**Connecting to the external SAP BTP Destination**

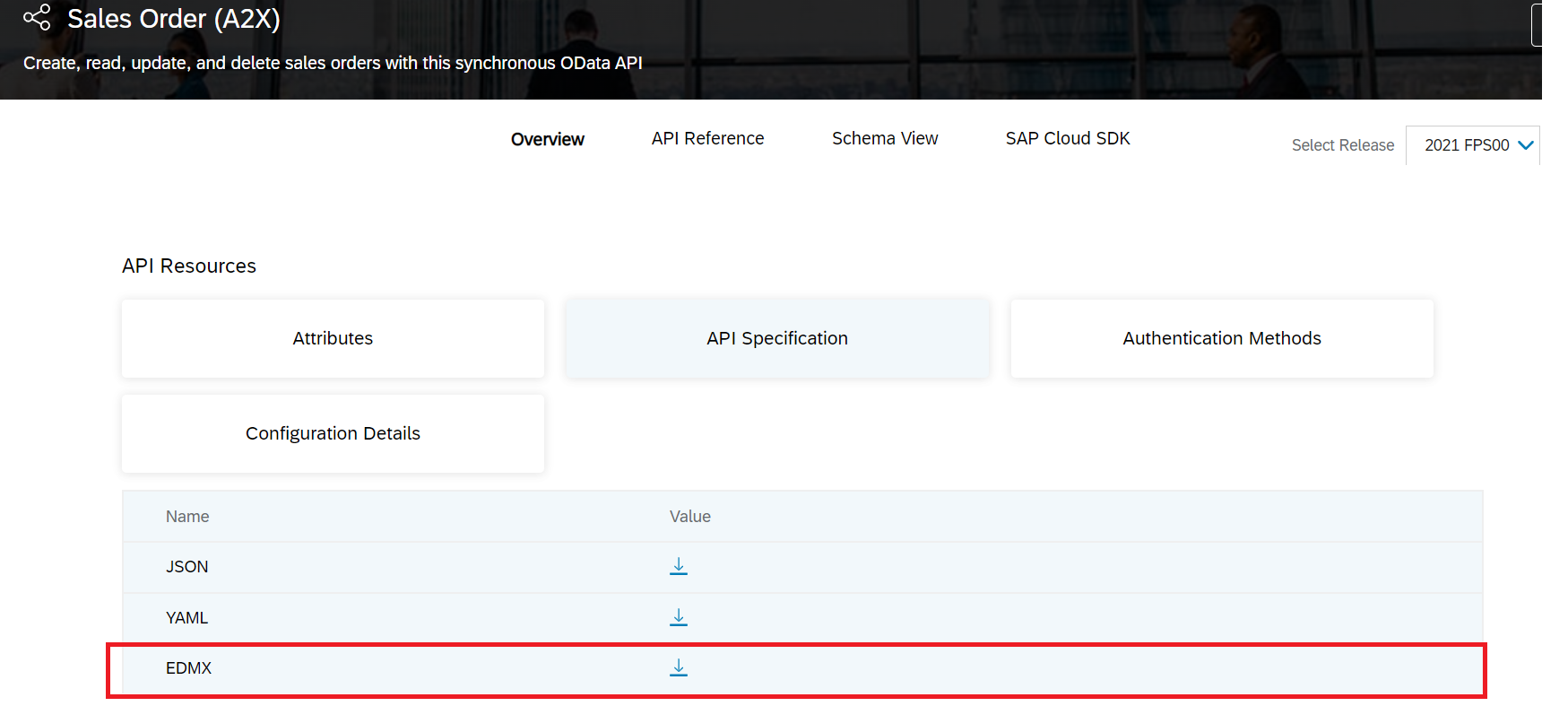
**Prerequisites**: completed Environment Setup tutorial

**Result**: Configured destination service that consumes external service and returns SalesOrder data.

**1. Importing SalesOrder OData service**

To define a SalesOrder model, we can download and import API reference from SAP API Business Hub to our CAP application.

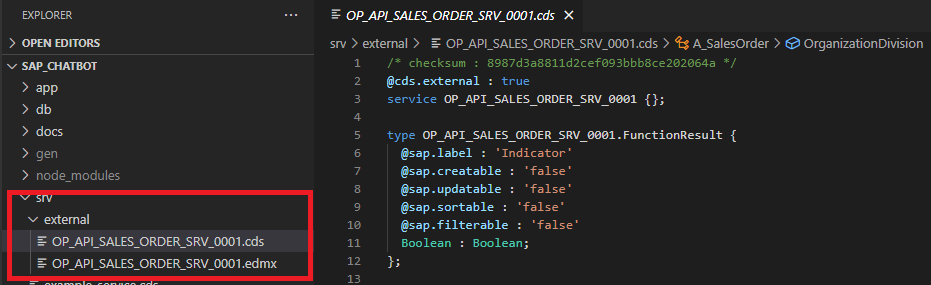
1. Download API specification of SalesOrder OData in EDMX format from *https://api.sap.com/api/OP\_API\_SALES\_ORDER\_SRV\_0001/overview*



1. Move the downloaded file to the project root folder and run the following command to import API specification.

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds import <filename>.edmx | |  |

This should have created a new folder called *external* with two files as seen in the screenshot.



**2. Modifying our example service to use newly imported API specification**

We are going to use the newly imported API to define a projection on the SalesOrder data.

1. Replace the content of the example-service.cds file to define a service endpoint/entity *TestSaleOrders* with the projection of the imported SalesOrder data.

using OP\_API\_SALES\_ORDER\_SRV\_0001.A\_SalesOrder as salesorder

from './external/OP\_API\_SALES\_ORDER\_SRV\_0001';

service ExampleService {

@readonly

entity TestSaleOrders as projection on salesorder {

key SalesOrder, LastChangeDate, CreationDate, TotalNetAmount

};

}

1. Create a new *example-service.js* Javascript handler. The name must match the name of the cds service so that CAP knows that it should invoke methods/routines when the *TestSaleOrders* entity is being read/written/updated.

const cds = require('@sap/cds');

module.exports = cds.service.impl(async function() {

const service = await cds.connect.to('OP\_API\_SALES\_ORDER\_SRV\_0001'); // connect to the external destination

const { TestSaleOrders } = this.entities;

this.before('\*', (req) => {

console.debug('>>>', req.method, req.target.name)

});

this.on('READ', TestSaleOrders, request => {

return service.tx(request).run(request.query);

});

});

The code gets called once the TestSaleOrders entity is being requested using a GET request. It connects to the BTP destination and returns all results in *this.on()* part.

1. Since CAP does not support ODataV2 by default, we need to use middleware to enable it.

Add the middleware package to the project by running:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | npm add @sap/cds-odata-v2-adapter-proxy | |  |

Create a new file *server.js* under *srv* folder with:

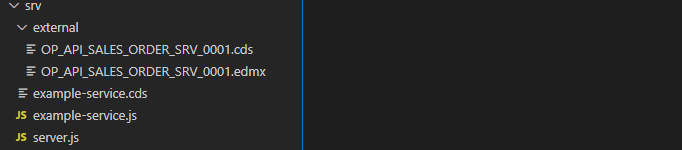
const proxy = require('@sap/cds-odata-v2-adapter-proxy')

const cds = require('@sap/cds')

cds.on('bootstrap', app => app.use(proxy()))

module.exports = cds.server

The *srv* folder structure after step 2 should look like this:



**3. Define Destination service**

The destination service can be created either from the BTP cockpit or in the *mta.yaml* file. We are going to use the *mta.yaml* approach based on which CAP will create the services upon deployment of the application.

1. Generate security descriptor file *xs-security.json* using the command below:

|  |  |  |  |
| --- | --- | --- | --- |
| Shell/Bash | |  |  |
| 1 | | cds compile srv/ --to xsuaa > xs.security.json | |  |

1. Define the destination service with provided credentials and properties. Since we are connecting to an OnPremise destination, we need to additionally define a connectivity service. In the end, add the destination service, connectivity service and authentication service to the *“requires”* part of the *mta.yaml* file. The file after the *“requires”* part should look something like this:

...

requires:

- name: my-destination-service

- name: sales-xsuaa

- name: my-connectivity-service

resources:

- name: my-destination-service

type: org.cloudfoundry.managed-service

parameters:

config:

HTML5Runtime\_enabled: true

version: 1.0.0

init\_data:

instance:

destinations:

- Authentication: BasicAuthentication

Name: ERP\_IDA\_SO\_SRV

ProxyType: OnPremise

CloudConnectorLocationId: SCC01

Type: HTTP

URL: <your URL>

User: <your User>

Password: <your Password>

HTML5.ForwardAuthToken: true

service: destination

service-plan: lite

- name: sales-xsuaa

type: org.cloudfoundry.managed-service

parameters:

path: ./xs-security.json

service: xsuaa

service-plan: application

- name: my-connectivity-service

type: org.cloudfoundry.managed-service

parameters:

service-plan: lite

service: connectivity

1. Adjust *package.json* file to make a correspondence between the model specification and the destination.

. . .

},

"rules": {

"no-console": "off",

"require-atomic-updates": "off"

}

},

"cds": {

"requires": {

"OP\_API\_SALES\_ORDER\_SRV\_0001": {

"kind": "odata-v2",

"model": "srv\\external\\OP\_API\_SALES\_ORDER\_SRV\_0001",

"credentials": {

"destination": "ERP\_IDA\_SO\_SRV"

}

}

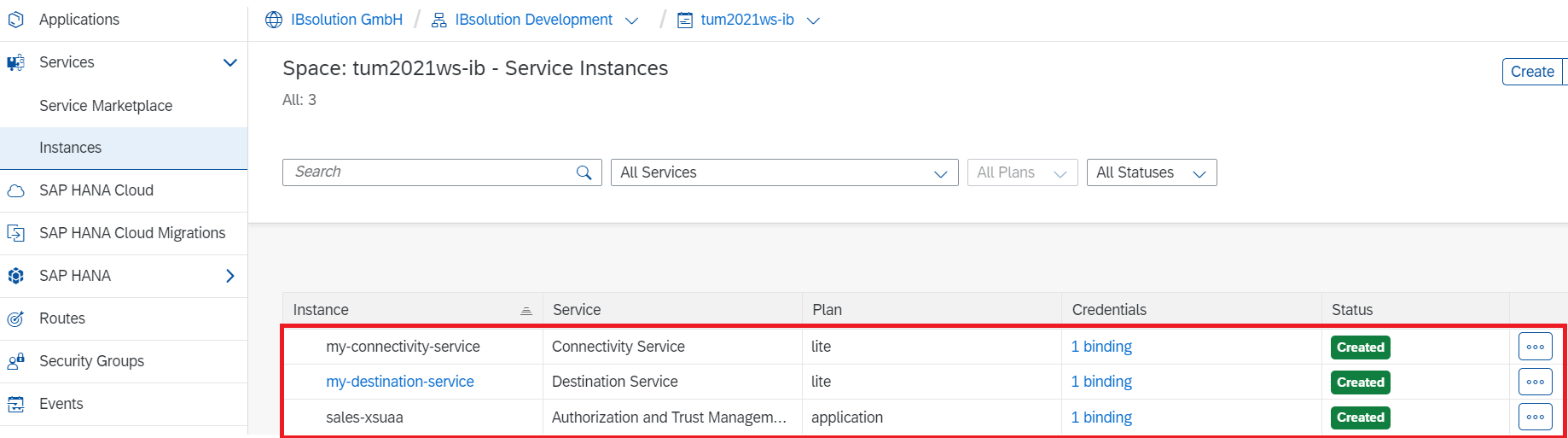
}

}

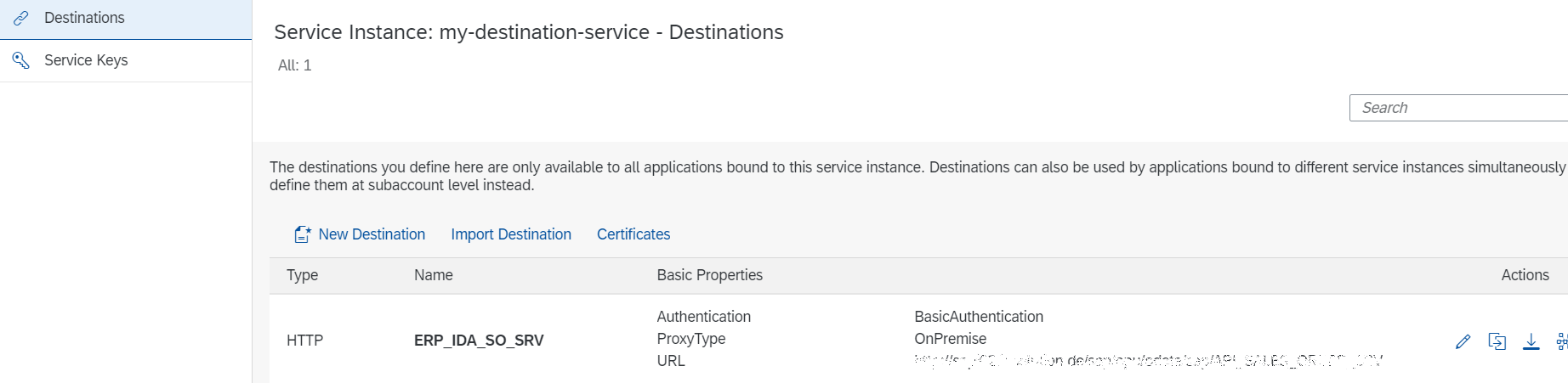
**4. Test the destination by deploying the application to Cloud Foundry**

To test the destination service, we need to deploy the application to the Cloud Foundry and check if the GET request for the defined entity endpoint executes successfully.

1. Build and deploy the application to the Cloud Foundry as illustrated in the Environment Setup tutorial. Once the application is running, navigate to your application service instances. You should see following services.



Click on *my-destination-service* and navigate to Destinations, you should see your destination.



1. Navigate to the <url>/example/TestSaleOrders and check if the request executes successfully. If the request returns 401 Forbidden or 502 Bad Gateway code, make sure the credentials are filled correctly. You should get a response in JSON format with TestSaleOrders entities.

