

**IBM z/OS Connect (OpenAPI 3.0)**

# **Developing and Administering RESTful APIs for Db2 Native REST Services**



**IBM**  
IBM Z  
Wildfire Team –  
Washington System Center

*Lab Version Date: July 27, 2022*

## Table of Contents

<b>Overview .....</b>	<b>3</b>
<b>Db2 REST services and z/OS Connect.....</b>	<b>4</b>
Creating and testing Db2 REST Services.....	4
<b>The z/OS Connect Designer Container environment.....</b>	<b>13</b>
Setup required Windows Subsystems for Linux support.....	14
Docker Engine.....	15
Podman.....	15
The container configuration file.....	16
Connecting to Db2 and the required server XML configuration.....	17
Basic security and the required server XML configuration .....	17
Accessing the z/OS Connect Designer log and trace files .....	19
<b>Managing the z/OS Connect Designer container .....</b>	<b>21</b>
Using Docker commands.....	21
Creating a new container.....	21
Refresh an existing container .....	22
Using Podman commands .....	23
Designer problem determination.....	24
<b>Developing a z/OS Connect APIs that accesses Db2 .....</b>	<b>26</b>
Configure the POST method for URI path /employees.....	28
Configure the GET method for URI path /employees/{employee}.....	43
<b>Testing the API's POST and GET methods .....</b>	<b>48</b>
<b>Complete the configuration of the API (Optional).....</b>	<b>52</b>
Configure the PUT method for URI path /employees/{employee} .....	52
Configure the GET method for URI path /employees/details/{employee} .....	56
Configure the DELETE method for URI path /employees/{employee} .....	61
Configure the GET method for URI path /roles/{job} .....	65
<b>Testing APIs deployed in a z/OS Connect Designer container .....</b>	<b>70</b>
Using Postman.....	71
Using cURL .....	78
Using the API Explorer.....	81
<b>Deploying and installing APIs in a z/OS Connect Native Server .....</b>	<b>87</b>
Moving the API Web Archive file from the container to a z/OS OMVS directory .....	87
Updating the server xml .....	88
Defining the required RACF EJBRole resources .....	89
<b>Testing APIs deployed in a native z/OS server .....</b>	<b>90</b>
Using Postman.....	90
Using cURL .....	95
Using the API Explorer.....	97
<b>Using TLS to access Db2.....</b>	<b>102</b>
Creating the RACF resources .....	102
Configuring the Db2 AT-TLS inbound policy .....	104
Activating the AT-TLS configuration .....	124
Test the TLS connection from the z/OS Connect Designer to a Db2 subsystem .....	127
Using a JSSE key store from the Designer .....	128
Server certificate authentication.....	131
Client certificate (mutual) authentication.....	137
Using RACF key rings from the native z/OS server.....	144
<b>Additional information and samples.....</b>	<b>145</b>
JCL to define and load the Db2 table USER1.EMPLOYEE.....	145
Client Certificate Requests .....	146
Suggestions for customizing the Linux shell environment.....	147
Useful commands for managing containers .....	148
AT-TLS policy file .....	150
The contents of the employees.yaml file .....	151

**Important:** On the desktop there is a file named *OpenAPI 3 development APIs CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

## Overview

The objective of these exercises is to gain experience with working with *z/OS Connect* and the *z/OS Connect Designer*. This exercise is offered in conjunction with a Washington Systems Center Wildfire workshop for *z/OS Connect*. For information about scheduling this workshop in your area contact your IBM representative.

**Important – You do not need any skills with Db2 to perform this exercise. Even if Db2 is not relevant to your current plans, performing the steps in this exercise will give additional experience using the *z/OS Connect Designer* to developing and administer APIs.**

### General Exercise Information and Guidelines

- ✓ This exercise requires using *z/OS* user identities *Fred*, *USER1* and *USER2*. The *Designer* passwords for these identities are *fredpwd*, *user1* and *user2* respectively and are case sensitive. The RACF password for these users are *FRED*, *USER1* and *USER2* respectively and are case insensitive.
- ✓ The IP address of the image where the *z/OS Connect Designer* containers are running has been configured in the *../etc/hosts* file as *designer.washington.ibm.com*. This was done so the host name that appears in this exercise would be consistent regardless of whether the containers were running in a local Windows Subsystem for Linux image using a loop back adapter or a remote server image.
- ✓ Any time you have any questions about the use of screens, features or tools do not hesitate to reach out for assistance.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *OpenAPI 3 development APIs CopyPaste* file on the desktop
- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see when performing this exercise. These differences should not impact the completion of this exercise. For example, the text might reference host name *designer.washington.ibm.com* when a screen shot shows the host as *designer.ibm.com* or even *localhost*. All of these names resolve to the same IP address. Another example is that a section of a page has been expanded for display purposes. If a section or screen shot does not look exactly as what you are observing, consider maximizing or minimizing that section

## ***Db2 REST services and z/OS Connect***

Accessing a Db2 REST service from z/OS Connect differs from the ways in which z/OS Connect accesses the resources of other z/OS subsystems. Other subsystem's resources are accessed by using their normal subsystem interfaces (e.g., OTMA, IPIC, JMS, etc.).

A z/OS Connect server accesses Db2 not as a Db2 client using JDBC, but rather as a RESTful client accessing an existing Db2 REST service. This may raise the question as to what value-add does z/OS Connect provide if z/OS Connect can only access an existing Db2 REST service? The answer is that (1) the REST services support provided by Db2 only supports the POST method with only a few administrative services that support the GET method. There is no support for PUT or DELETE methods normally expected for a robust RESTful API service. Another reason (2) is that the API function of transforming JSON request or response messages, e.g., assigning values or removing fields from the interface is not available when using the Db2 native REST Services directly.. And finally (3) z/OS Connect provides security mechanism (e.g., OAUTH and JWT tokens) not available with Db2. If a full function RESTful API with support for the major HTTP methods (POST, PUT, GET and DELETE), or transforming JSON payloads and/or additional authentication methods are required, then z/OS Connect is the solution

### ***Creating and testing Db2 REST Services***

Db2 REST services are defined either using a Db2 provided RESTful administrative service (DB2ServiceManager) or by using the Db2 BIND command using an update provided in Db2 PTF UI51748 and APAR PI98649 (PTF UI584231 or UI58425). Only the latter technique that will be shown in this exercise.

1. Log onto TSO and access data set USER1.ZCEE.CNTL
2. Select member *DB2REST1* in *USER1.ZCEE.CNTL* and right- mouse button clicking and submit this job for execution. It should complete with a completion code of 0.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//          DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT DD *
      SELECT EMPNO AS "employeeNumber", FIRSTNME AS "firstName",
             MIDINIT AS "middleInitial", LASTNAME as "lastName",
             WORKDEPT AS "department", PHONENO AS "phoneNumber",
             JOB AS "job"
      FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN DD *
      DSN SYSTEM(DSN2)
      BIND SERVICE("zCEEService") -
      NAME("selectEmployee") -
      SQLENCODING(1047) -
      DESCRIPTION('Select an employee from table USER1.EMPLOYEE')
/*
```

This defines a Db2 native REST Services that select a single row from table USER1.EMPLOYEE based on the employee number (column EMPNO).

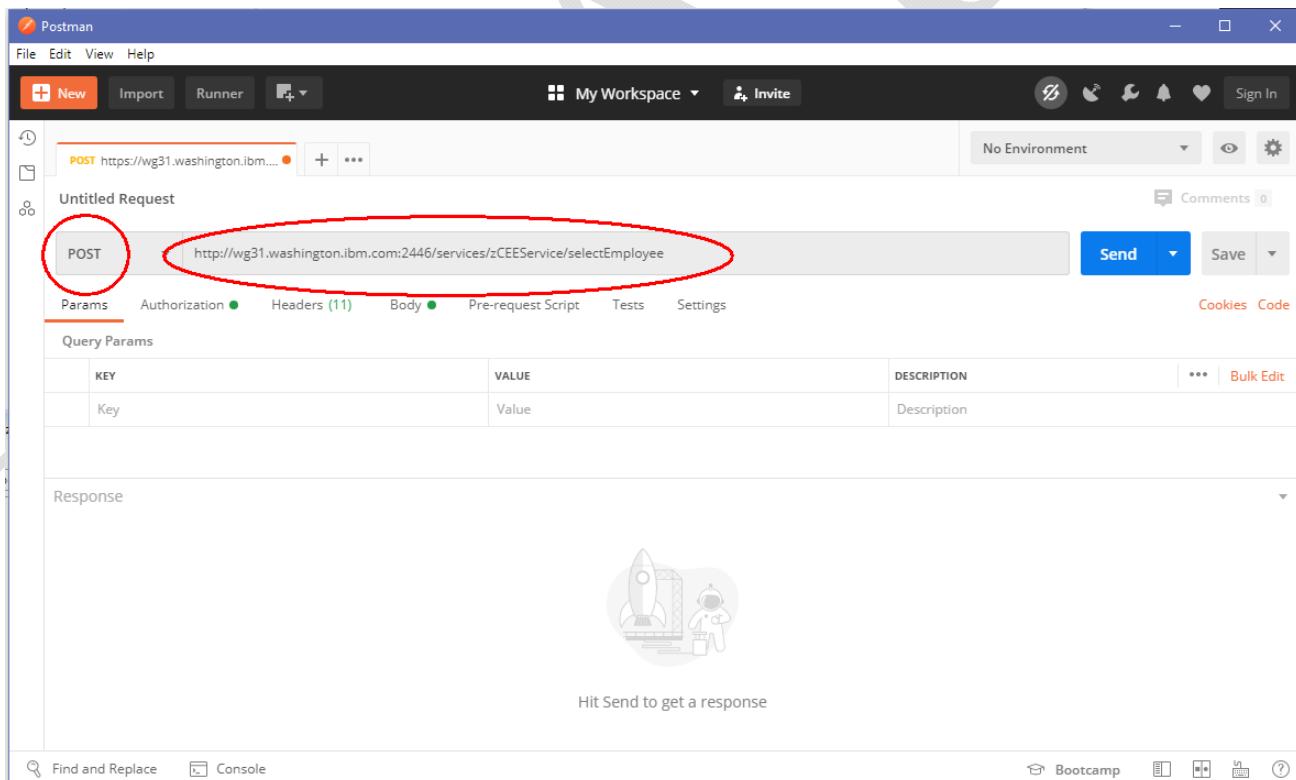
**Important:** The DBA creating this native Db2 REST service is excluding other table columns, e.g., SEX, SALARY, BONUS, COMMISION, etc. from the selection by omitting these columns from the SELECT statement. The DBA's use of the AS cause will also ensure the assigning of meaningful JSON property names rather than the original Db2 column names to the JSON request and response messages.

**Tech-Tip:** The input to DD DSNSTMT can be a CALL, DELETE, INSERT, SELECT, TRUNCATE, UPDATE, or WITH SQL statement.

To delete a service created by using the Db2 BIND command use the Db2 FREE command, e.g., FREE SERVICE("zCEEService"."selectEmployee")

**Tech-Tip:** A minimum of EXECUTE authority on package zCEEService.selectEmployee would be required to have the ability to execute this service.

3. Open the *Postman* tool icon on the desktop and if necessary reply to any prompts and close any welcome messages, use the down arrow to select **POST** and enter <http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee> in the URL area (see below).



**Tech-Tip:** If the above Postman view is not displayed select *File* on the toolbar and then choose *New Tab* on the pull down. Alternatively, if the *Launchpad* view is displayed, click on the *Create a request* option.

4. No *query* or *path* parameters are required so next select the *Authorization* tab to enter an authorization identity and password. Use the pull down arrow to select *Basic Auth* and enter **USER1** as the *Username* and USER1's password as the *Password*.

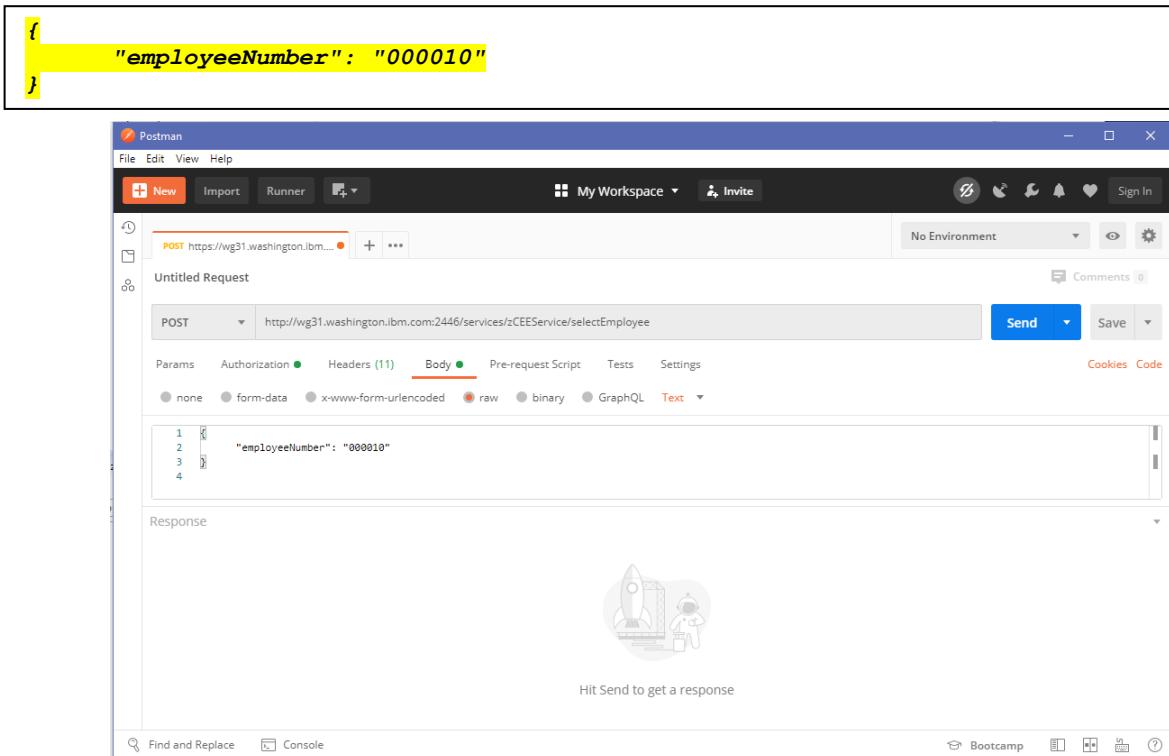
The screenshot shows the Postman interface with a POST request to <http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee>. The 'Authorization' tab is selected. A dropdown menu labeled 'TYPE' shows 'Basic Auth' selected, with other options like 'Inherit auth from parent', 'No Auth', 'API Key', 'Bearer Token', 'Digest Auth', 'OAuth 1.0', 'OAuth 2.0', 'Hawk Authentication', and 'AWS Signature'. To the right, there are fields for 'Username' (containing 'USER1') and 'Password' (containing '.....'). A red circle highlights the 'Basic Auth' selection in the dropdown.

5. Next select the *Headers* tab and under *KEY* use the code assist feature to enter ***Content-Type*** and under *VALUE* use the code assist feature to enter ***application/json***.

The screenshot shows the Postman interface with a POST request to <http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee>. The 'Headers' tab is selected, showing a table with one row. The 'Key' column contains 'Content-Type' with a checked checkbox, and the 'Value' column contains 'application/json'. Both the 'Content-Type' key and the 'application/json' value are circled in red.

**Tech-Tip:** Code assist simply means that when text is entered in field, all the valid values for that field that match the typed text will be displayed. You can select the desired value for the field from the list displayed and that value will populate that field.

6. Next select the *Body* tab and select the *raw* radio button and enter the JSON message below in the *Body* area and press the **Send** button.



The screenshot shows the Postman interface with a yellow box highlighting the JSON code in the Body tab:

```

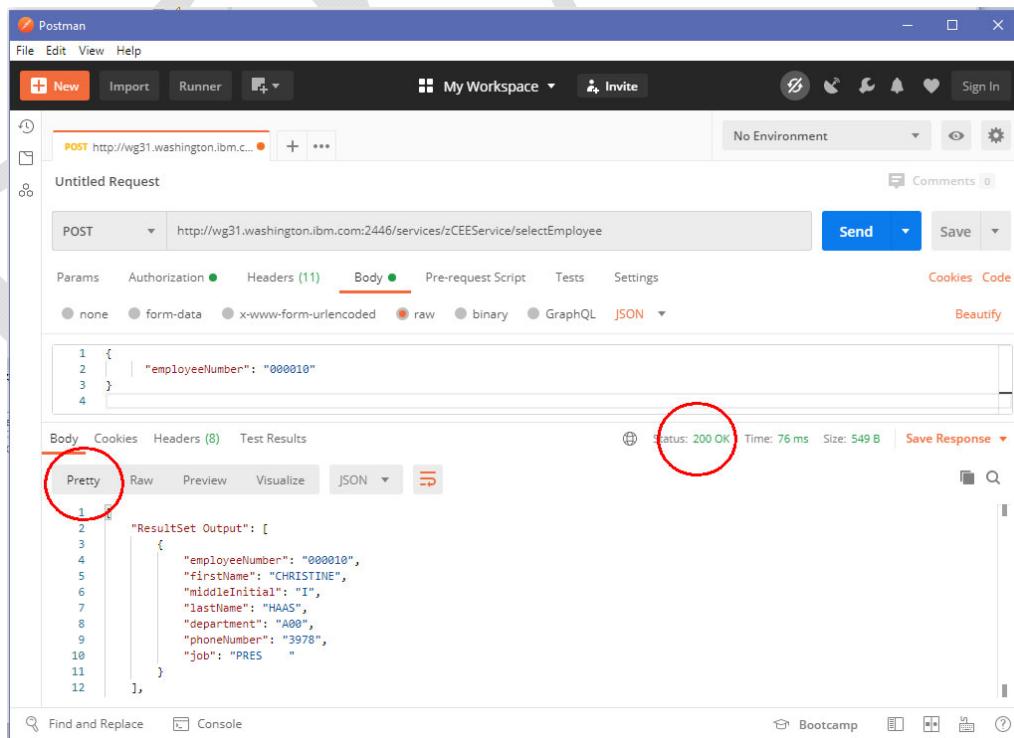
t
"employeeNumber": "000010"
j

```

The Postman interface includes the following details:

- Request Method:** POST
- URL:** http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee
- Headers:** (11) (highlighted in red)
- Body:** (highlighted in red)
- Params:** none
- Authorization:** (green dot)
- Headers (11):** (green dot)
- Body (raw):** (selected)
- Text:** (radio button)
- Code:** (radio button)
- JSON:** (radio button)
- Response:** Placeholder with a rocket icon and text: "Hit Send to get a response".
- Send:** (blue button)
- Save:** (button)
- Comments:** (button)
- Find and Replace:** (button)
- Console:** (button)

7. Pressing the **Send** button invokes the API. The Status of request should be *200 OK* and pressing the *Pretty* tab will display the response



The screenshot shows the Postman interface after sending the request, with a red circle highlighting the **Pretty** tab in the response panel:

**Status:** 200 OK

**Body:**

```

1 {
2   |   "employeeNumber": "000010"
3 }

```

**Pretty** (highlighted with a red circle)

**Raw**

**Preview**

**Visualize**

**JSON**

**ResultSet Output:** [

1, {

2 | "employeeNumber": "000010",

3 | "firstName": "CHRISTINE",

4 | "middleInitial": "I",

5 | "lastName": "HAAS",

6 | "department": "A00",

7 | "phoneNumber": "3978",

8 | "job": "PRES"

9 | },

10 | ],

11 | ]

12 | ]

**Find and Replace:**

**Console:**

8. Submit member DB2REST2 in *USER1.ZCEE.CNTL* for execution. It should complete with a completion code of 0.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//          DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT DD *
DELETE FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN DD *
      DSN SYSTEM(DSN2)

BIND SERVICE("zCEEService") -
  NAME("deleteEmployee") -
  SOLENCODING(1047) -
  DESCRIPTION('Delete an employee from table USER1.EMPLOYEE')
/*

```

This creates a user Db2 native REST Service named *deleteEmployee* that deletes a row from table *USER1.EMPLOYEE* using the same JSON request message used in Step 6. Optionally test this service by using same Postman session with URL

<http://wg31.washington.ibm.com:2446/services/zCEEService/deleteEmployee> and the JSON request message below.

```
t
  "employeeNumber": "000340"
j
```

You should see this result in the response area.

```
{
  "Update Count": 1,
  "StatusCode": 200,
  "StatusDescription": "Execution Successful"
}
```

9. Submit member DB2REST3 in *USER1.ZCEE.CNTL* for execution. It should complete with a completion code of 0.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//          DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT DD *
      SELECT EMPNO AS "employeeNumber", FIRSTNME AS "firstName",
             MIDINIT AS "middleInitial", LASTNAME as "lastName",
             WORKDEPT AS "department", PHONENO AS "phoneNumber",
             JOB AS "job"
      FROM USER1.EMPLOYEE WHERE JOB = :job AND WORKDEPT = :department
//SYSTSIN DD *
      DSN SYSTEM(DSN2)

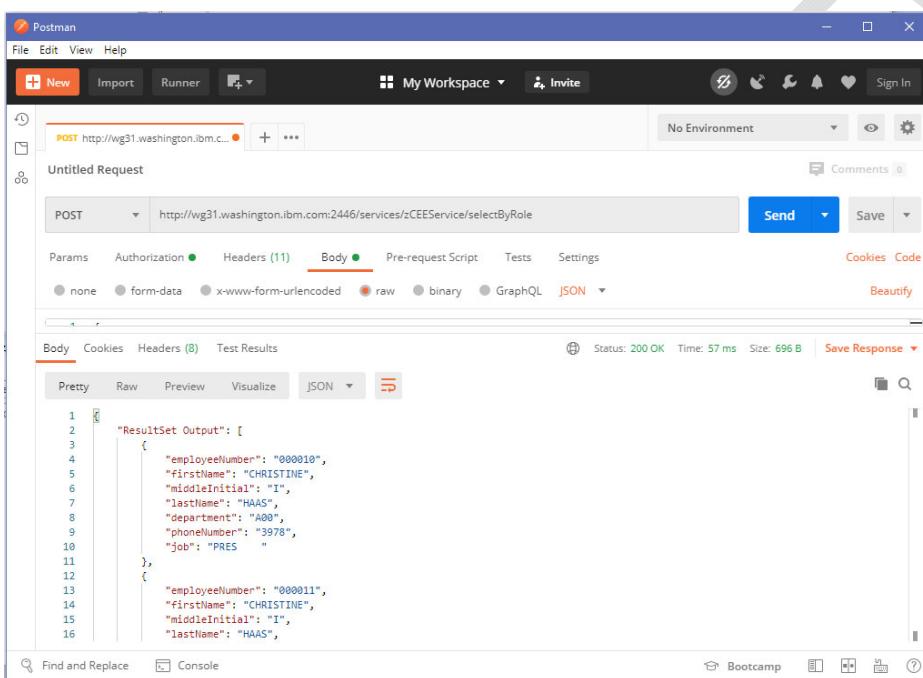
      BIND SERVICE("zCEEService") -
      NAME("selectByRole") -
      SQLENCODING(1047) -
      DESCRIPTION('Select an employee based on job and department')
/*

```

This creates a user Db2 native REST Service named *selectByRole* that selects rows from table *USER1.EMPLOYEE* based on the contents of the *WORKDEPT* and *JOB* columns.

**Important:** The DBA creating this native Db2 REST service is excluding other table columns, e.g. *SEX*, *SALARY*, *BONUS*, *COMMISION*, etc. from the selection by omitting these columns from the *SELECT* statement. The DBA's use of the *AS* cause will also ensure the assigning of meaningful JSON property names rather than the original Db2 column names to the JSON request and response messages.

10. Test this service by using a URL of <http://wg31.washington.ibm.com:2446/services/zCEEService/selectByRole> and a JSON request message of:



The screenshot shows the Postman application interface. A POST request is made to <http://wg31.washington.ibm.com:2446/services/zCEEService/selectByRole>. The request body contains the following JSON:

```

{
  "job": "PRES",
  "department": "A00"
}

```

The response status is 200 OK, with a response time of 57 ms and a size of 696 B. The response body is displayed in JSON format:

```

1   "ResultSet Output": [
2     {
3       "employeeNumber": "000010",
4         "firstName": "CHRISTINE",
5         "middleInitial": "T",
6         "lastName": "HAAS",
7         "department": "A00",
8         "phoneNumber": "3978",
9         "job": "PRES"
10    },
11    {
12      "employeeNumber": "000011",
13        "firstName": "CHRISTINE",
14        "middleInitial": "T",
15        "lastName": "HAAS",
16    }
]

```

11. Submit member DB2REST5 in *USER1.ZCEE.CNTL* for execution. It should complete with a completion code of 0.

```

//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//          DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYOUT=*
//SYSPRINT DD SYOUT=*
//SYSUDUMP DD SYOUT=*
//DSNSTMT DD *
      INSERT INTO USER1.EMPLOYEE
        (EMPNO,FIRSTNAME,MIDINIT,LASTNAME,WORKDEPT,PHONENO,
         HIREDATE,JOB,EDLEVEL,SEX,BIRTHDATE,SALARY,BONUS,COMM)
      VALUES (:employeeNumber, :firstName, :middleInitial, :lastname,
              :department, :phoneNumber, :hireDate, :job,
              :educationLevel, :sex, :birthDate,
              :salary, :bonus, :commission)
//SYSTSIN DD *
DSN SYSTEM(DSN2)
BIND SERVICE("zCEEService") -
NAME("insertEmployee") -
SQLENCODING(1047) -
DESCRIPTION('Insert an employee into table USER1.EMPLOYEE')
/*

```

This creates a user Db2 native REST Service named *insertEmployee* that inserts a new row into table USER1.EMPLOYEE).

**Tech-Tip:** The host variables specified in the VALUES clause will determine the JSON request and response message property names.

12. Test this service by using a URL of

<http://wg31.washington.ibm.com:2446/services/zCEEService/insertEmployee>

```
{  
    "employeeNumber": "999999",  
    "firstName": "Matt",  
    "middleInitial": "T",  
    "lastName": "Johnson",  
    "department": "A00",  
    "phoneNumber": "9999",  
    "hireDate": "2013-01-01",  
    "job": "Staff",  
    "educationLevel": "27",  
    "sex": "M",  
    "birthDate": "1985-06-18",  
    "salary": "100000",  
    "bonus": "15000",  
    "commission": "10000"  
}
```

You should see this result in the response area.

```
{  
    "Update Count": 1,  
    "StatusCode": 200,  
    "StatusDescription": "Execution Successful"  
}
```

13. Next submit job *DB2REST6* to create two new Db2 native services. Db2 native REST service *updateEmployee* updates the SALARY, BONUS and COMM columns in the Db2 table. Db2 native REST service *displayEmployee* will display all the columns of the table (remember Db2 native REST service *selectEmployee* only returns a subset of the columns).

```
//BIND EXEC PGM=IKJEFT01, DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT DD *
  UPDATE USER1.EMPLOYEE
    SET SALARY = :salary, BONUS = :bonus, COMM = :commission
    WHERE EMPNO = :employeeNumber
//SYSTSIN DD *
DSN SYSTEM(DSN2)
BIND SERVICE("zCEEService") -
NAME("updateEmployee") SQLENCODING(1047) -
DESCRIPTION('Insert an employee row into table USER1.EMPLOYEE')
//BIND EXEC PGM=IKJEFT01, DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT DD *
  SELECT * FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN DD *
DSN SYSTEM(DSN2)
BIND SERVICE("zCEEService") -
NAME("displayEmployee") SQLENCODING(1047) -
DESCRIPTION('Display an employee row in table USER1.EMPLOYEE')
('Select an employee from table USER1.EMPLOYEE')
```

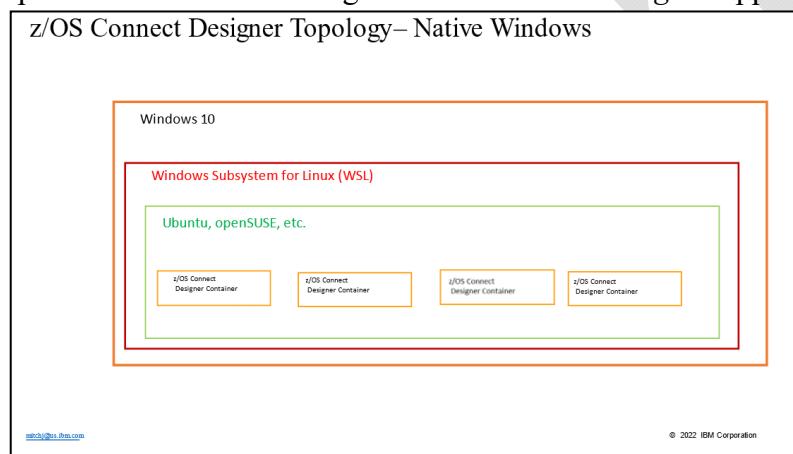
This completes the creation of the Db2 native REST Services that will be used in an API. In the next section, the OpenAPI 3 specification document will be imported into the *z/OS Connect Designer* and used to develop an API.

But before we continue; go back and review the Db2 response messages. Notice that there is a pattern in the response messages from the Db2 REST services. When a Db2 resources is added, updated, or deleted, the response message included an *Update Count* field. The field contains the number of Db2 resources affected by this invoking this service. In the same token, when Db2 resources were retrieved, the Db2 resources were returned in a list or array (e.g., *Resultset Output*) containing one or more list elements. These fields will be used in the *z/OS Connect Designer* to know if a specific REST method was successful or not.

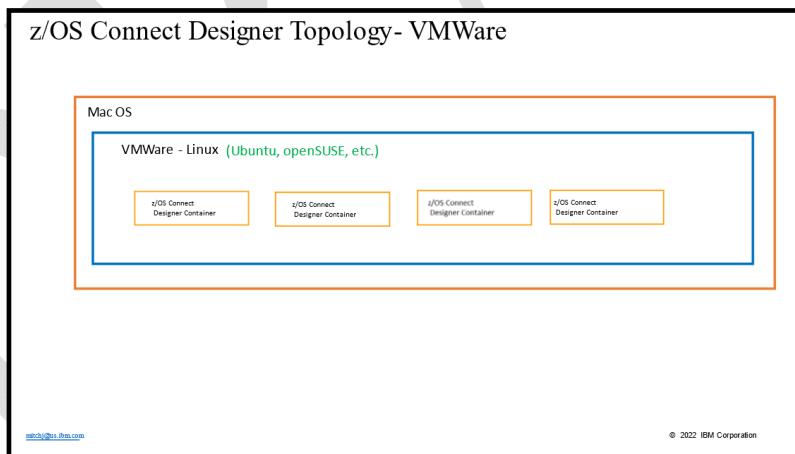
## The z/OS Connect Designer Container environment

APIs for accessing Db2 resources are developed using the *z/OS Connect Designer* which is an application that runs in a Liberty server environment. Each Liberty server with a *z/OS Connect Designer (Designer)* installed can only support the development of a single API. This should not be an issue since these Liberty servers are running in containers and multiple containers can run concurrently on a single image. Currently these containers must be running in a host Linux environment. And again, limiting containers to a Linux host is not a major issue because even if a native Linux environment is not available, current versions of Windows and Mac OS can host Linux host images using either *Windows Subsystems for Linux (WSL)* or virtual images.

For example, consider the diagram below. This diagram is showing a Windows image with *WSL* enabled. The *WSL* environment allows the installation of a Linux distribution (*Ubuntu* in our case), and this *Ubuntu* host image can support multiple container each running the *z/OS Connect Designer* application in a Liberty server.

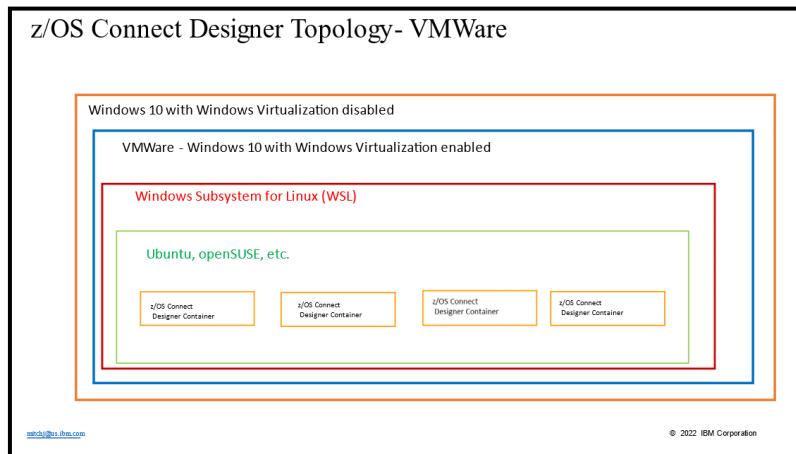


This diagram shows another scenario for a Mac OS image where VMWare Fusion<sup>©</sup> or Parallels<sup>©</sup> can be used to run a virtual Linux image in which where the Liberty *Designer* containers can run.



## IBM z/OS Connect (OpenAPI 3.0)

Another example is shown here where a Windows VMWare© can be used to host a Windows image which has WSL configured in which a host Linux image is running with multiple Liberty *Designer* containers running.



## Setup required Windows Subsystems for Linux support

When running on Windows, the environment for the *Designer* containers required the enablement of *Windows Subsystem for Linux (WSL)*. WSL was configured using the instructions at URL <https://docs.microsoft.com/en-us/windows/wsl/setup/environment>.

The screenshot shows a Microsoft browser window with the URL <https://docs.microsoft.com/en-us/windows/wsl/setup/environment>. The page title is "Set up a WSL development environment". The content includes a "Get started" section with instructions for setting up WSL, mentioning the Windows Subsystem for Linux comes with the Windows operating system but must be enabled and a Linux distribution installed. It also provides instructions for using the simplified --install command.

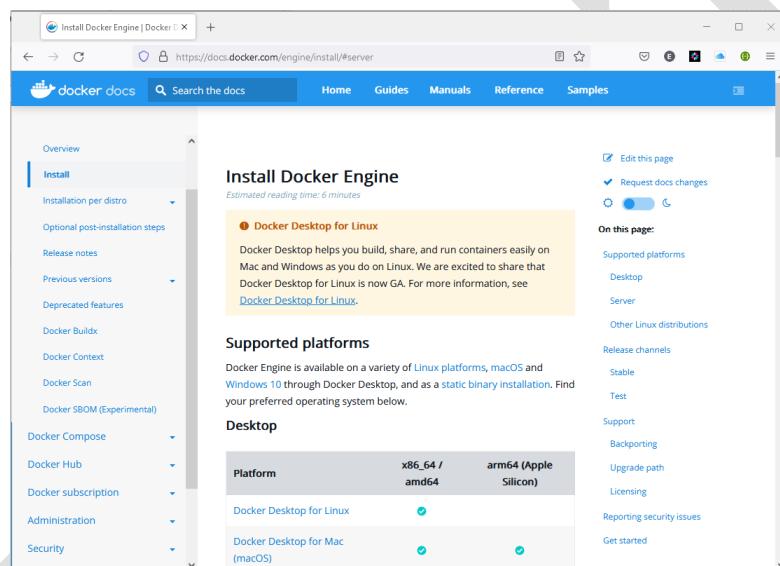
A Linux distribution can be installed following the instructions at URL <https://docs.microsoft.com/en-us/windows/wsl/install>

The screenshot shows a Microsoft browser window with the URL <https://docs.microsoft.com/en-us/windows/wsl/install>. The page title is "Install Linux on Windows with WSL". The content includes a "Prerequisites" section stating you must be running Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11. It also includes a note about checking the Windows version and build number using the Windows logo key + R, type winver, select OK.

Once the host Linux distribution is installed, a container runtime product is required. The z/OS Connect product documentation refers to using *Docker Desktop* and other products. Know that the *Docker Desktop* is a licensed product with yearly cost per license. If the license cost are not issue then by all means, follow the product documentation, and use *Docker Desktop*. But for our purposes, *Docker Desktop* was not an option and z/OS *Connect Designer* containers for this exercise run in containers support by products that do not require licenses, *Docker Engine* and *Podman*.

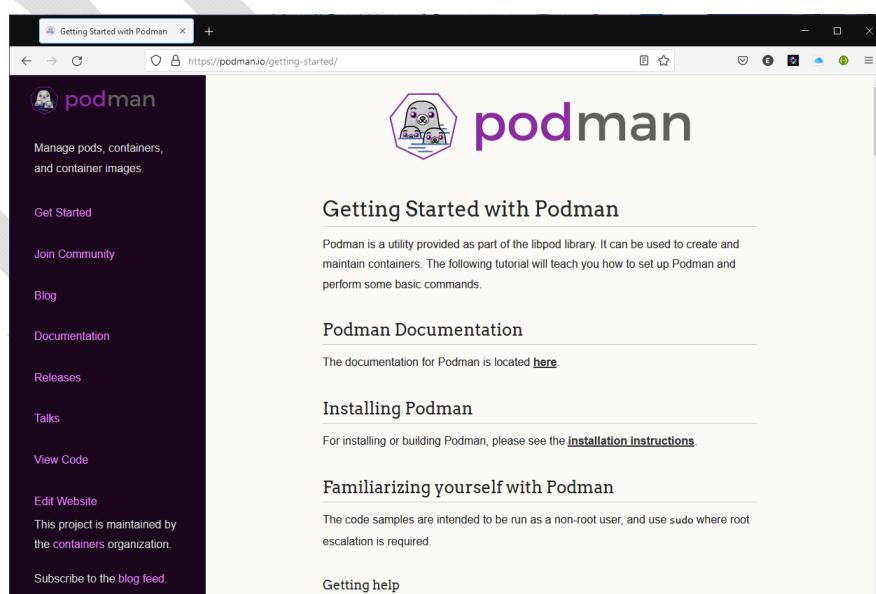
## **Docker Engine**

As stated earlier, Docker Engine was used for this exercise since it does not require a license. Docker Engine was installed the host Linux image using the information at URL <https://docs.docker.com/engine/install/#server>



## **Podman**

An alternative to using *Docker Engine* is to use *Podman* which also not require a license. *Podman* was installed in the host Linux image using the information at URL <https://docs.docker.com/engine/install/#server>



## The container configuration file

Regardless of whether *Docker Engine*, *Docker Desktop*, *Podman* or some other container runtime product is being used, the container's environment required configuration.

First, the container requires that Db2 related environment variable be provided. These variables are used to customize the Db2 related server XML configuration elements. For this exercise, the container was configured with these environment variables set in the *docker-compose.yaml* file (in **bold**).

```
version: "3.2"
services:
  zosConnect:
    image: icr.io/zosconnect/ibm-zcon-designer:3.0.57
    user: root
    environment:
      - CICS_USER=USER1
      - CICS_PASSWORD=USER1
      - CICS_HOST=wg31.washington.ibm.com
      - CICS_PORT=1491
      - DB2_USERNAME=USER1
      - DB2_PASSWORD=USER1
      - DB2_HOST=wg31.washington.ibm.com
      - DB2_PORT=2446
      - HTTP_PORT=9080
    ports:
      - "9449:9443"
      - "9086:9080"
    volumes:
      - ./project:/workspace/project
      - ./logs/:/logs/
      - ./certs:/output/resources/security/
```

Connecting to a Db2 subsystem requires the addition of a *zosconnect\_db2Connection* configuration element to the container's Liberty configuration. And since this API has role-based security elements configured, additional configuration elements for a basic registry and authorization roles are also required. These Liberty configuration elements are described next.

## ***Connecting to Db2 and the required server XML configuration***

The `zosconnect_db2Connection` element used to connect to a Db2 subsystem in this exercise looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="Db2 zosconnect_db2Connection">
  <featureManager>
    <feature>zosconnect:db2-1.0</feature>
  </featureManager>

  <zosconnect_db2Connection id="db2Conn">
    host="${DB2_HOST}"
    port="${DB2_PORT}"
    credentialRef="commonCredentials" />

  <zosconnect_credential id="commonCredentials">
    user="${DB2_USERNAME}"
    password="${DB2_PASSWORD}" />

</server>
```

Notice the environment variables `${DB2_HOST}`, `${DB2_PORT}`, `${DB2_USERNAME}` and `${DB2_PASSWORD}` are set to the values provided in the `docker-compose.yaml` file.

## ***Basic security and the required server XML configuration***

The `basicRegistry` and `authorization-roles` elements used in this exercise looks like this:

```
<server description="basic security">

  <!-- Enable features -->
  <featureManager>
    <feature>appSecurity-2.0</feature>
    <feature>restConnector-2.0</feature>
  </featureManager>

  <webAppSecurity allowFailOverToBasicAuth="true" />

  <basicRegistry id="basic" realm="zosConnect">
    <user name="Fred" password="fredpwd" />
    <user name="user1" password="user1" />
    <user name="user2" password="user2" />
    <group name="Manager">
      <member name="Fred"/>
    </group>
    <group name="Staff">
      <member name="Fred"/>
      <member name="user1"/>
    </group>
  </basicRegistry>

  <administrator-role>
    <group>Manager</group>
  </administrator-role>

  <authorization-roles id="zCERRoles">
    <security-role name="Manager"> <group name="Manager" /> </security-role>
    <security-role name="Staff"> <group name="Staff" /> </security-role>
  </authorization-roles>
</server>
```

In the above configuration, identity *Fred* is a member of the *Manager* and *Staff* group. Identities *USER1* and *USER2* are members of the *Staff* group. Identity *USER2* is not a member of any role-based groups.

The role names *Manager* and *Staff* correspond to the values that appear in the API's specification document . In this example, a default role of *Manager* is defined in the root of the OpenAPI definition. Each of the GET operations defines a role of *Staff*. So only users in or with access to the *Staff* role all allowed to perform the GET methods. And only users in or with access to the *Manager* role all allowed to perform the POST, PUT and DELETE methods. A user with only *Staff* access will receive an HTTP 403 (Forbidden) response if they try to invoke one of these privileged methods.

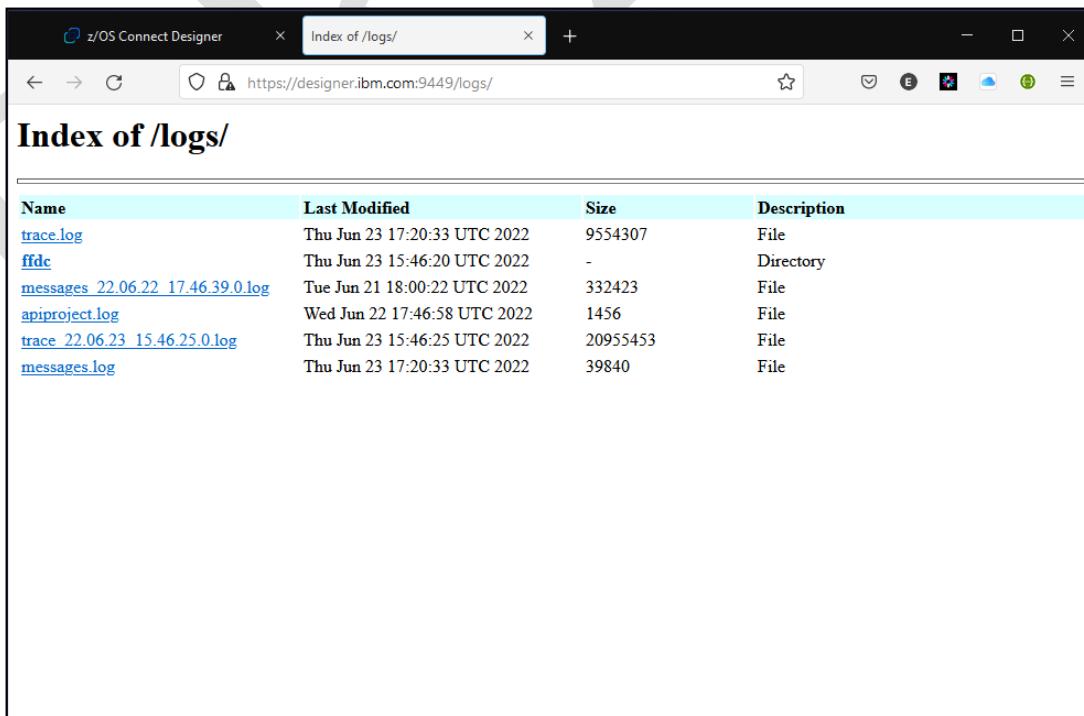
```
openapi: 3.0.0
x-ibm-zcon-roles-allowed:
- Manager
security:
- BasicAuth: []
- BearerAuth: []
paths:
  "/roles/{job}":
    get:
      x-ibm-zcon-roles-allowed:
        - Staff
  /employees:
    post:
  /employees/details/{employee}:
    get:
      x-ibm-zcon-roles-allowed:
        - Staff
  "/employees/{employee}":
    get:
      x-ibm-zcon-roles-allowed:
        - Staff
    delete:
    put:
```

## Accessing the z/OS Connect Designer log and trace files

The Liberty server in which the z/OS Connect Designer has been further customized with the addition of the server XML configuration elements below. These XML configuration elements enables the Liberty server to become a file server.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="Default server">
<webApplication id="resources-dropins" name="dropins"
  location="/opt/ibm/wlp/usr/servers/defaultServer/dropins">
  <web-ext context-root="dropins"
    enable-file-serving="true" enable-directory-browsing="true">
    <file-servering-attribute name="extendDocumentRoot"
      value="/opt/ibm/wlp/usr/servers/defaultServer/dropins" />
  </web-ext>
</webApplication> >
<webApplication id="resources-logs" name="logs"
  location="/logs">
  <web-ext context-root="logs"
    enable-file-serving="true" enable-directory-browsing="true">
    <file-servering-attribute name="extendDocumentRoot"
      value="/logs" />
  </web-ext>
</webApplication> >
</server>
```

This is very useful because this allows the viewing of the server's log and trace file from a browser. This means an API developer using *z/OS Connect Designer* in one tab of browser will be able to monitor the messages and/or traces in other browser tabs as they are developing or testing their API. To access the server's logs directory, start with the same host and port as the *Designer* but with the URI path to */logs*. Double clicking on a file such as *trace.log* or *messages.log* allows the real time monitoring of trace messages or server messages by clicking the browser's refresh button.



For example, using this technique the details of a SQL request and any SQL errors will appear in a *trace.log*. In this case this is information not returned in the response message but written to the trace by the service provider. This is very useful when the expected results are not returned.



A screenshot of a web browser window showing the contents of a trace log at <https://localhost:9449/logs/trace.log>. The browser tabs include 'z/OS Connect Designer', 'OpenAPI UI', and the active tab 'localhost:9449/logs/trace.log'. The page displays a list of log entries from June 21, 2022, at 14:20:05 UTC. The entries are primarily from the 'org.apache.cxf.jaxrs.impl.ResponseImpl@dc04a5de' class, showing various database asset operations like 'db2AssetWrapper Entry', 'constructHeadersObject Entry', and 'setAsset Entry'. One entry specifically handles an SQL error with code -180, which is highlighted with a red box. The log also includes JSON payloads for database operations and service descriptions. At the bottom of the log, there are search and navigation controls: 'Highlight All', 'Match Case', 'Match Diacritics', 'Whole Words', '1 of 34 matches', and 'Reached end of page, continued'.

```

[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > db2AssetWrapper Entry
[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > constructHeadersObject Entry
[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > {}
[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < constructHeadersObject Exit
[6/21/22 14:20:05:750 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < db2AssetWrapper Exit
[6/21/22 14:20:05:750 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < {"headers":
{"connection":"close","Content-Language":"en-US","Content-Length":228,"content-type":"application/json; charset=UTF-8","Date":"Tue, 21 Jun 2022
14:20:08 GMT","Server":"DB2 DDF Native REST, DSNILOC, DSNIJEMG 10/02/19 U165644","X-Correlation-ID":"C0A80067.H41C.DB80633B32BF","X-Powered-
By":"DB2 for z/OS","Content-Type":"application/json; charset=UTF-8"},"body":{"Status":500,"StatusDescription":"Service
zCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-180, SQLSTATE=22007, Message=THE DATE, TIME, OR TIMESTAMP VALUE *N IS
INVALID Error Location:DSNLJXUS:6","cookies":{},"statusCode":500}
[6/21/22 14:20:05:750 UTC] 000007c3 id=9223352b com.ibm.zosconnect.engine.impl.ResponseDataImpl > setAsset Entry
[6/21/22 14:20:05:750 UTC] 000007c3 id=9223352b com.ibm.zosconnect.engine.impl.ResponseDataImpl < setAsset Exit
[6/21/22 14:20:05:750 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < invoke Exit
[6/21/22 14:20:05:750 UTC] 000007c3 id=810e88ae com.ibm.zosconnect.engine.impl.ProviderInvokeOperationImpl < invokeEndpoint Exit
com.ibm.zosconnect.engine.impl.ResponseDataImpl@9223352b > createResponseJson Entry
[6/21/22 14:20:05:750 UTC] 000007c3 id=810e88ae com.ibm.zosconnect.engine.impl.ProviderInvokeOperationImpl > createResponseJson Entry

```

# Managing the z/OS Connect Designer container

## Using Docker commands

The section explores the use of Docker and other commands when creating and administering *Designer* containers. They are provided more reference purposes since their execution is not required to complete the remainder of the exercise. That said, there is one set commands described here that can be used to ‘refresh’ a container or start over if you want to restart the API development process. These commands have been combined in a script file named *refreshDockerContainer*. If anytime you wish to restart, open an Ubuntu terminal shell, and enter command ***refreshDockerContainer employees***. To open an Ubuntu terminal shell, click on the Ubuntu icon (see below) in the Windows task bar.



## Creating a new container

- \_\_\_ 1. Access an *Ubuntu* terminal shell.
- \_\_\_ 2. Make new Linux directory for the container  
**`mkdir /home/workstation/docker/sandbox`**
- \_\_\_ 3. Change location to the new directory  
**`cd /home/workstation/docker/sandbox`**
- \_\_\_ 4. Make a configuration path  
**`mkdir -p project/src/main/liberty/config`**
- \_\_\_ 5. Copy server XML configuration file from the Linux to the container’s config directory  
**`cp /mnt/c/z/openApi3/xml/* project/src/main/liberty/config`**
- \_\_\_ 6. Make the certs and logs subdirectories  
**`mkdir certs`**  
**`mkdir logs`**
- \_\_\_ 7. Copy the base docker-compose.yaml file into current directory  
**`cp /mnt/c/z/openApi3/yaml/docker-compose.yaml .`**
- \_\_\_ 8. Edit docker-compose.yaml file and make the ports unique  
**`sed -i "s/9080:9080/9086:9080/" docker-compose.yaml`**  
**`sed -i "s/9443:9443/9449:9443/" docker-compose.yaml`**
- \_\_\_ 9. Start the container  
**`docker-compose up -d`**
- \_\_\_ 10. Copy server XML override files from Linux into a container’s configuration directory  
**`docker cp /mnt/c/z/openApi3/xml/. sandbox_zosConnect_1:/config/configDropins/overrides`**

**Tech-Tip:** The above commands can be placed in a script, e.g., *createDockerContainer*, see below. Since this script has been placed in directory /home/workstation/bin and this directory has been added to the PATH environment variable, the script can be invoked in any directory by simply entering ***createDockerContainer containerName httpPort httpsPort*** (where *containerName* is the name of the container without the *zosConnect\_1* suffix and *httpPort* and *httpsPort* are the ports to be configured for HTTP and HTTPS connections to the container).

```
echo on
[ -z "$2" ] && HTTP_port=9080 || HTTP_port=$2
[ -z "$3" ] && HTTPS_port=9443 || HTTPS_port=$3
echo creating container "$1"_zosConnect_1 with HTTP_port="$HTTP_port" and
HTTPS_port="$HTTPS_port"
mkdir $containerHome/docker/"$1"
cd $containerHome/docker/"$1"
mkdir certs
mkdir logs
mkdir -p project/src/main/liberty/config
cp /mnt/c/z/openApi3/xml/* project/src/main/liberty/config
cp /mnt/c/z/openApi3/yaml/docker-compose.yaml .
sed -i "s/9080:9080/$HTTP_port:9080/" docker-compose.yaml
sed -i "s/9443:9443/$HTTPS_port:9443/" docker-compose.yaml
docker-compose up -d
docker cp /mnt/c/z/openApi3/xml/. "$1"_zosConnect_1:/config/configDropins/overrides
```

## 11. Display the status of the active containers

***docker ps***

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
0a53e5d3a7af	icr.io/zosconnect/ibm-zcon-designer:3.0.57	"/opt/ibm/helpers/ru..."	About an hour ago	Up About an hour	0.0.0.0:9086->9080/tcp, :::9086->9080/tcp, 0.0.0.0:9449->9443/tcp, :::9449->9443/tcp employees_zosConnect_1

## Refresh an existing container

These set of commands can be used to restart the development of API.

- 1. Stop the container

***docker stop sandbox\_zosConnect\_1***

- 2. Remove the container

***docker container rm sandbox\_zosConnect\_1***

- 3. Change to the host Linux directory of the container

***cd /home/workstation/docker/sandbox***

- 4. Remove the existing *project* directory and all subdirectories (this is required to delete the previous API artifacts.)

***rm -r project/\****

5. Make a new *project* directory and *config* subdirectory

**mkdir -p project/src/main/liberty/config**

6. Copy XML configuration files into the *config* subdirectory.

**cp /mnt/c/z/openApi3/xml/\* project/src/main/liberty/config**

7. Start the container

**docker-compose up -d**

8. Copy the XML configuration files into the container's configuration *overrides* directory

**docker cp /mnt/c/z/openApi3/xml/. sandbox\_zosConnect\_1:/config/configDropins/overrides**

**Tech-Tip:** The above commands can be placed in a script, e.g., *refreshDockerContainer*, see below.  
Since this script has been placed in directory /home/workstation/bin and this directory has been added to the PATH environment variable, the script can be invoked in any directory by simply entering *refreshDockerContainer containerName*

(Where *containerName* is the name of the container without the *zosConnect 1 suffix*).

```
echo refreshing container "$1"_zosConnect_1
docker stop "$1"_zosConnect_1
docker container rm "$1"_zosConnect_1
cd $containerHome/docker/"$1"
rm -r project/*
mkdir -p project/src/main/liberty/config
cp /mnt/c/z/openApi3/xml/* project/src/main/liberty/config
docker-compose up -d
docker cp /mnt/c/z/openApi3/xml/. "$1"_zosConnect_1:/config/configDropins/overrides
```

## Using Podman commands

Note, *Podman* commands are the same as *Docker* commands. Just replace the *docker* command with *podman*.

## Designer problem determination

In this section, we will explore various scenarios using tracing to resolve API development issues, the trace output was created using this trace specification.

```
<logging traceSpecification="zosConnectCics=all:zosConnectDb2=all"/>
```

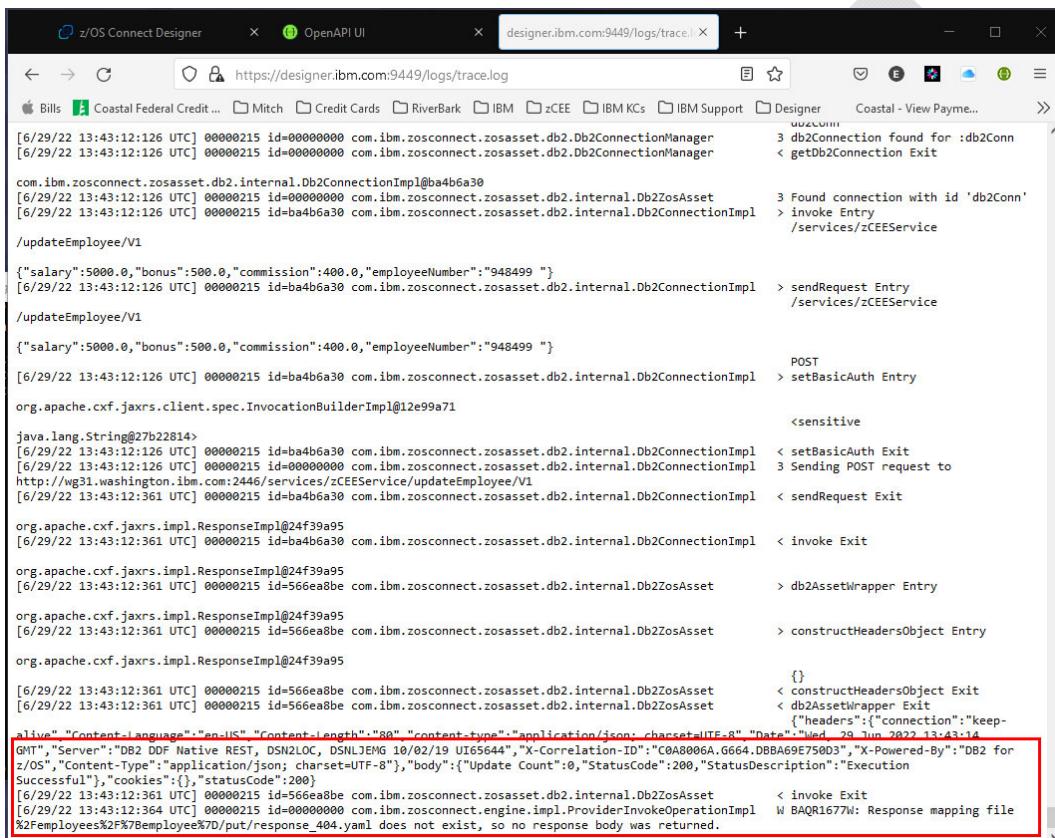
1. Invoking a method returned an HTTP 404 response and no other response. There should have been a message displayed to state that a record for this employee did not exist. A review of the trace showed the Db2 REST service did return an HTTP 404 status code and a status message. The status message indicated that the request Db2 REST service did not exist. The resource not found (HTTP 404) was the REST service, not the employee record.

```

[6/29/22 12:21:08:558 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > sendRequest
Entry
/zCEEService/displayEmployee/V1
{"employeeNumber": "948499"}
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl >
setShouldRebuild Entry
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl <
setShouldRebuild Exit
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > setBasicAuth
Entry
org.apache.cxf.jaxrs.client.spec.InvocationBuilderImpl@20fb0e37
java.lang.String@5eeee633>
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < sensitive
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < setBasicAuth
Exit
[6/29/22 12:21:08:573 UTC] 00000226 id=00000000 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl 3 Sending POST
request to http://wg31.washington.ibm.com:2446/services/zCEEService/displayEmployee/V1
[6/29/22 12:21:08:779 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < sendRequest
Exit
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < invoke Exit
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > db2AssetWrapper
Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset >
constructHeadersObject Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < db2AssetWrapper
Exit
{"headers":
{"connection": "close", "Content-Language": "en-US", "Content-Length": "158", "content-type": "application/json; charset=UTF-8", "Date": "Wed, 29 Jun 2022 12:21:11 GMT", "Server": "DB2 DDF Native REST, DSN2LOC, DSNLJEMG 10/02/19 UI65644", "X-Correlation-ID": "C0A8006A.G64A.DBBA578FD7F8", "X-Powered-By": "DB2 for z/OS", "Content-Type": "application/json; charset=UTF-8"}, "body": {"StatusCode": 404, "StatusDescription": "Service zCEEService.displayEmployee.(V1) execution failed due to the service is undefined. Error Location:DSNLJACC:86"}, "cookies": {}, "statusCode": 404}
[6/29/22 12:21:08:780 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < invoke Exit

```

2. Here is another situation. Invoking a method returned an HTTP 404 response and no other details. There should have been a message displayed to state that a record for this employee did not exist. A review of the trace showed the Db2 REST service did return an HTTP 200 code but not the results array. In this case a review of trace showed the issue was that the response mapping for this situation, *response\_404.yaml*, did not exist in the API. It was this resource not being found which generated the HTTP 404 (not found) condition, not the employee record and not the Db2 REST service.



```

z/OS Connect Designer      OpenAPI UI      designer.ibm.com:9449/logs/trace.log
[6/29/22 13:43:12:126 UTC] 00000215 id=00000000 com.ibm.zosconnect.zosasset.db2.Db2ConnectionManager
[6/29/22 13:43:12:126 UTC] 00000215 id=00000000 com.ibm.zosconnect.zosasset.db2.Db2ConnectionManager
com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl@ba4b6a30
[6/29/22 13:43:12:126 UTC] 00000215 id=00000000 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset
[6/29/22 13:43:12:126 UTC] 00000215 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl
/updateEmployee/V1
{"salary":5000.0,"bonus":500.0,"commission":400.0,"employeeNumber":"948499 "}
[6/29/22 13:43:12:126 UTC] 00000215 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > sendRequest Entry
/services/zEEService
/updateEmployee/V1
{"salary":5000.0,"bonus":500.0,"commission":400.0,"employeeNumber":"948499 "}
[6/29/22 13:43:12:126 UTC] 00000215 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > POST
setBasicAuth Entry
org.apache.cxf.jaxrs.client.spec.InvocationBuilderImpl@12e99a71
<sensitive
java.lang.String@27b22814>
[6/29/22 13:43:12:126 UTC] 00000215 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < setBasicAuth Exit
[6/29/22 13:43:12:126 UTC] 00000215 id=00000000 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl 3 Sending POST request to
http://wg31.washington.ibm.com:2446/services/zEEService/updateEmployee/V1
[6/29/22 13:43:12:361 UTC] 00000215 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < sendRequest Exit
org.apache.cxf.jaxrs.impl.ResponseImpl@24f39a95
[6/29/22 13:43:12:361 UTC] 00000215 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < invoke Exit
org.apache.cxf.jaxrs.impl.ResponseImpl@24f39a95
[6/29/22 13:43:12:361 UTC] 00000215 id=566ea8be com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > db2AssetWrapper Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@24f39a95
[6/29/22 13:43:12:361 UTC] 00000215 id=566ea8be com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > constructHeadersObject Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@24f39a95
[6/29/22 13:43:12:361 UTC] 00000215 id=566ea8be com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < constructHeadersObject Exit
[6/29/22 13:43:12:361 UTC] 00000215 id=566ea8be com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < db2AssetWrapper Exit
[6/29/22 13:43:12:361 UTC] 00000215 id=566ea8be com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset {"headers":{"connection":"keep-alive","Content-Language":"en-US","Content-Length":"30","content-type":"application/json; charset=UTF-8","date":"Wed, 29 Jun 2022 13:43:14 GMT","Server":"DB2 DDF Native REST, DSNI2LOC, DSNIJEMG 10/02/19 11:05:54 AM","X-Correlation-ID":"C0A8006A.G664,DBBA69E750D3","X-Powered-By":"DB2 for z/OS","Content-Type":"application/json; charset=UTF-8"}, "body":{"Update Count":0,"StatusCode":200,"StatusDescription":"Execution Successful"},"cookies":{},"statusCode":200}
[6/29/22 13:43:12:361 UTC] 00000215 id=566ea8be com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < invoke Exit
[6/29/22 13:43:12:364 UTC] 00000215 id=00000000 com.ibm.zosconnect.engine.impl.ProviderInvokeOperationImpl W BAQR1677W: Response mapping file %2F%2Femployees%2F%2Femployee%70/put/response_404.yaml does not exist, so no response body was returned.

```

## Developing a z/OS Connect APIs that accesses Db2

This section of the exercise provides an opportunity to compose and test an API that accesses Db2.

*But before actually starting the exercise, let's do some container housekeeping.*

- Open an Ubuntu terminal shell by clicking on the Ubuntu icon (see below) in the Windows task bar.

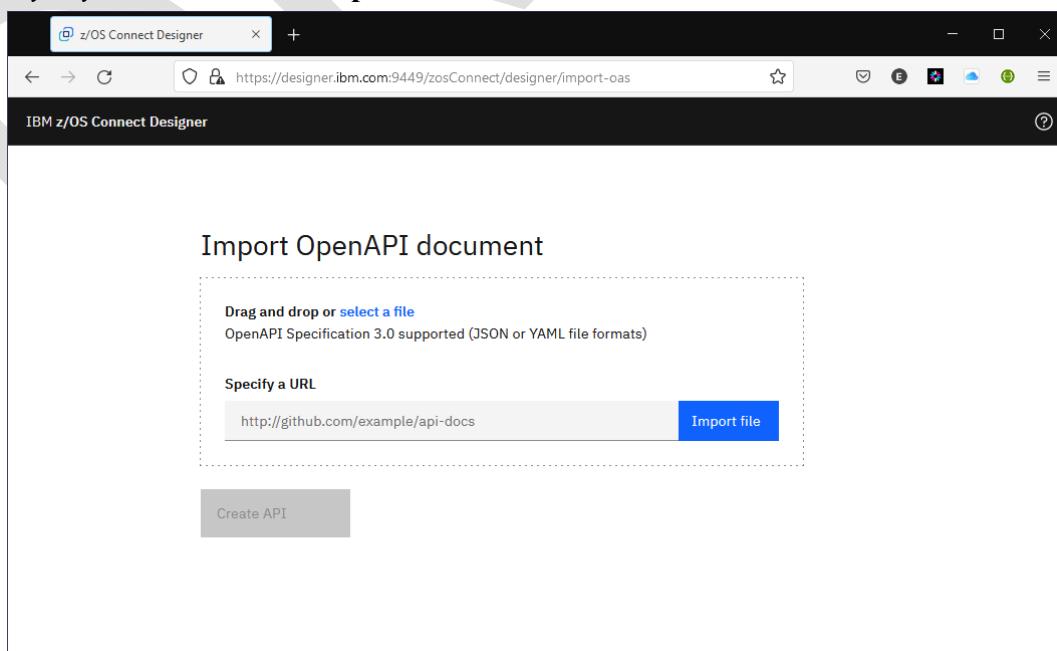


- Switch to root authority by entering command `sudo su` and replying to the password prompt with root's password of `passw0rd`.
- On Windows the Docker daemon needs to be running. Enter command `dockerd &` and wait until you see a message Daemon has completed initialization
- This exercise uses the `employees_zosConnect_1` container. Enter command `docker ps` to see a list of the active containers.
- If this container does not appear in list, enter command `docker start employees_zosConnect_1` to start this container.
- Repeat the `docker ps` command and verify this command now appears in the list of active containers.

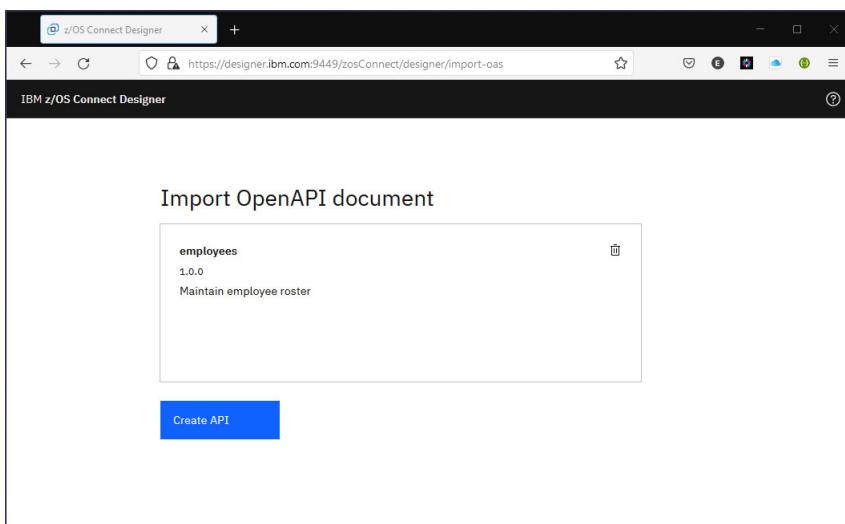
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
27dcfdb8760	icr.io/zosconnect/ibm-zcon-designer:3.0.58	"/opt/ibm/helpers/run..."	11 days ago	Up 2 seconds	0.0.0.0:9086->9080/tcp, :::9086->9080/tcp, 0.0.0.0:9449->9443/tcp, :::9449->9443/tcp

Container housekeeping is now complete.

- Start by opening the Firefox browser and going to URL  
<https://designer.washington.ibm.com:9449/zosConnect/designer/>
- The first window you will see in a 'fresh' Designer environment gives you the opportunity to import an OpenAPI document. On the *Import OpenAPI document* window, click on **select a file** and traverse in the *File Upload* window to the directory where the specification document files are stored, e.g., `C:/z/openApi3/yaml`. Select file `employee.yaml` and click the **Open** button to continue.



3. On the next *Import OpenAPI document* window, click the **Create API** button to complete the importation of the specification document file into the *Designer*.



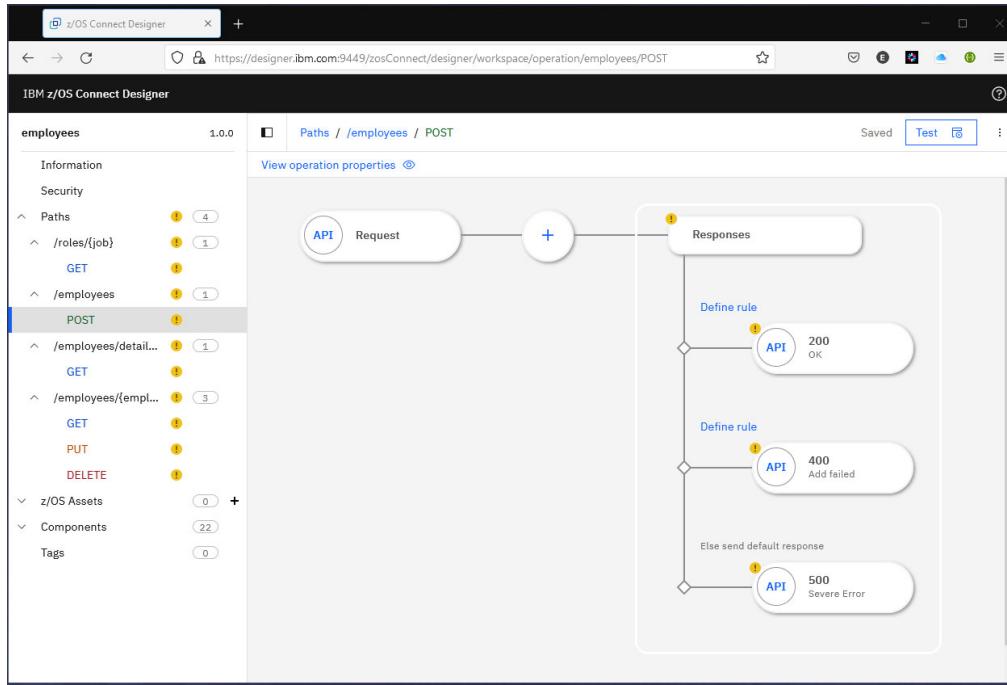
4. The next *Designer* page to be displayed will be the details of the API provided by the specification document. Expand the **Paths** on the left-hand side and you will see the URI paths of the API. Expand the URI paths will display the individual methods of each path. For example, expanding URI paths */employees* and */employees/{employee}* will display the *POST*, *GET*, *PUT* and *DELETE* methods associated with these URI paths (see below).

*Important:* When using this tool, monitor the upper right-hand corner of the page. You will see either status of either **Saved** or **Saving**. It is suggested that you wait until changes are saved before continuing using the Designer.

**Tech-Tip:** The yellow exclamation marks simply indicate the underlying configuration for this element is incomplete. As the exercise progresses, the exclamation marks will disappear.

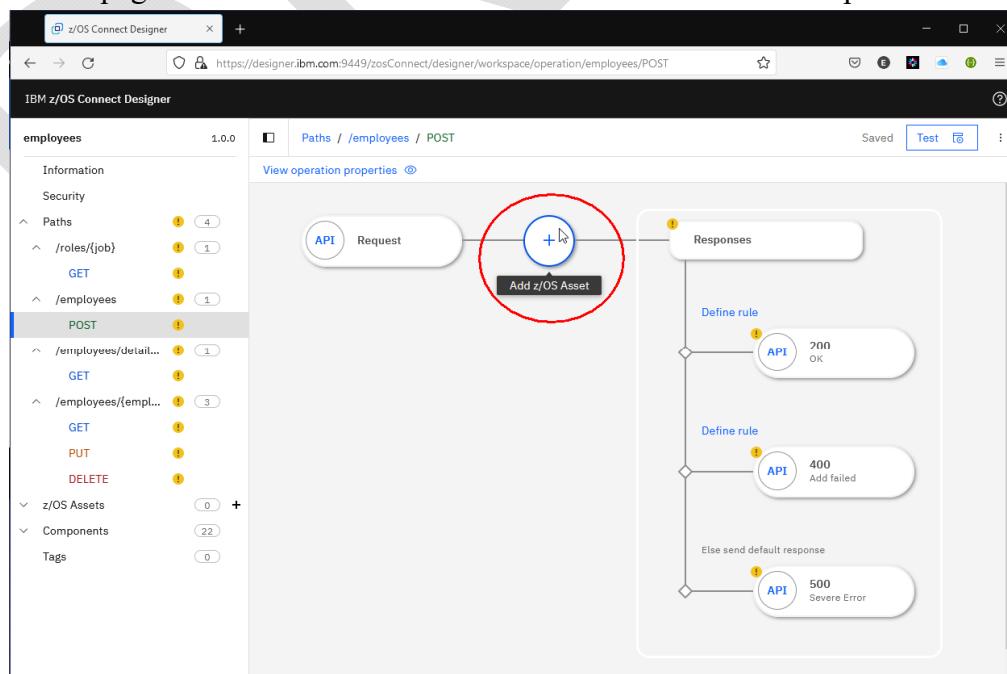
## Configure the POST method for URI path /employees

1. Selecting a method will display the operation properties of the method. Start with the *POST* method under */employees* and by selecting it, the view like the one below will appear.

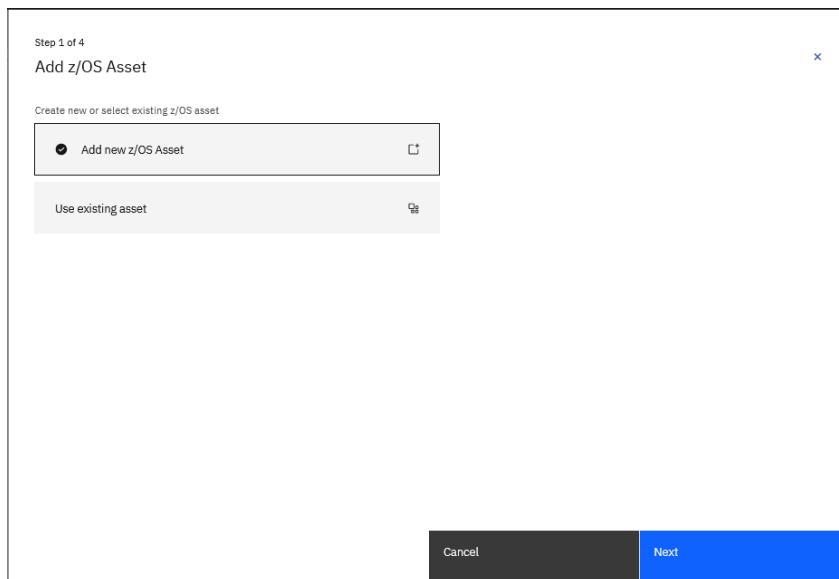


In the example above, we see that the specification document defined 3 responses for this method. One is a 200-status code which indicate the invocation of the method (an insert) was successful. A 400-status code which indicates, in this case, that the request to insert an employee record failed. And finally, a 500-status code which indicates a severe error has occurred while processing the request.

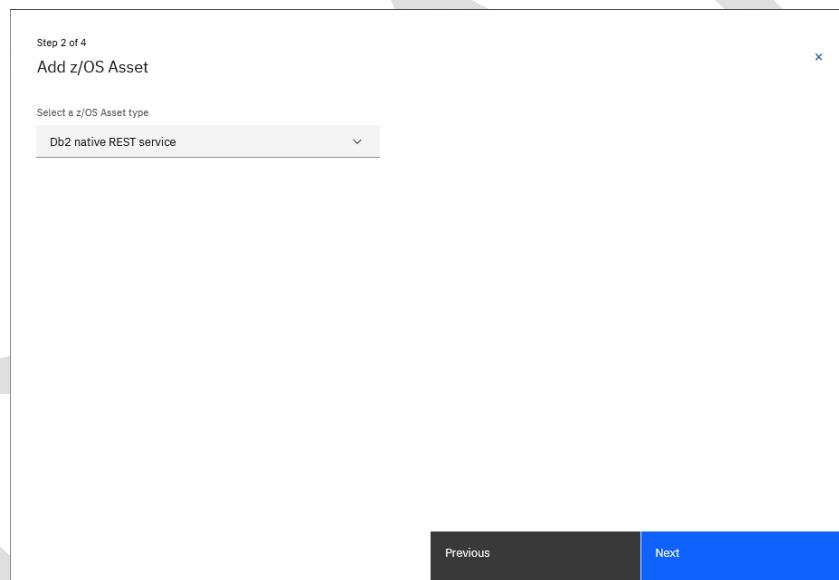
2. The first step in configuring this method for this URI path is to associate it with a z/OS asset or resource. Click the plus sign on the page to start the association of a z/OS asset with this URI path.



3. On the *Add z/OS Asset (Step 1 of 4)* page, select the *Add new z/OS Asset* and press **Next** to continue.



4. On the *Add z/OS Asset (Step 2 of 4)* page, use the pull-down arrow and select *Db2 native REST service* and press **Next** to continue.



5. On the *Add z/OS Asset (Step 3 of 4)* page, use the pull-down arrow and select the *db2conn* Db2 connection. This action will cause the full page to be displayed. Press **Import from Db2 service manager** to access Db2 and to list the available Db2 REST services. Note that the *collection ID*, *service name* and *version* can be used to filter the list.

Step 3 of 4  
Add z/OS Asset

Select a Db2 connection  
db2Conn

**Import from Db2 service manager**

Db2 native REST service collection ID  
e.g. SYSIBMSERVICE

Db2 native REST service name  
e.g. myService

Db2 native REST service version (optional)  
e.g. V1

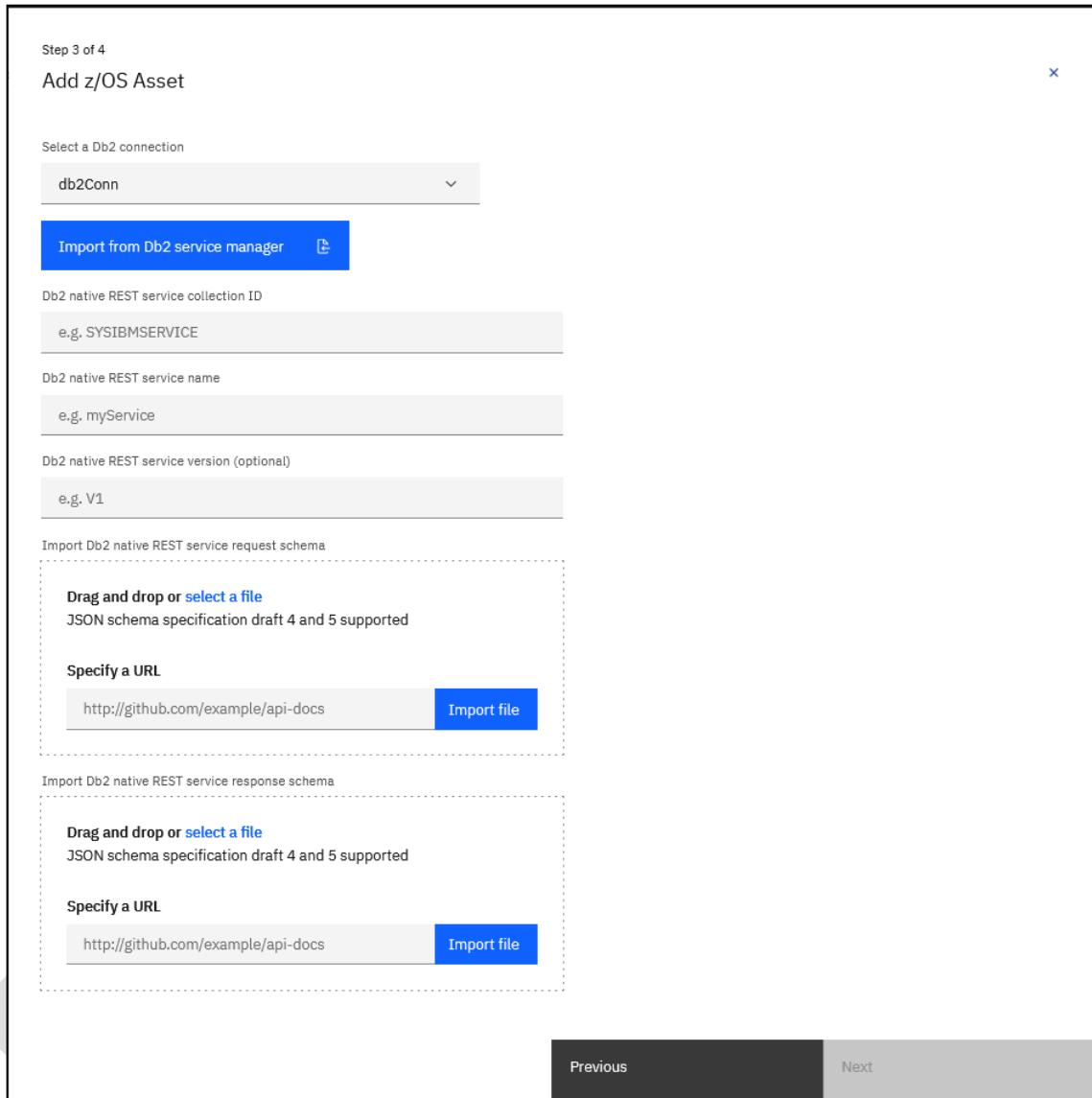
Import Db2 native REST service request schema  
Drag and drop or **select a file**  
JSON schema specification draft 4 and 5 supported

Specify a URL  
 **Import file**

Import Db2 native REST service response schema  
Drag and drop or **select a file**  
JSON schema specification draft 4 and 5 supported

Specify a URL  
 **Import file**

**Previous** **Next**



**Tech-Tip:** The name *db2conn* is the name of the *zosconnect\_db2Connection* configuration element described earlier in this exercise.

6. A list of the available Db2 REST services will be displayed. In this case we want to associate the method with the Db2 REST service *insertEmployee* in the *zCEEService* collection. In this screen shot below, this service is on the first page. If it had not been, there would have been a need to use the page forward arrow at the bottom to go to page 2 of the list to display other pages of the list.

Add z/OS Asset / Import Db2 native REST service

### Import Db2 native REST service

Select a Db2 connection

db2Conn

Service name	Version	Collection ID	Path	Description	Status
addEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Add the details of an ind...	<span style="color: green;">Available</span>
deleteEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Remove the details of a...	<span style="color: green;">Available</span>
getEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of a spec...	<span style="color: green;">Available</span>
getEmployees	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of all em...	<span style="color: green;">Available</span>
updateEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Update the details of an ...	<span style="color: green;">Available</span>
deleteEmployee	V1	zCEEService	/services/zCEEService/d...	Delete an employee fro...	<span style="color: green;">Available</span>
displayEmployee	V1	zCEEService	/services/zCEEService/d...	Display an employee in t...	<span style="color: green;">Available</span>
insertEmployee	V1	zCEEService	/services/zCEEService/i...	Insert an employee into ...	<span style="color: green;">Available</span>
selectByDepartments	V1	zCEEService	/services/zCEEService/s...	Select employees by de...	<span style="color: green;">Available</span>
selectByRole	V1	zCEEService	/services/zCEEService/s...	Select an employee has...	<span style="color: green;">Available</span>

Items per page: 10 ▾ 1–10 of 12 items

1 ▾ of 2 pages < >

Previous Import

7. Select the radio button beside the *insertEmployee* service under *Service Name* and in collection *zCEEService* see below. Press the **Import** button to have the Db2 native REST service information retrieved from Db2.

Service name	Version	Collection ID	Path	Description	Status
addEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Add the details of an ind...	Available
deleteEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Remove the details of a...	Available
getEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of a spec...	Available
getEmployees	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of all em...	Available
updateEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Update the details of an...	Available
deleteEmployee	V1	zCEEService	/services/zCEEService/...	Delete an employee fro...	Available
displayEmployee	V1	zCEEService	/services/zCEEService/...	Display an employee in ...	Available
<b>insertEmployee</b>	V1	zCEEService	/services/zCEEService/i...	Insert an employee into...	Available
selectByDepartments	V1	zCEEService	/services/zCEEService/s...	Select employees by de...	Available
selectByRole	V1	zCEEService	/services/zCEEService/s...	Select an employee bas...	Available

Previous      **Import**

8. This will return you back to the *Add z/OS Asset (Step 3 of 4)* page with the details (request and response schema, etc.) populated from the Db2 service repository. Click **Next** to continue.

Step 3 of 4  
Add z/OS Asset

Select a Db2 connection  
db2Conn

Import from Db2 service manager

Db2 native REST service collection ID  
zCEEService

Db2 native REST service name  
insertEmployee

Db2 native REST service version (optional)  
V1

Import Db2 native REST service request schema

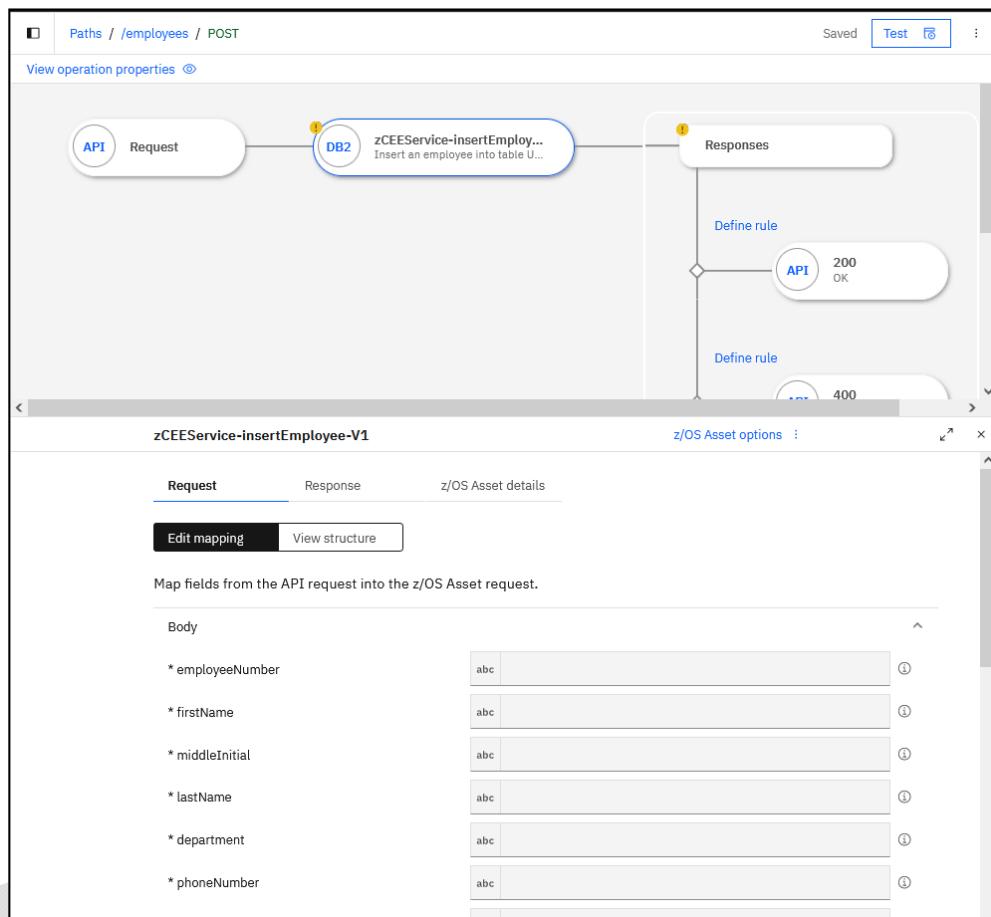
```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "employeeNumber": { "type": [
      "null",
      "string"
    ], "maxLength": 6, "description": "Nullble CHAR(6)" },
    "firstName": { "type": [
      "null",
      "string"
    ]... }
  }
}
```

Import Db2 native REST service response schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "Update Count": { "type": [
      "integer",
      "multipleOf": 1,
      "minimum": 0,
      "maximum": 32767,
      "description": "Update Count" },
    "StatusDescription": { "type": ...
  }
}
```

Previous      **Next**

9. On the *Add z/OS Asset (Step 4 of 4)* page, we are given an opportunity to rename or update the description of the z/OS asset. In this exercise, simply press **Add z/OS Asset** to continue. Eventually you see a brief message that the asset has been added successfully and the operation properties page will reflect the z/OS asset request mapping details (see below). On this page we are seeing the request properties from the Db2 request schema. What needs to be done is to map these fields to the request schema properties of the API as defined in the specification document that described the entire API.



**Note:** It is very important that when working with mapping fields that the field has been properly selected. A properly selection field will be displayed in a blue box as shown below.

10. Now map the Db2 request message fields with the corresponding API request message fields. But first become familiar with the fields in the API request message for this method. To display the API's request message fields, select any container field, and then select the *Insert a mapping tool* (see below).

The screenshot shows the 'zCEEService-insertEmployee-V1' interface with the 'Edit mapping' tab active. Below it, a note says 'Map fields from the API request into the z/OS Asset request.' The 'Body' section contains two fields: '\* employeeNumber' and '\* firstName'. The 'employeeNumber' field has its value set to 'abc'. The 'Insert a mapping tool' icon is circled in red.

11. This will display the header and body mappings available for this method of the API. Become familiar with the mappings available in the body of the request message (you will have to use the scroll bar to see all the available mappings). Knowledge of the mappings in the body will help greatly in the next few steps.

The screenshot shows the 'Available mappings' dialog box. It lists 'headers' and 'body' sections. Under 'body', there is a list of API objects, each with a name and value. The objects listed are: 'Object {}', 'employeeNumber abc', 'firstName abc', 'middleInitial abc', and 'lastName abc'. The entire dialog box is highlighted with a black rectangle.

*There are two ways to map the Db2 request schema with the corresponding specification document API request fields. Both will be demonstrated in this section of the exercise. Use whichever method you prefer when mapping fields later in this exercise.*

12. Start by selecting an empty Db2 request message field and start typing the corresponding API request message field name. For example, entering the string **em** in the area beside *employeeNumber* will eventually match a field in the API request message whose name includes the same characters, and that schema field will be displayed in the drop-down list (see below).

The screenshot shows the 'zCEEService-insertEmployee-V1' interface with the 'Edit mapping' tab active. In the 'Body' section, the 'employeeNumber' field is selected and contains 'abc em'. A red circle highlights the 'employeeNumber' field. A dropdown menu is open over the 'em' part, showing a list of matching API fields. The top item in the list is 'employeeNumber abc'. The 'Insert a mapping tool' icon is also circled in red.

13. Select the field and the area beside the Db2 request schema property name this will cause the API's request message field name to populate the area and complete the mapping.

Body	
* employeeNumber	abc <span style="border: 1px solid #ccc; border-radius: 50%; padding: 2px;">API</span> employeeNumber
* firstName	abc
* middleInitial	abc
* lastName	abc
* department	abc
* phoneNumber	abc

**Tech-Tip:** The icon can be used to maximize or reset this area of the page.

14. The alternate mapping method is to select the *Insert a mapping* tool (see below).

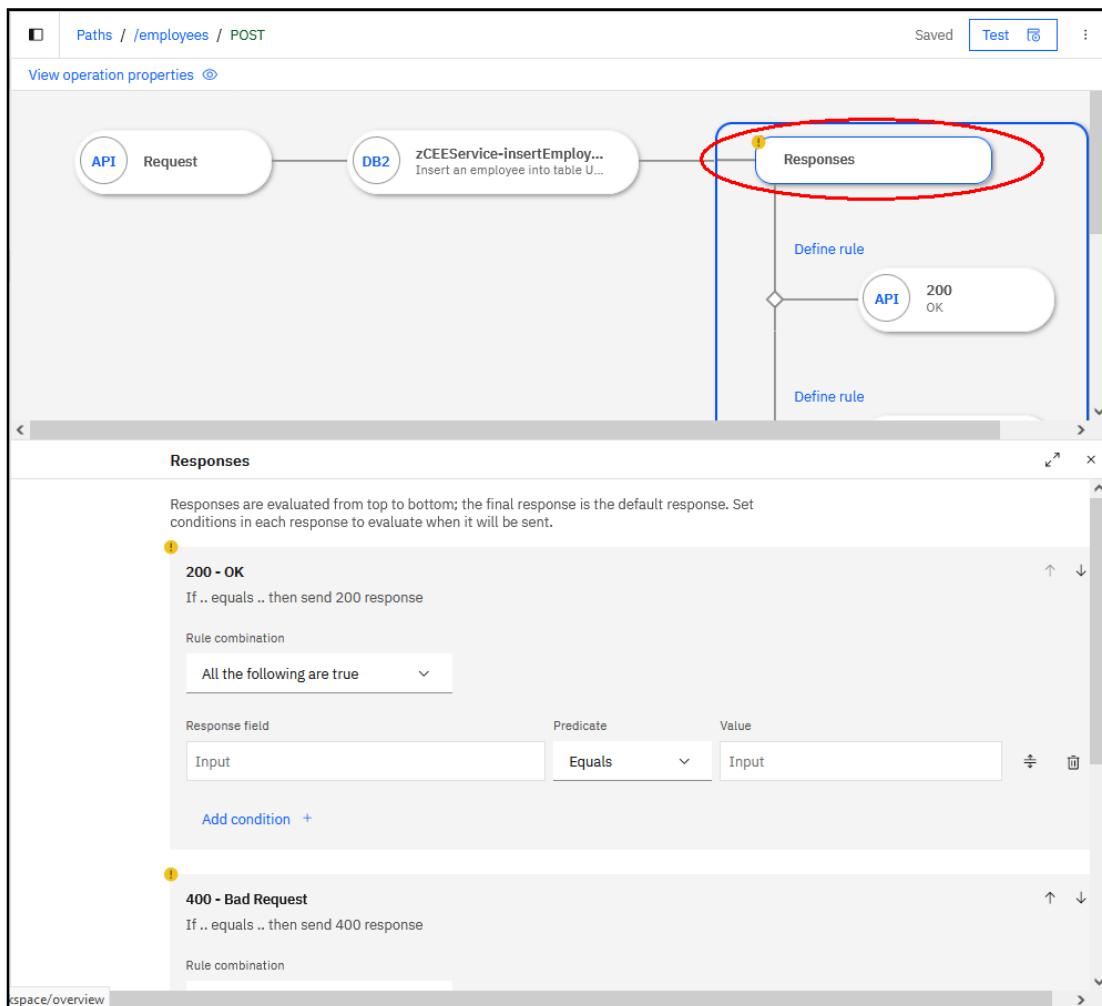
15. This will display a list of available mapping fields. Since this is a request message and the fields are in the request *body*. Scroll up or down and choose appropriate field from the fields from the *body*, not a query parameter, nor a path parameter. In this case, the field to select is the *firstName* field.

16. Use either technique to complete the mappings. When completed, the results should look something like this the page below.

The screenshot shows the 'zCEEService-insertEmployee-V1' mapping interface. It has tabs for Request, Response, and z/OS Asset details, with Request selected. Below the tabs are buttons for Edit mapping and View structure. A note says 'Map fields from the API request into the z/OS Asset request.' The main area is titled 'Body' and lists fields with their mappings:

API Field	z/OS Asset Field
* employeeNumber	abc [API] employeeNumber
* firstName	abc [API] firstName
* middleInitial	abc [API] middleInitial
* lastName	abc [API] lastName
* department	abc [API] departmentCode
* phoneNumber	abc [API] phoneNumber
* hireDate	abc [API] dateOfHire
* job	abc [API] job
* educationLevel	123 [API] educationLevel
* sex	abc [API] sex
* birthDate	abc [API] dateOfBirth
* salary	123 [API] salary
* bonus	123 [API] lastBonus
* commission	123 [API] lastCommission

17. The next step is to provide the configuration to evaluate the responses that come back in the Db2 REST service response message. Select the *Responses* box in the *view operation properties* page.



18. Maximize the *Responses* area of the browser's page (see below).

Responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record was inserted successfully. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was inserted or not. So, we need another way to determine whether a row was really inserted. Fortunately, a Db2 REST service returns another response field, *Update Count*, which we can check the value to see how many rows were affected by this request.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) the value of *Update Count* is set to either 1 or zero.

The screenshot shows the 'Responses' section of the API configuration interface. The top bar indicates the path is 'Paths / /employees / POST'. The 'Responses' section contains three entries:

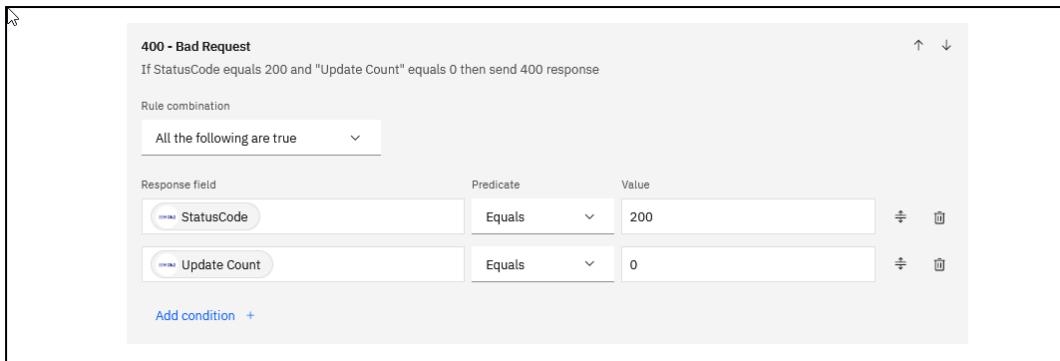
- 200 - OK**: If .. equals .. then send 200 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input. Add condition: +
- 400 - Bad Request**: If .. equals .. then send 400 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input. Add condition: +
- 500 - Internal Server Error**: Else send default response.

19. Under the *200 – OK* response, Enter the string ***Stat*** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter ***200*** in the *Input* field for *Value*.

20. Next add a condition check for the value of *Update Count* by clicking on *Add condition* in the *200 – OK* evaluation.

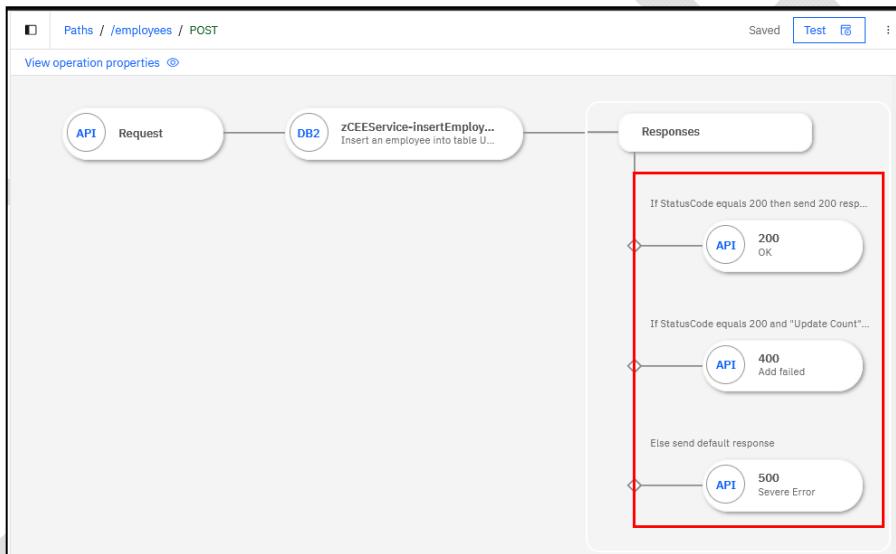
21. Use the same technique described above to add a check for response field *Update Count*. Leave the *Predicate* as *Equals* and set the value to ***1*** below:

22. For the *400 – Bad Request* check, add a check for ***Status Code*** equaling ***200*** and a check for Update Count equaling ***0***



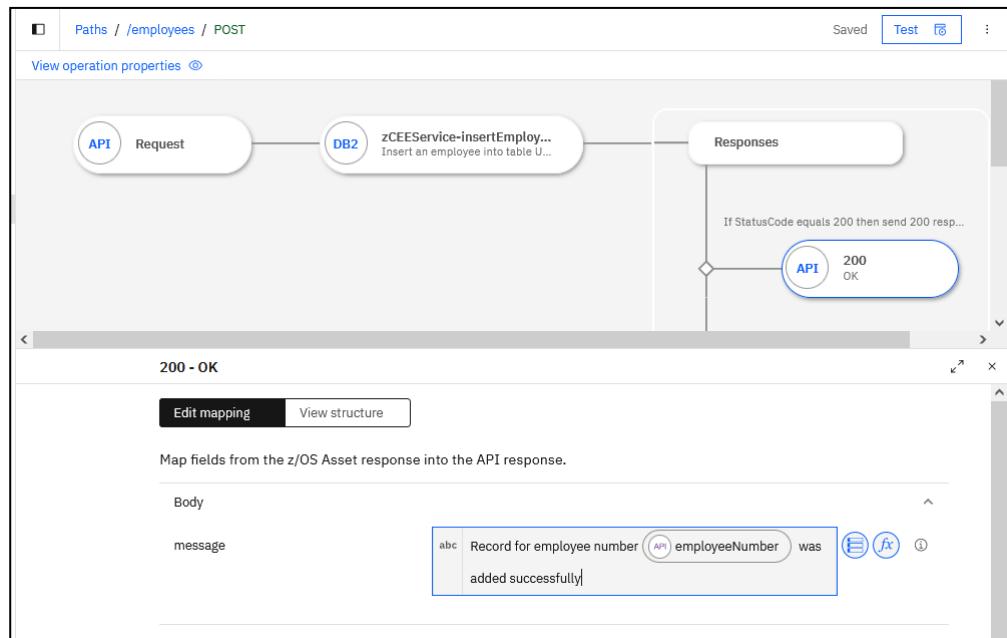
23. If neither of these connections are met, simply return with a HTTP 500 status code.

24. Next the API response messages need to be configured for each of these potential status codes.



25. Select the response for *200 OK* paste the text below in the *message* area.

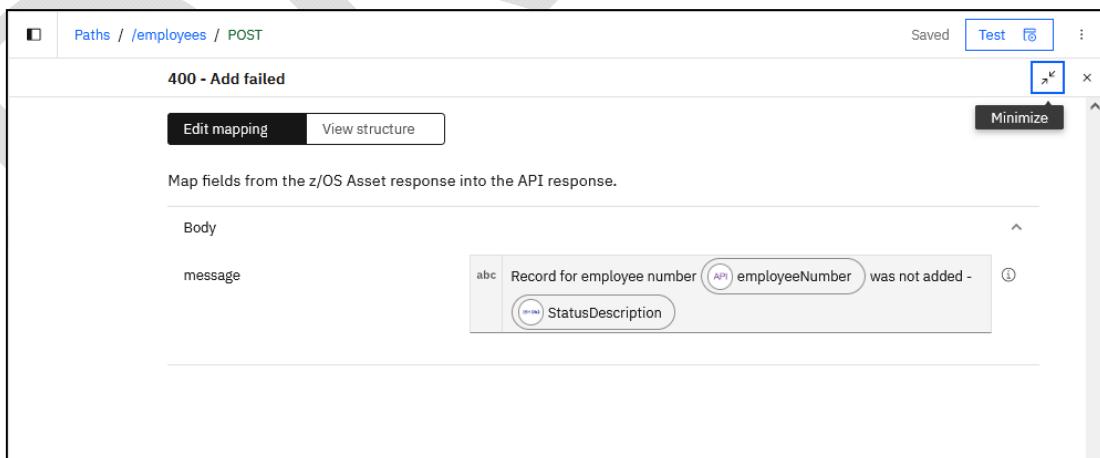
**Record for employee number {{\\$apiRequest.body.employeeNumber}} was added successfully**



The same techniques used to map API response with the Db2 REST request message can be used to insert Db2 REST response files into text like this message which is then subsequently mapped to a field in the API response message. There is flexibility is building complex text strings based on the fields in the Db2 REST response message.

26. Select the response for *400 Add failed* response mapping and paste the text below in the *message* area.

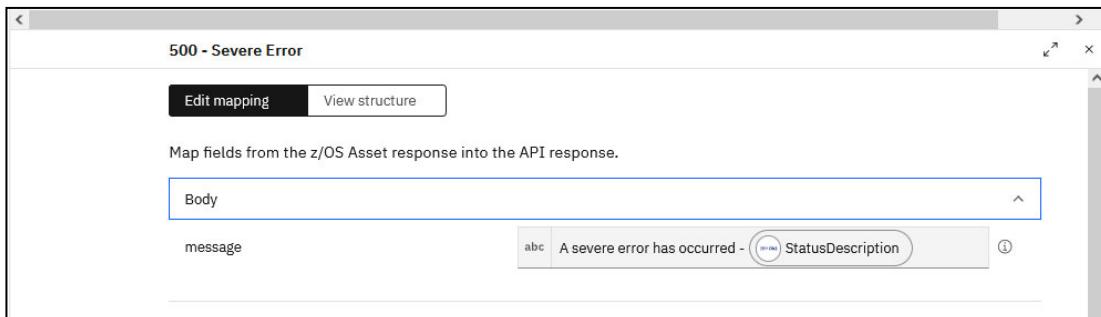
**Record for employee number {{\\$apiRequest.body.employeeNumber}} was not added - {{\\$zosAssetResponse.body.StatusDescription}}**



**Tech-Tip:** Db2 REST response property field *StatusDescription* provides more information regarding the issue that caused the insert to fail.

27. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

**A severe error has occurred - {\$zosAssetResponse.body.StatusDescription}**

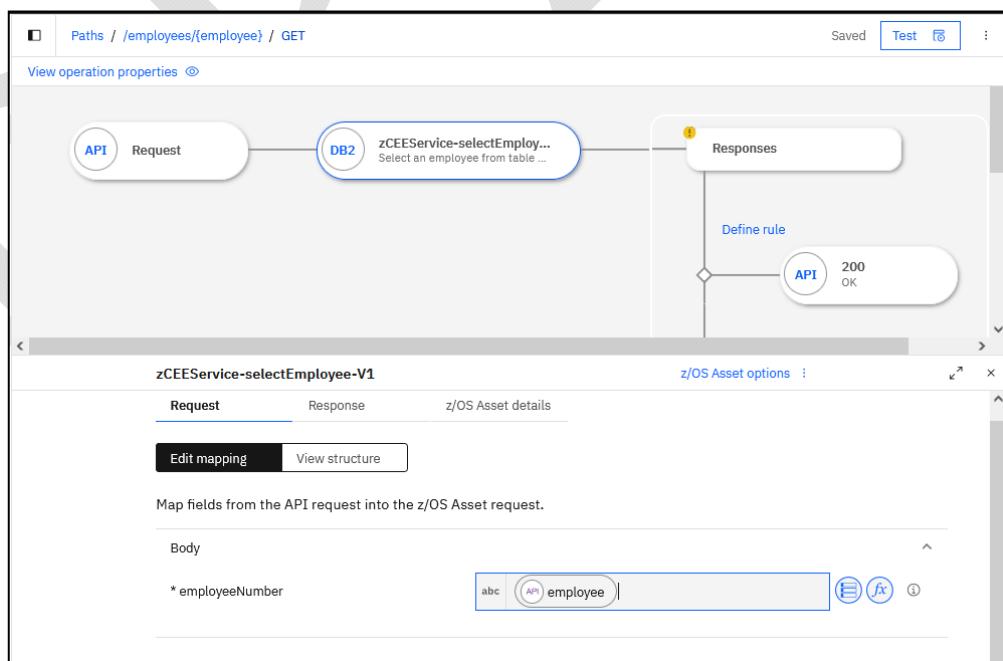


This completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

### Configure the GET method for URI path /employees/{employee}

Now let's repeat the process and complete the configuration for the *GET* method of URI Path */employees/{employee}*

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-selectEmployee* to this method by using the same steps as performed before. Map the path parameter *employee* to the Db2 REST service request message field *employeeNumber* as shown below.



2. Maximize the *Responses* area of the browser's page (see below).

Again, responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record a row or rows were returned as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was selected or not. So, we need another way to determine whether a row was really selected. In this case a Db2 REST service will return the rows selected in a list or array. We are going to take advantage of function that will return the number of elements in the list or array, e.g., \$count. If the result of invoking the function against a list returns a non-zero values, the list or array contains elements. If the result is zero, no elements are in the list and therefore no rows were selected.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and the value of invoking the \$count function against the list of returned rows.

The screenshot shows the 'Responses' configuration for a specific API endpoint. The interface includes a header with 'Paths / /employees/{employee} / GET', 'Saved', 'Test', and a menu icon. The main area is titled 'Responses' with a note: 'Responses are evaluated from top to bottom; the final response is the default response. Set conditions in each response to evaluate when it will be sent.' Below this are three defined rules:

- 200 - OK**: A condition 'If .. equals .. then send 200 response' is set under 'Rule combination' 'All the following are true'. The response field 'Input' is compared to 'Input' using the 'Equals' predicate.
- 404 - Not Found**: A condition 'If .. equals .. then send 404 response' is set under 'Rule combination' 'All the following are true'. The response field 'Input' is compared to 'Input' using the 'Equals' predicate.
- 500 - Internal Server Error**: A condition 'Else send default response' is set.

Under the **200 – OK** response, Enter the string **Stat** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter **200** in the *Input* field for *Value*.

Next add a condition check for the value of invoking the function \$count against the array of rows returned by the Db2 REST service *Count* by clicking on Add condition in the **200 – OK** evaluation and entering the string below in the area for the new check of a Response field.

**\$count(\$zosAssetResponse.body."ResultSet Output")**

And set the *Predicate* to *Is greater than or equal to a Value of 1*.

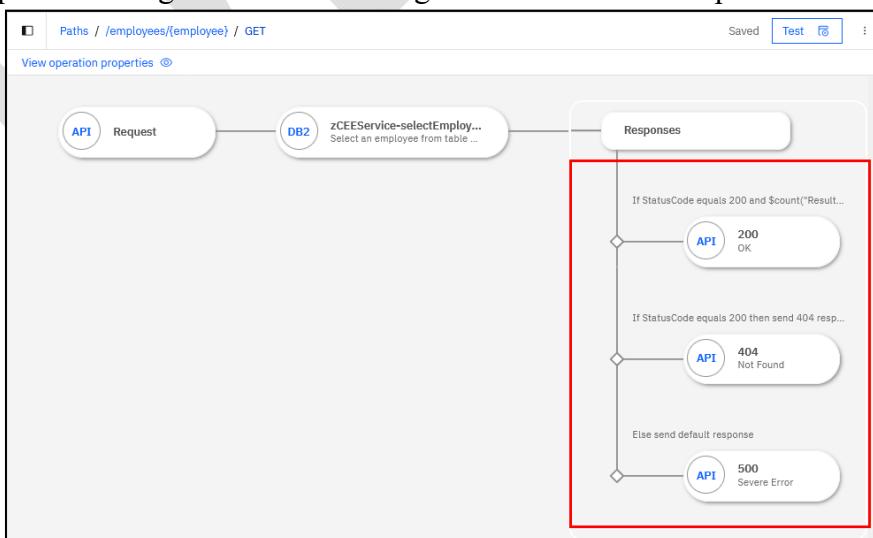
Response field	Predicate	Value
StatusCode	Equals	200
\$count(ResultSet Output)	Is greater than or equal to	1

3. For the **404 – Not Found** check, add a check for *StatusCode* equaling **200** and a check for the count equaling zero.

Response field	Predicate	Value
StatusCode	Equals	200
count(ResultSet Output)	Equals	0

4. If neither of these connections are met, simply return with a HTTP 500 status code.

5. Next the API response messages need to be configured for each of these potential status codes.



6. Select the response for *200 OK* and map the fields from the Db2 REST response message. Start by mapping the *ResultSet Output* field from the Db2 REST response message to the API response field *results Output*. This must be done first to be able to access the elements in the array.

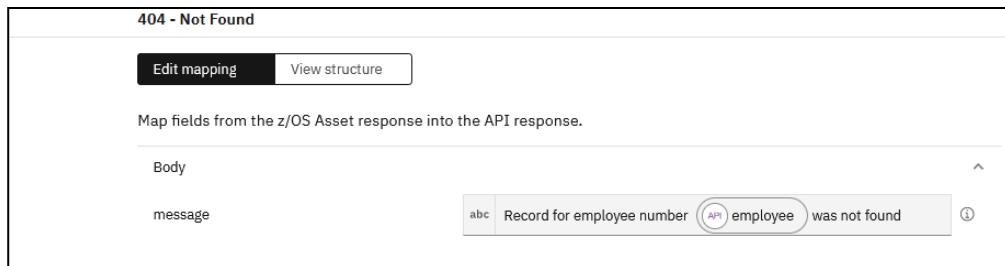
The screenshot shows the 'Body' section of the mapping interface. The 'zosAssetResponse / Object' node has a child node 'ResultSet Output' highlighted with a blue border. A tooltip '[] Res' is displayed above the mapping area. The left sidebar lists properties: employeeNumber, name, departmentCode, phoneNumber, and job.

7. Be careful at this point to ensure you are selecting fields in the *ResultSet output* array in the *body* of the *zosAssetResponse*. The same property name may appear in another one of the available mappings, e.g., *apiRequest*, *ResultSet Row item*, etc. and if a property is selected from one of these mappings, the results will be unpredictable.
8. Complete the mapping for the other properties. Notice the mapping of the Db2 REST response properties *firstName*, *middleInitial* and *lastName* into the single API response property *name*.

The screenshot shows the 'Body' section of the mapping interface. The 'zosAssetResponse / Object' node has a child node 'ResultSet Output' highlighted with a blue border. Other properties like employeeNumber, name, departmentCode, phoneNumber, and job are also mapped. The 'Edit mapping' button is visible at the top left.

9. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

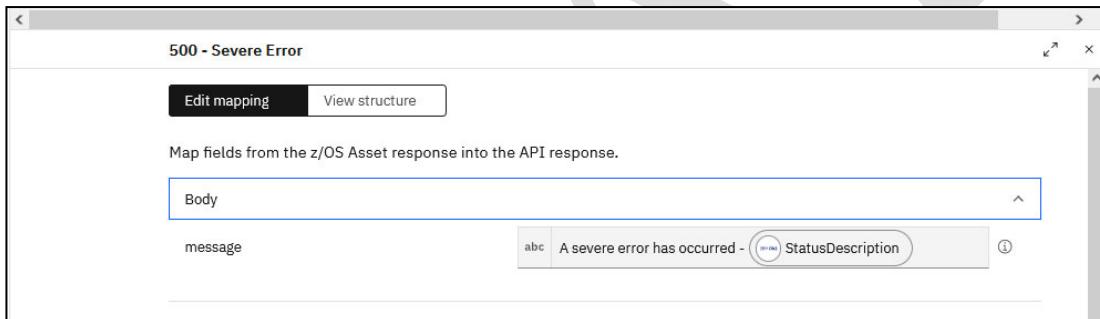
**Record for employee number `>{{$apiRequest.pathParameters.employee}}` was not found**



Notice that the mapping for the property in the message was from the API request message and not the Db2 REST response message.

10. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

**A severe error has occurred -  `{{$zosAssetResponse.body.StatusDescription}}`**



The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

**Tech-Tip:** Note that the exclamation mark has disappeared from *zCEEService-insertEmployee* on the operation page.

## Testing the API's POST and GET methods

As the API was being developed, the changes have been saved and a Web Archive (WAR) file was generated with each change. If the upper right-hand corner of the browser page there will be a **Test** button. Clicking this button will open an API Explorer page. All the URI paths and methods in the original OpenAPI 3 specification document will be displayed, but only the *POST* for */employees* and the *GET* for */employees/{employee}* have been created. Executing one of the other methods will return an HTTP 404 because the components required to execute these methods cannot be found in the WAR.

Let's test what has been developed so far.

1. Click the Test button to open the API Explorer.

The screenshot shows the Open Liberty API Explorer interface. At the top, it says "Liberty REST APIs 1.0.0 OAS3". Below that, it says "Discover REST APIs available within Liberty". Under "Servers", it shows "https://localhost:9449". The main area displays the "employee roster" endpoint. It lists five methods: GET /roles/{job}, POST /employees, GET /employees/details/{employee}, PUT /employees/{employee}, and DELETE /employees/{employee}. The "POST" method is highlighted in green, while the others are in blue, orange, and red respectively. There is also a section for "employeeRos" and a "Schemas" dropdown.

**Tech Tip:** You may be challenged by browser because the digital certificate used by the *Designer* is self-signed Click the **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button. Next you may see a prompt you for a userid and password. If you do see the prompt, enter the username **Fred** and password **frepwd** (case matters) and click **OK**. Remember we are using basic security, and this is the user identity and password defined in the server.xml file.

2. Click on *Post /employees* URI path to display the request body view of the URI path.

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "string",
  "lastName": "string",
  "departmentCode": "string",
  "phoneNumber": "string",
  "dateHire": "string",
  "job": "string",
  "educationLevel": 32767,
  "sex": "string",
  "dateOfBirth": "string",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

3. Next press the **Try it out** button to enable the entry of an authorization string and a request message body

Name	Description
Authorization string (header)	Authorization

Request body required application/json

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "string",
  "lastName": "string",
  "departmentCode": "string",
  "phoneNumber": "string",
  "dateHire": "string",
  "job": "string",
  "educationLevel": 32767,
  "sex": "string",
  "dateOfBirth": "string",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

Servers

These path-level options override the global server options.

/

Execute

4. Enter the JSON request message below in the *Request body* section and press the **Execute** button.

```
{
  "employeeNumber": "948478",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

5. Security was enabled in the original specification document, so you will be required to sign in with one of the identities defined in the basicSecurity.xml file explored earlier. Use **Fred** for the *Username* and **fredpwd** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.
6. Scroll down the view and you should see the *Response body* with the expected successful message.

The screenshot shows the IBM z/OS Connect API interface. At the top, there is a text input field containing a JSON object. Below it is a toolbar with a dropdown menu, a blue 'Execute' button, and a 'Clear' button. The main area is divided into sections: 'Responses' (which is currently empty), 'Curl' (containing a command-line example of the POST request), 'Request URL' (set to <https://localhost:9449/employees>), and 'Server response'. The 'Server response' section shows a status code of 200, a 'Response body' containing the message 'Record for employee number 948478 was added successfully', and a 'Response headers' section with various HTTP headers like Content-Language, Content-Length, Content-Type, Date, X-Firefox-Spdy, and X-Powered-By.

7. Press the **Execute** button again and observe the results. A row for this employee number already existed in the employee roster (a Db2 tables) so the request failed with an HTTP 500.

The screenshot shows the API interface with the following details:

- Curl:**

```
curl -X 'POST' \
  'https://localhost:9449/employees' \
  -H 'accept: application/json' \
  -H 'Content-type: application/json' \
  -d '{
    "employeeNumber": "948478",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0065",
    "dateOfHire": "10/15/1980",
    "job": "Staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
  }'
```
- Request URL:** `https://localhost:9449/employees`
- Server response:**

Code	Details
500	Error: Internal Server Error

**Response body:**

```
{
  "message": "A severe error has occurred - Service xCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-803, SQLSTATE=23505, Message=AN INSERTED OR UPDATED VALUE IS INVALID BECAUSE INDEX IN INDEX SPACE EMPLOYEECA CONSTRAINS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS.          RID OF EXISTING ROW IS X'00000000229'. Error Location:DSNLJXUS:6"
}
```

**Response headers:**

```
content-language: en-US
content-length: 400
content-type: application/json
date: Fri,17 Jun 2022 17:18:19 GMT
x-firebase-spdy: h2
x-powered-by: Servlet/4.0
```

8. Scroll down and click on `GET /employees/{employee}` URI path to display the request body view of the URI path for this method. Next click on the **Try it out** button to enable the entry of data for this method. Enter **948478** as the employee identity and press the **Execute** button to retrieve a subset of data for this employee.

The screenshot shows the API interface with the following details:

- Execute** button is visible at the top.
- Responses** section:
  - Curl:**

```
curl -X 'GET' \
  'https://localhost:9449/employees/948478' \
  -H 'accept: application/json'
```
  - Request URL:** `https://localhost:9449/employees/948478`
- Server response:**

Code	Details
200	Response body

**Response body:**

```
{
  "results output": [
    {
      "employeeNumber": "948478",
      "name": "Matt T Johnson",
      "departmentCode": "C00",
      "phoneNumber": "0065",
      "job": "Staff"
    }
  ]
}
```

**Response headers:**

```
content-language: en-US
content-length: 133
content-type: application/json
date: Fri,17 Jun 2022 17:31:03 GMT
x-firebase-spdy: h2
x-powered-by: Servlet/4.0
```

9. Try this again using number **121212** and observe the results. You see the message that the employee was not found.

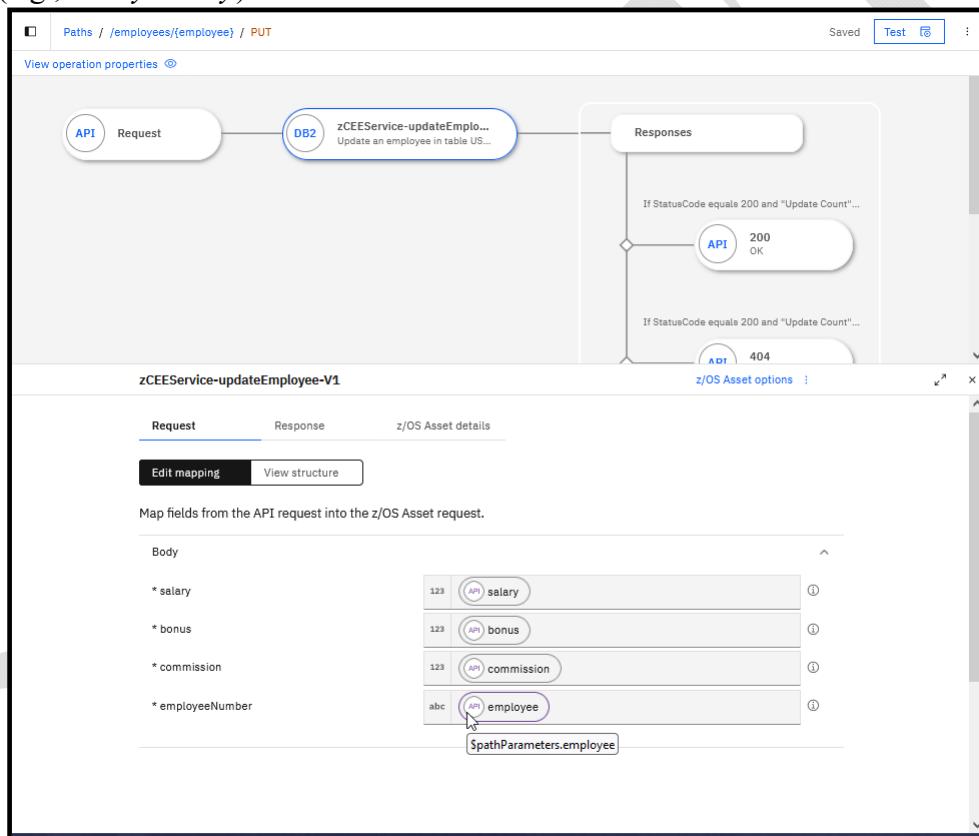
## Compete the configuration of the API (Optional)

To be able to fully test all the URI paths and methods the other methods need to be configured. Otherwise, you may advance to the section *Deploying and Installing APIs in a z/OS Connect Native Server*.

### Configure the PUT method for URI path /employees/{employee}

Now add support for updated a subset of the details of an employee record by completing the configuration for the *PUT* method of URI Path */employees/{employee}*

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-updateEmployee* to this method. Map the API request path parameter field *employee* (*\$pathParameters.employee*) to the DB2 REST request message field *employeeNumber* as shown below. Map *salary*, *bonus*, and *commission* fields from the body of the API request message (e.g., *\$body.salary*)



**Tech Tip:** Hover each of the properties and you see that all the properties except *employee* are mapped to the Db2 REST service from the *body* of the API request message. Property *employee* is mapped to the Db2 REST service from the path parameter.

2. Maximize the *Responses* area of the browser's page (see below).

Responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record was updated as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was updated or not. So, we need another way to determine whether a row was really updated. Again, we will use the *Update Count* response field to check the value to see how many rows were affected by this request.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of *Update Count*.

The screenshot shows the 'Responses' configuration for a PUT endpoint at '/employees/{employee}'. The interface includes a header with 'Saved' and 'Test' buttons, and a sidebar with a 'Paths' section.

**Responses**

Responses are evaluated from top to bottom; the final response is the default response. Set conditions in each response to evaluate when it will be sent.

- 200 - OK**  
If .. equals .. then send 200 response  
Rule combination: All the following are true
 

Response field	Predicate	Value
Input	Equals	Input

[Add condition +](#)
- 404 - Not Found**  
If .. equals .. then send 404 response  
Rule combination: All the following are true
 

Response field	Predicate	Value
Input	Equals	Input

[Add condition +](#)
- 500 - Internal Server Error**  
Else send default response

At the bottom of the responses section, there is a link: [/space/assets/zCEEService-deleteEmployee-V1](#)

3. Under the *200 – OK* response, Enter the string ***Stat*** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, as long as the string entered matches any portion of the field name, that field will be displayed). In this case, select the *Status Code* field. Leave the *Predicate* as *Equals* and enter ***200*** in the *Value* field.

Next add a condition check for the value of the *Update Count* Db2 REST response property by clicking on the *Add condition* in the *200 – OK* evaluation and entering the string below in the area for the new check of a Response field. Enter property ***Update Count*** for the Response field name. Set the *Predicate* to *Is greater than or equal to* and a value of ***1***.

The screenshot shows the configuration interface for a 200-OK response. The rule combination is set to "All the following are true". There are two conditions defined:

Response field	Predicate	Value
Status Code	Equals	200
Update Count	Is greater th...	1

An "Add condition" button is visible at the bottom left.

4. For the *404 – Not Found* check, add a check for *Status Code* equaling ***200*** and a check for *Update Count* equaling zero.

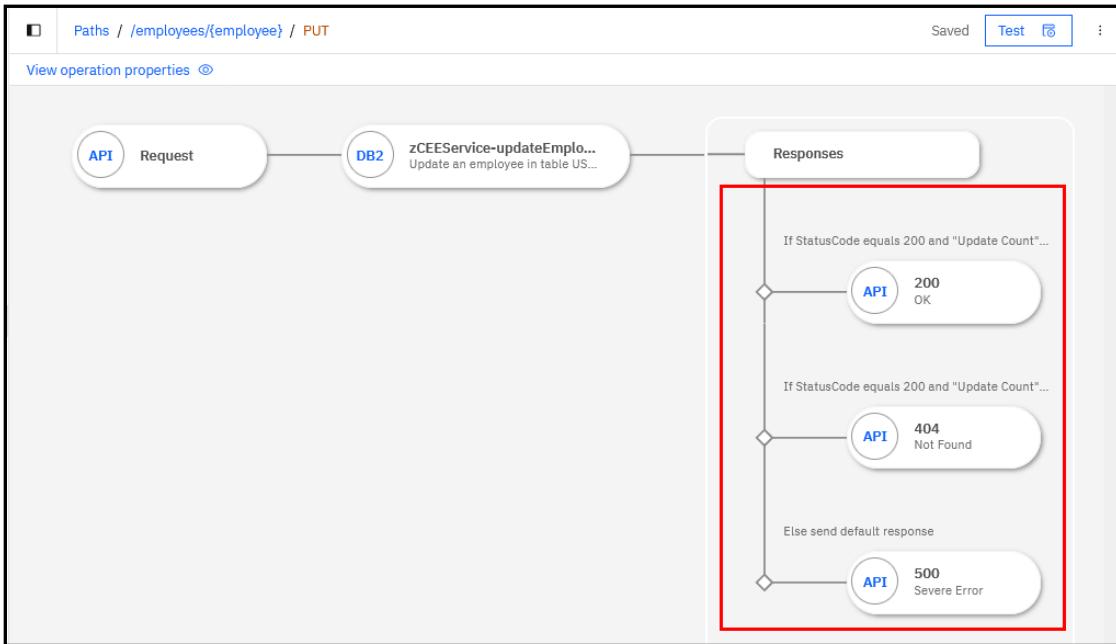
The screenshot shows the configuration interface for a 404-Not Found response. The rule combination is set to "All the following are true". There are two conditions defined:

Response field	Predicate	Value
Status Code	Equals	200
Update Count	Equals	0

An "Add condition" button is visible at the bottom left.

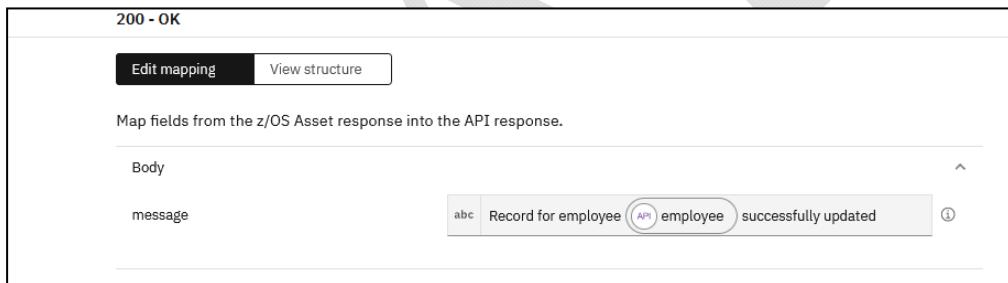
5. If neither of these connections are met, simply return with a HTTP 500 status code.

6. Next the API response messages need to be configured for each of these potential status codes.



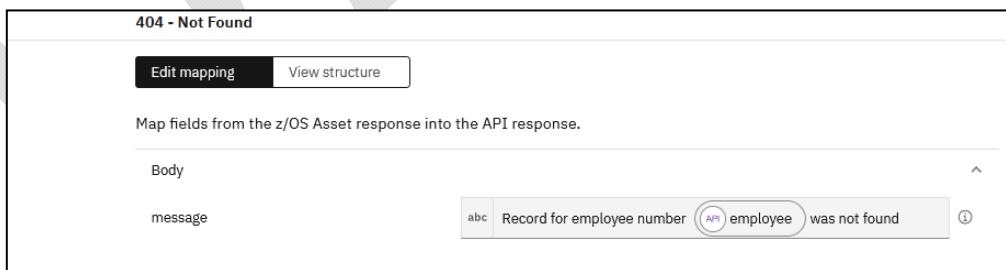
7. Select the response for *200 OK* paste the text below in the *message* area.

**Record for employee {{\\$apiRequest.pathParameters.employee}} successfully updated**



8. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

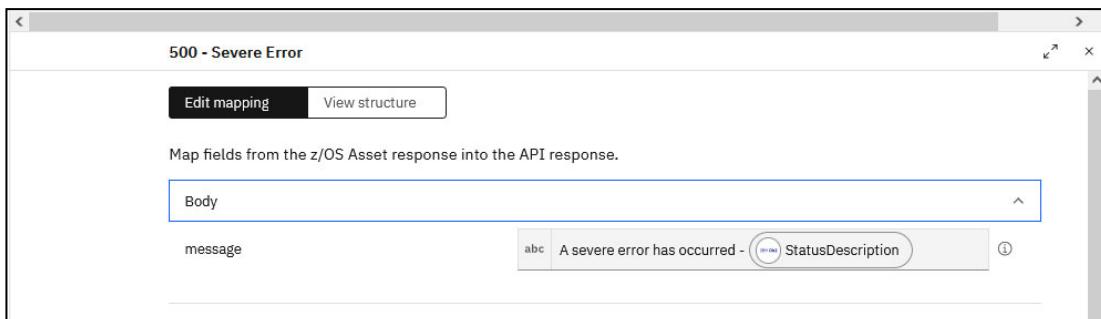
**Record for employee number {{\\$apiRequest.pathParameters.employee}} was not found**



Notice that the mapping for the property in the message was from the API request message and not the Db2 REST response message.

9. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

**A severe error has occurred - {{zosAssetResponse.body.StatusDescription}}**

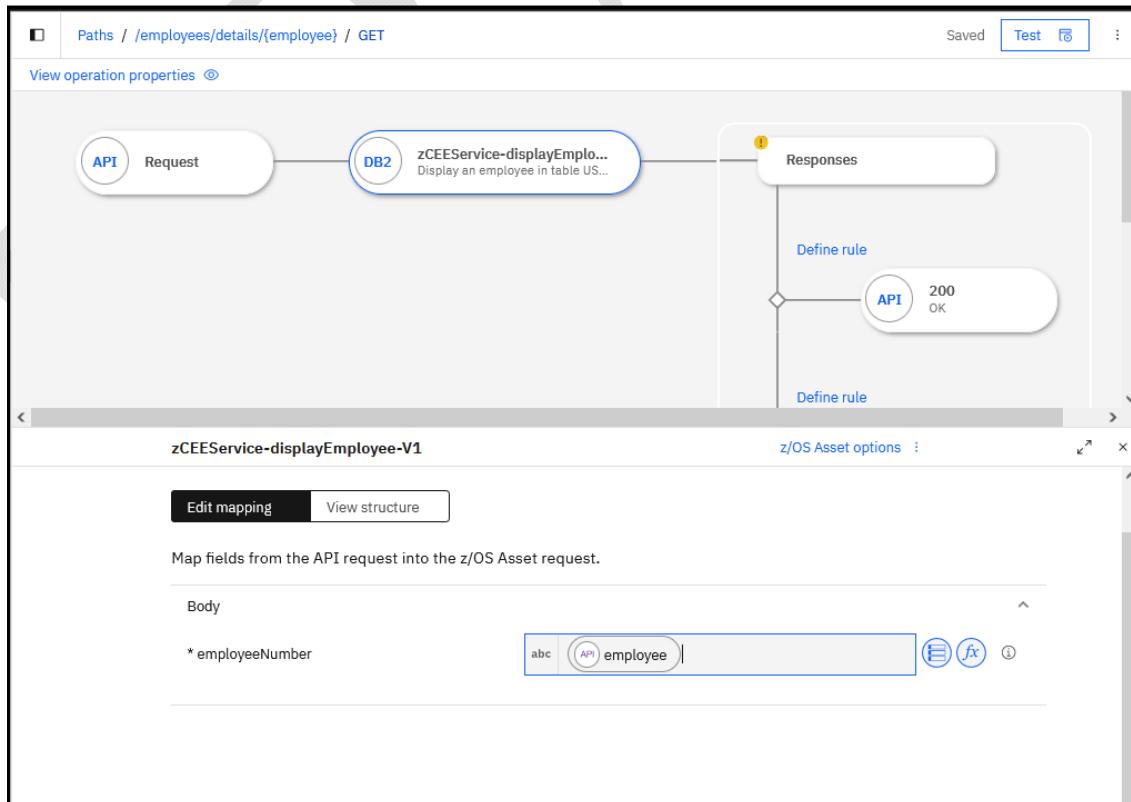


The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

### Configure the GET method for URI path /employees/details/{employee}

Now let's repeat the process and complete the configuration for the *GET* method of URI Path */employees/details/{employee}*

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-displayEmployee* to this method. Map the API request field *employee* to the DB2 REST request message field *employeeNumber* as shown below.



2. Maximize the *Responses* area of the browser's page (see below).

Again, responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record a row or rows were returned as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was selected or not. So, we need another way to determine whether a row was really selected. A Db2 REST service will return the rows selected in a list or array. We are going to take advantage of function that will return the number of elements in the list or array, e.g., \$count. If the result of invoking the function returns a non-zero values, the list or array contains elements. If the result is zero, no rows were selected.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of invoking the \$count function against the array of returned rows.

3. Under the *200 – OK* response, Enter the string ***Stat*** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter ***200*** in the *Input* field for *Value*.

Next add a condition check for the value of invoking the function \$count against the array of rows returned by the Db2 REST service *Count* by clicking on Add condition in the *200 – OK* evaluation and entering the string below in the area for the new check of a Response field.

***\$count(\$zosAssetResponse.body."ResultSet Output")***

And set the *Predicate* to *Is greater than or equal to a Value of 1*.

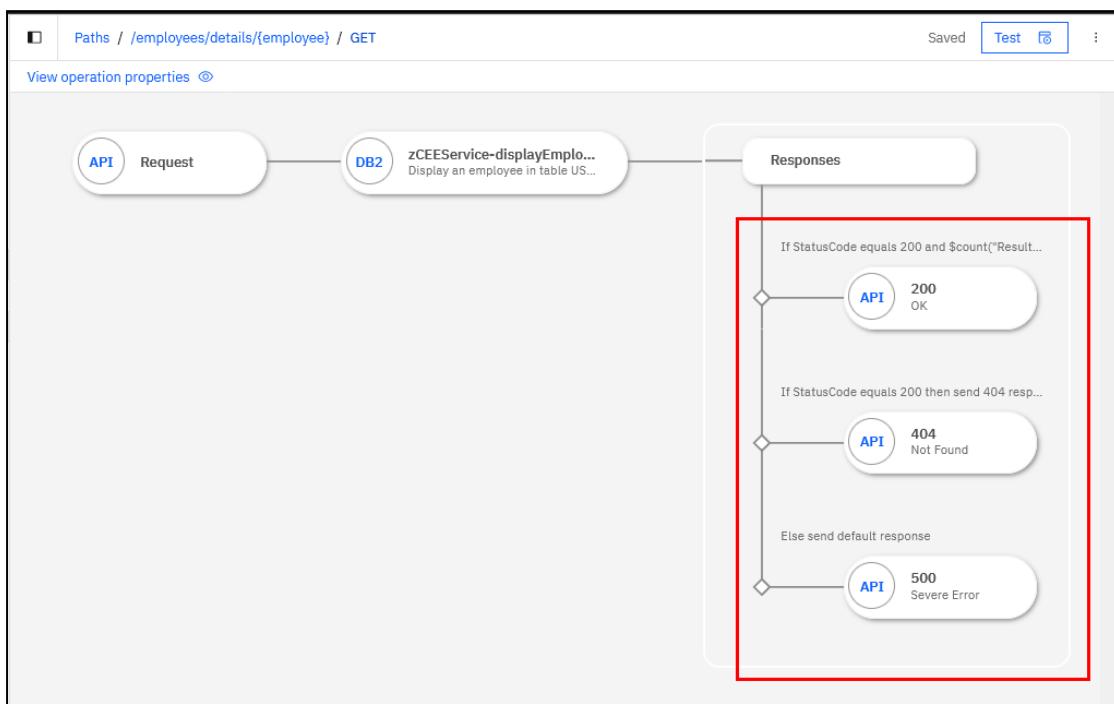
Response field	Predicate	Value
StatusCode	Equals	200
\$count(ResultSet Output)	Is greater than or equal to	1

4. For the *404 – Not Found* check, add a check for *StatusCode* equaling ***200*** and a check for the count equaling zero.

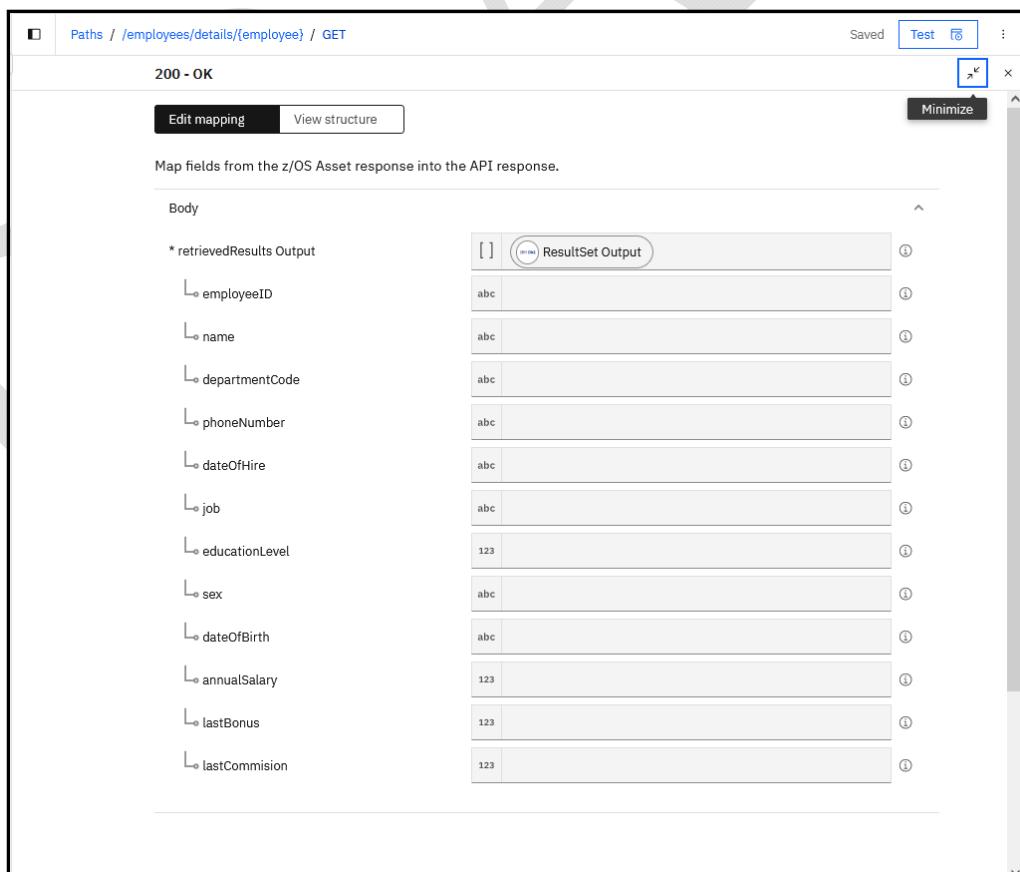
Response field	Predicate	Value
StatusCode	Equals	200
count(ResultSet Output)	Equals	0

5. If neither of these connections are met, simply return with a HTTP 500 status code.

6. Next the API response messages need to be configured for each of these potential status codes.



7. Select the response for **200 OK** and map the fields from the Db2 REST response message. Start by mapping the **ResultSet Output** field from the Db2 REST response message to the API response field **results Output**. This must be done first to access the elements in the array.



8. Be careful at this point to ensure you are selecting fields in the *ResultSet* output array in the *body* of the *zosAssetResponse*. The same property name may appear in another one of the available mappings, e.g., *apiRequest*, *ResultSet Row item*, etc. and if a property is selected from one these mappings, the results will be invalid.
9. Complete the mapping for the other properties. Notice the mapping of the Db2 REST response properties *firstName*, *middleInitial* and *lastName* into the API response property *name*.

The screenshot shows the IBM z/OS Connect API mapping interface. The path is set to `/employees/details/{employee}` and the method is `GET`. The status is `200 - OK`. There are two buttons: `Edit mapping` and `View structure`. A note says: "Map fields from the z/OS Asset response into the API response." The mapping table lists fields from the z/OS asset response and their corresponding API field names:

z/OS Asset Response Field	API Field Name
* retrievedResults Output	[ ] (None) ResultSet Output
└ employeeID	abc (None) EMPNO
└ name	abc (None) FIRSTNME (None) MIDINIT (None) LASTNAME
└ departmentCode	abc (None) WORKDEPT
└ phoneNumber	abc (None) PHONENO
└ dateOfHire	abc (None) HIREDATE
└ job	abc (None) JOB
└ educationLevel	123 (None) EDLEVEL
└ sex	abc (None) SEX
└ dateOfBirth	abc (None) BIRTHDATE
└ annualSalary	123 (None) SALARY
└ lastBonus	123 (None) BONUS
└ lastCommission	123 (None) COMM   <span style="border: 1px solid blue; padding: 2px;">Edit</span> <span style="border: 1px solid blue; padding: 2px;">Fix</span>

10. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

**Record for employee number `>{{$apiRequest.pathParameters.employee}}` was not found**

Notice that the mapping for the property in the message was from the API request message and not the Db2 REST service response message.

11. Finally in the *500 – Severe Error* response mapping paste the following in the area for the *message* property

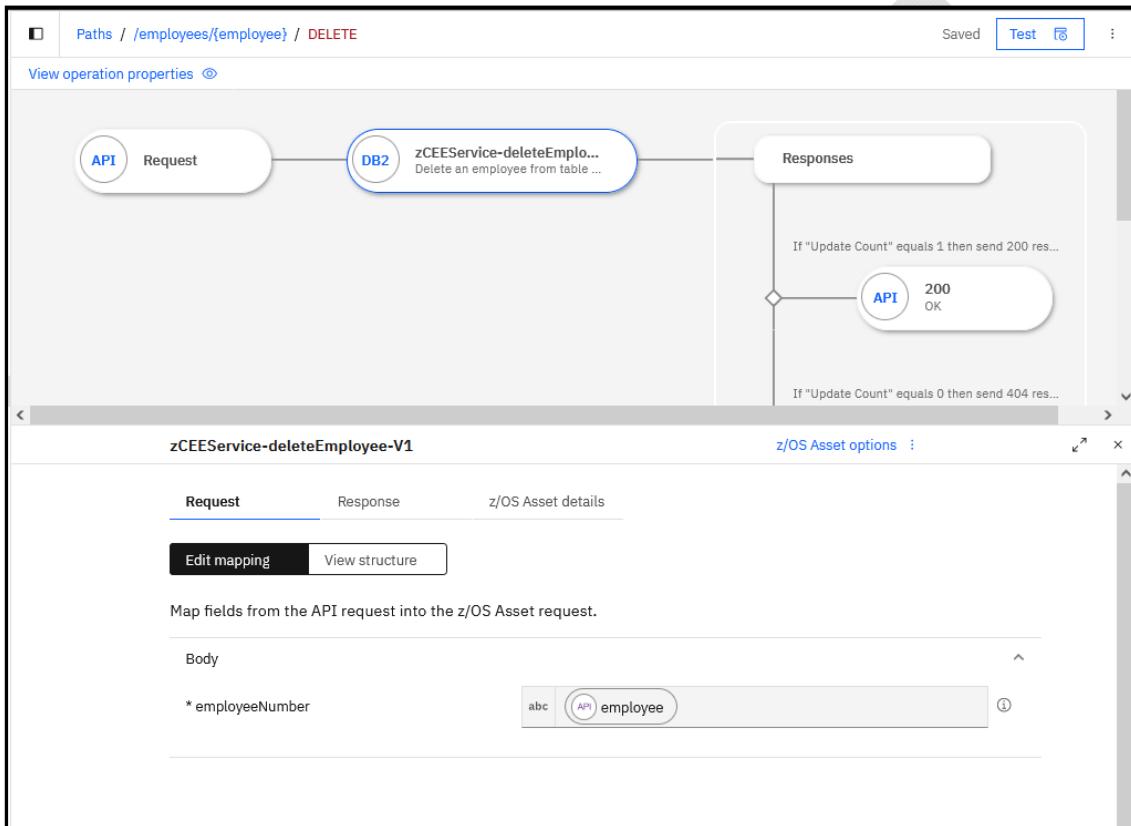
**A severe error has occurred - `{{$zosAssetResponse.body.StatusDescription}}`**

The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

## Configure the **DELETE** method for URI path **/employees/{employee}**

Now let's repeat the process and complete the configuration for the **DELETE** method of URI Path **/employees/{employee}**

1. Start by adding a new z/OS Asset for Db2 REST service **zCEEService-deleteEmployee** to this method. Map the API request field **employee** to the DB2 REST request message field **employeeNumber** as shown below.



2. Maximize the *Responses* area of the browser's page (see below).

Responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record was updated as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was updated or not. So, we need another indication whether a row was really updated. Again, we will use the *Update Count* response field to check the value to see how many rows were affected by this request.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of *Update Count*.

3. Under the *200 – OK* response, Enter the string ***Stat*** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter ***200*** in the *Value* field for *Value*.

Next add a condition check for the value of the *Update Count* Db2 REST response property by clicking on the *Add condition* in the *200 – OK* evaluation and entering the string below in the area for the new check of a Response field. Enter property ***Update Count*** for the Response field name. Set the *Predicate* to *Is greater than or equal to* and a value of ***1***.

200 - OK  
If StatusCode equals 200 and "Update Count" is greater than or equal to 1 then send 200 response

Rule combination  
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Is greater th...	1

Add condition +

4. For the *404 – Not Found* check, add a check for *StatusCode* equaling ***200*** and a check for an update count equaling zero.

404 - Not Found  
If StatusCode equals 200 and "Update Count" equals 0 then send 404 response

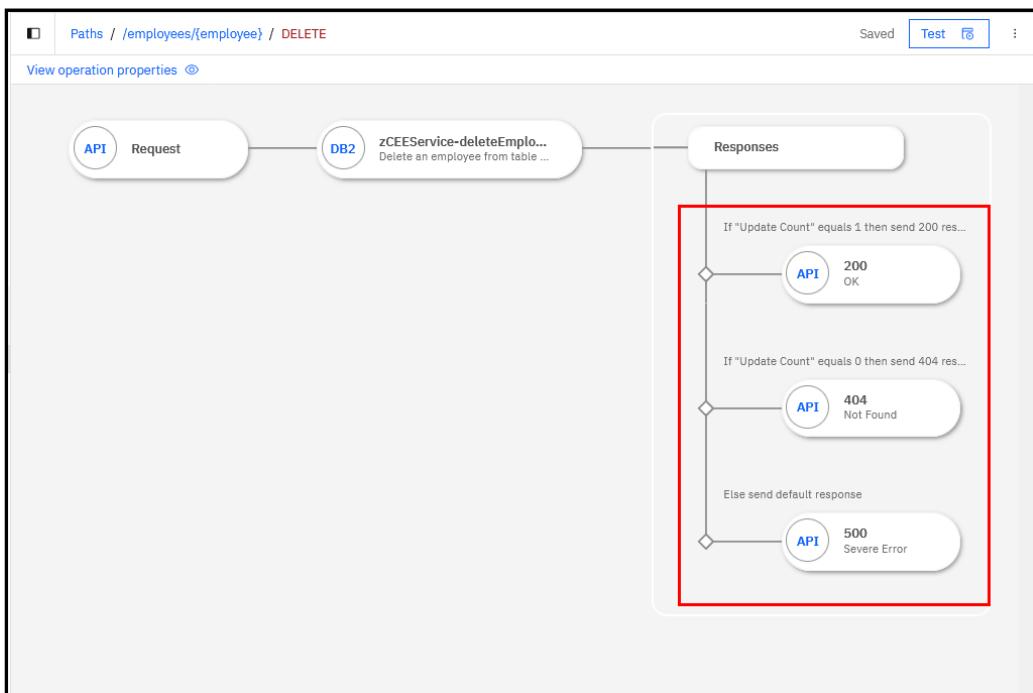
Rule combination  
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Equals	0

Add condition +

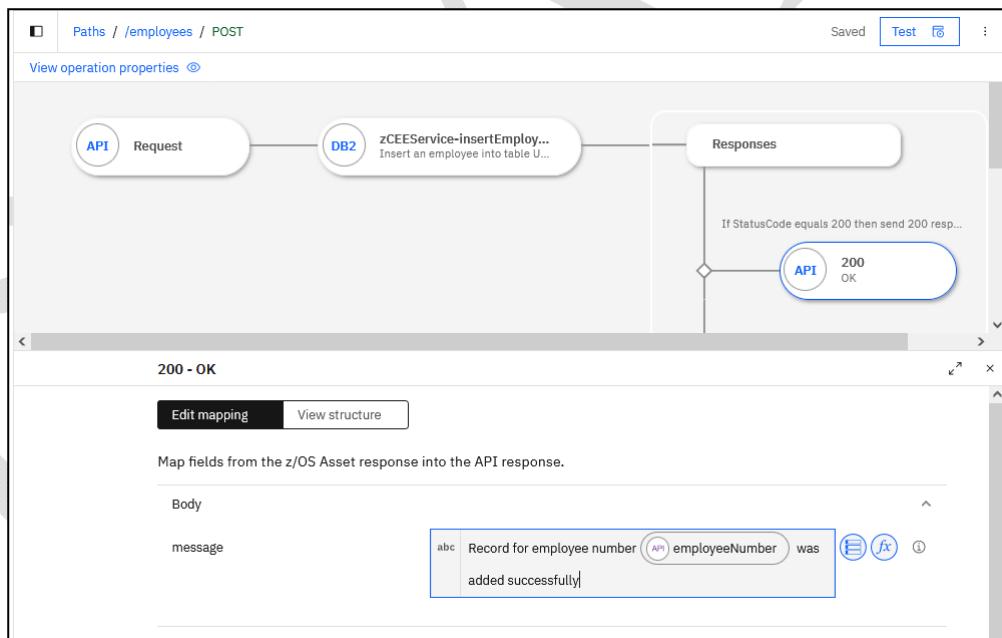
5. If neither of these connections are met, simply return with a HTTP 500 status code.

6. Next the API response messages need to be configured for each of these potential status codes.



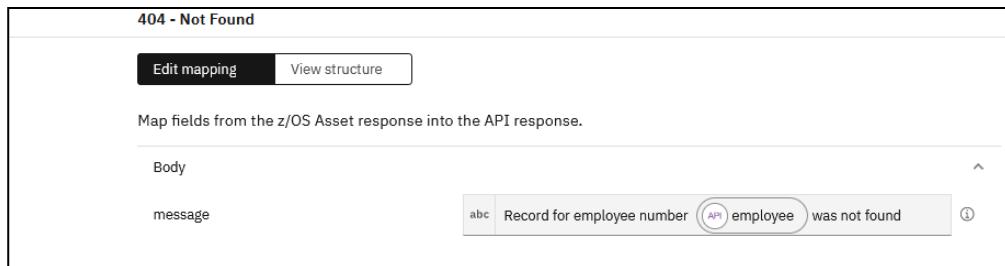
28. Select the response for *200 OK* paste the text below in the *message* area.

**Record for employee number {{\\$apiRequest.body.employeeNumber}} was deleted successfully**



12. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

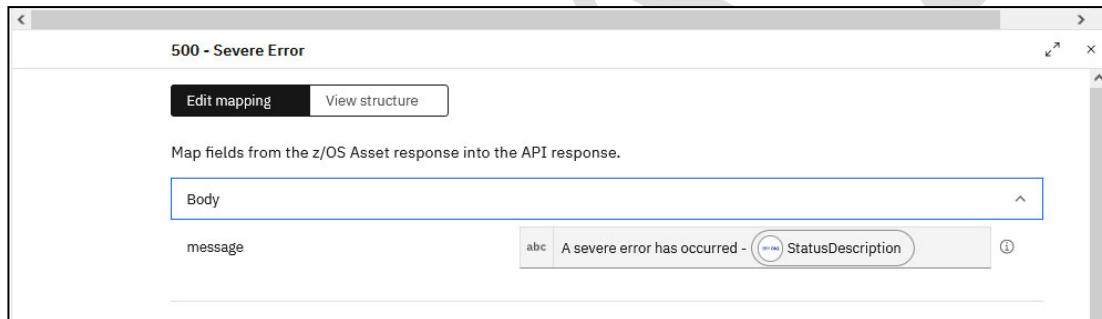
**Record for employee number `>{{$apiRequest.pathParameters.employee}}` was not found**



Notice that the mapping for the property in the message was from the API request message and not the Db2 REST response message.

13. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

**A severe error has occurred -  `{{$zosAssetResponse.body.StatusDescription}}`**



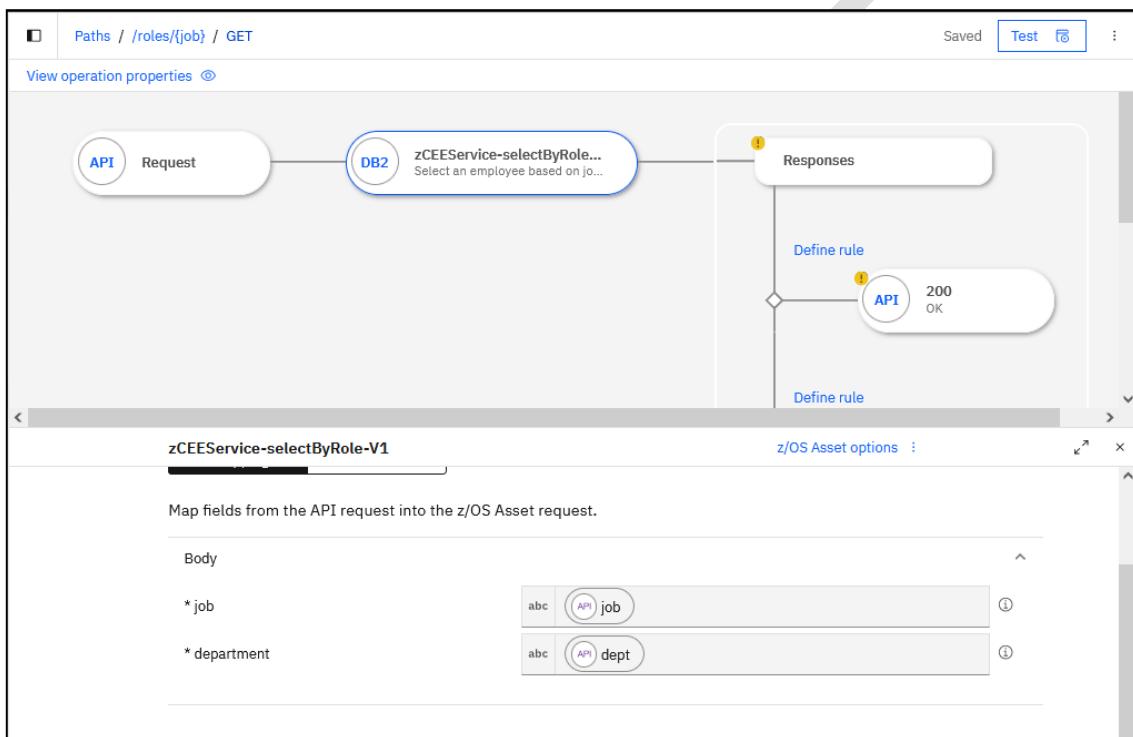
**Tech-Tip:** Db2 REST response property field *StatusDescription* provides more information regarding the issue that caused the insert to fail.

The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

## Configure the GET method for URI path /roles/{job}

Now complete the configuration for the *GET* method of URI Path */roles/{job}*. This method includes both a path parameter and a query parameter.

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-selectByRole* to this method. Map the API request path parameter *job* to the DB2 REST server request message field *job* and the API query parameter *department* to the Db2 REST service request message field *dept*.



2. Maximize the *Responses* area of the browser's page (see below).

Again, responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record a row or rows were returned as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was selected or not. So, we need another indication whether a row was really selected. A Db2 REST service will return the rows selected in a list or array. We are going to take advantage of function that will return the number of elements in the list or array, e.g., \$count. If the result of invoking the function returns a non-zero values, the list or array contains elements. If the result is zero, no rows were selected.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of invoking the \$count function against the array of returned rows.

3. Under the *200 – OK* response, Enter the string ***Stat*** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter ***200*** in the *Input* field for *Value*.

Next add a condition check for the value of invoking the function \$count against the array of rows returned by the Db2 REST service *Count* by clicking on Add condition in the *200 – OK* evaluation and entering the string below in the area for the new check of a Response field.

***\$count(\$zosAssetResponse.body."ResultSet Output")***

And set the *Predicate* to *Is greater than or equal to a Value of 1*.

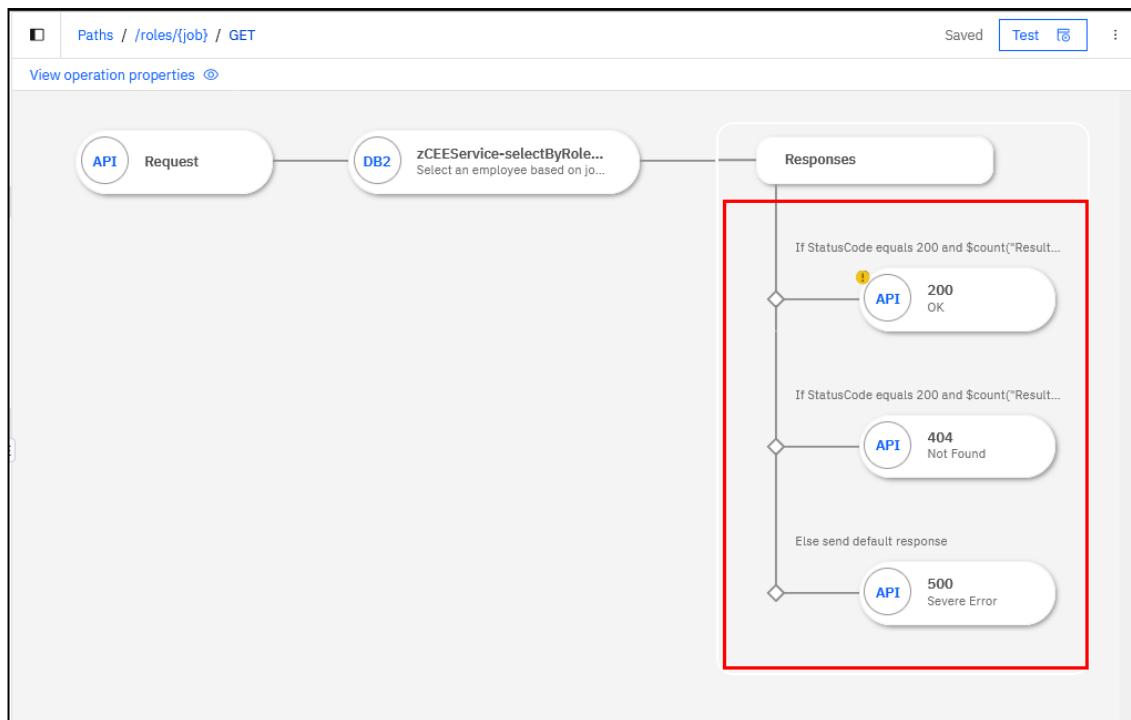
Response field	Predicate	Value
StatusCode	Equals	200
\$count(ResultSet Output)	Is greater than or equal to	1

4. For the *404 – Not Found* check, add a check for *StatusCode* equaling ***200*** and a check for count equaling zero.

Response field	Predicate	Value
StatusCode	Equals	200
count(ResultSet Output)	Equals	0

5. If neither of these connections are met, simply return with a HTTP 500 status code.

6. Next the API response messages need to be configured for each of these potential status codes.



7. Select the response for *200 OK* and map the fields from the Db2 REST response message. Start by mapping the *ResultSet Output* field from the Db2 REST response message to the API response field *results Output*. This must be done first to be able to access the elements in the array.

The screenshot shows the mapping configuration for the `200 - OK` response. It has two tabs: `Edit mapping` (selected) and `View structure`. The mapping instructions are: "Map fields from the z/OS Asset response into the API response." The `Body` section shows the mapping of `* results Output` to an array of `ResultSet Output` objects. Each element in the array has fields for `employeeNumber`, `name`, `department`, `phoneNumber`, and `job`, all mapped to the value "abc".

8. Be careful at this point to ensure you are selecting fields in the *ResultSet output* array in the *body* of the *zosAssetResponse*. The same property name may appear in another one of the available mappings, e.g., *apiRequest*, *ResultSet Row item*, etc. and if a property is selected from one these mappings, the results will be invalid.
9. Complete the mapping for the other properties. Notice the mapping of the Db2 REST response properties *firstName*, *middleInitial* and *lastName* into the API response property *name*.

The screenshot shows the configuration interface for an API endpoint. The path is set to `/roles/{job} / GET`. The status is `200 - OK`. There are two buttons: `Edit mapping` (which is highlighted) and `View structure`. A note below says: "Map fields from the z/OS Asset response into the API response." The `Body` section contains the following mappings:

- \* results Output: An array containing a single item labeled `ResultSet Output`.
- \* employeeNumber: An array containing a single item labeled `employeeNumber`.
- \* name: An array containing three items labeled `firstName`, `middleInitial`, and `lastName`.
- \* department: An array containing a single item labeled `department`.
- \* phoneNumber: An array containing a single item labeled `phoneNumber`.
- \* job: An array containing a single item labeled `job`.

10. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

**No records were found**

404 - Not Found

Edit mapping View structure

Map fields from the z/OS Asset response into the API response.

Body

message abc No record were not found ⓘ

11. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

**A severe error has occurred - {{\\$zosAssetResponse.body.StatusDescription}}**

500 - Severe Error

Edit mapping View structure

Map fields from the z/OS Asset response into the API response.

Body

message abc A severe error has occurred - StatusDescription ⓘ

The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

## Testing APIs deployed in a z/OS Connect Designer container

The deployed APIs are accessible as long as the *Designer* container is active, even when the *Designer* is not opened in a browser. In fact, there are advantages in this behavior when testing security roles. This section will demonstrate using common HTTP clients to test APIs specifically with security enabled.

We know the URI paths of the API from the initial page of the *API Explorer* displayed when testing in the *Designer*. From this page the first part of the URL can be determined, e.g.,

<https://designer.washington.ibm.com:9449/>. This along with the URI path of each methods provides the URL we need to use to invoke a method. For example, to invoke the GET to display the additional details of an employee record in any client, the URL will be

`https://designer.washington.ibm.com:9449/employee/{employee}`

The screenshot shows the Open Liberty API Explorer interface. At the top, it says "Liberty REST APIs 1.0.0 OAS3". Below that, it says "Discover REST APIs available within Liberty". Under "Servers", it lists "https://designer.ibm.com:9449". The main area displays the "employee roster" endpoint with the following methods:

- GET /roles/{job} Retrieve a list of employees based on job and department code
- POST /employees Insert a new employee record into the employee roster
- GET /employees/details/{employee} Display additional details of an employee record
- GET /employees/{employee} Display details of an employee record
- PUT /employees/{employee} Update a subset of details of an employee record
- DELETE /employees/{employee} Delete an employee record

From this display, the methods and URLs required to access the API deployed in this container are:

- GET `https://designer.washington.ibm.com:9449/employee/{job}`
- POST `https://designer.washington.ibm.com:9449/employees`
- GET `https://designer.washington.ibm.com:9449/employees/details/{employee}`
- GET `https://designer.washington.ibm.com:9449/employee/{employee}`
- PUT `https://designer.washington.ibm.com:9449/employee/{employee}`
- DELETE `https://designer.washington.ibm.com:9449/employee/{employee}`

## Using Postman

Start a Postman session using the Postman icon on the desktop.

1. Open the *Postman* tool icon on the desktop and if necessary reply to any prompts and close any welcome messages. Set *Authorization* type to basic and use *Fred* as the *Username* and *fredpwd* as the *Password* and add a header property of ***Content-Type*** with a value of *application/json*. Then use the down arrow in the *Body* tab to select **GET** and enter [\*\*https://designer.washington.ibm.com:9449/employees/details/000010\*\*](https://designer.washington.ibm.com:9449/employees/details/000010) in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.

The screenshot shows the Postman interface with a successful API call. The URL in the request field is <https://designer.ibm.com:9449/employees/details/000010>. The response status is circled in red as 200 OK. The response body is a JSON object containing employee details:

```

1  {
2      "retrievedResults": [
3          {
4              "employeeID": "000010",
5              "name": "CHRISTINE I HAAS",
6              "departmentCode": "A00",
7              "phoneNumber": "3978",
8              "dateOfHire": "1965-01-01",
9              "job": "PRES",
10             "educationLevel": 18,
11             "sex": "F",
12             "dateOfBirth": "1933-08-14",
13             "annualSalary": 52750.0,
14             "lastBonus": 1000.0,
15             "lastCommission": 4220.0
16         }
17     ]
18 }
```

2. Next enter an invalid employee number such as 121212,

[\*\*https://designer.washington.ibm.com:9449/employees/details/121212\*\*](https://designer.washington.ibm.com:9449/employees/details/121212)

in the URL area (see below) and press **Send**. You should see results like below in the response Body area.

The screenshot shows the Postman interface with an unsuccessful API call. The URL in the request field is <https://designer.ibm.com:9449/employees/details/121212>. The response status is circled in red as 404 Not Found. The response body is a JSON object with a message:

```

1  {
2      "message": "Record for employee number 121212 was not found"
3  }
```

## IBM z/OS Connect (OpenAPI 3.0)

If you invoked the underlying Db2 REST service (`/services/zCEEService/displayEmployee`), you would receive an HTTP 200 with an empty *ResultSet Output* array.

The screenshot shows the Postman application interface. A POST request is made to `http://wg31.washington.ibm.com:2446/services/zCEEService/displayEmployee`. The request body contains the JSON object `{"employeeNumber": "121212"}`. The response status is 200 OK, and the response body is a JSON object with the key `"ResultSet Output"` pointing to an empty array, indicating execution was successful.

This demonstrates one of the advantages of using z/OS Connect APIs as a gateway to Db2 REST services.

3. Next use Postman to invoke a *POST* method with URL

**<https://designer.washington.ibm.com:9449/employees>** and the JSON below for the request message.

```
{  
  "employeeNumber": "948499",  
  "firstName": "Matt",  
  "middleInitial": "T",  
  "lastName": "Johnson",  
  "departmentCode": "C00",  
  "phoneNumber": "0065",  
  "dateOfHire": "10/15/1980",  
  "job": "Staff",  
  "educationLevel": 21,  
  "sex": "M",  
  "dateOfBirth": "06/18/1960",  
  "salary": 3999.99,  
  "lastBonus": 399.99,  
  "lastCommission": 119.99  
}
```

Pressing the **Send** button invokes the POST method of URI path */employees* which in turns invokes the Db2 REST service *zCEEService.insertEmployee*.

The screenshot shows the Postman application interface. A POST request is made to <https://designer.ibm.com:9449/employees>. The request body is a JSON object containing employee details:

```

1 "employeeNumber": "948499",
2 "firstName": "Matt",
3 "middleInitial": "T",
4 "lastName": "Johnson",
5 "departmentCode": "C00",
6 "phoneNumber": "0065",
7 "dateOfHire": "10/15/1980",
8

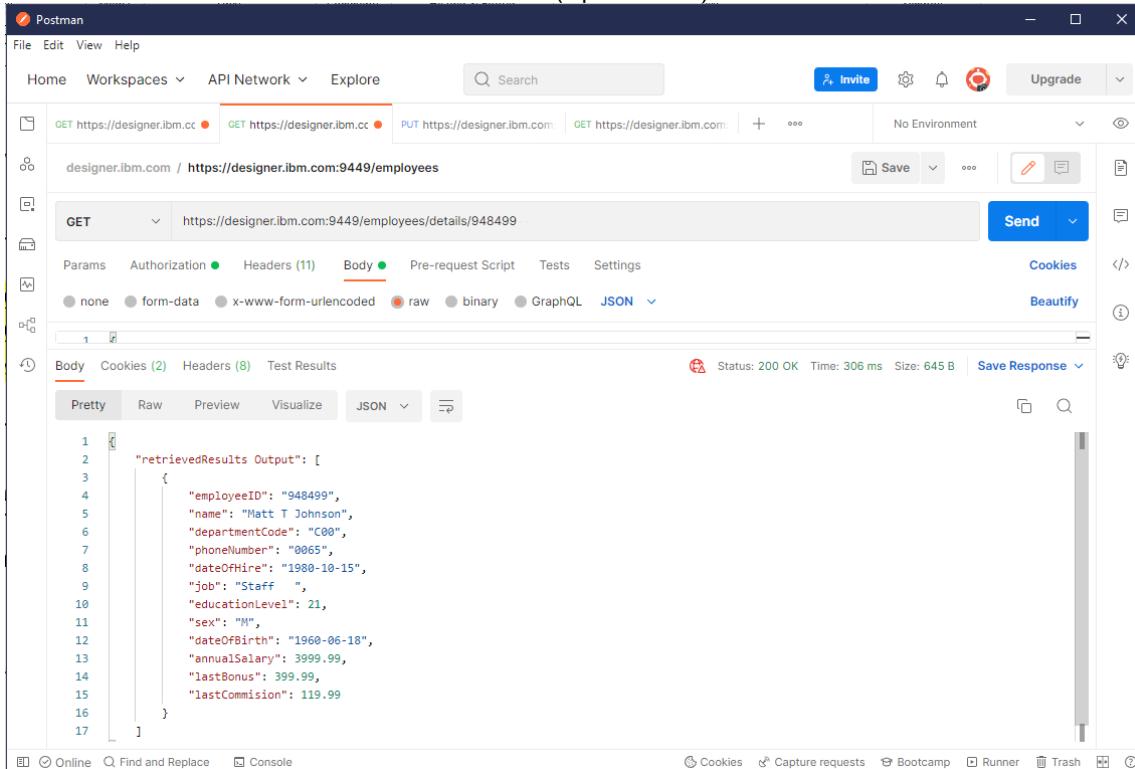
```

The response status is 200 OK, and the message is: "Record for employee number 948499 was added successfully".

4. Invoke the GET method with URL

<https://designer.washington.ibm.com:9449/employees/details/948499> to display the record just inserted.

## IBM z/OS Connect (OpenAPI 3.0)



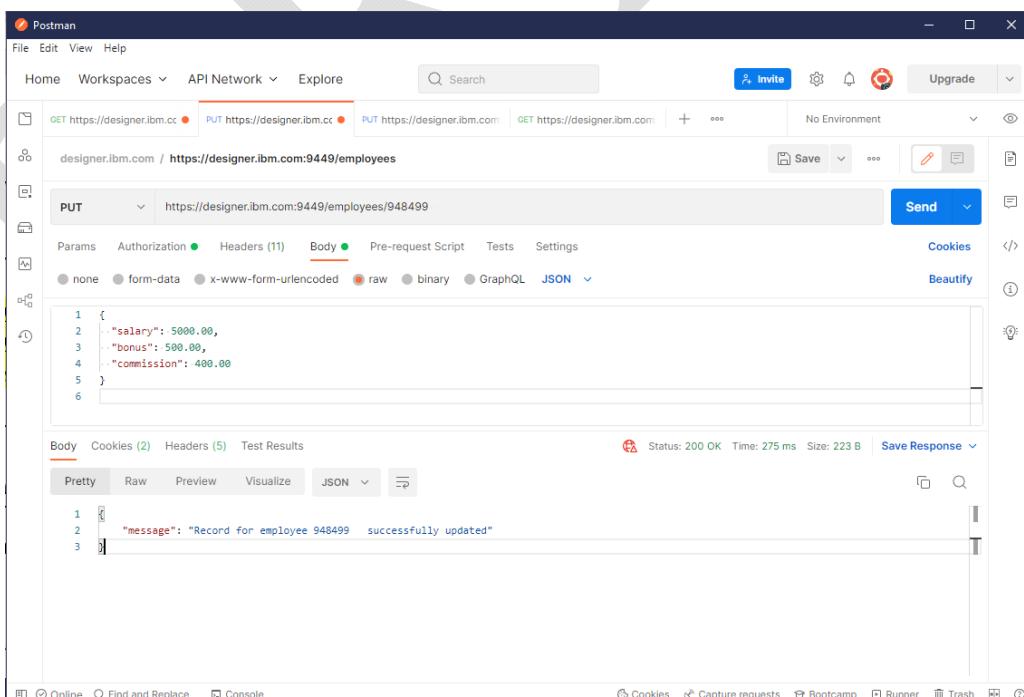
The screenshot shows the Postman interface with a successful GET request to <https://designer.ibm.com:9449/employees/948499>. The response status is 200 OK, time is 306 ms, and size is 645 B. The response body is a JSON object:

```
1 "retrievedResults": [  
2   {  
3     "employeeID": "948499",  
4     "name": "Matt T Johnson",  
5     "departmentCode": "C00",  
6     "phoneNumber": "00651",  
7     "dateOfHire": "1980-10-15",  
8     "job": "Staff ",  
9     "educationLevel": 21,  
10    "sex": "M",  
11    "dateOfBirth": "1960-06-18",  
12    "annualSalary": 3999.99,  
13    "lastBonus": 399.99,  
14    "lastCommission": 119.99  
15  }  
16 ]  
17 ]
```

5. Invoke a **PUT** method with URL <https://designer.washington.ibm.com:9449/employees/948499> and the JSON below for the request message to update the *salary*, *bonus*, and *commission* columns of this row.

```
{  
  "salary": 5000.00,  
  "bonus": 500.00,  
  "commission": 400.00  
}
```

6.



The screenshot shows the Postman interface with a successful PUT request to <https://designer.ibm.com:9449/employees/948499>. The response status is 200 OK, time is 275 ms, and size is 223 B. The response body is a JSON object:

```
1 {  
2   "salary": 5000.00,  
3   "bonus": 500.00,  
4   "commission": 400.00  
5 }  
6 ]
```

The response message is: "Record for employee 948499 successfully updated".

7. Display the updated record by using a *GET* method with URL

<https://designer.washington.ibm.com:9449/employees/details/948499>

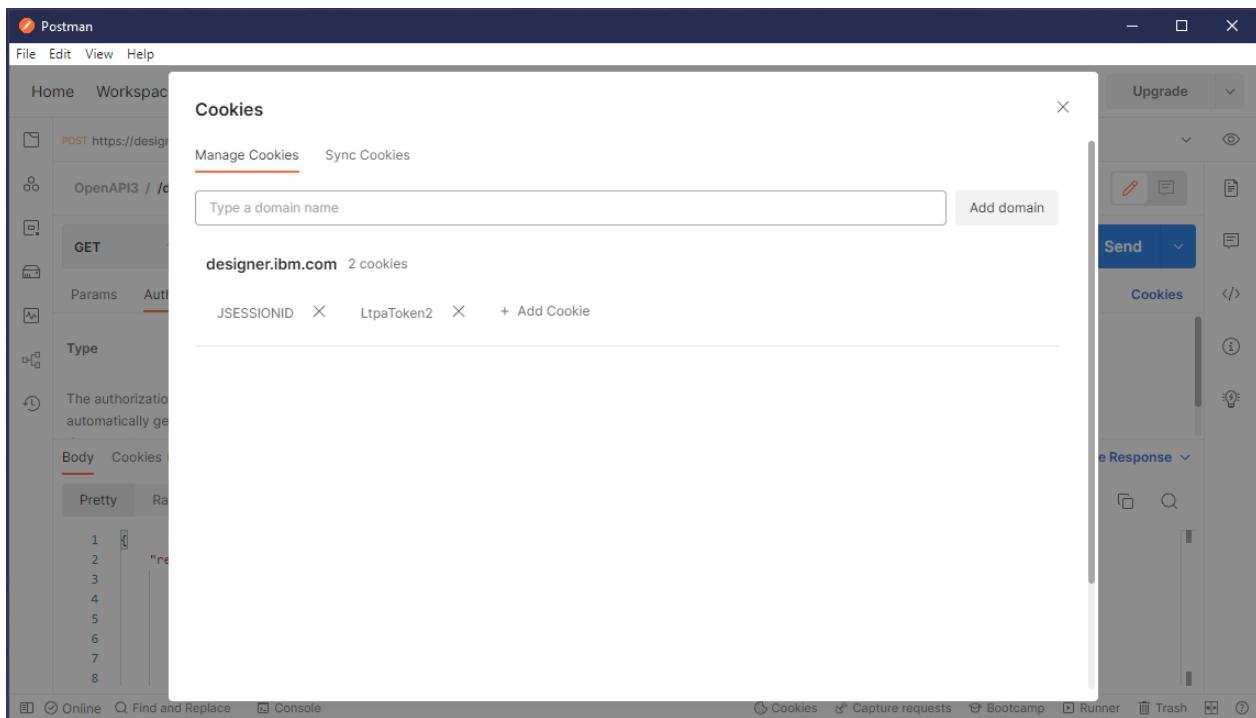
```

1  {
2      "retrievedResults Output": [
3          {
4              "employeeID": "948499",
5              "name": "Matt T Johnson",
6              "departmentCode": "C00",
7              "phoneNumber": "0065",
8              "dateOfHire": "1980-10-15",
9              "job": "Staff",
10             "educationLevel": 21,
11             "sex": "M",
12             "dateOfBirth": "1960-06-18",
13             "annualSalary": 5000.0,
14             "lastBonus": 500.0,
15             "lastCommission": 400.0
16         }
17     ]
18 }
```

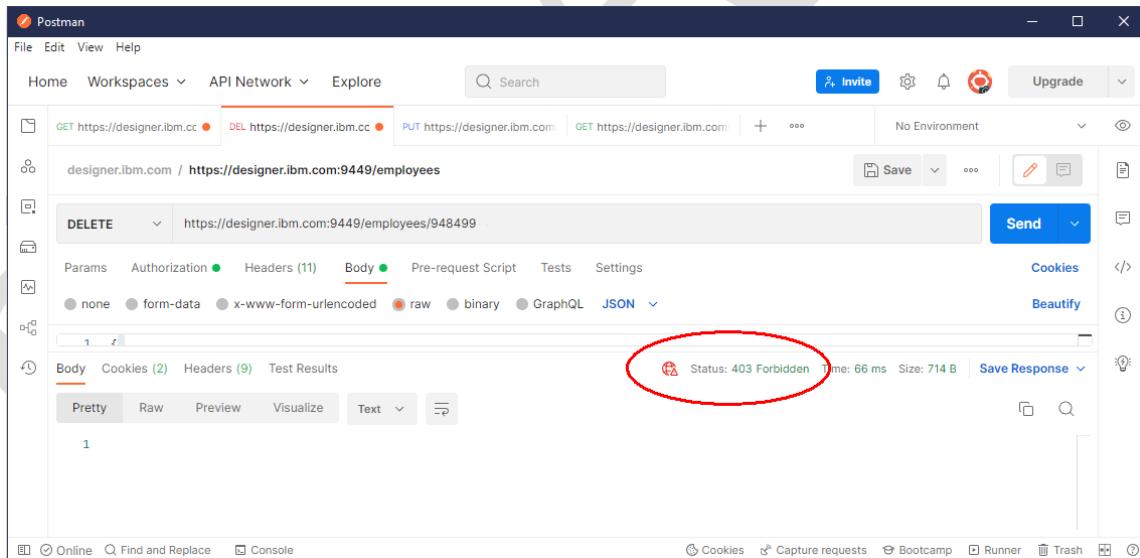
8. Up until this point you have been using the role assigned to user *Fred*. Now experiment using user *user1* and/or *user2*. Before we can use other credentials we have to clear the credentials that cached by *Postman*. Unless this is done, *Postman* will continue to use the credentials for *Fred* regardless of what is provided in the authorization header.

9. To clear the *Postman* cached security tokens, click on the *Cookies* section of the *Postman* window and

IBM z/OS Connect (OpenAPI 3.0)  
And delete any *JSESSIONID* and *LtpaToken2* cookies displayed.



Test various methods using Username *user1* and *user2* and observe the results. Remember, *user1* can only invoke *GET* methods and *user2* can not invoke any method. For example, if you try to delete a record as *user1*, you should see an HTTP Status code of 403 (Forbidden).



While an delete request by Fred is successful.

The screenshot shows the Postman application interface. A DELETE request is made to `https://designer.ibm.com:9449/employees/948499`. The response status is 200 OK, and the message "record deleted" is displayed in red text. The interface includes tabs for Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings. Authorization is set to Basic Auth with a password. The Body tab shows the JSON response:

```
1 "message": "record deleted"
```

## Using cURL

*Client for URL (cURL)* is a common tool for driving REST client request to APIs. In this section, the *curl* command will be used to test the API's methods deployed into the *z/OS Connect Designer*'s container and more importantly, demonstrate role-based security. *Postman* caches security credentials between tests and the cached credentials must be cleared if the identity being used is changed. *cURL* does not this caching of credentials and therefore it is easier to change security credentials between request with *cURL* than with *Postman*.

1. Start a DOS command prompt session and go to directory *c:\z\openapi3*, e.g., *cd \z\openapi3*.

2. Enter the *curl* command below and observe the response.

```
curl -X GET -w " - HTTP CODE ${http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure
https://designer.washington.ibm.com:9449/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w " - HTTP CODE ${http_code}" --user Fred:fredpwd --
header "Content-Type: application/json" --insecure
https://localhost:9449/employees/details/000010
{"retrievedResults Output": [{"employeeID": "000010", "name": "CHRISTINE I HAAS",
"departmentCode": "A00", "phoneNumber": "3978", "dateOfHire": "1965-01-01", "job": "PRES",
",
"educationLevel": 18, "sex": "F", "dateOfBirth": "1933-08-14", "annualSalary": 52750.0,
"lastBonus": 1000.0, "lastCommision": 4220.0}]} - HTTP CODE 200
```

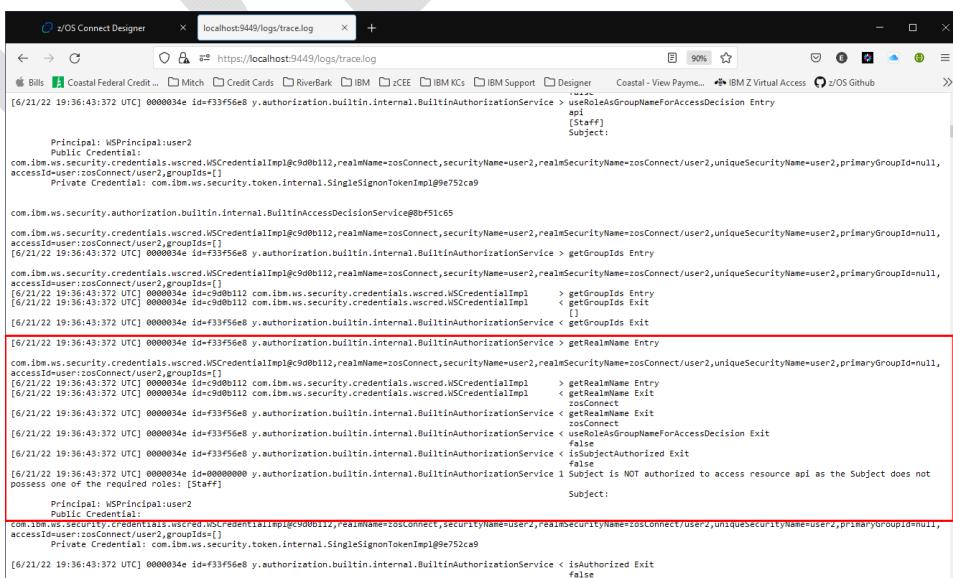
Fred is a member of the *Staff* group and has *Staff* access to the **Staff** role. Any identity with **Staff** access can invoke one of the GET methods.

3. Enter the *curl* command below and observe the response.

```
curl -X GET -w " - HTTP CODE ${http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure
https://designer.washington.ibm.com:9449/employees/details/000010
```

```
curl -X GET -w " - HTTP CODE ${http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure
https://localhost:9449/employees/details/000010
```

A review of the *trace.log* file will show the HTTP 403 (Forbidden) occurred because the identity *user2* is not a member of the *Staff* group.



**Tech-Tip:** If you had provided an invalid password, e.g., `-user user2:userx`, the request would have failed with an HTTP status of 401, Unauthorized.

4. Enter the `curl` command below and observe the response.

```
curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @insertEmployee.json https://designer.washington.ibm.com:9449/employees/
```

```
c:\z\openapi3>curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @insertEmployee.json https://designer.washington.ibm.com:9449/employees/
{"message":"Record for employee number 948489 was added successfully"} - HTTP CODE 200
```

In the above command, the file `insertEmployee.json` has the contents below:

```
{
  "employeeNumber": "948489",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

5. Enter the `curl` command below to invoke the `GET` method with URI path `/roles/{job}`.

```
curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9449/roles/PRES?dept=A00
```

```
curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9449/roles/PRES?dept=A00
{"results Output": [{"employeeNumber": "[\"000010\", \"000011\"]", "name": "[\"CHRISTINE\", \"CHRISTINE\"]", "job": "[\"PRES\", \"PRES\"]"}, {"employeeNumber": "[\"000010\", \"000011\"]", "name": "[\"CHRISTINE\", \"CHRISTINE\"]", "job": "[\"PRES\", \"PRES\"]"}]} - HTTP CODE 200
```

Now try using user2's credentials.

6. Enter the *curl* command below to invoke the *DELETE* method with URI path */employees/{employee}*.

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9449/employees/948489
```

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9449/employees/948489
{"message":"record deleted"} - HTTP CODE 200
```

7. Enter the *curl* command below to invoke the *DELETE* method again with URI path */employees/{employee}*.

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user user1:user1 https://designer.washington.ibm.com:9449/employees/948489
```

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user user1:user1 https://designer.washington.ibm.com:9449/employees/948489
- HTTP CODE 403
```

8. Enter the *curl* command below to invoke the *PUT* with URI path */employees/{employee}*.

```
curl -X PUT -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @updateEmployee.json https://designer.washington.ibm.com:9449/employees/948489
```

```
curl -X PUT -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @updateEmployee.json https://designer.washington.ibm.com:9449/employees/948489
- HTTP CODE 200
```

In the above command, the file *updateEmployee.json* has the contents below:

```
{
  "salary": 5000.00,
  "bonus": 500.00,
  "commission": 400.00
}
```

Invoke other curl commands varying the credentials , methods, etc. until you feel comfortable the API is working as intended.

## Using the API Explorer

The API Explorer was used in the Designer to test the APIs as they were being developed. The API Explorer All the URI paths and methods in the original OpenAPI 3 specification document will be displayed, but only the *POST* for */employees* and the *GET* for */employees/{employee}* have been created. Executing one of the other methods will return an HTTP 404 because the components required to execute these methods cannot be found in the WAR.

1. Using the Firefox browser, go to URL <https://designer.washington.ibm.com:9449/api/explorer> to start the API Explorer.

**Tech Tip:** You may be challenged by browser because the digital certificate used by the *Designer* is self-signed Click the **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button. Next you may see a prompt you for a userid and password. If you do see the prompt, enter the username **Fred** and password **frepwd** (case matters) and click **OK**. Remember we are using basic security, and this is the user identity and password defined in the server.xml file.

## IBM z/OS Connect (OpenAPI 3.0)

Click on *Post /employees* URI path to display the request body view of the URI path.

The screenshot shows the z/OS Connect Designer OpenAPI UI. The browser title bar says "z/OS Connect Designer" and "OpenAPI UI". The address bar shows "https://localhost:9449/api/explorer/". The main content area displays the "employee roster" section. Under the "POST /employees" endpoint, there is a "Request body" section with a "Try it out" button and a dropdown set to "application/json". Below this is a "request body" example with a "Schema" link. The schema code is as follows:

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "string",
  "lastName": "string",
  "departmentCode": "string",
  "phoneNumber": "string",
  "dateOfBirth": "string",
  "job": "string",
  "educationLevel": 32767,
  "sex": "string",
  "dateOfBirth": "string",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

2. Next press the **Try it out** button to enable the entry of an authorization string and a request message body

This is a screenshot of the "Try it out" dialog for the POST /employees endpoint. It has a "Parameters" tab at the top. The "Authorization" parameter is defined with the value "Authorization" and type "string (header)". Below this is a "Request body" section with a "schema" link and a JSON schema example. The schema is identical to the one shown in the previous screenshot. At the bottom of the dialog, there is a "Servers" section with a note about overriding global server options, a URL input field with a dropdown, and a large blue "Execute" button.

3. Enter the JSON request message below in the *Request body* section and press the **Execute** button.

```
{
  "employeeNumber": "948478",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

4. Security was enabled in the original specification document, so you will be required to sign in with one of the identities defined in the basicSecurity.xml file explored earlier. Use **Fred** for the *Username* and **fredpwd** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.
5. Scroll down the view and you should see the *Response body* with the expected successful message.

The screenshot shows the IBM z/OS Connect API testing interface. At the top, there is a message: "These path-level options override the global server options." Below this is a search bar and a toolbar with "Execute" and "Clear" buttons. The main area is titled "Responses". Under "Responses", there is a "Curl" section containing the cURL command used to make the request:

```
curl -X 'POST' \
  'https://localhost:9449/employees' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d {
    "employeeNumber": "948478",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0065",
    "dateOfHire": "10/15/1980",
    "job": "Staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
  }
```

Below the curl command is the "Request URL" field, which contains "https://localhost:9449/employees". Under "Server response", the status code is 200, and the "Response body" is shown as:

```
{
  "message": "Record for employee number 948478 was added successfully"
}
```

There are "Download" and "Copy" buttons next to the response body. The "Response headers" section shows the following:

```
content-language: en-US
content-length: 113
content-type: application/json
date: Fri, 17 Jun 2022 17:15:13 GMT
x-firefox-spdy: h2
x-powered-by: Servlet/4.0
```

6. Press the **Execute** button again and observe the results. A row for this employee number already existed in the employee roster (a Db2 tables) so the request failed with an HTTP 500.

The screenshot shows the 'Responses' tab of the API interface. Under the 'Curl' section, a command is shown to POST to the '/employees' endpoint with JSON data for employee 948478. The 'Request URL' is https://localhost:9449/employees. The 'Server response' section shows a 500 Internal Server Error. The response body contains an error message: "A severe error has occurred - Service xCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-803, SQLSTATE=23505, Message=AN INSERTED OR UPDATED VALUE IS INVALID BECAUSE INDEX IN INDEX SPACE EMPLOYEECA CONSTRAINS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS. RID OF EXISTING ROW IS X'0000000229'. Error Location:DSNLJXUS:6". Response headers include content-language: en-US, content-length: 400, content-type: application/json, date: Fri, 17 Jun 2022 17:18:19 GMT, x-firefox-spdy: h2, and x-powered-by: Servlet/4.0.

7. Scroll down and click on *GET /employees/{employee}* URI path to display the request body view of the URI path for this method. Next click on the **Try it out** button to enable the entry of data for this method. Enter **948478** as the employee identity and press the **Execute** button to retrieve a subset of data for this employee.

The screenshot shows the 'Responses' tab of the API interface. Under the 'Curl' section, a command is shown to GET the '/employees/948478' endpoint. The 'Request URL' is https://localhost:9449/employees/948478. The 'Server response' section shows a 200 OK response. The response body contains a JSON object with a 'results output' array containing one element, which is another JSON object with fields: employeeNumber, name, departmentCode, phoneNumber, and job. Response headers include content-language: en-US, content-length: 133, content-type: application/json, date: Fri, 17 Jun 2022 17:31:03 GMT, x-firefox-spdy: h2, and x-powered-by: Servlet/4.0.

8. Try this again using number **121212** and observe the results. You see the message that the employee was not found.

9. Expand the **PUT** method and enter press the **Try it out** button.

10. Enter **948478** in the *employee* field and paste the JSON below in the request body area.

```
{
  "salary": 5000.00,
  "bonus": 500.00,
  "commission": 400.00
}
```

11. Press the **Execute** button.

The screenshot shows the 'Server response' interface. It includes tabs for 'Code' (set to 200) and 'Details'. Under 'Response body', there is a JSON object with a single key 'message': "Record for employee 948478 successfully updated". Below the body, 'Response headers' are listed, including content-language: en-US, content-type: application/json, date: Mon, 18 Jul 2022 22:33:29 GMT, x-firebase-spdy: h2, and x-powered-by: Servlet/4.0. There are also 'Copy' and 'Download' buttons.

12. Expand the **GET** method for URI path */employees/details/{employee}* and enter press the **Try it out** button.

13. Enter **948478** in the *employee* field and press the **Execute** button. Observe that the updates values have been applied.

The screenshot shows the 'Server response' interface. It includes tabs for 'Code' (set to 200) and 'Details'. Under 'Response body', there is a JSON object with a key 'retrievedResultsOutput' containing a list of employee details. One entry in the list includes fields like employeeID: "948478", name: "Mitch T Johnson", departmentCode: "C001", phoneNumber: "0000", dateOfBirth: "1980-06-18", job: "Staff", educationLevel: 21, sex: "M", dateHire: "1980-06-18", annualSalary: 5000, lastBonus: 500, and lastCommission: 400. Below the body, 'Response headers' are listed, including content-language: en-US, content-type: application/json, date: Mon, 18 Jul 2022 22:33:29 GMT, x-firebase-spdy: h2, and x-powered-by: Servlet/4.0. There are also 'Copy' and 'Download' buttons.

14. Expand the **DELETE** method for URI path */employees/{employee}* and enter press the **Try it out** button. Observe the record has been deleted.

The screenshot shows the 'Server response' interface. It includes tabs for 'Code' (set to 200) and 'Details'. Under 'Response body', there is a JSON object with a single key 'message': "record deleted". Below the body, 'Response headers' are listed, including content-language: en-US, content-type: application/json, date: Mon, 18 Jul 2022 22:33:29 GMT, x-firebase-spdy: h2, and x-powered-by: Servlet/4.0. There are also 'Copy' and 'Download' buttons.

15. Repeat either of the two **GET** method request for employee **948478** and you see a message that the record could not be found.

Try other methods using other rows in the table. The initial contents of the Db2 table are shown below.

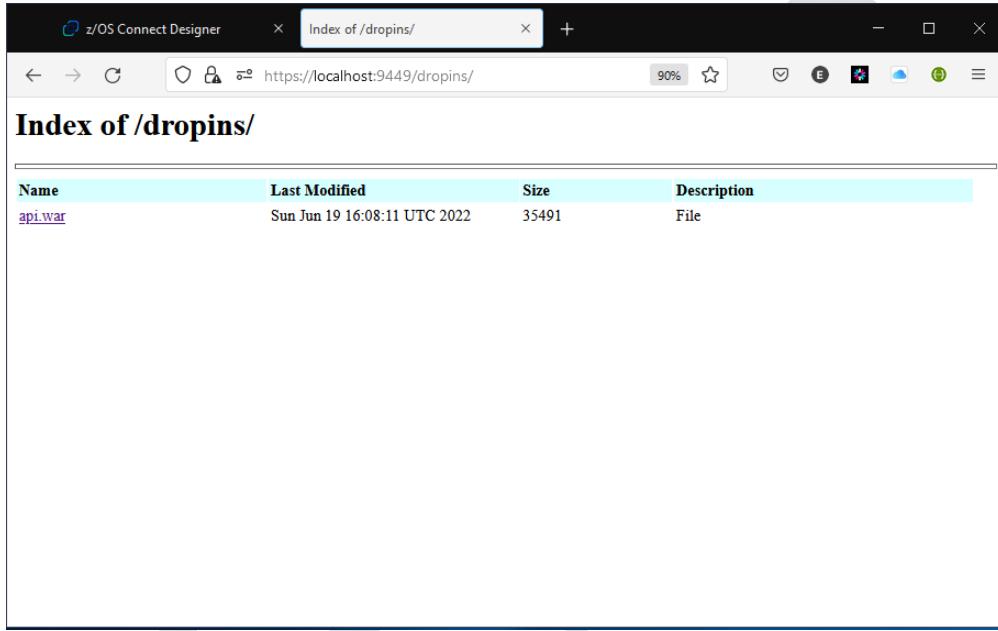
EMPN	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	Birthdate	Salary	Bonus	COMM
000011	CHRISTINE	I	HAAS	A00	A1A1	1965-01-01	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
000110	VINCENZO	G	LUCCHESI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
000180	MARYLYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
000260	SYBIL	V	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

## Deploying and installing APIs in a z/OS Connect Native Server

As the *z/OS Connect Designer* is being used to develop the API from specification file, a Web Archive (WAR) file is being constantly regenerated and being automatically deployed to the z/OS Connect server embedded in the *Designer*. This section of the exercises provides details on how this WAR file can be extracted from the *Designer* container, moved to a zOS OMVS directory, and then added to a native z/OS Connect server.

### Moving the API Web Archive file from the container to a z/OS OMVS directory

- 1. The first step is to use the file serving capability added the Liberty server's configuration. Use a web browser to access URL <https://designer.washington.ibm.com:9449/dropins>.



Double click the *api.war* file and save the file in local directory, e.g., *c:\z\openApi3\wars*. Specify a *File* name of *employees.war*.

- 2. Open a DOS command prompt and use the change directory command to go to directory *C:\z\openApi3\wars*, e.g., **cd C:\z\openApi3\wars**
- 3. Start a file transfer session with the WG31 host using the *ftp* command, e.g., **ftp wg31**
- 4. Logon as USER1 and then use the *cd* command to change to directory to data set */var/zcee/openApi3/apps*, e.g. **cd /var/zcee/openApi3/apps**
- 5. Toggle prompting off by entering command **prompt**
- 6. Enter binary mode transmission but entering command **bin**
- 7. Perform multiple put requests by using the multiple put command, **mput employees.war**

8. When the last transfer has completed enter the **quit** command.

```
c:\z\openApi3> cd wars
c:\z\openApi3\wars>ftp wg31.washington.ibm.com
Connected to wg31.washington.ibm.com.
220-FTP 16:26:23 on 2018-02-15.
220 Connection will close if idle for more than 200 minutes.
User (wg31.washington.ibm.com:(none)): user1
331 Send password please. user1
Password:
230 USER1 is logged on. Working directory is "USER1.".
ftp> cd /var/zcee/openApi3
250 HFS directory /var/zcee/openApi3/apps is the current working directory
ftp> prompt
Interactive mode Off .
ftp> bin
200 Representation type is Image
ftp> mput employees.war
200 Port request OK.
125 Storing data set /var/zcee/openApi3/apps/employees.war
250 Transfer completed successfully.
ftp: 35491 bytes sent in 0.39Seconds 90.77Kbytes/sec.
ftp> quit
```

These steps have moved the WAR file from the Designer container to the OMVS directory accessible by the z/OS Connect native server.

### **Updating the server xml**

The next step is to add a *webApplication* server XML configuraton element for the API to the OpenAPI 3 server's configuration.

1. Edit OMVS file **server.xml** in directory */var/zcee/openApi3* and add this configuration element.

```
<webApplication id="db2" contextRoot="/db2" name="db2API"
location="${server.config.dir}apps/employees.war"/>
```

The addition of this configuration adds the web application found in the *employees.war* file to the server's configuration. The context root of */db2* is prepended is to the URI paths of each URI path found in the web application to ensure the uniqueness of this API's URI paths versus the URI paths of other APIs installed in the server.

2. Use MVS modify command **F ZCEEAPI3,REFRESH,CONFIG** to have the server XML changes installed.

**Tech-Tip:** To refresh an application using the MVS modify command **F ZCEEAPI3,REFRESH,APPS**

This completes the installation of the API's web application.

## Defining the required RACF EJBRole resources

The API has been installed but the required RACF EJBRoles have not been defined and access permitted. This section describes the steps required to complete the RACF configuration required to execute the API.

Remember the specification file defined two roles for invoking the methods of this API, *Manager* and *Staff*. In the basicSecurity.xml configuration file we saw how we configured these roles and granted access to the roles in a Liberty internal registry. On z/OS we want to use RACF. The names of the required RACF EJBRoles are derived by combining information from 3 sources. The first is the *profilePrefix* attribute of the server XML *safCredentials* configuration element. In our case, the value of *profilePrefix* is **ATSZDFLT**. The next source is the name of the web application. The name of the web application is either derived from information in the specification or the *name* attribute provided on the *webApplication* configuration element. In our case, this value should be **db2API**. The final source is the role name provided in the specification document, **Manager** or **Staff**. So, two EJBRoles need to be defined, **ATSZDFLT.db2API.Manager** and **ATSZDFLT.db2API.Staff**.

1. Use the RACF RDEFINE command to define EJBROLE **ATSZDFLT.db2API.Manager**.

```
rdefine ejbrole ATSZDFLT.db2API.Manager uacc(none)
```

2. Use the RACF RDEFINE command to define EJBROLE **ATSZDFLT.db2API.Staff**.

```
rdefine ejbrole ATSZDFLT.db2API.Staff uacc(none)
```

3. Use the RACF PERMIT command to permit identity FRED READ access to EJBROLE **ATSZDFLT.db2API.Manager**.

```
permit ATSZDFLT.db2API.Manager class(ejbrole) id(fred) acc(read)
```

4. Use the RACF PERMIT command to permit identity FRED READ access to EJBROLE **ATSZDFLT.Db2API.STAFF**.

```
permit ATSZDFLT.db2API.Staff class(ejbrole) id(fred) acc(read)
```

5. Use the RACF PERMIT command to permit identity USER1 READ access to EJBROLE **ATSZDFLT.Db2API.STAFF**.

```
permit ATSZDFLT.db2API.Staff class(ejbrole) id(user1) acc(read)
```

6. Use the RACF SETROPTS command to refresh the EJBRole instorage profiles.

```
setropts raclist(ejbrole) refresh
```

Now we are ready to test the invoking of the methods of this API.

## Testing APIs deployed in a native z/OS server

### Using Postman

Start a Postman session using the Postman icon on the desktop.

1. Open the *Postman* tool icon on the desktop and if necessary reply to any prompts and close any welcome messages. Set *Authorization* type to basic and use *fred* as the *Username* and *fred* as the *Password* as we did when directly testing the Db2 REST services and add a header property of **Content-Type** with a value of *application/json*. Then use the down arrow in the *Body* tab to select **GET** and enter <https://wg31.washington.ibm.com:9445/db2/employees/details/000010> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.

The screenshot shows the Postman application window. In the top navigation bar, there are tabs for Home, Workspaces, API Network, and Explore. Below the navigation bar, there is a search bar and a toolbar with various icons. The main workspace shows a list of requests. One request is selected, which is a GET request to the URL <https://wg31.washington.ibm.com:9445/db2/employees/details/000010>. The request details panel shows the method (GET), URL, and various configuration options like Headers, Body, and Settings. The Body tab is selected, showing the JSON response. The response body is a JSON object with a single key "retrievedResults\_Output" containing an array of employee details. The status bar at the bottom right indicates "Status: 200 OK". A red oval highlights this status message.

```

1 {
  "retrievedResults_Output": [
    {
      "employeeID": "000010",
      "name": "CHRISTINE I HAAS",
      "departmentCode": "A00",
      "phoneNumber": "3978",
      "dateOfHire": "1965-01-01",
      "job": "PRES",
      "educationLevel": 18,
      "sex": "F",
      "dateOfBirth": "1933-08-14",
      "annualSalary": 52750.0,
      "lastBonus": 1000.0,
      "lastCommision": 4220.0
    }
  ]
}

```

Notice what is different from the earlier testing when the API was deployed in the *Designer* container. First the credentials were for the RACF credentials (*fred*) with access to the RACF EJBRole. The other major differences was of the context root of */db2* in the URI path. This was the value of the *contextRoot* attribute in the *webApplication* configuration element that defined this application in the z/OS Connect server.

2. Next enter an invalid employee number such as 121212,

<https://wg31.washington.ibm.com:9445/db2/employees/details/121212> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.

The screenshot shows the Postman application interface. A GET request is made to <https://wg31.washington.ibm.com:9445/db2/employees/details/121212>. The response status is 404 Not Found, and the message is "Record for employee number 121212 was not found".

If you invoked the underlying Db2 REST service (`/services/zCEEService/displayEmployee`) you would receive an HTTP 200 with an empty *ResultSet Output* array.

The screenshot shows the Postman application interface. A POST request is made to <http://wg31.washington.ibm.com:2446/services/zCEEService/displayEmployee> with the body: {"employeeNumber": "121212"}. The response status is 200 OK, and the message is "Execution Successful".

The API created in the *Designer* essentially intercepted the response from the Db2 service and was to determine no results were found and returned an HTTP 404 (not found) to the client rather than an HTTP 200 (OK).

## IBM z/OS Connect (OpenAPI 3.0)

Optional, experiment using *Postman* to invoke other methods of the API. For example, if you want to invoke a *POST* with URI path */employees*, use the JSON below for the request message.

```
{  
    "employeeNumber": "948489",  
    "firstName": "Matt",  
    "middleInitial": "T",  
    "lastName": "Johnson",  
    "departmentCode": "C00",  
    "phoneNumber": "0065",  
    "dateOfHire": "10/15/1980",  
    "job": "Staff",  
    "educationLevel": 21,  
    "sex": "M",  
    "dateOfBirth": "06/18/1960",  
    "salary": 3999.99,  
    "lastBonus": 399.99,  
    "lastCommission": 119.99  
}
```

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, Home, Workspaces, API Network, Explore, and a search bar. The main workspace shows a 'New Request' card for 'wg31.washington.ibm.com / New Request'. The request method is set to POST, and the URL is https://wg31.washington.ibm.com:9445/db2/employees. The 'Body' tab is selected, showing a JSON payload identical to the one above. The 'Send' button is visible. Below the request details, the response pane shows a status of 200 OK with a response message: "Record for employee number 948489 was added successfully". The bottom navigation bar includes Online, Find and Replace, Console, Cookies, Capture requests, Bootcamp, Runner, Trash, and a help icon.

Or if you want to do a *PUT* with URI path /db2/employees/{employee} use this JSON request message.

<https://wg31.washington.ibm.com:9445/db2/employees/948489>

```
{
  "salary": 5000.00,
  "bonus": 500.00,
  "commission": 400.00
}
```

The screenshot shows the Postman application window. The top navigation bar includes File, Edit, View, Help, Home, Workspaces, API Network, Explore, and a search bar. The main workspace shows a list of environments with 'wg31.washington.ibm.com' selected. A PUT request is being prepared to the URL <https://wg31.washington.ibm.com:9445/db2/employees/948489>. The request details panel shows the method (PUT), URL, and various tabs like Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings. The Body tab is active, displaying a JSON payload:

```
{
  "salary": 5000.00,
  "bonus": 500.00,
  "commission": 400.00
}
```

The response panel shows a status of 200 OK, a time of 574 ms, and a size of 221 B. The response body is displayed in Pretty format:

```

1
2   "message": "Record for employee 948489 successfully updated"
3

```

3. Up until this point you have been using the role assigned to user *Fred*. Now experiment using user *user1* and/or *user2*. Before we can use other credentials we have to clear the credentials that cached by *Postman*. Unless this is done, *Postman* will continue to use the credentials for *Fred* regardless of what is provided in the authorization header

4. To clear the *Postman* cached tokens, click on the *Cookies* section of the *Postman* window and

The screenshot shows the Postman interface with a GET request to `https://designer.ibm.com:9449/employees/948478`. The 'Cookies' tab is circled in red. The response status is 200 OK.

```

1
2   "results": [
3     {
4       "employeeNumber": "948478",
5       "name": "Matt T Johnson",
6       "departmentCode": "C00",
7       "phoneNumber": "0065",
8       "job": "Staff"
  
```

And delete any *JSESSIONID* and *LtpaToken2* cookies displayed.

The screenshot shows the 'Cookies' dialog box in Postman. It lists two cookies for the domain `designer.ibm.com`: `JSESSIONID` and `LtpaToken2`.

Test various methods using Username *user1* and *user2* and observe the results. Remember, *user1* can only invoke GET methods and *user2* can not invoke any method.

## Using cURL

*Client for URL (cURL)* is a common tool for driving REST client request to APIs. In this section, the *curl* command will be used to test the API's methods deployed into the *z/OS Connect Designer*'s container and more importantly, demonstrate role-based security. *Postman* caches security credentials between tests and the cached credentials must be cleared if the identity being used is changed. *cURL* does not this caching of credentials and therefore it is easier to change security credentials between request with *cURL* than with *Postman*.

1. Start a DOS command prompt session and go to directory *c:\z\openapi3*, e.g., *cd \z\openapi3*.

2. Enter the *curl* command below and observe the response.

```
curl -X GET -w "%{http_code}" --user Fred:fred --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/db2/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w "%{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://localhost:9449/employees/details/000010
{"retrievedResults": [{"employeeID": "000010", "name": "CHRISTINE I HAAS", "departmentCode": "A00", "phoneNumber": "3978", "dateOfHire": "1965-01-01", "job": "PRES ", "educationLevel": 18, "sex": "F", "dateOfBirth": "1933-08-14", "annualSalary": 52750.0, "lastBonus": 1000.0, "lastCommision": 4220.0}]} - HTTP CODE 200
```

Fred is a member of the *Staff* group and has *Staff* access to the *Staff* role. Any identity with *Staff* access can invoke one of the GET methods.

3. Enter the curl command below and observe the response.

```
curl -X GET -w "%{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/db2/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w "%{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://localhost:9449/employees/details/000010
- HTTP CODE 403
```

**Tech-Tip:** If you had provided an invalid password, e.g., *--user user2:userx*, the request would have failed with an HTTP status of 401, *Unauthorized*.

4. Enter the curl command below and observe the response.

```
curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred --data @insertEmployee.json https://wg31.washington.ibm.com:9445/db2/employees/
```

In the above command, the file insertEmployee.json has the contents below:

```
{
  "employeeNumber": "948489",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

```
c:\z\openapi3>curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred --data @insertEmployee.json https://wg31.washington.ibm.com:9445/employees/
```

5. Enter the curl command below to invoke the *GET* method with URI path */roles/{job}*.

```
curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred https://wg31.washington.ibm.com:9445/db2/roles/PRES?dept=A00
```

6. Enter the curl command below to invoke the *DELETE* method with URI path */employees/{employee}*.

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred https://wg31.washington.ibm.com:9445/db2/employees/000012
```

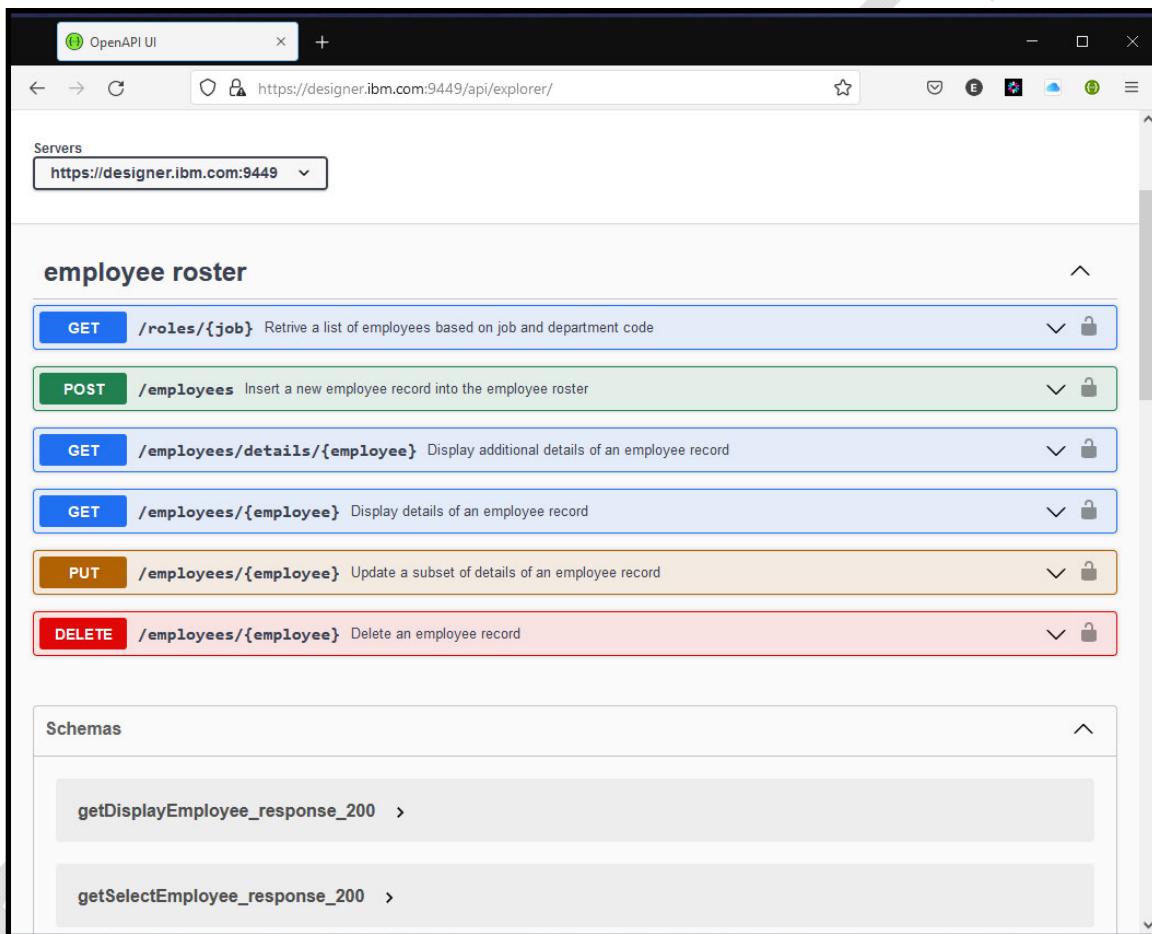
7. Enter the curl command below to invoke the *DELETE* method with URI path */employees/{employee}*.

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user user1:user1 https://wg31.washington.ibm.com:9445/db2/employees/000012
```

## Using the API Explorer

The API Explorer was used in the Designer to test the APIs as they were being developed. The API Explorer All the URI paths and methods in the original OpenAPI 3 specification document will be displayed, but only the *POST* for */employees* and the *GET* for */employees/{employee}* have been created. Executing one of the other methods will return an HTTP 404 because the components required to execute these methods cannot be found in the WAR.

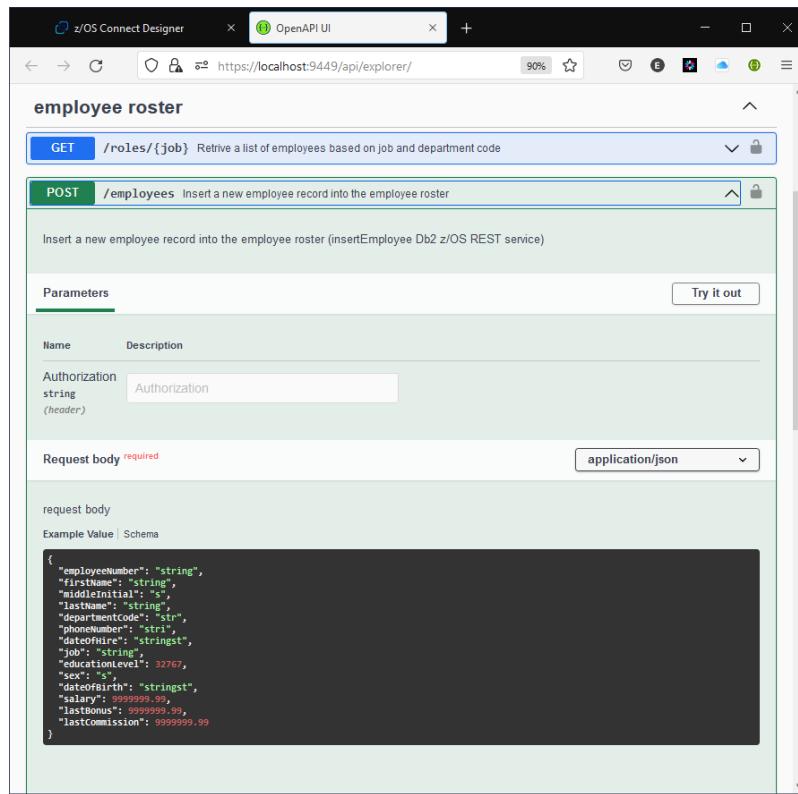
16. Using the Firefox browser, go to URL <https://wg31.washington.ibm.com:9445/api/explorer> to start the API Explorer.



**Tech Tip:** You may be challenged by browser because the digital certificate used by the *Designer* is self-signed Click the **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button. Next you may see a prompt you for a userid and password. If you do see the prompt, enter the username **Fred** and password **frepwd** (case matters) and click **OK**. Remember we are using basic security, and this is the user identity and password defined in the server.xml file.

## IBM z/OS Connect (OpenAPI 3.0)

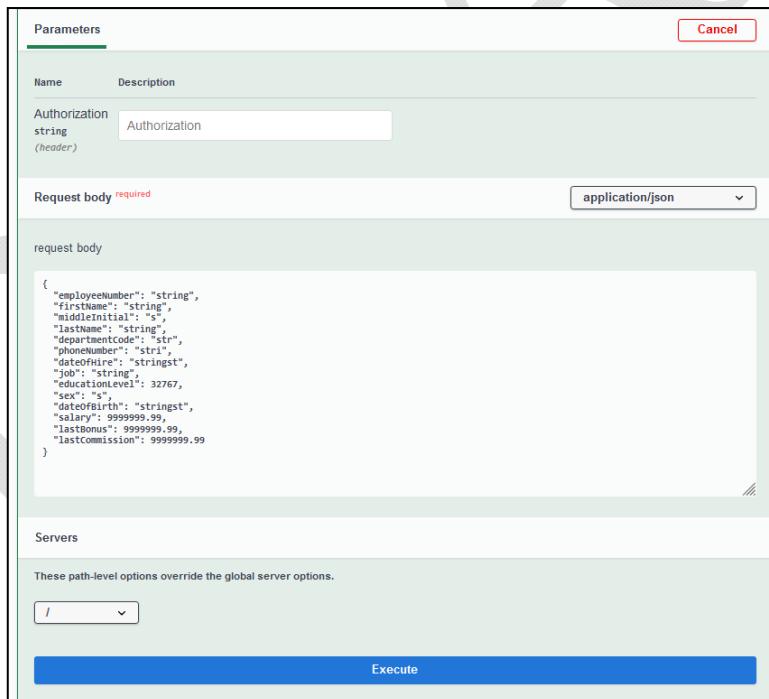
Click on *Post /employees* URI path to display the request body view of the URI path.



The screenshot shows the z/OS Connect Designer interface with the OpenAPI UI tab selected. The URL is https://localhost:9449/api/explorer/. The main area displays the 'employee roster' section with two endpoints: a GET /roles/{job} endpoint and a POST /employees endpoint. The POST endpoint is highlighted. Below it, a detailed description of the POST /employees endpoint is shown, including its purpose ('Insert a new employee record into the employee roster (insertEmployee Db2 z/OS REST service)'), parameters (Authorization header), and request body schema. The request body schema is defined as a JSON object with the following fields:

```
{ "employeeNumber": "string", "firstName": "string", "middleInitial": "s", "lastName": "string", "departmentCode": "st", "phoneNUmber": "string", "dateOfHire": "string", "job": "string", "educationLevel": 32767, "sex": "s", "dateOfBirth": "string", "salary": "999999.99", "lastBonus": "999999.99", "lastCommission": "999999.99" }
```

17. Next press the **Try it out** button to enable the entry of an authorization string and a request message body



This screenshot shows the 'Try it out' dialog for the POST /employees endpoint. It contains the same parameters and request body schema as the previous screenshot. At the top right is a red 'Cancel' button. At the bottom is a large blue 'Execute' button.

18. Enter the JSON request message below in the *Request body* section and press the **Execute** button.

```
{
    "employeeNumber": "948478",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0065",
    "dateOfHire": "10/15/1980",
    "job": "Staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
}
```

19. Security was enabled in the original specification document, so you will be required to sign in with one of the identities defined in the basicSecurity.xml file explored earlier. Use **Fred** for the *Username* and **fredpwd** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

20. Scroll down the view and you should see the *Response body* with the expected successful message.

The screenshot shows the IBM z/OS Connect API Explorer interface. At the top, there is a status bar with the message "These path-level options override the global server options." Below this is a search bar with the value "/". To the right of the search bar are two buttons: "Execute" and "Clear".

The main area is titled "Responses". Under "Responses", there is a "curl" section containing the command used to make the request:

```
curl -X 'POST' \
  'https://localhost:9449/employees' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "employeeNumber": "948478",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0065",
    "dateOfHire": "10/15/1980",
    "job": "Staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
}'
```

Below the curl command is the "Request URL" field, which contains the URL "https://localhost:9449/employees".

Under "Server response", there is a table with two columns: "Code" and "Details". A single row is shown for code 200, with the details "Response body". The response body is displayed in a dark box:

```
{
  "message": "Record for employee number 948478 was added successfully"
}
```

To the right of the response body are two buttons: "Copy" and "Download".

At the bottom of the "Server response" section is a "Response headers" table, showing the following headers:

content-language: en-US
content-length: 71
content-type: application/json
date: Fri, 17 Jun 2022 17:15:13 GMT
x-firfox-spy: n2
x-powered-by: Servlet/4.0

21. Press the **Execute** button again and observe the results. A row for this employee number already existed in the employee roster (a Db2 tables) so the request failed with an HTTP 500.

The screenshot shows the 'Responses' tab of the API interface. Under the 'Curl' section, a command is shown to post data to the '/employees' endpoint. The 'Request URL' is https://localhost:9449/employees. The 'Server response' section shows a 500 Internal Server Error. The error message is: "message": "A severe error has occurred - Service xCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-803, SQLSTATE=23505, Message=AN INSERTED OR UPDATED VALUE IS INVALID BECAUSE INDEX IN INDEX SPACE EMPLOYEECA CONSTRAINS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS. RID OF EXISTING ROW IS X'00000000229'. Error Location:DSNLXUS:6". Below the error message are 'Download' and 'Copy' buttons. The 'Response headers' section lists standard HTTP headers like Content-Language, Content-Length, Content-Type, Date, X-Firefox-Spdy, and X-Powered-By.

22. Scroll down and click on *GET /employees/{employee}* URI path to display the request body view of the URI path for this method. Next click on the **Try it out** button to enable the entry of data for this method. Enter **948478** as the employee identity and press the **Execute** button to retrieve a subset of data for this employee.

The screenshot shows the 'Responses' tab of the API interface. Under the 'Curl' section, a command is shown to get the employee with ID 948478. The 'Request URL' is https://localhost:9449/employees/948478. The 'Server response' section shows a 200 OK response. The response body contains a JSON object with a 'results output' array containing one element, which is another JSON object with fields: employeeNumber, name, departmentCode, phoneNumber, and job. Below the response body are 'Download' and 'Copy' buttons. The 'Response headers' section lists standard HTTP headers like Content-Language, Content-Length, Content-Type, Date, X-Firefox-Spdy, and X-Powered-By.

23. Try this again using number **121212** and observe the results. You see the message that the employee was not found.

The initial contents of the Db2 table are shown below.

EMPNO	FIRSTNAME	MIDDLENAME	LASTNAME	WORKDEPT	PHONEENO	HIREDATE	JOB	EDLEVEL	SEX	Birthdate	Salary	Bonus	COMM
000011	CHRISTINE	I	HAAS	A00	A1A1	1965-01-01	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
000110	VINCENZO	G	LUCCHESI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
000120	SEAN	O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00	
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
000260	SYBIL	V	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

## Using TLS to access Db2

### Creating the RACF resources

In this section, the certificates and key ring required for TLS connections to Db2 will be imported into and/or defined to RACF.

The certificate authority public certificate, labeled as *Db2 CA*, be imported into RACF as a CERTAUTH certificate. There are also personal certificates signed by the same certificate authority that will be used as Db2's server certificate (labeled *DB2USER*) and sent to the client to be used as a client certificate (labeled *designer*) for use when mutual authentication is enabled. The personal certificates are provided by the certificate authority in encrypted files since they contain private keys. This means that the certificate authority has also provide a password or pass phrase that can be used to decrypt the contents of these files.

The certificate authority (CERTAUTH) certificate and the personal certificate labeled DB2USER will be connected to the RACF key ring used by Db2 as the server endpoint. *Note that Db2 actually depends on AT-TLS to act as the server endpoint on behalf of Db2.*

The CERTAUTH certificate will also be provided to Db2 clients in order for them to verify the server certificate provided by Db2 during the handshake. Only clients with the public key of the certificate authority that signed the Db2's server certificate in their trust stores will be able perform TLS handshakes with Db2.

Private keys in a personal certificate are by both server and client endpoints to encrypt a message during the TLS handshake. The other endpoint uses the public key of the sending endpoint to decrypt the message encrypted with the private key. Being able to successfully decrypt a message encrypted with a private key, verifies the identity of the sending endpoint (asymmetric encryption) since only the holder of a private key should the owner of the certificate.

**Tech-Tip:** In this exercise we are using RACF as our certificate authority. This is unusual and you probably will be using a non-RACF certificate authority, either internal your organization or one of the well-known certificate authorities like Verisign or Entrust. So, you probably not be using RACF to create certificate authority certificates or using certificate authority certificates to sign certificate request. More than likely you will only be using the RACDCERT command to create keyrings and connecting certificates to these key rings.

1. Browse data set *USER1.ZCEE30.CNTL*. You should see the members in that data set.

2. Browse member **DB2TLS3**. The job contains the RACF RACDCERT commands below. Submit the job for execution.

```

/* Add CERTAUTH and personal certificates to RACF */
racdcert CERTAUTH withlabel('DB2 CA') -
    add('USER1.DB2CA.PEM')

racdcert id(db2user) withlabel('DB2USER') -
    add('USER1.DB2USER.P12') password('secret')

racdcert id(USER1) withlabel('Designer') -
    add('USER1.DESIGNER.P12') password('secret')

racdcert id(USER2) withlabel('USER2') -
    add('USER1.USER2.P12') password('secret')

racdcert id(ATSSERV) Withlabel('ATSSERV Db2 Cert') -
    add('USER1.ATSSERV.P12') password('secret')

setr raclist(digtcert,digtnmap) refresh

/* Create DB2 key ring and connect CA and personal certificates */
racdcert id(db2user) addring(Db2.KeyRing)

racdcert id(db2user) connect(ring(Db2.KeyRing) +
    label('DB2 CA') certauth usage(certaauth))

/* Connect default personal certificate */
racdcert id(db2user) connect(ring(Db2.KeyRing) +
    label('DB2USER') default

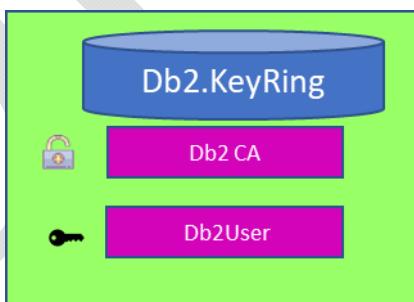
setropts raclist(digtring,digtnmap) refresh

```

The commands in this job:

- Imports (adds) the certificate authority and personal certificates to RACF. The personal certificates are added because if they are to be used in TLS handshakes, they have to be associated with valid RACF identities
- Define a key ring to be used for TLS handshakes.
- Connect the Db2 server personal certificate to this key ring.
- Connect the CA public certificate used to sign the Db2 server's certificate to this key ring.
- The in-storage profile for digital certificates resources are refreshed.

When finished, the Db2 key ring looks something like this:



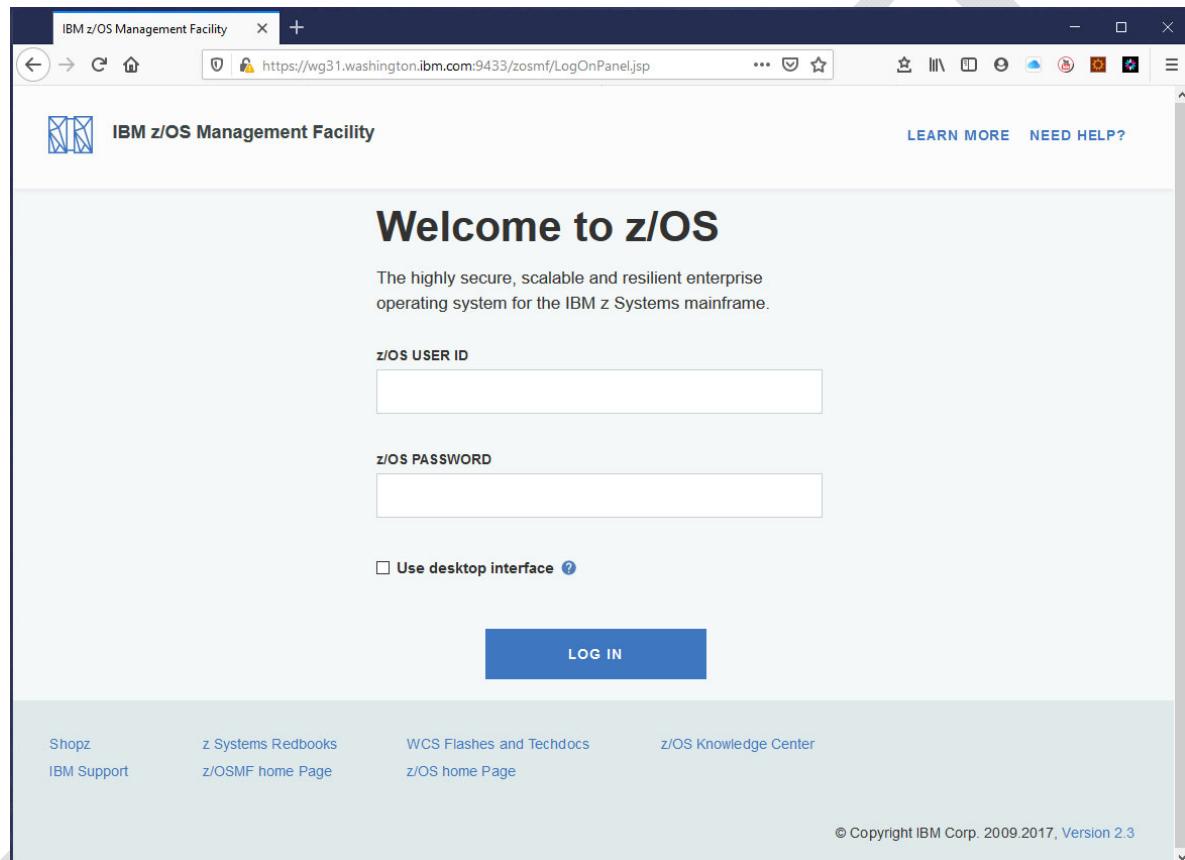
**Tech-Tip:** Both server certificates and client certificates are personal certificates. The difference is the role of the certificate during the handshake. The endpoint that initiates the connection is the client and its personal certificate is referred to as a client certificate. The endpoint that receives the connection request is the server and its personal certificate is referred to as a server certificate. A personal certificate can be used as either a client or a server certificate at any time as long as the certificate contains a private key that can be used to encrypt handshake messages.

## Configuring the Db2 AT-TLS inbound policy

Db2 depends on AT-TLS to perform the handshakes and the encrypting and decrypting of messages. So, the RACF resources created in the previous section are not configured in Db2 but rather AT-TLS. The required AT-TLS configuraton steps are done in this section

z/OSMF will be used in this section to configure the AT-TLS configuration for the desired outbound policy.

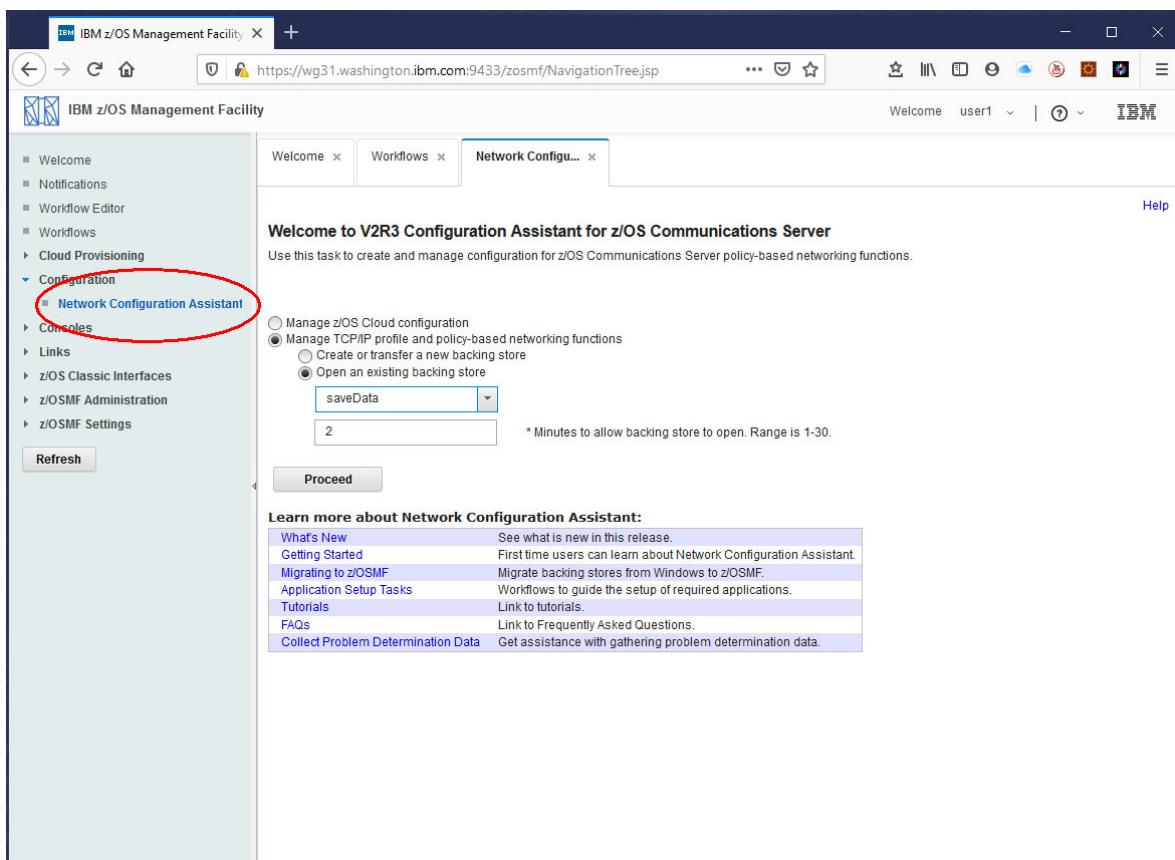
1. In a Firefox browser, enter URL <https://wg31.washington.ibm.com:9433/zosmf> and you should see the *IBM z/OS Management Facility* window.



Note that some of the AT-TLS configurations steps described here may have been performed in another exercise.

2. Enter *USER1* as the *z/OS USER ID* and *USER1*'s password and click the **LOG IN** button.

3. The *Welcome* screen should be displayed. On the left-hand side expand the *Configuration* tab to expose the *Network Configuration Assistance* option. Select this option to expose the *Network Configuration* tab.

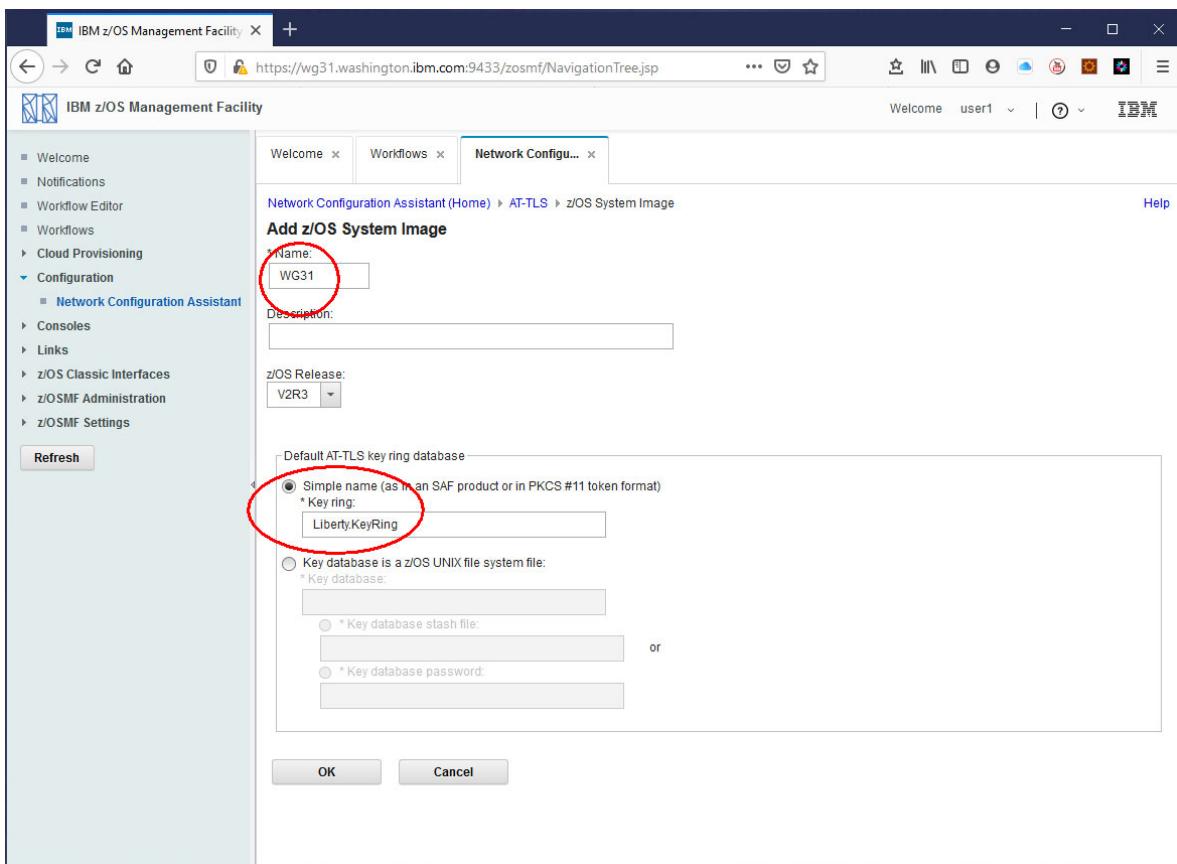


4. Select the radio button beside *Create or transfer a new backing store* option and click the **Proceed** button.
5. On the next screen select the radio button beside *Create a New Backing Store File* and enter **USER1** in the area beside *File Name*. Press the **OK** button and press the **OK** button on the Information pop-up.

6. On the *Network Configuration* tab use the pull-down arrow to select *AT-TLS* as the *TCP/IP technology to configure*.

7. Select the radio button beside the *Default - System Group* and use the *Action* pull-down button to select *Add z/OS System Image* option.

8. On the *Add z/OS System Image* window enter **WG31** for the image *Name* and check the radio button beside *Simple name (as in an SAF product...)*. Enter **Liberty.KeyRing** as the default AT-TLS key ring name. Click **OK** to continue.

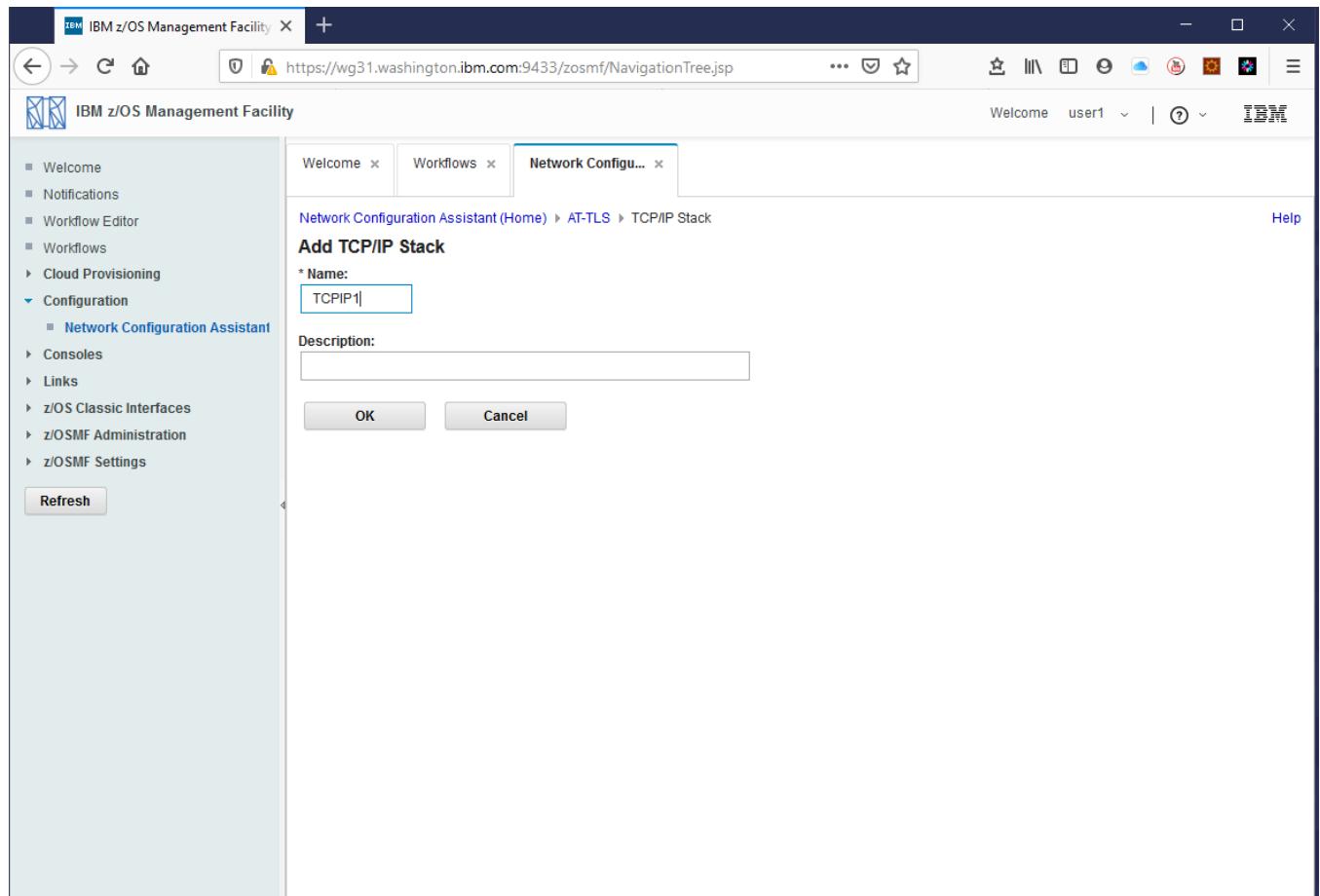


**Tech Tip:** The value for the key ring will be used if an explicit key ring is not provided for a policy.

We recommend establishing a naming convention for key rings with each SAF identity by using the same key ring name. Using this name as an example, you could create a unique key ring named *Liberty.KeyRing* for SAF identities USER1, USER2, FRED, etc. Each user's key ring would have the same name but a different set of connected certificates. One default key ring specified at the image level covers all users.

9. On the *Proceed to the Next Step?* pop-up click the **Proceed** button.

10. The *Add TCP/IP Stack* screen should be displayed. Select this option to expose the *Network Configuration* tab. Enter **TCPIP1** as the name of the stack. Click **OK** to continue.



**Tech-Tip:** The value for the stack name was determined by the TCPIP Name display by entering the MVS command D TCPIP.

```
EZAOP50I TCPIP STATUS REPORT 007
COUNT      TCPIP NAME      VERSION      STATUS
-----      -----      -----
      1      TCPIP1          CS V2R3        ACTIVE
*** END TCPIP STATUS REPORT ***
EZAOP41I 'DISPLAY TCPIP' COMMAND COMPLETED SUCCESSFULLY
```

11. Before any TCP/IP stack rules can be added, *Traffic Descriptors*, *Address Groups* and *Requirement Maps* need to be defined. Click **Cancel** on the *Proceed to the Next Step?* displayed at this time.

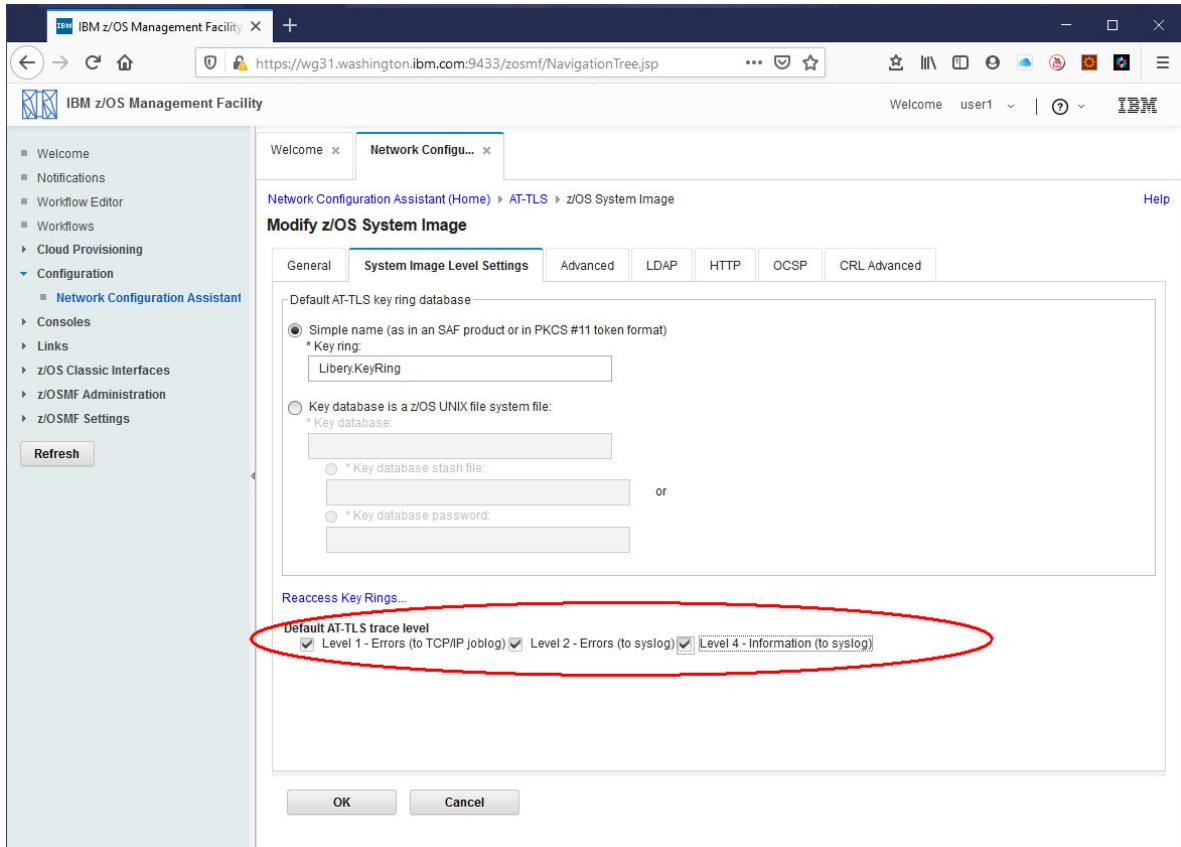


12. This will display the window below:

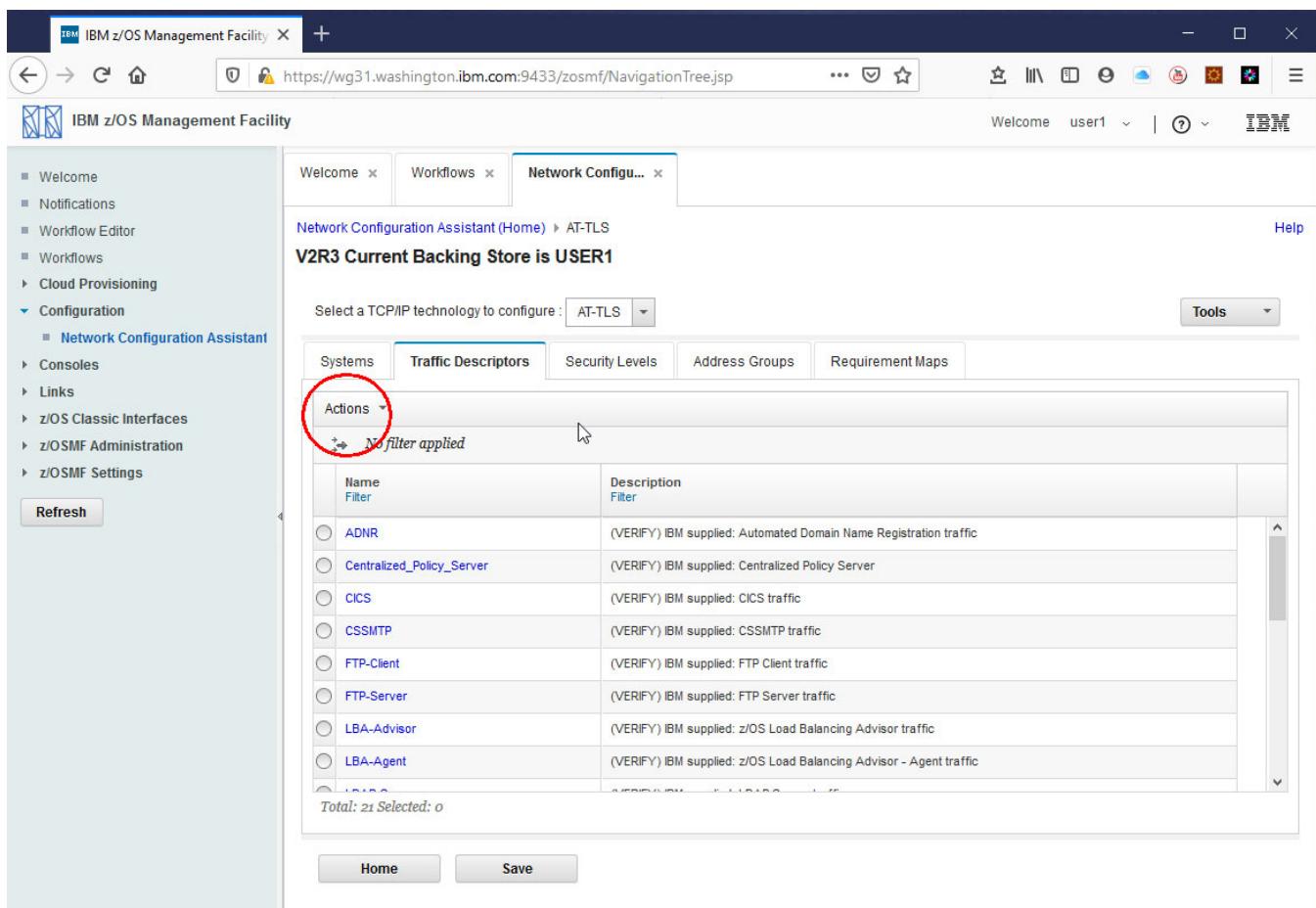
System Group or Sysplex / System Image / Stack	Type Filter	Status Filter	Install Status Filter	Release Filter
Default	System Group	Complete	N/A	V2R3
WG31	System Image	Complete	N/A	V2R3
TCPIP1	Stack	Incomplete	N/A	V2R3

**Tech Tip:** The **Incomplete** warning will be addressed shortly.

13. Select the radio button beside *WG31* and use the *Actions* pull-down to select *Properties*. On the *Modify z/OS System Image* window, select the *System Image Level Settings* tab. Check all the trace level boxes as shown below. This is being done so we can confirm AT-TLS is being invoked and identify issues. Press **OK** to continue.



14. Select the *Traffic Descriptors* tab and use the *Actions* pull-down to select *New*.



The screenshot shows the IBM z/OS Management Facility interface. The left sidebar has a tree view with nodes like Welcome, Notifications, Workflow Editor, Workflows, Cloud Provisioning, Configuration (Network Configuration Assistant is expanded), Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. A Refresh button is at the bottom of the sidebar.

The main area has tabs: Welcome, Workflows, and Network Configu... (which is active). Below the tabs, it says "Network Configuration Assistant (Home) > AT-TLS". It also says "V2R3 Current Backing Store is USER1".

A dropdown menu labeled "Select a TCP/IP technology to configure" is set to "AT-TLS". Below it is a "Tools" dropdown.

The central part of the screen shows a table with tabs: Systems, Traffic Descriptors (which is selected and highlighted in blue), Security Levels, Address Groups, and Requirement Maps. The "Actions" dropdown is circled in red. The table header includes "Name Filter" and "Description Filter".

The table lists various traffic descriptors:

Name Filter	Description Filter
ADNR	(VERIFY) IBM supplied: Automated Domain Name Registration traffic
Centralized_Policy_Server	(VERIFY) IBM supplied: Centralized Policy Server
CICS	(VERIFY) IBM supplied: CICS traffic
CSSMTP	(VERIFY) IBM supplied: CSSMTP traffic
FTP-Client	(VERIFY) IBM supplied: FTP Client traffic
FTP-Server	(VERIFY) IBM supplied: FTP Server traffic
LBA-Advisor	(VERIFY) IBM supplied: z/OS Load Balancing Advisor traffic
LBA-Agent	(VERIFY) IBM supplied: z/OS Load Balancing Advisor - Agent traffic
...	...

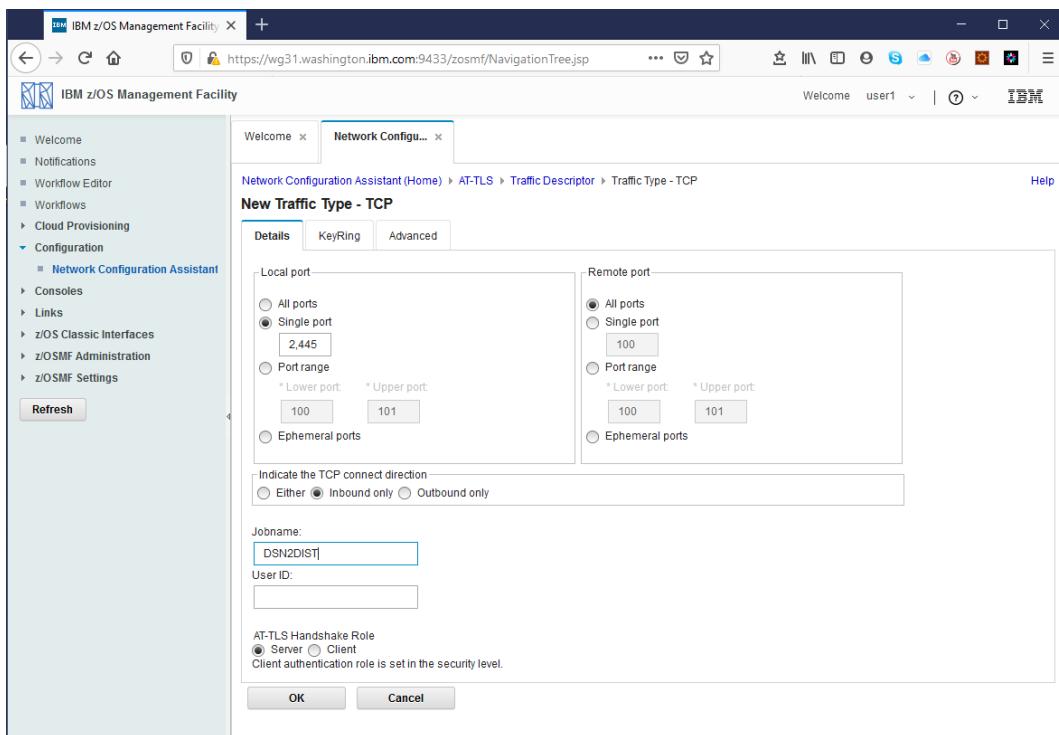
Total: 21 Selected: 0

At the bottom are "Home" and "Save" buttons.

15. On the *New Traffic Descriptor* window, enter **Db2Server** as the *Name*. Use the *Actions* pull-down and select *New* to start the definition of a new traffic descriptor type.

The screenshot shows the IBM z/OS Management Facility interface. The left sidebar has a tree view with 'Configuration' expanded, showing 'Network Configuration Assistant' selected. The main area has tabs for 'Welcome' and 'Network Configu...'. Below is a breadcrumb trail: Network Configuration Assistant (Home) > AT-TLS > Traffic Descriptor. A large heading says 'New Traffic Descriptor'. It includes a note about traffic descriptors containing details of traffic types mapped to security levels. A car icon with 'TN3270' is shown. A form has 'Name' set to 'Db2Server'. A table below is empty, stating 'There is no data to display.'

16. On the *New Traffic Type - TCP* window, select the radio button beside *Single ports* under *Local port* and enter **2445** as the port number. Select the radio button *All ports* under *Remote port*. Select the radio button beside *Inbound only* under *Indicate the TCP connection direction*. Enter **DSN2DIST** in the area under *Jobname* and finally select the radio button beside *Server* under *AT-TLS Handshake Role*. Next click on the *KeyRing* tab to continue.

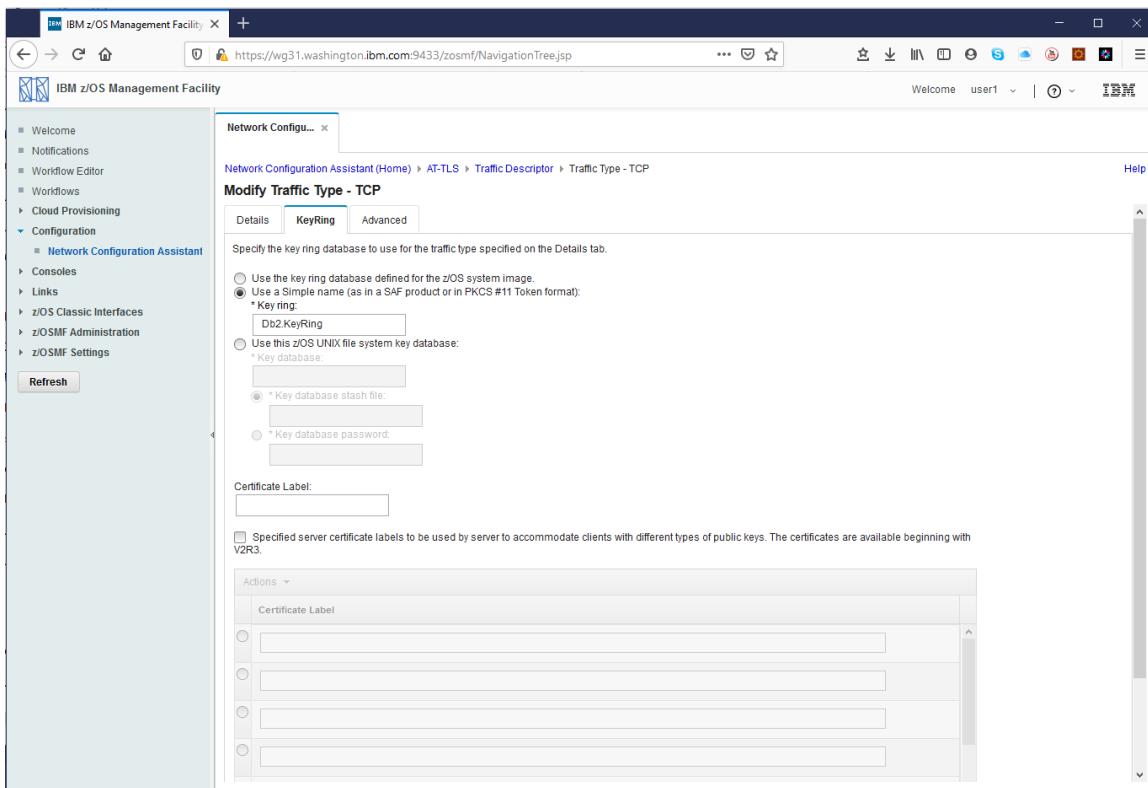


**Tech-Tip:** This traffic definition is triggered when a client attempts to connect to port 2445. Port 2445 was identified as the Db2 SECPORT in startup message.

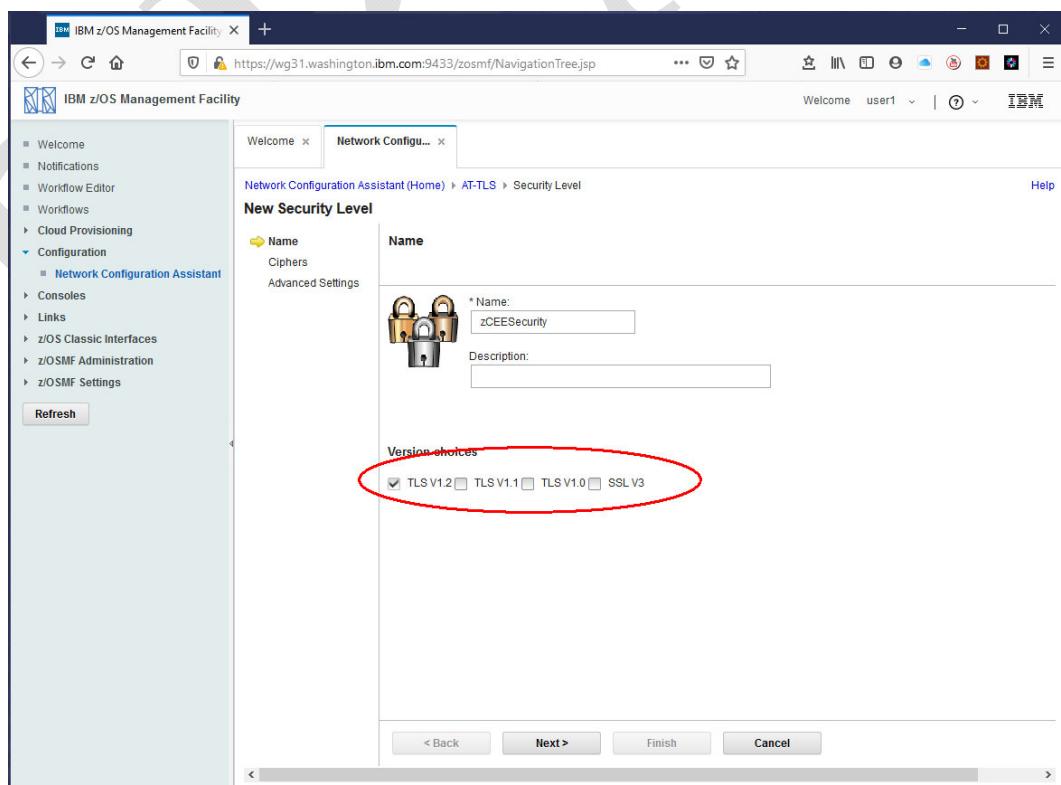
```
DSNL004I -DSN2 DDF START COMPLETE
LOCATION DSN2LOC
LU USIBMWZ.DSN2APPL
GENERICLU -NONE
DOMAIN WG31.WASHINGTON.IBM.COM
TCPPORT 2446
SECPORT 2445
RESPORT 2447
IPNAME -NONE
OPTIONS:
PKGREL = COMMIT
```

If all the defined conditions are met, AT-TLS will act as a surrogate for the server during a TLS handshake. Note the *KeyRing* tab can be used to specify the name of the key ring to be used for this handshake, e.g., Db2.KeyRing. Otherwise, the default is to use the same key ring name defined for the z/OS System image, e.g., Liberty.KeyRing.

17. On the *KeyRing* tab, select the radio button beside *Use a Simple name* and enter **Db2.KeyRing** as the key ring name. Click **OK** twice to continue.



18. Next, click the Security Levels tab and use the *Actions* pull-down button and select the *New* option. On the *New Security Level* windows, enter **zCEESecurity** for the *Name* and check the box beside *TLS V1.2* and uncheck the other boxes. Click **Next** to display the *Cipher selection* options. Click **Next** to display the *Advanced Settings* options exploring as you like but there is no need to make any changes. Click **Finish** to continue.



19. Next, click the *Requirement Maps* tab. Use the *Actions* pull-down button to select the *New* option.

The screenshot shows the IBM z/OS Management Facility interface. The left sidebar has a tree view with nodes like Welcome, Notifications, Workflow Editor, Workflows, Cloud Provisioning, Configuration (with Network Configuration Assistant selected), Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. The main area has tabs: Welcome, Workflows, Network Configu..., Network Configuration Assistant (Home) > AT-TLS, V2R3 Current Backing Store is USER1, Select a TCP/IP technology to configure : AT-TLS, Tools, Systems, Traffic Descriptors, Security Levels, Address Groups, Requirement Maps (selected). Below these tabs is a table with columns Name and Description. A row for 'AT-TLS\_Sample' is selected, showing 'IBM supplied: AT-TLS sample: CICS and TN3270'. At the bottom are buttons for Home and Save.

**Tech Tip:** The key ring specified here belongs to identity DB2USER This is the identity under which the z/OS Connect server is running. This ring has these certificates connected.

Certificate Label Name	Cert Owner	USAGE	DEFAULT
DB2 CA	CERTAUTH	CERTAUTH	NO
zCEE CA	CERTAUTH	CERTAUTH	NO
DB2USER	ID (DB2USER)	PERSONAL	YES

The z/OS Connect out bound JSSE key ring has these certificates connected.

Ring:			
>zCEE.KeyRing<			
Certificate Label Name	Cert Owner	USAGE	DEFAULT
zCEE CA	CERTAUTH	CERTAUTH	NO
Liberty CA	CERTAUTH	CERTAUTH	NO
zCEE Client Cert	ID (LIBSERV)	PERSONAL	YES
zCEE-CertAuth	CERTAUTH	CERTAUTH	NO
DB2 CA	CERTAUTH	CERTAUTH	NO

20. On the *New Requirement Map* window, enter ***Db2RequirementMap*** as the *Name*. Use the pull-down arrows to select *Db2Server* as the *Traffic Descriptor* and *zCEESecurity* as the *Security Level* for this map. Click **OK** to continue.

The screenshot shows the IBM z/OS Management Facility Network Configuration Assistant interface. The left sidebar has a tree view with nodes like Welcome, Notifications, Workflow Editor, Workflows, Cloud Provisioning, Configuration (selected), Network Configuration Assistant (selected), Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. A Refresh button is also present. The main area has tabs for Welcome and Network Configu... (which is selected). Below that, the Network Configuration Assistant (Home) > AT-TLS > Requirement Map path is shown. The title is "New Requirement Map". It includes a graphic of locks and a brief description: "A requirement map is an object that maps each IP traffic type (traffic descriptor) to a specific level of security (security level)." Instructions for adding a mapping are provided: "To add a new mapping to the requirement map: 1. Click the 'Add Row' action or use an existing row 2. Click a table cell to select a traffic descriptor from the list 3. Click a table cell to select a security level from the list". The "Name" field contains "Db2RequirementMap". The "Description" field is empty. Below is a "Mappings table" with columns for Actions, Traffic Descriptor, and Security Level. It shows one row where "Db2Server" is selected as the traffic descriptor and "zCEESecurity" is selected as the security level. There are two more rows below it, both with "Select a traffic descriptor" and "Select a security level" in their respective fields. At the bottom of the table, it says "Total: 3 Selected: 0".

21. Select the radio button beside *Db2RequirementMap* and use the *Actions* pull-down to select the *View Details* options to display the window below. Review the details and click the **Close** button to continue.

The screenshot shows the IBM z/OS Management Facility interface. The left sidebar has a tree view with nodes like Welcome, Notifications, Workflow Editor, Workflows, Cloud Provisioning, Configuration (selected), Network Configuration Assistant (under Configuration), Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. A Refresh button is also present. The main area has tabs: Welcome (selected) and Network Configu... (another tab). Below the tabs, the URL https://wg31.washington.ibm.com:943 is shown. The main content area displays the 'View Details' window for 'Application / Requirement Map: Db2RequirementMap'. It includes a 'Requirement map summary' table:

Traffic Descriptor	AT-TLS Security Level
Db2Server	zCEESecurity

Below this is a 'Requirement Map traffic - Shown in Configured Order' table:

Traffic Descriptor		AT-TLS Security Level				
Name	Protocol	Local Port	Remote Port	Connect Direction	Name	Ciphers
Db2Server	TCP	2445	All Ports	Inbound	zCEESecurity	---

Below these tables is a section titled 'Security Level Details' with a sub-section 'Security Level: zCEESecurity' containing the following details:

Type:  
AT-TLS  
Encryption:  
System SSL V3 Defaults  
Use TLS Version 1.0:

At the bottom of the window are 'Close' and 'Back to Top' buttons.

22. Click the **Save** button to save the configuration.

23. When the save has completed, click on the *Systems* tab to return to this window.

The screenshot shows the IBM z/OS Management Facility Network Configuration Assistant window. The left sidebar navigation menu includes Welcome, Notifications, Workflow Editor, Workflows, Cloud Provisioning, Configuration (with Network Configuration Assistant selected), Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. A Refresh button is also present. The main content area has tabs for Welcome, Network Configu..., and Network Configuration Assistant (Home) > AT-TLS. A message at the top says "V2R3 Current Backing Store is USER1". Below this, a dropdown menu says "Select a TCP/IP technology to configure : AT-TLS". The main panel displays a table titled "Systems" with columns: System Group or Sysplex / System Image / Stack, Type Filter, Status Filter, Install Status Filter, and Release Filter. The table contains three rows: Default (System Group, Complete, N/A, V2R3), WG31 (System Image, Complete, N/A, V2R3), and TCPIP1 (Stack, Incomplete, N/A, V2R3). At the bottom of the panel are "Home" and "Save" buttons. The status bar at the bottom of the window shows "Total: 3 Selected: 1".

24. Select the radio button beside *TCP1P1* and use the *Actions* pull-down to select *Rules*. This is where these definitions will be tied together. Use the *Actions* pull-down again and select *New* to create a new connectivity rule. Enter ***Db2ServerRule*** for the *Connectivity rule name* and press **Next** to continue.

The screenshot shows the IBM z/OS Management Facility Network Configuration Assistant interface. The left sidebar lists various management options like Welcome, Notifications, Workflows, Cloud Provisioning, Configuration, Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. The main panel is titled 'New Connectivity Rule' under the 'Data Endpoints' tab. It prompts for a 'Connectivity rule name' which is set to 'Db2ServerRule'. Below this, it asks to 'Select the address groups of the host endpoints of the traffic you want to protect'. Under 'Local data endpoint', the 'Address group' radio button is selected, with 'All\_IPv4\_Addresses' chosen from a dropdown. There is also an option for 'IPv4 or IPv6 address, subnet, or range' with a text input field and examples provided. A similar section for 'Remote data endpoint' is shown on the right. At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

25. On the *New Connectivity Rule – Requirement Map* window, select the radio button beside *Select an existing requirement map* and use the pull-down to select *Db2RequirementMap*. This should automatically populate the mapping table with *Db2Server* as the traffic descriptor and *zCEESecurity* as the security level. Press **Next** and then **Finish** to continue.

The screenshot shows the IBM z/OS Management Facility Network Configuration Assistant interface. The left sidebar has a tree view with 'Network Configuration Assistant' selected. The main area shows the 'New Connectivity Rule' step. Under 'Requirement Map', the 'Select an existing requirement map' radio button is selected, and 'Db2RequirementMap' is listed in the dropdown. A 'Mappings table' below shows a single entry: 'Db2Server' in the Traffic Descriptor column and 'zCEESecurity' in the Security Level column. Navigation buttons at the bottom include '< Back', 'Next >', 'Finish', and 'Cancel'.

Traffic Descriptor	Security Level
Db2Server	zCEESecurity

26. Press **Close** to return to this window. Note that the status of the configuration is now complete.

The screenshot shows the IBM z/OS Management Facility Network Configuration Assistant interface. The left sidebar has a tree view with nodes like Welcome, Notifications, Workflow Editor, Workflows, Cloud Provisioning, Configuration (selected), Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. A 'Refresh' button is at the bottom of the sidebar. The main area has tabs for Welcome and Network Configuration (selected). Below that, it says 'Network Configuration Assistant (Home) > AT-TLS'. It displays 'V2R3 Current Backing Store is USER1'. A dropdown menu 'Selected a TCP/IP technology to configure : AT-TLS' is open. Below it is a table with columns: Actions, System Group or Sysplex / System Image / Stack Filter, Type Filter, Status Filter, Install Status Filter, and Release Filter. There are three rows: Default (System Group, Complete, N/A, N/A), WG31 (System Image, Complete, N/A, V2R3), and TCPPI1 (Stack, Complete, Never installed, V2R3). The TCPPI1 row is selected, indicated by a dashed blue border around its cells. At the bottom of the table, it says 'Total: 3 Selected: 1'. Buttons for Home and Save are at the bottom of the main area.

27. Select the radio button beside *TCPPI1* and use the *Actions* pull-down to select *Install All Files for This Group*.

28. On the *List of Configuration Files for All Systems Images in Group Default* window, select *WG31* and use the *Actions* pull-down to select *Install*.

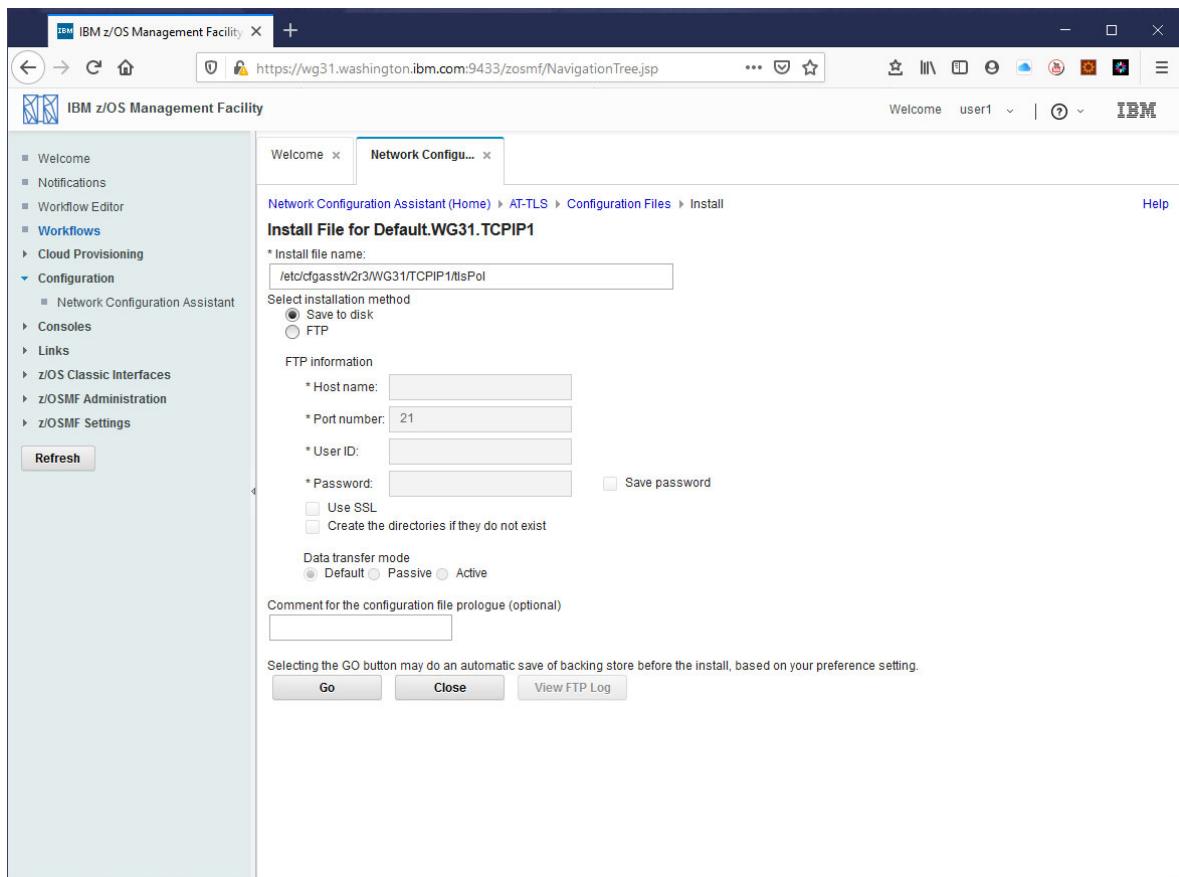
The screenshot shows the IBM z/OS Management Facility interface. The left sidebar has a tree view with nodes like Welcome, Notifications, Workflow Editor, Workflows (selected), Cloud Provisioning, Configuration (selected), Network Configuration Assistant, Consoles, Links, z/OS Classic Interfaces, z/OSMF Administration, and z/OSMF Settings. The main area has tabs for Welcome and Network Configu... (which is selected). Below that is a breadcrumb trail: Network Configuration Assistant (Home) > AT-TLS > Configuration Files. The title is "List of Configuration Files for All System Images In Group Default". A table lists one row:

Action	System Image	Configuration Type	Status	Last Install	Configured File Name	Configured Host Name	Configured Installation Method
Actions	WG31	TCPPIP1 - AT-TLS Policy	Never installed	Never	/etc/cfgasst /V2r3WG31 /TCPPIP1/tlsPol		Save to disk

Total: 1 Selected: 1

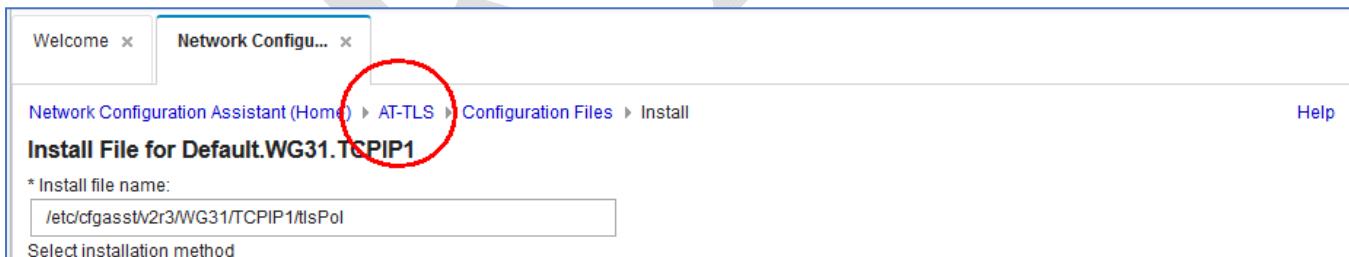
Close

29. On the *Install File for Default.WG31.TCPIP1* window, click the **GO** button to continue.



30. Click **OK** twice to continue.

31. Next, click on *AT-TLS* as shown below to return to the primary window.



32. The AT-TLS configuration has been completed and is installed, but it is not active yet.

## Activating the AT-TLS configuration

The AT-TLS configuration has been saved in an OMVS file but has not been installed in the active policy agent task (e.g., PAGENT).

1. This instance of the policy agent has been configured to use the *SYSLOGD* daemon task to log messages.

```
//PAGENT EXEC PGM=PAGENT,REGION=0K,TIME=NOLIMIT,
//      PARM='ENVAR("_CEE_ENVFILE_S=DD:STDENV")/ -I SYSLOGD'
```

2. The *SYSLOGD* daemon has been configured to write all log messages to file */var/syslogd/syslogall.log* (see the *syslog.conf* file in the */etc* subdirectory) as shown below:

```
#####
#
# Write all messages with priority err and higher to log file errors.
#
#*.err          /var/log/%Y/%m/%d/errors
*.*            /var/syslogd/syslogall.log
#
```

3. Use ISPF option 3.4 to access directory */var/syslogd* and the *v* line command to view *syslogall.log*. Go to the bottom of the file and you will see something like what is shown below:

```

VIEW      /SYSTEM/var/syslogd/syslogall.log          Columns 00063 00134
Command ==> - Using catalog '/usr/lib/nls/msg/C/ftpmsg.cat' for FTP messages.
003388 YFT18I Using catalog '/usr/lib/nls/msg/C/ftpmsg.cat' for FTP messages.
003389 Y2697I IBM FTP CS V2R3 19:44:07 on 03/23/20
003390 Y2640I Using dd:SYSFTP=SYS1.TCPPARMS(FTPDATA) for local site configurat
003391 YFT47I dd:SYSFTP=SYS1.TCPPARMS(FTPDATA) file, line 10: Ignoring keyword
003392 YFT47I dd:SYSFTP=SYS1.TCPPARMS(FTPDATA) file, line 11: Ignoring keyword
003393 YFT47I dd:SYSFTP=SYS1.TCPPARMS(FTPDATA) file, line 13: Ignoring keyword
003394 YFT47I dd:SYSFTP=SYS1.TCPPARMS(FTPDATA) file, line 49: Ignoring keyword
003395 YFT47I dd:SYSFTP=SYS1.TCPPARMS(FTPDATA) file, line 54: Ignoring keyword
003396 YFT21I Using catalog '/usr/lib/nls/msg/C/ftpdrply.cat' for FTP replies.
003397 YFT26I Using 7-bit conversion derived from 'ISO8859-1' and 'IBM-1047' fo
003398 YFT32I Using the same translate tables for the control and data connecti
003399 YFT09I system information for WG31: z/OS version 2 release 3 (3906)
003400 pFixLevel: Fix level: NONEFND Data: EZB0ECPR
003401 pFixLevel: Fix level: HIP6230 Data: EZAFTPDA EZAFTPDI EZAFTP4 EZAFTPAG
003402 pFixLevel: Fix level: " Data: EZAFTPQ1 EZAFTPXC EZAFTPBU EZAFTPDF
003403 pFixLevel: Fix level: " Data: EZAFTPDH EZAFTPDM EZAFTPEA EZAFTPED
003404 pFixLevel: Fix level: " Data: EZAFTPJE EZAFTPFR EZAFTPET EZAFTPGU
003405 pFixLevel: Fix level: " Data: EZAFTPQV EZAFTPNX EZAFTPSD EZAITUTI
003406 pFixLevel: Fix level: UI53145 Data: EZAFTPNY
003407 pFixLevel: Fix level: UI56159 Data: EZAFTPEP
003408 pFixLevel: Fix level: UI57631 Data: EZAFTP5
003409 pFixLevel: Fix level: 24/ 24 Data: OBJECTS PROCESSED. AV-BUFR: 0005087
003410 Y2700I Using port FTP control (21)
003411 Y2701I Inactivity time is 12000
003412 Y2702I Server-FTP: Initialization completed at 19:44:07 on 03/23/20.
003413 YFT41I Server-FTP: process id 83886182, server job name FTPSERVE
003414 nning on 0.0.0.0 port 22.
***** Bottom of Data *****

```

4. Start the policy agent task using MVS command ***S PAGENT***
5. Exit the *syslogall.log* view session and reopen the file do a find for sting ***EZZ8431I PAGENT STARTING***. You should see these messages.

```
003414 0.0.0 port 22.
003415 main: EZZ8431I PAGENT STARTING
003416 main: Compiled on Sep 26 2016 at 18:37:59
003417 main: Use environment PAGENT_CONFIG_FILE = '/etc/pagent.conf'
003418 main: List all environment variables:
003419 main:   EXPORT '_CEE_ENVFILE S=DD:STDENV'
003420 main:   EXPORT 'LIBPATH=/usr/lib:.'
003421 main:   EXPORT 'PAGENT_CONFIG_FILE=/etc/pagent.conf'
003422 main:   EXPORT 'PAGENT_LOG_FILE=SYSLOGD'
003423 main:   EXPORT 'TZ=EST5EDT'
003424 main:   EXPORT 'GSK_TRACE=0xFFFF'
003425 main: using code page 'IBM-1047'
003426 main: Using log level 511
```

6. Do a find for the character string TTLSRule, e.g. ***f TTLSRule*** and you see multiple occurrences where the AT-TLS configuration elements added earlier are being processed.

```
003515 _profile: Processing Image TTLS config file: '/etc/cfgasst/v2r3/WG31/
003516 Processing: 'TTLSRule                                     Db2ServerRule~1'
003517 Processing: 'TTLSSGroupAction                         gAct1~Db2Server'
003518 Processing: 'TTLSEnvironmentAction                   eAct1~Db2Server'
003519 Processing: 'TLSConnectionAction                     cAct1~Db2Server'
003520 Processing: 'TLSConnectionAdvancedParms             cAdv1~Db2Server'
003521 Processing: 'TLSKeyringParms                        keyR~WG31'
003522 Processing: 'IpAddrSet                            addr1'
003523 Processing: 'PortRange                           portR1'
003524 Processing: 'PortRange                           portR2'
003525 _profile: Finished processing Image TTLS config file
003526 Processing TTLS Group action 'gAct1~ Db2Server'
003527 Processing TTLS Connection action 'cAct1~Db2Server'
003528 Processing TTLS Environment action 'eAct1~Db2Server'
003529 ocessing TTLS rule 'Db2Server~1'
```

7. Go the bottom of this file and you see these messages

```
EZD1579I PAGENT POLICIES ARE NOT ENABLED FOR TCPIP1 : TTLS
EZZ8771I PAGENT CONFIG POLICY PROCESSING COMPLETE FOR TCPIP1 : QOS
EZZ8771I PAGENT CONFIG POLICY PROCESSING COMPLETE FOR TCPIP1 : TTLS
EZD1586I PAGENT HAS INSTALLED ALL LOCAL POLICIES FOR TCPIP1
Finished main config file update
```

**Tech-Tip:** If a policy or otherwise changed the new or updated policy can be installed with an MVS modify command, ***F PAGENT,REFRESH***

8. The policy agent is active. The policies have been loaded, but there is one remaining step. The TCPIP stack has not been modified to enable TTLS. On this image this has been configured this way so the AT-TLS is disabled by default and must be explicitly enabled. This is done by an MVS VARY command.

**V TCPIP,,OBEY,SYS1.TCPPARMS(TTLS)**

Where the contents of SYS1.TCPPARMS(TTLS) is simply TCPCONFIG TTLS

Issue this command and you should see these messages in the console, see below:

**V TCPIP,,OBEY,SYS1.TCPPARMS (TTLS)**

```
EZZ0060I PROCESSING COMMAND: VARY TCPIP,,OBEY,SYS1.TCPPARMS (TTLS)
EZZ0300I OPENED OBEYFILE FILE 'SYS1.TCPPARMS (TTLS)'
EZZ0309I PROFILE PROCESSING BEGINNING FOR 'SYS1.TCPPARMS (TTLS)'
EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'SYS1.TCPPARMS (TTLS)'
EZZ0053I COMMAND VARY OBEY COMPLETED SUCCESSFULLY
```

**Tech-Tip:** AT-TLS can also be disabled with a VARY command,  
**V TCPIP,,OBEY,SYS1.TCPPARMS(NOTTLS)** where the contents of SYS1.TCPPARMS(NOTTLS) is  
TCPCONFIG NOTTLS

## **Test the TLS connection from the z/OS Connect Designer to a Db2 subsystem**

In the following steps you will download the Db2 CERTAUTH public key certificate exported from RACF in a previous section and add it to a *Designer* trust key store that will be used to perform a TLS handshake with Db2 (via AT-TLS) and used to access Db2 from the *Designer* using TLS.

- \_\_\_ 1. On the Windows desktop, open DOS command prompt by clicking on the DOS icon.
- \_\_\_ 2. Enter the command: **cd c:\z\openApi3\certs**
- \_\_\_ 3. Enter the command: **ftp wg31.washington.ibm.com**
- \_\_\_ 4. Logon with *USER1* and the password for that ID
- \_\_\_ 5. Enter the command: **prompt off**
- \_\_\_ 6. Enter the command: **mget db2ca.pem**
- \_\_\_ 7. Enter the command: **bin**
- \_\_\_ 8. Enter the command: **mget designer.p12**
- \_\_\_ 9. Enter the command: **quit**

```
Microsoft Windows [Version 10.0.19044.1766]
(c) Microsoft Corporation. All rights reserved.

c:\z>cd c:\z\openApi3

c:\z\openapi3>ftp wg31.washington.ibm.com
Connected to wg31.washington.ibm.com.
220-FTP 13:34:13 on 2022-07-05.
220 Connection will close if idle for more than 200 minutes.
501 command OPTS aborted -- no options supported for UTF8
User (wg31.washington.ibm.com:(none)): user1
331 Send password please.
Password:
230 USER1 is logged on. Working directory is "USER1.".
ftp> prompt
Interactive mode Off .
ftp> mget db2ca.pem
200 Representation type is Ascii NonPrint
200 Port request OK.
125 Sending data set USER1.DB2CA.PEM
250 Transfer completed successfully.
ftp: 1244 bytes received in 0.00Seconds 414.67Kbytes/sec.
ftp> bin
200 Representation type is Image
ftp> mget designer.p12
200 Representation type is Image
200 Port request OK.
125 Sending data set USER1.DESIGNER.P12
250 Transfer completed successfully.
ftp: 3416 bytes received in 0.11Seconds 31.05Kbytes/sec.
ftp> quit
221 Quit command received. Goodbye.
```

**Tech-Tip:** The significance of the P12 file extensions of the certificate is that this extension usually indicates the file contains a private key and will require a password or pass phrase to be able to access this certificate.

- \_\_\_ 10. On the Windows desktop, open an Ubuntu terminal shell by clicking on the *Ubuntu* icon in the task bar.
- \_\_\_ 11. In the Ubuntu session, switch to *root* access by entering the command: *sudo su*
- \_\_\_ 12. Enter root's password of *passw0rd*.

### ***Using a JSSE key store from the Designer***

In this section a JSSE trust store will be created for use in TLS handshakes with Db2. Note that JSSE supports the use of the same or different key store files for both the personal and CERTAUTH repositories.

First, create a Db2 trust store and import the Db2 CA PEM into the JSSE trust store.

- \_\_\_ 1. In an Ubuntu session, copy the CERTAUTH public key certificate file (*DB2CA.PEM*) from Windows into the Linux directory mapped to a container's directory (see the *docker-compose.yaml* file).
- ```
cp /mnt/c/z/openApi3/certs/DB2CA.PEM /home/workstation/docker/employees/certs
```
- \_\_\_ 2. In an Ubuntu session, import the CERTAUTH public key into the container's Db2 trust key store (this will create the JSSE keystore file).

```
docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/db2TrustStore.jks -storetype PKCS12 -storepass changeit -importcert -file /output/resources/security/DB2CA.PEM -alias Db2CA
```

**Tech-Tip:** The command, *docker exec -it cscvinc\_zosConnect\_1* provides a command interface for invoking command in the container. In this case, the *docker exec* command is being used to invoke the *keytool* command in the container.

Note that there will be a prompt as to whether the certificate is to be trusted or not, reply **yes**.

```
root:/home/workstation/docker/employees:> docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/db2TrustStore.jks -storetype PKCS12 -storepass changeit -importcert -file /output/resources/security/DB2CA.PEM -alias Db2CA
Owner: CN=Db2 CA, OU=ATS, O=IBM
Issuer: CN=Db2 CA, OU=ATS, O=IBM
Serial number: 0
Valid from: 7/1/22 5:00 AM until: 1/1/23 4:59 AM
Certificate fingerprints:
    MD5: 3A:40:CE:00:0C:84:36:D4:77:4F:7A:61:92:73:6C:F8
    SHA1: 6A:92:D9:C4:9C:A8:D5:5E:AB:5D:1D:82:1D:AF:5D:7D:63:8F:47:1E
    SHA256: CF:6F:A2:B7:CA:FA:56:44:ED:A8:6A:07:FD:E2:05:3D:FF:78:D2:88:C0:87:C0:D3:9E:0A:79:71:7A:18:AF:A2
    Signature algorithm name: SHA256withRSA
    Version: 3

Extensions:

#1: ObjectId: 2.16.840.1.113730.1.13 Criticality=false
0000: 16 30 47 65 6e 65 72 61 74 65 64 20 62 79 20 74 .0Generated.by.t
0010: 68 65 20 53 65 63 75 72 69 74 79 20 53 65 72 76 he.Security.Serv
0020: 65 72 20 66 6f 72 20 7a 2f 4f 53 20 28 52 41 43 er.for.z.OS..RAC
0030: 46 29 F.

#2: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  Key_CertSign
  Crl_Sign
]
#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
  KeyIdentifier [
  0000: be f3 b3 48 61 b0 07 fd d0 d4 e0 da 0d b5 4f 4b ...Ha.....OK
  0010: a9 6c 61 27 .la.
]
]
#4: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

Trust this certificate? [no]: yes
Certificate was added to keystore
```

### 3. To view the contents of this key store using this command

```
docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/db2TrustStore.jks -storetype PKCS12 --storepass changeit -list
```

```
root:/home/workstation/docker/employees:> docker exec -it employees_zosConnect_1 keytool -list -keystore /output/resources/security/db2TrustStore.jks -storetype PKCS12 --storepass changeit

Keystore type: PKCS12
Keystore provider: IBMJCE

Your keystore contains 1 entries

db2ca, Jul 1, 2022, trustedCertEntry,
Certificate fingerprint (SHA1): 6A:92:D9:C4:9C:A8:D5:5E:AB:5D:1D:82:1D:AF:5D:7D:63:8F:47:1E
```

This JSSE keystore contains one trustedCertEntry(CERTAUTH) certificate (labeled *db2ca*)

4. Now list the contents of the details of the *db2ca* certificate using this command

```
docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/db2TrustStore.jks -storetype PKCS12 --storepass changeit -list -alias db2ca -v
```

```
root:/home/workstation/docker/employees:> docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/db2KeyStore.jks -storetype PKCS12 --storepass changeit -list -alias db2ca -v
Alias name: db2ca
Creation date: Jul 1, 2022
Entry type: trustedCertEntry

Owner: CN=Db2 CA, OU=ATS, O=IBM
Issuer: CN=Db2 CA, OU=ATS, O=IBM
Serial number: 0
Valid from: 7/1/22 5:00 AM until: 1/1/23 4:59 AM
Certificate fingerprints:
MD5: 3A:40:CE:00:0C:84:36:D4:77:4F:7A:61:92:73:6C:F8
SHA1: 6A:92:D9:C4:9C:A8:D5:5E:AB:5D:1D:82:1D:AF:5D:7D:63:8F:47:1E
SHA256: CF:6F:A2:B7:CA:FA:56:44:ED:A8:6A:07:FD:E2:05:3D:FF:78:D2:88:C0:87:C0:D3:9E:0A:79:71:7A:18:AF:A2
Signature algorithm name: SHA256withRSA
Version: 3

Extensions:
#1: ObjectId: 2.16.840.1.113730.1.13 Criticality=false
0000: 16 30 47 65 6e 65 72 61 74 65 64 20 62 79 20 74 .0Generated.by.t
0010: 68 65 20 53 65 63 75 72 69 74 79 20 53 65 72 76 he.Security.Serv
0020: 65 72 20 66 6f 72 20 7a 2f 4f 53 20 28 52 41 43 er.for.z.OS..RAC
0030: 46 29 F.

#2: ObjectId: 2.5.29.15 Criticality=true
KeyUsage [
  Key_CertSign
  Crl_Sign
]
#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdIdentifier [
  KeyIdentifier [
    0000: be f3 b3 48 61 b0 07 fd d0 d4 e0 da 0d b5 4f 4b ...Ha.....OK
    0010: a9 6c 61 27 .la.
  ]
]
#4: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]
```

In this output we are seeing that even though it is a self-signed certificate also (refer to the distinguished names) we imported this certificate as a trustedCertEntry (e.g., CERTAUTH) certificate. Also note that this certificate has *Key\_CertSign* as a *KeyUsage* attribute. This means that it can be used to validate personal certificates signed by the *Issuer, CN=Db2 CA, OU=ATS, O=IBM*

This is a diagram of what we just created.



This is the minimum required to enable sever authentication TLS support between a *Designer* container and Db2.

## Server certificate authentication

The container in which the *Designer* is running needs to be updated with new environment variables. Adding environment variables to a container requires updating the *docker-compose.yaml* file with the new variables and the stopping and then restarting the container with a *docker-compose* or *podman-compose* commands technique.

Update the *docker-compose.yaml* file to add three new environment variables, *DB2SSL\_PORT* and *DB2TRUSTSTORE\_PASSWORD*, *DB2KEYSTORE\_PASSWORD* with values as shown below:

```
version: "3.2"
services:
  zosConnect:
    image: icr.io/zosconnect/ibm-zcon-designer:3.0.57
    user: root
    environment:
      - CICS_USER=USER1
      - CICS_PASSWORD=USER1
      - CICS_HOST=wg31.washington.ibm.com
      - CICS_PORT=1491
      - DB2_USERNAME=USER1
      - DB2_PASSWORD=USER1
      - DB2_HOST=wg31.washington.ibm.com
      - DB2_PORT=2446
      - DB2TRUSTSTORE_PASSWORD=changeit
      - DB2KEYSTORE_PASSWORD=secret
      - DB2SSL_PORT=2445
      - HTTP_PORT=9080
    ports:
      - "9449:9443"
      - "9086:9080"
    volumes:
      - ./project:/workspace/project
      - ./logs/:/logs/
      - ./certs:/output//resources/security/
```

**Tech Tip:** The value for the trust store password (*changeit*) is the value of *-STORPASS* used in the keytool command when the trust store was initially created. The value for the key store password is *secret* (the encryption password provided by the certificate authority and used in the earlier *RACDDERT ADD* commands). The *Db2KEYSTORE* environment variable will be used in a later section of the exercise.

1. If you are comfortable with the vi editor, the *docker-compose.yaml* file in the container's base Linux directory can be updated directly and then proceed to Step 4. Otherwise, using an Ubuntu session, copy the container's *docker-compose.yaml* file to a Windows directory using command:

```
cp /home/workstation/docker/employees/docker-compose.yaml /mnt/c/openapi3/
```

2. Update the *docker-compose.yaml* file as described above using your favorite Windows editor.  
 3. In an Ubuntu session, copy the update YAML file back to the Linux directory for this container.

```
cp /mnt/c/openapi3/docker-compose.yaml /home/workstation/docker/employees
```

4. The container needs to be stopped and restarted in order for the new environment variables to be recognized. Stop the container with command:

```
docker stop employees_zosConnect_1
```

5. Restart the container with this command (*invoke this command while in directory /home/workstation/docker/employees*).

```
docker-compose up -d
```

Now the *zosconnect.db2Connection* entry for the Db2 connection needs to have additional attributes added for TLS.

6. Copy the Windows file *db2tlsServer.xml* to the container configuration directory as file *db2.xml*.

```
cp /mnt/c/z/openapi3/tls/db2TlsServer.xml  
/home/workstation/docker/employees/project/src/main/liberty/config/db2.xml
```

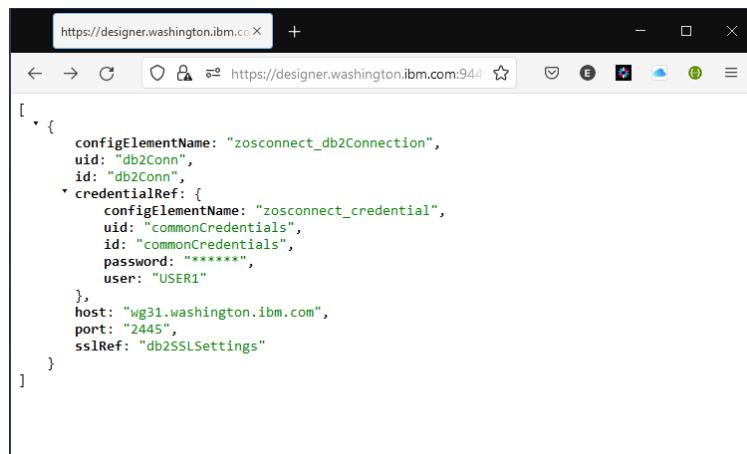
Below are the contents of *db2TlsServer.xml*. The boldfaced lines are the additions made to the original contents of db2.xml for adding TLS support between the *Designer* and Db2.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="DB2 SSL">
  <featureManager>
    <feature>zosconnect:db2-1.0</feature>
  </featureManager>
  <zosconnect_credential id="commonCredentials" />
    user="${DB2_USERNAME}" password="${DB2_PASSWORD}" />
  <zosconnect_db2Connection id="db2Conn">
    host="${DB2_HOST}" port="${DB2_PORT}"
    credentialRef="commonCredentials"
    sslRef="db2SSLSettings" />
    <ssl id="db2SSLSettings"
      keyStoreRef="db2TrustStore"
      trustStoreRef="db2TrustStore" />
  <keyStore id="db2TrustStore"
    location="/output/resources/security/db2TrustStore.jks"
    password="${DB2TRUSTSTORE_PASSWORD}" type="PKCS12" />
</server>
```

The above configuration defines a TLS repertoire (*db2SSLSettings*) for use when connection to a Db2. Note the *sslRef* in the *zosconnect\_db2Connection* configuration elements references *db2SSLSettings*. The TLS repertoire has attributes *keyStoreRef* and *trustStoreRef* referencing the same *keystore* element. This is where we could have made a distinction between personal certificates and CERTAUTH key store files. The *keystore* element, *db2KeyStore*, identifies the key store we created in the previous section. This key store contains our self-signed personal certificate and the Db2 *CERTAUTH* certificate.

7. Enter the URL below in a browser to verify the updated `zosconnect_db2Connection` configuraton

[https://designer.washington.ibm.com:9449/ibm/api/config/zosconnect\\_db2Connection](https://designer.washington.ibm.com:9449/ibm/api/config/zosconnect_db2Connection)



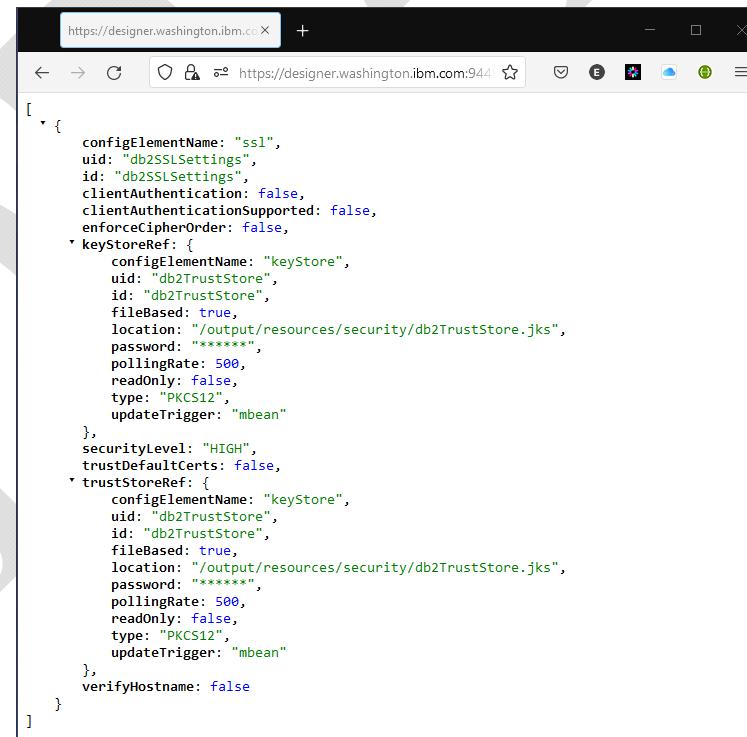
```
[{"configElementName": "zosconnect_db2Connection", "uid": "db2Conn", "id": "db2Conn", "credentialRef": {"configElementName": "zosconnect_credential", "uid": "commonCredentials", "id": "commonCredentials", "password": "*****", "user": "USER1"}, "host": "wg31.washington.ibm.com", "port": "2445", "sslRef": "db2SSLSettings"}]
```

If a logon is required, use **Fred** as the *Username* and **fredpwd** as the *Password*.

**Tech-Tip:** Installing the *restConnector-2.0* feature and providing an *administrator-role* configuration element, (see the *basicSecurity.xml file*), enabled the displaying of configuration elements in a web browser. This is good method for reviewing for syntax or other configuration errors in the installed server XML configuration element names particularly when other configuration elements are referenced.

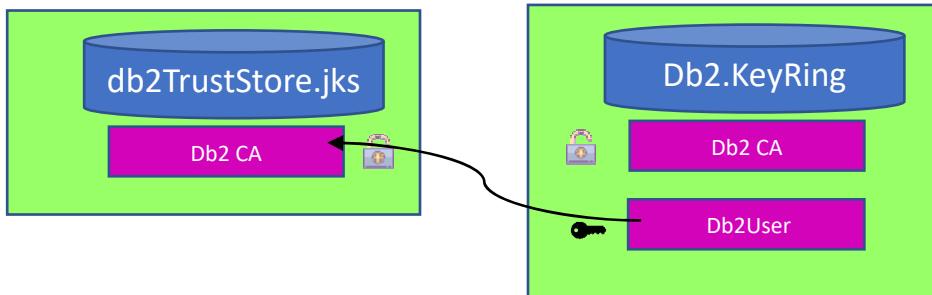
Enter the URL below in a browser to verify the updated `zosconnect_db2Connection` configuraton

<https://designer.washington.ibm.com:9449/ibm/api/config/ssl?id=db2SSLSettings>



```
[{"configElementName": "ssl", "uid": "db2SSLSettings", "id": "db2SSLSettings", "clientAuthentication": false, "clientAuthenticationSupported": false, "enforceCipherOrder": false, "keyStoreRef": {"configElementName": "keyStore", "uid": "db2TrustStore", "id": "db2TrustStore", "fileBased": true, "location": "/output/resources/security/db2TrustStore.jks", "password": "*****", "pollingRate": 500, "readOnly": false, "type": "PKCS12", "updateTrigger": "mbean"}, "securityLevel": "HIGH", "trustDefaultCerts": false, "trustStoreRef": {"configElementName": "keyStore", "uid": "db2TrustStore", "id": "db2TrustStore", "fileBased": true, "location": "/output/resources/security/db2TrustStore.jks", "password": "*****", "pollingRate": 500, "readOnly": false, "type": "PKCS12", "updateTrigger": "mbean"}, "verifyHostname": false}]
```

This diagram shows the TLS handshake flow as it currently configured. When a client requests a connection, Db2 (via AT-TLS) will send its server certificate to the client where it will be validated with a local CERTAUTH certificate. The client will validate the server certificate and an encrypted connection will be established (this is not mutual authentication).



To directly invoke a Db2 REST service, entering this curl command in a Windows DOS command session in directory c:\z\openapi3 with the **-v** trace will display the handshake flow in the **bold red** below:

```
curl -X POST -w "- HTTP CODE %{http_code}" --user user1:user1 --header "Content-Type: application/json" --data @selectEmployee.json --cacert certs\DB2CA.PEM https://wg31.washington.ibm.com:2445/services/zCEEService/selectEmployee -v
```

```
c:\z\openapi3>curl -X POST -w "- HTTP CODE %{http_code}" --user user1:user1 --header "Content-Type: application/json" --data @selectEmployee.json --cacert certs\DB2CA.PEM https://wg31.washington.ibm.com:2445/services/zCEEService/selectEmployee -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 9.82.31.201:2445...
* Connected to wg31.washington.ibm.com (9.82.31.201) port 2445 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*   CAfile: DB2CA.pem
*   Capath: none
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Finished (20):
*   SSL connection using TLSv1.2 / AES256-SHA
* ALPN, server did not agree to a protocol
* Server certificate:
*   subject: O=IBM; OU=ATS; CN=wg31.washington.ibm.com
*   start date: Jul 1 05:00:00 2022 GMT
*   expire date: Jan 1 04:59:59 2023 GMT
*   common name: wg31.washington.ibm.com (matched)
*   issuer: O=IBM; OU=ATS; CN=Db2 CA
*   SSL certificate verify ok.
* Server auth using Basic with user 'user1'
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
> POST /services/zCEEService/selectEmployee HTTP/1.1
> Host: wg31.washington.ibm.com:2445
> Authorization: Basic dXNlcjE6dXNlcjE=
> User-Agent: curl/7.82.0
> Accept: */
> Content-Type: application/json
> Content-Length: 34
>
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Connection: keep-alive
< Content-Length: 227
< Content-Type: application/json; charset=UTF-8
< Date: Sat, 2 Jul 2022 20:25:13 GMT
< X-Powered-By: DB2 for z/OS
< Server: DB2 DDF Native REST, DSN2LOC, DSNLJEMG 10/02/19 UI65644
< Content-Language: en-US
< X-Correlation-ID: COA8006A.G887.DBBE89586028
<
{
  "ResultSet": [
    {
      "employeeNumber": "000050",
      "firstName": "JOHN",
      "middleInitial": "B",
      "lastName": "GEYER",
      "department": "E01",
      "phoneNumber": "6789",
      "job": "MANAGER"
    }
  ],
  "statusCode": 200,
  "statusDescription": "Execution Successful"
}
* Connection #0 to host wg31.washington.ibm.com left intact
- HTTP CODE 200
```

Note the server certificate subject distinguished name in the trace matches the distinguished name of the personal certificate connect to the Db2 key ring. Also note that the issuer or signer of the personal certificate is the Db2 CERTAUTH.

Test the TLS connection between the *Designer* and Db2 using a curl command.

```
curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://designer.washington.ibm.com:9449/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://designer.washington.ibm.com:9449/employees/details/000010
>{"retrievedResults Output": [{"employeeID": "000010", "name": "CHRISTINE I HAAS", "departmentCode": "A00", "phoneNumber": "3978", "dateOfHire": "1965-01-01", "job": "PRES", "educationLevel": 18, "sex": "F", "dateOfBirth": "1933-08-14", "annualSalary": 52750.0, "lastBonus": 1000.0, "lastCommision": 4220.0}]} - HTTP CODE 200
```

A review of the AT-TLS trace shows a successful handshake.

```
Jul 1 19:57:17 wg31/TCPIP    TCPIP1    TTLS[345]: 14:57:17 TCPIP1    EZD1281I TTLS Map    CONNID: 00002F0F LOCAL: 192.168.17.201..2445 REMOTE: 192.168.0.106..1570 JOBNAME: DSN2DIST USERID: DB2USER TYPE: InBound STATUS: Enabled RULE: Db2ServerRule~1 ACTIONS: gAct1~Db2Server eAct1~Db2Server cAct1~Db2Server
Jul 1 19:57:17 wg31/TCPIP    TCPIP1    TTLS[345]: 14:57:17 TCPIP1    EZD1283I TTLS Event GRPID: 00000001 ENVID: 00000003 CONNID: 00002F0F RC: 0 Initial Handshake 0000005011527E10 0000005011522750 TLSV1.2 F0F0F3F5
```

And a review of the Liberty traces shows the handshake flow.

RSA
Socket[addr=wg31.washington.ibm.com/9.82.31.201, port=2445, localport=41164]
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Target host: wg31.washington.ibm.com
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
<b>Certificate information:</b>
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
<b>Subject DN:</b> CN=wg31.washington.ibm.com, OU=ATS, O=IBM
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
<b>Issuer DN:</b> CN=Db2 CA, OU=ATS, O=IBM
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Serial number: 1
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
<b>Certificate information:</b>
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
<b>Subject DN:</b> CN=Db2 CA, OU=ATS, O=IBM
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
<b>Issuer DN:</b> CN=Db2 CA, OU=ATS, O=IBM
[7/3/22 14:25:27:564 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Serial number: 0
[7/3/22 14:25:27:565 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Delegating to X509TrustManager implementation: com.ibm.jssse2.br
[7/3/22 14:25:27:566 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
<b>Server is trusted by all X509ExtendedTrustManager.</b>
[7/3/22 14:25:27:566 UTC] 00000808 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
checkServerTrusted Exit
[7/3/22 14:25:27:811 UTC] 00000808 id=ca4fabf7 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl <
sendRequest Exit
org.apache.cxf.jaxrs.impl.ResponseImpl@f64b8103

In the trace below, the `--cacert` attribute was omitted, and the handshake failed as expected.

```
c:\z\openapi3>curl -X POST -w " - HTTP CODE %{http_code}" --user user1:user1 --header "Content-Type: application/json" --data @selectEmployee.json https://wg31.washington.ibm.com:2445/services/zCEEService/selectEmployee -v
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 9.82.31.201:2445...
* Connected to wg31.washington.ibm.com (9.82.31.201) port 2445 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* CAfile: C:\z\software\nonIBM\cURL\curl-7.82.0-win64-mingw\bin\curl-ca-bundle.crt
* Capath: none
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS header, Unknown (21):
* TLSv1.2 (OUT), TLS alert, unknown CA (560):
* SSL certificate problem: self-signed certificate in certificate chain
* Closing connection 0
curl: (60) SSL certificate problem: self-signed certificate in certificate chain
More details here: https://curl.se/docs/sslcerts.html

curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.
- HTTP CODE 000
```

In the SYSLOG on the z/OS side these messages appear in the console log

```
EZD1287I TTLS Error RC: 435 Initial Handshake 514
LOCAL: 192.168.17.201..2445
REMOTE: 192.168.0.106..16795
JOBNAME: DSN2DIST RULE: Db2ServerRule 1
USERID: DB2USER GRPID: 00000004 ENVID: 00000017 CONNID: 000098F5
DSNL511I -DSN2 DSNIENO TCP/IP CONVERSATION FAILED 515
    TO LOCATION ::192.168.0.106
    IPADDR=::192.168.0.106 PORT=16795
    SOCKET=RECV RETURN CODE=1121 REASON CODE=77A9733D
```

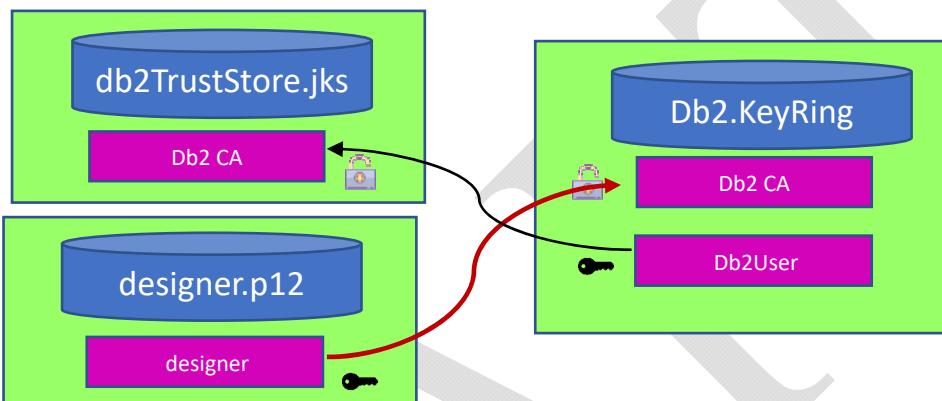
An explanation for a return of **435** (Certification authority is unknown) can be found at URL  
<https://www.ibm.com/docs/en/zos/2.3.0?topic=codes-ssl-function-return>

This completes the basic configuration of just a server handshake,

## Client certificate (mutual) authentication

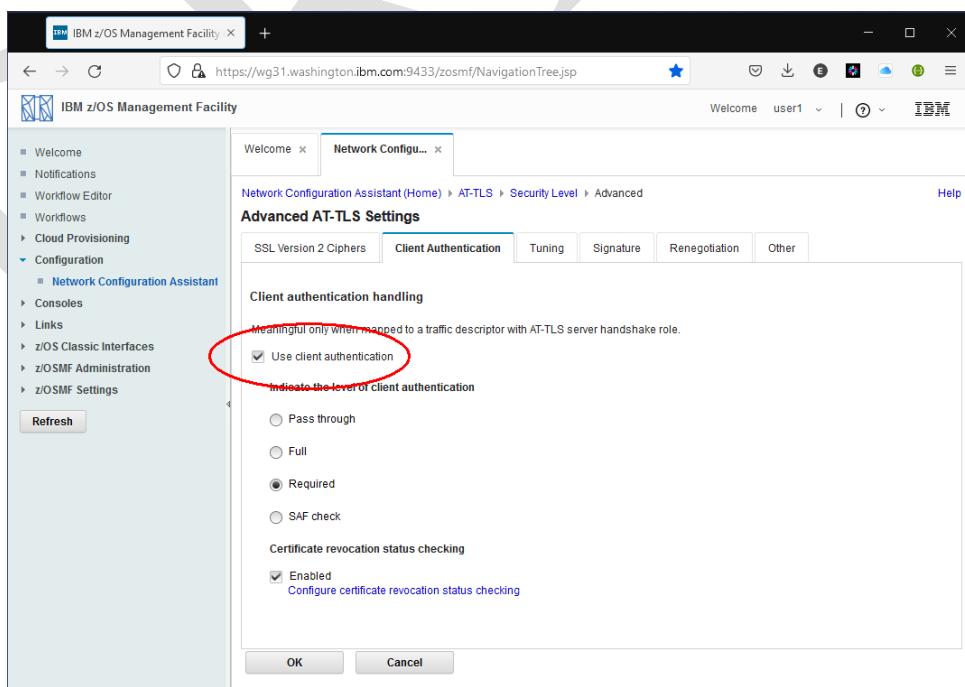
Now let's look at implementing mutual authentication. Mutual authentication occurs when the server endpoint requires the client to provide a proof of its identity by using a digital certificate. When mutual authentication is enabled, the client's personal certificate must have a private key.

Initially the client will send the public portion of its personal certificate (including a public key) to the server where the public contents of certificate will be validated with a local CERTAUTH certificate. Verification of the client is ensured because the client will then create a message digest (hash) of all the messages exchanged with the server. This digest is then encrypted with the client private key and sent to the server. The server uses the client public key to decrypt the digest. If the decrypted digest matches the server's digest derived from its record of the messages exchanged, then the client has proven its identity.



To configure mutual authentication in AT-TLS go to the *Security Level* tab and modify security level *zCEESecurity*. Select the *Advanced Settings* tab and then the **Advanced Settings** button to open the *Client Authentication* tab. Then check the box beside Use client authentication and the save and install the updated policy.

The updates can be activated by refreshing the policy agent task with MVS modify command **F PAGENT,REFRESH**



A *curl* test that directly invokes a Db2 REST service at this time would fail as shown here:

```
c:\z\openapi3>curl -X POST -w " - HTTP CODE %{http_code}" --user user1:user1 --header
"Content-Type: application/json" --data @selectEmployee.json --cacert DB2CA.pem
https://wg31.washington.ibm.com:2445/services/zCEEService/selectEmployee -v
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 9.82.31.201:2445...
* Connected to wg31.washington.ibm.com (9.82.31.201) port 2445 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* CAfile: DB2CA.pem
* Capath: none
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Request CERT (13):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS header, Unknown (21):
* OpenSSL SSL_connect: Connection was reset in connection to wg31.washington.ibm.com:2445
* Closing connection 0
curl: (35) OpenSSL SSL_connect: Connection was reset in connection to
wg31.washington.ibm.com:2445
- HTTP CODE 000
```

In the SYSLOG on the z/OS side these messages appear in the console log

```
EZD1287I TTLS Error RC: 403 Initial Handshake 447
LOCAL: 192.168.17.201..2445
REMOTE: 192.168.0.106..1127
JOBNAME: DSN2DIST RULE: Db2ServerRule 1
USERID: DB2USER GRPID: 00000003 ENVID: 000000011 CONNID: 000075B3
DSNL511I -DSN2 DSNLIE NO TCP/IP CONVERSATION FAILED 448
    TO LOCATION ::192.168.0.106
    IPADDR=::192.168.0.106 PORT=1127
    SOCKET=RECV RETURN CODE=1121 REASON CODE=77A9733D
```

An explanation for a return of **403** (No certificate received from partner) can be found at URL  
<https://www.ibm.com/docs/en/zos/2.3.0?topic=codes-ssl-function-return>

1. In an Ubuntu session, copy the user's private keycertificate file (*DB2CA.PEM*) from Windows into the Linux directory mapped to a container's directory (see the *docker-compose.yaml*) file.

```
cp /mnt/c/z/openApi3/certs/DESIGNER.P12 /home/workstation/docker/employees/certs
```

2. In an Ubuntu session, copy the Windows file *db2TlsMutual.xml* to the container configuration directory as file *db2.xml*.

```
cp /mnt/c/z/openapi3/tls/db2TlsMutual.xml
/home/workstation/docker/employees/project/src/main/liberty/config/db2.xml
```

Below are the contents of *db2TlsServer.xml*. The boldfaced lines are the additions made to the original contents of db2.xml for adding TLS support between the *Designer* and Db2.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="Db2 zosconnect_db2Connection ">
  <featureManager>
    <feature>zosconnect:db2-1.0</feature>
  </featureManager>

  <zosconnect_db2Connection id="db2Conn">
    host="${DB2_HOST}"
    port="${DB2SSL_PORT}"
    credentialRef="commonCredentials"
    sslRef="db2SSLSettings" />

    <zosconnect_credential id="commonCredentials">
      user="${DB2_USERNAME}"
      password="${DB2_PASSWORD}" />

    <ssl id="db2SSLSettings">
      keyStoreRef="db2KeyStore"
      trustStoreRef="db2TrustStore" />
    <keyStore id="db2TrustStore">
      location="/output/resources/security/db2TrustStore.jks"
      password="${DB2TRUSTSTORE_PASSWORD}" type="PKCS12" />
    <keyStore id="db2KeyStore">
      location="/output/resources/security/DESIGNER.P12"
      password="${DB2KEYSTORE_PASSWORD}" type="PKCS12" />
  </zosconnect_db2Connection>
</server>
```

The above configuration defines a TLS repertoire (*db2SSLSettings*) for use when connection to a Db2. Note the *sslRef* in the *zosconnect\_db2Connection* configuration elements references *db2SSLSettings*. The TLS repertoire has attributes *keyStoreRef* and *trustStoreRef* referencing different *keystore* elements. The *keystore* element, **db2TrustStore**, identifies the key store we used for server authentication in the previous section. This key store contains Db2 *CERTAUTH* certificate. The *keystore* element, **db2KeyStore**, identifies the key store that contains this client's personal certificate.

3. In an Ubuntu session, stop the container using this command:

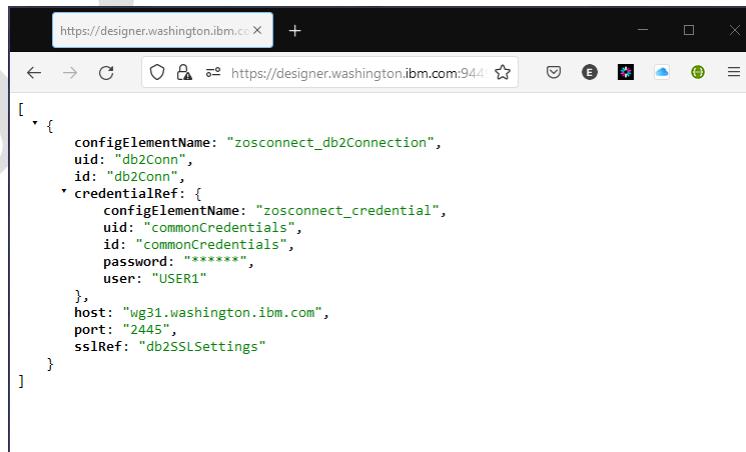
***docker stop employees\_zosConnect\_1***

4. In an Ubuntu session, restart the container using this command:

***docker start employees\_zosConnect\_1***

5. Enter the URL below in a browser to verify the updated *zosconnect\_db2Connection* configuraton

***[https://designer.washington.ibm.com:9449/ibm/api/config/zosconnect\\_db2Connection](https://designer.washington.ibm.com:9449/ibm/api/config/zosconnect_db2Connection)***



If a logon is required, use ***Fred*** as the *Username* and ***fredpwd*** as the *Password*

**Tech-Tip:** Installing the *restConnector-2.0* feature and providing an *administrator-role* configuration element, (see the *basicSecurity.xml* file), enabled the displaying of configuration elements in a web browser. This is good method for reviewing for syntax or other configuration errors in the installed server XML configuration element names particularly when other configuration elements are referenced.

6. Enter the URL below in a browser to verify the updated zosconnect\_db2Connection configuraton

**<https://designer.washington.ibm.com:9449/ibm/api/config/ssl?id=db2SSLSettings>**



```

https://designer.washington.ibm.com:9449/ibm/api/config/ssl?id=db2SSLSettings

[ {
  configElementName: "ssl",
  uid: "db2SSLSettings",
  id: "db2SSLSettings",
  clientAuthentication: false,
  clientAuthenticationSupported: false,
  enforceCipherOrder: false,
  * keyStoreRef: {
    configElementName: "keyStore",
    uid: "db2KeyStore",
    id: "db2KeyStore",
    fileBased: true,
    location: "/output/resources/security/DESIGNER.P12",
    password: "*****",
    pollingRate: 500,
    readOnly: false,
    type: "PKCS12",
    updateTrigger: "mbean"
  },
  securityLevel: "HIGH",
  trustDefaultCerts: false,
  * trustStoreRef: {
    configElementName: "keyStore",
    uid: "db2TrustStore",
    id: "db2TrustStore",
    fileBased: true,
    location: "/output/resources/security/db2TrustStore.jks",
    password: "*****",
    pollingRate: 500,
    readOnly: false,
    type: "PKCS12",
    updateTrigger: "mbean"
  },
  verifyHostname: false
}
]

```

### IBM z/OS Connect (OpenAPI 3.0)

To demonstrate the new TLS handshake flow, invoke a Db2 REST service by entering this *curl* command in a Windows DOS command session in directory c:\z\openapi3 with the **-v** trace will display the additions related to mutual authentication in the handshake flow in the **bold red** below:

```
curl -X POST -w "- HTTP CODE %{http_code}" --user user1:user1 --header "Content-Type: application/json" --data @selectEmployee.json --cacert certs\DB2CA.PEM --cert DESIGNER.P12:secret --cert-type P12 https://wg31.washington.ibm.com:2445/services/zCEEService/selectEmployee -v
```

```
c:\z\openapi3>curl -X POST -w "- HTTP CODE %{http_code}" --user user1:user1 --header "Content-Type: application/json" --data @selectEmployee.json --cacert certs\DB2CA.PEM https://wg31.washington.ibm.com:2445/services/zCEEService/selectEmployee -v --cert DESIGNER.P12:secret --cert-type P12
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 9.82.31.201:2445...
* Connected to wg31.washington.ibm.com (9.82.31.201) port 2445 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* CAfile: DB2CA.PEM
* CAPATH: none
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Request CERT (13):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Certificate (11):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, CERT verify (15):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / AES256-SHA
* ALPN, server did not agree to a protocol
* Server certificate:
* subject: O=IBM; OU=ATS; CN=WG31.WASHINGTON.IBM.COM
* start date: Jul 4 05:00:00 2022 GMT
* expire date: Jan 1 04:59:59 2024 GMT
* common name: WG31.WASHINGTON.IBM.COM (matched)
* issuer: O=IBM; OU=ATS; CN=DB2 CA
* SSL certificate verify ok.
* Server auth using Basic with user 'user1'
* TLSv1.2 (OUT), TLS header, Supplemental data (23):
> POST /services/zCEEService/selectEmployee HTTP/1.1
> Host: wg31.washington.ibm.com:2445
> Authorization: Basic dXNlcjB6dXNlcjE=
> User-Agent: curl/7.82.0
> Accept: */
> Content-Type: application/json
> Content-Length: 34
>
* TLSv1.2 (IN), TLS header, Supplemental data (23):
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Connection: keep-alive
< Content-Length: 227
< Content-Type: application/json; charset=UTF-8
< Date: Tue, 5 Jul 2022 20:00:23 GMT
< X-Powered-By: DB2 for z/OS
< Server: DB2 DDF Native REST, DSN2LOC, DSNLJEMG 10/02/19 UI65644
< Content-Language: en-US
< X-Correlation-ID: C0A8006A.H0F5.DBC249645DE9
<
{ "ResultSet"
Output": [{"employeeNumber": "000050", "firstName": "JOHN", "middleInitial": "B", "lastName": "GEYER", "department": "E01", "phoneNumber": "6789", "job": "MANAGER"}], "statusCode": 200, "StatusDescription": "Execution Successful"}* Connection #0 to host wg31.washington.ibm.com left intact
- HTTP/1.1 200 OK
```

Test the TLS connection between the *Designer* and Db2 using a curl command.

```
curl -X GET -w "- HTTP CODE %{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://designer.washington.ibm.com:9449/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w "- HTTP CODE %{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://designer.washington.ibm.com:9449/employees/details/000010
{"retrievedResults Output": [{"employeeID": "000010", "name": "CHRISTINE I HAAS", "departmentCode": "A00", "phoneNumber": "3978", "dateOfHire": "1965-01-01", "job": "PRES", "educationLevel": 18, "sex": "F", "dateOfBirth": "1933-08-14", "annualSalary": 52750.0, "lastBonus": 1000.0, "lastCommision": 4220.0}]} - HTTP CODE 200
```

A review of the AT-TLS trace shows a successful handshake.

```
Jul 1 19:57:17 wg31/TCPIP    TCPIP1    TTLS[345]: 14:57:17 TCPIP1    EZD1281I TTLS Map    CONNID: 00002F0F LOCAL: 192.168.17.201..2445 REMOTE: 192.168.0.106..1570 JOBNAME: DSN2DIST USERID: DB2USER TYPE: InBound STATUS: Enabled RULE: Db2ServerRule~1 ACTIONS: gAct1~Db2Server eAct1~Db2Server cAct1~Db2Server
Jul 1 19:57:17 wg31/TCPIP    TCPIP1    TTLS[345]: 14:57:17 TCPIP1    EZD1283I TTLS Event GRPID: 00000001 ENVID: 00000003 CONNID: 00002F0F RC: 0 Initial Handshake 0000005011527E10 0000005011522750 TLSV1.2 F0F0F3F5
```

## IBM z/OS Connect (OpenAPI 3.0)

And a review of the Liberty traces show the completion of the mutual authentication handshake flow.

```
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Target host: wg31.washington.ibm.com
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Certificate information:
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Subject DN: CN=WG31.WASHINGTON.IBM.COM, OU=ATS, O=IBM
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Issuer DN: CN=DB2 CA, OU=ATS, O=IBM
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Serial number: 1
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Certificate information:
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Subject DN: CN=DB2 CA, OU=ATS, O=IBM
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Issuer DN: CN=DB2 CA, OU=ATS, O=IBM
[7/6/22 13:35:51:321 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Serial number: 0
[7/6/22 13:35:51:322 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Delegating to X509TrustManager implementation: com.ibm.jsse2.br
[7/6/22 13:35:51:350 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
Server is trusted by all X509ExtendedTrustManager.
[7/6/22 13:35:51:350 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
checkServerTrusted Exit
[7/6/22 13:35:51:353 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
chooseClientAlias Entry
RSA

CN=DB2 CA, OU=ATS, O=IBM
CN=zCEE CA, OU=ATS, O=IBM
CN=WG31.WASHINGTON.IBM.COM, OU=ATS, O=IBM

Socket[addr=wg31.washington.ibm.com/9.82.31.201,port=2445,localport=42506]
[7/6/22 13:35:51:353 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
chooseClientAlias Exit
[7/6/22 13:35:51:353 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
chooseClientAlias Entry
RSA

CN=DB2 CA, OU=ATS, O=IBM
CN=zCEE CA, OU=ATS, O=IBM
CN=WG31.WASHINGTON.IBM.COM, OU=ATS, O=IBM

[7/6/22 13:35:51:353 UTC] 00000040 id=00000000 com.ibm.websphere.ssl.JSSEHelper
getOutboundConnectionInfo Entry
[7/6/22 13:35:51:353 UTC] 00000040 id=00000000 com.ibm.ws.ssl.config.ThreadContext
getOutboundConnectionInfo
[7/6/22 13:35:51:353 UTC] 00000040 id=00000000 com.ibm.websphere.ssl.JSSEHelper
getOutboundConnectionInfo Exit
null
[7/6/22 13:35:51:354 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
chooseClientAlias (from JSSE) Exit
<

designer
[7/6/22 13:35:51:354 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
getPrivateKey Entry
>

designer
[7/6/22 13:35:51:354 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
getX509KeyManager -> com.ibm.jsse2.bn
[7/6/22 13:35:51:354 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
getPrivateKey -> true Exit
[7/6/22 13:35:51:354 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
getCertificateChain: designer Entry
[7/6/22 13:35:51:354 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
getX509KeyManager -> com.ibm.jsse2.bn
[7/6/22 13:35:51:354 UTC] 00000040 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
getCertificateChain Exit
[

Version: V3
Subject: CN=Designer, OU=ATS, O=IBM
Signature Algorithm: SHA256withRSA, OID = 1.2.840.113549.1.1.11
Key: IBMJCE RSA Public Key:
```

This completes the basic configuraton of just a mutual authentication handshake.

## Using RACF key rings from the native z/OS server

To enable TLS connections between a z/OS Connect native server a RACF key ring will have to be created and the CERTAUTH certificate and a personal for the native server's RACF identity (if mutual authentication is required) connected to this key ring.

1. Invoke these are the commands to add the certificates provide by the certificate authority to a new key ring. These commands do not include the command to connect a personal certificate to the key ring.

```
racdcert id(atsserv) addring(Db2Client.KeyRing)
```

```
racdcert id(atsserv) connect(ring(Db2Client.KeyRing) label('DB2 CA') certauth usage(certauth))
```

```
setropts raclist(digtring,digtnmap) refresh
```

2. Update the current contents `/var/zcee/openapi3/includes/db2Api3.xml` with the contents in bold below:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="DB2 SSL">
  <featureManager>
    <feature>zosconnect:db2-1.0</feature>
  </featureManager>
  <zosconnect_credential id="commonCredentials" />
  user="${DB2_USERNAME}" password="${DB2_PASSWORD}" />
  <zosconnect_db2Connection id="db2Conn"
    host="${DB2_HOST}" port="${DB2_PORT}"
    credentialRef="commonCredentials"
    sslRef="db2SSLSettings" />
  <ssl id="db2SSLSettings"
    keyStoreRef="db2KeyRing"
    trustStoreRef="db2KeyRing" />
  <keyStore id="db2KeyRing"
    location="safkeyring:///Db2Client.KeyRing"
    password="password" type="JCERACEFKS"
    fileBased="false" readOnly="true" />
</server>
```

3. Change the value of the `DB2_PORT` variable `/var/zcee/openapi3/includes/openApi3.xml` from 2446 to **2445**.

```
<variable name="DB2_HOST" value="wg31.washington.ibm.com"/>
<variable name="DB2_PORT" value="2445"/>
<variable name="DB2_USERNAME" value="USER1"/>
<variable name="DB2_PASSWORD" value="USER1"/>
```

4. Update the server's configuration using a MVS modify command:

**F ZCEEAPI3,REFRESH,CONFIG**

5. Repeat some of the tests perform in section *Testing APIs deployed in a native z/OS server*, see page 90

**Congratulations, you have completed this exercise.**

## Additional information and samples

This section contains samples of using command to generate self-signed certificates and having them signed and then imported into a JSSE keystore. The details of the AT-TLS policy that was generated using zOSMF. And the contents of the original YAML file used to develop the Open API 3 API.

### ***JCL to define and load the Db2 table USER1.EMPLOYEE***

Below is the JCL used to load the Db2 table accessed by this API. This table was based on the standard Db2 sample employee with some constraints removed.

```
//EMPLOYEE EXEC PGM=IKJEFT01, DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
  DSN SYSTEM(DSN2)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA12) -
    LIB('DSN2.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN   DD *
  DROP TABLE USER1.EMPLOYEE;
  COMMIT;

CREATE TABLE USER1.EMPLOYEE
  (EMPNO      CHAR(6)          NOT NULL,
   FIRSTNAME  VARCHAR(12)       NOT NULL,
   MIDINIT    CHAR(1)          NOT NULL,
   LASTNAME   VARCHAR(15)       NOT NULL,
   WORKDEPT   CHAR(3)          ,
   PHONENO    CHAR(4)          ,
   HIREDATE   DATE             ,
   JOB        CHAR(8)          ,
   EDLEVEL    SMALLINT         ,
   SEX        CHAR(1)          ,
   BIRTHDATE  DATE             ,
   SALARY     DECIMAL(9, 2)     ,
   BONUS      DECIMAL(9, 2)     ,
   COMM       DECIMAL(9, 2)     ,
   PRIMARY KEY(EMPNO));

GRANT ALL PRIVILEGES ON TABLE USER1.EMPLOYEE TO USER1;

//LOAD    EXEC DSNUPROC,PARM='DSN2,DSNTEX',COND=(4,LT)
//SORTLIB  DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTOUT  DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SORTWK01 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SORTWK02 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SORTWK03 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//SORTWK04 DD UNIT=SYSDA,SPACE=(4000,(20,20),,ROUND)
//DSNTRACE DD SYSOUT=*
//SYSRECEM DD DSN=DSN1210.DB2.SDSNSAMP(DSN8LEM),
//           DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(4000,(50,50),,ROUND)
//SYSIN   DD *

LOAD DATA INDDN(SYSRECEM) CONTINUEIF(72:72)='X'
  INTO TABLE USER1.EMPLOYEE
  (EMPNO      POSITION( 1)  CHAR(6),
   FIRSTNAME  POSITION( 8)  VARCHAR,
   MIDINIT    POSITION(21)  CHAR(1),
   LASTNAME   POSITION(23)  VARCHAR,
   WORKDEPT   POSITION(36)  CHAR(3),
   PHONENO    POSITION(40)  CHAR(4),
   HIREDATE   POSITION(45)  DATE EXTERNAL,
   JOB        POSITION(56)  CHAR(8),
   EDLEVEL    POSITION(65)  INTEGER EXTERNAL(2),
   SEX        POSITION(68)  CHAR(1),
   BIRTHDATE  POSITION(80)  DATE EXTERNAL,
   SALARY     POSITION(91)  INTEGER EXTERNAL(5),
   BONUS      POSITION(97)  INTEGER EXTERNAL(5),
   COMM       POSITION(103) INTEGER EXTERNAL(5))
ENFORCE NO
```

## Client Certificate Requests

1. Create a self-signed certificate (and indirectly create a local keystore)

```
docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/zceeuusr.p12 -storetype PKCS12 -storepass changeit -genkey -keysize 2048 -alias zceeuusr -dname "CN=zceeuusr.washington.ibm.com, O=IBM, C=US" -keyalg RSA -validity 365
```

2. Now list the contents of the details of the *designer* certificate store using this command

```
docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/zceeuusr.p12 -storetype PKCS12 --storepass changeit -list -alias zceeuusr -v
```

```
root:/home/mitchj/docker/employees/certs:> docker exec -it employees_zosConnect_1 keytool -keystore /output/resources/security/zceeuusr.p12 -storetype PKCS12 --storepass changeit -list -alias zceeuusr -v
Alias name: zceeuusr
Creation date: Jul 8, 2022
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=zceeuusr, O=IBM, C=US
Issuer: CN=zceeuusr, O=IBM, C=US
Serial number: 2b86ea89
Valid from: 7/8/22 7:20 PM until: 7/8/23 7:20 PM
Certificate fingerprints:
    MD5: 69:44:17:E0:BE:B9:0A:AE:2F:25:6C:41:1E:79:B2:5D
    SHA1: AB:7B:77:C6:80:2F:B9:AA:AB:48:DD:20:30:6F:7D:8B:C9:D1:8F:85
    SHA256: EB:85:F7:F4:33:78:CF:3A:B6:1B:82:91:88:DD:0A:1E:3B:83:E9:6F:9F:33:A5:91:A2:DD:0B:B0:A1:AA:84:E3
    Signature algorithm name: SHA256withRSA
    Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 4b 34 2b f8 ed 42 27 63 22 4e 06 9c 31 d5 a7 7f K4...B.c.N..1...
0010: d7 81 5b 0e ....
]
]]
```

Note in the above output that the *Issuer* and *Owner* have the same distinguished name, the very definition of a self-signed certificate.

3. Create a certificate request from the self-signed certificate

```
docker exec -it employees_zosConnect_1 keytool -certreq -alias zceeuusr -keystore /output/resources/security/zceeuusr.p12 -storetype PKCS12 -storepass changeit -file /output/resources/security/zceeuusr.csr
```

Send the certificate request file *zceeuusr.csr* to the certificate authority for signing.

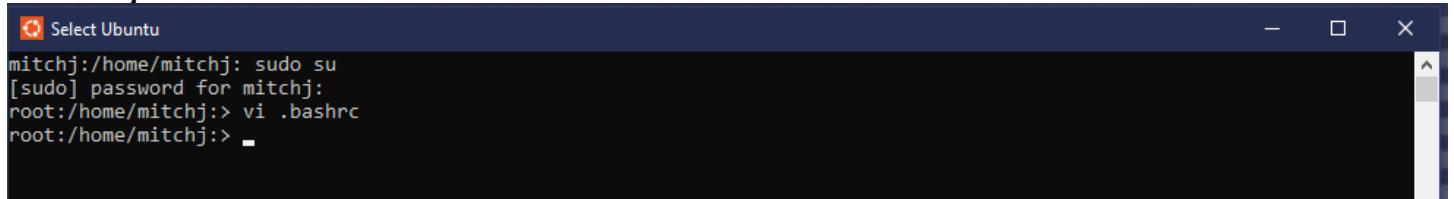
4. Import the signed personal certificate into the local keystore

```
docker exec -it employees_zosConnect_1 keytool -importcert -file /output/resources/security/ZCEEUSR.ARM -alias zceeuusr -storetype PKCS12 --noprompt -keystore /output/resources/security/db2TrustStore.jks --storepass changeit
```

## Suggestions for customizing the Linux shell environment

- Adding these lines to file `.bashrc` in your Linux home directory add path details to the Linux command prompt

```
PS1='\$LOGNAME':\$PWD:> '
export PATH=.:$PATH:/mnt/c/z/openApi3/bin
export containerHome=/home/workstation
```

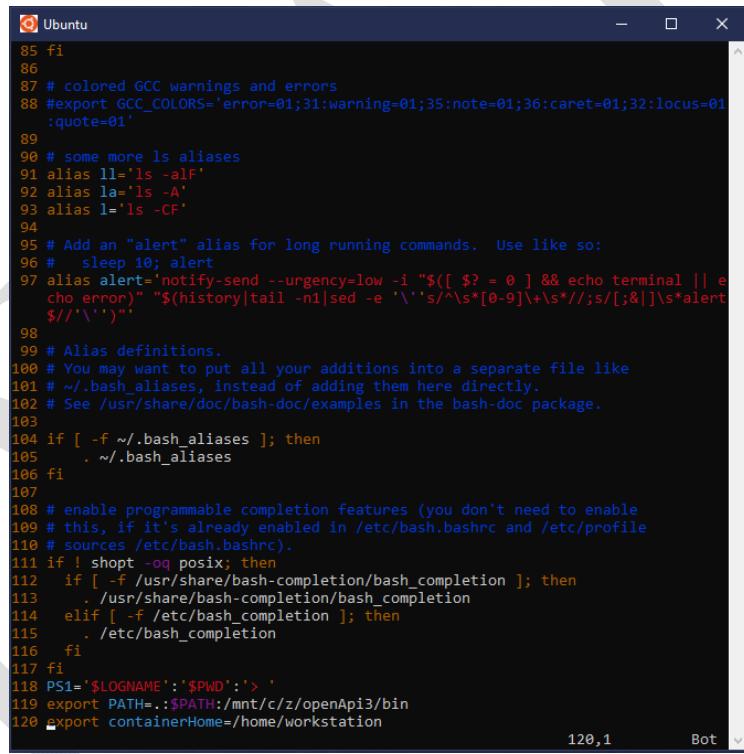


A screenshot of a terminal window titled "Select Ubuntu". The window shows a command-line session where a user has run "vi .bashrc" to edit the file. The terminal shows the command "root:/home/mitchj:>" followed by the text being added to the file:

```
mitchj:/home/mitchj: sudo su
[sudo] password for mitchj:
root:/home/mitchj:> vi .bashrc
root:/home/mitchj:>
```

- Creating a file named `.exrc` in your Linux home directory improves the `vi` editor experience

```
set showmode
set redraw
set wrapmargin=3
set nu
```



A screenshot of a terminal window titled "Ubuntu". The window shows the contents of the `.exrc` file. The file contains various `set` commands to improve the `vi` editor's behavior. The terminal shows the command "root:/home/mitchj:>" followed by the file content:

```
85 fi
86
87 # colored GCC warnings and errors
88 #export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01
89 :quote=01'
90 # some more ls aliases
91 alias ll='ls -alF'
92 alias la='ls -A'
93 alias l='ls -CF'
94
95 # Add an "alert" alias for long running commands. Use like so:
96 # sleep 10; alert
97 alias alert='notify-send --urgency=low -i "$( [ \$? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/\;[^;]*\;\'')"'"
98
99 # Alias definitions.
100 # You may want to put all your additions into a separate file like
101 # ~/.bash_aliases, instead of adding them here directly.
102 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
103
104 if [ -f ~/.bash_aliases ]; then
105   . ~/.bash_aliases
106 fi
107
108 # enable programmable completion features (you don't need to enable
109 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
110 # sources /etc/bash.bashrc).
111 if ! shopt -oq posix; then
112   if [ -f /usr/share/bash-completion/bash_completion ]; then
113     . /usr/share/bash-completion/bash_completion
114   elif [ -f /etc/bash_completion ]; then
115     . /etc/bash_completion
116   fi
117 fi
118 PS1='\$LOGNAME':\$PWD:> '
119 export PATH=.:$PATH:/mnt/c/z/openApi3/bin
120 export containerHome=/home/workstation
```

## Useful commands for managing containers

- Start a new container or update an existing container using a *docker-compose-yaml* file  
`docker-compose -f /home/mitchj/docker/sandbox/docker-compose.yaml up -d`
- Start a new container using *docker-compose-yaml* while in directory /home/mitchj/docker/sandbox  
`docker-compose up -d`
- Stop the container using docker-compose command while in directory /home/mitchj/docker/sandbox  
`docker-compose down`
- Start the sandbox container regardless of current directory  
`docker start sandbox-zosConnect-1`
- Stop the sandbox container regardless of current directory  
`docker stop sandbox-zosConnect-1`
- Copy server XML override files from Linux into a container directory\*  
`docker cp /home/mitchj/xml/. sandbox_zosConnect_1:/config/configDropins/overrides`
- List the active containers  
`docker ps`

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS			NAMES	
<code>97756ede6692</code>	<code>icr.io/zosconnect/ibm-zcon-designer:3.0.55</code>	<code>"/opt/ibm/helpers/r..."</code>	26 hours ago	Up 26 hours
<code>0.0.0.0:9088-&gt;9080/tcp, :::9088-&gt;9080/tcp, 0.0.0.0:9429-&gt;9443/tcp, :::9429-&gt;9443/tcp</code>				<code>employees_zosConnect_1</code>
<code>642f17a4063a</code>	<code>icr.io/zosconnect/ibm-zcon-designer:3.0.55</code>	<code>"/opt/ibm/helpers/r..."</code>	47 hours ago	Up 20 hours
<code>0.0.0.0:9082-&gt;9080/tcp, :::9082-&gt;9080/tcp, 0.0.0.0:9445-&gt;9443/tcp, :::9445-&gt;9443/tcp</code>				<code>sandbox_zosConnect_1</code>

- List all active and stopped containers  
`docker ps -a`
- Remove a container by name or container ID  
`docker rm sandbox_zosconnect_1`  
or  
`docker rm 642f17a4063a`
- Start a bash shell in the container  
`docker exec -it sandbox_zosConnect_1 bash`
- List the installed images  
`docker images`

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
<code>icr.io/zosconnect/ibm-zcon-designer</code>	<code>3.0.57</code>	<code>386f4ac8cbd0</code>	25 hours ago	<code>1.16GB</code>
<code>icr.io/zosconnect/ibm-zcon-designer</code>	<code>3.0.56</code>	<code>cf167f4230b5</code>	6 weeks ago	<code>1.57GB</code>
<code>icr.io/zosconnect/ibm-zcon-designer</code>	<code>3.0.55</code>	<code>be9c9101f533</code>	2 months ago	<code>1.52GB</code>
<code>hello-world</code>	<code>latest</code>	<code>feb5d9fea6a5</code>	8 months ago	<code>13.3kB</code>

- Remove an installed image
- Install the Podman *podman-compose* command
- Display the details of a container

***docker container inspect sandbox-zosConnect-1***

- Create a copy of a container
- Copy the configuration XML override file from Linux into the container
- Copy the war files and from the container
- Copy the configuration XML files from the container into Linux
- Pull in a new (download) a z/OS Connect Designer image
- Save the z/OS Connect Docker image to a file
- Copy the z/OS Connect Docker image file to a Windows directory location
- Use FTP to move the image file from the original image to the target Linux image
- Load the z/OS Connect Docker image on the Linux image

***docker load < 3.0.57.tar.gz***

## *AT-TLS policy file*

```
##  
## AT-TLS Policy Agent Configuration file for:  
##   Image: WG31  
##   Stack: TCPIP1  
##  
## Created by the IBM Configuration Assistant for z/OS Communications  
Server  
## Version 2 Release 3  
## Backing Store = USER1  
## Install History:  
## 2022-07-06 13:53:25 : Save To Disk  
##  
## End of Network Configuration Assistant information  
TTLSRule           Db2ServerRule~1  
{  
    LocalAddrSetRef      addr1  
    RemoteAddrSetRef     addr1  
    LocalPortRangeRef    portR1  
    Jobname              DSN2DIST  
    Direction            Inbound  
    Priority             255  
    TTLSGroupActionRef   gAct1~Db2Server  
    TTLSEnvironmentActionRef eAct1~Db2Server  
    TTLSConnectionActionRef cAct1~Db2Server  
}  
TTLSGroupAction      gAct1~Db2Server  
{  
    TTLSEnabled          On  
    Trace                7  
}  
TTLSEnvironmentAction eAct1~Db2Server  
{  
    HandshakeRole        Server  
    EnvironmentUserInstance 0  
    TTLSKeyringParmsRef  keyR1  
}  
TTLSConnectionAction  cAct1~Db2Server  
{  
    HandshakeRole        Server  
    TTLSConnectionAdvancedParmsRef cAdv1~Db2Server  
    CtraceClearText      Off  
    Trace                7  
}  
TTLSConnectionAdvancedParms  cAdv1~Db2Server  
{  
    SSLv3                Off  
    TLSv1                Off  
    TLSv1.1              Off  
    SecondaryMap         Off  
    TLSv1.2              On  
}  
TTLSKeyringParms      keyR1  
{  
    Keyring              Db2.KeyRing  
}  
IpAddrSet             addr1  
{  
    Prefix               0.0.0.0/0  
}  
PortRange             portR1  
{  
    Port                 2445  
}
```

## *The contents of the employees.yaml file*

```

openapi: 3.0.0
info:
  description: "Db2 Employees sample table"
  version: 1.0.0
  title: employees
servers:
- url: /
x-ibm-zcon-roles-allowed:
- Manager
security:
- BasicAuth: []
- BearerAuth: []
paths:
  /employees:
    post:
      tags:
        - db2employee
      operationId: postInsertEmployee
      parameters:
        - name: Authorization
          in: header
          required: false
          schema:
            type: string
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/postInsertEmployee_request"
        description: request body
        required: true
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postInsertEmployee_response_200"
        "400":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postInsertEmployee_response_400"
        "500":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postInsertEmployee_response_500"
  /employees/department:
    get:
      tags:
        - db2employee
      operationId: getSelectByDepartments
      parameters:
        - name: Authorization
          in: header
          required: false
          schema:
            type: string
        - name: dept1
          in: query
          required: false
          schema:
            type: string
            maxLength: 3
        - name: dept2
          in: query
          required: false
          schema:
            type: string
            maxLength: 3

```

```
responses:
  "200":
    description: OK
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/getSelectByDepartments_response_200"
  "404":
    description: OK
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/getSelectByDepartments_response_404"
  "500":
    description: OK
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/getSelectByDepartments_response_500"
"/employees/details/{employee}":
get:
  tags:
    - db2employee
  operationId: getDisplayEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  responses:
    "200":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/getDisplayEmployee_response_200"
    "404":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/getDisplayEmployee_response_404"
    "500":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/getDisplayEmployee_response_500"
"/employees/{employee}":
get:
  tags:
    - db2employee
  operationId: getSelectEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  responses:
    "200":
      description: OK
      content:
        application/json:
```

## IBM z/OS Connect (OpenAPI 3.0)

```
schema:
  $ref: "#/components/schemas/getSelectEmployee_response_200"
"404":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/getSelectEmployee_response_404"
"500":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/getSelectEmployee_response_500"
put:
  tags:
    - db2employee
  operationId: putInsertEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  requestBody:
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/putInsertEmployee_request"
    description: request body
    required: true
  responses:
    "200":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/putInsertEmployee_response_200"
    "404":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/putInsertEmployee_response_404"
    "500":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/putInsertEmployee_response_500"
delete:
  tags:
    - db2employee
  operationId: deleteDeleteEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  responses:
    "200":
      description: OK
      content:
        application/json:
```

## IBM z/OS Connect (OpenAPI 3.0)

```
schema:
  $ref: "#/components/schemas/deleteDeleteEmployee_response_200"
"404":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/deleteDeleteEmployee_response_404"
"500":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/deleteDeleteEmployee_response_500"
"/employees/roles/{job}":
  get:
    tags:
      - db2employee
    operationId: getSelectByRole
    parameters:
      - name: Authorization
        in: header
        required: false
        schema:
          type: string
      - name: job
        in: path
        required: true
        schema:
          type: string
          maxLength: 8
      - name: dept
        in: query
        required: true
        schema:
          type: string
          maxLength: 3
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/getSelectByRole_response_200"
      "404":
        description: OK
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/getSelectByRole_response_404"
      "500":
        description: OK
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/getSelectByRole_response_500"
components:
  schemas:
    getSelectEmployee_response_200:
      type: object
      required:
        - ResultSet Output
        - StatusCode
        - StatusDescription
      properties:
        ResultSet Output:
          type: array
          items:
            type: object
            description: ResultSet Row
            properties:
              employeeNumber:
                type: string
                description: CHAR(6)
                maxLength: 6
              firstName:
                type: string
```

Additional information and samples

© Copyright IBM Corporation 2022 All rights reserved  
Mitch Johnson (mitchj@us.ibm.com)

## IBM z/OS Connect (OpenAPI 3.0)

```
    description: VARCHAR(12)
    maxLength: 12
  middleInitial:
    type: string
    description: CHAR(1)
    maxLength: 1
  lastName:
    type: string
    description: VARCHAR(15)
    maxLength: 15
  department:
    type: string
    description: Nullable CHAR(3)
    maxLength: 3
  phoneNumber:
    type: string
    description: Nullable CHAR(4)
    maxLength: 4
  job:
    type: string
    description: Nullable CHAR(8)
    maxLength: 8
  required:
    - department
    - employeeNumber
    - firstName
    - job
    - lastName
    - middleInitial
    - phoneNumber
  StatusDescription:
    type: string
    description: Service invocation status description
  StatusCode:
    type: integer
    description: Service invocation HTTP status code
    minimum: 100
    maximum: 600
  description: Service selectEmployee invocation HTTP response body
getSelectEmployee_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be found
getSelectEmployee_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
deleteDeleteEmployee_response_200:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record deleted
putInsertEmployee_request:
  type: object
  required:
    - bonus
    - commission
    - salary
  properties:
    salary:
      type: number
      description: Nullable DECIMAL(9, 2)
      minimum: -99999999.99
      maximum: 99999999.99
    bonus:
      type: number
      description: Nullable DECIMAL(9, 2)
      minimum: -99999999.99
      maximum: 99999999.99
```

Additional information and samples

© Copyright IBM Corporation 2022 All rights reserved  
Mitch Johnson (mitchj@us.ibm.com)

```

commission:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
description: Service updateEmployee invocation HTTP request body
deleteDeleteEmployee_response_404:
  type: object
  properties:
    message:
      type: string
example:
  message: Record could not be found to be deleted
deleteDeleteEmployee_response_500:
  type: object
  properties:
    message:
      type: string
example:
  message: A severe error has occurred
putInsertEmployee_response_200:
  type: object
  required:
    - StatusCode
    - StatusDescription
    - Update Count
  properties:
    Update Count:
      type: integer
      description: Update Count
      minimum: 0
      maximum: 32767
    StatusDescription:
      type: string
      description: Service invocation status description
    StatusCode:
      type: integer
      description: Service invocation HTTP status code
      minimum: 100
      maximum: 600
description: Service updateEmployee invocation HTTP response body
putInsertEmployee_response_404:
  type: object
  properties:
    message:
      type: string
example:
  message: Record could not be inserted
putInsertEmployee_response_500:
  type: object
  properties:
    message:
      type: string
example:
  message: A severe error has occurred
getDisplayEmployee_response_200:
  type: object
  required:
    - ResultSet Output
    - StatusCode
    - StatusDescription
  properties:
    ResultSet Output:
      type: array
      items:
        type: object
        description: ResultSet Row
        properties:
          EMPNO:
            type: string
            description: CHAR(6)
            maxLength: 6
          FIRSTNAME:
            type: string
            description: VARCHAR(12)
            maxLength: 12
          MIDINIT:

```

## IBM z/OS Connect (OpenAPI 3.0)

```
type: string
description: CHAR(1)
maxLength: 1
LASTNAME:
  type: string
  description: VARCHAR(15)
  maxLength: 15
WORKDEPT:
  type: string
  description: Nullable CHAR(3)
  maxLength: 3
PHONENO:
  type: string
  description: Nullable CHAR(4)
  maxLength: 4
HIREDATE:
  type: string
  description: Nullable DATE yyyy-[m]m-[d]d
  minLength: 8
  maxLength: 10
  pattern: ^(?![0]{4})([0-9]{4})-(0?[1-9]|1[0-2])-(0?[1-9]|1[2][0-9]|3[0-1])$
```

```
JOB:
  type: string
  description: Nullable CHAR(8)
  maxLength: 8
EDLEVEL:
  type: integer
  description: Nullable SMALLINT
  minimum: -32768
  maximum: 32767
SEX:
  type: string
  description: Nullable CHAR(1)
  maxLength: 1
BIRTHDATE:
  type: string
  description: Nullable DATE yyyy-[m]m-[d]d
  minLength: 8
  maxLength: 10
  pattern: ^(?![0]{4})([0-9]{4})-(0?[1-9]|1[0-2])-(0?[1-9]|1[2][0-9]|3[0-1])$
```

```
SALARY:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
BONUS:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
COMM:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
required:
- BIRTHDATE
- BONUS
- COMM
- EDLEVEL
- EMPNO
- FIRSTNME
- HIREDATE
- JOB
- LASTNAME
- MIDINIT
- PHONENO
- SALARY
- SEX
- WORKDEPT
StatusDescription:
  type: string
  description: Service invocation status description
StatusCode:
  type: integer
  description: Service invocation HTTP status code
  minimum: 100
```

Additional information and samples

© Copyright IBM Corporation 2022 All rights reserved  
Mitch Johnson (mitchj@us.ibm.com)

157 of 161

## IBM z/OS Connect (OpenAPI 3.0)

```
maximum: 600
description: Service displayEmployee invocation HTTP response body
getDisplayEmployee_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
getDisplayEmployee_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
postInsertEmployee_request:
  type: object
  required:
    - birthDate
    - bonus
    - commission
    - department
    - educationLevel
    - employeeNumber
    - firstName
    - hireDate
    - job
    - lastName
    - middleInitial
    - phoneNumber
    - salary
    - sex
  properties:
    employeeNumber:
      type: string
      description: Nullable CHAR(6)
      maxLength: 6
    firstName:
      type: string
      description: Nullable VARCHAR(12)
      maxLength: 12
    middleInitial:
      type: string
      description: Nullable CHAR(1)
      maxLength: 1
    lastName:
      type: string
      description: Nullable VARCHAR(15)
      maxLength: 15
    department:
      type: string
      description: Nullable CHAR(3)
      maxLength: 3
    phoneNumber:
      type: string
      description: Nullable CHAR(4)
      maxLength: 4
    hireDate:
      type: string
      description: Nullable DATE yyyy-[m]m-[d]d
      minLength: 8
      maxLength: 10
      pattern: ^(?![0]{4})(([0-9]{4})-(0?[1-9]|1[0-2])-(0?[1-9]|1[0-2][0-9]|3[0-1]))$ 
    job:
      type: string
      description: Nullable CHAR(8)
      maxLength: 8
    educationLevel:
      type: integer
      description: Nullable SMALLINT
      minimum: -32768
      maximum: 32767
    sex:
      type: string
      description: Nullable CHAR(1)
      maxLength: 1
```

Additional information and samples

© Copyright IBM Corporation 2022 All rights reserved  
Mitch Johnson (mitchj@us.ibm.com)

## IBM z/OS Connect (OpenAPI 3.0)

```
birthDate:  
  type: string  
  description: Nullable DATE yyyy-[m]m-[d]d  
  minLength: 8  
  maxLength: 10  
  pattern: ^(?:[0]{4}) ([0-9]{4})-(0?[1-9]|1[0-2])-(0?[1-9]|1[1-2][0-9]|3[0-1])$  
salary:  
  type: number  
  description: Nullable DECIMAL(9,2)  
  minimum: -9999999.99  
  maximum: 9999999.99  
bonus:  
  type: number  
  description: Nullable DECIMAL(9,2)  
  minimum: -9999999.99  
  maximum: 9999999.99  
commission:  
  type: number  
  description: Nullable DECIMAL(9,2)  
  minimum: -9999999.99  
  maximum: 9999999.99  
description: Service insertEmployee invocation HTTP request body  
postInsertEmployee_response_200:  
  type: object  
  required:  
    - StatusCode  
    - StatusDescription  
    - Update Count  
  properties:  
    Update Count:  
      type: integer  
      description: Update Count  
      minimum: 0  
      maximum: 32767  
    StatusDescription:  
      type: string  
      description: Service invocation status description  
    StatusCode:  
      type: integer  
      description: Service invocation HTTP status code  
      minimum: 100  
      maximum: 600  
description: Service insertEmployee invocation HTTP response body  
postInsertEmployee_response_400:  
  type: object  
  properties:  
    message:  
      type: string  
  example:  
    message: Record could not be inserted  
postInsertEmployee_response_500:  
  type: object  
  properties:  
    message:  
      type: string  
  example:  
    message: A severe error has occurred  
getSelectByRole_response_200:  
  type: object  
  required:  
    - ResultSet Output  
    - StatusCode  
    - StatusDescription  
  properties:  
    ResultSet Output:  
      type: array  
      items:  
        type: object  
        description: ResultSet Row  
        properties:  
          employeeNumber:  
            type: string  
            description: CHAR(6)  
            maxLength: 6  
          firstName:  
            type: string  
            description: VARCHAR(12)
```

Additional information and samples

© Copyright IBM Corporation 2022 All rights reserved  
Mitch Johnson (mitchj@us.ibm.com)

```
maxLength: 12
middleInitial:
  type: string
  description: CHAR(1)
  maxLength: 1
lastName:
  type: string
  description: VARCHAR(15)
  maxLength: 15
department:
  type: string
  description: Nullable CHAR(3)
  maxLength: 3
phoneNumber:
  type: string
  description: Nullable CHAR(4)
  maxLength: 4
job:
  type: string
  description: Nullable CHAR(8)
  maxLength: 8
required:
- department
- employeeNumber
- firstName
- job
- lastName
- middleInitial
- phoneNumber
StatusDescription:
  type: string
  description: Service invocation status description
StatusCode:
  type: integer
  description: Service invocation HTTP status code
  minimum: 100
  maximum: 600
  description: Service selectByRole invocation HTTP response body
getSelectByRole_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
getSelectByRole_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
getSelectByDepartments_response_200:
  type: object
  required:
- StatusCode
- StatusDescription
  properties:
    ResultSet 1 Output:
      type: array
      description: Stored Procedure ResultSet 1 Data
      items:
        type: object
        description: ResultSet Row
        properties: {}
  Anonymous ResultSets:
    type: integer
    description: Number of Anonymous ResultSets
    minimum: 0
    maximum: 1
  StatusDescription:
    type: string
    description: Service invocation status description
  StatusCode:
    type: integer
    description: Service invocation HTTP status code
    minimum: 100
```

IBM z/OS Connect (OpenAPI 3.0)

```
maximum: 600
description: Service selectByDepartments invocation HTTP response body
getSelectByDepartments_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
getSelectByDepartments_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
```

DRAFT