IBM – CICS and DevOps Workshop

Lab – z/OS Connect OAS3 API for CICS applications

Lab Version V1.1

Oct 2022

Please send any comments on this lab exercise to:
George GE
ypge@au1.ibm.com

Overview

The use of REST APIs for z/OS has matured as a key mechanism for modernizing and unlocking the value of existing applications and data. This maturity has grown the number and diversity of development teams engaging with the mainframe. This requires a development and deployment model that supports agility and enables self-service provisioning.

IBM z/OS Connect now includes cloud-native development support and API first mapping for creating OpenAPI 3 interfaces to z/OS applications and data. To achieve this, we have added two new components; a new container-based deployment model that is known as the IBM z/OS Connect Server and a powerful new browser-based tooling that is known as the z/OS Connect Designer.

Different from the previous Eclipse-based API toolkit for OpenAPI2 (or the “swagger” APIs), the latest z/OS Connect Designer is a container image that’s freely available on the IBM container registry (icr.io). It also adopts the top-down approach, which means you will start with an OpenAPI3 specification document and work your way down, importing your existing z/OS backend service data structure and then using the powerful mapping capability to meet the two in the middle.

Scenario

In this lab scenario, you are an API developer. You are tasked to complete the implementation of a z/OS API that exposes the business functions of a backend CICS application called “catalog manger”. This will allow the front-end cloud application to use standard RESTful API to look up the items in the catalogue, inquire about the detail of any item and place an order.

The API interface has been set in the specification, so you will need to complete additional mapping to match it with the copybook structure of the CICS program. Also, the API project is hosted in an enterprise unified Git-based repository on GitLab, you will explore the DevOps pipeline that is used to drive the build, deploy and test the API once the changes are committed.

Lab environment setup

In our lab, we have set up the z/OS Connect Designer container on the Linux development server which you can access via the browser on the Windows desktop by navigating to this URL:

```
https://workshop.dev:9443/zosConnect/designer
```

It is running in a docker container with the API project directory mounted into it. The API project is in the Linux local file system, as API developers make changes in the z/OS Connect Designer, these changes are updated on the file system in the API Project directory. These files are the source code for your API and are tracked by Git, with its remote origin pointing to a GitLab repo.

Lab Step Overview

Part 1: Explore the CICS catalog manager application

In this step, you will log in to the target CICS region using a 3270 emulator to test the Catalog Manager application that is to be API enabled. Catalog Manager is a CICS-supplied sample and traditionally uses 3270 BMS interface as the main interface.

Part 2: Log in to the z/OS Connect Designer

In this lab exercise, you will use the browser to open the z/OS Connect Designer and get familiar with the Designer interface and project layout.

Part 3: Define the operation & basic mapping for the “inquiry single item” service.

In this step, you will start working on implementing the mappings for the “inquiry single item” service using the z/OS Connect Designer in the browser.

Part 4: Test the API using the built-in OpenAPI3 testing tool

In this part of the lab, you will use the built-in Liberty OpenAPI testing tool to perform API testing in your development environment to validate the API call and examine the response payload from the z/OS CICS program.

Part 5: Fine-tune the API data format using advanced mapping capability.

In this part of the lab, you've done the initial test and noticed that additional work is needed to format the returned data from the CICS program to meet the requirement. You then make the changes in the Designer and then test it again to confirm it's now complete.

Part 6: Commit and push the API project to the Git repository

In this part of the lab exercise, you're confident all the changes made is satisfactory and ready to be committed to the remote Git server.

Part 7: Explore the DevOps pipeline that automates the build, deploy and testing

In this part of the lab exercise, you will review the steps and results of the automated CI/CD pipeline that's triggered by the commit and ensure now the new version of the API is passing the build, deploy and test stages.

Part 8: Summary

This is a recap of the steps performed in this lab exercise.

Part 1: Explore the CICS catalog manager application

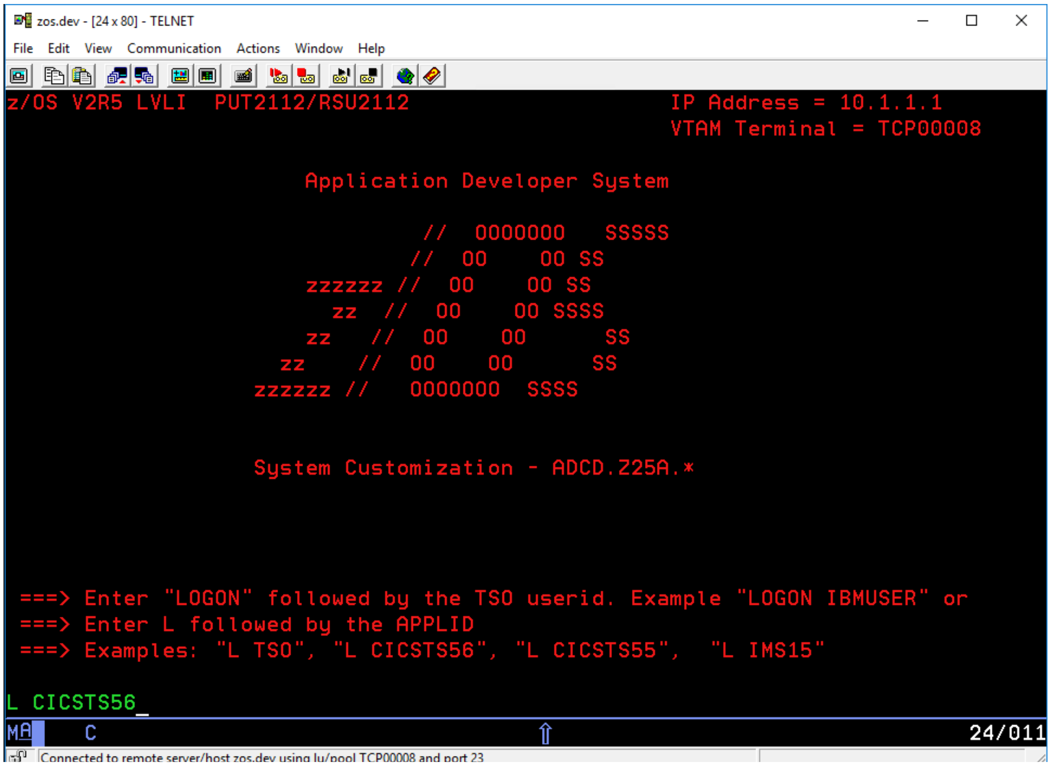
In this step, you will log in to the target CICS region using a 3270 emulator to review the Catalog Manager application that is to be API enabled. Catalog Manager is a CICS-supplied sample and traditionally uses 3270 BMS as the user interface.

Under its cover, the presentation logic drives `EXEC CICS LINK` to a program called `DFH0XCMN` and by passing a COMMAREA to drive several business logics for inquiry catalogue, look up single item detail and place an order.

Start the Personal Communication emulator and log into CICS

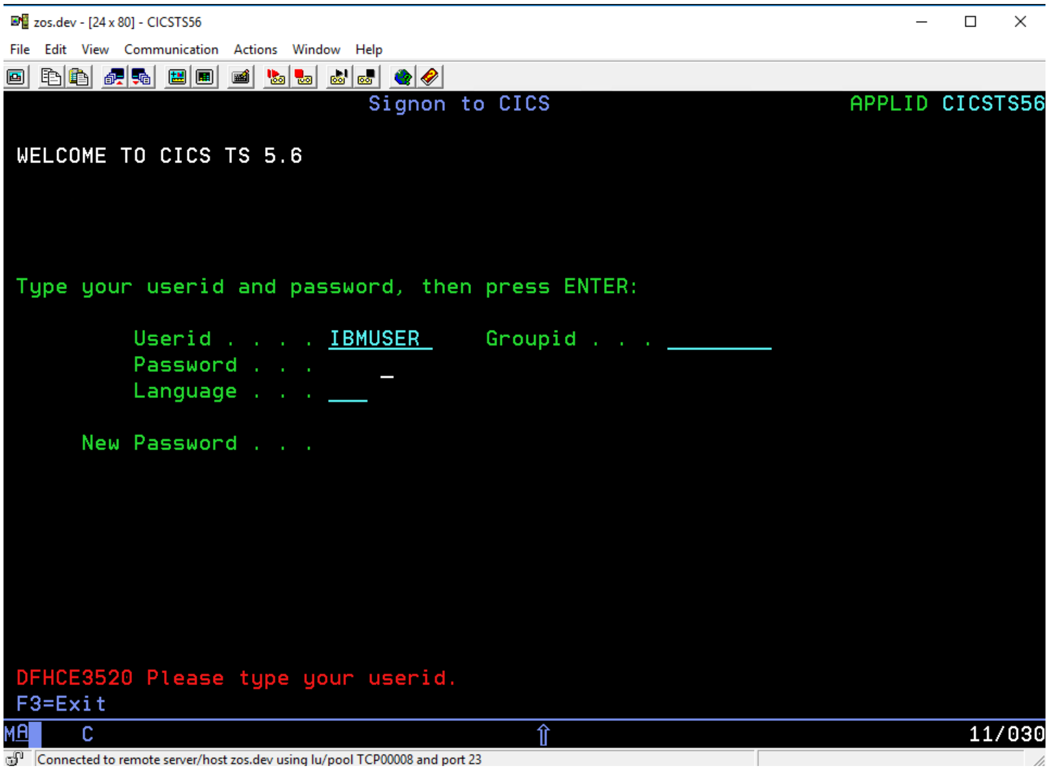
___**1.** From the desktop, **double-click** the Personal Communication icon to start PCOMM if it is not already running.

___**2.** When you start the PCOMM, type `L CICSTS56` then press **Enter** on the initial screen to log into CICS.


zos.dev - [24 x 80] - TELNET
File Edit View Communication Actions Window Help

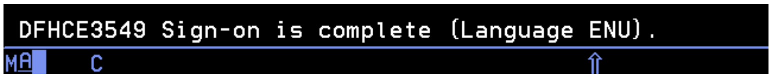
```
z/OS V2R5 LVLI  PUT2112/RSU2112                           IP Address = 10.1.1.1
                                                          VTAM Terminal = TCP00008

                       Application Developer System

                                //  OOOOOOO   SSSSS
                               //  OO    OO SS
                       zzzzzz //  OO    OO SS
                         zz  //  OO    OO SSSS
                       zz   //  OO    OO      SS
                     zz    //  OO    OO      SS
                   zzzzzz //   OOOOOOO  SSSS


                   System Customization - ADCD.Z25A.*



 ===> Enter "LOGON" followed by the TSO userid. Example "LOGON IBMUSER" or
 ===> Enter L followed by the APPLID
 ===> Examples: "L TSO", "L CICSTS56", "L CICSTS55",  "L IMS15"

L CICSTS56_
MA     C                                                                 24/011
Connected to remote server/host zos.dev using lu/pool TCP00008 and port 23
```

___**3.** You will be presented with the CICS login screen, type `IBMUSER` and `SYS1` as the userid and password and press **Enter** to log in. Please note that the password will not be shown on the screen.


zos.dev - [24 x 80] - CICSTS56
File Edit View Communication Actions Window Help

```
                               Signon to CICS                     APPLID CICSTS56

 WELCOME TO CICS TS 5.6




 Type your userid and password, then press ENTER:

          Userid . . . . IBMUSER     Groupid . . . ________
          Password . . .    _
          Language . . . ___

      New Password . . .







 DFHCE3520 Please type your userid.
 F3=Exit
MA     C                                                                 11/030
Connected to remote server/host zos.dev using lu/pool TCP00008 and port 23
```

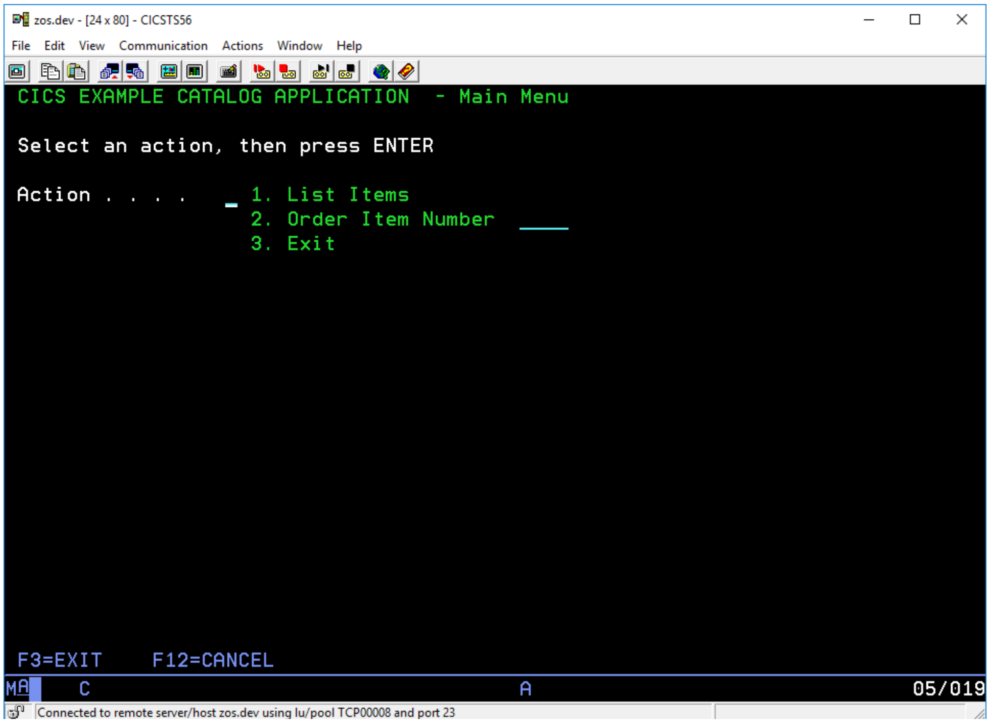
___**4.** If you log in successfully, the message at the bottom of the screen indicates you're now ready to start transactions from the CICS terminal.



```
DFHCE3549 Sign-on is complete (Language ENU).
MA     C                                    ⇧
```

Start the catalog manager EGUI transaction

___**5.** In the CICS 3270 emulator session, enter the transaction id `EGUI` and press **Enter**. The application menu is displayed:

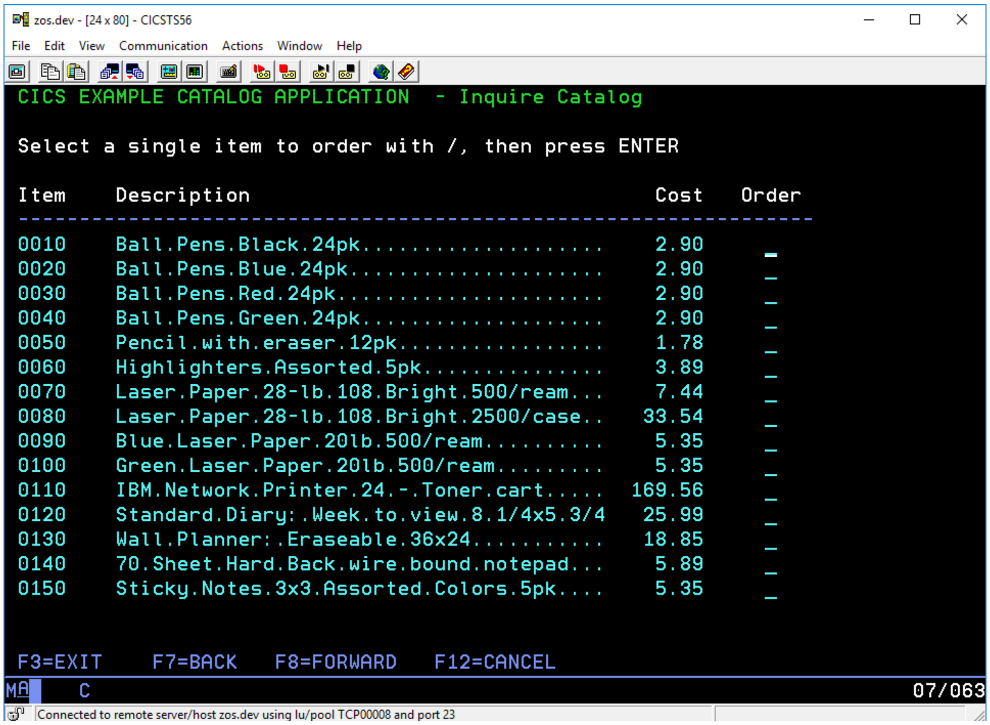


zos.dev - [24 x 80] - CICSTS56
File Edit View Communication Actions Window Help

```
CICS EXAMPLE CATALOG APPLICATION  - Main Menu

Select an action, then press ENTER

Action . . . .   _ 1. List Items
                   2. Order Item Number  ____
                   3. Exit












F3=EXIT     F12=CANCEL
MA     C                                  A                        05/019
```

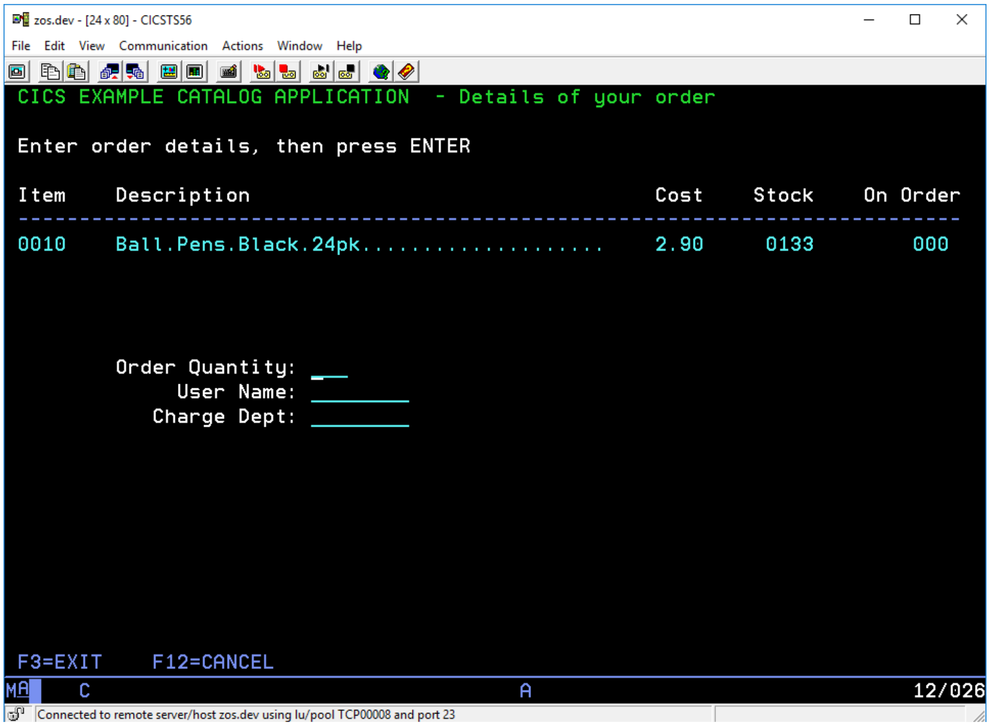
Connected to remote server/host zos.dev using lu/pool TCP00008 and port 23

___**6.** Type `1` and press **Enter** to select the **List Items** option. The application displays a list of items in the catalog.


zos.dev - [24 x 80] - CICSTS56
File Edit View Communication Actions Window Help

```
CICS EXAMPLE CATALOG APPLICATION  - Inquire Catalog

Select a single item to order with /, then press ENTER

Item    Description                                 Cost   Order
-----------------------------------------------------------------
0010    Ball.Pens.Black.24pk....................    2.90     _
0020    Ball.Pens.Blue.24pk.....................    2.90     _
0030    Ball.Pens.Red.24pk......................    2.90     _
0040    Ball.Pens.Green.24pk....................    2.90     _
0050    Pencil.with.eraser.12pk.................    1.78     _
0060    Highlighters.Assorted.5pk...............    3.89     _
0070    Laser.Paper.28-lb.108.Bright.500/ream...    7.44     _
0080    Laser.Paper.28-lb.108.Bright.2500/case..   33.54     _
0090    Blue.Laser.Paper.20lb.500/ream..........    5.35     _
0100    Green.Laser.Paper.20lb.500/ream.........    5.35     _
0110    IBM.Network.Printer.24.-.Toner.cart.....  169.56     _
0120    Standard.Diary:.Week.to.view.8.1/4x5.3/4   25.99     _
0130    Wall.Planner:.Eraseable.36x24...........   18.85     _
0140    70.Sheet.Hard.Back.wire.bound.notepad...    5.89     _
0150    Sticky.Notes.3x3.Assorted.Colors.5pk....    5.35     _


F3=EXIT    F7=BACK    F8=FORWARD    F12=CANCEL
MA     C                                                                07/063
Connected to remote server/host zos.dev using lu/pool TCP00008 and port 23
```

___**7.** Type `/` in the **Order** column next to an item and press **Enter** to place an order for that item. For example, if you selected item `0010`, the following screen is displayed.


zos.dev - [24 x 80] - CICSTS56
File Edit View Communication Actions Window Help

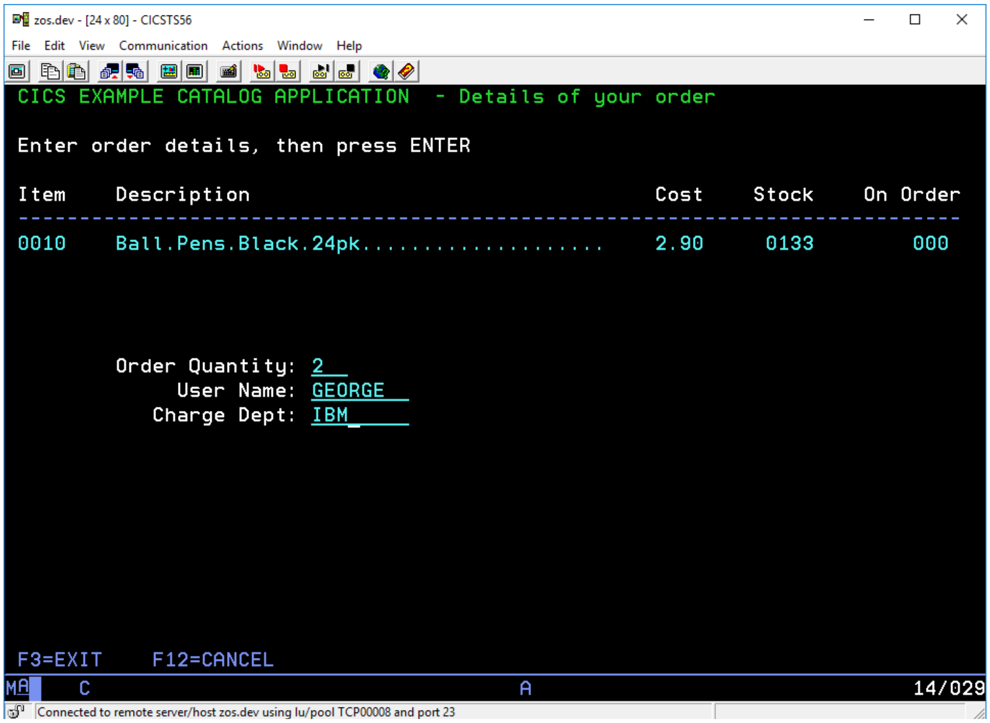
```
CICS EXAMPLE CATALOG APPLICATION  - Details of your order

Enter order details, then press ENTER

Item    Description                                 Cost    Stock    On Order
-----------------------------------------------------------------------------
0010    Ball.Pens.Black.24pk....................    2.90     0133         000




        Order Quantity: ___
             User Name: ________
           Charge Dept: ________









F3=EXIT    F12=CANCEL
MA     C                                     A                          12/026
Connected to remote server/host zos.dev using lu/pool TCP00008 and port 23
```

___**8.** If stock levels are sufficient to fulfil the order, enter the following information:

a) In the **Order Quantity** field, specify the number of items you want to order, e.g.`1`

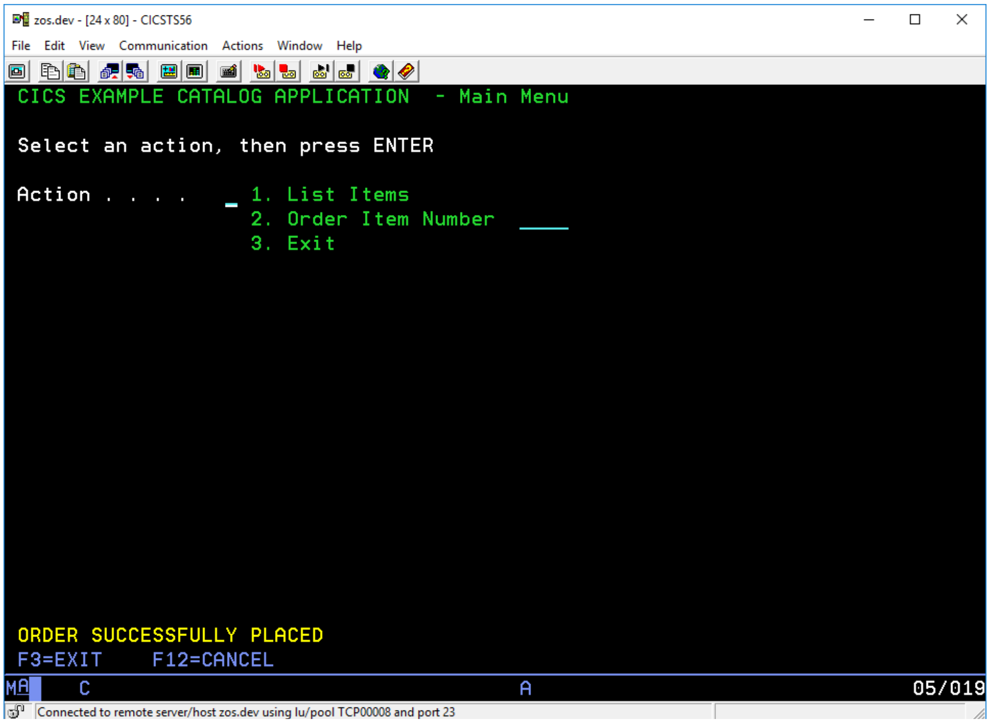
b) In the **User Name** field, enter a 1-to 8-character string. The sample application does not check the value that is entered here.

c) In the **Charge Dept** field, enter a 1-to 8-character string. The sample application does not check the value that is entered here.


zos.dev - [24 x 80] - CICSTS56
File Edit View Communication Actions Window Help

```
CICS EXAMPLE CATALOG APPLICATION  - Details of your order

Enter order details, then press ENTER

Item     Description                                  Cost     Stock     On Order
-----------------------------------------------------------------------------
0010     Ball.Pens.Black.24pk...................      2.90      0133          000




         Order Quantity: 2
              User Name: GEORGE
            Charge Dept: IBM










F3=EXIT     F12=CANCEL
MA     C                                    A                                14/029
Connected to remote server/host zos.dev using lu/pool TCP00008 and port 23
```

____**9.** Press **Enter** to submit the order and return to the main menu.


zos.dev - [24 x 80] - CICSTS56
File Edit View Communication Actions Window Help

```
CICS EXAMPLE CATALOG APPLICATION  - Main Menu

Select an action, then press ENTER

Action . . . .   _ 1. List Items
                   2. Order Item Number ____
                   3. Exit











ORDER SUCCESSFULLY PLACED
F3=EXIT     F12=CANCEL
MA     C                                     A                        05/019
```

Connected to remote server/host zos.dev using lu/pool TCP00008 and port 23

___**10.** Press **F3** to end the application.

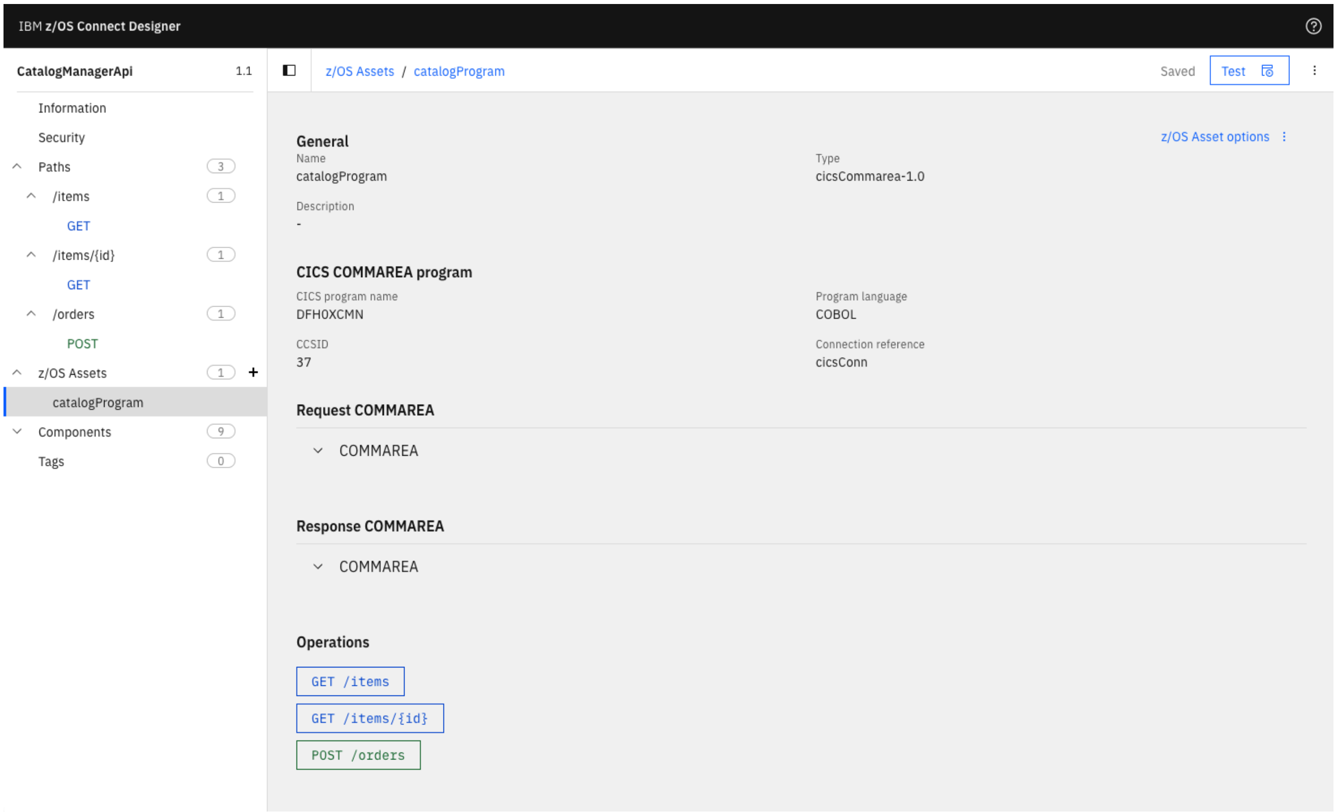
The CICS Catalog Manager sample application is successfully installed and configured, and you can use this sample with the CICS scenarios that are described in this documentation.

Part 2: Log in to the z/OS Connect Designer

Developing APIs with the z/OS Connect Designer.

The z/OS Connect Designer is a container-based Web UI tool that offers a wide set of capabilities:

- Support for enterprise standard OpenAPI 3.0 specification APIs.
- Allows contract first API creation by importing an OpenAPI 3 definition as a starting point.
- Supports JSONata, a functional expression language for complex mappings between the API schemas and the z/OS application and data interfaces.
- Supports the creation of OpenAPI 3 definitions.
- Includes a built-in isolated development server for iterative development and testing.



General

Name: catalogProgram

Type: cicsCommarea-1.0

Description: -

CICS COMMAREA program

CICS program name: DFH0XCMN

Program language: COBOL

CCSID: 37

Connection reference: cicsConn

Request COMMAREA

COMMAREA

Response COMMAREA

COMMAREA

Operations

`GET /items`

`GET /items/{id}`

`POST /orders`

Note:

The Designer image is used by an application developer to build and test the z/OS Connect API project. When this image runs in a container runtime such as Docker or Docker Desktop for Windows, A developer can use the z/OS Connect Designer in a web browser to work on the API project. When finished, check the project files into a source code management tool like Git, and subsequentially kick off the build process to build the API as a .war file. z/OS Connect Designer image is supported with any OCI-compliant container platforms like Docker and is currently available on amd64 (x86_64) architecture.

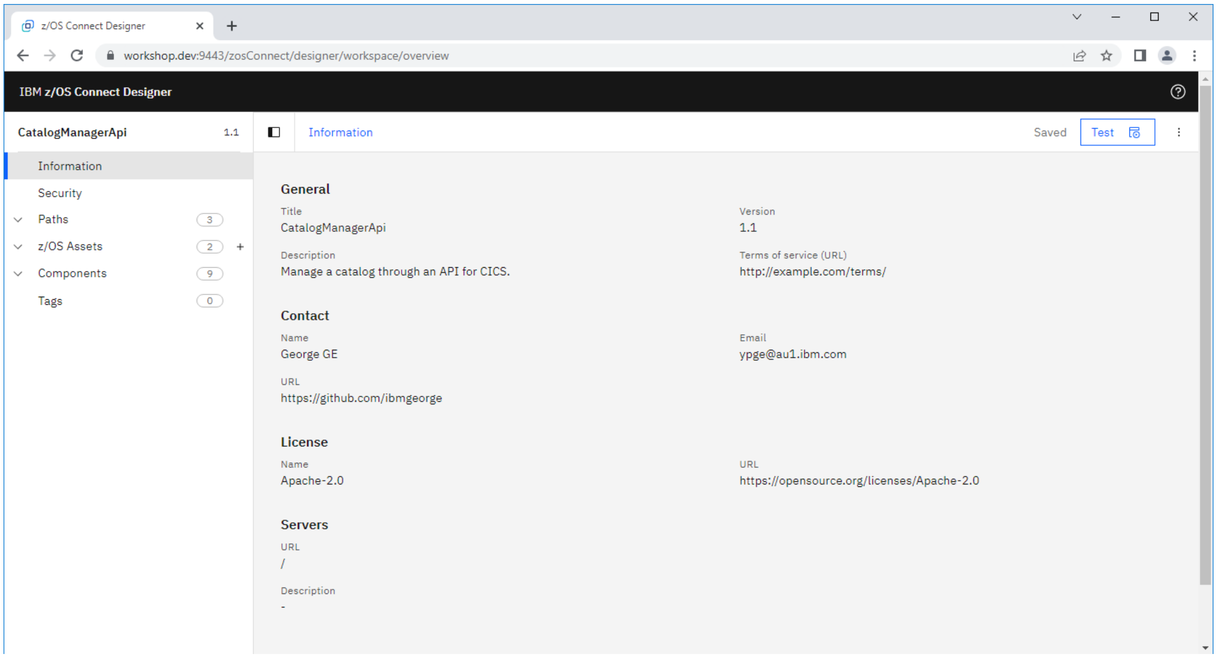
Logging in to the z/OS Connect Designer using the browser

___**1.** Double-click the Chrome browser icon on the desktop if it's not open.

___**2.** From the **Chrome** browser, click on the z/OS Connect Designer from the bookmark bar at the top or navigate to:

https://workshop.dev:9443/zosConnect/designer/workspace/overview

___**3.** The z/OS Connect Designer interface is now open.



General

Title
CatalogManagerApi

Version
1.1

Description
Manage a catalog through an API for CICS.

Terms of service (URL)
http://example.com/terms/

Contact

Name
George GE

Email
ypge@au1.ibm.com

URL
https://github.com/ibmgeorge

License

Name
Apache-2.0

URL
https://opensource.org/licenses/Apache-2.0

Servers

URL
/

Description
-

Navigate the Designer interface

___**4.** The **Information** tab on the left shows some of the basic information about the API defined in the specification.

___**5.** The **Security** tab shows the required authentication and authorization model for the API.

___**6.** **Paths** list all the API paths and associated operations defined in the specification. The one with the icon ✓ Ready indicates that the mapping for that operation is completed.

___**7.** Expand the **/items** then click on the **GET** operation will show mapping details for the "inquiry catalog" operation which has been completed as part of the sample.

___**8.** Expand the **z/OS Assets** to show the backend z/OS assets. We will create another z/OS asset representing the target CICS program later in the lab.

Note:

A z/OS asset is a representation of IBM Z service or data from a z/OS subsystem such as CICS and DB2 that you can expose as an API by using z/OS Connect Designer. We will be adding other subsystem for IMS and MQ etc. in the near future thru continuous delivery of z/OS Connect.

___**9.** Under **Components**, it lists all the artifacts in the z/OS Connect API projects.

Note:

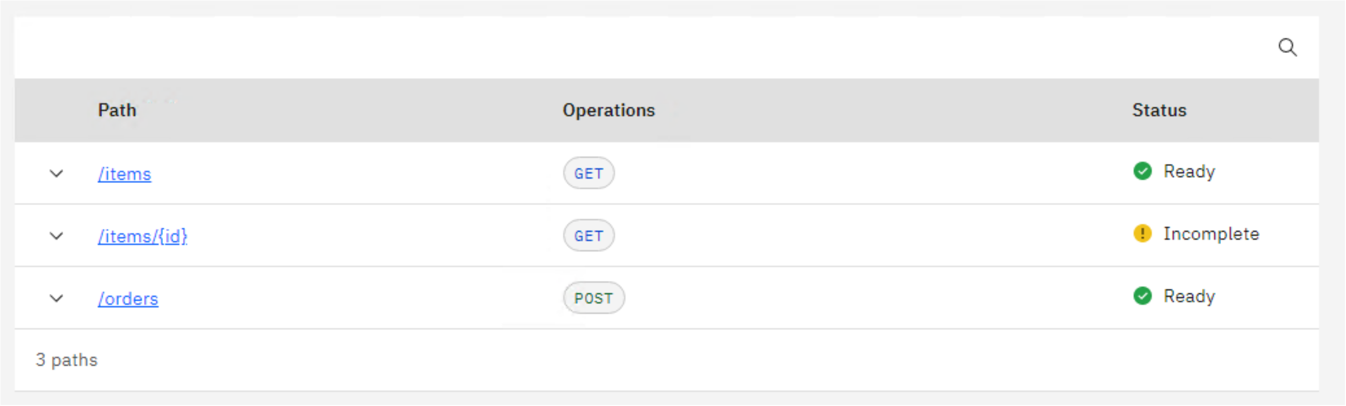
Since the API project has been cloned to your workspace on the Linux development, you will see that the detail of the API specification has already been imported, and some of the Paths has been completed.

Part 3: Define the operation & basic mapping for inquiry single item service.

As part of the lab scenario, the API project has two operations pre-configured and mapped already, we will be implementing the **/items/{id} GET** operation for the "inquiry single item" detail API in this part.

Review the API request definition

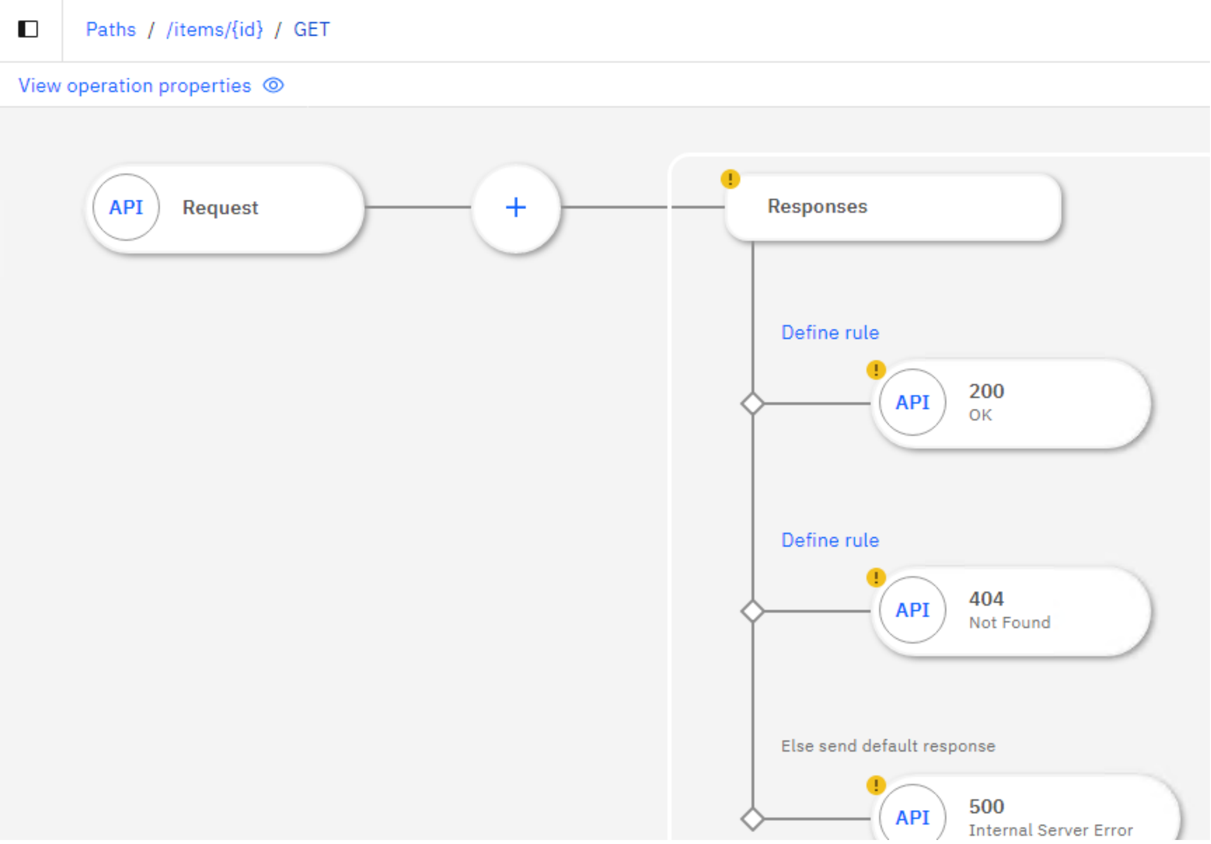
___**1.** From **the z/OS Connect Designer**, click **Paths** on the left, it shows both the **/items** and **/orders** operations are Ready, with the **/items/{id} GET** operation is incomplete.



Path	Operations	Status
/items	GET	✓ Ready
/items/{id}	GET	! Incomplete
/orders	POST	✓ Ready

3 paths

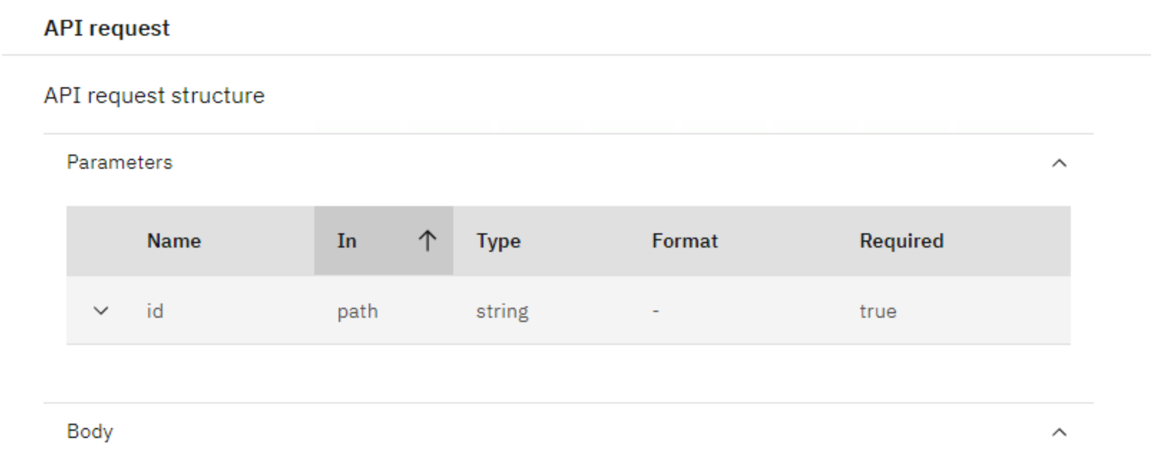
___**2.** Click the **down arrow** of this path and click **GET** operation button to open the mapping editor.



Paths / /items/{id} / GET
View operation properties
API Request
+
Responses
Define rule
API 200 OK
Define rule
API 404 Not Found
Else send default response
API 500 Internal Server Error

___**3.** Click the **Request** node, and you will be able to see what is expected on incoming API request defined in the API specification. In this example, the API client is expected to pass in a mandatory **id** parameter as a string in the path, which can then be used to build the input to the z/OS asset. There's no **body** payload required as this is typical with http GET operation.

So this API request should be a HTTP GET of the URI:
`https://hostname:ip/basepath/items/nnn` (nnn is the id of the item)



API request

API request structure

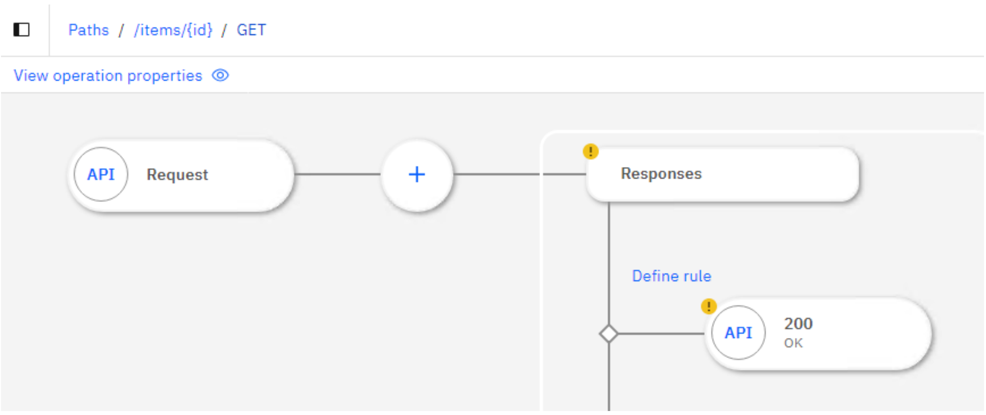
Parameters

	Name	In	Type	Format	Required
⌄	id	path	string	-	true

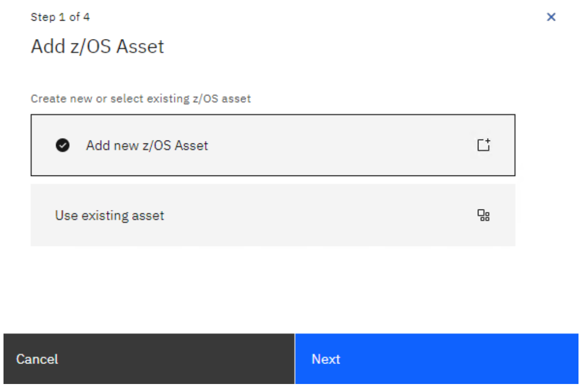
Body

Creating a new z/OS asset

___**4.** Now click the + button next to the **Request** node to configure the z/OS asset for this API request.



___**5.** It opens a wizard for the configuring the z/OS Asset, choose **Add new z/OS Asset** and click **Next**.



Step 1 of 4

Add z/OS Asset

Create new or select existing z/OS asset

● Add new z/OS Asset

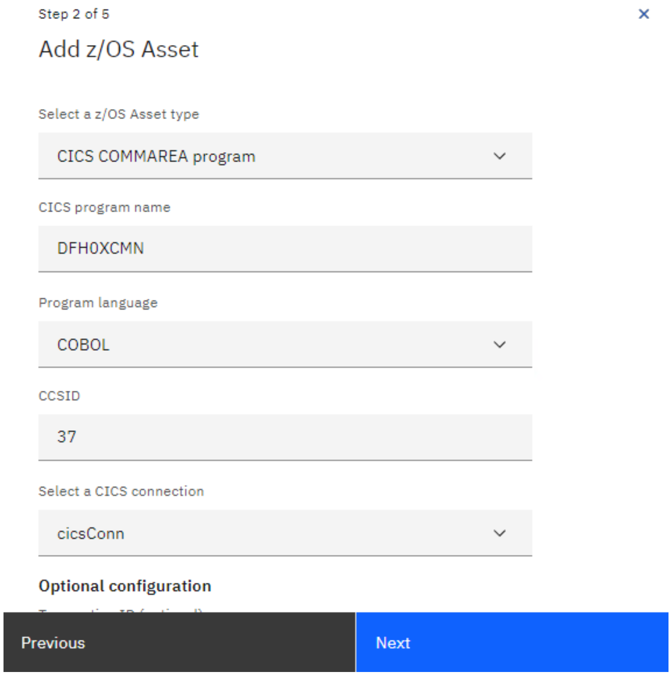
Use existing asset

___**6.** Now configure the z/OS Asset as below:

- Choose **CICS COMMAREA program** in the next screen

- Type in DFH0XCMN as the CICS program name
- Choose **COBOL** for the program language
- Specify 37 as the CCSID.
- Select the **cicsConn** from the dropdown menu for the CICS connection, which has been pre-configured in the API project that points to the backend z/OS CICS TS Server IPIC port number.

___**7.** Leave the rest of the optional configurations blank, it should look like this after you finish. Now click **Next**.


Step 2 of 5

Add z/OS Asset

Select a z/OS Asset type: CICS COMMAREA program

CICS program name: DFH0XCMN

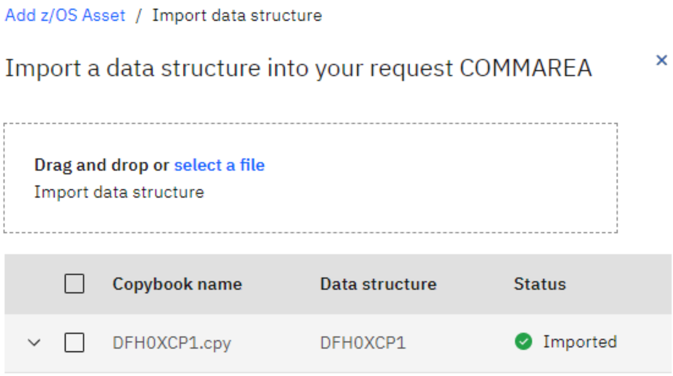
Program language: COBOL

CCSID: 37

Select a CICS connection: cicsConn

Optional configuration

___**8.** In the next screen, you can import the data structure for the CICS program, click the **Import data structure** button and select the **DFH0XCP1.cpy** file in your **Labs** folder on the desktop.


Add z/OS Asset / Import data structure

Import a data structure into your request COMMAREA

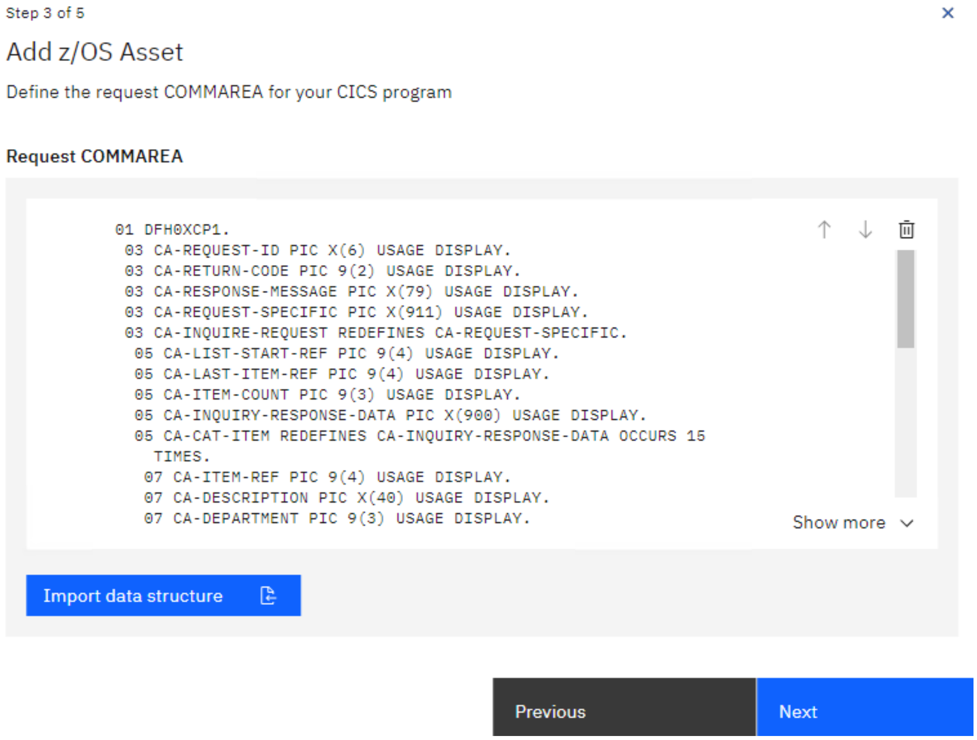
Drag and drop or select a file

Import data structure

	Copybook name	Data structure	Status
☐	DFH0XCP1.cpy	DFH0XCP1	✓ Imported

___**9.** Select the **DFH0XCP1** data structure from the **DFH0XCP1.cpy** copybook that you just imported to include it in your request COMMAREA to the CICS program. Click **Add**.

___**10.** Review the CICS Request COMMAREA then click **Next**.



Step 3 of 5

Add z/OS Asset

Define the request COMMAREA for your CICS program

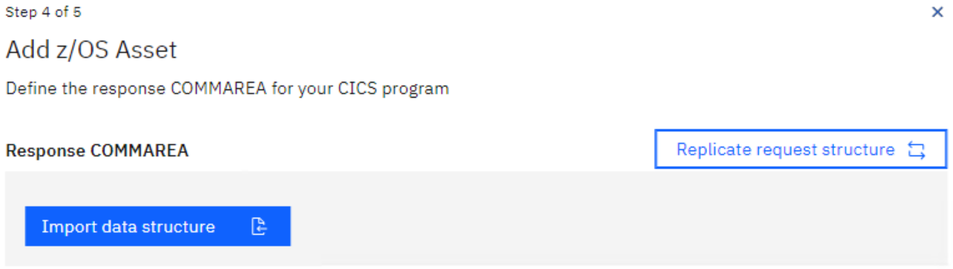
Request COMMAREA

```
01 DFH0XCP1.
 03 CA-REQUEST-ID PIC X(6) USAGE DISPLAY.
 03 CA-RETURN-CODE PIC 9(2) USAGE DISPLAY.
 03 CA-RESPONSE-MESSAGE PIC X(79) USAGE DISPLAY.
 03 CA-REQUEST-SPECIFIC PIC X(911) USAGE DISPLAY.
 03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
  05 CA-LIST-START-REF PIC 9(4) USAGE DISPLAY.
  05 CA-LAST-ITEM-REF PIC 9(4) USAGE DISPLAY.
  05 CA-ITEM-COUNT PIC 9(3) USAGE DISPLAY.
  05 CA-INQUIRY-RESPONSE-DATA PIC X(900) USAGE DISPLAY.
  05 CA-CAT-ITEM REDEFINES CA-INQUIRY-RESPONSE-DATA OCCURS 15
    TIMES.
   07 CA-ITEM-REF PIC 9(4) USAGE DISPLAY.
   07 CA-DESCRIPTION PIC X(40) USAGE DISPLAY.
   07 CA-DEPARTMENT PIC 9(3) USAGE DISPLAY.
```

Note:

Complex COBOL copybook structures are supported. For example, the use of REDEFINE to specify additional record layouts for a single block of storage area. And the use of Occurs Depending On (ODO) in COBOL to implement variable-size dynamic array, or nested array are some of the examples that are supported natively for the z/OS Asset.

___**11.** Define the response COMMAREA for the Catalog Manager **DFH0XCMN** program. Copy over your request COMMAREA structure for the response COMMAREA by clicking **Replicate request structure**, then click **Next**



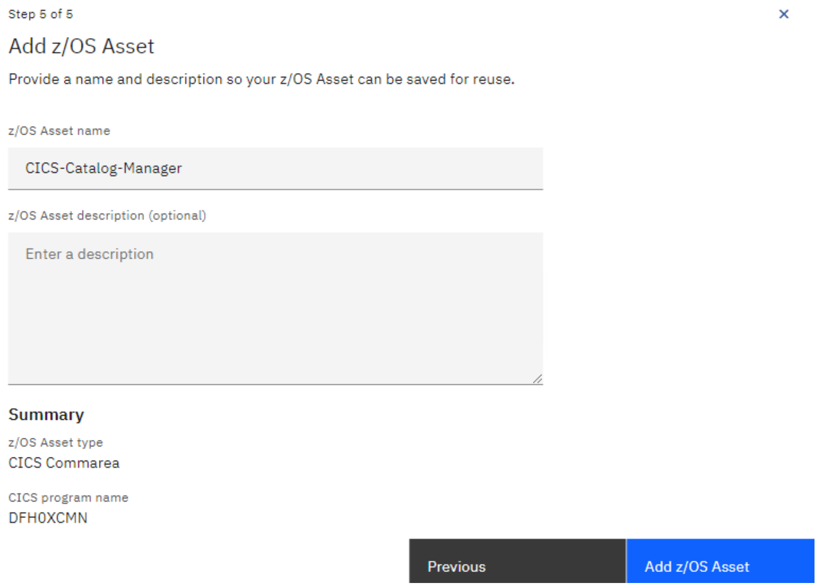
Step 4 of 5

Add z/OS Asset

Define the response COMMAREA for your CICS program

Response COMMAREA

___**12.** Enter a z/OS Asset name: `CICS-Catalog-Manager`, an optional description, review the z/OS Asset and click **Add z/OS Asset**.



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

CICS-Catalog-Manager

z/OS Asset description (optional)

Enter a description

Summary

z/OS Asset type

CICS Commarea

CICS program name

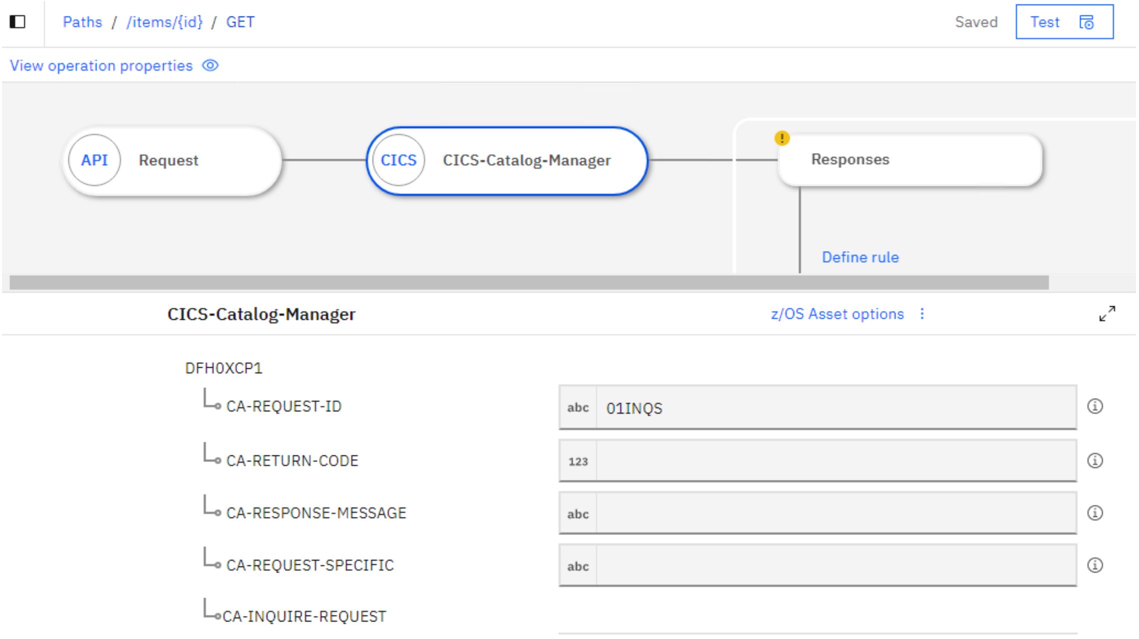
DFH0XCMN

Map the API request to the z/OS asset

In this task, you map parameters defined in the API request to fields in the z/OS Asset Request.

___13. Click the **CICS-Catalog-Manager** z/OS Asset node in the operation flow diagram. The z/OS Asset configuration window opens.

___14. Put `01INQS` into the **CA-REQUEST-ID** field, this value is case sensitive and will be populated in the input COMMAREA as required function code by the backend CICS program representing the function to look up single item details.

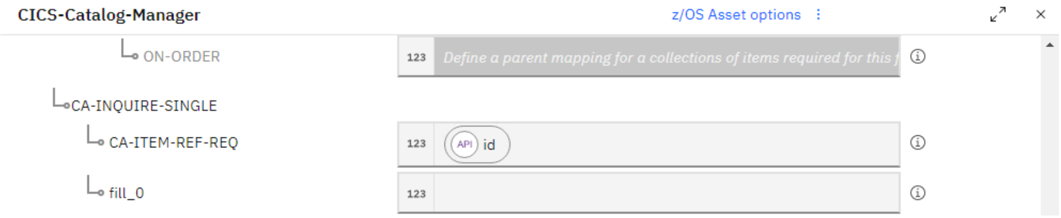


DFH0XCP1

CA-REQUEST-ID	abc	01INQS
CA-RETURN-CODE	123	
CA-RESPONSE-MESSAGE	abc	
CA-REQUEST-SPECIFIC	abc	
CA-INQUIRE-REQUEST		

___**15.** Map the API Request parameter **id** into the **CA-ITEM-REF-REQ** under the **CA-INQUIRE-SINGLE** z/OS Asset Request field. This can be done in either way.

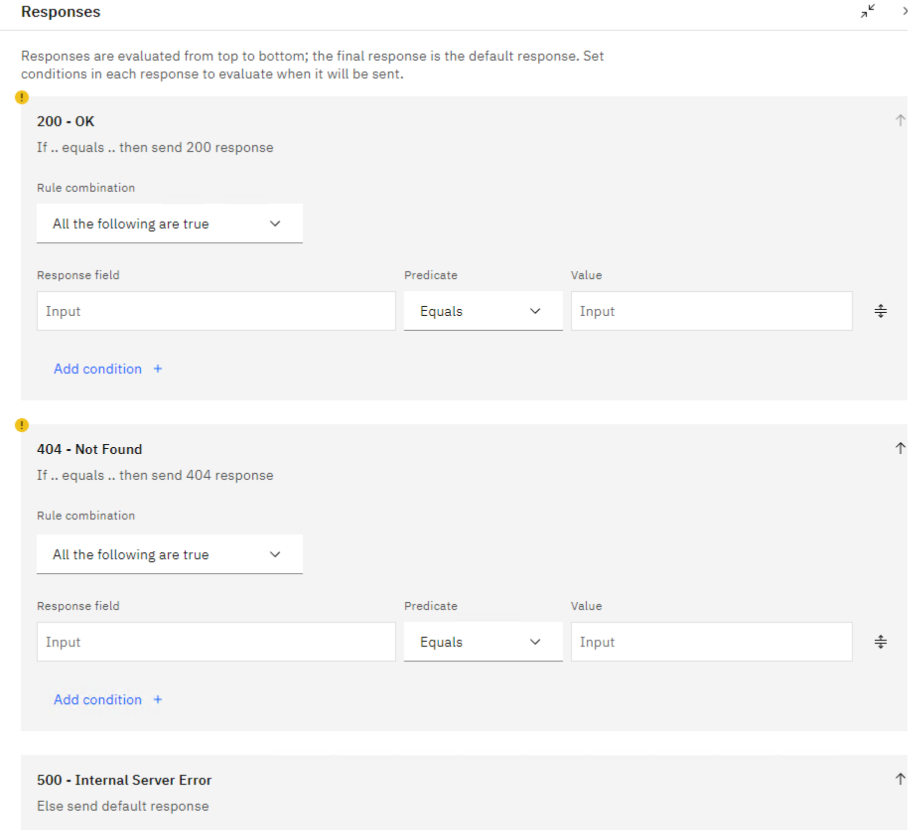
- Type id in the CA-ITEM-REF-REQ field. When you start typing, the Available Mappings menu opens with the available parameters. Select **id** from the list.
- You can also click the icon ☰ to select a path parameter from the list. When the list opens up, select the **id** path parameter under the **object**.



___**16.** Click the **X** button to exit this editing panel.

Define the response code criteria

___**17.** Click **Responses** on the operation flow diagram. The Responses configuration panel opens.


Responses
Responses are evaluated from top to bottom; the final response is the default response. Set conditions in each response to evaluate when it will be sent.
200 - OK
If .. equals .. then send 200 response
Rule combination
All the following are true
Response field
Predicate
Value
Input
Equals
Input
Add condition +
404 - Not Found
If .. equals .. then send 404 response
Rule combination
All the following are true
Response field
Predicate
Value
Input
Equals
Input
Add condition +
500 - Internal Server Error
Else send default response

Note:

Responses are evaluated from top to bottom where the final response is the default response. Each response has the following properties:

- A condition with three fields, response, predicate, and value.
- One or more conditions.

You can change the order of the responses by using the and next to each response case.

The sequence of the conditions within a response can be changed. Click to change the position in the sequence.

Conditions can be deleted. Click to delete a condition.

The default order of this responses is such that 200 - OK is the first to be evaluated, followed by 404 – Not Found, followed by 500 - Internal server error is the last and therefore the default response.

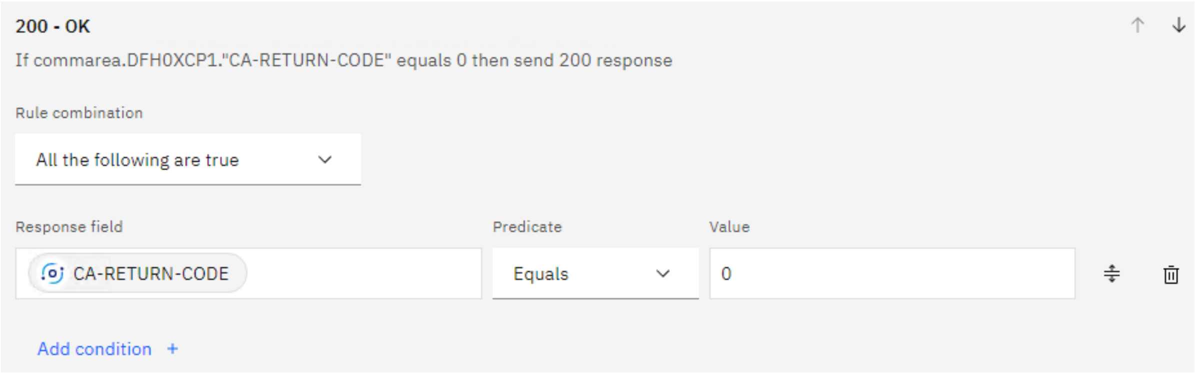
In the following steps, you can either manually enter the values or click ☰ and select values from the **Available mappings** menu.

The menu lists options for Path Parameters, zosAssetResponse, and error. Click next to each of these **Available mappings** menu options to use them to build the expression for your Response Field condition.

___**18.** Set the conditions for the **200 - OK** response which indicates success response.

- Select **All the following are true**
- Set the input the condition of: **CA-RETURN-CODE** equals `0`.

A quick way to search for a field is to start typing `RETURN` and use the keyboard up/down and **Enter** key to select **CA-RETURN-CODE**



200 - OK

If commarea.DFH0XCP1."CA-RETURN-CODE" equals 0 then send 200 response

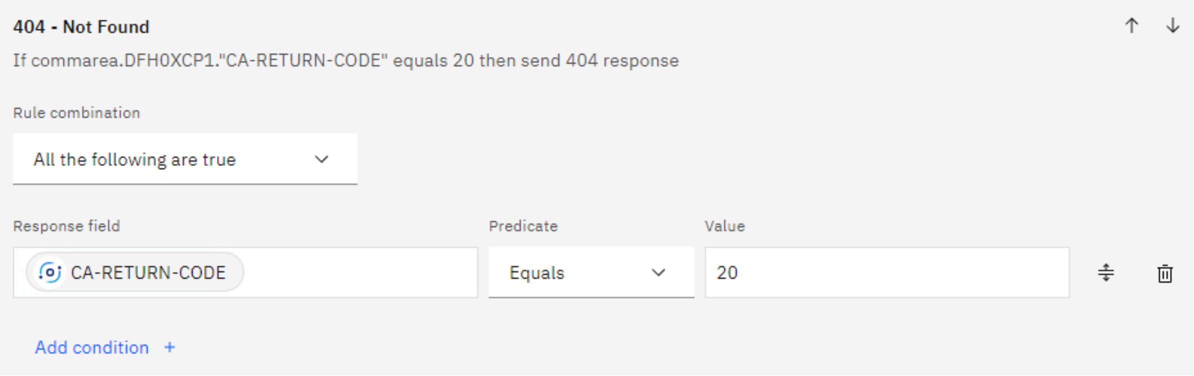
Rule combination: All the following are true

Response field	Predicate	Value
CA-RETURN-CODE	Equals	0

Note: if the yellow warning icon still appears after entering the condition, try to choose a different **Predict** and change it back to **Equal**. This is a known issue relates to the browser.

___**19.** Set the conditions for the **404 – Not Found** response which indicates an item is not found in the catalog.

- Select **All the following are true**
- Set the following conditions: **CA-RETURN-CODE** equals `20`
  Which is the return code this CICS program uses.



404 - Not Found

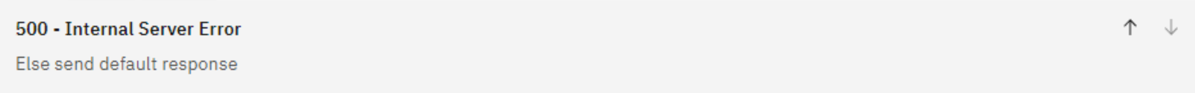
If commarea.DFH0XCP1."CA-RETURN-CODE" equals 20 then send 404 response

Rule combination

All the following are true

Response field	Predicate	Value
CA-RETURN-CODE	Equals	20

___**20.** The **500 - Internal server error** response is the default, so it has no conditions and must be the last entry in the table. Best practice is always configure **500 - Internal server error** as the default response to capture any errors in the conditional logic of the response.



500 - Internal Server Error

Else send default response

Define the Response fields mapping

The z/OS Asset response fields need to be mapped to the API response fields. In the previous task, you defined the order in which the response codes are checked. The next step is to map the actual data that is returned for each response code.

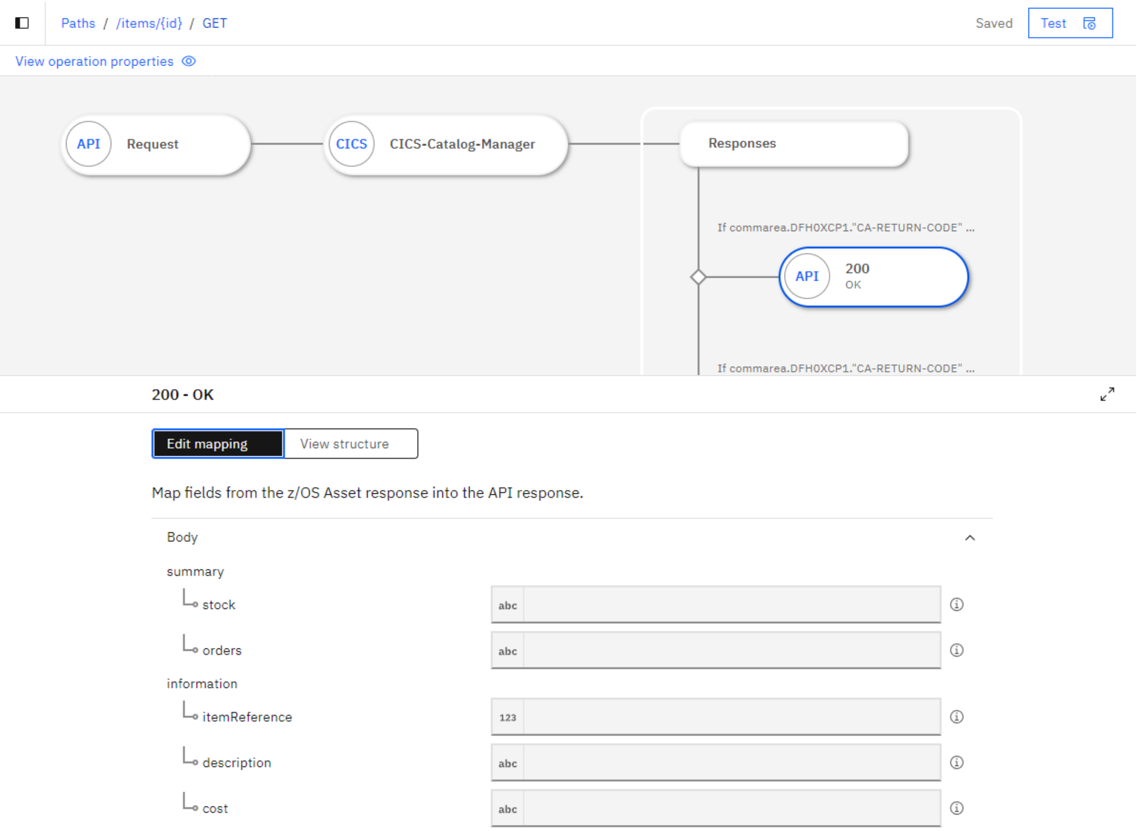
___**21.** Close the response configuration panel to go back to the **GET /items/{id}** operation flow diagram.

An amber exclamation mark ! indicates that the mapping is not defined.

Map the 200 responses.

___**22.** In the operation flow diagram, click the **200 – OK** response node.

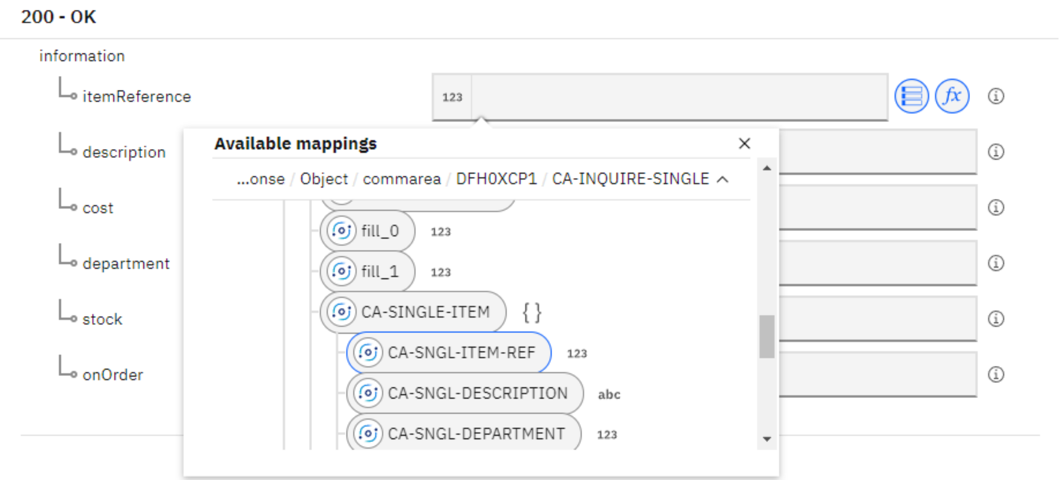
___**23.** The 200 response code indicates that the requested catalog items were found and the information is returned as normal in the payload. The item record data need to be mapped to the fields in the API response.


Paths / /items/{id} / GET
Saved
Test
View operation properties
API
Request
CICS
CICS-Catalog-Manager
Responses
If commarea.DFHOXCP1."CA-RETURN-CODE" ...
API
200
OK
If commarea.DFHOXCP1."CA-RETURN-CODE" ...
200 - OK
Edit mapping
View structure
Map fields from the z/OS Asset response into the API response.
Body
summary
stock
abc
orders
abc
information
itemReference
123
description
abc
cost
abc

___**24.** Let's leave the top **summary** section blank for now and focus on the mapping of the item **information** section.

___**25.** Click the **itemReference** field and click the **insert mapping icon** ☰ next to it.

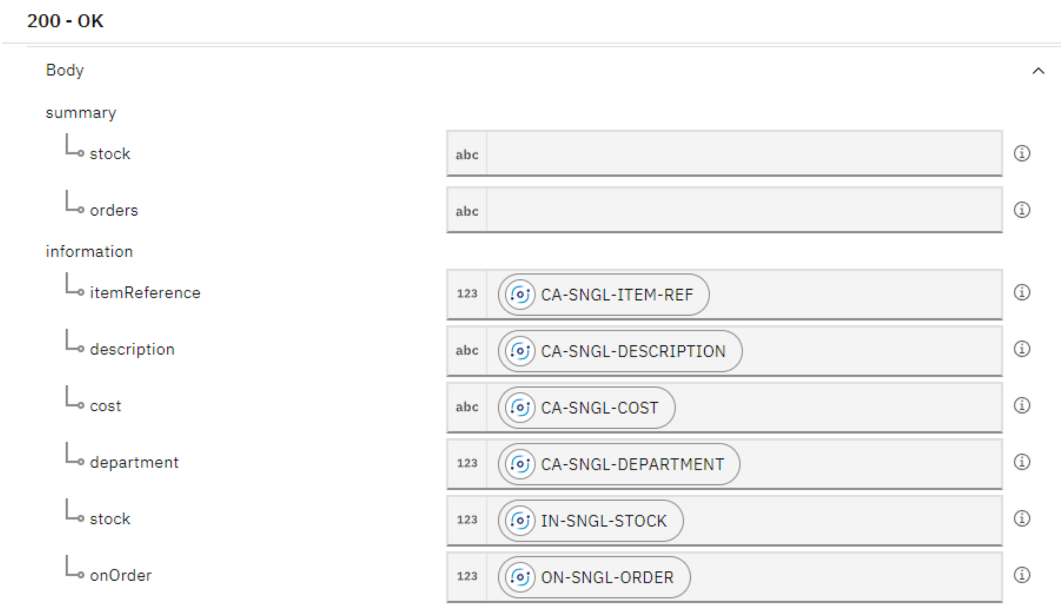
___**26.** Expand **zosAssetResponse** object and scroll down to locate **CA-INQUIRY-SINGLE** then click the **CA-SNGL-ITEM-REF** field. Like before, you can also start typing `SNGL` in the text field to quickly lookup the object from the dropdown search result.



___**27.** Repeat the process to populate all API response fields under **information** from the **zosAssetResponse** object:

API response field	zosAssetResponse
information→itemReference	CA-SNGL-ITEM-REF
information→description	CA-SNGL-DESCRIPTION
information→cost	CA-SNGL-COST
information→department	CA-SNGL-DEPARTMENT
information→stock	IN-SNGL-STOCK
information→onOrder	ON-SNGL-ORDER

___**28.** You should have this after all the fields are mapped.



200 - OK

Body

summary

- stock: abc
- orders: abc

information

- itemReference: 123 CA-SNGL-ITEM-REF
- description: abc CA-SNGL-DESCRIPTION
- cost: abc CA-SNGL-COST
- department: 123 CA-SNGL-DEPARTMENT
- stock: 123 IN-SNGL-STOCK
- onOrder: 123 ON-SNGL-ORDER

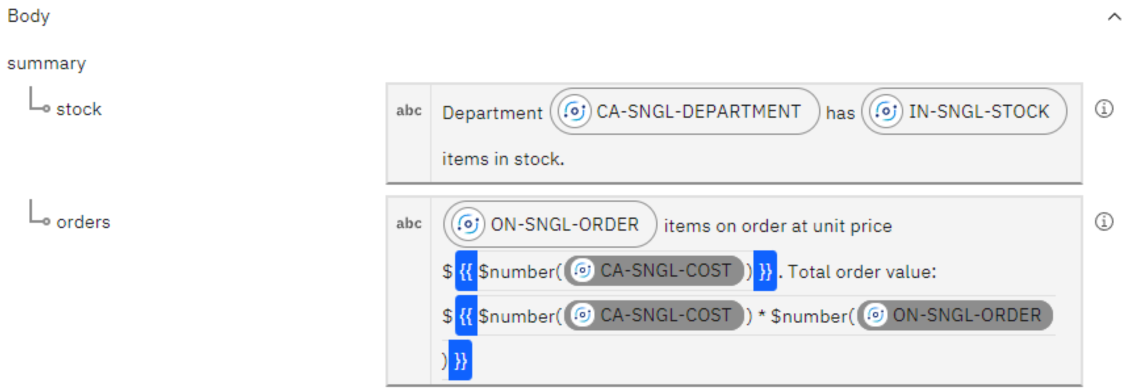
___**29.** Now we will populate the summary section to provide additional information for the API user.

You can either create the complex mapping by start typing and choose the fields from dropdown selection menu as you go **or** copy the entire mapping definition source provided down below which includes the **text**, **data field reference** as well as **JSONata functions** and paste them into the field.

Pay special attention to using the straight double-quote symbol.

API response field	Mapping definition
summary→stock	`Department {{$zosAssetResponse.commarea.DFH0XCP1."CA-INQUIRE-SINGLE"."CA-SINGLE-ITEM"."CA-SNGL-DEPARTMENT"}} has {{$zosAssetResponse.commarea.DFH0XCP1."CA-INQUIRE-SINGLE"."CA-SINGLE-ITEM"."IN-SNGL-STOCK"}} items in stock.`
summary→orders	`{{$zosAssetResponse.commarea.DFH0XCP1."CA-INQUIRE-SINGLE"."CA-SINGLE-ITEM"."ON-SNGL-ORDER"}} items on order at unit price ${{$number($zosAssetResponse.commarea.DFH0XCP1."CA-INQUIRE-SINGLE"."CA-SINGLE-ITEM"."CA-SNGL-COST")}}. Total order value: ${{$number($zosAssetResponse.commarea.DFH0XCP1."CA-INQUIRE-SINGLE"."CA-SINGLE-ITEM"."CA-SNGL-COST") * $number($zosAssetResponse.commarea.DFH0XCP1."CA-INQUIRE-SINGLE"."CA-SINGLE-ITEM"."ON-SNGL-ORDER")}}`

___30. If done correctly, the mapping editor should automatically parse and display the data field items, JSONata function and other text strings correctly. If it's not showing up exactly as below, double-check the code pasted in the text box.



Body

summary

stock

abc Department CA-SNGL-DEPARTMENT has IN-SNGL-STOCK items in stock.

orders

abc ON-SNGL-ORDER items on order at unit price $ {{ $number(CA-SNGL-COST) }}. Total order value: $ {{ $number(CA-SNGL-COST) * $number(ON-SNGL-ORDER) }}

Note:

JSONata is an open source expression language that is used for querying and transforming JSON data. z/OS Connect uses the JSONata to reformat and restructure JSON data that is contained in a response. You enter a JSONata expression directly to the mapping field for the response. For more information, see https://Github.com/IBM/JSONata4Java.

z/OS Connect application developers use JSONata to achieve the following benefits:

> Provide a much richer and more advanced set of mapping capabilities than exists today in z/OS Connect V3.
> Enable application developers to map source files where they can write direct JSONata queries and functions without the need for a client.
> Offer a documented and curated open source library that is maintained with active collaborators.
> Enable synergy with other IBM tools (like IBM AppConnect) that also use JSONata to provide a consistent experience for your application developers.

You can use JSONata to create sophisticated queries that are expressed in a compact and intuitive notation. A rich complement of built-in operators and functions is provided for manipulating and combining extracted data. The results of queries can be formatted into any JSON output structure by using familiar JSON object and array syntax. Coupled with the facility to create user-defined functions, advanced expressions can be built to handle any JSON query and transformation task.

JSONata is used to extract meaningful data that is buried in potentially large JSON structures. It can be applied to virtually any problem that involves querying and transforming JSON data, and is able to do the following:

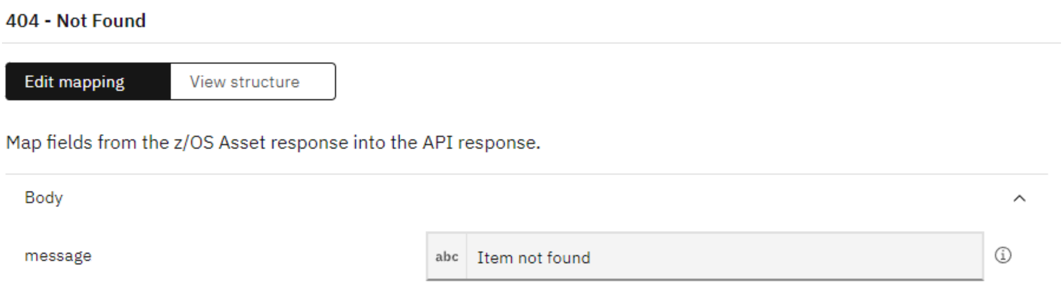
>Manipulate strings.
>Combine and aggregate numeric data.
>Query and extract values.
>Create complex JSON output structures that enable complex data transformation tasks.

Map the 404 and 500 responses.

Now that we've done the mapping for the 200 - OK response, we also need to map the rest of the error condition responses.

___**31.** Click the **404 – Not Found** node

___**32.** Type `Item not found` in the message field which is the only field to be returned as required by the API specification.



404 - Not Found

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

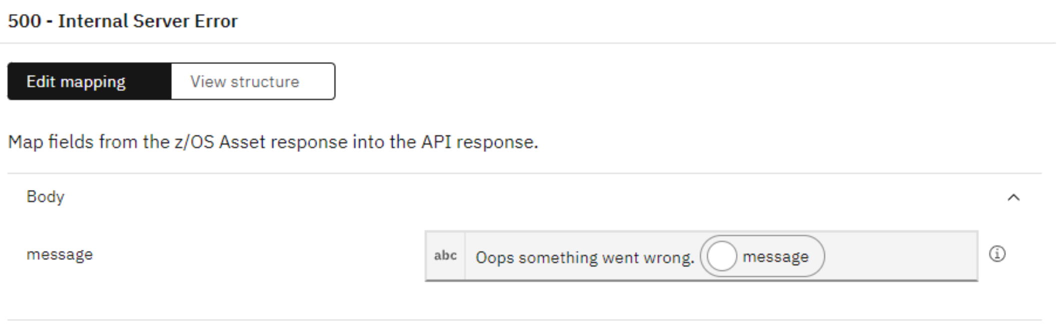
message abc Item not found

___**33.** Moving on to click the **500 – Internal Server Error** node

___**34.** Type `Oops something went wrong.` into the message field.

___**35.** And it's a good idea to include some detail runtime message to tell the user about the error.

You can then click on the Add a mapping icon ☰, select the **error** object then click the **message** field to append it to the response message.



500 - Internal Server Error

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

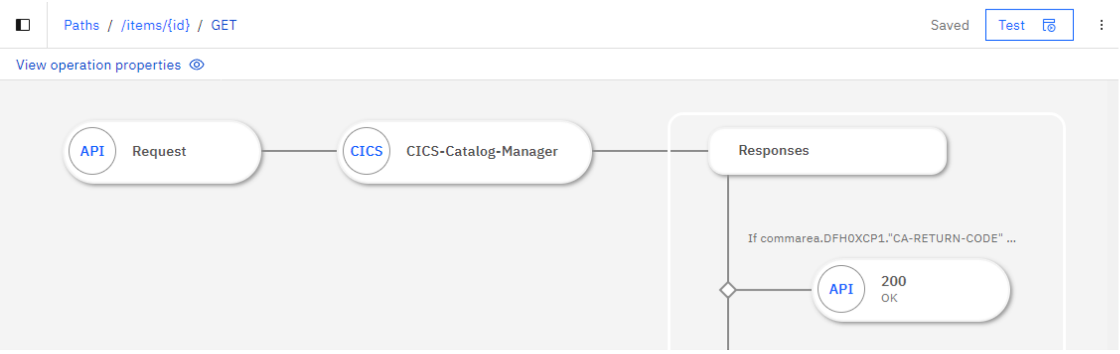
message abc Oops something went wrong. ○ message

___**36.** Once it's done you've completed the mapping for the **/items/{id}** operation and all changes are saved automatically so you can move on to the next part to test it.

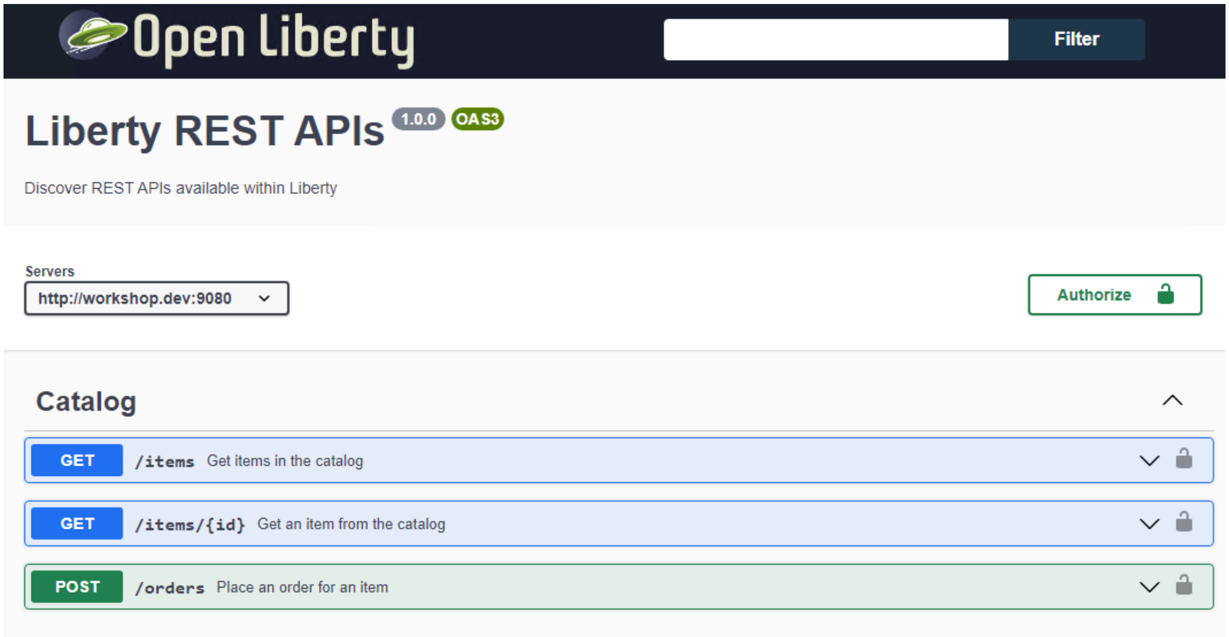
Part 4: Test the API using built-in OpenAPI3 testing tool

In this part of the lab, you will test the API mapping in the built-in API testing tool.

___**1.** On the upper right corner of the z/OS Connect Designer window, click the blue **Test** button.

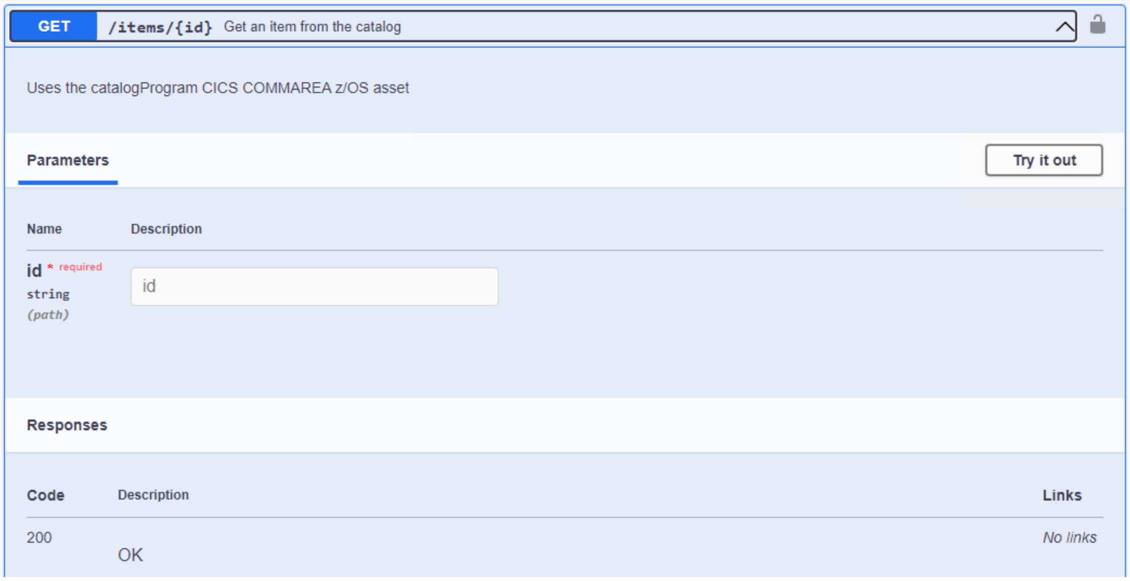

Paths / /items/{id} / GET
Saved
Test
View operation properties
API Request
CICS CICS-Catalog-Manager
Responses
If commarea.DFHOXCP1."CA-RETURN-CODE" ...
API 200 OK

___2. The **Open Liberty REST API Client** opens your OpenAPI definition in a new tab in your browser, and ready to be tested.


Open Liberty
Filter
Liberty REST APIs 1.0.0 OAS3
Discover REST APIs available within Liberty
Servers
http://workshop.dev:9080
Authorize
Catalog
GET /items Get items in the catalog
GET /items/{id} Get an item from the catalog
POST /orders Place an order for an item

___3. The **Open Liberty REST Client** lists the paths of your z/OS Connect API that are ready to test. It acts as an HTTP API client to invoke your OpenAPI by sending a RESTful request to your API endpoints.

___4. To test the **GET /items/{id}** operation, click to expand the **GET /items/{id}** operation, which reveals the detail of the API call format, possible return codes and sample responses.



GET /items/{id} Get an item from the catalog

Uses the catalogProgram CICS COMMAREA z/OS asset

Parameters

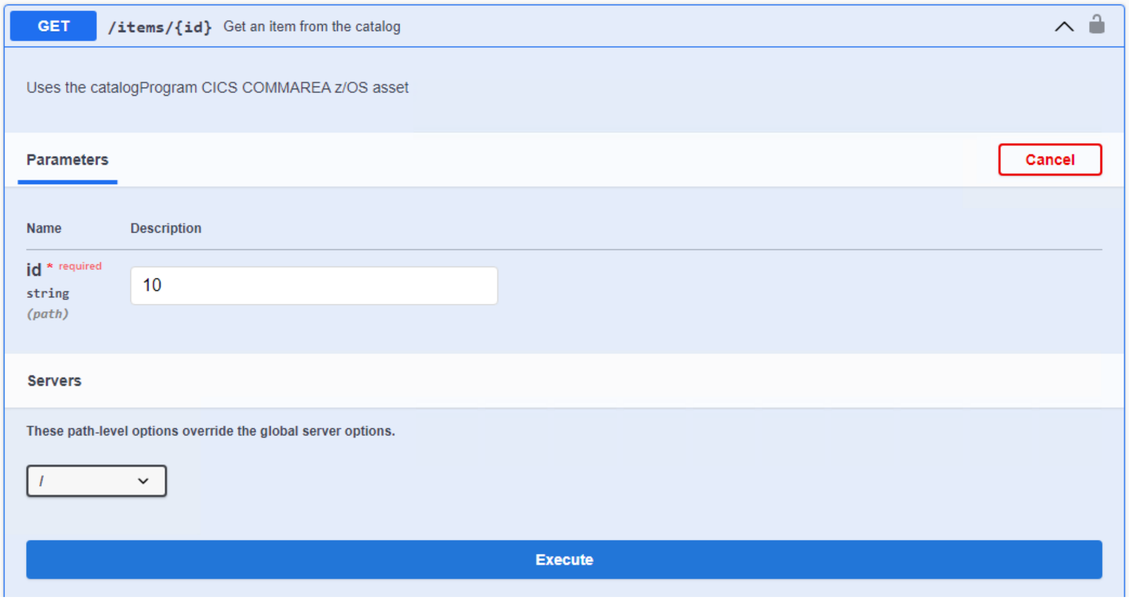
Name	Description
id * required string *(path)*	id

Responses

Code	Description	Links
200	OK	*No links*

___**5.** Click **Try it out** button for the operation.

___**6.** Input an id for an item that exists in the catalog in the id field. For example, `10`.



GET /items/{id} Get an item from the catalog

Uses the catalogProgram CICS COMMAREA z/OS asset

Parameters

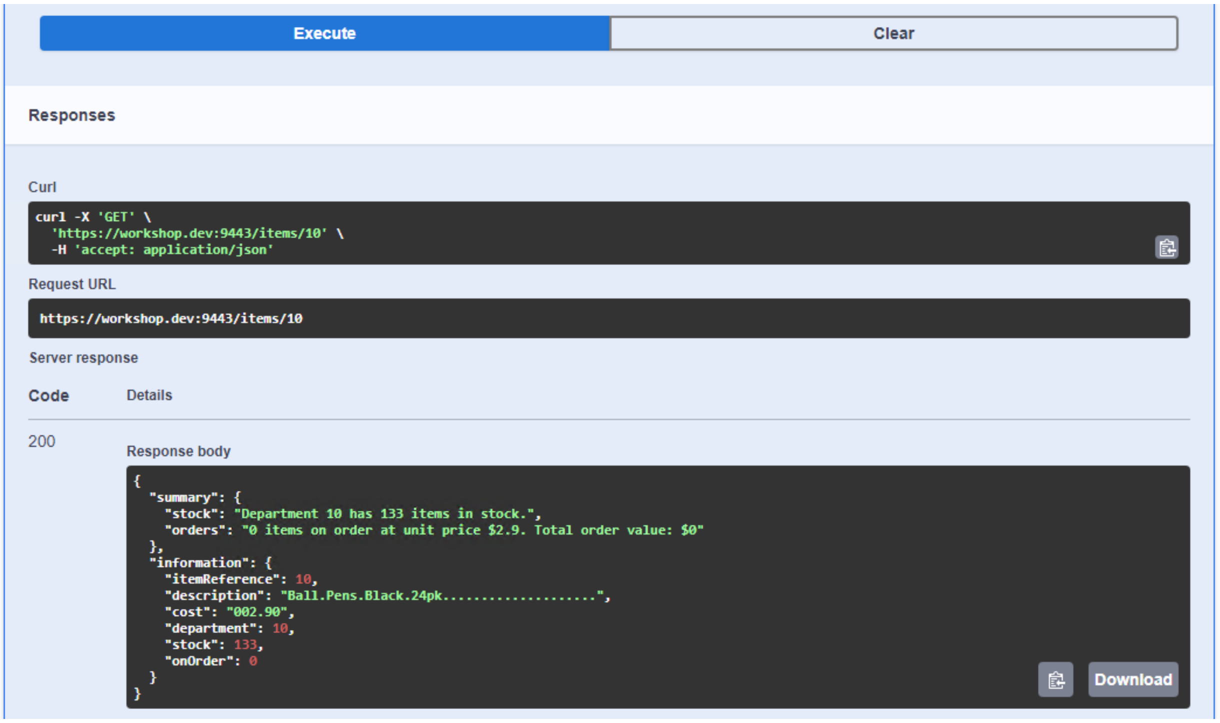
Name	Description
id * required string *(path)*	10

Servers

These path-level options override the global server options.

/

___**7.** Click **Execute** button to invoke the API using provided value, you can now examine the response from the API.



Curl

```
curl -X 'GET' \
  'https://workshop.dev:9443/items/10' \
  -H 'accept: application/json'
```

Request URL

```
https://workshop.dev:9443/items/10
```

Server response

Code | Details
200 | Response body

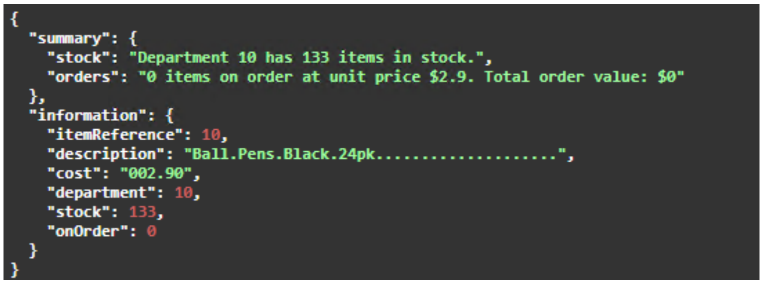
```
{
  "summary": {
    "stock": "Department 10 has 133 items in stock.",
    "orders": "0 items on order at unit price $2.9. Total order value: $0"
  },
  "information": {
    "itemReference": 10,
    "description": "Ball.Pens.Black.24pk....................",
    "cost": "002.90",
    "department": 10,
    "stock": 133,
    "onOrder": 0
  }
}
```

___**8.** You can also try to input an invalid id for example `99` to test if your API is returning 404 return code as expected.

___**9.** Congratulates you’ve just tested your new API is working as expected.

Part 5: Fine-tune the API data format using advanced mapping capability.

In the previous part we’ve successfully tested the API, however there’s a few minor things we could do to make it more user friendly to the API consumers. Here are the new requirements:



```
{
  "summary": {
    "stock": "Department 10 has 133 items in stock.",
    "orders": "0 items on order at unit price $2.9. Total order value: $0"
  },
  "information": {
    "itemReference": 10,
    "description": "Ball.Pens.Black.24pk....................",
    "cost": "002.90",
    "department": 10,
    "stock": 133,
    "onOrder": 0
  }
}
```

- API user requested to include a current timestamp at the end of the stock summary information
- You noticed in the item description it’s returning a fixed-length string with all the dots as the placeholder. It is the way it’s designed in CICS program but you want it to be replaced by space also remove all the trailing spaces to be more readable.

- The item cost is fixed-length string due to the copybook definition with leading zeros.

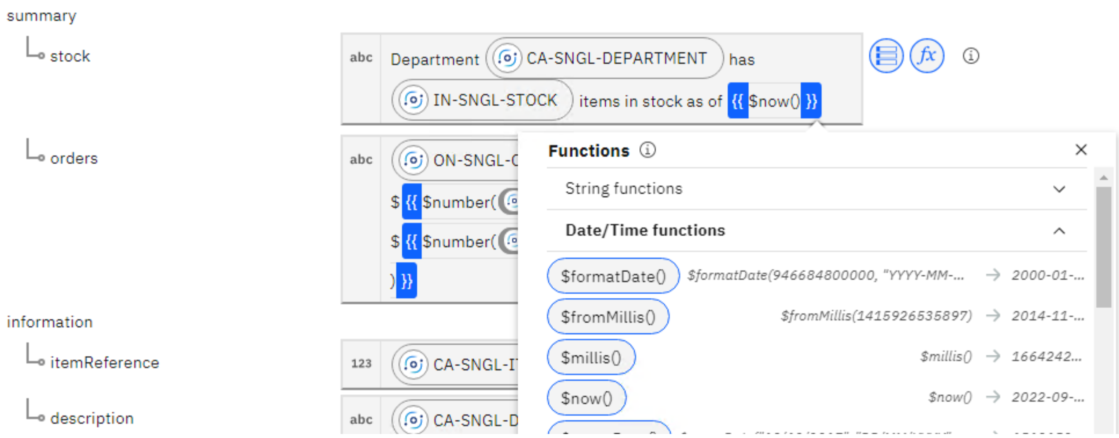
Let's get on to the tasks.

___**1.** Switch back to the **z/OS Connect Designer** browser tab to continue working on implementing the remaining changes to the API mapping.

___**2.** Click on the **200 – OK** node to open the mapping editor.

___**3.** Place the cursor at the end of the **summary → stock** field and add " `as of` " (with a space on both side) then click on the **insert function icon** fx.

___**4.** Expand **Date/Time functions** and select **$now()** to add the function to the end.



___**5.** By default **$now()** function returns ISO 8601 format which looks like `2022-09-27T03:13:50.876Z`, you can optionally add a formatting parameter to the **$now()** function between the parentheses to specify a desired format of the timestamp. For example make it:

```
$now('[D01]/[M01]/[Y0001] [h#1]:[m01][P]')
```

This way it will produce a timestamp in the format of `27/09/2022 3:00am`

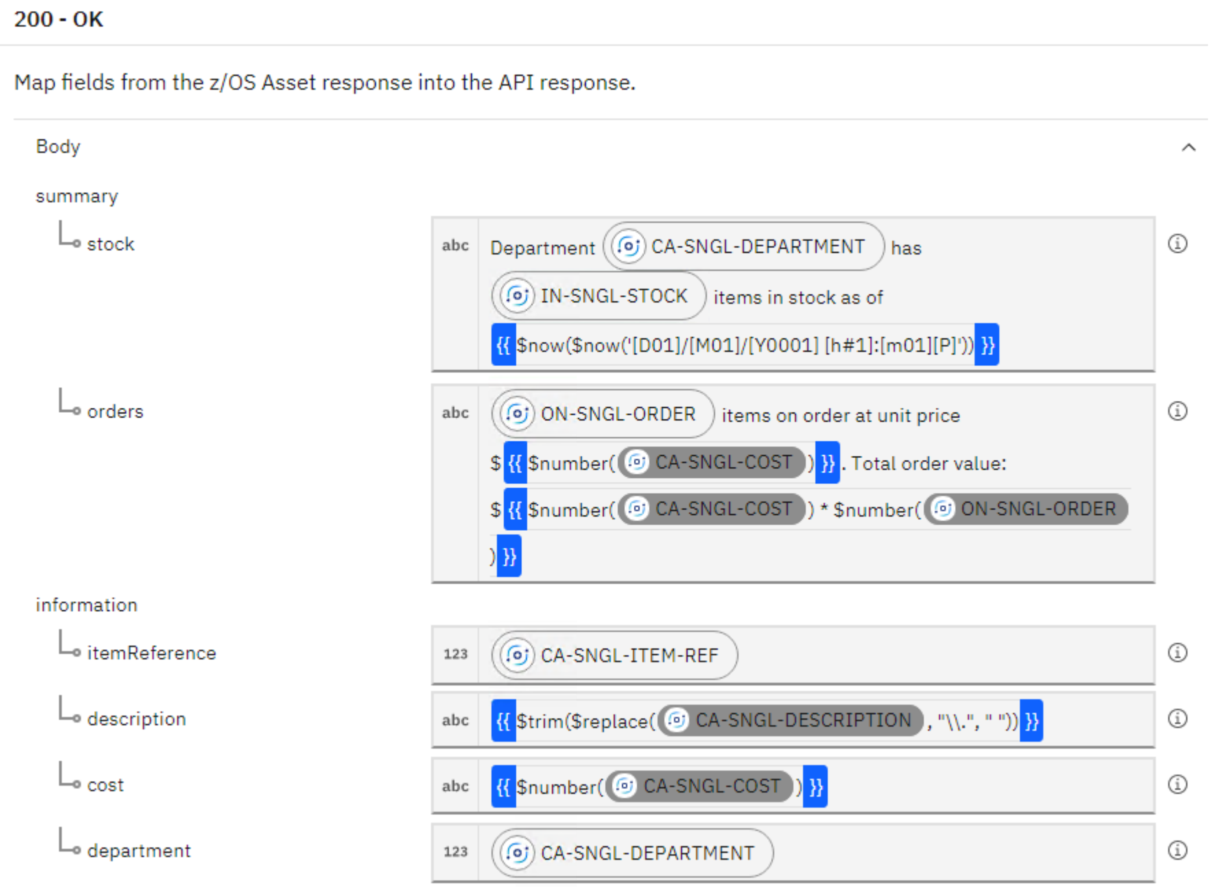
___**6.** Delete the **CA-SNGL-DESCRIPTION** field for **information → description** then copy and paste the below definition into the text area.

```
{{$trim($replace($zosAssetResponse.commarea.DFH0XCP1."CA-INQUIRE-
SINGLE"."CA-SINGLE-ITEM"."CA-SNGL-DESCRIPTION", "\\.", " "))}}
```

Note: these are two functions nested together, first to use `$replace` to match the string using a RegEx pattern for any character of "."(dot) to be replaced by " " (space) then use the `$trim()` function to remove the trailing whitespace. Both are built-in JSONata functions.

___**7.** Lastly, click on the **CA-SNGL-COST** field and select **Apply a function** in the pop-up panel, choose the **Casting functions** and click on **$number** to covert the string format cost into a numeric format, which will make it a real number in the API response.

___**8.** Once it's done the mapping should look like this.



200 - OK

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

Body

summary

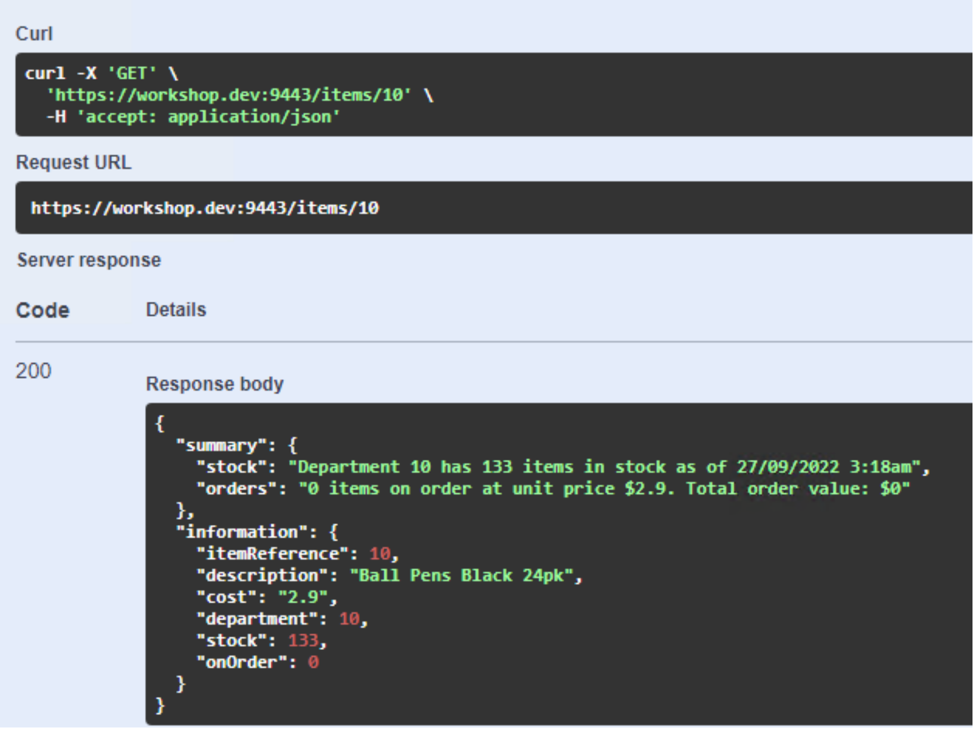
stock	abc	Department CA-SNGL-DEPARTMENT has IN-SNGL-STOCK items in stock as of {{ $now($now('[D01]/[M01]/[Y0001] [h#1]:[m01][P]')) }}
orders	abc	ON-SNGL-ORDER items on order at unit price $ {{ $number(CA-SNGL-COST) }}. Total order value: $ {{ $number(CA-SNGL-COST) * $number(ON-SNGL-ORDER) }}

information

itemReference	123	CA-SNGL-ITEM-REF
description	abc	{{ $trim($replace(CA-SNGL-DESCRIPTION , "\\.", " ")) }}
cost	abc	{{ $number(CA-SNGL-COST) }}
department	123	CA-SNGL-DEPARTMENT

___**9.** Head back to the API Testing tab. If you've closed it, you can reopen it by clicking the **Test** button at the top.

___**10.** Try to invoke the same API again and now the API response is exactly what we wanted.



Curl

```
curl -X 'GET' \
  'https://workshop.dev:9443/items/10' \
  -H 'accept: application/json'
```

Request URL

```
https://workshop.dev:9443/items/10
```

Server response

Code	Details
200	Response body

```
{
  "summary": {
    "stock": "Department 10 has 133 items in stock as of 27/09/2022 3:18am",
    "orders": "0 items on order at unit price $2.9. Total order value: $0"
  },
  "information": {
    "itemReference": 10,
    "description": "Ball Pens Black 24pk",
    "cost": "2.9",
    "department": 10,
    "stock": 133,
    "onOrder": 0
  }
}
```

Part 6: Check in and push the API project to the Git repository

In previous part of the lab exercise, we have completed the mapping for the “single item inquiry” API and tested that it’s working as expected. Now we’re ready to check it into the Git repository.

As mentioned at the beginning, our API project is hosted on a remote Linux development server where the z/OS Connect Designer container runs. The project directory was cloned from a remote Git repo on GitLab server, and it’s mounted into the Designer container for development. After we’ve done all the code changes, we’ll then need to commit all the updates in the file system directory back to the Git repo and push it to the remote GitLab repository to make it available to everyone.

Note:

As part of this lab scenario, we are using VSCode IDE to commit the changes on the remote Linux server, but you can also use SSH to log into the remote Linux shell and issue Git commit and Git push commands manually from the command prompt.

Furthermore, you also don’t have to run it on a remote environment, if you have a physical Windows or MacOS PC, you can also set up the z/OS Connect Designer container using Docker Desktop locally and use any IDE or command line to commit and push the changes in the project directory. We opted for remote server for the lab because of a limitation for the lab environment doesn’t support Windows Docker software.

Another option is to use a full cloud-native development environment to develop your zOS Connect API then commit all the changes in a web-based IDE like Openshift CodeReady Workspace (recently renamed to DevSpace), with this approach there's zero installation required to setup the Docker and the Git commits can also be done in the browser IDE.

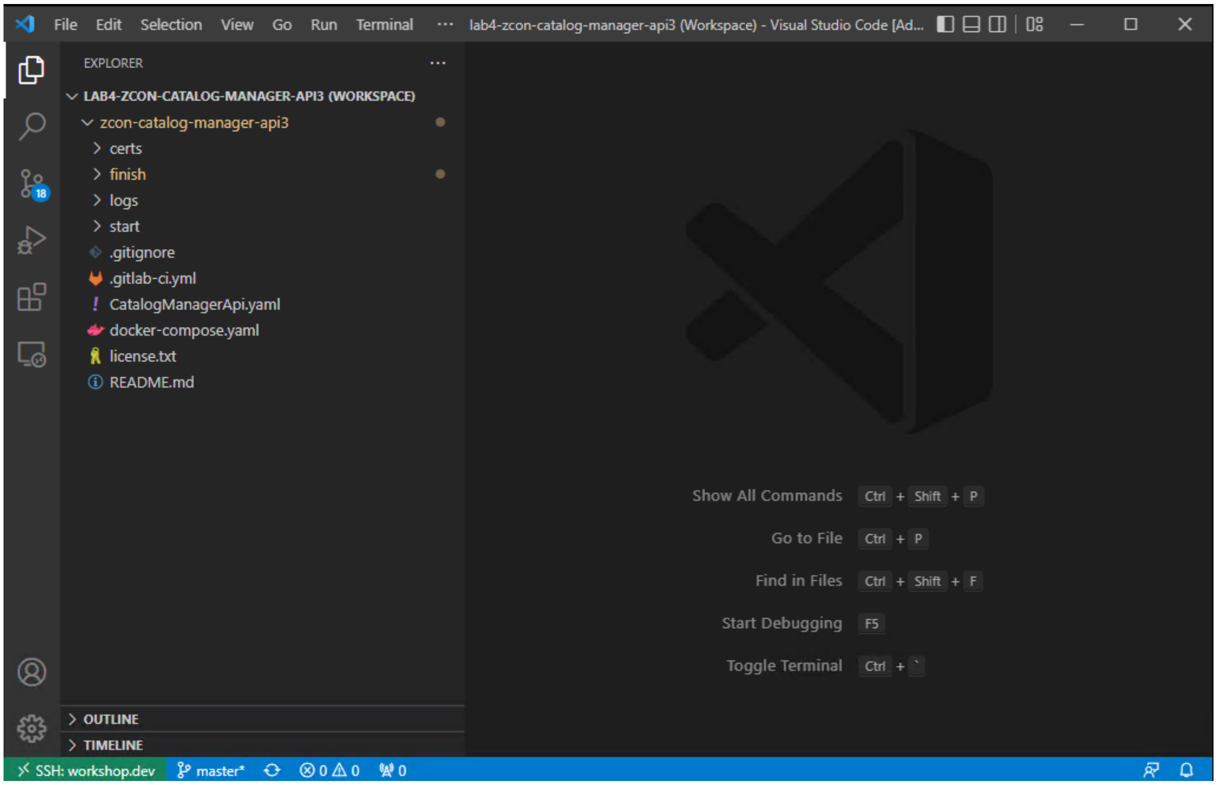
You can read more details about this setup in this blog or refer to the product manual.

https://medium.com/@ibmgeorge/cloud-based-development-for-z-os-connect-openapi3-apis-121bde3a6655

Open VSCode and connect to the Linux development server

___**1.** Double click **Lab - zOS Connect OAS3 API VSCode** icon on the desktop to open the VSCode workspace for z/OS Connect API lab.

___**2.** The VSCode will automatically open the remote project directory on the Linux server (**workshop.dev**) using SSH as indicated in the **green status bar** at the bottom.



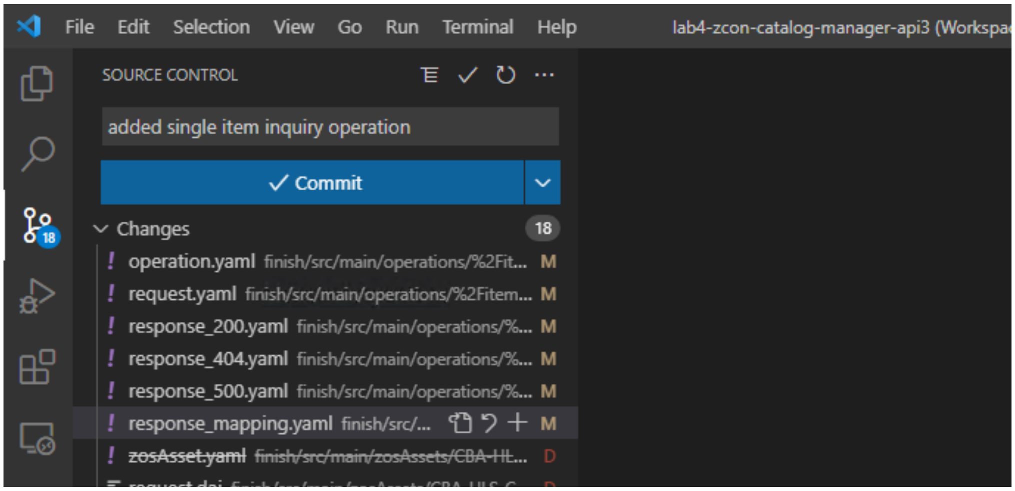
___3. Click the **Source Control** button in the left navbar where you can review all the changed you just made via the z/OS Connect Designer in previous steps.

___4. Feel free to explore all the mapping and API definition files changed, as you will find out these are all YAML documents. Being an open format means it's easy to understand and you can even make changes without going thru the Designer graphic user interface for repetitive tasks, this can be really powerful for automation.

Commit the changes and push them to the remote GitLab repository

___5. Input some text description for the **commit message**. For example:

```
added single item inquiry operation
```



added single item inquiry operation

! operation.yaml finish/src/main/operations/%2Fit... M
! request.yaml finish/src/main/operations/%2Fitem... M
! response_200.yaml finish/src/main/operations/%... M
! response_404.yaml finish/src/main/operations/%... M
! response_500.yaml finish/src/main/operations/%... M
! response_mapping.yaml finish/src/... M
! ~~zosAsset.yaml finish/src/main/zosAssets/CBA-HL...~~ D

___6. Click the **Commit** button to commit the changes to your local Git repository

___7. Click the **Sync Changes** button again to push the changes to the remote Git repository hosted on GitLab server.

___8. Type in your GitLab credentials `ibmdev/Passw0rd` to complete the Git push operation

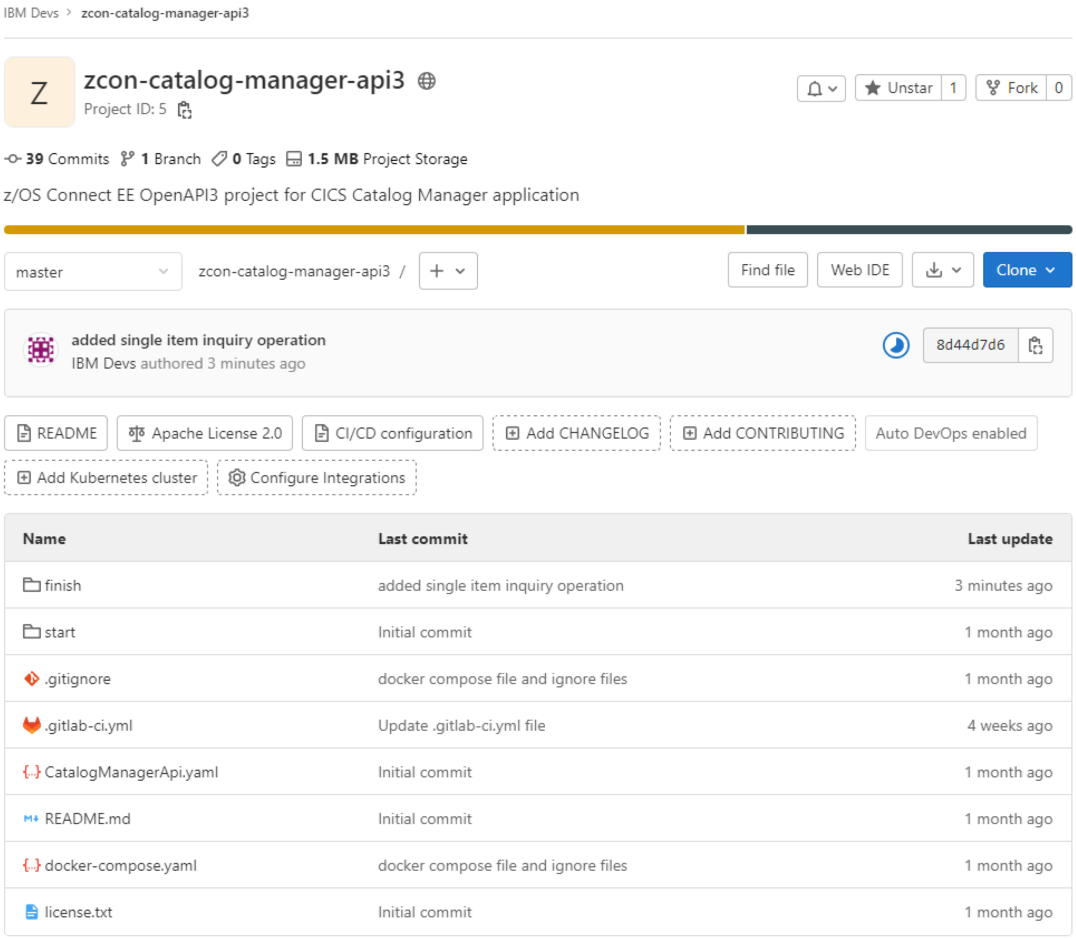
___9. Now all the changes have been pushed to the remote Git repository and we will examine the DevOps pipeline it has triggered in the next part of the lab.

Part 7: Explore the DevOps pipeline that automates the build, deploy and testing

In this final part of the lab exercise, we will review the DevOps pipeline that we've set up on the GitLab server for building, deploying and testing the API on the z/OS.

Log in to the GitLab project dashboard

___**1.** Open the **Chrome** browser and click the **GitLab** bookmark, log in to the GitLab using `ibmdev/Passw0rd` then select **zcon-catalog-manager-api3** project.



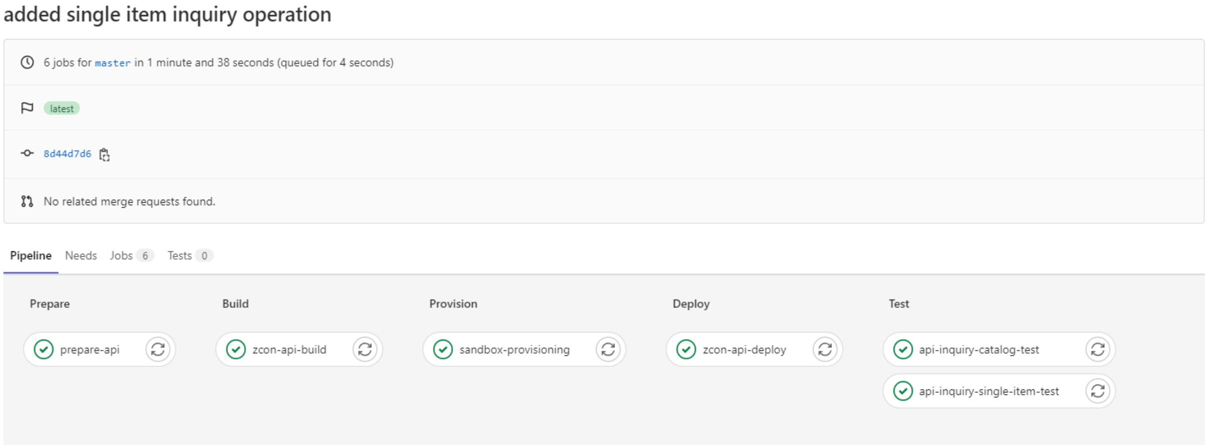
z/OS Connect EE OpenAPI3 project for CICS Catalog Manager application

added single item inquiry operation
IBM Devs authored 3 minutes ago
8d44d7d6

Name	Last commit	Last update
finish	added single item inquiry operation	3 minutes ago
start	Initial commit	1 month ago
.gitignore	docker compose file and ignore files	1 month ago
.gitlab-ci.yml	Update .gitlab-ci.yml file	4 weeks ago
CatalogManagerApi.yaml	Initial commit	1 month ago
README.md	Initial commit	1 month ago
docker-compose.yaml	docker compose file and ignore files	1 month ago
license.txt	Initial commit	1 month ago

___**2.** You can see the most recent commit you made from the previous step and the status of the pipeline triggered by the commit. You should see a successful pipeline execution with icon ✓ in a few minutes after the commit is pushed.

___**3.** Click the Green icon ✓ to review the jobs in the pipeline.



added single item inquiry operation

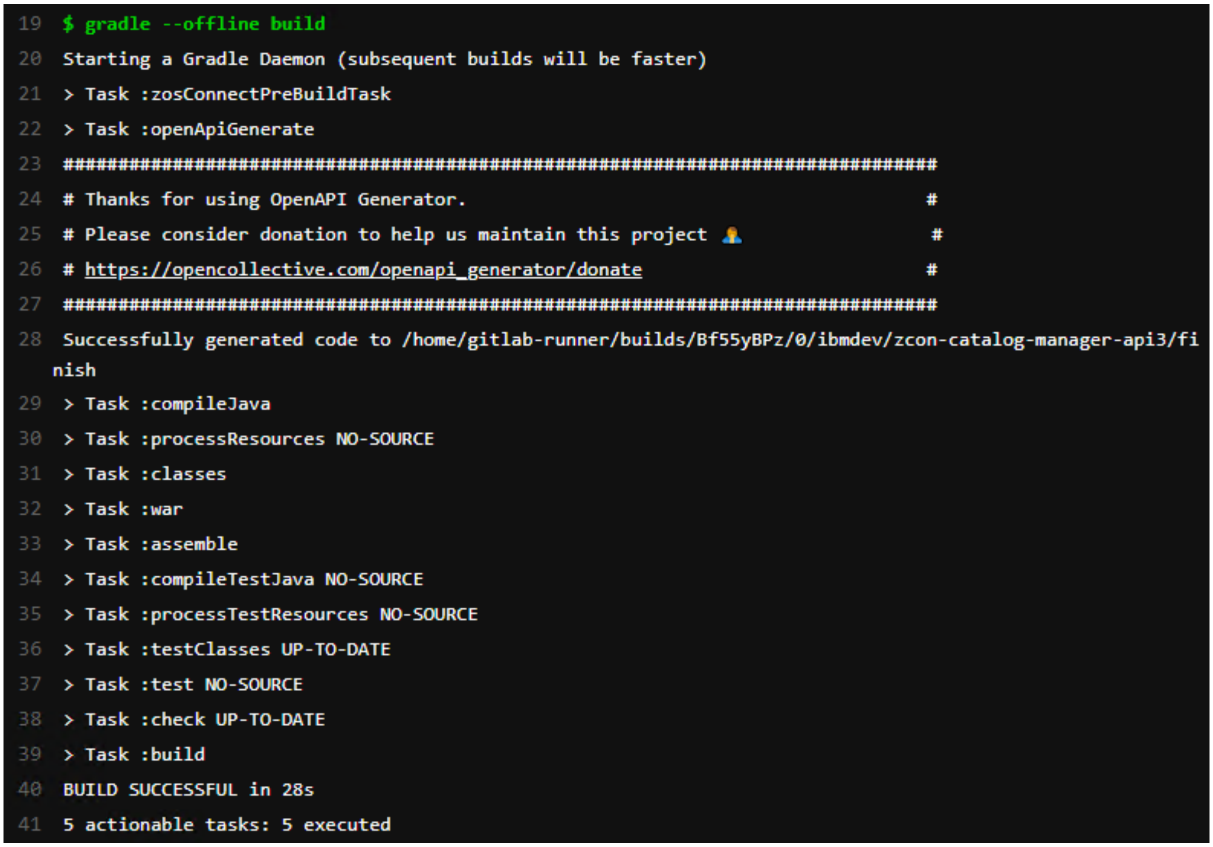
6 jobs for master in 1 minute and 38 seconds (queued for 4 seconds)

latest

8d44d7d6

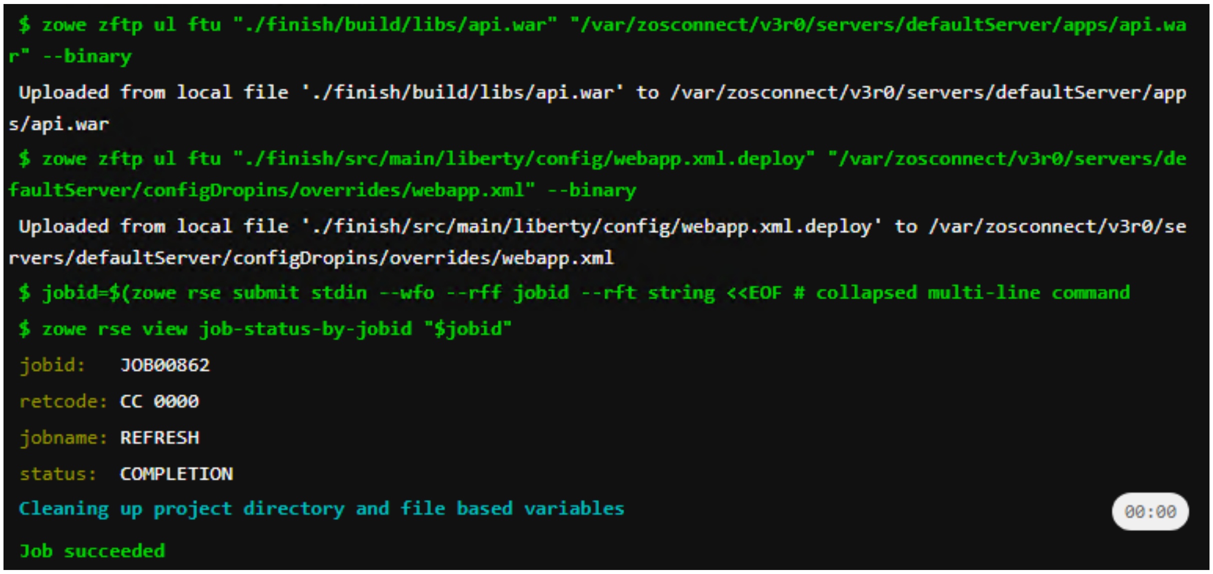
No related merge requests found.

___**4.** Click on the **zcon-api-build** job to check the **gradle** build result which is a popular open source build engine from Apache that z/OS Connect utilizes to build API project into the deployable API artifact which is a .war archive.



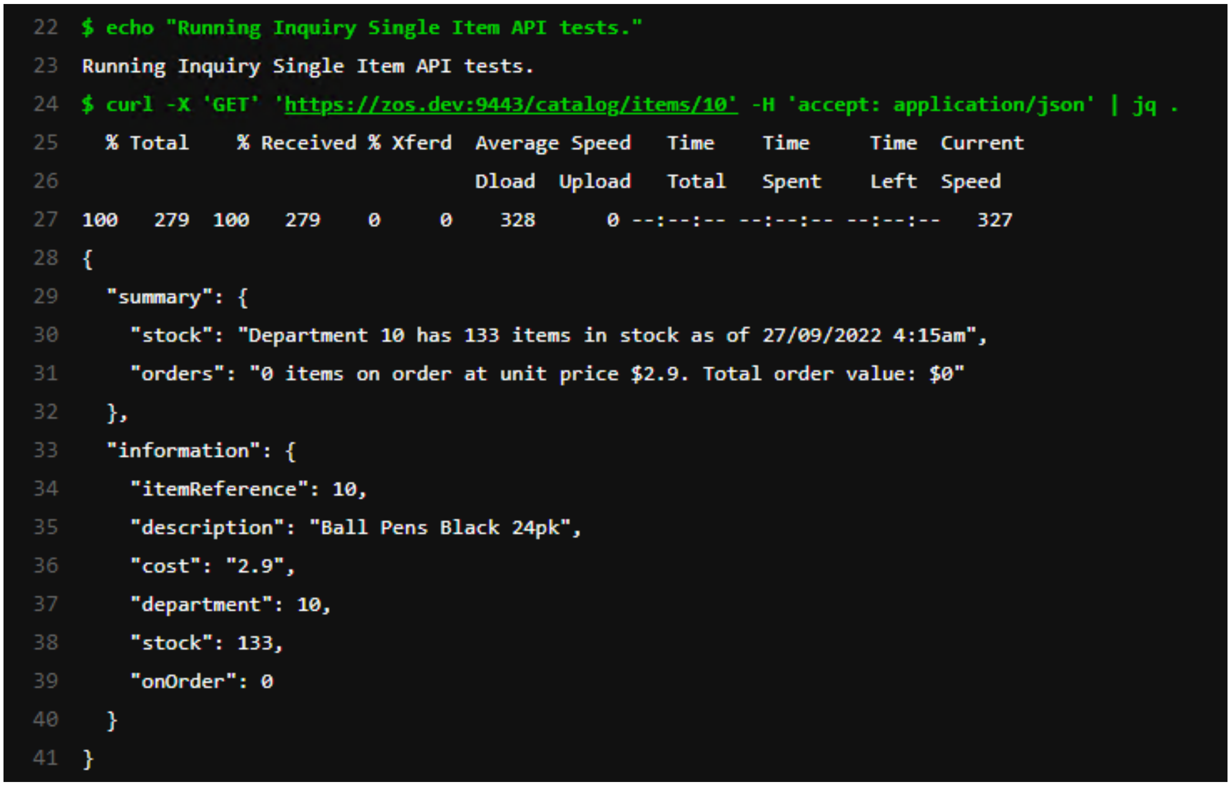
```
$ gradle --offline build
Starting a Gradle Daemon (subsequent builds will be faster)
> Task :zosConnectPreBuildTask
> Task :openApiGenerate
################################################################################
# Thanks for using OpenAPI Generator.                                          #
# Please consider donation to help us maintain this project 🙏                 #
# https://opencollective.com/openapi_generator/donate                          #
################################################################################
Successfully generated code to /home/gitlab-runner/builds/Bf55yBPz/0/ibmdev/zcon-catalog-manager-api3/finish
> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes
> Task :war
> Task :assemble
> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test NO-SOURCE
> Task :check UP-TO-DATE
> Task :build
BUILD SUCCESSFUL in 28s
5 actionable tasks: 5 executed
```

___**5.** In the **zcon-api-deploy** job you can check the deployment is done by issuing a **zowe cli** command to upload the **.war** file into USS file system of the z/OS and then submits a job to issue **MVS MODIFY** command to refresh the server.



```
$ zowe zftp ul ftu "./finish/build/libs/api.war" "/var/zosconnect/v3r0/servers/defaultServer/apps/api.wa
r" --binary
Uploaded from local file './finish/build/libs/api.war' to /var/zosconnect/v3r0/servers/defaultServer/app
s/api.war
$ zowe zftp ul ftu "./finish/src/main/liberty/config/webapp.xml.deploy" "/var/zosconnect/v3r0/servers/de
faultServer/configDropins/overrides/webapp.xml" --binary
Uploaded from local file './finish/src/main/liberty/config/webapp.xml.deploy' to /var/zosconnect/v3r0/se
rvers/defaultServer/configDropins/overrides/webapp.xml
$ jobid=$(zowe rse submit stdin --wfo --rff jobid --rft string <<EOF # collapsed multi-line command
$ zowe rse view job-status-by-jobid "$jobid"
jobid:   JOB00862
retcode: CC 0000
jobname: REFRESH
status:  COMPLETION
Cleaning up project directory and file based variables                                          00:00
Job succeeded
```

___**6.** Lastly, click on the **api-inquiry-single-item-test** job to check the API test result, which is done by simply issuing **curl** command line to send an API call to the z/OS Connect server on the mainframe. In real world scenario it can kick off a proper API testing tool to perform any sophisticated API testing.



```
$ echo "Running Inquiry Single Item API tests."
Running Inquiry Single Item API tests.
$ curl -X 'GET' 'https://zos.dev:9443/catalog/items/10' -H 'accept: application/json' | jq .
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   279  100   279    0     0    328      0 --:--:-- --:--:-- --:--:--   327
{
  "summary": {
    "stock": "Department 10 has 133 items in stock as of 27/09/2022 4:15am",
    "orders": "0 items on order at unit price $2.9. Total order value: $0"
  },
  "information": {
    "itemReference": 10,
    "description": "Ball Pens Black 24pk",
    "cost": "2.9",
    "department": 10,
    "stock": 133,
    "onOrder": 0
  }
}
```

___**7.** You have now completed the z/OS Connect OAS3 API lab.

Part 8: Summary

Congratulations! In this lab you've successfully completed the following tasks and implemented an OpenAPI Specification 3 API for the CICS Catalog Manager program.

- ✓ Explore the CICS catalog manager application
- ✓ Log in to the z/OS Connect Designer
- ✓ Define the operation & basic mapping for "inquiry single item" service.
- ✓ Test the API using the built-in OpenAPI3 testing tool
- ✓ Fine-tune the API data format using advanced mapping capability.
- ✓ Check-in and push the API project into the Git repository
- ✓ Explore the DevOps pipeline that automates the build, deploy and testing