



# Red Hat Integration 2020-Q2

## Developing and Deploying API Provider Integrations

Developing and Deploying API Provider Integrations



# Red Hat Integration 2020-Q2 Developing and Deploying API Provider Integrations

---

Developing and Deploying API Provider Integrations

## 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

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

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> 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<sup>®</sup> 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

Define and deploy API Provider integrations with Red Hat Fuse Online, API Designer, and 3scale API Management.

---

## Table of Contents

|   |           |
|---|-----------|
| <b>PREFACE</b> .....  | <b>3</b>  |
| <b>CHAPTER 1. OVERVIEW OF DEVELOPING AND DEPLOYING API PROVIDER INTEGRATIONS</b> .....              | <b>4</b>  |
| 1.1. BENEFIT, OVERVIEW, AND WORKFLOW FOR CREATING API PROVIDER INTEGRATIONS                         | 5         |
| 1.2. HOW OPENAPI OPERATIONS RELATE TO API PROVIDER INTEGRATION FLOWS                                | 7         |
| <b>CHAPTER 2. CONFIGURING RED HAT INTEGRATION PRODUCTS FOR API DEVELOPMENT AND DEPLOYMENT</b> ..... | <b>9</b>  |
| 2.1. CONFIGURING FUSE ONLINE TO ENABLE 3SCALE DISCOVERY OF APIS                                     | 9         |
| 2.2. CONFIGURING OPENSIFT TO ENABLE SERVICE DISCOVERY   | 10        |
| 2.2.1. Configuring Service Discovery with an OpenShift OAuth server                                 | 11        |
| 2.2.2. Configuring Service Discovery with an RH-SSO server (Keycloak)                               | 12        |
| 2.2.3. Configuring Service Discovery without an OAuth server  | 13        |
| 2.3. AUTHORIZING 3SCALE ACCESS TO AN OPENSIFT PROJECT   | 14        |
| <b>CHAPTER 3. CREATING A FUSE ONLINE INTEGRATION THAT IS TRIGGERED BY AN API CALL</b> .....         | <b>16</b> |
| 3.1. CREATING AN API PROVIDER INTEGRATION   | 16        |
| 3.1.1. Creating a REST API definition in API Designer   | 17        |
| 3.1.2. Resolving validation issues in API Designer  | 22        |
| 3.2. DEFINING THE OPERATION FLOWS FOR AN API PROVIDER INTEGRATION                                   | 23        |
| <b>CHAPTER 4. IMPORTING THE API INTO 3SCALE</b> .....   | <b>27</b> |
| 4.1. ABOUT SERVICE DISCOVERY  | 27        |
| 4.1.1. Criteria for a discoverable service  | 27        |
| 4.2. IMPORTING DISCOVERED SERVICES  | 28        |



# PREFACE

Define and deploy REST APIs using Red Hat Fuse Online, API Designer, and 3scale API Management.

# CHAPTER 1. OVERVIEW OF DEVELOPING AND DEPLOYING API PROVIDER INTEGRATIONS

An application programming interface (API) is a set of tools, definitions, and protocols for building application software. An API lets products or services communicate with other products and services without having to know how they are implemented.

As a business user who wants to share data between different applications and services, you can use the Red Hat Integration distributed integration platform to:

- Develop an API provider integration and connect to REST APIs.
- Create applications that consume the data or functionality exposed by REST APIs.
- Connect to products and services, including legacy systems and the Internet of Things (IoT).
- Open up access to your resources while maintaining security and control. How you open access and to whom is up to you.

Red Hat Integration is a set of agile and flexible integration and messaging technologies that are designed to provide API connectivity, data transformation, service composition and orchestration, real-time messaging, cross-datacenter message streaming, and API management to connect applications across hybrid cloud architectures and enable API-centric business services.

Red Hat Integration includes the following products:

## Fuse Online

Red Hat Fuse is a distributed, cloud-native integration platform. Fuse Online is Red Hat's web-based Fuse distribution. It is provided pre-installed on the OpenShift Online Professional tier (and also available for installing on an on premise OpenShift cluster). Fuse Online is for business users who prefer low code development.

## API Designer

Red Hat provides this lightweight version of Apicurio (<https://www.apicur.io/>), enabling you to create API definitions in OpenAPI format. The API Designer is accessible within Fuse Online when you edit an API provider integration.

## 3scale API Management

With Red Hat 3scale API Management, you can set access policies, centralize control, and provide high availability for your APIs.

## Prerequisites

You should have working knowledge of the following concepts:

- Fuse Online concepts
- REST API concepts
- 3scale API Management concepts

## Overview of steps

1. In Fuse Online, create an integration that is triggered by an API call.
  - a. Start the integration with a REST API service. Specify an existing REST API definition or create a new one in the API designer.



Fuse Online creates an execution path, referred to as an integration flow, for each REST API operation.

- b. In each operation's flow, add the connections and other steps that execute that operation. Each REST API client call invokes an operation and triggers execution of only the integration flow that executes that operation.
  - c. Publish the Fuse Online integration, which makes it available as an API service on OpenShift.
2. In 3scale API Management, discover the published API service. You can then secure the API, configure access policies, and launch it.

## 1.1. BENEFIT, OVERVIEW, AND WORKFLOW FOR CREATING API PROVIDER INTEGRATIONS

An API provider integration starts with a REST API service. This REST API service is defined by an OpenAPI 3 (or 2) document that you provide when you create an API provider integration. After you publish an API provider integration, Fuse Online deploys the REST API service on OpenShift. The benefit of an API provider integration is that REST API clients can invoke calls that trigger execution of the integration.

### Multiple execution flows

An API provider integration has multiple execution paths, referred to as flows. Each operation that the OpenAPI document defines has its own flow. In Fuse Online, for each operation that the OpenAPI document defines, you add connections and other steps to the execution flow for that operation. These steps process the data as required for the particular operation.

### Example execution flow

For example, consider a human resources application that calls a REST API service that Fuse Online has made available. Suppose the call invokes the operation that adds a new employee. The operation flow that handles this call could:

- Connect to an application that creates an expense report for new employee equipment.
- Connect to a SQL database to add an internal ticket for setting up new equipment.
- Connect to Google mail to send a message to the new employee that provides orientation information.

### Ways to trigger execution

There are many ways to call the REST APIs that trigger integration execution, including:

- A web browser page that takes data input and generates the call.
- An application that explicitly calls the REST APIs, such as the **curl** utility.
- Other APIs that call the REST API, for example, a webhook.

### Ways to edit a flow

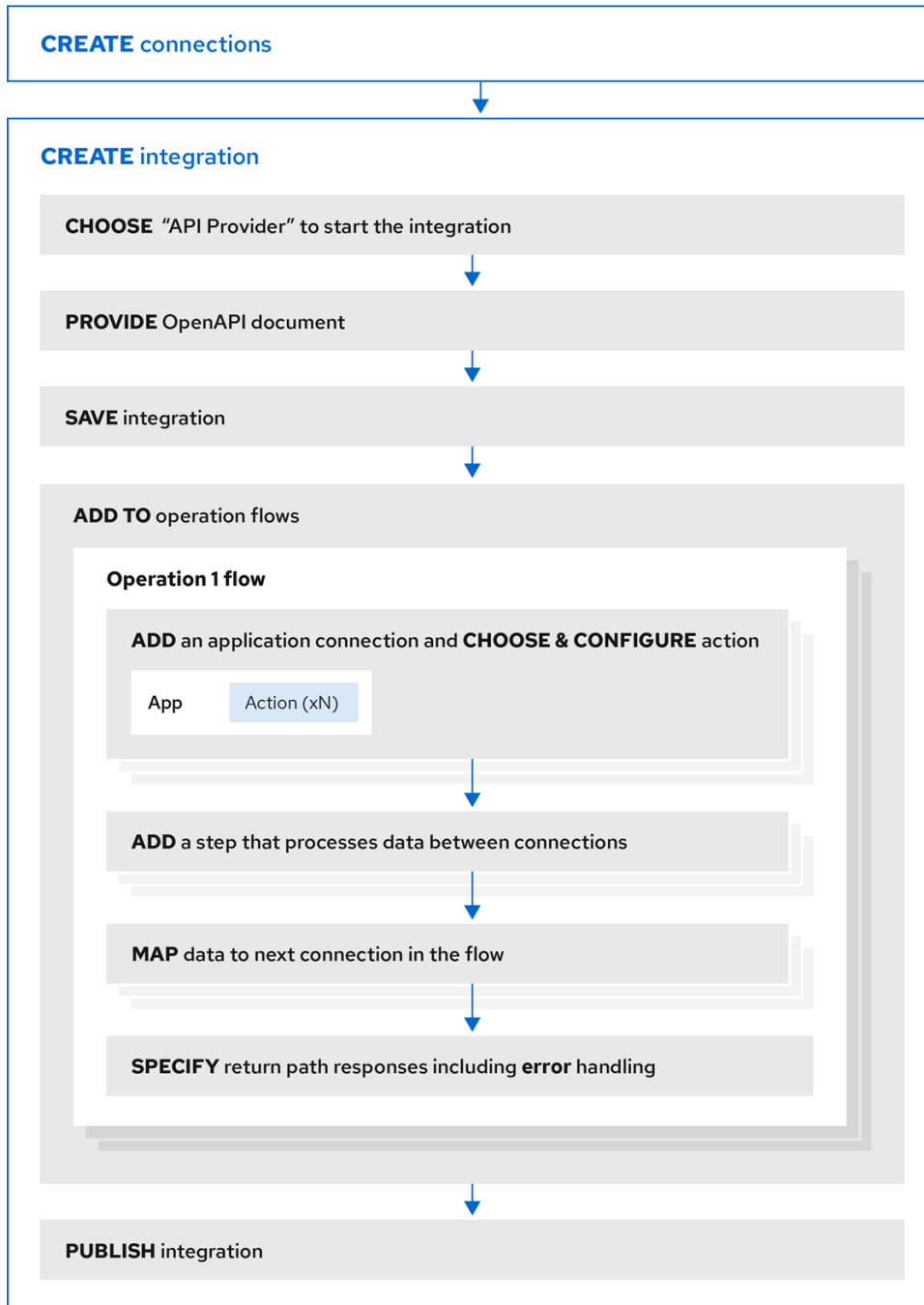
For each operation, you can edit its flow by:

- Adding connections to the applications that need to process the data.
- Adding steps between connections, including split, aggregate, and data mapping steps.

- Mapping connection error messages to return codes in the HTTP response that finishes the flow. The response goes to the application that invoked the call that triggered execution of the integration.

## Workflow for creating an API provider integration

The **general** workflow for creating an API provider integration is shown in the following diagram:



Fuse\_14\_1019

## Publishing an API provider integration

After you publish an API provider integration, in the integration's summary page, Fuse Online displays the external URL for your REST API service. This external URL is the base URL that clients use to call your REST API services.

For Fuse Online environments on OCP, Red Hat 3scale discovery of API provider integrations might be enabled. In this case, 3scale publishes the URL for invoking services.

## Testing an API provider integration

To test an API provider integration's flows, you can use the **curl** utility. For example, the following **curl** command triggers execution of the flow for the **Get Task by ID** operation for the REST API service URL: <https://i-task-api-proj319352.6a63.fuse-ignite.openshiftapps.com/api/>.

The HTTP **GET** command is the default request so there is no requirement to specify **GET**. The last part of the URL specifies the ID of the task to get:

```
curl -k https://i-task-api-proj319352.6a63.fuse-ignite.openshiftapps.com/api/todo/1
```

## 1.2. HOW OPENAPI OPERATIONS RELATE TO API PROVIDER INTEGRATION FLOWS

An API provider integration's OpenAPI document defines the operations that REST API clients can call. Each OpenAPI operation has its own API provider integration flow. Consequently, each operation can also have its own REST API service URL. Each URL is defined by the API service's base URL and optionally by a subpath. REST API calls specify an operation's URL to trigger execution of the flow for that operation.

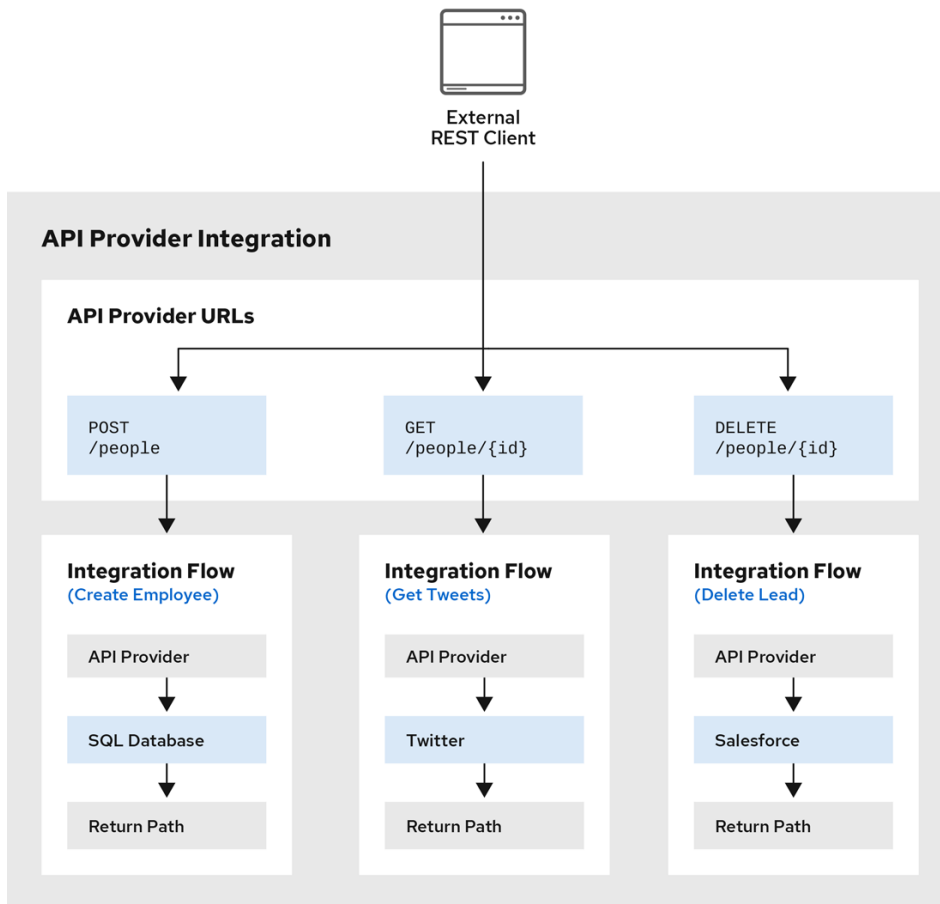
Your OpenAPI document determines which HTTP verbs (such as **GET**, **POST**, **DELETE** and so on) you can specify in calls to your REST API service URLs. Examples of calls to API provider URLs are in the [instructions for trying out the API provider quickstart example](#).

Your OpenAPI document also determines the possible HTTP status codes that an operation can return. An operation's return path can handle only the responses that the OpenAPI document defines. For example, an operation that deletes an object based on its ID might define these possible responses:

```
"responses": {
  "204": {
    "description": "Task deleted"
  },
  "404": {
    "description": "No Record found with this ID"
  },
  "500": {
    "description": "Server Error"
  }
}
```

### Illustration of an API provider integration example

The following diagram shows an API provider integration that processes data about people. An external REST API client invokes the REST API URLs that are deployed by the API provider integration. Invocation of a URL triggers execution of the flow for one REST operation. This API provider integration has 3 flows. Each flow can use any connection or step that is available in Fuse Online. The REST API along with its flows is one Fuse Online API provider integration, which is deployed in one OpenShift pod.



Fuse\_19\_1019

## Editing the OpenAPI document while creating an API provider integration

After you specify an OpenAPI document for your API provider integration, you can update the document as needed while you define the execution flows for the API operations. To do this, click **View/Edit API Definition** in the upper right of a page in which you are editing the API provider integration. This displays your OpenAPI document in the API Designer editor. Edit and save the document to make changes that are reflected in Fuse Online.

Considerations while editing the OpenAPI document:

- operationId properties for synchronization**  
 Synchronization between the versions of the OpenAPI document in the API Designer editor and in the Fuse Online integration editor depend on a unique **operationId** property that is assigned to each operation that is defined in the document. You can assign a specific **operationId** property value to each operation, or use the one that Fuse Online generates automatically.
- Request and response definitions**  
 In each operation's definition, you can supply a JSON schema that defines the operation's request and response. Fuse Online uses the JSON schema:
  - As the basis for the operation's input and output data shapes
  - To display operation fields in the data mapper
- No cyclic schema references**  
 A JSON schema for an API provider integration operation cannot have cyclic schema references. For example, a JSON schema that specifies a request or response body cannot reference itself as a whole nor reference any part of itself through intermediate JSON schemas.

## CHAPTER 2. CONFIGURING RED HAT INTEGRATION PRODUCTS FOR API DEVELOPMENT AND DEPLOYMENT

Your OpenShift Container Platform (OCP) on premise administrator must configure the Red Hat Integration products for API development and deployment:

- Deploy Fuse Online and 3scale API Management on the same OpenShift cluster (requires OpenShift cluster admin permission).
- Configure Fuse Online to enable 3scale discovery of APIs (requires permission to install Fuse Online to an OpenShift project).
- Configure 3scale for service discovery (requires 3scale admin permission).

In addition, the following configurations are required:

- For Fuse Online, the default behavior is that APIs are not exposed for automatic discovery in 3scale. You must enable discovery for the OpenShift project in which you install Fuse Online.
- A 3scale API Management administrator must configure 3scale for service discovery. For example, the administrator must ensure that users have proper permissions to view cluster projects that contain discoverable services.

### 2.1. CONFIGURING FUSE ONLINE TO ENABLE 3SCALE DISCOVERY OF APIS

If you create an API provider integration, you might want to enable discovery of the API for that integration in Red Hat 3scale. The default behavior is that APIs are not exposed for automatic discovery in 3scale. When you enable discovery, you must provide a URL for a 3scale user interface.

To configure Fuse Online to enable 3scale discovery of APIs before you install Fuse Online, see [Descriptions of custom resource attributes that configure Fuse Online](#).

After installation, you can enable discovery by updating the **syndesis** custom resource. Instructions for doing this are in this topic. When you enable discovery, it applies to only the OpenShift project that you are connected to when you update the resource.

Turning on 3scale service discovery means that:

- The default behavior is that 3scale publishes API provider integrations. When 3scale publishes an API provider integration:
  - Fuse Online does not provide an external URL for an API provider integration that is running.
  - The API is accessible only through 3scale. Configuration in 3scale is required to expose the endpoint. For details, see [Red Hat 3scale API Management, Service Discovery](#).
- The creator of an API provider integration can disable 3scale discovery for that integration. In other words, each API provider integration creator can choose whether that integration's API is discoverable.

#### Prerequisites

- Fuse Online is installed on OCP on-site.

- The **oc** client tool is installed and it is connected to the OCP cluster in which Fuse Online is installed.
- You have permission to install Fuse Online in the project for which you want to enable discovery of APIs.

## Procedure

1. Log in to OpenShift with an account that has permission to install Fuse Online. For example:  
**oc login -u developer -p developer**
2. Switch to an OpenShift project in which Fuse Online is running. You are enabling discovery for only this project. For example:  
**oc project my-fuse-online-project**
3. Edit the **syndesis** custom resource:
  - a. Invoke the following command, which typically opens the resource in an editor:  
**oc edit syndesis**
  - b. Edit the resource by setting **managementUrlFor3scale** to the URL for your 3scale user interface. The result looks like this:

```
spec:
  components:
    server:
      features:
        managementUrlFor3scale: https://3scale-admin.apps.mycluster.com
```

- c. Save the resource.
4. Optional. To confirm that discovery is turned on for the project that you switched to, invoke the following command:  
**oc describe dc/syndesis-server**

When discovery is turned on, the output from this command shows that the **OPENSIFT\_MANAGEMENT\_URL\_FOR3SCALE** environment variable is set to the URL that you specified in the custom resource.

## Results

This change to the **syndesis** custom resource triggers **syndesis-operator**, which is responsible for installing Fuse Online, to redeploy **syndesis-server**. In the OpenShift project that you switched to, the new default behavior is that APIs are exposed for discovery in 3scale.

Do not edit the **syndesis-server DeploymentConfig** object to set the **OPENSIFT\_MANAGEMENT\_URL\_FOR3SCALE** environment variable. This does not work because **syndesis-operator** reverts your change. The **syndesis-operator** ensures that Fuse Online is deployed only and always according to the **syndesis** custom resource.

## 2.2. CONFIGURING OPENSIFT TO ENABLE SERVICE DISCOVERY

As a 3scale administrator, you can configure Service Discovery with or without an Open Authorization (OAuth) server.

If you configure 3scale Service Discovery with an OAuth server, this is what happens when a user signs in to 3scale:

- The user is redirected to the OAuth Server.
- If the user is not already logged in to the OAuth Server, the user is prompted to log in.
- If it is the first time that the user implements 3scale Service Discovery with SSO, the OAuth server prompts for authorization to perform the relevant actions.
- The user is redirected back to 3scale.

To configure Service Discovery with an OAuth server, you have the following options:

- [Configuring Service Discovery with an OpenShift OAuth server](#)
- [Configuring Service Discovery with an RH-SSO server \(Keycloak\)](#)

If you [configure Service Discovery without an OAuth server](#), when a user signs in to 3scale, the user is not redirected. Instead, the 3scale Single Service Account provides a seamless authentication to the cluster for the Service Discovery. All 3scale tenant administration users have the same access level to the cluster while discovering API services through 3scale.

### 2.2.1. Configuring Service Discovery with an OpenShift OAuth server

As a 3scale system administrator, you can allow users to individually authenticate and authorize 3scale to discover APIs by using OpenShift built-in OAuth server.

#### Prerequisites

- You must deploy 3scale 1.1 to an OpenShift cluster (version 3.11 or later).
- To deploy 3scale to OpenShift, you need to use [3scale-amp-openshift-templates](#).
- 3scale users that want to use Service Discovery in 3scale must have access to the OpenShift cluster.

#### Procedure

1. Create an OpenShift OAuth client for 3scale. For more details, see the [OpenShift Authentication documentation](#). In the following example, replace **<provide-a-client-secret>** with a secret that you generate and replace **<3scale-master-domain-route>** with the URL to access the 3scale Master Admin Portal.

```
$ oc project default
$ cat <<-EOF | oc create -f -
kind: OAuthClient
apiVersion: v1
metadata:
  name: 3scale
secret: "<provide-a-client-secret>"
redirectURIs:
  - "<3scale-master-domain-route>"
grantMethod: prompt
EOF
```

- Open the 3scale Service Discovery settings file:

```
$ oc project <3scale-project>
$ oc edit configmap system
```

- Configure the following settings:

```
service_discovery.yml:
production:
  enabled: true
  authentication_method: oauth
  oauth_server_type: builtin
  client_id: '3scale'
  client_secret: '<choose-a-client-secret>'
```

- Ensure that users have proper permissions to view cluster projects containing discoverable services.

To give an administrator user, represented by `<user>`, the view permission for the `<namespace>` project containing a service to be discovered, use this command:

```
oc adm policy add-role-to-user view <user> -n <namespace>
```

- After modifying **configmap**, you must redeploy the **system-app** and **system-sidekiq** pods to apply the changes.

```
oc rollout latest dc/system-app
oc rollout latest dc/system-sidekiq
```

- Check the status of the rollout to ensure it has finished:

```
oc rollout status dc/system-app
oc rollout status dc/system-sidekiq
```

### Additional note

For more information about OpenShift OAuth tokens, [Configuring the internal OAuth server](#).

## 2.2.2. Configuring Service Discovery with an RH-SSO server (Keycloak)

As a system administrator, you can allow users to individually authenticate and authorize 3scale to discover services by using [Red Hat Single Sign-On for OpenShift](#).

For an example about configuring OpenShift to use the RH-SSO deployment as the authorization gateway for OpenShift, you can refer to this [workflow](#).

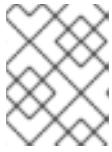
### Prerequisites

- You must deploy 3scale 1.1 to an OpenShift cluster (version 3.11 or later).
- To deploy 3scale to OpenShift, you need to use [3scale-amp-openshift-templates](#).
- 3scale users that want to use Service Discovery in 3scale must have access to the OpenShift cluster.



## Procedure

1. Create an OAuth client for 3scale in Red Hat OAuth server (Keycloak).



### NOTE

In the client configuration, verify that the **username** maps to **preferred\_username**, so that OpenShift can link accounts.

2. Edit 3scale Service Discovery settings.

```
$ oc project <3scale-project>
$ oc edit configmap system
```

3. Verify that the following settings are configured, where `<the-client-secret-from-Keycloak>` is the value that Keycloak generated automatically when you created the OAuth client.

```
service_discovery.yml:
  production:
    enabled: true
    authentication_method: oauth
    oauth_server_type: rh_sso
    client_id: '3scale'
    client_secret: '<the-client-secret-from-Keycloak>'
```

4. Make sure that users have proper permissions to view cluster projects containing discoverable services.  
For example, to give `<user>` view permission for the `<namespace>` project, use this command:

```
oc adm policy add-role-to-user view <user> -n <namespace>
```

5. After modifying **configmap**, you must redeploy the **system-app** and **system-sidekiq** pods to apply the changes.

### Additional note:

- Token lifespan: By default, session tokens expire after one minute, as indicated in [Keycloak - Session and Token Timeouts](#). However, it is recommended to set the timeout to an acceptable value of one day.

## 2.2.3. Configuring Service Discovery without an OAuth server

To configure the 3scale Service Discovery without an OAuth server, you can use 3scale Single Service Account to authenticate to OpenShift API service.

### Prerequisites

- You must deploy 3scale 1.1 to an OpenShift cluster (version 3.11 or later).
- To deploy 3scale to OpenShift, you need to use [3scale-amp-openshift-templates](#).
- 3scale users that want to use Service Discovery in 3scale must have access to the OpenShift cluster.

## Procedure

1. Verify that the 3scale project is the current project.

```
$ oc project <3scale-project>
```

2. Open the 3scale Service Discovery settings in an editor.

```
$ oc edit configmap system
```

3. Verify that the following settings are configured.

```
service_discovery.yml:
  production:
    enabled: <%= cluster_token_file_exists = File.exists?(cluster_token_file_path =
'/var/run/secrets/kubernetes.io/serviceaccount/token') %>
    bearer_token: "<%= File.read(cluster_token_file_path) if cluster_token_file_exists %>"
    authentication_method: service_account
```

4. Provide the 3scale deployment **amp** service account with the relevant permissions to view projects containing discoverable services by following one of these options:

- Grant the 3scale deployment **amp** service account with **view** cluster level permission.

```
oc adm policy add-cluster-role-to-user view system:serviceaccount:<3scale-project>:amp
```

- Apply a more restrictive policy as described in [OpenShift - Service Accounts](#).

## 2.3. AUTHORIZING 3SCALE ACCESS TO AN OPENSIFT PROJECT

As an OpenShift project administrator, you can authorize a 3scale user to access a namespace when the OAuth token is not valid.

### Prerequisites

- You need to have the credentials as an OpenShift project administrator.
- The OpenShift administrator has configured Service Discovery for the OpenShift cluster. For example, for Fuse Online APIs, the OpenShift administrator must set the Fuse Online service's **CONTROLLERS\_EXPOSE\_VIA3SCALE** environment variable to **true**.
- The 3scale administrator has configured the 3scale deployment for Service Discovery as described in [Section 4.1, "About Service Discovery"](#).
- You know the API service name and its namespace of the OpenShift project.
- The API service is deployed on the same OpenShift cluster where 3scale is installed.
- The API has the correct annotations that enable Service Discovery, as described in [Section 4.1, "About Service Discovery"](#).

### Procedure

1. Click the *Authenticate to enable this option* link.

2. Log in to OpenShift using the namespace administrator credentials.
3. Authorize access to the 3scale user, by clicking **Allow selected permissions**.

### Next steps

See the [Red Hat 3scale API Management documentation](#) for information about managing the API.

## CHAPTER 3. CREATING A FUSE ONLINE INTEGRATION THAT IS TRIGGERED BY AN API CALL

To trigger execution of an integration on demand, start the integration with a REST API service that you provide. Integrations that start this way are referred to as *API provider integrations*. An API provider integration allows REST API clients to invoke commands that trigger execution of the integration.

When Fuse Online publishes an API provider integration, any client with network access to the integration endpoints can trigger execution of the integration.

By default, Fuse Online annotates an API provider integration's API service definition for use with 3scale.

The following topics provide information and instructions for creating API provider integrations:

- [Section 3.1, "Creating an API provider integration"](#)
- [Section 3.2, "Defining the operation flows for an API provider integration"](#)

### 3.1. CREATING AN API PROVIDER INTEGRATION

To create an API provider integration, provide an OpenAPI document (**.json**, **.yaml**, or **.yml** file) that defines the operations that the integration can perform. Fuse Online creates an execution flow for each operation. Edit the flow for each operation to add connections and steps that process integration data according to the requirements for that operation.

#### Prerequisites

- You are able to provide or define an OpenAPI document for the REST API operations that you want the integration to perform.  
To experiment, [download the raw version of the \*\*task-api.json\*\* file](#), which is an OpenAPI document for an API provider quickstart. You can upload this file when Fuse Online prompts you to provide an OpenAPI document. Alternatively, you can specify the URL for the raw **task-api.json** file, which is <https://raw.githubusercontent.com/syndesio/syndesis-quickstarts/1.10/api-provider/task-api.json>.
- You have a plan for the flow for each OpenAPI operation.
- You created a connection for each application or service that you want to add to an operation's flow.

#### Procedure

1. In Fuse Online, in the left navigation panel, click **Integrations**.
2. Click **Create Integration**.
3. On the **Choose a connection** page, click **API Provider**.
4. On the **Start integration with an API call** page:
  - If you have an OpenAPI document that defines the REST API operations, upload the OpenAPI document.

- If you need to define the OpenAPI document, select **Create a new OpenAPI 3.x document** or **Create a new OpenAPI 2.x document**
5. Click **Next**.
- If you uploaded a document, review or edit it:
    - a. Click **Review/Edit** to open the API Designer editor.
    - b. Review and edit as needed.  
Optionally, if your document uses the OpenAPI 2 specification, you can click **Convert to OpenAPI 3** if you want the API Designer to convert your document to conform with the OpenAPI 3 specification.
    - c. In the upper right, click **Save** or **Cancel** to close the editor.
    - d. Click **Next**.
  - If you are creating a document, then in the API Designer editor that Fuse Online opens:
    - a. Define the OpenAPI document as described in [Creating a REST API definition in API Designer](#).
    - b. In the upper right, click **Save**, which closes the editor.
    - c. Click **Next**.

## Result

Fuse Online displays a list of the operations that the OpenAPI document defines.

## Next step

For each operation, [define a flow that executes that operation](#) .

### 3.1.1. Creating a REST API definition in API Designer

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

#### 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.

#### Prerequisite

- 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}**.

#### Procedure


1. Click **New API**. A new API page opens.  
By default, API Designer uses the OpenAPI 3 specification. If you want to use the OpenAPI 2 specification, click the arrow next to the **New API** button and then select **OpenAPI 2**



## NOTE

If you open an API based on the OpenAPI 2 specification, you can use the API Designer's **Convert to OpenAPI 3** option to convert the API to comply with the OpenAPI 3 specification.

### 2. To change the API name:

- Hover the cursor over the name and then click the edit icon (  ) that appears.
- Edit the name. For example, type **Task API**.
- Click the checkmark icon to confirm the name change.

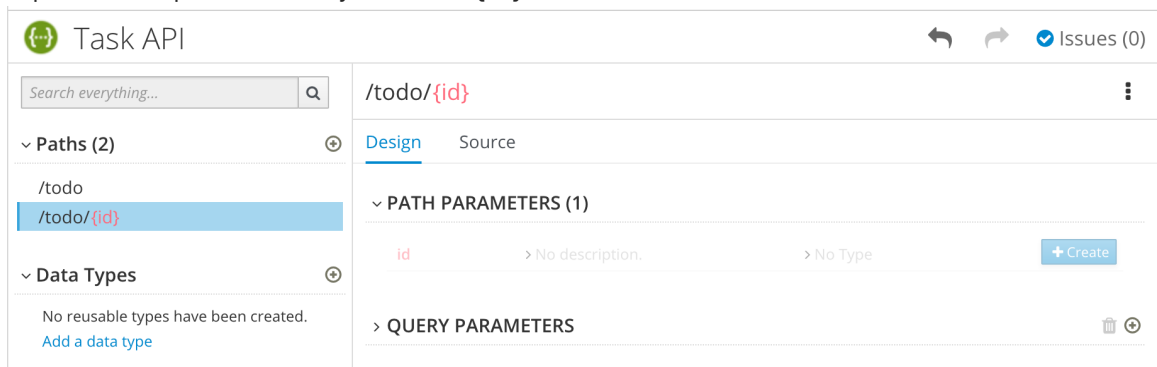
### 3. Optionally:

- Provide a version number and a description.
- Add your contact information (name, email address, and URL).
- Select a license.
- Define tags.
- Define one or more servers.
- Configure a security scheme.
- Specify security requirements.

### 4. 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'. The top bar includes a search bar, a refresh icon, a back icon, and 'Issues (0)'. The main area is divided into two panels. The left panel shows a list of paths: '/todo' and '/todo/{id}', with the latter selected. Below the paths is a 'Data Types' section with a message: 'No reusable types have been created. Add a data type'. The right panel shows the details for the selected path '/todo/{id}'. It has tabs for 'Design' and 'Source'. Under 'PATH PARAMETERS (1)', there is a parameter 'id' with a description '> No description.' and a type '> No Type'. A '+ Create' button is visible next to the parameter. Below this is a 'QUERY PARAMETERS' section with a trash icon and a plus icon.

### 5. Specify the type of any path parameters.

For the example **id** parameter:

- In the **Paths** list, click **/todo/{id}**.  
The **id** parameter appears in the **PATH PARAMETERS** section.
- 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, under 'Paths (2)', the path `/todo/{id}` is selected. Below it, the 'Data Types' section shows 'No reusable types have been created.' and a link 'Add a data type'. On the right, the 'PATH PARAMETERS (1)' section shows a parameter `id` with the description '> The ID of the task to find.' and the type 'integer as int32'. Below this, a configuration box shows 'Type: Integer' and 'as 32-Bit Integer'.

6. 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 the 'Task API'. The 'Data Types (1)' section shows a new data type '</> Todo' selected. The 'Properties (3)' section shows the following properties:

| Property Name | Description       | Type               |
|---------------|-------------------|--------------------|
| completed     | > No description. | > boolean          |
| id            | > No description. | > integer as int32 |
| task          | > No description. | > string           |

Below the properties, the 'EXAMPLE' section shows the following JSON schema:

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

7. Optionally, you can add edit the schema properties and add new ones.
8. For the Task Management API example, create another data type named **Task** with one property named **task** of type **string**.

9. 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 3.1. Task Management API operations

| Path         | Operation   | Description        | Request Body  | Response   |
|--------------|-------------|--------------------|---|--|
| <b>/todo</b> | <b>POST</b> | Create a new task. | Media Type: <b>application/json</b><br>Data Type: <b>Task</b> | <ul style="list-style-type: none"> <li>Status Code: <b>201</b><br/>Description: <b>Task created</b><br/>Response Body: Media Type: <b>application/json</b><br/>Data Type: <b>Todo</b></li> </ul> |
| <b>/todo</b> | <b>GET</b>  | Get all tasks.     | <i>Not applicable</i>   | <ul style="list-style-type: none"> <li>Status Code: <b>200</b><br/>Description: <b>Task deleted</b></li> <li>Status Code: <b>400</b><br/>Description: <b>Task not deleted</b></li> </ul>         |



| Path              | Operation     | Description          | Request Body                   | Response   |
|-------------------|---------------|----------------------|--------------------------------|--|
| <b>/todo/{id}</b> | <b>GET</b>    | Get a task by ID.    | <i>Not applicable</i>          | <ul style="list-style-type: none"> <li>● Status Code: <b>200</b><br/>Description: <b>Task found for ID</b><br/>Response Body: Media Type: <b>application/json</b> Data type: <b>Task</b></li> <li>● Status Code: <b>404</b><br/>Description: <b>No task with provided identifier found.</b></li> </ul> |
| <b>/todo/{id}</b> | <b>UPDATE</b> | Update a task by ID. | Request Body type: <b>Task</b> | <ul style="list-style-type: none"> <li>● Status Code: <b>200</b><br/>Description: <b>Completed</b></li> <li>● Status Code: <b>400</b><br/>Description: <b>Task not updated</b></li> </ul>  |
| <b>/todo/{id}</b> | <b>DELETE</b> | Delete a task by ID. | <i>Not applicable</i>          | <ul style="list-style-type: none"> <li>● Status Code: <b>200</b><br/>Description: <b>Task deleted</b></li> <li>● Status Code: <b>400</b><br/>Description: <b>Task not deleted</b></li> </ul>   |

10. Resolve any issues, as described in [Resolving validation issues in API Designer](#) .

#### Additional resources

- For information about the OpenAPI Specification, go to: <https://github.com/OAI/OpenAPI-Specification>

### 3.1.2. Resolving validation issues in API Designer

When you create and edit an API, API Designer identifies issues that you must resolve with an exclamation (!) icon and also with a list of issues in the API Designer title bar.

#### Prerequisites

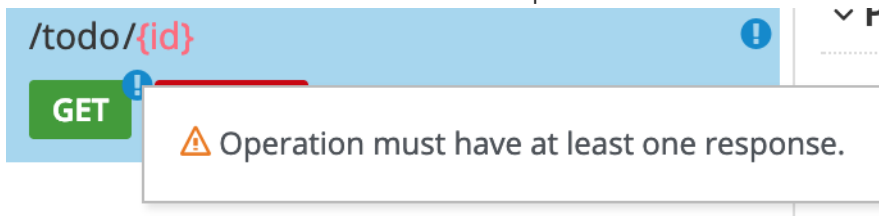
- Open an API in API Designer.

#### 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:



3. 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**.

The screenshot shows the API Designer interface for the endpoint `/todo/{id}` with a **GET** operation. The interface includes a **Design** tab and a **Source** tab. Below the endpoint information, there are sections for **INFO**, **PATH PARAMETERS (1)**, **REQUEST BODY**, **QUERY PARAMETERS**, and **RESPONSES**. The **RESPONSES** section is expanded, showing a warning icon and the text: "No responses have been defined. [Add a response](#)".

After you type a description for the response, the issue is resolved and the exclamation icon disappears:

TaskAPI

Search everything... Q

↩ ↻ Issues (0)

/todo/{id} GET

Design Source

id The unique identifier of the task integer Override

> REQUEST BODY

> QUERY PARAMETERS

> RESPONSES (1)

201 Created Task created. No Type

Description  
Task created.

4. For a summary of all issues:

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

TaskAPI

↩ ↻ Issues (3)

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

↩ ↻ Issues (3)

Validation Problems X

⚠ **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](#)

## 3.2. DEFINING THE OPERATION FLOWS FOR AN API PROVIDER INTEGRATION

The OpenAPI document that defines your REST API service defines the operations that the service can perform. After you create an API provider integration, you can edit the flow for each operation.

Each operation has exactly one flow. In an operation flow, you can add connections to other applications and services, as well as steps that operate on data between connections.

As you add to operation flows, you might find that you need to update the OpenAPI document that the API provider integration is based on. To do this, click **View/Edit API Definition** in the upper right of a page in which you are editing your API provider integration. This displays your document in the API Designer editor. In your OpenAPI definition, as long as each operation has a unique **operationId** property, you can save your updates in API Designer and Fuse Online can synchronize the API provider integration's flow definitions to have your updates.

## Prerequisites

- You created an API provider integration, gave it a name, and saved it.
- You created a connection to each application or service that you want an operation flow to connect to. For details, see the [information about creating connections](#).
- Fuse Online is displaying the list of operations that the API defines.


## Procedure

1. In the **Operations** list page, for the operation whose flow you want to define, click **Create flow**.
2. For each connection that you want to add to this flow:
  - a. In the flow visualization, click the plus sign to add a connection at that location.
  - b. Click the connection that you want to add.
  - c. Select the action that you want this connection to perform.
  - d. Configure the action by entering data in the labeled fields.
  - e. Click **Next**.

Add all desired connections to the flow before you continue.

3. In this operation flow, to process data between connections:
  - a. In the flow visualization, click the plus sign where you want to add a step.
  - b. Click the step that you want to add.
  - c. Configure the step by entering data in the labeled fields.
  - d. Click **Next**.  
For help, see [Adding steps between connections](#).

If you want to add another step that processes data between connections, repeat this subset of instructions.

4. Map data to fields in the next connection:
  - a. In the flow visualization, check for data type mismatch  icons, which indicate that the

connection cannot process the incoming data. You need to add a data mapper step here.

- b. For each data mismatch icon in the flow visualization:
  - i. Click the plus sign that is just before that step.
  - ii. Click **Data Mapper**.
  - iii. Define the needed mappings. For help, see [Mapping integration data to fields in the next connection](#).
  - iv. Click **Done** to add the data mapper step to the flow.
5. In the flow visualization, on the **Provided API Return Path** step, click **Configure**.
 

Every API provider integration finishes each operation flow by sending a response to the REST API caller that triggered execution of the operation flow. The response contains one of the return codes that you configure for the **Provided API Return Path** step that finishes the operation's flow. Configure the return path step as follows:

  - a. Under **Default Response**, in the **Return Code** field, accept the default response that Fuse Online displays, or click the down caret and scroll to select the default response that you want. The flow sends this response when execution of the operation flow does not return any of the configured error responses. Typically, the default response return code indicates a successful operation.
  - b. Under **Error Handling**, indicate whether you want to include the error message in the body of the returned message.
 

During development, you typically want to return the error message. In production, however, you might want to hide the error message if it contains sensitive or proprietary information. The error message is a JSON formatted string that contains **responseCode**, **category**, **message**, and **error** elements, for example:

```

{
  responseCode: 404,
  category: "ENTITY_NOT_FOUND_ERROR",
  message: "SQL SELECT did not SELECT any records"
  error: SYNDESIS_CONNECTION_ERROR
}
          
```

Note that during development, the most reliable way to know that an error happened is to check the **HTTP\_RESPONSE STATUS** header in the response to the caller. You can also check the integration pod's log for **INFO** messages. The integration's **Activity** log shows a successful exchange and errors are not always visible in the **Activity** log.
  - c. Under **Error Response Codes**, Fuse Online displays an entry for each error that a connection in the flow might return. For each error, accept the **200 All is good** default return code or click to select another HTTP status return code.
 

The return codes that you can select from, are the return codes that the OpenAPI document defines for the operation that this flow executes. If Fuse Online does not display a return code that you need, you can edit the OpenAPI document to add it.

To do this, in the upper right, click **View/Edit API Definition**. Edit the OpenAPI document as needed. When you are done, save the OpenAPI document. Fuse Online returns to editing the **Provided API Return Path** and reflects any changes that you saved.
  - d. Click **Next** to complete configuration of the return path.

6. When this flow has all needed connections and steps and there are no data mismatch icons, or when you no longer want to edit the flow for now, do one of the following:
  - **Publish** – To start running the integration, in the upper right, click **Publish**. This builds the integration, deploys the REST API service to OpenShift, and makes the integration available to be executed. You can publish the integration each time that you complete the creation of an operation's flow or each time that you edit an operation's flow.
  - **Save** – To display the list of operations, in the upper right, click **Save**.

Repeat this procedure to edit another operation's flow.

### Testing API provider integrations

- Testing API provider integrations running on one of these platforms:
  - OpenShift Online
  - OpenShift Dedicated
  - OpenShift Container Platform when **API discovery is disabled**

You can use the **curl** utility to confirm that the integration is working as expected. In the **curl** command, specify the external URL that Fuse Online displays after it publishes the API provider integration. For examples of doing this, see [Testing the example API provider quickstart integration](#).

- Testing API provider integrations running on OpenShift Container Platform when **API discovery is enabled**

Red Hat 3scale publishes your API provider integration. To test the integration, open the 3scale dashboard to obtain the integration's URL.

You can disable discovery for an API provider integration if, for example, you do not want Red Hat 3scale to control access to the integration's API or you want to test the API provider integration in Fuse Online. If you disable discovery, Fuse Online republishes the integration and provides an external URL for invoking and testing integration execution. To do this, in Fuse Online go to the integration's summary page. On this page, click **Disable discovery**. Fuse Online republishes the integration and provides the integration's URL. For examples of how to test an integration, see [Testing the example API provider quickstart integration](#) . After testing, you can re-enable discovery for the API provider integration so that 3scale publishes it.

You can enable or disable discovery for each API provider integration.

## CHAPTER 4. IMPORTING THE API INTO 3SCALE

With Red Hat 3scale API Management's service discovery feature, you can import services from OpenShift.

### 4.1. ABOUT SERVICE DISCOVERY

Using Service Discovery, you can scan for discoverable API services that are running in the same OpenShift cluster and automatically import the associated API definitions into 3scale.

You can also update the API integration and the Open API Specification at any time, to later resynchronize them with the cluster.

Service Discovery offers the following features:

- Uses the cluster API to query for services that are properly annotated for discovery.
- Configures 3scale to access the service using an internal endpoint inside the cluster.
- Imports, as 3scale ActiveDocs, the OpenAPI Specification associated with the service.
- Supports OpenShift and Red Hat Single Sign-On (RH SSO) authorization flows.
- Works with Red Hat Fuse, starting with Fuse version 7.2.

When you import a discoverable service, it keeps its namespace within the project it belongs to. The imported service becomes a new customer-facing API, product, and its corresponding internal API, backend.

- For 3scale on premises, the 3scale API provider may have its own namespace and services. Discovered services can co-exist with 3scale existing and native services.
- Fuse discoverable services are deployed to the Fuse production namespace.

#### 4.1.1. Criteria for a discoverable service

If you want 3scale find an API in an OpenShift cluster, said API must meet the criteria for each element below:

##### Content-Type header

The API specification's **Content-Type** header must be one of the following values:

- **application/swagger+json**
- **application/vnd.oai.openapi+json**
- **application/json**

##### OpenShift Service Object YAML definition

- The OpenShift Service Object YAML definition must include the following metadata:
  - The **discovery.3scale.net** label: (required) Set 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.
  - discovery.3scale.net/scheme:** (required) The scheme part of the URL where the service is hosted. Possible values are "http" or "https".
  - discovery.3scale.net/port:** (required) The port number of the service within the cluster.
  - discovery.3scale.net/path:** (optional) The relative base path of the URL where the service is hosted. You can omit this annotation when the path is at root, "/".
  - discovery.3scale.net/description-path:** The path to the OpenAPI service description document for the service.

For example:

```

metadata:
  annotations:
    discovery.3scale.net/scheme: "https"
    discovery.3scale.net/port: '8081'
    discovery.3scale.net/path: "/api"
    discovery.3scale.net/description-path: "/api/openapi/json"
  labels:
    discovery.3scale.net: "true"
  name: i-task-api
  namespace: fuse

```

- If you are an OpenShift user with administration privileges, you can view the API service's YAML file in the OpenShift Console:
  1. Select **Applications > Services**.
  2. Select the service, for example **i-task-api**, to open its Details page.
  3. Select **Actions > Edit YAML** to open the YAML file.
  4. When you have finished viewing it, select **Cancel**.

### Clusters with the **ovs-networkpolicy** plugin

- To allow traffic between the OpenShift and 3scale projects, clusters that have the **ovs-networkpolicy** plugin require **NetworkPolicy** objects created within their application project.
- For more information about configuring a **NetworkPolicy** object, see [About network policy](#)



#### NOTE

When you create an API provider integration in Fuse Online, the API automatically includes these required annotations.

## 4.2. IMPORTING DISCOVERED SERVICES

From the OpenShift cluster, you can import a new API service that conforms to the OpenAPI Specification. This API can be managed with 3scale.



## Prerequisites

- The OpenShift administrator has configured Service Discovery for the OpenShift cluster. For example, the OpenShift administrator must have enabled 3scale discovery by editing the Fuse Online custom resource to specify the URL for their 3scale user interface.
- The 3scale administrator has configured the 3scale deployment for Service Discovery as described in [About Service Discovery](#).
- The 3scale administrator has granted your 3scale user or service account (depending on the configured authentication mode) the necessary privileges to view the API service and its namespace. For more details, you can see [Authorizing 3scale access to an OpenShift project](#).
- The API has the correct annotations that enable Service Discovery, as described in [Criteria for a discoverable service](#).
- The API service is deployed on the same OpenShift cluster where 3scale is installed.
- You know the API's service name and its namespace (OpenShift project).

## Procedure

1. Log in to the 3scale Administration Portal.
2. From the Admin Portal's Dashboard, click **New API**.
3. Choose **Import from OpenShift**.
  - If the OAuth token is not valid, the OpenShift project administrator should authorize access to the 3scale user as described in [Authorizing 3scale access to an OpenShift project](#).
4. In the **Namespace** field, specify or select the OpenShift project that contains the API, for example **fuse**.
5. In the **Name** field, type or select the name of an OpenShift service within that namespace, for example **i-task-api**.
6. Click **Create Service**.
7. Wait for the new API service to be asynchronously imported into 3scale. A message appears in the upper right section of the Admin Portal: **The service will be imported shortly. You will receive a notification when it is done.**

## Next steps

See the [Red Hat 3scale API Management documentation](#) for information about managing the API.