

IBM z/OS Connect (OpenAPI 3.0)

Developing and Administering RESTful APIs for CICS REST Services



IBM
IBM Z
Wildfire Team –
Washington System Center

Table of Contents

Overview	3
The CSCVINC CICS application	4
The z/OS Connect Designer Container environment	4
Setup required Windows Subsystems for Linux support	6
Docker Engine.....	7
Podman.....	7
The container configuration file.....	8
Connecting to a CICS region and the required server XML configuration.....	9
Basic security and the required server XML configuration	9
Accessing the z/OS Connect Designer log and trace files	11
Managing the z/OS Connect Designer container	13
Using Podman commands.....	13
Creating a new container.....	13
Refresh an existing container	14
Using Docker commands.....	15
Designer problem determination.....	16
Developing a z/OS Connect APIs that accesses a CICS program.....	18
Configure the POST method for URI path /employee	20
Configure the GET method for URI path /employee/{employee}.....	38
Testing the API's POST and GET methods.....	44
Complete the configuration of the API (Optional).....	49
Configure the PUT method for URI path /employee/{employee}.....	49
Configure the DELETE method for URI path /employee/{employee}	54
Testing APIs deployed in a z/OS Connect Designer container	59
Using Postman.....	60
Using cURL	67
Using the API Explorer.....	70
Deploying and installing APIs in a z/OS Connect Native Server	75
Moving the API Web Archive file from the container to a z/OS OMVS directory	75
Updating the server xml	77
Defining the required RACF EJBRole resources	78
Testing APIs deployed in a native z/OS server	79
Using Postman.....	79
Using cURL	84
Using the API Explorer.....	87
Using TLS to access CICS.....	92
Creating the RACF resources	92
Enabling CICS TLS support	94
Test the TLS connection from the z/OS Connect Designer to a CICS subsystem	96
Using a JSSE key store from the Designer	97
Server certificate authentication.....	100
Client certificate (mutual) authentication	104
Using RACF key rings from the native z/OS server.....	110
Additional information and samples.....	111
Client Certificate Requests	111
Suggestions for customizing the Linux shell environment.....	112
Useful commands for managing containers	113
The contents of the cscvinc.yaml file	115

Important: On the desktop there is a file named *OpenAPI 3 Development for CICS 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 CICS to perform this exercise. Even if CICS 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.

The CSCVINC CICS application

In this exercise we are developing an API to access a CICS application program named CSCVINC. This application is a COBOL application that uses a channel/container interface. This application, when invoked, accesses the channel provided by CICS and looks for the container in that channel. The name of the container is not hard coded in the application, so the application will accept a container with any name.

The container has a field named ACTION. The contents of this field determines what function the application will perform. Based on the value of ACTION, the application will access the CICS sample FILEA VSAM data set and either insert a new record (I), retrieve an existing record (G), update an existing record (U) or delete an existing record (D).

The application returns the results of the request including the values of the CICS EIBRESP and EIBRESP3 response code from the EXEC CICS WRITE, EXEC CICS READ, EXE CICS REWRITE or EXEC CICS DELETE API performed by the program. Also returned is the identity under which the CICS transaction executed. The response container name will have the same name as the request container.

The COBOL copy book for the request container is shown here.

```
01 Request-Container.
  03 ACTION          PIC X(1).
  03 USERID          PIC X(8).
  03 FILEA-AREA.
    05 STAT            PIC X.
    05 NUMB            PIC X(6).
    05 NAME            PIC X(20).
    05 ADDRX           PIC X(20).
    05 PHONE           PIC X(8).
    05 DATEX           PIC X(8).
    05 AMOUNT          PIC X(8).
    05 COMMENT         PIC X(9).
```

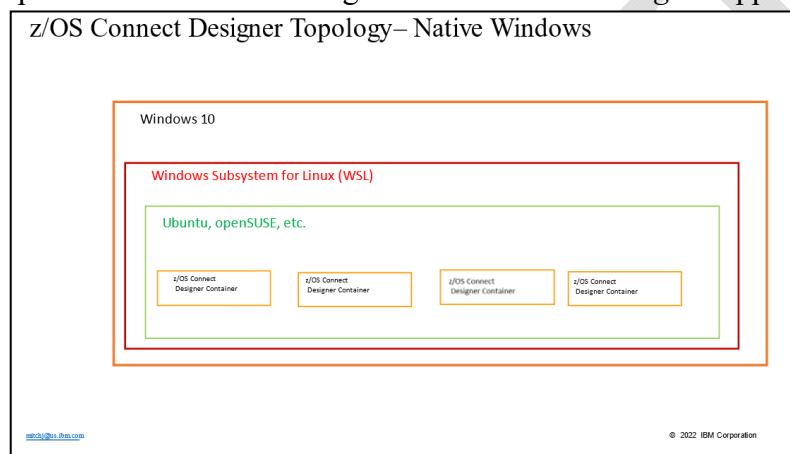
The COBOL copy book for the response container is shown here.

```
01 Response-Container.
  03 ACTION          PIC X(1).
  03 CEIBRESP        PIC S9(8) COMP.
  03 CEIBRESP2       PIC S9(8) COMP.
  03 USERID          PIC X(8).
  03 FILEA-AREA.
    05 STAT            PIC X.
    05 NUMB            PIC X(6).
    05 NAME            PIC X(20).
    05 ADDRX           PIC X(20).
    05 PHONE           PIC X(8).
    05 DATEX           PIC X(8).
    05 AMOUNT          PIC X(8).
    05 COMMENT         PIC X(9).
```

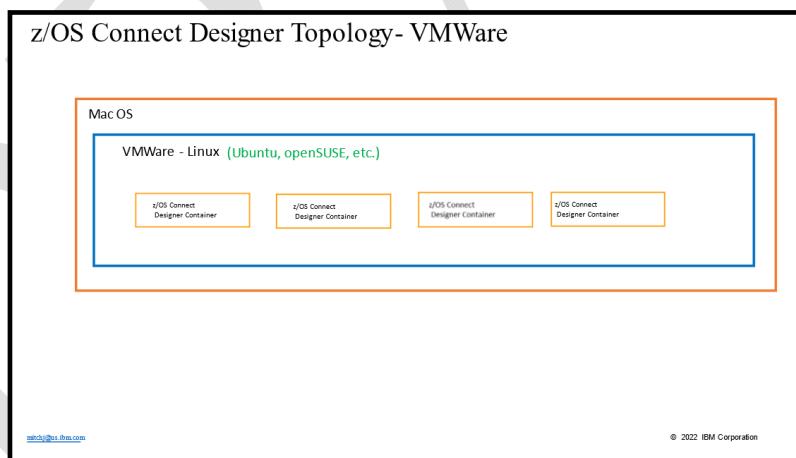
The z/OS Connect Designer Container environment

APIs for accessing CICS 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.

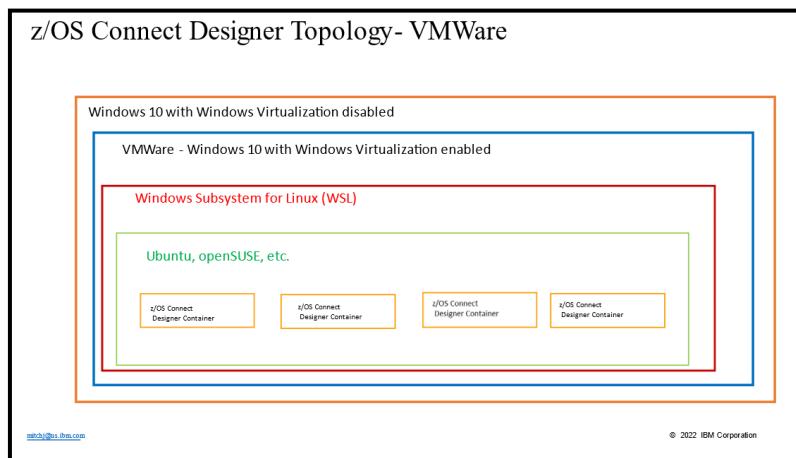


This diagram shows another scenario for a Mac OS image where VMWare Fusion[©] or Parallels[©] can be used to run a virtual Linux image in which where the Liberty *Designer* containers can run.



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> were followed.

A screenshot of a Microsoft Edge browser window showing the "Set up a WSL development environment" article from Microsoft Docs. The page includes a sidebar with navigation links like "WSL Documentation", "Install", and "Tutorials". The main content area has a heading "Get started" and text explaining the setup process for WSL. A "Show more" button is visible on the right side of the page.

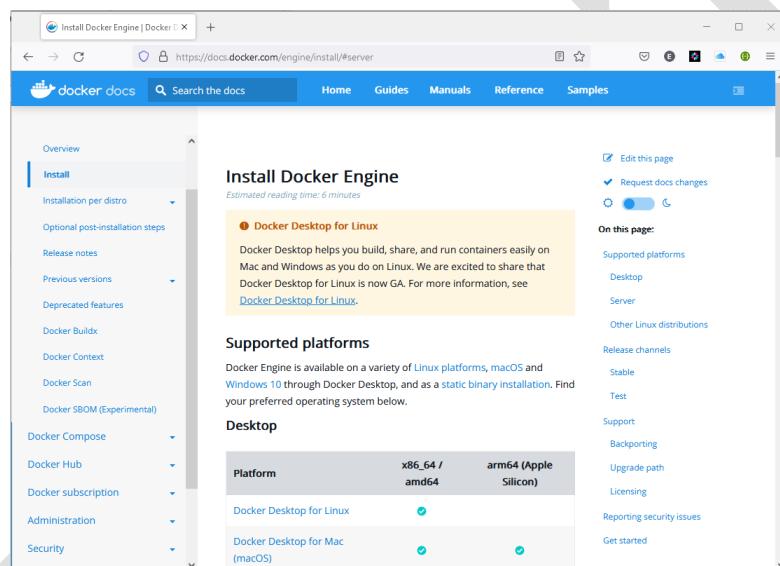
A Linux distribution can be installed following the instructions at URL <https://docs.microsoft.com/en-us/windows/wsl/install>

A screenshot of a Microsoft Edge browser window showing the "Install Linux on Windows with WSL" article from Microsoft Docs. The page includes a sidebar with navigation links like "WSL Documentation", "Install", and "Tutorials". The main content area has a heading "Prerequisites" and text explaining the requirements for installing WSL. A "Show more" button is visible on the right side of the page.

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*.

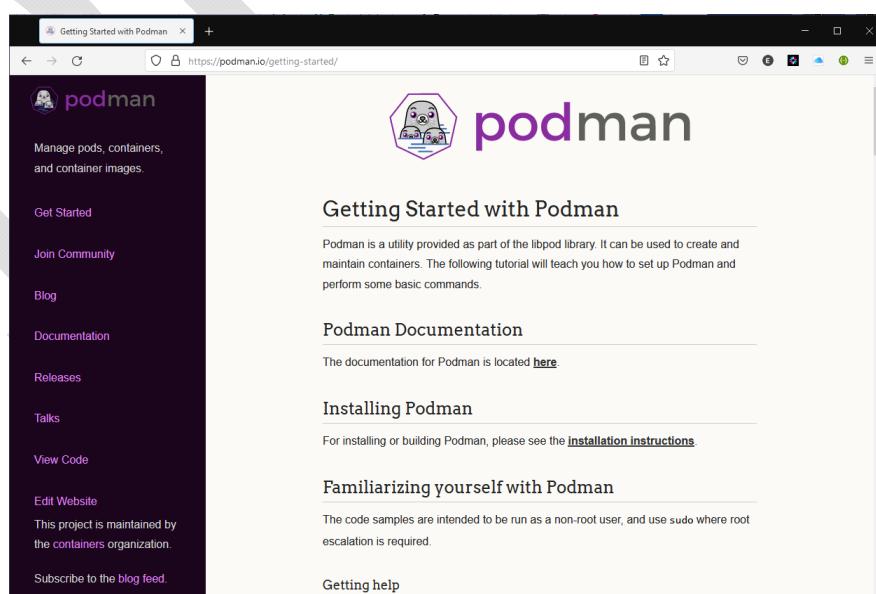
Docker Engine

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 CICS related environment variable be provided. These variables are used to customize the CICS 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
      - CICS_PORT=2446
      - HTTP_PORT=9080
    ports:
      - "9447:9443"
      - "9084:9080"
    volumes:
      - ./project:/workspace/project
      - ./logs/:/logs/
      - ./certs:/output/resources/security/
```

Connecting to a CICS region requires the addition of a `zosconnect_cicsIpicConnection` 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 a CICS region and the required server XML configuration

The `zosconnect_cicsIpicConnection` element used to connect to a CICS region in this exercise looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="IPIC connection to CICS">
  <featureManager>
    <feature>zosconnect:cics-1.0</feature>
  </featureManager>

  <zosconnect_cicsIpicConnection id="cicsConn"
    host="${CICS_HOST}"
    port="${CICS_PORT}"
    authDataRef="cicsCredentials" />

  <zosconnect_authData id="cicsCredentials"
    user="${CICS_USER}"
    password="${CICS_PASSWORD}" />

</server>
```

Notice the environment variables `${CICS_HOST}`, `${CICS_PORT}`, `${CICS_USER}` and `${CICS_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="zCEERoles">
    <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.

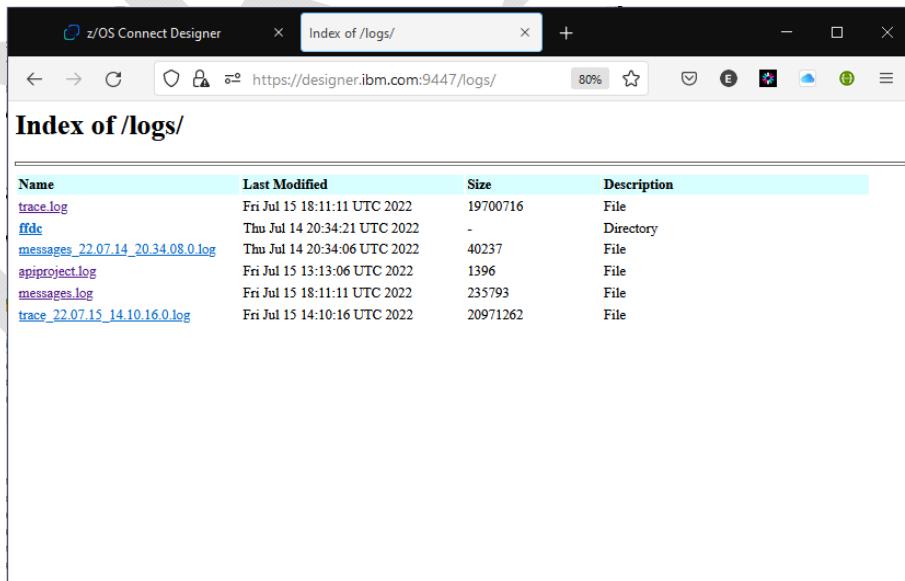
```
openapi: 3.0.0
info:
  description: "CICS Employee Sample VSAM Application"
  version: 1.0.0
  title: Employee
x-ibm-zcon-roles-allowed:
- Manager
paths:
  /employee:
    post:
      -----
      "/employee/{employee}":
        get:
          tags:
            - Employee
          operationId: getEmployeeSelectService
          x-ibm-zcon-roles-allowed:
            - Staff
      put:
        tags:
          - Employee
        operationId: putEmployeeUpdateService
        x-ibm-zcon-roles-allowed:
          - Staff
      delete:
        tags:
          - Employee
        operationId: putEmployeeUpdateService
        x-ibm-zcon-roles-allowed:
          - Staff
      -----
```

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 IBM z/OS Connect Designer UI. The URL in the address bar is `designer.ibm.com:9447/logs/trace.log`. The page displays a log of system events and SQL requests. A red box highlights a section of the log where a SQL query is shown being processed. The log entries include timestamps, thread IDs, and detailed information about session states and database operations. At the bottom of the log, there is a JSON object representing a response container. The search bar at the bottom left contains the text "cicsconn".

```

[7/15/22 18:22:43:224 UTC] 00000044 id=ad10b3fc com.ibm.zosconnect.cics.internal.conn.isc.Connection < sessionPoolIdle Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=a5b3000b com.ibm.zosconnect.cics.internal.conn.isc.SessionPool < deallocateSession Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.Session 3 Session state updated from FREEING to IDLE
[7/15/22 18:22:43:224 UTC] 00000044 id=2047cc94 com.ibm.zosconnect.cics.internal.conn.isc.Session < deallocateSession Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.Conversation 3 updateConStatus, Old state:CICS_COMPLETE, New state:DEALLOCATED
[7/15/22 18:22:43:224 UTC] 00000044 id=15190930 com.ibm.zosconnect.cics.internal.conn.isc.Conversation < updateConState Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=e8067058 com.ibm.zosconnect.cics.internal.conn.isc.SessionManager < endConversation Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=95c3f4d8 com.ibm.zosconnect.cics.ServerEISCIRequest < executeEISCI Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=95c3f4d8 com.ibm.zosconnect.cics.ServerEISCIRequest false
[7/15/22 18:22:43:224 UTC] 00000044 id=95c3f4d8 com.ibm.zosconnect.cics.ServerEISCIRequest > RequestInThread com.ibm.zosconnect.cics.ServerEISCIRequest@95c3f4d8
[7/15/22 18:22:43:224 UTC] 00000044 id=95c3f4d8 com.ibm.zosconnect.cics.CiscspicConnection > workEnded Entry
[7/15/22 18:22:43:224 UTC] 00000044 id=00000000 com.ibm.zosconnect.cics.CiscspicConnection 3 Work ended, remaining work in progress is 0
[7/15/22 18:22:43:225 UTC] 00000044 id=95c3f4d8 com.ibm.zosconnect.cics.CiscspicConnection < workEnded Exit
[7/15/22 18:22:43:225 UTC] 00000044 id=95c3f4d8 com.ibm.zosconnect.cics.CiscspicConnection < flow Exit
[7/15/22 18:22:43:225 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset < flowRequest Exit
[7/15/22 18:22:43:225 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset > getContainer Entry
[7/15/22 18:22:43:225 UTC] 00000044 id=2148441a com.ibm.zosconnect.cics.Channel < getContainer Exit
[7/15/22 18:22:43:225 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset < com.ibm.zosconnect.cw.metadata.transaction.Message@968c70cb
[7/15/22 18:22:43:225 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset > [0@4a269000],len=97
[0000] E2000000 00000000 50C3CC3 E2E4E2C5 D940F1F2 F1F2F1F2 40404040 40404040
[0000] 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
[0000] 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
[0000] 40
[7/15/22 18:22:43:242 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset < transformBytestoJson Exit
[7/15/22 18:22:43:242 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset {"Response-Container":{"ACTION":"S","CEIBRESP":13,"CEIBRESP2":88,"USERID":"CICSUSER","FILEA-AREA":"STAT","","NUMB":"121212","NAME":"","ADDRX":"","PHONE":"","DATEX":"","AMOUNT":"","COMMENT":""}}
[7/15/22 18:22:43:243 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsChannelZosAsset < invoke Exit

```

cicsconn

Highlight All Match Case Match Diacritics Whole Words 2 of 90 matches

Managing the z/OS Connect Designer container

Using Podman commands

The section explores the use of Podman 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 *refreshPodmanContainer*. If anytime you wish to restart, open an Ubuntu terminal shell, and enter command ***refreshPodmanContainer employee***. 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/podman/sandbox
- ___ 3. Change location to the new directory
cd /home/workstation/podman/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/9080:9080/9084:9080/" docker-compose.yaml
sed -i "s/9443:9443/9447:9443/" docker-compose.yaml
- ___ 9. Start the container
podman-compose up -d
- ___ 10. Copy server XML override files from Linux into a container’s configuration directory
podman 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., *createPodmanContainer*, 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 ***createPodmanContainer 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/podman/"$1"
cd $containerHome/podman/"$1"
mkdir certs
mkdir logs
mkdir -p project/src/main/liberty/config
cp /mnt/c/z/openApi3/xml/* project/src/main/liberty/config
cp /mnt/c/z/openApi3/yaml/docker-compose.yaml .
sed -i "s/9080:9080/$HTTP_port:9080/" docker-compose.yaml
sed -i "s/9443:9443/$HTTPS_port:9443/" docker-compose.yaml
podman-compose up -d
podman cp /mnt/c/z/openApi3/xml/. "$1"_zosConnect_1:/config/configDropins/overrides
```

11. Display the status of the active containers

podman ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
0a53e5d3a7af	icr.io/zosconnect/ibm-zcon-designer:3.0.57	"/opt/ibm/helpers/run..."	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 cscvinc_zosConnect_1

Refresh an existing container

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

1. Stop the container

podman stop sandbox_zosConnect_1

2. Remove the container

podman container rm sandbox_zosConnect_1

3. Change to the host Linux directory of the container

cd /home/workstation/podman/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. Remove the container

```
podman rm sandbox_zosConnect_1
```

8. Start the container

```
podman-compose up -d
```

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

```
podman 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., *refreshPodmanContainer*, 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 *refreshPodmanContainer containerName*

(Where *containerName* is the name of the container without the *_zosConnect_1 suffix*).

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

Using Docker commands

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

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 500 with no EIB response code or identity. A review of the trace shows that the CICS transaction abended with an abend code of MIJO.

2. Invoking a method returned an HTTP 500 with no other information, for example using the curl as shown below.

```
c:\z\openApi3>curl -X DELETE -w "- HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9447/employee/948478  
- HTTP CODE 500
```

d

The trace showed a response came back from CICS with non-zero EIB response codes. But the mapping for this situation was not present, i.e., the *response 500.yaml* did not exist.

z/OS Connect Designer designer.ibm.com:9447/logs/trace.log 80%

https://designer.ibm.com:9447/logs/trace.log

```
[7/16/22 17:47:40] [401:858 UTC] 00000525 com.ibm.zosconnect.cics.internal.com.lsc.Conversation < updateConversation Exit  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=e6867958 com.ibm.zosconnect.cics.internal.com.lsc.SessionManager < endConversation Exit  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=223dieb9 com.ibm.zosconnect.cics.ServerECIRequest < executeISIP Exit  
    false  
3 requestFinished(com.ibm.zosconnect.cics.ServerECIRequest@223dieb9)  
[7/16/22 17:47:40] [401:858 UTC] 00000525 com.ibm.zosconnect.cics.CiscIpicConnection > workended Entry  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=eaeb3c3d com.ibm.zosconnect.cics.CiscIpicConnection 3 work ended, remaining work in progress is 0  
[7/16/22 17:47:40] [401:858 UTC] 00000525 com.ibm.zosconnect.cics.CiscIpicConnection < workended Exit  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=eaeb3c3d com.ibm.zosconnect.cics.CiscIpicConnection < flow Exit  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=eaeb3c3d com.ibm.zosconnect.cics.CiscIpicConnection < getContainer Exit  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=eaeb3c3d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset > getContainer Entry  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=6db2a3f com.ibm.zosconnect.cics.Channel < getContainer Exit  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=6db2a3f com.ibm.zosconnect.cics.Channel > transformByBytesToJson Entry  
[7/16/22 17:47:40] [401:858 UTC] 00000525 id=6db2a3f com.ibm.zosconnect.cics.ChannelContainer@6dbe1c83 com.onw.zosconnect.wv.metadata.transaction.Message@2ed7d4dc  
[6dbe1c83] len=97  
[0020] C4000000 00000000 50C3C93 E2E4E2C5 00000F94 F8F4F7F8 00000000 00000000  
[0020] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
[0040] 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
[0060] 00  
< transformByBytesToJson Exit  
[7/16/22 17:47:40] [401:859 UTC] 00000525 id=6db2a3f com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset < Response-Container> { "ACTION": "D", "CEIBRESP1": "13", "CEIBRESP2": "00", "USERID": "CICSUSER", "FILEA-AREA": "  
{"STATE": "", "NAME": "6db2a3f", "NAME2": "", "ADDRESS": "", "PHONE": "", "DATE": "", "MOUNT": "", "COMMENT": ""}}  
[7/16/22 17:47:40] [401:859 UTC] 00000525 id=6db2a3f com.ibm.zosconnect.zosasset.cics.internal.CicsChannelZosAsset < invoke Exit  
[7/16/22 17:47:40] [401:860 UTC] 00000525 id=6db2a3f com.ibm.zosconnect.engine.impl.ProviderInvokeOperationImpl W MQ2Q1677W: Response mapping file %2fEmployee%2f%2fEmployee%7D/delete/response_500.yaml does not exist, so no response body was returned.  
[7/16/22 17:47:40] [401:866 UTC] 00000525 id=df574b com.ibm.ws.security.collector.CollectorUtils < getCallerPrincipal Entry  
    null  
    true  
    false  
    > getCallerSubject Entry  
[7/16/22 17:47:40] [401:866 UTC] 00000525 id=bbb4552f com.ibm.ws.security.context.SubjectManager < getCallerSubject Entry  
[7/16/22 17:47:40] [401:866 UTC] 00000525 id=59a3a19b com.ibm.ws.security.context.internal.SubjectThreadContext < getCallerSubject Entry  
[7/16/22 17:47:40] [401:866 UTC] 00000525 id=59a3a19b com.ibm.ws.security.context.internal.SubjectThreadContext < getCallerSubject Exit  
    Subject:  
  
    Principal: W$principal$Fred  
    Public Credential: com.ibm.ws.security.credentials.wscred.$SCredentialImpl@2fe744c,realmName=zosConnect,securityName=Fred,realmSecurityName=zosConnect/Fred,uniqueSecurityName=Fred,primaryGroupId=group:zosConnect  
/Manager,accessIdUser:zosConnect/Fred,groupIds:[group:zosConnect/Manager, group:zosConnect/Staff]  
    Private Credential: com.ibm.ws.security.token.internal.SignleSignOnTokenImpl@297f0e18  
  
[7/16/22 17:47:40] [401:866 UTC] 00000525 id=bbb4552f com.ibm.ws.security.context.SubjectManager < getCallerSubject Exit  
    Subject:  
  
    Principal: W$principal$Fred  
    Public Credential: com.ibm.ws.security.credentials.wscred.$SCredentialImpl@2fe744c,realmName=zosConnect,securityName=Fred,realmSecurityName=zosConnect/Fred,uniqueSecurityName=Fred,primaryGroupId=group:zosConnect  
/Manager,accessIdUser:zosConnect/Fred,groupIds:[group:zosConnect/Manager, group:zosConnect/Staff]  
    Private Credential: com.ibm.ws.security.token.internal.SignleSignOnTokenImpl@297f0e18  
  
[7/16/22 17:47:40] [401:866 UTC] 00000525 id=00000000 com.ibm.ws.security.authentication.utility.SubjectHelper < init> Entry  
[7/16/22 17:47:40] [401:866 UTC] 00000525 id=d3f66b2e com.ibm.ws.security.authentication.utility.SubjectHelper < init> Entry  
    com.ibm.ws.security.authentication.utility.SubjectHelper@f0fa0a  
yaml
```

The point of this section is to use the *trace.log* and *messages.out* file when developing an API identify issues with the API and the interactions with the CICS region.

Developing a z/OS Connect APIs that accesses a CICS program

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

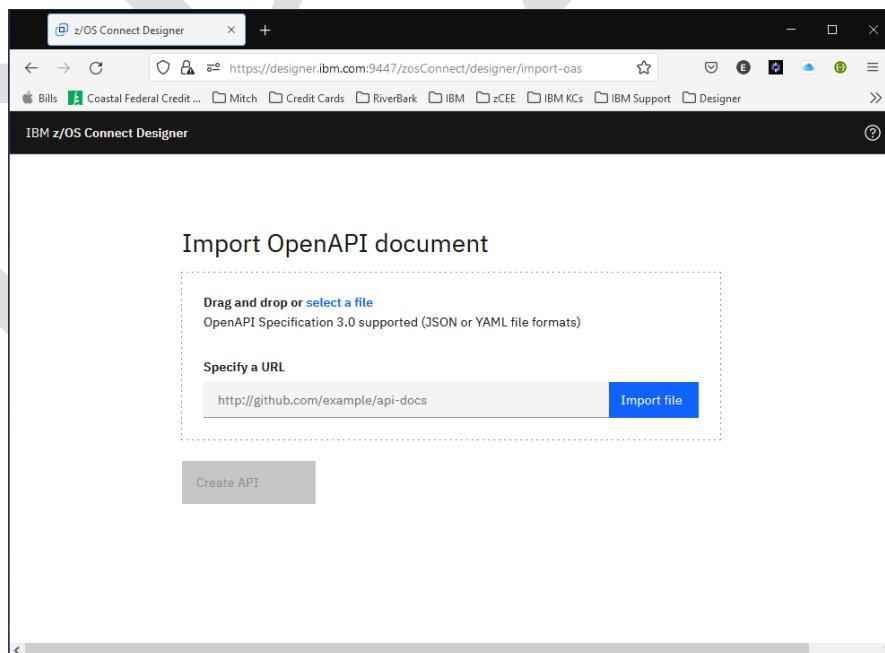


- Switch to root authority by entering command `sudo su` and replying to the password prompt with root's password of `passw0rd`.
- This exercise uses the `cscvinc_zosConnect_1` container. Enter command `podman ps` to see a list of the active containers.
- If this container does not appear in list, enter command `podman start cscvinc_zosConnect_1` to start this container.
- Repeat the `podman ps` command and verify this command now appears in the list of active containers.

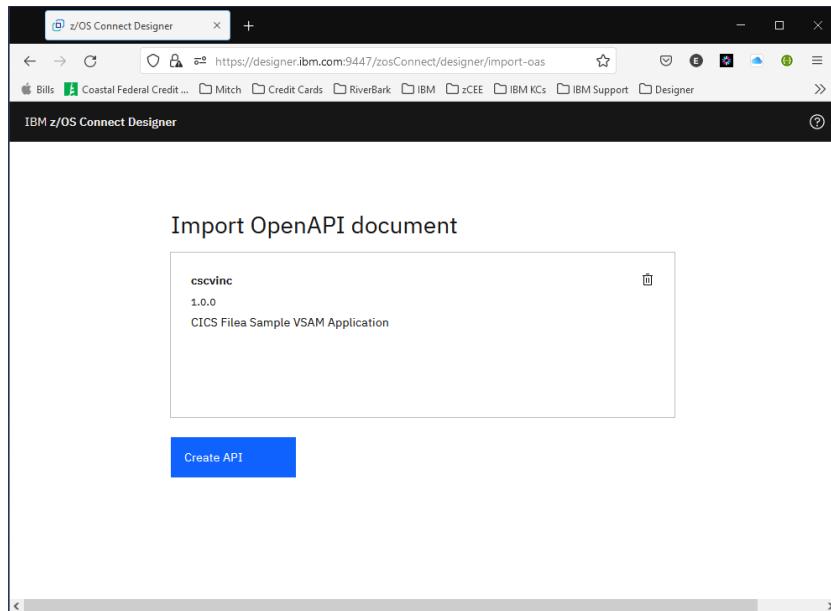
CONTAINER ID	IMAGE NAMES	COMMAND	CREATED	STATUS	PORTS
27dcfdb8760	icr.io/zosconnect/ibm-zcon-designer:3.0.58	"/opt/ibm/helpers/run..."	11 days ago	Up 2 seconds	0.0.0.0:9086->9080/tcp, :::9086->9080/tcp, 0.0.0.0:9449->9443/tcp, :::9449->9443/tcp employees_zosConnect_1

Container housekeeping is now complete.

- Open the Firefox browser and go to URL <https://designer.washington.ibm.com:9447/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 *cscvinc.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 */employee* and */employee/{employee}* will display the *POST*, *GET*, *PUT* and *DELETE* methods associated with these URI paths (see below).

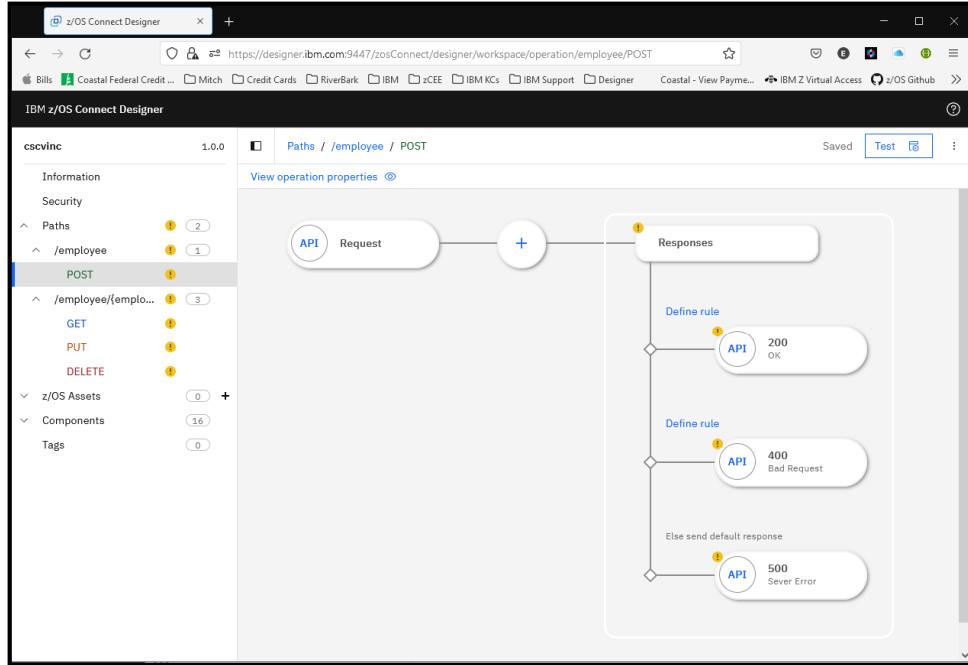
Section	Detail
General	Title: cscvinc, Version: 1.0.0, Description: CICS Filea Sample VSAM Application, Terms of service (URL): -
Contact	Name: -, Email: -, URL: -
License	Name: -, URL: -
Servers	URL: /

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.

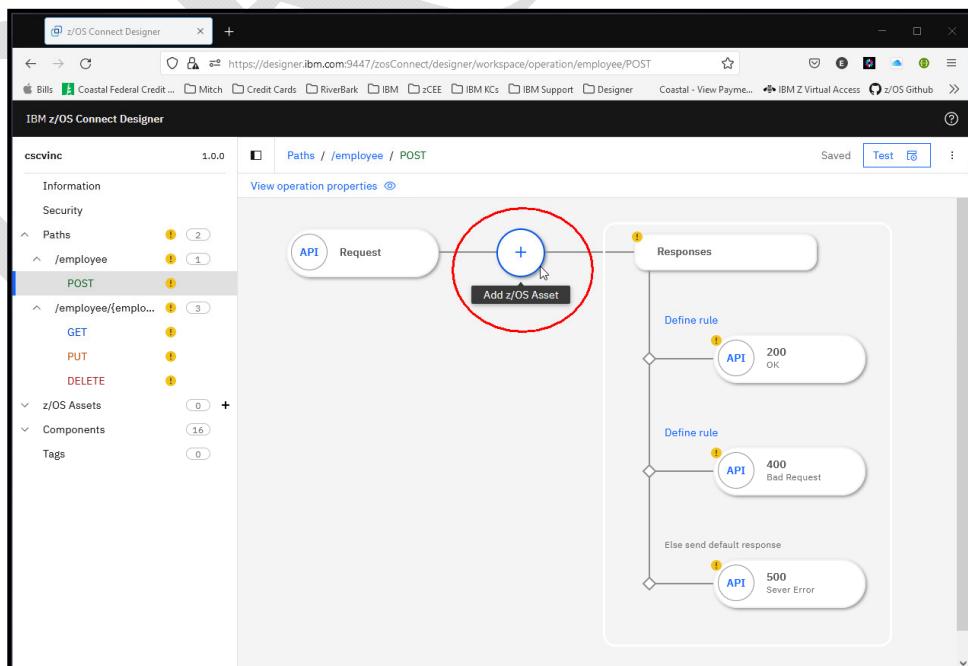
Configure the POST method for URI path /employee

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

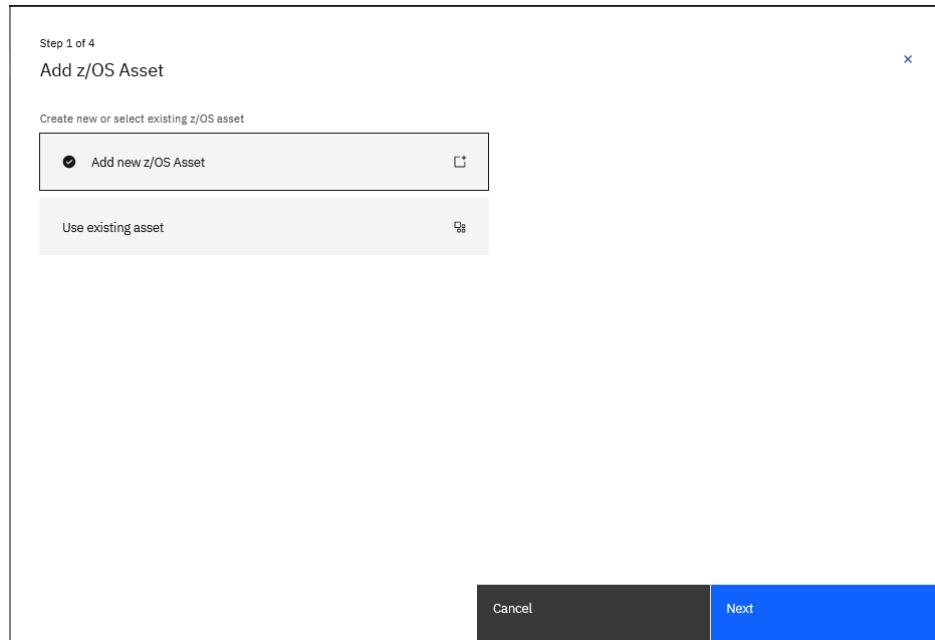


In the example above, we see that the specification document defined 3 responses for this method. One is a 200-status code which indicate the invocation of the method (an insert) was successful. A 400-status code which indicates, in this case, that the insert of the record failed. And finally, a 500-status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

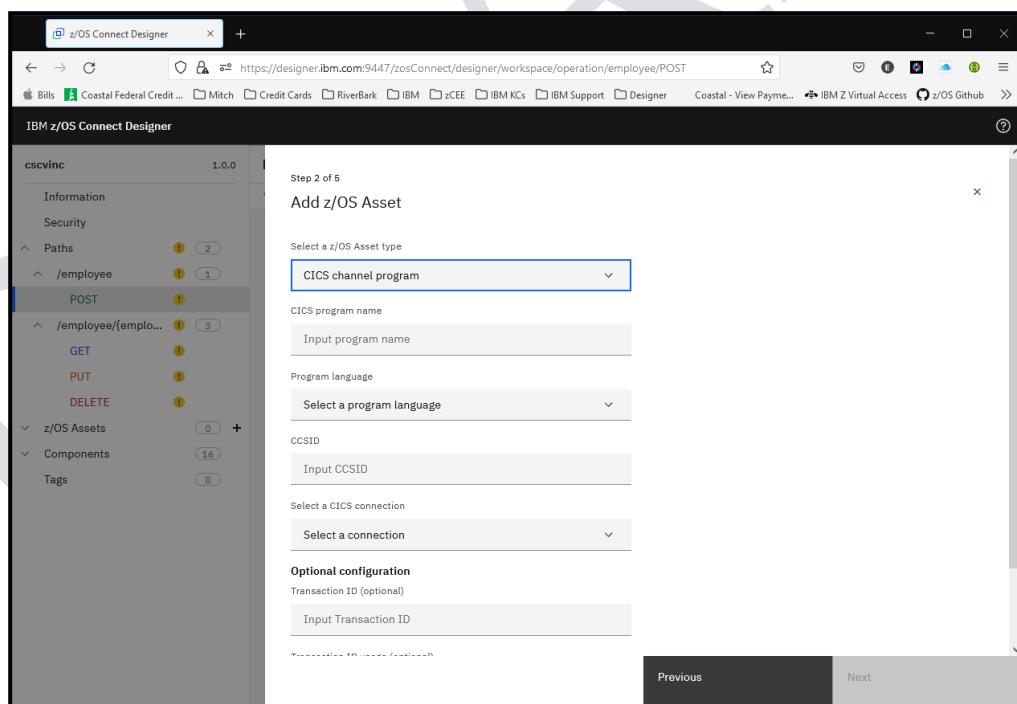
2. The first step in configuring this method for this URI path is to associate it with a z/OS asset or resource. Click the plus sign on the page to start the association of a z/OS asset with this URI path.



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 *CICS channel program*. This action will display additional items on the page.



5. On the *Add z/OS Asset (Step 2 of 5)* page, enter a *CICS program name* of **CSCVINC**, use the pull-down arrow and select the **COBOL** as the *program language*. Enter **037** as the CCSID and use the pull-down arrow and select the **cicsConn** as the *CICS connection*. This enables the **Next** button. Press **Next** to continue.

Step 2 of 5
Add z/OS Asset

Select a z/OS Asset type
CICS channel program

CICS program name
CSCVINC

Program language
COBOL

CCSID
037

Select a CICS connection
cicsConn

Optional configuration

Transaction ID (optional)
Input Transaction ID

Transaction ID usage (optional)
Select usage

Previous **Next**

Tech-Tip: The name **cicsConn** is the name of the `zosconnect cicsIpicConnection` configuration element described earlier in this exercise.

6. On the *Add z/OS Asset (Step 3 of 5)* page, details of the request container will be provided. Press the **Add Container** button and enter a name of **containerCscvinc** for the *CICS container* name.
 7. Next press the **Import Data structure** to start the definition of the layout of this container.

Step 3 of 5
Add z/OS Asset

Define the request CICS channel program

Request channel

CICS container name
containerCscvinc

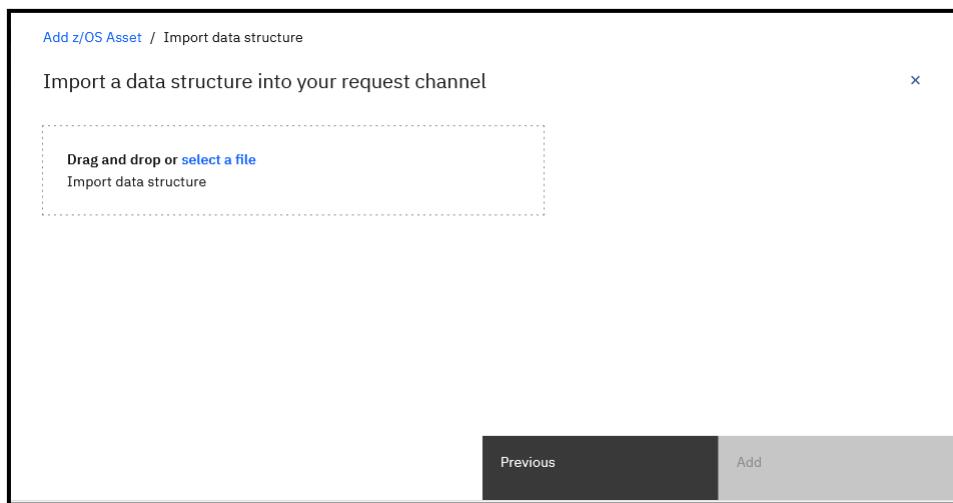
CICS container type
BIT

Import data structure

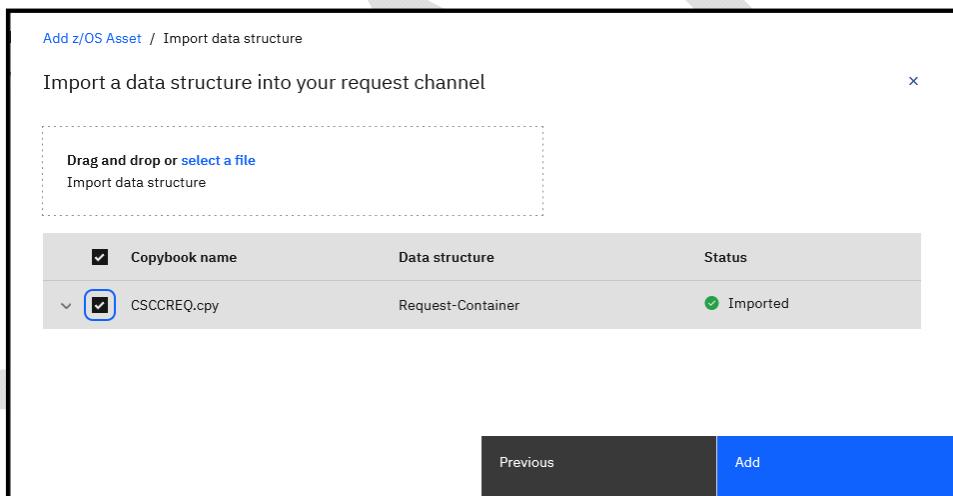
Add container

Previous **Next**

8. On the *Add z/OS Asset / Import data structure* page, press the *select a file* area to start the import of the COBOL structure that represent the request container.



9. Traverse to directory *c:/z/cicslab* and select file *CSCCREQ.cpy*. This file contains the COBOL source of the request container. Click the **Open** button to import the COBOL source into the *Designer*. Check the box beside *CSCCREQ.cpy* and click the **Add** button to continue.



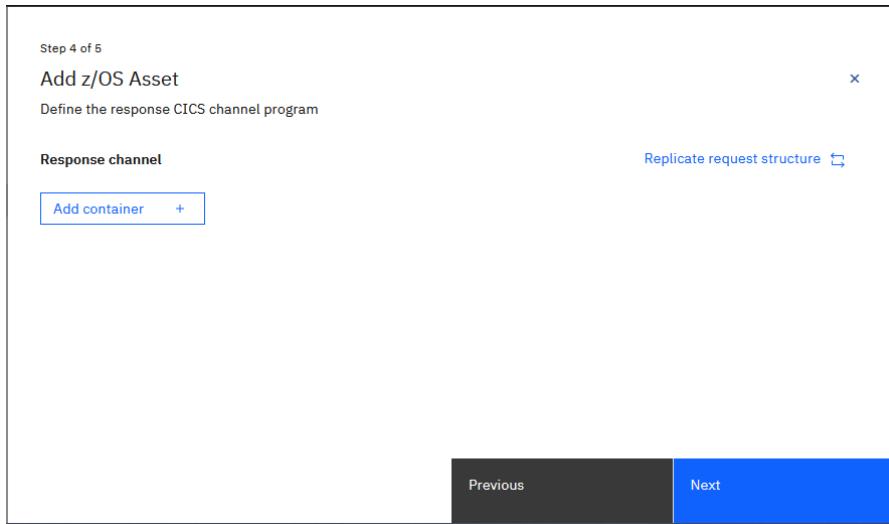
10. On the *Add z/OS Asset (Step 3 of 6)* page, the COBOL structure of this container will be displayed (see below). Additional input or request containers be added by clicking the **Add container** button and repeating this process. Since this application expects only one input or request container, click the **Next** button to continue.

The screenshot shows the 'Add z/OS Asset' interface at Step 3 of 5. The title bar says 'Add z/OS Asset' and 'Define the request CICS channel program'. The main area is titled 'Request channel' and contains a table for defining CICS container types. A row is selected for 'containerCscvinc' with 'BIT' as the type. Below the table, a large block of COBOL code is displayed:

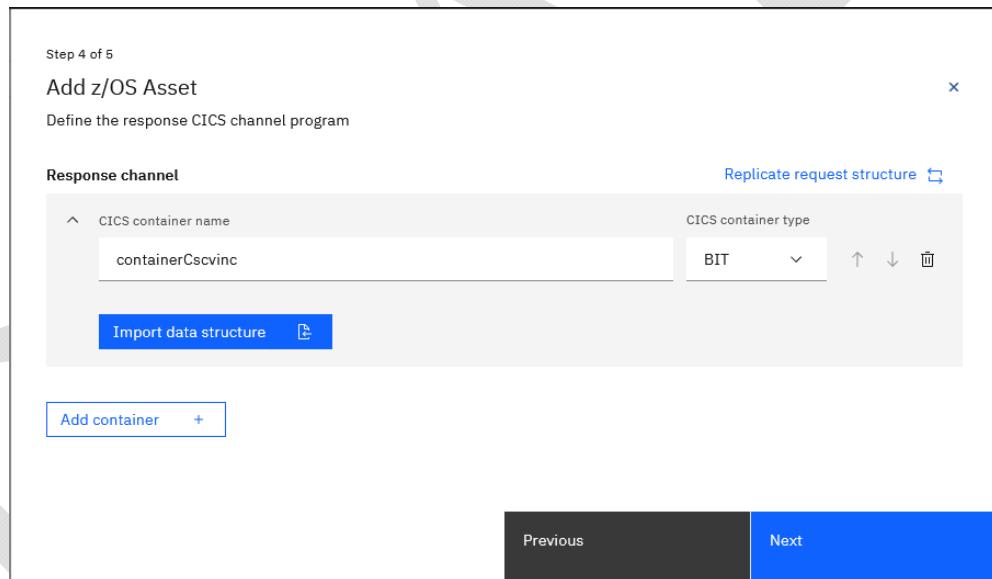
```
01 Request-Container.  
03 ACTION PIC X USAGE DISPLAY.  
03 USERID PIC X(8) USAGE DISPLAY.  
03 FILEA-AREA.  
05 STAT PIC X USAGE DISPLAY.  
05 NUMB PIC X(6) USAGE DISPLAY.  
05 NAME PIC X(20) USAGE DISPLAY.  
05 ADDRX PIC X(20) USAGE DISPLAY.  
05 PHONE PIC X(8) USAGE DISPLAY.  
05 DATEX PIC X(8) USAGE DISPLAY.  
05 AMOUNT PIC X(8) USAGE DISPLAY.  
05 COMMENT PIC X(9) USAGE DISPLAY.
```

Below the code, there's a blue 'Import data structure' button with a file icon. At the bottom left is an 'Add container' button with a plus sign. At the bottom right, there are 'Previous' and 'Next' buttons, with 'Next' being highlighted in blue.

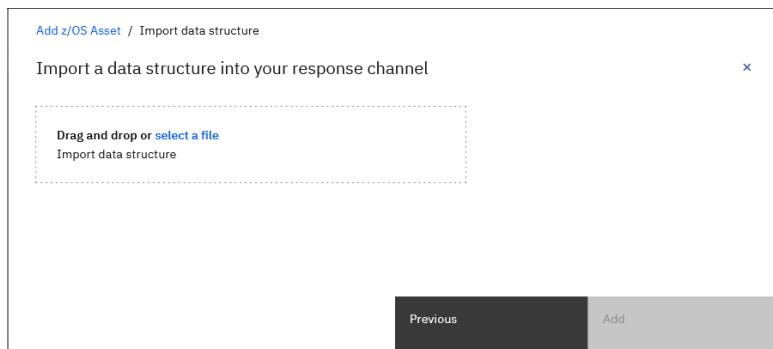
11. The response container is configured on the *Add z/OS Asset (Step 4 of 5)* page. If the response container that exactly matches the request container, then clicking the *Replicate request structure* will create a response container with the same details as the request container. In this case, the response contains does not match the request container, so click the **Add container** button to add the details of the response container.



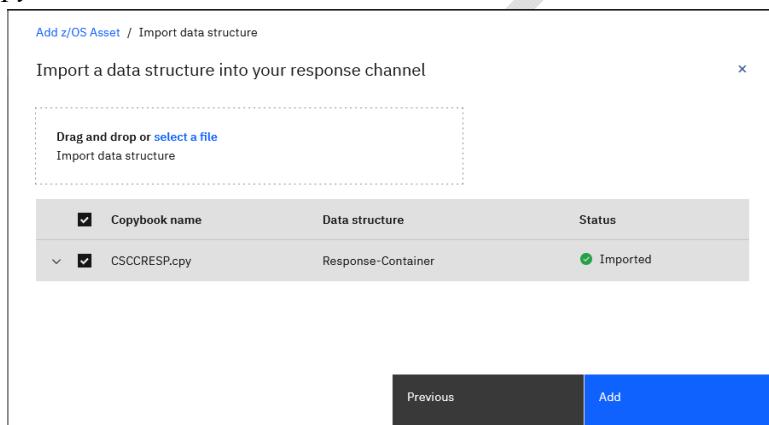
12. This action will cause the **Import data structure** to appear. Enter *containerCscvinc* as the *CICS container name* (remember the application returns a response container using the same name are the request container). Click on the **Import data structure** button to import the layout of the response container.



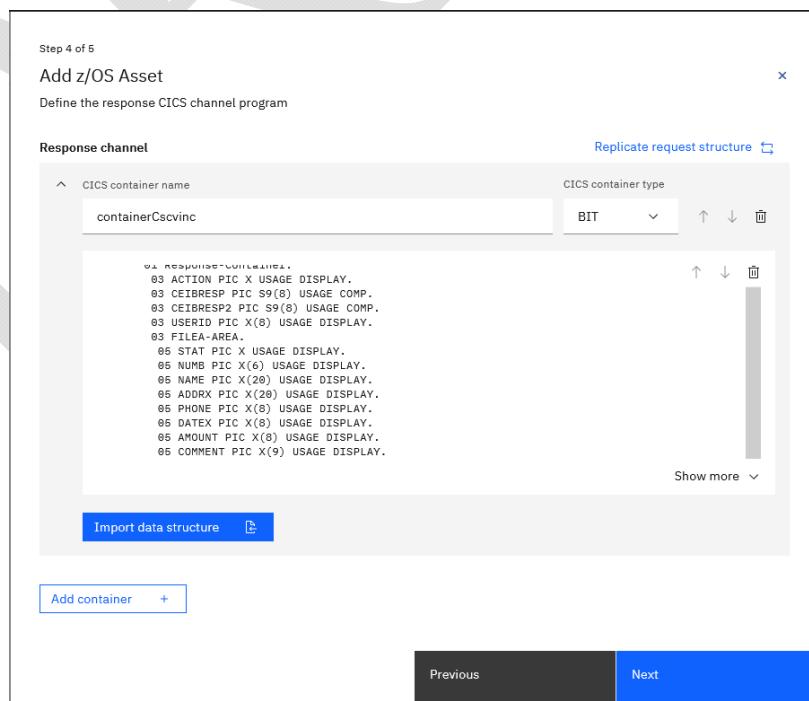
13. On the *Add z/OS Assert/Import data structure* page. Click on the *select a file* area to continue.



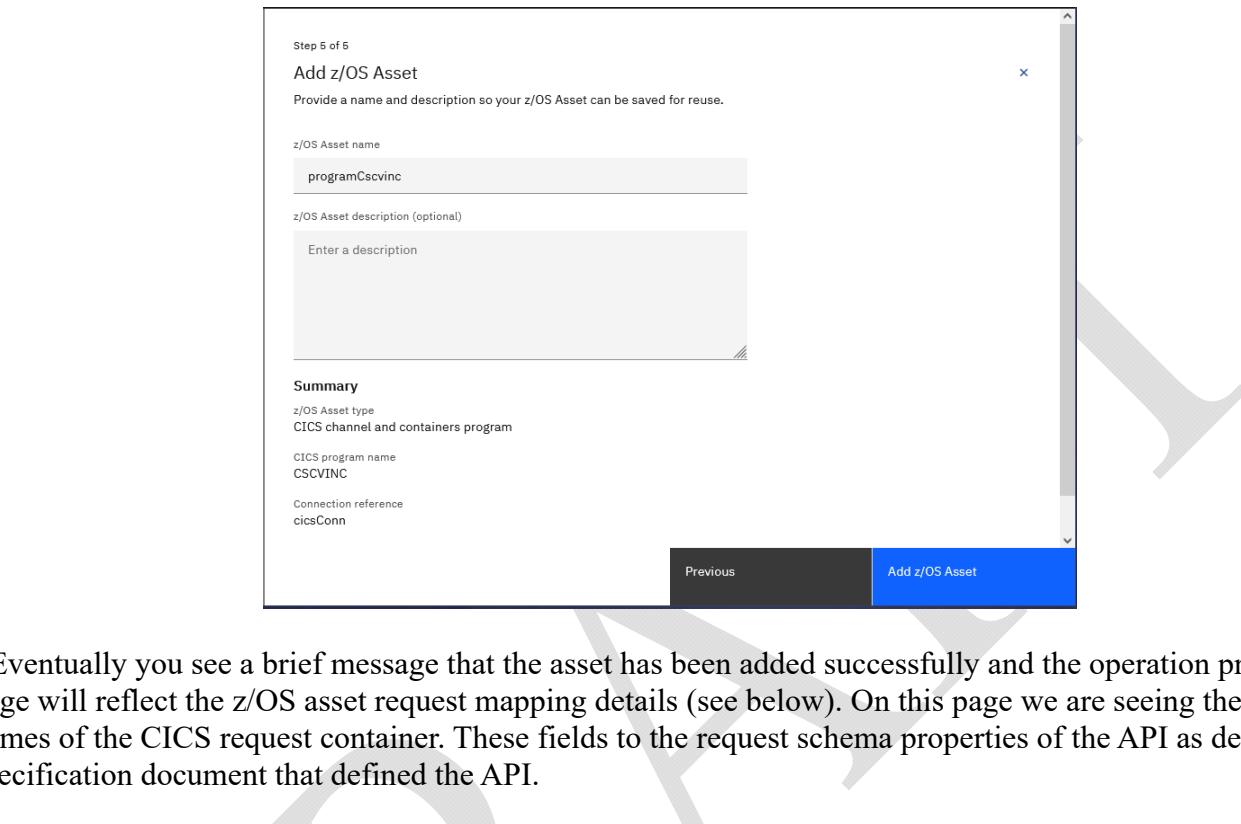
14. Traverse to directory *c:/z/cicslab* and select file *CSCCRESP.cpy* which contains the COBOL source of the response container. Click the **Open** button to import the COBOL source into the *Designer*. Check the box beside *CSCCRESP.cpy* and click the **Add** button to continue.



15. On the *Add z/OS Asset (Step 4 of 5)* page, the COBOL structure of this container will be displayed (see below). Additional input or request containers be added by clicking the **Add container** button and repeating this process. This application only produces one response container, so click the **Next** button to continue.



16. On the *Add z/OS Asset (Step 5 of 5)* page, we are required to provide a name and/or optionally provide a description of the z/OS asset. In this exercise, enter a value of **programCscvinc** as the name of the z/OS asset (a CICS application) and click the **Add z/OS Asset** button to complete the definition of this program as an asset and to continue.



Step 5 of 5
Add z/OS Asset
Provide a name and description so your z/OS Asset can be saved for reuse.

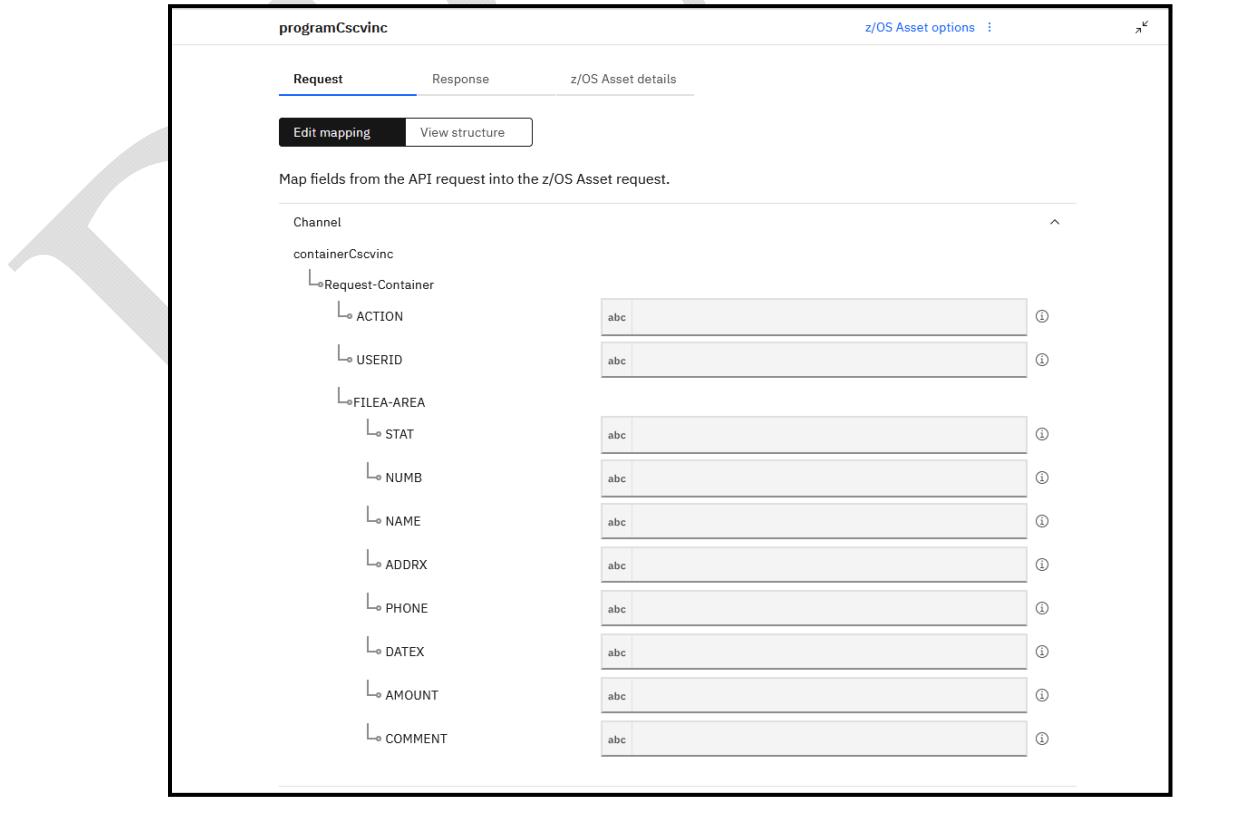
z/OS Asset name
programCscvinc

z/OS Asset description (optional)
Enter a description

Summary
z/OS Asset type
CICS channel and containers program
CICS program name
CSCVINC
Connection reference
cicsConn

Previous **Add z/OS Asset**

17. 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 field names of the CICS request container. These fields map to the request schema properties of the API as defined in the specification document that defined the API.



programCscvinc **z/OS Asset options**

Request **Response** **z/OS Asset details**

Edit mapping **View structure**

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

Channel	Container	Field	Value
containerCscvinc	Request-Container	ACTION	abc
		USERID	abc
	FILEA-AREA	STAT	abc
		NUMB	abc
		NAME	abc
		ADDRX	abc
		PHONE	abc
		DATEX	abc
		AMOUNT	abc
		COMMENT	abc

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.

The screenshot shows the 'Request' tab of the IBM z/OS Connect interface. The path 'programCscvinc' is selected. The 'Request' tab is active. The 'z/OS Asset details' tab is visible at the top right. Below the tabs, there are two buttons: 'Edit mapping' (highlighted in black) and 'View structure'. A large red oval highlights the 'ACTION' field under the 'Request-Container' section. The 'ACTION' field contains the value 'abc' and has a blue border. To its right are three icons: a clipboard, a magnifying glass, and a question mark. Below the 'ACTION' field are other fields: 'USERID', 'FILEA-AREA', 'STAT', 'NUMB', and 'NAME', each with a corresponding input field containing 'abc'.

18. Now map the CICS container 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).

This screenshot is similar to the previous one, showing the 'Request' tab for 'programCscvinc'. The 'ACTION' field under 'Request-Container' is highlighted in blue and has a red oval around it. The other fields ('USERID', 'FILEA-AREA', 'STAT', 'NUMB', 'NAME') are also present but not highlighted.

19. 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. The title bar says 'Available mappings' and the path is '...ServiceOperation employeeData request employeeDetails'. The list contains the following items, each with a circled 'API' icon and a blue border around the value:

- (API) employeeDetails abc
- (API) employeeNumber abc
- (API) name abc
- (API) address abc
- (API) phoneNumber abc
- (API) date abc

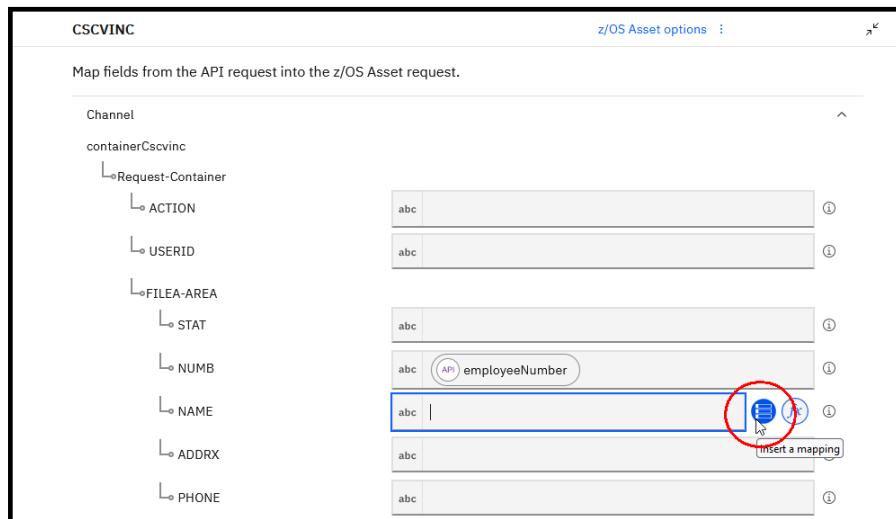
There are two ways to map the CICS request container fields to the corresponding specification document API request fields. Both will be demonstrated in this section of the exercise. Use whichever method you prefer when mapping fields later in this exercise.

20. Select an empty CICS request container field and start typing the corresponding API request message field name. For example, entering the string **em** in the area beside NUMB 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).

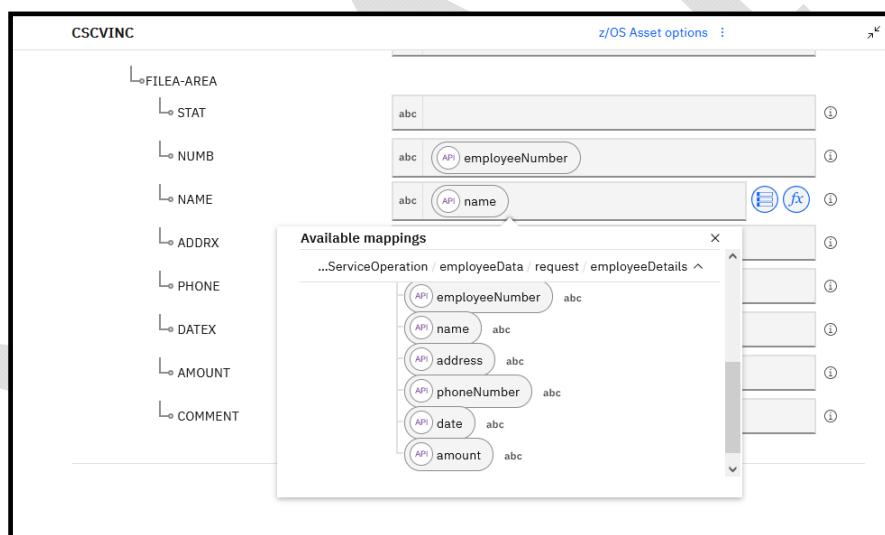
21. Select the field and this will cause the API's request message field name to populate the area beside field name in the container area and complete the mapping.

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

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



23. 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 request body *name* field.



24. Use either technique to complete the mappings of the container fields to the API request body fields. When completed, the results should look something like this the page below.

The screenshot shows the 'Request' tab of the IBM z/OS Connect mapping interface. The top navigation bar includes 'z/OS Asset options' and a back arrow. Below the tabs are 'Edit mapping' and 'View structure' buttons. A header message says 'Map fields from the API request into the z/OS Asset request.' The main area shows a hierarchical tree of fields:

- Channel
- containerCscvinc
 - Request-Container
 - ACTION (abc)
 - USERID (abc)
 - FILEA-AREA
 - STAT (abc)
 - NUMB (abc) (API) employeeNumber
 - NAME (abc) (API) name
 - ADDRX (abc) (API) address
 - PHONE (abc) (API) phoneNumber
 - DATEX (abc) (API) date
 - AMOUNT (abc) (API) amount (highlighted with a blue border)
 - COMMENT (abc)

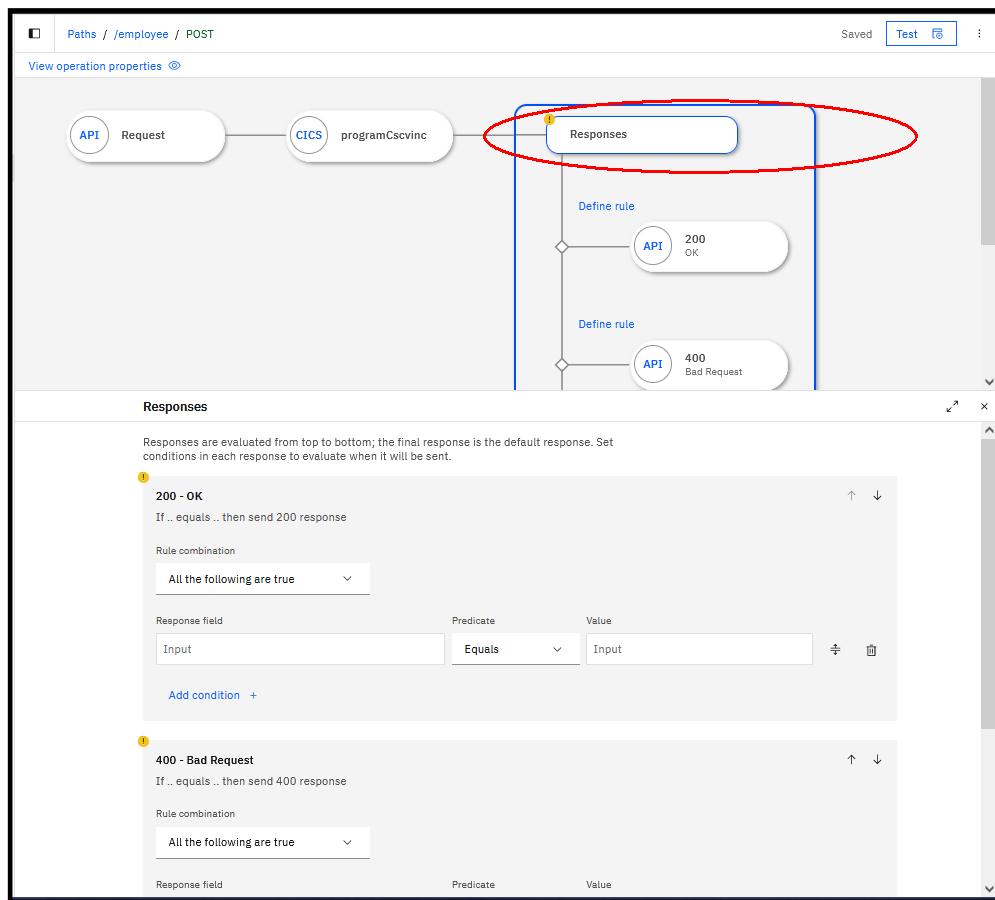
25. Note that not all of the container fields have been mapped from the API request message but by providing request message field names. For the *ACTION* field, the application requires that this field have a value of **I** in order for a record to be inserted in the VSAM data sets. The *USERID* is used by another client application and has no meaning for us, so leave this field blank. Set the value of the *STAT* field to an asterisk * and set the value of the *COMMENT* field to **`{{\$substring($now(), 2,8)}}`** (the current date).

CSCVINC

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

Channel		z/OS Asset options :
containerCscvinc		
└ Request-Container		
└ ACTION	abc I	(i)
└ USERID	abc	(i)
└ FILEA-AREA		
└ STAT	abc *	(i)
└ NUMB	abc API employeeNumber	(i)
└ NAME	abc API name	(i)
└ ADDRX	abc API address	(i)
└ PHONE	abc API phoneNumber	(i)
└ DATEX	abc API date	(i)
└ AMOUNT	abc API amount	(i)
└ COMMENT	abc <code>{{\\$substring(\$now(), 14,8)}}</code>	(i)

26. The next step is to evaluate the responses that come back in the CICS program response container. Select the *Responses* box in the *View operation properties* page.



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

The contents of the CEIBRESP and CEIBRESP2 fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was inserted as intended. The application will set the CEIBRESP and CEIBRESP2 fields to zero if a record was successfully insert into the VSAM data set. Otherwise, the insert failed. We are going to check and provide a message if a duplicate record condition (CEIBRESP = 14 and CEIBRESP2=150) was raised. Otherwise a message indicating a severe error has occurred along with the values of CEIBRESP and CEIBRESP2 fields.

The screenshot shows the 'Responses' configuration for a POST path. It includes three response rules:

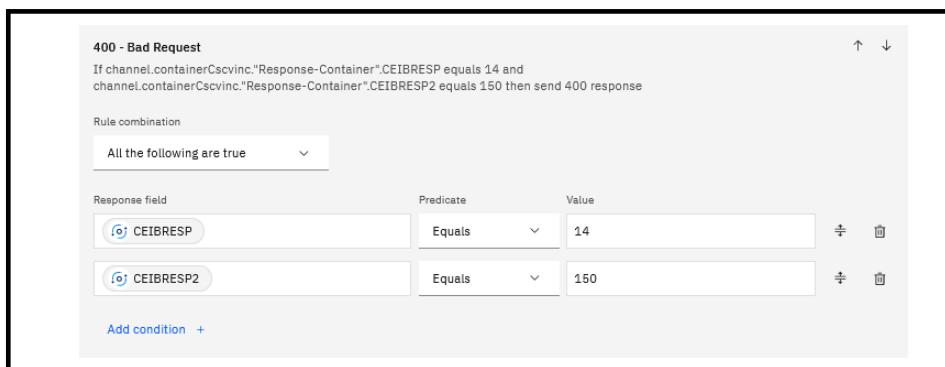
- 200 - OK**: If .. equals .. then send 200 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input. Add condition: +
- 400 - Bad Request**: If .. equals .. then send 400 response. Rule combination: All the following are true. Response field: Input,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

28. Under the *200 – OK* response, Enter the string ***CEI*** in the *Input* area under *Response field*. This will display all the fields in the CICS response container which match this string (position of the string in the field name does not matter). If the string matches any portion of the field name, that field will be displayed. In this case, map or select the ***CEIBRESP*** field in the CICS response container. Leave the *Predicate* as *Equals* and enter ***0*** in the *Value* field for *Value*.

29. Next add a condition check for the value of response container field ***CEIBRESP2*** by clicking on *Add condition* in the *200 – OK* evaluation.

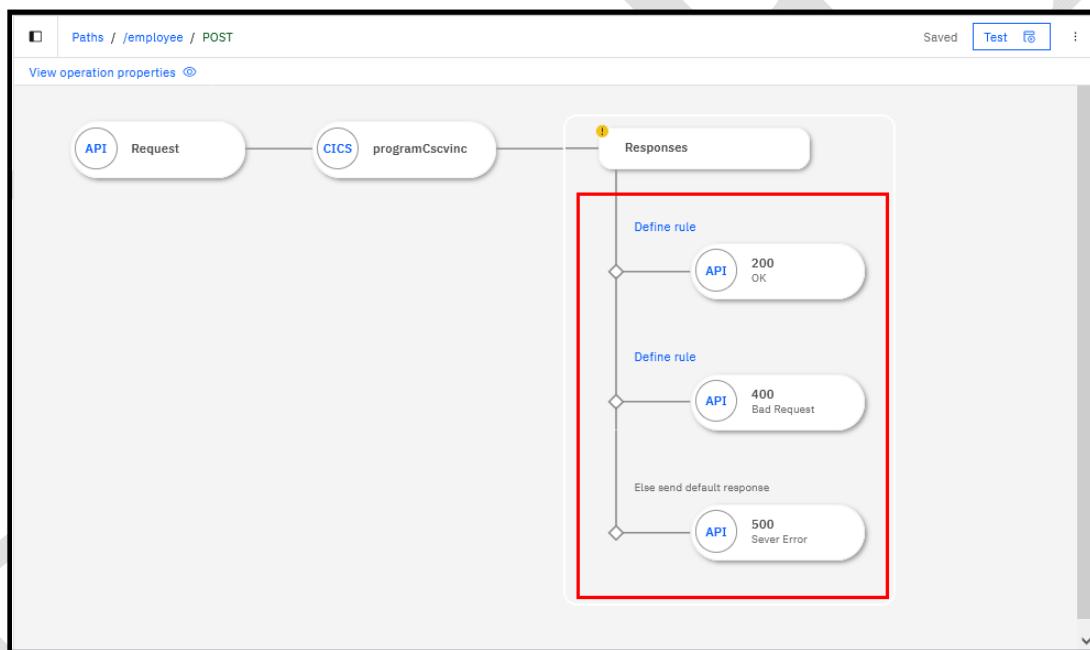
30. Use the same technique described above to add a check for response container field ***CEIBRESP2***. Leave the *Predicate* as *Equals* and set the value to ***0*** as shown below:

- ___ 31. For the *400 – Bad Request* check, add a check for container field ***CEIBRESP*** equaling ***14*** and response container ***CEIBRESP2*** equaling ***150***.



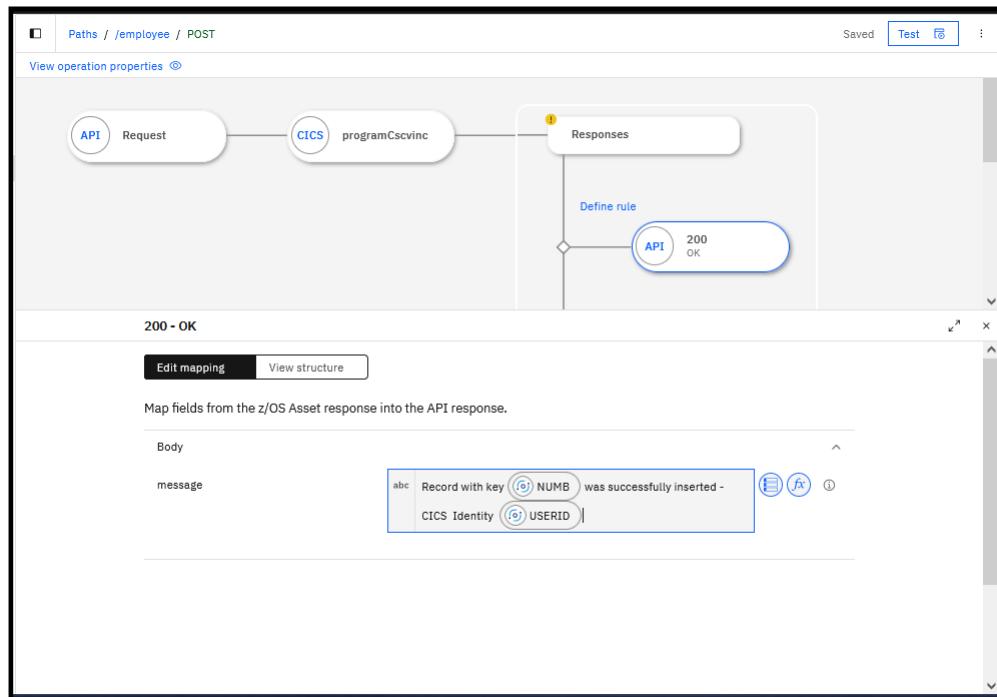
- ___ 32. If neither of these connections are met, simply return with a HTTP 500 status code.

- ___ 33. Next the API response messages need to be configured for each of these potential status codes.



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

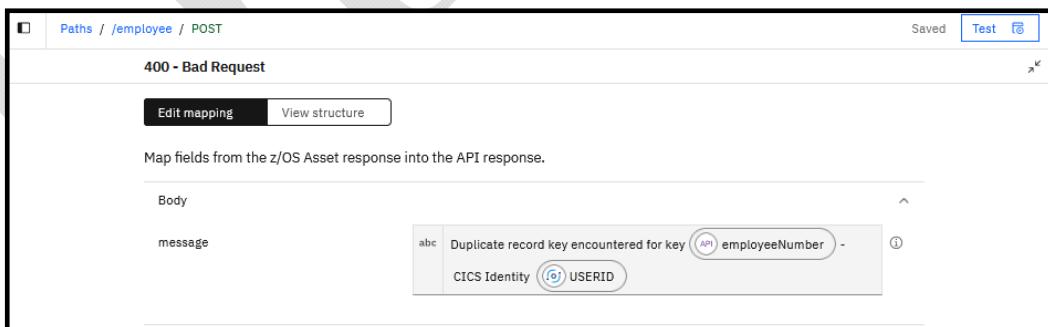
**Record with key {{\\$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEAREA".NUMB}} was successfully inserted - CICS Identity
{{\\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**



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

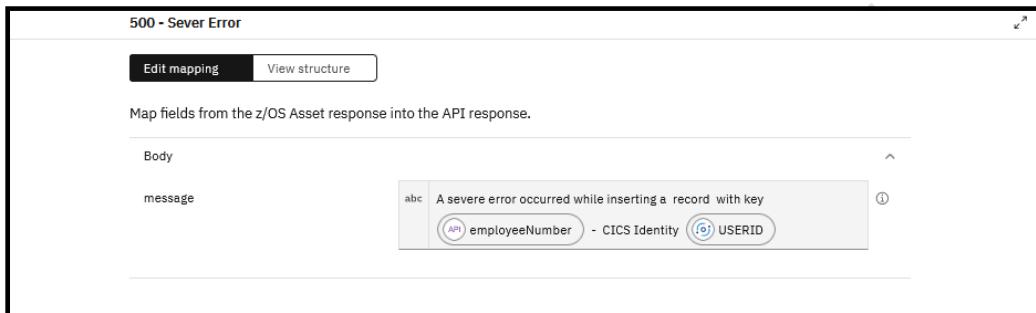
35. Select the response for *400 Add failed* response mapping and paste the text below in the *message* area.

**Duplicate record key encountered for key
{{\\$apiRequest.body.EmployeeInsertServiceOperation.employeeData.request.employeeDetails.employeeNumber}} - CICS Identity {{\\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**



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

```
A severe error occurred while inserting a record with key
{{zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} - CICS
Identity {{zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
{{zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP}}
{{zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2}}
```

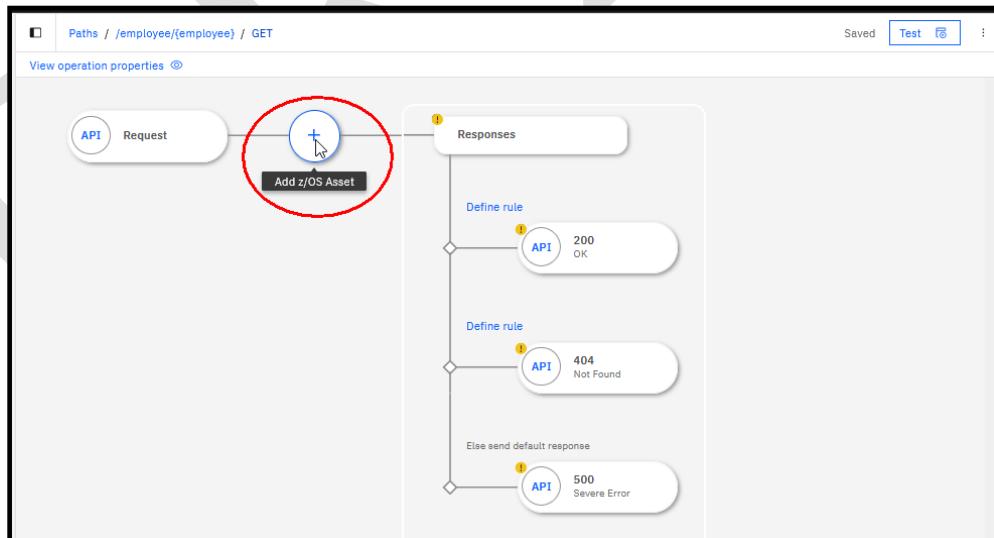


The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

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

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

1. Start by adding an existing z/OS Asset for CICS program *CSCVINC* to the GET method by using the same steps as performed before.



2. On the *Step 1 of 2* page, select *Use existing asset* and click **Next**.

3. On the *Step 2 of 2* page, select the radio button beside *programCSCVINC* and click the **Add z/OS Asset** button.

4. Use the *Insert a mapping tool* to display the API request mapping for this method. Notice that there is no Body mapping. There is only mappings for headers properties and path parameters.

5. So, the only mapping from the API request for this method is to map the path parameter *employee* to the container field *NUMB*. Use your preferred technique to do this mapping and set the value of *ACTION* to *S*.

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

The contents of the *CEIBRESP* and *CEIBRESP2* fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was returned as intended. The application returns the values of the EIBRESP and EIBRESP2 from the EXECI CICS API in the container *CEIBRESP* and *CEIBRESP2* fields. Their values will be zero if a record was successfully read from the VSAM data set. Otherwise, the record did not exist, e.g., *CEIBRESP*=13 and *CEIBRESP2*=80 or some other error occurred will be returned.

The screenshot shows the 'Responses' configuration for a GET request to the path '/employee/{employee}'. The interface is divided into three main sections corresponding to different HTTP status codes:

- 200 - OK**: If the input equals the input, then send a 200 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input.
- 404 - Not Found**: If the input equals the input, then send a 404 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input.
- 500 - Internal Server Error**: Else send default response.

7. Under the *200 – OK* response, use your preferred mapping technique to check the ***CEIBRESP*** and ***CEIBRESP2*** response fields are equal to *0*.

200 - OK
If channel.containerCscvinc."Response-Container".CEIBRESP equals 0 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 0 then send 200 response

Rule combination
All the following are true

Response field	Predicate	Value
<code>!o: CEIBRESP</code>	Equals	0
<code>!o: CEIBRESP2</code>	Equals	0

Add condition +

8. For the *404 – Not Found* check, use your preferred mapping technique to check the ***CEIBRESP*** container response field is equal to a value of *13* and the ***CEIBRESP2*** container response field is equal to a value of *80*.

404 - Not Found
If channel.containerCscvinc."Response-Container".CEIBRESP equals 13 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 80 then send 404 response

Rule combination
All the following are true

Response field	Predicate	Value
<code>!o: CEIBRESP</code>	Equals	13
<code>!o: CEIBRESP2</code>	Equals	80

Add condition +

9. If none of these connections are met, simply return with a HTTP 500 status code.

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



11. Select the response for *200 OK* and map the fields from the CICS response container. Start by mapping the message API response field the message below:

Record with key \${zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEAREA".NUMB} was successfully retrieved - CICS identity \${zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}

The screenshot shows the 'Edit mapping' view for a GET request to '/employee/{employee}'. The response status is '200 - OK'. The mapping table lists fields from the 'zosAssetResponse' message, specifically the 'FILEAREA' and 'USERID' fields, which are mapped to the 'NUMB' and 'USERID' fields in the 'EmployeeSelectServiceOperationResponse' message respectively.

zosAssetResponse	EmployeeSelectServiceOperationResponse
FILEAREA.NUMB	NUMB
USERID	USERID

12. Complete the mapping for the other container fields to the corresponding API response message fields.

The screenshot shows the 'Edit mapping' view for a GET request to '/employee/{employee}'. The response status is '200 - OK'. The mapping table now includes all fields from the 'zosAssetResponse' message, mapping them to the corresponding fields in the 'EmployeeSelectServiceOperationResponse' message. The 'NUMB' field is highlighted with a blue border.

zosAssetResponse	EmployeeSelectServiceOperationResponse
FILEAREA.NUMB	NUMB
FILEAREA.NAME	NAME
FILEAREA.ADDRX	ADDRX
FILEAREA.PHONE	PHONE
FILEAREA.DATEX	DATEX
FILEAREA.AMOUNT	AMOUNT
FILEAREA.COMMENT	COMMENT

13. 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 response container field. Since the record was not found, the response container field for NUMB is empty.

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

**A severe error occurred while retrieving a record with key
`{{$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}}` - CICS Identity
 `{{$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}`
 `{{$zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP}}`:
 `{{$zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2}}`**

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 `programCscvinc` 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 */employee* and the *GET* for */employee/{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 displays the URL `https://designer.ibm.com:9447/api/explorer/`. The main content area is titled "Employee". It lists four methods for the `/employee` endpoint:

- POST** `/employee`
- GET** `/employee/{employee}`
- PUT** `/employee/{employee}`
- DELETE** `/employee/{employee}`

Below the methods, under the "Schemas" heading, are four schema definitions:

- `postEmployeeInsertService_request`
- `deleteEmployeeDeleteService_response_200`
- `putEmployeeUpdateService_request_EmployeeUpdateServiceOperation_employeeData`
- `deleteEmployeeDeleteService_response_500`

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 **fredpwd** (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 /employee* URI path to display the request body view of the URI path.

The screenshot shows the Open Liberty OpenAPI UI interface. The top navigation bar has tabs for 'z/OS Connect Designer' and 'OpenAPI UI'. The main title is 'Open Liberty' with 'Liberty REST APIs 1.0.0 OAS3'. Below it, a sub-header says 'Discover REST APIs available within Liberty'. A 'Servers' dropdown is set to 'https://designer.ibm.com:9447'. The main content area is titled 'Employee' and shows a 'POST /employee' endpoint. The 'Parameters' section includes an 'Authorization' header. The 'Request body' section is marked as 'required' and has a 'Try it out' button. It shows a JSON schema for 'EmployeeInsertServiceoperation' with fields like 'employeeNumber', 'name', 'address', 'phoneNumber', 'date', and 'amount'. The 'Responses' section shows a 200 OK status with 'No links'.

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

This screenshot shows the same 'Employee' endpoint from the previous screen, but with the 'Try it out' button now highlighted in red. The 'Request body' section is visible, showing the JSON schema for the employee data.

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

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948478",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

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 HTTP 200 - success message.

The screenshot shows the IBM API Designer interface with the following details:

- Request URL:** <https://designer.ibm.com:9447/employee>
- Server response:**
 - Code:** 200
 - Details:** Response body contains: {"message": "Record with key 948478 was successfully inserted - CICS Identity CICSUSER"}
 - Response headers:** content-language: en-US, content-length: 89, content-type: application/json, date: Thu, 21 Jul 2022 19:57:08 GMT, x-firebase-spdy: h2, x-powered-by: Servlet/4.0

7. Press the **Execute** button again and observe the results. A row for this employee number already existed in the VSAM data set so the request failed with an HTTP 400 – bad request.

```

Responses

Curl
curl -X 'POST' \
  'https://designer.ibm.com:9447/employee' \
  -H 'Accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "EmployeeInsertServiceOperation": {
      "employeeData": {
        "request": {
          "employeeDetails": {
            "employeeNumber": "948478",
            "date": "12/31/22",
            "amount": "$100.00",
            "address": "RTP NC",
            "phoneNumber": "0005",
            "name": "M Johnson"
          }
        }
      }
    }
  }'

Request URL
https://designer.ibm.com:9447/employee

Server response
Code Details

400 Error: Bad Request

Response body
{
  "message": "Duplicate record key encountered for key 948478 - CICS Identity CICSUSER"
}

Response headers

```

8. Scroll down and click on *GET /employee/{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.

```

Responses

Curl
curl -X 'GET' \
  'https://designer.ibm.com:9447/employee/948478' \
  -H 'Accept: application/json'

Request URL
https://designer.ibm.com:9447/employee/948478

Server response
Code Details

200 Response body
{
  "summary": {
    "message": "Record with key 948478 was successfully retrieved - CICS identity CICSUSER"
  },
  "detail": {
    "EmployeeSelectServiceOperationResponse": {
      "employeeData": {
        "response": {
          "employeeDetails": {
            "employeeNumber": "948478",
            "name": "M Johnson",
            "address": "RTP NC",
            "phoneNumber": "0005",
            "date": "12/31/22",
            "amount": "$100.00",
            "comment": "$57.08 .58"
          }
        }
      }
    }
  }
}

Response headers
content-language: en-US
content-length: 353
content-type: application/json
date: Thu, 14 Jul 2022 20:08:37 GMT
x-fairfax-spy: 12
x-powered-by: Servlet/4.0

Responses

```

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

The screenshot shows the 'Responses' section of the API designer. It includes a 'Curl' command, a 'Request URL' (https://designer.ibm.com:9447/employee/121212), and a 'Server response' block. The response code is 404, indicating 'Error: Not Found'. The response body contains the JSON object: { "message": "Record for employee number 121212 was not found" }. The response headers include: content-language: en-US, content-type: application/json, date: Thu, 14 Jul 2022 20:03:26 GMT, x-firefox-spy: h2, and x-powered-by: Servlet/4.0.

The table below list the contents of the VSAM data set.

stat	numb	name	addrx	Phone	datex	amount	comment
Y	000100	S. D. BORMAN	SURREY, ENGLAND	32156778	26 11 81	\$0100.11	*****
Y	000102	J. T. CZAYKOWSKI	WARWICK, ENGLAND	98356183	26 11 81	\$1111.11	*****
Y	000104	M. B. DOMBEY	LONDON, ENGLAND	12846293	26 11 81	\$0999.99	*****
Y	000106	A. I. HICKSON	CROYDON, ENGLAND	19485673	26 11 81	\$0087.71	*****
Y	000111	ALAN TULIP	SARATOGA, CALIFORNIA	46120753	01 02 74	\$0111.11	*****
Y	000762	SUSAN MALAIKA	SAN JOSE, CALIFORNIA	22312121	01 06 74	\$0000.00	*****
Y	000983	J. S. TILLING	WASHINGTON, DC	34512120	21 04 75	\$9999.99	*****
Y	001222	D.J.VOWLES	BOBLINGEN, GERMANY	70315551	10 04 73	\$3349.99	*****
Y	001781	TINA J YOUNG	SINDELFINGEN, GERMANY	70319990	21 06 77	\$0009.99	*****
Y	003210	B.A. WALKER	NICE, FRANCE	12345670	26 11 81	\$3349.99	*****
N	003214	PHIL CONWAY	SUNNYVALE, CAL.	34112120	00 06 73	\$0009.99	*****
N	003890	BRIAN HARDER	NICE FRANCE	00000000	28 05 74	\$0009.99	*****
N	004004	JANET FOUCHE	DUBLIN, IRELAND	71112121	02 11 73	\$1259.99	*****
N	004445	DR. P. JOHNSON	SOUTH BEND, S.DAK.	61212120	26 11 81	\$0009.99	*****
N	004878	ADRIAN JONES	SUNNYVALE, CALIF.	32212120	10 06 73	\$5399.99	*****
N	005005	A. E. DALTON	SAN FRANCISCO, CA.	00000001	01 08 73	\$0009.99	*****
N	005444	ROS READER	SARATOGA, CALIF.	67712120	20 10 74	\$0809.99	*****
N	005581	PETE ROBBINS	BOSTON, MASS.	41312120	11 04 74	\$0259.99	*****
Y	006016	SIR MICHAEL ROBERTS	NEW DELHI, INDIA	70331211	21 05 74	\$0009.88	*****
N	006670	IAN HALL	NEW YORK, N.Y.	21212120	31 01 75	\$3509.88	*****
Y	006968	J.A.L. STAINFORTH	WARWICK, ENGLAND	56713821	26 11 81	\$0009.88	*****
N	007007	ANDREW WHARMBY	STUTTGART, GERMANY	70311000	10 10 75	\$5009.88	*****
N	007248	M. J. AYRES	REDWOOD CITY, CALF.	33312121	11 10 75	\$0009.88	*****
Y	007779	MRS. A. STEWART	SAN JOSE, CALIF.	41512120	03 01 75	\$0009.88	*****
Y	009000	P. E. HAVERCAN	WATERLOO, ONTARIO	09876543	21 01 75	\$9000.00	*****
Y	100000	M. ADAMS	TORONTO, ONTARIO	03415121	26 11 81	\$0010.00	*****
Y	111111	C. BAKER	OTTAWA, ONTARIO	51212003	26 11 81	\$0011.00	*****
Y	200000	S. P. RUSSELL	GLASGOW, SCOTLAND	63738290	26 11 81	\$0020.00	*****
Y	222222	DR E. GRIFFITHS	FRANKFURT, GERMANY	20034151	26 11 81	\$0022.00	*****
Y	300000	V. J. HARRIS	NEW YORK, U.S.	64739801	26 11 81	\$0030.00	*****
Y	333333	J.D. HENRY	CARDIFF, WALES	78493020	26 11 81	\$0033.00	*****
Y	400000	C. HUNT	MILAN, ITALY	25363738	26 11 81	\$0040.00	*****
Y	444444	D. JACOBS	CALGARY, ALBERTA	77889820	26 11 81	\$0044.00	*****
Y	500000	P. KINGSLEY	MADRID, SPAIN	44454640	26 11 81	\$0000.00	*****
Y	555555	S.J. LAZENBY	KINGSTON, N.Y.	39944420	26 11 81	\$0005.00	*****
Y	600000	M.F. MASON	DUBLIN, IRELAND	12398780	26 11 81	\$0010.00	*****
Y	666666	R. F. WALLER	LA HULPE, BRUSSELS	42983840	26 11 81	\$0016.00	*****
Y	700000	M. BRANDON	DALLAS, TEXAS	57984320	26 11 81	\$0002.00	*****
Y	777777	L.A. FARMER	WILLIAMSBURG, VIRG.	91876131	26 11 81	\$0027.00	*****
Y	800000	P. LUPTON	WESTEND, LONDON	24233389	26 11 81	\$0030.00	*****
Y	888888	P. MUNDY	NORTHAMPTON, ENG.	23691639	26 11 81	\$0038.00	*****
Y	900000	D.S. RENSHAW	TAMPA, FLA.	35668120	26 11 81	\$0040.00	*****
Y	999999	ANJI STEVENS	RALEIGH, N.Y.	84591639	26 11 81	\$0049.00	*****

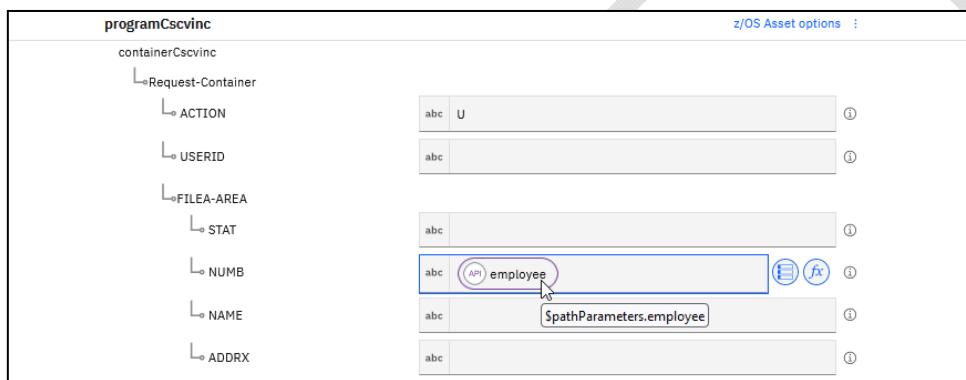
Compete the configuration of the API (Optional)

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*.

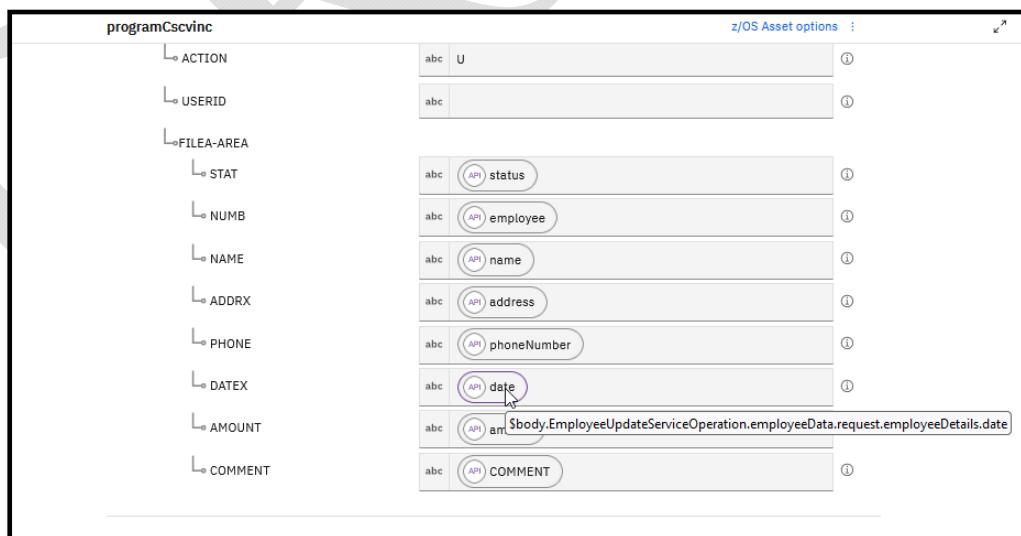
Configure the PUT method for URI path /employee/{employee}

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 */employee/{employee}*

- ___ 1. Start by adding the existing z/OS Asset for *programCscvinc* to this method.
- ___ 2. Now map the API path parameter field *employee* to the CICS container field *Numb* as shown below. Next set the ACTION container field to **U**.



- ___ 3. Map the API request message body fields *status*, *name*, *address*, *phoneNumber*, *date*, *amount*, and *COMMENT* to the CICS container fields *STAT*, *NAME*, *ADDRX*, *PHONE*, *DATEX*, *AMOUNT* and *COMMENT* as shown below.



- ___ 4. Maximize the *Responses* area of the browser's page (see below).

The contents of the *CEIBRESP* and *CEIBRESP2* fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was

updated as intended. The application returns the values of the EIBRESP and EIBRESP2 from the EXECI CICS API in the container *CEIBRESP* and *CEIBRESP2* fields. Their values will be zero if a record was successfully updated in the VSAM data set. Otherwise, the record did not exist, e.g., CEIBRESP=13 and CEIBRESP2=80 or some other error occurred will be returned.

The screenshot shows the configuration of response rules for a PUT endpoint at `/employee/{employee}`. The interface includes sections for Responses, Rule combination, Response field, Predicate, and Value. The responses are evaluated from top to bottom, with the final response being the default response.

- 200 - OK:** If .. equals .. then send 200 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input. Add condition +
- 404 - Not Found:** If .. equals .. then send 404 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.

- ___ 5. Under the **200 – OK** response, use your preferred mapping technique to check the **CEIBRESP** and **CEIBRESP2** response fields are equal to **0**.

200 - OK
If channel.containerCscvinc."Response-Container".CEIBRESP equals 0 and
channel.containerCscvinc."Response-Container".CEIBRESP2 equals 0 then send 200 response

Rule combination
All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	0
CEIBRESP2	Equals	0

Add condition +

- ___ 6. For the **404 – Not Found** check, use your preferred mapping technique to check the **CEIBRESP** container response field is equal to a value of **13** and the **CEIBRESP2** container response field is equal to a value of **80**.

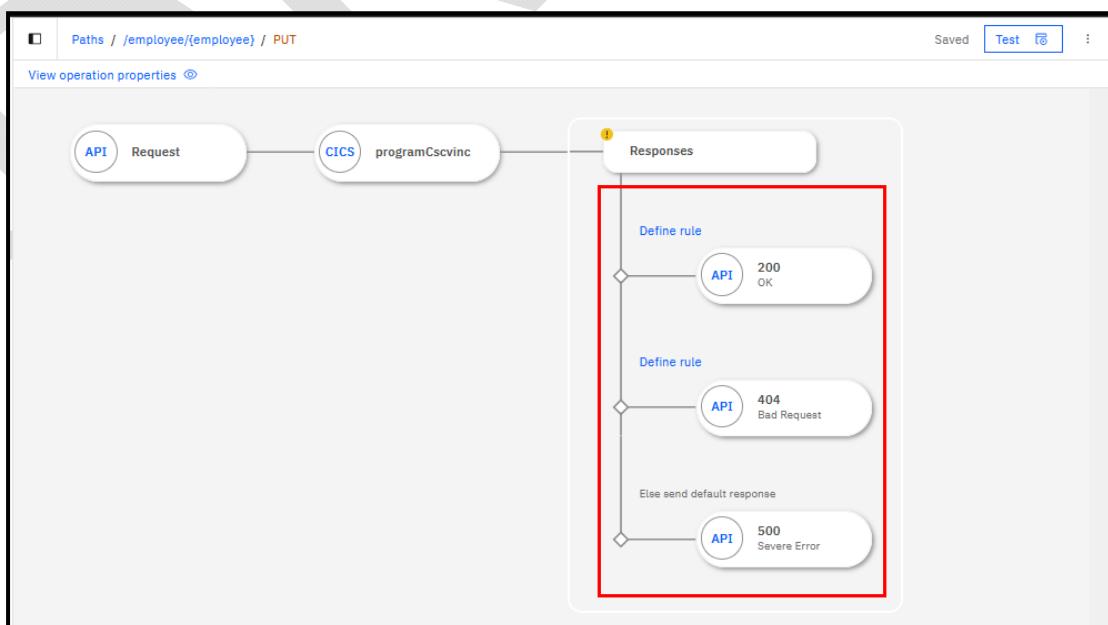
404 - Not Found
If channel.containerCscvinc."Response-Container".CEIBRESP equals 13 and
channel.containerCscvinc."Response-Container".CEIBRESP2 equals 80 then send 404 response

Rule combination
All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	13
CEIBRESP2	Equals	80

Add condition +

- ___ 7. If none of these connections are met, simply return with a HTTP 500 status code.
___ 8. Next the API response messages need to be configured for each of these potential status codes.



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

```
Record with key {{zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} was successfully updated - CICS identity
{{zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
```

200 - OK

[Edit mapping](#) [View structure](#)

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

Body

message

abc Record with key (API) NUMB was successfully updated - CICS identity (API) USERID

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

```
Record with key
{{$apiRequest.body.EmployeeUpdateServiceOperation.employeeData.request.employeeDetails.employeeNumber}} was not found - CICS Identity
{{zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
```

404 - Bad Request

[Edit mapping](#) [View structure](#)

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

Body

message

abc Record with key (API) employeeNumber was not found - CICS Identity (API) USERID

Notice that the mapping for the employeeNumber in the message was from the API request message and not the CICS response container.

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

```
A severe error occurred while inserting a record with key
{{zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} -
CICS Identity {{zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
{{zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP}}:
{{zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2}}
```

The screenshot shows the 'Edit mapping' screen for a 500 - Severe Error response. The 'Body' section contains a 'message' field with the following JSON path expression:

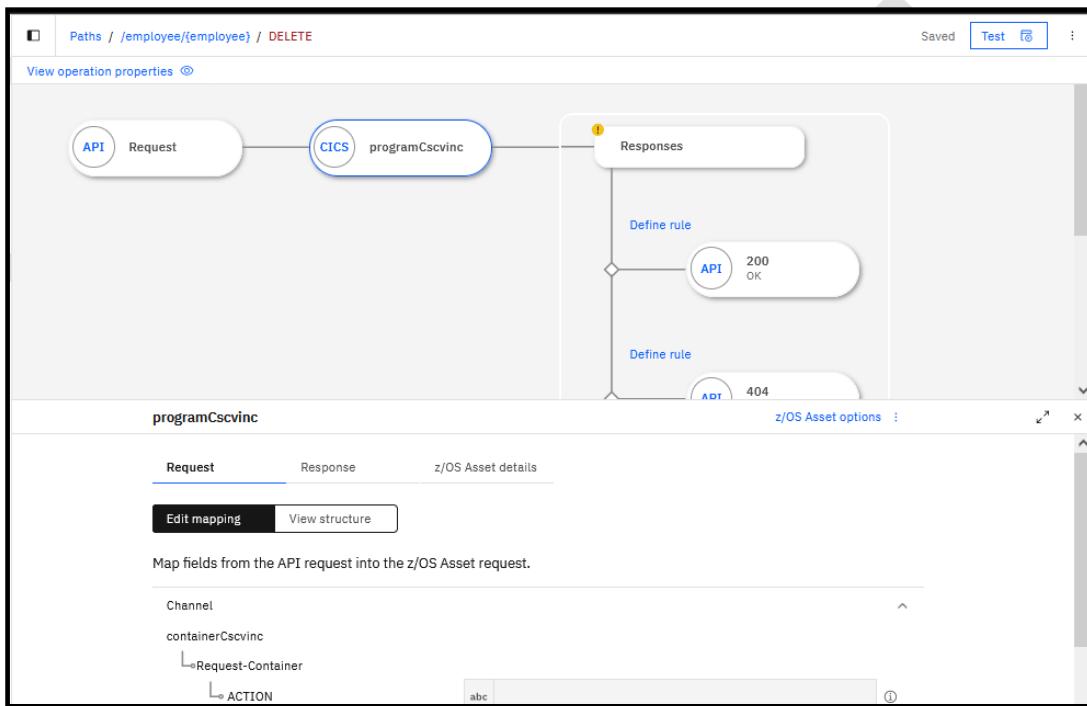
```
abc A severe error occurred while inserting a record with key $NUMB - CICS Identity $USERID : $CEIBRESP $CEIBRESP2
```

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.

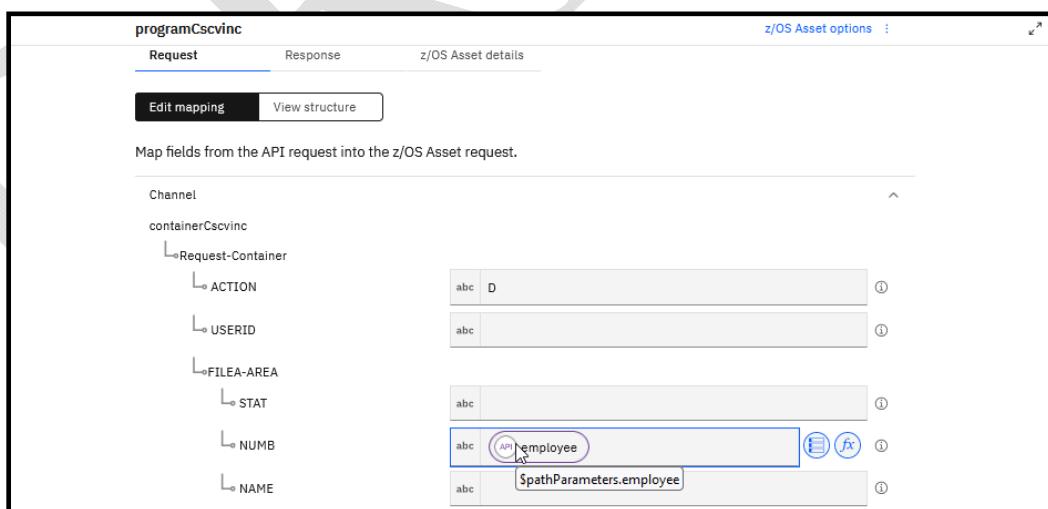
Configure the **DELETE** method for URI path **/employee/{employee}**

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

1. Start by adding the existing *programCscvinc* z/OS Asset to this method.



2. Now map the API path parameter field *employee* to the CICS container field *Numb* as shown below and enter a **D** in the *ACTION* field of the request container.



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

The contents of the *CEIBRESP* and *CEIBRESP2* fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was deleted as intended. The application returns the values of the EIBRESP and EIBRESP2 from the EXECI CICS API in the container *CEIBRESP* and *CEIBRESP2* fields. Their values will be zero if a record was successfully deleted from the VSAM data set. Otherwise, the record did not exist, e.g., *CEIBRESP*=13 and *CEIBRESP2*=80 or some other error occurred will be returned.

The screenshot shows the 'Responses' configuration for a DELETE path. The interface is divided into three main sections: 200 - OK, 404 - Not Found, and 500 - Internal Server Error.

- 200 - OK:** If .. equals .. then send 200 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input.
- 404 - Not Found:** If .. equals .. then send 404 response. Rule combination: All the following are true. Response field: Input, Predicate: Equals, Value: Input.
- 500 - Internal Server Error:** Else send default response.

- ___ 4. Under the *200 – OK* response, use your preferred mapping technique to check the ***CEIBRESP*** and ***CEIBRESP2*** response fields are equal to ***0***.

200 - OK
If channel.containerCscvinc."Response-Container".CEIBRESP equals 0 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 0 then send 200 response

Rule combination
All the following are true

Response field	Predicate	Value
(o) CEIBRESP	Equals	0
(o) CEIBRESP2	Equals	0

Add condition +

- ___ 5. For the *404 – Not Found* check, use your preferred mapping technique to check the ***CEIBRESP*** container response field is equal to a value of ***13*** and the ***CEIBRESP2*** container response field is equal to a value of ***80***.

404 - Not Found
If channel.containerCscvinc."Response-Container".CEIBRESP equals 13 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 80 then send 404 response

Rule combination
All the following are true

Response field	Predicate	Value
(o) CEIBRESP	Equals	13
(o) CEIBRESP2	Equals	80

Add condition +

- ___ 6. If none of these connections are met, simply return with a HTTP 500 status code.

- ___ 7. Next the API response messages need to be configured for each of these potential status codes.

Paths / employee/{employee} / DELETE
View operation properties

Request → CICS programCscvinc → Responses

Responses (highlighted by a red box):

- Define rule: API 200 OK
- Define rule: API 404 Not Found
- Else send default response: API 500 Not Found

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

```
Record with key {{\$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} was successfully deleted - CICS identity
{{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
```

The screenshot shows the 'Edit mapping' screen for a 200 OK response. At the top, there are two buttons: 'Edit mapping' (which is highlighted in black) and 'View structure'. Below these buttons, a message states: 'Map fields from the z/OS Asset response into the API response.' Under the 'Body' section, there is a 'message' field. To the right of the 'message' field, a preview box displays the mapped JSON: 'abc Record with key (API) NUMB was successfully deleted - CICS identity (API) USERID'. There are three small icons in the top right corner of the preview box: a blue square with a white circle, a blue square with a white checkmark, and a blue square with a white question mark.

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

```
Record with key {{$apiRequest.pathParameters.employee}} was not found - CICS Identity
{{$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
```

The screenshot shows the 'Edit mapping' screen for a 404 Not Found response. At the top, there are two buttons: 'Edit mapping' (highlighted in black) and 'View structure'. Below these buttons, a message states: 'Map fields from the z/OS Asset response into the API response.' Under the 'Body' section, there is a 'message' field. To the right of the 'message' field, a preview box displays the mapped JSON: 'abc Record with key (API) employee was not found - CICS Identity (API) USERID'. There is one small icon in the top right corner of the preview box: a blue square with a white question mark.

Notice that the mapping for the employeeNumber in the message was from the API request message and not the CICS response container.

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

```
A severe error occurred while deleting a record with key
{{zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} -
CICS Identity {{zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
{{zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP}}:
{{zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2}}
```

The screenshot shows the 'Edit mapping' interface for a 500 - Not Found response. Under the 'Body' section, there is a 'message' field containing the following JSON template:

```
abc A severe error occurred while deleting a record with key
  {{osAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} -
  CICS Identity {{osAssetResponse.channel.containerCscvinc."Response-Container".USERID}}
  {{osAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP}}:
  {{osAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2}}
```

Each variable in the template has an exclamation mark next to it, indicating a warning or error in the mapping configuration.

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.

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:9447>. 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:9447/employee/{employee}>

The screenshot shows a web browser window with the title "z/OS Connect Designer" and the tab "OpenAPI UI". The address bar shows the URL <https://designer.ibm.com:9447/api/explorer/>. The main content area is titled "Liberty REST APIs 1.0.0 OAS3". Below it, a sub-section titled "Employee" lists four methods:

- POST** /employee
- GET** /employee/{employee}
- PUT** /employee/{employee}
- DELETE** /employee/{employee}

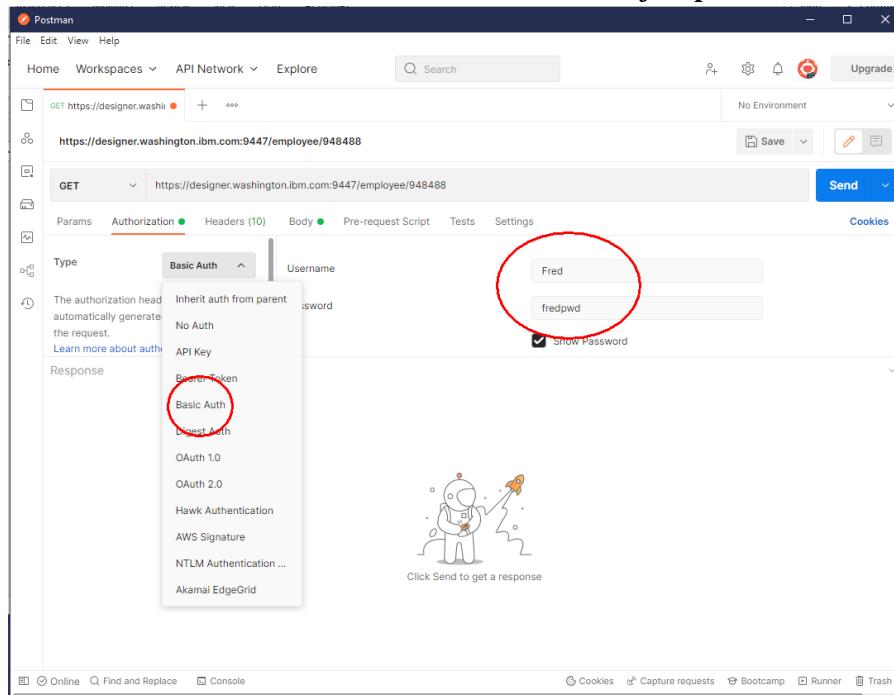
From this display, the methods and URLs required to access the API deployed in this container are:

- POST <https://designer.washington.ibm.com:9447/employee>
- GET <https://designer.washington.ibm.com:9447/employee/{employee}>
- PUT <https://designer.washington.ibm.com:9447/employee/{employee}>
- DELETE <https://designer.washington.ibm.com:9447/employee/{employee}>

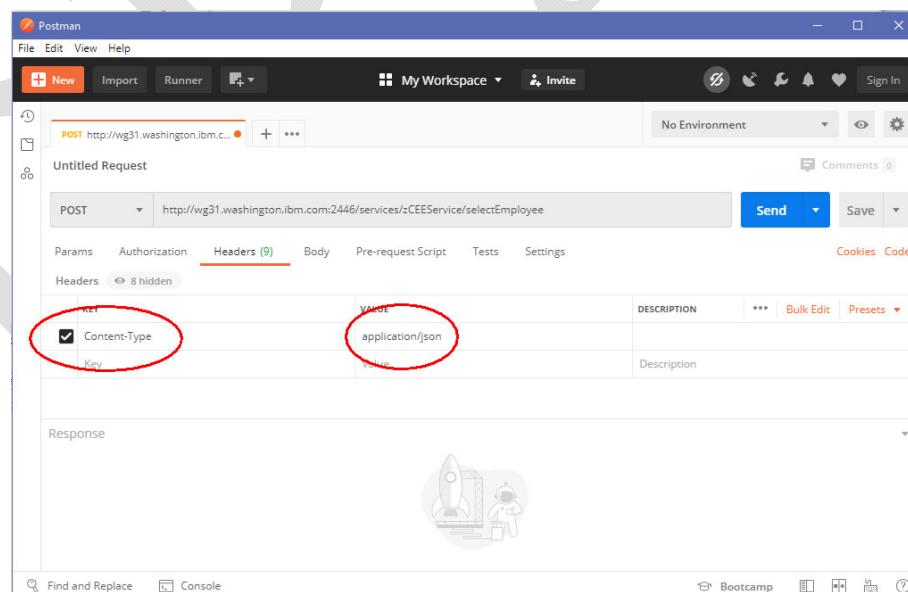
Using Postman

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. Select the *Authorization* tab to enter an authorization identity and password. Use the pull down arrow to select *Basic Auth* and enter **Fred** as the *Username* and **fredpwd** as the *Password*.

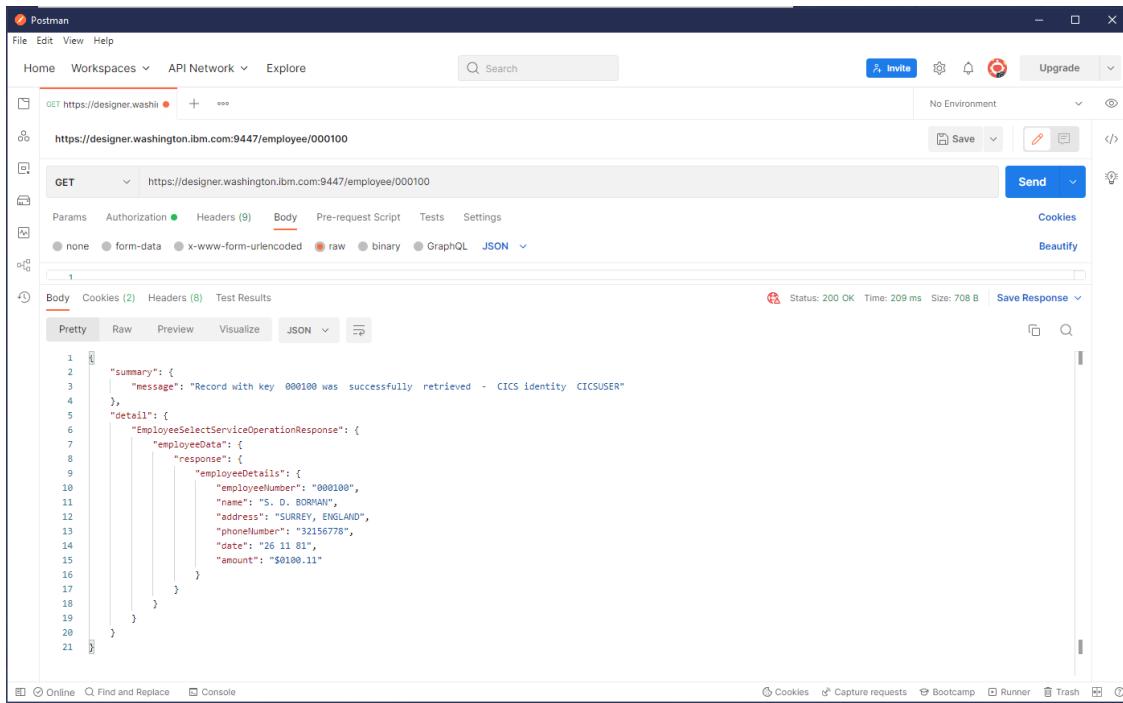


2. Next select the *Headers* tab and under *KEY* use the code assist feature to enter **Content-Type** and under *VALUE* use the code assist feature to enter **application/json**.



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.

3. Then go to the *Body* tab and use the down arrow to select **GET** and enter <https://designer.washington.ibm.com:9447/employee/000100> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.



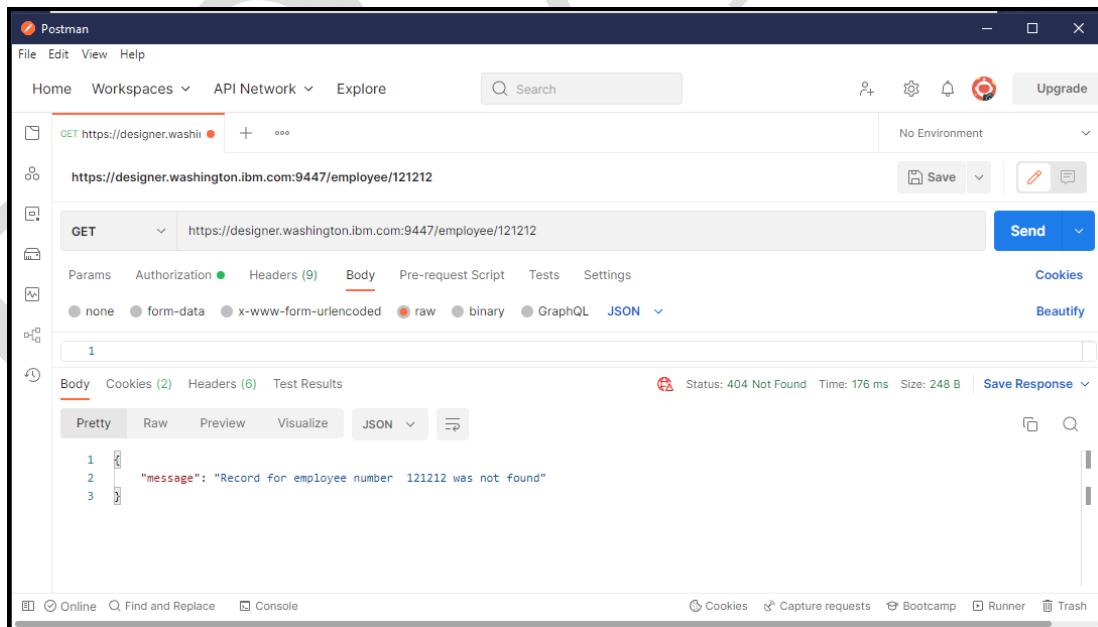
```

1
2   "summary": {
3     "message": "Record with key 000100 was successfully retrieved - CICS identity CICSUSER"
4   },
5   "detail": [
6     "EmployeeSelectServiceOperationResponse": {
7       "employeeData": {
8         "response": {
9           "employeeDetails": {
10             "employeeNumber": "000100",
11             "name": "S. D. BORNAN",
12             "address": "SURREY, ENGLAND",
13             "phoneNumber": "32156787",
14             "date": "26 11 81",
15             "amount": "$0100.11"
16           }
17         }
18       }
19     }
20   }
21 }
```

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

<https://designer.washington.ibm.com:9447/employee/121212>

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



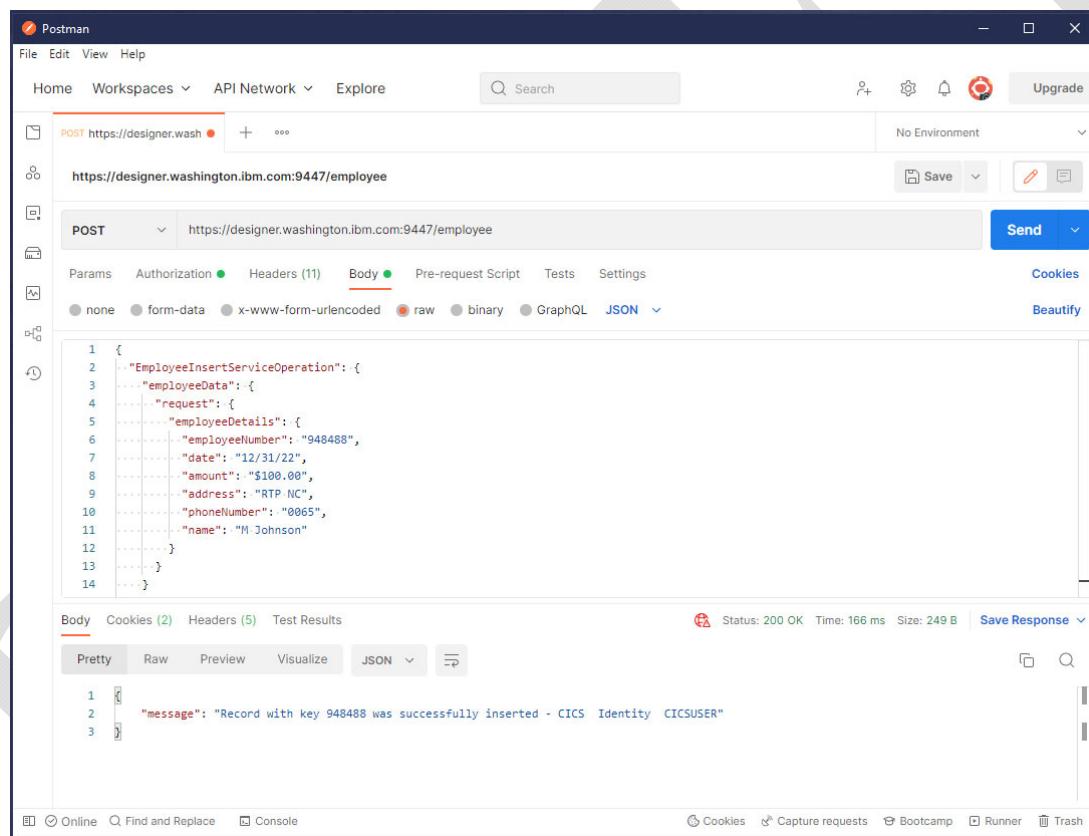
```

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

5. Next use *Postman* to invoke other methods of the API. For example, if you want to invoke a *POST* method with URL <https://designer.washington.ibm.com:9447/employee> and the JSON below for the request message.

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948488",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

Pressing the **Send** button invokes the POST method of URI path */employee*.



6. Invoke the *GET* method with URL <https://designer.washington.ibm.com:9447/employee/948488> to display the record just inserted.

```

1 {
2   "summary": {
3     "message": "Record with key 948488 was successfully retrieved - CICS identity CICSUSER"
4   },
5   "detail": {
6     "EmployeeSelectServiceOperationResponse": {
7       "employeeData": {
8         "response": {
9           "employeeDetails": {
10             "employeeNumber": "948488",
11             "name": "M Johnson",
12             "address": "RTP NC",
13             "phoneNumber": "0065",
14             "date": "12/31/22",
15             "amount": "$100.00",
16             "comment": "22-07-17"
17           }
18         }
19       }
20     }
21   }
22 }
```

Status: 200 OK Time: 174 ms Size: 514 B | Save Response

7. Invoke a *PUT* method with URL <https://designer.washington.ibm.com:9447/employee/948488> and the JSON below for the request message to update the *salary*, *bonus*, and *commission* columns of this row.

```
{
  "EmployeeUpdateServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "status": "s",
          "name": "A Johnson",
          "address": "Apex NC",
          "phoneNumber": "0065",
          "date": "01/31/23",
          "amount": "500",
          "COMMENT": "updated"
        }
      }
    }
  }
}
```

IBM z/OS Connect (OpenAPI 3.0)

The screenshot shows the Postman application interface. A PUT request is made to <https://designer.washington.ibm.com:9447/employee/948488>. The request body contains JSON data representing an employee update. The response status is 200 OK, and the message indicates the record was successfully updated.

```
PUT https://designer.washington.ibm.com:9447/employee/948488
{
    "EmployeeUpdateServiceOperation": {
        "employeeData": {
            "request": {
                "employeeDetails": {
                    "status": "s",
                    "name": "A Johnson",
                    "address": "Apex NC",
                    "phoneNumber": "0065",
                    "date": "01/31/23",
                    ...
                }
            }
        }
    }
}

{
    "message": "Record with key 948488 was successfully updated - CICS identity CICSUSER"
}
```

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

<https://designer.washington.ibm.com:9447/employee/948488>

The screenshot shows the Postman application interface. A GET request is made to <https://designer.washington.ibm.com:9447/employee/948488>. The response status is 200 OK, and the JSON response body includes a summary message and detailed employee data.

```
GET https://designer.washington.ibm.com:9447/employee/948488
{
    "summary": {
        "message": "Record with key 948488 was successfully retrieved - CICS identity CICSUSER"
    },
    "detail": {
        "EmployeeSelectServiceOperationResponse": {
            "employeeData": {
                "response": {
                    "employeeDetails": {
                        "employeeNumber": "948488",
                        "name": "A Johnson",
                        "address": "Apex NC",
                        "phoneNumber": "0065",
                        "date": "01/31/23",
                        "amount": "500"
                    }
                }
            }
        }
    }
}
```

9. 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.

10. To clear the *Postman* cached security 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.

```

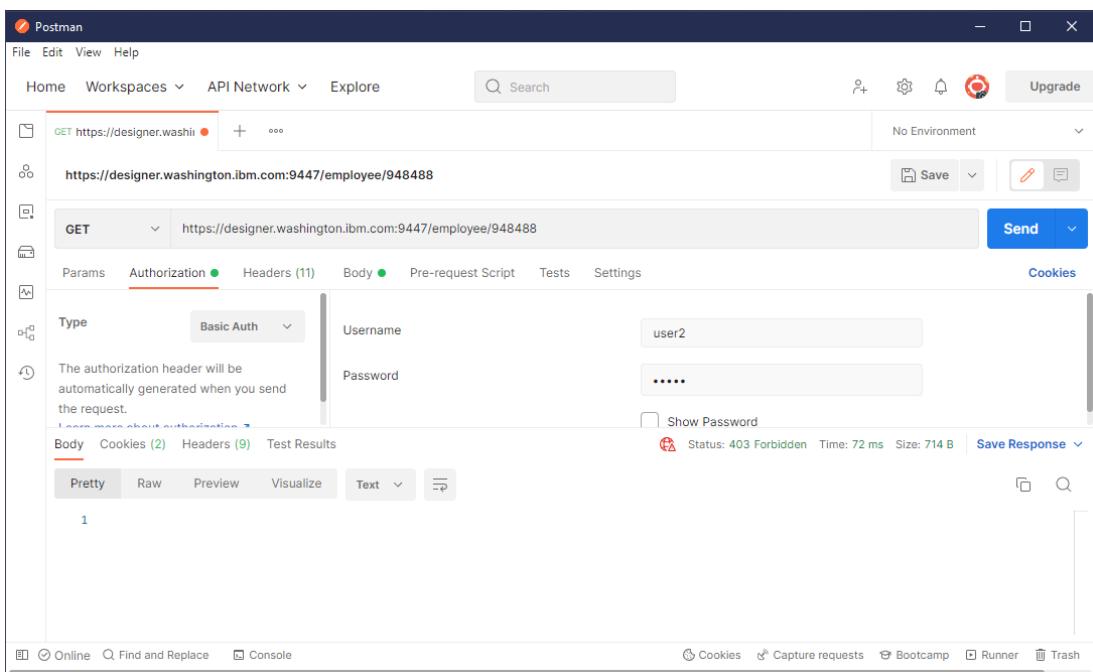
1
2   "results": [
3     {
4       "employeeNumber": "948478",
5       "name": "Matt T Johnson",
6       "departmentCode": "C00",
7       "phoneNumber": "0065",
8       "job": "Staff"
  
```

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

The screenshot shows the 'Cookies' section of Postman. It lists two cookies for the domain `designer.ibm.com`: *JSESSIONID* and *LtpaToken2*. Both cookie entries have their close (X) icons highlighted.

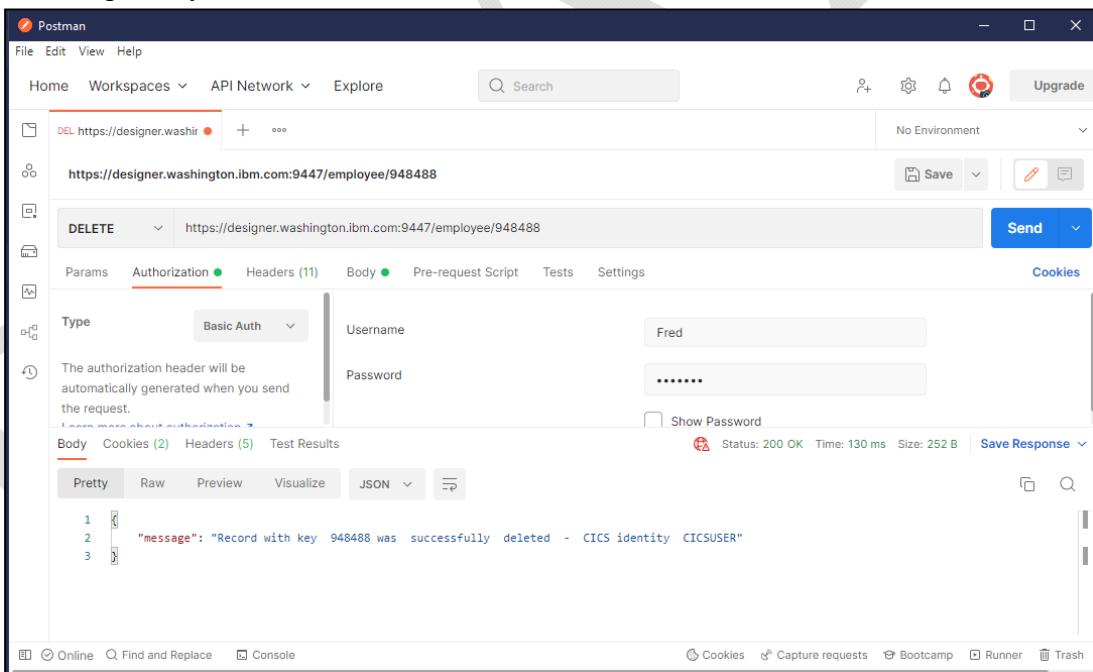
IBM z/OS Connect (OpenAPI 3.0)

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).



The screenshot shows the Postman application interface. A GET request is being made to `https://designer.washington.ibm.com:9447/employee/948488`. The Authorization tab is selected, showing 'Basic Auth' selected. The 'Username' field contains 'user2' and the 'Password' field contains '*****'. The response status is 403 Forbidden.

While a delete request by Fred is successful.



The screenshot shows the Postman application interface. A DELETE request is being made to `https://designer.washington.ibm.com:9447/employee/948488`. The Authorization tab is selected, showing 'Basic Auth' selected. The 'Username' field contains 'Fred' and the 'Password' field contains '*****'. The response status is 200 OK, and the response body is: "message": "Record with key 948488 was successfully deleted - CICS identity CICSUSER".

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:9447/employee/000100
```

```
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:9447/employee/000100
{"summary":{"message":"Record with key 000100 was successfully retrieved - CICS identity CICSUSER"},"detail":{"EmployeeSelectServiceOperationResponse":{"employeeData":{"response":{"employeeDetails":{"employeeNumber":"000100","name":"S. D. BORMAN","address":"SURREY, ENGLAND","phoneNumber":"32156778","date":"26 11 81","amount":"$0100.11"}}}}} - 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:9447/employee/000100
```

```
c:\z\openApi3>curl -X GET -w " - HTTP CODE ${http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://designer.washington.ibm.com:9447/employee/000100
- HTTP CODE 403
```

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.

```
com.ibm.ws.security.credentials.vaultedIdProviderImpl[id=09614a49,realmName=zosConnect,securityName=user2,realmSecurityName=zosConnect/user2,uniqueSecurityName=user2,primaryGroupID=null,accessIdUser:zosConnect/user2,groupIds=[]]
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 > authorization.builtin.BuiltinAuthorizationService > getGroups Exit
com.ibm.ws.security.credentials.vaultedIdProviderImpl[id=09614a49,realmName=zosConnect,securityName=user2,realmSecurityName=zosConnect/users,uniqueSecurityName=user2,primaryGroupID=null,accessIdUser:zosConnect/user2,groupIds=[]]
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.credentials.vaultedIdProviderImpl < getGroups Exit
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.credentials.vaultedIdProviderImpl < getGroups Exit
[]
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 > authorization.builtin.BuiltinAuthorizationService > getGroups Exit
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 > authorization.builtin.BuiltinAuthorizationService > getGroupName Entry
com.ibm.ws.security.credentials.vaultedIdProviderImpl[id=09614a49,realmName=zosConnect,securityName=user2,realmSecurityName=zosConnect/user2,uniqueSecurityName=user2,primaryGroupID=null,accessIdUser:zosConnect/user2,groupIds=[]]
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.credentials.vaultedIdProviderImpl < getGroupName Exit
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.credentials.vaultedIdProviderImpl < disconnect
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 > authorization.builtin.BuiltinAuthorizationService > isAuthorizedForAccessDecision Exit
false
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 > authorization.builtin.BuiltinAuthorizationService > isSubjectAuthorized Exit
false
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 > authorization.builtin.BuiltinAuthorizationService > isSubjectAuthorized [Subject:
Principal: idPrincipal:user2
Public Credential:
com.ibm.ws.security.credentials.vaultedIdProviderImpl[id=09614a49,realmName=zosConnect,securityName=user2,realmSecurityName=zosConnect/user2,uniqueSecurityName=user2,primaryGroupID=null,accessIdUser:zosConnect/user2,groupIds=[]]
Private Credential: com.ibm.ws.security.token.internal.SingleSignOnToken@9728ef
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 > authorization.builtin.BuiltinAuthorizationService > isUnauthorized Exit
false
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.authorization.util.AuthorizedAuthUtil < isAuthentication ENTRY WSPrincipalUser
[7/16/22 17:24:51+138 UTC] 0000020d id=715f7e74 com.ibm.ws.security.authentication.principals.WSPrincipalUser < getname Entry
[7/16/22 17:24:51+138 UTC] 0000020d id=715f7e74 com.ibm.ws.security.authentication.principals.WSPrincipalUser < getname Exit
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.authorization.util.AuthorizedAuthUtil < isAuthentication RETURN false
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.authorization.util.AuthorizedAuthUtil < getCallerPrincipal Entry
false
null
true
false
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.context.SubjectManager < getCallerSubject Entry
[7/16/22 17:24:51+138 UTC] 0000020d id=09614a49 com.ibm.ws.security.context.internal.SubjectThreadContent < getCallerSubject Exit
[7/16/22 17:24:51+138 UTC] 0000020d id=229ff518 com.ibm.ws.security.context.internal.SubjectThreadContent < getCallerSubject Exit
]
```

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 @insertCscvinc.json https://designer.washington.ibm.com:9447/employee/
```

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

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948499",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

```
c:\z\openApi3>curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @insertCscvinc.json https://designer.washington.ibm.com:9447/employee/
{"message": "Record with key 948499 was successfully inserted - CICS Identity CICSUSER"} - HTTP CODE 200 {"message": "Record for employee number 948489 was added successfully"} - HTTP CODE 200
```

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

```
curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9447/employee/948499
```

```
c:\z\openApi3>curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9447/employee/948478
{"summary": {"message": "Record with key 948478 was successfully retrieved - CICS identity CICSUSER"}, "detail": {"EmployeeSelectServiceOperationResponse": {"employeeData": {"response": {"employeeDetails": {"employeeNumber": "948478", "name": "name", "address": "RTP NC", "phoneNumber": "0065", "date": "12/31/22", "amount": "$100.00"} }}}}} - HTTP CODE 200
```

Now try using user2's credentials and see that it fails with an HTTP 403.

```
curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user user2:user2 https://designer.washington.ibm.com:9447/employee/948499
```

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

```
curl -X PUT -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user Fred:fredpwd --data @updateCscvinc.json
https://designer.washington.ibm.com:9447/employee/948499
```

```
c:\z\openApi3>curl -X PUT -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --insecure
--user Fred:fredpwd --data @updateCscvinc.json https://designer.washington.ibm.com:9447/employee/948499
{"message": "Record with key 948478 was successfully updated - CICS identity CICSUSER"} - HTTP CODE 200
```

The file *updateCscvinc.json* contains the contents below:

```
{
  "EmployeeUpdateServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "status": "s",
          "name": "A Johnson",
          "address": "Apex NC",
          "phoneNumber": "0065",
          "date": "01/31/23",
          "amount": "500",
          "COMMENT": "updated"
        }
      }
    }
  }
}
```

7. Confirm the updates were made by using a *GET* method to retrieve this record.

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

```
curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user Fred:fredpwd https://designer.washington.ibm.com:9447/employee/948499
```

```
c:\z\openApi3>curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user Fred:fredpwd https://designer.washington.ibm.com:9447/employee/948478
{"message": "Record with key 948499 was successfully deleted - CICS identity CICSUSER"} - HTTP CODE
200
```

Repeat the command again and observe the results

```
c:\z\openApi3>curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user Fred:fredpwd https://designer.washington.ibm.com:9447/employee/948478
{"message": "Record with key 948499 was not found - CICS Identity CICSUSER"} - HTTP CODE 404
```

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

```
curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user user1:user1 https://designer.washington.ibm.com:9447/employee/948499
```

```
c:\z\openApi3>curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user user1:user1 https://designer.washington.ibm.com:9447/employee/948499
- HTTP CODE 403
```

The HTTP 403 is cause by user1 not having DELETE authority.

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 */employee* and the *GET* for */employee/{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:9447/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 **fredpwd** (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 `DELETE /employee/{employee}` URI path to display the request parameters.

`DELETE /employee/{employee}`

Parameters

Name	Description
Authorization string (header)	Authorization
employee * required string (path)	employee

[Try it out](#)

3. Press the Try it out button.

4. Enter a value of **948488** and press the Execute button.

`DELETE /employee/{employee}`

Parameters

Name	Description
Authorization string (header)	Authorization
employee * required string (path)	948488

Servers

These path-level options override the global server options.

[Cancel](#) [Execute](#)

5. Enter a value of **948488** and press the Execute button .

6. If the record exist, a message like the one below will appear.

Code	Details
200	<p>Response body</p> <pre>{ "message": "Record with key 948488 was successfully deleted - CICS identity CICSUSER" }</pre> <p>Copy Download</p> <p>Response headers</p> <pre> content-language: en-US content-length: 92 content-type: application/json date: Sun, 17 Jul 2022 18:45:30 GMT x-firefox-spdy: h2 x-powered-by: Servlet/4.0 </pre>

7. Otherwise, a message that the record was not found will appear.

Code	Details
404	<p>Error: Not Found</p> <p>Response body</p> <pre>{ "message": "Record with key 948488 was not found - CICS Identity CICSUSER" }</pre> <p>Copy Download</p> <p>Response headers</p> <pre> content-language: en-US content-length: 77 content-type: application/json date: Sun, 17 Jul 2022 18:47:16 GMT x-firefox-spdy: h2 x-powered-by: Servlet/4.0 </pre>

8. Click on *POST /employee* URI path to display the request body view of the URI path.

The screenshot shows the Open Liberty REST APIs UI. The top navigation bar has tabs for 'z/OS Connect Designer' and 'OpenAPI UI'. The main title is 'Open Liberty' with 'Liberty REST APIs 1.0.0 OAS3'. Below it says 'Discover REST APIs available within Liberty'. A dropdown 'Servers' shows 'https://designer.ibm.com:9447'. The main content area is titled 'Employee' and shows a 'POST /employee' endpoint. It includes sections for 'Parameters' (with an 'Authorization' header), 'Request body (required)' (set to 'application/json'), and 'Responses' (200 OK). The 'Request body' section contains a JSON schema:

```
{
  "EmployeeInsertServiceOperation": {
    "request": {
      "employeeDetails": {
        "employeeNumber": "string",
        "name": "string",
        "middleName": "string",
        "phoneNumber": "string",
        "date": "string",
        "amount": "string"
      }
    }
  }
}
```

9. Next press the *Try it out* button to enable the entry of a request message body

This screenshot is similar to the previous one, but the 'Try it out' button in the top right corner of the 'Request body' panel is now highlighted in red, indicating it is active. The rest of the interface remains the same, showing the 'Employee' endpoint details and the JSON schema for the request body.

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

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948488",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

11. Use a Click on *GET* for URI path `/employee/{employee}` to display the GET method parameters. Press the **Try it out** button Enter **948488** for the employee and press the **Execute** button to display this record.

The screenshot shows the API Designer interface with the following details:

- Code** tab selected.
- Details** tab visible.
- HTTP Method**: GET
- URI Path**: /employee/{employee}
- Path Parameters**: {employee} = 948488
- HTTP Response**:
 - Status**: 200
 - Response Body** (JSON):


```
{
            "summary": {
              "message": "Record with key 948488 was successfully retrieved - CICS identity CICSUSER"
            },
            "detail": {
              "EmployeeSelectServiceOperationResponse": {
                "employeeData": {
                  "response": {
                    "employeeDetails": {
                      "employeeNumber": "948488",
                      "name": "M Johnson",
                      "address": "RTP NC",
                      "phoneNumber": "0065",
                      "date": "12/31/22",
                      "amount": "$100.00",
                      "comment": "22-07-17"
                    }
                  }
                }
              }
            }
          }
```
 - Actions**: Download

Security was enabled in the original specification document, so you will be required to sign in with a RACF identity. Use **Fred** for the *Username* and **fred** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

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

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

14. Enter **948488** in the area beside *employee* and enter the JSON below in the request body area and press the **Execute** button.

```
t
"EmployeeUpdateServiceOperation": {
  "employeeData": {
    "request": {
      "employeeDetails": {
        "status": "s",
        "name": "A Johnson",
        "address": "Apex NC",
        "phoneNumber": "0065",
        "date": "01/31/23",
        "amount": "500",
        "COMMENT": "updated"
      } } } }
```

Curl

```
curl -X 'PUT' \
'https://designer.washington.ibm.com:9447/employee/948488' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"EmployeeUpdateServiceOperation": {
  "employeeData": {
    "request": {
      "employeeDetails": {
        "status": "s",
        "name": "A Johnson",
        "address": "Apex NC",
        "phoneNumber": "0065",
        "date": "01/31/23",
        "amount": "500",
        "COMMENT": "updated"
      } } }
};'
```

Request URL

<https://designer.washington.ibm.com:9447/employee/948488>

Server response

Code	Details
200	<p>Response body</p> <pre>{ "message": "Record with key 948488 was successfully updated - CICS identity CICSUSER"</pre> <p>Copy Download</p> <p>Response headers</p> <pre>content-language: en-US content-length: 92 content-type: application/json date: Sun,17 Jul 2022 19:05:33 GMT x-firefox-spdy: h2 x-powered-by: Servlet/4.0</pre>

15. Use the *GET* method to confirm the updates have taken place.

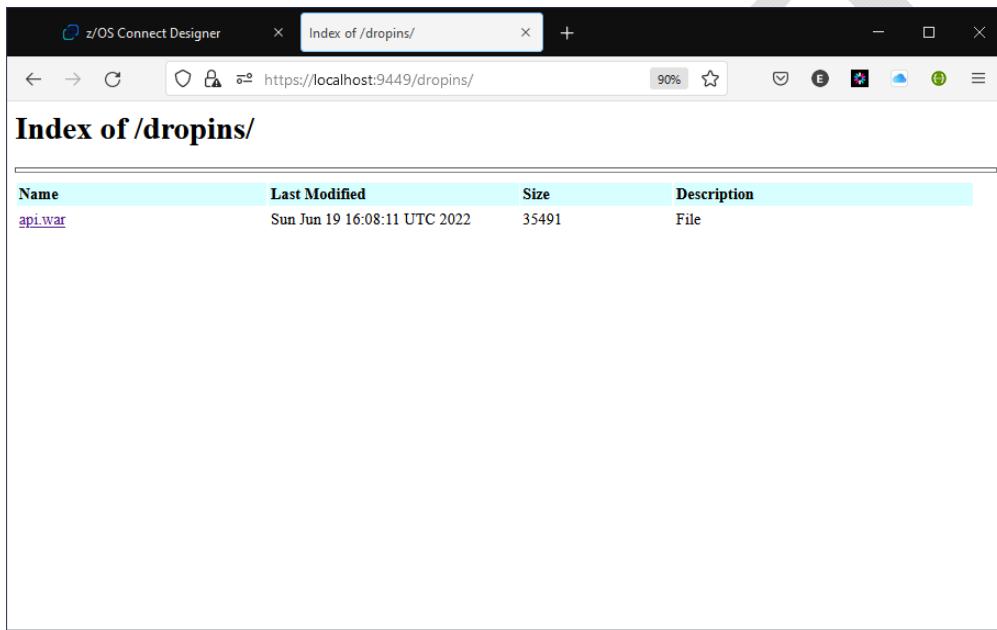
16. Use the *DELETE* method to delete employee **948488** and confirm the record has been deleted.

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:9447/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 *cscvinc.war*.

2. Open a DOS command prompt and use the change directory command to go to directory *C:\z\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 cscvinc.war
200 Port request OK.
125 Storing data set /var/zcee/openApi3/apps/employee.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="cics" contextRoot="/cics" name="cicsAPI"  
location="${server.config.dir}apps/cscvinc.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 */cics* 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 **cicsApi**. The final source is the role name provided in the specification document, **Manager** or **Staff**. So, two EJBRoles need to be defined, **ATSZDFLT.cicsApi.Manager** and **ATSZDFLT.cicsApi.Staff**.

- ___ 1. Use the RACF RDEFINE command to define EJBROLE **ATSZDFLT.cicsApi.Manager**.

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

- ___ 2. Use the RACF RDEFINE command to define EJBROLE **ATSZDFLT.cicsApi.Staff**.

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

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

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

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

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

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

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

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

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

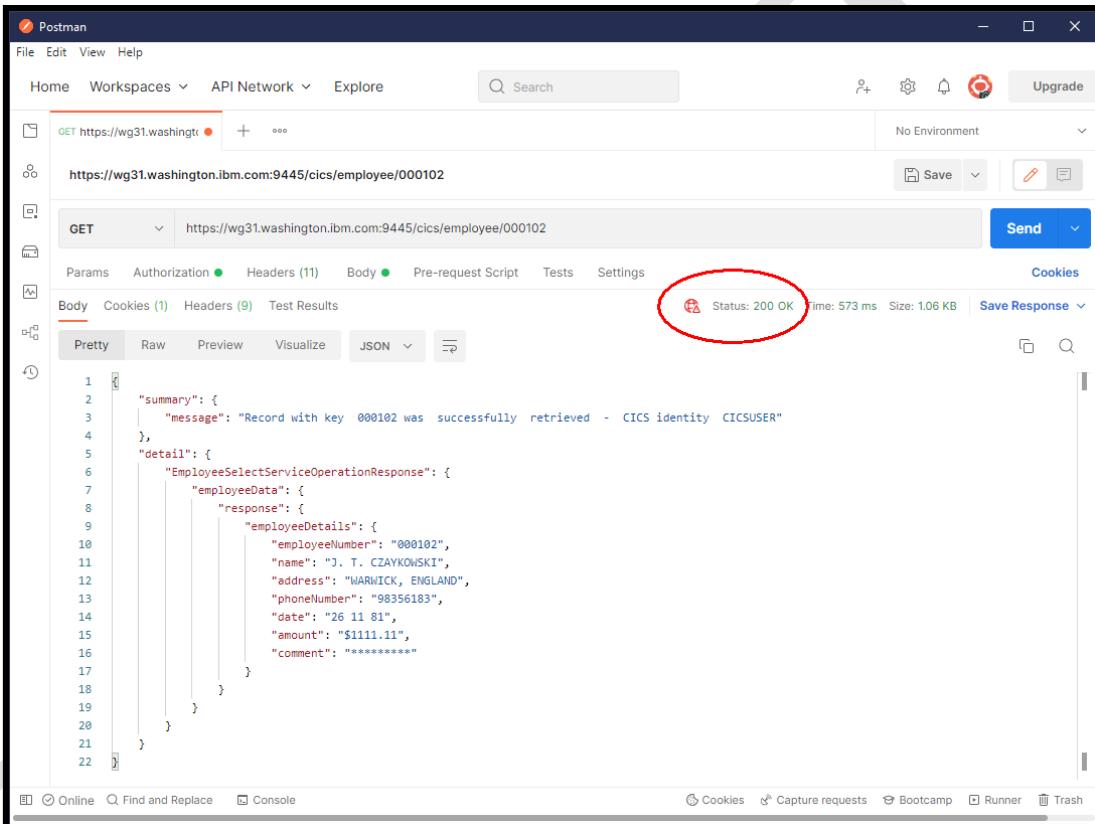
Now we are ready to test the invoking of the methods of this API.

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* 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/cics/employee/000102> 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 red circle highlighting the status bar at the top right, which displays "Status: 200 OK". The main area shows a JSON response with the following content:

```

1  {
2      "summary": {
3          "message": "Record with key 000102 was successfully retrieved - CICS identity CICSUSER"
4      },
5      "detail": {
6          "EmployeeSelectServiceOperationResponse": {
7              "employeeData": {
8                  "response": {
9                      "employeeDetails": {
10                         "employeeNumber": "000102",
11                         "name": "J. T. CZAYKOISKI",
12                         "address": "WARRICK, ENGLAND",
13                         "phoneNumber": "98356183",
14                         "date": "26 11 81",
15                         "amount": "$1111.11",
16                         "comment": "*****"
17                     }
18                 }
19             }
20         }
21     }
22 }
```

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 */cics* 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/cics/employee/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/cics/employee/121212>. The response status is 404 Not Found, with a message: "Record for employee number 121212 was not found".

3. Invoke a *POST* with URI path */employee*, use the JSON below for the request message.

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948498",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

The screenshot shows the Postman interface with a successful POST request to <https://wg31.washington.ibm.com:9445/cics/employee>. The JSON body contains employee details for ID 948498, and the response status is 200 OK.

```

1 {
2   "EmployeeInsertServiceOperation": {
3     "employeeData": {
4       "request": {
5         "employeeDetails": {
6           "employeeNumber": "948498",
7           "date": "12/31/22",
8           "amount": "$100.00",
9           "address": "RTP NC",
10          "phoneNumber": "0065",
11          "name": "M Johnson"
12        }
13      }
14    }
15  }
16}
  
```

Status: 200 OK Time: 569 ms Size: 249 B Save Response

4. Perform a *PUT* with URI path `/cics/employee/{employee}` use this JSON request message.

<https://wg31.washington.ibm.com:9445/cics/employee/948498>

```

{
  "EmployeeUpdateServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "status": "s",
          "name": "A Johnson",
          "address": "Apex NC",
          "phoneNumber": "0065",
          "date": "01/31/23",
          "amount": "500",
          "COMMENT": "updated"
        }
      }
    }
  }
}
  
```

IBM z/OS Connect (OpenAPI 3.0)

The screenshot shows the Postman application interface. A PUT request is made to `https://wg31.washington.ibm.com:9445/cics/employee/948498`. The request body is a JSON object representing an employee update:

```
1 {
2     "EmployeeUpdateServiceOperation": {
3         "employeeData": {
4             "request": {
5                 "employeeDetails": {
6                     "status": "s",
7                     "name": "A Johnson",
8                     "address": "Apex NC",
9                     "phoneNumber": "0065",
10                    "date": "01/31/23",
11                    "amount": "500",
12                    "COMMENT": "updated"
13                }
14            }
15        }
16    }
17 }
```

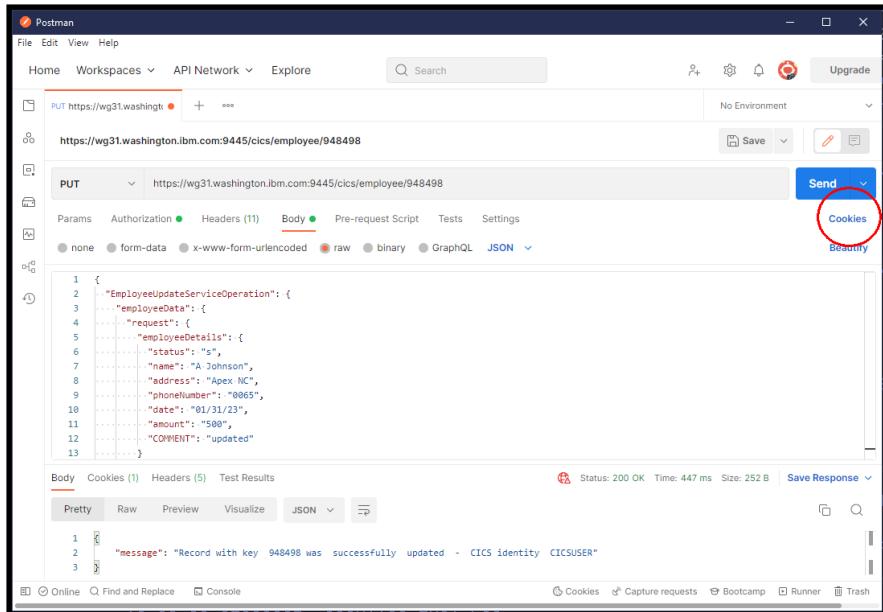
The response status is 200 OK, and the message in the response body is: "Record with key 948498 was successfully updated - CICS identity CICSUSER".

5. Perform a *DELETE* with URI path `/cics/employee/948498`.

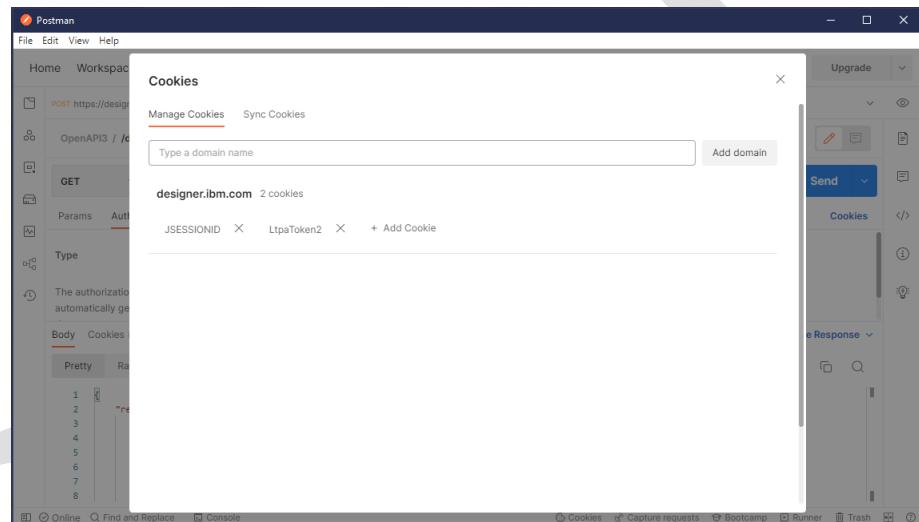
The screenshot shows the Postman application interface. A DELETE request is made to `https://wg31.washington.ibm.com:9445/cics/employee/948498`. The response status is 200 OK, and the message in the response body is: "Record with key 948498 was successfully deleted - CICS identity CICSUSER".

6. 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.

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



8. 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.

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*.

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

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

```
curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fred --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/cics/employee/222222
```

```
c:\z\openApi3>curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fred --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/cics/employee/222222
{"summary": {"message": "Record with key 222222 was successfully retrieved - CICS identity CICSUSER"}, "detail": {"EmployeeSelectServiceOperationResponse": {"employeeData": {"response": {"employeeDetails": {"employeeNumber": "222222", "name": "DR E. GRIFFITHS", "address": "FRANKFURT, GERMANY", "phoneNumber": "20034151", "date": "26 11 81", "amount": "$0022.00", "comment": "*****"} }}}}} - 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.

12. 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://wg31.washington.ibm.com:9445/cics/employee/222222
```

```
c:\z\openApi3>curl -X GET -w " - HTTP CODE %{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/cics/employee/222222
- HTTP CODE 403
```

A review of the SYSLOG will show that RACF access to the EJBRole protecting this method prevented USER2 from accessing this method.

```
ICH408I USER(USER2 ) GROUP(SYS1 ) NAME(WORKSHOP USER2 )
ATSZDFLT.cicsAPI.Staff CL(EJROLE )
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

13. 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 @insertCscvinc.json
https://wg31.washington.ibm.com:9445/cics/employee/
```

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948478",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

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

```
c:\z\openApi3>curl -X POST -w " - HTTP CODE ${http_code}" --header "Content-Type:
application/json" --insecure --user Fred:fred --data @insertCscvinc.json
https://wg31.washington.ibm.com:9445/cics/employee/
{"message":"Record with key 948478 was successfully inserted - CICS Identity
CICSUSER"} - HTTP CODE 200
```

14. Enter the curl command below to invoke the *PUT* method with URI /employee/{employee} to update a record.

```
curl -X PUT -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user Fred:fred --data @updateCscvinc.json
https://wg31.washington.ibm.com:9445/cics/employee/948478
```

```
c:\z\openApi3>curl -X PUT -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --
insecure --user Fred:fred --data @updateCscvinc.json
https://wg31.washington.ibm.com:9445/cics/employee/948478
>{"message":"Record with key 948478 was successfully updated - CICS identity CICSUSER"} - HTTP
CODE 200
```

In the above command, file *updateCscvinc.json* contains this content.

```
{
  "EmployeeUpdateServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "status": "s",
          "name": "A Johnson",
          "address": "Apex NC",
          "phoneNumber": "0065",
          "date": "01/31/23",
          "amount": "500",
          "COMMENT": "updated"
        }
      }
    }
  }
}
```

15. Repeat the GET request to observe that the updates have been applied.
16. Enter the curl command below to invoke the *DELETE* method with URI path */employee/{employee}*.

```
curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" -insecure --user Fred:fred https://wg31.washington.ibm.com:9445/cics/employee/948478
```

```
c:\z\openApi3>curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred https://wg31.washington.ibm.com:9445/cics/employee/948478
{"message":"Record with key 948478 was successfully deleted - CICS identity CICSUSER"} - HTTP
CODE 200
```

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

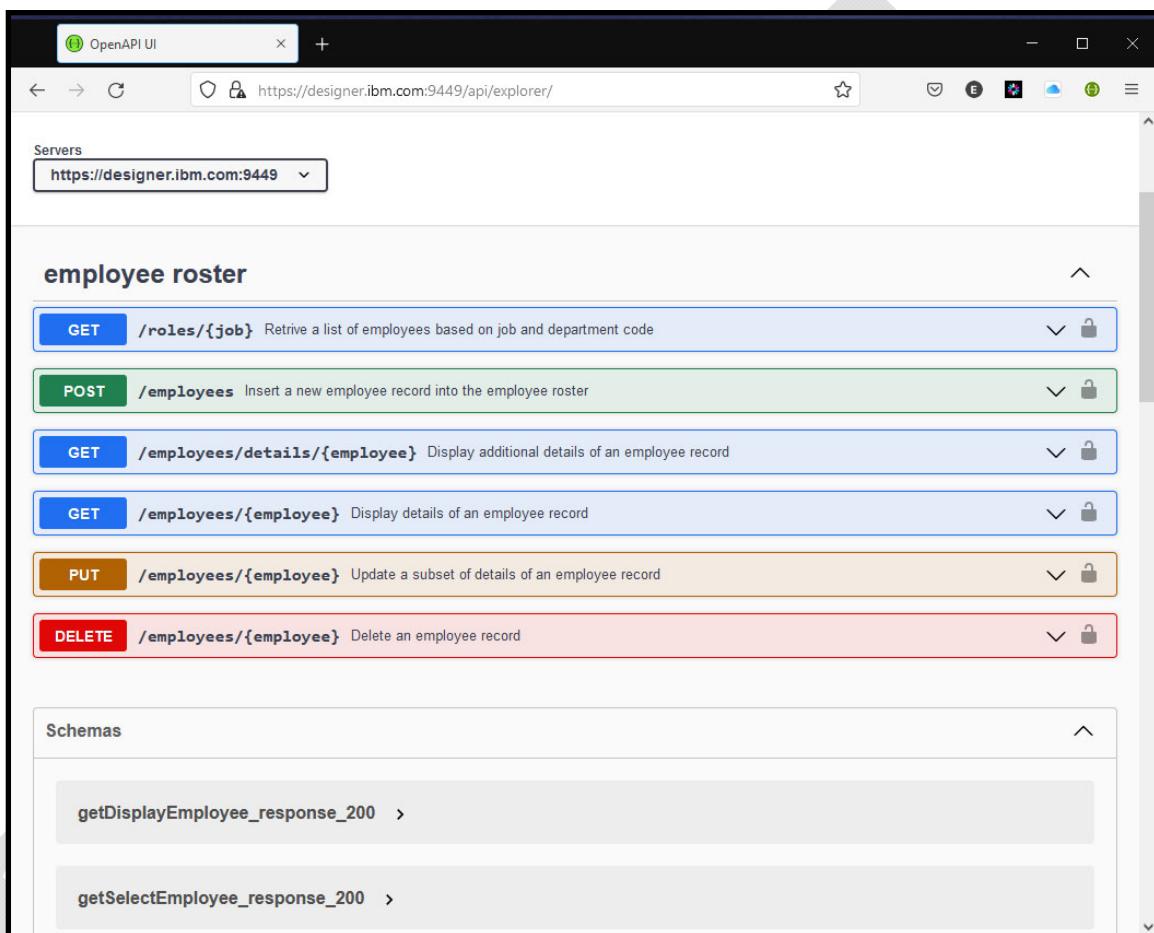
```
curl -X DELETE -w " - HTTP CODE ${http_code}" --header "Content-Type: application/json" -insecure --user user1:user1 https://wg31.washington.ibm.com:9445/cics/employee/948478
```

What HTTP status code occurred and why did it occur?

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 */employee* and the *GET* for */employee/{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.

17. 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 /employee* 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 in the browser is 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. Below the 'Request body' section, there is a 'request body' example value and a schema definition:

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "s",
  "lastName": "string",
  "departmentCode": "st",
  "phoneNumber": "string",
  "dateOfBirth": "string",
  "job": "string",
  "educationLevel": 3267,
  "sex": "s",
  "dateOfBirth": "string",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

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

The screenshot shows a modal dialog box titled 'Parameters'. It contains the same 'Authorization' parameter and 'Request body' section as the main UI, but with a 'Cancel' button in the top right corner. At the bottom of the dialog, there is a 'Servers' section with a note about overriding global server options, a path input field containing '/', and a large blue 'Execute' button.

19. 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
}
```

20. 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.

21. 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" (in blue) and "Clear".

The main area is titled "Responses". Under "curl", the command used for the POST request is displayed:

```
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, the "Request URL" is shown as <https://localhost:9449/employees>. The "Server response" section shows a 200 status code with the following details:

Code	Details
200	Response body

The response body is:

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

There are "Download" and "Copy" buttons next to the response body. Below the response body, the "Response headers" section lists:

- content-language: en-US
- content-length: 71
- content-type: application/json
- date: Fri, 17 Jun 2022 17:15:13 GMT
- X-firebase-spdy: h2
- x-powered-by: Servlet/4.0

22. 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.

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 zCEEService.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:DSN1JXUS: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
```

23. Scroll down and click on *GET /employee/{employees}* 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.

Responses

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
```

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

The table below list the contents of the VSAM data set.

stat	numb	name	addrx	Phone	datex	amount	comment
Y	000100	S. D. BORMAN	SURREY, ENGLAND	32156778	26 11 81	\$0100.11	*****
Y	000102	J. T. CZAYKOWSKI	WARWICK, ENGLAND	98356183	26 11 81	\$1111.11	*****
Y	000104	M. B. DOMBEY	LONDON, ENGLAND	12846293	26 11 81	\$0999.99	*****
Y	000106	A. I. HICKSON	CROYDON, ENGLAND	19485673	26 11 81	\$0087.71	*****
Y	000111	ALAN TULIP	SARATOGA, CALIFORNIA	46120753	01 02 74	\$0111.11	*****
Y	000762	SUSAN MALAIKA	SAN JOSE, CALIFORNIA	22312121	01 06 74	\$0000.00	*****
Y	000983	J. S. TILLING	WASHINGTON, DC	34512120	21 04 75	\$9999.99	*****
Y	001222	D.J.VOWLES	BOBLINGEN, GERMANY	70315551	10 04 73	\$3349.99	*****
Y	001781	TINA J YOUNG	SINDELFINGEN, GERMANY	70319990	21 06 77	\$0009.99	*****
Y	003210	B.A. WALKER	NICE, FRANCE	12345670	26 11 81	\$3349.99	*****
N	003214	PHIL CONWAY	SUNNYVALE, CAL.	34112120	00 06 73	\$0009.99	*****
N	003890	BRIAN HARDER	NICE FRANCE	00000000	28 05 74	\$0009.99	*****
N	004004	JANET FOUCHE	DUBLIN, IRELAND	71112121	02 11 73	\$1259.99	*****
N	004445	DR. P. JOHNSON	SOUTH BEND, S.DAK.	61212120	26 11 81	\$0009.99	*****
N	004878	ADRIAN JONES	SUNNYVALE, CALIF.	32212120	10 06 73	\$5399.99	*****
N	005005	A. E. DALTON	SAN FRANCISCO, CA.	00000001	01 08 73	\$0009.99	*****
N	005444	ROS READER	SARATOGA, CALIF.	67712120	20 10 74	\$0809.99	*****
N	005581	PETE ROBBINS	BOSTON, MASS.	41312120	11 04 74	\$0259.99	*****
Y	006016	SIR MICHAEL ROBERTS	NEW DELHI, INDIA	70331211	21 05 74	\$0009.88	*****
N	006670	IAN HALL	NEW YORK, N.Y.	21212120	31 01 75	\$3509.88	*****
Y	006968	J.A.L. STAINFORTH	WARWICK, ENGLAND	56713821	26 11 81	\$0009.88	*****
N	007007	ANDREW WHARMBY	STUTTGART, GERMANY	70311000	10 10 75	\$5009.88	*****
N	007248	M. J. AYRES	REDWOOD CITY, CALF.	33312121	11 10 75	\$0009.88	*****
Y	007779	MRS. A. STEWART	SAN JOSE, CALIF.	41512120	03 01 75	\$0009.88	*****
Y	009000	P. E. HAVERCAN	WATERLOO, ONTARIO	09876543	21 01 75	\$9000.00	*****
Y	100000	M. ADAMS	TORONTO, ONTARIO	03415121	26 11 81	\$0010.00	*****
Y	111111	C. BAKER	OTTAWA, ONTARIO	51212003	26 11 81	\$0011.00	*****
Y	200000	S. P. RUSSELL	GLASGOW, SCOTLAND	63738290	26 11 81	\$0020.00	*****
Y	222222	DR E. GRIFFITHS	FRANKFURT, GERMANY	20034151	26 11 81	\$0022.00	*****
Y	300000	V. J. HARRIS	NEW YORK, U.S.	64739801	26 11 81	\$0030.00	*****
Y	333333	J.D. HENRY	CARDIFF, WALES	78493020	26 11 81	\$0033.00	*****
Y	400000	C. HUNT	MILAN, ITALY	25363738	26 11 81	\$0040.00	*****
Y	444444	D. JACOBS	CALGARY, ALBERTA	77889820	26 11 81	\$0044.00	*****
Y	500000	P. KINGSLEY	MADRID, SPAIN	44454640	26 11 81	\$0000.00	*****
Y	555555	S.J. LAZENBY	KINGSTON, N.Y.	39944420	26 11 81	\$0005.00	*****
Y	600000	M.F. MASON	DUBLIN, IRELAND	12398780	26 11 81	\$0010.00	*****
Y	666666	R. F. WALLER	LA HULPE, BRUSSELS	42983840	26 11 81	\$0016.00	*****
Y	700000	M. BRANDON	DALLAS, TEXAS	57984320	26 11 81	\$0002.00	*****
Y	777777	L.A. FARMER	WILLIAMSBURG, VIRG.	91876131	26 11 81	\$0027.00	*****
Y	800000	P. LUPTON	WESTEND, LONDON	24233389	26 11 81	\$0030.00	*****
Y	888888	P. MUNDY	NORTHAMPTON, ENG.	23691639	26 11 81	\$0038.00	*****
Y	900000	D.S. RENSHAW	TAMPA, FLA.	35668120	26 11 81	\$0040.00	*****
Y	999999	ANJI STEVENS	RALEIGH, N.Y.	84591639	26 11 81	\$0049.00	*****

Using TLS to access CICS

Creating the RACF resources

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

The certificate authority public certificate, labeled as *CICS CA*, be imported into RACF as a CERTAUTH certificate. There are also a personal certificate signed by the same certificate authority that will be used as CICS's server certificate (labeled *CICS Server Cert*) and another personal certificate that will be used as a client certificate (labeled *USER1 CICS Cert*) 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 when they are being imported in to RACF.

The certificate authority (CERTAUTH) certificate and the personal certificate labeled *CICS Server Cert* will be connected to the RACF key ring used by CICS as the server endpoint.

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

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 be 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. More than likely you will only be using the RACDCERT command to create keyrings and connect certificates to these key rings.

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

2. Browse member **CICSTLS3**. The job contains the RACF RACDCERT commands below. Submit the job for execution.

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

racdcert id(CICSSTC) Withlabel('CICS Client Cert') -
    add('USER1.CICSCLT.P12') password('secret')

racdcert id(CICSSTC) Withlabel('CICS Server Cert') -
    add('USER1.CICSSRV.P12') password('secret')

racdcert id(USER1) withlabel('USER1 CICS Cert') -
    add('USER1.CICSUSR1.P12') password('secret')

racdcert id(USER2) withlabel('USER2 CICS Cert') -
    add('USER1.CICSUSR2.P12') password('secret')

setropts raclist(digtcert,digtnmap) refresh

/* Create CICS key ring and connect CA and personal certificates */
racdcert id(cicsstc) addring(CICS.KeyRing)

racdcert id(cicsstc) connect(ring(CICS.KeyRing) +
    label('CICS CA') certauth usage(certauth))

racdcert id(cicsstc) connect(ring(CICS.KeyRing) +
    label('CICS CA') certauth usage(certauth))

/* Connect default personal certificate */
racdcert id(cicsstc) connect(ring(CICS.KeyRing) +
    label('CICS Client Cert') default

/* Connect default personal certificate */
racdcert id(cicsstc) connect(ring(CICS.KeyRing) +
    label('CICS Server Cert')

setropts raclist(digtring,digtnmap) refresh

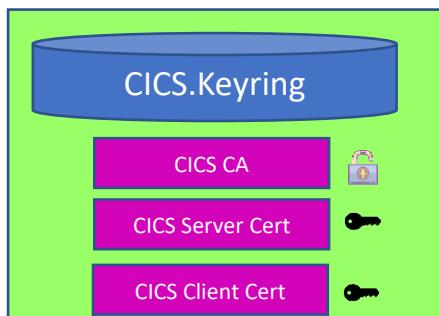
permit irr.digtcert.liststring +
    class(facility) id(cicsstc) access(read)

permit irr.digtcert.list +
    class(facility) id(cicsstc) access(read)
```

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 CICS server personal certificate to this key ring.
- Connect the CA public certificate used to sign the CICS server's certificate to this key ring.
- The in-storage profile for digital certificates resources are refreshed.

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



Using TLS to access CICS

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.

Enabling CICS TLS support

The next step is to update the CICS's regions initialization parameters to add support for TLS. This involves adding system initialization table (SIT) parameters.

1. Edit member *CICSZ* in data set *CICSTS.CICSZ.SYSIN*.
2. Add the following system initialization overrides anywhere before the .END line.

```
KEYRING=CICS.KeyRing,
ENCRYPTION=STRONG,
MAXSSLTCBS=8,
SSLDELAY=600,
```

3. Use the CEDA transaction to change the *SCHEME* and *PORT* of the *BAQURIMP* Urimap resource from *HTTP* to *HTTPS* and from *9120* to *9443*, see below.

CEDA COPY TCPIPS(IPICTLSS) G(IPICTLSS) TO(SYSPGRP)

IBM z/OS Connect (OpenAPI 3.0)

Below is the definition of this TCPIPSERVICE.

```

OVERTYPE TO MODIFY
CEDA ALter TCPIPSERVICE( IPICTLS )
TCPIPSERVICE : IPICTLS
GR0up : SSLGRP
DEscription ==>
Urm ==> DFHISAIPI
P0rtnumber ==> 01493 1-65535
Status ==> Open Open | Closed
PR0tocol ==> IPic Http | Eci | User | IPic
TRansaction ==> CISS
Backlog ==> 00000 0-32767
TSqprefix :
Host ==> ANY
(Mixed Case) ==>
Ipaddress ==> ANY
SPecifcps ==>
S0cketclose ==> No No | 0-240000 (HHMMSS)
MAXPersist ==> No No | 0-65535
+ MAXDataLEN ==> 000032 3-524288

SYSID=CICS APPLID=CICSS5Z

PF 1 HELP 2 COM 3 END      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
M A F 14/024
Connected to remote server/host wg31a using lu/pool TCP00105 and port 23

```



```

OVERTYPE TO MODIFY
CEDA ALter TCPIPSERVICE( IPICTLS )
+ Backlog ==> 00000 0-32767
TSqprefix :
Host ==> ANY
(Mixed Case) ==>
Ipaddress ==> ANY
SPecifcps ==>
S0cketclose ==> No No | 0-240000 (HHMMSS)
MAXPersist ==> No No | 0-65535
MAXDataLEN ==> 000032 3-524288
SECURITY
SSI ==> Yes Yes | No | Clientauth | Attlsware
CErificate ==> CICS Server Cert
(Mixed Case)
PRIvacy : Notsupported | Required | Supported
CIPhers ==> 3538392F3233
(Mixed Case)
+ AUthenticate ==> No | Basic | Certificate | AUTORegister

SYSID=CICS APPLID=CICSS5Z

PF 1 HELP 2 COM 3 END      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
M A F 06/022
Connected to remote server/host wg31a using lu/pool TCP00105 and port 23

```

Shutdown CICS with a ***CEMT PE SHUTDOWN*** command, using either an MVS modify command or in a CICS terminal session.

4. When CICS is no longer active, restart the region with MVS command ***S CICSZ***.

Test the TLS connection from the z/OS Connect Designer to a CICS subsystem

In the following steps you will download the CICS 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 CICS from the *Designer*.

- ___ 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 CICSCA.PE<**
- ___ 7. Enter the command: **bin**
- ___ 8. Enter the command: **mget CICSUSR1.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 CICSCA.PEM
200 Representation type is Ascii NonPrint
200 Port request OK.
125 Sending data set USER1.CICSCA.PEM
250 Transfer completed successfully.
ftp: 1244 bytes received in 0.00Seconds 414.67Kbytes/sec.
ftp> bin
200 Representation type is Image
ftp> mget CICSUSR1.p12
200 Representation type is Image
200 Port request OK.
125 Sending data set USER1.CICSUSR1.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 CICS. Note that JSSE supports the use of the same or different key store files for both the personal and CERTAUTH repositories.

First, create a CICS trust store and import the CICS CA public key file into the JSSE trust store.

- ___ 1. In an Ubuntu session, copy the CERTAUTH public key certificate file (*CICSCA.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/CICSCA.PEM /home/workstation/podman/cscvinc/certs
```

Tech-Tip: The directory */home/workstation/podman/cscvinc/certs* has been mapped to container directory */output/resources/security* in the docker-compose.yaml used to create the cscvinc container.

- ___ 2. In the same Ubuntu session, import the CERTAUTH public key into the container's CICS key store (this will create the JSSE keystore file).

```
podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsTrustStore.jks -storetype PKCS12 -storepass changeit -importcert -file /output/resources/security/CICSCA.PEM -alias CICSCA
```

Tech-Tip: The command, *podman exec -it cscvinc_zosConnect_1* provides a command interface for invoking command in the container. In this case, the *podman 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/mitchj/podman/cscvinc/certs:> podman exec -it cscvinc_zosConnect_1 keytool -keystore
/output/resources/security/cicsTrustStore.jks -storetype PKCS12 -storepass changeit -importcert -file
/output/resources/security/CICSCA.PEM -alias CICSCA
Error: can only create exec sessions on running containers: container state improper
root:/home/mitchj/podman/cscvinc/certs:> podman start cscvinc_zosConnect_1
cscvinc_zosConnect_1
root:/home/mitchj/podman/cscvinc/certs:> podman exec -it cscvinc_zosConnect_1 keytool -keystore
/output/resources/security/cicsTrustStore.jks -storetype PKCS12 -storepass changeit -importcert -file
/output/resources/security/CICSCA.PEM -alias CICSCA
Owner: CN=CICS CA, OU=ATS, O=IBM
Issuer: CN=CICS CA, OU=ATS, O=IBM
Serial number: 0
Valid from: 7/20/22 5:00 AM until: 1/1/24 4:59 AM
Certificate fingerprints:
      MD5: 1E:E3:84:FB:72:B7:6E:62:44:CC:78:7E:D9:8C:F2:30
      SHA1: 4D:CF:C3:BE:7D:B9:DE:C2:F4:6B:86:E3:14:B9:5B:25:9B:F8:02:CB
      SHA256: 75:9C:9B:D7:6D:8D:D2:09:9A:4D:B0:82:9A:CB:3A:09:49:60:CB:84:60:DE:ED:6F:7F:36:15:01:95:85:71:50
      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: 0b b8 de 32 a7 84 bc 71 4e 42 d4 77 09 57 2e 4e ...2...qNB.w.W.N
    0010: 59 98 fc c5 Y...
]
]
#4: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

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

```

Tech-Tip: All Podman commands(*podman*) need to be entered in Ubuntu sessions.

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

podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsTrustStore.jks -storetype PKCS12 --storepass changeit -list

```

root:/home/mitchj/podman/cscvinc/certs:> podman exec -it cscvinc_zosConnect_1 keytool -keystore
/output/resources/security/cicsTrustStore.jks -storetype PKCS12 --storepass changeit -list

Keystore type: PKCS12
Keystore provider: IBMJCE

Your keystore contains 1 entry

cicsca, Jul 23, 2022, trustedCertEntry,
Certificate fingerprint (SHA1): 4D:CF:C3:BE:7D:B9:DE:C2:F4:6B:86:E3:14:B9:5B:25:9B:F8:02:CB

```

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

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

```
podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsTrustStore.jks -storetype PKCS12 --storepass changeit -list -alias cicsca -v
```

```
root:/home/mitchj/podman/cscvinc/certs:> podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsTrustStore.jks -storetype PKCS12 --storepass changeit -list -alias cicsca -v
Alias name: cicsca
Creation date: Jul 23, 2022
Entry type: trustedCertEntry

Owner: CN=CICS CA, OU=ATS, O=IBM
Issuer: CN=CICS CA, OU=ATS, O=IBM
Serial number: 0
Valid from: 7/20/22 5:00 AM until: 1/1/24 4:59 AM
Certificate fingerprints:
MD5: 1E:E3:84:FB:72:B7:6E:62:44:CC:78:7E:D9:8C:F2:30
SHA1: 4D:CF:C3:BE:7D:B9:DE:C2:F4:6B:86:E3:14:B9:5B:25:9B:F8:02:CB
SHA256: 75:9C:9B:D7:6D:8D:D2:09:9A:4D:B0:82:9A:CB:3A:09:49:60:CB:84:60:DE:ED:6F:7F:36:15:01:95:85:71:50
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
]
1

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdIdentifier [
  KeyIdentifier [
    0000: 0b b8 de 32 a7 84 bc 71 4e 42 d4 77 09 57 2e 4e ...2...qNB.w.W.N
    0010: 59 98 fc c5 Y...
  ]
]
1

#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=CICS CA, OU=ATS, O=IBM*

This diagram shows what we just created.



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

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, *CICSSL_PORT*, *CICSTRUSTSTORE_PASSWORD* and *CICSKEYSTORE_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
      - CICSTRUSTSTORE_PASSWORD=changeit
      - CICSKEYSTORE_PASSWORD=secret
      - CICSSL_PORT=1493
      - DB2_USERNAME=USER1
      - DB2_PASSWORD=USER1
      - DB2_HOST=wg31.washington.ibm.com
      - DB2_PORT=2446
      - HTTP_PORT=9080
    ports:
      - "9447:9443"
      - "9084:9080"
    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 *CICSKEYSTORE* 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/podman/cscvinc/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/podman/cscvinc

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

podman stop cscvinc_zosConnect_1

5. Remove the container with this command. This remove the container but leaves contents of the project directory as is. This means the contents of this directory structure (the API definition) will be remapped when the container is restarted with the *podmon-compose* command.

podman rm cscvinc_zosConnect_1

6. Restart the container with this command (*invoke this command while in directory /home/workstation/podman/employee*).

podman-compose up -d

Now the *zosconnect.cicsIpicConnection* entry for the CICS connection needs to have additional attributes added for TLS.

7. Copy the Windows file *cicsTlsServer.xml* to the container configuration directory as file *cics.xml*.

***cp /mnt/c/z/openapi3/tls/cicsTlsServer.xml
/home/workstation/podman/cscvinc/project/src/main/liberty/config/cics.xml***

Below are the contents of *cicsTlsServer.xml*. The boldfaced lines are the additions made to the original contents of *cics.xml* for adding TLS support between the *Designer* and CICS.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="IPIC connection to CICS">
    <featureManager>
        <feature>zosconnect:cics-1.0</feature>
    </featureManager>

    <zosconnect_cicsIpicConnection id="cicsConn"
        host="${CICS_HOST}"
        port="${CICSSL_PORT}"
        sslCertsRef="cicsSSLSettings"
        authDataRef="cicsCredentials" />

    <ssl id="cicsSSLSettings"
        keyStoreRef= "cicsTrustStore"
        trustStoreRef= "cicsTrustStore" />
    <keyStore id= "cicsTrustStore"
        location="/output/resources/security/cicsTrustStore.jks"
        password="${CICSTRUSTSTORE_PASSWORD}" type="PKCS12" />

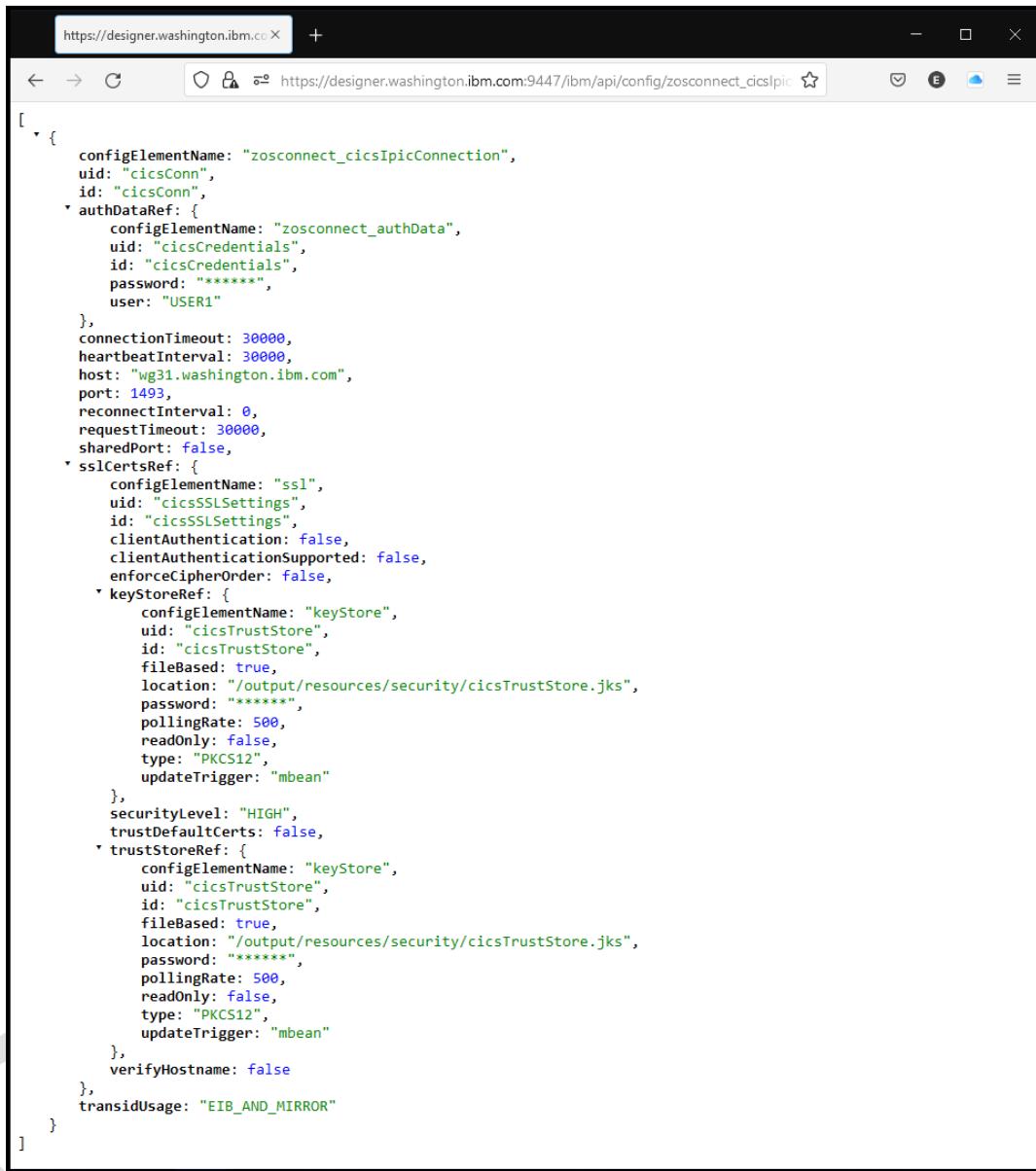
    <zosconnect_authData id="cicsCredentials"
        user="${CICS_USER}"
        password="${CICS_PASSWORD}" />

</server>
```

The above configuration defines a TLS repertoire (*cicsSSLSettings*) for use when connection to a CICS. Note the *sslRef* in the *zosconnect_cicsIpicConnection* configuration elements references *cicsSSLSettings*. 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, *cicsTrustStore*, identifies the key store we created in the previous section. This key store contains the CICS *CERTAUTH* certificate.

8. Enter the URL below in a browser to verify the updated `zosconnect_cicsIpicConnection` configuration

https://designer.washington.ibm.com:9447/ibm/api/config/zosconnect_cicsIpicConnection



The screenshot shows a browser window with the URL `https://designer.washington.ibm.com:9447/ibm/api/config/zosconnect_cicsIpicConnection`. The page displays a JSON object representing the configuration for a CICS IPIC connection. The configuration includes details such as host, port, reconnect interval, and security settings like SSL certificates and keystores.

```

[{"zosconnect_cicsIpicConnection": {
    "authDataRef": {
        "cicsConn": {
            "uid": "cicsConn",
            "id": "cicsConn",
            "password": "*****",
            "user": "USER1"
        }
    },
    "connectionTimeout": 30000,
    "heartbeatInterval": 30000,
    "host": "wg31.washington.ibm.com",
    "port": 1493,
    "reconnectInterval": 0,
    "requestTimeout": 30000,
    "sharedPort": false,
    "sslCertsRef": {
        "cicsSSLSettings": {
            "clientAuthentication": false,
            "clientAuthenticationSupported": false,
            "enforceCipherOrder": false
        }
    },
    "keyStoreRef": {
        "cicsTrustStore": {
            "fileBased": true,
            "location": "/output/resources/security/cicsTrustStore.jks",
            "password": "*****",
            "pollingRate": 500,
            "readOnly": false,
            "type": "PKCS12",
            "updateTrigger": "mbean"
        }
    },
    "securityLevel": "HIGH",
    "trustDefaultCerts": false,
    "trustStoreRef": {
        "cicsTrustStore": {
            "fileBased": true,
            "location": "/output/resources/security/cicsTrustStore.jks",
            "password": "*****",
            "pollingRate": 500,
            "readOnly": false,
            "type": "PKCS12",
            "updateTrigger": "mbean"
        }
    },
    "verifyHostname": false
},
"transidUsage": "EIB_AND_MIRROR"}]

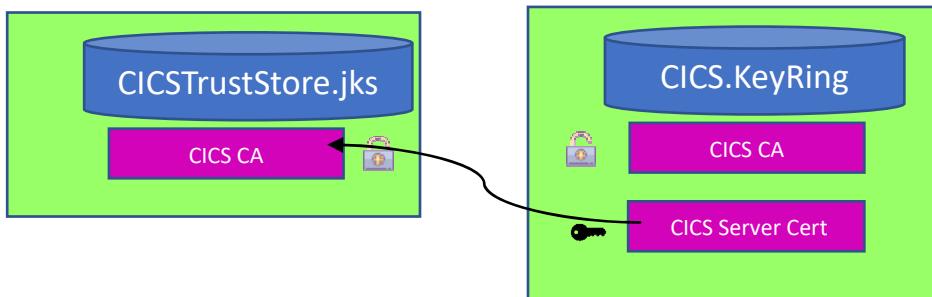
```

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.

IBM z/OS Connect (OpenAPI 3.0)

This diagram shows the TLS handshake flow as it currently configured. When a client requests a connection, CICS 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).



Test the TLS connection between the *Designer* and CICS 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:9447/employee/000100
```

```
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/employee/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
```

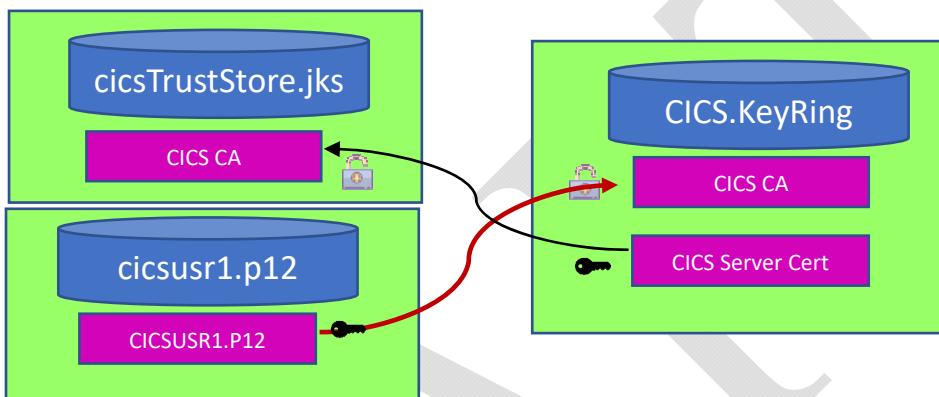
And a review of the Liberty traces shows the handshake flow.

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.



1. In an Ubuntu session, copy the user's private key certificate file (*CICSUSR1.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/CICSUSR1.P12 /home/workstation/podman/cscvinc/certs
```

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

```
cp /mnt/c/z/openapi3/tls/cicsTlsMutual.xml  
/home/workstation/podman/cscvinc/project/src/main/liberty/config/cics.xml
```

The `zosconnect_cicsIpicConnection` element need values to match the IPCONN resource for `zosConnectNetworkid` and `zosConnectApplid`. The `authDataRef` attribute is no longer needed.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="IPIC connection to CICS">
  <featureManager>
    <feature>zosconnect:cics-1.0</feature>
  </featureManager>

  <zosconnect_cicsIpicConnection id="cicsConn">
    host="${CICS_HOST}"
    port="${CICSSL_PORT}"
    zosConnectNetworkid="DESIGNER"
    zosConnectApplid="DESIGNER"
    sslCertsRef="cicsSSLSettings"/>

    <ssl id="cicsSSLSettings">
      keyStoreRef= "cicsKeyStore"
      trustStoreRef= "cicsTrustStore" />
    <keyStore id= "cicsTrustStore">
      location="/output/resources/security/cicsTrustStore.jks"
      password="${CICSTRUSTSTORE_PASSWORD}" type="PKCS12" />
    <keyStore id= "cicsKeyStore">
      location="/output/resources/security/CICSUSR1.P12"
      password="${CICSKEYSTORE_PASSWORD}" type="PKCS12" />
    </ssl>
  </zosconnect_cicsIpicConnection>
</server>
```

The above configuration defines a TLS repertoire (`cicsSSLSettings`) for use when connection to a CICS. Note the `sslRef` in the `zosconnect_cicsIpicConnection` configuration elements references `cicsSSLSettings`. The TLS repertoire has attributes `keyStoreRef` and `trustStoreRef` referencing different `keystore` elements. The `keystore` element, `cicsTrustStore`, identifies the key store we used for server authentication in the previous section. This key store contains CICS `CERTAUTH` certificate. The `keystore` element, `cicsKeyStore`, identifies the key store that contains this client's personal certificate, in this case, the P12 file can be a key store.

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

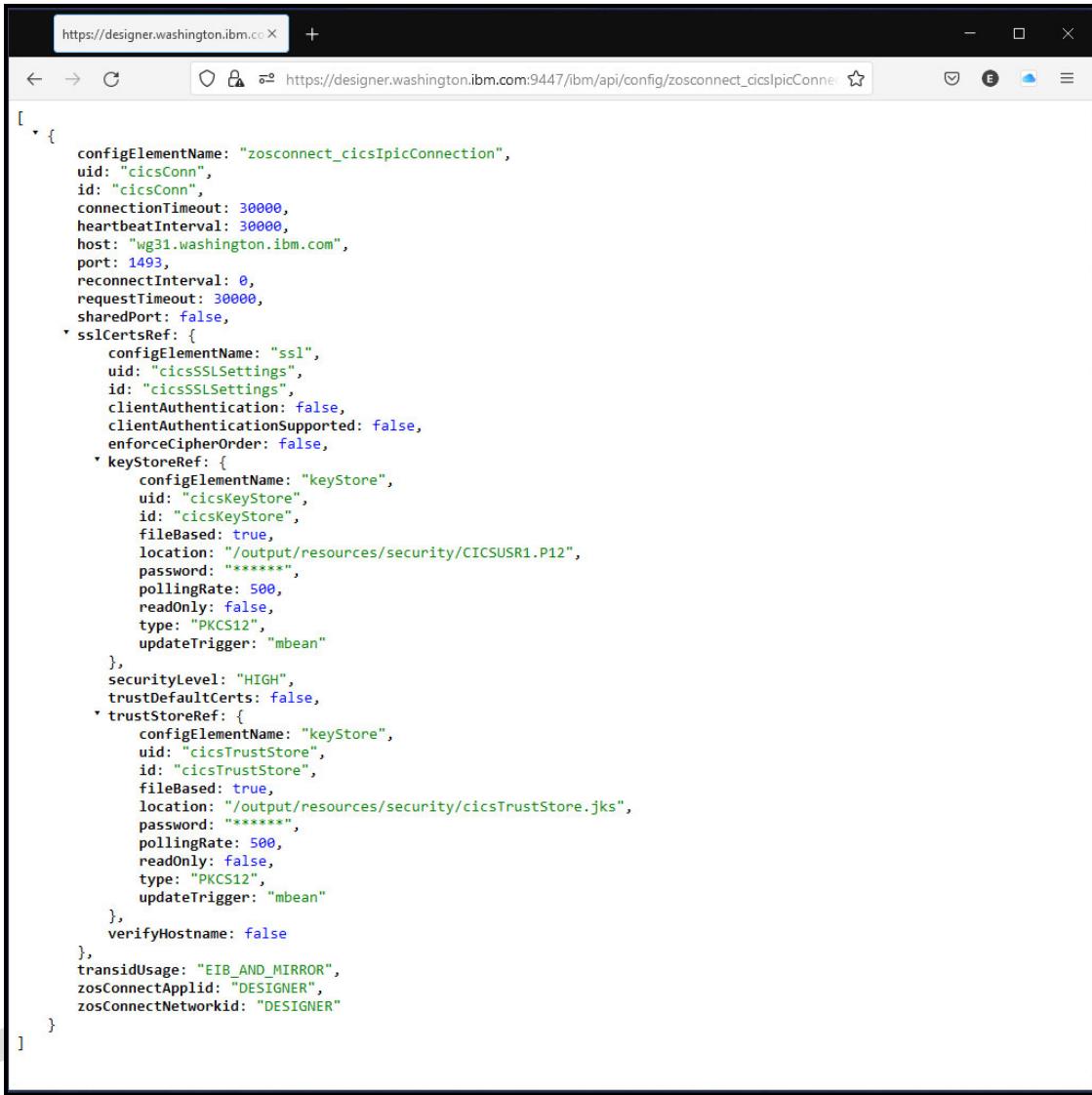
`podman stop cscvinc_zosConnect_1`

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

`podman start cscvinc_zosConnect_1`

5. Enter the URL below in a browser to verify the updated `zosconnect_cicsIpicConnection` configuration

https://designer.washington.ibm.com:9447/ibm/api/config/zosconnect_cicsIpicConnection



```
[{"configElementName": "zosconnect_cicsIpicConnection", "uid": "cicsConn", "id": "cicsConn", "connectionTimeout": 30000, "heartbeatInterval": 30000, "host": "wg31.washington.ibm.com", "port": 1493, "reconnectInterval": 0, "requestTimeout": 30000, "sharedPort": false, "sslCertsRef": {"configElementName": "ssl", "uid": "cicsSSLSettings", "id": "cicsSSLSettings", "clientAuthentication": false, "clientAuthenticationSupported": false, "enforceCipherOrder": false}, "keyStoreRef": {"configElementName": "keyStore", "uid": "cicsKeyStore", "id": "cicsKeyStore", "fileBased": true, "location": "/output/resources/security/CICSUSR1.P12", "password": "*****", "pollingRate": 500, "readOnly": false, "type": "PKCS12", "updateTrigger": "mbean"}, "securityLevel": "HIGH", "trustDefaultCerts": false, "trustStoreRef": {"configElementName": "keyStore", "uid": "cicsTrustStore", "id": "cicsTrustStore", "fileBased": true, "location": "/output/resources/security/cicsTrustStore.jks", "password": "*****", "pollingRate": 500, "readOnly": false, "type": "PKCS12", "updateTrigger": "mbean"}, "verifyHostname": false}, {"transidUsage": "EIB_AND_MIRROR", "zosConnectApplid": "DESIGNER", "zosConnectNetworkid": "DESIGNER"}]
```

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. The CICS IPCONN and TCPIPServices need to be updated.

The IPCONN resources needs to have values for *Applid* and *Networkid* that will match the *zosconnect_cicsIpIpicConnection* element as a value of *Certuser* for *Linkauth*.

```

WG31                               CICS RELEASE = 0710
File Edit Settings View Communication Actions Window Help
OVERTYPE TO MODIFY
CEDA ALter Ipconn( IPICTLs )
Ipconn      : IPICTLs
Group       : SSLGRP
DEscription ==>
IPIC CONNECTION IDENTIFIERS
Applid      ==> DESIGNER
Networkid   ==> DESIGNER
Host        ==>
(Mixed Case) ==>
Port        ==> No           No | 1-65535
Tcpipservice ==> IPICTLs
HA          ==> No           No | Yes
IPIC CONNECTION PROPERTIES
Receivecount ==> 001           1-999
SENDcount   ==> 000           0-999
QueueLimit  ==> No            No | 0-9999
Maxqtime    ==> No            No | 0-9999
+ OPERATIONAL PROPERTIES
SYSID=CICS APPLID=CICS532

PF 1 HELP 2 COM 3 END      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
MA C 06/022
Connected to remote server/host wg31a using lu/pool TCP00113 and port 23

WG31                               CICS RELEASE = 0710
File Edit Settings View Communication Actions Window Help
OVERTYPE TO MODIFY
CEDA ALter Ipconn( IPICTLs )
+ AUTOconnect ==> No           No | Yes
INSERVICE   ==> Yes          Yes | No
SECURITY
SSL         ==> Yes          No | Yes
CERTIFICATE ==>             (Mixed Case)
CIPHERS    ==> 3538392F3233
(Mixed Case)
Linkauth    ==> Certuser     Secuser | Certuser
SECURITYNAME ==
Userauth    ==> Local        Local | Identify | Verify | Defaultuser
IDPROP      ==> Optional    Notallowed | Optional | Required
RECOVERY
XINACTION  ==> Keep        Keep | Force
MIRROR TASK PROPERTIES
MIRRORLIFE ==> Request     Request | Task | Uow
DEFINITION SIGNATURE
+ DEFINETIME : 07/24/22 10:03:09
SYSID=CICS APPLID=CICS532

PF 1 HELP 2 COM 3 END      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
MA A 04/022
Connected to remote server/host wg31a using lu/pool TCP00114 and port 23

```

The value for *Userauth* is important. When *Userauth* is set to a value of **Local** as above, the CICS transaction will run under the identity associate with the client certificate sent from the *Designer* container. If *Userauth* is set to a value of **Identity**, the identity authenticated by Designer will be propagated to CICS and this distributed identity will need to be map to a valid RACF identity because the CICS transaction will be started using the mapped identity, using RACMAP commands like the ones below:

```
RACMAP ID(FRED) MAP USERDIDFILTER(NAME('Fred')) REGISTRY(NAME('*')) WITHLABEL('zCEE FRED')
RACMAP ID(USER1) MAP USERDIDFILTER(NAME('user1')) REGISTRY(NAME('*')) WITHLABEL('zCEE user1')
```

The TCPIPSERVICE will need to specify a value of ***ClientAuth*** for the **SSL** attribute.

```

OVERTYPE TO MODIFY
CEDA Alter TCPIPSERVICE( IPICTLS )
+ SECURITY
  SSL      ==> Clientauth
  CErtificate ==> CICS Server Cert
  (Mixed Case)
  PRIVacy   :
  Ciphers    ==> 3538392F3233
  (Mixed Case)
  AUTHenticate ==>
    Realm     ==>
    (Mixed Case)
    ATtachsec  ==>
  DNS CONNECTION BALANCING
  DNsgroup   :
  GRPcritical : No           No | Yes
  DEFINITION SIGNATURE
+  DEFinetime   : 07/20/22 14:35:27

CICS RELEASE = 0710
SYSID=CICS APPLID=CICS532
PF 1 HELP 2 COM 3 END       6 CCSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
M8 C 05/022
Connected to remote server/host wg31a using lu/pool TCP00113 and port 23

```

7. Test the TLS connection between the *Designer* and CICS 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:9447/employee/000100
```

Results when the IPCONN *Userauth* attribute set to **Local**, the identity mapped to the identity is used,

```
c:\z>curl -X GET -w "- HTTP CODE %{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://designer.washington.ibm.com:9447/employee/000100
{"summary":{"message":"Record with key 000100 was successfully retrieved - CICS identity
USER1"},"detail":{"EmployeeSelectServiceOperationResponse":{"employeeData":{"response":{"employeeDetails":{"employeeNumber":"000100","name":"S. D. BORMAN","address":"SURREY,
ENGLAND","phoneNumber":"32156778","date":"26 11 81","amount":"$0100.11","comment":*****}}}}} - 
HTTP CODE 200
```

Results when the IPCONN *Userauth* attribute set to **Identify**, the identity asserted from the Designer is used.

```
c:\z>curl -X GET -w "- HTTP CODE %{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://designer.washington.ibm.com:9447/employee/000100
{"summary":{"message":"Record with key 000100 was successfully retrieved - CICS identity
FRED"},"detail":{"EmployeeSelectServiceOperationResponse":{"employeeData":{"response":{"employeeDetails":{"employeeNumber":"000100","name":"S. D. BORMAN","address":"SURREY,
ENGLAND","phoneNumber":"32156778","date":"26 11 81","amount":"$0100.11","comment":*****}}}}} - 
HTTP CODE 200
```

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

```

3 Certificate information:
[7/23/22 23:33:01:979 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3   Subject DN: CN=CICS Server, OU=ATS, O=IBM
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3   Issuer DN: CN=CICS CA, OU=ATS, O=IBM
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3   Serial number: 1
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3 Certificate information:
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3   Subject DN: CN=CICS CA, OU=ATS, O=IBM
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3   Issuer DN: CN=CICS CA, OU=ATS, O=IBM
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3   Serial number: 0
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3 Delegating to X509TrustManager implementation: com.ibm.jsse2.br
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
3 Server is trusted by all X509ExtendedTrustManager.
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509TrustManager
< checkServerTrusted Exit
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
> chooseClientAlias Entry
RSA
CN=CICS CA, OU=ATS, O=IBM
CN=CICS Client, OU=ATS, O=IBM
CN=CICS Server, OU=ATS, O=IBM
Socket[addr=wg31.washington.ibm.com/9.82.31.201,port=1493,localport=44896]
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
< chooseClientAlias Exit
[7/23/22 23:33:01:980 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
> chooseClientAlias Entry
RSA
CN=CICS CA, OU=ATS, O=IBM
CN=CICS Client, OU=ATS, O=IBM
CN=CICS Server, OU=ATS, O=IBM
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.websphere.ssl.JSSEHelper
> getOutboundConnectionInfo Entry
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.config.ThreadContext
3 getOutboundConnectionInfo
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.websphere.ssl.JSSEHelper
< getOutboundConnectionInfo Exit
null
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
< chooseClientAlias (from JSSE) Exit
user1 cics cert
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
> getPrivateKey Entry
user1 cics cert
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
3 getX509KeyManager -> com.ibm.jsse2.bn
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
< getPrivateKey -> true Exit
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
> getCertificateChain: user1 cics cert Entry
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
3 getX509KeyManager -> com.ibm.jsse2.bn
[7/23/22 23:33:01:981 UTC] 00000098 id=00000000 com.ibm.ws.ssl.core.WSX509KeyManager
< getCertificateChain Exit

[
[
  Version: V3
  Subject: CN=USER1 Client Cert, 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 to not include the commands to connect a personal certificate to the key ring.

```
racdcert id(atsserv) addring(CICSClient.KeyRing)
```

```
racdcert id(atsserv) connect(ring(CICSClient.KeyRing) label('CICS CA') certauth usage(certauth))
```

```
setropts raclist(digtring,digtnmap) refresh
```

2. Update the current contents `/var/zcee/openapi3/includes/cicsApi3.xml` with the contents in bold below:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="CICS SSL">
  <featureManager>
    <feature>zosconnect:cics-1.0</feature>
  </featureManager>
  <zosconnect_credential id="commonCredentials"
    user="${CICS_USER}" password="${CICS_PASSWORD}" />
  <zosconnect_cicsIpicConnection id="cicsIpicConn"
    host="${CICS_HOST}" port="${CICS_PORT}"
    sslRef="cicsSSLSettings" />
  <ssl id="cicsSSLSettings"
    keyStoreRef="cicsKeyRing"
    trustStoreRef="cicsKeyRing" />
  <keyStore id="cicsKeyRing"
    location="safkeyring:///CICSClient.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
</server>
```

3. Change the value of the `CICS_PORT` variable `/var/zcee/openapi3/includes/openApi3.xml` from 1491 to **1493**.

```
<variable name="CICS_HOST" value="wg31.washington.ibm.com"/>
<variable name="CICS PORT" value="1493"/>
```

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 79

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. And the contents of the original YAML file used to develop the Open API 3 API.

Client Certificate Requests

1. Create a self-signed certificate (and indirectly create a local keystore)

```
podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsKeyStore.jks -storetype PKCS12 -storepass changeit -genkey -keysize 2048 -alias cicsusr -dname "CN=user1, O=IBM, C=US" -keyalg RSA -validity 365
```

2. Now list the contents of the details of the *designer* certificate store using this command

```
podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsKeyStore.jks -storetype PKCS12 --storepass changeit -list -alias designer -v
```

```
root:/home/workstation/podman/employee:> podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsKeyStore.jks -storetype PKCS12 --storepass changeit -list -alias designer -v
Alias name: designer
Creation date: Jul 1, 2022
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=user1, O=IBM, C=US
Issuer: CN=user1, O=IBM, C=US
Serial number: 5bc5192c
Valid from: 7/1/22 1:52 PM until: 7/1/23 1:52 PM
Certificate fingerprints:
    MD5: BC:26:9F:73:A6:31:54:DD:17:BC:F8:BA:1B:28:46:63
    SHA1: D4:62:23:4F:59:AB:E7:1C:59:74:2E:15:77:19:1D:4A:C7:F4:D1:6F
    SHA256: 6B:49:8A:4B:62:B9:11:EC:EA:29:8B:CD:56:6E:3C:B4:80:50:2E:43:1F:64:BE:D5:93:A1:02:9B:76:B0:E3:20
    Signature algorithm name: SHA256withRSA
    Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 76 40 fc 00 ad fc 2b c5 b1 ab cc 9e 4a 34 fc 9b v.....J4...
0010: 93 b8 e8 c4 ....
]
]
```

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

```
podman exec -it cscvinc_zosConnect_1 keytool -keystore /output/resources/security/cicsKeyStore.jks -storetype PKCS12 -certreq -alias cicsusr -file /output/resources/security/user1.arm
```

Send the certificate request file to the certificate authority for signing.

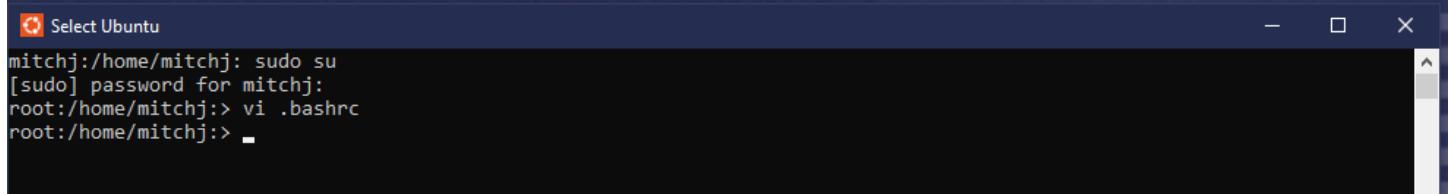
4. Import the signed personal certificate into the local keystore

```
podman exec -it cscvinc_zosConnect_1 keytool -importcert -file /output/resources/security/user1.PEM -alias cicsusr -storetype PKCS12 --noprompt -keystore /output/resources/security/cicsKeyStore.jks
```

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

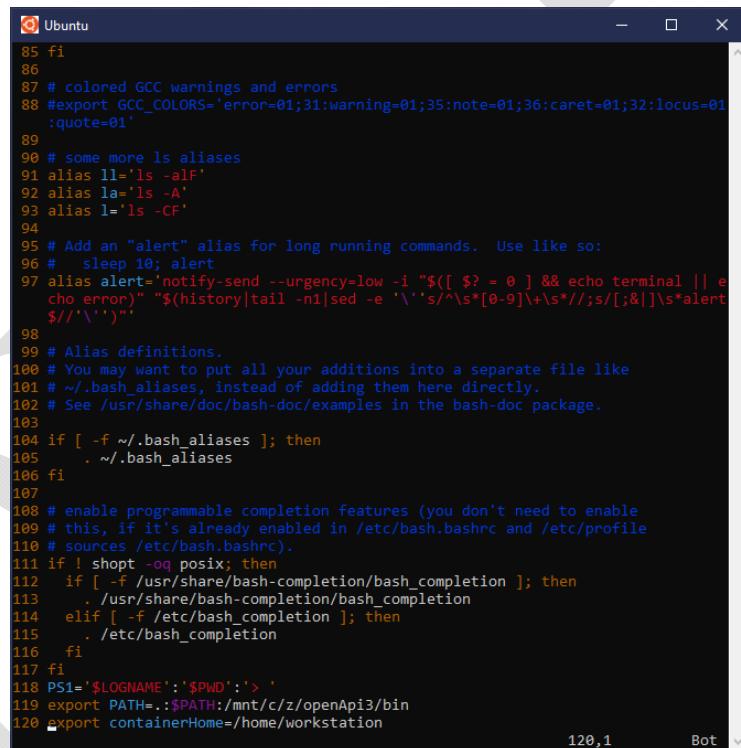
```
PS1='${LOGNAME}:\$PWD':'>
export PATH=.:$PATH:/mnt/c/z/openApi3/bin
export containerHome=/home/workstation
```



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
```



```
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
:quote=01'
89
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 "$( [ $? = 0 ] && echo terminal || e
cho error)" "$(history|tail -n1|sed -e '\''s/^[\s*][\s*\w*]*[\s*//;s/[;:&|]\s*alert
$//'\''')"
98
99 # Alias definitions.
100 # You may want to put all your additions into a separate file like
101 # ~/.bash_aliases, instead of adding them here directly.
102 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
103
104 if [ -f ~/.bash_aliases ]; then
105     . ~/.bash_aliases
106 fi
107
108 # enable programmable completion features (you don't need to enable
109 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
110 # sources /etc/bash.bashrc).
111 if ! shopt -q posix; then
112     if [ -f /usr/share/bash-completion/bash_completion ]; then
113         . /usr/share/bash-completion/bash_completion
114     elif [ -f /etc/bash_completion ]; then
115         . /etc/bash_completion
116     fi
117 fi
118 PS1='${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
`podman-compose -f /home/mitchj/podman/sandbox/docker-compose.yaml up -d`
- Start a new container using *docker-compose-yaml* while in directory /home/mitchj/podman/sandbox
`podman-compose up -d`
- Stop the container using podman-compose command while in directory /home/mitchj/podman/sandbox
`podman-compose down`
- Start the sandbox container regardless of current directory
`podman start sandbox-zosConnect-1`
- Stop the sandbox container regardless of current directory
`podman stop sandbox-zosConnect-1`
- Copy server XML override files from Linux into a container directory*
`podman cp /home/mitchj/xml/.sandbox_zosConnect_1:/config/configDropins/overrides`
- List the active containers
`podman 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/run..."</code>	26 hours ago	Up 26 hours
<code>0.0.0.0:9088->9080/tcp, :::9088->9080/tcp, 0.0.0.0:9429->9443/tcp, :::9429->9443/tcp</code>			<code>cscvinc_zosConnect_1</code>	
<code>642f17a4063a</code>	<code>icr.io/zosconnect/ibm-zcon-designer:3.0.55</code>	<code>"/opt/ibm/helpers/run..."</code>	47 hours ago	Up 20 hours
<code>0.0.0.0:9082->9080/tcp, :::9082->9080/tcp, 0.0.0.0:9445->9443/tcp, :::9445->9443/tcp</code>			<code>sandbox_zosConnect_1</code>	

- List all active and stopped containers
`podman ps -a`
- Remove a container by name or container ID
`podman rm sandbox_zosconnect_1`
or
`podman rm 642f17a4063a`
- Start a bash shell in the container
`podman exec -it sandbox_zosConnect_1 bash`
- List the installed images
`podman 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

podman 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

***podman cp sandbox_zosConnect_1:/workspace/project/build/libs/api.war
/home/mitchj/wars/cscvinc.war***

- 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 Podman image to a file
- Copy the z/OS Connect Podman 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 Podman image on the Linux image

podman save icr.io/zosconnect/ibm-zcon-designer:3.0.57 | gzip > 3.0.57.tar.gz

The contents of the cscvinc.yaml file

```
openapi: 3.0.0
info:
  description: "CICS Employee Sample VSAM Application"
  version: 1.0.0
  title: Employee
x-ibm-zcon-roles-allowed:
- Manager
paths:
  /employee:
    post:
      tags:
        - Employee
      operationId: postEmployeeInsertService
      parameters:
        - name: Authorization
          in: header
          required: false
          schema:
            type: string
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/postEmployeeInsertService_request"
        description: request body
        required: true
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postEmployeeInsertService_response_200"
        "400":
          description: Bad Request
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postEmployeeInsertService_response_400"
        "500":
          description: Sever Error
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postEmployeeInsertService_response_500"
```

```
"/employee/{employee}":
get:
tags:
- Employee
operationId: getEmployeeSelectService
x-ibm-zcon-roles-allowed:
- Staff
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/getEmployeeSelectService_response_200"
"404":
description: Not Found
content:
  application/json:
    schema:
      $ref: "#/components/schemas/getEmployeeSelectService_response_404"
"500":
description: Severe Error
content:
  application/json:
    schema:
      $ref: "#/components/schemas/getEmployeeSelectService_response_500"
put:
tags:
- Employee
operationId: putEmployeeUpdateService
x-ibm-zcon-roles-allowed:
- Staff
parameters:
- name: Authorization
  in: header
  required: false
  schema:
    type: string
```

```

- name: employee
  in: path
  required: true
  schema:
    type: string
requestBody:
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/putEmployeeUpdateService_request"
description: request body
required: true
responses:
  "200":
    description: OK
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/putEmployeeUpdateService_response_200"
  "404":
    description: Bad Request
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/putEmployeeUpdateService_response_404"
  "500":
    description: Severe Error
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/putEmployeeUpdateService_response_500"
delete:
  tags:
    - Employee
  operationId: deleteEmployeeDeleteService
  x-ibm-zcon-roles-allowed:
    - Manager
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/deleteEmployeeDeleteService_response_200"  
"404":  
  description: Not Found  
  content:  
    application/json:  
      schema:  
        $ref: "#/components/schemas/deleteEmployeeDeleteService_response_404"  
"500":  
  description: Not Found  
  content:  
    application/json:  
      schema:  
        $ref: "#/components/schemas/deleteEmployeeDeleteService_response_500"  
servers:  
  - url: /  
components:  
schemas:  
  getEmployeeSelectService_response_404:  
    type: object  
    properties:  
      message:  
        type: string  
    example:  
      message: Record could not be found  
  getEmployeeSelectService_response_200:  
    type: object  
    properties:  
      summary:  
        $ref: '#/components/schemas/getEmployeeSelectService_response_200_message'  
      detail:  
        $ref: '#/components/schemas/getEmployeeSelectService_response_200_detail'  
  getEmployeeSelectService_response_200_message:  
    type: object  
    properties:  
      message:  
        type: string  
    example:  
      message: record retrieved  
  getEmployeeSelectService_response_200_detail:  
    type: object  
    properties:  
      EmployeeSelectServiceOperationResponse:  
        type: object  
        properties:  
          employeeData:
```

```
type: object
properties:
  response:
    type: object
    properties:
      employeeDetails:
        type: object
        properties:
          employeeNumber:
            type: string
            maxLength: 6
          name:
            type: string
            maxLength: 20
          address:
            type: string
            maxLength: 20
          phoneNumber:
            type: string
            maxLength: 8
          date:
            type: string
            maxLength: 8
          amount:
            type: string
            maxLength: 8
          comment:
            type: string
            maxLength: 9
required:
  - employeeData
getEmployeeSelectService_response_500:
  type: object
  properties:
    message:
      type: string
example:
  message: A severe error has occurred
deleteEmployeeDeleteService_response_404:
  type: object
  properties:
    message:
      type: string
example:
  message: Record could not be found to be deleted
deleteEmployeeDeleteService_response_500:
  type: object
  properties:
    message:
      type: string
```

example:

 message: A severe error has occurred

deleteEmployeeDeleteService_response_200:

 type: object

 properties:

 message:

 type: string

example:

 message: Record deleted

putEmployeeUpdateService_request:

 type: object

 properties:

 EmployeeUpdateServiceOperation:

 type: object

 properties:

 employeeData:

 type: object

 properties:

 request:

 type: object

 properties:

 employeeDetails:

 type: object

 properties:

 status:

 type: string

 maxLength: 1

 employeeNumber:

 type: string

 maxLength: 6

 name:

 type: string

 maxLength: 20

 address:

 type: string

 maxLength: 20

 phoneNumber:

 type: string

 maxLength: 8

 date:

 type: string

 maxLength: 8

 amount:

 type: string

 maxLength: 8

 COMMENT:

 type: string

 maxLength: 9

required:

 - employeeData

putEmployeeUpdateService_response_404:

type: object

properties:

message:

type: string

example:

message: Record update failed

putEmployeeUpdateService_response_500:

type: object

properties:

message:

type: string

example:

message: A severe has occurred

putEmployeeUpdateService_response_200:

type: object

properties:

message:

type: string

example:

message: Record updated

postEmployeeInsertService_request:

type: object

properties:

EmployeeInsertServiceOperation:

type: object

properties:

employeeData:

type: object

properties:

request:

type: object

properties:

employeeDetails:

type: object

properties:

employeeNumber:

type: string

maxLength: 6

name:

type: string

maxLength: 20

address:

type: string

maxLength: 20

phoneNumber:

type: string

maxLength: 8

date:

type: string

```
maxLength: 8
amount:
  type: string
  maxLength: 8
required:
- employeeData
postEmployeeInsertService_response_400:
type: object
properties:
  message:
    type: string
example:
  message: Employee insert failed
postEmployeeInsertService_response_500:
type: object
properties:
  message:
    type: string
example:
  message: Severe Error Occurred
postEmployeeInsertService_response_200:
type: object
properties:
  message:
    type: string
example:
  message: Record successfully inserted
```