**ESPM on CF**

The objective of this document is to show how a Java application (ESPM) running on SAP Cloud Platform (classic) can be made to run out of the box (without making any major changes) on SAP Cloud Foundry (CF). We have used ESPM (Enterprise Sales and Procurement model) as the sample for this purpose as ESPM is a reference application for java based application development on HCP in [openSap course](https://open.sap.com/courses/hcp2-1?tracking_user=5CTjbb8ra7fgagYGU9hXqj&tracking_type=news&tracking_id=1N95n4kXQd9iRYR0Swh8I) too.

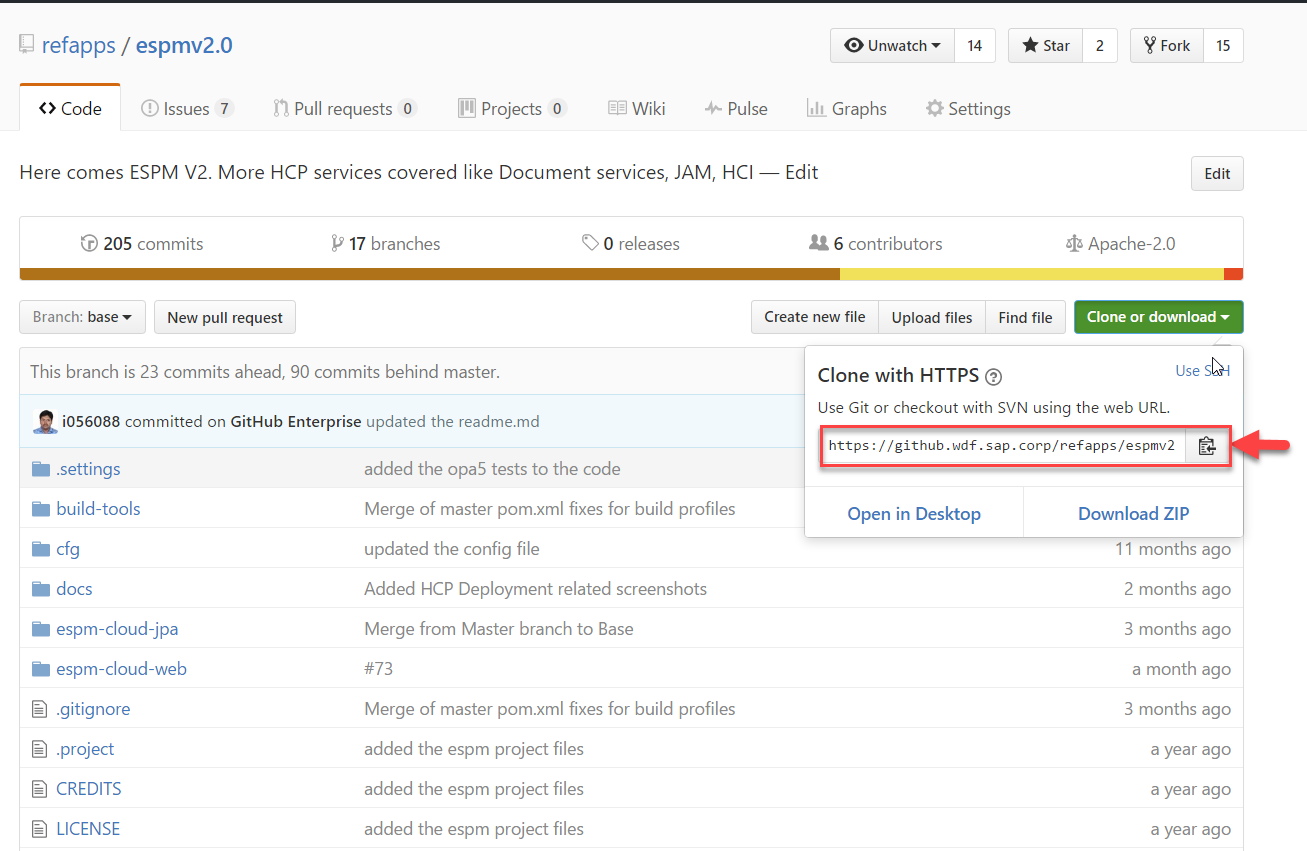
For directly deploying a monolithic application that does not consume any other platform services of classical HCP, call SAP internal sites (eg: <https://github.wdf.sap.corp/> ), the same steps can be followed.

The business scenario for ESPM is that of an eCommerce site that sells electronic products. It involves two personas.

* Customers who can order products and provide reviews on the products.
* Retailer who can then accept the sales orders created against orders created by customers. The Retailer can also update the product stock information. Only a person assigned with the ‘retailer’ role can access the retailer page of ESPM.

Below are the steps that can be followed to make ESPM to run on CF.

1. Go to <https://github.wdf.sap.corp/refapps/espmv2.0> Clone the **Base** branch in your eclipse workspace.



2.Deploying an application on HCP gives us the advantage of connector classes that can easily connect to a HANA database. These are not available on CF. As a workaround for this, we need to add certain dependencies to our Java project to enable us to connect to the database. For connecting to HANA database, the dependency required is “**“ngdbc”**”. So add it to pom.xml of “**espmv2.0 “** and **“espm-cloud -web”**

<dependency>

<groupId>com.sap.db.jdbc</groupId>

<artifactId>ngdbc</artifactId>

<version>2.0.13<version>

</dependency>

We also need to add the dependencies that will help us parse the applications environmental variables which is available in the form of json.

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.8.7</version>

</dependency>

3. Unlike on classical HCP, In Cloud foundry, we have various backend services like postgres, mongodb, HANA etc. We can use any of these services depending on our usecase. In our code, we need to parse the credentials (host, username, password etc) of the required service to connect to them. These credentials are available in the environmental variable called ‘VCAP\_SERVICES’ of the application. In ESPM application, We need to parse the VCAP\_SERVICES to fetch the credential of HANA service on CF. Below is the code snippet for the same. This has to be added to the file -

**espm-cloud-web/src/main/java/com/sap/espm/model/web/JpaEntityManagerFactory.java** as this is where we connect to the database.

**private** **static** JsonNode readCredentialsFromEnvironment() **throws** IOException {

ObjectMapper mapper = **new** ObjectMapper();

JsonNode actualObj = mapper.readTree(System.*getenv*("VCAP\_SERVICES"));

**return** actualObj.get("hana").get(0).get("credentials");

}

The above code fetched the VCAP\_SERVICES from the environment of the application, looks for the ‘hana’ service, fetched the credentials.

**public** **static** **synchronized** EntityManagerFactory getEntityManagerFactory()

**throws** NamingException, SQLException, IOException {

**if** (*entityManagerFactory* == **null**) {

InitialContext ctx = **new** InitialContext();

BasicDataSource ds = **new** BasicDataSource();

JsonNode credentials;

credentials = *readCredentialsFromEnvironment*();

ds.setDriverClassName(credentials.get("driver").asText());

ds.setUrl(credentials.get("url").asText());

ds.setUsername(credentials.get("user").asText());

ds.setPassword(credentials.get("password").asText());

Map<String, Object> properties = **new** HashMap<String, Object>();

properties.put(PersistenceUnitProperties.***NON\_JTA\_DATASOURCE***, ds);

*entityManagerFactory* = Persistence.*createEntityManagerFactory*(

***PERSISTENCE\_UNIT\_NAME***, properties);

Utility.*setEntityManagerFactory*(*entityManagerFactory*);

}

**return** *entityManagerFactory*;

}

The credentials that are required to connect to HANA db are fetched and are added to the hashmap ‘properties’ and then provided to entityMagerFactory.



4. Now build the application.

Go to the root folder of ESPM project. And run the command “**mvn clean install**”

5. Now to deploy the application on CF

Create your own trial space for the SAP internal Cloud Foundry system on AWS landscape if you already don’t have one 🡪<https://account.int.sap.hana.ondemand.com/cockpit> by setting your global Id, password.

6. Download command line interface(CLI) - <https://github.com/cloudfoundry/cli>

It is the official command line client for [Cloud Foundry](https://cloudfoundry.org/).

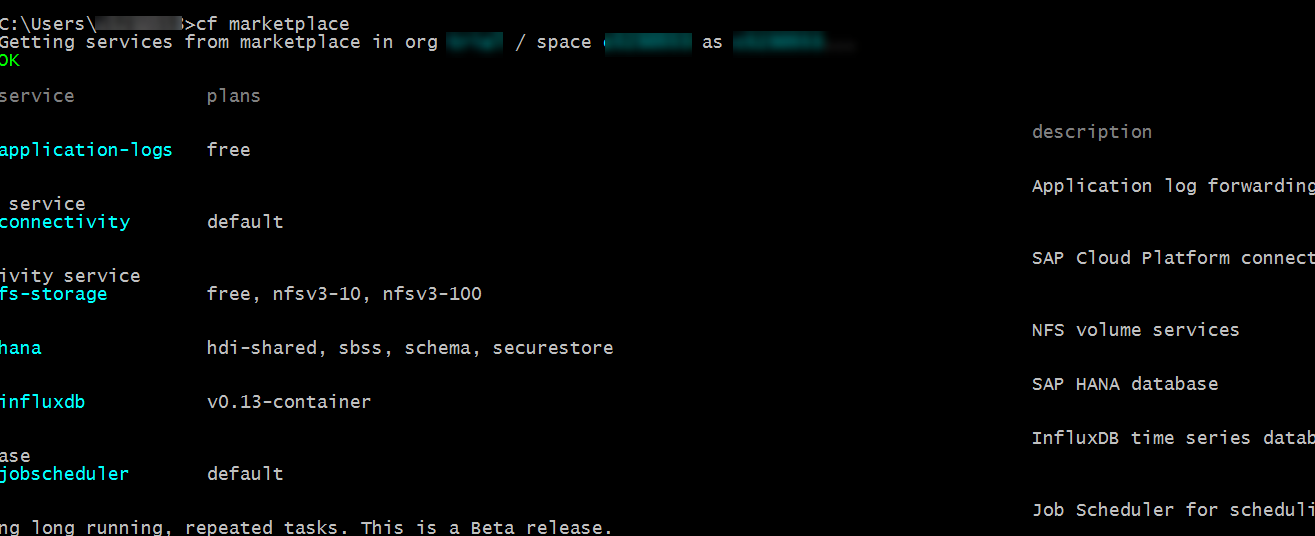
7. Go to command prompt, Set CF API and login.

‘cf api [https://api.cf.sap.hana.ondemand.com/’](https://api.cf.sap.hana.ondemand.com/)

‘cf login’

8. Next step is to create a service instance of HANA. For this, first see the various plans available for the same.

‘CF marketplace’ gives you the list of services, their plans.



9. For ESPM we need to create a HANA service with schema plan.

The schema plan is basically a plain schema, which you need to manage on your own. You might want to use this plan if your application uses an OR Mapper concept and a framework creates the necessary database resources on the fly by itself.

For this, enter the following command:

**cf create-service hana schema <service instance name>**

service instance can be any name of your choice

10. By default, the cf push command deploys an application using a manifest.yml file in the current working directory. This file contains all the information about your application (like application name, host, path of your war file, services).

Create a file **manifest.yml** inside espm-cloud-web. Inside the manifest.yml file give the HANA instance name that you have created.

The host should be unique across the cf landscape. You can append it with your user id for example.

---

applications:

- name: espm

host: <I number>-espm

memory: <memory you want to allocate for your application>

buildpack: java\_buildpack

path: <path of your war file>

services:

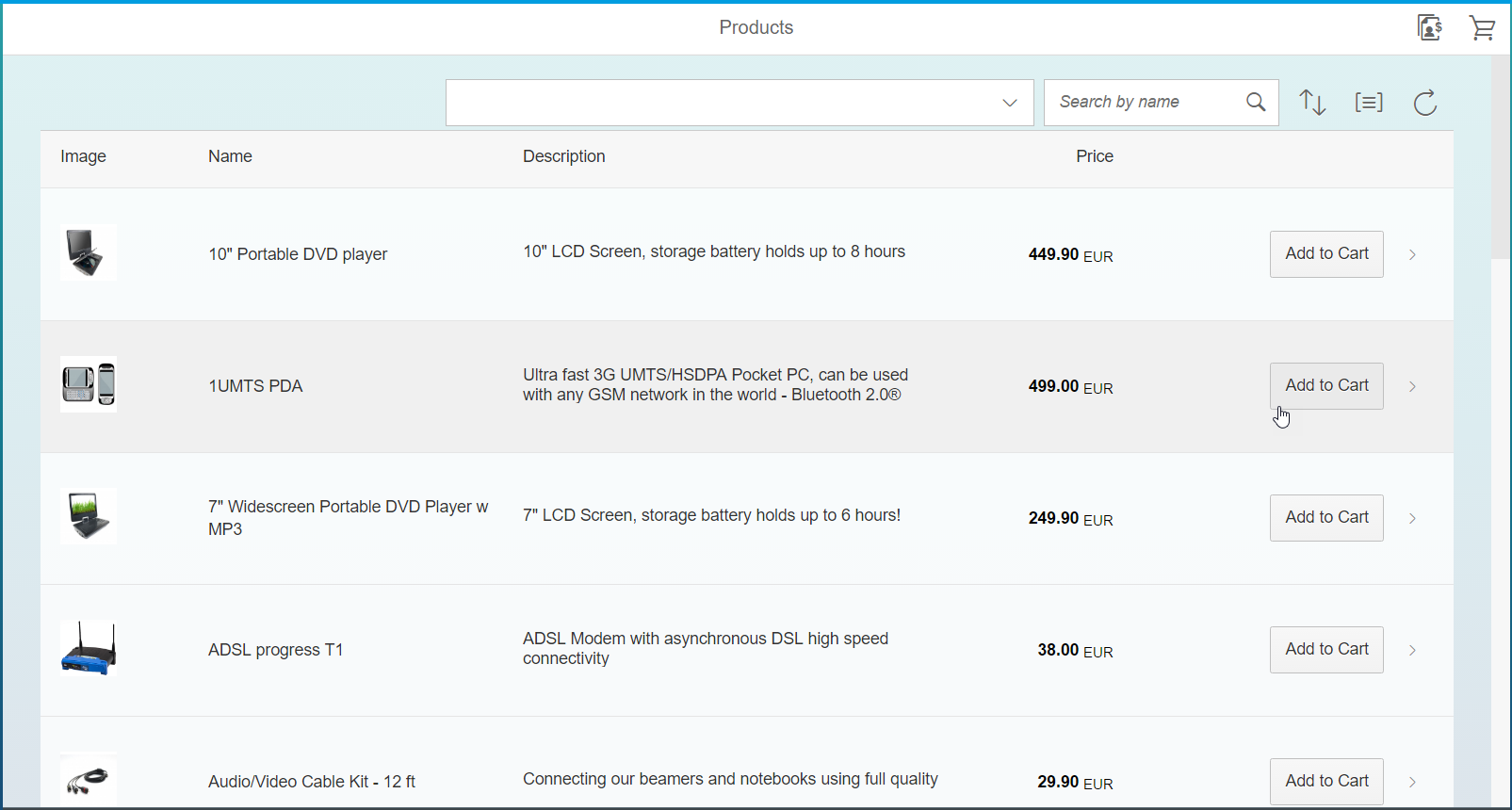
- <Service instance name>

11. Push your application from the folder that contains the manifest file using following command:-

**‘cf push’**

11. Paste the URL of the application on the browser and you will be able to view the **webshop** . 

Below is the view that a customer will be able to see.



**Retailer Configurations-**

For retailer, we need certain roles and authentication to access the page.

1. In CF we have the concept of approuter . All calls to the application running, on CF go through approuter. Thus to add any restriction to these calls, configure the approuter.

Approuter is currently available as a node module. Thus create a “package.json” file in espm-cloud-web ->src - > main - > webapp -> retailer. Paste the below code snippet.

{

"name": *"espm\_approuter"*,

"dependencies": {

"approuter": *"2.4.0"*

},

"scripts": {

"start": *"node node\_modules/approuter/approuter.js"*

}

}

2. Next step is to create xs-app.json which contains all the configuration information used by the application router. Here the route defines how an incoming request path is forwarded to a destination.

Thus create “xs-app.json” in espm-cloud-web ->src - > main - > webapp -> retailer which should look like this.

{

"welcomeFile": *"index.html"*,

"logout": {

"logoutEndpoint": *"/do/logout"*

},

"routes": [{

"source": *"/espm-cloud-web/espm.svc/secure/"*,

"destination": *"java"*,

"scope": *"$XSAPPNAME.Create"*

},

{

"source": *"^/(.\*)"*,

"localDir": *"./"*,

"authenticationType": *"xsuaa"*

}]

}

3. Next step is to create “xs-security.json”. Here we declare scope and Role Templates required to access the application. Place the xs-security.json file in “espm-cloud-web”. Below code snippet shows the xs-security.json for ESPM. xsappname should be a unique name. Enter a name of your choice in the highlighted code.

{

"xsappname" :” *<xs app name> “,*

"scopes" : [

{

"name" : *"$XSAPPNAME.Display"*,

"description" : *"display"*

},

{

"name" : *"uaa.user"*,

"description" : *"uaa.user"*

},

{

"name" : *"$XSAPPNAME.Create"*,

"description" : *"create"*

},

{

"name" : *"$XSAPPNAME.Delete"*,

"description" : *"delete"*

}

],

"role-templates": [

{

"name" : *"retailer"*,

"description" : *"add roles"*,

"scope-references" : [

*"$XSAPPNAME.Create"*, *"uaa.user"*

]

}

]

}

4. xsuaa is a service that is used for user authentication and authorization. We use this service coupled with the Application Router for securing an application. XS-UAA issues authorization codes and JWT tokens.

In Command line interface (CLI) create a service instance of XSUAA service using following command:-

**"cf create-service xsuaa default espm-uaa -c xs-security.json"**

Here, **espm-uaa** - is the name of the service instance.

5. Move to the folder “espm-cloud-web >src - > main - > webapp -> retailer” and perform “**npm install --registry=http://nexus.wdf.sap.corp:8081/nexus/content/groups/build.milestones.npm -proxy=null “** to fetch the AppRouter dependencies.

6. Add the approuter module as a new application in your manifest file, add uaa service instance name in your espm application.

---

applications:

- name: espm

host: <I number>-espm

memory: <memory you want to allocate for your application>

buildpack: java\_buildpack

path: <path of your war file>

services:

- <hana service instance name>

- <uaa service instance name>

- name: <I number>-approuter

host: <I number>-approuter

path: <path to retailer>

memory: <memory you want to allocate for your application>

env:

destinations: >

[

{"name":"java", "url":"https://<I number>-espm.cfapps.sap.hana.ondemand.com", "forwardAuthToken": true }

]

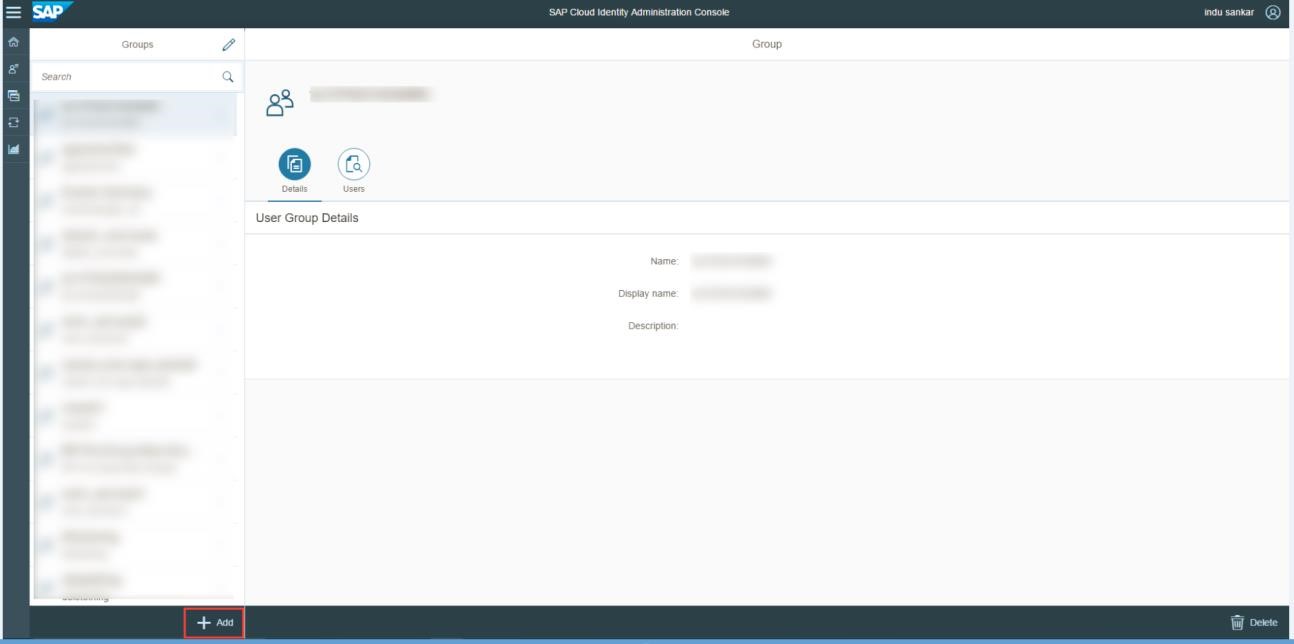
services:

- <uaa service instance name>

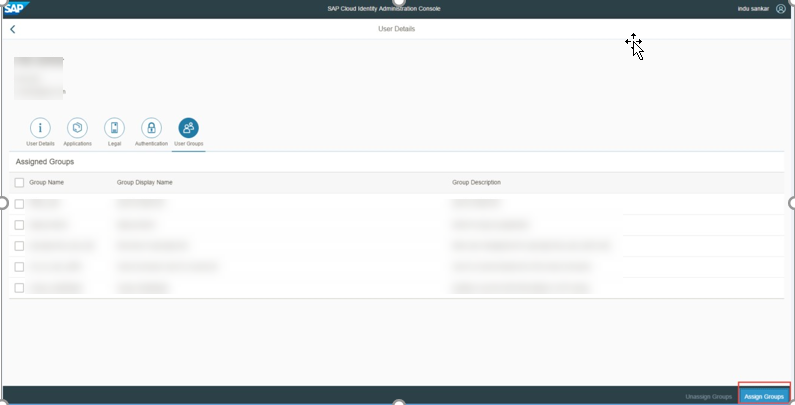
7. Push the application using “**cf push”** command.

8. For your user to be able to access the secured parts of the application (that are restricted to certain roles like admin/privileged etc.) you need to assign your user to the role (admin etc.) and give the scopes (create, edit, delete etc ) for this role in your business application. But this is not a direct step. It involves the below steps:-

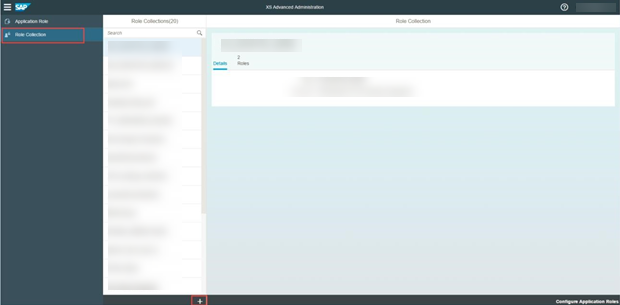
* Create a user group
  + Open <https://xs2security.accounts400.ondemand.com/admin/#/groups>. This requires admin access to xs2security.
  + Click on ‘Add’.



* Enter the name, display name and description for the new group. Save
* Assign the group to your user
  + Open <https://xs2security.accounts400.ondemand.com/admin/#/users>. Find your user. You can search by the email id with which you have registered in XSUAA.
  + Go to User Groups tab and click on Assign Groups.



* Select the group and click on “save”.
* Check for your application in role-builder application
* Open [https://application-role-builder.cfapps.sap.hana.ondemand.com](https://application-role-builder.cfapps.sap.hana.ondemand.com/)
* Once logged into the application, you can search for ‘your application ‘on the left hand side and view the corresponding role-template(s) that you configured via the xs-security.json file . Also, you may also add/modify new roles or templates. Here ‘your application’ is the xsappname that is defined in the xs-security.json file.
* Create a role collection
* Open the role collection tab. Click on ‘+’ button to create a new role collection.



* Add application role to role collection
* Click on ‘Add application Role’. Select your project, role template.
* Assign group to the role collection
* Open the URL <https://saml-idp.cfapps.sap.hana.ondemand.com/index.html>
  + Select useridp from SAML identity provider list.
  + In role collections tab, select the role collection that you created in the above step. Enter the group that you created as the ‘value’.
  + Save changes.
* Now you have admin access to Retailer, you will be able to open the Retailer page.

URL: <app url>/retailer

