CIRC CAP APPLICATION

Language Used:- Node.js

Abstract

A document to describe the steps I used to create the CIRC Application. The Servces added, entities used as well as steps I had used for the deployment of the Application

Gupta, Nikhil

(Scholar@SAP,2020)

nikhil.gupta05@sap.com

+91-7204893465

TABLE OF CONTENTS

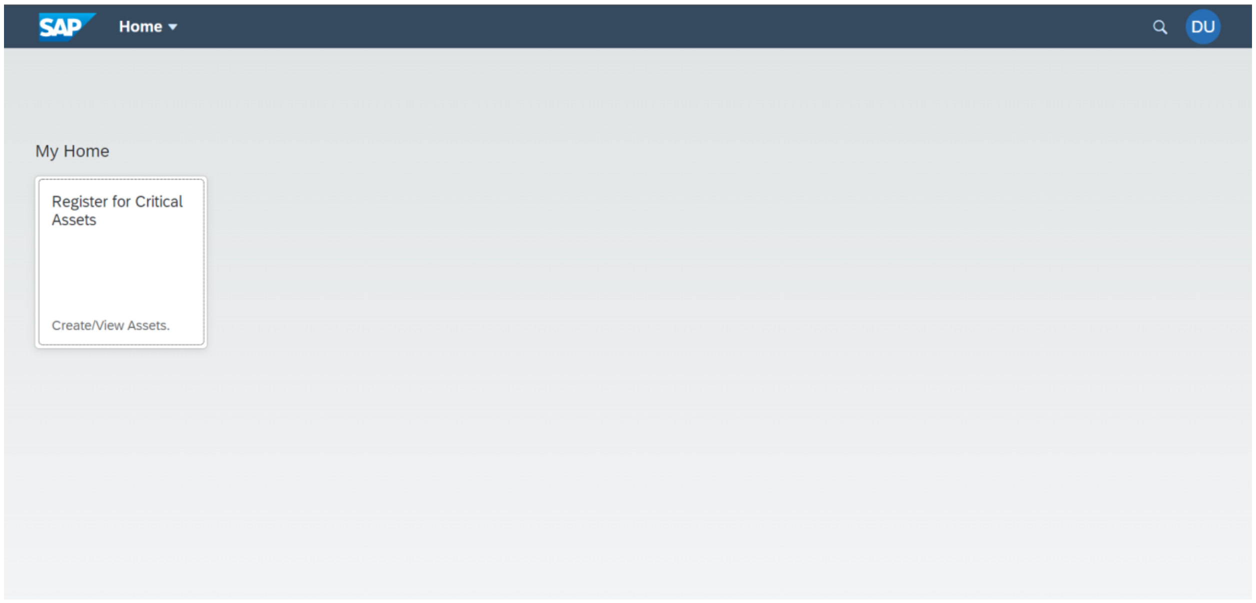
INTRODUCTION

The document will walk through the various steps I have used for achieving the UI design of the application, as well as the databases creation, the entities which I have used, services which have been deployed as well as the various steps which I have used to deploy the CAP application on CF.

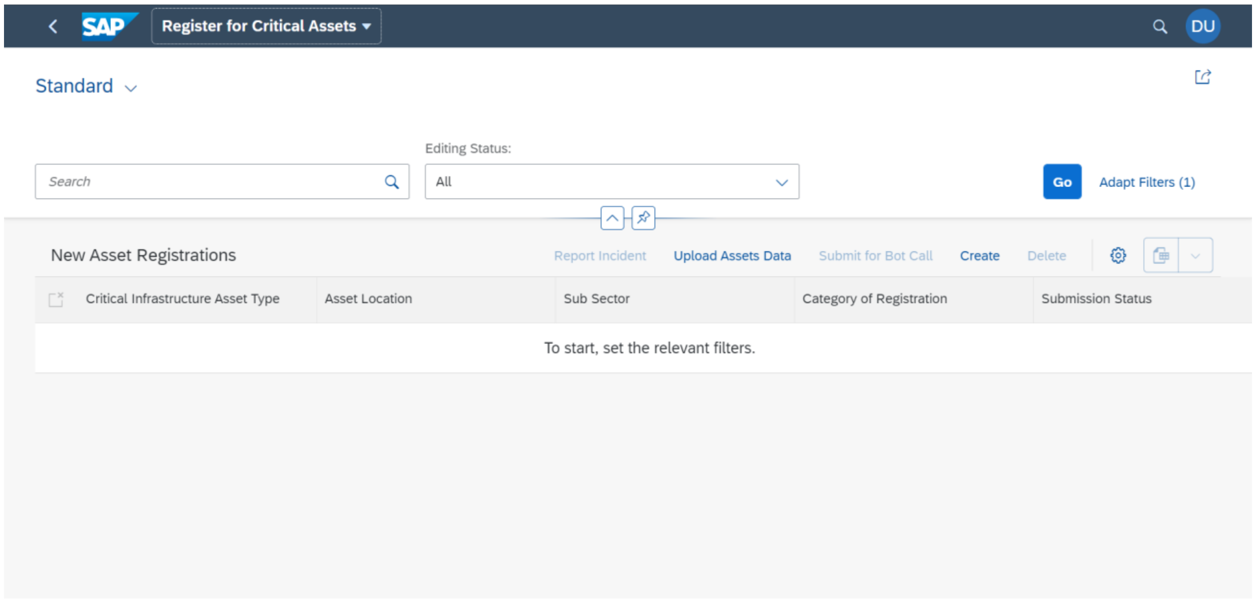
The end application looks like:-

All the pages will be talked in detail later.

1. Cover Page:- (Which consists the Main Fiori Title)

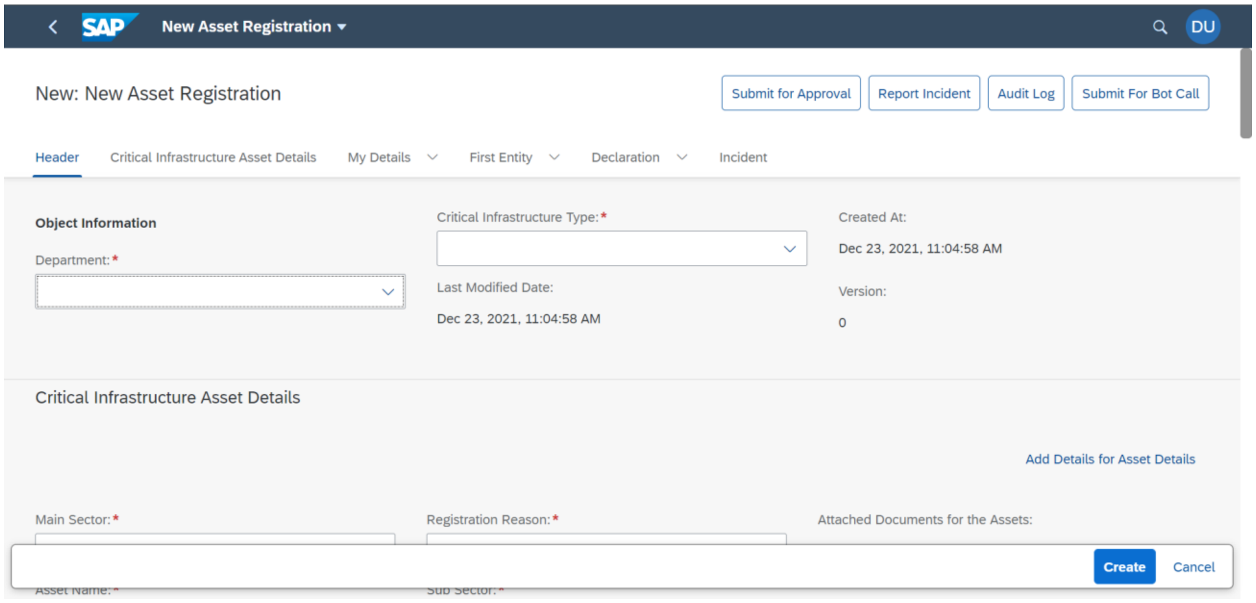


2. The Fiori List Page:-



To start, set the relevant filters.

3. The Fiori Object Page



New: New Asset Registration

Object Information

Department:*

Critical Infrastructure Type:*

Last Modified Date: Dec 23, 2021, 11:04:58 AM

Created At: Dec 23, 2021, 11:04:58 AM

Version: 0

Critical Infrastructure Asset Details

Main Sector:*

Registration Reason:*

Attached Documents for the Assets:

Asset Name:*

Sub Sector:*

ENTITIES USED

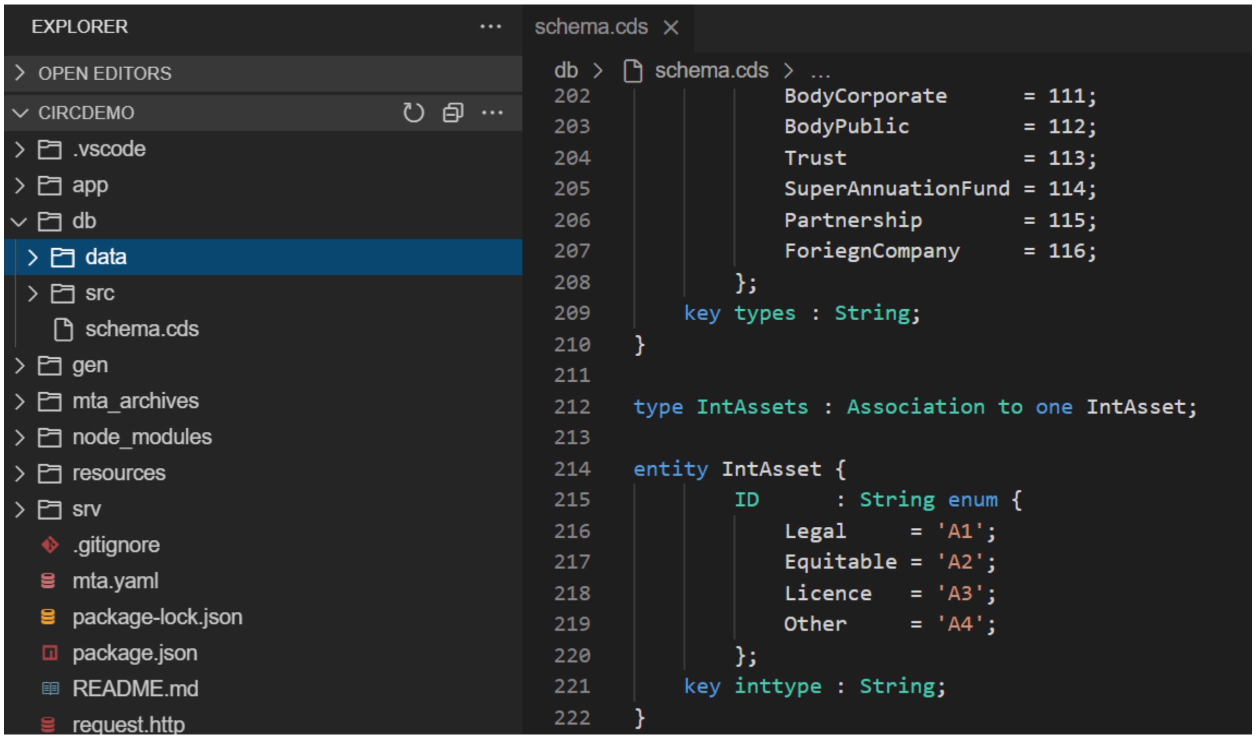
I have used 2 main Entities for the whole design and HANA-Db design. The two entities used are:-

1. NewReg :- For the whole registration page(the object as well as the list page)
2. Incident :- For the Incident which can reported for a particular asset. This is an association to the NewReg entity.

The other entities which have been used, have been used for the purpose of hard-coded data types. Some also used for the drop-down options. The Entities used for the drop downs are:-

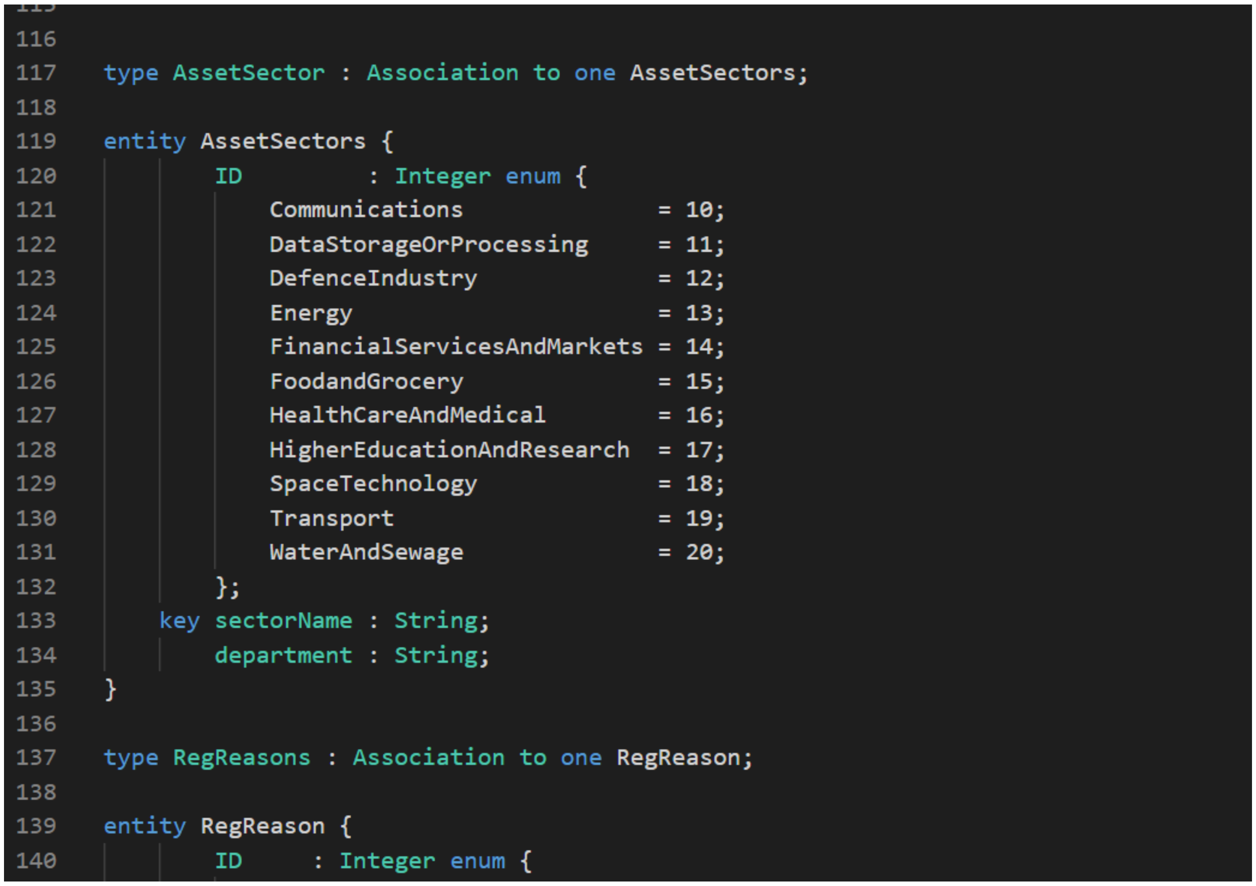
1. AssetSectors → For the Drop-down of Main Sectors
2. RegReason → For the Registration Reason
3. SubSector → For the SubSectors (The codes correspond to the one's associated with the departments declared in AssetSectors)
4. EntType → For the Type of Asset which has been registered.
5. IntAsset → For the Type of Interest in Asset

The following are all defined in the **schema.cds** in the *db* folder.



```
                BodyCorporate      = 111;
                BodyPublic         = 112;
                Trust              = 113;
                SuperAnnuationFund = 114;
                Partnership        = 115;
                ForiegnCompany     = 116;
            };
        key types : String;
    }

    type IntAssets : Association to one IntAsset;

    entity IntAsset {
            ID      : String enum {
                Legal     = 'A1';
                Equitable = 'A2';
                Licence   = 'A3';
                Other     = 'A4';
            };
        key inttype : String;
    }
```

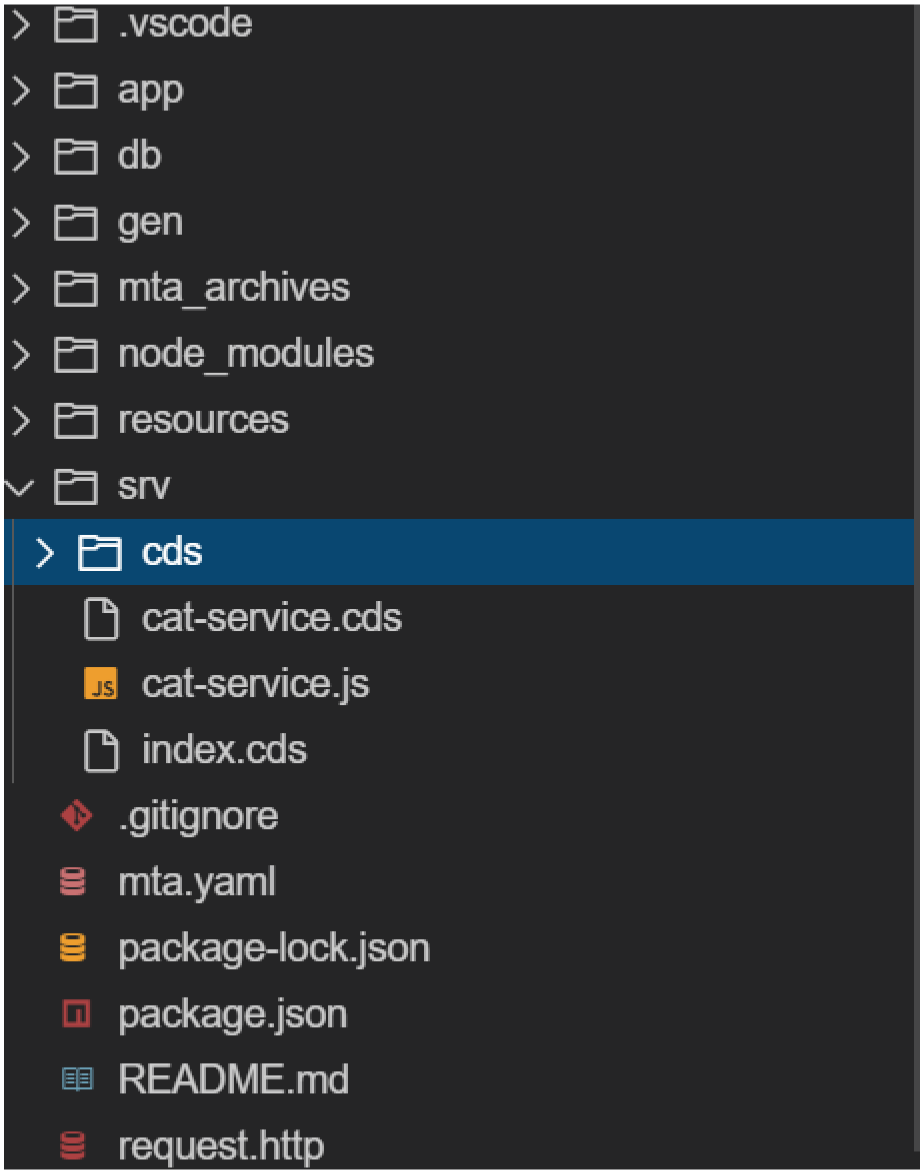
Small snippet of the schema.cds:-



```
type AssetSector : Association to one AssetSectors;

entity AssetSectors {
        ID          : Integer enum {
            Communications                = 10;
            DataStorageOrProcessing       = 11;
            DefenceIndustry               = 12;
            Energy                        = 13;
            FinancialServicesAndMarkets = 14;
            FoodandGrocery                = 15;
            HealthCareAndMedical          = 16;
            HigherEducationAndResearch   = 17;
            SpaceTechnology               = 18;
            Transport                     = 19;
            WaterAndSewage                = 20;
        };
    key sectorName : String;
        department : String;
}

type RegReasons : Association to one RegReason;

entity RegReason {
        ID      : Integer enum {
```

SERVICES USED

To find the services which has been used please refer to the **cat-service.cds** and **cat-service.js** files in the *srv* folder.

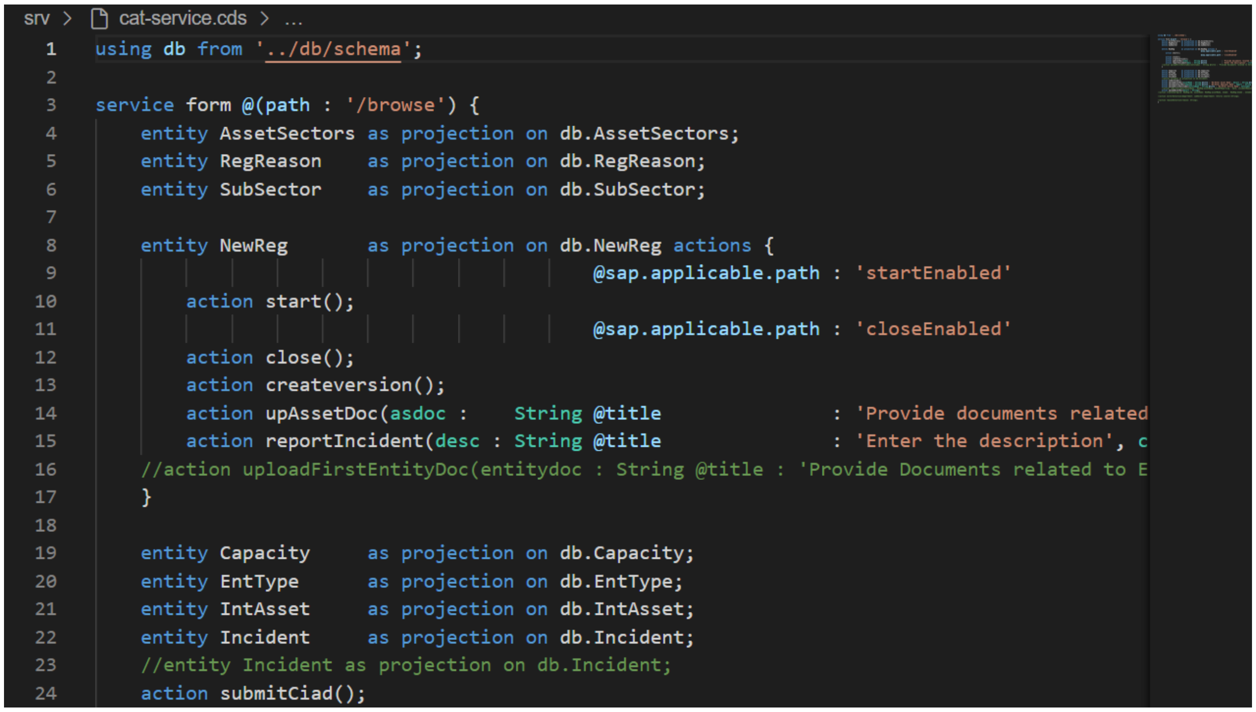


```
> .vscode
> app
> db
> gen
> mta_archives
> node_modules
> resources
v srv
  > cds
    cat-service.cds
    cat-service.js
    index.cds
  .gitignore
  mta.yaml
  package-lock.json
  package.json
  README.md
  request.http
```

As you can see there is a folder called **cds** in the *srv* folder and there are 3 files :- **cat-service.cds**, **cat-service.js** and **index.cds**. The focus for this would be mainly **cat-service.cds** and **cat-service.js**. The cds folder and index.cds will be covered later.

1. cat-service.cds :-The main use of this file is to define the service which is going to be used for running the app as well as various odata services. Please note, that all the entities you define **should** be projected/exposed in this file or else it might give an error while running the app in local as well as on the cloud.

   A small snippet of the file:-



srv > cat-service.cds > ...

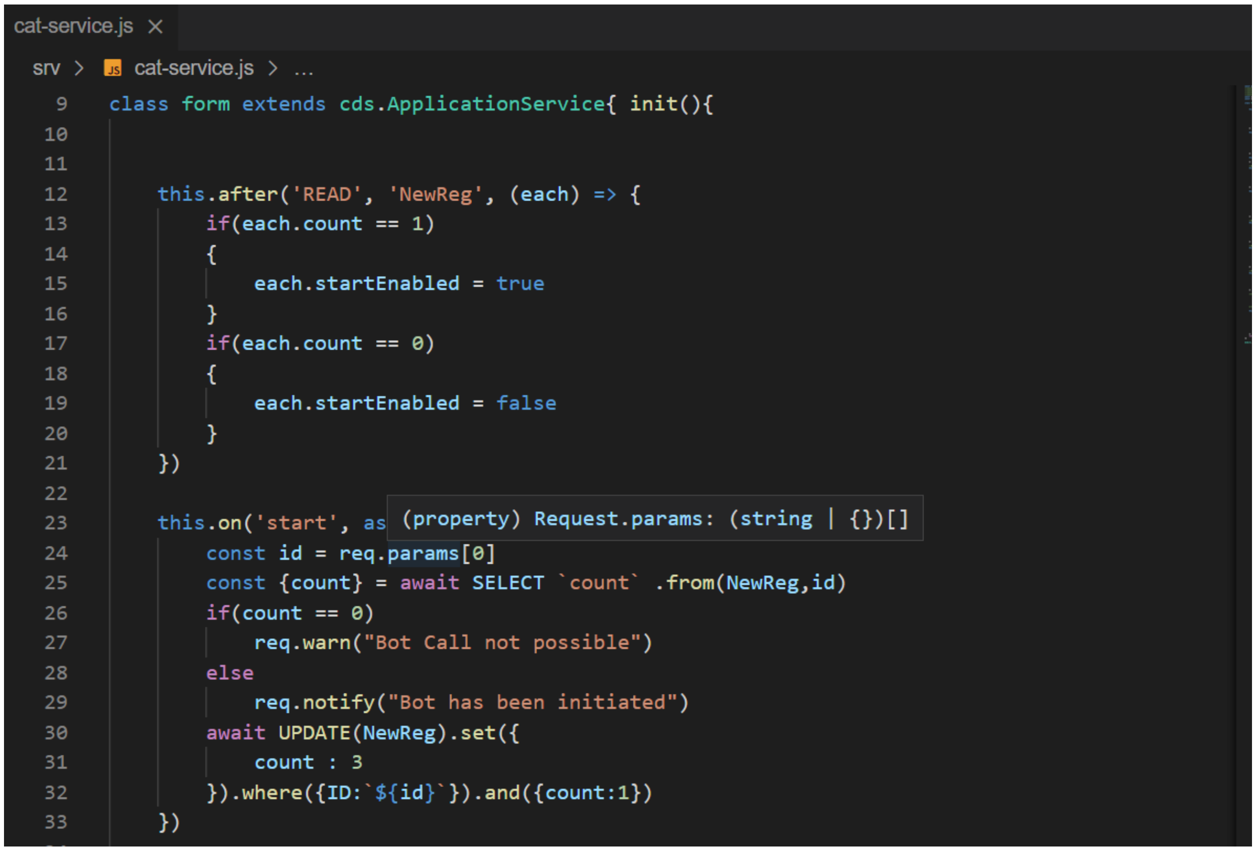
```
using db from '../db/schema';

service form @(path : '/browse') {
    entity AssetSectors as projection on db.AssetSectors;
    entity RegReason    as projection on db.RegReason;
    entity SubSector    as projection on db.SubSector;

    entity NewReg       as projection on db.NewReg actions {
                                            @sap.applicable.path : 'startEnabled'
        action start();
                                            @sap.applicable.path : 'closeEnabled'
        action close();
        action createversion();
        action upAssetDoc(asdoc :    String @title                 : 'Provide documents related
        action reportIncident(desc : String @title                 : 'Enter the description', c
    //action uploadFirstEntityDoc(entitydoc : String @title : 'Provide Documents related to E
    }

    entity Capacity     as projection on db.Capacity;
    entity EntType      as projection on db.EntType;
    entity IntAsset     as projection on db.IntAsset;
    entity Incident     as projection on db.Incident;
    //entity Incident as projection on db.Incident;
    action submitCiad();
```

2. cat-service.js :- This file contains all the custom logic which I have incorporated in the application so far.
   The functionalities I have incorporated so far are:-
   a. Bot call intitation
   b. Submit The form – Auto approval
   c. Upload Documents for – Assets, Contact Details, Entity Proof
   d. Report Incident Button – On the list page (This needs further work, as the very basic format is what has been coded so far)
   e. Versioning – Audit Log

   Please note that this file will also be used for the API-bot call (Future-Scope) The functionalities have 1 or more functions associated to them. The functions will be called in the object page which I have created using annotations.

Do refer to the object page which has been coded to know which function is corresponding to which functionality. I have tried commenting most of the functions.

A small snippet of the file is given below:-



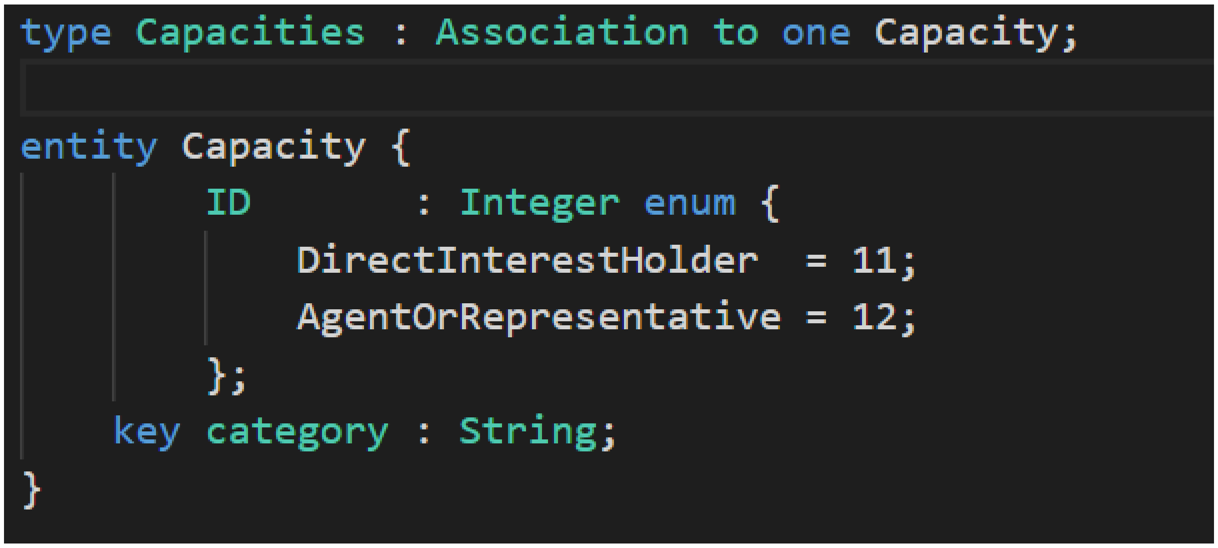
```
class form extends cds.ApplicationService{ init(){


    this.after('READ', 'NewReg', (each) => {
        if(each.count == 1)
        {
            each.startEnabled = true
        }
        if(each.count == 0)
        {
            each.startEnabled = false
        }
    })

    this.on('start', as (property) Request.params: (string | {})[]
        const id = req.params[0]
        const {count} = await SELECT `count` .from(NewReg,id)
        if(count == 0)
            req.warn("Bot Call not possible")
        else
            req.notify("Bot has been initiated")
        await UPDATE(NewReg).set({
            count : 3
        }).where({ID:`${id}`}).and({count:1})
    })
```

THE DROP-DOWNS FUNCTIONALITY USED

The drop-downs which have been incorporated in this app, have been hard coded and there is no pre-defined functionality to do so.

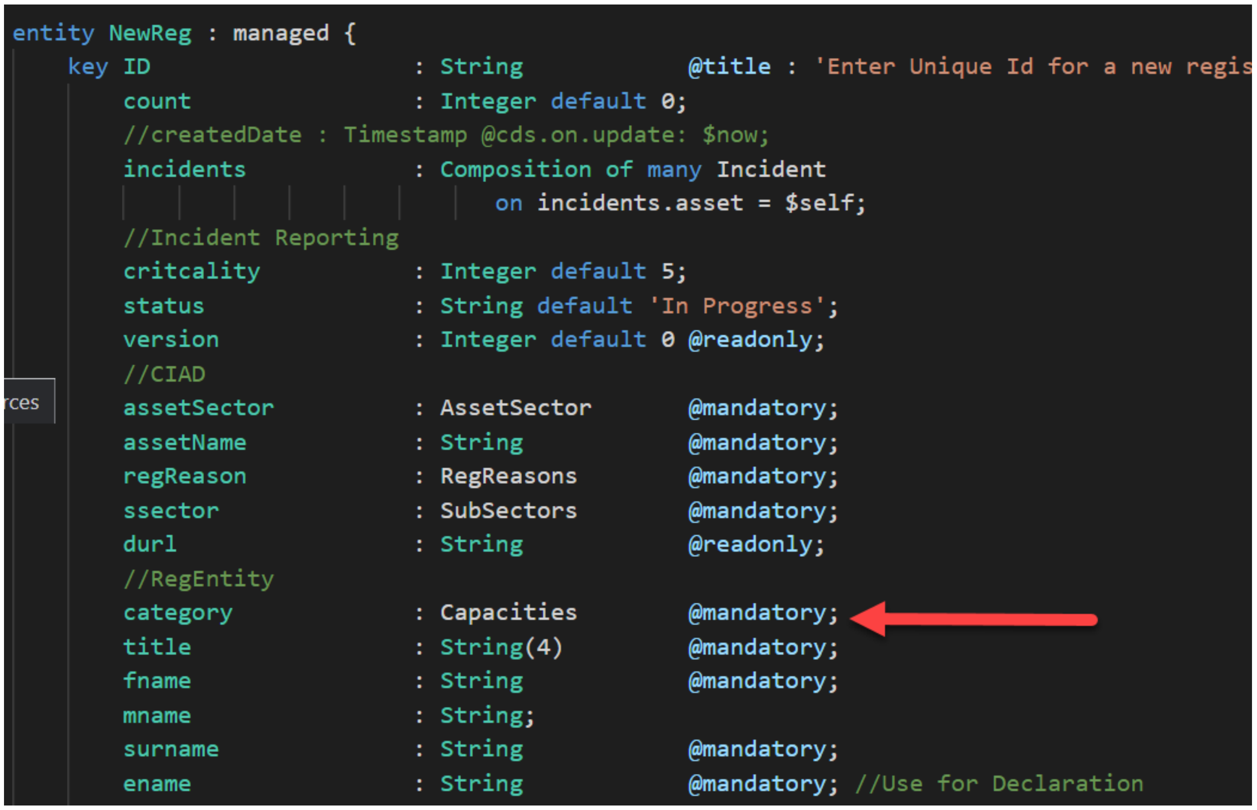
I will be taking an example to show how the drop-down could be achieved. The steps which you have to follow are the following:-

1. First created a user-define data type for which you need the drop down in the **schema.cds** file.



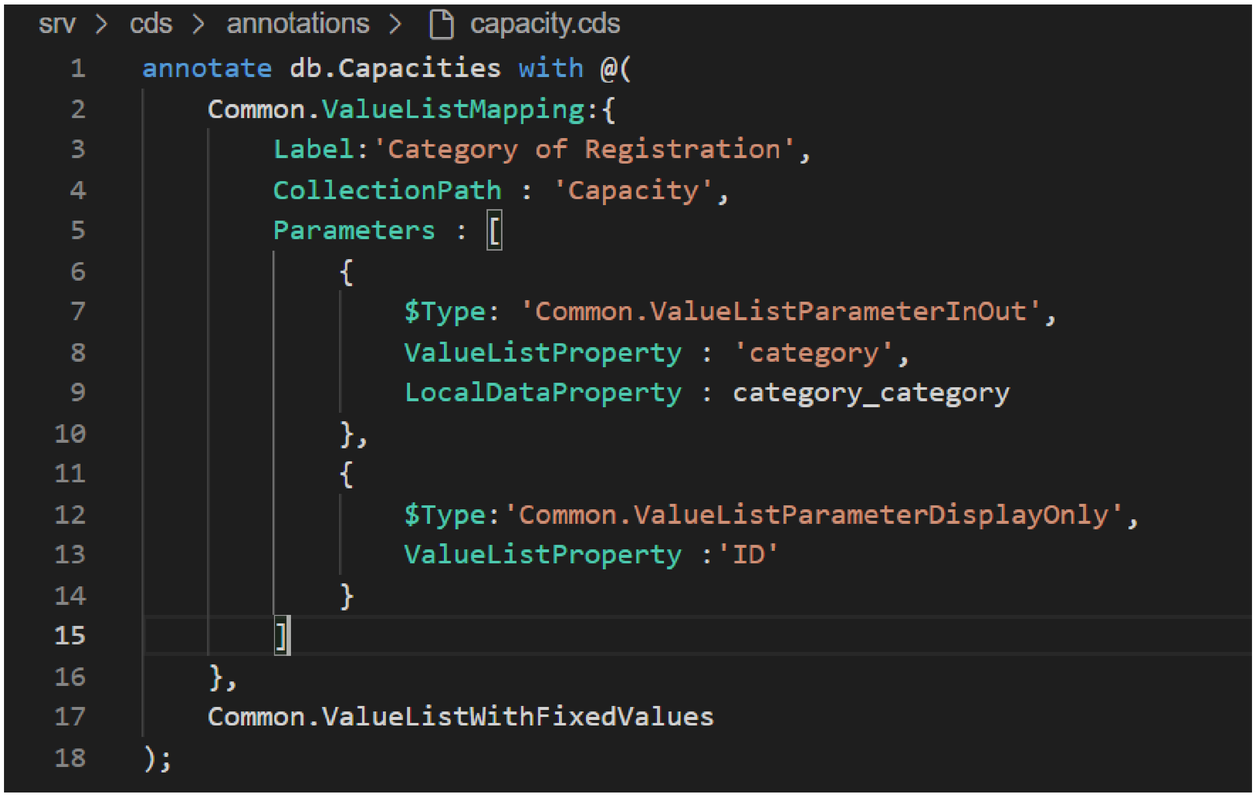
```
type Capacities : Association to one Capacity;

entity Capacity {
        ID        : Integer enum {
            DirectInterestHolder   = 11;
            AgentOrRepresentative = 12;
        };
    key category : String;
}
```

2. Incorporate the created type in the entity where you wish to use the same in the **schema.cds** file.



```
entity NewReg : managed {
    key ID                     : String          @title : 'Enter Unique Id for a new regis
        count                  : Integer default 0;
        //createdDate : Timestamp @cds.on.update: $now;
        incidents              : Composition of many Incident
                                    on incidents.asset = $self;
        //Incident Reporting
        critcality             : Integer default 5;
        status                 : String default 'In Progress';
        version                : Integer default 0 @readonly;
        //CIAD
        assetSector            : AssetSector     @mandatory;
        assetName              : String          @mandatory;
        regReason              : RegReasons      @mandatory;
        ssector                : SubSectors      @mandatory;
        durl                   : String          @readonly;
        //RegEntity
        category               : Capacities      @mandatory;
        title                  : String(4)       @mandatory;
        fname                  : String          @mandatory;
        mname                  : String;
        surname                : String          @mandatory;
        ename                  : String          @mandatory; //Use for Declaration
```

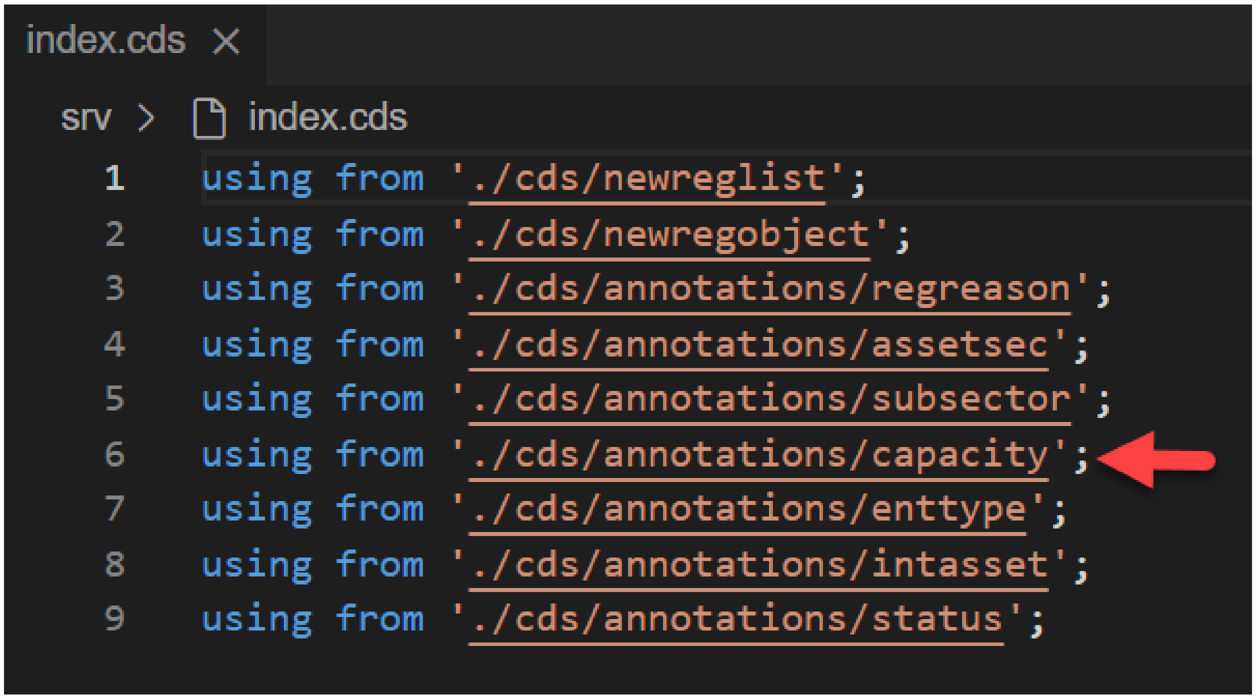
3. In the *srv/cds* folder create another folder called annotations.
4. In the annotations folder create a file with the same name as that of the user-defined entity. In our case Entity is Capacity, so the file name would be capacity.cds and write the following code:
   Note: Please use the Entity name and not the user-defined data type for the filename.



srv > cds > annotations > capacity.cds

```
annotate db.Capacities with @(
    Common.ValueListMapping:{
        Label:'Category of Registration',
        CollectionPath : 'Capacity',
        Parameters : [
            {
                $Type: 'Common.ValueListParameterInOut',
                ValueListProperty : 'category',
                LocalDataProperty : category_category
            },
            {
                $Type:'Common.ValueListParameterDisplayOnly',
                ValueListProperty :'ID'
            }
        ]
    },
    Common.ValueListWithFixedValues
);
```

5. Create a file **index.cds** in the srv folder.
6. Make sure whatever file you use is included in this file as the compiler will know what all files to include. We can include the files using the following format.



index.cds

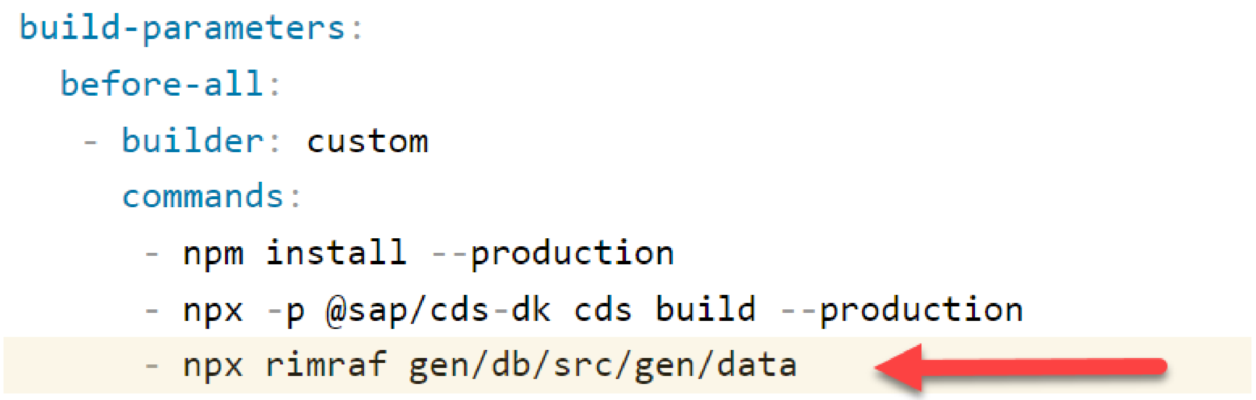
srv > index.cds

```
using from './cds/newreglist';
using from './cds/newregobject';
using from './cds/annotations/regreason';
using from './cds/annotations/assetsec';
using from './cds/annotations/subsector';
using from './cds/annotations/capacity';
using from './cds/annotations/enttype';
using from './cds/annotations/intasset';
using from './cds/annotations/status';
```

DEPLOYING THE APPLICATION ON CLOUD FOUNDRY

There are various steps which needs to be followed for the deploying of the application on Cloud Foundry. And all the steps needs to be followed, or else it will give you errors. In this, I will be focusing on the steps to go for when you have to deploy the Application for the very first on the CF.

1. The first step before doing anything should be, logging into the CF.
   a. In the terminal, first use the *cf login* command
   b. Enter your credentials and choose the dev space in which you would like to deploy your application.
2. You should then add HANA to your project, which can be done by the simple command : *add hana.*
   In case you have already deployed and changed your schema and want to re-deploy, please use *add hana --force.*
3. We need to generate the MTA file next. This can be done by using the command :- *cds add mta.*
4. In the **mta.yaml** file which has been generated. Add the following line, under the commands section in the build-parameters:before-all section.



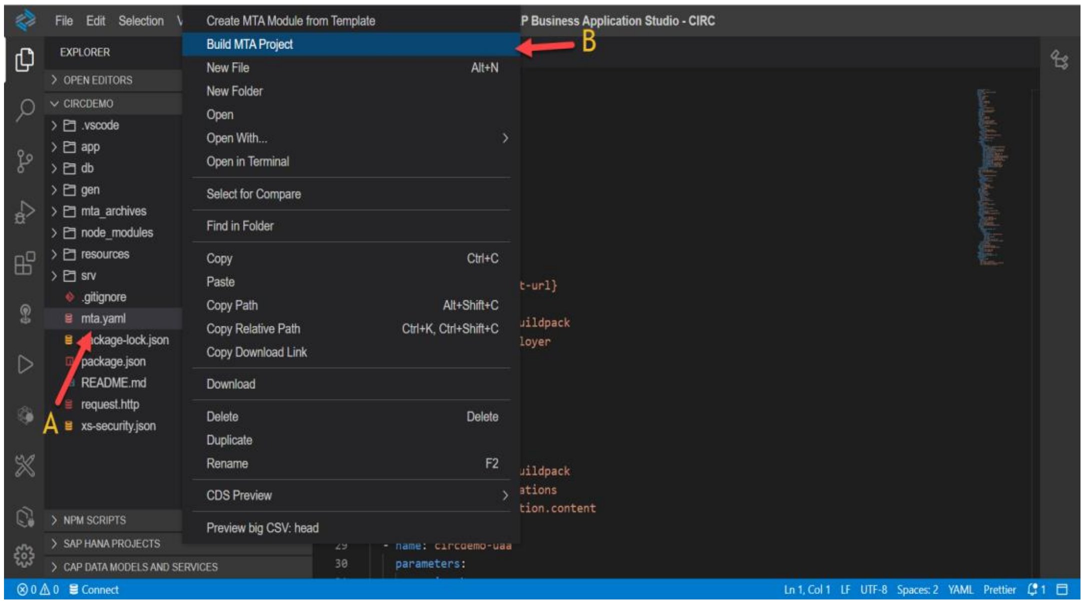
```
build-parameters:
  before-all:
   - builder: custom
     commands:
       - npm install --production
       - npx -p @sap/cds-dk cds build --production
       - npx rimraf gen/db/src/gen/data
```

5. In the **mta.yaml** file again, add the following section under the resources section. As there is no in-built security in the app. This is the only thing which

is required.

```
resources:
   ...
 - name: cpapp-uaa
   type: org.cloudfoundry.managed-service
   parameters:
     service: xsuaa
     service-plan: application
     path: ./xs-security.json
```

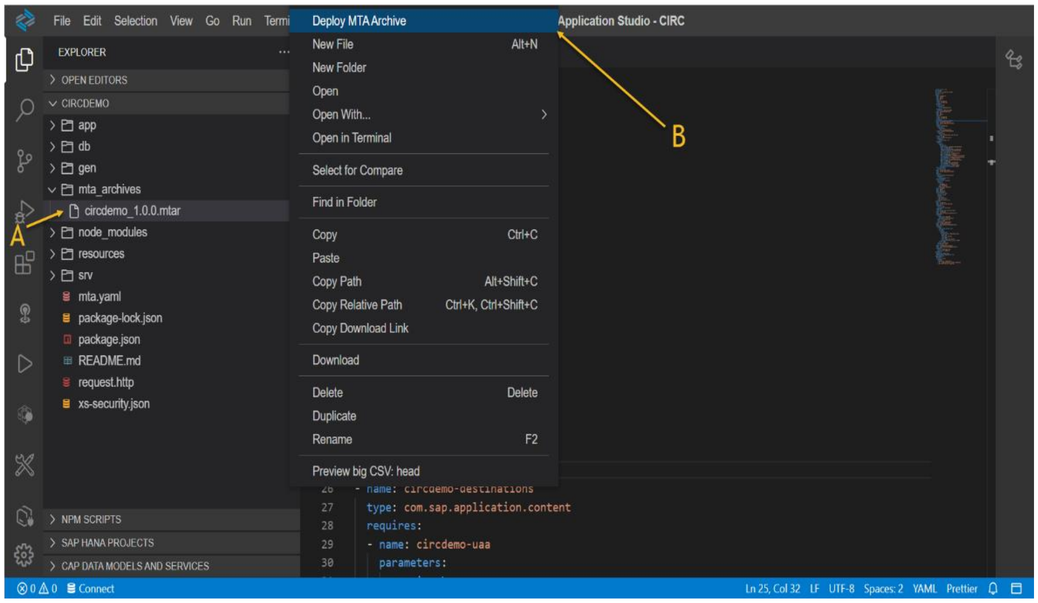
6. The next step is to deploy the **mta.yaml** file. This can be done by either of the two steps:-
   a. Using the command :- *mbt build -t ./* (This will create the **mtar** file in the base project.
   b. Using the GUI:-



   The following method will create a **mta_archives** folder which will contain the **mtar** file.

7. Once the **mtar** file is built, without any errors we need to deploy the **mtar** file on the cloud. The same as the previous step. There are two ways in which this can be achieved. They are:-
   a. Using the command :- *cf deploy <filename>.mtar*

b. Using the GUI:-

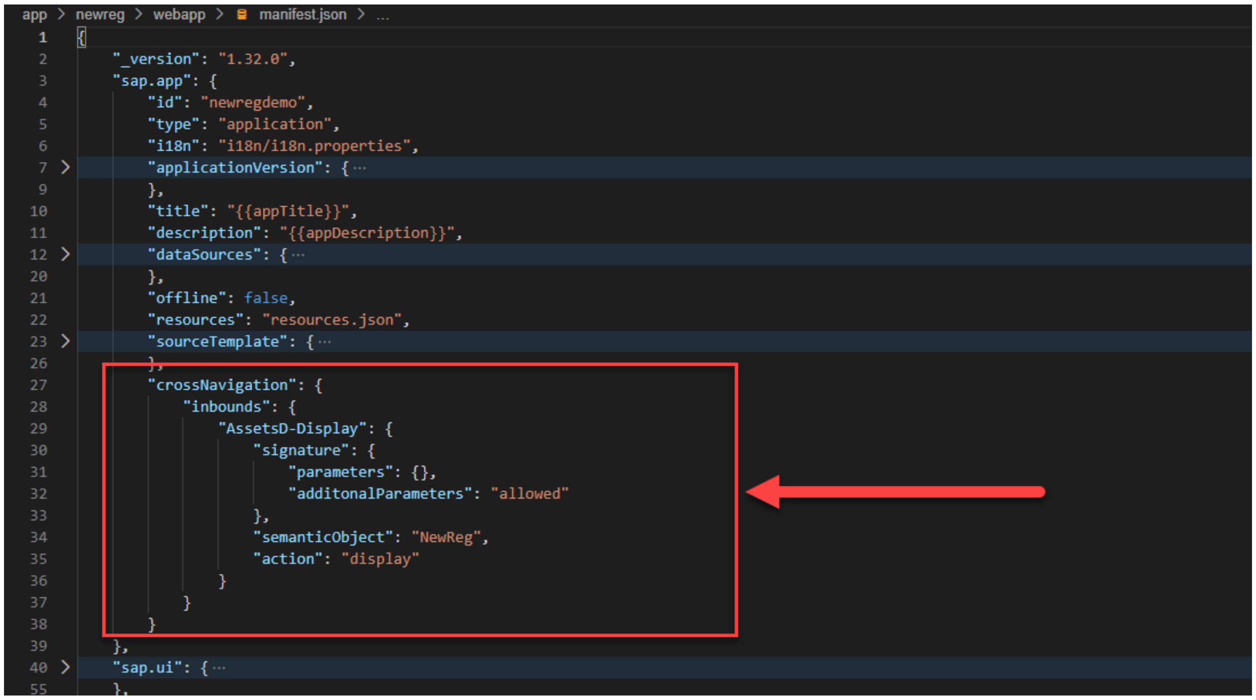


```
- name: circdemo-destinations
  type: com.sap.application.content
  requires:
  - name: circdemo-uaa
    parameters:
```

c. The above two deploy two things in your CF. One would be your hana-db deployer and the other being srv-deployer.

d. Use the command <u>*cf services* </u>and make sure that both these are running and the deploy terminal has no error while deploying.

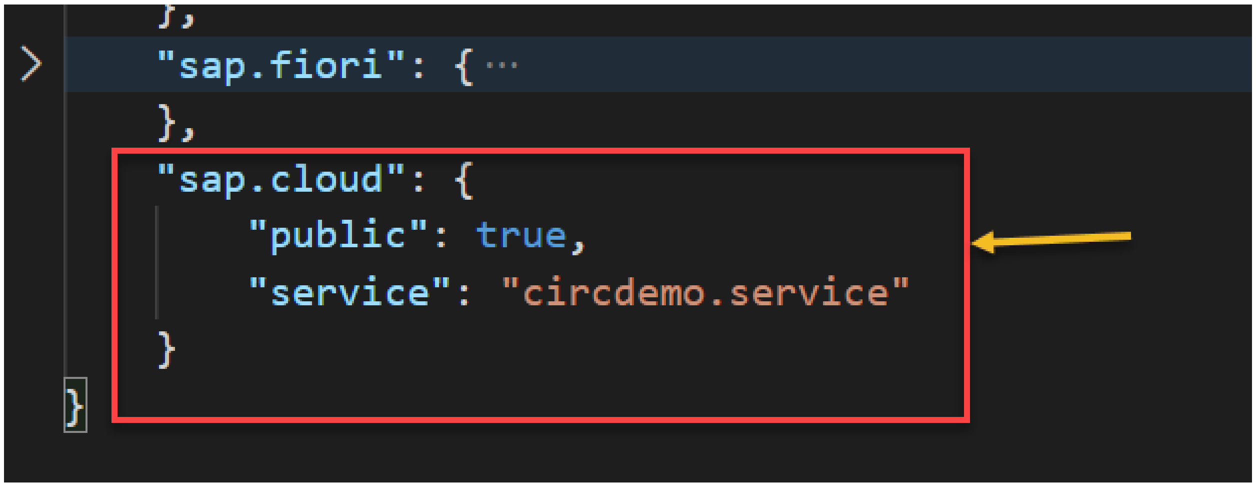
8. Once successfully deployed, you will notice that its just an Odata framework with no data in it as such. The next few steps will show how you can get the Fiori layout as well as the data in the file.
9. In the **app/newreg/webapp/manifest.json** add the <u>cross navigation</u> part to the code.



app > newreg > webapp > manifest.json > ...

```
{
    "_version": "1.32.0",
    "sap.app": {
        "id": "newregdemo",
        "type": "application",
        "i18n": "i18n/i18n.properties",
        "applicationVersion": {···
        },
        "title": "{{appTitle}}",
        "description": "{{appDescription}}",
        "dataSources": {···
        },
        "offline": false,
        "resources": "resources.json",
        "sourceTemplate": {···
        },
        "crossNavigation": {
            "inbounds": {
                "AssetsD-Display": {
                    "signature": {
                        "parameters": {},
                        "additonalParameters": "allowed"
                    },
                    "semanticObject": "NewReg",
                    "action": "display"
                }
            }
        }
    },
    "sap.ui": {···
    },
```

10. Add the following section at the end of the **app/newreg/webapp/manifest.json** file.



```
    },
    "sap.fiori": {···
    },
    "sap.cloud": {
        "public": true,
        "service": "circdemo.service"
    }
}
```

11. Add the following section in the **mta.yaml** file under the resources section

```
- name: [redacted]-destination
  type: org.cloudfoundry.managed-service
  parameters:
    service: destination
    service-plan: lite
    config:
      HTML5Runtime_enabled: true
```

12. Add the following section in the **mta.yaml** file under the resources section

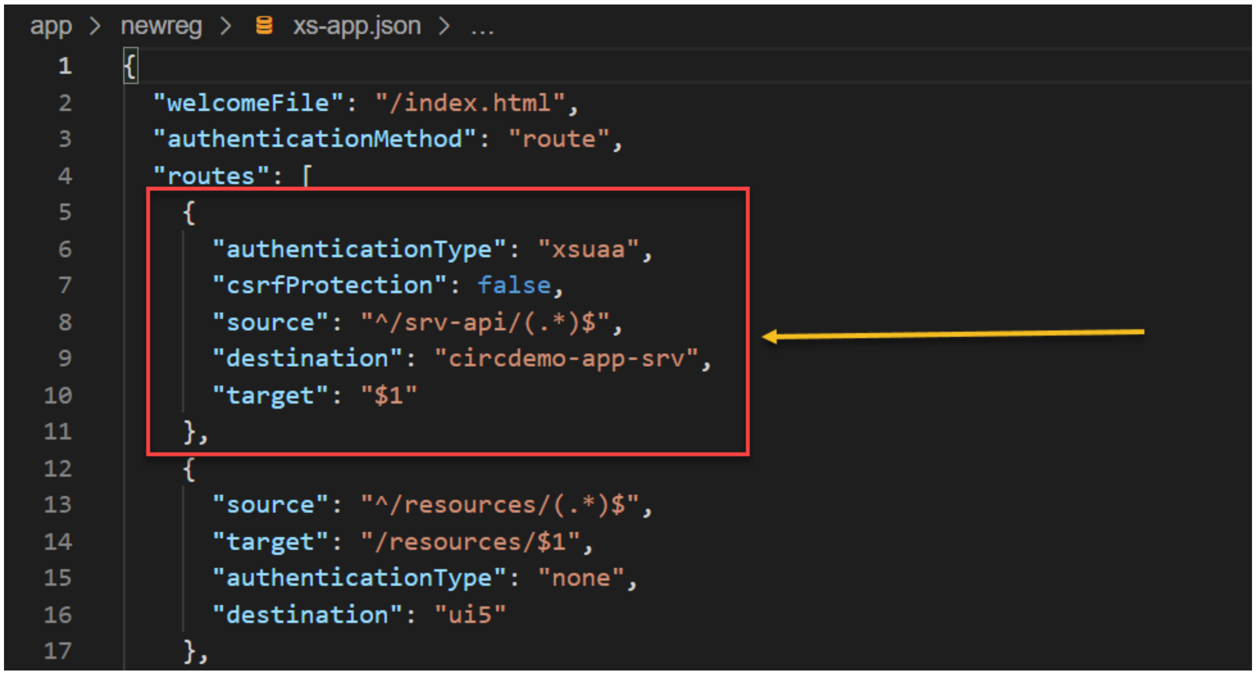
```
resources:
   ...
 - name: [redacted]-html5-repo-host
   type: org.cloudfoundry.managed-service
   parameters:
     service: html5-apps-repo
     service-plan: app-host
```

13. Add the three destinations in the **mta.yaml** file under the modules section.

```yaml
- name: cpapp-destinations
  type: com.sap.application.content
  requires:
    - name: cpapp-uaa
      parameters:
        service-key:
          name: cpapp-uaa-key
    - name: cpapp-html5-repo-host
      parameters:
        service-key:
          name: cpapp-html5-repo-host-key
    - name: srv-api
    - name: cpapp-destination
      parameters:
        content-target: true
  parameters:
    content:
      instance:
        destinations:
          - Authentication: OAuth2UserTokenExchange
            Name: cpapp-app-srv
            TokenServiceInstanceName: cpapp-uaa
            TokenServiceKeyName: cpapp-uaa-key
            URL: '~{srv-api/srv-url}'
            sap.cloud.service: cpapp.service
          - Name: cpapp-html5-repo-host
            ServiceInstanceName: cpapp-html5-repo-host
            ServiceKeyName: cpapp-html5-repo-host-key
            sap.cloud.service: cpapp.service
          - Authentication: OAuth2UserTokenExchange
            Name: cpapp-uaa
            ServiceInstanceName: cpapp-uaa
            ServiceKeyName: cpapp-uaa-key
            sap.cloud.service: cpapp.service
        existing_destinations_policy: update
  build-parameters:
    no-source: true
```

14. Before proceeding any further, please add the following under the **app/newreg/xs-app.json** file under the routes section or else while the fiori

app is being deployed it will give a lot of errors.



app > newreg > xs-app.json > ...

```
{
  "welcomeFile": "/index.html",
  "authenticationMethod": "route",
  "routes": [
    {
      "authenticationType": "xsuaa",
      "csrfProtection": false,
      "source": "^/srv-api/(.*)$",
      "destination": "circdemo-app-srv",
      "target": "$1"
    },
    {
      "source": "^/resources/(.*)$",
      "target": "/resources/$1",
      "authenticationType": "none",
      "destination": "ui5"
    },
```

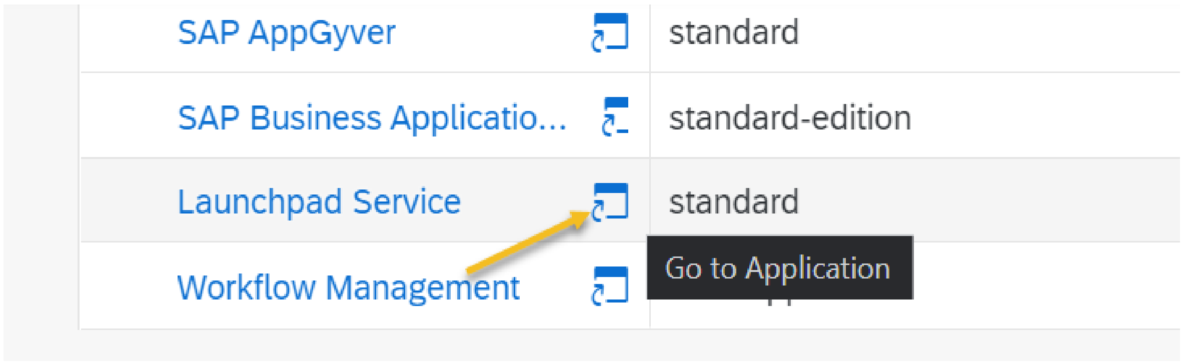
15. In the **app** folder of the project, open a terminal and run the following command: *npm install - - global @sap/ux-ui5-tooling*
16. Go back to the project root folder and open a terminal for the root folder. Run the following command: *npm install - -global mta*
17. Navigate to the **app/newreg** folder and run the command: *fiori add deploy-config cf*
18. Rebuild the **mta.yaml** file from either of the two methods in step (6)
19. Deploy the **.mtar** file by either of the two methods in step (7)
20. The next step is to check whether the HTML application has been deployed in the CF.

SAP LAUNCHPAD SERVICE

The following are the steps which we need to follow to check if the app that we have deployed is available on the SAP Launchpad or not.

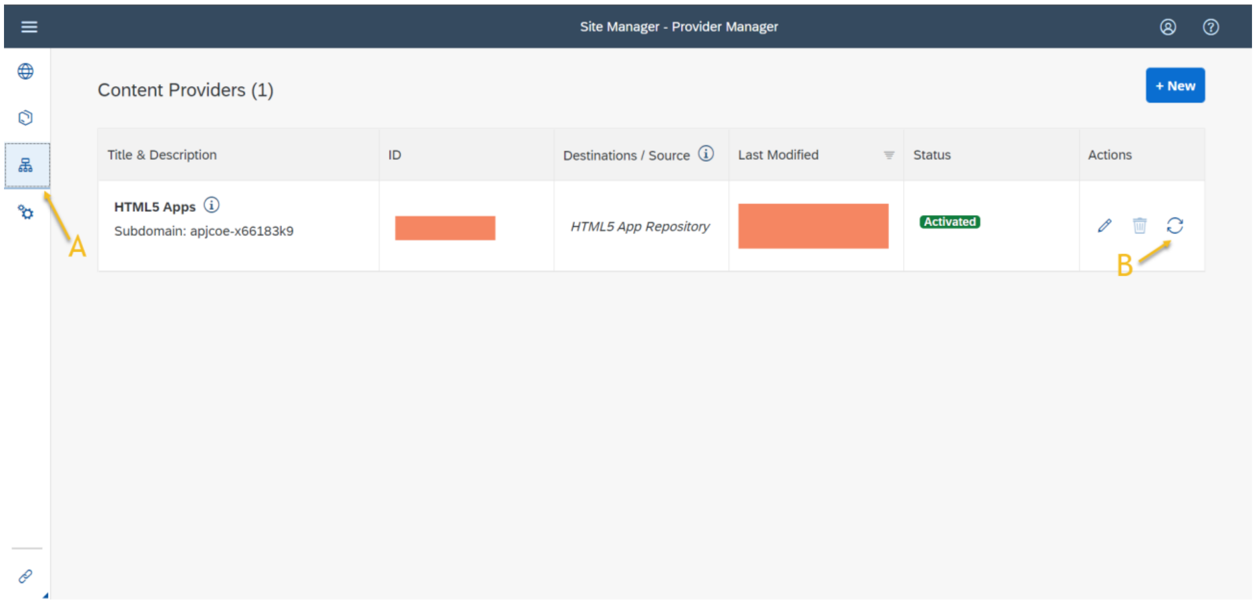
The steps are:-

1. Launch the SAP Launchpad application from your Global/Trial Account, under the instances/subscriptions.



SAP AppGyver	standard
SAP Business Applicatio...	standard-edition
Launchpad Service	standard
Workflow Management	

2. In the Content Providers tab, Please refresh the HTML5 Apps which are being shown.

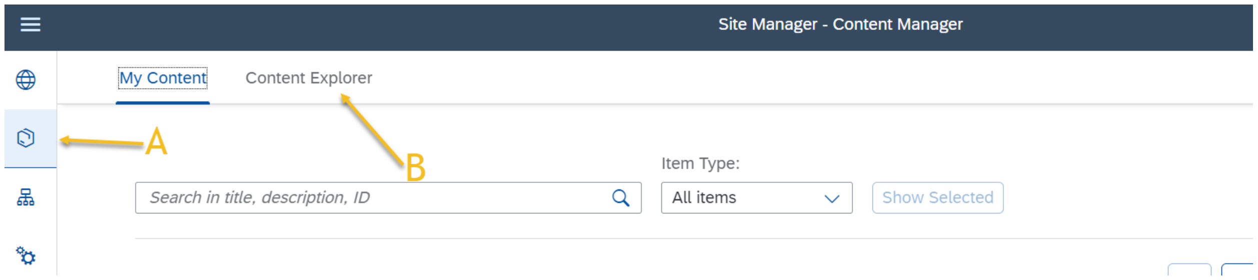


Site Manager - Provider Manager

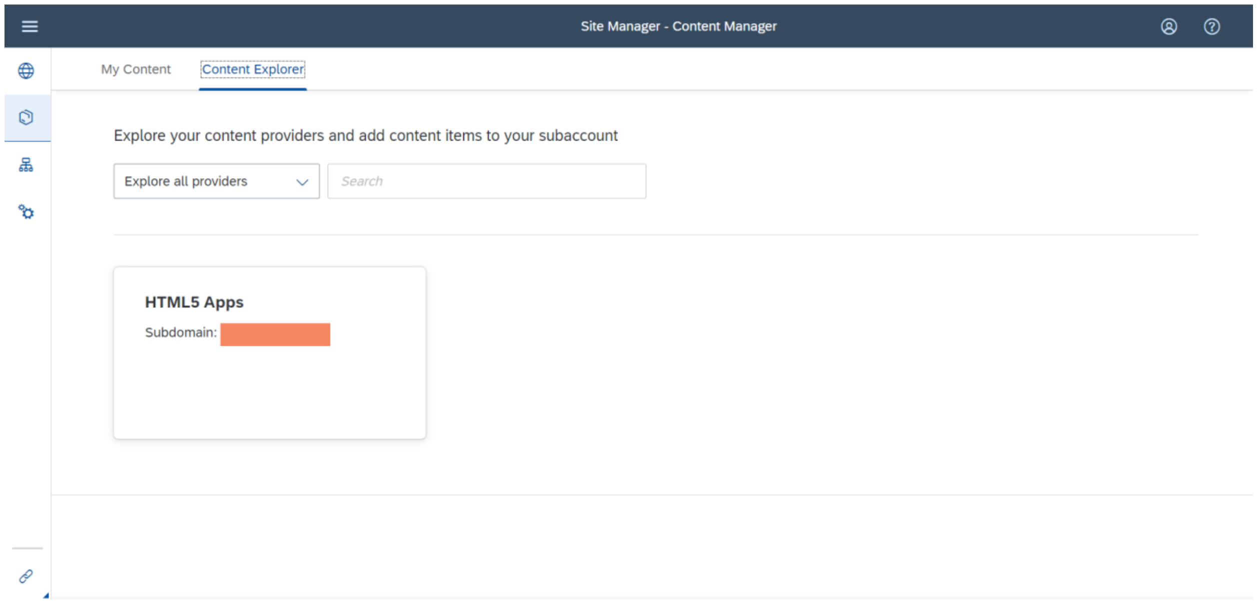
Content Providers (1)

Title & Description	ID	Destinations / Source	Last Modified	Status	Actions
HTML5 Apps Subdomain: apjcoe-x66183k9	[redacted]	*HTML5 App Repository*	[redacted]	Activated	

3. In the Content Manager tab of the Launchpad Service, go to the Content Explorer tab.


Site Manager - Content Manager
My Content
Content Explorer
A
B
Item Type:
Search in title, description, ID
All items
Show Selected

4. In the HTML5 Apps check if your application has shown up



Explore your content providers and add content items to your subaccount

Subdomain: [redacted]

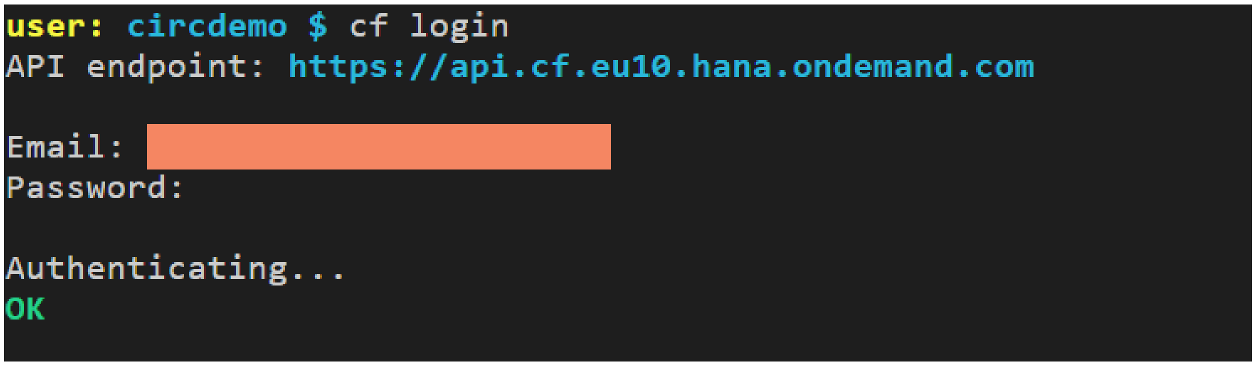
5. Make the visibility of your site to everyone, create a new site and the Fiori tile which you have created will show up.

STEPS TO RE-DEPLOY AN APPLICATION

In case your app has already been deployed, we can follow the under given steps to re-deploy the application on CF.

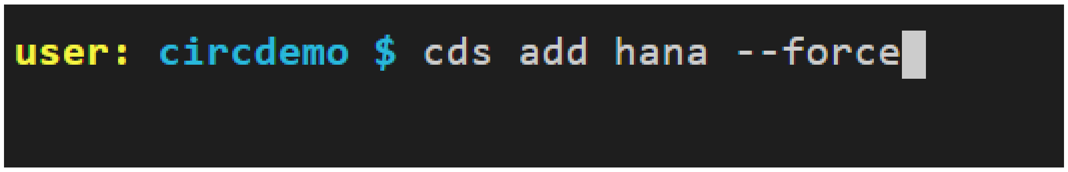
The steps are:

1. Login to Cloud Foundry.



```
user: circdemo $ cf login
API endpoint: https://api.cf.eu10.hana.ondemand.com

Email: [redacted]
Password:

Authenticating...
OK
```

2. Check if the target dev is correct after authentication
3. Add HANA again by using the force option.



```
user: circdemo $ cds add hana --force
```

4. Build the **mta.yaml** file.
5. On successful build, deploy the **.mtar** file.

REFERENCES

1. https://developers.sap.com/tutorials/btp-app-hana-cloud-setup.html

2. https://developers.sap.com/tutorials/cap-service-deploy.html