Red Hat 3scale API Management 2.8

Providing APIs in the Developer Portal

A properly configured Developer Portal provides plenty of functionalities for API management.
A properly configured Developer Portal provides plenty of functionalities for API management.
Abstract

This guide documents the uses of the Developer Portal on Red Hat 3scale API Management 2.8.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE</td>
<td>3</td>
</tr>
<tr>
<td>PART I. OPENAPI SPECIFICATION (OAS)</td>
<td>4</td>
</tr>
<tr>
<td>CHAPTER 1. CREATING A NEW SERVICE BASED ON OAS</td>
<td>5</td>
</tr>
<tr>
<td>1.1. INTRODUCTION</td>
<td>5</td>
</tr>
<tr>
<td>1.2. PREREQUISITES</td>
<td>5</td>
</tr>
<tr>
<td>1.3. FEATURES OF OPENAPI SPECIFICATION</td>
<td>5</td>
</tr>
<tr>
<td>1.4. USING OPENAPI SPECIFICATION</td>
<td>5</td>
</tr>
<tr>
<td>1.4.1. Detecting OpenAPI definition from the filename path</td>
<td>6</td>
</tr>
<tr>
<td>1.4.2. Detecting OpenAPI definition from a URL</td>
<td>6</td>
</tr>
<tr>
<td>1.4.3. Detecting OpenAPI definition from stdin</td>
<td>6</td>
</tr>
<tr>
<td>PART II. API DOCUMENTATION</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 2. ADDING SPECIFICATIONS TO 3SCALE</td>
<td>8</td>
</tr>
<tr>
<td>2.1. NAVIGATE TO SERVICE SPECIFICATIONS IN ACTIVEDOC</td>
<td>8</td>
</tr>
<tr>
<td>2.2. CREATE A SERVICE SPECIFICATION</td>
<td>8</td>
</tr>
<tr>
<td>2.3. WORKING WITH YOUR FIRST ACTIVEDOC</td>
<td>9</td>
</tr>
<tr>
<td>CHAPTER 3. CREATE AN OAS SPECIFICATION</td>
<td>11</td>
</tr>
<tr>
<td>3.1. ABOUT OPENAPI SPECIFICATION (OAS)</td>
<td>11</td>
</tr>
<tr>
<td>3.2. 3SCALE ACTIVEDOC AND OAS</td>
<td>11</td>
</tr>
<tr>
<td>3.3. CREATING THE SPECIFICATION OF YOUR API</td>
<td>11</td>
</tr>
<tr>
<td>3.3.1. Learning by example: the Petstore API</td>
<td>11</td>
</tr>
<tr>
<td>3.3.2. More on the OAS specification</td>
<td>12</td>
</tr>
<tr>
<td>3.3.2.1. OAS object</td>
<td>12</td>
</tr>
<tr>
<td>3.3.2.2. Info object</td>
<td>13</td>
</tr>
<tr>
<td>3.3.2.3. Paths object</td>
<td>13</td>
</tr>
<tr>
<td>3.3.3. OAS design and editing tools</td>
<td>13</td>
</tr>
<tr>
<td>3.3.4. ActiveDocs auto-fill of API keys</td>
<td>13</td>
</tr>
<tr>
<td>CHAPTER 4. ACTIVEDOC &amp; OAUTH</td>
<td>16</td>
</tr>
<tr>
<td>4.1. PREREQUISITES</td>
<td>16</td>
</tr>
<tr>
<td>4.2. CLIENT CREDENTIALS AND RESOURCE OWNER FLOWS</td>
<td>16</td>
</tr>
<tr>
<td>CHAPTER 5. PUBLISH ACTIVEDOC IN THE DEVELOPER PORTAL</td>
<td>20</td>
</tr>
<tr>
<td>CHAPTER 6. UPGRADE SWAGGER UI 2.1.3 TO 2.2.10</td>
<td>21</td>
</tr>
</tbody>
</table>
This guide provides information about features and functionalities to boost your Developer Portal.
PART I. OPENAPI SPECIFICATION (OAS)
CHAPTER 1. CREATING A NEW SERVICE BASED ON OAS

1.1. INTRODUCTION

This documentation outlines the features of OpenAPI 2.0 specification (OAS) in Red Hat 3scale 2.8 and provides steps to update an existing service or create a new one.

1.2. PREREQUISITES

- OpenAPI Specification (OAS)
- A 3scale 2.8 instance tenant credentials (token or provider_key)

1.3. FEATURES OF OPENAPI SPECIFICATION

NOTE

ActiveDocs are created/updated when importing OpenAPI (OAS)

- Service’s system_name can be passed as an option parameter and defaults to info.title field from OAS.
- Methods are created for each operation from the OAS.
  - Method names are taken from operation.operationId field.
- All existing mapping rules are deleted before importing a new API definition.
  - Methods will be not deleted if they exist before running the command.
- Mapping rules are created on each operation from the OAS.
- The OpenAPI definition resource can be provided by one of the following channels:
  - Filename in the available path
  - URL format - toolbox will try to download from given address.
  - Read from stdin standard input stream.

1.4. USING OPENAPI SPECIFICATION

NAME
openapi - Import API definition in OpenAPI specification

USAGE
3scale import openapi [opts] -d <dst> <spec>

DESCRIPTION
Using an API definition format like OpenAPI, import to your 3scale API

OPTIONS
-d --destination=<value> 3scale target instance.
1.4.1. Detecting OpenAPI definition from the filename path

The allowed formats are json and yaml. The format is automatically detected from filename extension.

```
$ 3scale import openapi -d <destination> /path/to/your/spec/file.[json|yaml|yml]
```

1.4.2. Detecting OpenAPI definition from a URL

The allowed formats are json and yaml. The format is automatically detected from URL’s path extension.

```
$ 3scale import openapi -d <destination> http[s]://domain/resource/path.[json|yaml|yml]
```

1.4.3. Detecting OpenAPI definition from stdin

The command line parameter for the OpenAPI resource is `-`. The allowed formats are json and yaml. The format is automatically detected internally with parsers.

```
$ tool_to_read_openapi_from_source | 3scale import openapi -d <destination> -
```
CHAPTER 2. ADDING SPECIFICATIONS TO 3SCALE

By the end of the section, you will have ActiveDocs set up for your API.

3scale offers a framework to create interactive documentation for your API.

With OpenAPI Specification (OAS) 2.0 (based on the OpenAPI Specification (OAS)) you will have a functional documentation for your API, which will help your developers explore, test and integrate with your API.

2.1. NAVIGATE TO SERVICE SPECIFICATIONS IN ACTIVEDOCS

Navigate to [your_API_name] → ActiveDocs in your Admin Portal. This will lead you to the list of your service specifications for your API (initially empty).

You can add as many service specifications as you want. Typically, each service specification corresponds to one of your APIs. For example, 3scale has specifications for each 3scale API, such as Service Management, Account Management, Analytics, and Billing.

2.2. CREATE A SERVICE SPECIFICATION

When you add a new service spec, you will have to provide:

- Name
- System name (required to reference the Service specification from the Developer Portal)
- Whether you want the specification to be public or not
- A description that is only meant for your own consumption
- API JSON spec, which you can see in the figure below.

The API JSON specification is the "secret ingredient" of ActiveDocs.

You must generate the specification of your API according to the specification proposed by OpenAPI Specification (OAS). In this tutorial we assume that you already have a valid OpenAPI Specification (OAS) 2.0-compliant specification of your API.
2.3. WORKING WITH YOUR FIRST ACTIVEDOC

Once you have added your first ActiveDoc, you can see it listed in [your_API_name] → ActiveDocs. You can edit it as necessary, delete it, or switch it from public to private. You can also detach it from your API or attach it to any other API. You can see all your ActiveDocs (attached to an API or not) in Audience → Developer Portal → ActiveDocs.

You can also preview what your ActiveDocs looks like by clicking on the name you gave the service specification (in the example it was called it Pet Store). You can do this even if the specification is not public yet.

This is what your ActiveDoc will look like:
ActiveDocs

Preview Service Spec (2.0)

Available Actions:
- Publish
- Edit
- Delete

Pet Store

A sample API that uses a pet store as an example to demonstrate API specification.

default

<table>
<thead>
<tr>
<th>Method</th>
<th>Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/user-key</td>
</tr>
<tr>
<td>GET</td>
<td>/app-id</td>
</tr>
<tr>
<td>GET</td>
<td>/client-id</td>
</tr>
</tbody>
</table>

[ BASE URL: , API VERSION: 1.0.0 ]
CHAPTER 3. CREATE AN OAS SPECIFICATION

This section will help you to create an OpenAPI Specification (OAS) 2.0-compliant specification for your REST API, which is required to power ActiveDocs on your Developer Portal. If you only want to read the code, all the examples are on OAS Petstore example source code.

3.1. ABOUT OPENAPI SPECIFICATION (OAS)

3scale ActiveDocs are based on the specification of RESTful web services called Swagger (from Wordnik). This example is based on the Extended OpenAPI Specification Petstore example and draws all the specification data from the OpenAPI Specification 2.0 specification document.

OAS is not only a specification. It also provides a full feature framework around it:

1. Servers for the specification of the resources in multiple languages (NodeJS, Scala, and others).
2. A set of HTML/CSS/Javascript assets that take the specification file and generate the attractive UI.
3. A OAS codegen project, which allows generation of client libraries automatically from a Swagger-compliant server. Support to create client-side libraries in a number of modern languages.

3.2. 3SCALE ACTIVEDOCS AND OAS

ActiveDocs is not an OAS replacement but an instantiation of it. With ActiveDocs, you don’t have to run your own OAS server or deal with the UI components of the interactive documentation. The interactive documentation is served and rendered from your 3scale Developer Portal.

The only thing you need to do is to build a Swagger-compliant specification of your API, add it on your Admin Portal, and the interactive documentation will be available. Your developers will be able to launch requests against your API through your Developer Portal.

If you already have a Swagger-compliant specification of your API, you can add it in your Developer Portal (see the tutorial on the ActiveDocs configuration).

3scale extended the OAS specification in several ways to accommodate certain features that were needed for our own interactive API documentation:

- Auto-fill of API keys
- OAS proxy to allow calls to non-CORS enabled APIs

3.3. CREATING THE SPECIFICATION OF YOUR API

We recommend that you first read the specification from the original source. See the OAS Specification.

On the OAS site there are multiple examples of specifications. If you like to learn by example, you can follow the example of the Petstore API by the OAS API Team.

3.3.1. Learning by example: the Petstore API

The Petstore API is an extremely simple API. It is meant as a learning tool, not for production.
The Petstore API is composed of 4 methods:

- **GET /api/pets** - returns all pets from the system
- **POST /api/pets** - creates a new pet in the store
- **GET /api/pets/{id}** - returns a pet based on a single ID
- **DELETE /api/pets/{id}** - deletes a single pet based on the ID

Because Petstore is integrated with 3scale API Management, you have to add an additional parameter for authentication – for example, the standard User Key authentication method (there are others) sent in the headers.

You need to add the parameters:

`user_key: {user_key}`

The `user_key` will be sent by the developers in their requests to your API. The developers will obtain those keys on your Developer Portal. On receiving the key, you must to perform the authorization check against 3scale using the Service Management API.

For your developers, the documentation of your API represented in cURL calls would look like this:

```bash
curl -X GET "http://example.com/api/pets?tags=TAGS&limit=LIMIT" -H "user_key: {user_key}"
curl -X POST "http://example.com/api/pets" -H "user_key: {user_key}" -d "{"name": "NAME", "tag": "TAG", "id": ID }"
curl -X GET "http://example.com/api/pets/{id}" -H "user_key: {user_key}"
curl -X DELETE "http://example.com/api/pets/{id}" -H "user_key: {user_key}"
```

However, if you want the documentation to look like the OAS Petstore Documentation, you must create a Swagger-compliant specification like the associated Petstore `swagger.json` file. You can use this specification out-of-the-box to test your ActiveDocs. But remember that this is not your API.

### 3.3.2. More on the OAS specification

The OAS specification relies on a resource declaration that maps to a hash encoded in JSON. Take the Petstore `swagger.json` file as an example and go step by step.

#### 3.3.2.1. OAS object

This is the root document object for the API specification. It lists all the highest level fields.

**WARNING**

The host must be a domain and not an IP address. 3scale will proxy the requests made against your Developer Portal to your host and render the results. This requires your host and basePath endpoint to be whitelisted by us for security reasons. You can only declare a host that is your own. 3scale reserves the right to terminate your account if we detect that you're proxying a domain that does not belong to you. This means that local host or any other wildcard domain will not work.
3.3.2.2. Info object

The Info object provides the metadata about the API. This will be presented in the ActiveDocs page.

3.3.2.3. Paths object

The paths object holds the relative paths to the individual endpoints. The path is appended to the basePath to construct the full URL. The paths may be empty, due to ACL constraints.

Parameters that are not objects use primitive data types. In Swagger, these are based on the types supported by the JSON-Schema Draft 4. There is an additional primitive data type “file” but it will work only if the API endpoint has CORS enabled (so the upload won't go through api-docs gateway). Otherwise, it will get stuck on the gateway level.

Supported datatypes

Currently OAS supports the following DataTypes:

- integer with possible formats: int32 and int64. Both formats are signed.
- number with possible formats: float and double
- string with possible formats (besides the unformatted version): byte, date, date-time, password
- boolean

3.3.3. OAS design and editing tools

The following tools are useful for designing and editing your OpenAPI specification:

- The open source Apicurio Studio enables you to design and edit your OpenAPI-based APIs in a web-based application. Apicurio Studio provides a design view so you do not need detailed knowledge of the OpenAPI specification. The source view enables expert users to edit directly in YAML or JSON. For more details, see Getting Started with Apicurio Studio.
  Red Hat also provides a lightweight version of Apicurio Studio named API Designer, which is included with Fuse Online on OpenShift. For more details, see Developing and Deploying API Provider Integrations.

- The JSON Editor Online is useful if you are very familiar with the JSON notation. It gives a pretty format to compact JSON and provides a JSON object browser.

- The Swagger Editor enables you to create and edit your OAS API specification written in YAML in your browser and preview it in real time. You can also generate a valid JSON specification, which you can upload later in your 3scale Admin Portal. You can use the live demo version with limited functionality or deploy your own OAS Editor.

3.3.4. ActiveDocs auto-fill of API keys

Auto-fill of API keys is a useful extension to the OAS specification in 3scale ActiveDocs. In the parameters, you can define the x-data-threescale-name field with the following values depending on your API authentication mode:

- **user_keys**: Returns the user keys for applications of the services that use API key authentication only.
- **app_ids**: Returns the IDs for applications of the services that use App ID/App Key (OAuth and OpenID Connect are also supported for backwards compatibility).

- **app_keys**: Returns the keys for applications of services that use App ID/App Key (OAuth and OpenID Connect are also supported for backwards compatibility).

- **client_ids**: Returns the client IDs for applications of the services that use OAuth/OpenID Connect authentication only.

- **client_secrets**: Returns the client secrets for applications of the services that use OAuth/OpenID Connect authentication only.

### API key authentication example

The following example shows using `"x-data-threescale-name": "user_keys"` for API key authentication only:

```
"parameters": [
  {
    "name": "user_key",
    "description": "Your access API Key",
    "type": "string",
    "in": "query",
    "x-data-threescale-name": "user_keys",
    "required": true
  },
]
```

### App ID/App Key authentication example

For App ID/App Key authentication mode, specify `"x-data-threescale-name": "app_ids"` for the parameter that represents the application ID, and `"x-data-threescale-name": "app_keys"` for the parameter that represents the application key.

When you have declared your parameters, ActiveDocs will automatically prompt the ActiveDocs user to log in to the Developer Portal to get their keys as shown in the following screenshot:

![Screenshot of API key authentication example]

If the user is already logged in, ActiveDocs will show the latest five keys that could be relevant for them so that they can test right away without having to copy and paste their keys.
NOTE

The `x-data-threescale-name` field is an extension to the OAS specification that will be ignored outside the domain of ActiveDocs.
CHAPTER 4. ACTIVEDOCS & OAUTH

By the end of this tutorial, you will have a set of ActiveDocs that will allow your users to easily test and call your OAuth-enabled API from one place.

If you have an OAuth-enabled API, you will want to show off its capabilities to your users. How can you do this using ActiveDocs? While this is a bit trickier than usual, it’s still possible.

4.1. PREREQUISITES

Before you begin, you’ll need to have a Red Hat Single Sign-On instance set up, and OpenID Connect integration configured. See OpenID Connect integration documentation for information on how to set it up. Additionally, you will need to be familiar with how to set up ActiveDocs – see Add ActiveDocs and Create a (Swagger) spec.

4.2. CLIENT CREDENTIALS AND RESOURCE OWNER FLOWS

This first example is for an API using the OAuth 2.0 client credentials flow. This API accepts any path and returns information about the request (path, request parameters, headers, etc.). The Echo API is only accessible using a valid access token. Users of the API are only able to call it once they have exchanged their credentials (client_id and client_secret) for an access token.

In order for users to be able to call the API from ActiveDocs, they will need to request an access token. Since this is just a call to an OAuth authorization server, you can create an ActiveDocs spec for the OAuth token endpoint. This will allow you to call this endpoint from within ActiveDocs. In this case, for a client credentials flow, the Swagger JSON spec looks like this:

```json
{
    "swagger": "2.0",
    "info": {
        "version": "v1",
        "title": "OAuth for Echo API",
        "description": "OAuth2.0 Client Credentials Flow for authentication of our Echo API.",
        "contact": {
            "name": "API Support",
            "url": "http://www.swagger.io/support",
            "email": "support@swagger.io"
        }
    },
    "host": "red-hat-sso-instance.example.com",
    "basePath": "/auth/realms/realm-example/protocol/openid-connect",
    "schemes": [ "http"
    ],
    "paths": {
        "/token": {
            "post": {
                "description": "This operation returns the access token for the API. You must call this before calling any other endpoints.",
                "operationId": "oauth",
                "parameters": [ {
                    "name": "client_id",
                    "description": "Your client id",
                }]
            }
        }
    }
}
```
For a resource owner OAuth flow, you’ll probably also want to add parameters for a username and password, as well as any other parameters that you require in order to issue an access token. For this client credentials flow example, you’re just sending the `client_id` and `client_secret` – which can be populated from the 3scale values for signed-in users – as well as the `grant_type`.

Then in the ActiveDocs spec for our Echo API we need to add the `access_token` parameter instead of the `client_id` and the `client_secret`.

```json
{
  "swagger": "2.0",
  "info": {
    "version": "v1",
    "title": "Echo API",
    "description": "A simple API that accepts any path and returns information about the request",
    "contact": {
      "name": "API Support",
      "url": "http://www.swagger.io/support",
      "email": "support@swagger.io"
    }
  },
  "host": "echo-api.3scale.net",
  "basePath": "/v1/words",
  "schemes": ["http"
```
You can then include your ActiveDocs in the Developer Portal as usual. In this case, since you want to specify the order in which they display to have the OAuth endpoint first, it looks like this:

```
{% active_docs version: "2.0" services: "oauth" %}

<script type="text/javascript">
$(function () {
  window.swaggerUi.load(); // <-- loads first swagger-ui

  // do second swagger-ui

  var url = "/swagger/spec/echo-api.json";
  window.anotherSwaggerUi = new SwaggerUi({
    url: url,
    dom_id: "another-swagger-ui-container",
    supportedSubmitMethods: ['get', 'post', 'put', 'delete', 'patch'],
    onComplete: function(swaggerApi, swaggerUi) {
      $("#another-swagger-ui-container pre code").each(function(i, e) {hljs.highlightBlock(e)});
    },
    onFailure: function(data) {
      log("Unable to Load Echo-API-SwaggerUI");
    }
  });
});
</script>
```
}`
docExpansion: "list",
transport: function(httpClient, obj) {
    log("[swagger-ui]>>> custom transport.");
    return ApiDocsProxy.execute(httpClient, obj);
}
});

window.anotherSwaggerUi.load();

});
</script>
CHAPTER 5. PUBLISH ACTIVEDOCS IN THE DEVELOPER PORTAL

By the end of this tutorial, you will have published your ActiveDocs in your developer portal.

Once you are happy with your Swagger, and you have added it to 3scale, it is time to make it public and link it on your Developer Portal so it can be used by your API developers.

Add the following snippet to the content of any page of your Developer Portal. This must be done through the CMS of your Developer Portal. Note that SERVICE_NAME should be the system name of the service specification, pet_store in the example.

```html
<h1>Documentation</h1>
<p>Use our live documentation to learn about Echo API</p>
{% active_docs version: "2.0" services: "SERVICE_NAME" %}
{% cdn_asset /swagger-ui/2.2.10/swagger-ui.js %} {% cdn_asset /swagger-ui/2.2.10/swagger-ui.css %} {% include 'shared/swagger_ui' %}
<script type="text/javascript">
$(function () {
  // you have access to swaggerUi.options object to customize its behaviour
  // such as setting a different docExpansion mode
  window.swaggerUi.options['docExpansion'] = 'none';
  // or even getting the swagger specification loaded from a different url
  window.swaggerUi.options['url'] = "http://petstore.swagger.io/v2/swagger.json";
  window.swaggerUi.load();
});
</script>
```

**NOTE**

- You can specify only one service on one page. If you want to display multiple specifications, the best way is to do it on different pages.

- This snippet requires jQuery, which is typically already included in the main layout of your Developer Portal. If you remove it from there, make sure you add the jQuery dependency on the page with ActiveDocs.

- Make sure you have Liquid tags enabled on the CMS page.

- The version used in the Liquid tag `{{ '{% active_docs version: "2.0" ' }}%` should correspond to that of the Swagger spec.

- If you would like to fetch your specification from an external source, change the JavaScript code as follows:

  ```javascript
  $(function () {
    window.swaggerUi.options['url'] = "SWAGGER_JSON_URL";
    window.swaggerUi.load();
  });
  ```

You can see an example in the snippet in on line 14. Just make sure that this line is not inside the comments block.
CHAPTER 6. UPGRADE SWAGGER UI 2.1.3 TO 2.2.10

If you are using a version of 3scale that contains Swagger UI 2.1.3, you can upgrade to Swagger UI version 2.2.10.

Previous implementations of Swagger UI 2.1.3 in the 3scale developer portal rely on the presence of a single {% active_docs version: "2.0" %} liquid tag in the Documentation page. With the introduction of support for Swagger 2.2.10 in 3scale, the implementation method changes to multiple cdn_asset and include liquid tags.

NOTE

Previous versions of Swagger UI in 3scale will continue to be called using the legacy active_docs liquid tag method.

Perform the following steps to upgrade Swagger UI 2.1.3 to 2.2.10:

1. Log in to your 3scale AMP admin portal

2. Navigate to the Developer Portal → Documentation page, or the page in which you want to update your Swagger UI implementation

3. In the code pane replace the {% active_docs version: "2.0" %} liquid tag with the following assets with the cdn_asset liquid tag and the new partial shared/swagger_ui:

{% cdn_asset /swagger-ui/2.2.10/swagger-ui.js %}
{% cdn_asset /swagger-ui/2.2.10/swagger-ui.css %}
{% include 'shared/swagger_ui' %}

4. By default, Swagger UI loads the ActiveDocs specification published in APIs > ActiveDocs. Load a different specification by adding the following window.swaggerUi.options line before the window.swaggerUi.load(); line, where `<SPEC_SYSTEM_NAME>` is the system name of the specification you want to load:

window.swaggerUi.options['url'] = "{{provider.api_specs.<SPEC_SYSTEM_NAME>.url}}";