.168.64.43.nip.io**.
API Designer opens in a new web browser window or tab.



NOTE

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

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


CHAPTER 2. DESIGN AND DEVELOP AN API DEFINITION WITH API DESIGNER

You can use API Designer 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 API Designer service to your OpenShift project.

2.1. CREATING A REST API DEFINITION

The following steps describe how to create a REST API definition.



NOTE

- You can access the API Designer user interface from Fuse Online or Fuse on OpenShift.
- For Fuse on OpenShift only, API Designer is stateless which means that it does not save your work between OpenShift sessions. You need to save the API to your local file system between sessions.

About the example

The Task Management API example simulates a simple API that sales consultants might use to track the tasks that they need to do when interacting with customer contacts. Example "to-do" tasks might be "create an account for a new contact" or "place an order for an existing contact". To implement the Task Management API example, you create two paths - one for tasks and one for a specific task. You then define operations to create a task, retrieve all tasks or a specific task, update a task, and delete a task.

Prerequisites


- You know the endpoints for the API that you want to create. For the Task Management API example, there are two endpoints: **/todo** and **/todo/{id}**.
- For Fuse on OpenShift only, you created an OpenShift project and you added the API Designer service to your OpenShift project.

Procedure

1. If you are using Fuse Online, skip to step 2.
If you are using Fuse on OpenShift:
 - a. Log in to your OpenShift web console and then open the project that contains API Designer.
 - b. In the list of applications, click the URL for API Designer, for example **https://apidesigner-myproject.192.168.64.43.nip.io**
A new browser window or tab opens for API Designer.

**NOTE**

Because API Designer is a “light” version of the [Apicurio Studio open source project](#), “Apicurio” shows in the API Designer interface.

2. Click **New API**. A new API page opens.
3. 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 **Task API**.
 - c. Confirm the name change.
4. Optionally:
 - Add your contact information (name, email address, and URL).
 - Select a license.
 - Define tags.
 - Define a security scheme.
 - Specify security requirements.
5. Define a relative path to each individual endpoint of the API. The field name must begin with a slash (/).
For the Task Management API example, create two paths:

- A path for tasks: **/todo**
- A path for a specific task by ID: **/todo/{id}**

The screenshot shows the API Designer interface for 'Task API'. On the left, there is a search bar and a list of paths. The path '/todo/{id}' is selected. On the right, the details for this path are shown, including the path parameters section. The parameter 'id' is listed with a description 'No description.' and a type 'No Type'. A 'Create' button is visible next to the parameter.

6. Specify the type of any path parameters.
For the example **id** parameter:
 - a. In the **Paths** list, click **/todo/{id}**.
The **id** parameter appears in the **PATH PARAMETERS** section.
 - b. Click **Create**.
 - c. For the description, type: **The ID of the task to find**.
 - d. For the type, select **integer** as **32-Bit integer**.

The screenshot shows the API Designer interface. On the left, there is a sidebar with two sections: 'Paths (2)' and 'Data Types'. Under 'Paths (2)', the path `/todo/{id}` is selected. Under 'Data Types', a message states 'No reusable types have been created. Add a data type'. The main area is titled 'Design' and shows 'PATH PARAMETERS (1)'. A parameter named 'id' is defined with the description 'The ID of the task to find.' and the type 'integer as int32'. Below this, there are dropdown menus for 'Type: Integer' and 'as 32-Bit Integer'.

7. 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 the Task Management API example, type **Todo**.
- c. Optionally, you can provide an example from which API Designer creates the data type's schema. You can then edit the generated schema.
For the Task Management API example, start with the following JSON example:

```
{
  "id": 1,
  "task": "my task",
  "completed": false
}
```

- d. Optionally, you can choose to create a REST Resource with the data type.
- e. Click **Save**. If you provided an example, API Designer generates a schema from the example:

The screenshot shows the API Designer interface for a resource named 'Task API'. The main area is titled 'Design' and shows 'PROPERTIES (3)'. The properties are: 'completed' (boolean), 'id' (integer as int32), and 'task' (string). Below the properties, there is an 'EXAMPLE' section containing the JSON schema:

```
{
  "id": 1,
  "task": "my task",
  "completed": false
}
```

. The sidebar on the left shows 'Paths (2)' with `/todo/{id}` and 'Data Types (1)' with `</> Todo` selected.

8. Optionally, you can add edit the schema properties and add new ones.

9. For the Task Management API example, create another data type named **Task** with one property named **task** of type **string**.

The screenshot shows the Red Hat Fuse API Designer interface for a 'Task API'. The interface includes a search bar at the top left, a left sidebar with 'Paths (2)' and 'Data Types (2)', and a main panel for the 'Task' resource. The 'Task' resource is selected, showing its 'INFO' section with a description 'No description.' and a 'PROPERTIES (1)' section with a property 'task' of type 'string'.

10. For each path, define operations (GET, PUT, POST, DELETE, OPTIONS, HEAD, or PATCH). For the Task Management API example, define the operations as described in the following table:

Table 2.1. Task Management API operations

Path	Operation	Description	Request	Response
/todo	POST	Create a new task.	Request Body type: Task	Status Code: 201 Description: Task created Response Body: Todo type
/todo	GET	Get all tasks.	<i>Not applicable</i>	<ul style="list-style-type: none"> Status Code: 200 Description: Task deleted Status Code: 400 Description: Task not deleted
/todo/{id}	GET	Get a task by ID.	<i>Not applicable</i>	Status Code: 200 Description: Task found for ID Response Body: Todo type

Path	Operation	Description	Request	Response
/todo/{id}	UPDATE	Update a task by ID.	Request Body type: Task	<ul style="list-style-type: none"> ● Status Code: 200 Description: Completed ● Status Code: 400 Description: Task not updated
/todo/{id}	DELETE	Delete a task by ID.	<i>Not applicable</i>	<ul style="list-style-type: none"> ● Status Code: 200 Description: Task deleted ● Status Code: 400 Description: Task not deleted

11. Resolve any issues, as described in [Section 2.2, "Resolving issues in API Designer"](#) .
12. For Fuse on OpenShift only, save your API specification by clicking **Save As** and then select **JSON** or **YAML** format.
The JSON or YAML file is downloaded to your local download folder. The default filename is **openapi-spec** with the appropriate file extension.

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 API DESIGNER

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

Prerequisites

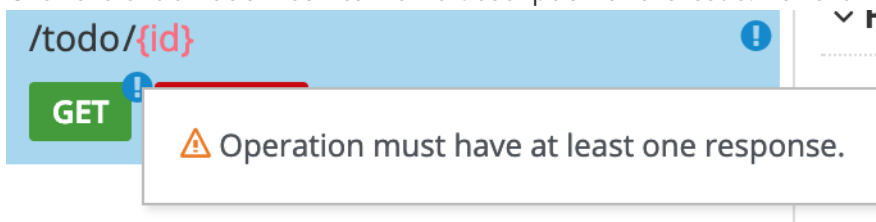
- Open an API in API Designer.

Procedure

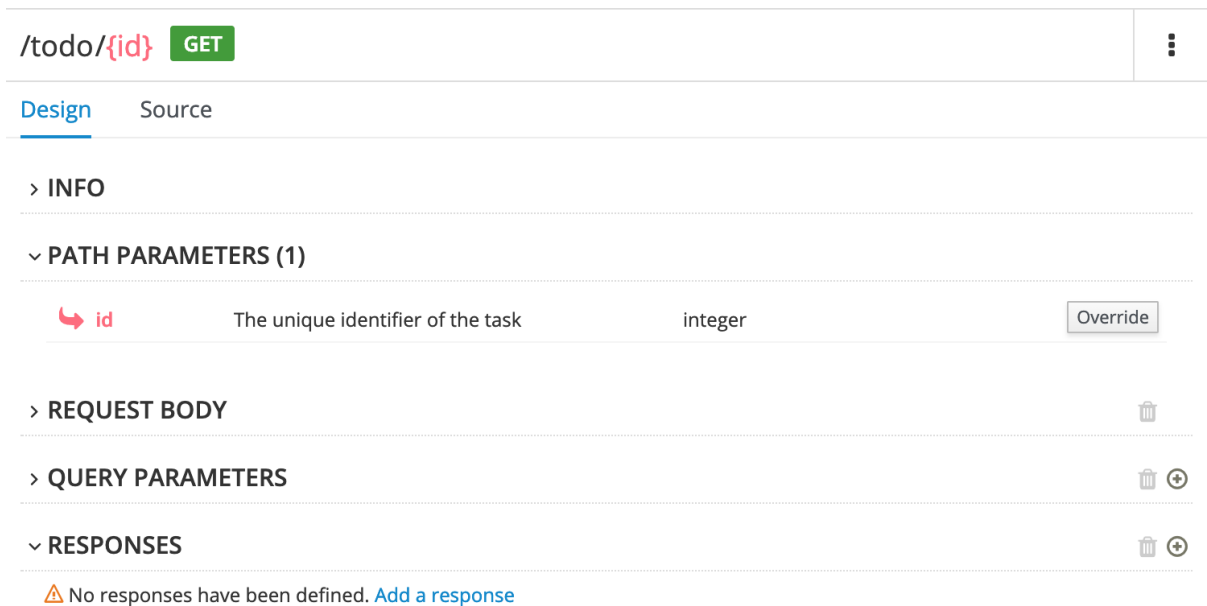
1. Find an issue indicated by an exclamation (!) icon. For example:



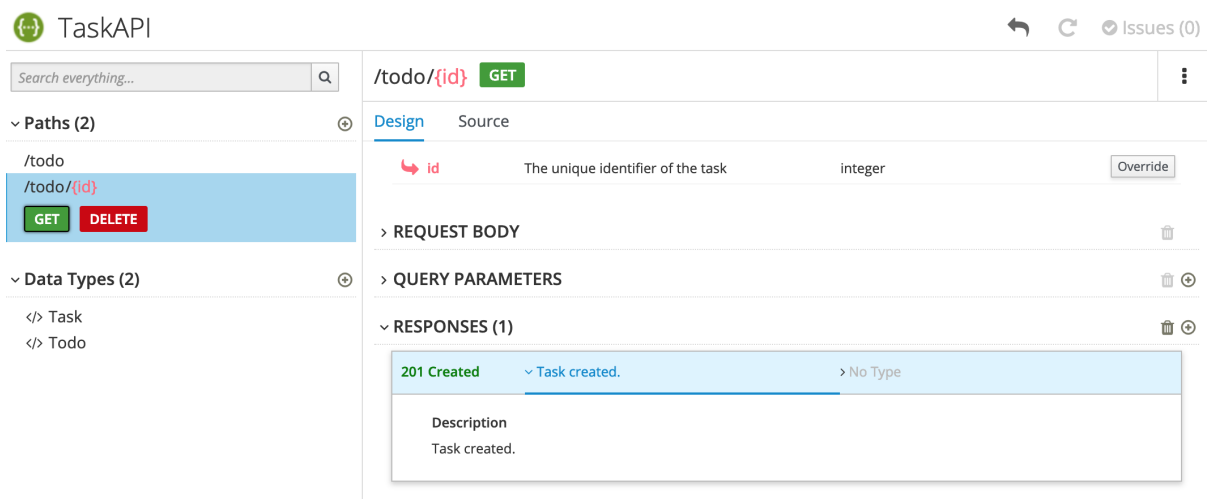
- Click the exclamation icon to view a description of the issue. For example:



- Based on the information provided by the issue description, navigate to the location of the issue and fix it.
For example, to fix the "Operation must have at least one response" issue, click the **GET** operation to open it and then click **Add a response**.



After you type a description for the response, 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.

A screenshot of the "Validation Problems" panel in the API Designer. The panel has a title bar with "Validation Problems" and a close button (X). Below the title bar, there are three error messages, each with an orange warning icon. The first two messages are identical: "Operation must have at least one response. When declaring an Operation (e.g. GET, PUT, POST, etc...) at least one Response MUST be included. Typically at least a 20x (success) response should be defined. Go to problem". The third message is: "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".

! Issues (3)

Validation Problems ×

⚠ Operation must have at least one response.
When declaring an Operation (e.g. GET, PUT, POST, etc...) at least one Response MUST be included. Typically at least a 20x (success) response should be defined.
[Go to problem](#)

⚠ Operation must have at least one response.
When declaring an Operation (e.g. GET, PUT, POST, etc...) at least one Response MUST be included. Typically at least a 20x (success) response should be defined.
[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 API Designer 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 API Designer.
- API Designer is installed and running on your local OpenShift cluster.
- You have an existing OpenShift project with API Designer 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 API Designer"](#)
- [Section 3.2, "Generating a Fuse Camel project from API Designer"](#)
- [Section 3.3, "Completing the API Designer-generated Camel project"](#)
- [Section 3.4, "Building and deploying a REST service"](#)

3.1. UPLOADING AN API DEFINITION TO API DESIGNER

You can upload an existing API definition to API Designer.

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 API Designer.
- API Designer is installed and running on your local OpenShift cluster.
- You have an existing OpenShift project with API Designer added as an application.

Procedure

1. In your OpenShift web console, open the project that contains API Designer.
2. Open the API Designer console. In the list of applications for the project, click the URL under apidesigner. For example: <https://apidesigner-myproject.192.168.64.38.nip.io>
The API Designer console opens in a separate web browser tab or window.
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

- 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 API Designer console.

3.2. GENERATING A FUSE CAMEL PROJECT FROM API DESIGNER

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

Prerequisites

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

Procedure

In the API Designer console:

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

API Designer 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 API DESIGNER-GENERATED CAMEL PROJECT

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

Procedure

1. Unzip the API Designer-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.OPENSIFT_IP_ADDR.nip.io.

7. Click the URL to access the service.

CHAPTER 4. PREPARE AN API SERVICE FOR 3SCALE DISCOVERY

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 API Designer are pre-configured to automatically provide the requisite annotations.

For Fuse projects that are not generated by API Designer, you must configure your project as described in [Section 4.1, “Adding annotations for Fuse projects that are not generated by API Designer”](#).

- 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 the service discovery section of the [Red Hat 3scale API Management Admin Portal 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 API DESIGNER

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 API Designer are pre-configured to automatically provide the required annotations.

Procedure

For a Fuse Rest DSL project that is not generated by API Designer, 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 see the service discovery section of the [Red Hat 3scale API Management Admin Portal 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 *Admin Portal Guide* 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	<p>The element that sets the discovery.3scale.net label to either true or false.</p> <p>If set to true, 3scale will try to discover this service.</p> <p>If set to false, 3scale will not try to discover this service.</p> <p>You can use this element as a switch, to temporary turn off 3scale discovery integration by setting it to "false".</p>	<p>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.</p>