Red Hat 3scale 2-saas

API Documentation

Make your API documentation clear, intuitive, and user friendly with 3scale ActiveDocs.

Last Updated: 2019-03-19

Red Hat 3scale 2-saas API Documentation

Make your API documentation clear, intuitive, and user friendly with 3scale ActiveDocs.

Legal Notice

Abstract

This guide provides instruction on creating API documentation with Red Hat 3scale 2-saas.

Table of Contents

CHAPTER 1. ADD YOUR SPEC TO 3SCALE

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

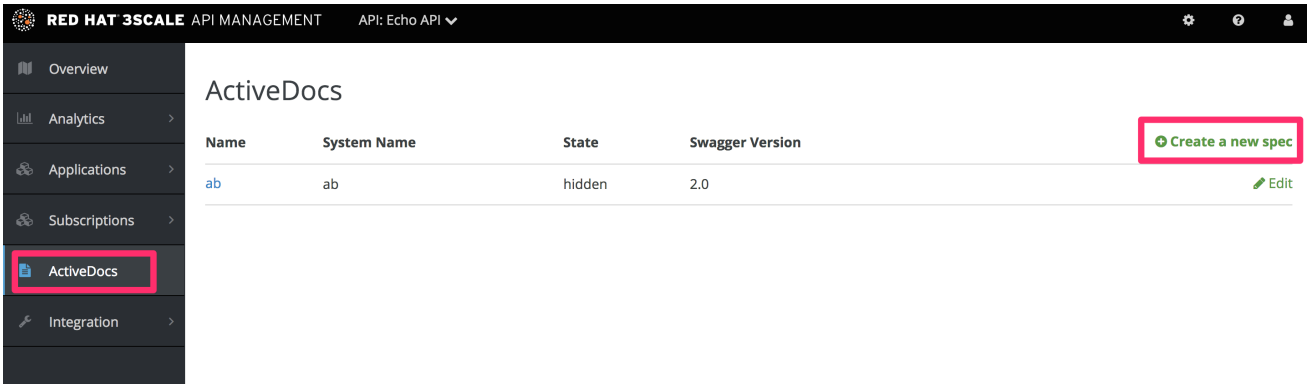
3scale offers a framework to create interactive documentation for your API.

With Swagger 2.0 (based on the Swagger Spec) you will have functional, attractive documentation for your API, which will help your developers to explore, to test and integrate with your API.

Every minute spent making your developers happy is a minute well invested on your API.

1.1. STEP 1: NAVIGATE TO SERVICE SPECS IN ACTIVEDOCS

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



You can add as many service specs as you want. Typically, each service spec corresponds to one of your APIs. For instance, at 3scale we have four different specs, one for each API of 3scale: Service Management, Account Management, Analytics, and Billing.

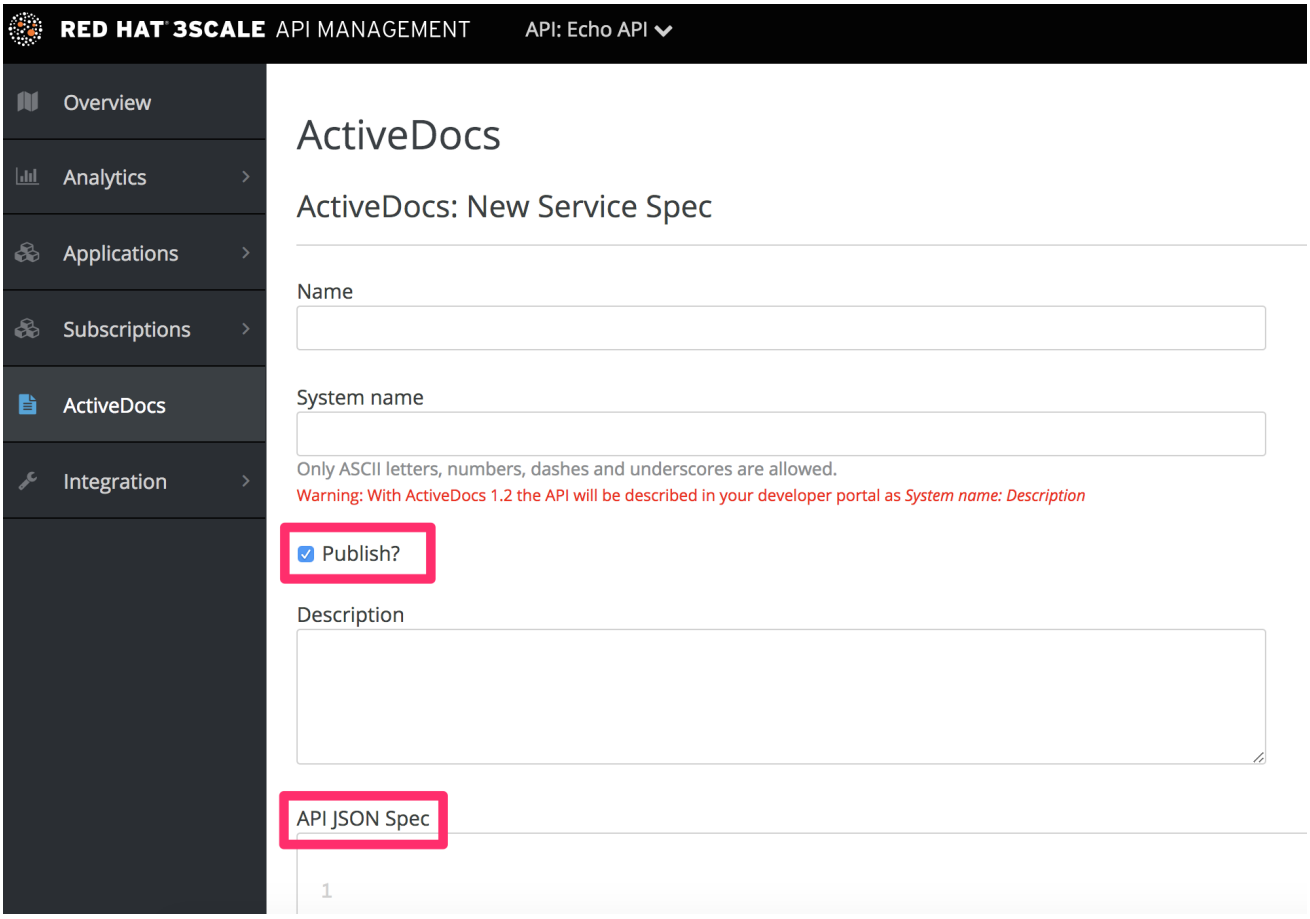
1.2. STEP 2: CREATE A SERVICE SPEC

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

- Name
- System name (required to reference the Servcie Spec from the Developer Portal)
- Whether you want the spec 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.

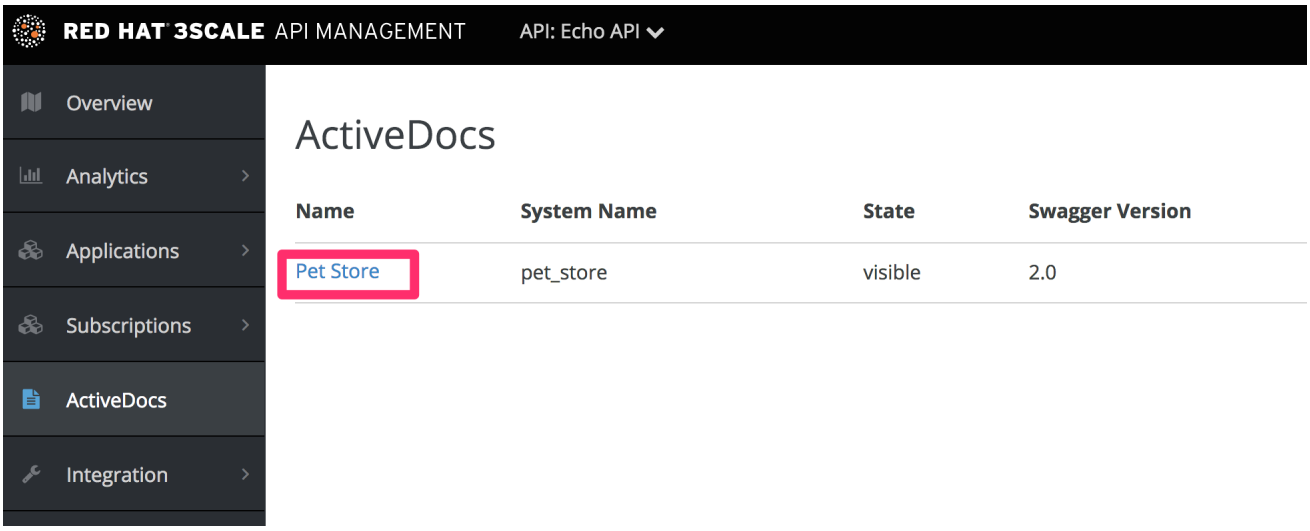
The API JSON spec is the "secret ingredient" of ActiveDocs.

You must generate the specification of your API according to the spec proposed by Swagger. In this tutorial we assume that you already have a valid Swagger 2.0-compliant specification of your API.



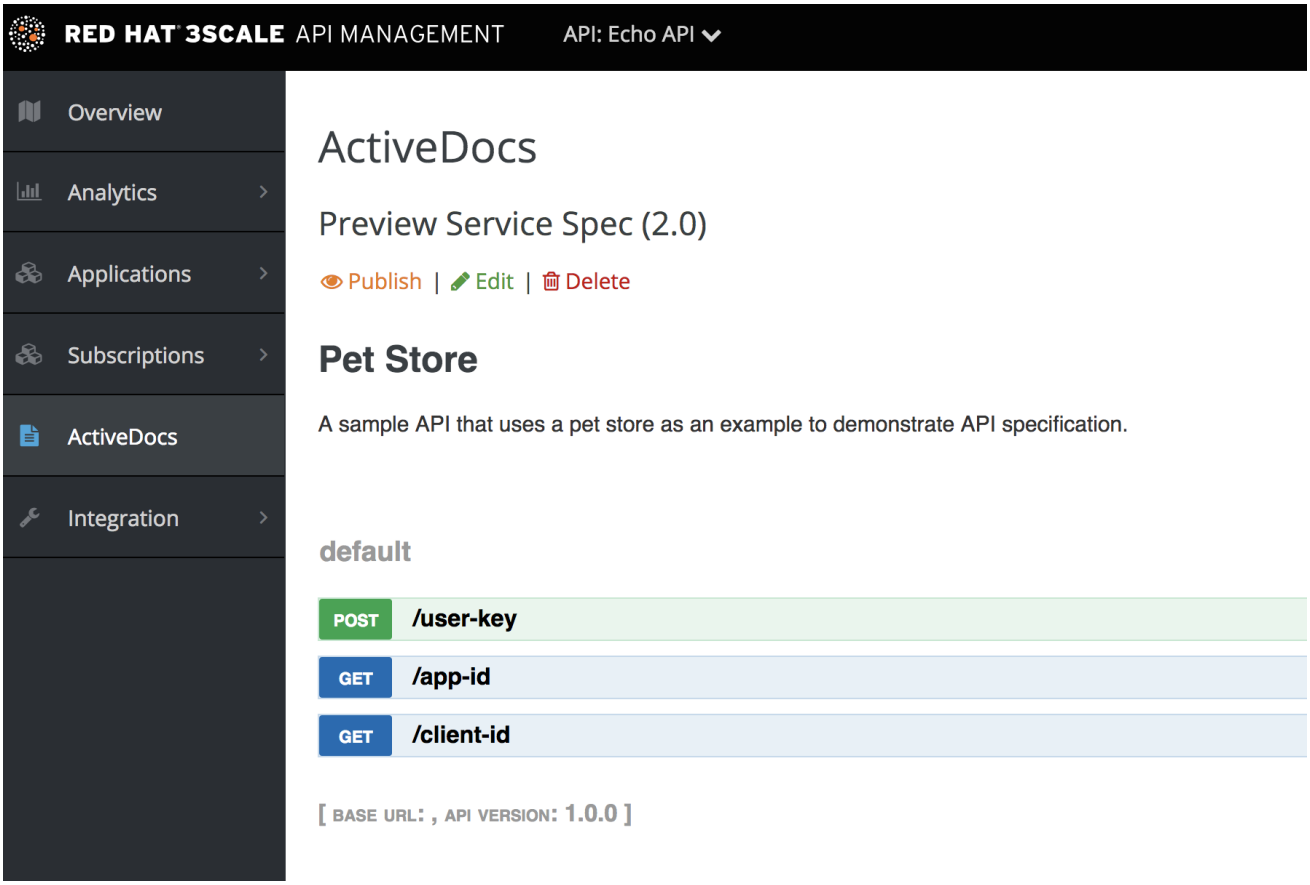
1.3. STEP 3: YOU GOT YOUR FIRST SWAGGER 2.0!

Once you've 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 spec (in the example it was called it Pet Store). You can do this even if the spec 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 2. CREATE A (SWAGGER) SPEC

This section will help you to create a Swagger 2.0-compliant spec 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 Swagger Petstore example source code.

2.1. CREDIT WHERE CREDIT IS DUE

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

Swagger 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 Swagger 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.

2.2. 3SCALE ACTIVEDOCS AND SWAGGER

ActiveDocs is not a Swagger replacement but an instantiation of it. With ActiveDocs, you don't have to run your own Swagger 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 all set. Your developers will be able to launch requests against your API through your Developer Portal.

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

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

1. Auto-fill of API keys
2. Swagger proxy to allow calls to non-CORS enabled APIs

2.3. READY TO CREATE THE SPEC OF YOUR API?

We recommend that you read first the original spec from the original source: the Swagger Specification

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

2.3.1. 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.

Upon receiving the key, you will have to do the authorization check against 3scale using the Service Management API. (This starts to get into the integration. Check the /get-started/fast-track-integration[guide on integration] and API lifecycle for more details.)

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

However, if you want the documentation to look awesome like the Swagger Petstore Documentation, you have to create the following Swagger-compliant spec:

You can use this spec out-of-the-box to test your ActiveDocs. But remember that this is not your API.

At first it might look a bit cumbersome, but the Swagger spec is not complex at all. Learn more in the next section.

2.3.2. More on the Swagger specification

The Swagger specification relies on a resource declaration that maps to a hash encoded in JSON. Take the above petstore3scale.json as example an go step by step…

2.3.3. Swagger 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.

2.3.3.1. Info object

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

2.3.3.2. Paths object

The paths object holds the relative paths to the individual endpoints. The path is appended to the basePath in order 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.

Currently Swagger supports the following *dataTypes*:

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

2.3.4. Useful tools

The JSON Editor Online is quite good. It gives a pretty format to compact JSON, and it also provides a browser of the JSON object. We really recommend it if you are not well versed with the JSON notation.

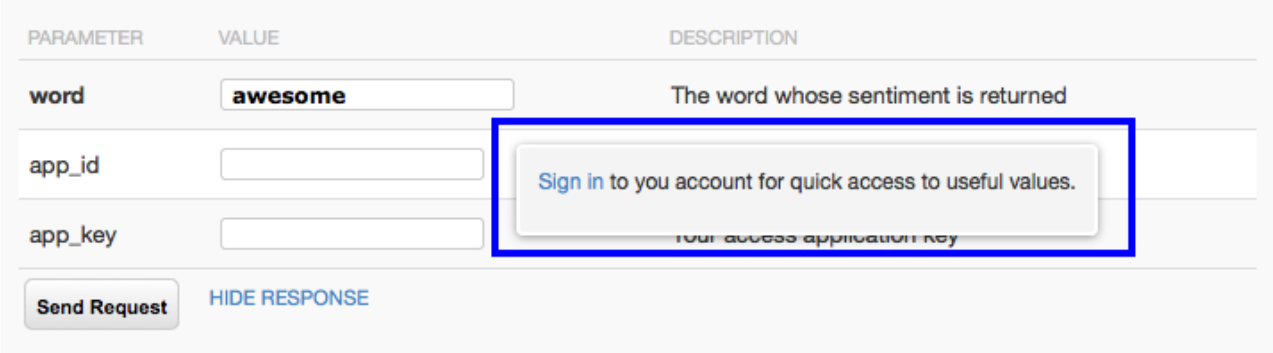
Another great tool is the Swagger Editor. It lets you create and edit your Swagger API specification written in YAML inside your browser and preview it in real time. You can also generate a valid JSON spec, which you can upload later in your 3scale Admin Portal. You can either use the live demo version with limited functionality or deploy your own Swagger Editor.

2.3.4.1. Extension to the Swagger spec: Auto-fill of API keys

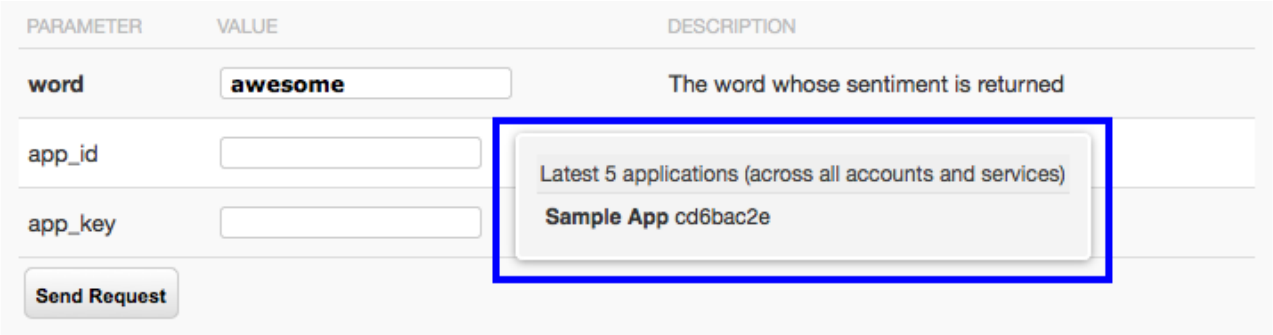
A very useful extension to the Swagger spec of 3scale's ActiveDocs is the auto-fill of the API keys. On the parameters, you can define the field **`x-data-threescale-name`** with values **`app_ids`**, **`app_keys`** or **`user_keys`** depending on the authentication mode your API is in.

For instance, for the authentication mode *App ID/ App Key* you might want to declare **`"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. Just like in the following snippet:

If you do so, ActiveDocs will automatically prompt the user of the ActiveDocs to log in to the Developer Portal to get their keys as shown in the screenshot below:



If the user is already logged in, it 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 around.



The field **`x-data-threescale-name`** is an extension to the Swagger spec which will be ignored outside the domain of ActiveDocs.

CHAPTER 3. 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.

3.1. 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.

3.2. 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 Credentails 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>
```

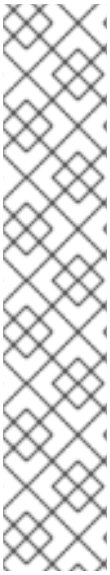
CHAPTER 4. PUBLISH YOUR ACTIVEDOCS IN YOUR DEVELOPER PORTAL

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

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

You'll have to 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 spec, 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" %}
<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>
```



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:

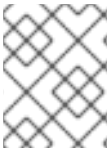
```
$(function () {
 window.swaggerUi.options['url'] = "SWAGGER_JSON_URL";
 window.swaggerUi.load();
});
```

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 5. 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}}";
```