Open Banking Workshop

Red Hat Workshop
March 2019

Table of Contents

Introduction

Duration and format

This is a workshop designed for approximately 2 hours delivery. The focus is on the adoption of the products to build a comprehensive financial solution. This is not an introduction to the different products used in the open banking solution portal, but it is focused on the modules and functionalities relevant for the FSI industry. Typically, instructors would talk through the introduction slides, and then for each hands-on lab, explain the steps needed to achieve the objective of the lab calling on the main functionalities of the products. At the end of each activity there will be a checkpoint with the attendees.

Timeline

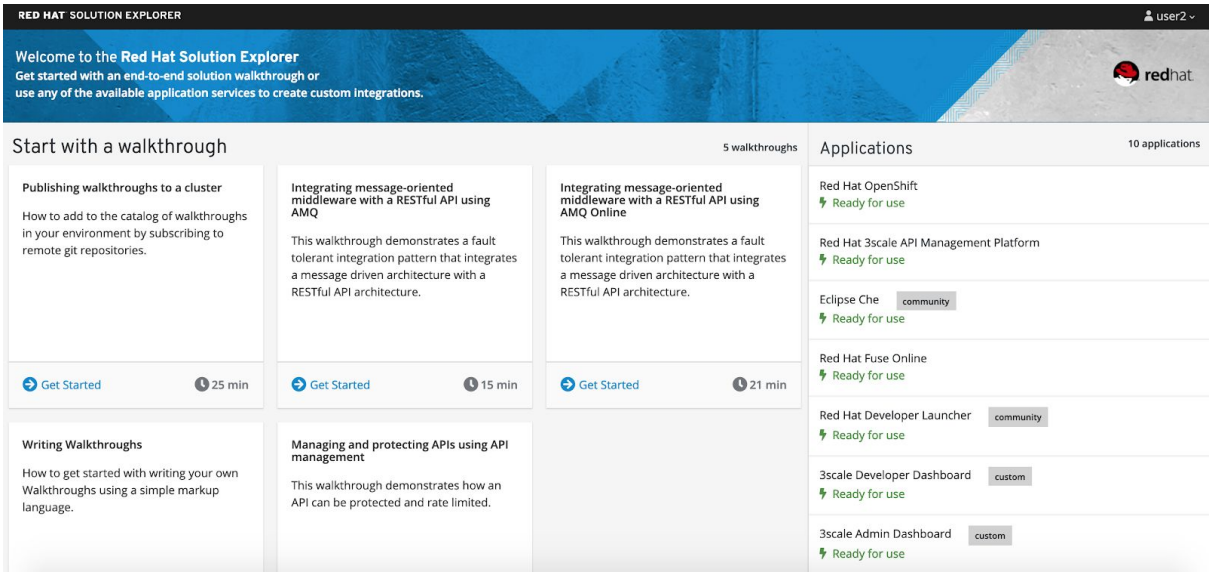
30 minutes	Architecture and key features
30 minutes	*Start of Practical Part 1* Financial Backend service demo and building blocks on Fuse Online [hands-on of the attendees in parallel]
25 minutes	Basics of Service Configuration and Integration in 3scale. Basics of Developer portal content publishing [hands-on of the attendees in parallel]
5 minutes	**CHECKPOINT**
15 minutes	**BREAK**
10 minutes	*Start of Practical Part 2* Introducing OIDC & OAuth [slides]
30 minutes	Basics of RH SSO and 3scale OIDC API authentication [hands-on of the attendees in parallel]
5 minutes	**CHECKPOINT**
10 minutes	(optional according to timing) OpenShift high level walkthrough [simple demo]
5 minutes	**Q&A and closing**

Practical Lab - Part 1

Log into the Red Hat Solution Explorer with your user id and the password “openshift”.

The URL has the following structure:
https://tutorial-web-app-webapp.apps.GUID.openshiftworkshop.com

(with the GUID replaced with the value provided by your instructor).



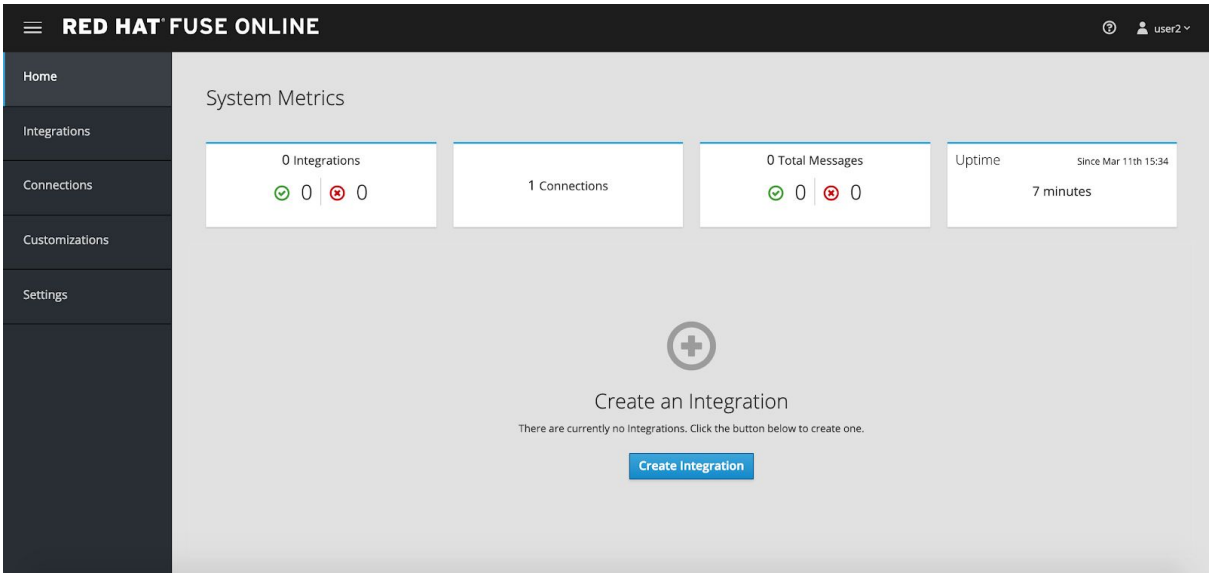
We will see how integr8ly environment works and what you get. It provides out of the box integration between products but it doesn’t change the base components that come with it. You can install yourself this type of environment starting from a clean or empty OpenShift installation.

FUSE ONLINE

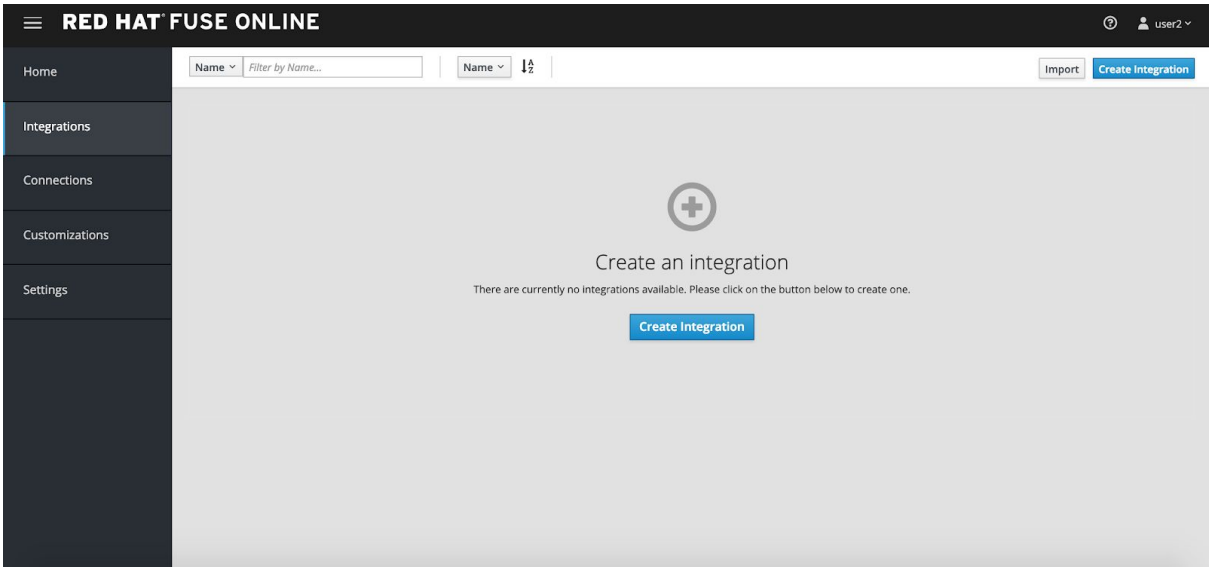
Let’s start our first lab session, by opening Red Hat Fuse Online. You will get there by clicking **Red Hat Fuse Online** link in the tutorial dashboard.

When using Fuse Online for the first time, you will have to authorize access by clicking on "Allow selected permissions" and having both permissions checkboxes selected.

We will see the main functionalities of it and then use it to build a simple integration block.

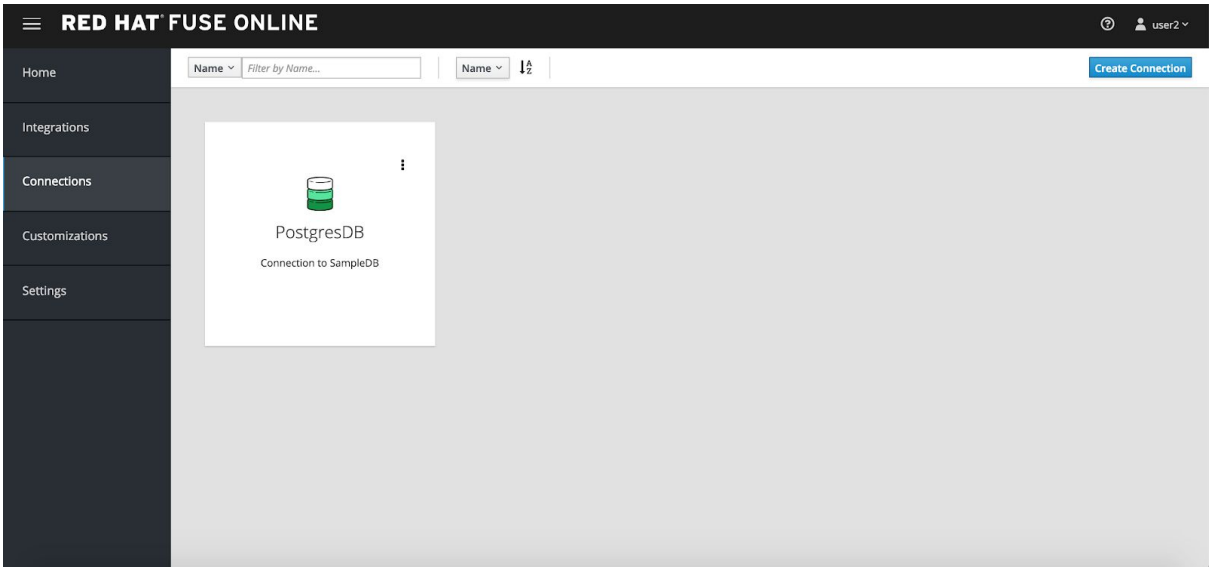


The main dashboard (**Home**) is the landing page and it is especially important when you need to check messages coming from the requests being sent and in general to give an overview of the deployed integration blocks.



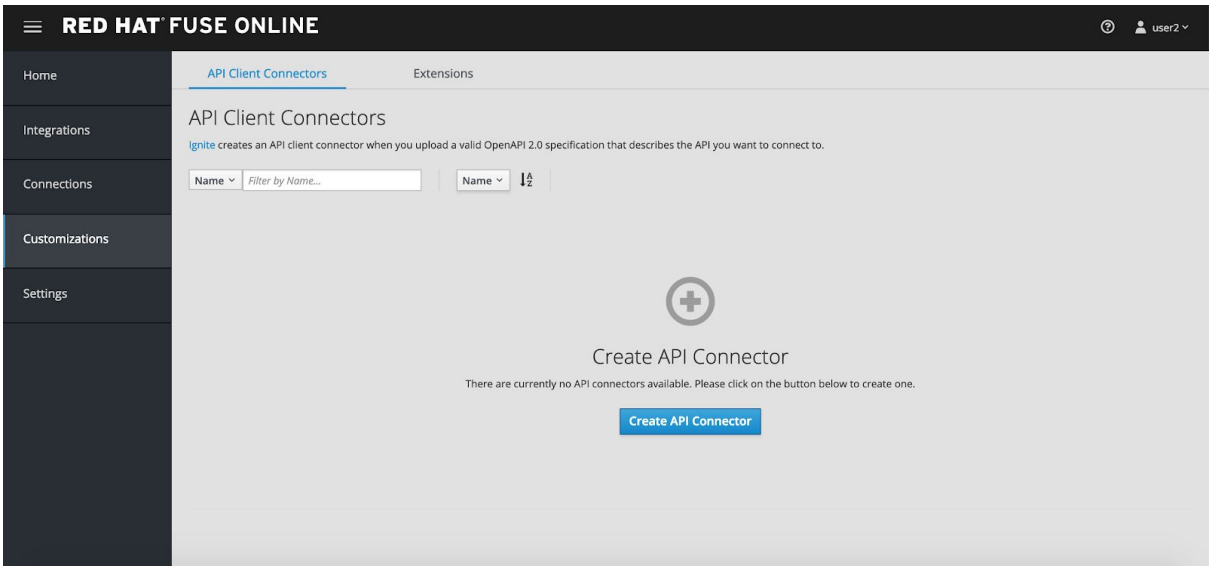
The integration window (**Integrations**) allows you to have a graphical representation of the mediation/transformation and is the first tool to onboard the citizen developers.

Behind the curtains there is Camel (which is a proven technology and deployed in highly transactional environment as well) and although the connectors available right now are just few the number will grow dramatically as this is the repository http://camel.apache.org/components.html .

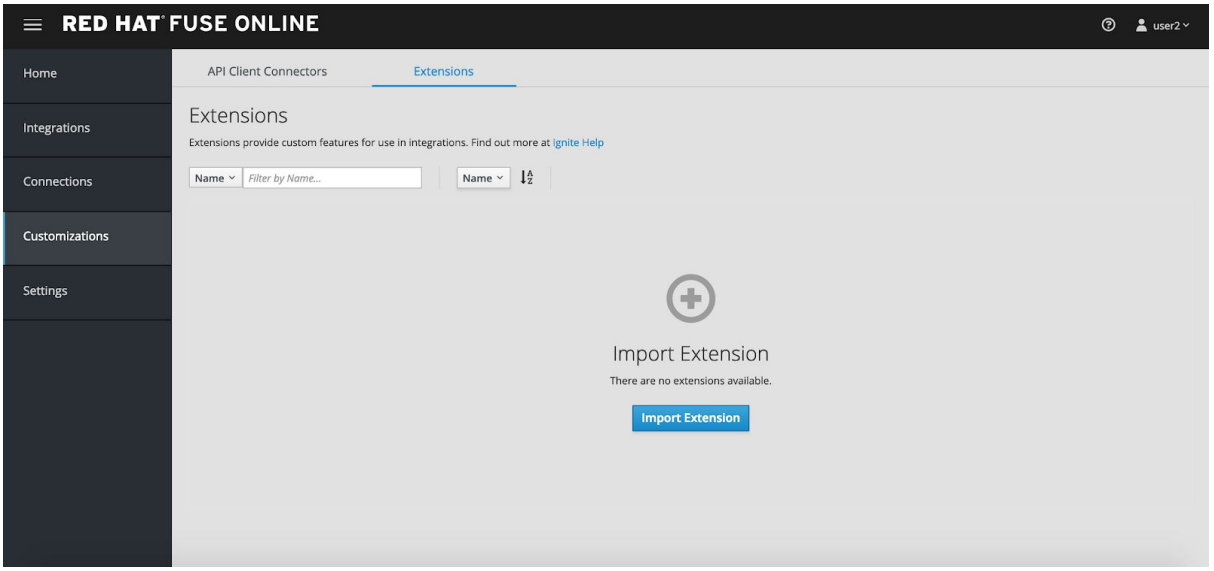


(**Connections**) you can start or end (or sometimes interpose) connections which are fully configured connectors. We will be showing the configuration of one connector during today's lab.

Various connectors are available and new connectors are coming in the future and will be ported onto Fuse Online.



(**Customizations**) there are two panes available here, the first one dedicated to API connections, which we will be using afterwards as Connection in the Integration.



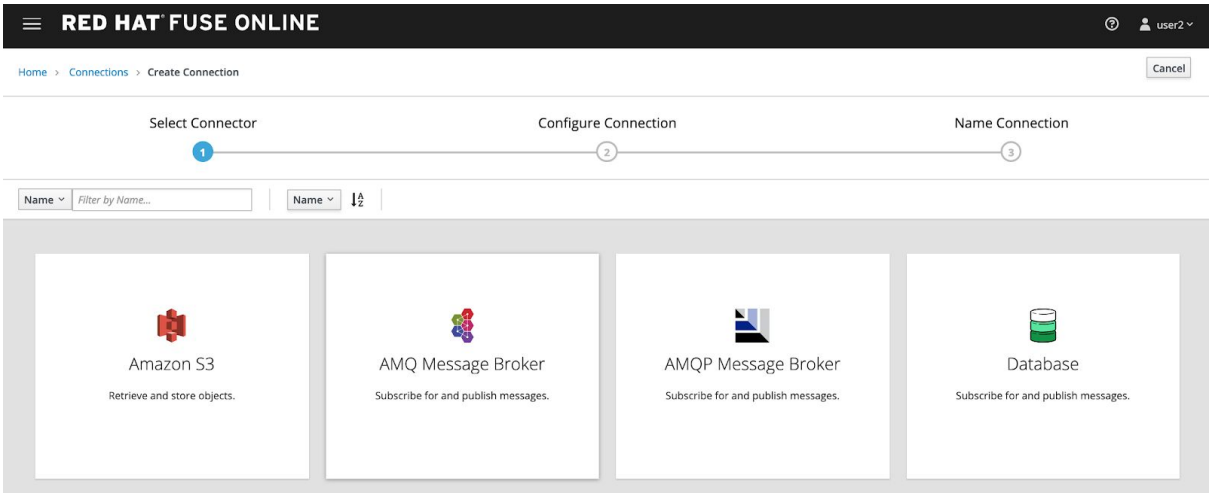
(**Extensions** panel) this is the part where you can extend the functionalities of the platform. You can compile a custom or community available extension to perform a specific functionality or connect to a specific system and in this easy way add transformation functionalities to the product.

Let's start INTEGRATING now!

This first session will take us to the creation of a simple REST service from a database source. This is quite a common scenario and one that goes well for quick prototyping service scenarios.

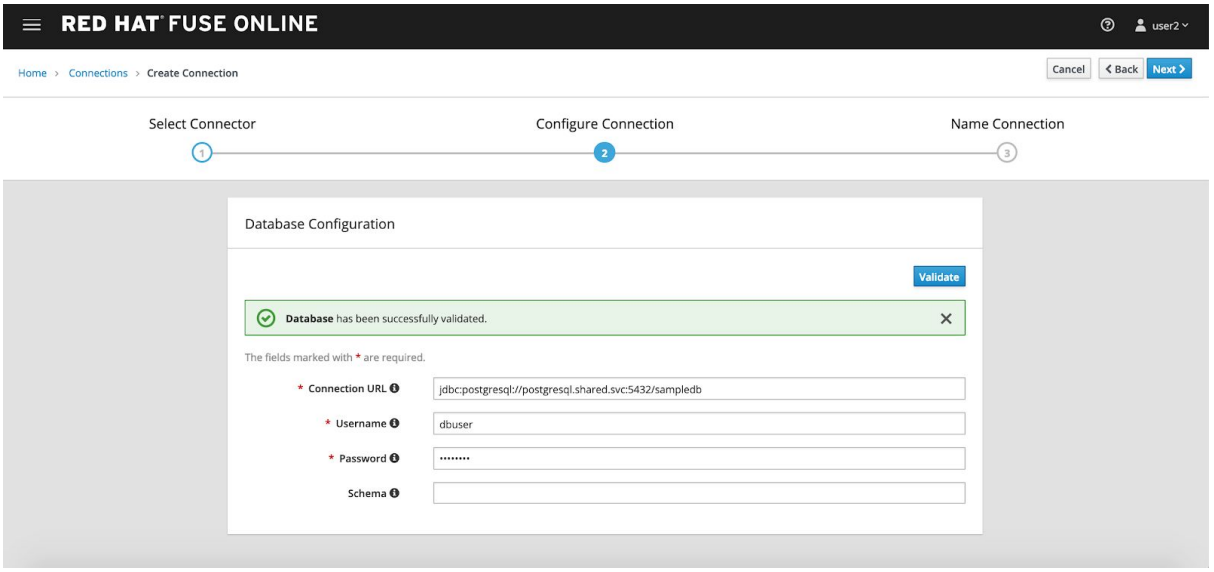
Let's start by creating the connection to an existing DB that I previously created. It is a DBaaS built on PostgreSQL and hosted on this environment. You could be running this anywhere else, as long as you have a public direct connection to the DB to use.

Under "Connections" click on "Create Connection" and select the "Database" connector.

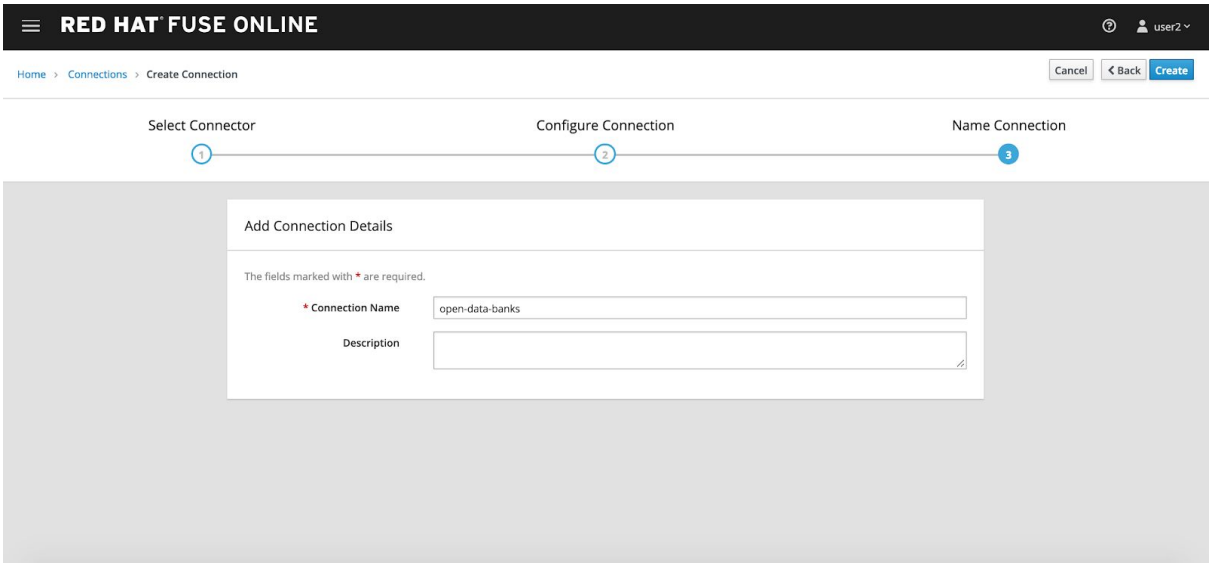


Use the following connection details and validate them by clicking on “Validate”:

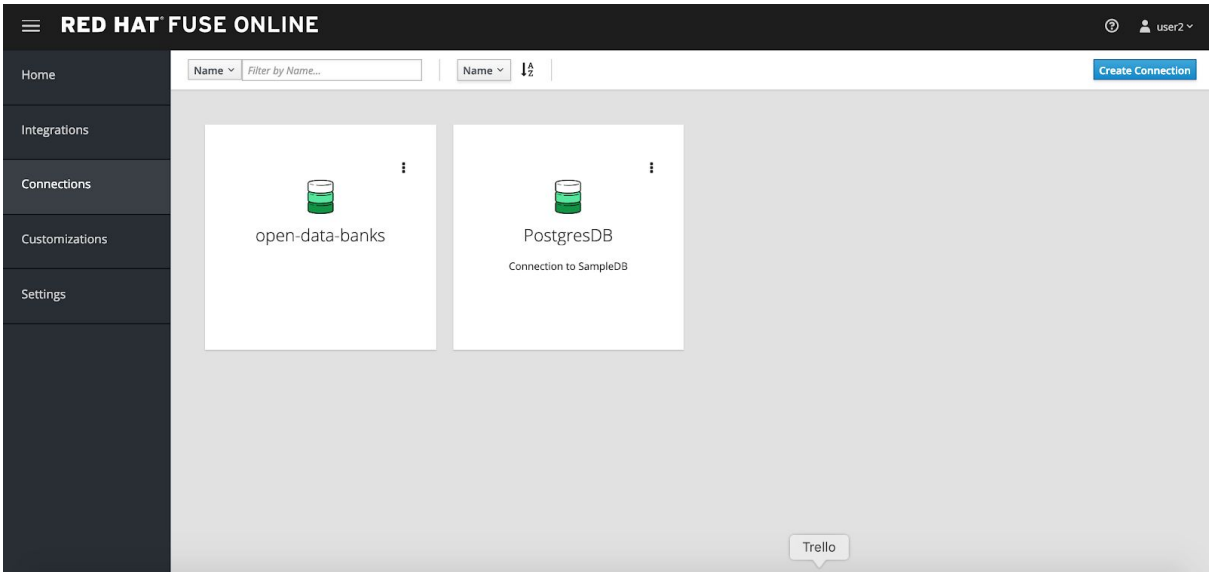
Connection URL: jdbc:postgresql://postgresql.shared.svc:5432/sampledb
Username: dbuser
Password: password



Click “Next” and provide “open-data-banks” as the “Connection Name”. Then click on “Create” to create the new connection.



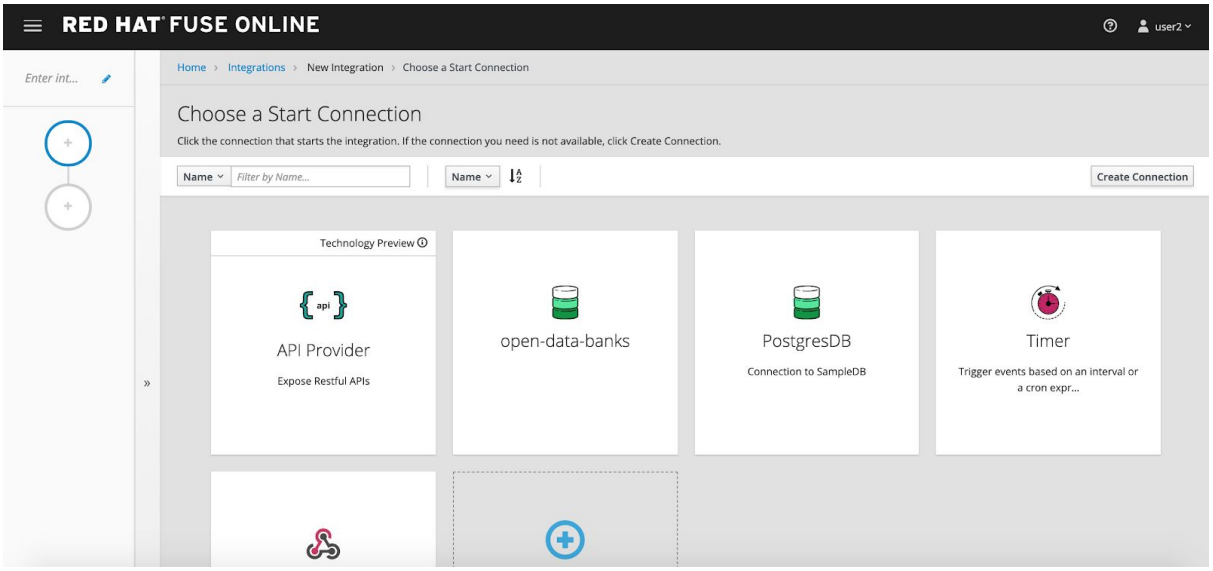
We can now use this connection as an integration starting, middle or finishing point.



Now that we have configured the end of the integration path we want to build and we will see where to find the start and we will put the two pieces together.

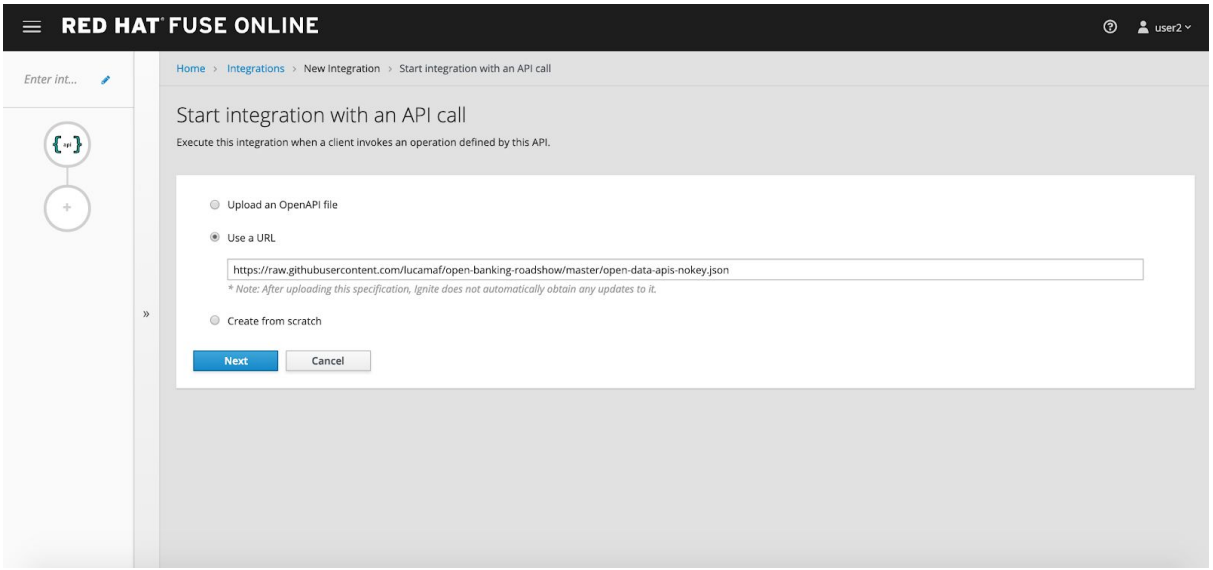
Under “Integrations” click on “Create Integration”

We will start by exposing a REST endpoint that will then get mapped to the backend datasource.

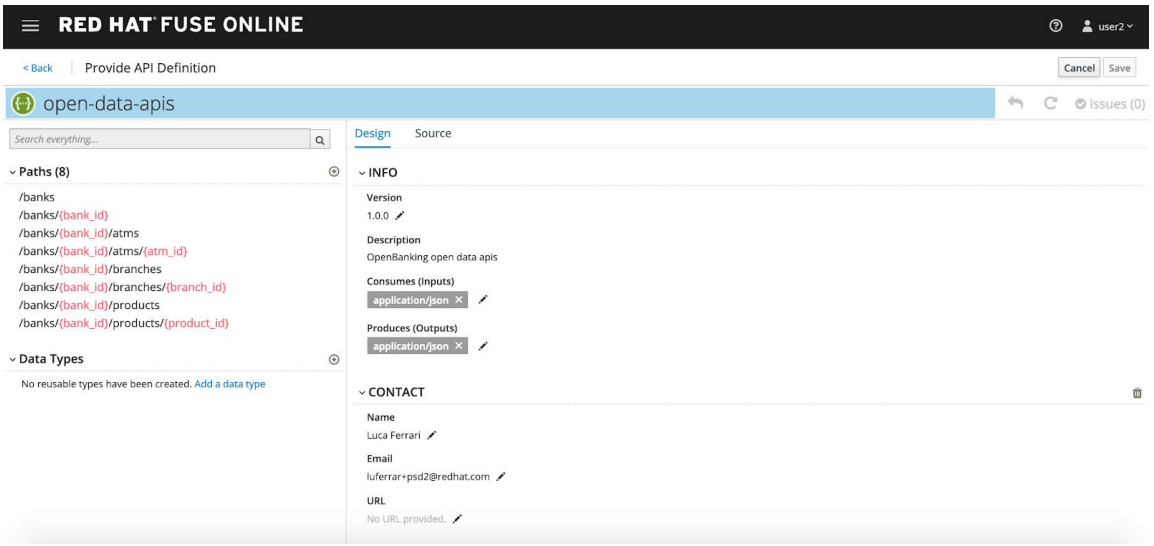


Use the “**API Provider”** connector with the following OpenAPI file (under “Use a URL”):
https://raw.githubusercontent.com/lucamaf/open-banking-roadshow/master/open-data-apis-nokey.json

Then click on “Next”.



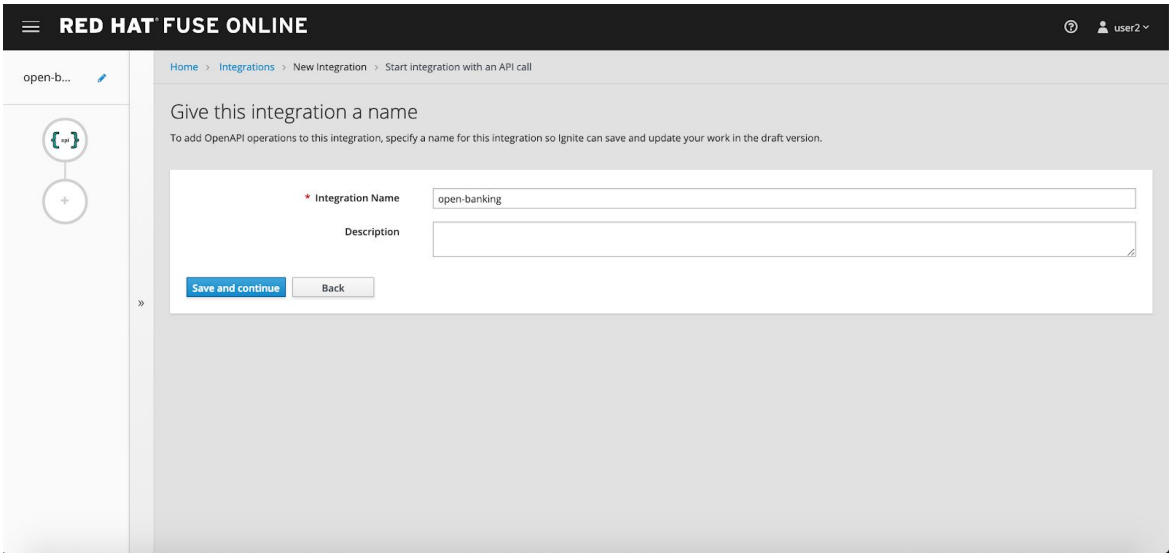
The definition is the same one seen on the Open Banking solution portal for Open Data APIs.
There is a validation happening on the API definition, but no error was identified so we can proceed with the configuration of the connector.
To show how easy it is to correct definitions click on “Review/Edit”.



This opens a window on Apicurito which is a scaled down version of Apicurio, our API Design platform. It is fairly easy to change elements graphically and also with the help of this tool an API team can start with a Design First approach when configuring the API. Also the same team doesn't need to know about the rules around OpenAPI specifications thanks to this tool.

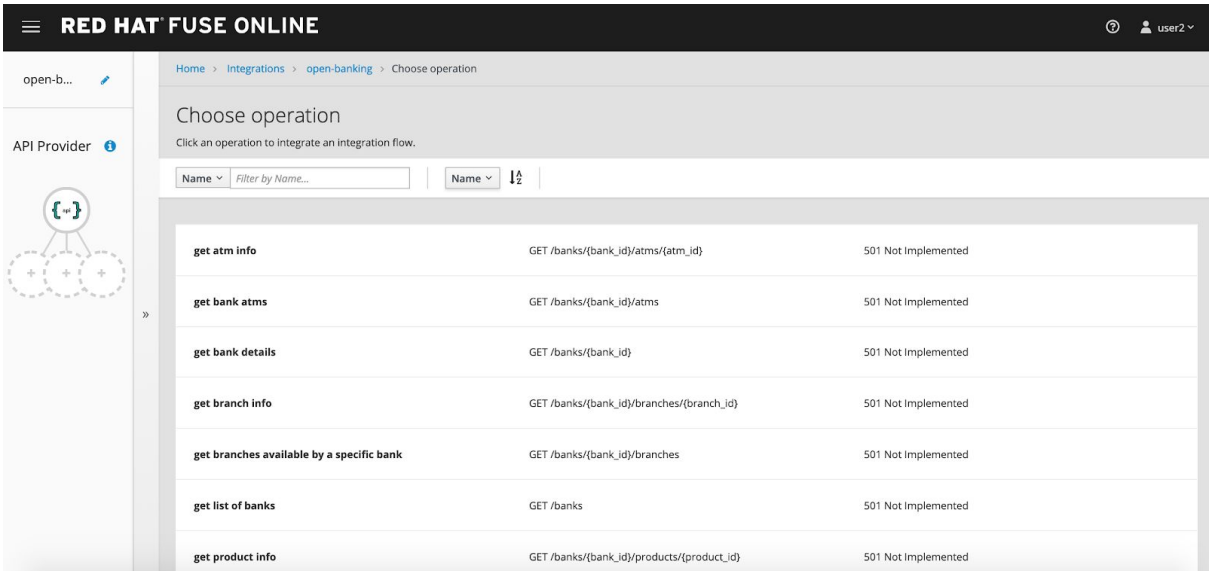
The service is exposed without any protection (that's one of the reasons we are going to be using 3scale later on) since the API definition is not adding any authentication on top of it by default.

Click on "Back" and then on "Next". Provide "open-banking" as the "Integration Name"

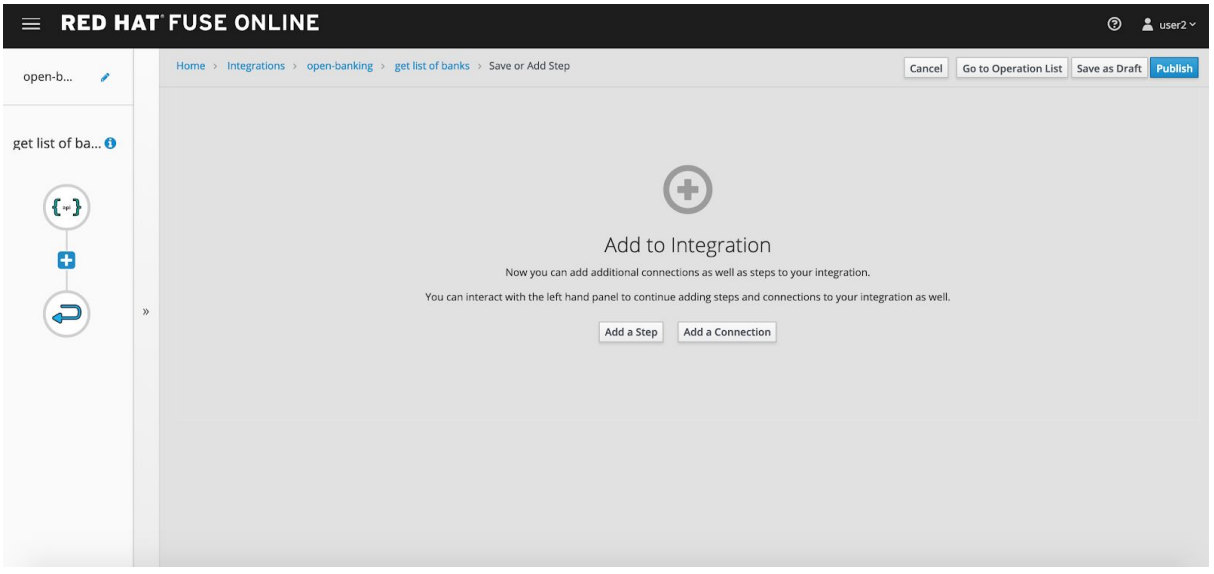


Click on “Save and continue”.

We are going to map just one of the endpoint exposed, in particular the *get banks* (/banks) one.



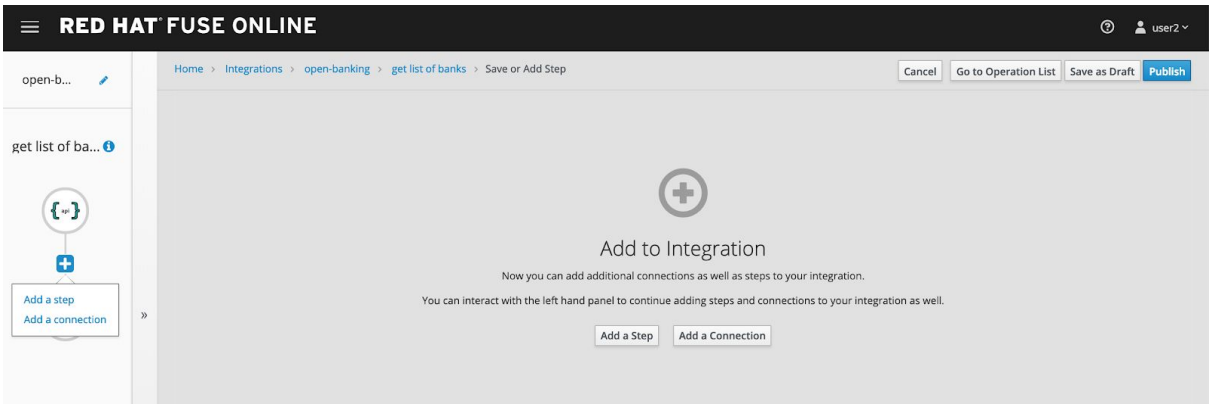
Click on **get list of banks**



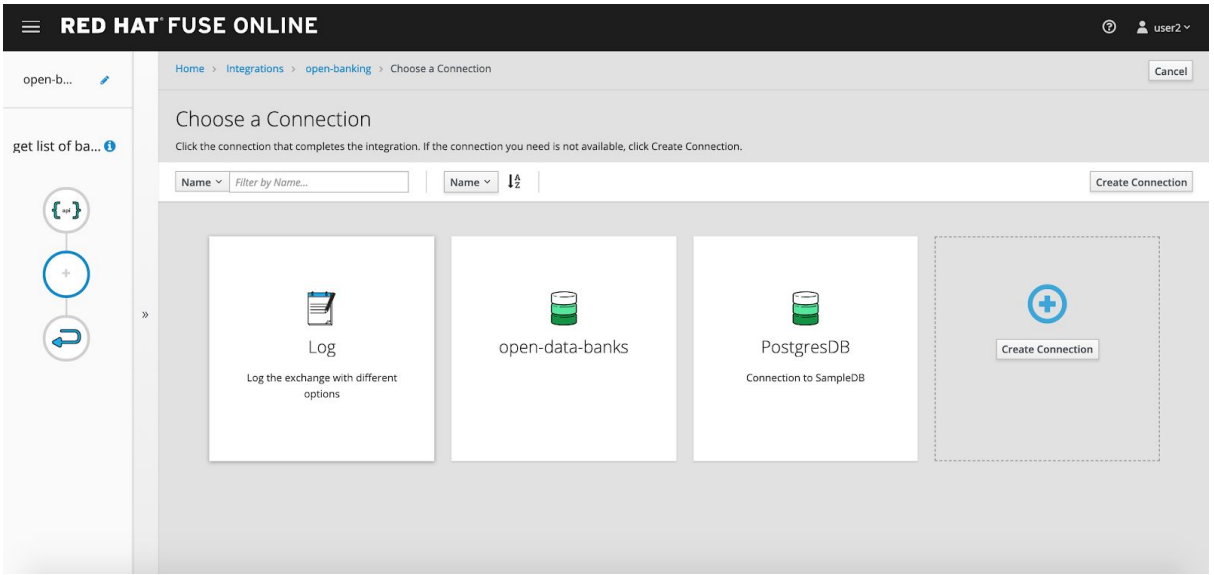
We now have a dumb pipe which connects an endpoint to receive user requests and return always 200 (all OK) - the return blue arrow symbol. Not very useful integration but a starter!

Let’s connect this front end to the database we previously configured as a terminating connection.

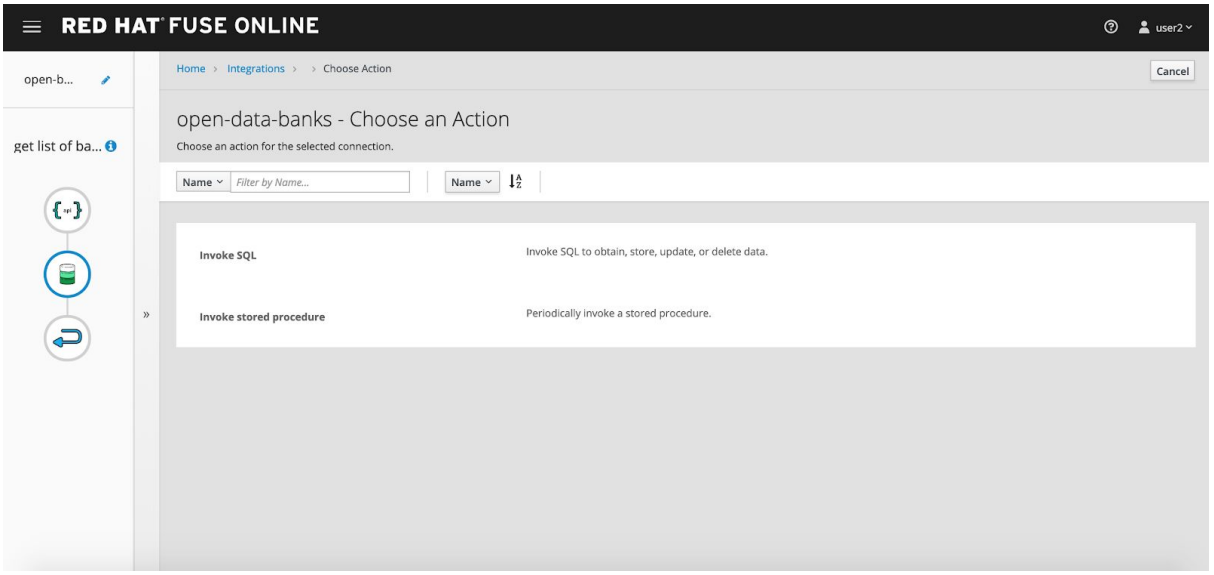
Hover over the blue + symbol and click on “Add a connection”.



Click on the previously configured data source “open-data-banks”.

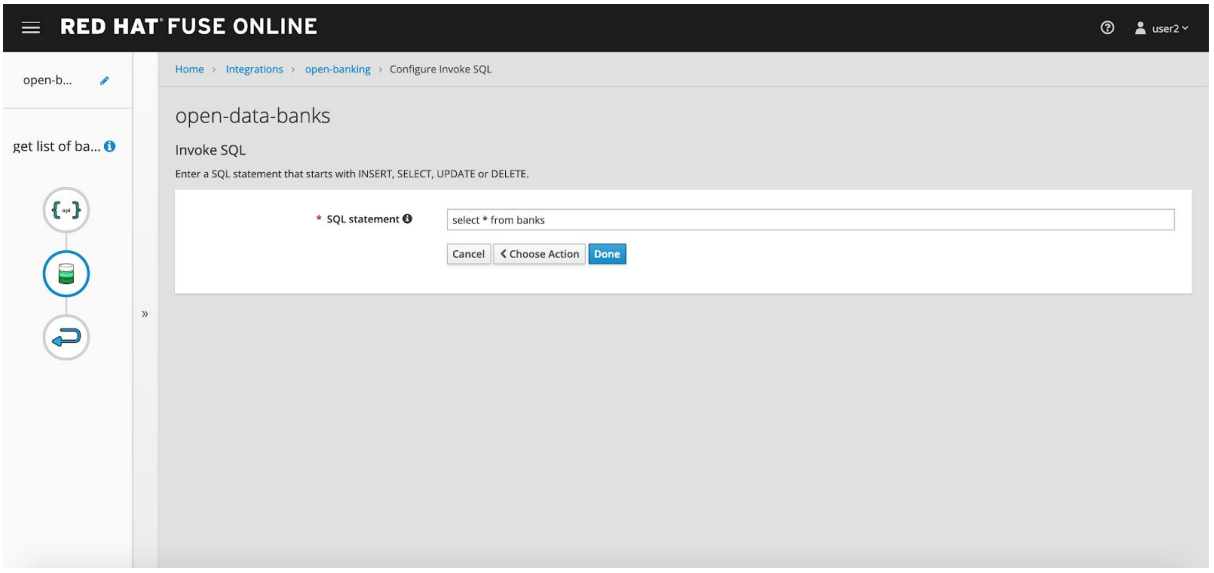


And click on “Invoke SQL” to invoke a simple SQL statement to return the data from a single table.



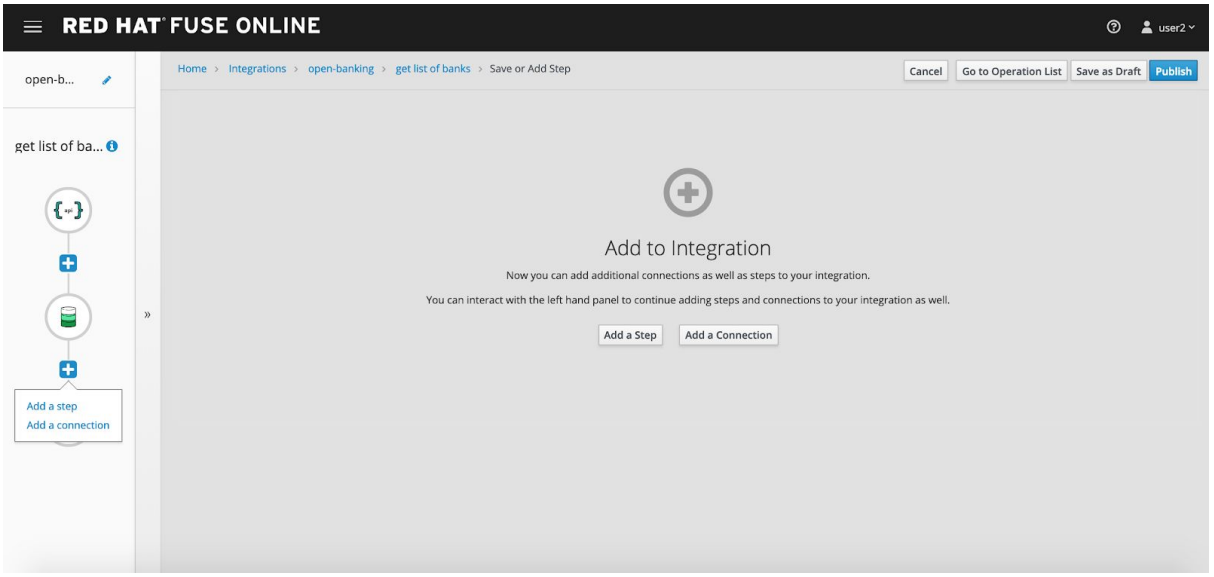
Add the statement “***select * from banks***” into the “SQL statement” field.

And click on “Done”.



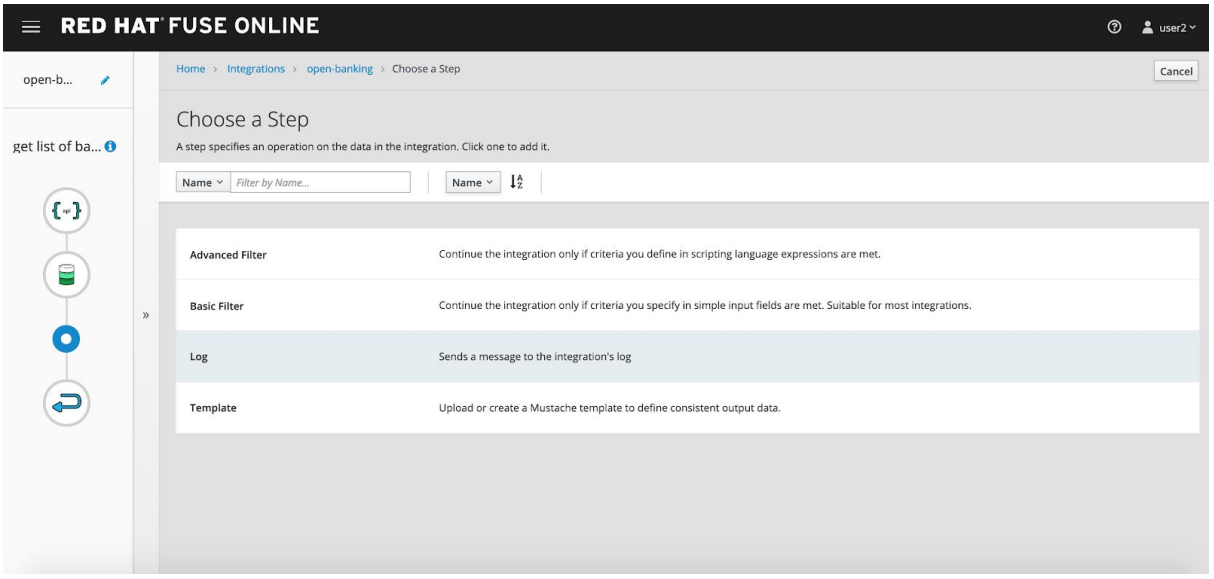
And now let’s add a simple log of the requests coming through.

Click on “Add a step” underneath the second + symbol.

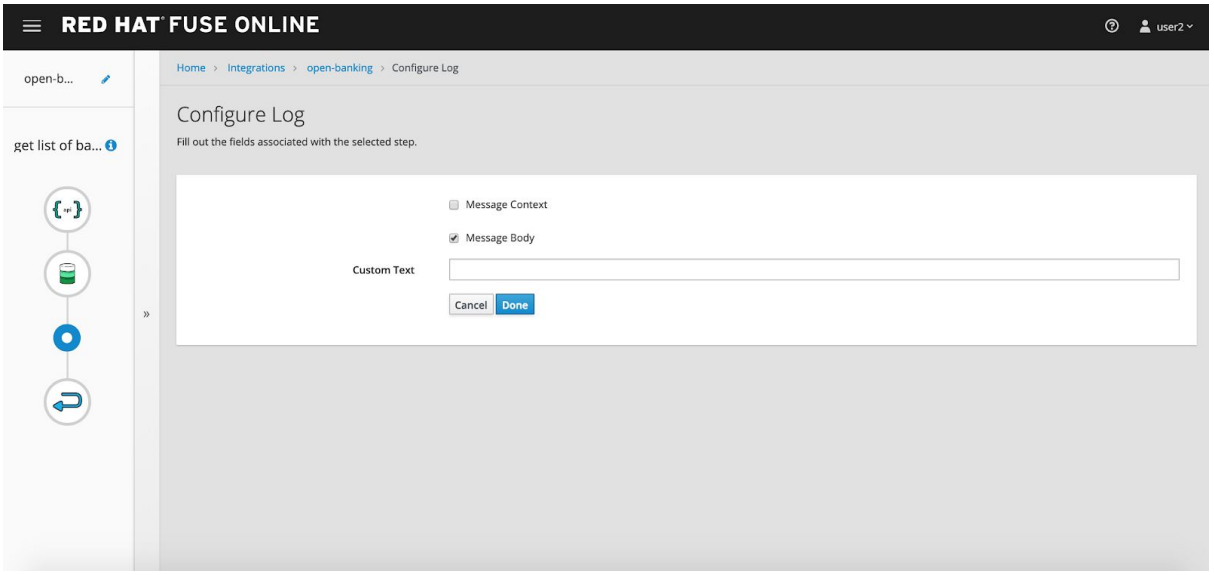


We are going to be sending a copy of the responses coming through to the integration log.

Select the “Log” step.

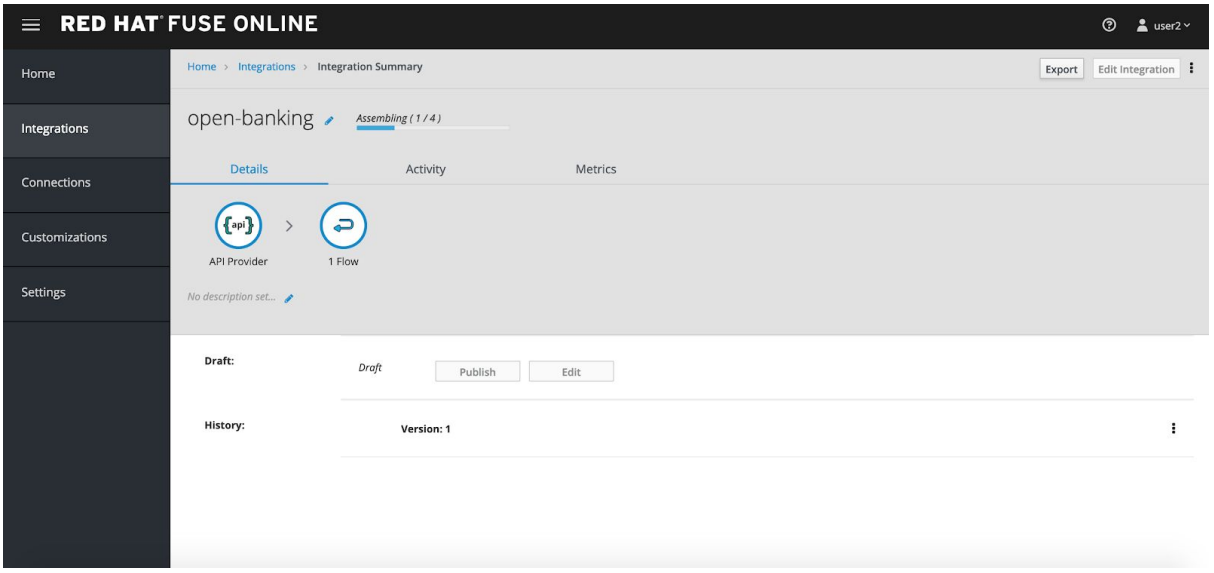


Select the “Message Body” option and click on “Done”.



We are now ready to deploy and expose this integration in our platform, to use it.

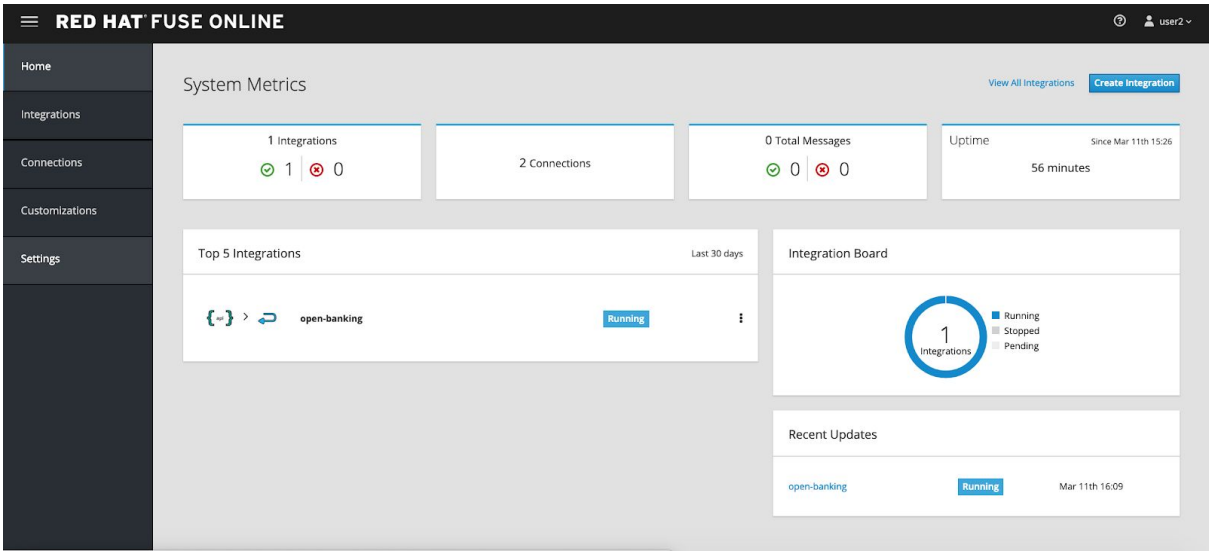
Hit “Publish”.



You can check the progress in building the integration changing through phases.

This will take about 5 to 10 minutes.

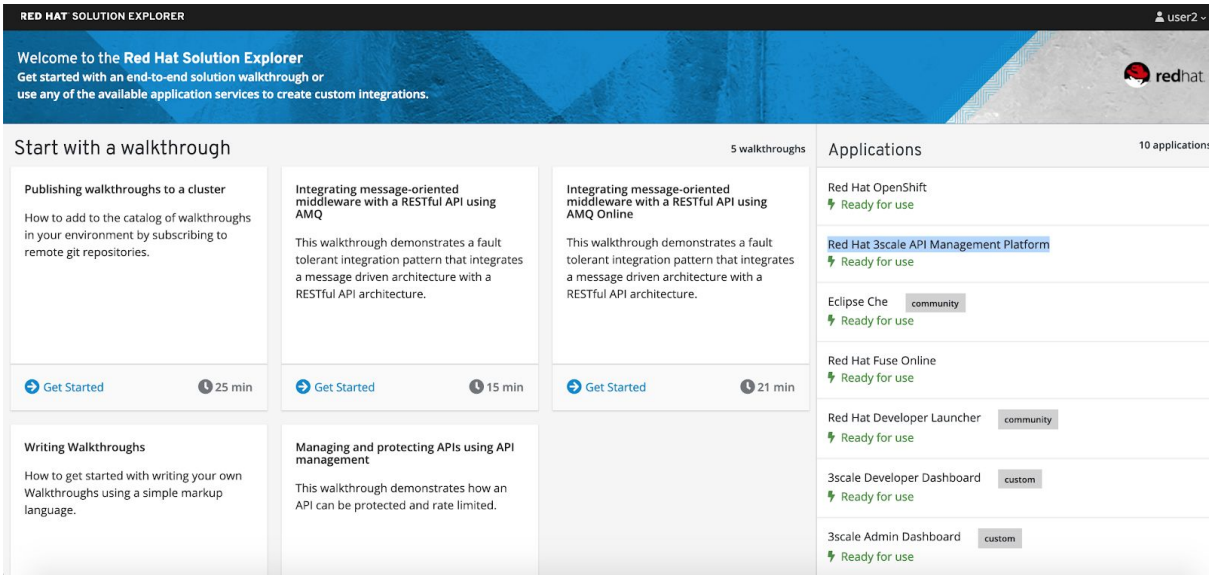
Go back to “Home” to view the recently deployed integration.



3SCALE API MANAGEMENT PLATFORM

Now let's manage our newly created integration endpoint in 3scale.

Open 3scale by clicking **Red Hat 3scale API Management Platform** link in the tutorial dashboard of the **Red Hat Solution Explorer**.

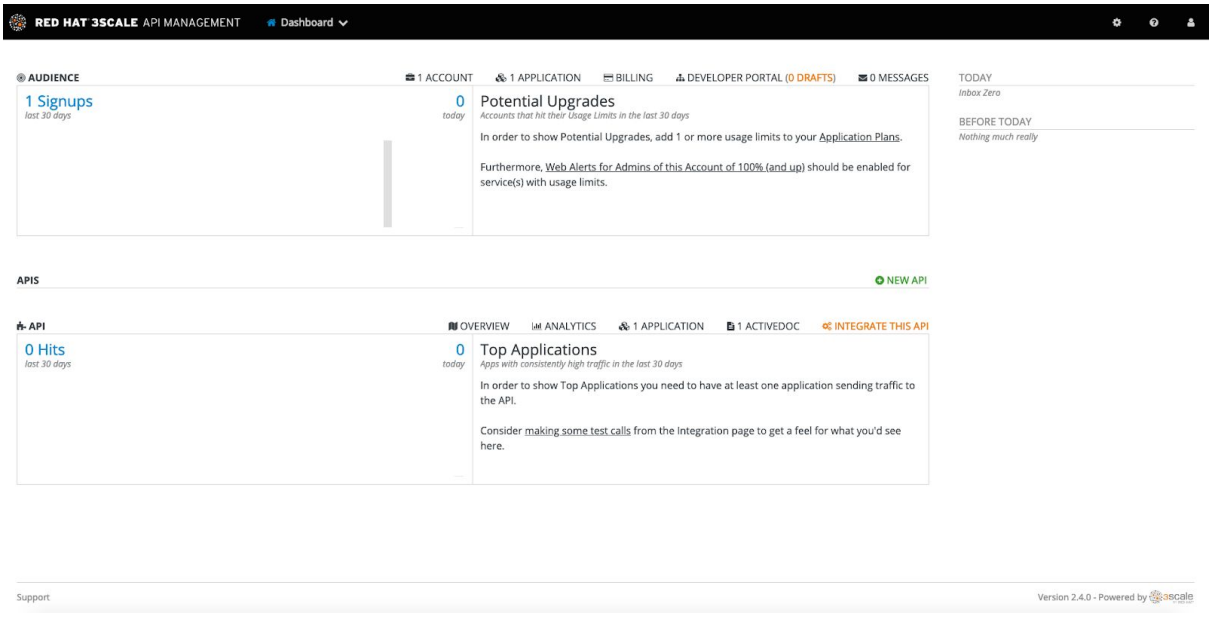


You will be presented with the login window and can use you previously used user credentials.

Introducing now 3scale, for the purpose of managing these exposed APIs, securing them and tracking their usage.

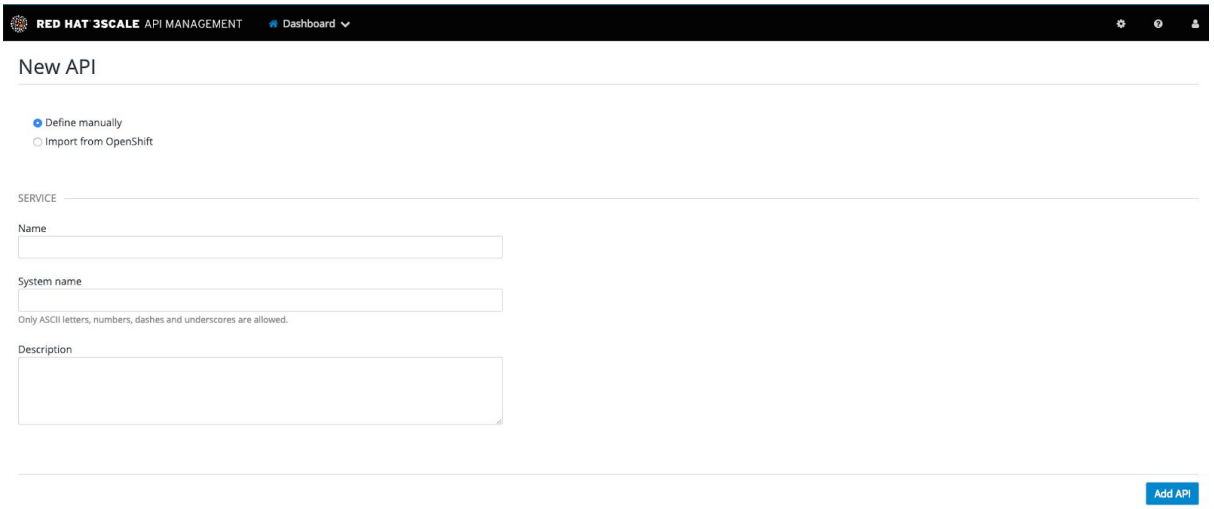
Dashboard

Everybody when logging in act as an administrator or API provider. The dashboard will show a summary of the trends in usage of the platform, in terms of developers signups, usage of APIs and message sent and received.



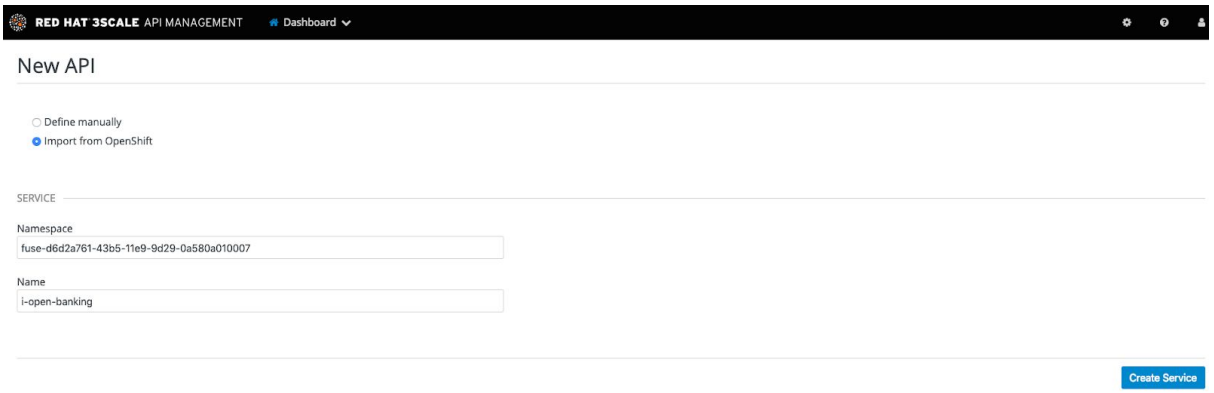
Let’s start managing and protecting the API we just created on Fuse Online.

In the dashboard click on the green button “New API” and select “Import from OpenShift” (click on “Authenticate to enable this option”). We are going to be using the auto-discovery feature we saw before. We would be helped by this feature and would save time configuring the service.

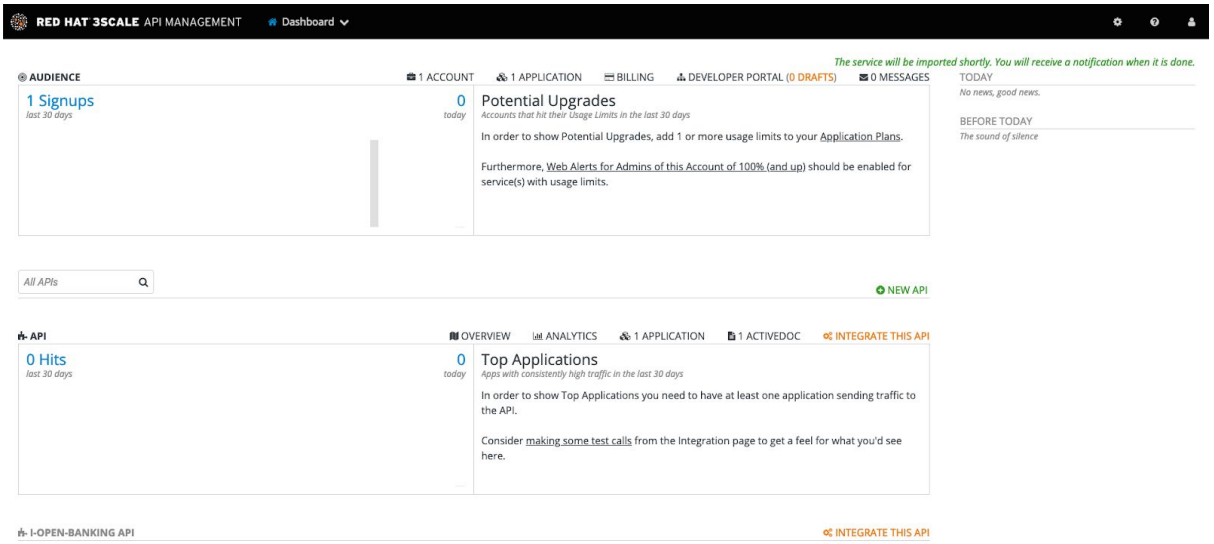


You are now presented with the permissions screen and you should click the Allow selected permissions button to proceed. This is to allow API management to go and look for services in our container platform.

You will now see the auto-discovered Fuse service appear in the Namespace field. Click the blue “Create Service” button and proceed in starting the service configuration.

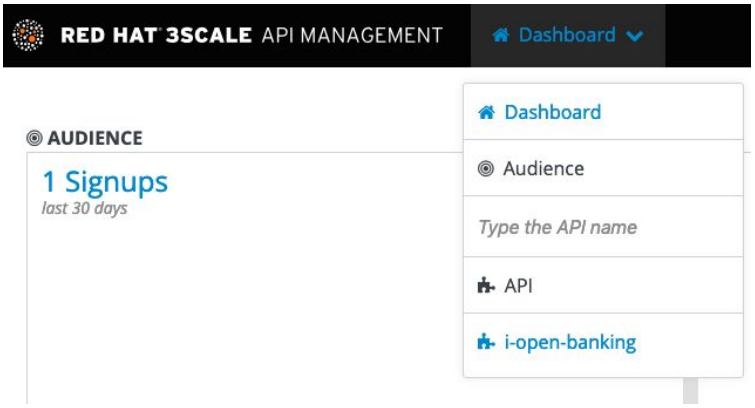


You will be redirected to the Dashboard view where a success message should appear as shown in the following screenshot.

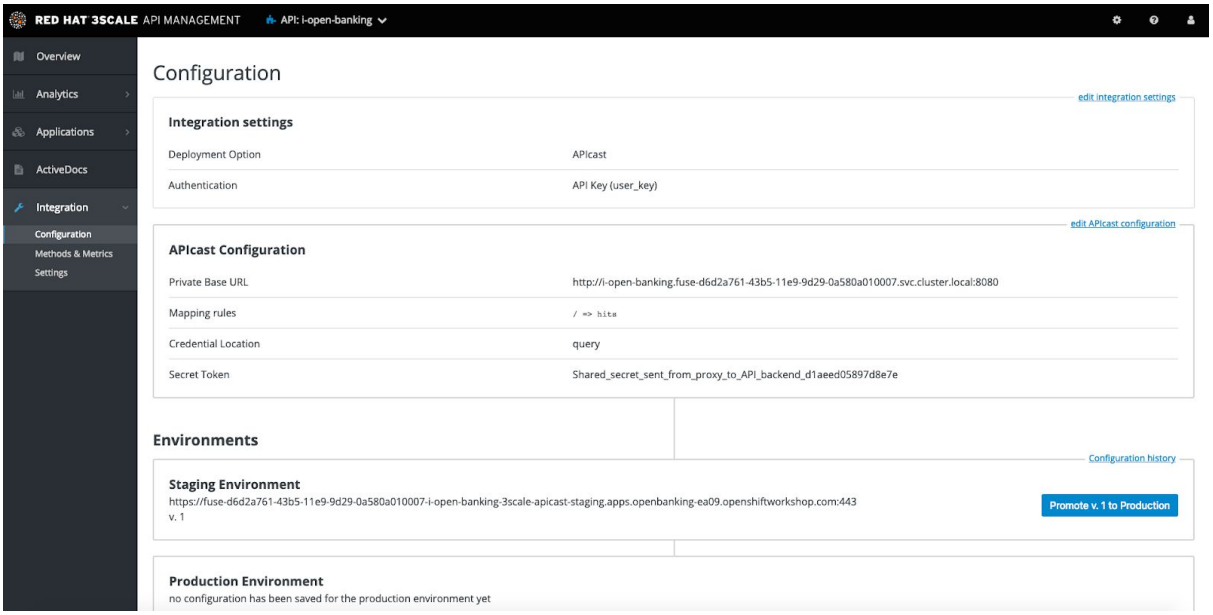


We can now proceed on changing the details of the configuration of the API and publish the update on the Developer Portal so that the public Developers can sign up for the open financial API.

In the menu click on the newly created API “i-open-banking”.

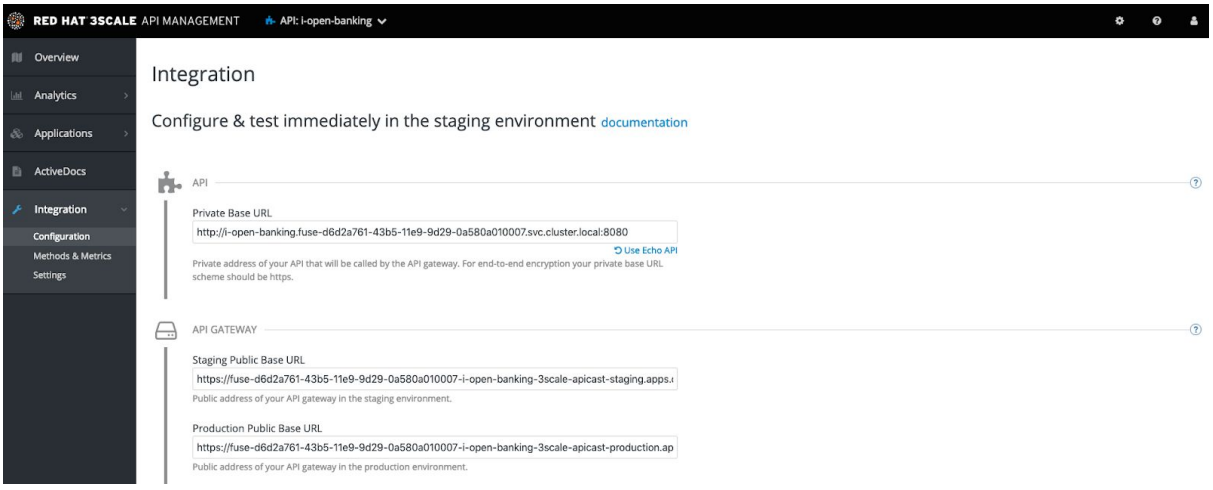


Click on “Integration” and “Configuration”.



Click on the link “edit APIcast configuration”.

This is where you configure the rest of the details of the mapped and protected Service. The private base URL should already be filled with the details coming from the auto-discovery feature.



We have the section where we map the Backend API (or in this case Integration service) as the Private Base URL.

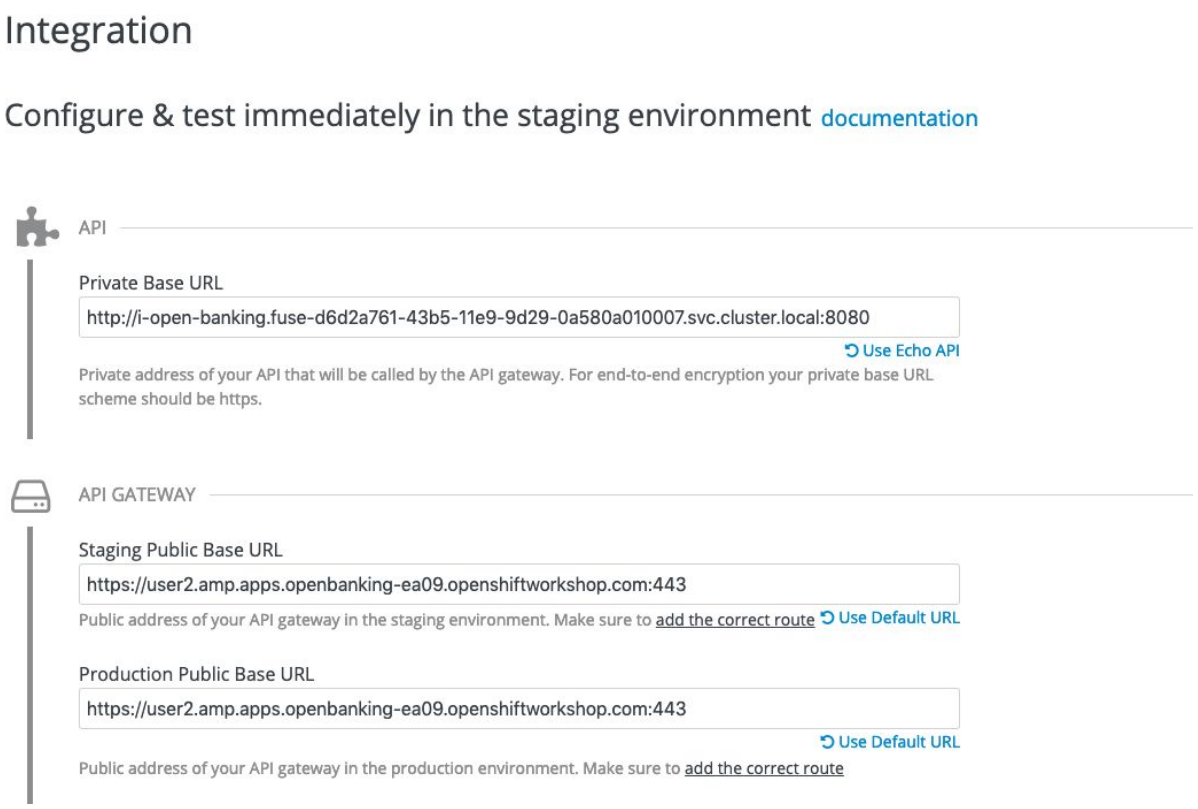
And we define two URLs where we expose the managed API on the staging fist and production gateways.

We will be changing the Staging and Public addresses of the gateway. In this case we are not going to use separate staging or public infrastructure so it can be the

same address (please notice the format of the staging and public base URL for the gateways):

`https://userX.amp.apps.GUID.openshiftworkshop.com:443`

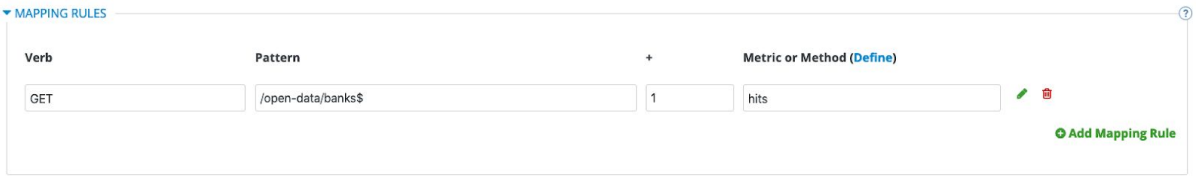
where you replace **userX** with your user ID and the **GUID** with the provided value from your instructor under “Staging Public Base URL” and “Production Public Base URL”.



We will now make sure we map a single endpoint or resource in 3scale and disallow any other endpoint (i.e. the other endpoints have not been implemented yet so we are protecting them from being exposed).

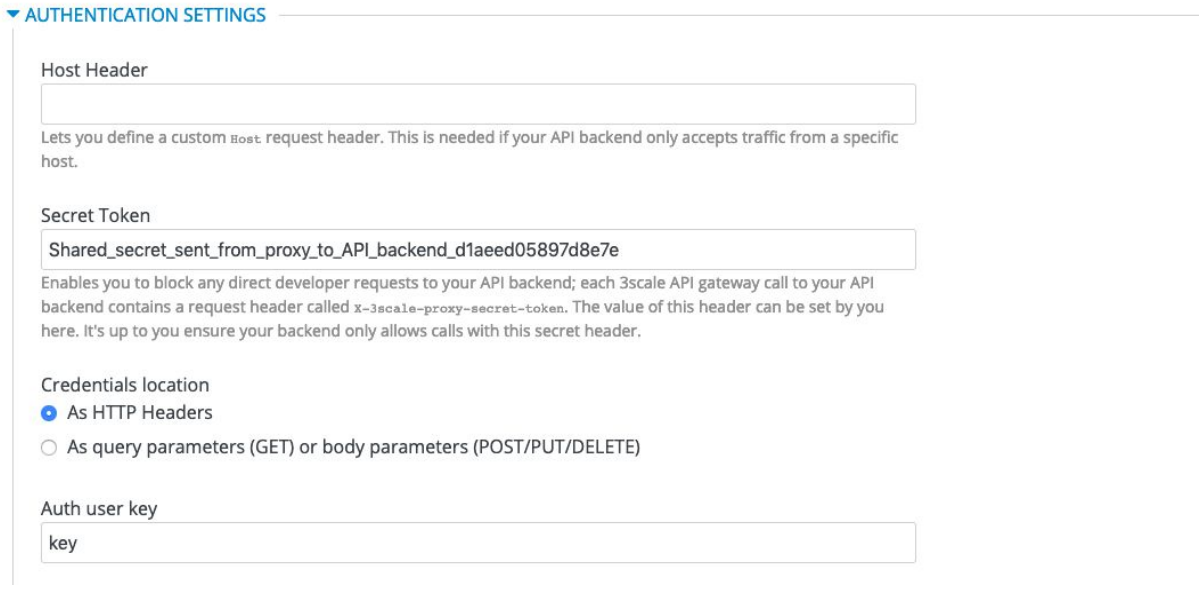
Under “Mapping Rules” click the green edit symbol.

- Under “Verb” select `GET`
- Under “Pattern” enter `/open-data/banks$`
  (notice the $ at the end of the path that will allow us to make sure users cannot improperly access any other sub-resource).
- Under “Metrics or Method” leave `hits`

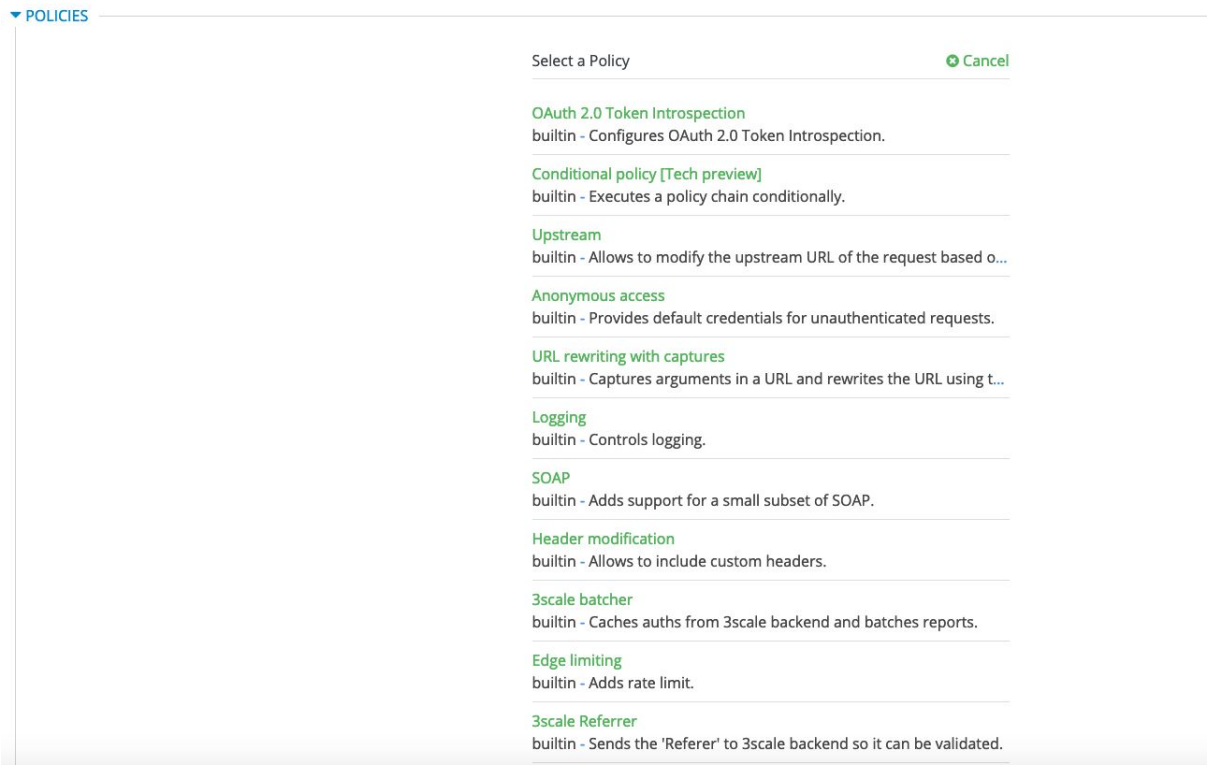


We can now check and change the configuration of authentication settings under the menu item “Authentication Settings”.

We see that we have already API key protection enabled, but we might want to pass this information as HTTP Header instead of the query parameter. We will also change the header name to ‘key’.

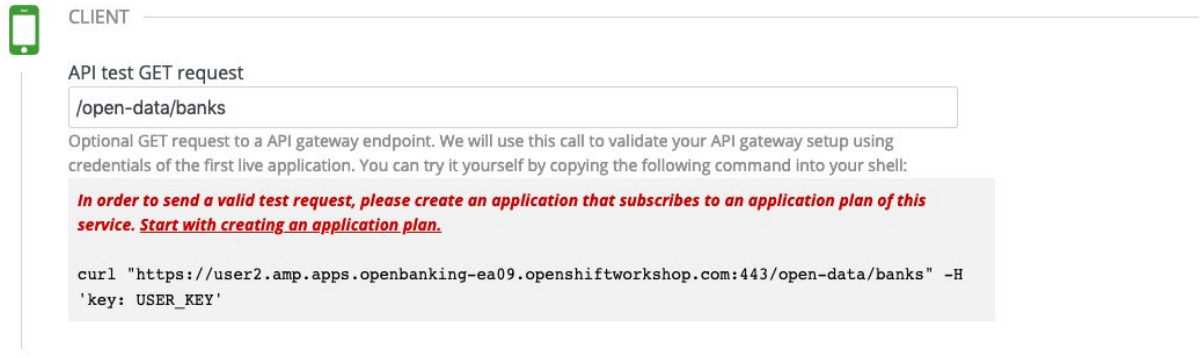


Under “Policies” you can add existing policies executed against API calls at the gateway level. Click on “Add Policy” to view the policies. For now we are not going to add new policies though, so click “Cancel” after that.



Update the “API test GET request” field with the same pattern you are mapping above (excluding the $):

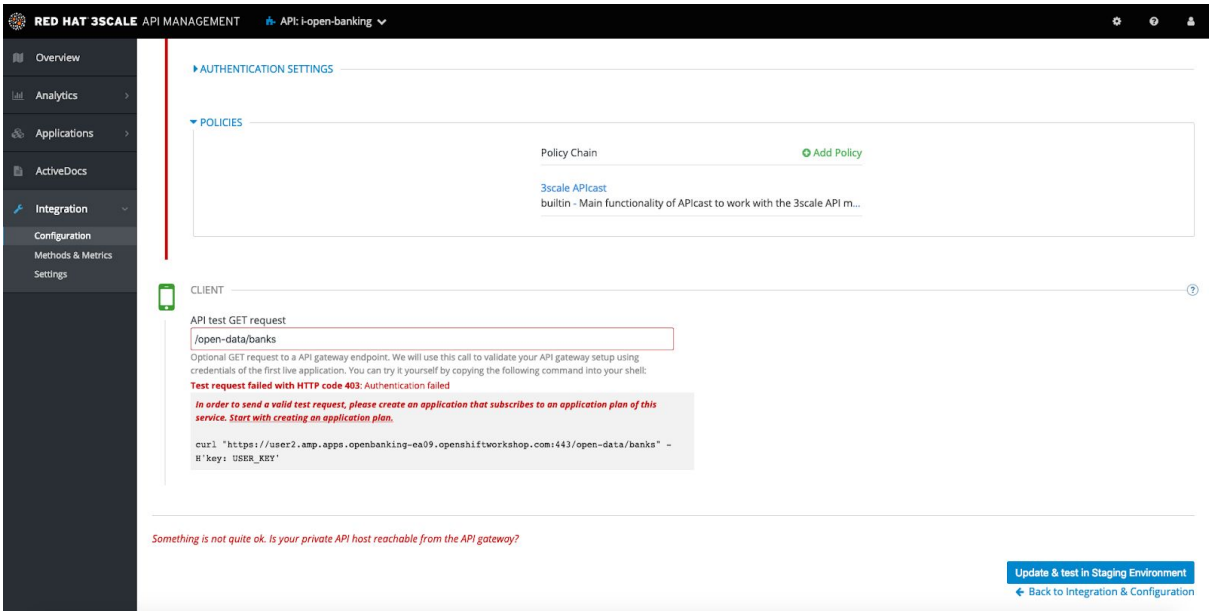
```
/open-data/banks
```



Hitting the big blue button will allow you to do two things at once:

- Update the service configuration on the platform
- Test the configuration just uploaded to the gateway.

The second one will fail since we are not providing any valid key, so we will get unauthorized request but the gateway will receive the updated configuration in any case.

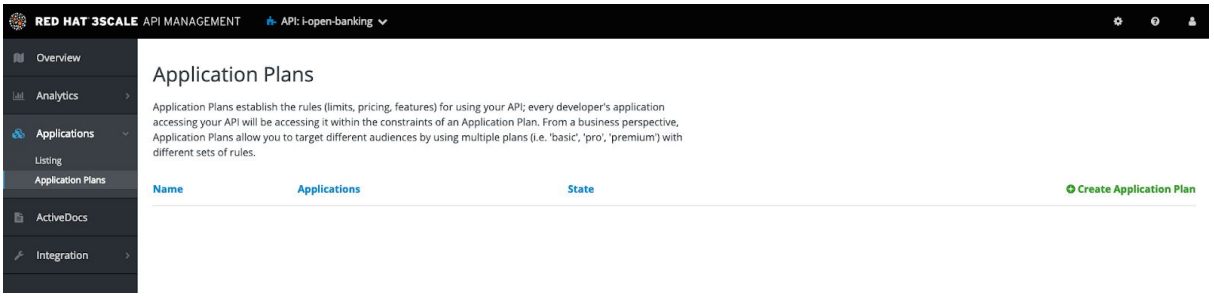


We will now fix the test request error as advised by the warning message.

Let’s switch to explaining the role of API contracts of Application Plans.

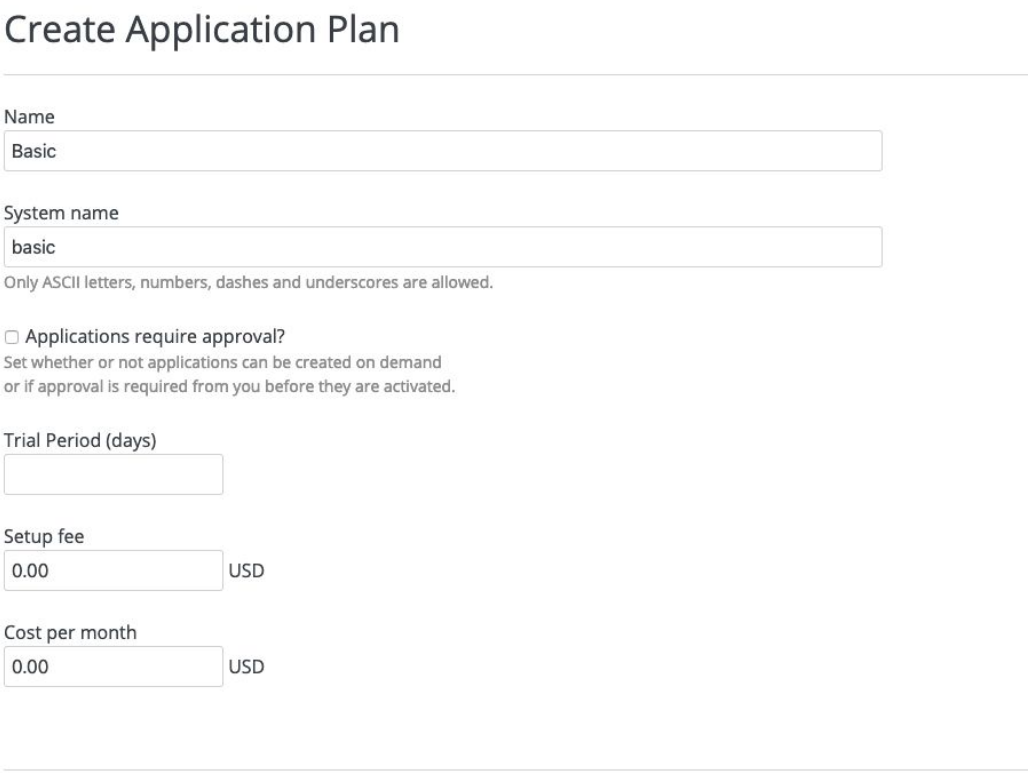
Within the red error message a link is generated *“Start with creating an application plan”*. Click this link and you will be redirected to the Application Plan overview.

Click on the green link “Create Application Plan”.



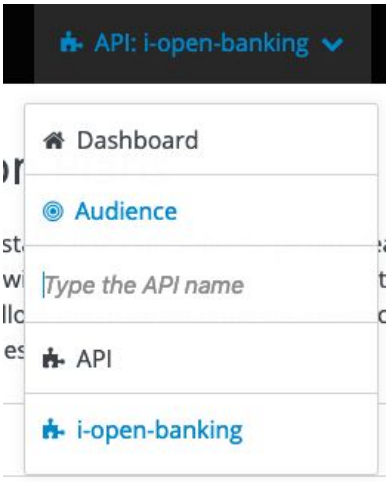
Fill out the Name field (“Basic”) and System Name field (“basic”) form and click the blue button to submit the form.

You can safely ignore for now the monetization options.

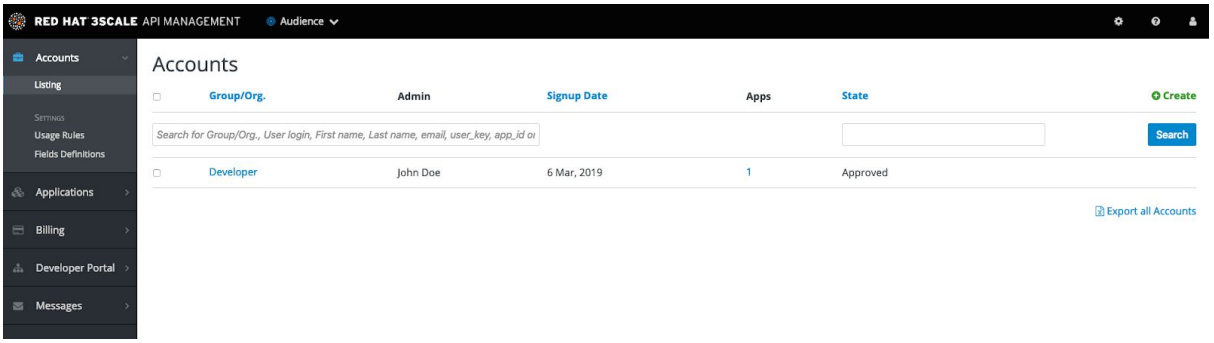


We see that we have one API contract (or Application Plan), but no application associated to it. The application plans are in “hidden” state by default, so let’s publish this one (by clicking on “Publish”) so that it is usable and visible on the Developer portal.

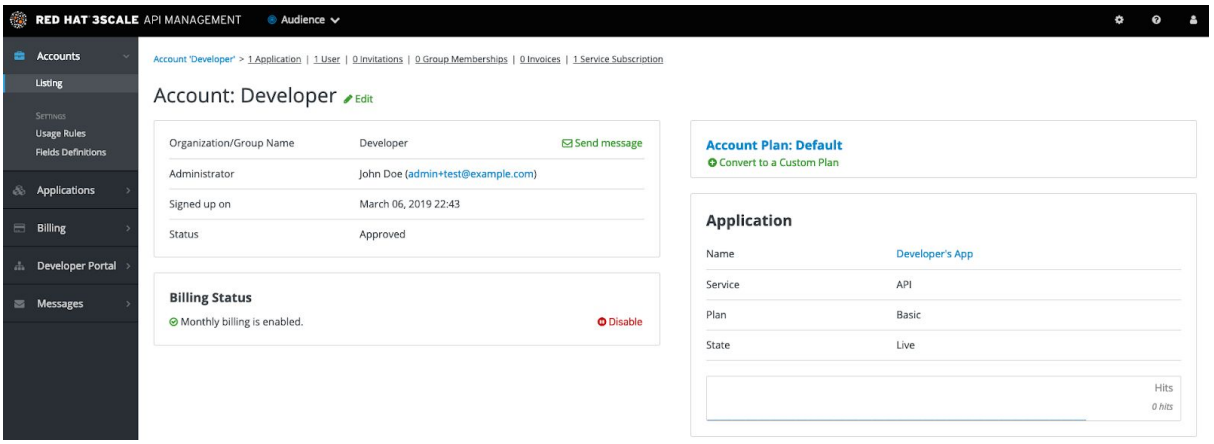
In the menu click on “Audience”.



From here we can see how we can - as Provider - approve or deny Developers’ Accounts registrations.

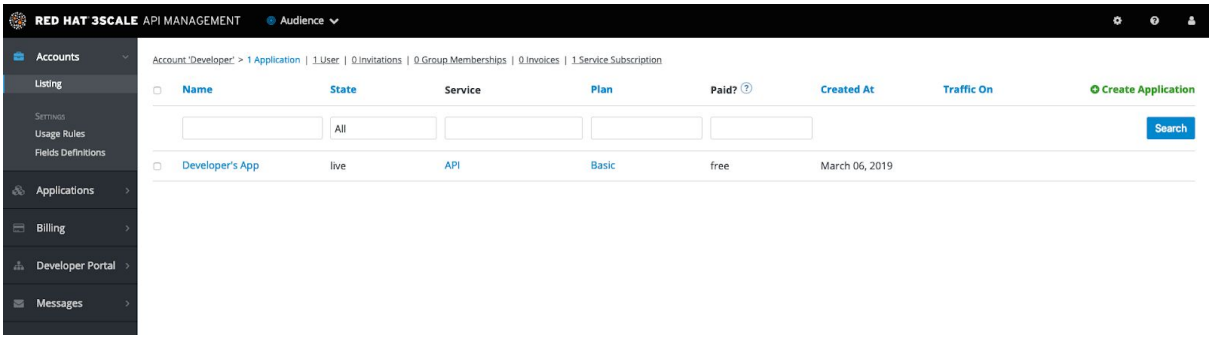


Let’s click on the default “Developer” Account.

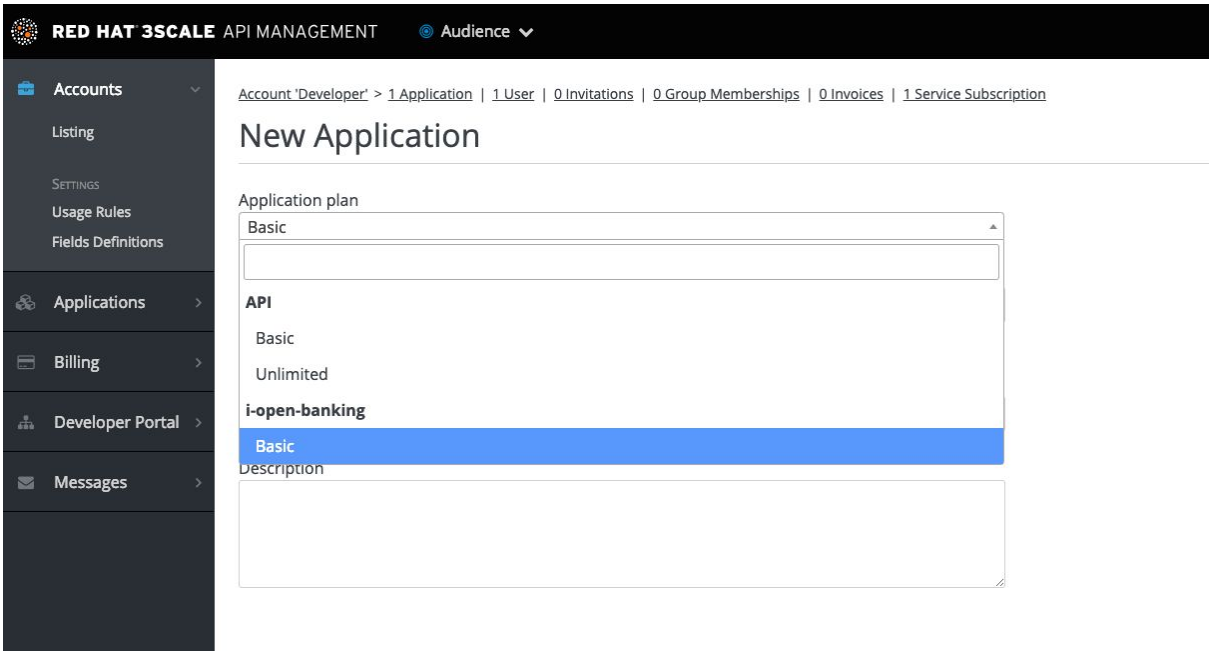


We can see that the Developer has the default application associated, but it’s subscribed to the default Service. We can also see the Developer user details.

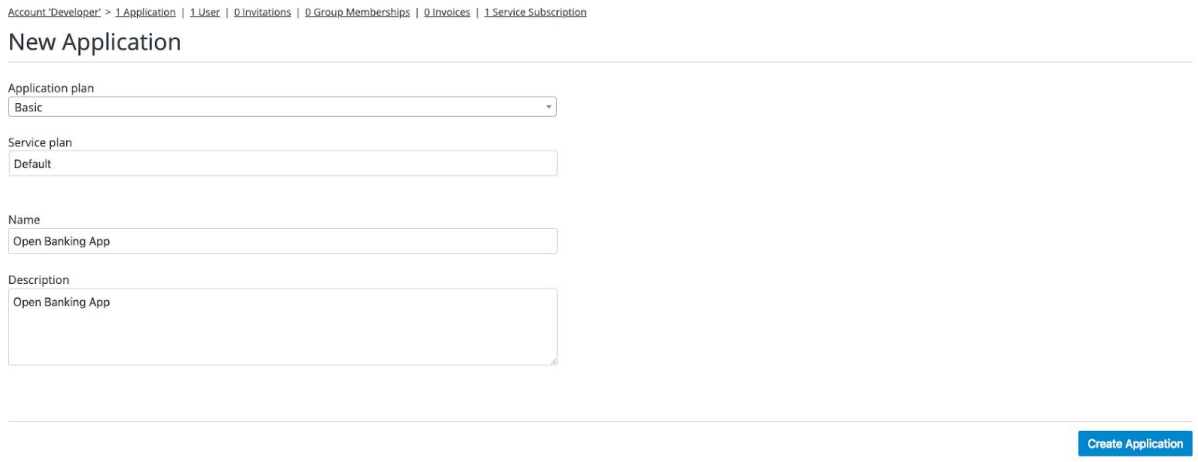
Let’s click on “1 Application” in the top level navigation and then on “Create Application”.



Here we can now subscribe the application to the Application plan “Basic” we created on our new Service “i-open-banking” from the drop down field available.

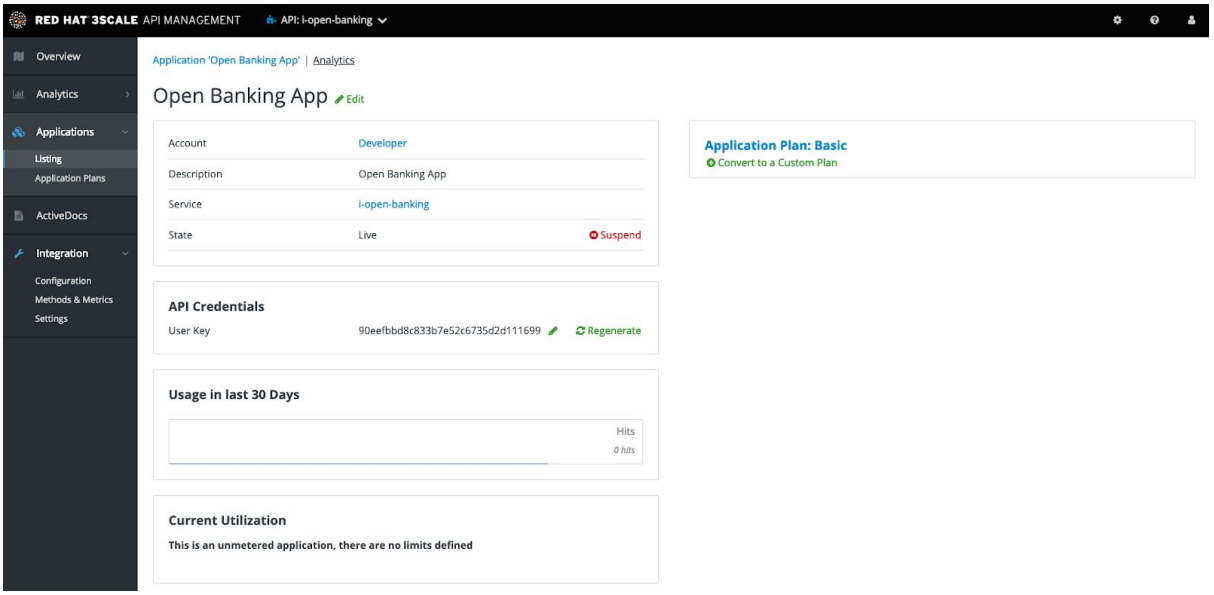


Let’s fill in the rest of the fields with the name “Open Banking App” and click the blue button “Create Application”.

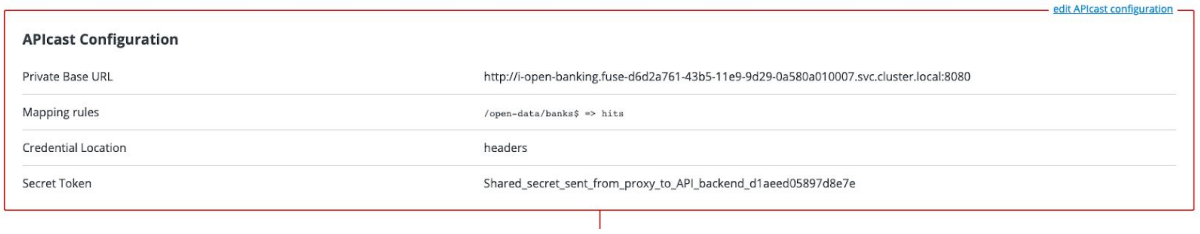


We now have an assigned key so we can go back to the Configuration window of the API service and make a successful test call.

In the menu on the overview window click on “Integration” and “Configuration”.

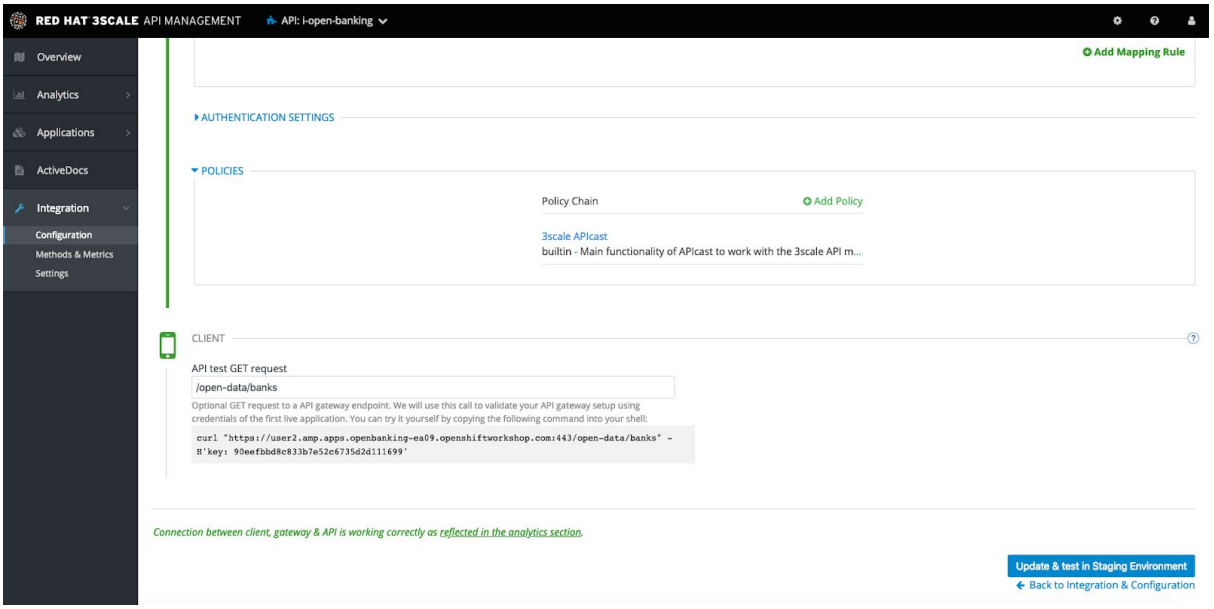


Click on “edit APIcast configuration”.



Scroll down to the bottom of the page and click on “Update & test in Staging Environment” again.

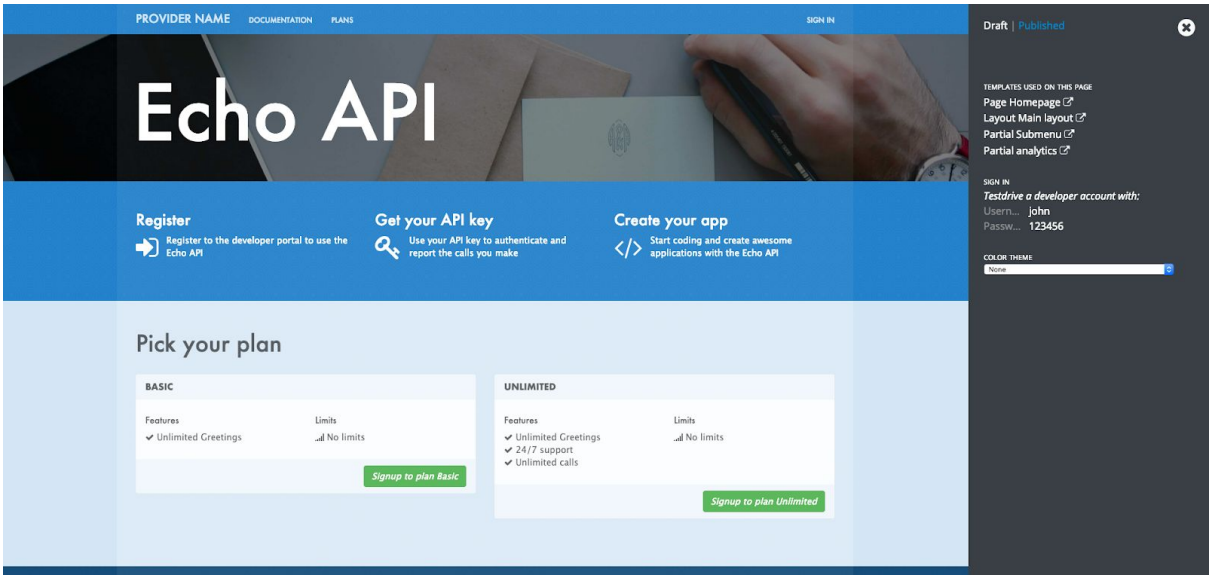
We now have a pre-populated key in the example curl statement, let’s try again testing the deployed configuration.



As we can see we turned the testing into a success.

Let's switch to the developers' point of view by accessing the Developer portal. You can access it by selecting in to the top menu in "Audience" -> "Developer Portal" -> "Visit Portal".

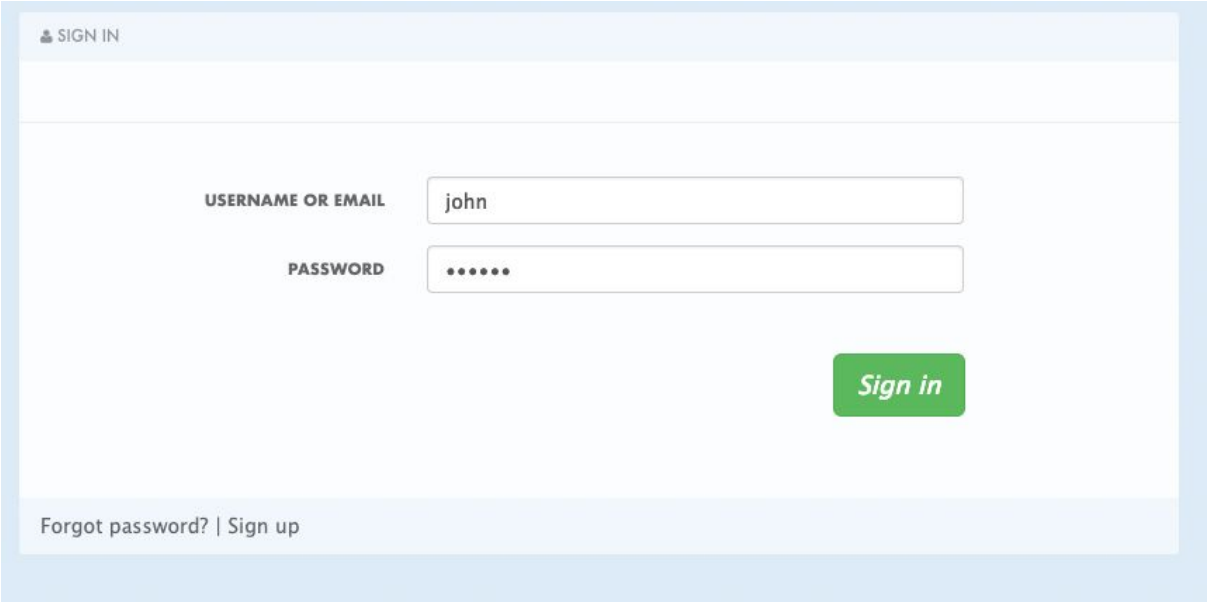
The sidebar allows us to edit pages of the Developer Portal live, but we are not interested in it now so we can close it.



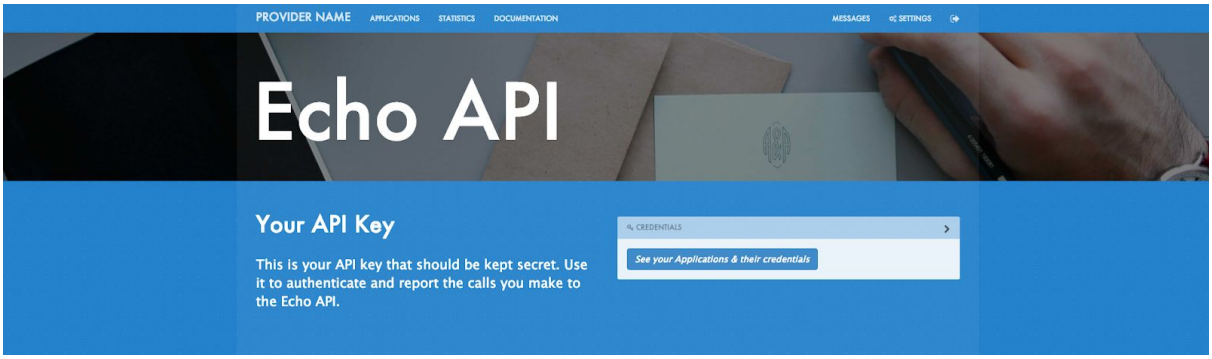
Let’s sign in with the default user credentials provided in the sidebar. This is the default developer user, created for the default developer account.

The username is `john`

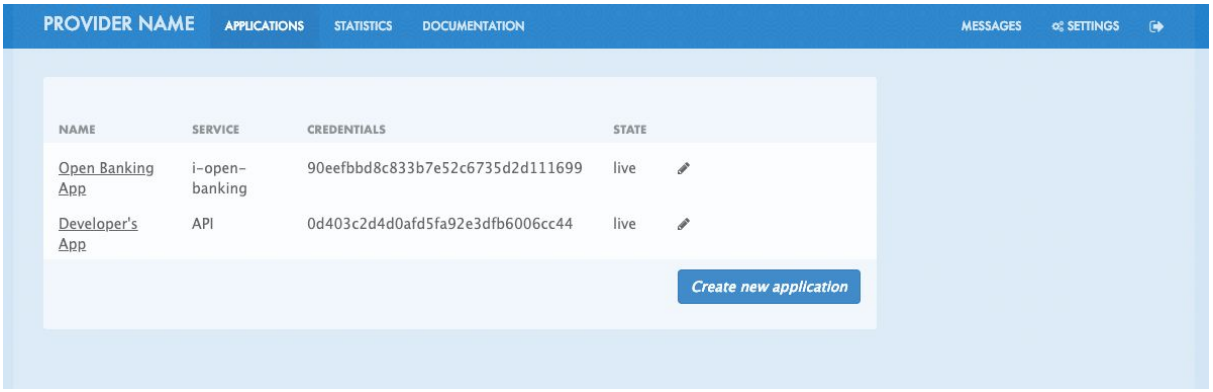
The password is `123456`



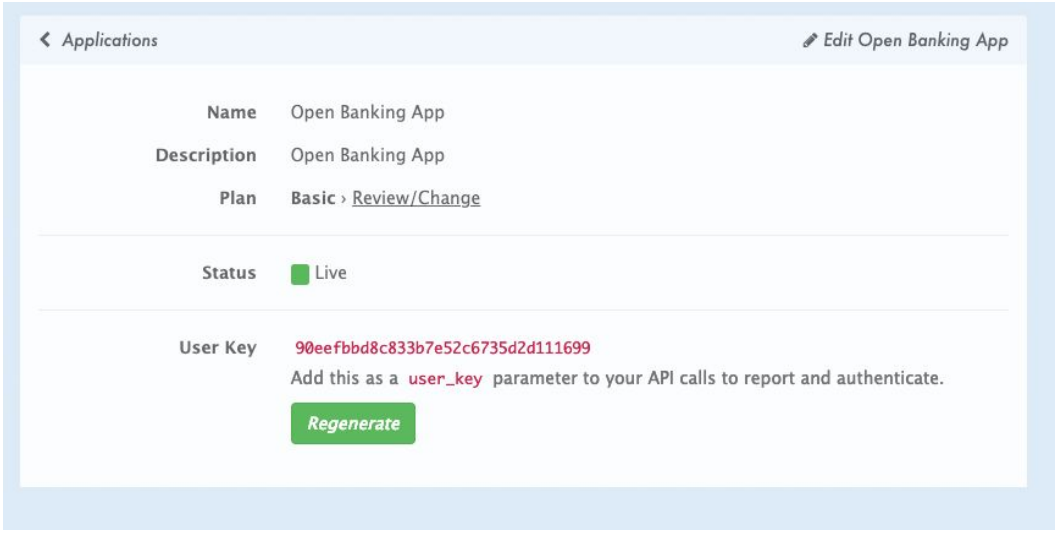
We are now logged in the developer’s dashboard.



Let’s see the applications by clicking on “See your applications & their credentials”.



Click on “Open Banking App”.



You can now use the credentials that were associated with the application and test the protected service.

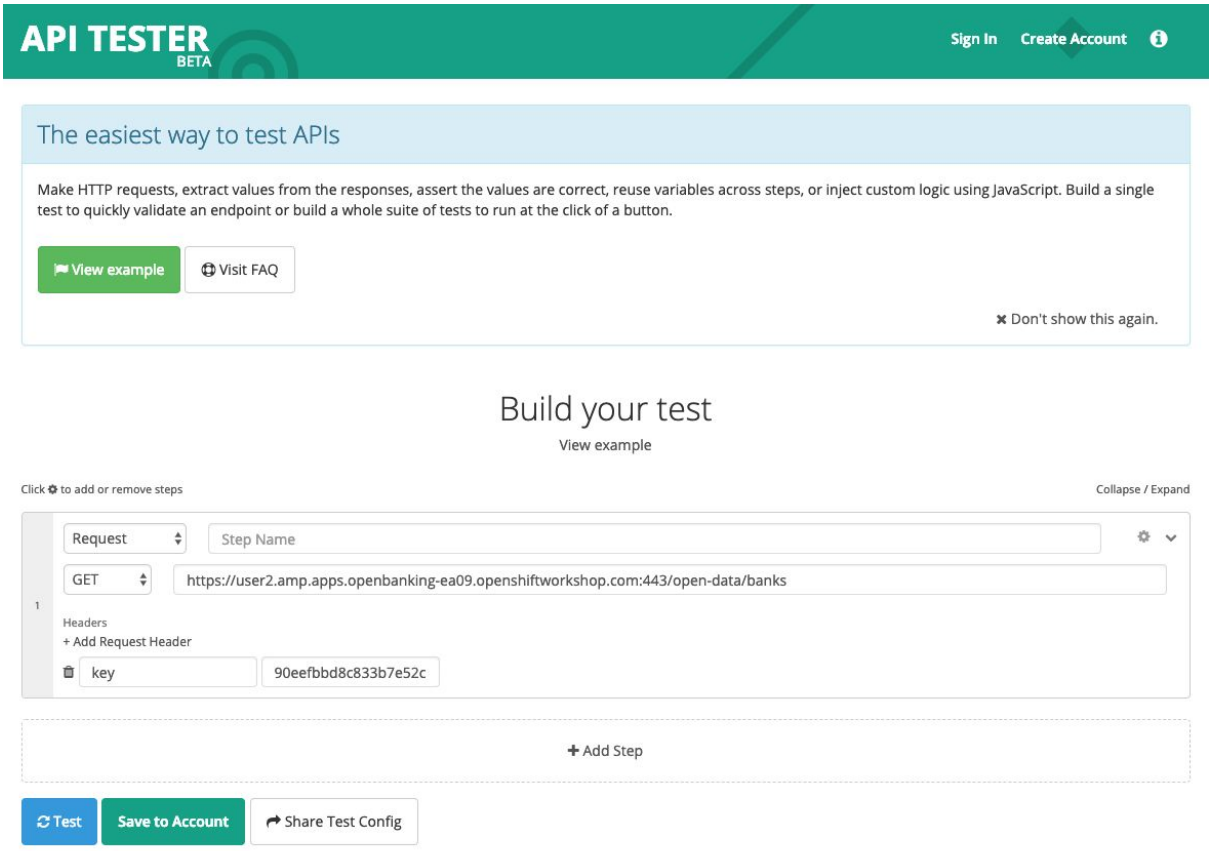
Let’s move to the online API testing tool, https://apitester.com

Use the URL for your API gateway:

```
https://userX.amp.apps.GUID.openshiftworkshop.com:443/open-data/banks
```

where you replace **userX** with your user ID and the **GUID** with the provided value from your instructor.

Also remember the the key Header and the associated value.



Click on “Test”.

As we can see we succeed with 200 OK !

PASS

N. Virginia
Seconds elapsed: 2

Results 1.10 s

Viewing a **Request** Step 1

Message

```
[Step 1] GET https://user2.amp.apps.openbanking-ea09.openshiftworkshop.com:443/open-data/banks passed
```

Connection Information

Request

Request Headers

```
GET /open-data/banks HTTP/1.1
Host: user2.amp.apps.openbanking-ea09.openshiftworkshop.com
Accept: */*
User-Agent: Mozilla/5.0 (compatible; Rigor/1.0.0; http://rigor.com)
key: 90eefbbd8c833b7e52c6735d2d111699
```

Response

Response Headers

```
HTTP/1.1 200 OK
Server: openresty/1.13.6.1
Date: Tue, 12 Mar 2019 05:33:12 GMT
Transfer-Encoding: chunked
X-Application-Context: application
Set-Cookie: 5edba771e9ee85040f19fe8af69fd058=2573d43e5703f514b749872ff136af35; path=/; HttpOnly
Cache-control: private
```

Response Body

{"short_name":"387832 5350","id":1,"address":"425-4002 Urna, Road","long_name":"Molestie Inc."}

Practical Lab - Part 2

Let’s now improve the security of the managed integration service with OIDC. API key is not really considered a safe API authentication protocol anymore and it is vulnerable to many attacks.

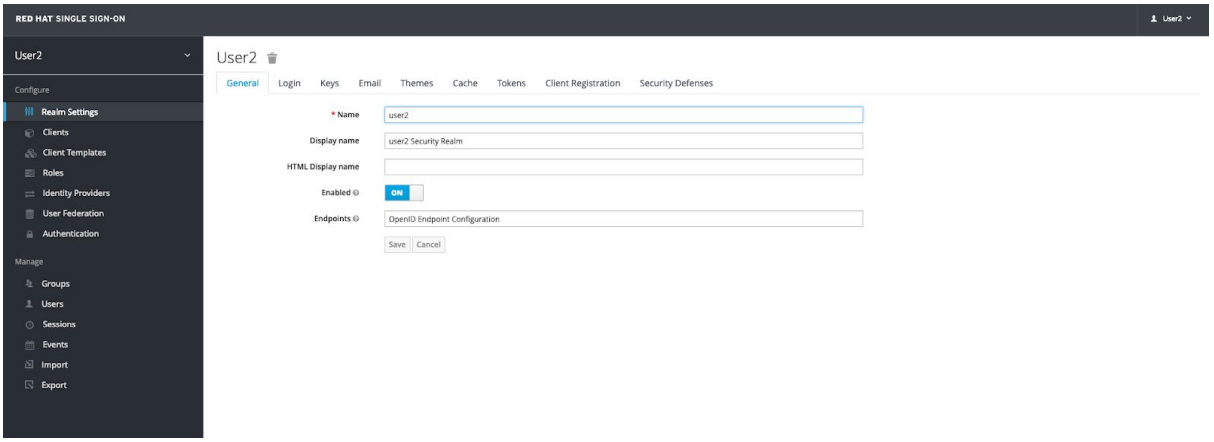
After introducing content around OAuth and OIDC, let’s see the main elements of RH SSO itself.

RED HAT SINGLE SIGN ON AND 3SCALE OIDC

Let’s start with RH SSO main dashboard
It has the following URL:

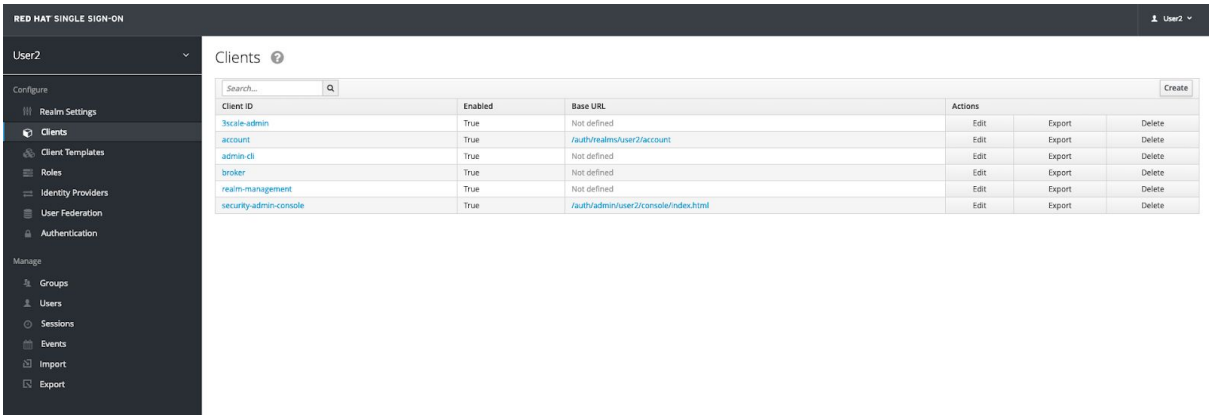
https://sso-sso.apps.GUID.openshiftworkshop.com/auth/admin/userX/console/#/realms/userX

where you replace the GUID with the provided GUID and userX with your user ID.



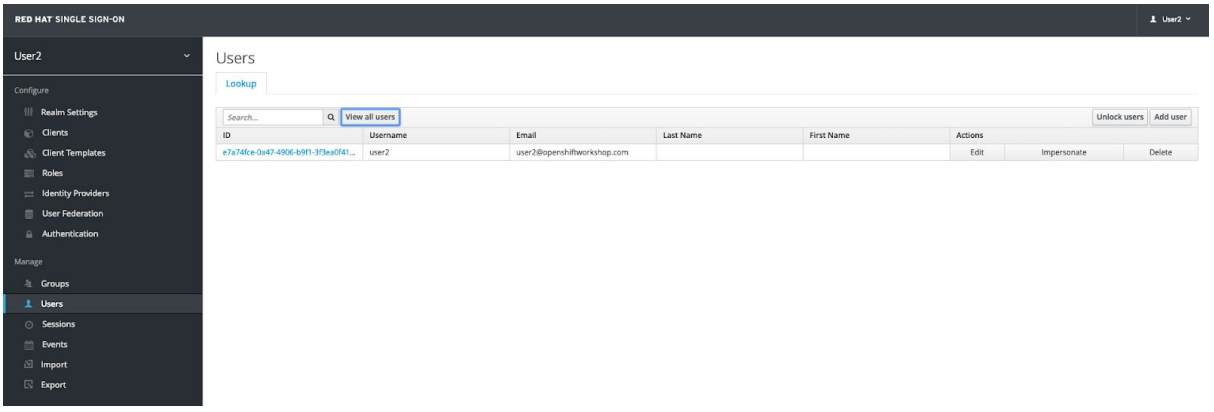
The realms are like separate instances of the platform, dedicated to separating users and applications. As we can see we can customize several aspects of the realm like the theme of the login page or the the tokens’ default parameters.

Let’s now take a look at the Clients section. Click on “Clients”.



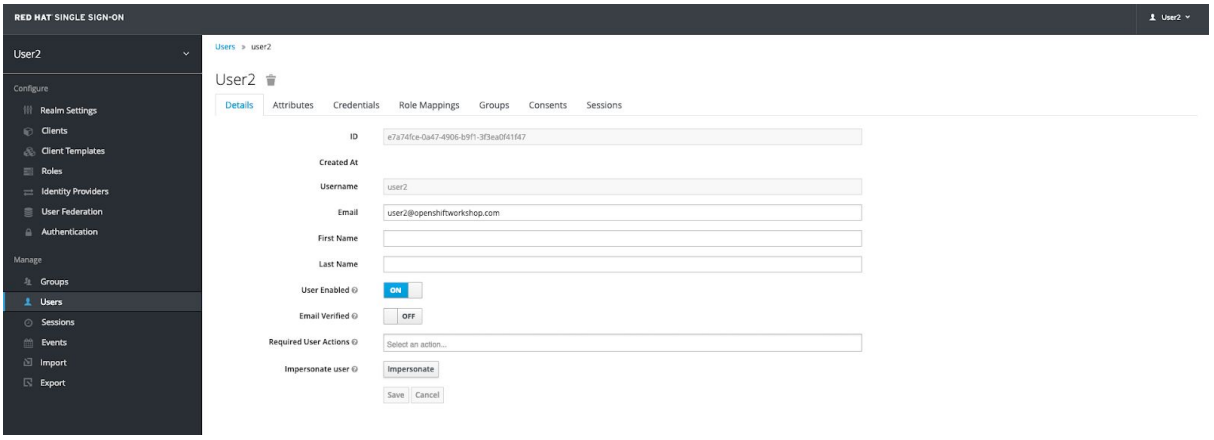
Here we can configure the web or mobile applications that will authenticate using RH SSO as an IDP (corresponding to applications in 3scale). As we can see there are some default clients dedicated to authentication in the integr8ly environment.

Click on “Users” and the link “View all users”.



Here we can see all the end users that are stored inside RH SSO, making it act as an IDM as well. This is the end user of the applications created in the Clients section and he will be able to authenticate through them. It is also the same user that each one is using to authenticate to the PaaS we are sharing.

Let’s open the users’ details.



We can see here the type of information stored along with basic user details. The user profile can be customized with additional attributes as well.

We will take advantage of one of the features available in OIDC and not in OAUTH which is dynamic client registration.

Normally to make sure an API web application authenticates with RH SSO, we would need to manually create the application on both platforms. With this feature, we let 3scale sync the applications to RH SSO, as well as obviously authenticating our API calls.

Let's create a special type of such Client in RH SSO.

Under "Client" click on on "Create".

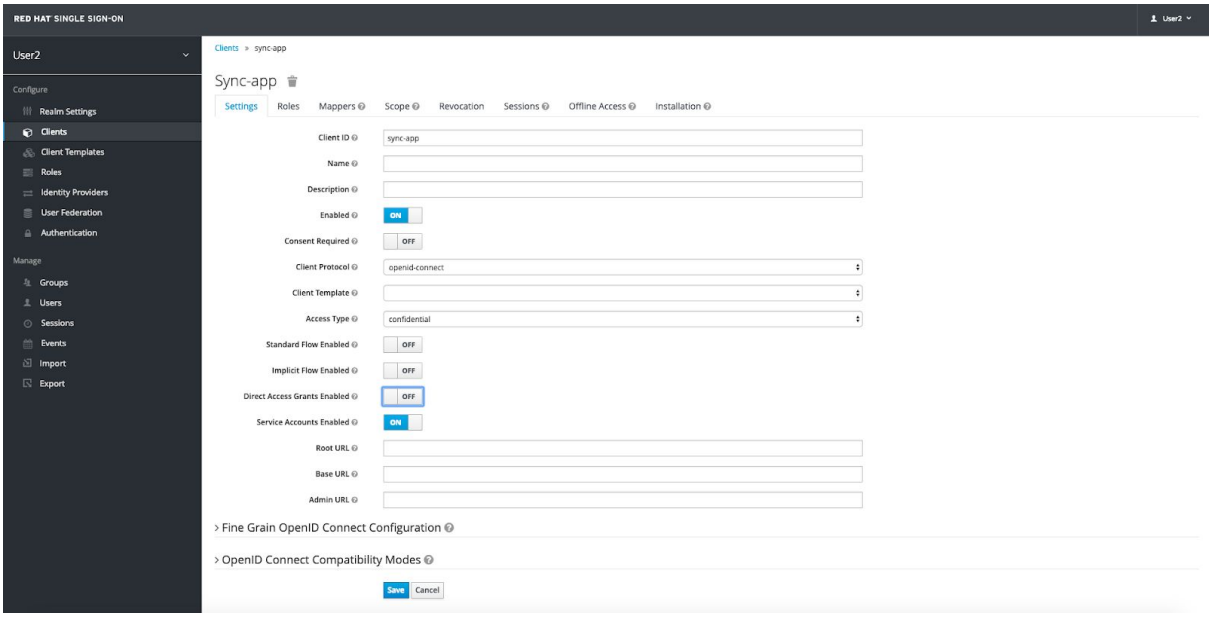
Under Client ID provide the name "sync-app" and click on "Save".

Now configure the other details required to let it communicate with 3scale.

- Under "Access Type" select "confidential"
- Set "Service Accounts Enabled" to "ON"

- Set “Standard Flow Enabled” to “OFF”
- Set “Direct Access Grants Enabled” to “OFF”

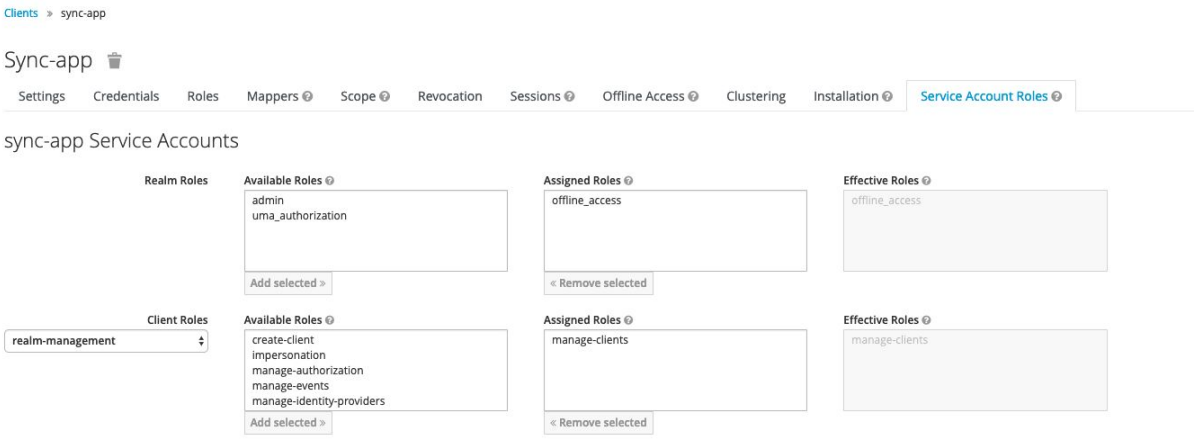
Click on “Save”.



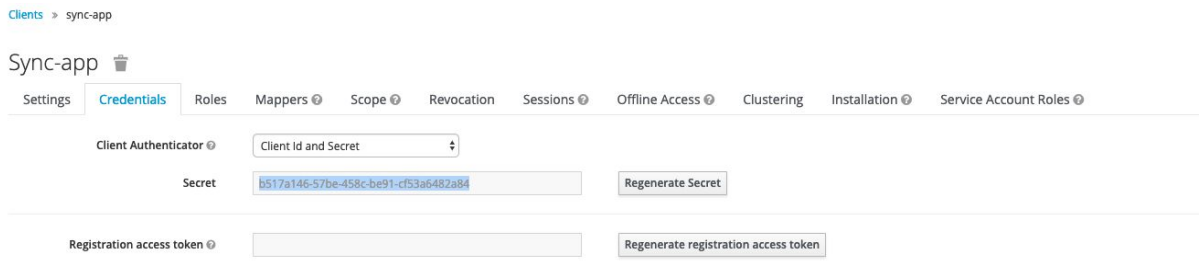
Move to the newly activated “Service Account Roles” tab.

Pick “realm-management” under “Client Roles”

Add “manage-clients” to the assigned roles in this window and by clicking “add selected”.



Finally, click on the “Credentials” tab and note down the “Secret”.

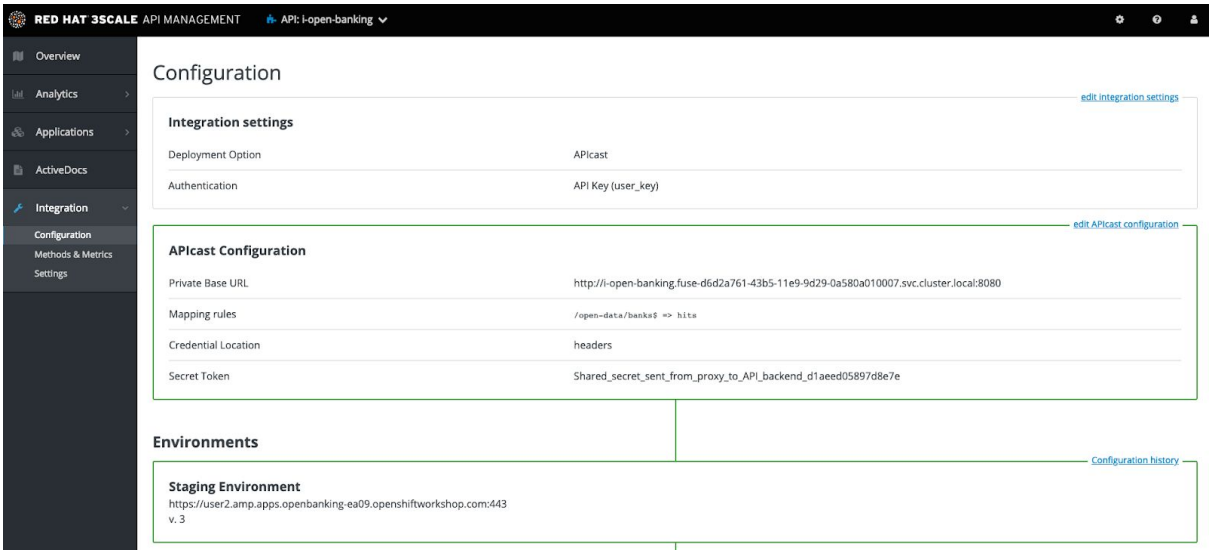


And now we are ready to use the client credentials inside 3scale OIDC configuration section.

To authenticate as we were an end user, we can just re-use our predefined user (already used to login in the rest of the components)

We have now all the elements to proceed with the corresponding configuration on API management to authenticate calls using our RH SSO.

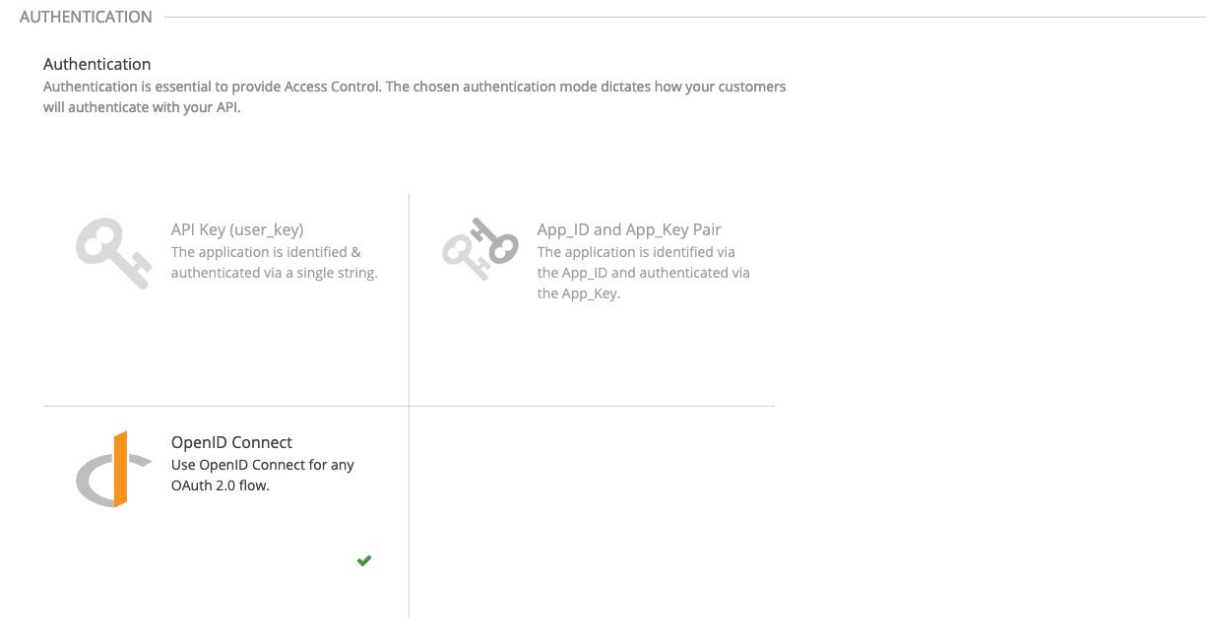
Let’s now switch back to 3scale to configure the API management side of OIDC authentication.



We can see that we have a fully configured API with API key as the Authentication method. We are going to change it to the more secure OpenID Connect, to ensure our financial data are protected from attacks performed when a key is compromised.

Click on “Edit integration settings”

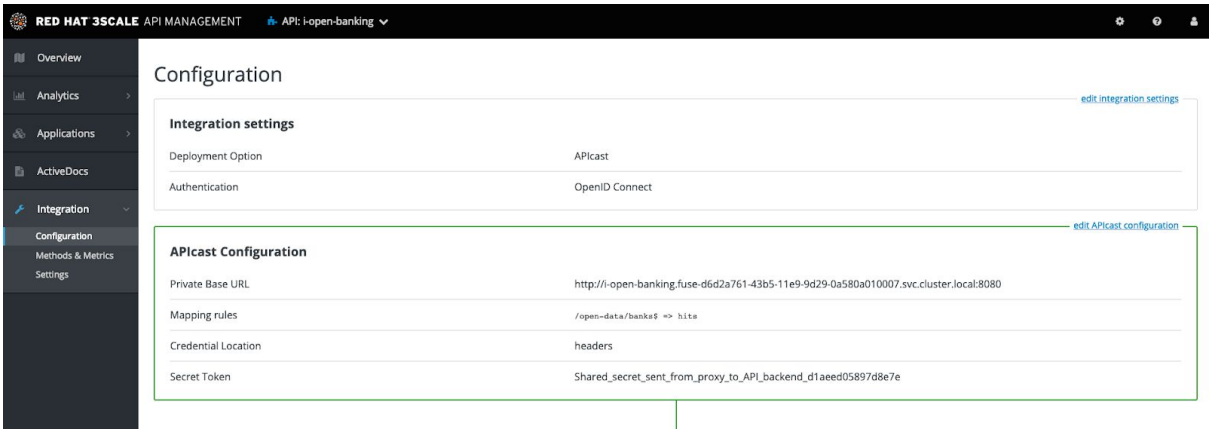
Under “Authentication” we are going to change the selected value to “OpenID Connect” and click “Update service”.



Clearly the platform is warning us that we have customers using this API and it might break their application, changing the authentication method. In a real world case, we would inform the developer in advance by using the messaging and notification functionality available within the platform.

We have now changed the authentication method, we are just left with configuring the correct IdP inside 3scale to make sure it is authenticating the requests with RH SSO.

Click on “edit APIcast configuration”.



As we see we have a dedicated field for this purpose now under “Authentication Settings” (**OpenID Connect Issuer**).

Let’s build a url of this format to use it:

```
http://client-id:client-secret@idp-public-endpoint
```

where

client-id:

sync-app

client secret:

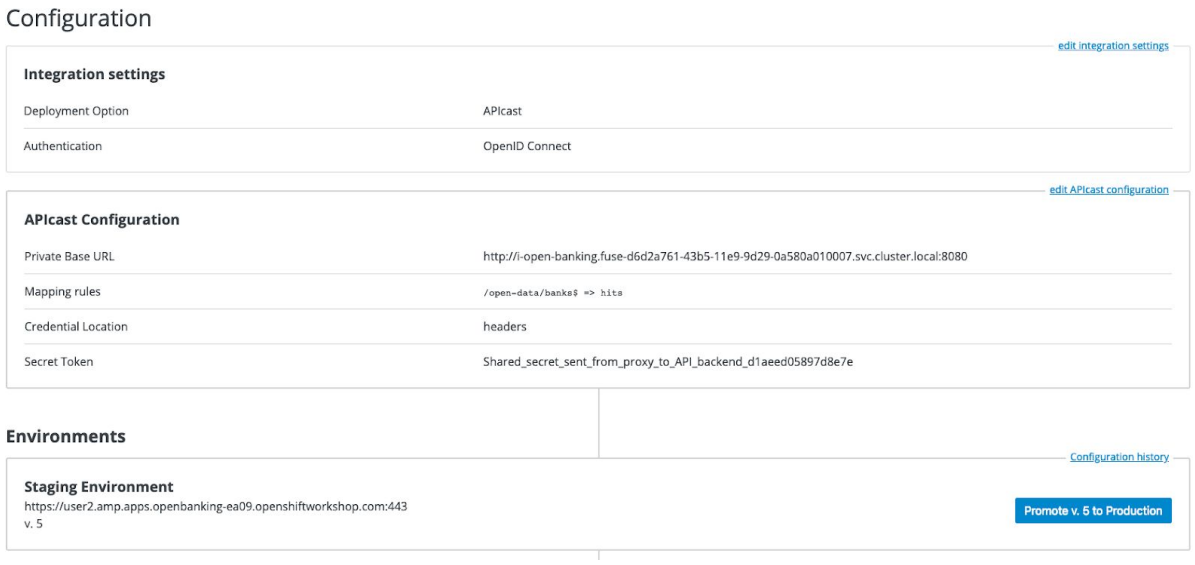
<defined-during-configuration>

idp-public-endpoint:

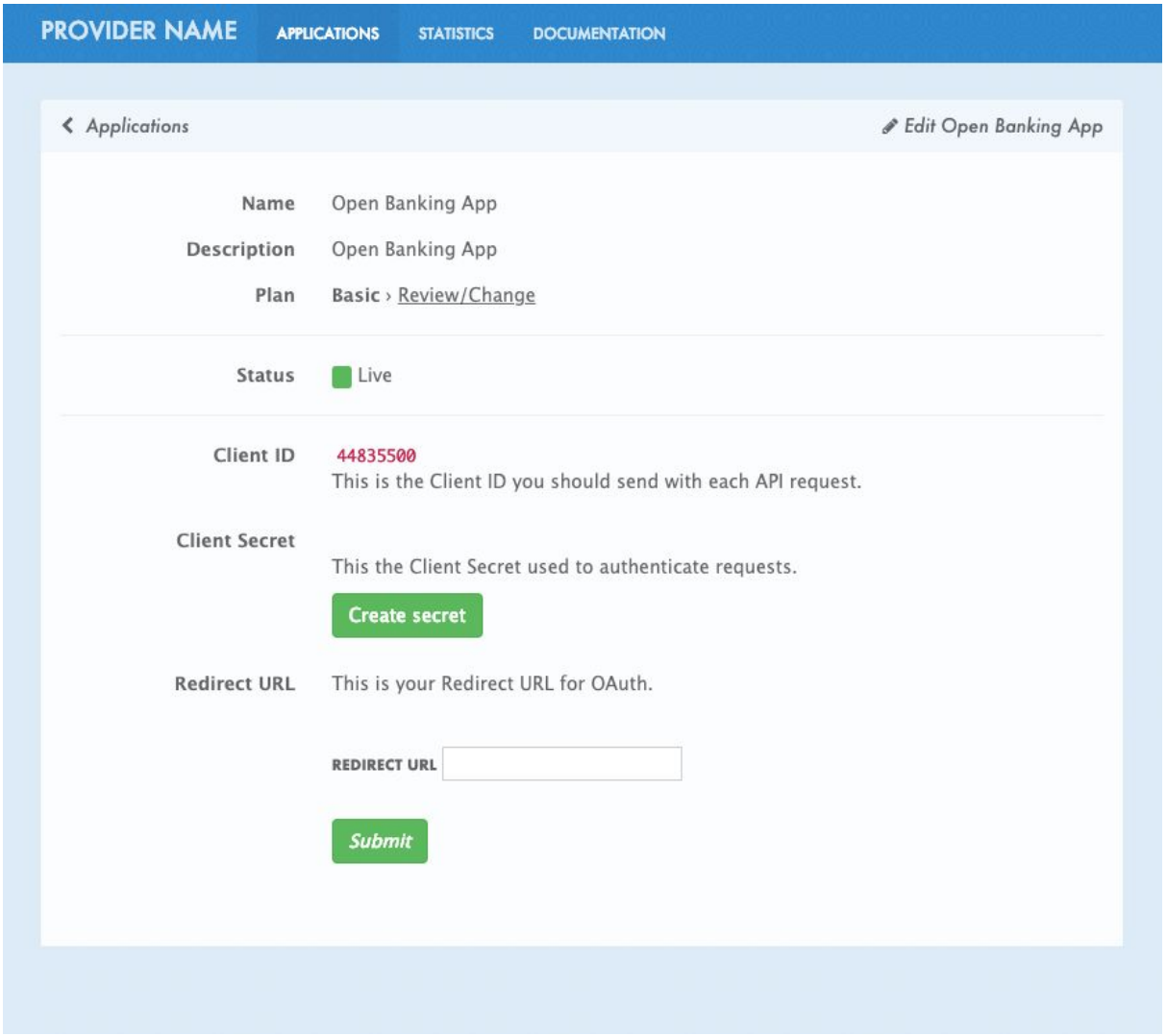
sso-http-sso.apps.GUID.openshiftworkshop.com/auth/realms/<userX>
(where you replace GUID with the GUID from the instructor and your user id)

Lastly, change the Credentials location to “As HTTP Headers”

And update the staging environment and promote the configuration to production by clicking the blue button Promote to production.

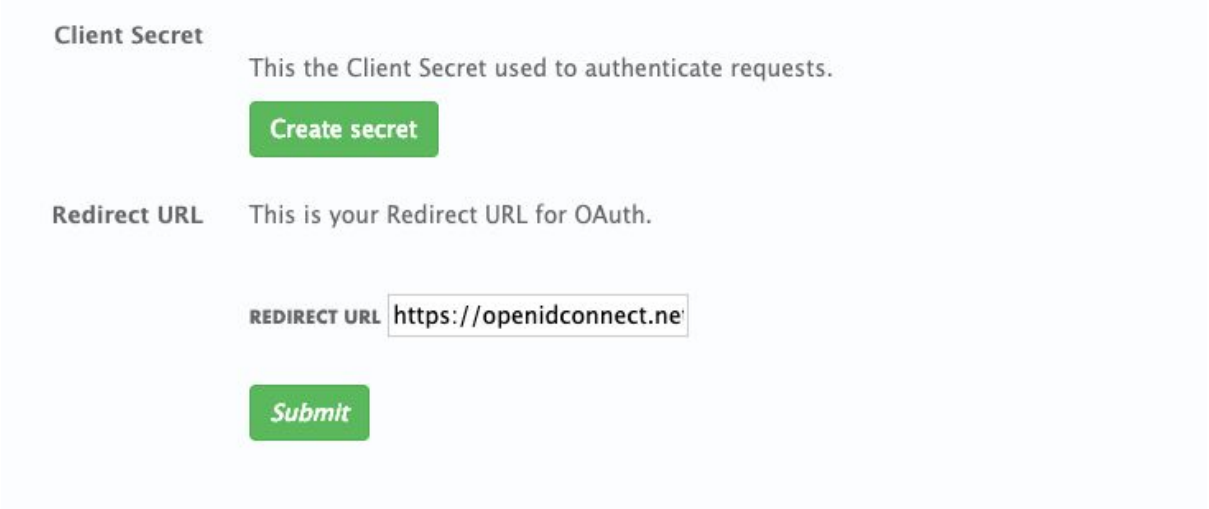


Let’s now switch user perspective and get in the shoes of the developer and open their Applications section.

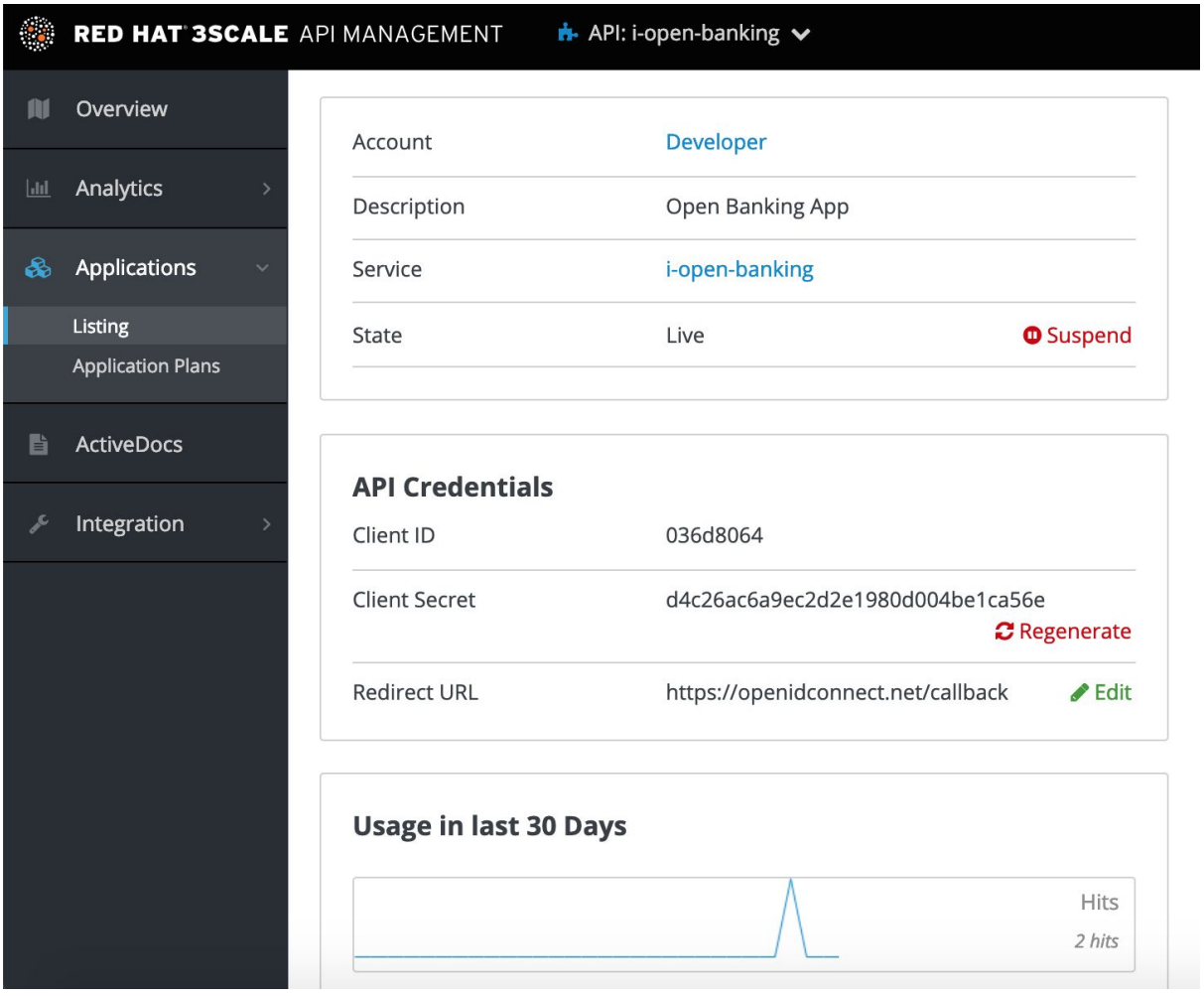


We can see the secret of their application is absent as is the redirect URL. We are going to generate the first and add as redirect URL the following URL:

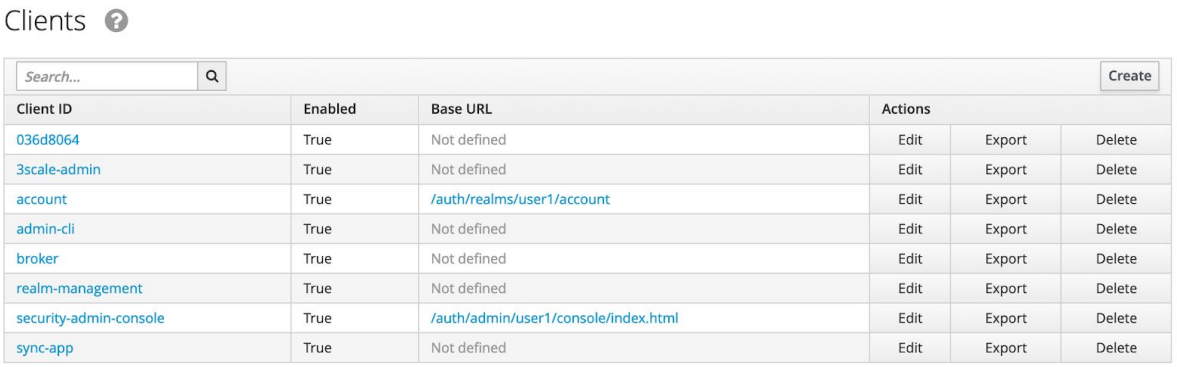
https://openidconnect.net/callback



Let’s make sure that the application is now aligned in terms of credentials both in 3scale and RH SSO.



Check the “Clients” section and ensure that the automatically created client ID is now present. In the screenshot below it is “036d8064”.



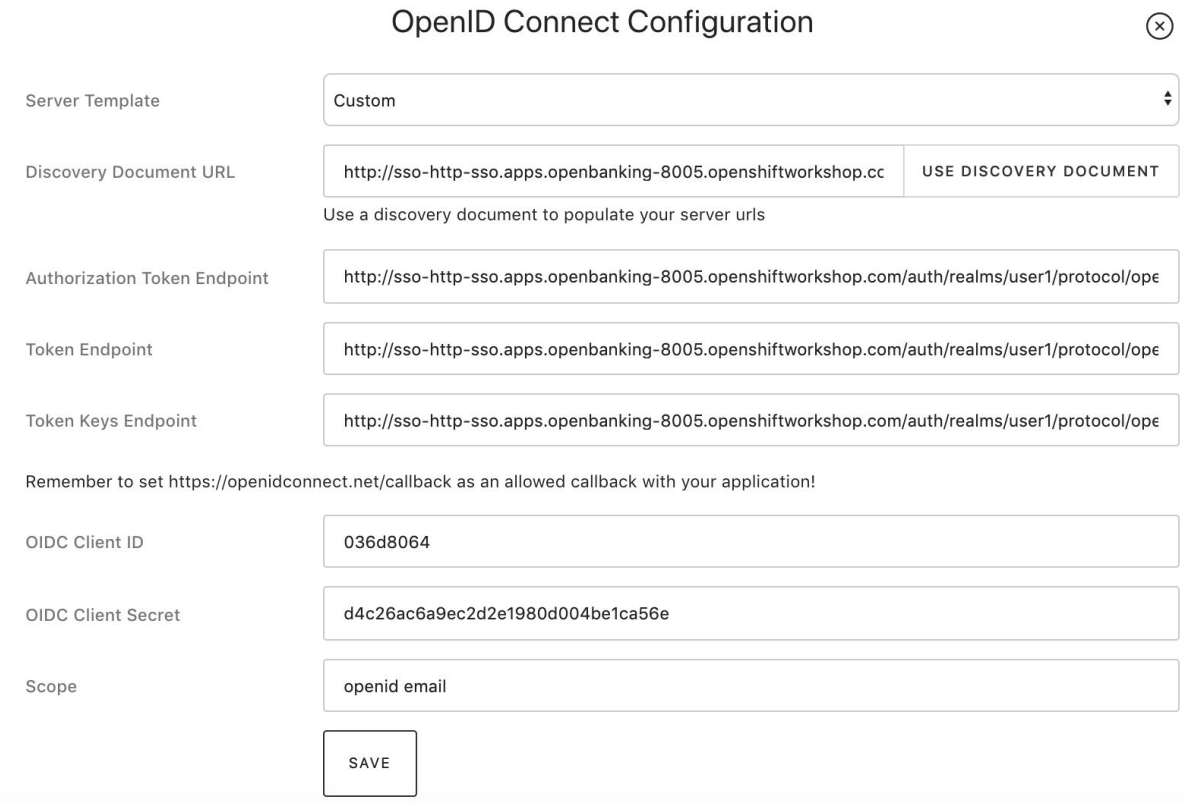
All looks good! Let’s now try to authenticate the end user, using OpenID Connect.

We are going to need a special web client, a little bit more intelligent than just the API tester:

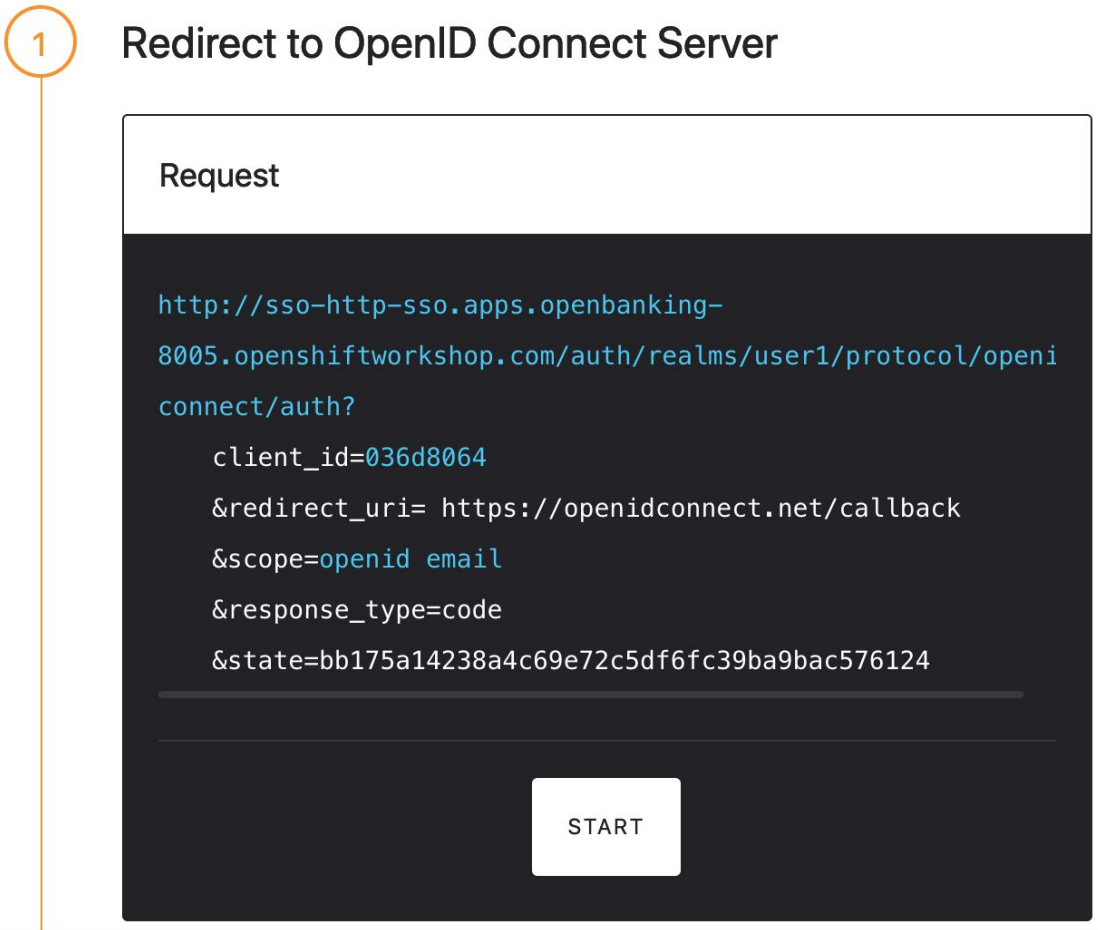
https://openidconnect.net/

Let’s configure it with the correct parameters from the previous steps by clicking on “Configuration”:

- Under “Server Template” select “Custpm”
- Under “Discovery Document URL” type the following URL: `http://sso-http-sso.apps.GUID.openshiftworkshop.com/auth/realms/userX/.well-known/openid-configuration`
- Click “Use Discovery Document”
- Fill in the “OIDC Client Id” with the client id from RHSSO
- Fill in the “OIDC Client Secret” with the secret from RHSSO
- Under Scope type in “openid email”
- Select “Save”



Now you are on the start page with the prefilled Request document:



Start the authentication flow by hitting start.

You are going to be redirected to the RH SSO login interface where you can use your user details and password we saw before (userX / openshift).

Once you login you will receive a temporary code to be exchanged for the final credentials or access token.

2 Exchange Code from Token

Your Code is


```
eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..VTYGpUa-
DVZHGZcukw_0hA.38CdI0wjX-
AvF02AhOD3WpvJCbxfwpvWw0CosuX8maHzXjiGLrH-Y-
PZltQ_fn41yhWkKjBOBr37wGi20zxjOtG351Vou4V698VyEoeFkWMnb1E6src
x4imFVrbgf48DdRh4lBzXJ_VyCsXudyDY8tUGgyDXQz9P48w9R4mOvR2AXIXD
I1heVHxTiNnTpK2z8i1dqVeG87oKSQT9iEdxkRzfuzj8fpYqGaia37.8nxaId
```


Now, we need to turn that access code into an access token, by having our server make a request to your token endpoint

Request

```
POST http://sso-http-sso.apps.openbanking-
8005.openshiftworkshop.com/auth/realms/user1/protocol/openi
connect/token
grant_type=authorization_code
&client_id=036d8064
&client_secret=d4c26ac6a9ec2d2e1980d004be1ca56e
&redirect_url=https://openidconnect.net/callback
&code=eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..VTYG
DVZHGZcukw_0hA.38CdI0wjX-
AvF02AhOD3WpvJCbxfwpvWw0CosuX8maHzXjiGLrH-Y-
PZltQ_fn41yhWkKjBOBr37wGi20zxjOtG351Vou4V698VyEoeFkWMnb1E6s
x4imFVrbgf48DdRh4lBzXJ_VyCsXudyDY8tUGgyDXQz9P48w9R4mOvR2AXI
I1heVHxTiNnTpK2z8i1dqVeG87oKSQT9iEdxkRzfuzj8fpYqGaia37.8nxa
```

EXCHANGE

Hit “Exchange”

Request

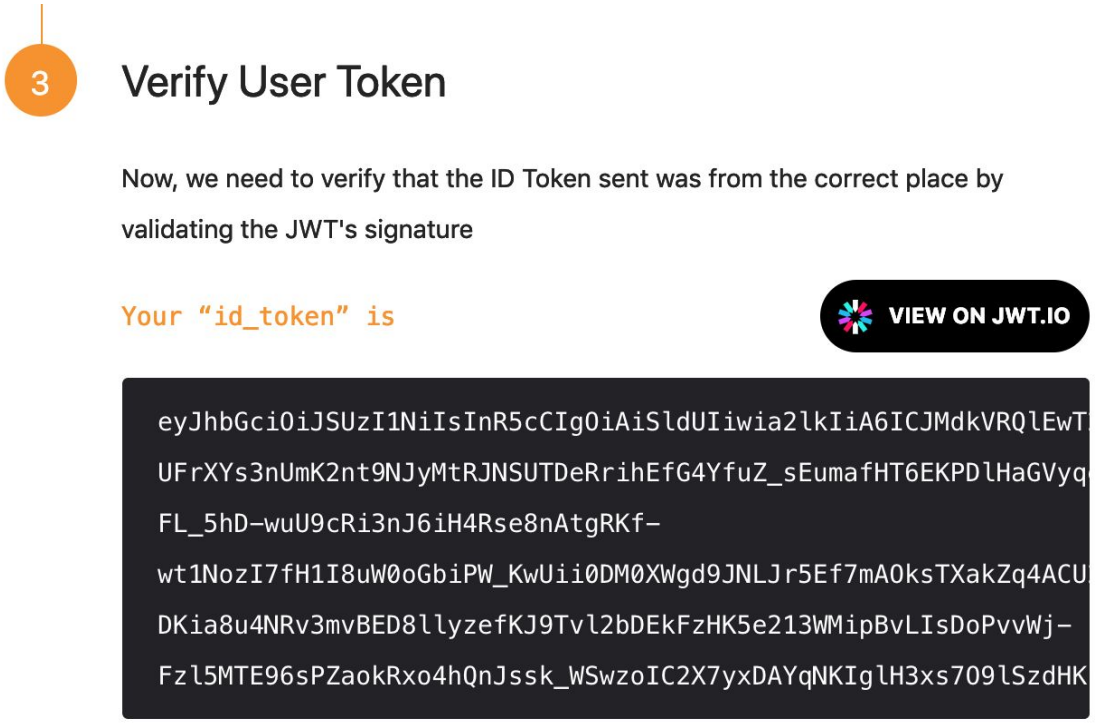
```
POST http://sso-http-sso.apps.openbanking-
8005.openshiftworkshop.com/auth/realms/user1/protocol/openi
connect/token
grant_type=authorization_code
&client_id=036d8064
&client_secret=d4c26ac6a9ec2d2e1980d004be1ca56e
&redirect_url=https://openidconnect.net/callback
&code=eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..VTYG
DVZHGZcukw_0hA.38CdI0wjX-
AvF02AhOD3WpvJCbxfwpvWw0CosuX8maHzXjiGLrH-Y-
PZltQ_fn41yhWkKjBOBr37wGi20zxjOtG351Vou4V698VyEoeFkWMnb1E6s
x4imFVrbgf48DdRh4lBzXJ_VyCsXudyDY8tUGgyDXQz9P48w9R4mOvR2AXI
I1heVHxTiNnTpK2z8i1dqVeG87oKSQT9iEdxkRzfuzj8fpYqGaia37.8nxa


HTTP/1.1 200
Content-Type: application/json
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwi
  "expires_in": 300,
  "refresh_expires_in": 1800,
  "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiw
  "token_type": "bearer",
  "id_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lk
  "not-before-policy": 0,
  "session_state": "45c16117-8802-4ca6-aa00-b535d9a26273",
  "scope": ""
}
```

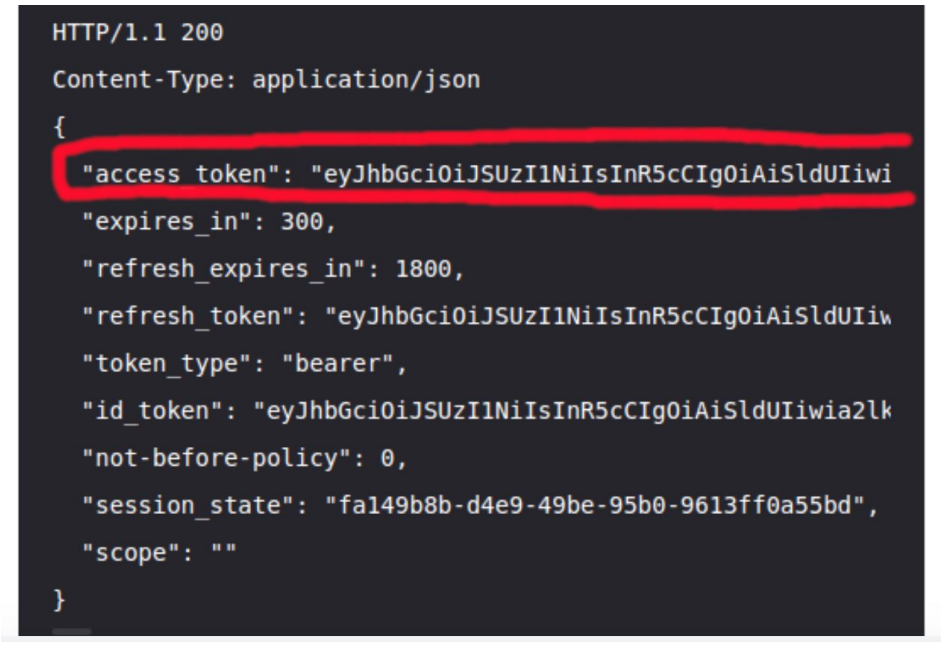
NEXT

You will receive the “access_token” which is an expiring credential that we will be using to authenticate with 3scale to get access to the configured API using OpenID Connect. We can see that another important piece of information is shown there regarding when this credential will expire “expires_in”.

We can hit NEXT and id_token will also be shown, which contains more user related details.



Copy the “access_token” value in the step 2 (the long string).



It should look something like this:

eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRa1RJX2VwS2IwNVpFSkp3ZTd1cnFQUWtjSERNRi1SMnhGcE1tZUJ2aC1Vln0.eyJqdGkiOilyYzJmZjQ5ZS01MDY4LTQ0MjQtYTRiNS05MWU3OTk3MTM0YTMiLCJleHAiOjE1NDczOTc1NTIsIm5iZiI6MCwiaWF0IjoxNTQ3Mzk2NjUyLCJpc3MiOiJodHRwczovL3NlY3VyZS1zc28tc3NvLmFwcHMub3Blbi1iYW5raW5nLm9wZW50cnkubWUvYXV0aC9yZWFsbXMvb3BlbnNoaWZ0IiwiYXVkIjoiNWJjOTRmNmEiLCJzdWIiOilyMzZjZDRhMy01MGM2LTQ3YjQtYWUzZC05ODdiYjA1ZTk4MzYiLCJ0eXAiOiJCZWFyZXIiLCJhenAiOiI1YmM5NGY2YSIsImF1dGhfdGltZSI6MTU0NzM5NTg3Niwic2Vzc2lvbl9zdGF0ZSI6ImZhMTQ5YjhiLWQ0ZTktNDliZS05NWIwLTk2MTNmZjBhNTViZCIsImFjciI6IjAiLCJhbGxvd2VkLW9yaWdpbnMiOltdLCJyZWFsbV9hY2Nlc3MiOnsicm9sZXMiOlsidW1hX2F1dGhvcmI6YXRpb24iXX0sInJlc291cmNlX2FjY2VzcyI6eyJhY2NvdW50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2NvdW50IiwibWFuYWdlLWFjY291bnQtbGlua3MiLCJ2aWV3LXByb2ZpbGUiXX19LCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJldmFsczk4QGV4YW1wbGUuY29tIiwiZW1haWwiOiJldmFsczk4QGV4YW1wbGUuY29tIn0.O7y6GDFq5CajAT0DkywEuQqEuD5H7_YMqrVC4AMPthZ-m_xZ_DAPBEqj3mmzp1o1JOo0_4pMxNgKpyyqCQiFY79GRS5IJE6aVrZK53rQkud5dlaZAE1-ryiD8CtP_MrQtsTS7bVKbaFyCXNyFfxy3c-TER8GnGG90OlYPXpy5M954slcp4CWXxA7ZwVEuQNRRs5w2G2TCjrFyQjCzsINFwDRtADjbMiY7kq1cwRB5qM9ipdEEligDnH8dietiOZgY24sK10vtowjz_CHuWr5W3474dAZVFC7utwSt_bNcoj1gENRcz5cP7fH7Nim8e4itWoSVPRVYcfDHyYb9zixQ

We are going to use this as a Header in our call towards the OpenID protected service.

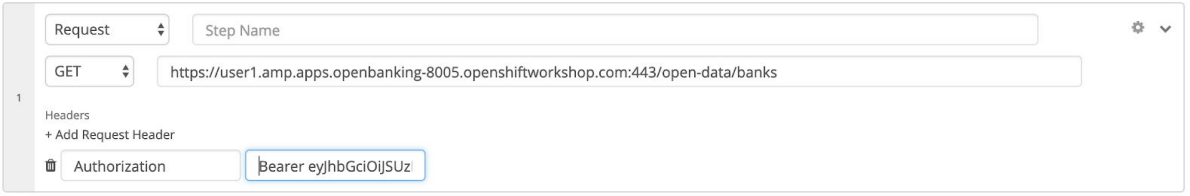
Let's go back to our API Tester (https://apitester.com) and add this as an Authorization header.

Use the URL for your API gateway:

`https://userX.amp.apps.GUID.openshiftworkshop.com:443/open-data/banks`

where you replace **userX** with your user ID and the **GUID** with the provided value from your instructor.

Under Header add the following values:
Name: **Authorization**
Value: **Bearer** < enter the access_token >



Let's hit **Test**

You should see a 200 response and the values in the "Response Body".

The work done by the API management behind the curtain is quite impressive:

- Check for the validity of the access token credentials (not expired, legit and associated to the correct application)
- Check for rate limits on the application triggering the call
- Apply monetization rules to the call
- Apply any additional policy that might modify the call in real time
- Report the traffic back to the analytics component