Preparing to deliver the workshop

Instructor requirements:

- Being very comfortable with 3scale both in terms of configuration and of operations. Knowing the basics of the Developer Portal
- Being comfortable with RH SSO and especially around OpenID Connect protocol
- Being able to use Postman (or similar REST client) to build REST request and REST authentication
- Having a basic knowledge of OpenShift

Attendees requirements:

- To use the GUI of OpenShift: https://docs.openshift.com/container-platform/3.11/architecture/infrastructure_components/web_console.html#browser-requirements
- Basic knowledge of REST protocol
- Basic knowledge of containers
- Basic understanding of OAuth social applications

Environment:
Environment reachable here:
https://tutorial-web-app-webapp.apps.open-banking-2a38.openshiftworkshop.com

End Users credentials:
user[1..100] / openshift

3scale dedicated instances here [X is your user id]:
https://userX-admin.apps.open-banking-2a38.openshiftworkshop.com
access using username and password above

dedicated gateway
https://userX.amp.apps.open-banking-2a38.openshiftworkshop.com

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.

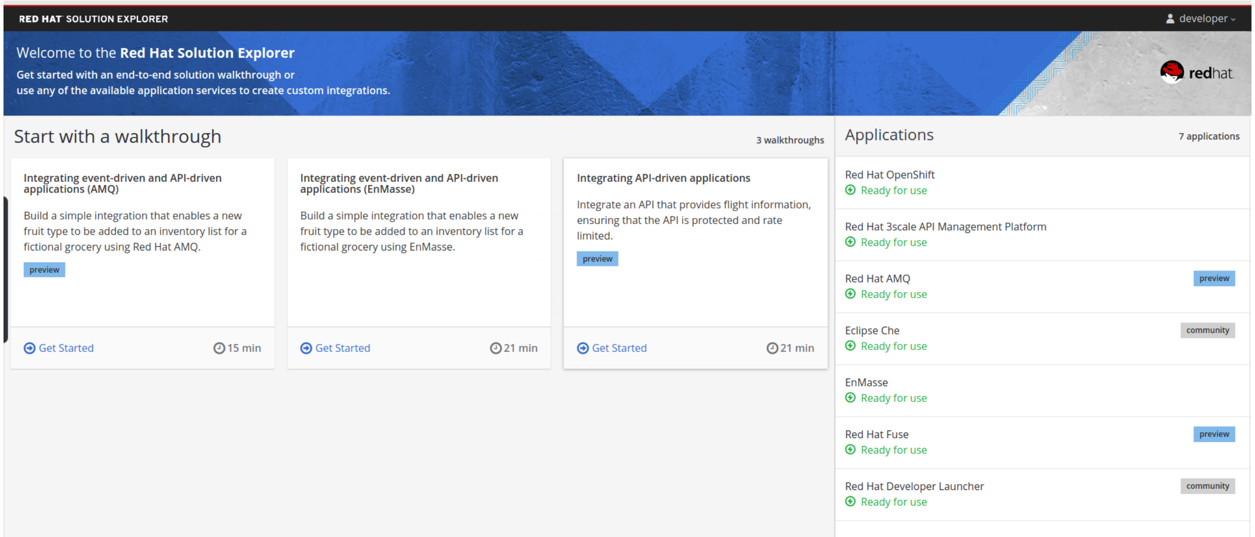
Timeline

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 Part 1

Integr8ly

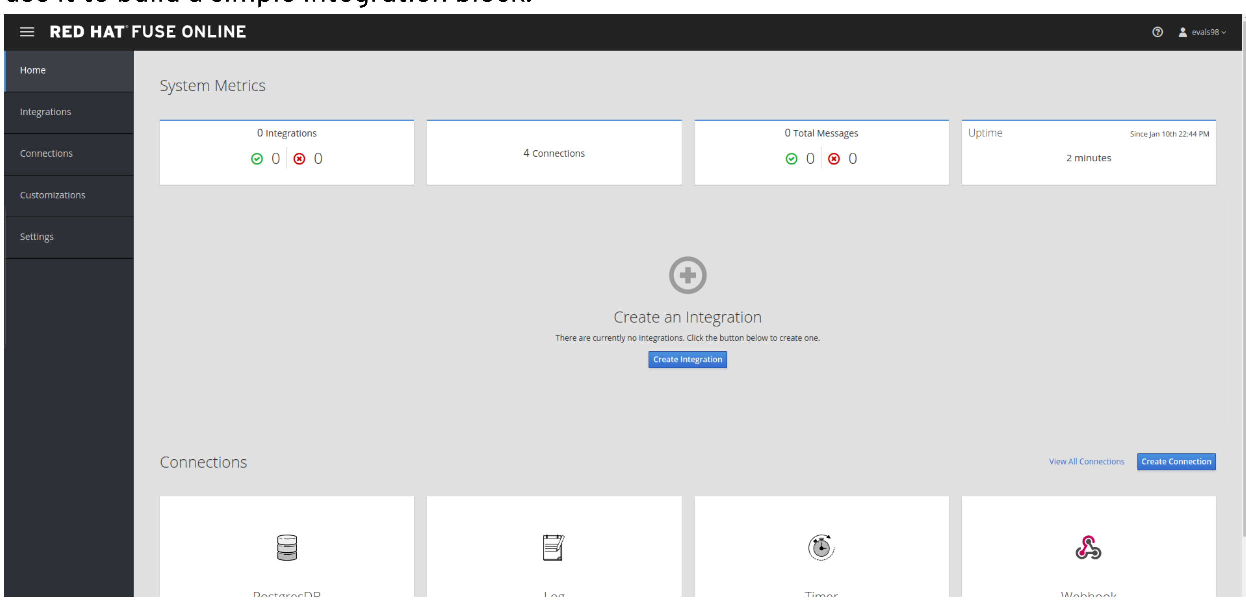
Login into the integr8ly environment main page
https://tutorial-web-app-webapp.apps.open-banking-2a38.openshiftworkshop.com



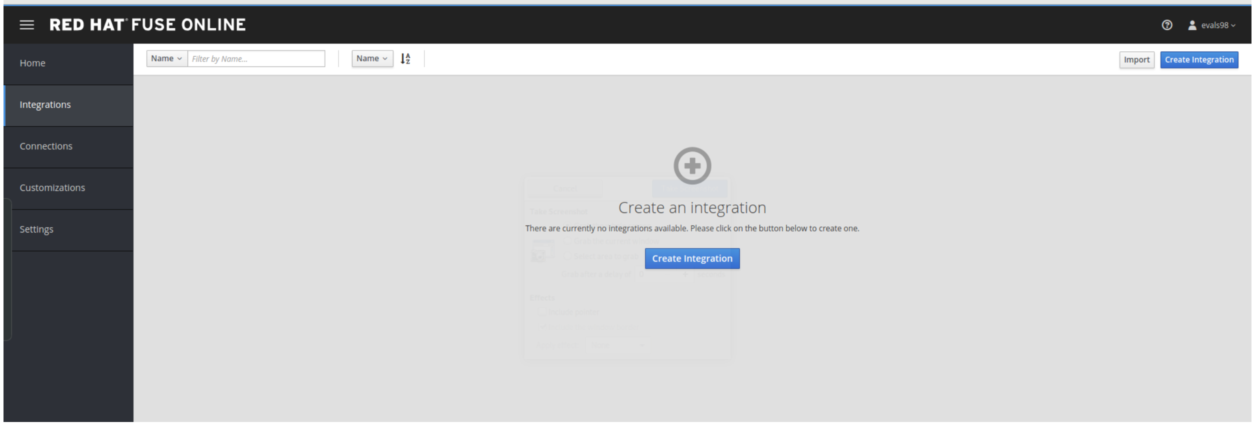
We will see how integr8ly environment works and what you get. It provides out of the box integration between products but it doesn't change the base components that come with it. You can install yourself this type of environment starting from a clean or empty OpenShift installation.

Fuse Online

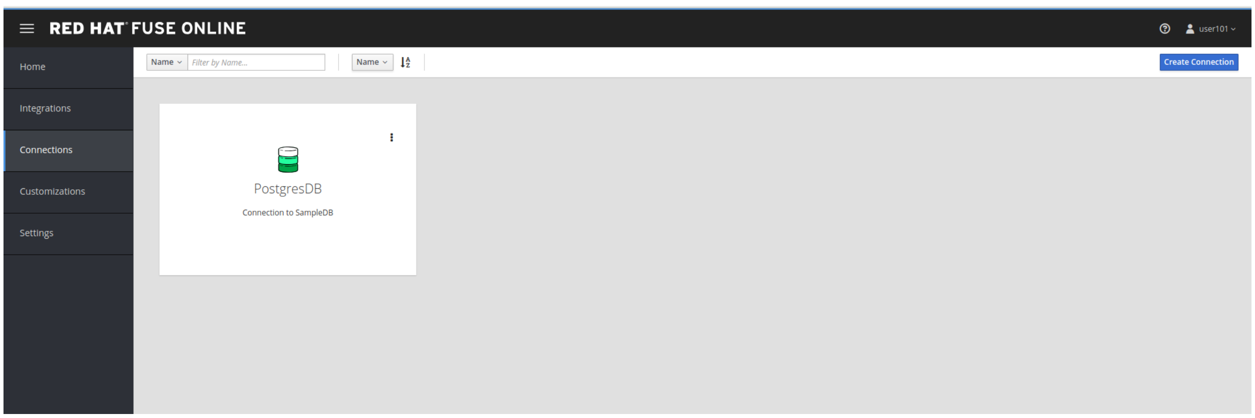
Let's start our first lab session, by opening Red Hat Fuse Online. You will get there by clicking **Red Hat Fuse** link in the tutorial dashboard. We will see the main functionalities of it and then use it to build a simple integration block.



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.

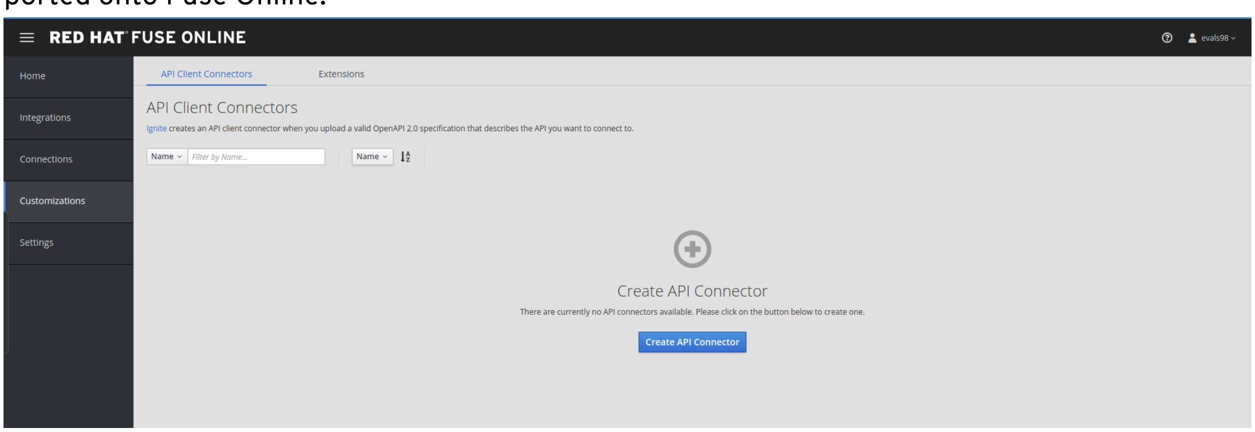


The integration window (**Integrations**) allows you to have a graphical representation of the mediation/transformation and is the first tool to onboard the citizen developers. Behind the curtains there is Camel (which is a proven technology and deployed in highly transactional environment as well) and 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 .

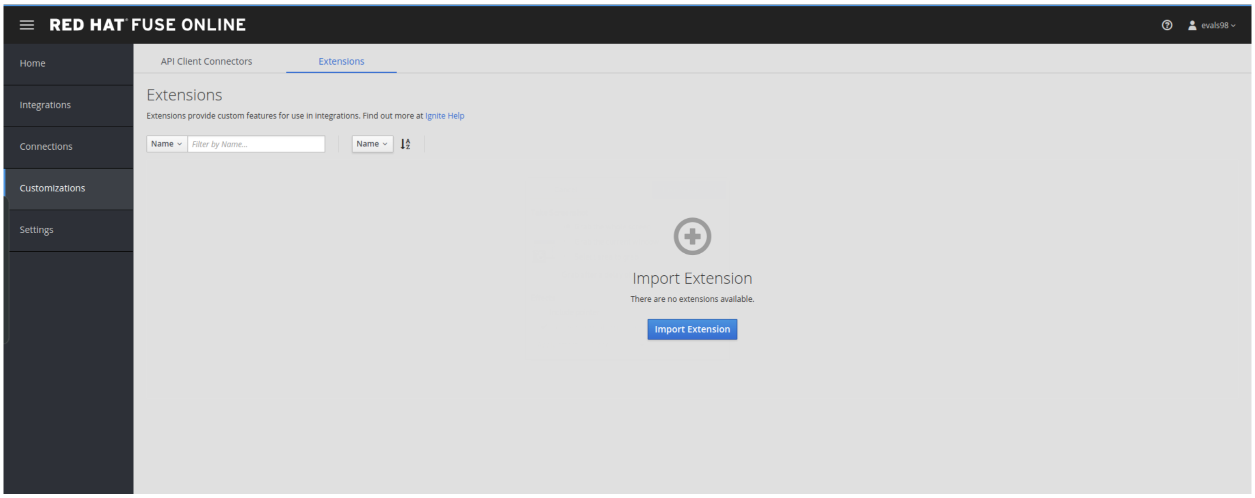


(**Connections**) you can start or end (or sometimes interpose) connections which are fully configured connectors. We will be showing the configuration of one connector during today's lab.

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.

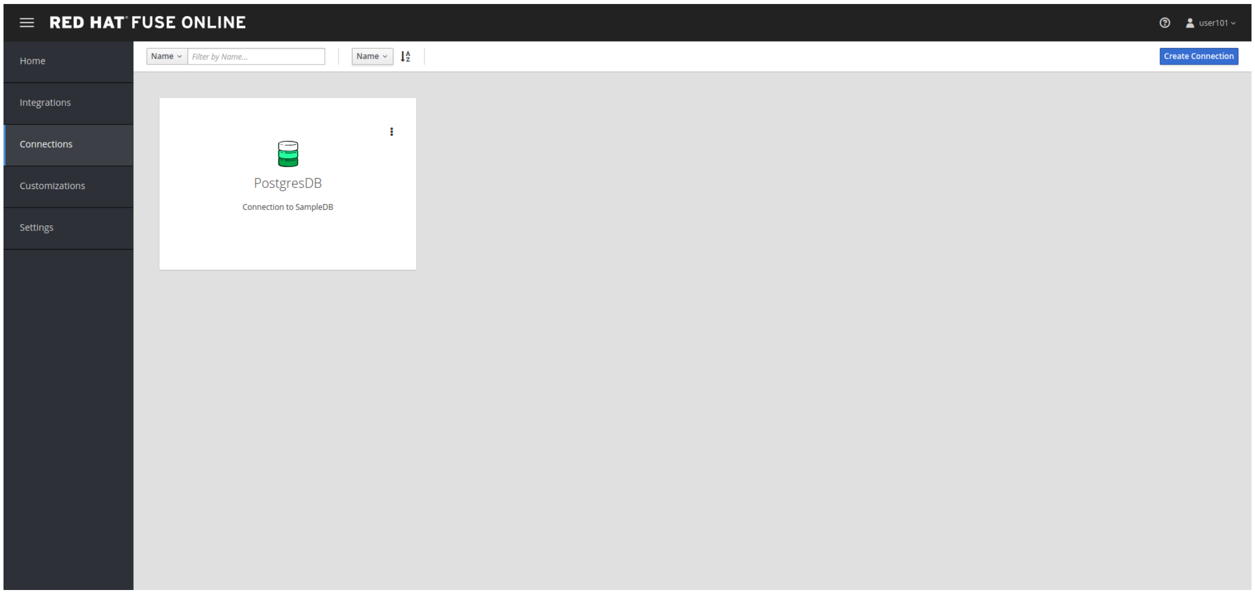


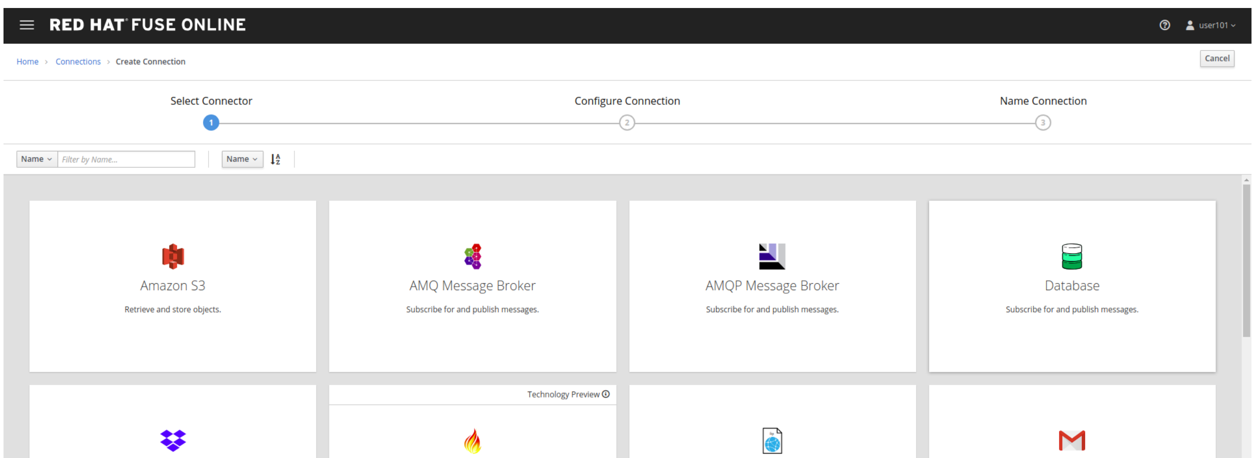
(**Extensions** panel) this is the part where you can extend the functionalities of the platform. You can compile a custom or community available extension to perform a specific functionality or connect to a specific system and in this easy way add transformation functionalities to the product.

LAB BEGINS
Let's start INTEGRATING now!

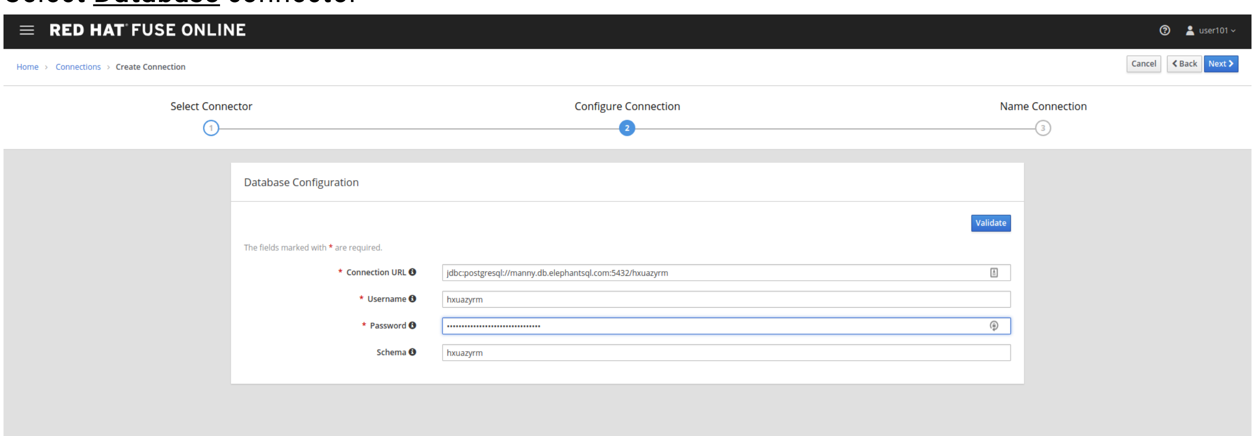
This first session will take us to the creation of a simple REST service from a database source. This is quite a common scenario and one that goes well for quick prototyping service scenarios. Let's start by creating the connection to an existing DB that I previously created. It is a DBaaS built on PostgreSQL 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.

Connections -> Create connection





Select **Database** connector

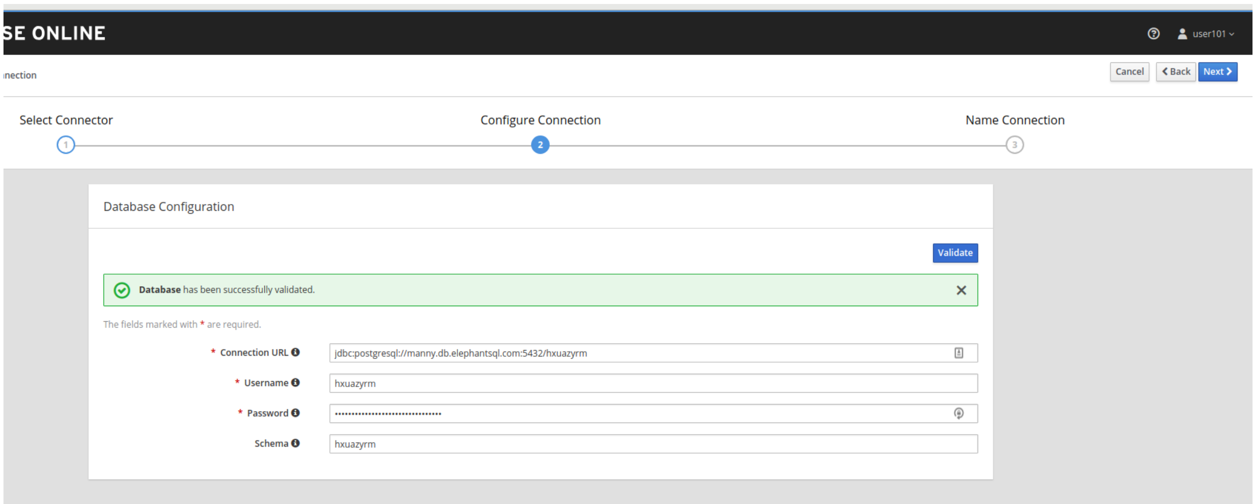


Use the following connection details and validate them:

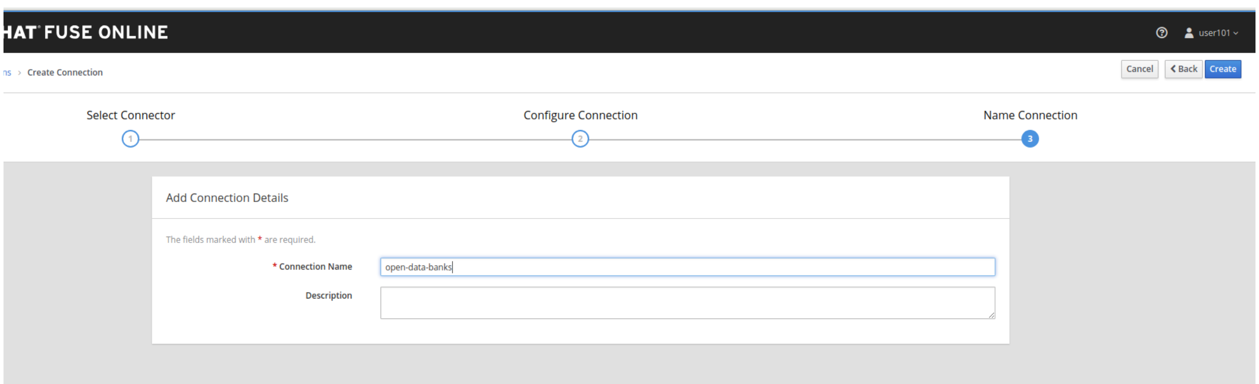
Connection details: jdbc:postgresql://postgresql.shared.svc:5432/sampledb

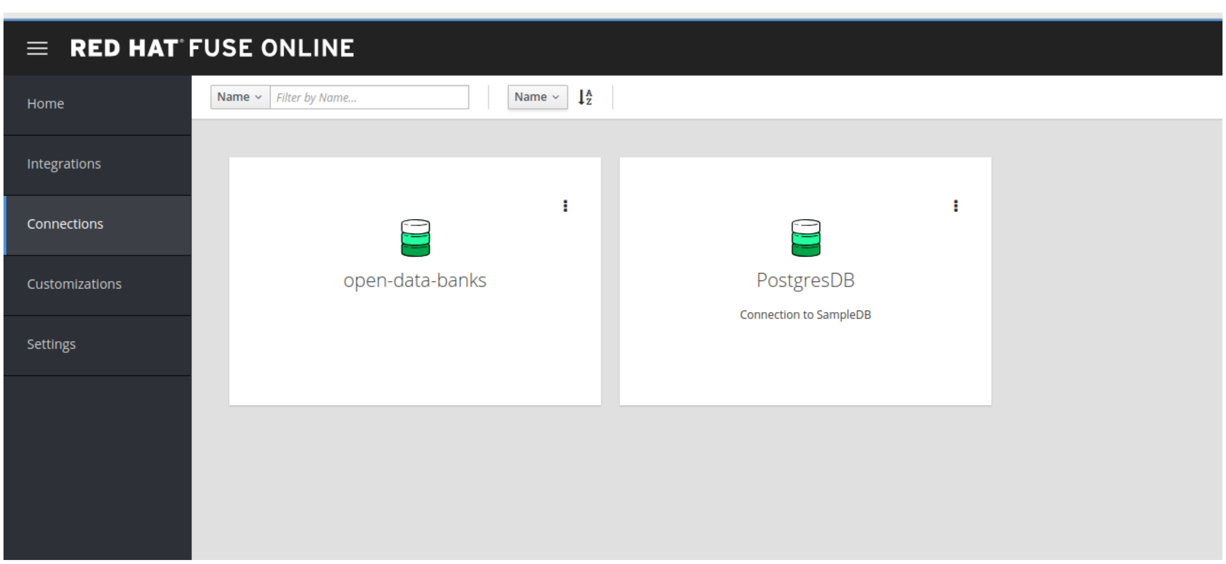
username: dbuser

password: password



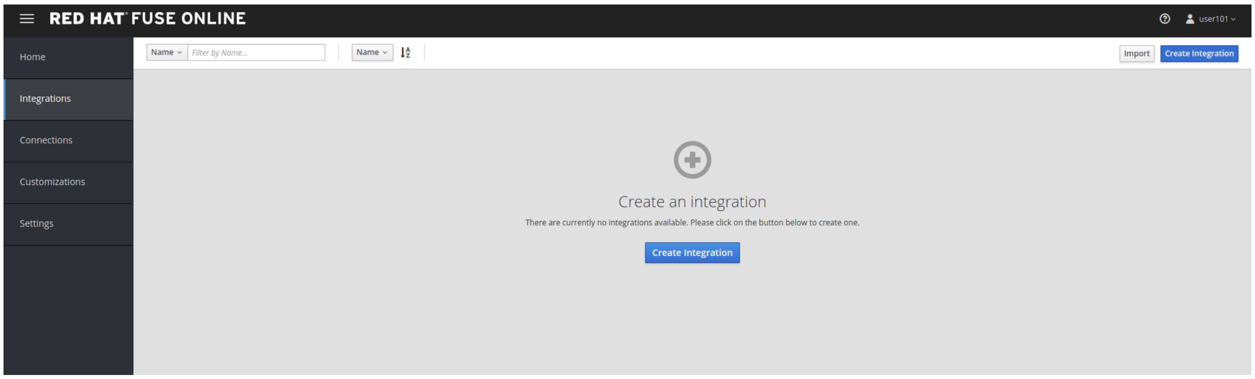
Let's name the connection and finalize the configuration of the connector (**Create**).





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

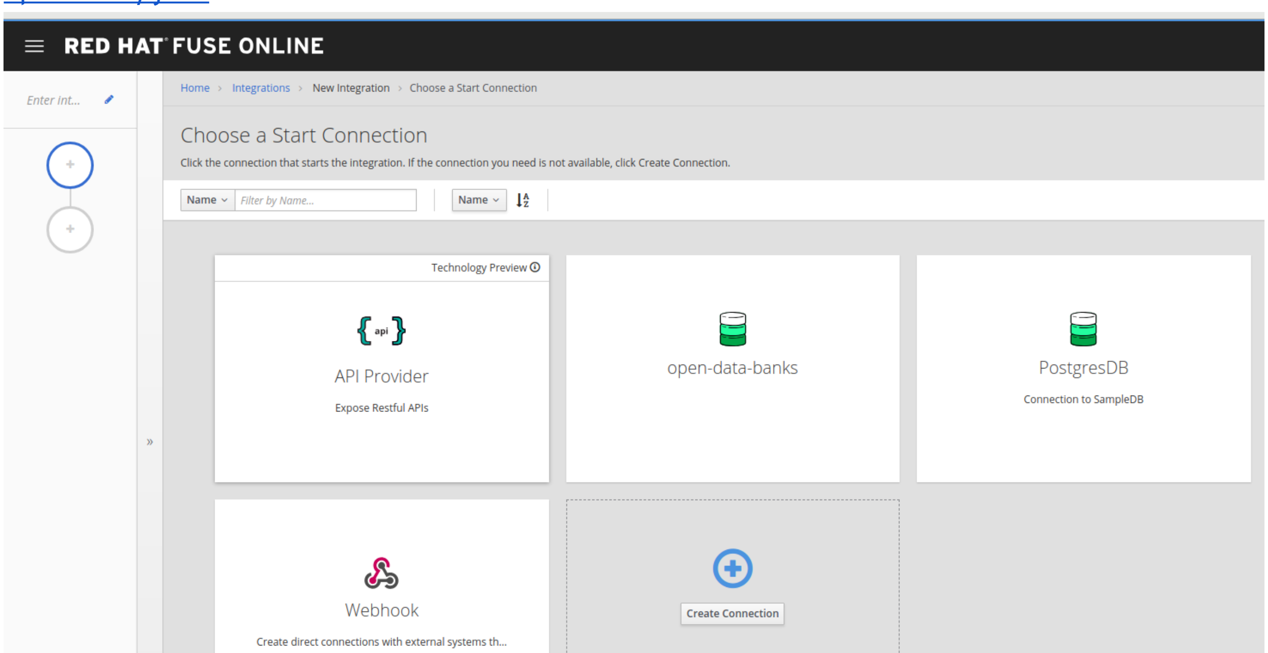
Now that we have configured the end of the integration path we want to build and we will see where to find the start and we will put the two pieces together. **Integrations -> create integration**

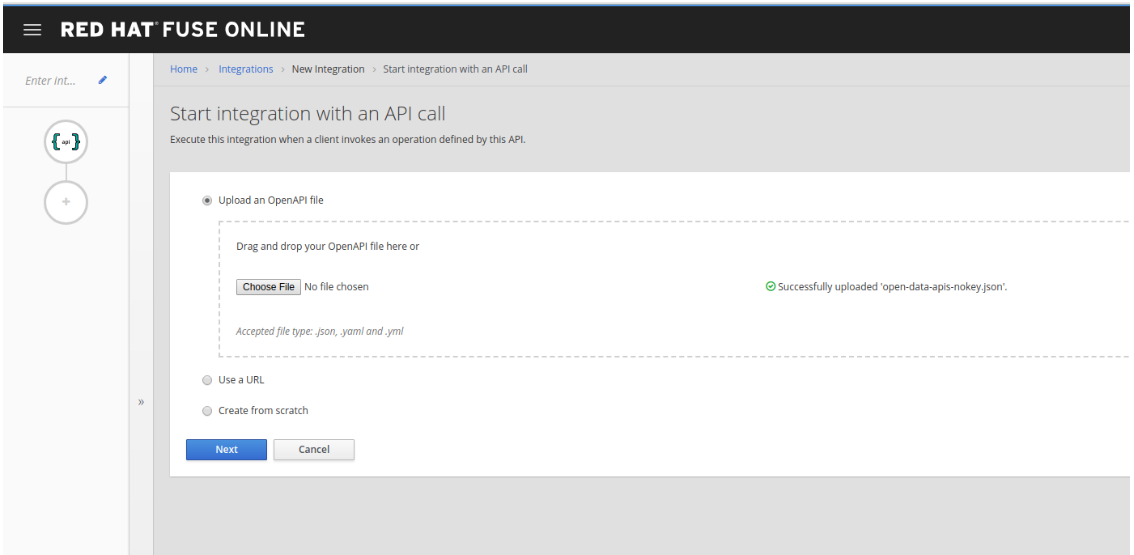


We will start by exposing a REST endpoint that will then get mapped to the backend datasource.

Use the **API Provider** connector with the following OpenAPI file:

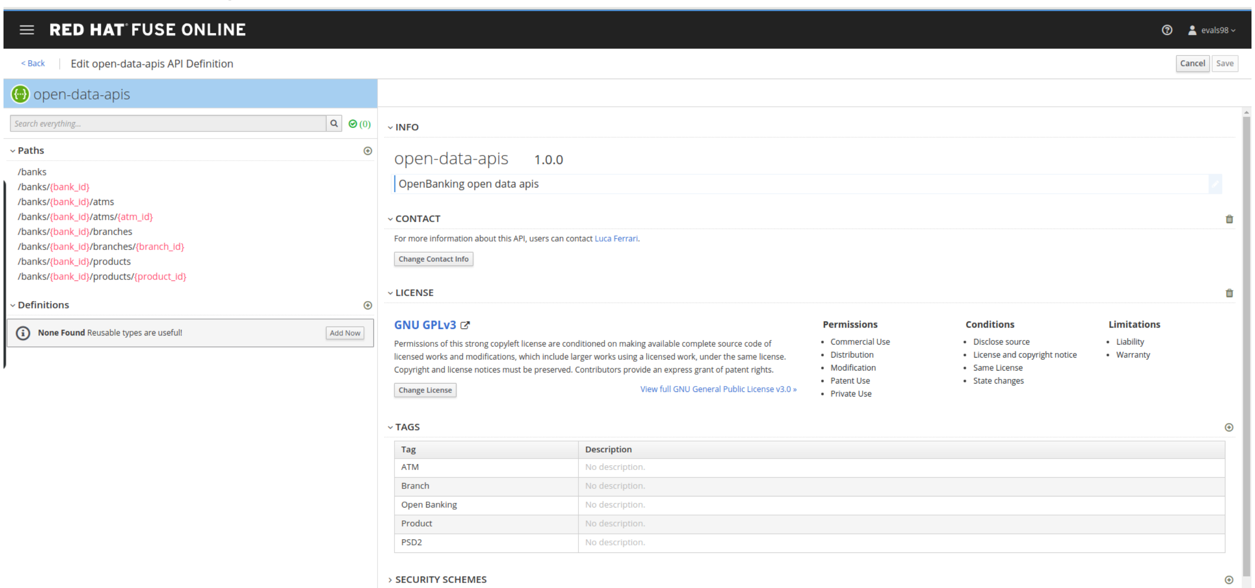
https://raw.githubusercontent.com/lucamaf/open-banking-roadshow/master/open-data-apis-nokey.json





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.

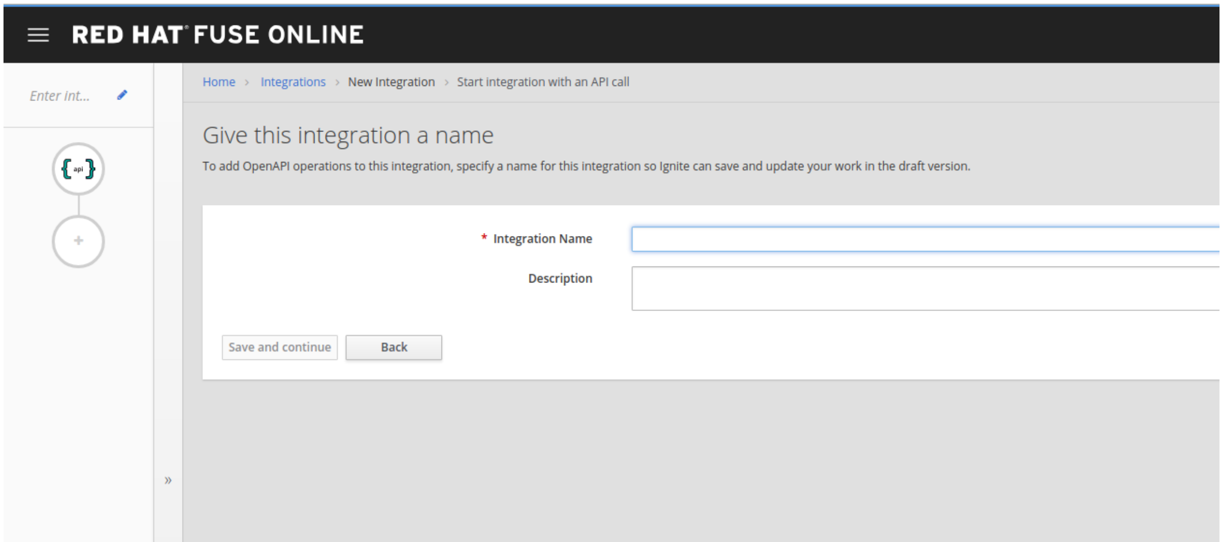
To show how easy it is to correct definitions **-> review/edit**



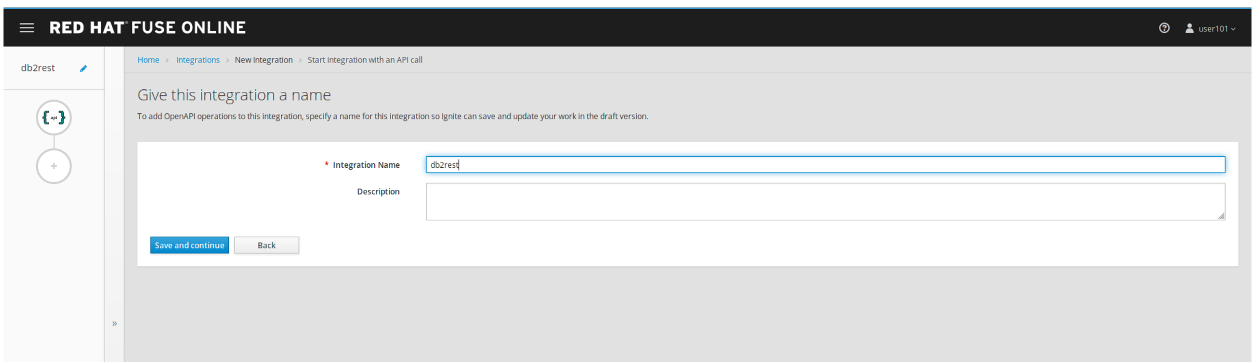
This opens a window on Apicurito which is a scaled down version of Apicurio, our API Design platform. It is fairly easy to change elements graphically and also with the help of this tool an API team can start with a Design First approach when configuring the API. Also the same team doesn't need to know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

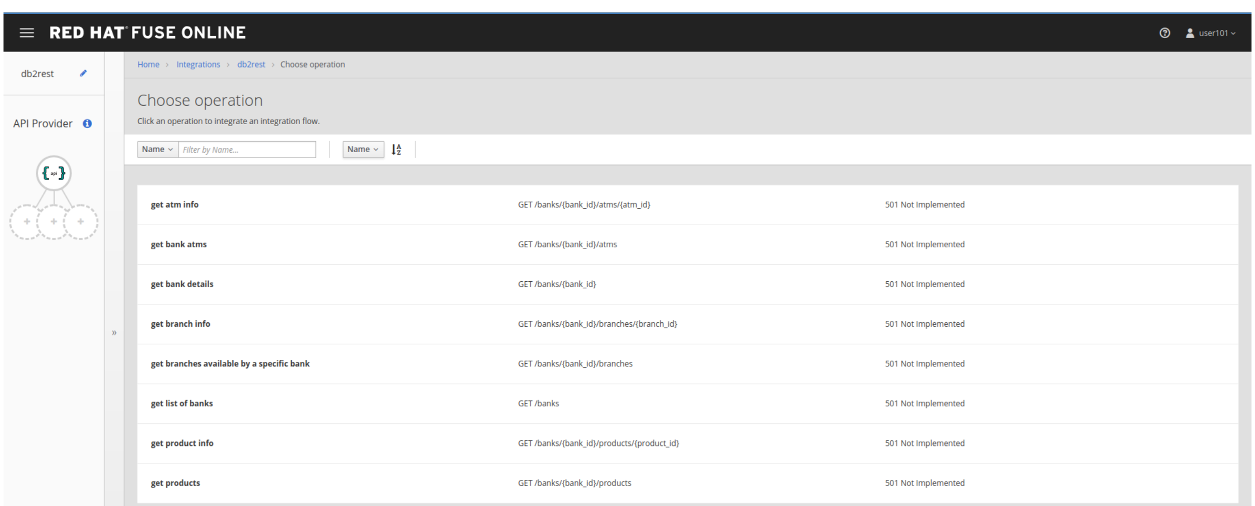
Name the integration.



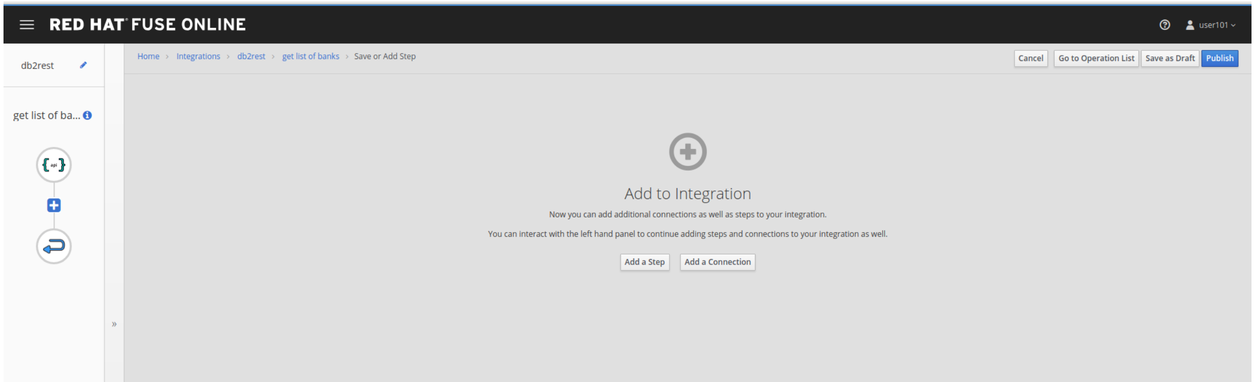
And **save and continue**



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

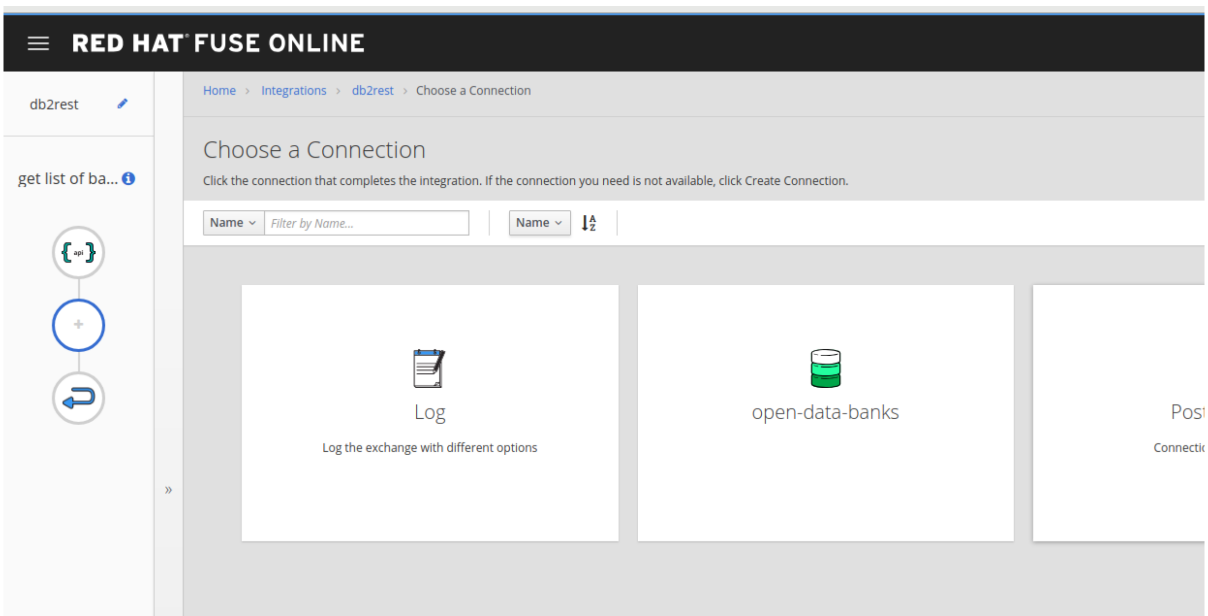


Click on **get list of banks**

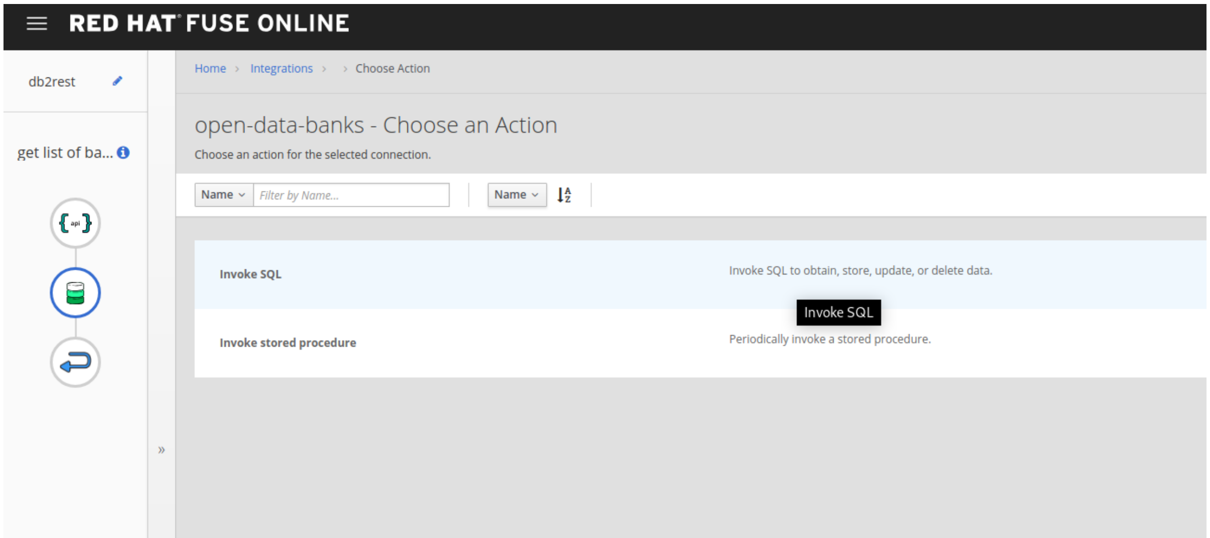


We now have a dumb pipe which connects an endpoint to receive user requests and return always 200 (all OK) - the return blue arrow symbol. Not very useful integration but a starter! Let's connect this front end to the database we previously configured as a terminating connection.

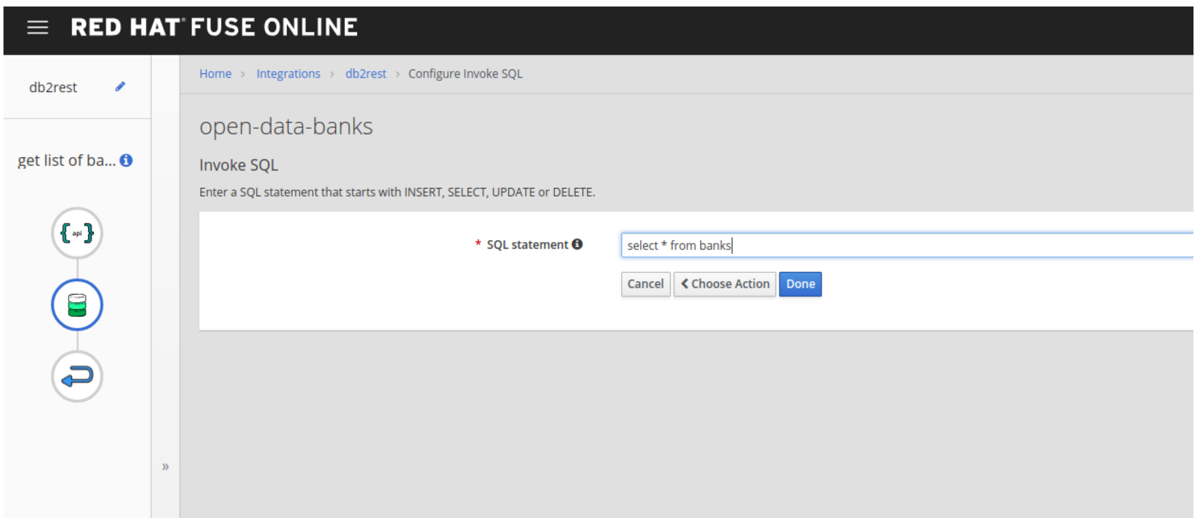
Add a connection



Click on the previously configured data source.

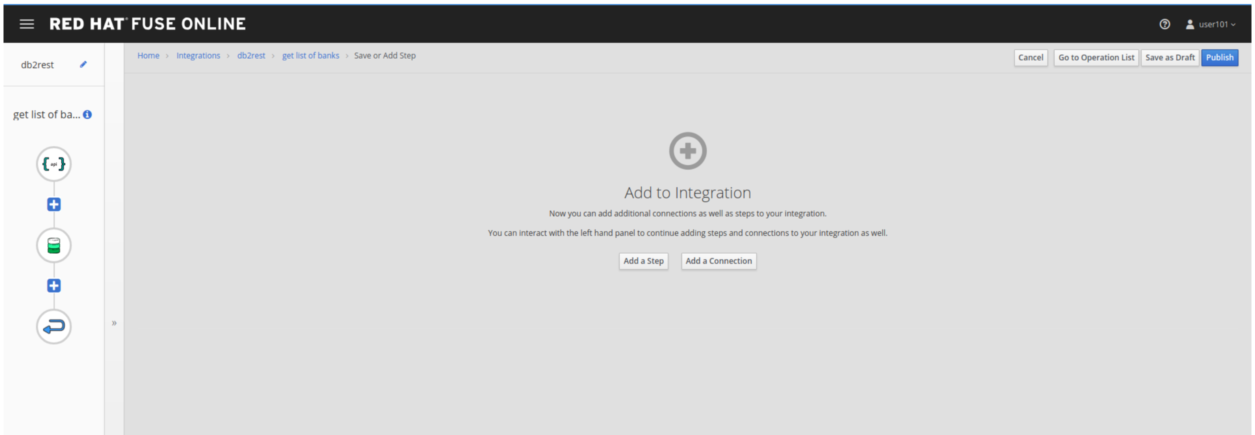


And we will invoke a simple SQL statement to return the data from a single table.

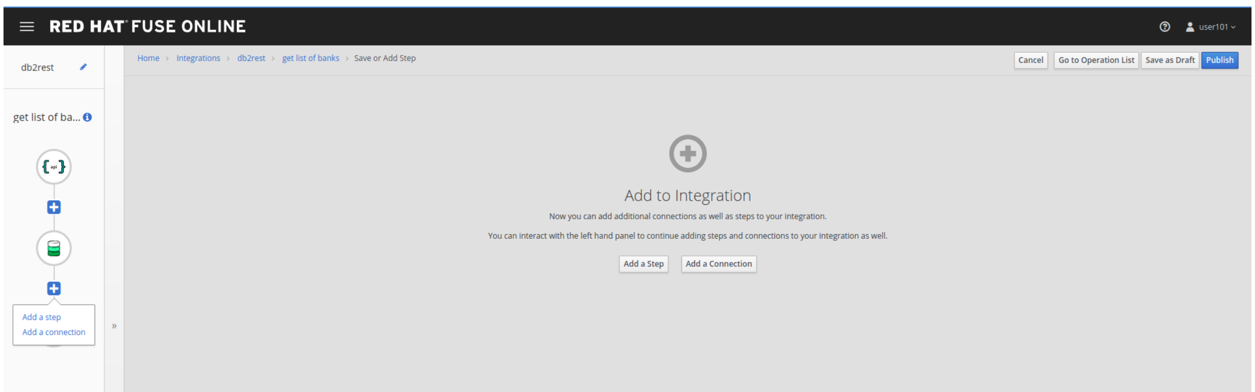


The simple statement to introduce is:
*select * from banks*

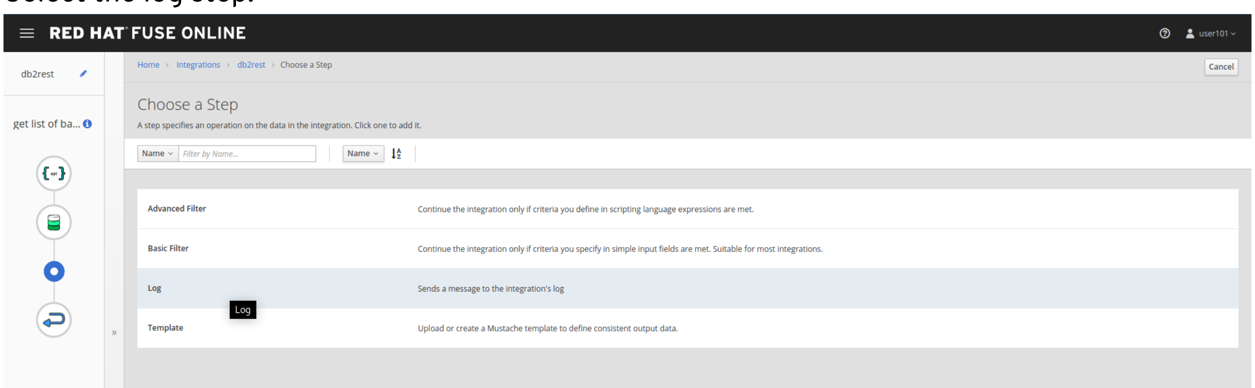
When you click **done** the statement will validate and you should be able to proceed with the configuration of the integration.



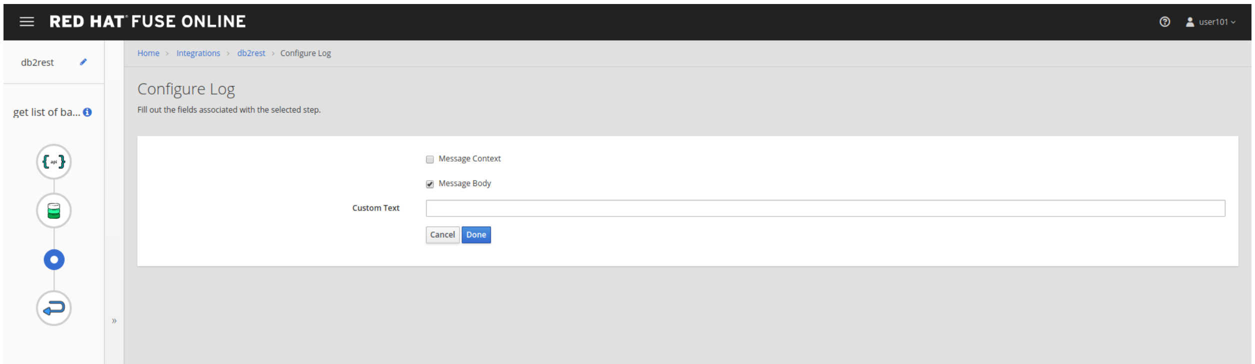
And now let's add a simple log of the requests coming through. Add a step.



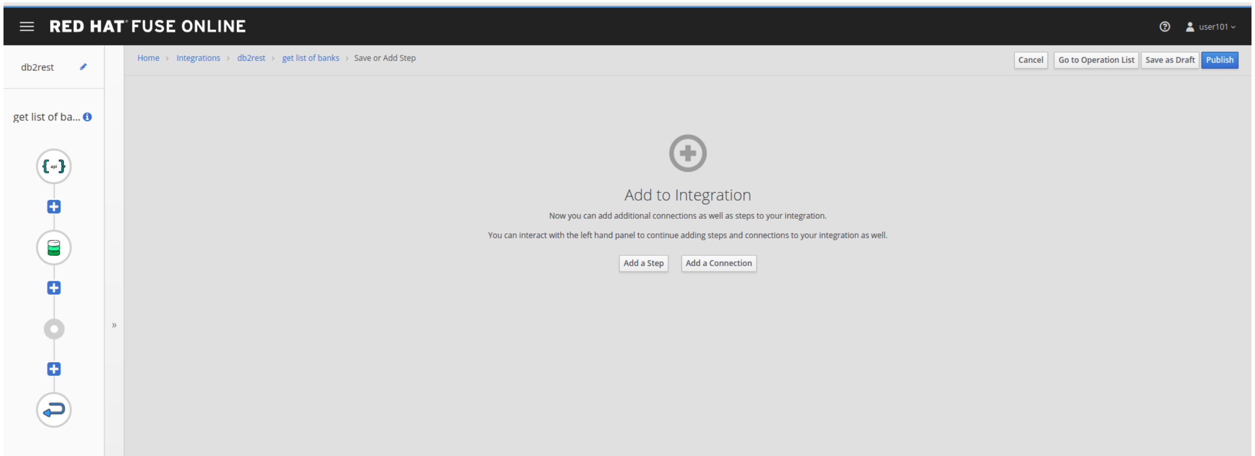
Select the log step.



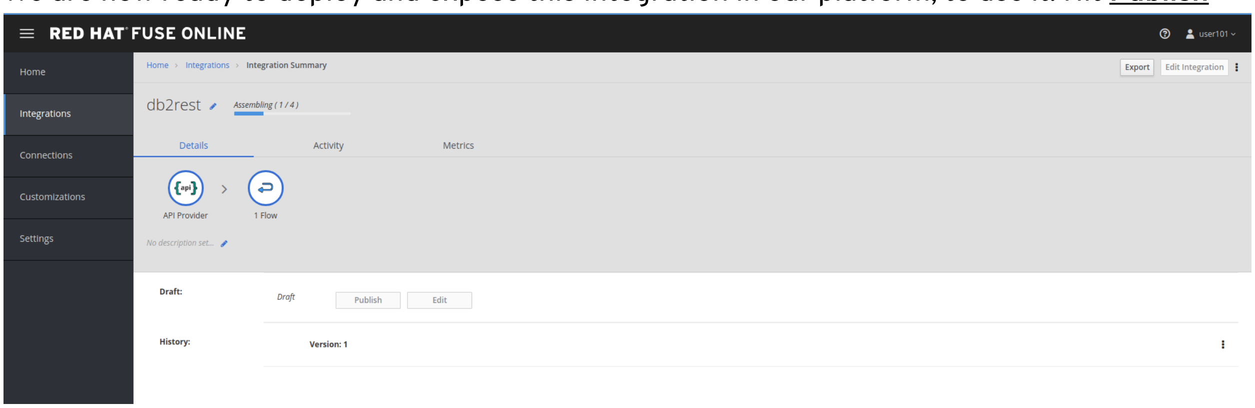
We are going to be sending a copy of the responses coming through to the integration log.



We are going to log just the message body.



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

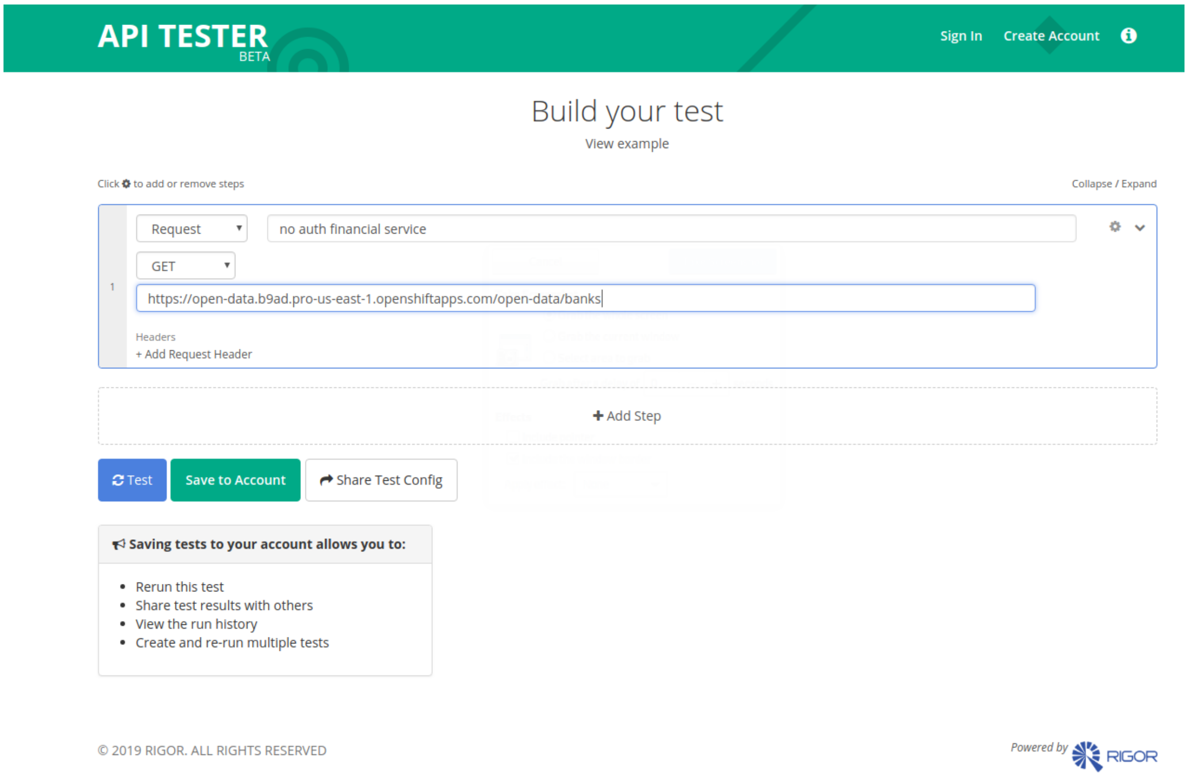


You can check the progress in building the integration changing through phases.
We can notice the platform is getting the required components and constructing the block.
When the building is completed we can test the Integration block.

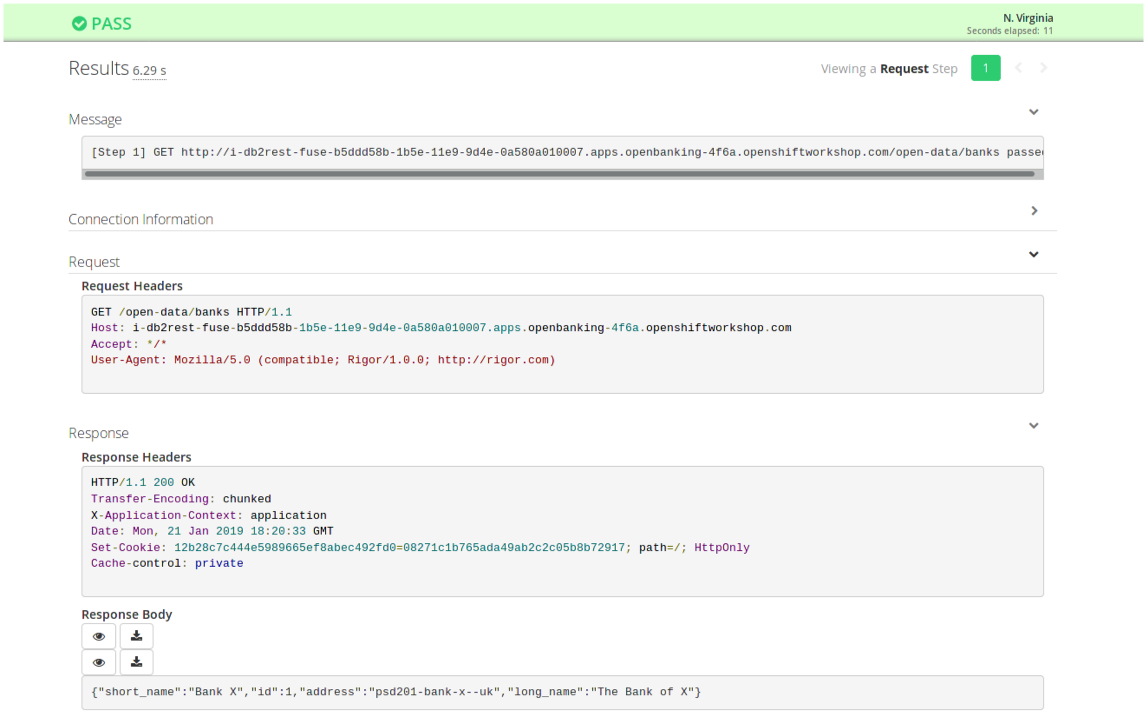
SINCE AUTO DISCOVERY FEATURE IS ACTIVE WE WILL NOT GET AUTOMATICALLY A URL WITH THE INTEGRATION BUILDING PROCESS, BUT API MANAGEMENT WILL BE ABLE TO SEE IT AND EXPOSE IT IN ANY CASE

Now that we have seen how to build the full integration you can test the integration just built, using this online tool https://apitester.com/ . You can use the following URL to test it http://i-db2rest-fuse-3fcd8e53-3e05-11e9-81f4-0a580a010007.apps.open-banking-2a38.openshiftworkshop.com/open-data/banks

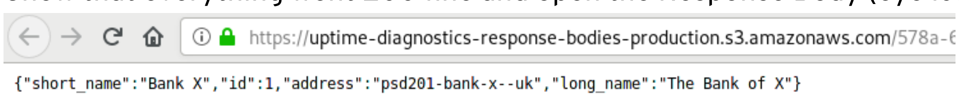
API tester works as a full Web or Mobile application but stripped down of the GUI.



Populate the fields with the following URL
http://i-db2rest-fuse-3fcd8e53-3e05-11e9-81f4-0a580a010007.apps.open-banking-2a38.openshiftworkshop.com/open-data/banks
and hit **Test**



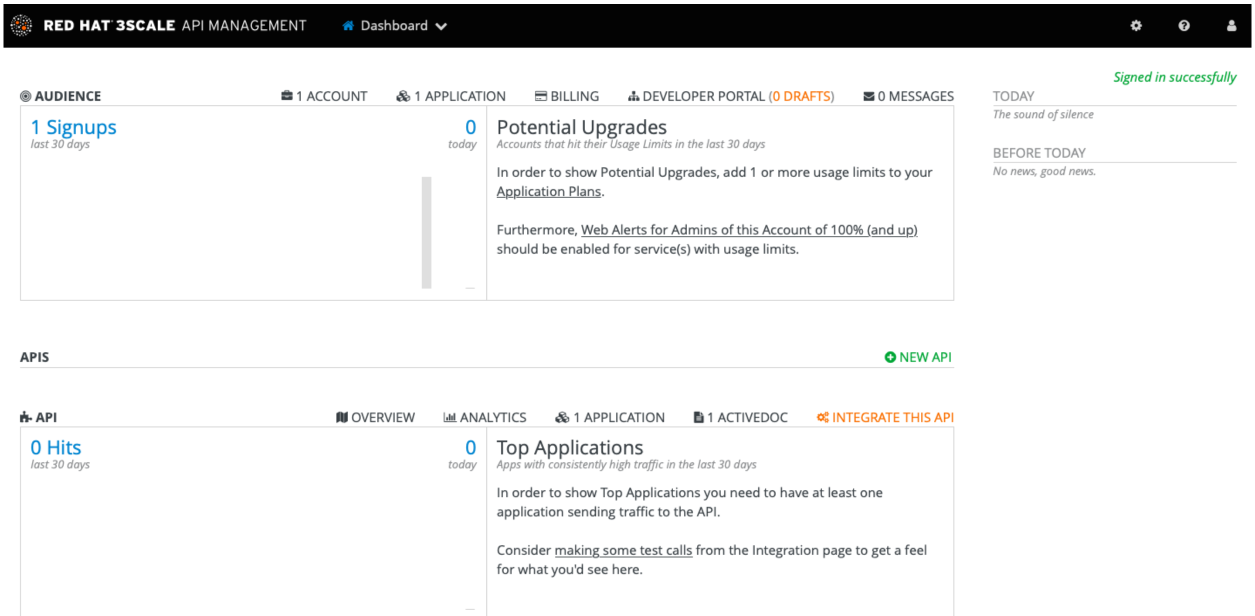
Show that everything went 200 fine and open the Response Body (eye icon)



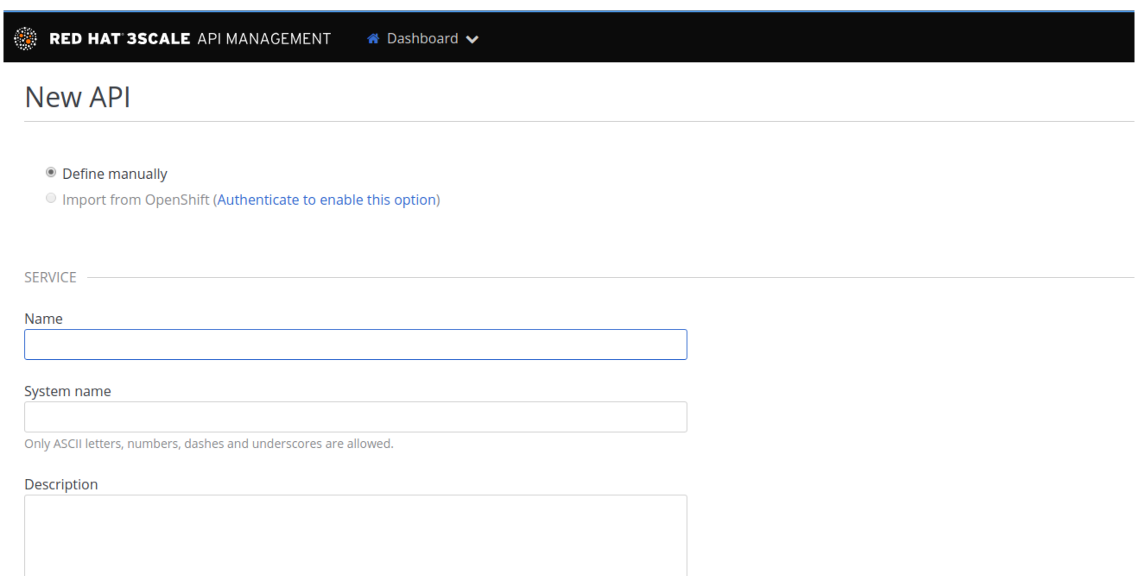
The response is in JSON format, with basic information given around banks (dummy data).

3scale

To reach 3scale as a user you can just go back to the main integr8ly tutorial dashboard and click on 3scale admin dashboard link (something like this https://userX-admin.apps.open-banking-2a38.openshiftworkshop.com/). 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.

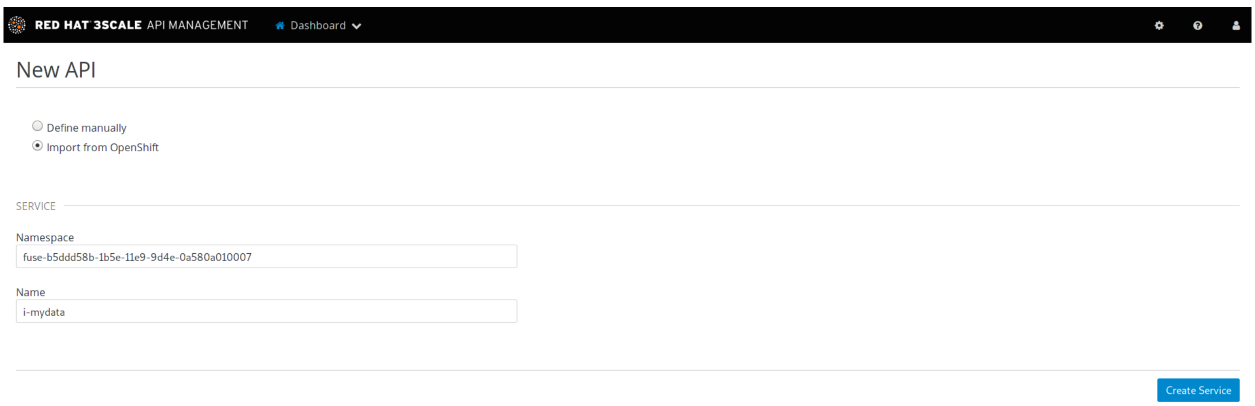


Let’s start managing and protecting the API we just created on Fuse Online.
Click on the green button **New API** and select **Import from OpenShift** (click on **Authenticate to enable this option**). We are going to be using the auto-discovery feature we saw before. We would be helped by this feature and would save time configuring the service.

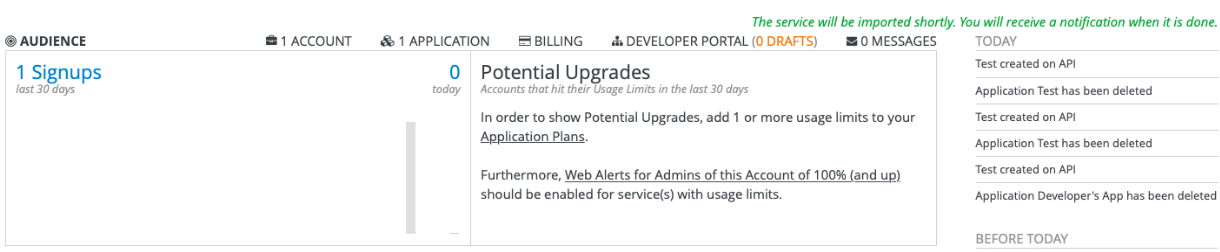


You are now presented with the permissions screen and you should click the *Allow selected permissions* button to proceed. This is to allow API management to go and look for services in our container platform.

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.



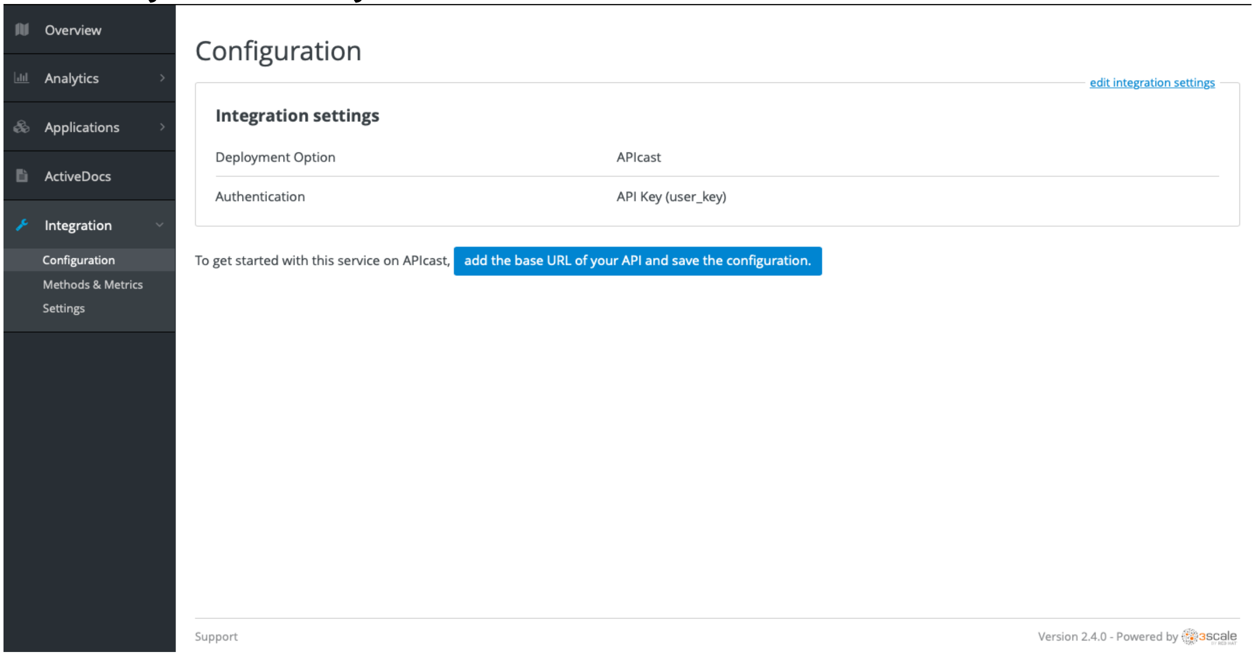
You will be redirected to the Dashboard view where a success message should appear as shown in the following screenshot.



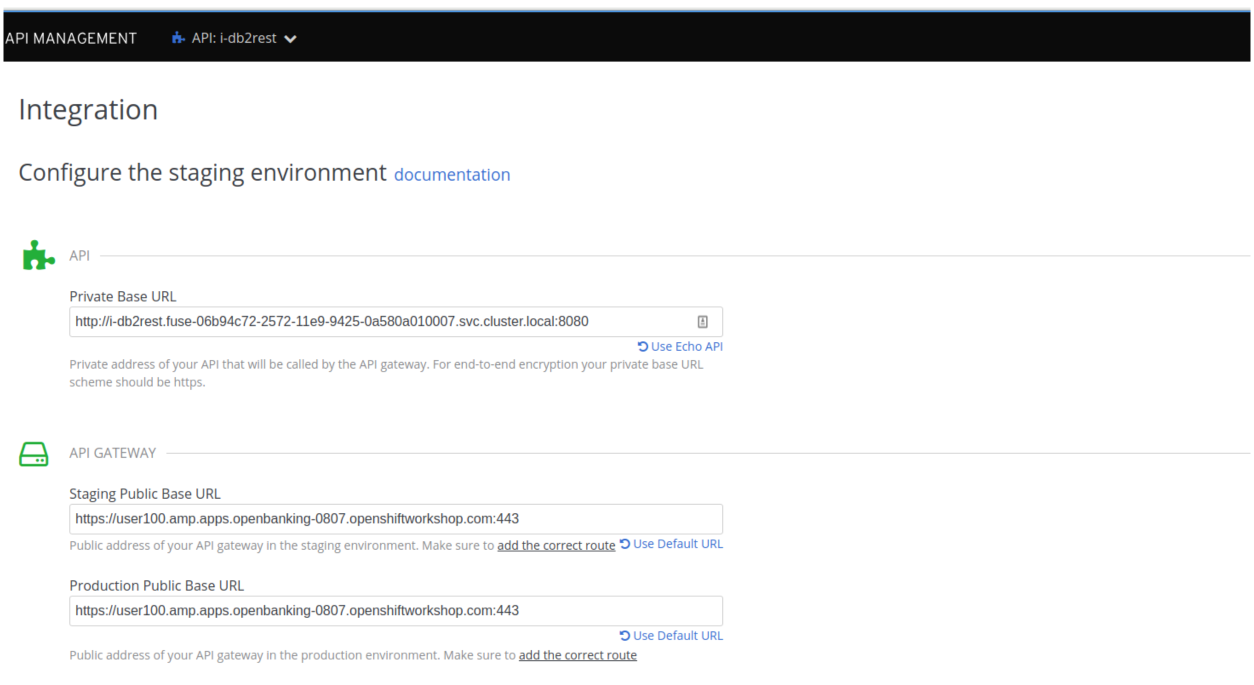
Once the service is created you will see that in the dropdown menu where you can also see **Audience.**

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.

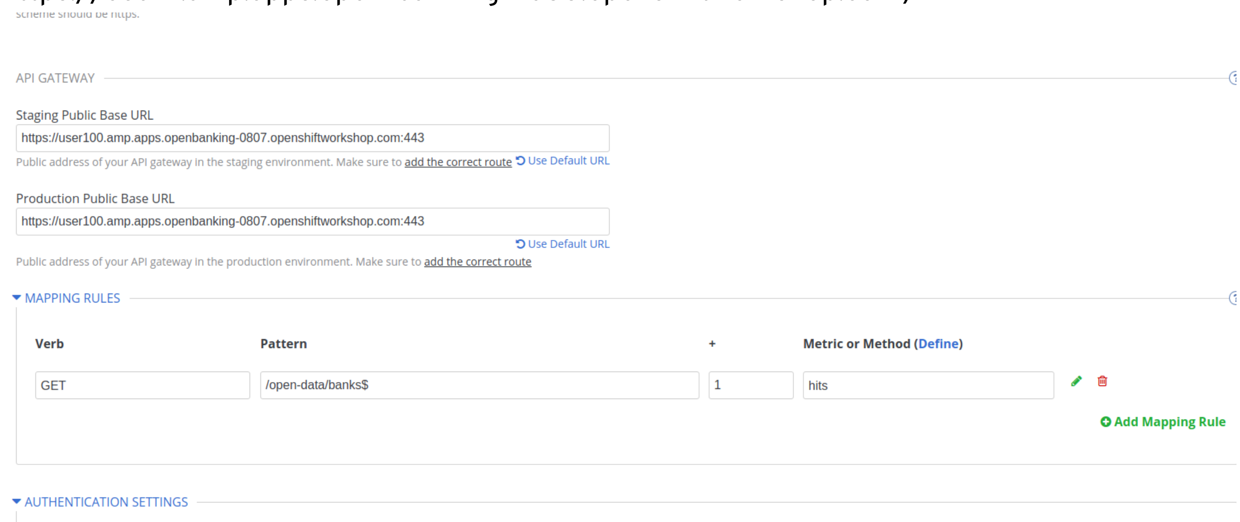
API-> Integration-> Configuration



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.

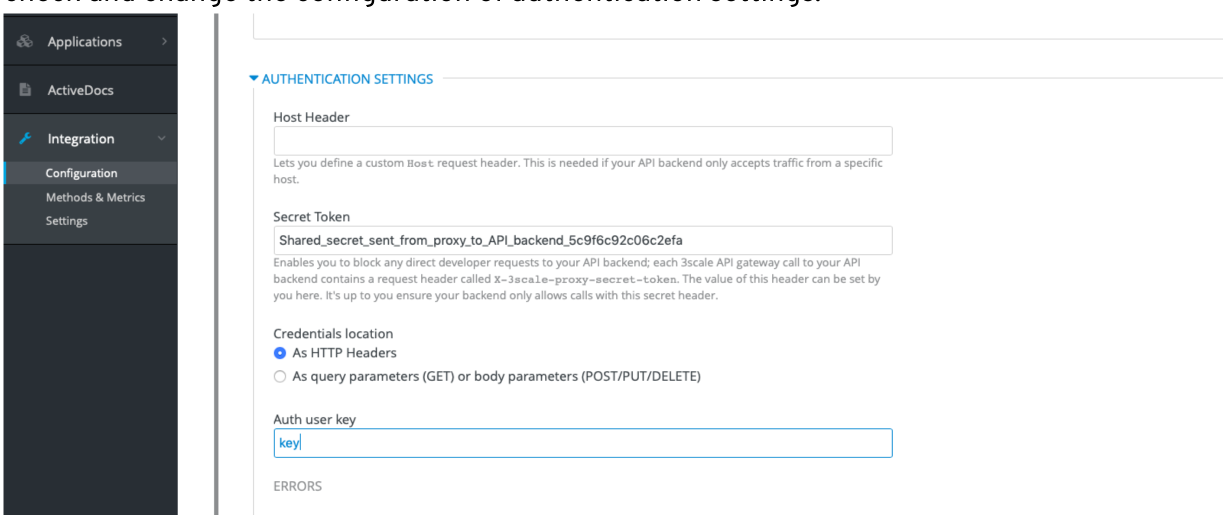


We have the section where we map the Backend API (or in this case Integration service) and then 2 URLs where we expose the managed API on the staging first and production gateways or infrastructure. We will be changing the Staging and Public addresses of the gateway. In this case we are not going to 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.open-banking-2a38.openshiftworkshop.com)

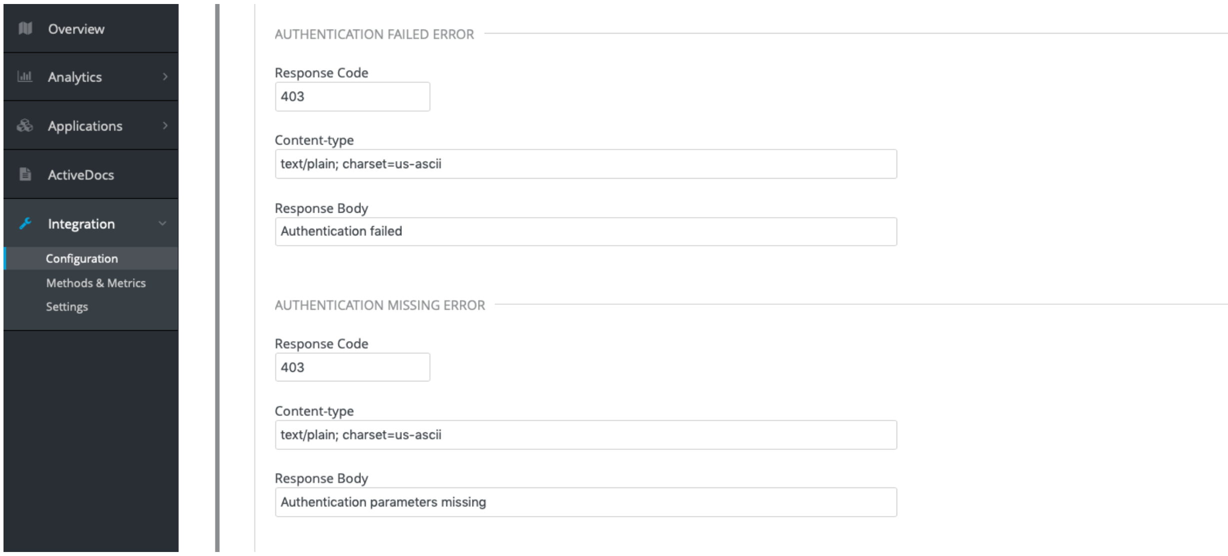


We will now make sure we map a single endpoint or resource in 3scale and disallow any other endpoint (i.e. the other endpoints have not been implemented yet so we are protecting them from being exposed).

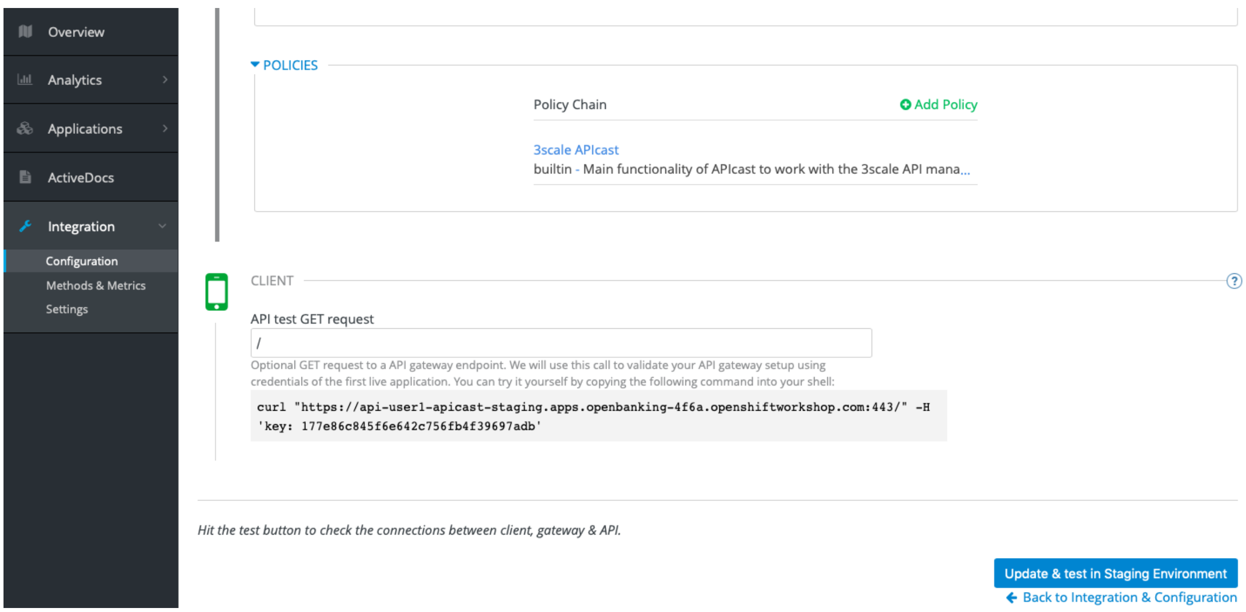
The endpoint you want to map is /open-data/banks$ (notice the $ at the end of the path that will allow us to make sure users cannot improperly access any other sub-resource). We can now check and change the configuration of authentication settings.



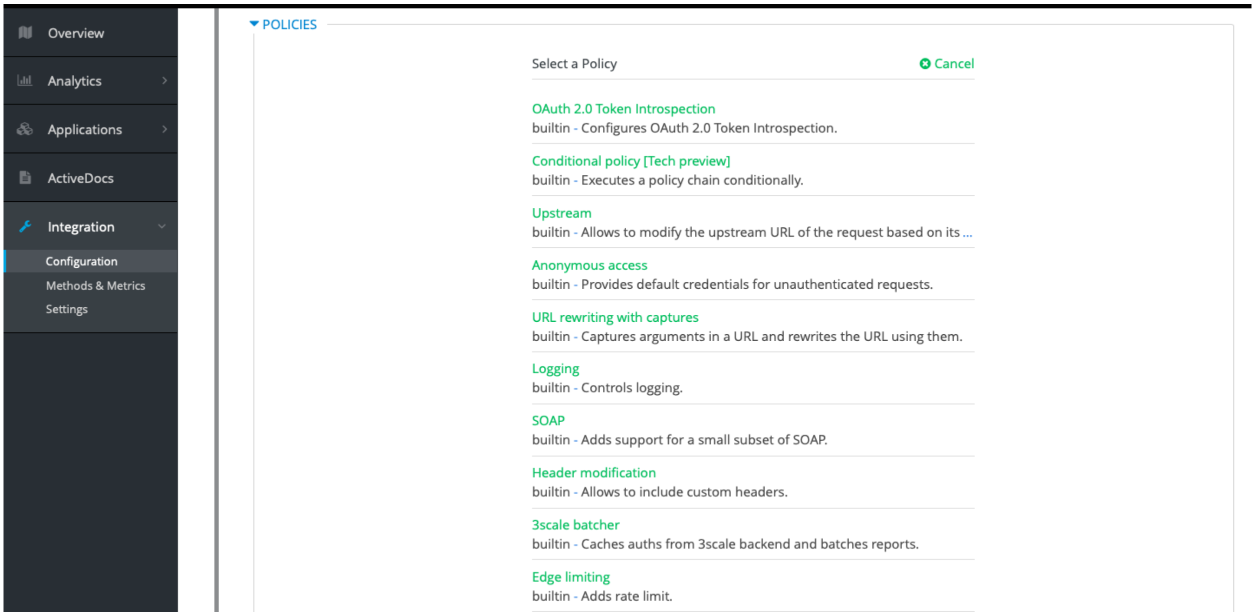
We see that we have already api key protection enabled, but we might want to pass this information as HTTP Header instead of HTTP parameter. We will also change the header name to ‘key’



You can also customize the error returned to the end user. **Policies**



These are like additional plugin that you can configure to adapt the service to your own preference. **Add Policy**

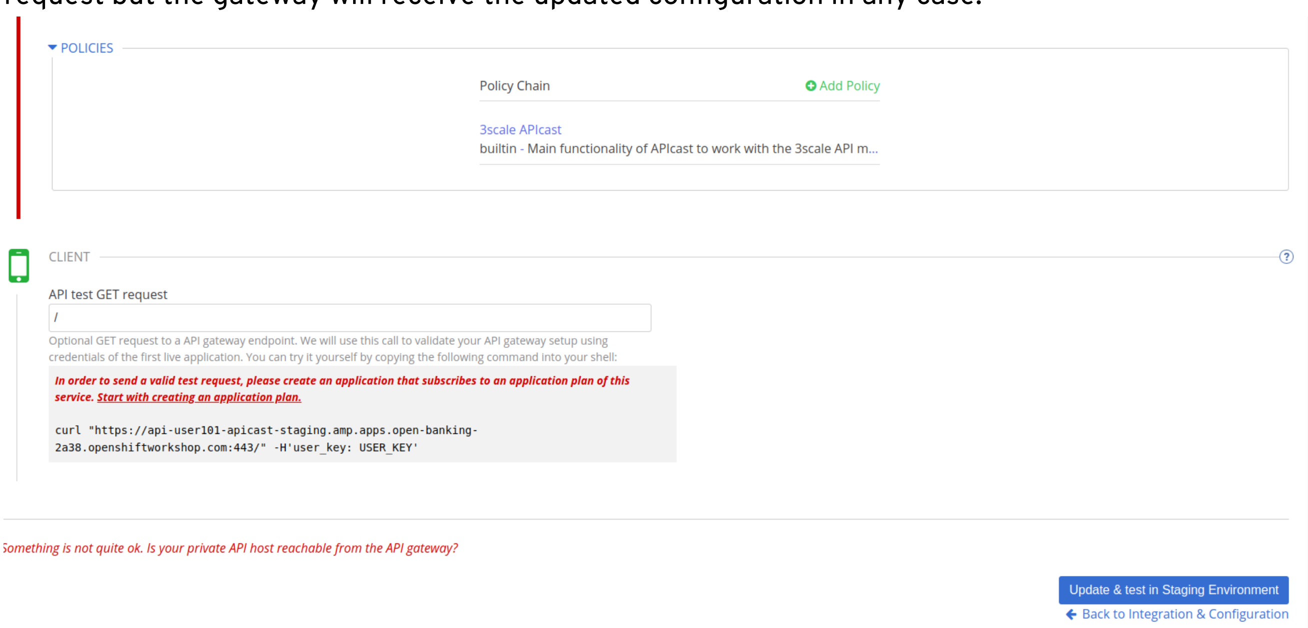


Several policies are available and the list is always increasing. Two policies / functionalities are of importance: SOAP policy to map SOAP services and advanced rate limit functionalities with edge limiting policy. Update the API test GET request field with the same pattern you are mapping above (excluding the $).

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.

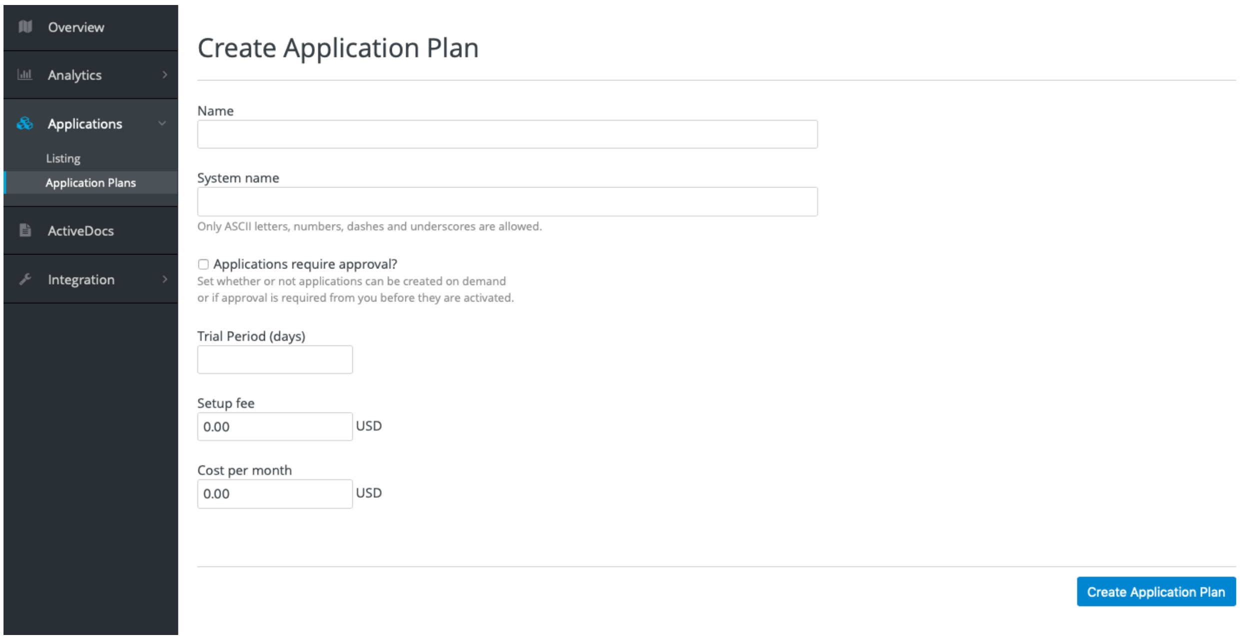


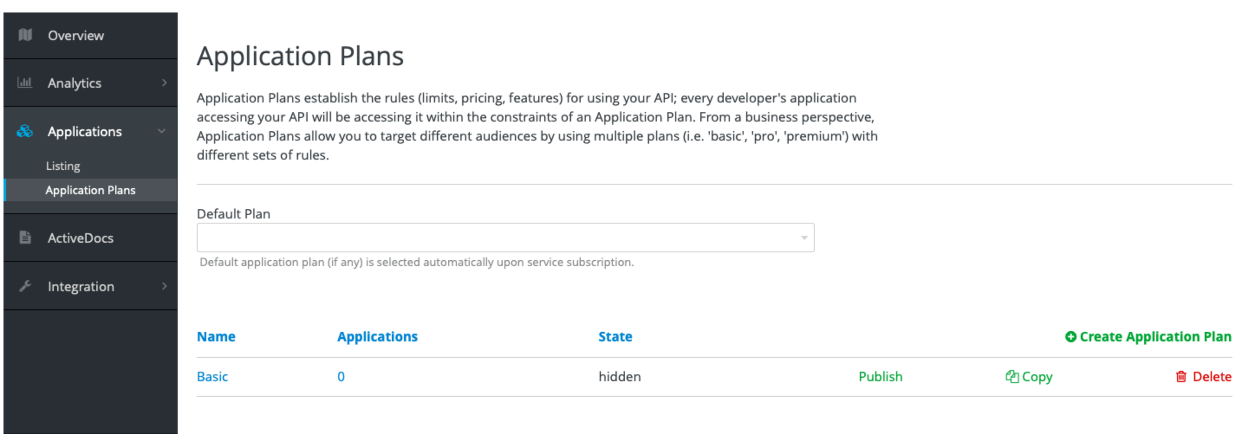
We will now fix the test request error as advised by the warning message.

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

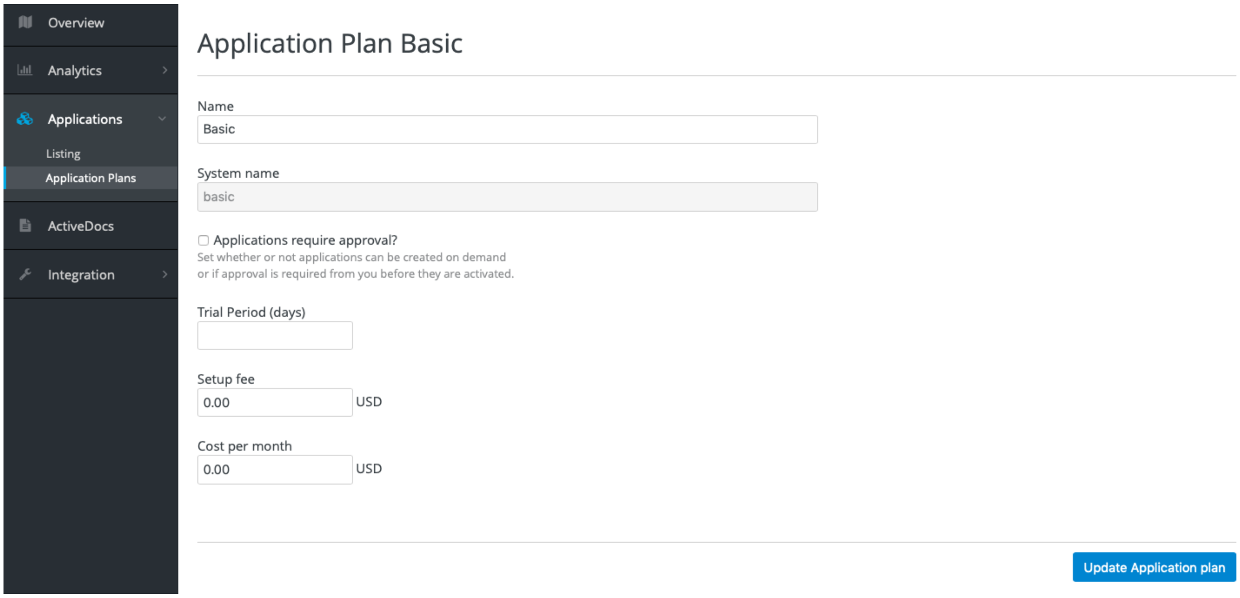
Within the red error message a link is generated “*create application plan*”. Click this link and you will be redirected to the **create application plan** form.

Fill out the **Name** and **System Name** fields on the **create application plan** form and click the blue button to submit the form.
You can safely ignore for now the monetization options and use whichever name you prefer.



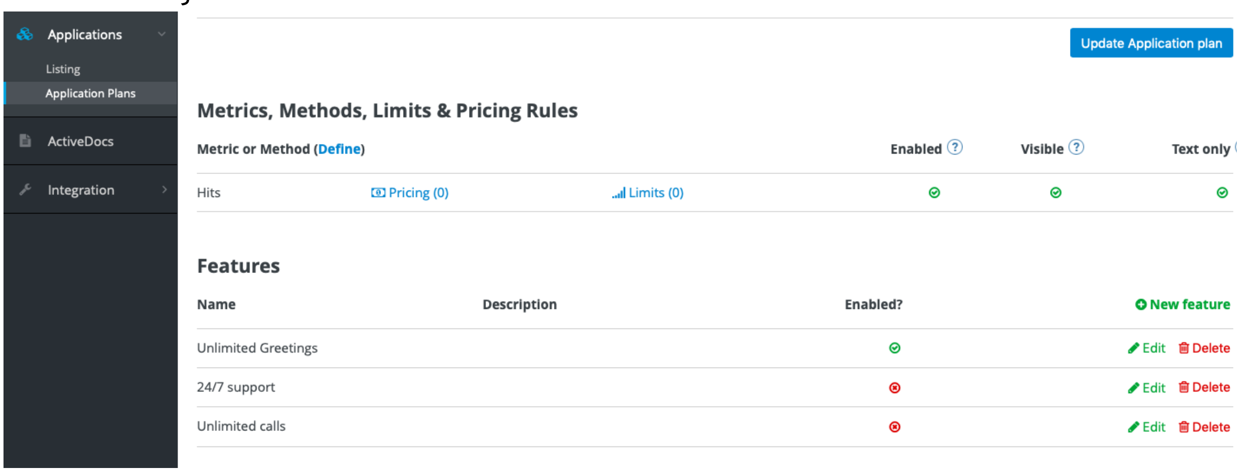


We see that we have 1 API contract (or Application Plan), but no application associated to it. The application plans are in **hidden** state by default, so let’s publish this one so that it is usable and visible on the Developer portal. Let’s open the application plan.

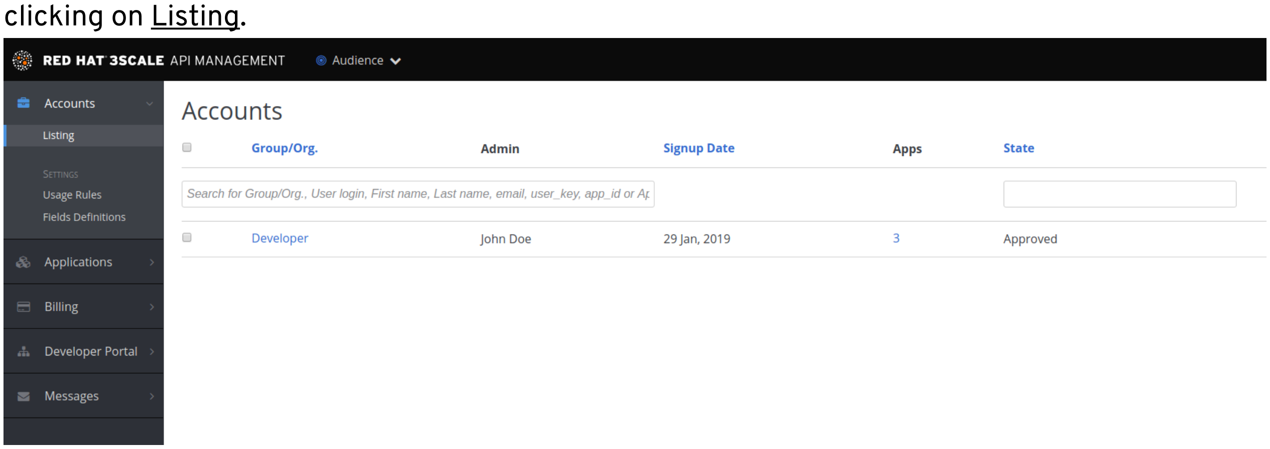


Main elements:

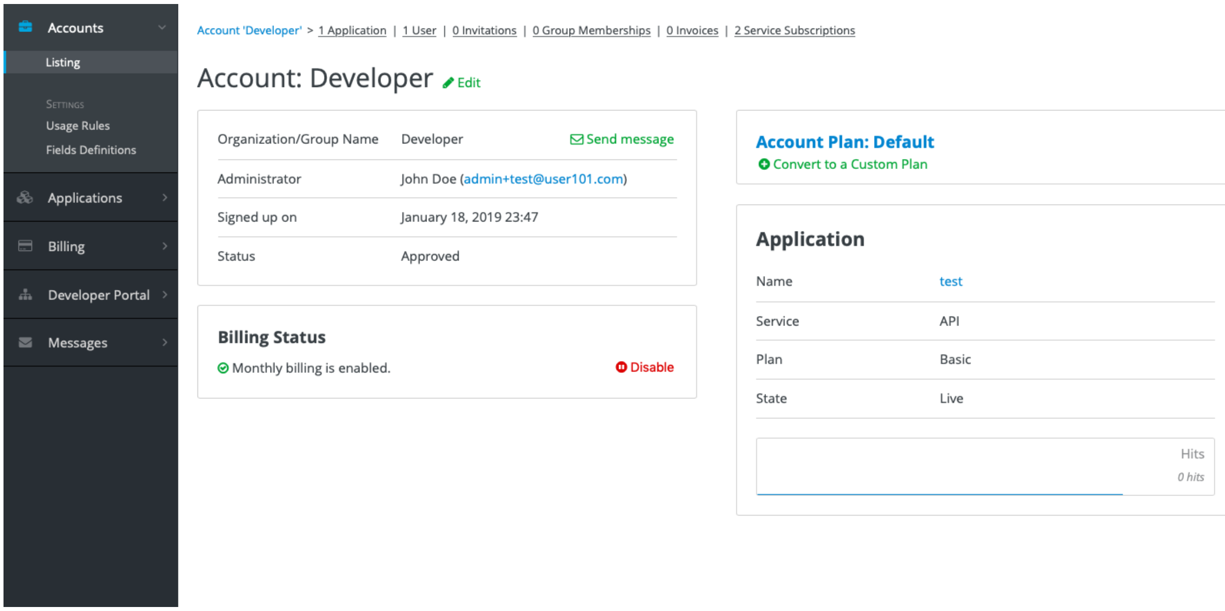
- Monetization settings (trial, setup, cost per month)
- Endpoint mapped (in this case generic Hits) and relative monetization and rate limiting settings



We can now switch to the Audience tab to create an Application to test the Configuration, by clicking on Listing.

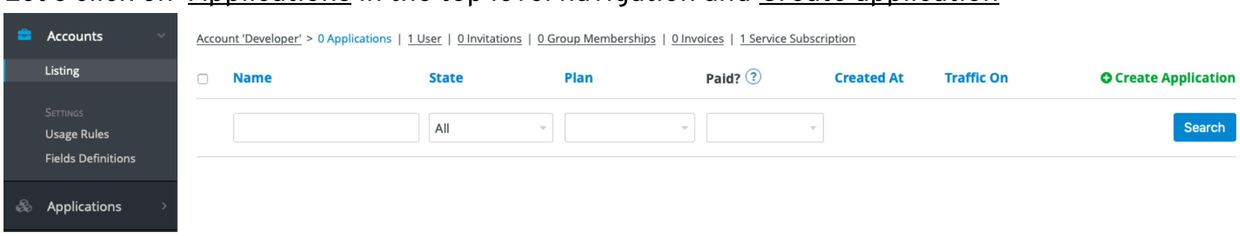


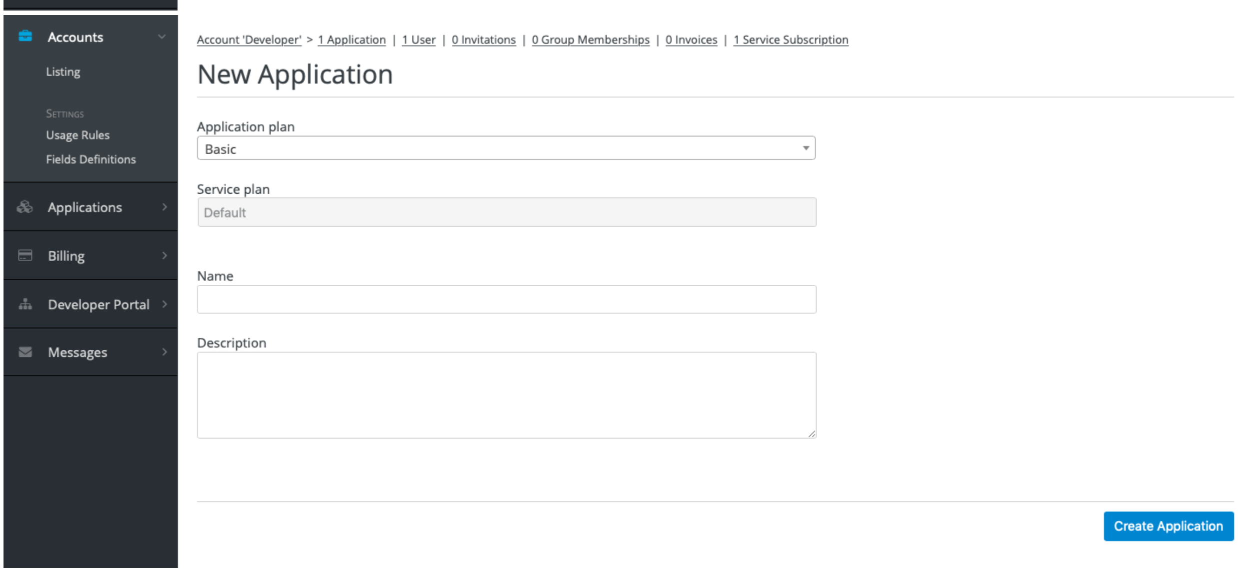
From here we can see how we can, as Provider, approve or deny Developers' Accounts registrations. Let's click on the default **Developer** Account



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.

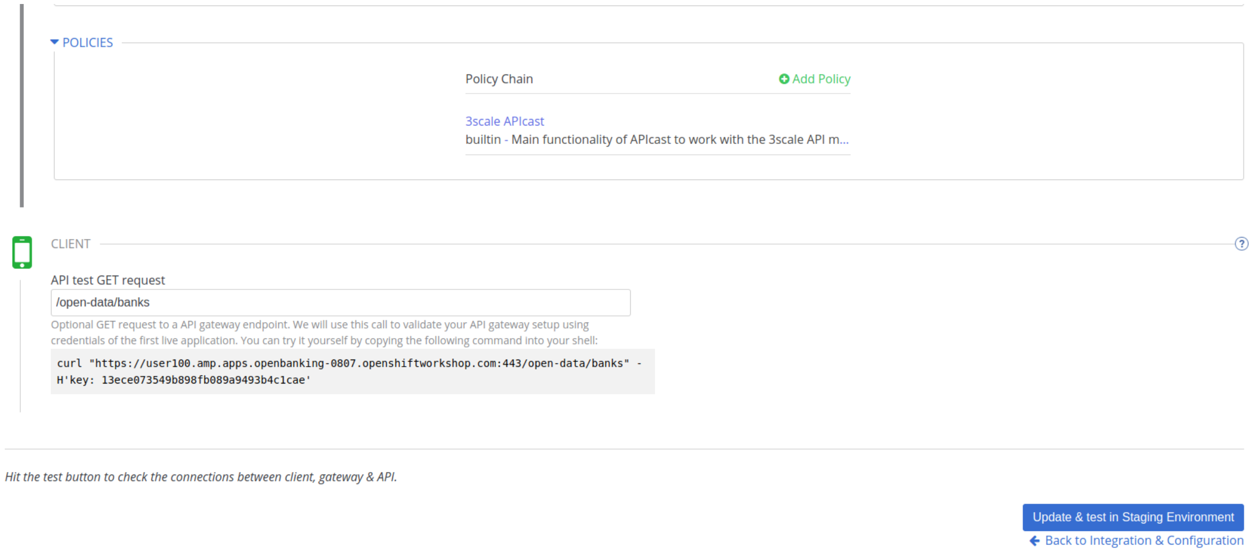
Let's click on Applications in the top level navigation and Create application



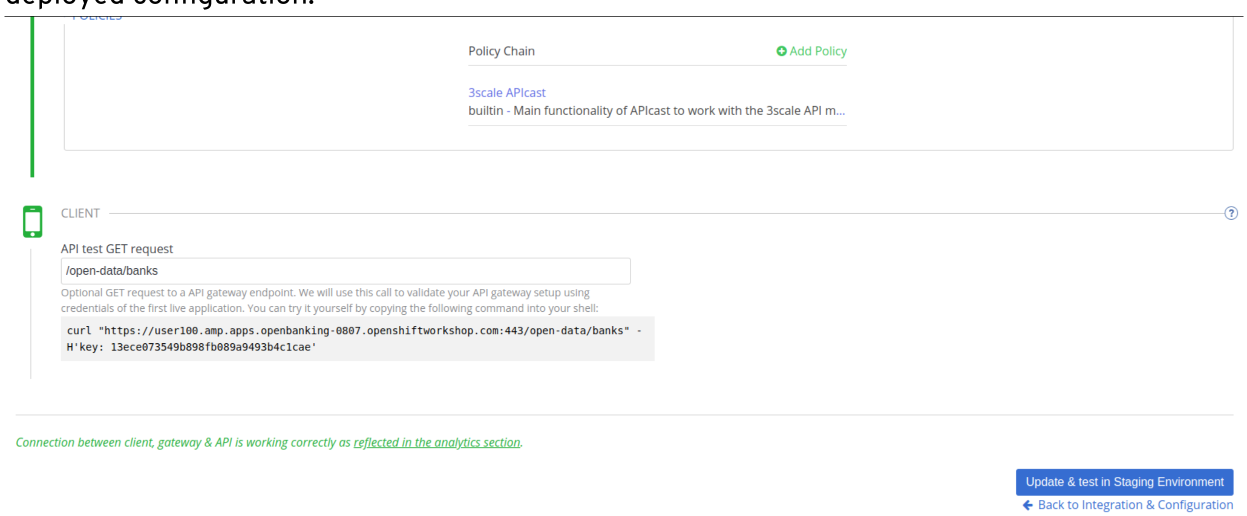


Here we can now subscribe the application to the **Application plan** we created on our new Service from the drop down field available. Let's fill in the rest of the fields with some basic details and click the big blue button: **Create Application**.

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. **API -> Integration -> edit Apicast configuration**



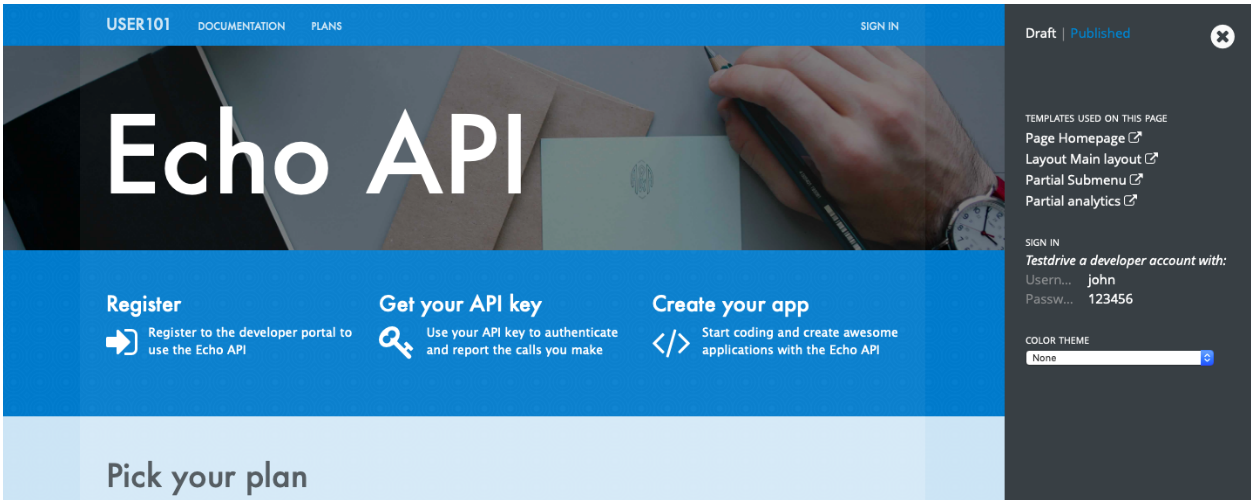
We now have a pre-populated key in the example curl statement, let’s try again testing the deployed configuration.



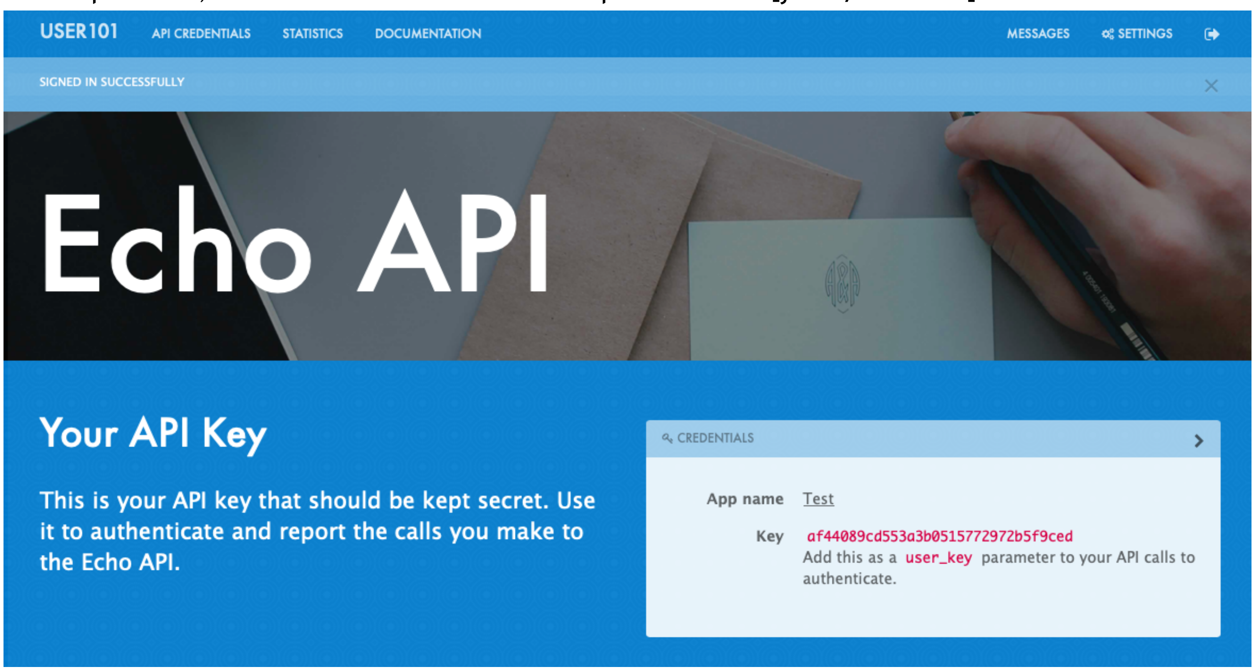
As we can see we turned the testing into a success.

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 Developer Portal**

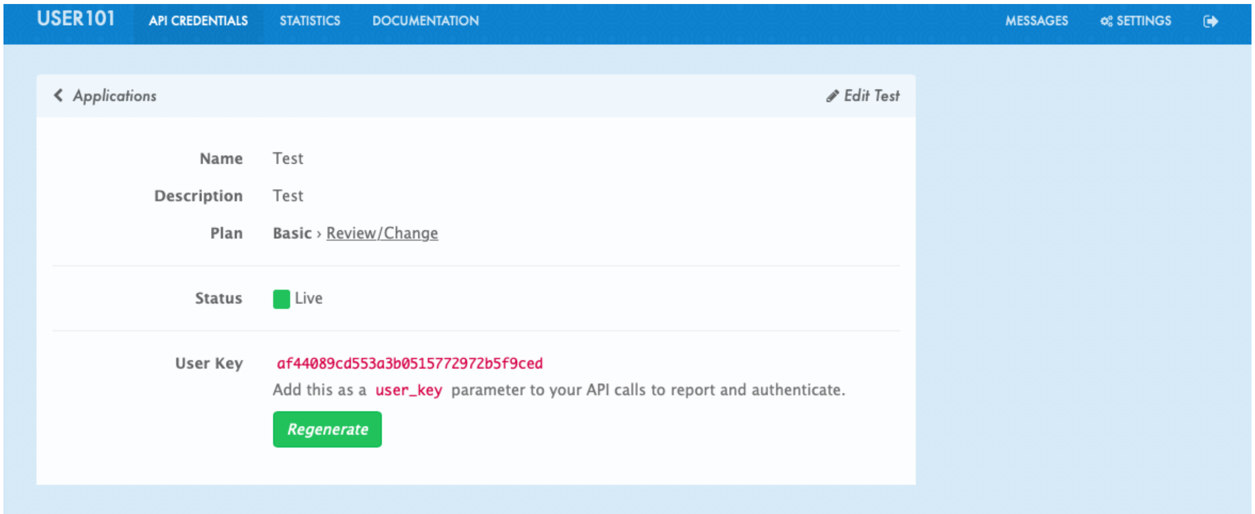
The sidebar allows us to edit pages of the Developer Portal live, but we are not interested in it now so we can close it.



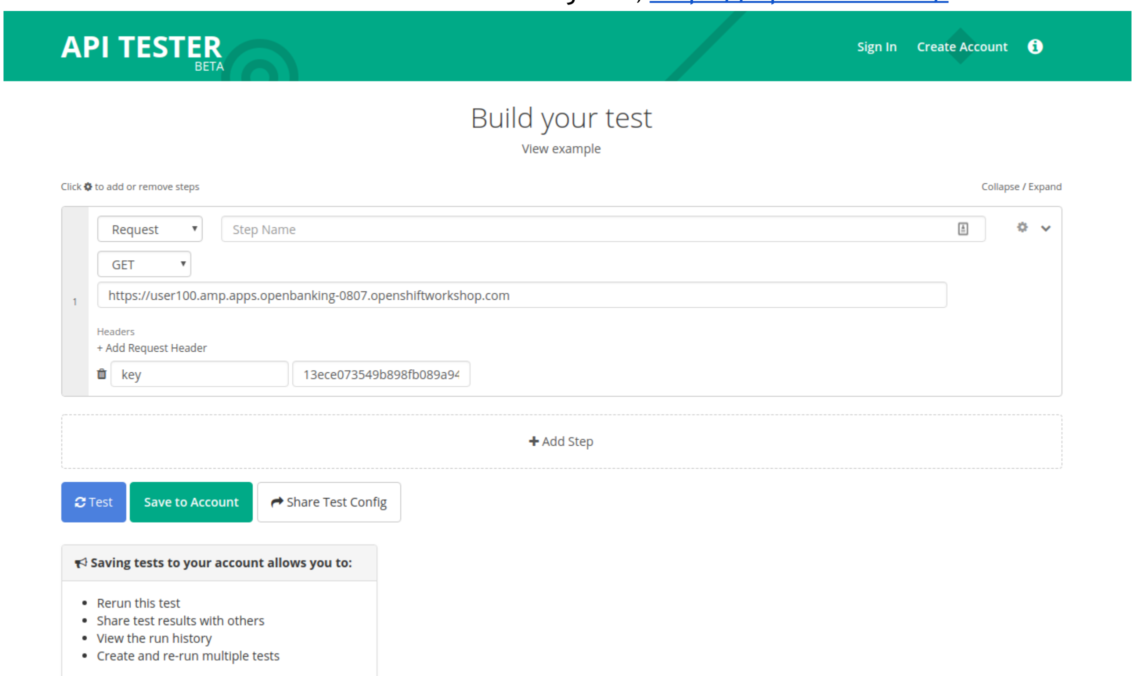
Let's sign in with the default user credentials provided in the sidebar. This is the default developer user, created for the default developer account [john / 123456]



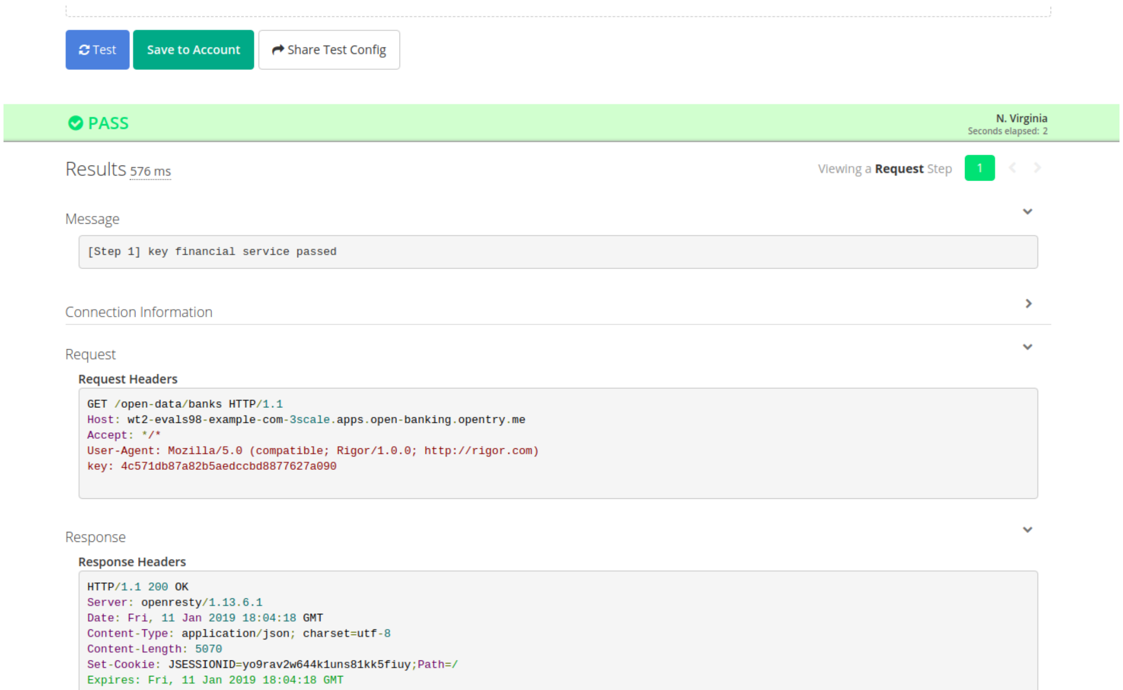
We are now logged in the developer's dashboard. Let's see the Applications I have created



I can now use the credential that I have 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, the following format should be configured in your service already: https://userX.amp.apps.open-banking-2a38.openshiftworkshop.com , remember the the key Header and the associated value.

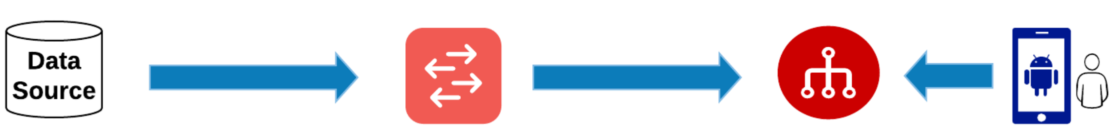


As we can see we succeed with 200 OK!

Let’s now just test with a wrong key or path then to confirm the role of API Management.

As expected we receive a Forbidden error.

Checkpoint



Break

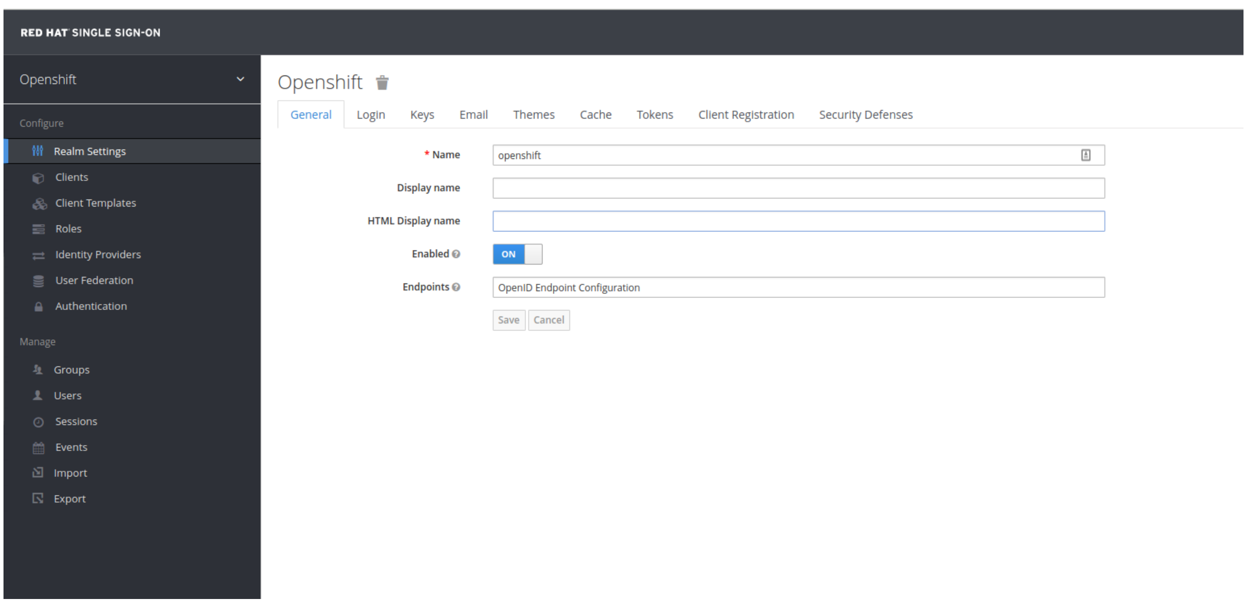
Practical Part 2

RH SSO and 3SCALE OIDC

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.

LAB BEGINS

Let's start with RH SSO main dashboard
https://sso-sso.apps.open-banking-2a38.openshiftworkshop.com/auth/admin/userX/console/#/realms/userX

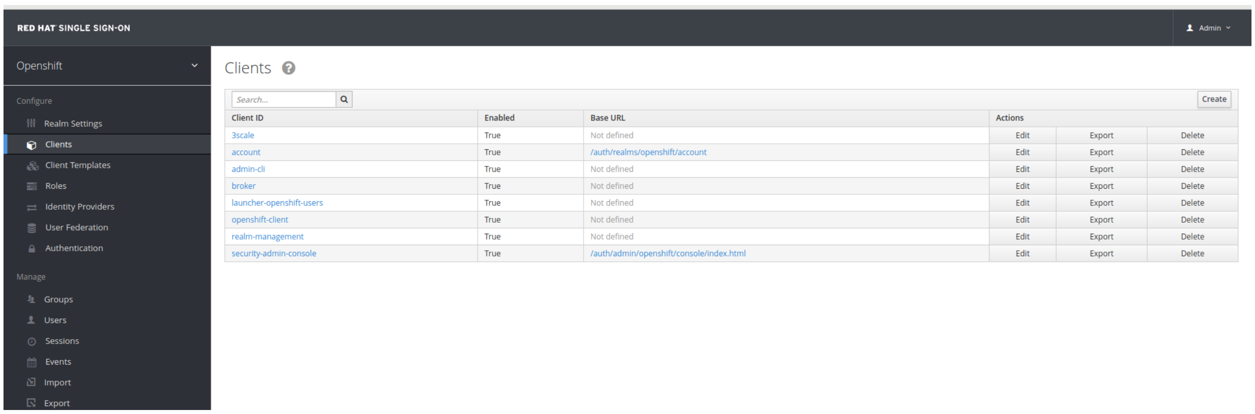


The realms are like separate instances of the platform, dedicated to separating users and applications. As we can see we can customize several aspects of the realm like the theme of the login page or the the tokens' default parameters. **Endpoints -> OpenID Endpoint Configuration**

```
{
    "issuer": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift",
    "authorization_endpoint": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/protocol/openid-connect/auth",
    "token_endpoint": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/protocol/openid-connect/token",
    "token_introspection_endpoint": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/protocol/openid-connect/token/introspect",
    "userinfo_endpoint": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/protocol/openid-connect/userinfo",
    "end_session_endpoint": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/protocol/openid-connect/logout",
    "jwks_uri": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/protocol/openid-connect/certs",
    "check_session_iframe": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/protocol/openid-connect/login-status-iframe.html",
    "grant_types_supported": [
        "authorization_code",
        "implicit",
        "refresh_token",
        "password",
        "client_credentials"
    ],
    "response_types_supported": [
        "code",
        "none",
        "id_token",
        "token",
        "id_token token",
        "code id_token",
        "code token",
        "code id_token token"
    ],
    "subject_types_supported": [
        "public",
        "pairwise"
    ],
    "id_token_signing_alg_values_supported": [
        "RS256"
    ],
    "userinfo_signing_alg_values_supported": [
        "RS256"
    ],
    "request_object_signing_alg_values_supported": [
        "none",
        "RS256"
    ],
    "response_modes_supported": [
        "query",
        "fragment",
        "form_post"
    ],
    "registration_endpoint": "https://secure-sso-sso.apps.open-banking.opentry.me/auth/realms/openshift/clients-registrations/openid-connect",
    "token_endpoint_auth_methods_supported": [
        "private_key_jwt",
        "client_secret_basic",
```

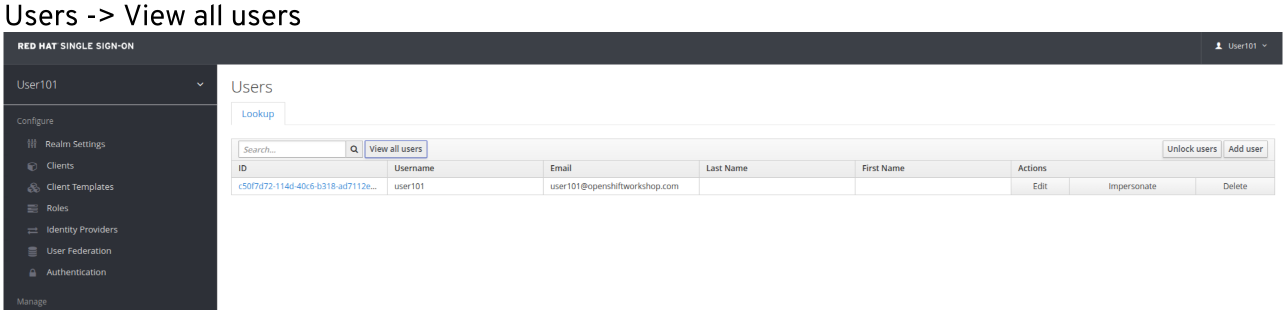

This is where we can find the public endpoints of the Realm exposed by RH SSO (we are going to be using this later).

Let’s now take a look at the Clients section.

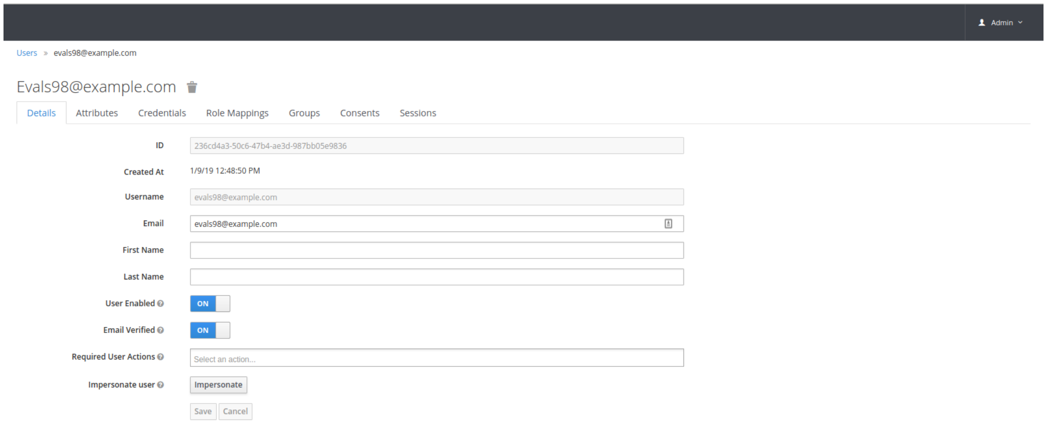


Here we can configure the web or mobile applications that will authenticate using RH SSO as an IDP (corresponding to applications in 3scale). As we can see there are some default clients dedicated to authentication in the integr8ly environment.

Users -> 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.

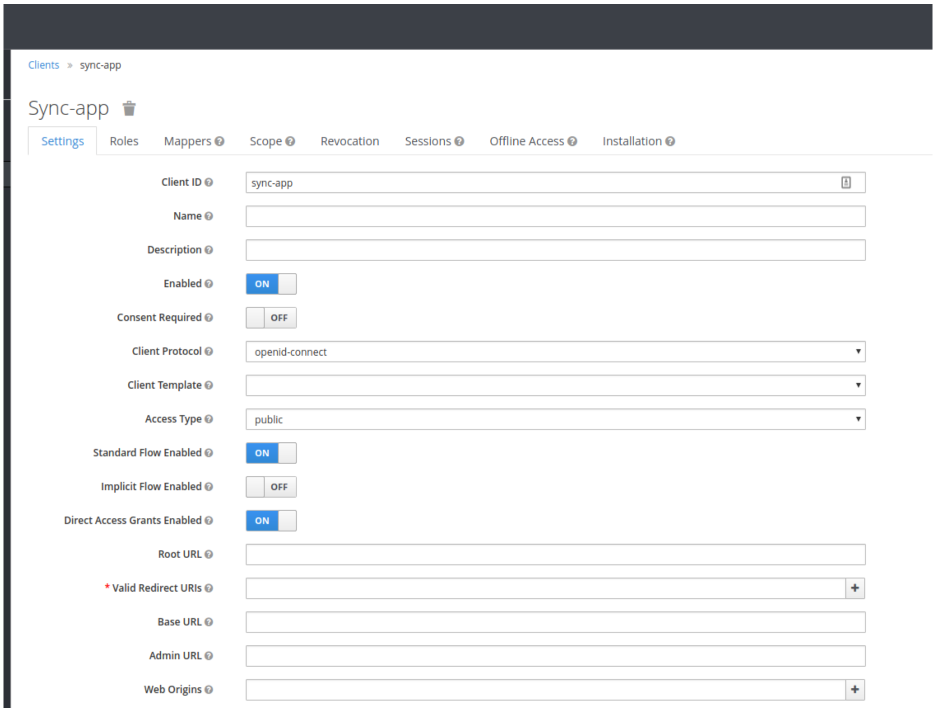


We can see here the type of information stored along with basic user details. The user profile can be customized with additional attributes as well.

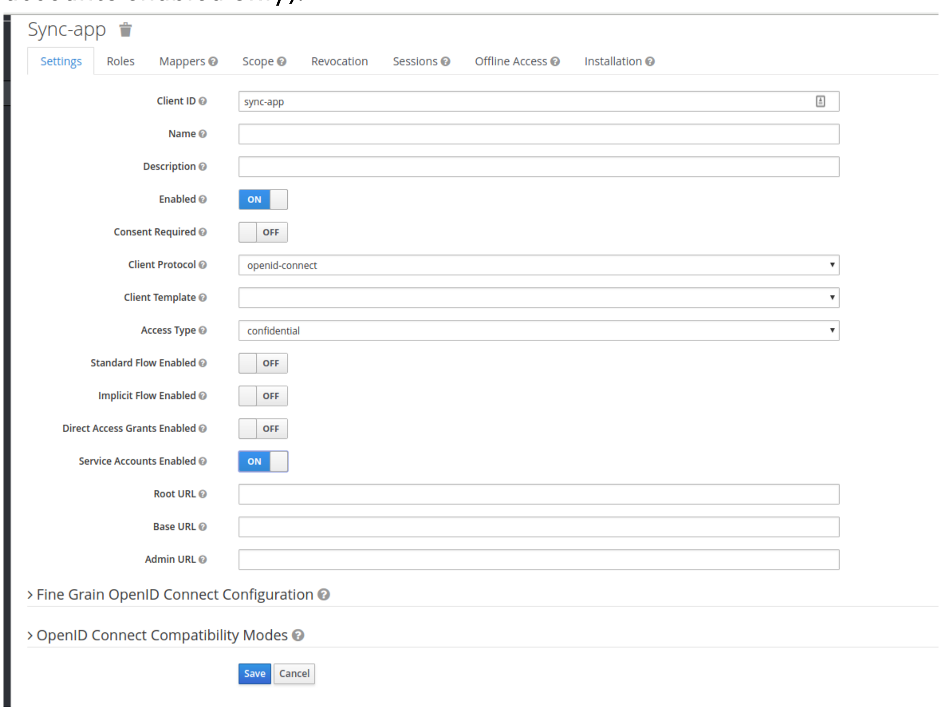
We will take advantage of one of the features available in OIDC and not in OAUTH which is dynamic client registration.

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. **Clients -> Create**

Let's call it sync-app and configure the other details required to let it communicate with 3scale.

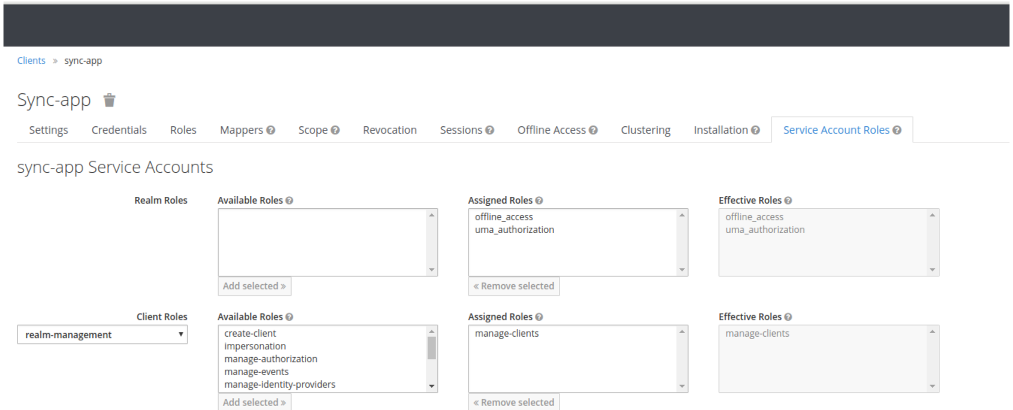


We are going to give it only the rights to create applications on behalf of 3scale (*service accounts enabled* only).



Save -> Service account roles

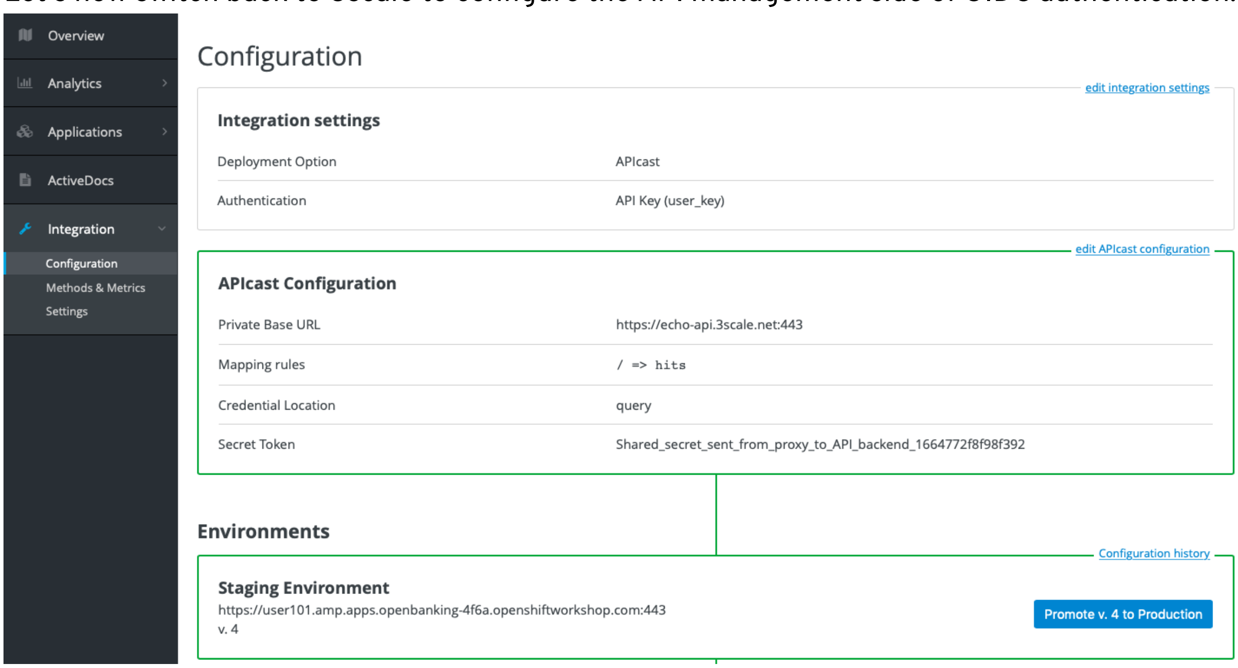
Add manage-clients to the assigned roles in this window, by picking realm-management in the Client roles menu, this special role allows it to create application on behalf of API management. Then click add selected



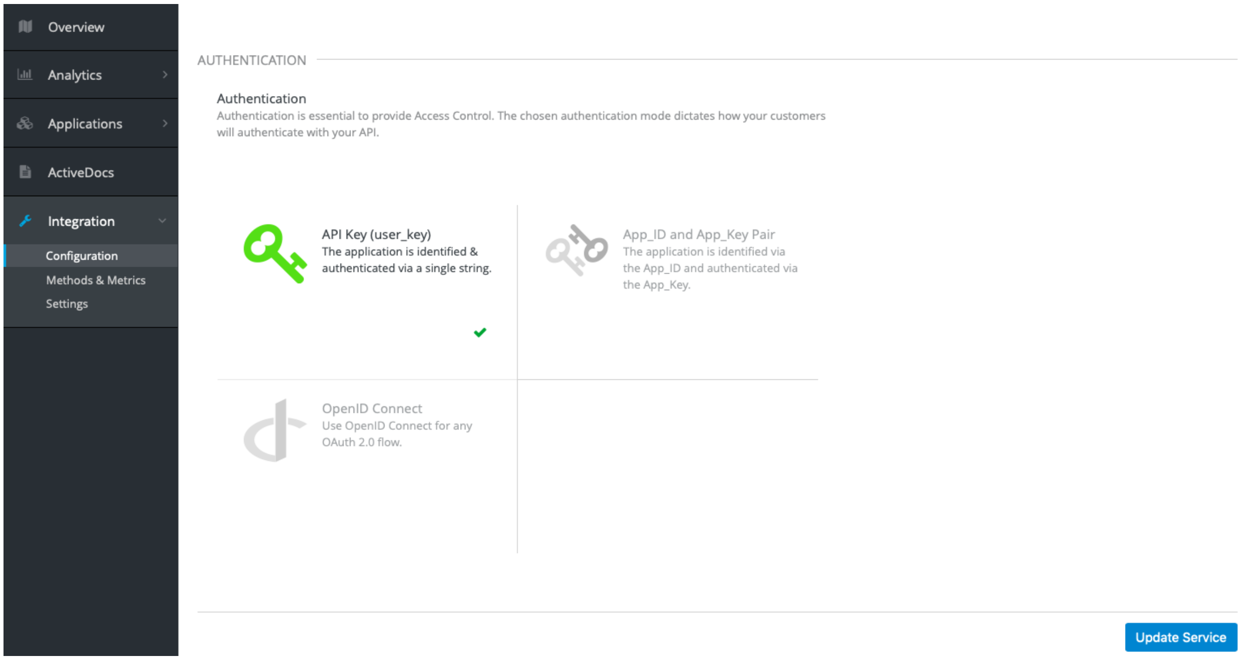
And now we are ready to use the client credentials inside 3scale OIDC configuration section. To authenticate as we were an end user, we can just re-use our predefined user (already used to login in the rest of the components)

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

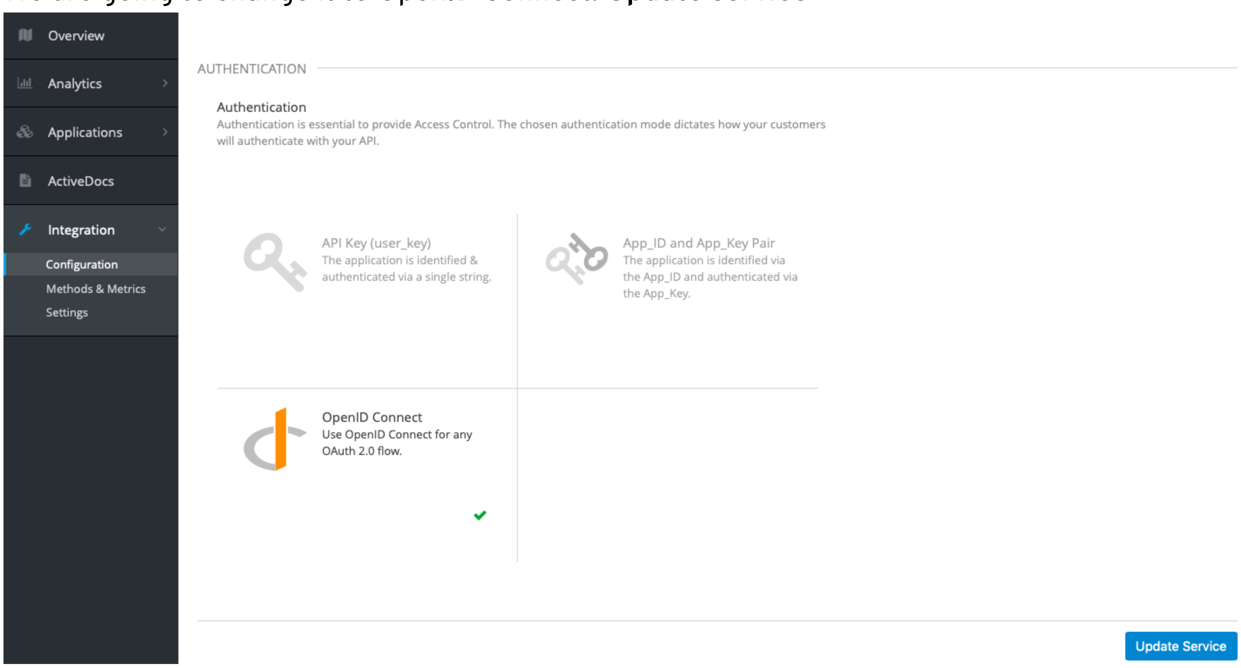
Let's now switch back to 3scale to configure the API management side of OIDC authentication.



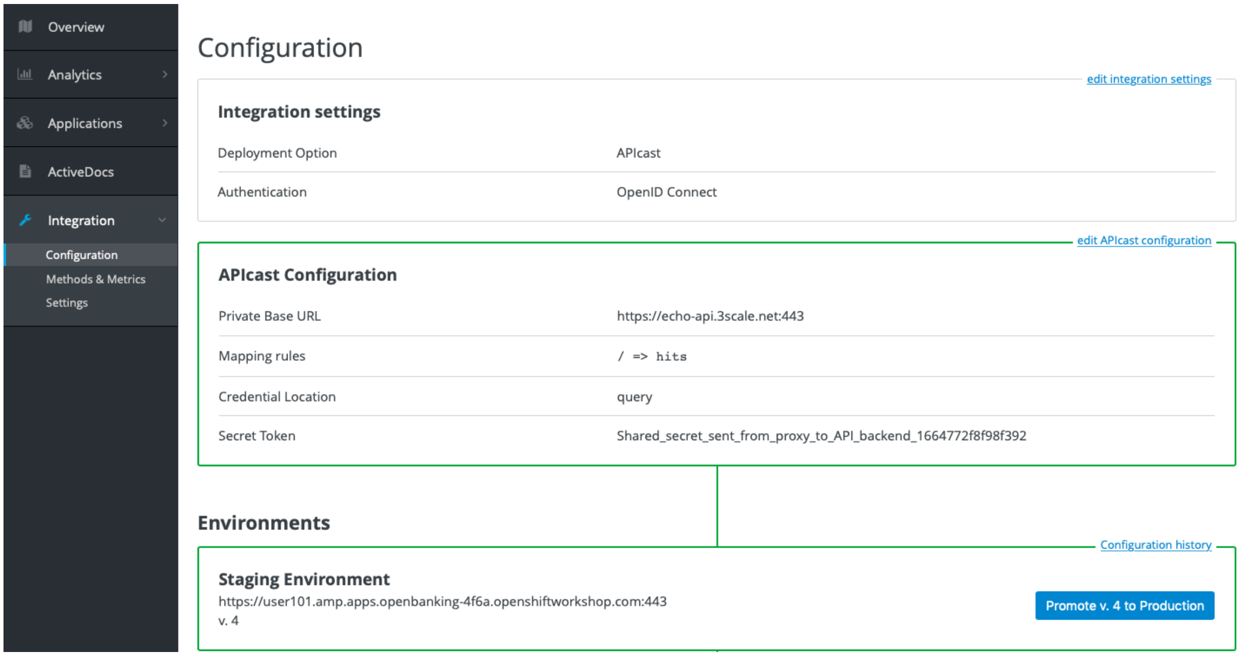
We can see that we have a fully configured API with API key as the Authentication method. We are going to change it to the more secure OpenID Connect, to ensure our financial data are protected from attacks performed when a key is compromised. **Edit integration settings**



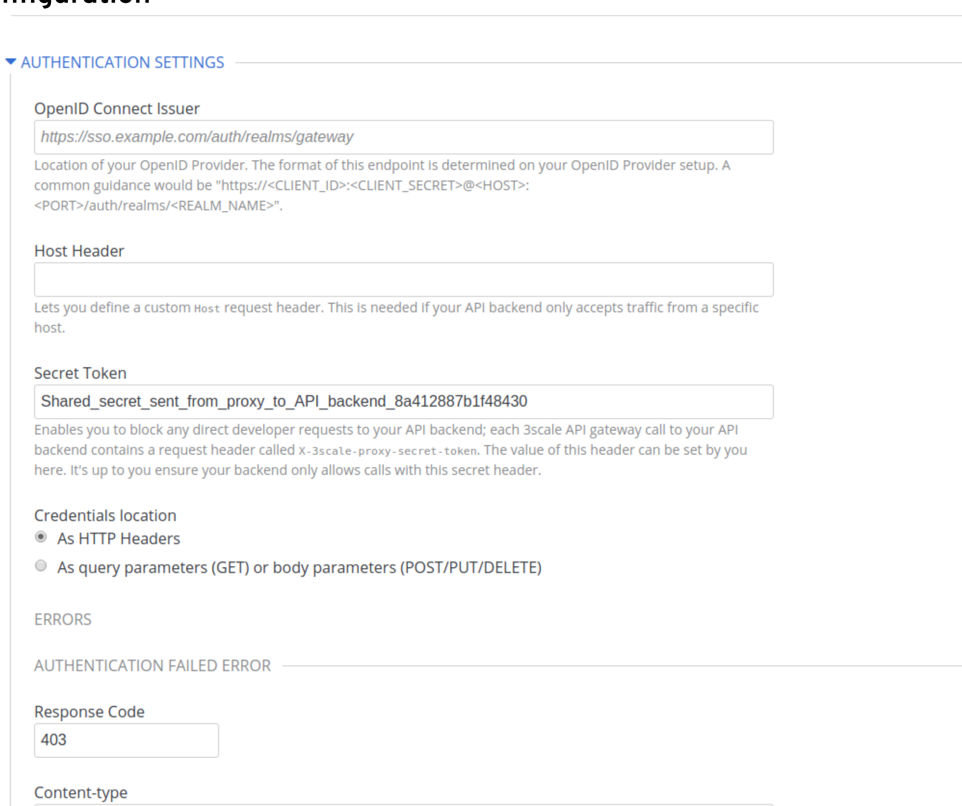
We are going to change it to OpenID Connect. **Update service**



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.



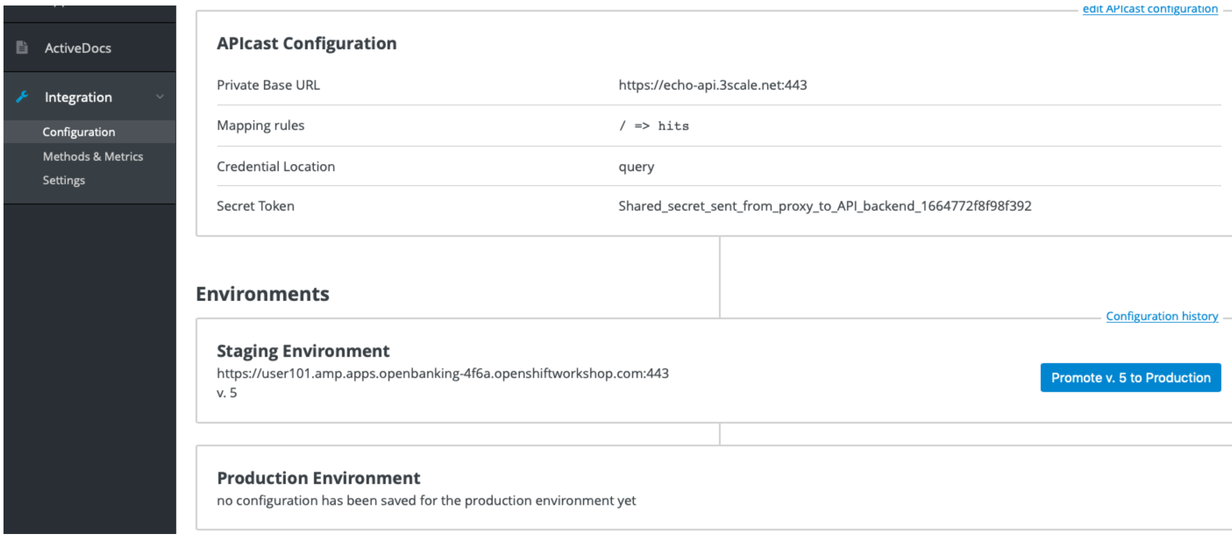
We have now changed the authentication method, we are just left with configuring the correct IdP inside 3scale to make sure it is authenticating the requests with RH SSO. **edit apicast configuration**



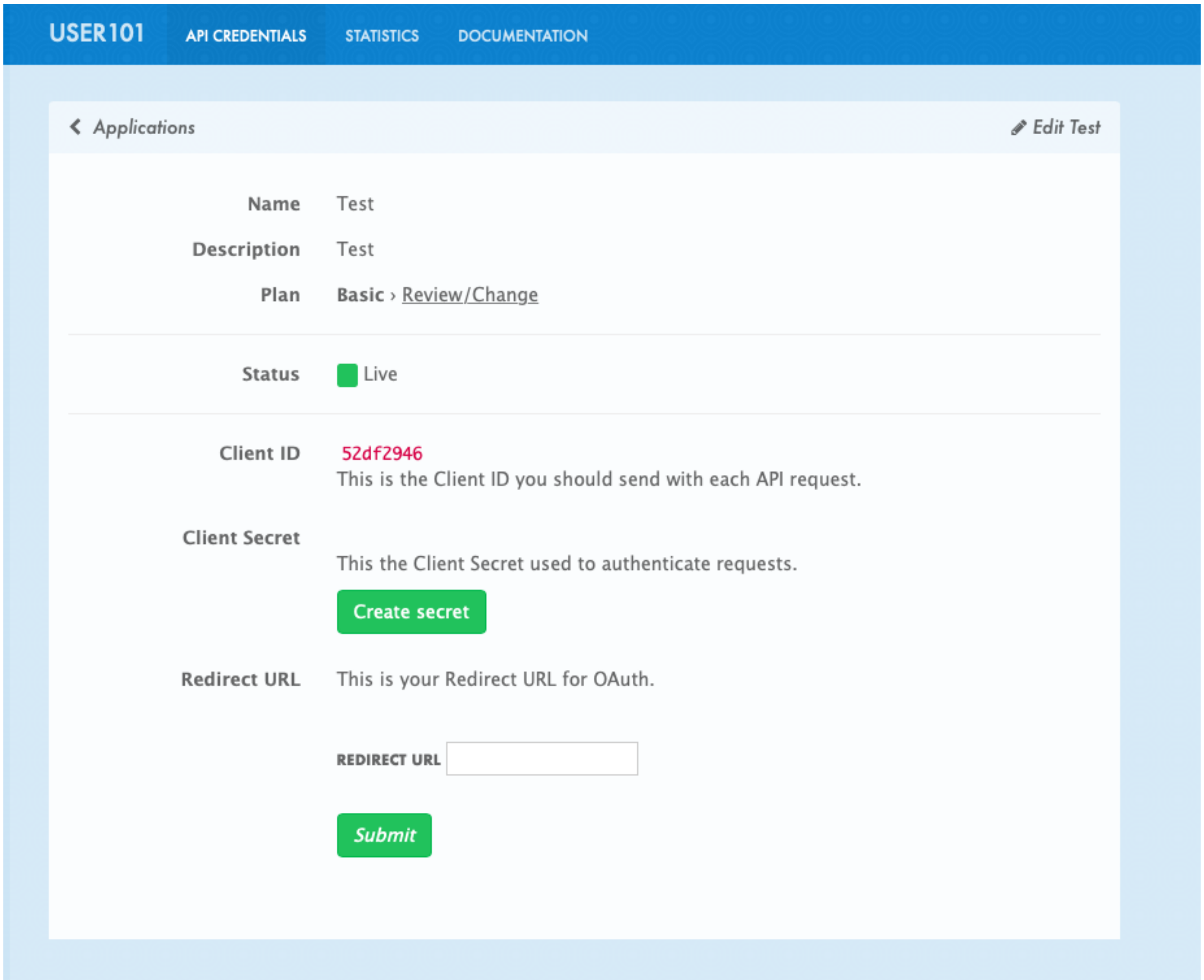
As we see we have a dedicated field for this purpose now: **OpenID Connect Issuer**

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

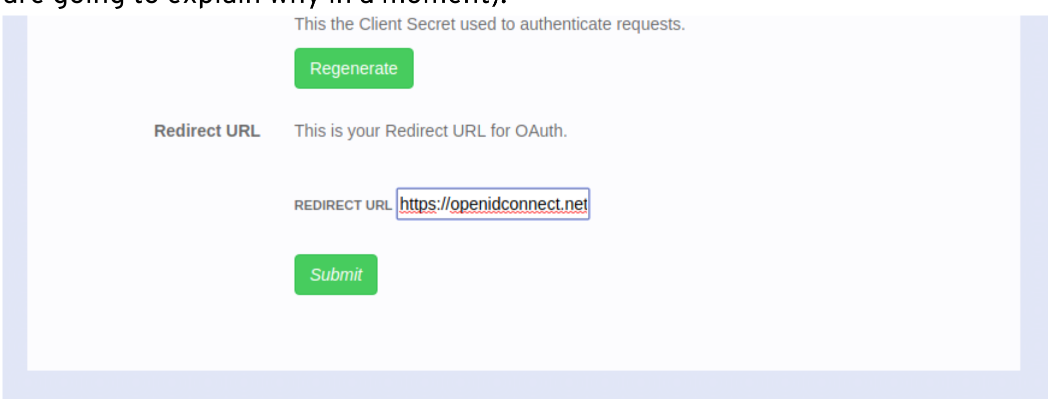
http://client-id:client-secret@<idp-public-endpoint>
where client-id: sync-app
client secret: <defined-during-configuration>
idp-public-endpoint:
sso-sso.apps.open-banking-2a38.openshiftworkshop.com/auth/realms/<userX>
Lastly, change the Credentials location to As HTTP Headers
And update the staging environment and promote the configuration to production by clicking the blue button *Promote to production*.



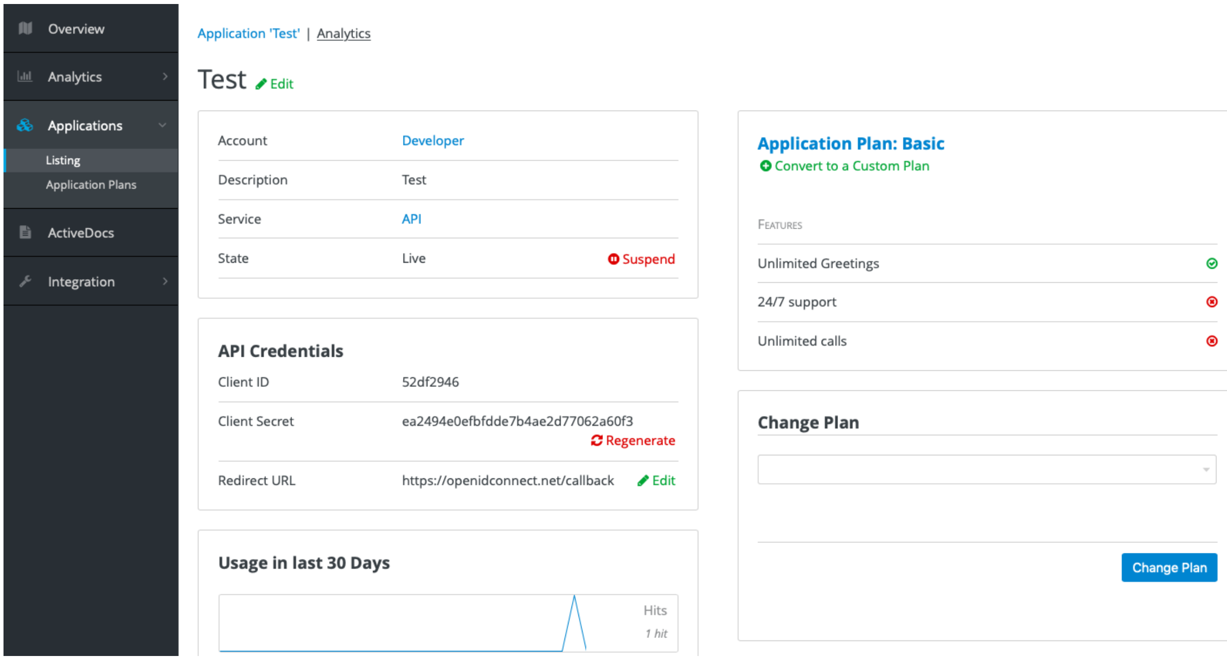
Let’s now switch user perspective and get in the shoes of the developer and open their Applications section.

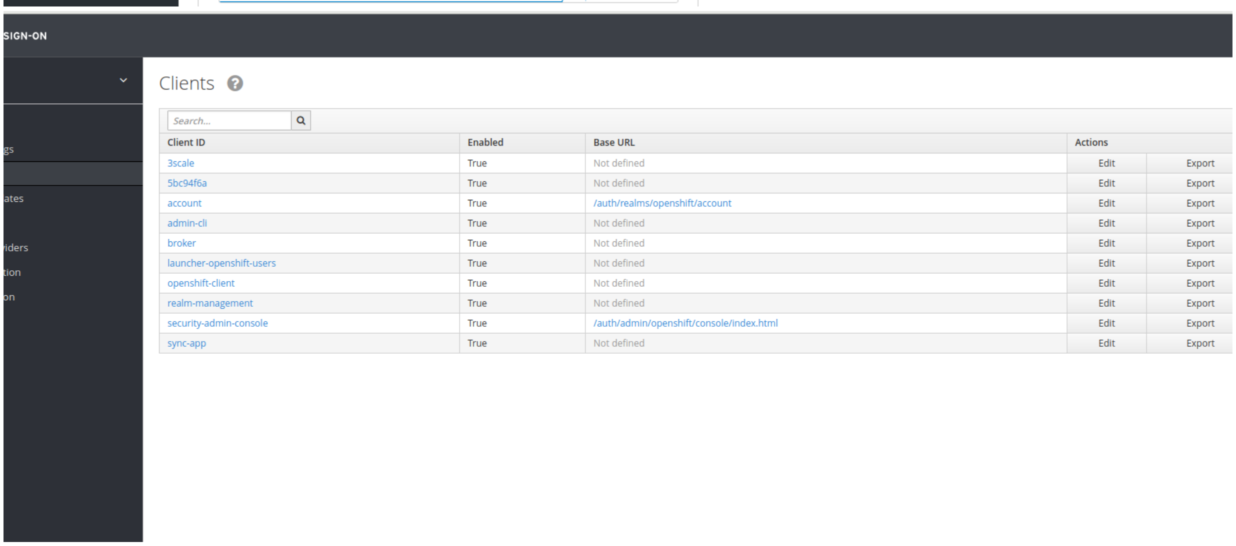


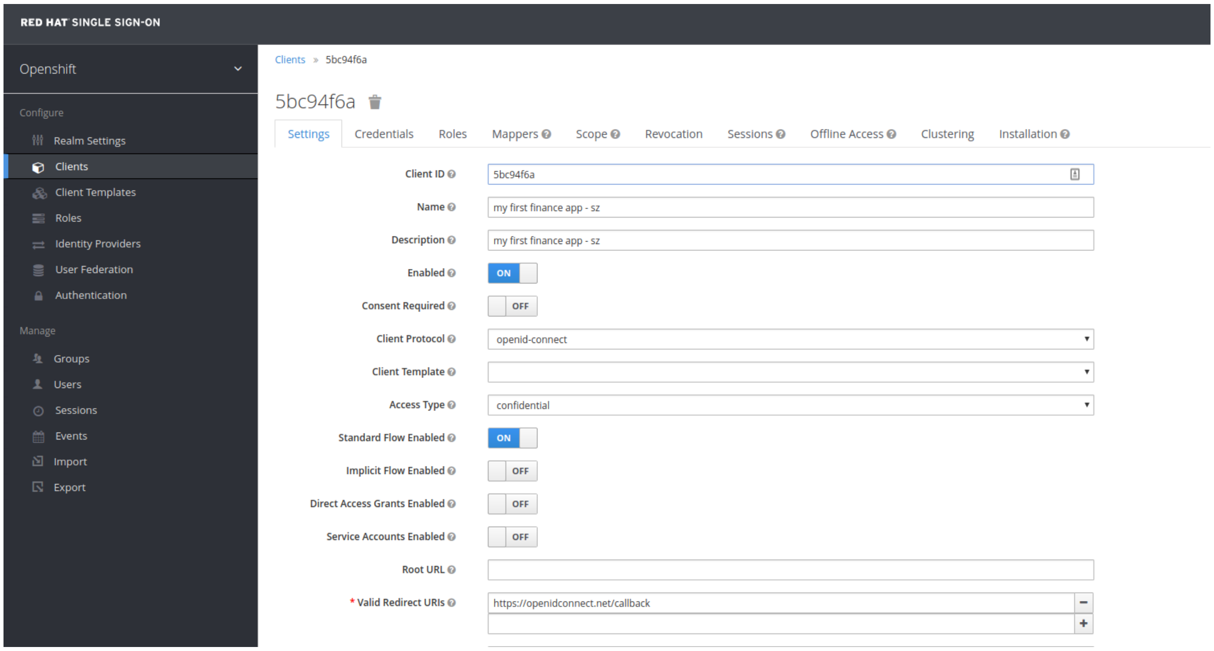
We can see the secret of their application is absent as is the redirect URL. We are going to generate the first and add as redirect url the following https://openidconnect.net/callback (we are going to explain why in a moment).



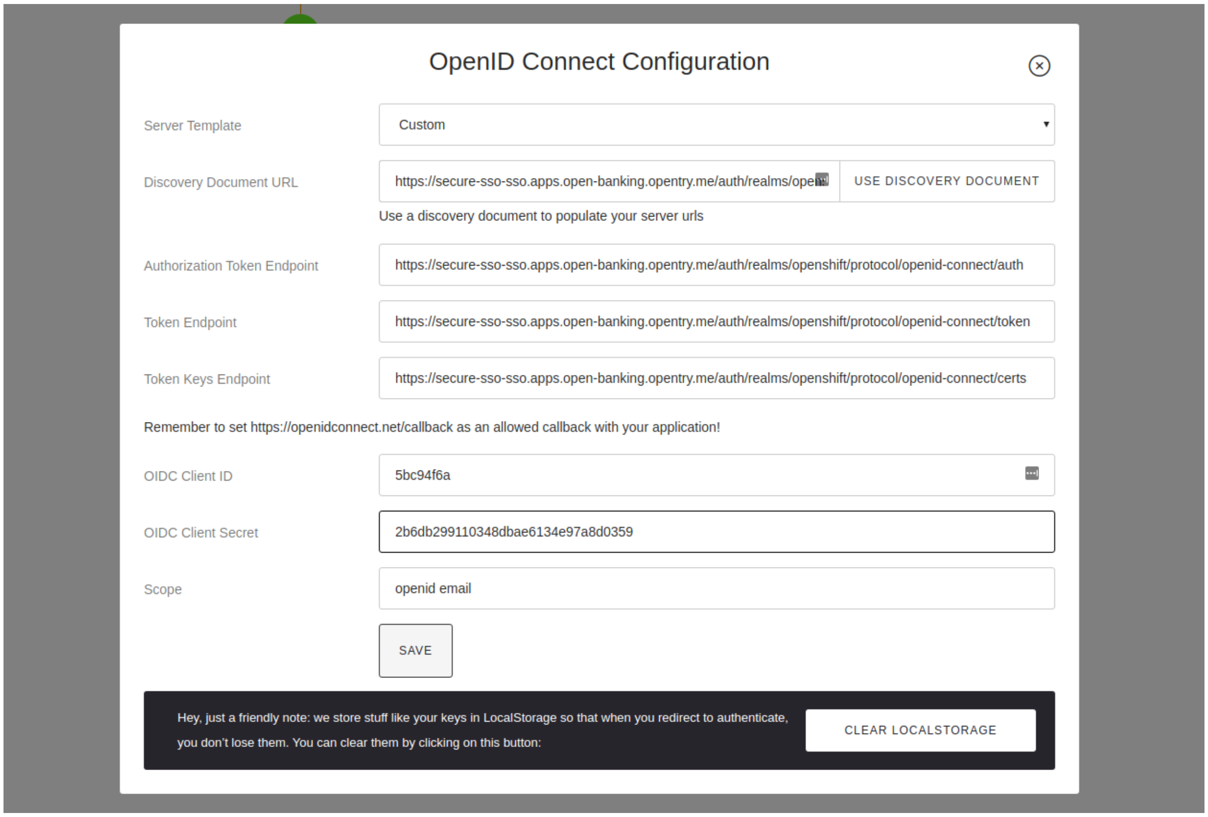
Let's make sure that the application is now aligned in terms of credentials both in 3scale and RH SSO.


Overview
Analytics
Applications
Listing
Application Plans
ActiveDocs
Integration
Application 'Test' | Analytics
Test Edit
Account Developer
Description Test
Service API
State Live Suspend
API Credentials
Client ID 52df2946
Client Secret ea2494e0efbfdde7b4ae2d77062a60f3
Regenerate
Redirect URL https://openidconnect.net/callback Edit
Usage in last 30 Days
Hits
1 hit
Application Plan: Basic
Convert to a Custom Plan
Features
Unlimited Greetings
24/7 support
Unlimited calls
Change Plan
Change Plan

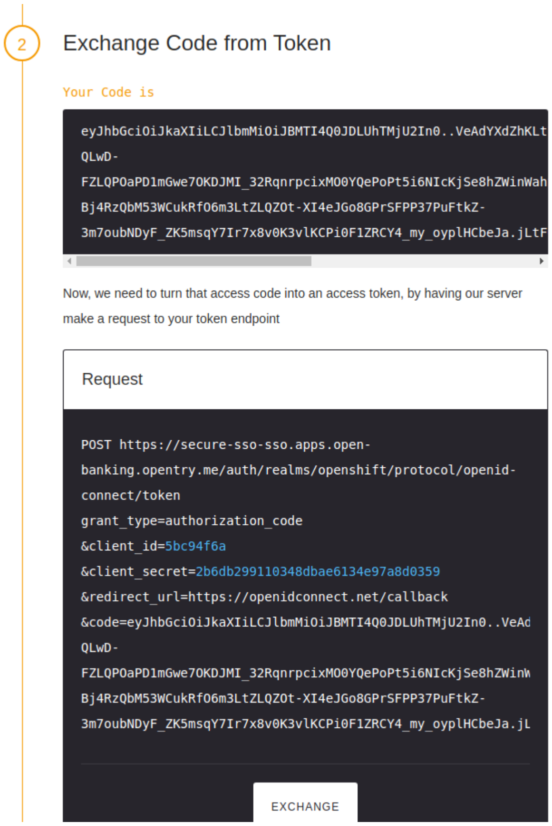

SIGN-ON
gs
ates
iders
tion
on
Clients
Search...
Client ID
Enabled
Base URL
Actions
3scale True Not defined Edit Export
5bc94f6a True Not defined Edit Export
account True /auth/realms/openshift/account Edit Export
admin-cli True Not defined Edit Export
broker True Not defined Edit Export
launcher-openshift-users True Not defined Edit Export
openshift-client True Not defined Edit Export
realm-management True Not defined Edit Export
security-admin-console True /auth/admin/openshift/console/index.html Edit Export
sync-app True Not defined Edit Export



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. **Configuration**
Let's change the server template to custom and input in the discovery URL the one we opened before in our RH SSO realm
https://sso-sso.apps.open-banking-2a38.openshiftworkshop.com/auth/realms/<userX>/.well-known/openid-configuration
And click on USE DISCOVERY DOCUMENT
We are going to use the client id and secret as from the application created in the 3scale developer portal / 3scale admin portal or RH SSO since they are all the same.
And lastly as scope we are going to add **openid** and **email**. **SAVE**



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.



Hit **Exchange**

Request

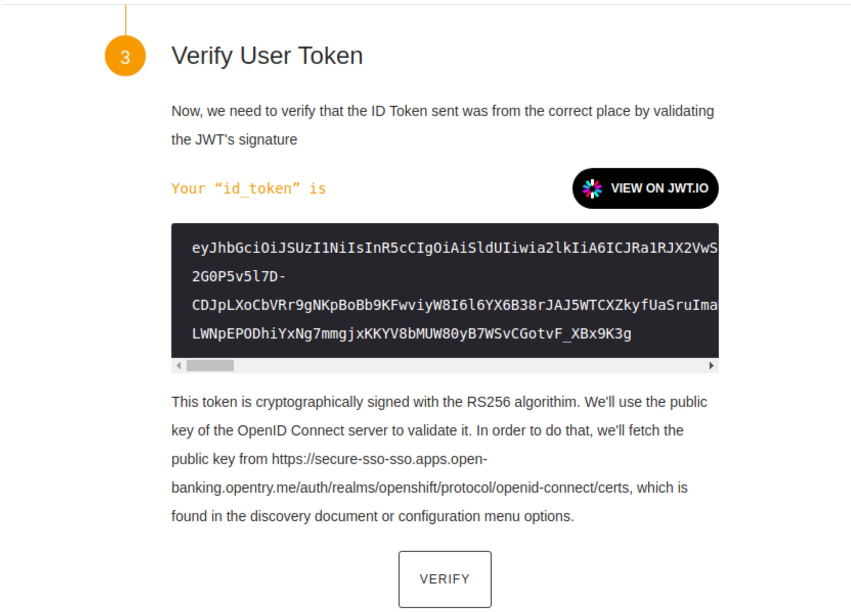
```
POST https://secure-sso-sso.apps.open-
banking.opentry.me/auth/realms/openshift/protocol/openid-
connect/token
grant_type=authorization_code
&client_id=5bc94f6a
&client_secret=2b6db299110348dbae6134e97a8d0359
&redirect_url=https://openidconnect.net/callback
&code=eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..VeAd
QLwD-
FZLQPOaPD1mGwe7OKDJMI_32RqnrpcixMO0YQePoPt5i6NIcKjSe8hZWinW
Bj4RzQbM53WCukRfO6m3LtZLQZOt-XI4eJGo8GPrSFPP37PuFtkZ-
3m7oubNDyF_ZK5msqY7Ir7x8v0K3vlKCPi0F1ZRCY4_my_oyplHCbeJa.jL

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": "fa149b8b-d4e9-49be-95b0-9613ff0a55bd",
  "scope": ""
}
```

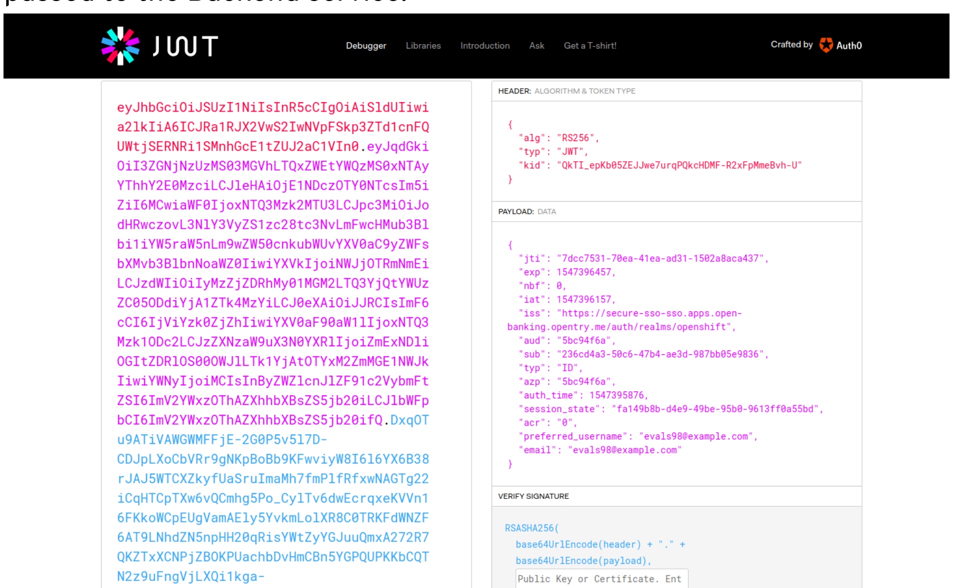
NEXT

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”.

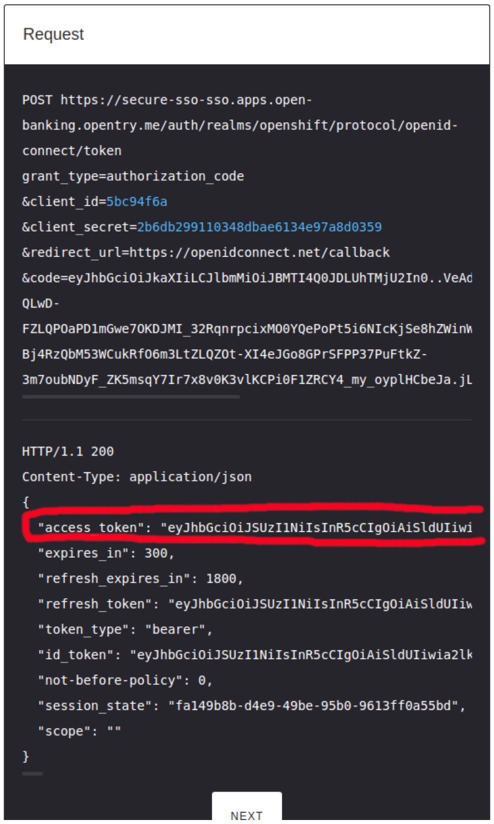
We can hit **NEXT** and id_token will also be shown, which contains more user related details.



We can decode the information on the website JWT.io and found our user details once again as passed to the Backend service.



Let's now go back to https://openidconnect.net/ website and copy the "access_token" value in the step 2 (the long string).

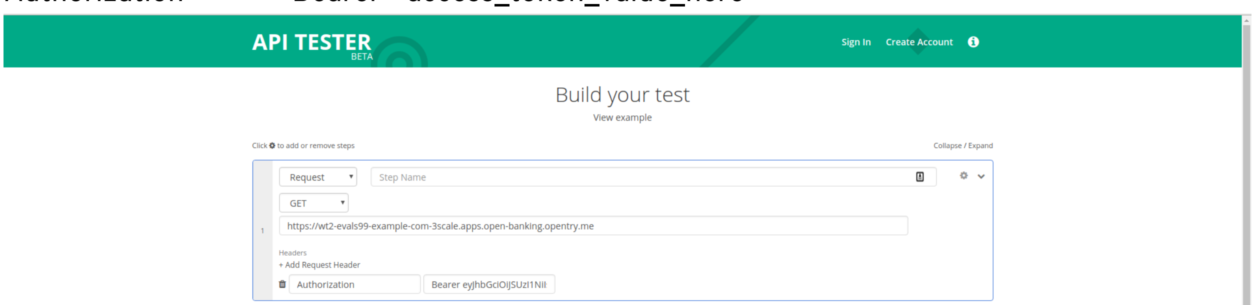


It should look something like this:

eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRa1RJX2VwS2IwNVpFSkp3ZTd1cnFQUWtjSERNRi1SMnhGcE1tZUJ2aC1VIn0.eyJqdGkiOiIyYzJmZjQ5ZS01MDY4LTQ0MjQtYTRiNS05MWU3OTk3MTM0YTMiLCJleHAiOjE1NDczOTc1NTIsIm5iZiI6MCwiaWF0IjoxNTQ3Mzk2NjUyLCJpc3MiOiJodHRwczovL3NlY3VyZS1zc28tc3NvLmFwcHMub3Blbi1iYW5raW5nLm9wZW50cnkubWUvYXV0aC9yZWFsbXMvb3BlbnNoaWZ0IiwiYXVkIjoiNWJjOTRmNmEiLCJzdWIiOiIyMzZjZDRhMy01MGM2LTQ3YjQtYWUzZC05ODdiYjA1ZTk4MzYiLCJ0eXAiOiJCZWFyZXIiLCJhenAiOiI1YmM5NGY2YSIsImF1dGhfdGltZSI6MTU0NzM5NTg3Niwic2Vzc2lvbl9zdGF0ZSI6ImZhMTQ5YjhiLWQ0ZTktNDliZS05NWIwLTk2MTNmZjBhNTViZCIsImFjciI6IjAiLCJhbGxvd2VkLW9yaWdpbnMiOltdLCJyZWFsbV9hY2Nlc3MiOnsicm9sZXMiOlsidW1hX2F1dGhvcml6YXRpb24iXX0sInJlc291cmNlX2FjY2Vzcyl6eyJhY2NvdW50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2NvdW50IiwibWFuYWdlLWFjY291bnQtbGlua3MiLCJ2aWV3LXByb2ZpbGUiXX19LCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJldmFsczk4QGV4YW1wbGUuY29tIiwiZW1haWwiOiJldmFsczk4QGV4YW1wbGUuY29tIn0.O7y6GDFq5CajAT0DkywEuQqEuD5H7_YMqrVC4AMPthZ-m_xZ_DAPBEqj3mmzp1o1JOo0_4pMxNgKpyyqCQiFY79GRS5IJE6aVrZK53rQkud5dlaZAE1-ryiD8CtP_MrQtsTS7bVKbaFyCXNyFfxy3c-TER8GnGG90OIYPXpy5M954slcp4CWXxA7ZwVEuQNRRs5w2G2TCjrFyQjCzslNFwDRtADjbMiY7kq1cwRB5qM9ipdEEligDnH8dietiOZgY24sK10vtowjz_CHuWr5W3474dAZVFC7utwStl_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 and add this as an Authorization header. The format is

Authorization Bearer <access_token_value_here>



Let’s hit **Test**

Request

Request Headers

```
GET /open-data/banks HTTP/1.1
Host: wt2-evals99-example-com-3scale.apps.open-banking.opentry.me
Accept: */*
User-Agent: Mozilla/5.0 (compatible; Rigor/1.0.0; http://rigor.com)
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJRa1RJX2VwS2IwNVpFSkp3ZTd1cnFQUWtjSERNRi1SMnhGcE1tZUJ2aC1VIn0.eyJqdGk
```

Response

Response Headers

```
HTTP/1.1 200 OK
Server: openresty/1.13.6.1
Date: Sun, 13 Jan 2019 16:26:57 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 5070
Set-Cookie: JSESSIONID=hq0uy865vua959siq06hh3ny;Path=/
Expires: Sun, 13 Jan 2019 16:26:57 GMT
Access-Control-Allow-Origin: *
Cache-Control: no-cache, private, no-store
Correlation-Id: hq0uy865vua959siq06hh3ny
Pragma: no-cache
X-Frame-Options: DENY
Strict-Transport-Security: max-age=31536000; includeSubdomains
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self'; script-src 'self'  'unsafe-inline' 'unsafe-eval'; img-src 'self' https://static.openbankproject.
Referrer-Policy: no-referrer-when-downgrade
Set-Cookie: 4e0f3fdf65272673fd1ac68b37832b37=020029bc3fa9aa2bdf56083d95b5e541; path=/; HttpOnly; Secure
Set-Cookie: 0cee625c053b39cab8739dead9521874=fd85f9e6e21ee8a00f6cad673c7c9e5b; path=/; HttpOnly; Secure
```

Response Body


```
{"banks":[{"id":"psd201-bank-x--uk","short_name":"Bank X","full_name":"The Bank of X","logo":"https://static.openbankproject.com/images/sandb
```


Variables

Search variables...

And success!


```
{
  "banks": [
    {
      "id": "psd201-bank-x--uk",
      "short_name": "Bank X",
      "full_name": "The Bank of X",
      "logo": "https://static.openbankproject.com/images/sandbox/bank_x.png",
      "website": "https://www.example.com",
      "bank_routing": {
        "scheme": "OBP",
        "address": "psd201-bank-x--uk"
      }
    },
    {
      "id": "psd201-bank-y--uk",
      "short_name": "Bank Y",
      "full_name": "The Bank of Y",
      "logo": "https://static.openbankproject.com/images/sandbox/bank_y.png",
      "website": "https://www.example.com",
      "bank_routing": {
        "scheme": "OBP",
        "address": "psd201-bank-y--uk"
      }
    },
    {
      "id": "at02-bank-x--01",
      "short_name": "Bank X",
      "full_name": "The Bank of X",
      "logo": "https://static.openbankproject.com/images/sandbox/bank_x.png",
      "website": "https://www.example.com",
      "bank_routing": {
        "scheme": "OBP",
        "address": "at02-bank-x--01"
      }
    },
    {
      "id": "at02-bank-y--01",
      "short_name": "Bank Y",
      "full_name": "The Bank of Y",
      "logo": "https://static.openbankproject.com/images/sandbox/bank_y.png",
      "website": "https://www.example.com",
      "bank_routing": {
        "scheme": "OBP",
        "address": "at02-bank-y--01"
      }
    }
```


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

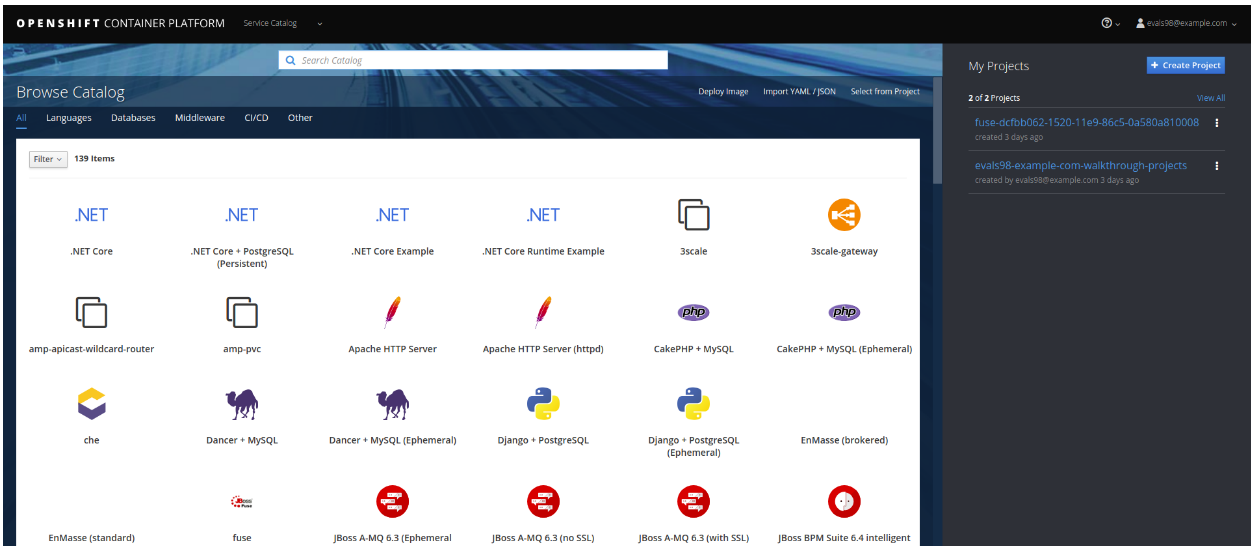
Checkpoint

Improved security to the highest grade possible while using standards.

OpenShift (optional)

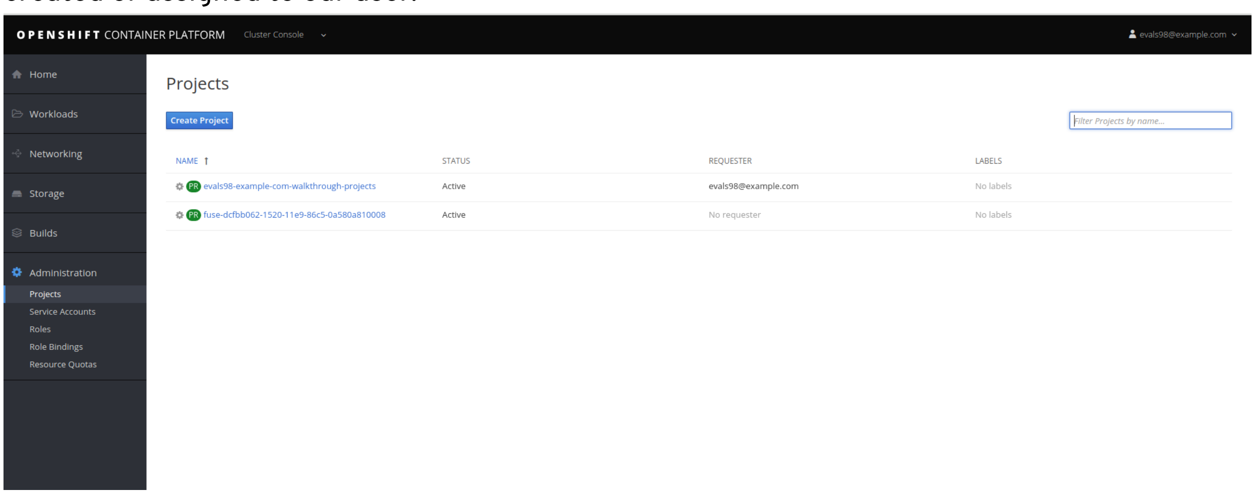
LAB BEGINS

As user you will login into openshift and it already looks evident that the end user has been profiled as developer on OpenShift as he has access only to Objects and Projects he created.

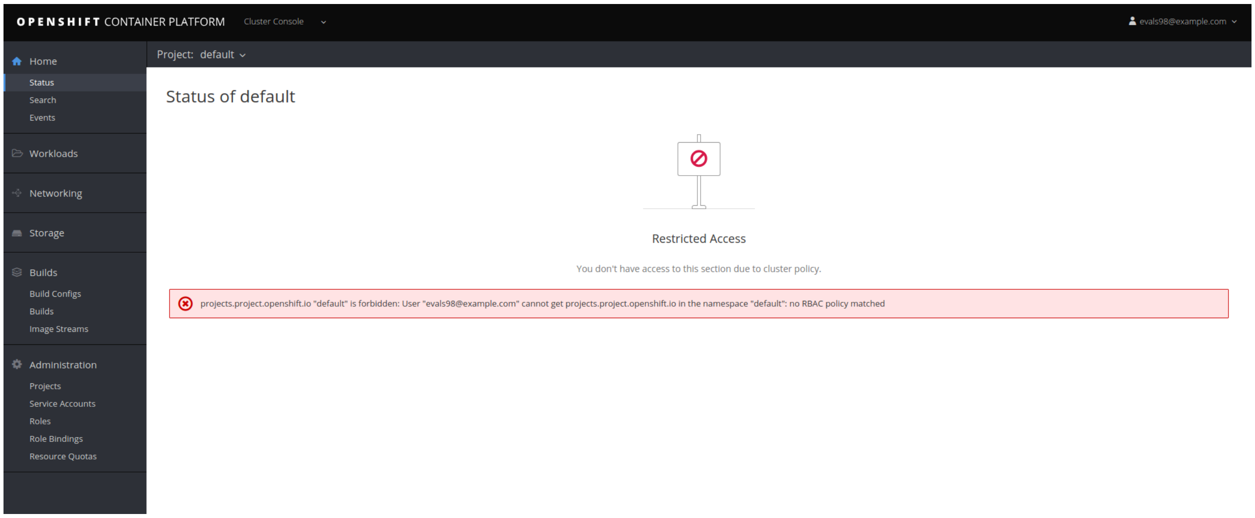


If we click on the fuse project we will be able to access to the Fuse Online installation dedicated to the user. We would also be able to see any integration project running alongside Fuse installation.

If we switch to the **Cluster console**, this will give us some Operations details on the project created or assigned to our user.

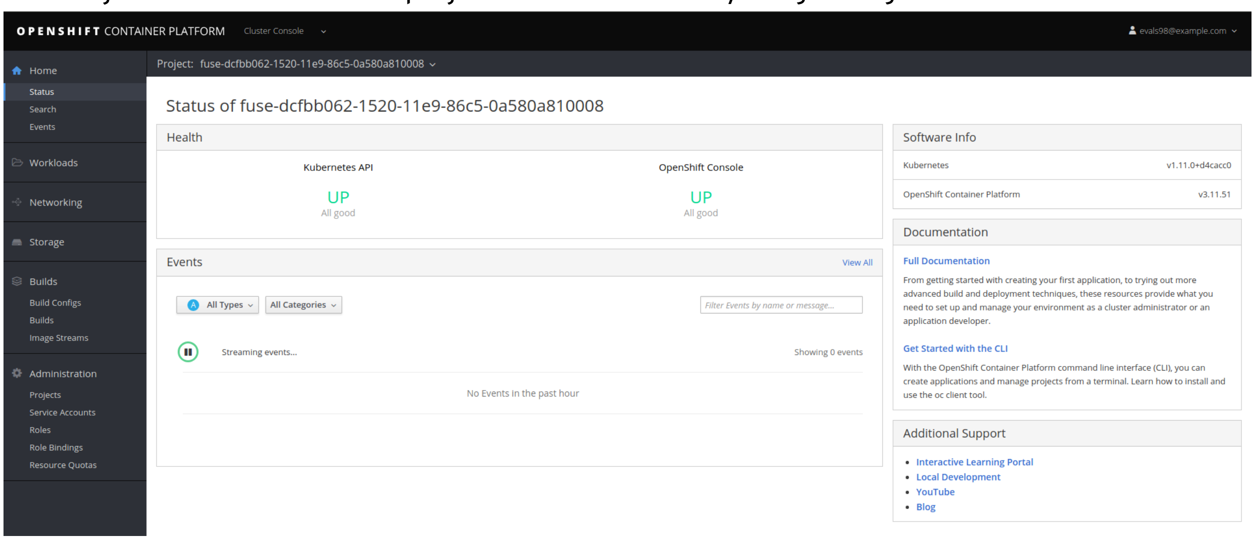


This type of console is also used by Operations administrators to check the health of OpenShift. We can see the RBAC in action if we click on **Home -> Status**

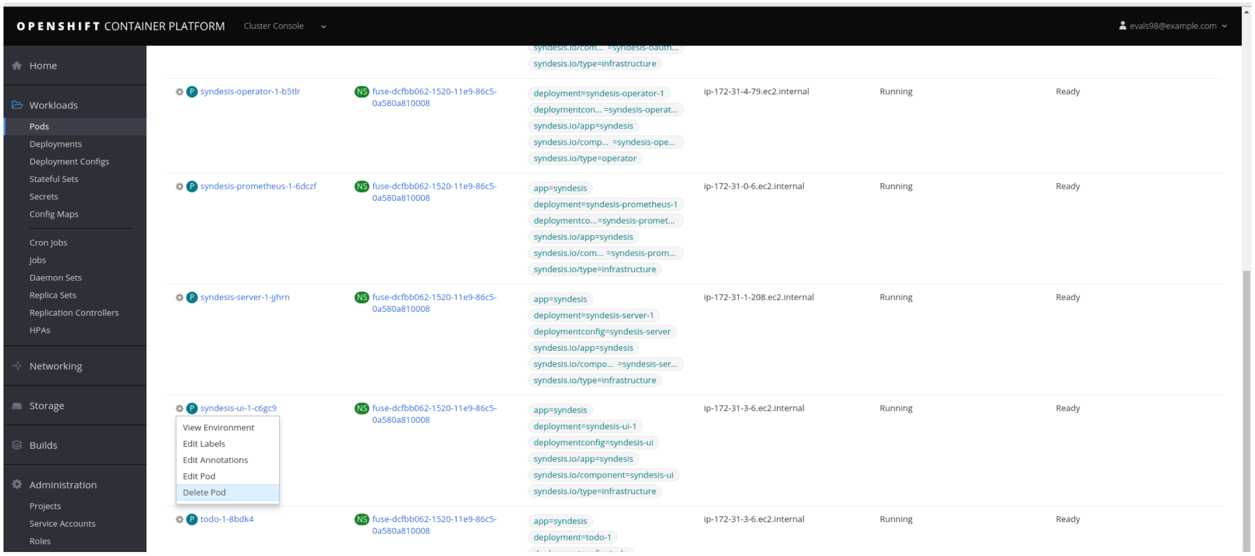


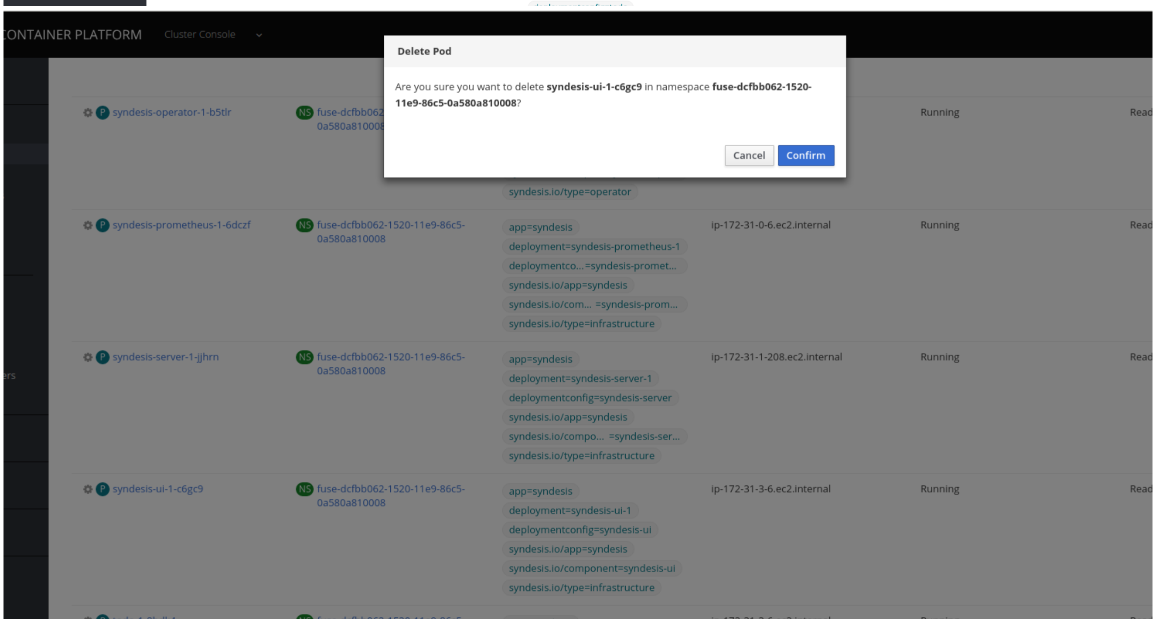
The **Project default** is excluded from the scope of any evals users, since it can contain system components and privileged objects.

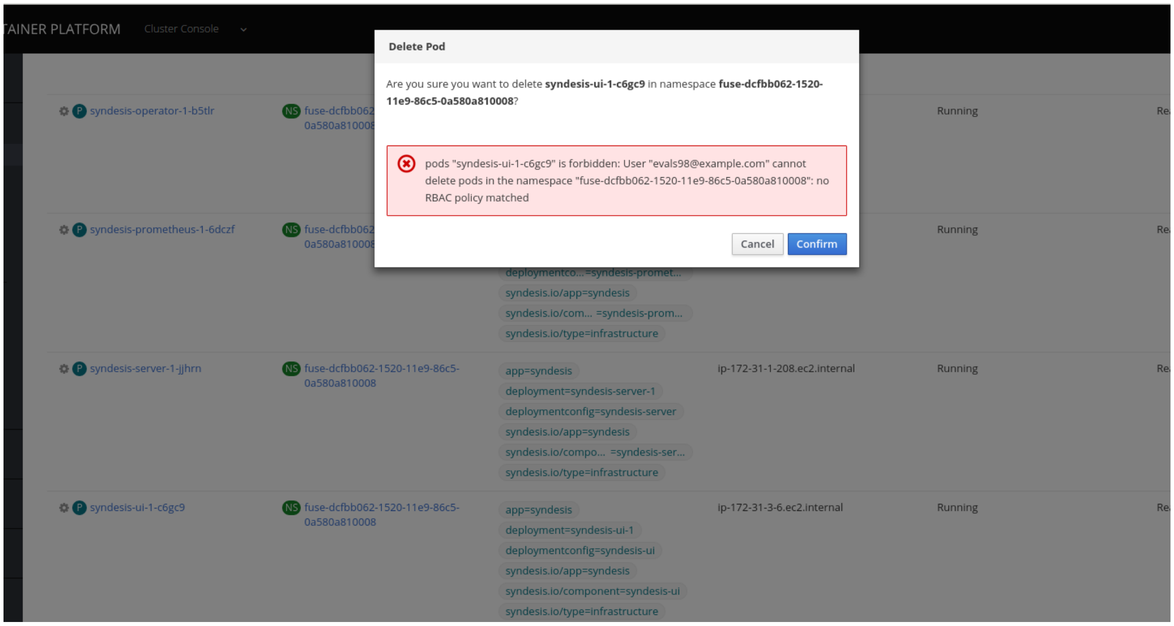
We can just switch to the Fuse project to see if there anything wrong with it in the cluster.



We will now try as bad intentioned user to change some parameters around the installed products.


OPENSHIFT CONTAINER PLATFORM
Cluster Console
evals98@example.com
Home
Workloads
Pods
Deployments
Deployment Configs
Stateful Sets
Secrets
Config Maps
Cron Jobs
Jobs
Daemon Sets
Replica Sets
Replication Controllers
HPAs
Networking
Storage
Builds
Administration
Projects
Service Accounts
Roles
syndesis-operator-1-b5tlr
fuse-dcfbb062-1520-11e9-86c5-0a580a810008
deployment=syndesis-operator-1
syndesis.io/app=syndesis
syndesis.io/type=operator
ip-172-31-4-79.ec2.internal
Running
Ready
syndesis-prometheus-1-6dczf
app=syndesis
deployment=syndesis-prometheus-1
syndesis.io/type=infrastructure
ip-172-31-0-6.ec2.internal
syndesis-server-1-jjhrn
deployment=syndesis-server-1
deploymentconfig=syndesis-server
ip-172-31-1-208.ec2.internal
syndesis-ui-1-c6gc9
View Environment
Edit Labels
Edit Annotations
Edit Pod
Delete Pod
deployment=syndesis-ui-1
deploymentconfig=syndesis-ui
syndesis.io/component=syndesis-ui
ip-172-31-3-6.ec2.internal
todo-1-8bdk4
deployment=todo-1

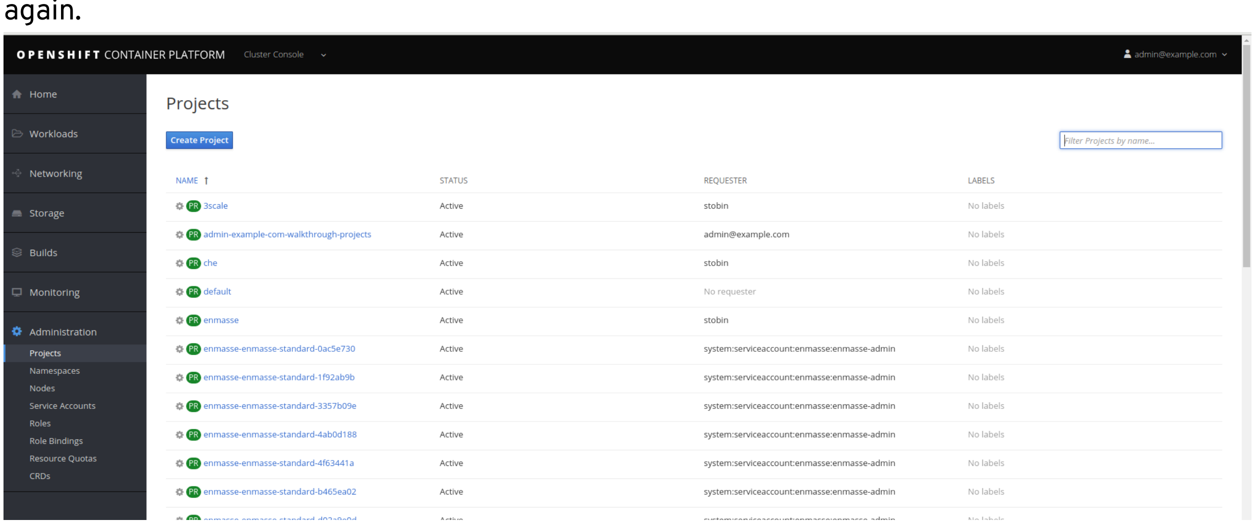

CONTAINER PLATFORM
Cluster Console
Delete Pod
Are you sure you want to delete syndesis-ui-1-c6gc9 in namespace fuse-dcfbb062-1520-11e9-86c5-0a580a810008?
Cancel
Confirm
syndesis-operator-1-b5tlr
syndesis.io/type=operator
syndesis-prometheus-1-6dczf
syndesis-server-1-jjhrn
syndesis-ui-1-c6gc9
Running



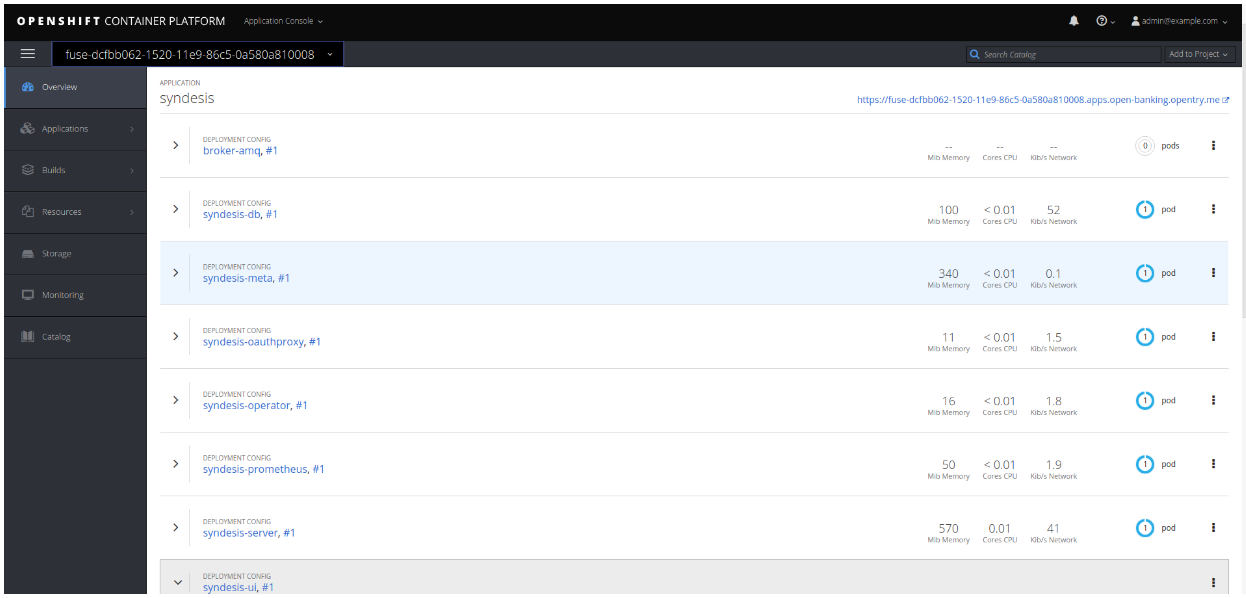
As we can see we tried to kill one of the running components of our integration platform with no success, because of the roles assigned to my user.

DEMO ONLY

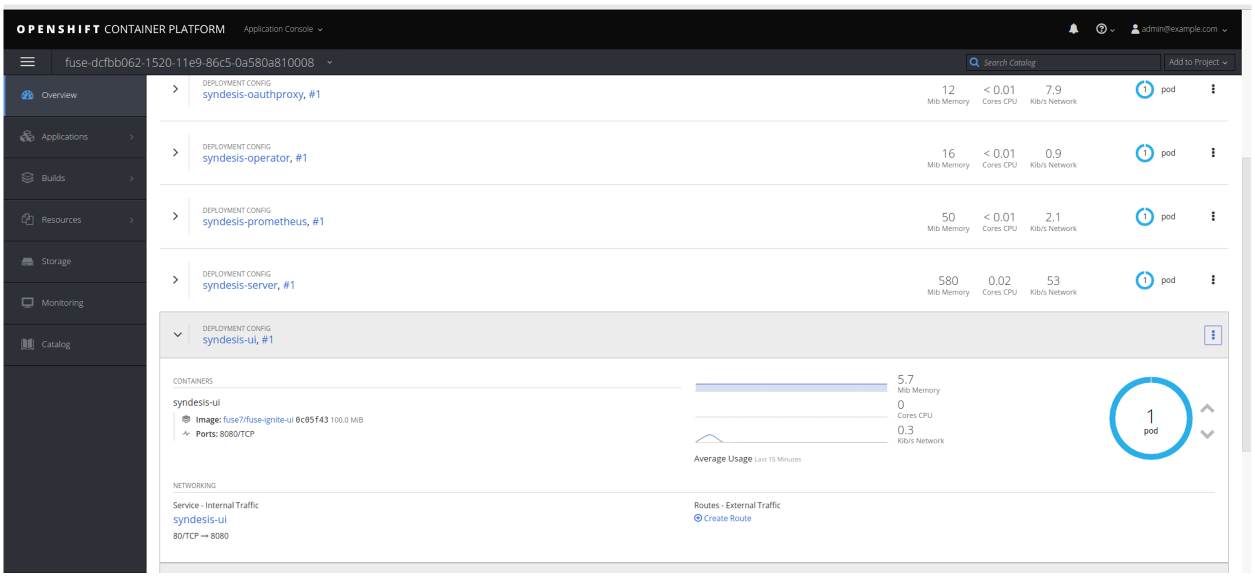
Let's see the magic introduced by OpenShift and login as administrator of the platform once again.

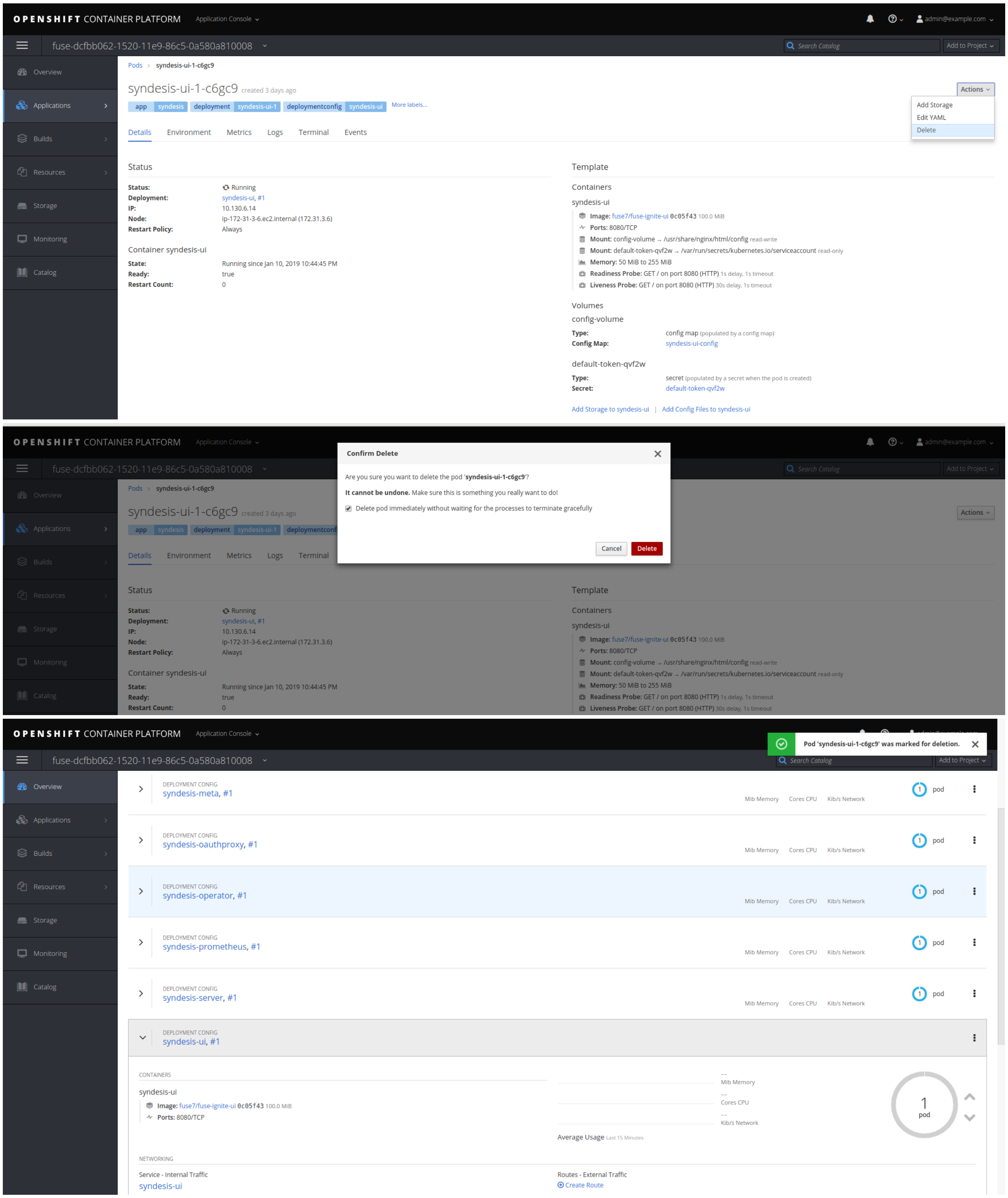


We now have full access to all the platforms from all users. We will open as admin one of the Fuse projects and open one of the components of Fuse Online.

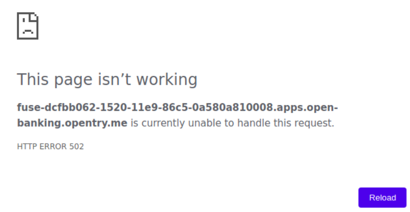


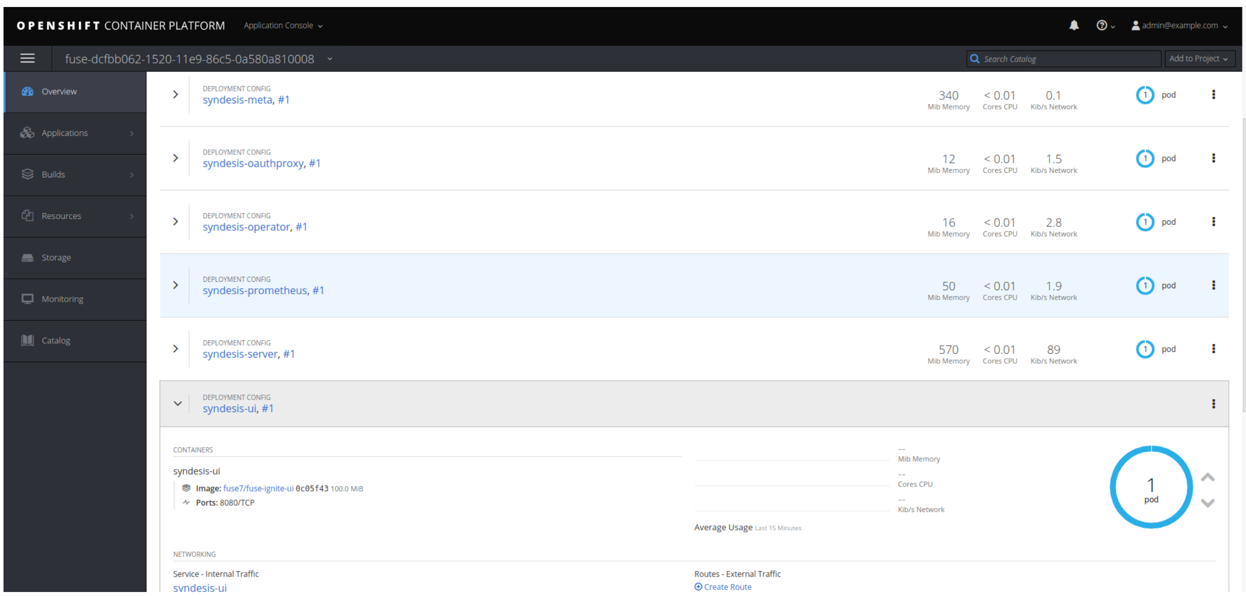
We are going to test the auto healing capabilities of the platform by killing one if its running components, in particular the one providing the UI service.

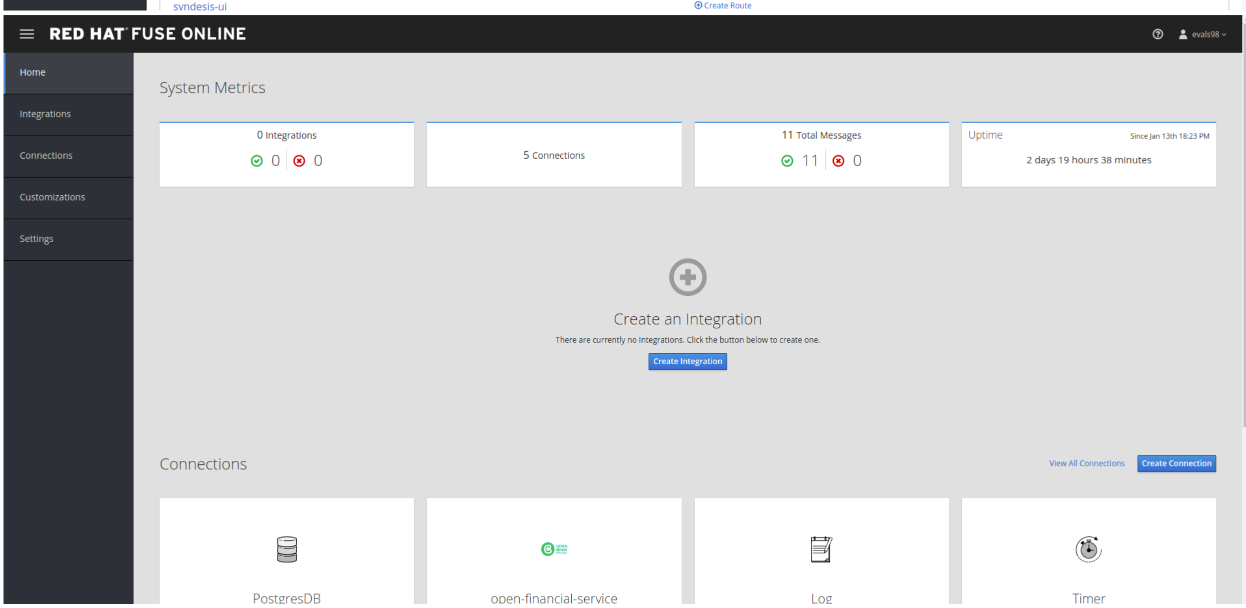




As you can see we just deleted a Pod and we will verify that UI is broken by accessing the interface of Fuse Online







As we can see the component auto-healed thanks to OpenShift features and in a few seconds we have a GUI running once again for the integration platform.

Q&A

Common issues

- openidconnect.net client might have an additional space in the redirect_uri field. That's a client bug, you can fix it by adding an additional redirect URIs in RH SSO with a space preceding the URL: " https://openidconnect.net/callback"
- The installation of RH SSO might have some certificate issues, so might need to use instead a RH SSO deployed somewhere else or using the HTTP only route as suggested in the tutorial