



Red Hat 3scale API Management 2.7

Providing APIs in the Developer Portal

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

Red Hat 3scale API Management 2.7 Providing APIs in the Developer Portal

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

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

This guide documents the uses of the Developer Portal on Red Hat 3scale API Management 2.7.

Table of Contents

PREFACE	3
PART I. OPENAPI SPECIFICATION (OAS)	4
CHAPTER 1. CREATING A NEW SERVICE BASED ON OAS	5
1.1. INTRODUCTION	5
1.2. PREREQUISITES	5
1.3. FEATURES OF OPENAPI SPECIFICATION	5
1.4. USING OPENAPI SPECIFICATION	5
1.4.1. Detecting OpenAPI definition from the filename path	6
1.4.2. Detecting OpenAPI definition from a URL	6
1.4.3. Detecting OpenAPI definition from stdin	6
PART II. API DOCUMENTATION	7
CHAPTER 2. ADDING SPECIFICATIONS TO 3SCALE	8
2.1. NAVIGATE TO SERVICE SPECIFICATIONS IN ACTIVEDOCS	8
2.2. CREATE A SERVICE SPECIFICATION	8
2.3. WORKING WITH YOUR FIRST ACTIVEDOC	9
CHAPTER 3. CREATE AN OAS SPEC	11
3.1. ABOUT OPENAPI SPECIFICATION (OAS)	11
3.2. 3SCALE ACTIVEDOCS AND OAS	11
3.3. CREATING THE SPECIFICATION OF YOUR API	11
3.3.1. Learning by example: the Petstore API	11
3.3.2. More on the OAS specification	12
3.3.2.1. OAS object	12
3.3.2.2. Info object	13
3.3.2.3. Paths object	13
3.3.3. Useful tools	13
3.3.3.1. Extension to the OAS specification: auto-fill of API keys	13
CHAPTER 4. ACTIVEDOCS & OAUTH	16
4.1. PREREQUISITES	16
4.2. CLIENT CREDENTIALS AND RESOURCE OWNER FLOWS	16
CHAPTER 5. PUBLISH ACTIVEDOCS IN THE DEVELOPER PORTAL	20
CHAPTER 6. UPGRADE SWAGGER UI 2.1.3 TO 2.2.10	21

PREFACE

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

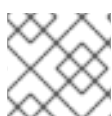
1.1. INTRODUCTION

This documentation outlines the features of OpenAPI 2.0 specification (OAS) in Red Hat 3scale 2.7 and provides steps to update an existing service or create a new one.

1.2. PREREQUISITES

- OpenAPI Specification (OAS)
- A 3scale 2.7 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.

```
Format: "http[s]://<authentication>@3scale_domain"
```

```
-t --target_system_name=<value>      Target system name
```

OPTIONS FOR IMPORT

```
-c --config-file=<value>             3scale toolbox
                                     configuration file
                                     (default:
                                     $HOME/.3scalerc.yaml)
-h --help                             show help for this command
-k --insecure                          Proceed and operate even
                                     for server connections
                                     otherwise considered
                                     insecure
-v --version                            Prints the version of this
                                     command
```

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> -
```

PART II. API DOCUMENTATION

CHAPTER 2. ADDING SPECIFICATIONS TO 3SCALE

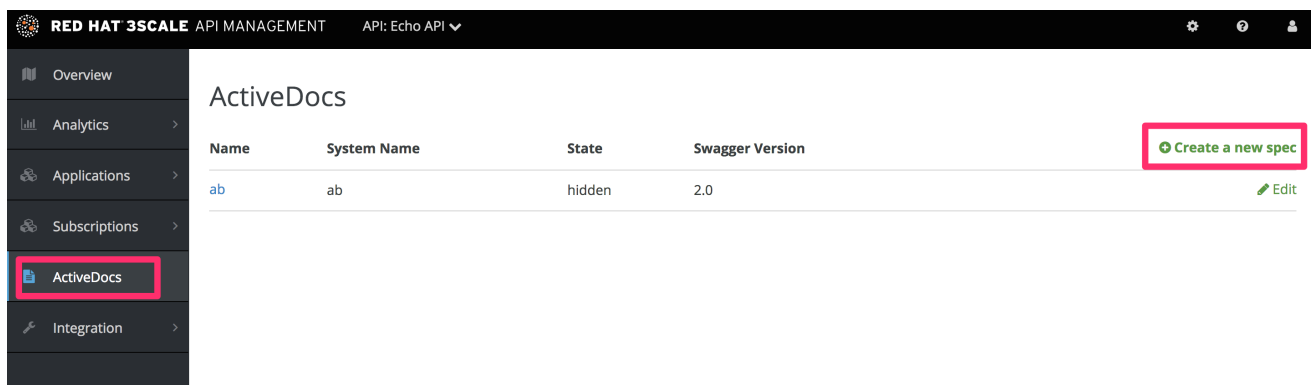
By the end of the section, you will have ActiveDocs set up for your API.

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.

RED HAT 3SCALE API MANAGEMENT API: Echo API ▾

Overview
Analytics >
Applications >
Subscriptions >
ActiveDocs
Integration >

ActiveDocs

ActiveDocs: New Service Spec

Name

System name

Only ASCII letters, numbers, dashes and underscores are allowed.
Warning: With ActiveDocs 1.2 the API will be described in your developer portal as System name: Description

Publish?

Description

API JSON Spec

1

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**

RED HAT 3SCALE API MANAGEMENT API: Echo API ▾


Overview
Analytics >
Applications >
Subscriptions >
ActiveDocs
Integration >

ActiveDocs

Name	System Name	State	Swagger Version
Pet Store	pet_store	visible	2.0

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:

 **RED HAT 3SCALE** API MANAGEMENT API: Echo API ▾

- Overview
- Analytics >
- Applications >
- Subscriptions >
- ActiveDocs**
- Integration >

ActiveDocs

Preview Service Spec (2.0)

[Publish](#) | [Edit](#) | [Delete](#)

Pet Store

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

default

POST	/user-key
GET	/app-id
GET	/client-id

[BASE URL: , API VERSION: 1.0.0]

CHAPTER 3. CREATE AN OAS SPEC

This section will help you to create a OpenAPI Specification 2.0-compliant (OAS) specification for your RESTful API, which is required to power ActiveDocs on your Developer Portal. If you only would like 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/Javascripts 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 a 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 original specification from the original source: 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

The Petstore API is integrated with 3scale, and for this reason you must add an additional parameter for authentication. For example, with the User Key authentication method, the parameter is sent in the header. For information about other authentication methods, see [Authentication patterns](#).

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:

```
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. You can learn more in the next section.

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
- plain string
- string with possible formats: byte, date, date-time, password and binary
- boolean

3.3.3. Useful tools

The [JSON Editor Online](#) is useful if you are very familiar with the JSON notation. It gives a pretty format to compact JSON, and it also provides a JSON object browser.

The [OAS Editor](#) is another useful tool. This 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.

3.3.3.1. Extension to the OAS specification: 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

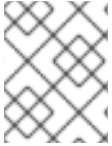
For App ID/App Key authentication mode, specify **"x-data-threescale-name": "app_ids"** for the parameter that represents the application ID, and **"x-data-threescale-name": "app_keys"** for the parameter that represents the application key.

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:

PARAMETER	VALUE	DESCRIPTION
word	<input type="text" value="awesome"/>	The word whose sentiment is returned
app_id	<input type="text"/>	<div style="border: 2px solid blue; padding: 5px; text-align: center;"> Sign in to you account for quick access to useful values. </div>
app_key	<input type="text"/>	

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.

PARAMETER	VALUE	DESCRIPTION
word	<input type="text" value="awesome"/>	The word whose sentiment is returned
app_id	<input type="text"/>	Latest 5 applications (across all accounts and services) Sample App cd6bac2e
app_key	<input type="text"/>	
<input type="button" value="Send Request"/>		



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 is still possible.

4.1. PREREQUISITES

Before you begin, you need 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 [Adding specifications to 3scale](#) 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:

```
{
  "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",
```

```

    "type": "string",
    "in": "query",
    "required": true
  },
  {
    "name": "client_secret",
    "description": "Your client secret",
    "type": "string",
    "in": "query",
    "required": true
  },
  {
    "name": "grant_type",
    "description": "OAuth2 Grant Type",
    "type": "string",
    "default": "client_credentials",
    "required": true,
    "in": "query",
    "enum": [
      "client_credentials",
      "authorization_code",
      "refresh_token",
      "password"
    ]
  }
]
}
}
}
}
}
}
}

```

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`.

```

{
  "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"
  ],
}

```

```

    "produces": [
      "application/json"
    ],
    "paths": {
      "/{word}.json": {
        "get": {
          "description": "This operation returns information about the request (path, request parameters,
headers, etc.),
          "operationId": "wordsGet",
          "summary": "Returns the path of a given word",
          "parameters": [
            {
              "name": "word",
              "description": "The word related to the path",
              "type": "string",
              "in": "path",
              "required": true
            },
            {
              "name": "access_token",
              "description": "Your access token",
              "type": "string",
              "in": "query",
              "required": true
            }
          ]
        }
      }
    }
  }
}

```

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");
    }
  });
}

```

```
    },  
    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.

```
<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 () {
  {% comment %}
  // 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";
  {% endcomment %}
  window.swaggerUi.load();
});
</script>
```

These are some additional considerations when publishing ActiveDocs in the Developer Portal:

- 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 included by default in the main layout of your Developer Portal. If you remove the jQuery dependency from the main layout, you must add this dependency on the page containing 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 want to fetch your specification from an external source, change the JavaScript code as follows:

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

Note that the line containing the source of the specification, `window.swaggerUi.options['url'] = "SWAGGER_JSON_URL";`, is outside of 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}}";
```