Red Hat 3scale API Management 2.5

Using the Developer Portal

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

Last Updated: 2020-01-28

Red Hat 3scale API Management 2.5 Using the Developer Portal

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

Legal Notice

Abstract

This guide documents the uses of the Developer Portal on Red Hat 3scale API Management 2.5.

Table of Contents

PART I. API DOCUMENTATION

CHAPTER 1. ADDING SPECIFICATIONS TO 3SCALE

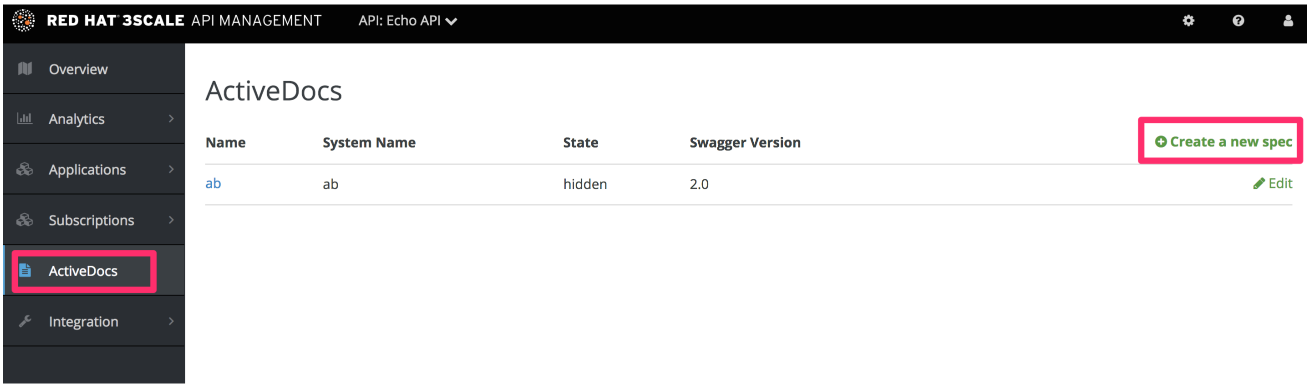
By the end of the section, you will have ActiveDocs set up for your API.

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.

1.1. 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).



Name	System Name	State	Swagger Version
ab	ab	hidden	2.0

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.

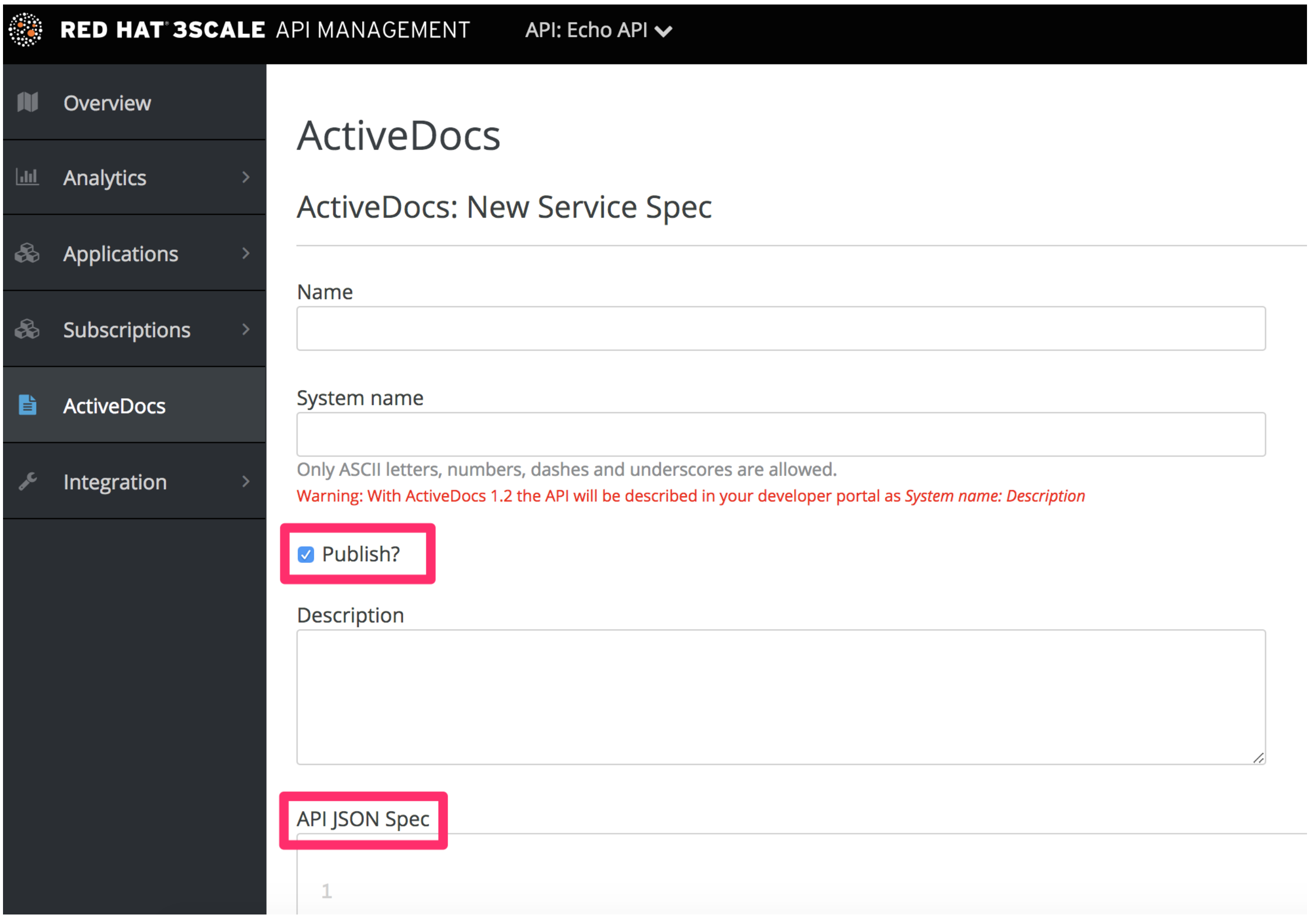
1.2. CREATE A SERVICE SPECIFICATION

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

- Name
- System name (required to reference the Servcie 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.



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*

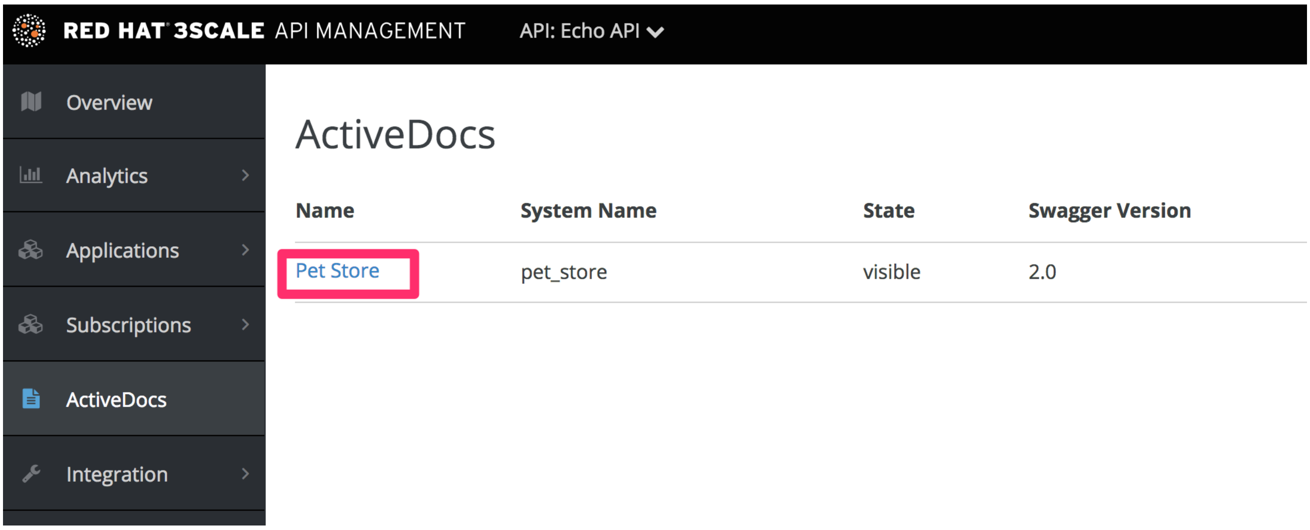
☑ Publish?

Description

API JSON Spec

1.3. 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**

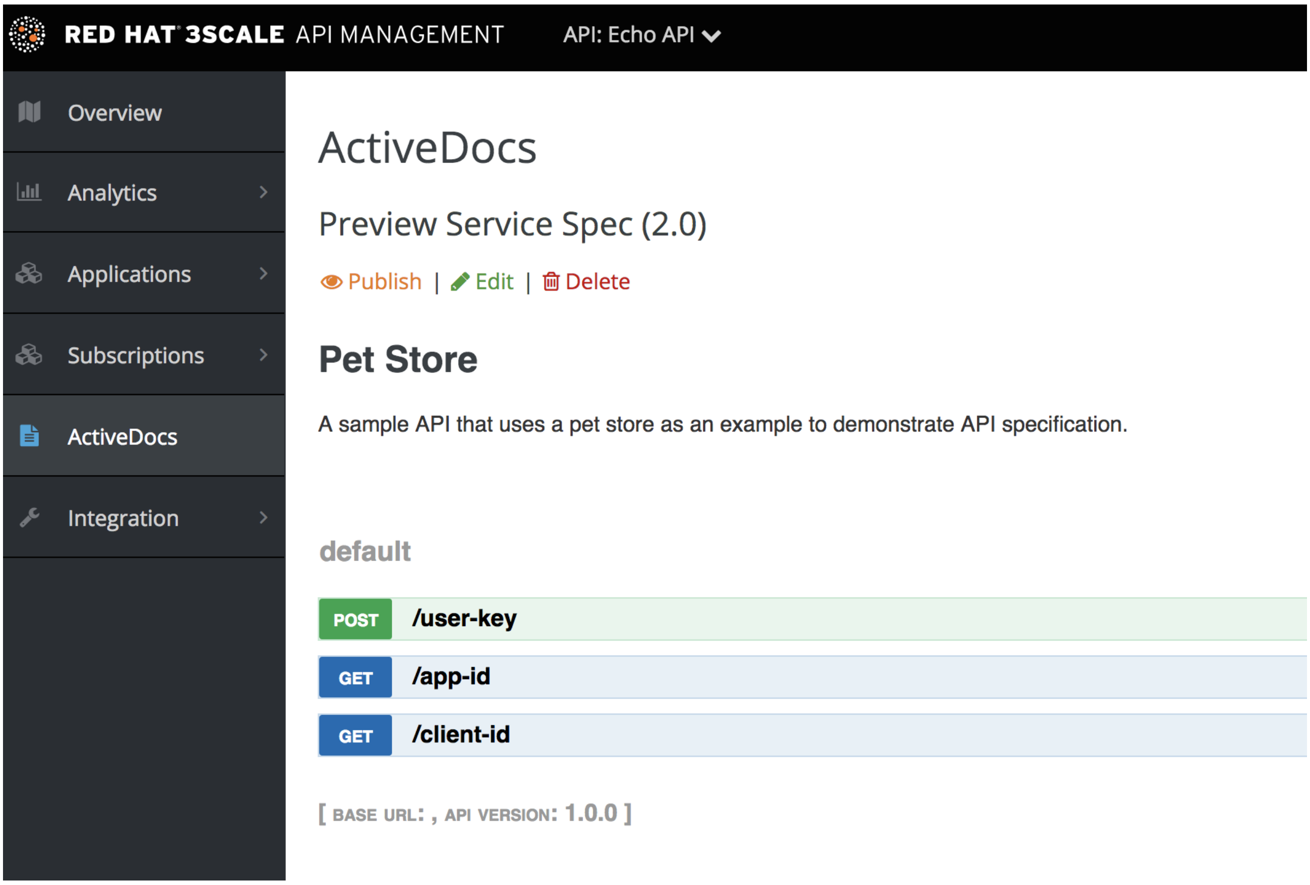


ActiveDocs

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:



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 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 .

2.1. 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 SpecificationPetstore 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.

2.2. 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

2.3. 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.

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. 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.

2.3.2. 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.

2.3.2.1. 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.

2.3.2.2. Info object

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

2.3.2.3. 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

2.3.3. 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.

2.3.3.1. 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_sercrets** - returns the client secrets for applications of the services that use OAuth/OpenID Connect authentication only.

API key authentication example

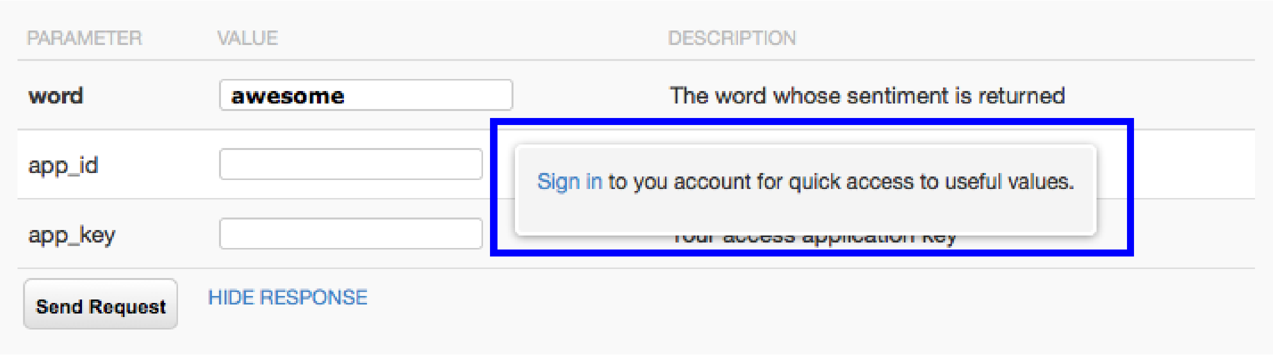
The following example shows using **"x-data-threescale-name": "user_keys"** for API key authentication only:

```
"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.

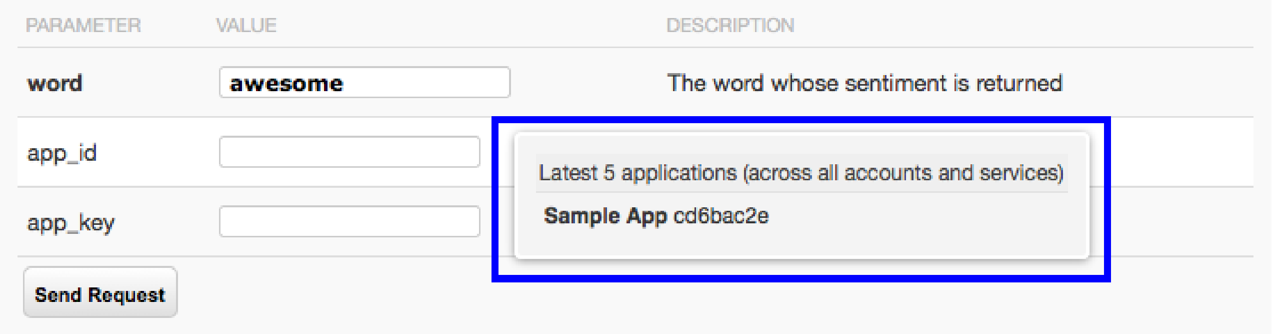
When you have declared your parameters, ActiveDocs will automatically prompt the ActiveDocs user to log in to the Developer Portal to get their keys as shown in the following screenshot:



PARAMETER	VALUE	DESCRIPTION
word	awesome	The word whose sentiment is returned
app_id		
app_key		Your access application key

Sign in to you account for quick access to useful values.

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	**awesome**	The word whose sentiment is returned
app_id		
app_key		

Latest 5 applications (across all accounts and services)

Sample App cd6bac2e

NOTE

The **x-data-threescale-name** field is an extension to the OAS specification that 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 will 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 ACTIVEDOCS IN THE 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}}";
```

PART II. API VERSIONING

CHAPTER 6. API VERSIONING

The 3scale API Management Platform allows API versioning. You have three ways to version your API correctly when you manage your API with 3scale. The following methods are examples of how you could version your API within the 3scale Gateway, which provides extra features due to 3scale's architecture.

6.1. GOAL

This guide is designed to give you enough information to implement an API versioning system within 3scale.

Suppose you have an API for finding songs. Users can search for their favorite songs by different keywords: artist, songwriter, song title, album title, and so on. Assume you had an initial version (v1) of the API and now you have developed a new, improved version (v2).

The following sections describe the three most typical ways of implementing an APR versioning system using 3scale:

- URL versioning
- Endpoint versioning
- Custom header versioning

6.2. PREREQUISITES

Complete the basics of connecting your API to 3scale before using this quick start guide.

6.3. URL VERSIONING

If you have different endpoints for searching songs (by artist, by song title, and so on), with URL versioning you would include the API version as part of the URI, for example:

1. api.songs.com/v1/songwriter
2. api.songs.com/v2/songwriter
3. api.songs.com/v1/song
4. api.songs.com/v2/song
5. and so on

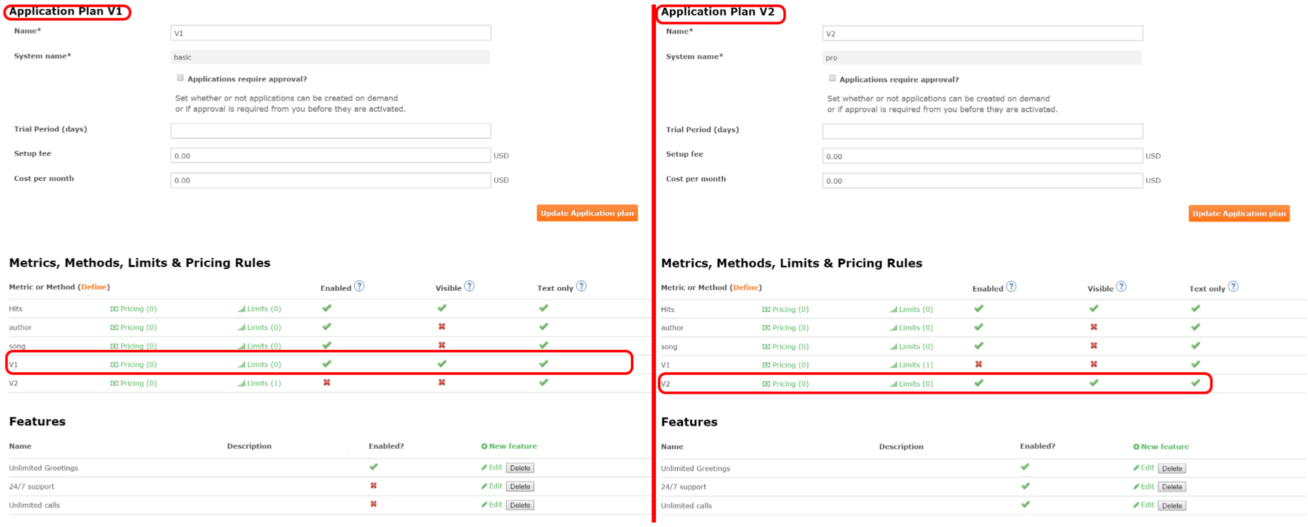
NOTE

When you use this method, you should have planned since v1 that you were going to version your API.

The 3scale Gateway would then extract the endpoint and the version from the URI. This approach allows you to set up application plans for any version/endpoint combination. You can then associate metrics with those plans and endpoints, and you can chart the usage for each endpoint on each version.

The following screen capture shows 3scale's flexibility.

Figure 6.1. Versioning Plan Feature



Application Plan V1

Name*	V1
System name*	basic
	□ Applications require approval? Set whether or not applications can be created on demand or if approval is required from you before they are activated.
Trial Period (days)	
Setup fee	0.00 USD
Cost per month	0.00 USD

Metrics, Methods, Limits & Pricing Rules

Metric or Method (Define)			Enabled	Visible	Text only
Hits	Pricing (0)	Limits (0)	✓	✓	✓
author	Pricing (0)	Limits (0)	✓	✗	✓
song	Pricing (0)	Limits (0)	✓	✗	✓
V1	Pricing (0)	Limits (0)	✓	✓	✓
V2	Pricing (0)	Limits (1)	✗	✗	✓

Features

Name	Description	Enabled?	New feature
Unlimited Greetings		✓	Edit Delete
24/7 support		✗	Edit Delete
Unlimited calls		✗	Edit Delete

Application Plan V2

Name*	V2
System name*	pro
	□ Applications require approval? Set whether or not applications can be created on demand or if approval is required from you before they are activated.
Trial Period (days)	
Setup fee	0.00 USD
Cost per month	0.00 USD

Metrics, Methods, Limits & Pricing Rules

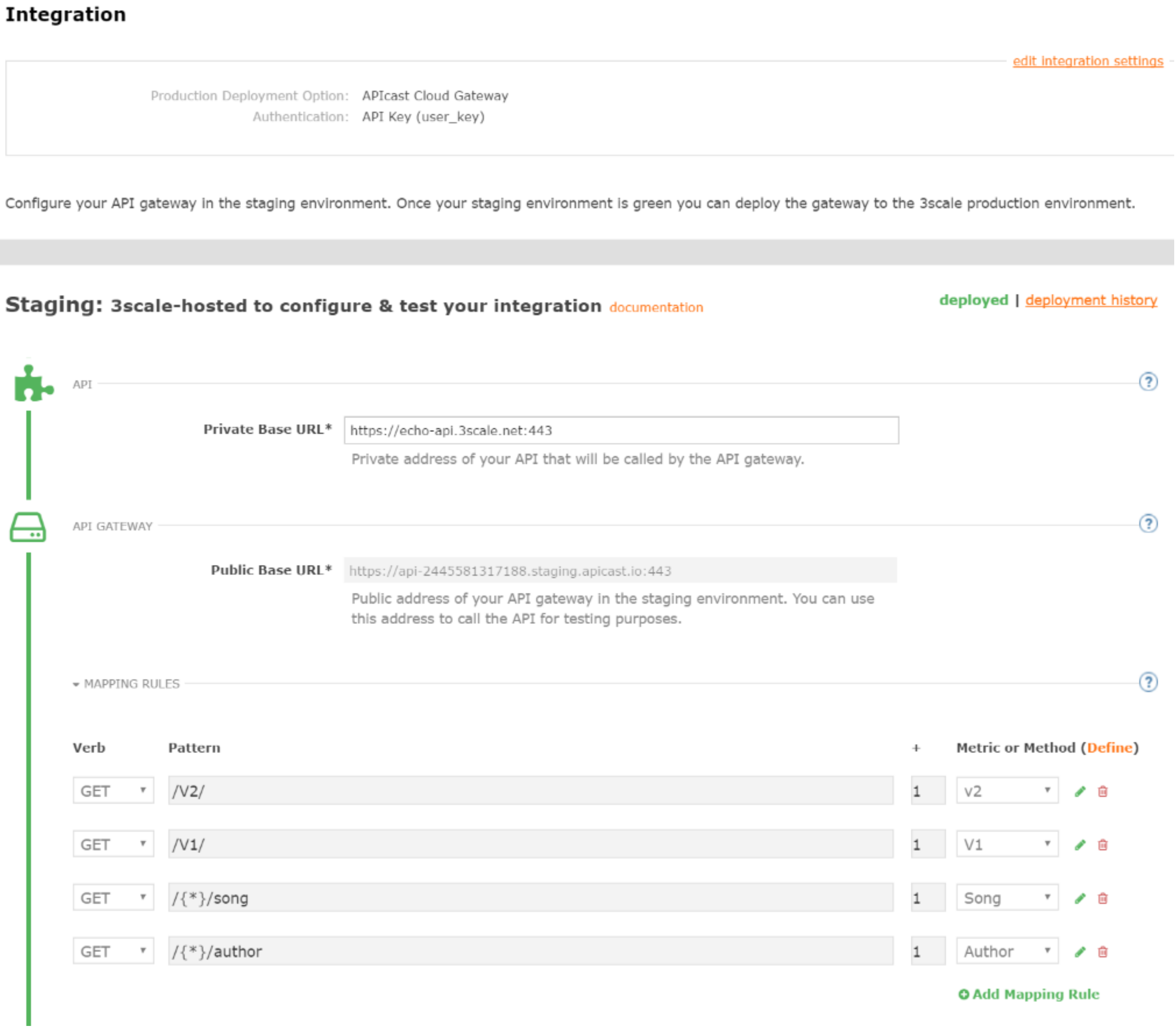
Metric or Method (Define)			Enabled	Visible	Text only
Hits	Pricing (0)	Limits (0)	✓	✓	✓
author	Pricing (0)	Limits (0)	✓	✗	✓
song	Pricing (0)	Limits (0)	✓	✗	✓
V1	Pricing (0)	Limits (1)	✗	✗	✓
V2	Pricing (0)	Limits (0)	✓	✓	✓

Features

Name	Description	Enabled?	New feature
Unlimited Greetings		✓	Edit Delete
24/7 support		✓	Edit Delete
Unlimited calls		✓	Edit Delete

The only thing left to do is go to **[your_API_name] > Integration > Configuration**in your 3scale Admin Portal and map your URIs to your metrics, as shown in the following diagram.

Figure 6.2. Mapping URIs to metrics



Integration

Production Deployment Option: APIcast Cloud Gateway
Authentication: API Key (user_key)

Configure your API gateway in the staging environment. Once your staging environment is green you can deploy the gateway to the 3scale production environment.

Staging: 3scale-hosted to configure & test your integration documentation

Private Base URL* https://echo-api.3scale.net:443
Private address of your API that will be called by the API gateway.

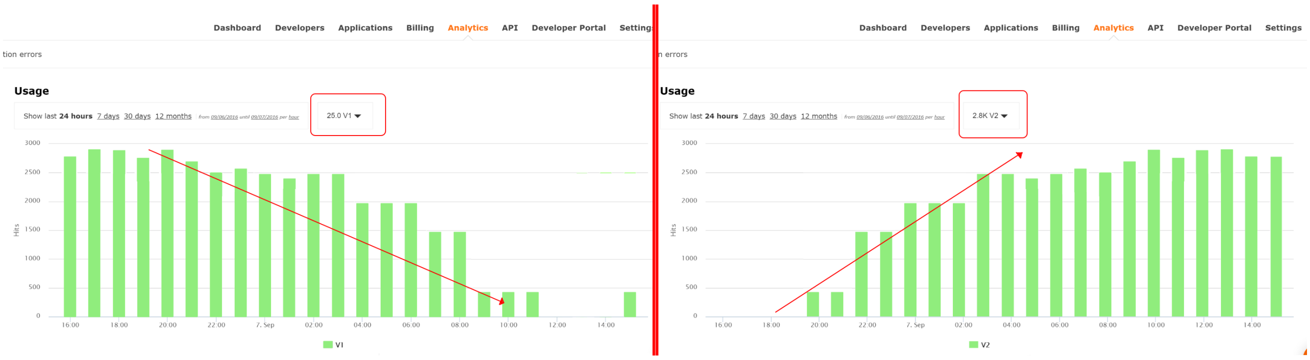
Public Base URL* https://api-2445581317188.staging.apicast.io:443
Public address of your API gateway in the staging environment. You can use this address to call the API for testing purposes.

Verb	Pattern	+	Metric or Method (Define)
GET	/V2/	1	v2
GET	/V1/	1	V1
GET	/{*}/song	1	Song
GET	/{*}/author	1	Author

You now have two different versions of your API, each with different features enabled. You also have full control and visibility on their usage.

If you want to communicate to all of your users that they should move to the API v2, you can send an internal note asking them to do so. You can monitor who makes the move and see how the activity on v1 decreases while the activity on v2 increases. By adding the metric in your authorization calls to 3scale, you can see how much overall traffic is hitting v1 vs. v2 endpoints and get an idea of when it is safe to deprecate v1.

Figure 6.3. Versioning



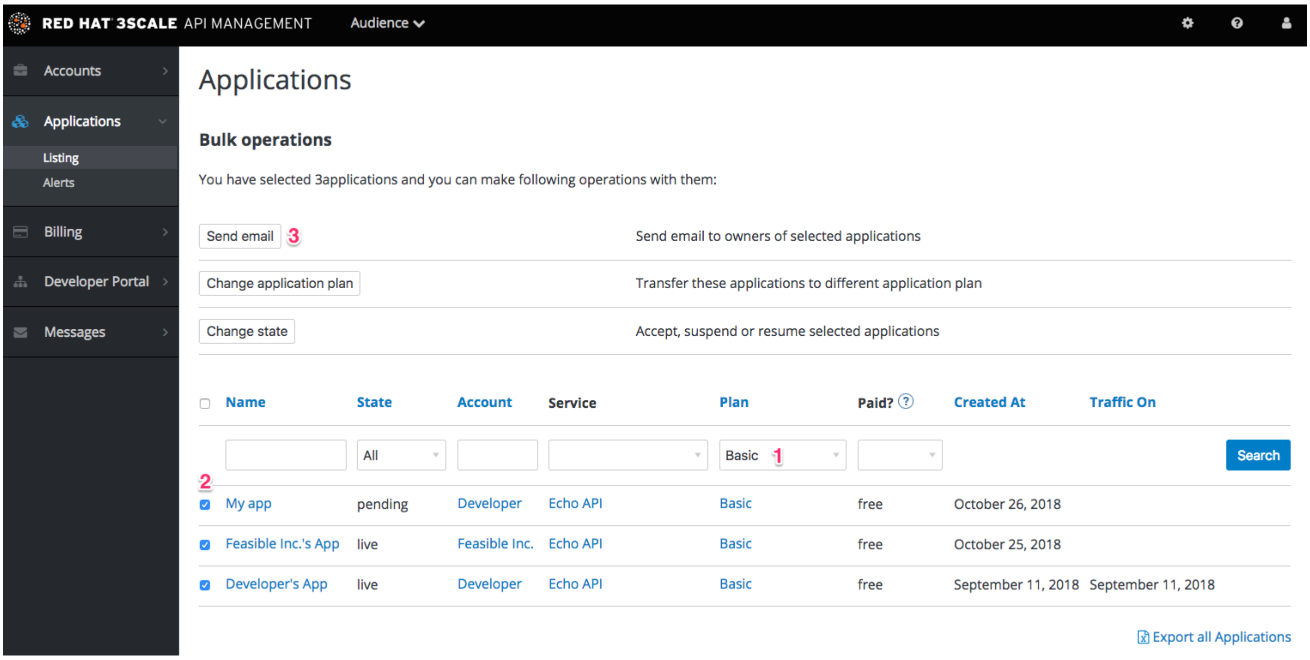
If some users continue to use v1, you can filter out only those users to send another internal note about switching to v2.

3scale provides a three-step method for sending deprecation notices.

1. Navigate to **Audience** > **Applications** > **Listing**and filter the list by the application plan that you want to send the deprecation note and click **Search**.
2. Click the multiselector to select all of the users for that particular version. New options display and allow you to perform bulk operations, such as **Send email**, **Change Application Plan**, and **Change State**.
3. Click **Send email** and follow the steps to send a deprecation notice to those customers who are still under the obsolete version.

The following image provides a visual reference.

Figure 6.4. Sending deprecation note



Applications

Bulk operations

You have selected 3applications and you can make following operations with them:

Send email	Send email to owners of selected applications
Change application plan	Transfer these applications to different application plan
Change state	Accept, suspend or resume selected applications

☐	Name	State	Account	Service	Plan	Paid?	Created At	Traffic On
		All			Basic			Search
☑	My app	pending	Developer	Echo API	Basic	free	October 26, 2018	
☑	Feasible Inc.'s App	live	Feasible Inc.	Echo API	Basic	free	October 25, 2018	
☑	Developer's App	live	Developer	Echo API	Basic	free	September 11, 2018	September 11, 2018

Export all Applications

For each authrep call that is made to an endpoint, you authenticate only once but report twice: once for the endpoint and once for the API version. There is no double-billing because the call can be authenticated only one time. For each call you make to any endpoint of a specific API version, you aggregate the hits on a convenient metric named after the version number (v1, v2, and so on), which you can use to compare full version traffic with each other.

6.4. ENDPOINT VERSIONING

You have the endpoint change for each version (api.cons.com/author_v1) with endpoint versioning. The gateway extracts the endpoint and the version from the endpoint itself. This method , as well as the previous method, allows the API provider to map external URLs to internal ones.

The endpoint versioning method can only be performed with the on-premise deployment method as it requires a URL rewrite using the LUA scripts that are provided as part of the on-premise configuration.

EXTERNAL		INTERNAL
api.songs.com/songwriter_v1	could be rewritten to	internal.songs.com/search_by_songwriter
api.songs.com/songwriter_v2	could be rewritten to	internal.songs.com/songwriter

Almost everything (mapping, application plans features, and so on.) works exactly the same as in the previous method.

6.5. CUSTOM HEADER VERSIONING

With custom header versioning, you use a header (that is, "x-api-version") instead of the URI to specify the version.

The gateway then extracts the endpoint from the path and the version from the header. Just as before, you can analyze and visualize any combination of path/version that you want. This approach has several inconveniences, regardless of the API management system you use. See API versioning methods, a brief reference for more information. Here are a few pointers on how 3scale works.

- Just like the previous method, custom header versioning can only be applied to on-premise hosted APIs because it requires some parsing/processing of the request headers to correctly route the authrep calls. This type of custom processing can only be done using Lua scripting.
- With this method, the fine-grained feature separation of the previous methods is much harder to achieve.
- One of the biggest advantages of using this methodology, and the main reason some API providers choose it, is because the URL and endpoints of your customers will never change. When a developer wants to switch from one API version to another, they only have to change the header. Everything else works the same.

PART III. API AUTHENTICATION

CHAPTER 7. AUTHENTICATION PATTERNS

By the end of this tutorial you will know how to set the authentication pattern on your API and the effect that this has on applications communicating with your API.

Depending on your API, you may need to use different authentication patterns to issue credentials for access to your API. These can range from API keys to openAuth tokens and custom configurations. This tutorial covers how to select from the available standard Authentication Patterns.

7.1. SUPPORTED AUTHENTICATION PATTERNS

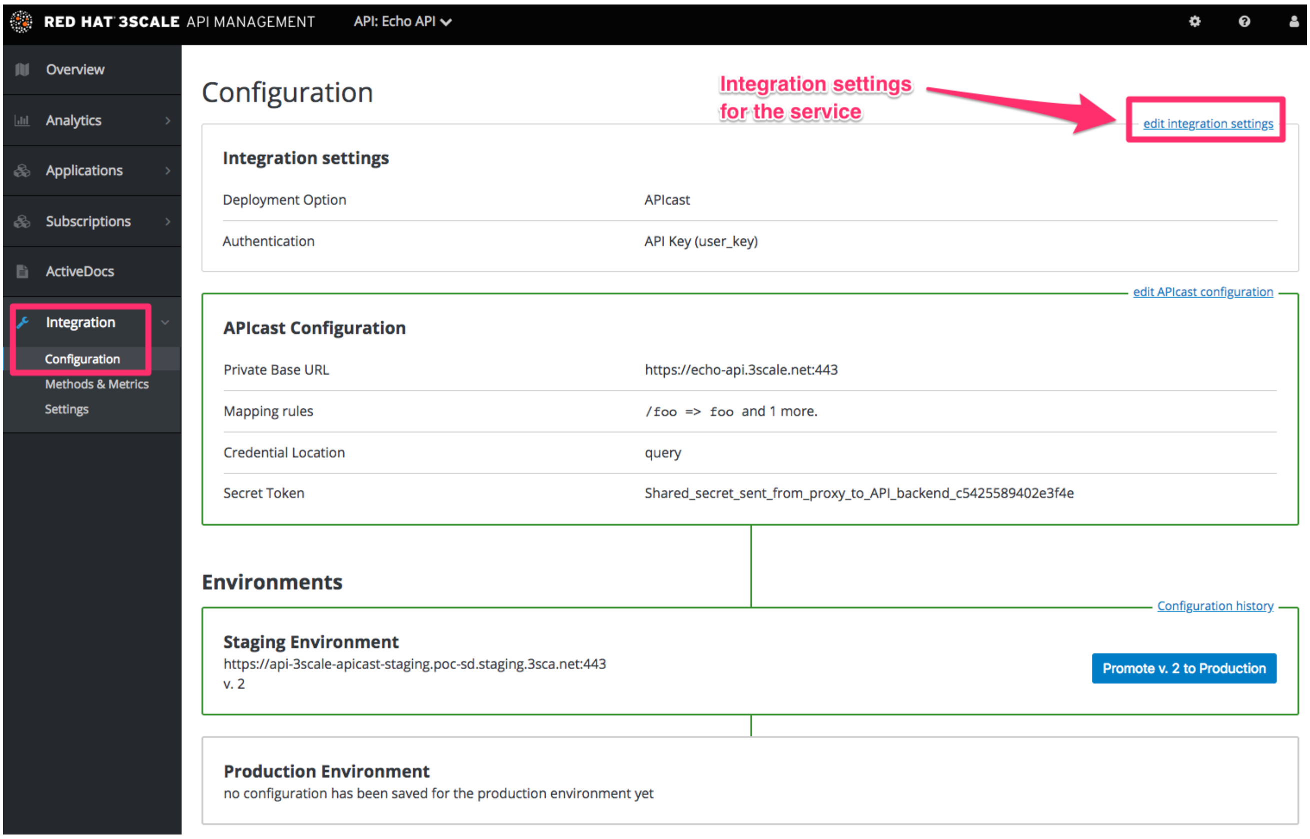
3scale supports the following authentication patterns out of the box:

- **Standard API Keys**: Single randomized strings or hashes acting as an identifier and a secret token.
- **Application Identifier and Key pairs**: Immutable identifier and mutable secret key strings.
- **OpenID Connect**

7.2. SETTING UP AUTHENTICATION PATTERNS

7.2.1. Select the authentication mode for your service

Navigate to the API service you want to work on (there may be only one service named API in which case select this). Go to the **Integration** section.


RED HAT 3SCALE API MANAGEMENT
API: Echo API
Overview
Analytics
Applications
Subscriptions
ActiveDocs
Integration
Configuration
Methods & Metrics
Settings
Integration settings for the service
edit integration settings
edit APIcast configuration
Configuration history
Promote v. 2 to Production

Configuration

Integration settings

Deployment Option	APIcast
Authentication	API Key (user_key)

APIcast Configuration

Private Base URL	https://echo-api.3scale.net:443
Mapping rules	`/foo => foo` and 1 more.
Credential Location	query
Secret Token	Shared_secret_sent_from_proxy_to_API_backend_c5425589402e3f4e

Environments

Staging Environment
https://api-3scale-apicast-staging.poc-sd.staging.3sca.net:443
v. 2

Production Environment
no configuration has been saved for the production environment yet

Each service that you operate can use a different authentication pattern, but only one pattern can be used per service.

IMPORTANT

You must not change the authentication pattern after the credentials have been registered because the behavior of the service may then become unpredictable. To change authentication patterns we recommend creating a new service and migrating customers.

7.2.2. Select the Authentication mode you want to use

To select an authentication mode, scroll to the AUTHENTICATION section. Here, you can choose one of the following options:

- API Key (user_key)
- App_ID and App_Key Pair
- OpenID Connect

7.2.3. Ensure your API accepts the correct types of credentials

Depending on the credential type chosen, you may need to accept different parameters in your API calls (key fields, IDs etc.). The names of these parameters may not be the same as those used internally at 3scale. The 3scale authentication will function correctly if the correct parameter names are used in calls to the 3scale backend.

7.2.4. Create an application to test credentials

To ensure that the credential sets are working, you can create a new application to issue credentials to use the API. Navigate to the Accounts area of your Admin Portal’s dashboard, click the account you want to use and click **new application**.

Filling out the form and clicking save will create a new application with credentials to use the API. You can now use these credentials to make calls to your API and records will be checked against the 3scale list of registered applications.

7.3. STANDARD AUTHENTICATION PATTERNS

3scale supports the authentication patterns detailed in the following sections.

7.3.1. API key

The simplest form of credential supported is the single API model. Here, each application with permissions on the API has a single (unique) long character string; example:

```
API-key = 853a76f7c8d5f4a1ee8bf10a4e0d1f13
```

By default, the name of the key parameter is **user_key**. You can use this label or choose another, such as **API-key**. If choosing another label, you need to map the value before you make the authorization calls to 3scale. The string acts as both, an identifier and a secret token, for use of the API. It is recommended that you use such patterns only in environments with low security requirements or with SSL security on API calls. Following are the operations that can be carried out on the token and application:

- Application Suspend: This suspends the applications access to the API and, in effect, all calls to the API with the relevant key will be suspended.

- Application Resume: Undoes the effect of an application suspend action.
- Key Regenerate: This action generates a new random string key for the application and associates it with the application. Immediately after this action is taken, calls with the previous token will cease to be accepted.

The latter action can be triggered from the API Administration in the Admin Portal and (if permitted) from the API Developers User console.

7.3.2. App_ID and App_Key pair

The API Key Pattern combines the identity of the application and the secret usage token in one token; however, this pattern separates the two. Each application using the API, issues an immutable initial identifier known as the **Application ID** (App ID). The App ID is constant and may or may not be secret. In addition, each application may have 1-n **Application Keys** (App_Keys). Each Key is associated directly with the App_ID and should be treated as secret.

```
app_id = 80a4e03 app_key = a1ee8bf10a4e0d1f13853a76f7c8d5f4
```

In the default setting, developers can create up to five keys per application. This allows a developer to create a new key, add it to their code, redeploy their application, and then disable old keys. This does not cause any application downtime the way an API Key Regeneration would.

Statistics and rate limits are always kept at the application ID level of granularity and not per API Key. If a developer wants to track two sets of statistics, they should create two applications rather than two keys.

It is also possible to change the mode in the system and allow applications to be created in the absence of application keys. In this case the 3scale system will authenticate access based on the App ID only (and no key checks are made). This mode is useful for widget type scenarios or where rate limits are applied to users rather than applications. In most cases you will want your API to enforce the presence of at least one application key per application present. This setting is available in **[your_API_name] > Integration > Settings**.

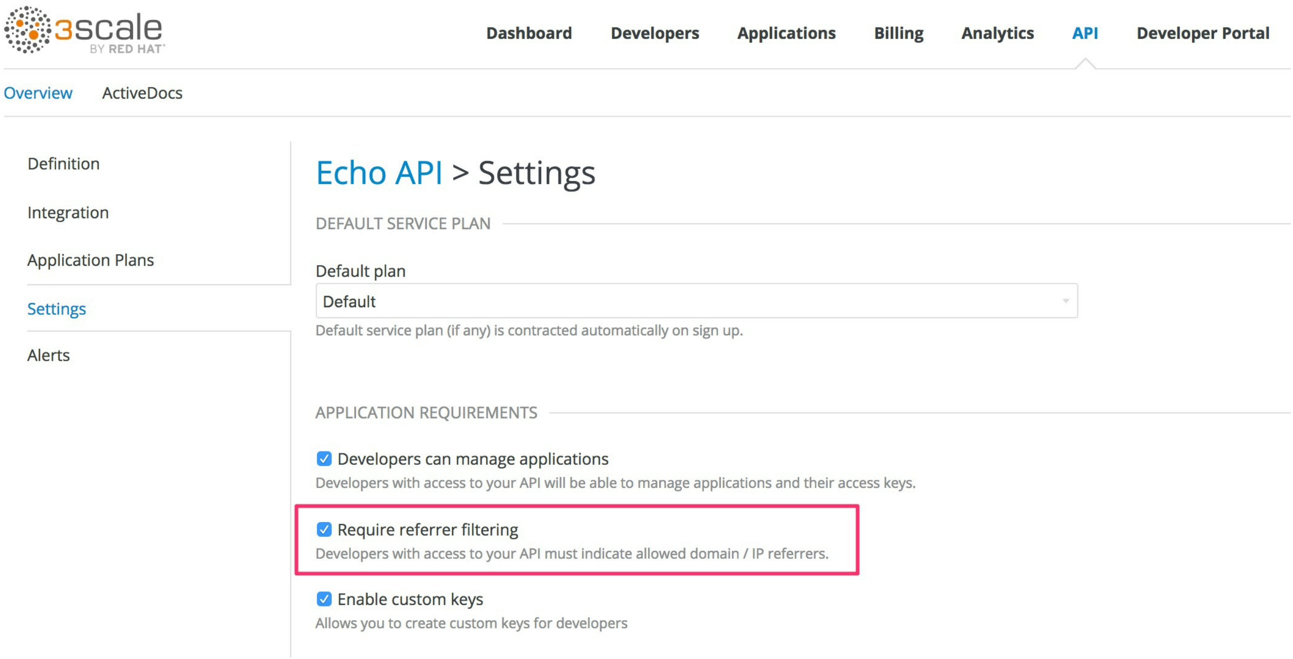
7.3.3. OpenID Connect

For information on OpenID Connect authentication, see the OpenID Connect integration section.

7.4. REFERRER FILTERING

3scale supports the Referrer Filtering feature that can be used to whitelist IP addresses or domain names from where an application can access the API. The API clients specify the referrer value in the **Referer** header. The purpose and the usage of the Referer header are described in the RFC 7231, section 5.5.2: Referer.

To enable the Referrer Filtering feature go to **[your_API_name] > Integration > Settings**, click the **Require referrer filtering** checkbox and click **Update Service**.



Echo API > Settings

DEFAULT SERVICE PLAN

Default plan: Default

Default service plan (if any) is contracted automatically on sign up.

APPLICATION REQUIREMENTS

☑ Developers can manage applications

Developers with access to your API will be able to manage applications and their access keys.

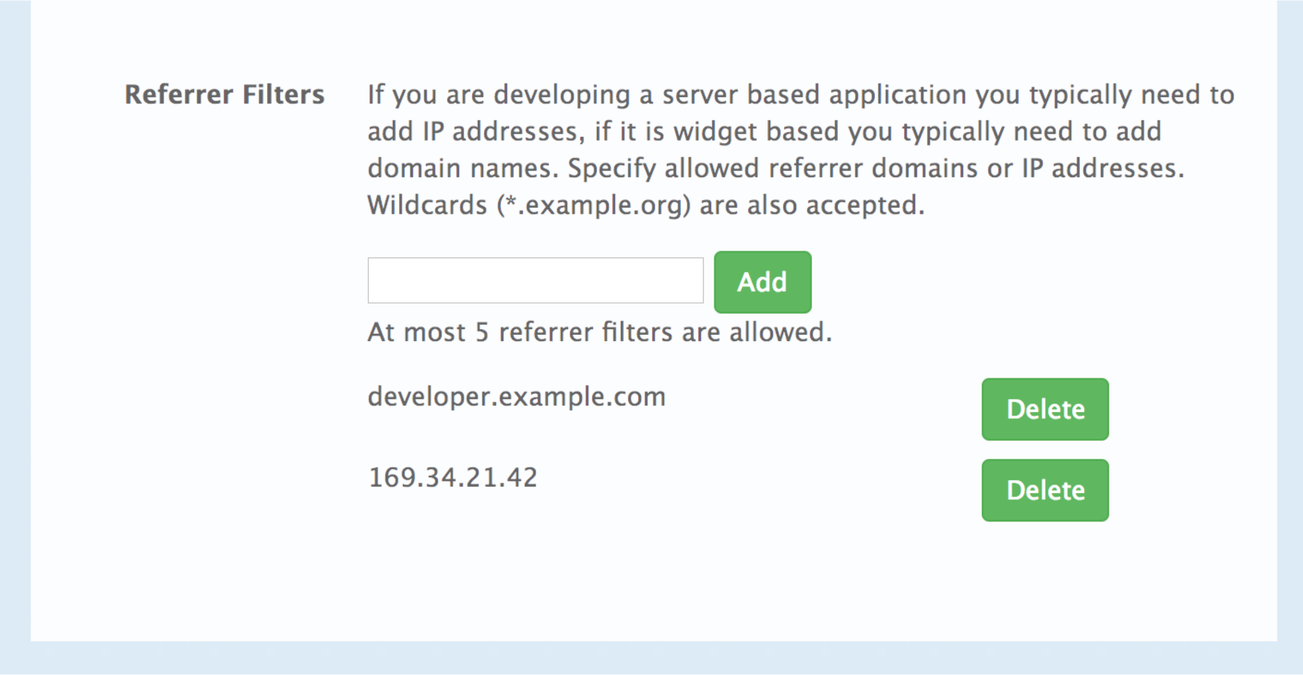
☑ Require referrer filtering

Developers with access to your API must indicate allowed domain / IP referrers.

☑ Enable custom keys

Allows you to create custom keys for developers

The developers with access to your API must configure allowed domain/IP referrers from the developer portal.



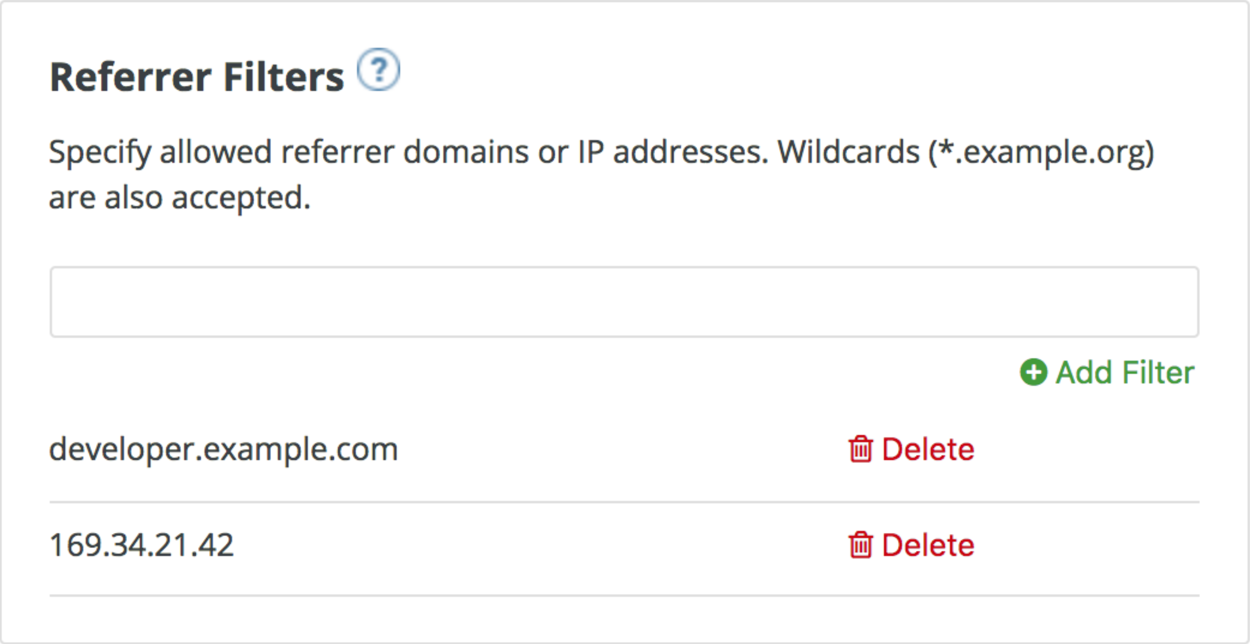
Referrer Filters If you are developing a server based application you typically need to add IP addresses, if it is widget based you typically need to add domain names. Specify allowed referrer domains or IP addresses. Wildcards (*.example.org) are also accepted.

At most 5 referrer filters are allowed.

developer.example.com

169.34.21.42

In the Admin Portal on the application details page for all applications that belong to this service a new **Referrer Filters** section displays. Here, the admin can also configure a whitelist of the allowed Referrer header values for this application.



Referrer Filters

Specify allowed referrer domains or IP addresses. Wildcards (*.example.org) are also accepted.

developer.example.com	Delete
169.34.21.42	Delete

You can set a maximum of five referrer values per application.

The value can only consist of Latin letters, numbers, and special characters *, ., and -. * can be used for wildcard values. If the value is set to *, any referrer value will be allowed, so the referrer check will be bypassed.

For the Referrer Filtering feature to work, the APIcast Referrer policy must be enabled in the service policy chain.

When the **Require referrer filtering** feature and the **3scale Referrer** policy are enabled, the authorization works as follows:

1. The applications that do not have Referrer Filters specified are authorized normally only using the provided credentials.

2. For the applications that have Referrer Filters values set, APIcast extracts the referrer value from the **Referer** header of the request and sends it as **referrer** param in the AuthRep (authorize and report) request to the Service Management API. The following table shows the AuthRep responses for different combination of the referrer filtering parameters.

referrer parameter passed?	Referrer Filters configured for the app?	Referrer parameter value	HTTP Response	Response body
Yes	Yes	matches referrer filter	200 OK	**<status><authorized>true</authorized></status>**
Yes	No	matches referrer filter	200 OK	**<status><authorized>true</authorized></status>**

referrer parameter passed?	Referrer Filters configured for the app?	Referrer parameter value	HTTP Response	Response body
Yes	Yes	does not match referrer filter	409 Conflict	**<status><authorized>false</authorized><reason>referrer "test.example.com" is not allowed</reason>** (**test.example.com** is an example)
Yes	No	does not match referrer filter	200 OK	**<status><authorized>true</authorized></status>**
Yes	Yes	*	200 OK	**<status><authorized>true</authorized></status>**
Yes	No	*	200 OK	**<status><authorized>true</authorized></status>**
No	Yes	–	409 Conflict	**<status><authorized>false</authorized><reason>referrer is missing</reason>**
No	No	–	200 OK	**<status><authorized>true</authorized></status>**

The calls that are not authorized by AuthRep are rejected by APIcast with an "Authorization Failed" error. You can configure the exact status code and the error message on the service Integration page.

CHAPTER 8. OPENID CONNECT INTEGRATION

3scale integrates with the 3rd-party Identity Providers (IdP) for authenticating the API requests using the OpenID Connect specification. OpenID Connect is built on top of OAuth 2.0 that complements the OAuth 2.0 Authorization framework with an authentication mechanism. When OpenID Connect authentication option is used, the API requests are authenticated using the access tokens in the JSON Web Token (JWT) format (RFC 7519).

The integration consists of the following two parts:

- Section 8.1, “JWT verification and parsing by APIcast”
- Section 8.2, “Client credentials synchronization by Zync”

Red Hat 3scale API Management fully supports both integration points with Red Hat Single Sign-On (RH-SSO) acting as the OpenID Provider. See the supported version of RH-SSO on the Supported Configurations page. APIcast integration is also tested with ForgeRock.

In both cases, you can configure the integration by specifying the *OpenID Connect Issuer* field in the APIcast Configuration on the Integration page of the service using OpenID Connect authentication option. For instructions, see Section 8.3, “Configure Red Hat Single Sign-On integration” .

8.1. JWT VERIFICATION AND PARSING BY APICAST

The API requests to the service using the OpenID Connect authentication mode should provide the access token in the JWT format, issued by the OpenID Provider, in the **Authorization** header using **Bearer** schema. The header should look like the following example:

```
Authorization: Bearer <JWK>
```

Example:

```
Authorization: Bearer:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2lkcC5leGFtcGxlLmNvbSIsInN1YiI
6ImFiYzEyMyIsIm5iZiI6MTUzNzg5MjQ5NCwiZXhwIjoxNTM3ODk2MDk0LCJpYXQiOjE1Mzc4OTI0OT
QsImp0aSI6ImlkMTIzNDU2IiwidHlwIjoiQmVhcmVyIn0.LM2PSmQ0k8mR7eDS_Z8iRdGta-Ea-
pJRrf4C6bAiKz-Nzhxpm7fF7oV3BOipFmimwkQ_-mw3kN--
oOc3vU1RE4FTCQGbzO1SAWHOZqG5ZUx5ugaASY-
hUHIohy6PC7dQl0e2NlAeqqg4MuZtEwrpESJW-
VnGdljrAS0HsXzd6nENM0Z_ofo4ZdTKvIKsk2KrdyVBOcjgVjYongtppR0cw30FwnpqfeCkuATeINN5OK
HXOibRA24pQyIF1s81nnmxLnjnVbu24SFE34aMGRXYzs4icMI8sK65eKxbvwV3PIG3mM0C4ilZPO26d
oP0YrLfVwFcqEirmENUAcHXz7NuvA
```

The JWT token contains a signature that the token’s receiver can verify and ensure that the token was signed by a known issuer and that its content has not been changed. 3scale supports RSA signature based on the public/private key pair. Here, the issuer signs the JWT token using a private key. APIcast verifies this token using a public key.

APIcast uses OpenID Connect Discovery for getting the JSON Web Keys (JWK) that can be used for verifying the JWT signature.

On each request, APIcast does the following:

1. Verifies the JWT token using the public key.

2. Validates the claims **nbf** and **exp**.

3. Verifies that the issuer specified in the claim **iss** (Issuer) is the same as the one configured in the *OpenID Connect Issuer* field.

4. Extracts the value of the **azp** or **aud** claim and uses it as the Client ID (that identifies the application in 3scale) to authorize the call through the Service Management API.

If any of the JWT validation or the authorization checks fail, APIcast returns an "Authenication failed" error. Otherwise, APIcast proxies the request to the API backend. The **Authorization** header remains in the request, so the API backend can also use the JWT token to check the user and client identity.

8.2. CLIENT CREDENTIALS SYNCHRONIZATION BY ZYNC

Using the Zync component, 3scale syncronizes the client (application) credentials between 3scale and the Red Hat Single Sign-On server configured through the *OpenID Connect Issuer* setting. Whenever a new application is created, updated, or deleted for a service configured to use OpenID Connect, Zync receives the corresponding event and communicates the change to the Red Hat Single Sign-On instance using RH-SSO API. The Section 8.3, “Configure Red Hat Single Sign-On integration” section provides the steps required to ensure that Zync has the correct credentials to use the RH-SSO API.

8.3. CONFIGURE RED HAT SINGLE SIGN-ON INTEGRATION

8.3.1. Configure Zync to use custom CA certificates

You must establish an SSL connection between Zync and Red Hat Single Sign-On. 3scale 2.2 and above supports custom CA certificates for Red Hat Single Sign-On with the **SSL_CERT_FILE** environment variable. This variable points to the local path of the certificates bundle. Configure it as follows:

1. Validate the new certificate with the following cURL command. The expected response is a JSON configuration of the realm. If validation fails it is an indicator that your certificate may not be correct.

   ```
   curl -v https://<secure-sso-host>/auth/realms/master --cacert customCA.pem
   ```

2. Add the certificate bundle to the Zync pod:

   a. Gather the existing content of the **/etc/pki/tls/cert.pem** file on the Zync pod. Run:

   ```
   oc exec <zync-pod-id> cat /etc/pki/tls/cert.pem > zync.pem
   ```

   b. Append the contents of the custom CA certificate file to **zync.pem**:

   ```
   cat customCA.pem >> zync.pem
   ```

   c. Attach the new file to the Zync pod as ConfigMap:

   ```
   oc create configmap zync-ca-bundle --from-file=./zync.pem
   ```

   ```
   oc set volume dc/zync --add --name=zync-ca-bundle --mount-path
   /etc/pki/tls/zync/zync.pem --sub-path zync.pem --source='{"configMap":{"name":"zync-ca-
   bundle","items":[{"key":"zync.pem","path":"zync.pem"}]}}'
   ```

```
oc patch dc/zync --type=json -p '[{"op": "add", "path":
"/spec/template/spec/containers/0/volumeMounts/0/subPath", "value":"zync.pem"}]'
```

3. After deployment, verify that the certificate is attached and the content is correct:

```
oc exec <zync-pod-id> cat /etc/pki/tls/zync/zync.pem
```

4. Configure the **SSL_CERT_FILE** environment variable on Zync to point to the new CA certificate bundle:

```
oc set env dc/zync SSL_CERT_FILE=/etc/pki/tls/zync/zync.pem
```

8.3.2. Configure Red Hat Single Sign-On

To configure Red Hat Single Sign-On, take the following steps:

1. Create a realm (**<REALM_NAME>**).
2. Create a client:
   a. Specify a client ID.
   b. In the *Client Protocol* field, select **openid-connect**.
3. To configure the client permissions, set the following values:
   a. *Access Type* to **confidential**.
   b. *Standard Flow Enabled* to **OFF**.
   c. *Direct Access Grants Enabled* to **OFF**.
   d. *Service Accounts Enabled* to **ON**.
4. Set the service account roles for the client:
   a. Navigate to the **Service Account Roles** tab of the client.
   b. In the *Client Roles* dropdown list, click **realm-management**.
   c. In the *Available Roles* pane, select the **manage-clients** list item and assign the role by clicking **Add selected >>**.
5. Note the client credentials:
   a. Make a note of the client ID (**<CLIENT_ID>**).
   b. Navigate to the **Credentials** tab of the client and make a note of the *Secret* field (**<CLIENT_SECRET>**).
6. Add a user to the realm:
   a. Click the **Users** menu on the left side of the window.
   b. Click **Add user**.
   c. Type the username, set the *Email Verified* switch to **ON**, and click **Save**.

d. On the **Credentials** tab, set the password. Enter the password in both the fields, set the *Temporary* switch to **OFF** to avoid the password reset at the next login, and click **Reset Password**.

e. When the pop-up window displays, click **Change password**.

8.3.3. Configure 3scale

After you have created and configured the client in Red Hat Single Sign-On, you must configure 3scale to work with Red Hat Single Sign-On.

To configure 3scale, take the following steps:

1. Enable OpenID Connect.

   a. Select the service on which you want to enable the OpenID Connect authentication, navigate to **[your_API_name] > Integration > Configuration**

   b. Select **edit integration settings**.

   c. Under the **Authentication** deployment options, select **OpenID Connect**.

   d. Click **Update Service** to save the settings.

2. Edit the APIcast Configuration:

   a. Navigate to **[your_API_name] > Integration > Configuration**

   b. Select **edit APIcast configuration**.

   c. Under the **Authentication Settings** heading, in the *OpenID Connect Issuer* field, enter the previously noted client credentials with the URL of your Red Hat Single Sign-On server (located at host **<RHSSO_HOST>** and port **<RHSSO_PORT>**).

   ```
   https://<CLIENT_ID>:<CLIENT_SECRET>@<RHSSO_HOST>:
   <RHSSO_PORT>/auth/realms/<REALM_NAME>
   ```

   d. To save the configuration, click **Update the Staging Environment**

8.4. OAUTH 2.0 SUPPORTED FLOWS

The API clients must get access tokens from the OpenID Connect issuer configured in 3scale, using any OAuth 2.0 flow that is supported by this OpenID provider. In case of Red Hat Signle Sign-On, the following flows are supported (the terms used in RH-SSO clients are specified in parenthesis):

- Authorization Code (*Standard Flow*)
- Resource Owner Password Credentials (*Direct Access Grants Flow*)
- Implicit (*Implicit Flow*)
- Client Credentials (*Service Accounts Flow*)

When clients under OpenID Connect (OIDC) are created in 3scale, the corresponding clients created by Zync in Red Hat Single Sign-On (RH SSO) have only the Authorization Code flow enabled. This flow is recommended as the most secure and suitable for most cases. However, it is possible to enable other flows.

8.4.1. How OAuth 2.0 supported flows work

The client gets the access token using the authorization request, or the token request, or both. The URLs that receive these requests can be discovered using the **.well-known/openid-configuration** endpoint of the OpenID provider, in the **"authorization_endpoint"** and **"token_endpoint"**, accordingly. Example: **https://<RHSSO_HOST>:<RHSSO_PORT>/auth/realms/<REALM_NAME>/.well-known/openid-configuration**.

8.4.2. Configuring OAuth 2.0 supported flows

You can configure allowed OAuth 2.0 flows for the 3scale API in the Admin Portal. When you create a new application, the basic integration is finished, including the OpenId Connect (OIDC) configuration.

To configure OAuth 2.0 supported flows, perform these steps:

1. Navigate to the Authentication Settings section: **[Your_API_name] > Integration > edit integration settings > Authentication**
2. Choose **OpenId Connect**.
3. The corresponding flows are enabled on the client on RH SSO side. You can view them by navigating through **[Your_API_name] > Integration > Edit APIcast configuration > Authentication Settings**
   - **standardFlowEnabled** (Authorization Code flow) [selected by default]
   - **implicitFlowEnabled** (Implicit flow)
   - **serviceAccountsEnabled** (Service Accounts Flow)
   - **directAccessGrantsEnabled** (Direct Access Grant Flow)
4. Choose one or multiple flows.
5. To save your changes, click **Update the Staging Environment**

8.5. TEST THE INTEGRATION

To test the integration, you must perform the steps listed in the following sections.

8.5.1. Test the client synchronization

To test the client synchronization, take the following steps:

1. Create an application for the service where you configured the OpenID Connect integration.
2. Note the client ID and the client Secret of the generated application.
3. Verify that the client with the same client ID and client secret is now present in the configured Red Hat Single Sign-On realm.
4. Update the Redirect URL of the application in the 3scale admin portal. Redirect URLs should be as specific as possible.

5. Verify that the *Valid Redirect URIs* field of the client in Red Hat Single Sign-On has been updated accordingly.

8.5.2. Test the API authorization flow

To test the APT authorization flow, take the following steps:

1. Get the access token from the Red Hat Single Sign-On server using an OAuth 2.0 flow that is enabled on the corresponding RH-SSO client.

2. Use the value of the **access_token** retrieved from RH-SSO in the **Authorization** header as follows: **Authorization: Bearer <access_token>**

If the token is correct and the corresponding application in 3scale is authorized, APIcast gateway returns a response from the API backend.

8.6. EXAMPLE OF THE INTEGRATION

The service "API" in 3scale is configured to use the OpenID Connect authentication. The *Public Base URL* on the service "API" is configured to be **https://api.example.com** and the *Private Base URL* is configured to be **https://internal-api.example.com**.

The *OpenID Connect Issuer* field is set to **https://zync:41dbb98b-e4e9-4a89-84a3-91d1d19c4207@idp.example.com/auth/realms/myrealm** in the API integration and the client **zync** in the realm **myrealm** has the correct Service Account roles.

In 3scale, there is an application having the **myclientid** client ID, **myclientsecret** client secret, and a **https://myapp.example.com** Redirect URL. In Red Hat Single Sign-On, in the **myrealm** realm, there also exists a client having a **myclientid** client ID, **myclientsecret** secret, and **https://myapp.example.com** *Valid Redirect URIs*. Standard Flow is enabled on this client. There is a user configured in the **myrealm** realm having the *myuser* username and *mypassword* password.

The flow is as follows:

1. Using the endpoint **https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/auth**, the application sends an Authorization request to RH-SSO. The application should provide the client ID **myclientsecret** and Redirect URL **https://myapp.example.com** with the request.

2. RH-SSO shows the login window, where the user must provide the user's credentials: Username *myuser* and password *mypassword*.

3. Depending on the configuration, and if it is the first time that the user is authenticating in this specific application, the consent window displays.

4. After the user is authenticated, the applciation sends a Token request to RH-SSO using the endpoint **https://idp.example.com/auth/realms/myrealm/protocol/openid-connect/token** and providing the client ID **myclientid**, client secret **myclientsecret** and Redirect URL **https://myapp.example.com**.

5. RH-SSO returns a JSON with an "access_token" field **eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A**.

6. The application sends an API request to **https://api.example.com** with the header **Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk…xBArNhqF-A**.

7. The application should receive a successful response from **https://internal-api.example.com**.

PART IV. OPENAPI SPECIFICATION (OAS)

CHAPTER 9. CREATING A NEW SERVICE BASED ON OPENAPI SPECIFICATION (OAS)

9.1. INTRODUCTION

This documentation outlines the features of OpenAPI 2.0 specification (OAS) in 3scale 2.5 and provides steps to update an existing service or create a new one.

9.2. PREREQUISITES

- OpenAPI Specification (OAS)
- A 3scale 2.5 instance tenant credentials (**token** or **provider_key**)

9.3. 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.

9.4. 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
```

9.4.1. 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]
```

9.4.2. 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]
```

9.4.3. 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> -
```