



# Accessing z/OS resources with REST APIs using IBM z/OS Connect

Mitch Johnson [mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

John Brefach [John.J.Brefach@ibm.com](mailto:John.J.Brefach@ibm.com)

Washington System Center



# Agenda

- An Introduction to REST API
- Enabling RESTful API to z/OS resources,
  - CICS programs
  - Db2 tables and stored procedures
  - IMS transactions
  - IMS data bases
  - MQ queues
- An overview of z/OS Connect API Provider Security
- Miscellaneous Topics

# Notes and Disclaimers



- **IBM z/OS Connect (OpenAPI 2)** refers to the types of APIs supported by the z/OS Connect product prior to service level V3.0.55. **IBM z/OS Connect (OpenAPI 3)** refers to support for an additional type of API as well as additional functions and features added at service level V3.0.55. Important - servers configured for OpenAPI 2 can will continue to operate as is with service level V3.0.55 and later.
- A z/OS Connect OpenAPI 2, or a z/OS Connect OpenAPI 3 icon will appear on slides where the information is specific to these products. Don't hesitate to ask questions as to why the icon does or does not appear on certain slides
- A Liberty icon will appear on slides where the information both products. Don't hesitate to ask questions as to why the icon does or does not appear on certain slides.
- There will be additional information on slides that will be designated as Tech/Tips. These contain information that hopefully will be useful to the reader.



## Accessing REST APIs from z/OS using IBM z/OS Connect

Mitch Johnson [mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

John Brefach [John.J.Brefach@ibm.com](mailto:John.J.Brefach@ibm.com)

Washington System Center

mitchj@us.ibm.com



## WebSphere Liberty Profile Administration

Managing WebSphere Liberty Profile servers for IBM z/OS Connect (OpenAPI 2 and OpenAPI 3), IBM MQ REST Console and zOSMF



## z/OS Connect OpenAPI 3

Designer and z/OS Native server Experiences and Observations

Mitch Johnson  
[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)  
Washington Systems Center



© 2017, 2023 IBM Corporation  
Slide 1

[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

<https://www.ibm.com/support/pages/mainframe-system-education-wildfire-workshops-washington-systems-center-wsc>

# z/OS Connect Wildfire Github Site

<https://ibm.biz/zCeeWorkshopMaterial>



GitHub - ibm-wsc/zCONNEE-Wildfire-Workshop

github.com/ibm-wsc/zCONNEE-Wildfire-Workshop

Product Solutions Open Source Pricing

Search or jump

ibm-wsc / zCONNEE-Wildfire-Workshop Public

Code Issues Pull requests Actions Projects Security Insights

master 1 Branch 0 Tags

Go to file Code

emitchj Delete containers/readme bc38acc · 9 hours ago 648 Commits

APIRequesters Add files via upload 5 months ago

AdminSecurity Add files via upload last month

COBOL Samples Add files via upload 9 months ago

JCL Samples Add files via upload 5 months ago

OpenAPI2 Add files via upload 2 months ago

OpenAPI3 Add files via upload 3 months ago

XML Samples Add files via upload 2 years ago

archive Add files via upload 4 months ago

containers Delete containers/readme 9 hours ago

Accessing REST APIs from zOS using IBM zOS ... Add files via upload 2 months ago

Accessing zOS resources with REST APIs using ... Add files via upload 2 months ago

README.md Update README.md 3 years ago

ZADMIN - zOS Connect Administration (Ope... Add files via upload 5 months ago

zOS Connect EE V3 Advanced Topics Guide.pdf Add files via upload 3 years ago

zOS Connect EE V3 Getting Started.pdf Add files via upload 3 years ago

zCONNEE-Wildfire-Workshop

master zCONNEE-Wildfire-Workshop / OpenAPI2 /

emitchj Add files via upload

Developing CICS API Requester Applications.pdf Add files via upload

Developing IMS API Requester Applications.pdf Add files via upload

Developing MVS Batch API Requester Application... Add files via upload

Developing RESTful APIs for Db2 DVM Services... Add files via upload

Developing RESTful APIs for Db2 REST Services.... Add files via upload

Developing RESTful APIs for HATS REST Service... Add files via upload

Developing RESTful APIs for IMS DVM Services... Add files via upload

Developing RESTful APIs for IMS Database RES... Add files via upload

Developing RESTful APIs for IMS Transactions.p... Add files via upload

master zCONNEE-Wildfire-Workshop / OpenAPI3 /

emitchj Delete admin

Developing RESTful APIs for Db2 REST Services.... Add files via upload

Developing RESTful APIs for a CICS Program.pdf Add files via upload

# IBM's Z Virtual Access (zVA) provides access to hand-on exercises

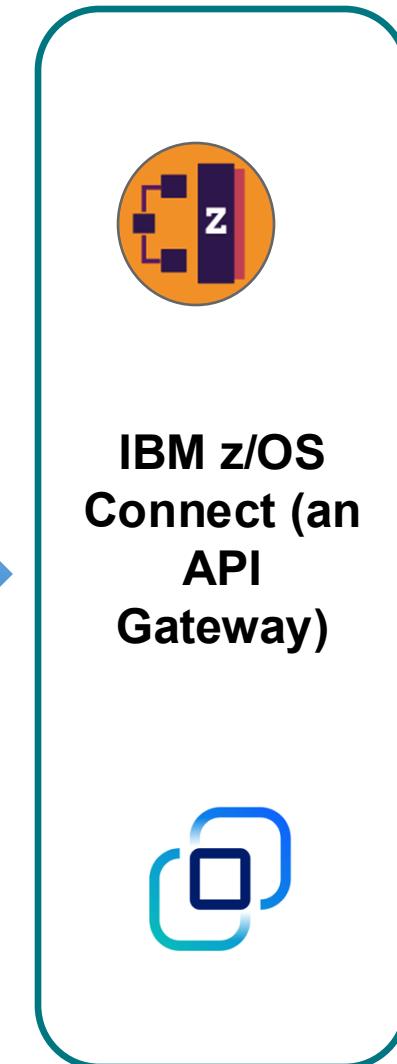


The screenshot displays a Windows desktop environment with two application windows open:

- zVA - Remote Desktop**: This window is titled "zVA - Remote Desktop" and contains the "zosexplorer - zva/package.xml - C:\Users\Administrator\zosexplorer - IBM Explorer for z/OS" tab. It features an "API Editor" interface with sections for "Describe your API" (Name: zva, Base path: /zva, Version: 1.0.0), "Contact Information", and a "Path" section with a "/newPath1" entry and four method definitions (POST, GET, PUT, DELETE).
- z/OS Connect Designer**: This window is titled "z/OS Connect Designer" and shows the URL "https://designer.washington.ibm.com:3451/zosConnect/designer/import-oas". It has a "Import OpenAPI document" interface with fields for "Drag and drop or select a file" (instructions: "OpenAPI Specification 3.0 supported (JSON or YAML file formats)") and "Specify a URL" (input field: "http://github.com/example/api-docs/openapi.json" and a "Import file" button).

The desktop taskbar at the bottom includes icons for File Explorer, Firefox, and Task View, along with system status indicators for battery level, signal strength, and date/time (9:17 AM, 1/29/2024).

# **z/OS Connect API provider support exposes z/OS resources to clients in the “cloud” via RESTful APIs**

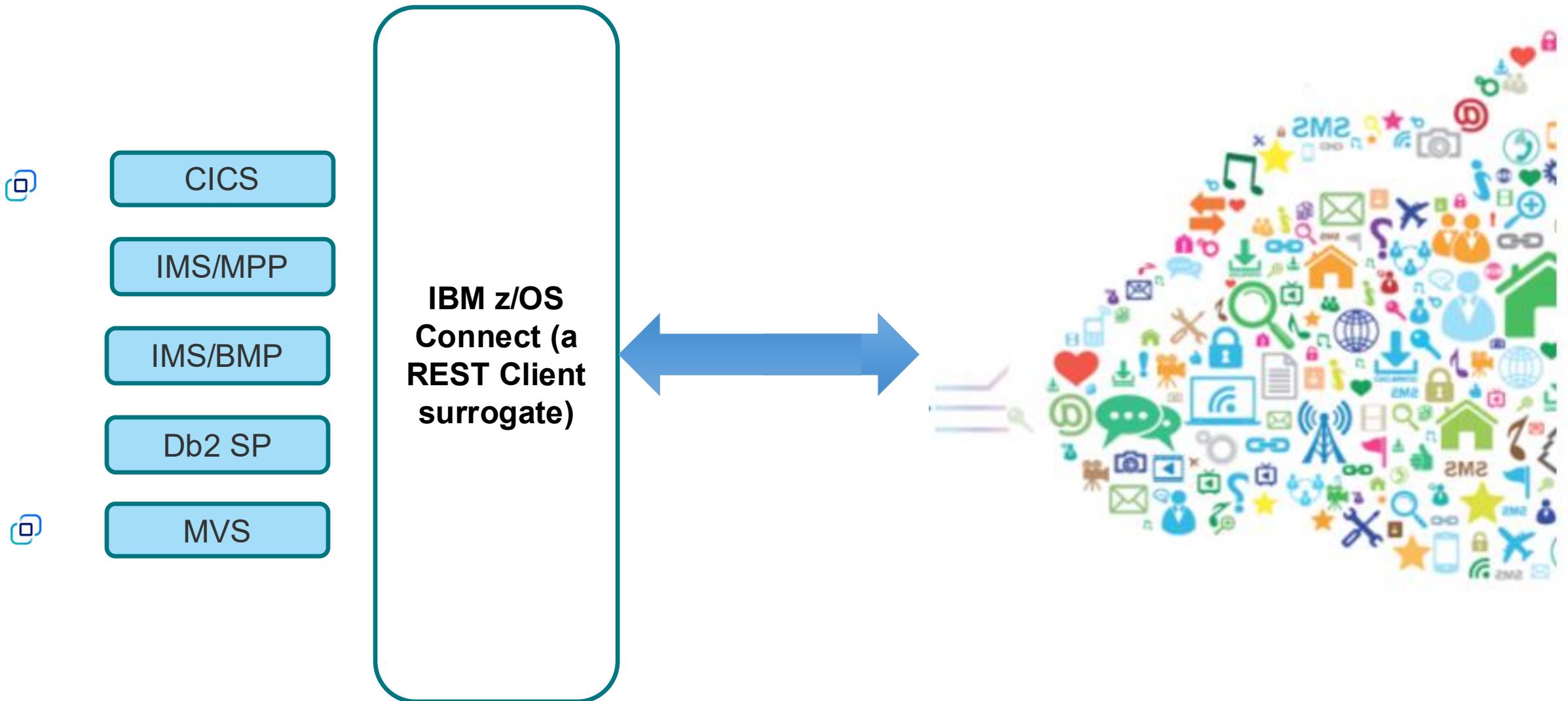


CICS	
IMS/TM	
IMS/DB	
Db2	
MQ	
MVS	
WAS	
IBM File Manager <sup>+</sup>	
HATS(3270) <sup>+</sup>	
IBM DVM <sup>+</sup>	
Custom*	

+ HCL and Rocket Software

\*Other Vendors or your own implementation

# Note: z/OS Connect API requester exposes external REST APIs in the “cloud” to z/OS applications



# **Let's start by defining REST and REST APIs**

And what makes an API “RESTful”?



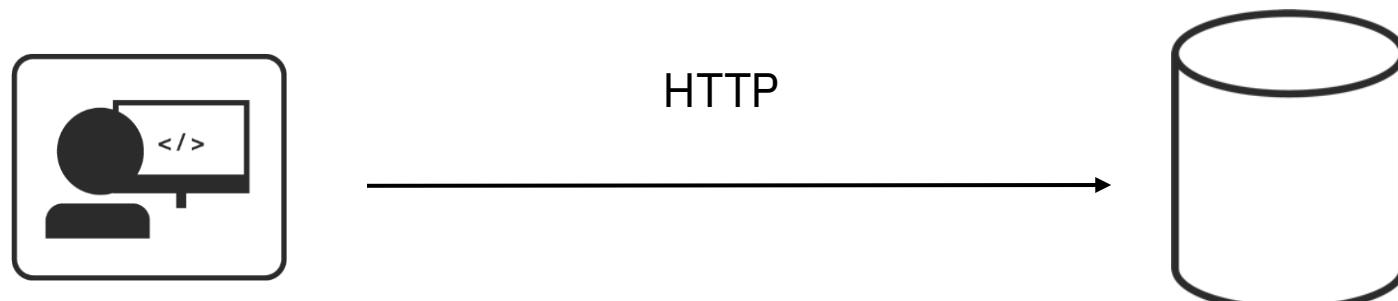
# REST is architectural programming style

**REST** is an acronym standing for **R**epresentational **S**tate **T**ransfer\*

An architectural programming style for **accessing** and **updating** data over the internet.

Typically using HTTP... but not all HTTP interfaces are “RESTful”.

Providing a simple and intuitive programming style for the end consumer (**the developer**).



\*Roy Fielding defined REST in his 2000 PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures" at UC Irvine. He developed the REST architectural style in parallel with HTTP 1.1 of 1996-1999, based on the existing design of HTTP 1.0 of 1996.



# All REST APIs follow some basic key principles

They use HTTP verbs to specify Create, Read, Update, Delete (CRUD) operations, for more information on CURD, see URL [https://en.wikipedia.org/wiki/Create,\\_read,\\_update\\_and\\_delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)

POST  
GET  
PUT  
DELETE

Uniform Resource Identifier (URI) path identifies the resource (s)

`http://<host>:<port>/path/parameter?name=value&name=value`

Use Path and Query parameters to refine the request

Uniform Resource Locator (URL) identifies the protocol, host and port and includes the URI path

GET `http://www.acme.com/customers/12345?personalDetails=true`  
RESPONSE: HTTP 200 OK  
BODY { "id" : 12345  
      "name" : "Joe Bloggs",  
      "address" : "10 Old Street",  
      "tel" : "01234 123456",  
      "dateOfBirth" : "01/01/1980",  
      "maritalStatus" : "married",  
      "partner" : "http://www.acme.com/customers/12346" }

The use JSON (Java Script Object Notation) to represent the data object in Request/Response message bodies



# What makes a REST API a RESTful API?

REST is an architectural style of development having these basic principles plus..

- It should be stateless (transaction management should be managed by the client)
- It should access and/or identify all server resources using only a **single** URI
- When performing the CRUD operations, the HTTP methods, GET, POST, PUT, DELETE should be used **as intended**
- It should return the result only in the form of **consistent and simple** JSON

When a REST API follows these basic principles, it is considered a RESTful API, whereas a REST API only follows some but not all the above principles

- The key is consistency, RESTful APIs follow these basic principles, REST APIs deviate from one or more of these principles



# RESTful Examples

**POST**

/account/ +

*(sends a JSON request message with Fred's details)*

**GET**

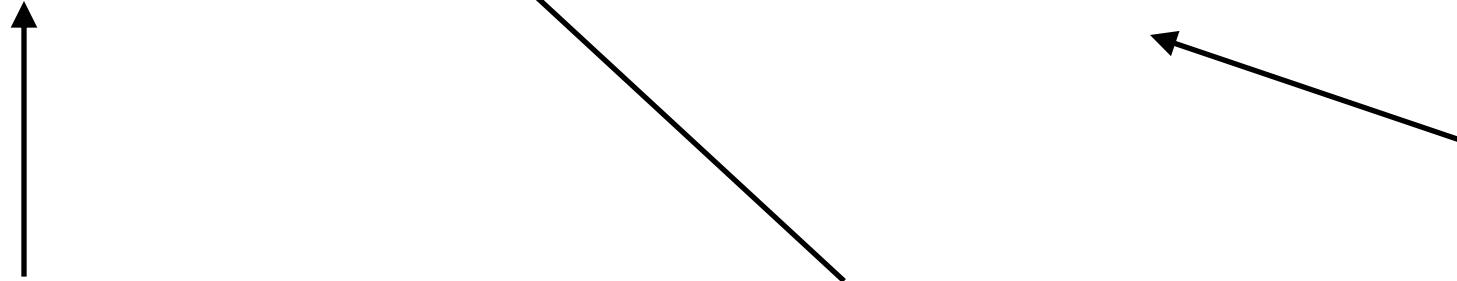
/account?person=Fred

*(returns a JSON response message with Fred's information)*

**PUT**

/account/Fred +

*(sends a JSON request message with changed information)*



**REST APIs are increasingly popular as an integration pattern because it is stateless, relatively lightweight, is relatively easy to program**

<https://martinfowler.com/articles/richardsonMaturityModel.html>



**Let's see one reason why,**

**Other solutions required that client code be tightly coupled with the resource since the resource's client code had to be integrated into the client application.**

## CICS client

```
24 COMMAREABUFFER commarea = new COMMAREABUFFER();
25 commarea.setNumb("111111");
26
27 //CICSConnection connection = new CICSConnection();
28 com.ibm.connector2.cics.ECIInteractionSpec interSpec = new com.ibm.connector2.cics.ECIInteractionSpec();
29
30 //      Create an interaction spec
31 interSpec.setFunctionName("CSCVINO");
32 interSpec.setCommareaLength(commarea.getSize());
33
34 //      Create a Connection Factory (non-Managed)
35 com.ibm.connector2.cics.ECIManagedConnectionFactory mcf = new com.ibm.connector2.cics.ECIManagedConnectionFactory();
36 mcf.setConnectionURL("tcp://ctfmvs09.rtp.raleigh.ibm.com");
37 //mcf.setConnectionURL("local:");
38 mcf.setServerName("NQA11C00");
39 mcf.setPortNumber("2006");
40
41 try {
42     //      Create an ECI connection factory using the nonMCF
43     com.ibm.connector2.cics.ECICreationFactory cf =
44         (com.ibm.connector2.cics.ECICreationFactory)mcf.createConnectionFactory();
45
46     //      Create a connection spec
47     com.ibm.connector2.cics.ECICreationSpec connectionSpec = new com.ibm.connector2.cics.ECICreationSpec();
48     connectionSpec.setUserName("userid");
49     connectionSpec.setPassword("p0ssword");
50
51     //      Obtain a connection from the connection factory
52     com.ibm.connector2.cics.ECICreation connection =
53         (com.ibm.connector2.cics.ECICreation)cf.getConnection(connectionSpec);
54
55     //      Create a interaction for the connection
56     com.ibm.connector2.cics.ECIInteraction interaction =
57         (com.ibm.connector2.cics.ECIInteraction) connection.createInteraction();
58
59     //      Execute the interaction
60     interaction.execute(interSpec, commarea, commarea);
61 }
```

## MQ client

```
PutMessage.java x
109
110     // Create a connection to the queue manager
111
112     if (bindingType.equals("client")) {
113         // Set MQ Environment variables/parameters
114         // qManager = "QMZB";
115         if (qManager.equals("QMZB")) {
116             com.ibm.mq.MQEnvironment.hostname = "mpx3";
117             com.ibm.mq.MQEnvironment.channel = "Client.to.QMZB";
118             com.ibm.mq.MQEnvironment.port = 1416;
119         }
120         if (qManager.equals("WMQ8")) {
121             com.ibm.mq.MQEnvironment.hostname = "localzos";
122             com.ibm.mq.MQEnvironment.channel = "Client.to.LOCALZOS";
123             com.ibm.mq.MQEnvironment.port = 1436;
124         }
125     }
126     com.ibm.mq.MQEnvironment.userID = userName;
127     com.ibm.mq.MQEnvironment.password = passWord;
128
129     qMgr = new MQQueueManager(qManager);
130     System.out.println("Success: Obtained a " + bindingType + " reference to the Queue manager: " + qManager);
131
132     // Set up the options on the queue we wish to open...
133     // Note. All WebSphere MQ Options are prefixed with MQC in Java.
134
135     int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;
136
137     // Now specify the queue that we wish to open,
138     // and the open options...
139
140     system_default_local_queue = qMgr.accessQueue(queueName,openOptions);
141     System.out.println("Success: Obtained a reference to the Queue: " + queueName);
142
143     sendMessage();
144
145     catch (MQException ex)
146     {
147         System.out.println("A WebSphere MQ error occurred : Completion code " +
148                           ex.completionCode + " Reason code " + ex.reasonCode);
149         if (ex.reasonCode == EMPTY_QUEUE)
150         {

```

The CICS client uses CICS provided Java classes (`com.ibm.connector2.cics.*`) and the MQ client uses MQ provided Java classes (`com.ibm.mq.*`)



# A key goal for REST is to have client code that is unaware of the resource's infrastructure

```
// Invoke the REST API using a GET method
URL url = new URL("https://wg31.washington.ibm.com:9453/cscvinc/employee/" + args[1]);
System.out.println("URL: " + url);
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Content-Type", "application/json");
byte[] bytesEncoded = Base64.encodeBase64(args[0].getBytes());
conn.addRequestProperty("Authorization", "Basic " + new String(bytesEncoded));
try {
    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
    }
    BufferedReader bufferReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
    while ((output = bufferReader.readLine()) != null) {
```

```
// Invoke the REST API using a GET method
URL url = new URL("https://wg31.washington.ibm.com:9453/db2/employee/" + args[1]);
System.out.println("URL: " + url);
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Content-Type", "application/json");
byte[] bytesEncoded = Base64.encodeBase64(args[0].getBytes());
conn.addRequestProperty("Authorization", "Basic " + new String(bytesEncoded));
try {
    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
    }
    BufferedReader bufferReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
    while ((output = bufferReader.readLine()) != null) {
```

These Java clients have the same application programming pattern.

- They provide an URL and a method and provide a request message.
- Then they use HTTP protocol to send a request to the API provider.
- A response message is returned after accessing a remote resource, e.g., a CICS program, a Db2 table, an IMS transaction or a MQ queue. The client application is unaware of the underlying infrastructure. No dependencies on coding for ECI, JDBC, OTMA, JMS, J2C, etc.

IMQ



## Just a FYI: This same pattern applies to the z/OS API requester solution

```
ZceeCICSGet.java ✘
55 // Invoke the REST API using a GET method
56 URL url = new URL("https://wg31.washington.ibm.com:9453/cscvinc/employee/" + args[1]);
57 System.out.println("URL: " + url);
58 HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
59 conn.setRequestMethod("GET");
60 conn.setRequestProperty("Content-Type", "application/json");
61 byte[] bytesEncoded = Base64.encodeBase64(args[0].getBytes());
62 conn.addRequestProperty("Authorization", "Basic " + new String(bytesEncoded));
63 try {
64     if (conn.getResponseCode() != 200) {
65         throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
66     }
67     BufferedReader bufferReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
68     while ((output = bufferReader.readLine()) != null) {
```

```
ZceeDb2Get.java ✘
52 // Invoke the REST API using a GET method
53 URL url = new URL("https://wg31.washington.ibm.com:9453/db2/employee/" + args[1]);
54 System.out.println("URL: " + url);
55 HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
56 conn.setRequestMethod("GET");
57 conn.setRequestProperty("Content-Type", "application/json");
58 byte[] bytesEncoded = Base64.encodeBase64(args[0].getBytes());
59 conn.addRequestProperty("Authorization", "Basic " + new String(bytesEncoded));
60 try {
61     if (conn.getResponseCode() != 200) {
62         throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
63     }
64     BufferedReader bufferReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
65     while ((output = bufferReader.readLine()) != null) {
```

**z/OS Connect API requester support provides the same pattern for COBOL applications running on z/OS. The application simply provides a URI path, a method and a request message in a working storage structure. Then passes the addresses of the working storage areas as a parameters in a COBOL CALL to the z/OS Connect server. The interaction with the remote API endpoint is encapsulated in the z/OS Connect server, therefore there is a minimum need for new or changed COBOL code.**



# These are the some of reasons why REST is popular today

## Ubiquitous Foundation

It's based on HTTP, which operates on TCP/IP, which is a ubiquitous networking topology.

## Relatively Lightweight

Compared to other technologies (for example, SOAP/WSDL), the REST/JSON pattern is relatively light protocol and data model, which maps well to resource-limited devices.

## Relatively Easy Development

Since the REST interface is so simple, developing the client involves very few things: an understanding of the URI requirements (path, parameters) and any JSON data schema.

## Increasingly Common

REST/JSON is becoming more and more a de facto "standard" for exposing APIs and Microservices. As more adopt the integration pattern, the more others become interested.

## Stateless

REST is by definition a stateless protocol, which implies greater simplicity in topology design. There's no need to maintain, replicate or route based on state.



# Let's start with how are REST API described

By using an OpenAPI specification document

The industry standard framework for describing REST APIs

The OpenAPI Initiative (OAI) was created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how APIs are described. As an open governance structure under the Linux Foundation, the OAI is focused on creating, evolving and promoting a vendor neutral description format. The OpenAPI Specification was originally based on the Swagger Specification, donated by SmartBear Software.

For more information see, [https://en.wikipedia.org/wiki/OpenAPI\\_Specification](https://en.wikipedia.org/wiki/OpenAPI_Specification)



## There are a number of tools available to aid in the creation and consumption of APIs

**Code Generation\*** - create stub code, to consume APIs from various languages, e.g., *Java data beans, COBOL copy books that describe the request and response messages.*

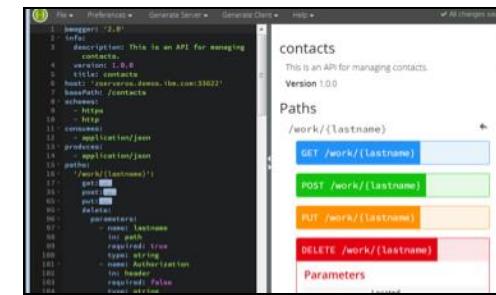


**Test UIs** - allows API consumers to easily browse and try APIs based on an OpenAPI document.



The screenshot shows the Swagger UI for a 'contacts' API. It lists four operations under the 'default' path: DELETE /work/{lastname}, GET /work/{lastname}, POST /work/{lastname}, and PUT /work/{lastname}. The 'DELETE' operation is highlighted in red. Below the operations, there is a 'VALID' button.

**Editors** - allows API developers to design their OpenAPI documents.



The screenshot shows the JSON Formatter interface displaying an OpenAPI specification for a 'contacts' API. The specification includes information about the API version (2.0), title ('contacts'), and contact details. It also defines paths for GET, POST, PUT, and DELETE requests to manage contacts, along with their respective parameters and descriptions.

\* z/OS Connect API Requester

+z/OS Connect, MQ REST support, Liberty, etc.

<https://swagger.io/>  
<https://jsonformatter.org/>

**Important** - You may have used or heard of the term Swagger with the use of APIs. Swagger was the original name of the specification but in 2016 the Swagger specification was renamed the OpenAPI Specification (OAS).



# What is the significance of the OpenAPI Specification to z/OS Connect?

The OpenAPI specification is the industry standard framework for describing REST APIs



- **z/OS Connect and Open API Specification 2, (formally known as Swagger 2.0), was supported initially by z/OS Connect**

Initially, accessing z/OS resources was the only desire for developing APIs .The interactions with the z/OS resources was driven by the layout of the CICS COMMAREA or CONTAINER, the IMS or MQ messages or the Db2 REST service.

- The details of the interactions with the z/OS resource determined the contents of the API request and response messages and the subsequent specification document.
- **z/OS Connect produces the specification document that describes the methods and request and response messages.**



- **z/OS Connect and Open API Specification 3, supported by z/OS Connect starting in March 2022 service, V3.0.55**

As companies mature their API strategy, they begin to introduce API governance boards to drive consistency in their API design. As more public APIs are created, government and industry standards bodies begin to regulate and drive for standardization. This drives the need for “API first” functional mapping capabilities within the integration platform. The external API design determined the layouts of the API request and response messages provided by the specification documents which was consumed by z/OS Connect to describe the z/OS resource interactions.

- The API details of the methods and layouts of request and response messages are provided in advance and access to the z/OS resource is driven by the API design
- **z/OS Connect consumes the specification document that describes the methods and request and response messages**

***Note: For our purposes, the terms OpenAPI 2.0 and Swagger 2.0 are interchangeable.***



# Contrast the Swagger 2 versus OpenAPI 3 specification differences

z/OS Connect  
OpenAPI2 tooling  
**produces** an  
OpenAPI 2  
specification  
document (aka  
OpenAPI 2.0),  
where the details of  
the methods and  
request/response  
messages in the API  
specification are  
driven by the nature  
of the z/OS resource  
(JSON format).

The image shows two side-by-side Notepad windows. The left window, titled 'cscvinc.json - Notepad', contains a Swagger 2 JSON specification. A red circle highlights the 'swagger: "2.0"' key at the top. The right window, titled 'cscvinc.yaml - Notepad', contains an OpenAPI 3 YAML specification. A red circle highlights the 'openapi: 3.0.1' key at the top. Both files describe a '/employee' endpoint with a 'get' method.

```

cscvinc.json - Notepad
File Edit Format View Help
{
  "swagger": "2.0",
  "info": {
    "description": "",
    "version": "1.0.0",
    "title": "cscvincapi"
  },
  "basePath": "/cscvincapi",
  "schemes": [
    "https",
    "http"
  ],
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "paths": {
    "/employee/{employee}": {
      "get": {
        "tags": [
          "cscvincapi"
        ],
        "operationId": "getCsvincSelectService",
        "parameters": [
          {
            "name": "Authorization",
            "in": "header",
            "required": false,
            "type": "string"
          },
          {
            "name": "employee",
            "in": "path",
            "required": true,
            "type": "string",
            "maxLength": 6
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "schema": {
              "$ref": "#/definitions/getCscvincSelectService_response_200"
            }
          },
          "404": {
            "description": "Not Found"
          }
        }
      }
    }
  }
}

cscvinc.yaml - Notepad
File Edit Format View Help
openapi: 3.0.1
info:
  title: cscvinc
  description: ""
  version: 1.0.0
servers:
- url: /cscvinc
x-ibm-zcon-roles-allowed:
- Manager
paths:
  /employee:
    get:
      tags:
        - cscvinc
      operationId: postCscvincInsertService
      x-ibm-zcon-roles-allowed:
        - Staff
      parameters:
        - name: Authorization
          in: header
          schema:
            type: string
      requestBody:
        description: request body
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/postCscvincInsertService_request'
            required: true
      responses:
        200:
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/postCscvincInsertService_response_200'
              x-codegen-request-body-name: postCscvincInsertService_request
      /employee/{employee}:
        get:
          tags:
            - cscvinc
          operationId: getCsvincSelectService
          x-ibm-zcon-roles-allowed:
            - Staff
          parameters:
            - name: Authorization
              in: header
              schema:
                type: string

```

z/OS Connect  
OpenAPI3 tooling  
**consumes** an  
OpenAPI3 specification  
document (aka OSA3)  
and the details of the  
methods and  
request/response  
messages are driven by  
the API specification  
(YAML format\*) and  
not the nature of the  
z/OS resource. Also,  
JSONata can be  
used to augment  
the API response

\*Yet Another Markup Language



## **What value does zOS Connect Add for REST API clients?**

Truly RESTful APIs to and from your mainframe.

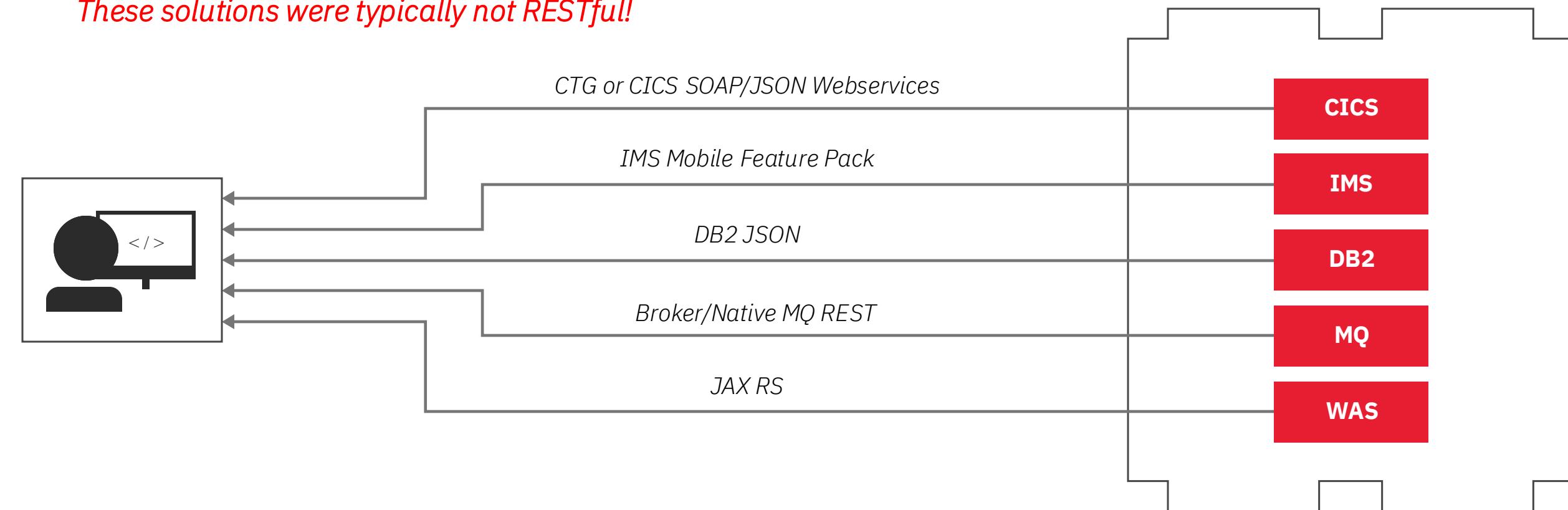


# There was support for REST to z/OS before z/OS Connect but..

Completely different configuration and management for each subsystem

Multiple endpoints for developers to call/maintain for access

*These solutions were typically not RESTful!*



# z/OS Connect provided a single-entry point to z/OS resources

Without the need to write new code or change any existing code.

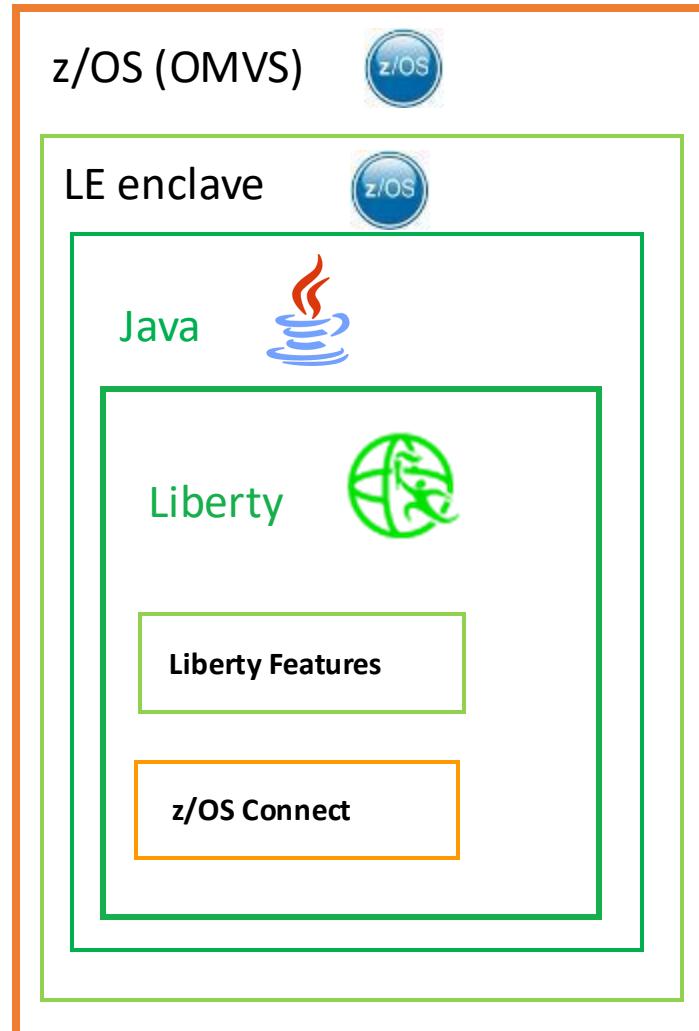
z/OS Connect provides

- Single Configuration Administration
- Single Security Administration
- With sophisticated mapping of truly RESTful APIs to existing mainframe and services data without writing any code.





# A z/OS Connect instance executes in a Liberty environment on z/OS



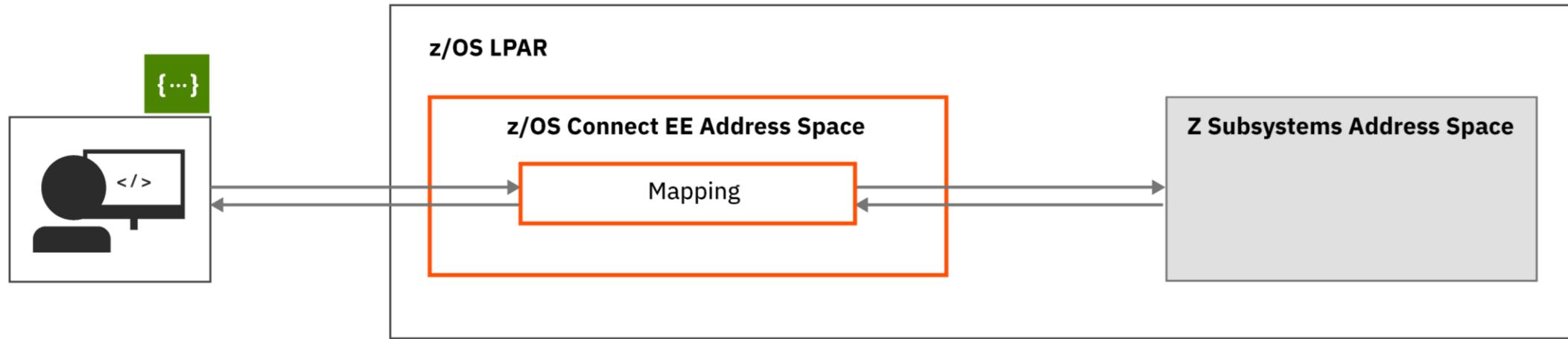
- That is started by an OMVS script that starts the Java environment (in an OMVS process).
- Runs in a Language Environment (LE) enclave configured to support OMVS and Java processes.
- Liberty is a Java application and on z/OS, there are features that provide access to z/OS services and facilities (e.g., SAF, WLM, RRS, SMF, JCL, started tasks, etc.).
- The Liberty Java application provides an execution environment for multiple concurrent Java applications.

Knowing the different layers and their relationship is important regarding

- Understanding which layer a configuration options, e.g., environment variables, etc., applies.
- Monitoring and understanding the health of the server
- Performing problem determination and performance tuning

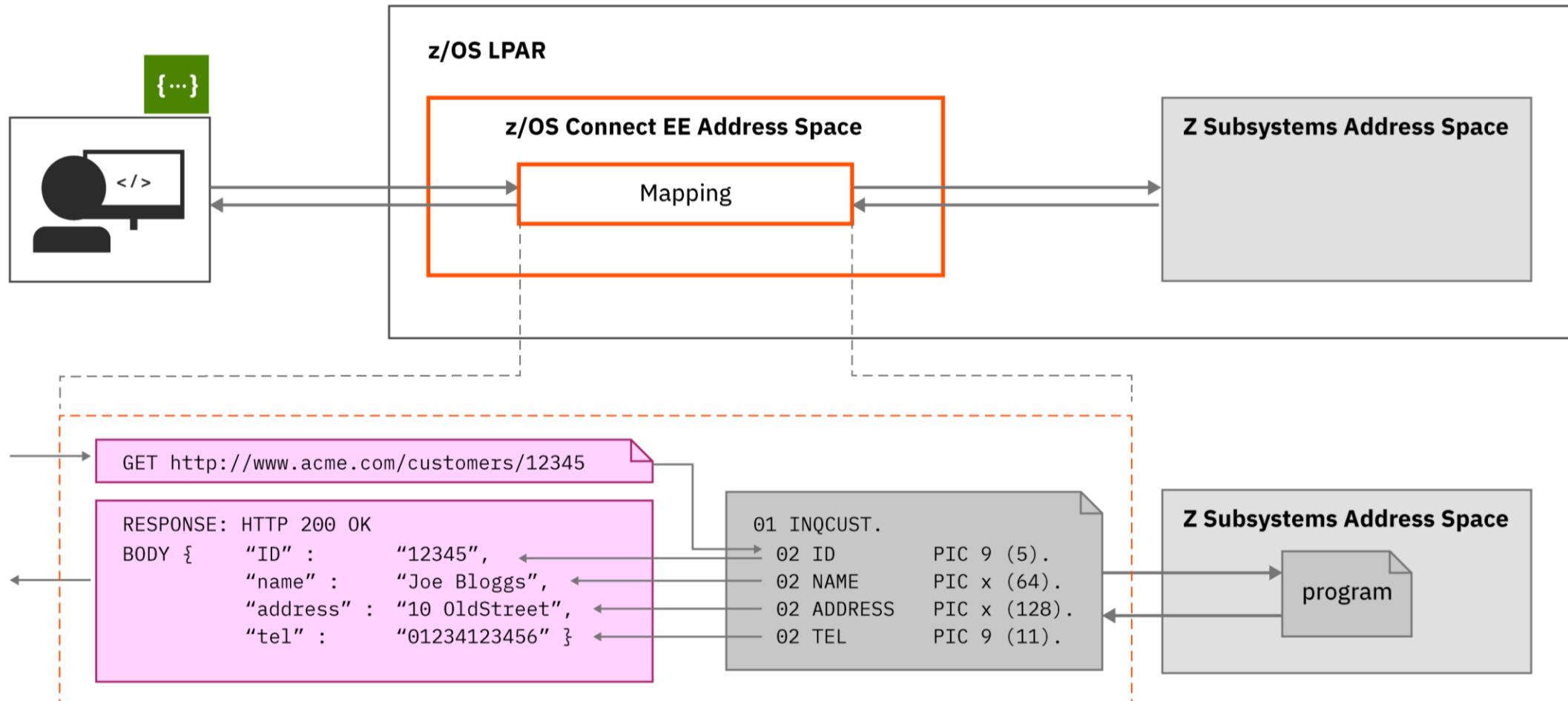
# Other than a RESTful interface, what else does z/OS Connect provide?

Automatic data mapping/transformation of the JSON message to a format the target's subsystem expects\*



\* Most z/OS subsystems depend on information in a serial data format and do not normally work with JSON request/response messages. Examples of serialized messages are CICS COMMAREAAs and CONTAINERS, IMS or MQ messages, or records stored in sequential or VSAM data sets. Data mapping and transformation refers to the process of converting JSON messages to a serialized layout (e.g., sequentially arranged in storage).

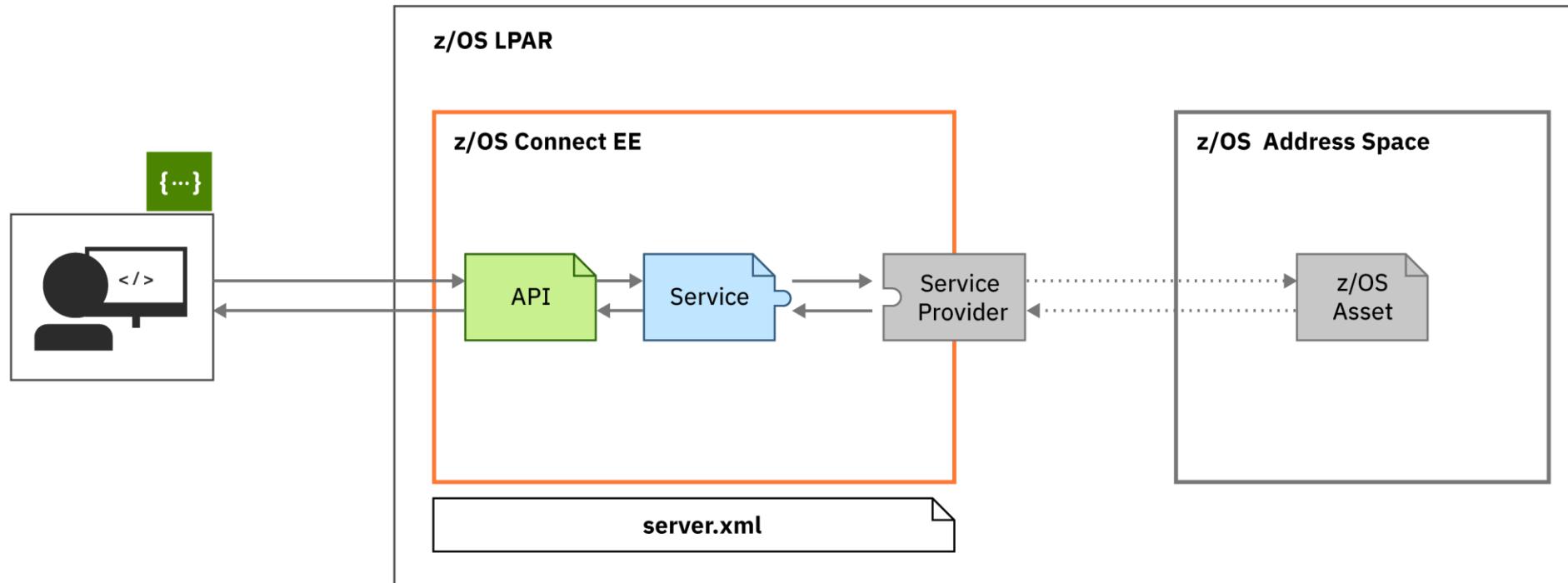
# Data mapping and transformation – a closer look



# Accessing an z/OS resource using an Open API 2 API



- z/OS Connect OpenAPI 2 APIs are developed using an Eclipse based API Toolkit to create Service Archive (SAR) and API ARchive (AAR) files.

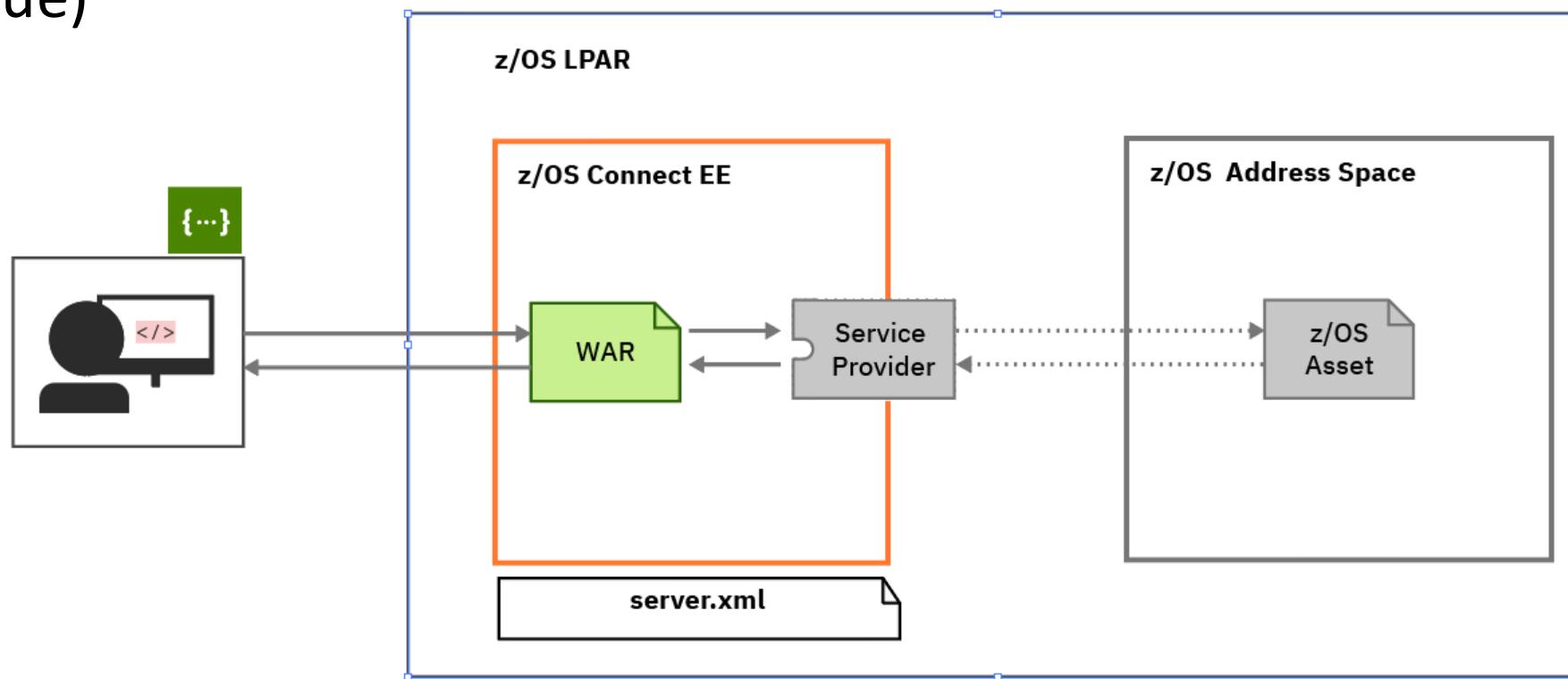


- The API is ready to be consumed and requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS resource (e.g., CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific instance of a resource (e.g., host and port, security)

# Accessing a z/OS resource using an Open API 3 API



- z/OS Connect OpenAPI 3 APIs are developed using the z/OS Connect Designer web browser tool to create Web ARchive (WAR) files (a traditional Java packaging technique)



- The WAR provides the RESTful interface is ready to be consumed by a client and it requires the client to have no knowledge that a z/OS resource is being accessed as well as the mapping and transformation for accessing the resource.
- The Service Provider configuration is tightly coupled to a specific instance of a resource (e.g., host and port)



# **z/OS Connect OpenAPI Tooling**

# z/OS Connect's OpenAPI 2 Palette versus the OpenAPI 3 API Designer



z/OS Connect API Toolkit (Eclipse)

The screenshot shows the API Editor interface. At the top, there are fields for 'Name' (new), 'Base path' (/new), and 'Version' (1.0.0). Below these are sections for 'Contact Information' and 'Path'. A path '/newPath1' is defined with four method mappings: POST, GET, PUT, and DELETE. Each method has a 'Service...' button, a 'Mapping...' button, and up/down arrows for reordering.

z/OS Connect Designer (Designer Container)

The screenshot shows the IBM z/OS Connect Designer interface. On the left, a sidebar displays the structure of the API specification, including 'Paths', 'z/OS Assets', 'Components', and other sections. The main panel shows two paths: '/employee' and '/employee/{employee}'. For each path, it lists supported operations (POST, GET, PUT, DELETE) and their status (Ready). The '/employee' path has a single POST operation, while the '/employee/{employee}' path has all four operations (GET, PUT, DELETE, POST).

The API toolkit is used to define the URI paths and methods.

The API specification provides the predefined URI Paths and methods.

# A distinction between OpenAPI 2 and OpenAPI 3 response message



The screenshot shows the Eclipse-based z/OS Connect API Toolkit interface. A red circle highlights the 'Body - cscvincSelectServiceOperationResponse' section, which contains fields like 'cscvincContainer', 'response', 'CEIBRESP', 'CEIBRESP2', 'USERID', and 'filea'. Below this is the 'HTTP Response' section, which includes 'HTTP Headers' and 'Body - cscvincSelectServiceOperationResponse'.



Eclipse based - z/OS Connect API Toolkit -  
The contents of the API's response message are  
directly derived from the fields exposed in  
the z/OS resource's response.

The screenshot shows the z/OS Connect Designer web browser interface. A red circle highlights the 'Body' section of the mapping configuration, which maps fields from the z/OS Asset response into the API response. It shows mappings for 'summary.message' (containing 'abc Record (NUMB) successfull retrieved by USERID'), 'detail.cscvincSelectServiceOperationResponse.response.CEIBRESP' (containing '123'), 'detail.cscvincSelectServiceOperationResponse.response.CEIBRESP2' (containing '123'), 'detail.cscvincSelectServiceOperationResponse.response.USERID' (containing 'abc'), 'detail.cscvincSelectServiceOperationResponse.filea.employeeNumber' (containing 'abc (NUMB)'), 'detail.cscvincSelectServiceOperationResponse.filea.name' (containing 'abc (NAME)'), 'detail.cscvincSelectServiceOperationResponse.filea.address' (containing 'abc (ADDRX)'), 'detail.cscvincSelectServiceOperationResponse.filea.phoneNumber' (containing 'abc phone'), 'detail.cscvincSelectServiceOperationResponse.filea.date' (containing 'abc (DATEX)'), 'detail.cscvincSelectServiceOperationResponse.filea.amount' (containing 'abc (AMOUNT)'), and 'detail.cscvincSelectServiceOperationResponse.filea.comment' (containing 'abc (COMMENT)').



Web browser - z/OS Connect Designer –  
The contents of the API's response message are derived  
from the z/O resource's response and JSONata coding,  
see URL <https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=connect-what-is-jsonata> for more  
information on JSONata. Note that there maybe  
additional fields, e.g., *message* that are not in the z/OS  
resource's response message.



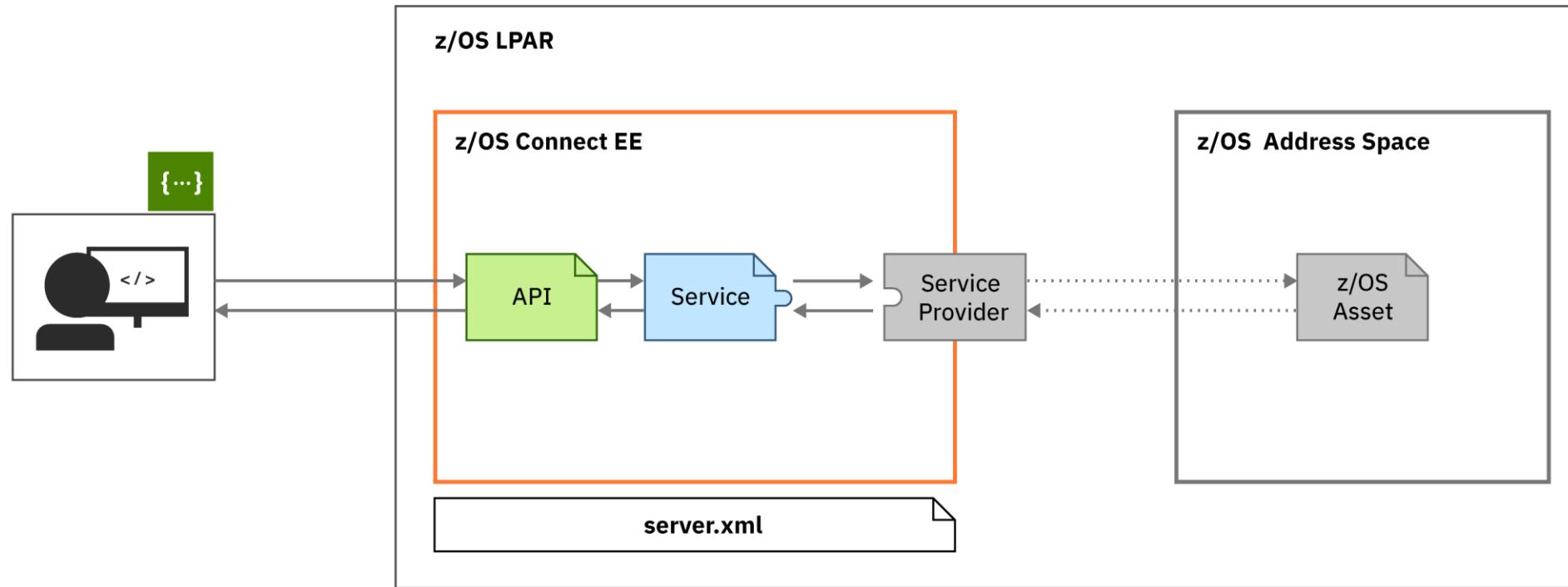
## OpenAPI 2

Creating **Services** using the Eclipse Toolkit



# Accessing a z/OS resource with an Open API 2 API

z/OS Connect OpenAPI 2 uses separate plug-and-play components



- The API provides the RESTful interface is ready to be consumed by a client and it requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS resource (e.g., CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific instance of a resource (e.g., host and port)

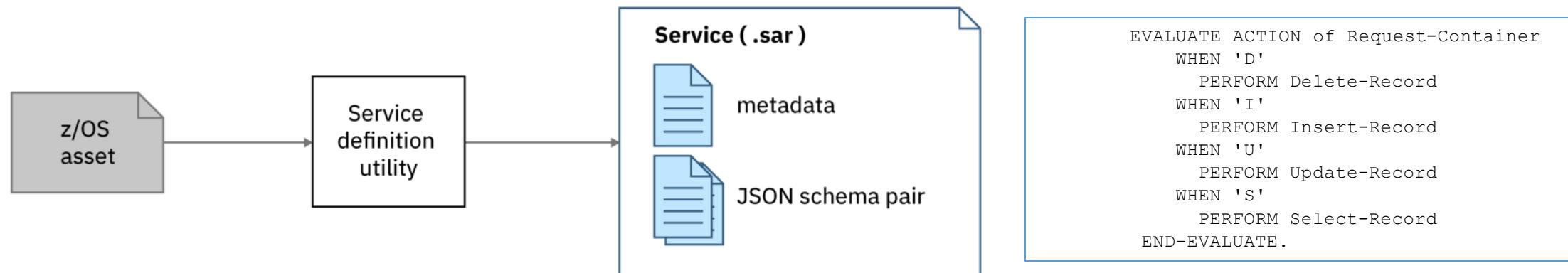


# Describing the interaction (service) with the z/OS resource

Start by creating a service to represent an interaction with the z/OS resource

To start mapping an API, z/OS Connect needs a representation of the underlying z/OS application: in a **Service Archive file (.sar)**.

The metadata consists of data mapping XML and provider specific configuration information



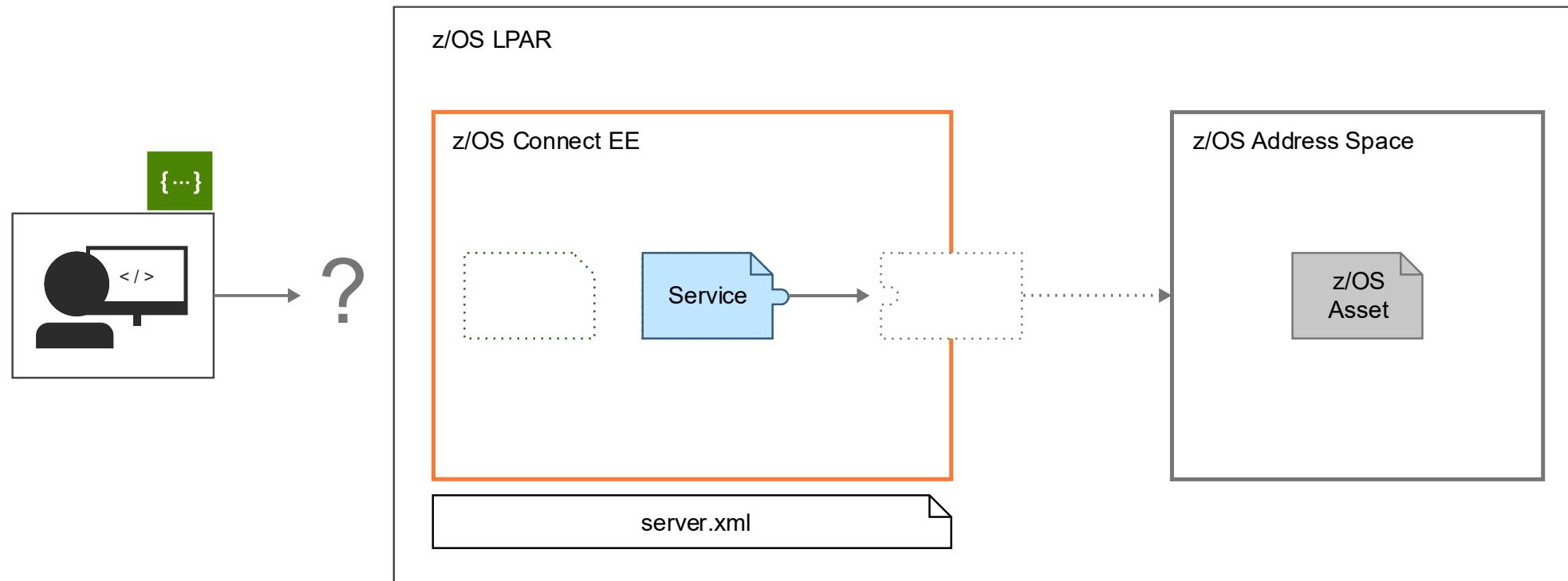
Use a system-appropriate utility to generate a service archive file for the z/OS application

- Eclipse based - API Toolkit (CICS, IMS TM, IMS DB, Db2 and MQ)
- Command line - z/OS Connect Build Toolkit (MVS Batch, IBM File Manager and HATS)
- Eclipse based - DVM Toolkit



# Deploy and export the service archive

Deploy to the server and export the service archive file for use in the API



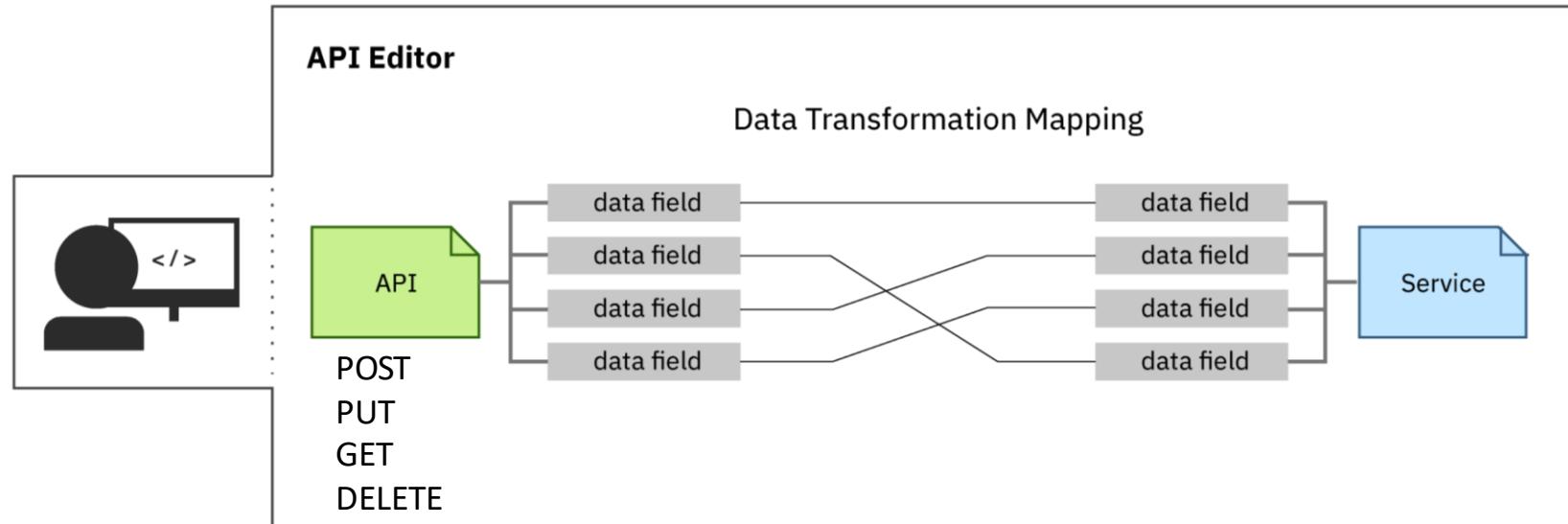
Use the right-click deploy in **the API toolkit**

- Deploy the service archive to the z/OS environment.
- Export the service archive to either another Eclipse project or a file system.



# Develop an API using the service interactions

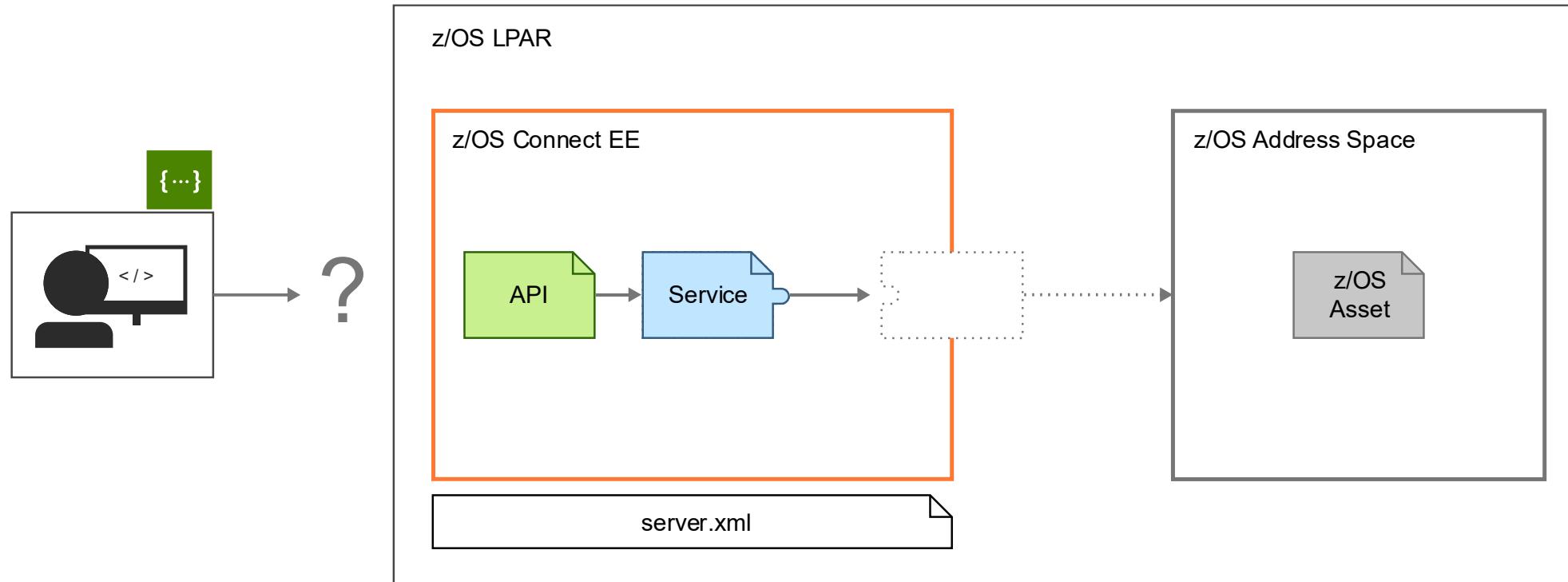
Import the service archive file it to create an API that consumes the service



- Import the service archive into the **API toolkit** and start designing the RESTful API.
- Provides additional data mapping
- Use the editor to describe the API and how it maps to underlying services.
- Add logic to interpret the results to add support for multiple HTTP response code

# Deploy the Open API 2 API

Deploy the API archive file to the server

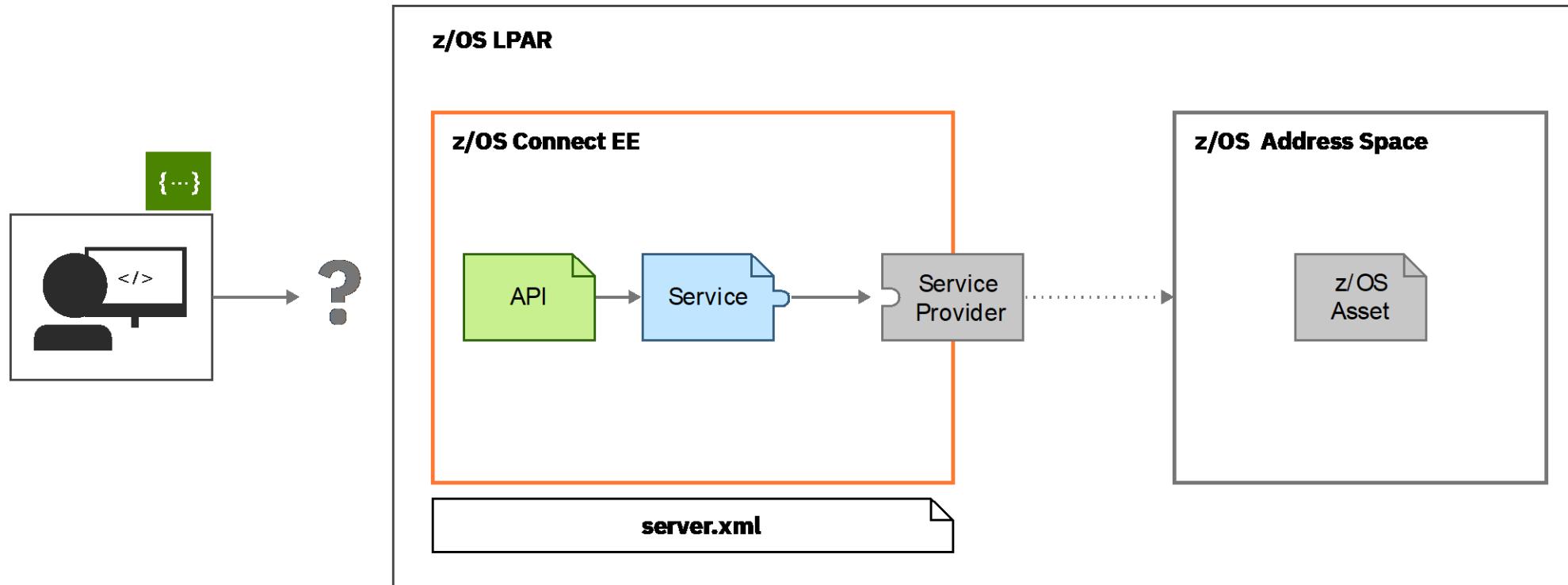


Deploy your API using the right-click deploy in **the API toolkit**



# Configure access to the z/OS subsystem

Configure the service provider

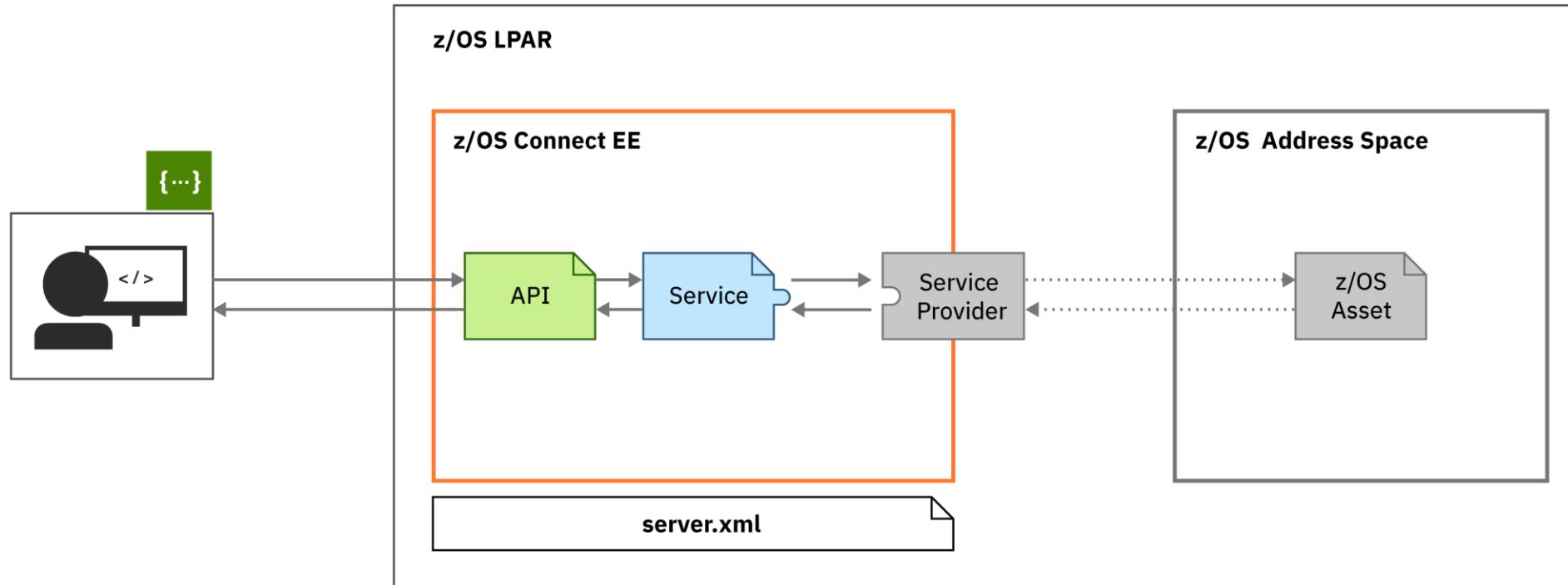


Configure the system-appropriate service provider to connect to your backend system in your **server.xml**.



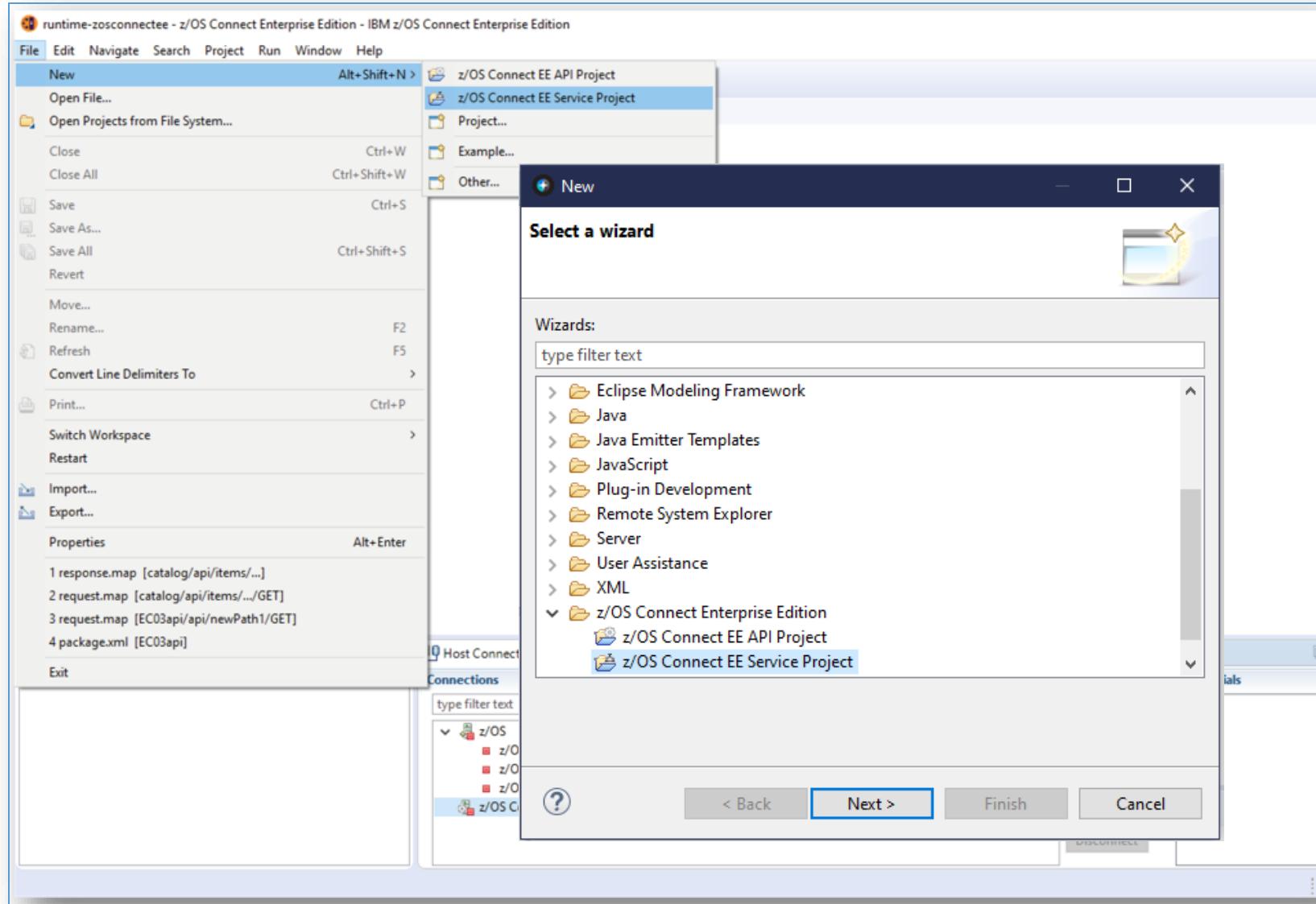
# Results: complete RESTful access to a z/OS resource

Done



- The API is ready to be consumed and requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS Asset (e.g., CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific instance of a resource (e.g., host and port, security)

# API toolkit – Creating Services for CICS, IMS TM, IMS DB, Db2 and MQ



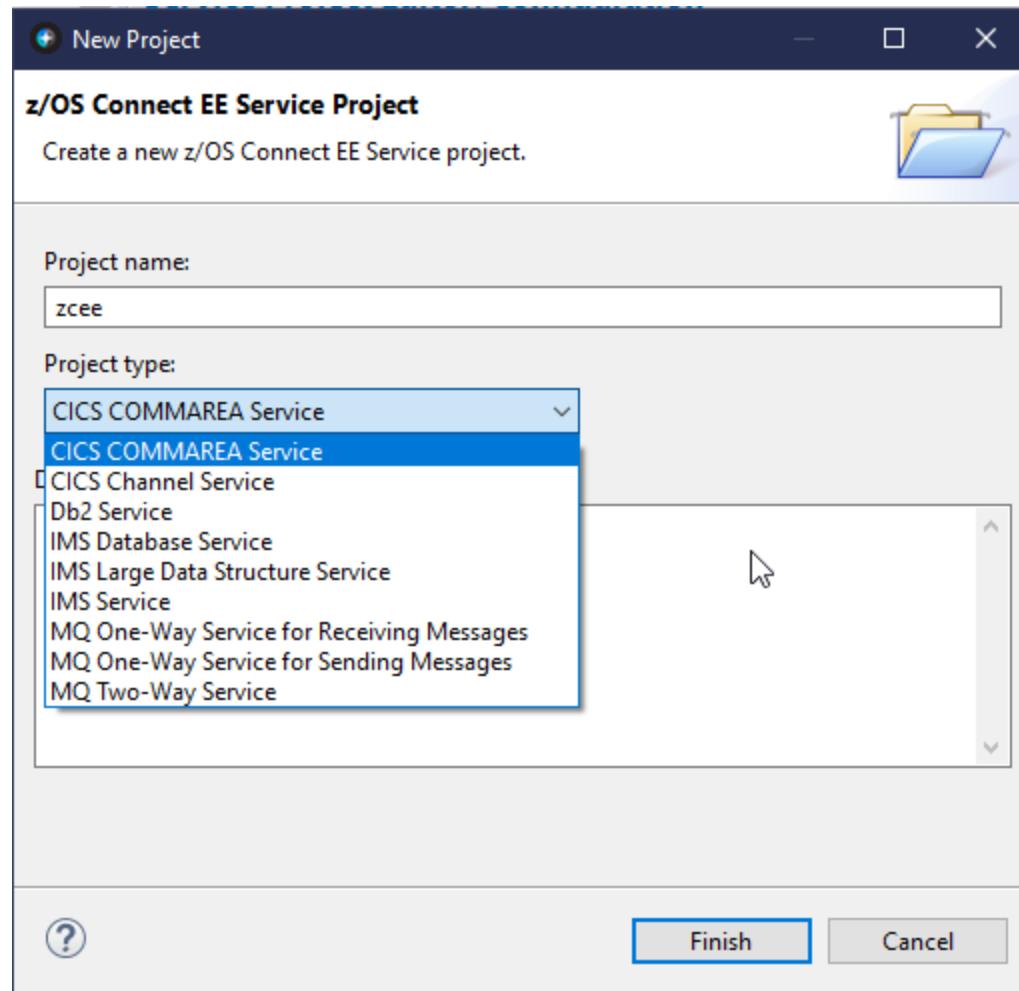
Use the **API toolkit** to create services through Eclipse-based tooling.

Services are described as Eclipse **Projects**, so they can be easily managed in source control.



# API toolkit – Creating Services for CICS, IMS TM, IMS DB, Db2 and MQ

## Service creation – a common interface



- CICS services that can invoke almost any CICS programs accessed by EXEC CICS LINK request (COMMAREA or CHANNEL). See URL <https://www.ibm.com/docs/en/cics-ts/6.x?topic=dpl-exception-conditions-link-command> for a list of EXEC CICS APIs not allowed in a program when invoked using a CICS Dynamic Program Link request.
- Db2 services that invoke a Db2 REST service.
- IMS DB services that access an IMS database.
- IMS TM services that sends a messages on an IMS message processing region.
- MQ services that use MQ request/reply queues for two-way services or access a single queue for MQ PUTs and MQ GETs on a either a local or remote queue manager

A common interface for service creation, irrespective of back-end subsystem.

# You start by identifying the format of the messages (copy books) used by the application



WG31 - 3270

File Edit Settings View Communication Actions Window Help

Menu Utilities Compilers Help

BROWSE USER1.ZCEE.SOURCE(CSCVINC) - 01.01 Line 0000000000 Col 001 080  
Command ==> Scroll ==> PAGE

\*\*\*\*\* Top of Data \*\*\*\*\*

PROCESS CICS  
IDENTIFICATION DIVISION.  
PROGRAM-ID. CSCVINC.

ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.

\* Container names  
01 WS-STORAGE.  
05 Input-Length PIC S9(8) COMP-4.  
05 Channel-Name PIC X(16) VALUE SPACES.  
05 Output-String PIC X(72) VALUE SPACES.  
05 Container-Name PIC X(16).  
05 Token PIC S9(8) COMP-5 SYNC.  
05 Message-to-Write PIC X(90).  
05 CSMT-Output-Area PIC X(121).  
05 Abs-Time PIC S9(15) COMP-3.  
05 Current-Date PIC X(8).  
05 Current-Time PIC X(8).  
05 EIBRespCode PIC S9(8) COMP.  
COPY CSCCREQ .  
COPY CSCCRESP .

PROCEDURE DIVISION.  
MAIN-PROCESSING SECTION.  
    INITIALIZE Request-Container.  
    INITIALIZE Response-Container.  
\* Save current CICS userid  
    EXEC CICS ASSIGN USERID(USERID of Response-Container)  
        END-EXEC.  
  
\* Obtain name of channel  
    EXEC CICS ASSIGN CHANNEL(Channel-Name)  
        END-EXEC.  
  
\* If no channel passed in, terminate with abend code NOCN

MA A 04/015

Connected to remote server/host wg31 using lu/pool TCP00105 and port 23

Adobe PDF on Documents\\*.pdf

sandbox

File Edit Settings View Communication Actions Window Help

Menu Utilities Compilers Help

BROWSE USER1.ZCEE.SOURCE(CSCCREQ) - 01.00 Line 0000000000 Col 001 080  
Command ==> Scroll ==> PAGE

\*\*\*\*\* Top of Data \*\*\*\*\*

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).  
\*\*\*\*\* Bottom of Data \*\*\*\*\*

sandbox

File Edit Settings View Communication Actions Window Help

Menu Utilities Compilers Help

BROWSE USER1.ZCEE.SOURCE(CSCCRESP) - 01.00 Line 0000000000 Col 001 080  
Command ==> Scroll ==> PAGE

\*\*\*\*\* Top of Data \*\*\*\*\*

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).  
\*\*\*\*\* Bottom of Data \*\*\*\*\*

MA A 04/015

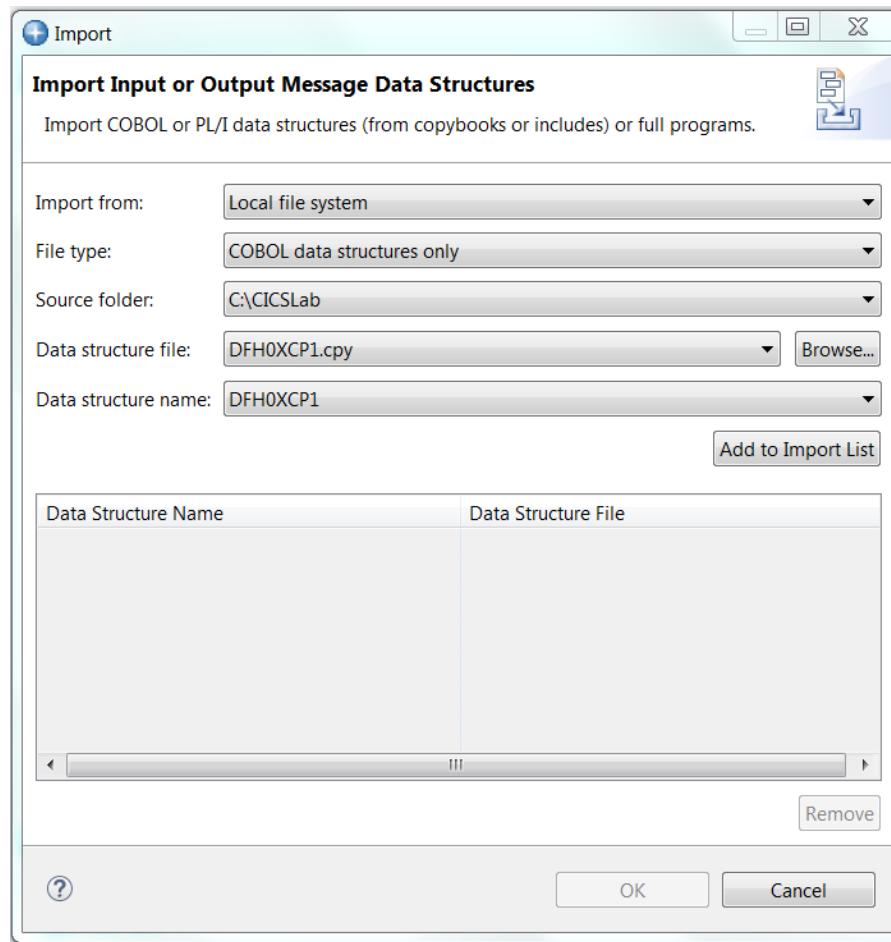
Connected to remote server/host sandbox using lu/pool TCP00003 and port 23

Adobe PDF on Documents\\*.pdf



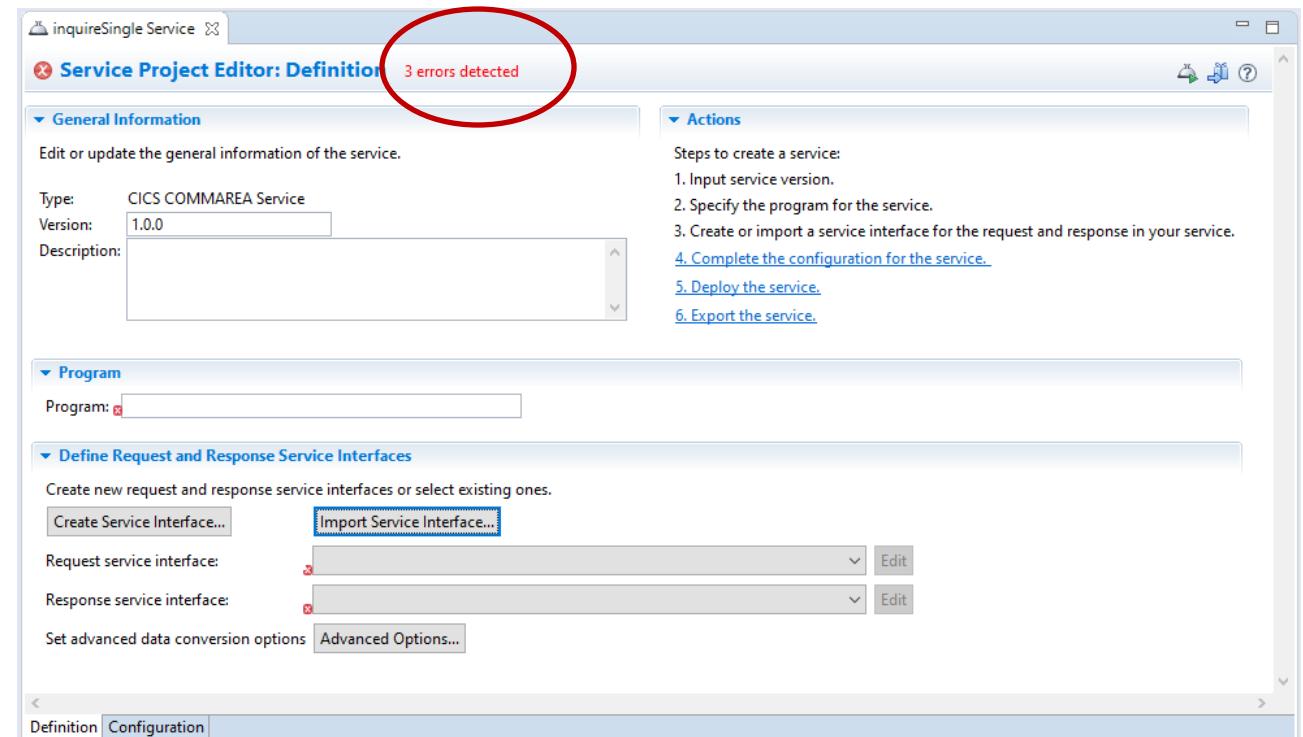
# API toolkit – Creating Services for CICS, IMS TM and MQ

for a message driven resource, e.g., a CICS COMMAREA or Container or IMS or MQ Message



Start by importing data structures into the service interface from the local file system or the workspace to create the request and response service interfaces.

The service interface supports complex data structures, including OCCURS DEPENDING ON and REDEFINES clauses.



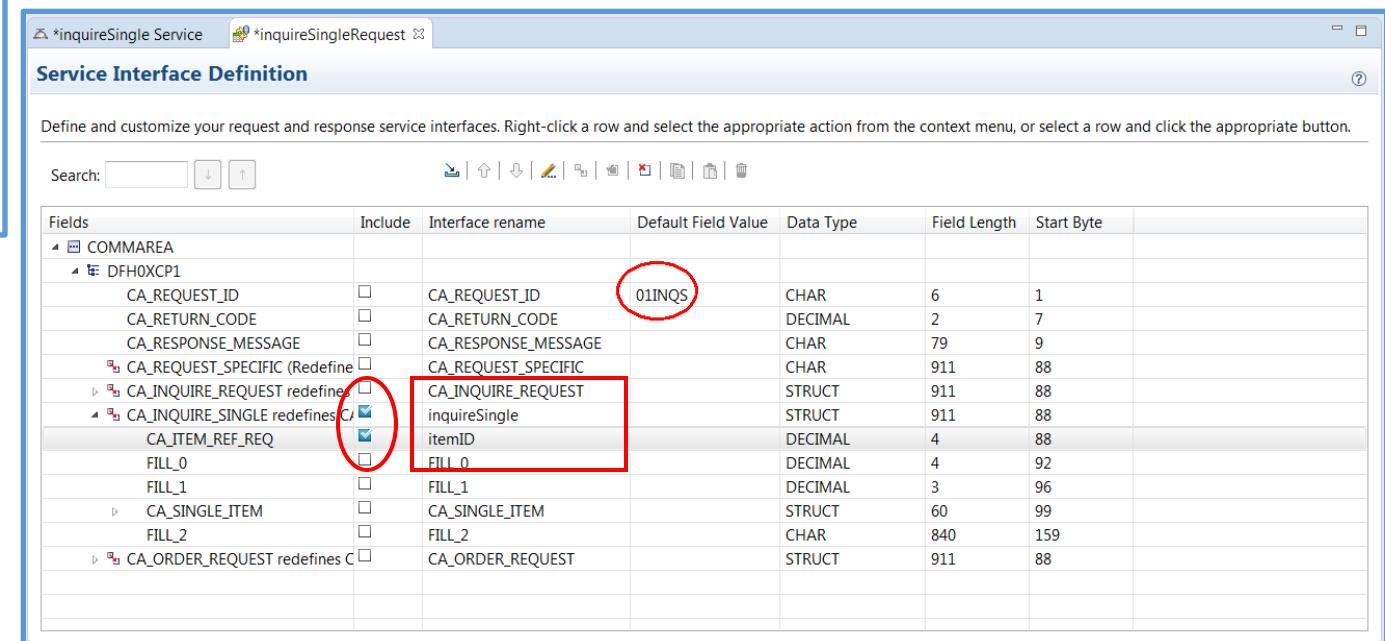


# API toolkit – Creating Services for CICS, IMS TM and MQ

The *service interface editor* allows creating and editing of a request service interface definition

```
*--  
* Check which operation is being requested  
*--  
  
* Uppercase the value passed in the Request Id field  
MOVE FUNCTION UPPER-CASE(CA-REQUEST-ID) TO CA-REQUEST-ID  
EVALUATE CA-REQUEST-ID  
    WHEN '01INQC'  
        Call routine to perform for inquire  
        PERFORM CATALOG-INQUIRE  
    WHEN '01INQS'  
        Call routine to perform for inquire for single item  
        PERFORM CATALOG-INQUIRE-SINGLE  
    WHEN '01ORDR'  
        Call routine to place order  
        PERFORM PLACE-ORDER  
    WHEN OTHER  
        Request is not recognised or supported  
        PERFORM REQUEST-NOT-RECOGNISED  
  
END-EVALUATE
```

See the imported data structure and then can **redact fields, rename fields**, and **add default values to fields** to make the service more consumable for an API developer.



Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID	01INQS	CHAR	6	1
CA_RETURN_CODE	<input type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	<input type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefine)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines C	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_INQUIRE_SINGLE redefines C	<input checked="" type="checkbox"/>	inquireSingle		STRUCT	911	88
CA_ITEM_REF_REQ	<input checked="" type="checkbox"/>	itemID		DECIMAL	4	88
FILL_0	<input type="checkbox"/>	FILL_0		DECIMAL	4	92
FILL_1	<input type="checkbox"/>	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	<input type="checkbox"/>	CA_SINGLE_ITEM		STRUCT	60	99
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840	159
CA_ORDER_REQUEST redefines C	<input type="checkbox"/>	CA_ORDER_REQUEST		STRUCT	911	88



# API toolkit – Creating Services for CICS, IMS TM, IMS DB and MQ

And creating and editing of a response message service interface definition

\*inquireSingleResponse

## Service Interface Definition

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Search:  Up Down Left Right New Edit Delete Print Copy Find Find Next Find Previous

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID		CHAR	6	1
CA_RETURN_CODE	<input checked="" type="checkbox"/>	returnCode		DECIMAL	2	7
CA_RESPONSE_MESSAGE	<input checked="" type="checkbox"/>	responseMessage		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefine)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefine	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_INQUIRE_SINGLE redefines CA_INQUIRE_REQUEST	<input checked="" type="checkbox"/>	inquireSingle		STRUCT	911	88
CA_ITEM_REF_REQ	<input type="checkbox"/>	CA_ITEM_REF_REQ		DECIMAL	4	88
FILL_0	<input type="checkbox"/>	FILL_0		DECIMAL	4	92
FILL_1	<input type="checkbox"/>	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	<input checked="" type="checkbox"/>	singleItem		STRUCT	60	99
CA_SNGL_ITEM_REF	<input checked="" type="checkbox"/>	itemReference		DECIMAL	4	99
CA_SNGL_DESCRIPTION	<input checked="" type="checkbox"/>	description		CHAR	40	103
CA_SNGL_DEPARTMENT	<input checked="" type="checkbox"/>	department		DECIMAL	3	143
CA_SNGL_COST	<input checked="" type="checkbox"/>	cost		CHAR	6	146
IN_SNGL_STOCK	<input checked="" type="checkbox"/>	inStock		DECIMAL	4	152
ON_SNGL_ORDER	<input checked="" type="checkbox"/>	onOrder		DECIMAL	3	156
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840	159
CA_ORDER_REQUEST redefine	<input type="checkbox"/>	CA_ORDER_REQUEST		STRUCT	911	88
CA_USERID	<input type="checkbox"/>	CA_USERID		CHAR	8	88
CA_CHARGE_DEPT	<input type="checkbox"/>	CA_CHARGE_DEPT		CHAR	8	96
CA_ITEM_REF_NUMBER	<input type="checkbox"/>	CA_ITEM_REF_NUMBER		DECIMAL	4	104
CA_QUANTITY_REQ	<input type="checkbox"/>	CA_QUANTITY_REQ		DECIMAL	3	108
FILL_3	<input type="checkbox"/>	FILL_3		CHAR	888	111

See the imported data structure and can **redact fields** and **rename fields**



# API toolkit – Creating Services for CICS

You can create multiple services definitions to the same resource

The screenshot shows two separate windows of the Service Interface Editor. Both windows have a header bar with tabs: 'cscvincSelectService Service' and 'cscvincSelectRequest'. The left window is titled 'Service Interface Editor' and contains a table of fields for the 'REQUEST\_CONTAINER' interface. The right window also has a 'Service Interface Editor' title and contains a table of fields for the same interface. A vertical blue sidebar runs between the two windows, with the text 'click the' at the top and 'Field Length' at the bottom. In the top-right cell of the first table (ACTION), there is a red circle around the value 'S'. In the top-right cell of the second table (ACTION), there is a red circle around the value 'I'.

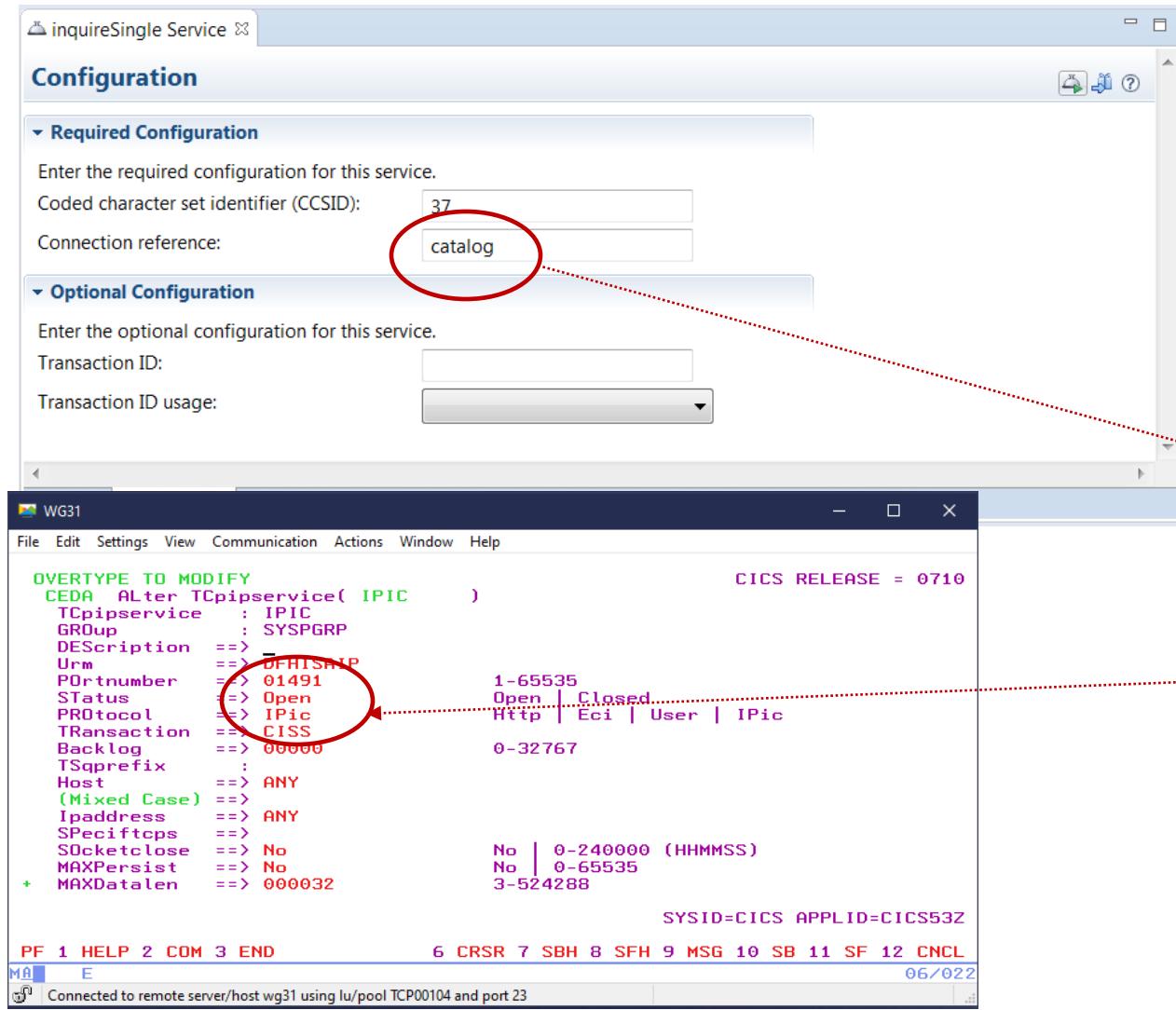
The screenshot shows the 'Service Project Editor: Definition' tab. It includes sections for 'General Information' (Type: CICS Channel Service, Version: 1.0.0), 'Actions' (a list of 6 steps), 'Program' (set to 'CSCVINC'), and 'Define Request and Response Service Interfaces' (Request service interface: 'cscvincSelectRequest.si', Response service interface: 'cscvincSelectResponse.si'). A red circle highlights the 'Program' field.

```
EVALUATE ACTION of Request-Container
  WHEN 'D'
    PERFORM Delete-Record
  WHEN 'I'
    PERFORM Insert-Record
  WHEN 'U'
    PERFORM Update-Record
  WHEN 'S'
    PERFORM Select-Record
END-EVALUATE.
```

# Accessing CICS uses a connection reference that uses CICS IP interconnectivity (IPIC)



The server.xml file is the key configuration file:



Features are functional building blocks. When configured here, that function becomes available to the Liberty server

```
catalog.xml
Design Source
1 <server description="CICS IPIC - catalog">
2
3 <!-- Enable features -->
4 <featureManager>
5   <feature>zosconnect:cicsService-1.0</feature>
6 </featureManager>
7
8 <zosconnect_cicsIpiconnection id="catalog">
9   host="wg31.washington.ibm.com"
10  port="1491"
11  transid="CSMI"
12  transidUsage="EIB_AND_MIRROR"/>
13
14 </server>
15
```

Define IPIC connection to CICS

# Tech-Tip : Accessing a CICS program – Transaction ID Usage



cscvincSelectService Service X

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Coded character set identifier (CCSID): 37

Connection reference: cscvinc

Optional Configuration

Enter the optional configuration for this service.

Transaction ID: MIJO

Transaction ID usage: EIB\_AND\_MIRROR (highlighted with a red circle)

Bidi configuration reference:

Use context containers:

Context containers HTTP headers:

Add another +

Definition Configuration

EIB\_ONLY

```
WG31 - 3270
File Edit Settings View Communication Actions Window Help
TRANSACTION: CSMI PROGRAM: DFHMIRS TASK: 0008501 APPLID: CICS53Z DISPLAY: 00
STATUS: PROGRAM INITIATION
```

```
EIBTIME = 104730
EIBDATE = 0122050
EIBTRNID = 'CSMI'
EIBTASKN = 8501
EIBTRMID = '/ABX'

EIBCPSON = 0
EIBCALEN = 0
EIBAID = X'00'
EIBFN = X'0000'
EIBRCODE = X'000000000000
+ EIBREQID = '.....'
```

```
ENTER: CONTINUE
PF1 : UNDEFINED PF2 :
PF4 : SUPPRESS DISPLAYS PF5 :
PF7 : SCROLL BACK PF8 :
PF10: PREVIOUS DISPLAY PF11:
```

```
M8 D
```

```
Connected to remote server/host wg31a using lu/pool TCP0012
```

```
WG31 - 3270
File Edit Settings View Communication Actions Window Help
TRANSACTION: MIJO PROGRAM: DFHMIRS TASK: 0008476 APPLID: CICS53Z DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
```

```
EXEC CICS LINK PROGRAM
PROGRAM ('CSCVINC')
SYNCONRETURN
CHANNEL ('Channel
NOHANDLE')
```

```
OFFSET:X'001CD6' LINE:
```

```
ENTER: CONTINUE
PF1 : UNDEFINED PF2 :
PF4 : SUPPRESS DISPLAYS PF5 :
PF7 : SCROLL BACK PF8 :
PF10: PREVIOUS DISPLAY PF11:
```

```
M8 D
```

```
Connected to remote server/host wg31a using lu/pool TCP
```

```
WG31 - 3270
File Edit Settings View Communication Actions Window Help
TRANSACTION: MIJO PROGRAM: CSCVINC TASK: 0008837 APPLID: CICS53Z
STATUS: PROGRAM INITIATION
```

```
EIBTIME = 181730
EIBDATE = 0122051
EIBTRNID = 'MIJO'
EIBTASKN = 8837
EIBTRMID = '/ABX'

EIBCPSON = 0
EIBCALEN = 0
EIBAID = X'00'
EIBFN = X'0E02' LINK
EIBRCODE = X'000000000000
+ EIBREQID = '.....'
```

```
EIBCPSON = 0
EIBCALEN = 0
EIBAID = X'00'
EIBFN = X'0E02' LINK
EIBRCODE = X'000000000000
+ EIBREQID = '.....'
```

```
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF
```

EIB\_AND\_MIRROR

```
WG31 - 3270
File Edit Settings View Communication Actions W
TRANSACTION: MIJO PROGRAM: DFH
STATUS: PROGRAM INITIATION
```

```
EIBTIME = 103914
EIBDATE = 0122050
EIBTRNID = 'MIJO'
EIBTASKN = 8482
EIBTRMID = '/ABX'

EIBCPSON = 0
EIBCALEN = 0
EIBAID = X'00'
EIBFN = X'0000'
EIBRCODE = X'000000000000
+ EIBREQID = '.....'
```

```
ENTER: CONTINUE
PF1 : UNDEFINED PF2 :
PF4 : SUPPRESS DISPLAYS PF5 :
PF7 : SCROLL BACK PF8 :
PF10: PREVIOUS DISPLAY PF11:
```

```
M8 D
```

```
Connected to remote server/host wg31a using lu/pool TCP
```

```
WG31 - 3270
File Edit Settings View Communication Actions W
TRANSACTION: MIJO PROGRAM: DFH
STATUS: ABOUT TO EXECUTE COM
```

```
EXEC CICS LINK PROGRAM
PROGRAM ('CSCVINC')
SYNCONRETURN
TRANSID ('MIJO')
CHANNEL ('Channel
NOHANDLE')
```

```
OFFSET:X'001CD6' LINE:
```

```
M8 D
```

```
Connected to remote server/host wg31a using lu/pool TCP
```

```
WG31 - 3270
File Edit Settings View Communication Actions W
TRANSACTION: MIJO PROGRAM: CSCVINC TASK: 0008949 APPLID: CICS53Z
STATUS: PROGRAM INITIATION
```

```
EIBTIME = 183613
EIBDATE = 0122051
EIBTRNID = 'MIJO'
EIBTASKN = 8949
EIBTRMID = '/ABD'

EIBCPSON = 0
EIBCALEN = 0
EIBAID = X'00'
EIBFN = X'0E02' LINK
EIBRCODE = X'000000000000
+ EIBREQID = '.....'
```

```
EIBCPSON = 0
EIBCALEN = 0
EIBAID = X'00'
EIBFN = X'0E02' LINK
EIBRCODE = X'000000000000
+ EIBREQID = '.....'
```

```
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DIS
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CON
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFIN
```

- **Transaction ID** attaches a CICS transaction (CSMI is the default) that starts the CICS DFHMIRS program.
- **Transaction ID Usage** attribute useful for:
  - Transaction security requirements
  - Db2 plan selection
  - Transaction classification and reporting



# API toolkit – Creating Services for IMS

## Creating a “GET” service interface request definition

```
*-----  
*      ROUTE TO REQUEST HANDLER  
*-----  
  
SPACE 1  
CLC KADD, IOCMD    IF COMMAND ADD ENTERED ?  
BE  TOADD          ...THEN, GOTO INSERT ENTRY  
CLC KUPD, IOCMD    IF COMMAND UPDATE ENTERED ?  
BE  TOUPD          ...THEN, GOTO UPDATE ENTRY  
CLC KDEL, IOCMD    IF COMMAND DEL ENTERED ?  
BE  TODEL          ...THEN, GOTO DELETE ENTRY  
CLC KDIS, IOCMD    IF COMMAND DIS ENTERED ?  
BE  TODIS          ...THEN, GOTO DISPLAY ENTRY  
CLC KTAD, IOCMD    IF TEST ADD WITH REPLY ?  
BE  TOTAD          ...THEN,  
B   INVREQ1        INVALID REQUEST
```

ivtnoDisplayRequest X

Service Interface Editor

Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

Search:  ↓ ↑

Fields

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length
ivtnoDisplayRequest					
Segment 1					
INPUT_MSG		phonebookRequest			
IN_LL	<input type="checkbox"/>	IN_LL		SHORT	2
IN_ZZ	<input type="checkbox"/>	IN_ZZ		SHORT	2
IN_TRANCODE	<input type="checkbox"/>	IN_TRANCODE		CHAR	10
IN_COMMAND	<input type="checkbox"/>	IN_COMMAND		CHAR	8
IN_LAST_NAME	<input checked="" type="checkbox"/>	lastName	IVTNO	CHAR	10
IN_FIRST_NAME	<input type="checkbox"/>	IN_FIRST_NAME	DISPLAY	CHAR	10
IN_EXTENSION	<input type="checkbox"/>	IN_EXTENSION		CHAR	10
IN_ZIP_CODE	<input type="checkbox"/>	IN_ZIP_CODE		CHAR	7

The service developer creates distinct services for each function.

DISPLAY (GET)  
DELETE (DELETE)  
ADD (POST)  
UPDATE (PUT)

ivtnoDisplayService Service X

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection profile: **IMSCONN**

Interaction profile: **IMSINTER** (circled in red)

Optional Configuration

Enter the optional configuration for this service.

IMS destination override:

Program name:

Definition Configuration

# IMS Connections and Interactions



ivtnoService Service Configuration

**Required Configuration**

Enter the required configuration for this service.

Connection profile: **IMSCONN**

Interaction profile: **IMSINTER**

**Optional Configuration**

Enter the optional configuration for this service.

IMS destination override:

Program name:

Overview Configuration

## Connection Profile

```
<server>
<imsmobile_imsConnection comment="" connectionFactoryRef="CF1"
connectionTimeout="-1" connectionType="IMSCONNECT" id="IMSCONN"/>
<connectionFactory containerAuthDataRef="Connection1_Auth" id="CF1">
<properties.gmoa hostName="wg31.washington.ibm.com"
portNumber="4000"/>
</connectionFactory>

<authData id="Connection1_Auth" password="encryptedPassword1"
user="userName1"/>
</server>
```

## Interaction Profile

```
<server>
<imsmobile_interaction comment="" commitMode="1" id="IMSINTER"
imsConnectCodepage="Cp1047" imsConnectTimeout="0"
imsDatastoreName="IVP1" interactionTimeout="-1"
ltermOverrideName="" syncLevel="0"/>
</server>
```

## IMS Connect HWSCFG

```
HWS=(ID=IMS14HWS,XIBAREA=100,RACF=Y,RRS=N)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,
TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP, ID=IVP1, MEMBER=HWSMEM, T MEMBER=OT
MAMEM)
```



# API toolkit – Creating Services for MQ

## Creating a MQ PUT (“POST”) service interface definition

The screenshot displays two windows from the API toolkit:

- Service Interface Editor:** Shows a table of fields for a service interface named "minilonServiceRequest". The table includes columns for Fields, Include, Interface Rename, Default Field Value, and Data Type. A red circle highlights the "Include" column for the "MINILOAN\_COMMAREA" section, and a red box highlights the "Interface Rename" column for the "loan application" section. The "Default Field Value" column contains values like "F" and "00005".
- Service Project Editor: Configuration:** Shows configuration settings for a service named "twoWay Service". A red circle highlights the "Request destination JNDI name" field, which contains "jms/requestQueue". Other fields include "Connection factory JNDI name: jms/qmgrCf", "Reply destination JNDI name: jms/replyQueue", "Wait interval: 2000", "MQMD format: MQSTR", "Coded character set identifier (CCSID): 37", "Is message persistent: false", "Reply selection: msgIDToCorrelID", and "Expiry: -1".

Again the service developer can then see the imported data structure and can **redact fields**, **rename fields**, and **add default values to fields** to make the service more consumable for an API developer.



# Using JMS to access MQ (One-Way)

mqGetService Service X

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection factory JNDI name: jms/qmgrCf

Destination JNDI name: jms/default

Coded character set identifier (CCSID): 37

Optional Configuration

Enter the optional configuration for this service.

Wait interval:

Message selector:

Definition Configuration

mqClient.xml

Read only Close

Design Source

```
<server description="MQ Service Provider">
<featureManager>
    <feature>zosconnect:mqService-1.0</feature>
</featureManager>
<variable name="wmqJmsClient.rar.location"
    value="/usr/lpp/mqm/V9R1M1/java/lib/jca/wmq.jmsra.rar"/>
<wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>
<zosconnect_services>
    <service name="mqPutService">
        <property name="useCallerPrincipal" value="false"/>
    </service>
</zosconnect_services>
<connectionManager id="ConMgr1" maxPoolSize="5"/>
<jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf">
    <connectionManagerRef>ConMgr1</connectionManagerRef>
    <properties.wmqJMS transportType="CLIENT"
        queueManager="ZMQ1"
        channel="LIBERTY.DEF.SVRCONN"
        hostName="wg31.washington.ibm.com"
        port="1422" />
</jmsConnectionFactory>
<jmsQueue id="q1" jndiName="jms/default">
    <properties.wmqJms
        baseQueueName="ZCEE.DEFAULT.MQZCEE.QUEUE"
        CCSID="37"/>
</jmsQueue>
</server>
```

# Using JMS to access MQ (Two-Way)



\*twoWay Service

## Service Project Editor: Configuration

**Required Configuration**

Enter the required configuration for this service.

Connection factory JNDI name: jms/qmgrCf

Request destination JNDI name: jms/requestQueue

Reply destination JNDI name: jms/replyQueue

Wait interval: 3000

MQMD format: MQSTR

Coded character set identifier (CCSID): 37

Is message persistent:

Reply selection: msgIDToCorrelID

Expiry: -1

Definition Configuration

mq.xml

Design Source

```
2 <featureManager>
3   <feature>zosconnect:mqService-1.0</feature>
4 </featureManager>
5
6 <variable name="wmqJmsClient.rar.location"
7   value="/usr/lpp/mqm/V9R1M1/java/lib/jca/wmq.jmsra.rar"/>
8 <wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>
9
10 <connectionManager id="ConMgr1" maxPoolSize="5"/>
11
12 <jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
13   connectionManagerRef="ConMgr1">
14   <properties.wmqJMS transportType="BROADCAST"
15     queueManager="QMZ1" />
16 </jmsConnectionFactory>
17
18 <jmsConnectionFactory id="qmgrCf2" jndiName="jms/qmgrCf2"
19   connectionManagerRef="ConMgr1">
20   <properties.wmqJMS transportType="CLIENT"
21     queueManager="QMQ1"
22       channel="LIBERTY.DEF.SVRCONN"
23       hostName="wg31.washington.ibm.com"
24       port="1422" />
25 </jmsConnectionFactory>
26
27 <jmsQueue id="q1" jndiName="jms/default">
28   <properties.wmqJms
29     baseQueueName="ZCONN2.DEFAULT.MQZCEE.QUEUE"
30     CCSID="37"/>
31 </jmsQueue>
32
33 <jmsQueue id="requestQueue" jndiName="jms/request">
34   <properties.wmqJms
35     baseQueueName="ZCONN2.TRIGGER.REQUEST"
36     targetClient="MQ"
37     CCSID="37"/>
38 </jmsQueue>
39
40 <jmsQueue id="replyQueue" jndiName="jms/replyQueue">
41   <properties.wmqJms
42     baseQueueName="ZCONN2.TRIGGER.RESPONSE"
43     targetClient="MQ"
44     CCSID="37"/>
45 </jmsQueue>
46
```



# API toolkit – Creating Services for IMS DB

## Creating a service project from the IMS Catalog

Service Project Editor: Definition

General Information

Type: IMS Database Service  
Version: 1.0.0  
Description:

Actions

Steps to create a service:

1. Input service version.
2. Specify the SQL command for the service.
3. Specify Database Connection Properties and Generate Service Interface.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

Enter or import SQL Command

SQL Command: `SELECT FIRSTNME, ZIPCODE, PHONENBR, A1111111  
FROM ATSVPA.A1111111 WHERE A1111111=?`

Specify Database Connection Properties and Generate Service Interface

Database Connection: wg31:5555 Database Name: DFSIVPA Generate Service Interface...

Definition Configuration

Use the IMS Catalog to assist with developing and testing SQL SELECT commands used for accessing IMS databases.

```
*-----  
* SEGMENT DESCRIPTION  
* ROOT ONLY DATABASE  
* BYTES 1-10 LAST NAME (CHARACTER) - KEY  
* BYTES 11-20 FIRST NAME (CHARACTER)  
* BYTES 21-30 INTERNAL PHONE NUMBER (NUMERIC)  
* BYTES 31-37 INTERNAL ZIP (CHARACTER)  
* BYTES 38-40 RESERVED  
*-----  
DBD NAME=IVPDB1,ACCESS=(HIDAM,OSAM)  
DATASET DD1=DFS IVD1,DEVICE=3380,SIZE=2048  
SEGM NAME=A1111111,PARENT=0,BYTES=40,RULES=(LLV,LAST),  
PTR=(TB,CTR)  
FIELD NAME=(A1111111,SEQ,U),BYTES=010,START=00001,TYPE=C  
FIELD NAME=FIRSTNME,BYTES=010,START=00011,TYPE=C  
FIELD NAME=PHONENBR,BYTES=010,START=00021,TYPE=C  
FIELD NAME=ZIPCODE,BYTES=7,START=00031,TYPE=C  
LCHILD NAME=(A1,IVPDB1I),POINTER=INDX,RULES=LAST  
DBDGEN  
FINISH  
END
```



# API toolkit – Creating Services for IMS DB

The Toolkit allows editing a service interface definitions for IM DB access\*

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length
selectByName_Response					
Segment 1					
Output Columns					
Result [0..*]	<input checked="" type="checkbox"/>	response			
FIRSTNME	<input checked="" type="checkbox"/>	result		ARRAY	0
ZIPCODE	<input checked="" type="checkbox"/>			CHAR	10
PHONENBR	<input checked="" type="checkbox"/>			CHAR	7
A1111111	<input checked="" type="checkbox"/>			CHAR	10
		firstName		CHAR	10
		zipCode		CHAR	7
		phoneNuber		CHAR	10
		lastName		CHAR	10

\*Using a slightly different process



# IMS Connection Factory in the server XML

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection profile: DFSIVPAConn

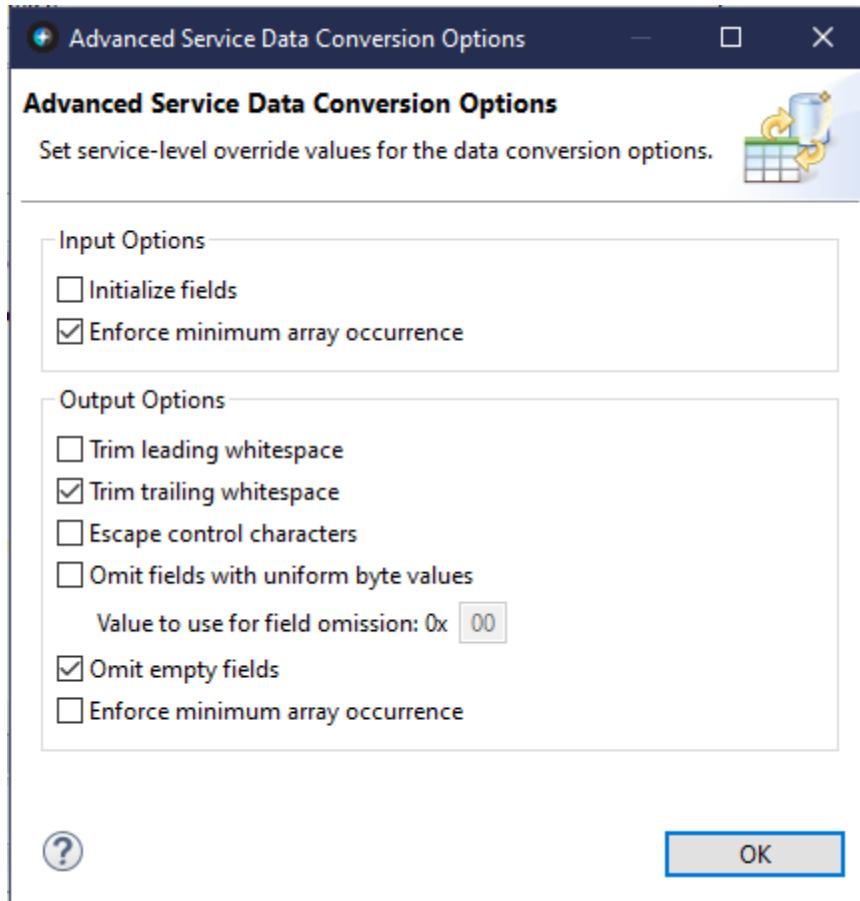
## ConnectionFactory

```
<connectionFactory id="DFSIVPAConn">
<properties.imsudbJLocal
    databaseName="DFSIVPA"
    datastoreName="IVP1"
    datastoreServer="wg31.washington.ibm.com"
    driverType="4"
    portNumber="5555"
    user="USER1"
    password="password"
    flattenTables="True"/>
</connectionFactory>
```

## IMS Connect HWSCFG

```
HWS=(ID=IMS14HWS,XIBAREA=100,RACF=N,RRS=N)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP, ID=IVP1, MEMBER=HWSMEM, TMEMBER=OTMAMEM)
IMSPLEX=(MEMBER=IMS14HWS, TMEMBER=PLEX1)
ODACCESS=(ODBMAUTOCONN=Y,
DRDAPORT=(ID=5555,PORTTMOT=6000), ODBMTMOT=6000)
```

# API toolkit – Advanced Data Conversion Options



## Request Messages:

- Initialize fields
- Enforce minimum array occurrence

## Response Messages:

- Trim leading whitespace
- Trim trailing whitespace
- Escape control characters
- Omit fields with uniform byte values
- Omit empty fields
- Enforce minimum array occurrence



# API toolkit – Creating Services for Db2

## Creating a service project from Db2 REST service by connecting to Db2

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

Import Db2 service from service manager

Db2 service manager connection: wg31:2446

Type to search...

Service Name	Version	Collection ID	Description
selectEmployee		SYSIBMSERVICE	Select an employee from table USER1.EMPLOYEE
deleteEmployee		zCEEService	Delete an employee from table USER1.E...
displayEmployee		zCEEService	Display an employee in table USER1.EM...
insertEmployee		zCEEService	Insert an employee into table USER1.EM...
selectByDepartments		zCEEService	Select employees by departments
selectByRole		zCEEService	Select an employee based on job and de...
selectEmployee	V1	zCEEService	Select an employee from table USER1.E...
selectEmployee	V2	zCEEService	Select an employee from table USER1.E...
updateEmployee		zCEEService	Update an employee in table USER1.EM...

?

Import Cancel

\*selectEmployee Service

### Service Project Editor: Definition

**General Information**

Edit or update the general information of the service.

Type: Db2 Service  
Version: 1.0.0  
Description:

**Actions**

Steps to create a service:

1. Input service version.
2. Import JSON schemas from a Db2 service manager or your local machine.
- [3. Complete the configuration for the service.](#)
- [4. Deploy the service.](#)
- [5. Export the service.](#)

**Define Db2 service**

Import a Db2 native REST service from a Db2 service manager. Alternatively, enter your Db2 service details and import the JSON schemas from your local machine.

Import from Db2 service manager...

Collection Id: SYSIBMSERVICE  
Db2 native REST service name: selectByRole  
Db2 native REST service version: V1  
Request JSON schema: request-schema.json  
Response JSON schema: response-schema.json

Import from local machine... Import from local machine...

The service developer retrieves details about the Db2 REST services

Note there is no service interface editor available



# Accessing a Db2 REST service resource

The screenshot shows the Service Project Editor for a project named "selectEmployee Service". The "Required Configuration" section displays a "Connection reference" input field containing "db2conn", which is circled in red. A red arrow points from this field to the "db2pass.xml" file on the right. The "db2pass.xml" file is an XML configuration file with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="DB2 REST">
  <zosconnect_zosConnectServiceRestClientConnection id="db2conn">
    <host>wg31.washington.ibm.com</host>
    <port>2446</port>
    <basicAuthRef>dsn2Auth</basicAuthRef>
  </zosconnect_zosConnectServiceRestClientConnection>
  <zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth">
    <appName>DSN2APPL</appName>
  </zosconnect_zosConnectServiceRestClientBasicAuth>
</server>
```

Annotations in red highlight the "db2conn" ID in the XML, the host and port values ("wg31.washington.ibm.com" and "2446"), and the "dsn2Auth" ID. On the left side of the editor, there is a table of connection parameters:

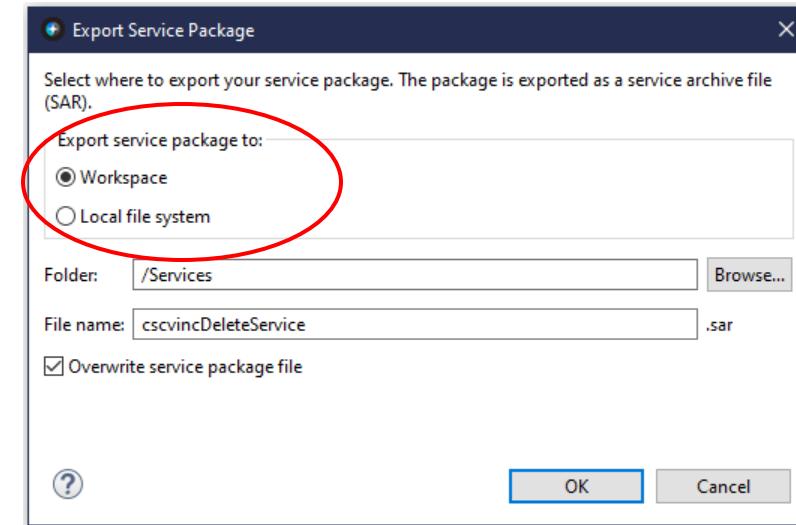
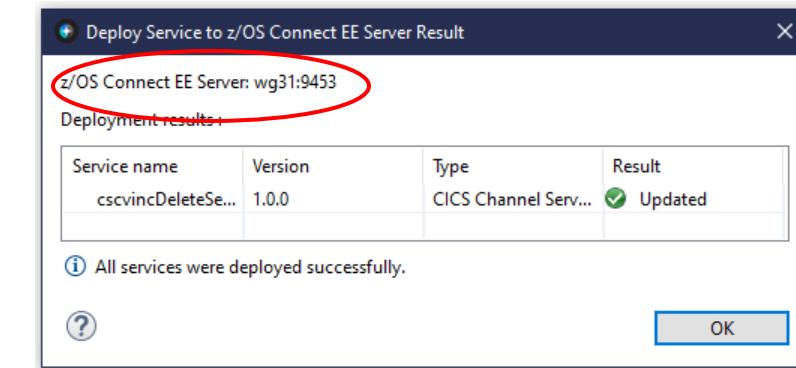
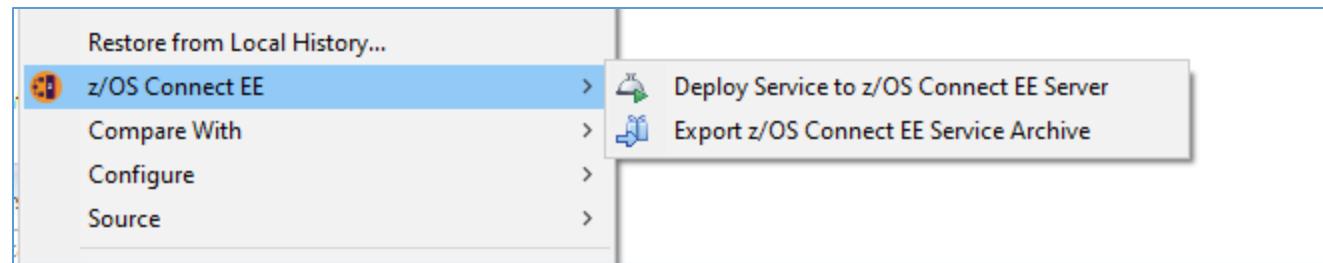
Parameter	Value
DSNL004I -DSN2 DDF START COMPLETE	
LOCATION	DSN2LOC
LU	
USIBMWZ.DSN2APPL	
GENERICLU	-NONE
DOMAIN	
WG31.WASHINGTON.IBM.COM	
TCPPORT	2446
SECPORT	2445
RESPORT	2447



# API toolkit – Services Editor

All services now be deployed to the server or exported for API development

Manage z/OS Connect server connections in the **Host Connections** view:





## Once we have a Service Archive (SAR) What's next?

Creating APIs using the Eclipse Tooling

*Remember: All service archives files are functionally equivalent regardless of how they are created or which resources they are intended to access*

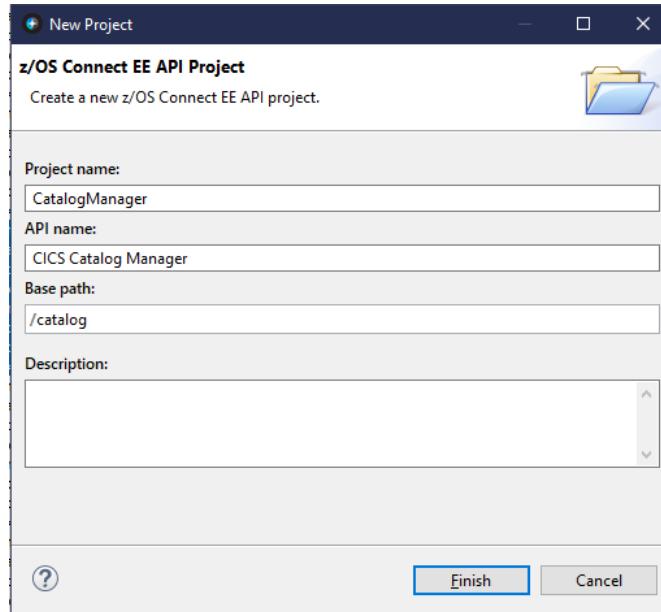


## **/api\_toolkit/api\_editor**

Quick and easy **API mapping**.



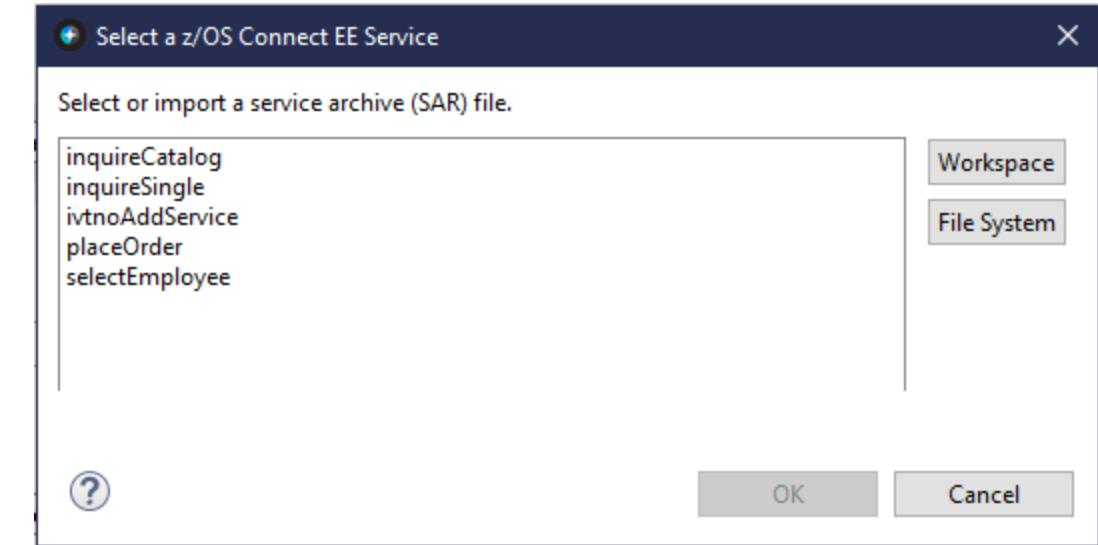
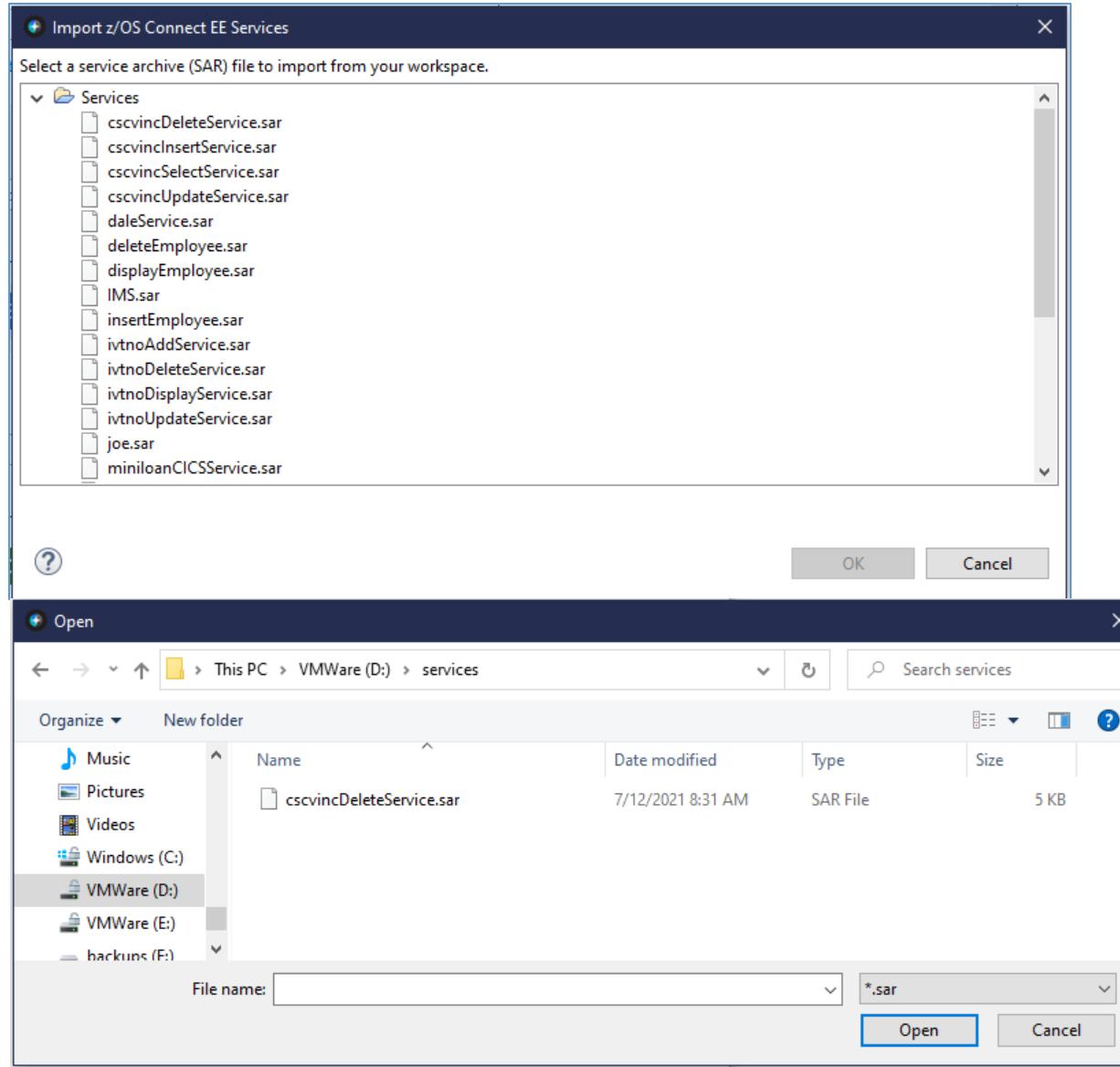
# API toolkit – API Editor



The screenshot shows the 'API Editor' window for the 'CICS Catalog Manager API'. The 'Describe your API' section contains fields for 'Name' (CICS Catalog Manager), 'Base path' (/catalog), and 'Version' (1.0.0). The 'Contact Information' section is collapsed. The main area shows a 'Path' entry (/newPath1) and a 'Methods (4)' section with four entries: POST, GET, PUT, and DELETE. Each method entry has fields for 'Service...', 'Mapping...', and navigation buttons (up, down, delete). A large red oval highlights the 'Methods (4)' section.



# Importing the service archive file from a filesystem or an Eclipse project





# API toolkit – API Editor

The screenshot shows the API toolkit API Editor interface. It displays three API endpoints:

- catalog API**:
  - Name:** catalog
  - Description:** (empty)
  - Base path:** /catalog
  - Version:** 1.0.0
  - Contact Information**: (empty)
  - Path:** /items?startItemID
    - Methods (2)**:
      - GET**: inquireCatalog
      - PUT**: ivtnoAddService
- order**:
  - Path:** /order
    - Methods (2)**:
      - POST**: placeOrder
      - PUT**: selectEmployee
- item/{itemID}**:
  - Path:** /item/{itemID}
  - Methods (1)**:
    - GET**: inquireSingle

The **API toolkit** is designed to encourage RESTful API design.

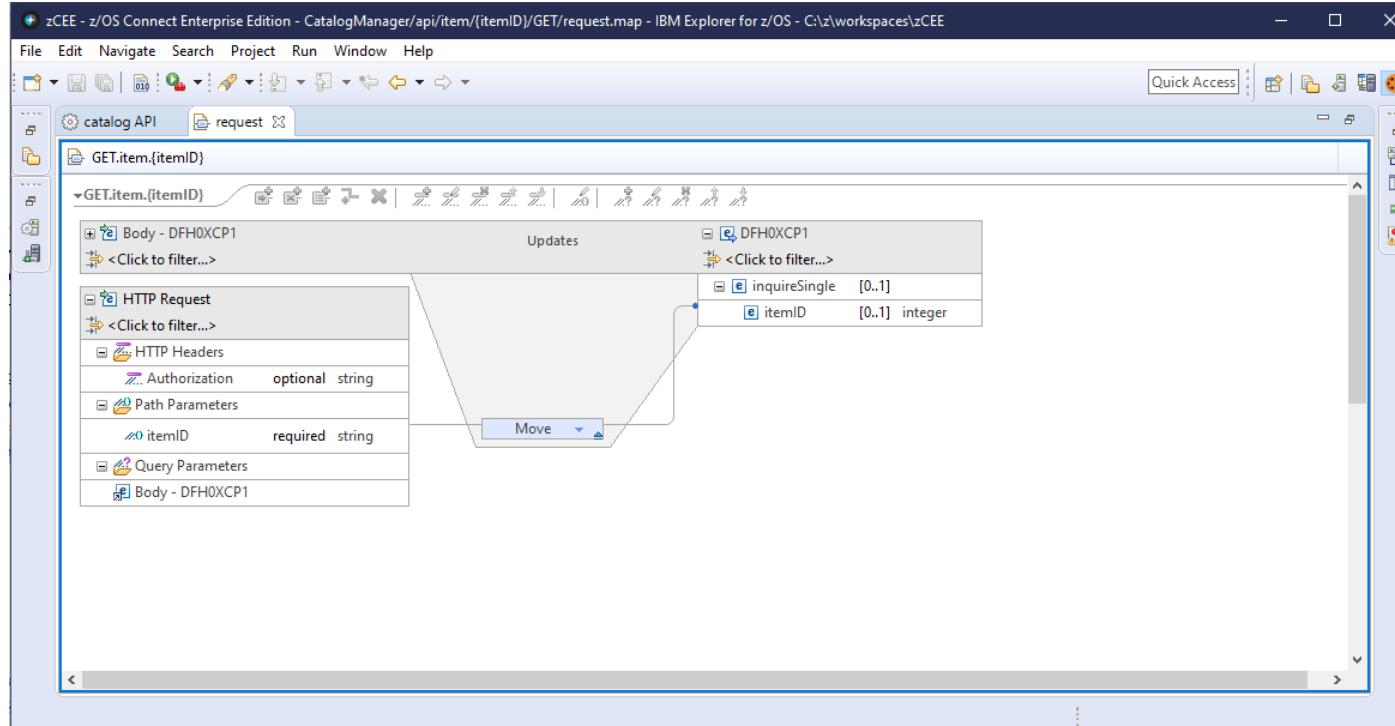
Once you define your API, you can map backend services to each request.

Your services are represented by a **.sar** files, which you import into the **API toolkit**, regardless of how the service archive file was generated.



# API toolkit – API Editor

API mapping: Assign values to the interface fields exposed by the service developer



Map both the request and response for each API.

Map path and query parameters to native data structures.

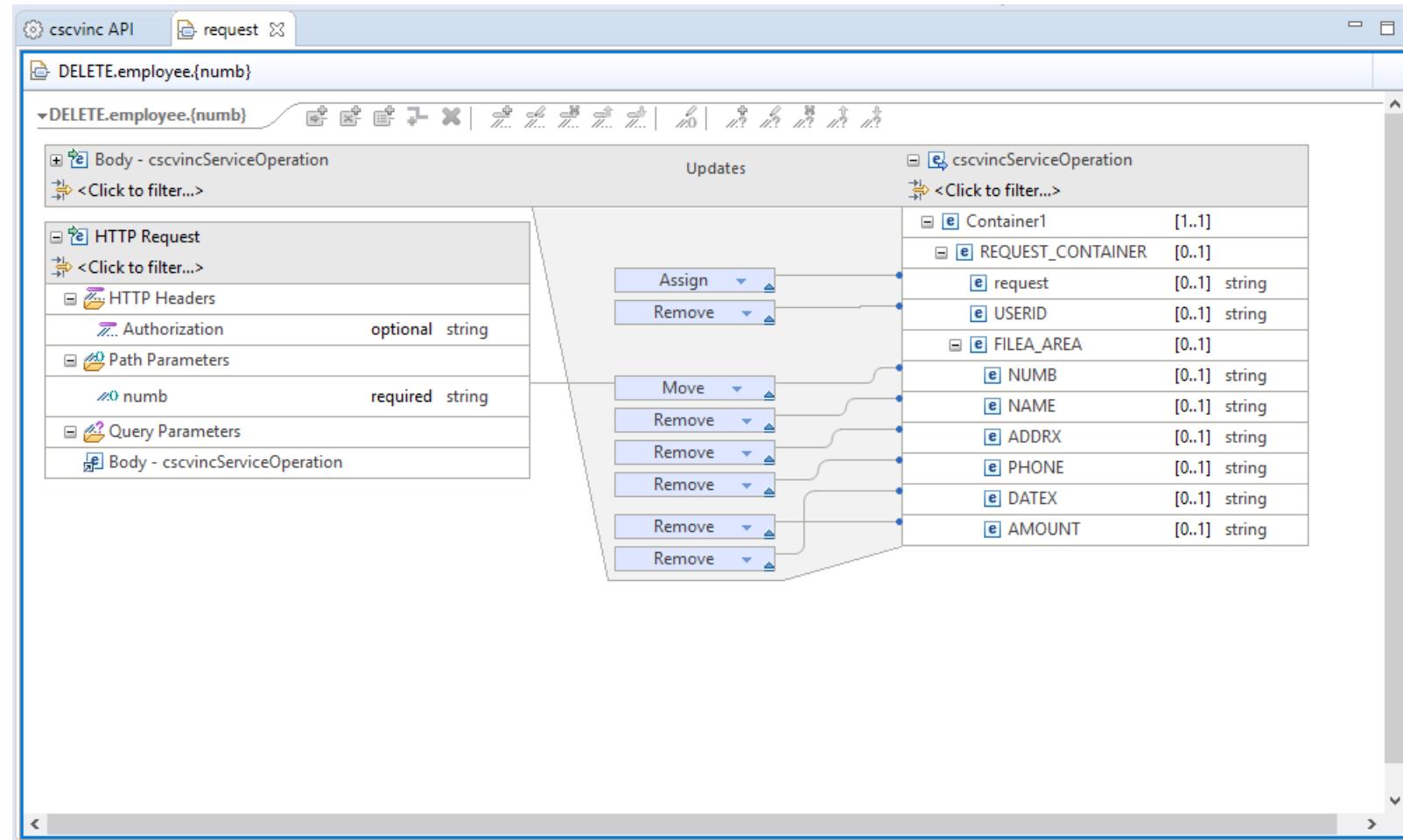
Assign static values to fields, useful for Op codes.

Remove unwanted fields to simplify the API (remember request was set to 01INQC in the SAR).



# API toolkit – API Editor

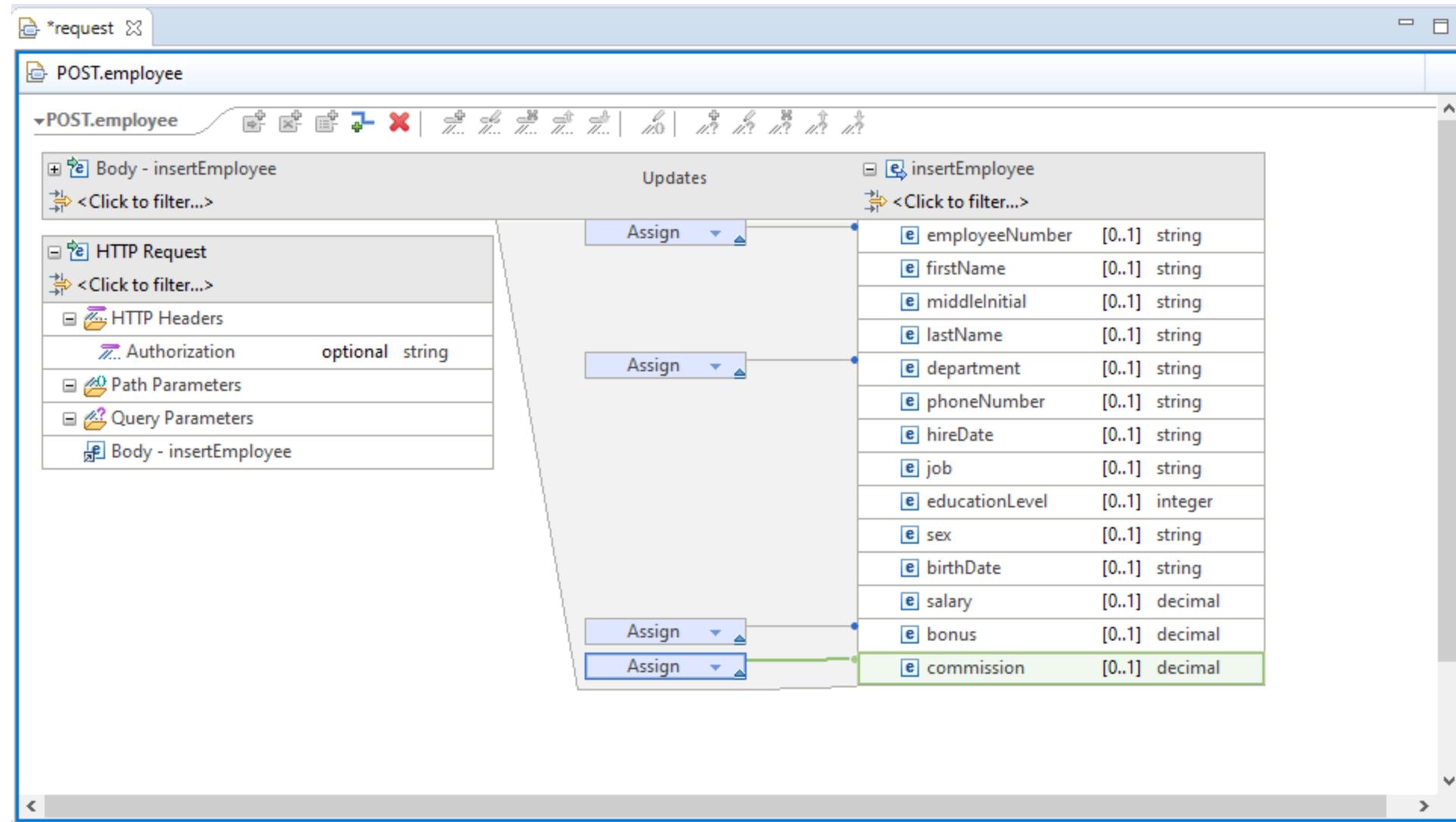
API mapping: Remove or assign values to the fields exposed by service developer





# API toolkit – API Editor and Db2 REST service

API mapping: Remove/Assign values columns exposed in Db2 REST service





# API toolkit – Header properties

API mapping: Allows adding HTTP header properties

The screenshot shows the API toolkit interface with two main sections: 'request' and 'Transferred'.

**request:**

- POST.queue
- Body - MQPUTOOperation
- HTTP Request
- HTTP Headers
  - Authorization (optional string)
  - ibm-mq-md-correlID (optional string)
- Path Parameters
- Query Parameters
- Body - MQPUTOOperation

**Transferred:**

- MQPUTOOperation
- mqmessage [1..1]
- stat [1..1] string
- numb [1..1] string
- name [1..1] string
- addrx [1..1] string
- phone [1..1] string
- datex [1..1] string
- amount [1..1] string
- comment [1..1] string

Retrieve specific message from queue based on correlation identity



# API toolkit : API definition with multiple response codes

The screenshot shows the API toolkit interface for defining an API operation. On the left, the path is set to `/employee/{employee}`. In the center, there are four methods: GET, POST, DELETE, and PUT. The GET method is selected, showing its configuration details. The service is `cscvincSelectService`, and the operation ID is `getCscvincSelectService`. Under the "Responses (2)" section, two responses are defined: a 404 response with a description "Not Found" and a 200 OK response. A modal window titled "Edit Response 404" is open, showing the response code as "404 - Not Found" and the description as "Not Found". Below this, rules are defined to determine when this response should be used:

- Rule 1: `!e/Container1/RESPONSE_CONTAINER/CEIBRESP` = 13 AND
- Rule 2: `!/Container1/RESPONSE_CONTAINER/CEIBRESP2` = 80

The summary of the rules is "Rule 1 AND Rule 2".

The **API toolkit** supports defining multiple response codes per API operation.

Separate mappings can be defined for each response code.

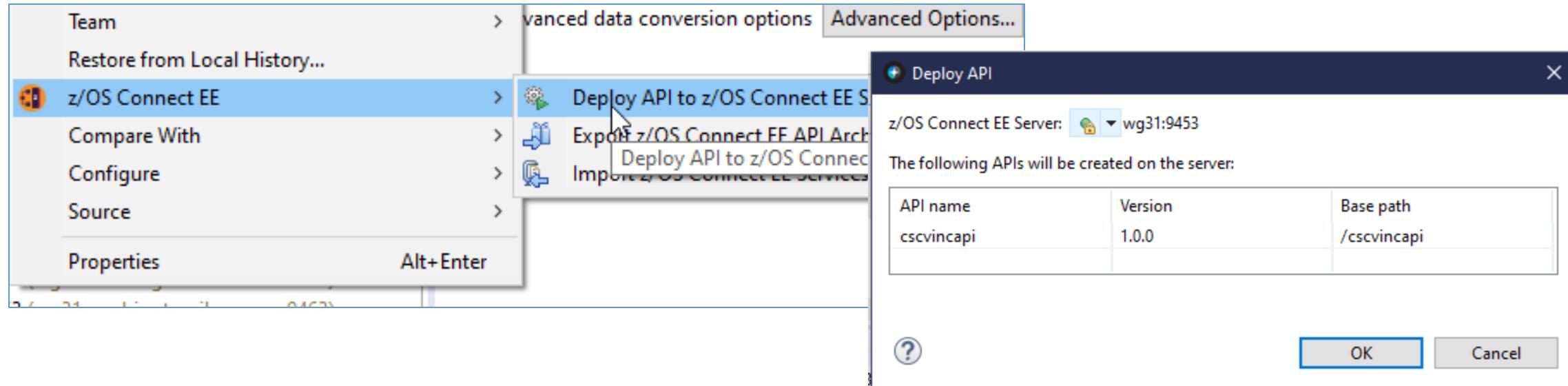
You can define rules based on fields in the service's return interface to tell z/OS Connect which response code to return



# API toolkit – API Editor

## API deployment

Manage z/OS Connect server connections in the **Host Connections** view:



**Right-click deploy to server** enables developers to quickly deploy, test, and iterate on their APIs.

**z/OS Connect servers view** allows you to start, stop, and remove APIs from a running server.



# This process “creates” and deploys the OpenAPI2 specification document



```
File Edit Format View Help
{
  "swagger": "2.0",
  "info": {
    "description": "",
    "version": "1.0.0",
    "title": "cscvincapi"
  },
  "basePath": "/cscvincapi",
  "schemes": [
    "https",
    "http"
  ],
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "paths": {
    "/employee/{employee)": {
      "get": {
        "tags": [
          "cscvincapi"
        ],
        "operationId": "getCscvincSelectService",
        "parameters": [
          {
            "name": "Authorization",
            "in": "header",
            "required": false,
            "type": "string"
          },
          {
            "name": "employee",
            "in": "path",
            "required": true,
            "type": "string",
            "maxLength": 6
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "schema": {
              "$ref": "#/definitions/getCscvincSelectService_response_200"
            }
          },
          "404": {
            "description": "Not Found",
          }
        }
      }
    }
  }
}
```

z/OS Connect OpenAPI2 tooling  
**produces** an OpenAPI 2 (Swagger 2.0) specification document, where the details of the methods and request/response messages in the API specification are driven by the nature of the z/OS asset (JSON format).



## Tech-Tip: You can create Service and API Archive files outside of the Eclipse tooling

- Use zconbt to create a service archive file

```
C:\z\workspaces\zCEE>...\\software\zconbt\bin\zconbt --  
projectDirectory=C:\z\workspaces\zcee\miniloanService --file=miniloan.sar  
BAQB0000I: z/OS Connect Enterprise Edition 3.0 Build Toolkit Version 1.10 (20220817-1609).  
BAQB0001I: Creating service archive from configuration file  
C:\z\workspaces\zcee\miniloanService\service.properties.  
BAQB1022I: IBM MQ for z/OS plugin for IBM z/OS Connect EE V3.0 build toolkit, code level is  
3.0.60.0(20220817-1609).  
BAQB0002I: Successfully created service archive file miniloan.sar
```

- Using zconbt to create an API archive file

```
C:\z\workspaces\zCEE>...\\software\zconbt\bin\zconbt --  
projectDirectory=C:/z/workspaces/zcee/miniloan --file miniloan.aar  
BAQB0000I: z/OS Connect Enterprise Edition 3.0 Build Toolkit Version 1.10 (20220817-1609).  
BAQB0028I: Creating API archive file from API project directory  
C:/z/workspaces/zcee/miniloan.  
BAQB0029I: Successfully created API archive file miniloan.aar.
```

The Eclipse tool kit is still required to develop the service or API.



# Deploying Service and API Archive files outside of the Eclipse tooling

- Use SAR or AAR files as request message and use HTTP POST
- Use the z/OS Connect administrative RESTful APIs
- Use Postman or cURL (client for URL)

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for File, Edit, View, Help, + New, Import, Runner, and a workspace dropdown. Below the header, there are several tabs for different requests: GET https://mpx3.washington.ibm.com..., POST https://mpx3.washington.ibm.com..., and POST https://wg31.washington.ibm.com... The main area is titled "Untitled Request" and shows a POST request to https://wg31.washington.ibm.com:9453/zosConnect/services. The "Body" tab is selected, showing the raw JSON response from the curl command. The response status is 201 Created, time is 1082 ms, and size is 1.02 KB. The JSON response is as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
{
  "zosConnect": {
    "serviceName": "selectEmployee",
    "serviceDescription": "Select a row from USER1.EMPLOYEE",
    "serviceProvider": "restclient-1.0",
    "serviceURL": "https://wg31.washington.ibm.com:9453/zosConnect/services/selectEmployee",
    "serviceInvokeURL": "https://wg31.washington.ibm.com:9453/zosConnect/services/selectEmployee?action=invoke",
    "dataXformProvider": "DATA_UNAVAILABLE",
    "serviceStatus": "Started"
  },
  "selectEmployee": {
    "receiveTimeout": 60000,
    "port": "2446",
    "host": "wg31.washington.ibm.com",
    "basicAuthConfigID": "dsn2Auth",
    "id": "Db2Conn",
    "httpMethod": "POST",
    "connectionTimeout": 30000,
    "uri": "/services/selectEmployee"
  }
}
```

- Deploy a service archive file  
`curl --data-binary @selectEmployee.sar  
--header "Content-Type: application/zip"  
https://mpxm:9453/zosConnect/services`

- Deploy an API archive file  
`curl --data-binary @Catalog.aar  
--header "Content-Type: application/zip"  
https://mpxm:9453/zosConnect/apis`

## Results:

```
{"zosConnect": {"serviceName": "selectEmployee", "serviceDescription": "Select a row from USER1.EMPLOYEE", "serviceProvider": "IBM_ZOS_CONNECT_SERVICE_REST_CLIENT-1.0", "serviceURL": "https://mpxm:9453/zosConnect/services/selectEmployee", "serviceInvokeURL": "https://mpxm:9453/zosConnect/services/selectEmployee?action=invoke", "dataXformProvider": "DATA_UNAVAILABLE", "serviceStatus": "Started"}, "selectEmployee": {"receiveTimeout": 0, "port": null, "host": null, "httpMethod": "POST", "connectionTimeout": 0, "uri": "/services/selectEmployee"}}
```



# Testing an API using the API toolkit

## Testing with Swagger UI

Test your deployed APIs directly with **Swagger UI** inside the editor.  
No need to export the Swagger doc to a separate tool.

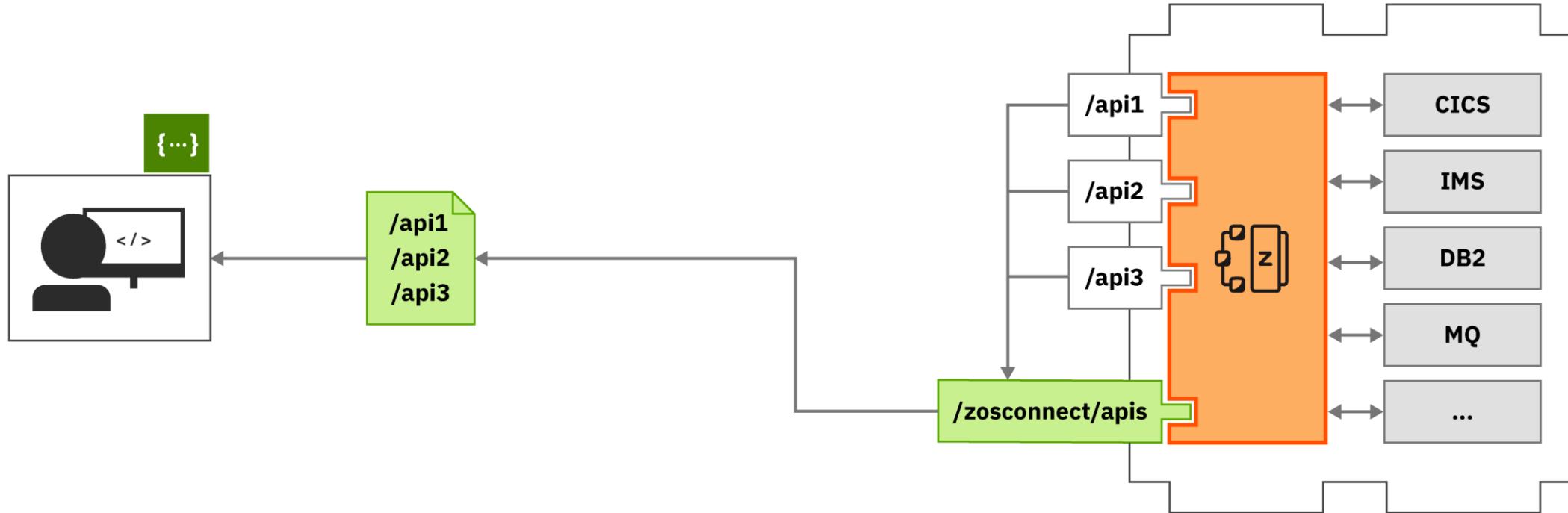
The screenshot shows the z/OS Connect EE Servers interface with the API toolkit open. On the left, under 'wg31:9443 (wg31.washington.ibm.com:9443)', there is a context menu for the 'cscvinc' API. The 'Open In Swagger UI' option is highlighted. The main window displays the Swagger UI for the 'cscvinc' service. It shows four endpoints: POST /employee, DELETE /employee/{employee}, GET /employee/{employee}, and PUT /employee/{employee}. The PUT endpoint is currently selected. To the right, a detailed view of the PUT /employee/{employee} endpoint is shown in a separate window. This view includes the 'Response Class (Status 200)' section, which contains the JSON schema for the response:

```
{  
  "cscvincSelectServiceOperationResponse": {  
    "cscvincContainer": {  
      "response": {  
        "CEIBRESP": 0,  
        "CEIBRESP2": 0,  
        "USERID": "string",  
        "file": {  
          "employeeNumber": "string",  
          "name": "string",  
        }  
      }  
    }  
  }  
}
```

The 'Parameters' section shows two parameters: 'Authorization' (header, string) and 'employee' (path, string). The 'Response Messages' section shows a 404 Not Found message with a 'Model | Example Value' link.



# API Documentation



APIs are discoverable via Swagger docs served from **z/OS Connect**.



## **Open API 3 z/OS Connect Designer**

Accessing z/OS resources from API using the z/OS Connect Designer



# With OpenAPI3, the specification description provides the details of the API

```
cscvinc.yaml - + File Edit View  
"/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  
Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8
```

```
cscvinc.yaml - + File Edit View  
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  
Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8
```



# Let's stop for a moment: z/OS Native Server Considerations – API Provider Deployment

Gradle plug-in and its requirements, see [Using Gradle with z/OS Connect](#).

**Important:** In order for multiple API project .war files to function when deployed to a single z/OS Connect native server, you must ensure that your OpenAPI specification includes a contextRoot defined within the servers section. The contextRoot attribute specifies the entry point of the deployed application. For example, to use a context root of /myContextRoot in the API's OpenAPI definition server's section:

```
openapi: 3.0.0
...
servers:
url: "https://localhost:9443/myContextRoot"
...
```

This definition must match the contextRoot attribute value of the webApplication element in your server.xml file. An example of the configuration might be:

```
<webApplication location="${server.config.dir}/apps/api.war" name="EmployeesApi" contextRoot="/myContextRoot"/>
```

If the server entry in the server section of the OpenAPI definition includes a contextRoot value, then this value must be specified in the contextRoot attribute of the corresponding webApplication element, even when only a single API is deployed to the z/OS Connect server.

Each API deployed to the same IBM z/OS Connect server requires a unique context root.

## 4. Copy the server configuration

Copy the server configuration files from the API project /scr/main/liberty/config directory into the \${server.config.dir}/configDrops/overrides directory of the server or another directory via FTP. For more information, see [Overview of IBM z/OS Connect Server configuration](#)

## 5. Deploy the generated API files to the z/OS Connect native server

Deploy API .war files into a permanent USS directory. An example directory, such as the one provided by the z/OS Connect native server template, is \${server.config.dir}/apps. API files can also be deployed to other directories, such as in separately mounted zFS file systems.

For each deployed API define a webApplication element in the configuration file. An example element is included in the supplied openApi3 server template. An example element might be the following:

```
<webApplication id="My API" location="${server.config.dir}/apps/api.war" name="MyAPI"/>
```

The samples as provided by z/OS Connect are not suitable as is for deployment to a z/OS Connect Native server.



# z/OS Server Considerations – API Context Root

<https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=building-zos-connect-apis-gradle>

## The drop-ins directory

The *drop-ins* directory, `/config/dropins` is a special directory that is supported by WebSphere® Application Server for Liberty. It files to be deployed and dynamically loaded into the running IBM z/OS Connect with no additional definitions that are required in the configuration file.

By default, z/OS Connect Designer deploys the API `.war` file to this directory. Using the same directory in your API container image creation of that image because the configuration remains the same.

## A directory other than drop-ins

This is required in any of the following situations:

- The API's OpenAPI definition server's section contains server entry that includes a context root value, which is not just `/`.
- Multiple APIs are to be deployed to the same IBM z/OS Connect container. Because the API `.war` file will be generated with a context root and multiple API `.war` files in the same server must have unique context root values.

You need to include a context root value (not `/`) in the API's OpenAPI definition server's section, for example to use a context root of `/MyCompany`:

```
openapi: 3.0.0
...
servers:
  url: https://localhost:9443/MyCompany
...
```

- Requests to start an API require authentication only, without authorization, so the authorization roles need to be mapped to the Web Application Server for Liberty special subject `ALL_AUTHENTICATED_USERS`. For more information, see [How to define authorization](#).

If you choose not to use the drop-ins directory, you must alter the configuration that is used in z/OS Connect Designer during the creation of the API container image.

**The question is when and where to make these updates?**

**The WSC recommends making the changes in the specification file before it is imported into the Designer.**

- **Explicit server XML configuration is required to define applications and add the context root**

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

# Also, the API's specification may require new or updated security roles



```
cscvinc.json - Notepad
File Edit Format View Help
{
  "swagger": "2.0",
  "info": {
    "description": "",
    "version": "1.0.0",
    "title": "cscvincapi"
  },
  "basePath": "/cscvincapi",
  "schemes": [
    "https",
    "http"
  ],
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "paths": {
    "/employee/{employee}": {
      "get": {
        "tags": [
          "Employee"
        ]
      }
    }
  }
}
```

```
cscvinc.yaml - Notepad
File Edit Format View Help
openapi: 3.0.1
info:
  title: cscvinc
  description: ""
  version: 1.0.0
servers:
- url: /cscvinc
x-ibm-zcon-roles-allowed:
- Manager
paths:
  /employee:
    post:
      tags:
        - cscvinc
      operationId: postCscvincInsertService
      x-ibm-zcon-roles-allowed:
        - Staff
      parameters:
        - name: Authorization
          in: header
          schema:
```

The WSC also recommends adding security roles to specification file before it is imported into the Designer.

```
{
  "name": "employee",
  "in": "path",
  "required": true,
  "type": "string",
  "maxLength": 6
}
],
"responses": {
  "200": {
    "description": "OK",
    "schema": {
      "$ref": "#/definitions/getCscvincSelectService_response_200"
    }
  },
  "404": {
    "description": "Not Found",
  }
}
```

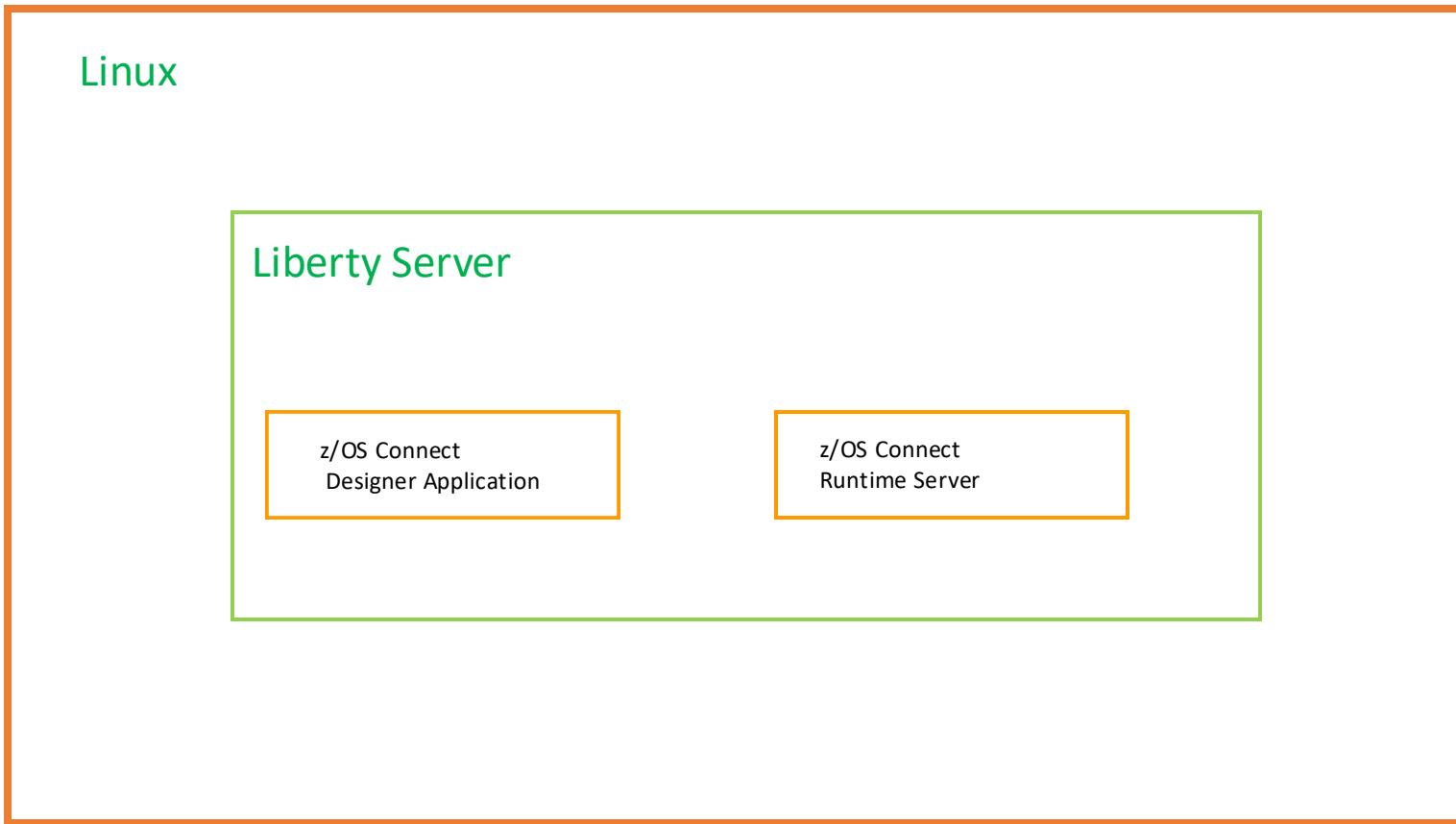
```
description: OK
content:
  application/json:
    schema:
      $ref: '#/components/schemas/postCscvincInsertService_response_200'
x-codegen-request-body-name: postCscvincInsertService_request
/employee/{employee}:
  get:
    tags:
      - cscvinc
    operationId: getEmployee
    x-ibm-zcon-roles-allowed:
      - Staff
    parameters:
      - name: Authorization
        in: header
        schema:
          type: string
```

# Let's explore the z/OS Connect Designer Topology



- A z/OS Connect Designer is composed of a self-contained Linux environment configured with a Liberty server running a z/OS Connect Designer application and a z/OS Connect runtime server, in total, known as a “container”.
- Running a Designer container requires a container runtime environment.

z/OS Connect Designer





# z/OS Connect Designer runtime options

one API per container, i.e., multiple Designer containers may be required

A Windows, Mac OS<sup>#</sup> or a Linux distribution base image

Docker Engine\*, Podman, etc.

(Ubuntu, openSUSE, Red Hat, Windows WSL, etc.)

z/OS Connect  
Designer Container

z/OS Connect  
Designer Container

z/OS Connect  
Designer Container

z/OS Connect  
Designer Container

The Washington Systems Center provides our recommendations for best practices for configuring and managing Designer containers at URL  
<https://ibm.biz/BdvrZh>

- For basic setup and management of the Designer environment, see *z/OS Connect Designer and Native Experiences*
- For a Podman example, see *Developing and Administering RESTful APIs for CICS REST Services*
- For a Docker engine example, *Developing and Administering RESTful APIs for Db2 REST Services*

#Warning, Machines with Mac M1 Pro processors may be problematic.

\*Docker Engine is not an option on Macs, use Podman

# Tech-Tip: Some suggested runtime environments that required no license



Screenshot of the Docker Engine documentation page for "Server" installation:

**Server**  
Docker provides `.deb` and `.rpm` packages from the following Linux distros and architectures:

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	s390x
CentOS	✓	✓		
Debian	✓	✓	✓	
Fedora	✓	✓		
Raspbian			✓	
RHEL				✓
SLES				✓
Ubuntu	✓	✓	✓	✓
Raspberry				
Binaries	✓	✓	✓	

**Other Linux distros**

**Note**  
While the instructions below may work, Docker doesn't test or verify installation on distro derivatives.

<https://docs.docker.com/engine/install/#server>

Screenshot of the Podman documentation "Get Started" page:

## Get Started with Podman

Podman is a utility provided as part of the libpod library. It can be used to create and maintain containers. The following tutorial will teach you how to set up Podman and perform some basic commands.

### First Things First: Installing Podman

For installing or building Podman, please see the installation instructions:

[Installation Instructions](#)

**Basic Resources**

- Installation Instructions
- Documentation
- Podman Troubleshooting Guide

**Getting Help**

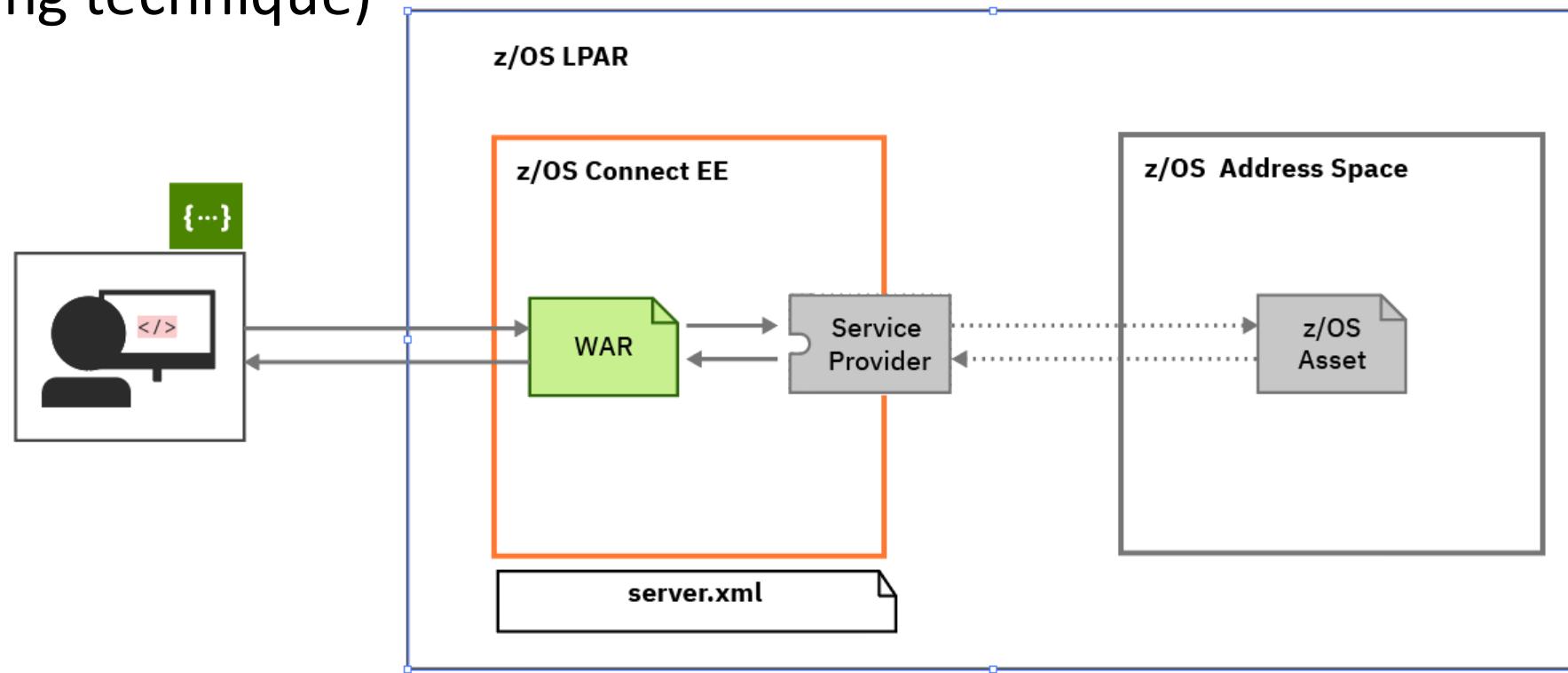
<https://podman.io/get-started>

**Important:** The command line interface (CLI) syntax is the same between the Docker Engine and Podman. Just change a Docker command from using the `docker` command to the `podman` command, e.g., `docker ps -a` when using Docker Engine becomes `podman ps -a` when using Podman.



# Accessing the CICS, Db2 or IMS resource using an Open API 3 API

- z/OS Connect OpenAPI 3 APIs are developed using a z/OS Connect Designer web browser tool. This tool generates Web ARchive (WAR) files (a traditional Java packaging technique)



- The WAR provides the RESTful interface is ready to be consumed by a client and it requires the client to have no knowledge that a z/OS resource is being accessed as well as the mapping and transformation for accessingg the resource.
- The Service Provider is tightly coupled to a specific instance of a resource (e.g., host and port)

# With OpenAPI 3 you start with an YAML file description of the API



The screenshot shows a web browser window for "JF Best YAML Viewer Online" at <https://jsonformatter.org/yaml-viewer>. The main area displays an OpenAPI 3.0.0 specification for a "CICS Filea Sample VSAM Application". The specification includes sections for info, paths, and components. The "paths" section contains a "post:" operation for "/employee". The "parameters" for this operation include an "Authorization" header. The "requestBody" is defined with a schema pointing to a component schema named "postCscvincInsertService\_request". The "responses" section includes a "200" response with an "OK" description and an "application/json" content type. To the right of the code editor is a "YAML Tree" panel showing the hierarchical structure of the YAML document, including nodes for object, paths, components, servers, and schemas. A central "Start" button is visible above the tree.

```
openapi: 3.0.0
info:
  description: "CICS Filea Sample VSAM Application"
  version: 1.0.0
  title: cscvinc
  x-ibm-zcon-roles-allowed:
    - Manager
paths:
  /employee:
    post:
      tags:
        - cscvinc
      operationId: postCscvincInsertService
      parameters:
        - name: Authorization
          in: header
          required: false
          schema:
            type: string
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/postCscvincInsertService_request"
      description: request body
      required: true
      responses:
        "200":
          description: OK
          content:
            application/json:
```

YAML Tree

```
object ► paths ► /employee/{employee} ►
object {6}
  openapi : 3.0.0
  info {3}
    description : CICS Filea Sample VSAM Application
    version : 1.0.0
    title : cscvinc
    x-ibm-zcon-roles-allowed [1]
      0 : Manager
  paths {2}
    /employee {1}
      post {5}
    /employee/{employee} {3}
      get {5}
      put {6}
      delete {5}
  servers [1]
    0 {1}
      url : /sandbox
  components {1}
    schemas {16}
```

© 2017, 2024 IBM Corporation

mitchj@us.ibm.com

Slide 87



# Import the description of an OpenAPI 3 API into the Designer

IBM z/OS Connect Designer

Employee 1.0.0

Paths / /employee/{employee} / GET

Saved Test :

Information

Security

Paths 2

/employee 1

POST 1

/employee/{emplo... 3

GET 1

PUT 1

DELETE 1

z/OS Assets 0 +

Components 16

Tags 0

View operation properties

Request → Responses

Define rule → 200 OK

Define rule → 404 Not Found

Else send default response → 500 Severe Error

```
graph LR; Request((API Request)) --> Responses[Responses]; Responses -- "Define rule" --> OK((200 OK)); Responses -- "Define rule" --> NotFound((404 Not Found)); Responses -- "Else send default response" --> SevereError((500 Severe Error))
```



# Describe the resource, a CICS program or IMS transaction

The image displays two side-by-side screenshots of the IBM z/OS Connect Designer application, illustrating the configuration of a z/OS asset.

**Left Screenshot (CICS Channel Program):**

- Panel:** IBM z/OS Connect Designer
- Title:** Step 2 of 5
- Section:** Add z/OS Asset
- Configuration:**
  - 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
- Buttons:** Previous (disabled), Next

**Right Screenshot (IMS Transaction):**

- Panel:** IBM z/OS Connect Designer
- Title:** Step 2 of 5
- Section:** Add z/OS Asset
- Configuration:**
  - Select a z/OS Asset type: IMS transaction
  - Transaction code: IVTNO
  - Program language: COBOL
  - Select an IMS connection: imsConn
- Buttons:** Previous, Next



# Or a Db2 REST service

IBM z/OS Connect Designer

EmployeesApi 1.1

Information Security

Paths /employees/{id}

GET PUT DELETE

/employees GET POST

z/OS Assets addEmployee deleteEmployee getEmployee getEmployees selectByRole updateEmployee

Components Tags

Step 3 of 4  
Add z/OS Asset

Select a Db2 connection db2Conn

Import from Db2 service manager

Db2 native REST service collection ID e.g. SYSIBMSERVICE

Db2 native REST service name e.g. myService

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

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

Specify a URL http://github.com/example/api-docs Import file

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

Previous Next

Add z/OS Asset / Import Db2 native REST service

Import Db2 native REST service

Select a Db2 connection db2Conn

Filter table

12 Db2 native REST service found

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

Items per page: 10 1-10 of 12 items Previous Import



# Map the API's method and request with the resource's input

IBM z/OS Connect Designer

cscvinc 1.0.0

Paths / /cscvinc/employee/{employee} / GET

View operation properties

API Request → CICS programCscvinc → Responses

If channel.cscvincContainer.Response-Contai... → API 200 OK

Else send default response

programCscvinc

Request Response z/OS Asset details

Edit mapping View structure

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

Channel

cscvincContainer

- Request-Container
  - ACTION abc S
  - USERID abc
  - FILEA-AREA
    - STAT abc
    - NUMB abc employee



# Map the API's request with the z/OS resource's input message

The screenshot shows the IBM z/OS Connect Designer interface. On the left, the navigation pane displays the 'EmployeesApi' project version 1.1, with sections for Information, Security, Paths, /employees/{id}, /employees, and z/OS Assets. The /employees path is currently selected, showing GET, PUT, and DELETE methods. The 'GET' method is highlighted.

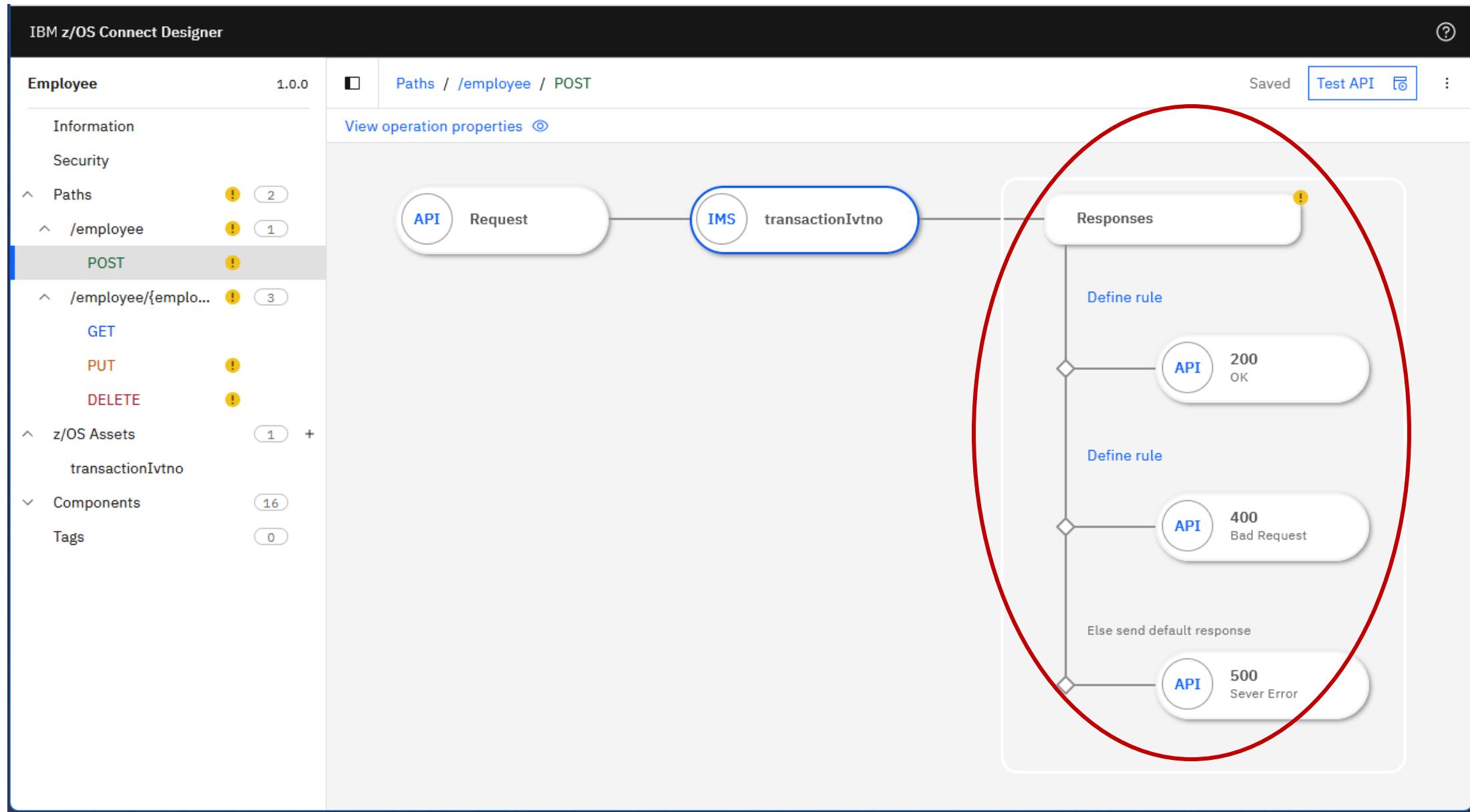
In the main workspace, under 'Paths / /employees / GET', the 'View operation properties' section shows a flow diagram. A blue rounded rectangle labeled 'Request' is connected to a blue rounded rectangle labeled 'DB2 getEmployees Get the details of all employees'. This connection is circled in red. An arrow points from the DB2 node to a 'Responses' section, which contains a condition: 'If statusCode equals 200 then send 200 response'. Below this, another blue rounded rectangle labeled 'API 200 OK' is shown.

Below the flow diagram, a detailed view for the 'getEmployees' operation is displayed. It has tabs for Request, Response, and z/OS Asset details. The Request tab is active, showing the mapping rule: 'Map fields from the API request into the z/OS Asset request.' Under the 'Body' section, there are two mapping rules:

- \* WORKDEPT: abc {{ \$exists(\$department) ? department : '' }}
- \* JOB: abc {{ \$exists(\$job) ? job : '' }}

A red arrow points from the text 'JSONata example' to the JSONata mapping rule for the WORKDEPT field.

# Map the resource's results to the API's response messages





# Map the z/OS resource's response to the API's response (200)

Paths / [/employee/{employee}](#) / GET

Saved Test :

200 - OK

Edit mapping View structure

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

Body

summary

  └ message

detail

  └ cscvincSelectServiceOperationResponse

    └\*cscvincContainer

      └ response

        └ CEIBRESP

        └ CEIBRESP2

        └ USERID

        └ filea

          └ employeeNumber

          └ name

          └ address

          └ phoneNumber

          └ date

          └ amount

          └ comment

abc Record NUMB successfull retrieved by USERID

123

123

abc

abc NUMB

abc NAME

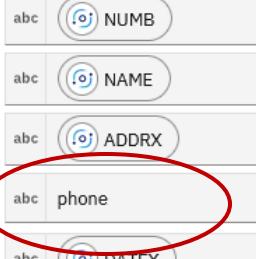
abc ADDRX

abc phone

abc DATEX

abc AMOUNT

abc COMMENT

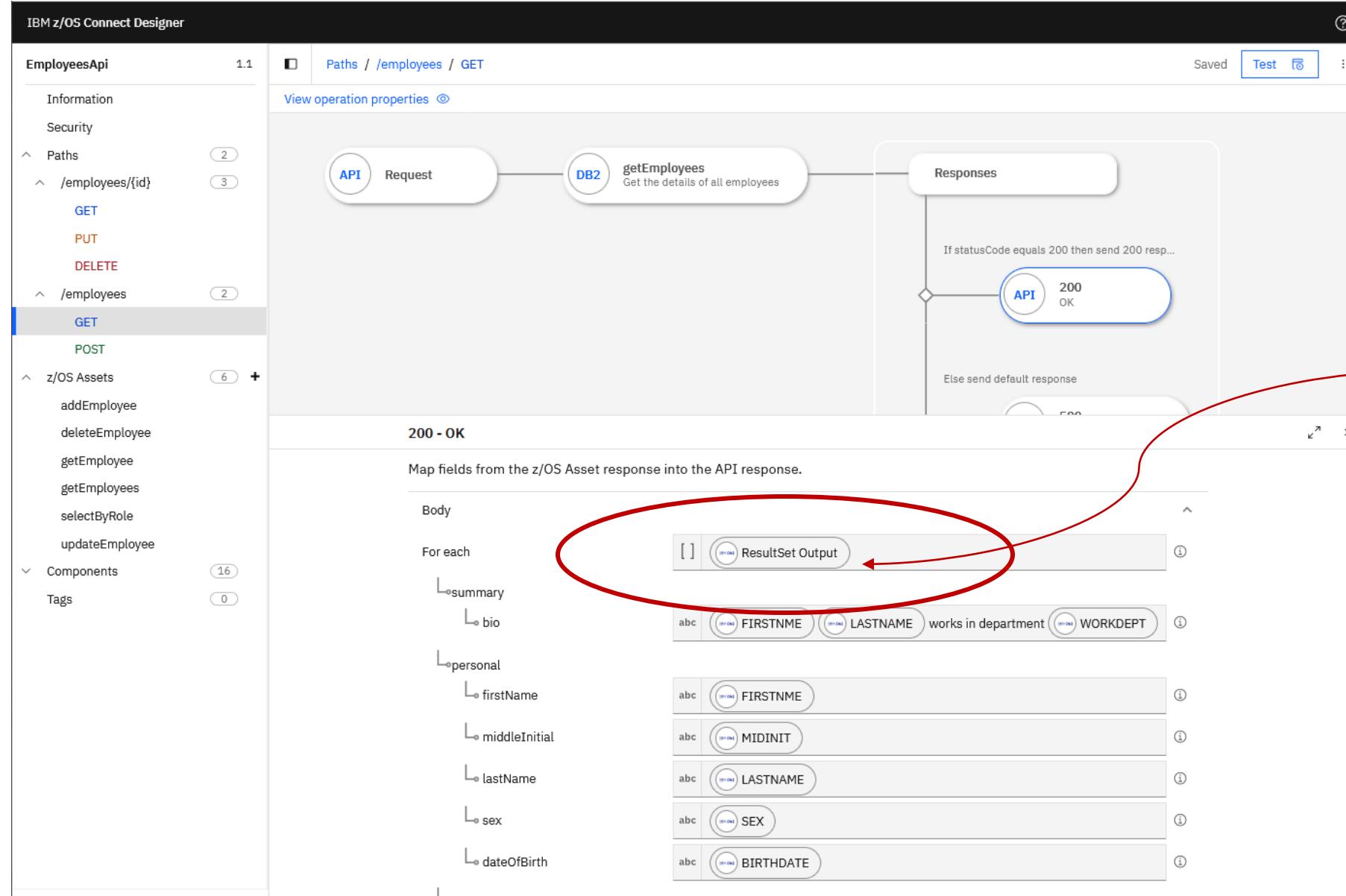


JSONata example showing the building of a complex message from the response.

Not recognized as a response message field. It is simply a text field..



# Map the z/OS resource response to the API's response message



*ResultSet Output* is a list or array of the results and the “For each” processes each element in the list.



# z/OS Connect Designer for OpenAPI 3 (404)

IBM z/OS Connect Designer

EmployeesApi 1.1

Paths / /employees/{id} / PUT

View operation properties

Saved Test : ?

Information

Security

Paths

/employees/{id}

GET

PUT

DELETE

/employees

z/OS Assets

Components

Tags

200 Updated

If "Update Count" equals 0 then send 404 res...

404 Not Found

Else send default response

500 Internal Server Error

404 - Not Found

Edit mapping View structure

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

Body

message

abc Employee not found ⓘ

A red oval highlights the 'Employee not found' message value in the mapping table.



# z/OS Connect Designer for OpenAPI 3 (500)

IBM z/OS Connect Designer

cscvinc 1.0.0

Paths /employee/{employee} / GET

View operation properties ⓘ

Saved Test ⌂ :

Information  
Security  
Paths (2)  
/employee (1)  
POST (1)  
/employee/{emplo... (3)  
GET (highlighted)  
PUT (1)  
DELETE (1)  
z/OS Assets (1)  
programCscvinc  
Components (16)  
Tags (0)

Request → CICS programCscvinc → Responses

If channel.containerCscvinc."Response-Conta..."

200 OK

500 - Severe Error

Body

message

A severe error has occurred message code ⓘ

```
graph LR; Request((API Request)) --> CICS((CICS programCscvinc)); CICS --> Responses((Responses)); Responses --> OK((200 OK));
```

```
graph TD; subgraph Body [Body]; message["message"]; end;
```

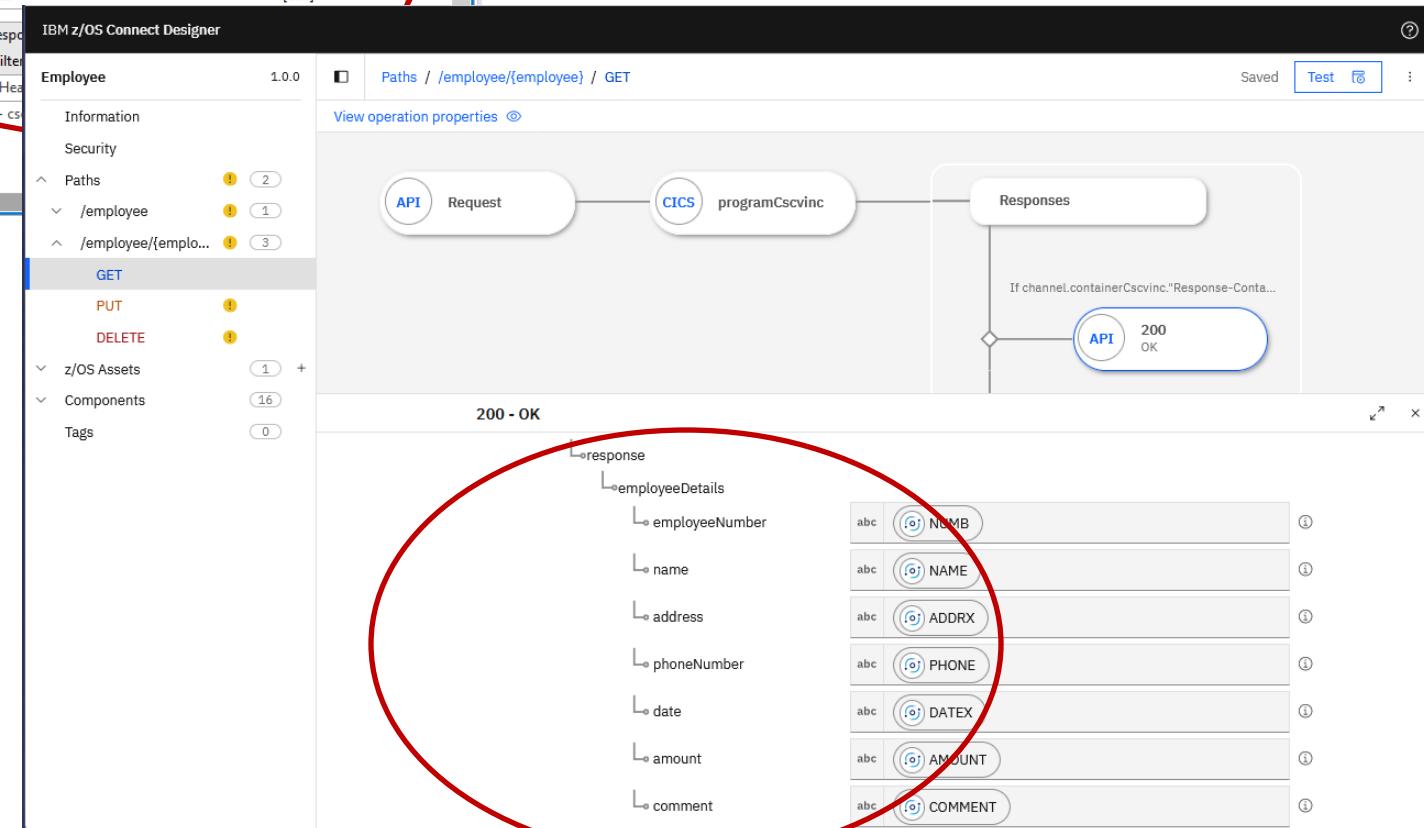
# Contrast the OpenAPI 2 versus an OpenAPI 3 Response Message



The screenshot shows two response message structures in the Eclipse-based toolkit. A red circle highlights the right-hand structure, which is labeled "Body - cscvincSelectServiceOperationResponse". This structure contains fields such as cscvincContainer, response, CEIBRESP, CEIBRESP2, USERID, and filea. The left-hand structure, labeled "cscvincSelectServiceOperationResponse", contains similar fields: cscvincContainer, response, CEIBRESP, CEIBRESP2, USERID, and filea. The right-hand structure is explicitly named as a "Body", while the left-hand one is part of a larger operation definition.

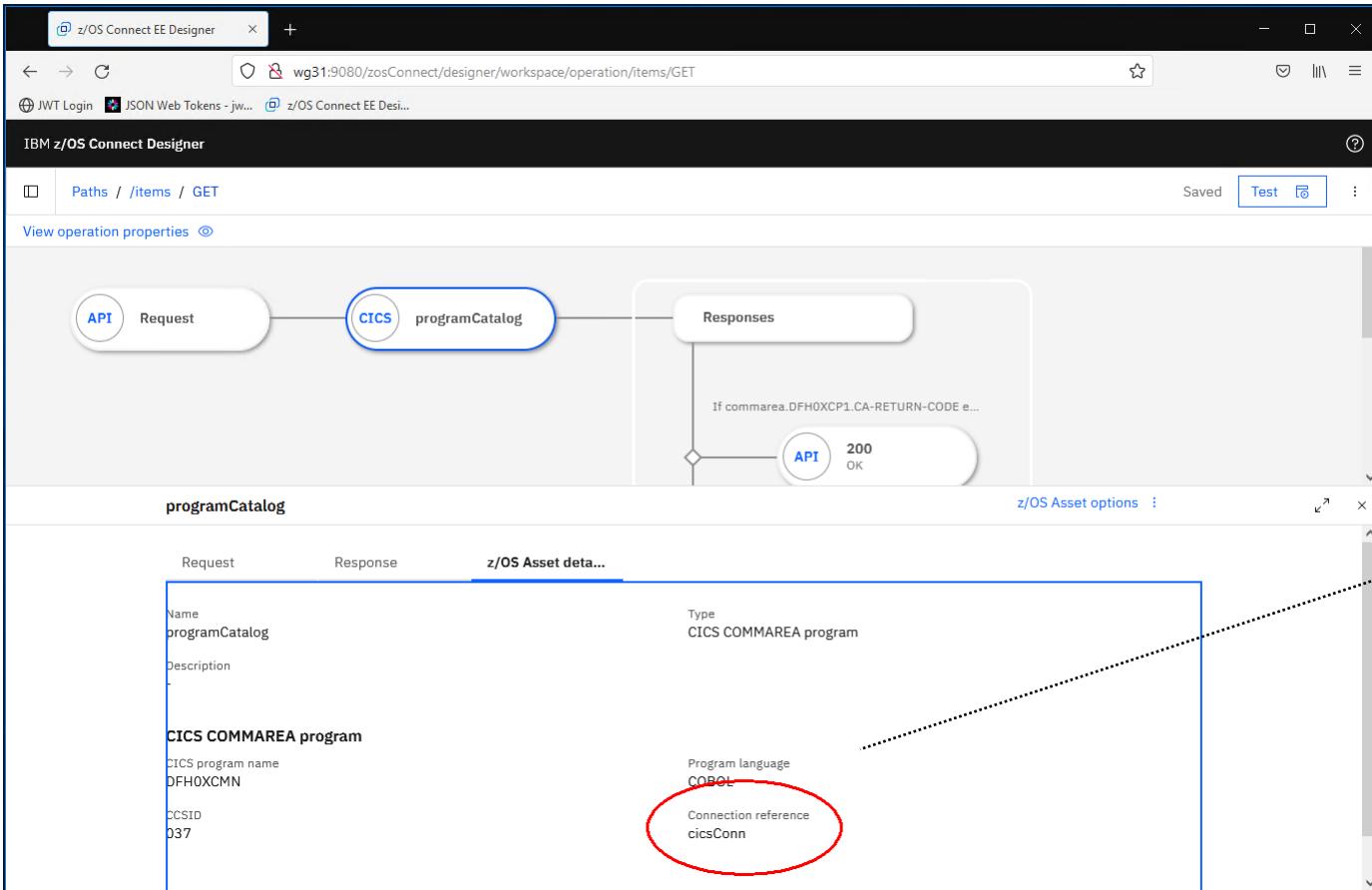
Eclipse based - z/OS Connect API Toolkit -  
The contents of the API's response message are  
directly derived from the z/OS resource's response and  
usually a subset of the resource's response contents.

Web browser - z/OS Connect Designer –  
The contents of the API's response  
message must be mapped to the z/O  
resource's response and JSONata coding,  
see URL  
<https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=connect-what-is-jsonata> for more information on JSONata.





# Server XML - Accessing a CICS program using IPIC



The screenshot shows the 'Server Config' interface for 'cics.xml'. It has tabs for 'Design' and 'Source'. The 'Source' tab displays the XML configuration:

```
<server description="CICS IPIC connections">
  <!-- Enable features -->
  <featureManager>
    <feature>zosconnect:cics-1.0</feature>
  </featureManager>
  <zosconnect_cicsIpicConnection id="cicsConn" host="${CICS_HOST}" port="${CICS_PORT}" />
</server>
```

A red circle highlights the 'cicsConn' connection reference in the XML code.

The connection references identifies a `zosconnect_cicsIpicConnection` configuration element which provides the connection details to a CICS region.

Note that this is a different CICS feature than the CICS feature for OpenAPI 2 server. The features are not interchangeable.



# Server XML - Accessing an IMS transaction

The screenshot shows the IBM z/OS Connect Designer interface. On the left, the navigation pane lists 'Employee' resources: 'Information', 'Security', 'Paths', '/employee' (with 1 warning), '/employee/{employee}' (with 3 warnings), and '/employee/{emp...}' (selected, with 1 warning). The 'Paths' section shows a flow from 'Request' to 'IMS transactionIvtno'. The 'transactionIvtno' asset is detailed as an 'IMS transaction' with 'Transaction code IVTNO' and 'Connection profile imsConn'. A red oval highlights 'imsConn'. On the right, a modal window titled 'Server Config' displays the 'ims.xml' file. The XML code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="zosconnect_imsConnection">
  <zosconnect_imsConnection id="imsConn">
    connectionFactoryRef="imsConnectionFactory"
    imsDatastoreName="${IMS_DATASTORE}" />
  <connectionFactory id="imsConnectionFactory">
    containerAuthDataRef="IMSCredentials">
      <properties.gmoa hostName="${IMS_HOST}" portNumber="${IMS_PORT}" />
    </connectionFactory>
    <authData id="IMSCredentials" user="${IMS_USER}" password="${IMS_PASSWORD}" />
  </server>
```

A red circle highlights the 'imsConn' connection profile reference in the XML.

The connection references identifies a `zosconnect_imsConnection` element. Which provides the connection details to IMS.



# Server XML - Accessing a Db2 REST service

The screenshot shows the z/OS Connect Designer interface. On the left, a workflow diagram for a 'getEmployee-V1' operation is displayed. It starts with an 'API Request' step, followed by a 'DB2 getEmployee-V1' step (which is annotated with 'Get the details of all employees'), and ends with an 'API Response' step (labeled '200 OK'). A condition 'If compareArea.DFHDXCP1.CA-RETURN-CODE e...' is shown branching from the DB2 step. Below the workflow, the 'getEmployee-V1' asset details are shown, including its name, description, type (Db2 native REST service), and connection reference 'db2Conn' (circled in red).

On the right, the 'Server Config' window is open, showing the 'db2.xml' configuration file. The 'Design' tab is selected, displaying the XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="Db2 Connections">
  <featureManager>
    <feature>zosconnect:db2-1.0</feature>
  </featureManager>
  <zosconnect_credential user="${DB2_USERNAME}" password="${DB2_PASSWORD}" id="commonCredentials" />
  <zosconnect_db2Connection id="db2Conn" host="${DB2_HOST}" port="${DB2_PORT}" credentialRef="commonCredentials" />
</server>
```

A red arrow points from the circled 'db2Conn' in the asset details to the 'zosconnect\_db2Connection' element in the XML. A callout box with the text 'Define connections to Db2 using variables defined in bootstrap.properties file' is positioned over the XML code.

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

The connection references identifies a `zosconnect_db2Connection` configuration element which provides the connection details to a DB2 DDF task.

Note that this is a new feature for accessing Db2.

# EJB roles for an OpenAPI 3 z/OS Connect API



```
openapi: 3.0.0
...
servers:
- url: /
x-ibm-zcon-roles-allowed:
- Manager
...
paths:
/items:
  get:
    operationId: itemsGet
    ...
/items/{id}:
  get:
    ...
    operationId: itemsIdGet
    x-ibm-zcon-roles-allowed:
    - Staff
/orders:
  post:
    ...
    operationId: ordersPost
    x-ibm-zcon-roles-allowed:
    - Staff
```

From the OpenApi document, the value for %role% would be either **Manager** or **Staff**.

So, the required SAF EJB roles to be defined would be:

- *BBGZDFLT.CatalogManager.Manager*
- *BBGZDFLT.CatalogManager.Staff*

REDFINE EJBCROLE *BBGZDFLT.CatalogManager.Manager*  
REDFINE EJBCROLE *BBGZDFLT.CatalogManager.Staff*

Access to use the GET method to invoke /items would require read access to EJB role *BBGZDFLT.CatalogManager.Manager*.

Access to use the GET method to invoke /items/{id} and the POST method to invoke /orders would require read access to EJB role *BBGZDFLT.CatalogManager.Staff*.

```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />

<webApplication id="CatalogManager" location="${server.config.dir}/apps/api.war" contextRoot="catalog"
name="CatalogManager" />

<safRoleMapper profilePattern=%profilePrefix%.%resourceName%.%role%
```



# What REST test tooling is available?



# Testing an API with Postman

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Edit', 'View', and 'Help' options, followed by 'Home', 'Workspaces', 'Reports', and 'Explore'. A search bar says 'Search Postman'. On the right of the header are icons for cloud, user, gear, and notifications, along with an 'Upgrade' button.

The main workspace shows a list of requests. The first request is selected, showing a 'GET' method and the URL 'https://wg31.washington.ibm.com:9453/cscvinc/employee/948478'. Below it, another request is shown with a 'GET' method and the URL 'https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111'.

The request details panel shows the 'Params' tab selected. It lists a single 'Query Params' entry:

KEY	VALUE	DESCRIPTION	...	Bulk Edit

The 'Body' tab is selected in the bottom navigation. The response body is displayed as JSON:

```
1  "cscvincSelectServiceOperationResponse": {  
2      "cscvincContainer": {  
3          "response": {  
4              "CEIBRESP": 0,  
5              "CEIBRESP2": 0,  
6              "USERID": "CICSUSER",  
7              "filea": {  
8                  "employeeNumber": "111111",  
9                  "name": "C. BAKER",  
10                 "address": "OTTAWA, ONTARIO",  
11                 "phoneNumber": "51212003",  
12                 "date": "26 11 81",  
13                 "amount": "$0011 .00"  
14             }  
15         }  
16     }  
17 }
```

The status bar at the bottom shows 'Status: 200 OK Time: 205 ms Size: 899 B' and a 'Save Response' button. The bottom navigation bar includes 'Find and Replace', 'Console', 'Bootcamp', 'Runner', 'Trash', and other icons.



# Testing an API with the API Explorer\*

The screenshot shows the IBM API Explorer interface. On the left, a sidebar lists various Liberty REST APIs: cscvinc, db2employee, filemgr, imsPhoneBook, jwtIvpDemoApi, miniloancics, mqapi, and phonebook. The cscvinc section is expanded, showing four operations: POST /cscvinc/employee, DELETE /cscvinc/employee/{employee}, GET /cscvinc/employee/{employee}, and PUT /cscvinc/employee/{employee}. The main area displays the API documentation for the cscvinc/employee endpoint. The URL is https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111. The response body is a JSON object:

```
{  
  "cscvincSelectServiceOperationResponse": {  
    "cscvincContainer": {  
      "response": {  
        "CE1BRESP": 0,  
        "CE1BRESP2": 0,  
        "USERID": "CICSUSER",  
        "filea": {  
          "employeeNumber": "111111",  
          "name": "C. BAKER",  
          "address": "OTTAWA, ONTARIO",  
          "phoneNumber": "51212003",  
          "date": "26 11 81",  
          "amount": "$0011.00"  
        }  
      }  
    }  
}
```

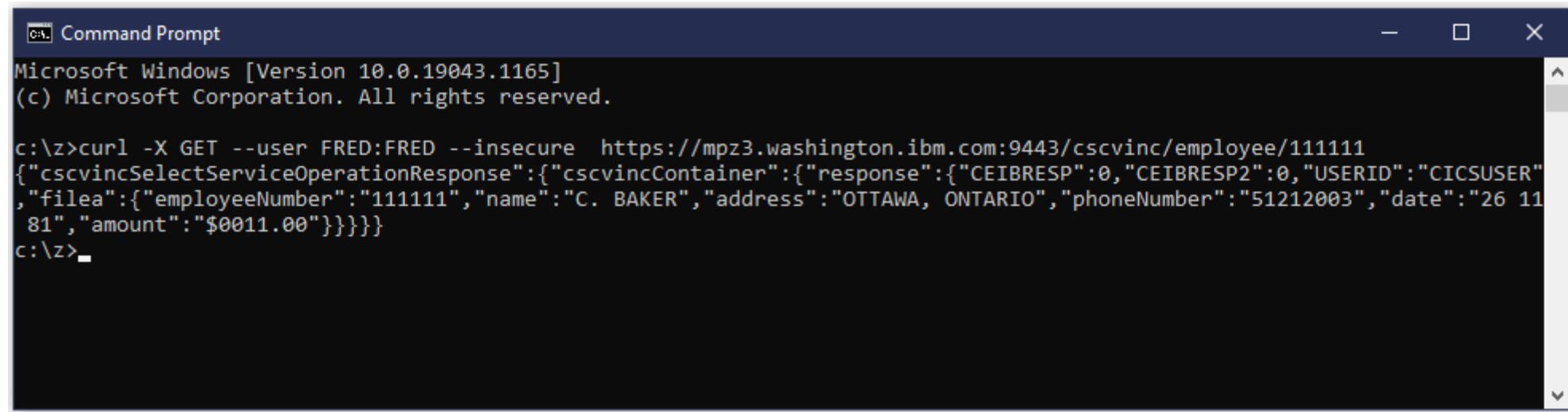
The response code is 200, and the response headers are:

```
{  
  "content-language": "en-US",  
  "content-length": "269",  
  "content-type": "application/json; charset=UTF-8",  
}
```

\*with zCEE V3.0.48 or later



# Testing an API using the cURL tool



The screenshot shows a Microsoft Windows Command Prompt window titled "Command Prompt". The window displays the following text:

```
Microsoft Windows [Version 10.0.19043.1165]
(c) Microsoft Corporation. All rights reserved.

c:\z>curl -X GET --user FRED:FRED --insecure https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111
{"cscvincSelectServiceOperationResponse":{"cscvincContainer":{"response":{"CEIBRESP":0,"CEIBRESP2":0,"USERID":"CICSUSER","filea":{"employeeNumber":"111111","name":"C. BAKER","address":"OTTAWA, ONTARIO","phoneNumber":"51212003","date":"26 11 81","amount":"$0011.00"}}}}
c:\z>
```

<https://curl.se/download.html>



# Tech-Tip: Examples of curl in JCL and in scripts, bat or commands files

```
*****  
/* SET SYMBOLS  
*****  
//EXPORT EXPORT SYMLIST=(*)  
// SET CURL= '/usr/lpp/rocket/curl'  
*****  
/* CURL Procedure  
*****  
//CURL PROC  
//CURL EXEC PGM=IKJEFT01,REGION=0M  
//SYSTSPRT DD SYSOUT=*  
//SYSERR DD SYSOUT=*  
//STDOUT DD SYSOUT=*  
// PEND  
*****  
/* STEP CURL - use cURL to deploy API cscvinc  
*****  
//DEPLOY EXEC CURL  
BPXBATCH SH export CURL=&CURL; +  
$CURL/bin/curl -X PUT -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc?status=stop+  
pped > null; +  
$CURL/bin/curl -X DELETE -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc > null; +  
$CURL/bin/curl -X POST -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
--data-binary @/u/johnson/cscvinc.aar +  
--header "Content-Type: application/zip" +  
https://wg31.washington.ibm.com:9445/zosConnect/apis  
*****  
/* STEP CURL - use curl to invoke the API cscvinc  
*****  
//INVOKE EXEC CURL  
//SYSTSIN DD *,SYMBOLS=EXECSYS  
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X GET -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
https://wg31.washington.ibm.com:9445/cscvinc/employee/000100
```

```
@echo off  
set TARGET=wg31.washington.ibm.com  
set FILE=cscvinc_1.0.0  
curl -X PUT --user user1:user1 --insecure  
https://%TARGET%:9483/zosConnect/apiRequesters/%FILE%?status=stopped  
curl -X DELETE --user user1:user1 --insecure  
https://%TARGET%:9483/zosConnect/apiRequesters/%FILE%  
curl -X POST --user user1:user1 --data-binary @%FILE%.ara --header  
"Content-Type: application/zip" --insecure  
https://%TARGET%:9483/zosConnect/apiRequesters  
curl -X GET -user user1:user1 --insecure  
https://%TARGET%:9483/cscvinc/employee/000100  
echo ''
```

Change the status of API to stopped

Delete or remove the API from the server

Deploy the API to the server

Execute the API



**/security**

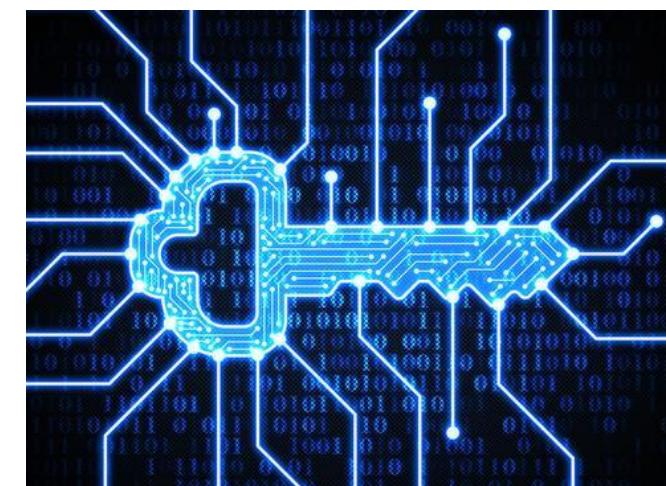
How is security implemented?



# General security terms or considerations

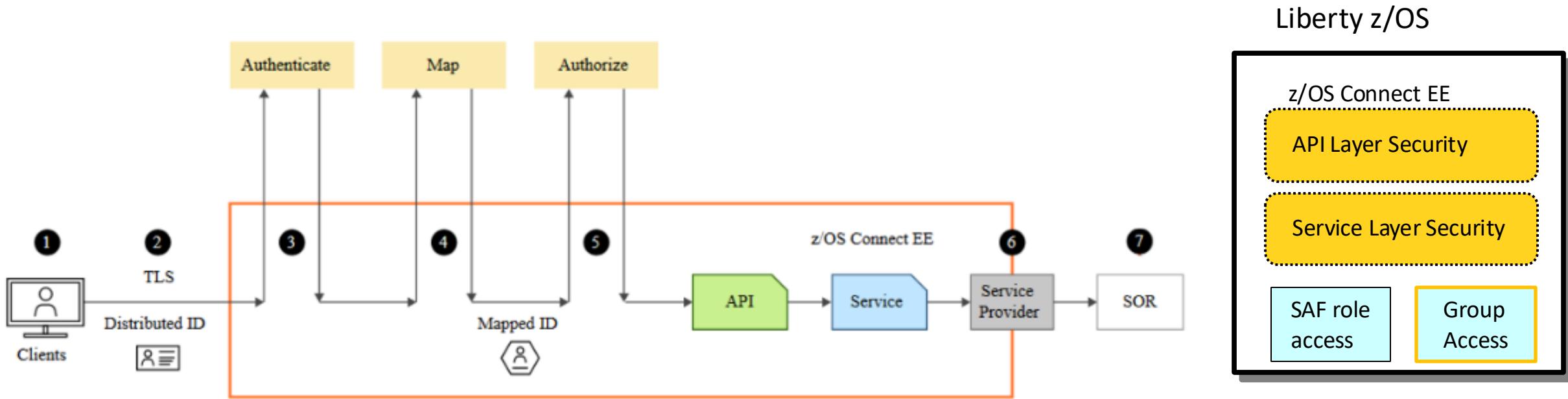
Security involves

- Identifying who or what is requesting access (**Authentication**)
  - Basic Authentication
  - Mutual Authentication using Transport Layer Security (TLS), formerly known as SSL
  - Third Party Tokens
- Ensuring that the message has not been altered in transit (**Data Integrity**) and ensuring the confidentiality of the message in transit (**Encryption**)
  - TLS (encrypting messages and using a digital signature)
- Controlling access (**Authorization**)
  - Is the authenticated identity authorized to access to z/OS Connect
  - Is the authenticated identity authorized to access a specific API, Services, etc.





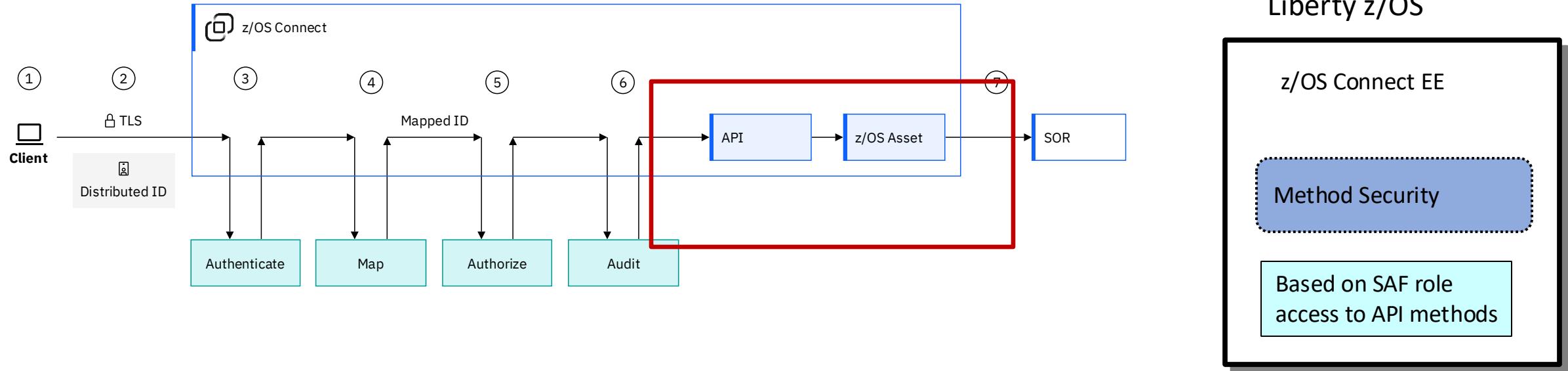
# Details of the OpenAPI2 z/OS Connect API Provider security flow



1. The credentials provided by the client
2. Secure the connection to the z/OS Connect server
3. Authenticate the client. This can be within the z/OS Connect server or by requesting verification from a third-party server
4. Map the authenticated identity to a user ID in the user registry
5. Authorize the mapped user ID to connect to z/OS Connect and optionally authorize user to invoke actions on APIs
6. Secure the connection to the System of Record (SoR) and provide security credentials to be used to invoke the program or to access the data resource
7. The program or database request may run in the SoR under the mapped ID



# Details of the OpenAPI 3 z/OS Connect API Provider security flow



The flow includes the following security steps that can be performed by IBM z/OS Connect. The credentials are provided by the client. These can be a user ID and password, a JWT, or a TLS certificate.

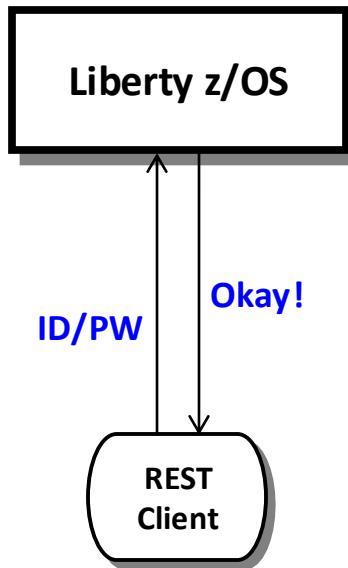
1. The credentials, including the identity, are passed on the connection between the client and IBM z/OS Connect. The identity is typically a distributed ID, such as an X.509 distinguished name and associated LDAP realm that originates from a remote system. Alternatively, the identity might be a SAF user ID. The data that is sent on the connection can be encrypted using TLS.
2. The client is authenticated. This can be within IBM z/OS Connect or by requesting verification from a third-party server.
3. The authenticated identity can be mapped to a user ID in the IBM z/OS Connect user registry.
4. The user is authorized to invoke the API operation if they have access to the role.
5. The API request is audited by using the Liberty Audit feature.
6. The authenticated user identity can be propagated to the System of Record (SoR) when the SoR supports this capability. Alternatively, the SoR connection can be configured to use a functional identity.



# Liberty Authentication

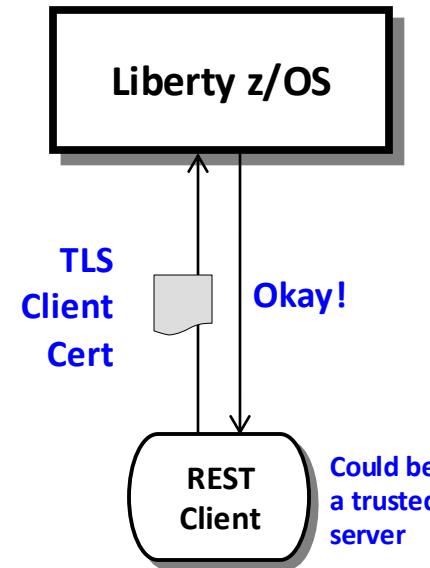
Several different ways this can be accomplished:

## Basic Authentication



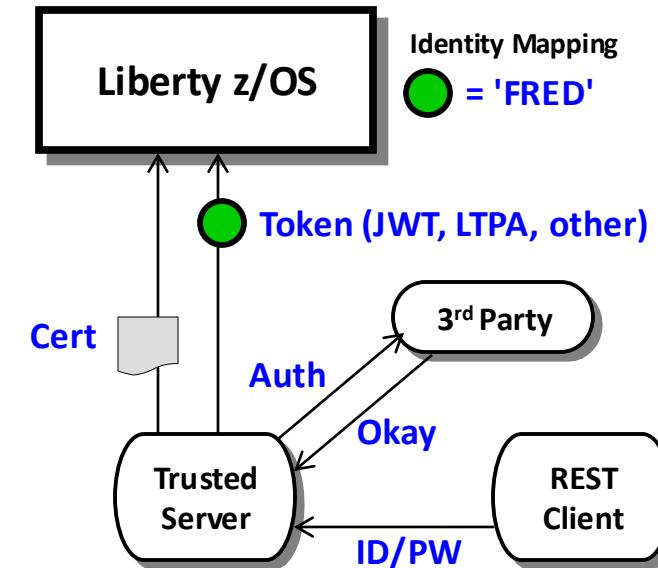
- Server prompts for ID/PW
- Client supplies ID/PW or ID/Passticket
- Server checks registry:
  - Basic (server.xml)
  - LDAP
  - SAF

## Client Certificate



- Server prompts for cert.
- Client supplies certificate
- Server validates cert and maps to an identity
- Registry options:
  - LDAP
  - SAF

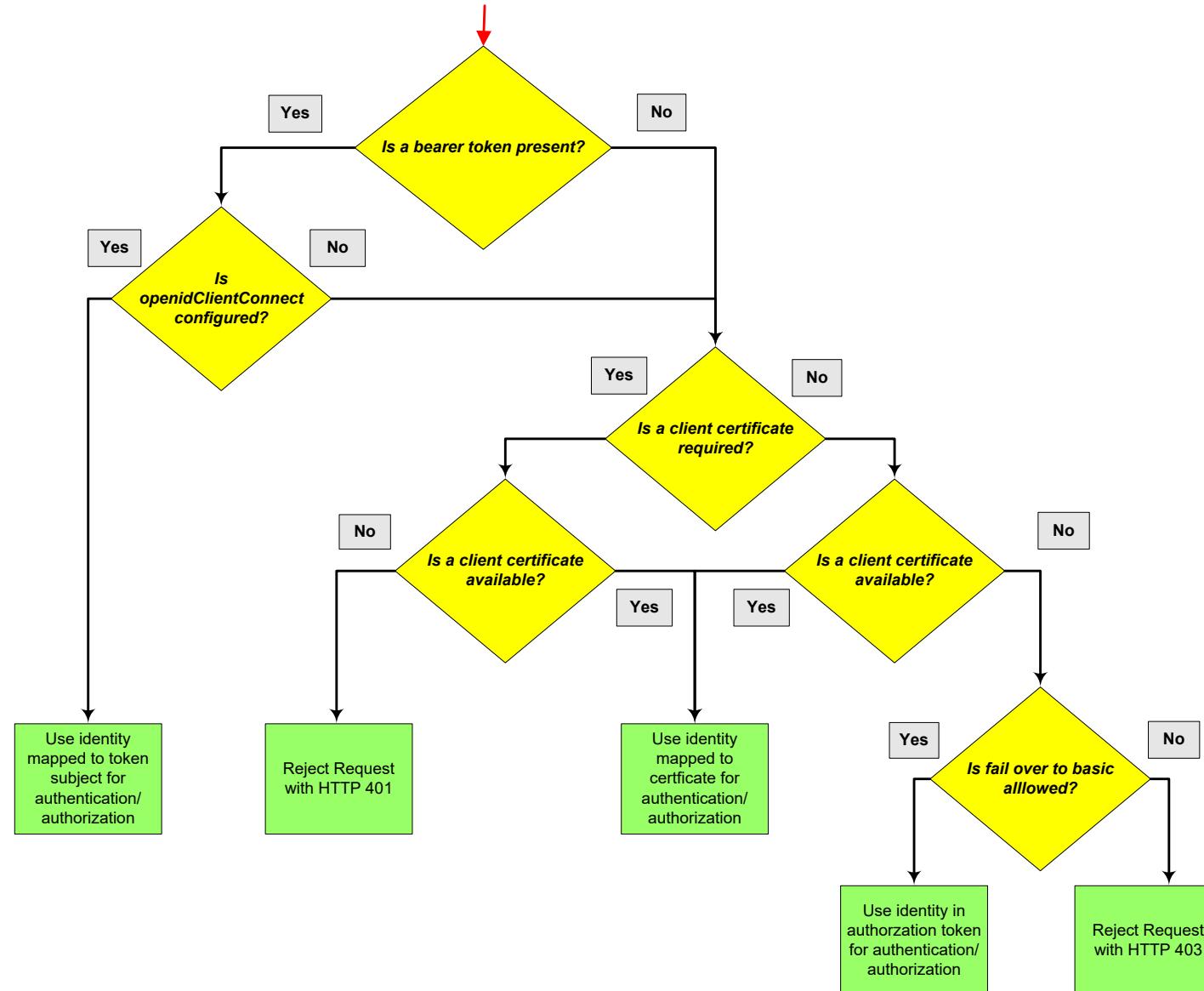
## Third Party Authentication



- Client authenticates to 3<sup>rd</sup> party sever
- Client receives a trusted 3<sup>rd</sup> party token
- Token flows to Liberty z/OS and is mapped to an identity
- Registry options:
  - LDAP
  - SAF



# Authentication credential precedence order for determining authorization identity





# Security for OpenAPI 2 z/OS Connect APIs are configured using z/OS Connect elements

The *requireAuth* attribute controls whether an inbound request must provide credentials using one of the three authentication methods, e.g., basic, client certificate, or third-party token.

```
<zosconnect_zosConnectManager  
    requireAuth="true|false"  
    requireSecure="true"/>
```

Globally, requires that users specify security credentials to be authenticated order to access APIs, services and API requesters, unless overridden on the specific resource definitions.

```
<zosconnect_zosConnectAPIs>  
    <zosConnectAPI name="catalog"  
        requireAuth="true|false"  
        requireSecure="true"/>  
</zosconnect_zosConnectAPIs>  
  
<zosconnect_services>  
    <service id="selectByEmployee"  
        name="selectEmployee"  
        requireAuth="true|false"  
        requireSecure="true"/>  
</zosconnect_services>  
  
<zosconnect_apiRequesters>  
    requireAuth="true|false"  
    <apiRequester name="cscvincapi_1.0.0"  
        requireAuth="true|false"  
        requireSecure="true"/>  
</zosconnect_apiRequesters>
```

Requires that users specify security credentials to be authenticated in order to access the API.

Requires that users specify security credentials to be authenticated in order to directly access the service. This attribute is ignored when the service is invoked from an API, then only the API requireAuth attribute is relevant.

Requires that users specify security credentials to be authenticated in order to access all API requesters. If the requireAuth attribute is not set, the global setting on the zosconnect\_zosConnectManager element is used instead, unless the requireAuth attribute is overridden on the specific API requester.

# Security for OpenAPI 3 z/OS Connect APIs is configured using Liberty elements



```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />  
  
<webApplication id="catalogManager" name="catalogManager"  
    location="${server.config.dir}/apps/api.war" contextRoot="/catalogManager" />  
  
<safRoleMapper profilePattern=%profilePrefix%.%resourceName%.%role%
```

The *name* attribute of the *webApplication* for the deployed WAR file determines the name of the EJBRoles used manage access to the API's methods.

*From the example OpenApi document, the value for %role% would be either Manager or Staff.*

So, the required SAF EJB roles to be defined would be:

- *BBGZDFLT.catalogManager.Manager*
- *BBGZDFLT.catalogManager.Staff*

```
REDEFINE EJBRULE BBGZDFLT.catalogManager.Manager  
REDEFINE EJBRULE BBGZDFLT.catalogManager.Staff
```

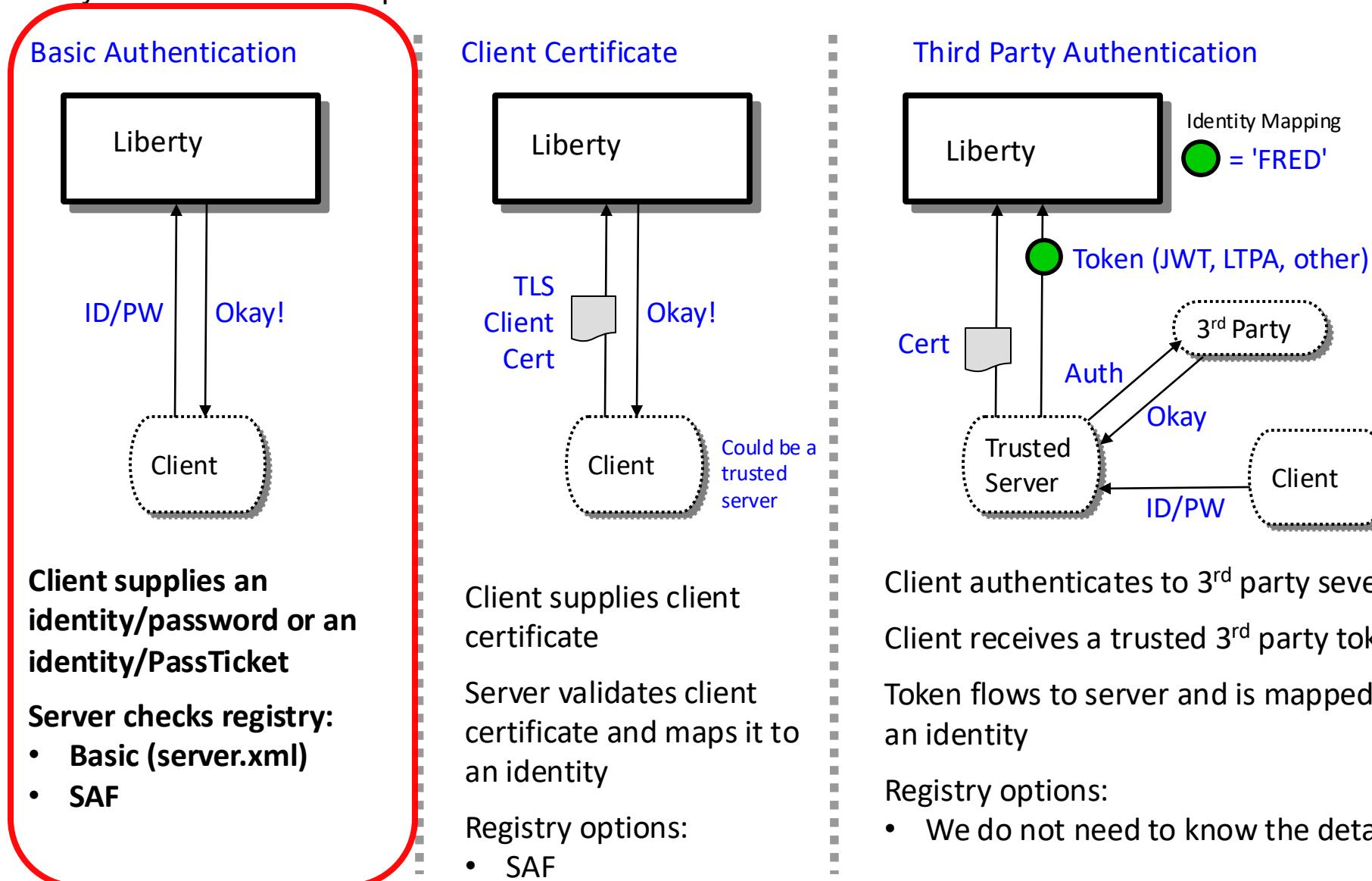
Access to use the GET method to invoke /items would require read access to EJB role *BBGZDFLT.catalogManager.Manager*.

Access to use the GET method to invoke /items/{id} and the POST method to invoke /orders would require read access to EJB role *BBGZDFLT.catalogManager.Staff*.



# Authentication - Basic Authentication

Several different ways this can be accomplished:





# Basic authentication – The client provides an identity and password

- The required server XML security configuration:

```
<featureManager>
    <feature>appSecurity-2.0</feature>
    <feature>zosSecurity-1.0</feature>
</featureManager>

<webAppSecurity allowFailOverToBasicAuth="true" />

<safRegistry id="saf" />
<safAuthorization racRouteLog="ASIS" />
<safCredentials unauthenticatedUser="WSGUEST"
    profilePrefix="BBGZDFLT" />
```

- When sending a request to a Liberty server running z/OS Connect, basic authentication information (identity and password) is provided in the HTTP header in a Basic *Authorization* token with the identity and password encoded using Base64.

- An example with Postman:

The screenshot shows the Postman interface for a GET request to <https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111...>. The 'Auth' tab is selected, and 'Basic Auth' is chosen from the dropdown. A warning message states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables.' Below the dropdown, it says: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'. The 'Username' field contains 'Fred' and the 'Password' field contains '....' with a 'Show Password' checkbox.

The screenshot shows the 'Headers' tab in the Postman interface. The 'Authorization' header is listed with the value 'Basic RnJIZDpmcmVk', which is circled in red. Other headers listed include 'Postman-Token', 'Host', 'User-Agent', 'Accept', 'Accept-Encoding', and 'Connection'. The 'Value' column for each header includes a small info icon and descriptive text like '<calculated when request is sent>' or 'keep-alive'.

**There are multiple ways to provide an identity and password**



- When sending a request to a Liberty server running z/OS Connect, basic authentication information (identity and password) is provided in the HTTP header in a *Basic Authorization* token with the identity and password encoded or formatted using Base64.
    - Examples using the API Explorer feature , cURL, and a Java client.

The screenshot shows a web browser with two tabs open: "REST API Documentation" and "Base64 Encode and Decode". The main content is a Liberty REST API documentation page for the "/cscvinc" service.

**API Endpoint:** /cscvinc/employee

**HTTP Methods:**

- POST:** /cscvinc/employee
- DELETE:** /cscvinc/employee/{employee}
- GET:** /cscvinc/employee/{employee}

**Response Class (Status 200):** OK

**Model Example Value:**

```
    "employeeNumber": "string",
    "name": "string",
    "address": "string",
    "phonenumber": "string",
    "date": "string",
    "amount": "string"
}
```

**Response Content Type:** application/json

**Parameters:**

Parameter	Value	Description
Authorization	Basic dXNlciJpWYXNzd29yZA==	
employee	000050	

**OR**

**Response Messages:**

HTTP Status Code	Reason
404	Not Found

**Try it out!**

A red oval highlights the "Authorization" parameter value. A red box highlights the "Sign in" button in a modal dialog box. The dialog box contains the following text:

mpz3.washington.ibm.com:9443  
This site is asking you to sign in.

Username: user1

Password:

Sign in Cancel

```
Command Prompt
Microsoft Windows [Version 10.0.19043.1165]
(c) Microsoft Corporation. All rights reserved.

c:\>curl -X GET --user FRED:FRED -j -secure https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111
>{"cscvincSelectServiceOperationResponse":{"cscvincContainer":{"response":{"CEIBRESP":0,"CEIBRESP2":0,"USERID":"CICSUSER","filea":{"employeeNumber":"111111","name":"C. BAKER","address":"OTTAWA, ONTARIO","phoneNumber":"51212003","date":"26 11 81","amount":"$0011.00"}}}}}
c:\>
```

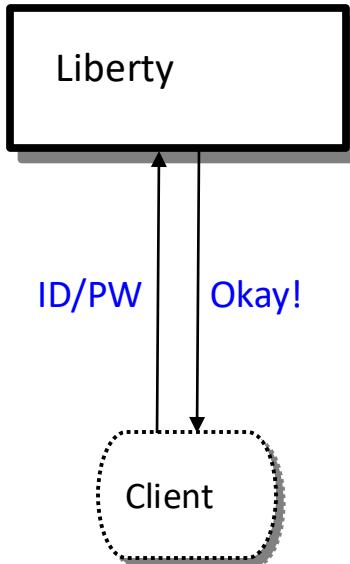
```
49     URL url = new URL("https://wg31.washington.ibm.com:9453/db2/department?dept1=C01&dept2=C01");
50     System.out.println("URL: " + url);
51     HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
52     conn.setRequestMethod("GET");
53     conn.setRequestProperty("Content-Type", "application/json");
54     byte[] bytesEncoded = Base64.encodeBase64("Fred:fredpwd".getBytes());
55     conn.addRequestProperty("Authorization", new String(bytesEncoded));
56
57     try {
58
59         if (conn.getResponseCode() != 200) {
60             throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
61         }
62         BufferedReader bufferReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
63         String output;
64         StringBuilder stringBuffer = new StringBuilder();
65         while ((output = bufferReader.readLine()) != null) {
66             stringBuffer.append(output);
67         }
68         JSONObject json = new JSONObject(stringBuffer.toString());
69         JSONArray jsonArray = json.getJSONArray("ResultSet 1 Output");
70         JSONObject jsonEntry = new JSONObject();
71         for (int index = 0; index < jsonArray.length(); index++) {
72             jsonEntry = jsonArray.getJSONObject(index);
73             if (jsonEntry.has("employeeNumber")){
```



# Authentication - TLS Mutual Authentication

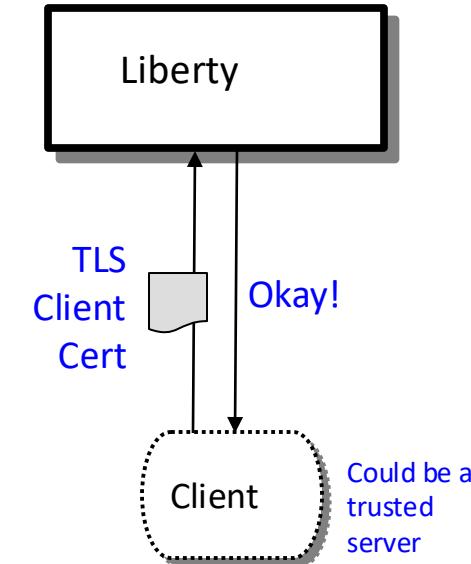
Several different ways this can be accomplished:

## Basic Authentication



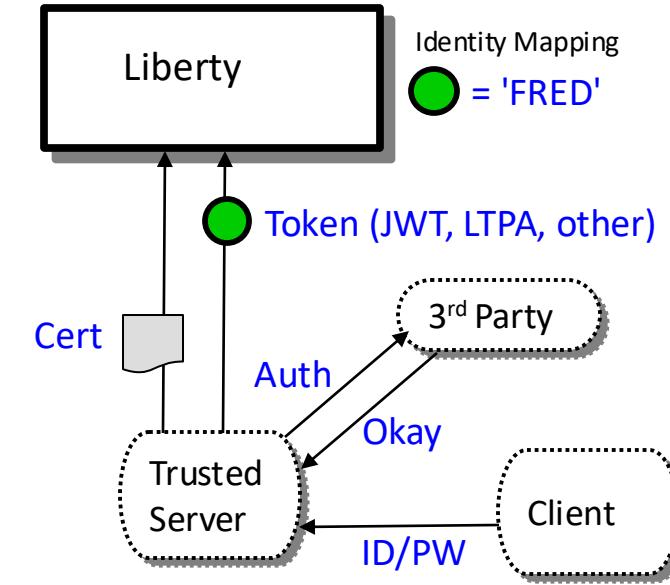
- Server prompts for ID/PW
- Client supplies ID/PW or ID/PassTicket
- Server checks registry:
  - Basic (server.xml)
  - SAF

## Client Certificate



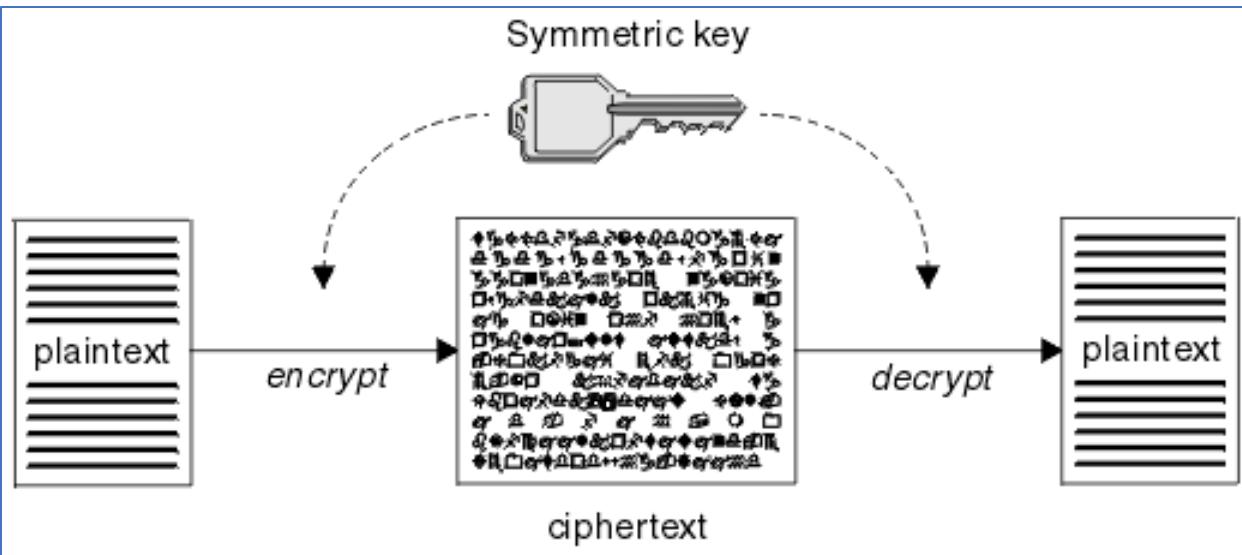
- Server prompts for client certificate.
- Client supplies personal certificate
- Server validates client certificate and maps it to an identity
- Registry options:
  - SAF

## Third Party Authentication

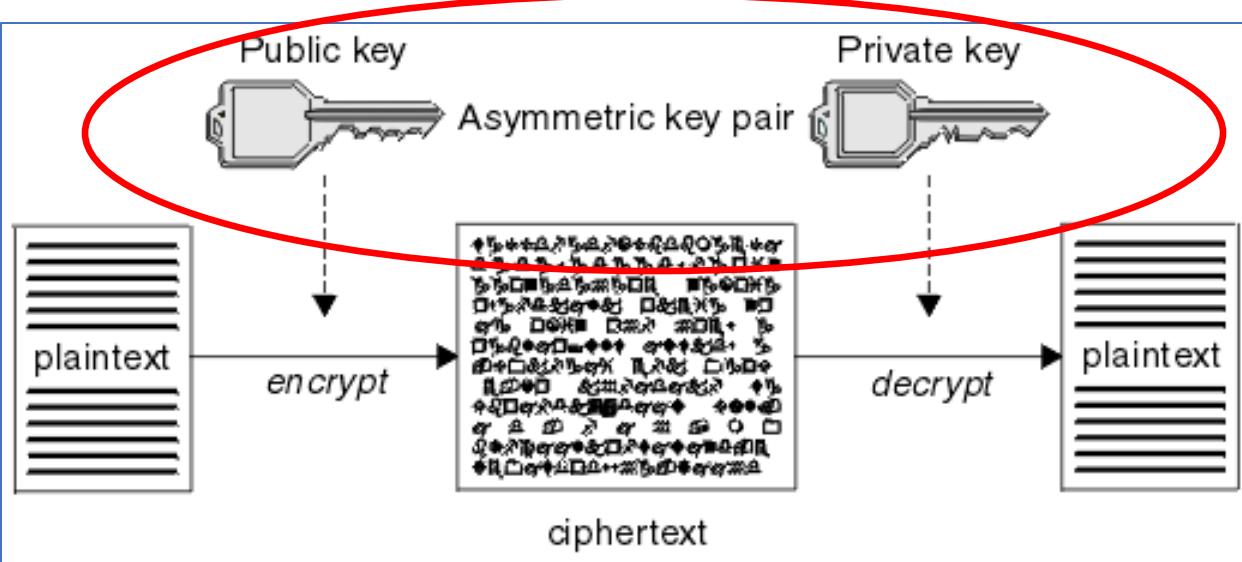


- Client authenticates to 3<sup>rd</sup> party sever
- Client receives a trusted 3<sup>rd</sup> party token
- Token flows to Liberty z/OS and is mapped to an identity
- Registry options:
  - We may not need to know these details.

# Tech-Tip: Symmetric key v. Asymmetric key pairs



A symmetric key is a key shared by the endpoints. Both endpoints use the same key to encrypt and decrypt messages.



An asymmetric key pair is the preferred solution. There is no risk of compromise by sending a symmetric or shared key outside of a protected communication flow.

A message encrypted with a public key can only be decrypted by endpoint that has the private key. The privacy of the messages is ensured.

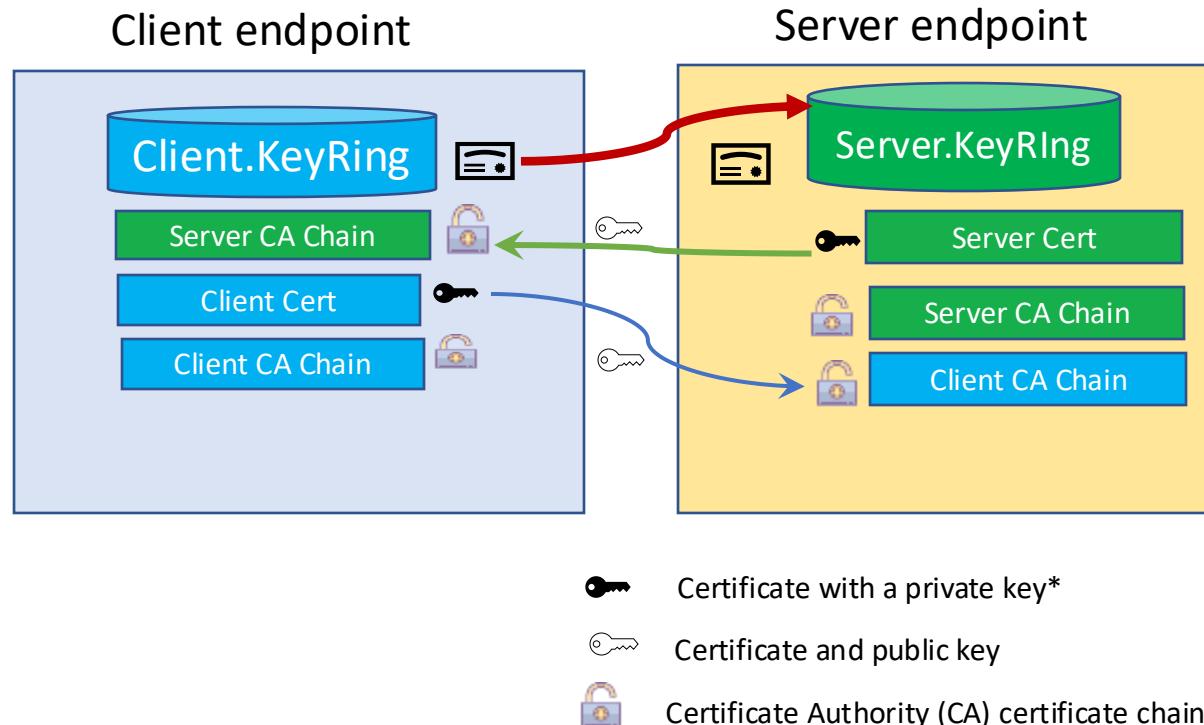
If an endpoint can successfully decrypt a message message encrypted received with a private key, the endpoint sending the message has successfully asserted its validity by proving it has the private used to encrypt the message.

# The basic TLS Handshake Flow (HTTPS)

The HTTPS protocol involves a TLS handshake –

Server Authentication (always occurs when HTTPS is the protocol)

Mutual Authentication (optional, at the request of the server endpoint)



\*For server and/or mutual authentication to work, the endpoint sending the client certificate must use a personal certificate with a private key. The private key is required to decrypt (or encrypt) a message digest that is sent from the other endpoint during the handshake flow. Generation of a message digest also requires access to the CA certificate used to sign the certificate.

#Refers to the set or of certificates used to issue the server or client personal certificate including any intermediate certificates up to and including the root CA.

# Liberty JSSE (HTTPS) server XML configuration



```
<!-- Enable features -->
<featureManager>
    <feature>transportSecurity-1.0</feature>
</featureManager>

<sslDefault sslRef="DefaultSSLSettings"
    outboundSSLRef="OutboundSSLSettings" />

<ssl id="DefaultSSLSettings"
    keyStoreRef="CellDefaultKeyStore"
    trustStoreRef="CellDefaultKeyStore"
    clientAuthenticationSupported="true"
    clientAuthentication="true"/>

<keyStore id="CellDefaultKeyStore"
    location="safkeyring:///Liberty.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />

<ssl id="OutboundSSLSettings"
    keyStoreRef="OutboundKeyStore"
    trustStoreRef="OutboundKeyStore"/>

<keyStore id="OutboundKeyStore"
    location="safkeyring:///zCEE.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
```

SSL repertoires

Key ring for server certificate  
send to for clients

Key ring for client connections to  
server endpoints

## OpenAPI 2

```
<zosconnect_zosConnectManager
    requireAuth="true"
    requireSecure="true|false"/>

<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="catalog"
        requireAuth="true"
        requireSecure="true|false"/>
</zosconnect_zosConnectAPIs>

<zosconnect_services>
    <service id="selectByEmployee"
        name="selectEmployee"
        requireAuth="true"
        requireSecure="true|false"/>
</zosconnect_services>

<zosconnect_apiRequesters>
    requireAuth="true|false"
    <apiRequester name="cscvincapi_1.0.0"
        requireAuth="true"
        requireSecure="true|false"/>
</zosconnect_apiRequesters>
```

safkeyring:///**KeyRing** v safkeyring://**owner/KeyRing**

**Tech/Tip:** Regarding *clientAuthentication* and *clientAuthenticationSupported*. Understand the implications of the interactions between these attributes. There may instances where you want to use HTTPS, but not always with mutual authentication Consider setting *clientAuthentication* to false when setting *clientAuthenticationSupported* to true.



# The Liberty JSSE server XML configuration for outbound connections

```
<!-- Enable features -->
<featureManager>
    <feature>transportSecurity-1.0</feature>
</featureManager>

<ssl id="cicsTLSSettings"
    keyStoreRef="CICSKeyStore"
    trustStoreRef="CICSKeyStore"
    clientKeyAlias="Liberty Client Cert"/>
<keyStore id="CICSKeyStore"
    location="safkeyring:///Liberty.CICS.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
<ssl id="db2TLSSettings"
    keyStoreRef="Db2KeyStore"
    trustStoreRef="Db2KeyStore"
    clientKeyAlias="Liberty Client Cert"/>
<keyStore id="Db2KeyStore"
    location="safkeyring:///Liberty.Db2.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
<ssl id="otherTLSSettings"
    keyStoreRef="OtherKeyStore"
    trustStoreRef="OtherKeyStore">
    <outboundConnection
        host="wg31.washington.ibm.com"
        port="9555"
        clientCertificate="Client Cert"/>
</ssl>
<keyStore id="OtherKeyStore"
    location="safkeyring:///Other.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
```

```
<sslDefault sslRef="DefaultSSLSettings"
    outboundSSLRef="OutboundSSLSettings" />

<zosconnect_authorizationServer sslCertsRef="SSL repertoire"/>
<zosconnect_cicsIpicConnection sslCertsRef="cicsTLSSettings"/>
<zosconnect_db2Connection sslCertsRef="db2TLSSettings"> *
<zosconnect_endpointConnect sslCertsRef= "SSL repertoire"/>
<zosconnect_zosConnectRestClient sslCertsRef="SSL repertoire"/>
<zosconnect_zosConnectServiceRestClientConnection sslCertsRef="SSL repertoire"/>
```

**F BAQSTRT,REFRESH,KEYSTORE**  
**F BAQSTRT,REFRESH,KEYSTORE, ID=CICSKeyStore**  
**F BAQSTRT,REFRESH,KEYSTORE, ID=Db2KeyStore**  
**F BAQSTRT,REFRESH,KEYSTORE, ID=OtherKeyStore**



# Tech/Tip: Combining TLS mutual and basic authentication

```
*****  
/* SET SYMBOLS  
*****  
/*EXPORT EXPORT SYMLIST=(*  
SET CURL= '/usr/lpp/rocket/curl'  
*****  
/* CURL Procedure  
*****  
/*CURL PROC  
/*CURL EXEC PGM=IKJEFT01,REGION=0M  
//SYSTSPRT DD SYSOUT=*  
//SYSERR DD SYSOUT=*  
//STDOUT DD SYSOUT=*  
// PEND  
*****  
/* STEP CURL - use cURL to deploy API cscvinc  
*****  
//DEPLOY EXEC CURL  
BPXBATCH SH export CURL=&CURL; +  
$CURL/bin/curl -X PUT -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc?status=stoped > null; +  
$CURL/bin/curl -X DELETE -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc > null; +  
$CURL/bin/curl -X POST -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
--data-binary @/u/johnson/cscvinc.aar +  
--header "Content-Type: application/zip" +  
https://wg31.washington.ibm.com:9445/zosConnect/apis  
*****  
/* STEP CURL - use cURL to invoke the API cscvinc  
*****  
//(INVOKE EXEC CURL  
//SYSTSIN DD *,SYMBOLS=EXECSYS  
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X GET -s +  
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +  
https://wg31.washington.ibm.com:9445/cscvinc/employee/000100
```

```
<httpEndpoint id="defaultHttpEndpoint"  
host="*"  
httpPort="9080"  
httpsPort="9443" />  
  
<sslDefault sslRef="DefaultSSLSettings"  
outboundSSLRef="DefaultSSLSettings" />  
  
<ssl id="DefaultSSLSettings"  
keyStoreRef="CellDefaultKeyStore"  
trustStoreRef="CellDefaultKeyStore"  
clientAuthenticationSupported="true"  
clientAuthentication="true"/>  
  
<keyStore id="CellDefaultKeyStore"  
location="safkeyring:///Liberty.KeyRing"  
password="password" type="JCERACFKS"  
fileBased="false" readOnly="true" />
```

```
<httpEndpoint id="AdminHttpEndpoint"  
host="*"  
httpPort="-1"  
httpsPort="9445"  
sslOptionsRef="mySSLOptions"/>  
  
<ssLOptions id="mySSLOptions"  
sslRef="BatchSSLSettings"/>  
  
<ssl id="BatchSSLSettings"  
keyStoreRef="CellDefaultKeyStore"  
trustStoreRef="CellDefaultKeyStore"  
clientAuthenticationSupported="true"  
clientAuthentication="false"/>
```

<https://www.rocketsoftware.com/platforms/ibm-z/curl-for-zos>



# Authentication - Third Party Authentication

Several different ways this can be accomplished:

## Basic Authentication



ID/PW

Okay!

Client

Server prompts for ID/PW

Client supplies ID/PW or  
ID/PassTicket

- Server checks registry:
- Basic (server.xml)
  - SAF

## Client Certificate



TLS  
Client  
Cert

Client

Could be a  
trusted  
server

Server prompts for client  
certificate.

Client supplies certificate

Server validates client  
certificate and maps to an  
identity

- Registry options:
- SAF

## Third Party Authentication



Token (JWT, LTPA, other)

Cert

Trusted  
Server

Identity Mapping  
= 'FRED'

3<sup>rd</sup> Party

Auth

Okay

ID/PW

Client

**Client authenticates to 3<sup>rd</sup> party sever**

**Client receives a trusted 3<sup>rd</sup> party token**

**Token flows to Liberty z/OS and is  
mapped to an identity**

**Registry options:**

- We may know these detail.



# Third Party Authentication Examples

Screenshot of the UPS Sign Up page (<https://wwwapps.ups.com/doapp/signup>). The page features a yellow header bar with the text "UPS is open for business: Service impacts related to Coronavirus" and a "More" link. The main content includes the UPS logo, a "Sign Up / Log in" button, a search bar, and a "Feedback" button. A section titled "Sign Up" provides links for existing users ("Log in") and suggests using one of these sites for sign-up: Google, Facebook, Amazon, Apple, and Twitter. Below this, there are fields for entering personal information: Name\*, Email\*, User ID\*, Password\*, and Phone. The "Password" field includes a "Show" link. The "Phone" field is a dropdown menu set to "US +1".

Screenshot of the myNCDMV Sign In page (<https://payments.ncdot.gov/auth>). The page has a "Log In" tab selected, with a "Sign Up" link nearby. It features a background image of autumn foliage. The login form includes fields for "Email Address" (with placeholder "name@example.com") and "Password" (with a "Show Password" link). There is also a "Remember Me" checkbox. Below the form are three social media sign-in options: "Continue with Apple", "Continue with Facebook", and "Continue with Google". A "Forgot Password" link is also present. At the bottom, there is a "Continue as Guest" link and a notice for public computer users about account sharing with Google, Apple, or Facebook.



# Open security standards

- **OAuth** is an open standard for access delegation, used as a way to grant websites or applications access to their information without requiring a password.
- **OpenID Connect** is an authentication layer on top of OAuth. It allows the verification of the identity of an end-user based on authentication performed by an authorization server.
- **JWT (JSON Web token)** defines a compact and self-contained way for securely transmitting information between parties as a JSON object

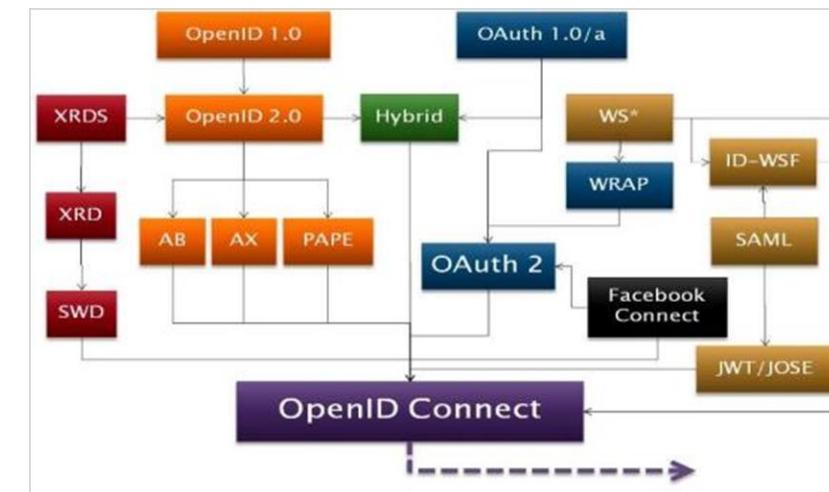
See the YouTube videos:

*OAuth 2.0 and OpenID Connect (in plain English)*

<https://www.youtube.com/watch?v=996OixerHze0>

*OpenID Connect on Liberty*

<https://www.youtube.com/watch?v=fuajCS5bG4c>



# What is a JWT (JSON Web Token) ?

- JWT is a compact way of representing claims that are to be transferred between two parties
- Normally transmitted via HTTP header
- Consists of three parts
  - Header
  - Payload
  - Signature

The screenshot shows the jwt.io debugger interface. At the top, it says "Encoded" and displays a long string of characters: eyJraWQiOiI0cWpYLWJrWE9Vd19GX... The bottom part of this string is circled in red, highlighting the timestamp "Mon Nov 02 2020 11:05:58 GMT-0500 (Eastern Standard Time)". To the right, under "Decoded", the token is split into "HEADER:" and "PAYLOAD:". The HEADER contains the kid and alg fields. The PAYLOAD contains the sub, token\_type, scope, azp, iss, aud, exp, iat, realmName, and uniqueSecurityName fields.

HEADER:
{ "kid": "4qjX- bkXOUw_F_uccjRMkB9ivMjXSQwj0RrkyRJq8DM", "alg": "RS256" }

PAYLOAD:
{ "sub": "Fred", "token_type": "Bearer", "scope": [ "openid", "profile", "email" ], "azp": "rpSsl", "iss": "https://wg31.washington.ibm.com:26213/ /oidc/endpoint/OP", "aud": "myZcee", "exp": 1604333158, "iat": 1604333858, "realmName": "zCEERealm", "uniqueSecurityName": "Fred" }

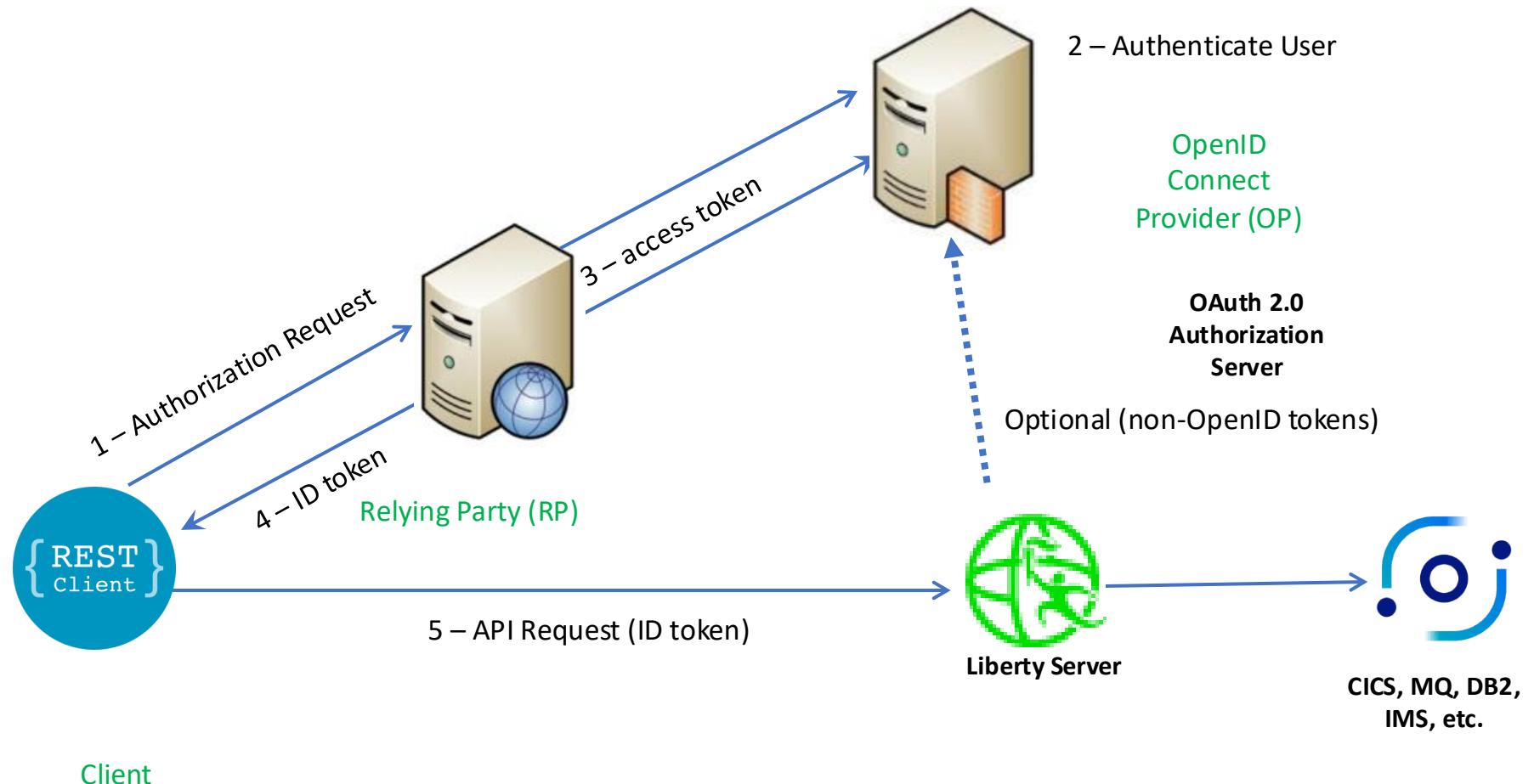
Values derived from the OAUTH configuration:

- signatureAlgorithm="RS256"
- accessTokenLifetime="300"
- resourceIds="myZcee"

<https://jwt.io>



# Typical Authorization Flow for an OpenID Connect token to a z/OS Connect API Provider





# Liberty OpenID Client identity mapping configuration attributes

## Decoded

EDIT THE PAYLOAD AND SECRET

```
HEADER: ALGORITHM & TOKEN TYPE

{
  "kid": "kvjtqdLMjOTWiJrjOr73fu2MMt-FjiQrxU0YBzJLR4o",
  "alg": "RS256"
}

PAYLOAD: DATA

{
  "sub": "auser",
  "token_type": "Bearer",
  "scope": [
    "openid",
    "profile",
    "email"
  ],
  "azp": "rpSsl",
  "iss": "https://wg31.washington.ibm.com:26213
/oidc/endpoint/OP",
  "aud": "myZcee",
  "exp": 1646761228,
  "iat": 1646760928,
  "realmName": "zCEERealm",
  "uniqueSecurityName": "auser"
}
```

```
<safRegistry id="saf" />
<safAuthorization racRouteLog="ASIS" />
<safCredentials unauthenticatedUser="WSGUEST"
  mapDistributedIdentities="true" ←
  profilePrefix="BBGZDFLT" />
```

Use distributed identity filters to map the distributed identities to SAF user IDs, using IDIDMAP resources and the RACMAP command.

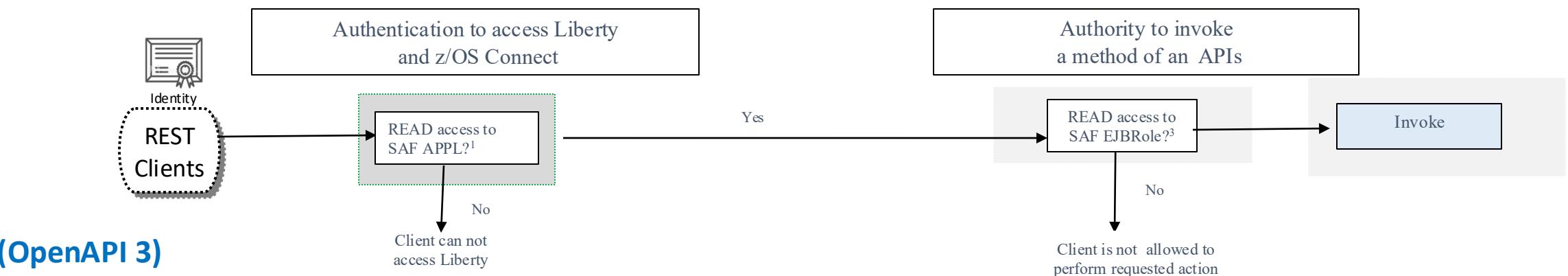
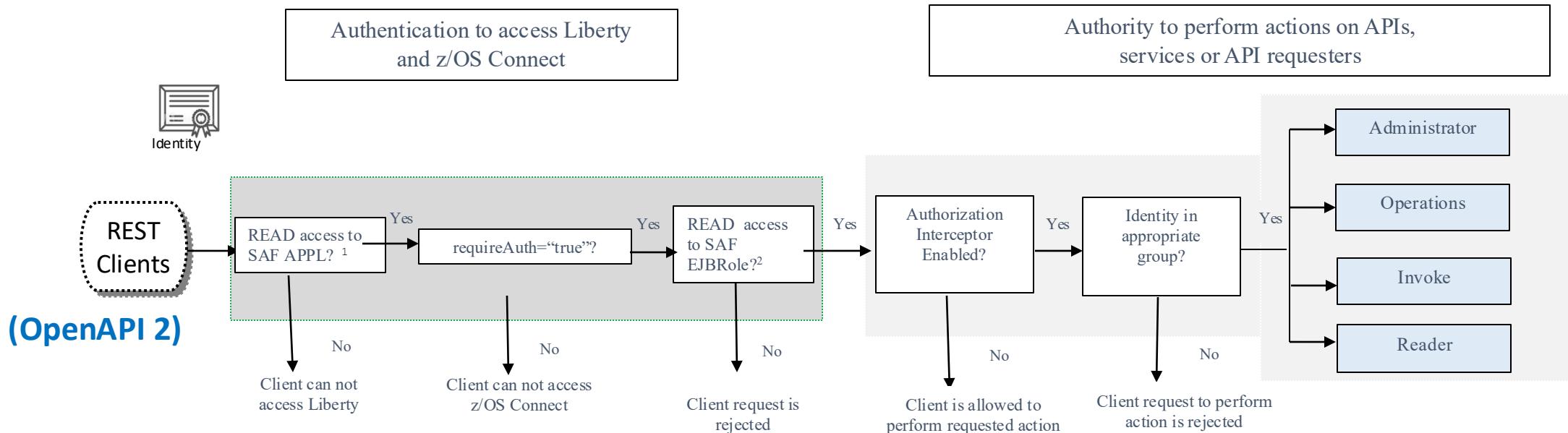
```
<authFilter id="ATSAuthFilter">
  <requestUrl id="ATSDemoUrl"
    name="ATSRefererUri"
    matchType="contains"
    urlPattern="/cscvinc/employee|/db2/employee|mqapi/loan"/>
</authFilter>
<openidConnectClient id="ATS"
  httpsRequired="true"
  authFilterRef="ATSAuthFilter"
  inboundPropagation="required"
  scope="openid profile email"
  audiences="myZcee"
  issuerIdentifier="https://wg31.washington.ibm.com:26213/oidc/endpoint/OP"
  mapIdentityToRegistryUser="false" ←
  signatureAlgorithm="RS256"
  userIdentityToCreateSubject="sub"
  trustAliasName="JWT-Signer-Certificate"
  trustStoreRef="jwtTrustStore"
  authnSessionDisabled="true"
  disableLtpaCookie="true">
</openidConnectClient>
<keyStore fileBased="false" id="jwtTrustStore"
  location="safkeyring:///JWT.KeyRing"
  password="password" readOnly="true" type="JCERACFKS"/>
```

Specifies whether to map the identity to a registry user. If this is set to false, then the user registry (SAF) is not used to create the user subject.

## Authorization

Once we have an identity, then what?

# Security flow with z/OS Connect EE



<sup>1</sup>RDEFINE APPL *profilePrefix*

<sup>2</sup>RDEFINE EJBROLE *profilePrefix.zos.connect.access.roles.zosConnectAccess*

<sup>3</sup>RDEFINE EJBROLE *profilePrefix.resourceName.role*

# z/OS Connect OpenAPI 2 Administrative RESTful APIs



## z/OS Connect administration API

Interface providing meta-data and life-cycle operations for z/OS Connect services, APIs and API requesters.

### APIs : Operations for working with APIs

Show/Hide | List Operations | Expand Operations

GET	/apis	Returns a list of all the deployed z/OS Connect APIs
POST	/apis	Deploys a new API into z/OS Connect
DELETE	/apis/{apiName}	Undeploys an API from z/OS Connect
GET	/apis/{apiName}	Returns detailed information about a z/OS Connect API
PUT	/apis/{apiName}	Updates an existing z/OS Connect API

### Services : Operations for working with services

Show/Hide | List Operations | Expand Operations

GET	/services	Returns a list of all the deployed z/OS Connect services
POST	/services	Deploys a new service into z/OS Connect
DELETE	/services/{serviceName}	Undeploys a service from z/OS Connect
GET	/services/{serviceName}	Returns detailed information about a z/OS Connect service
PUT	/services/{serviceName}	Updates an existing z/OS Connect service
GET	/services/{serviceName}/schema/{schemaType}	Returns the request or response schema for a z/OS Connect service

### API Requesters : Operations that work with API Requesters.

Show/Hide | List Operations | Expand Operations

GET	/apiRequesters	Returns a list of all the deployed z/OS Connect API Requesters
POST	/apiRequesters	Deploys a new API Requester into z/OS Connect and invoke an API Requester call
DELETE	/apiRequesters/{apiRequesterName}	Undeploys an API Requester from z/OS Connect
GET	/apiRequesters/{apiRequesterName}	Returns the detailed information about a z/OS Connect API Requester
PUT	/apiRequesters/{apiRequesterName}	Updates an existing z/OS Connect API Requester

# **z/OS Connect OpenAPI 2 Authorization Functions**



**Operations** - Ability to perform all z/OS Connect operations and actions except for function *Invoke*. The following operations/actions are allowed:

## **APIs:**

- *To obtain a list of all APIs (GET).*\*
- For a specific API, get its details and API Swagger document (GET) and *deploy (POST)\**, update (PUT), start(PUT), stop(PUT), and delete(DELETE) it.

## **Services:**

- *To obtain a list of all services or statistics for all services (GET).*\*
- For a specific service, get its details, request and response schemas, statistics (GET) and *deploy(POST)\**, update(PUT), start(PUT), stop(PUT), and delete(DELETE) it.

## **API Requesters:**

- *To obtain a list of all API requesters (GET).*\*
- For a specific API requester, get its details (GET) and *deploy (POST)\**, update(PUT), start(PUT), stop(PUT), and delete(DELETE) it.

\*These APIs use either the POST or GET method to invoke the REST APIs whose URIs have no path parameter. Therefore, the name of the API, or service or API Requester is ignored. For authorization, only the default or global groups list can be used since no specific group list can be determined (for deployment, the name is embedded in the archive file).

# z/OS Connect OpenAPI 2 Authorization Functions



**Reader** - Ability for:

**APIs:**

- *To obtain a list of all APIs (GET) . \**
- For a specific API, get its details and API Swagger document (GET).

**Services:**

- *To obtain a list of all services (GET) . \**
- For a specific service, get its details and request and response schemas (GET).

**API Requesters:**

- *To obtain a list of all API requesters (GET) . \**
- For a specific API requester, get its details (GET) .

**Invoke** - Ability to invoke user APIs, services and/or API requesters (POST,PUT,GET,DELETE,+).

**Admin** - All z/OS Connect actions are allowed, including all corresponding *Operations*, *Invoke*, and *Reader* actions configured for the same z/OS Connect resource.

\*These APIs use either the POST or GET method to invoke the REST APIs whose URIs have no path parameter. Therefore, the name of the API, service or API Requester is not available. For authorization, only the default or global groups list since no specific group list can be determined (for deployment, the name is embedded in the archive file).

# **z/OS Connect OpenAPI 2 Administrative Groups**



z/OS Connect uses group security for controlling authorization for accessing APIs. There are sets of default global groups for functional roles are configured in a `zosConnectManager` configuration element as shown below:

```
<zosconnect_zosConnectManager  
    globalInterceptorsRef="interceptorList_g"  
    globalAdminGroup="SYSPGRP" globalOperationsGroup="GBLOPERS"  
    globalInvokeGroup="GBLINVKE" globalReaderGroup="GBLRDR"/>
```

There are four classes of groups available controlling z/OS Connect functions, administration, operations, invoking and reader in our server. An authenticated identity membership in one or more of these groups provides access to the corresponding function to that identity.

There is also a way to provide an alternative set of groups for functional roles for specific APIs, services, and API requesters in subordinate configuration elements in our server.

```
<zoscConnectAPI name="cscvinc"  
    adminGroup="CSCADMIN" operationsGroup="CSCOPERS"  
    invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>  
  
<service name="cscvincSelectService"  
    adminGroup="CSCADMIN" operationsGroup="CSCOPERS"  
    invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>  
  
<apiRequester name="cscvinc_1.0.0"  
    adminGroup="CSCADMIN" operationsGroup="CSCOPERS"  
    invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>
```

# Interceptor - server XML example (OpenAPI 2)



```
<zosconnect_zosConnectManager
    globalInterceptorsRef="interceptorList_g"
    globalAdminGroup="SYSPGRP"
    globalOperationsGroup="GBLOPERS"
    globalInvokeGroup="GBLINVKE"
    globalReaderGroup="GBLRDR" />

<zosconnect_authorizationInterceptor id="auth" />
<zosconnect_auditInterceptor id="audit" />
<zosconnect_zosConnectInterceptors id="interceptorList_g"
    interceptorRef="auth"/>
<zosconnect_zosConnectInterceptors id="interceptorList_a"
    interceptorRef="auth,audit"/>

<zosconnect_zosConnectAPIs>
    <zsoConnectAPI name="catalog"
        runGlobalInterceptorsRef="true"
        adminGroup="aapigrp1,aapigrp2"
        operationsGroup="oapigrp1,oapigrp2"
        invokeGroup="iapigrp1,oapigrp2"
        readerGroup="rapigrp1,rapigrp2"/>
</zosconnect_zosConnectAPIs>

<zosconnect_apiRequesters>
    <apiRequester name="cscvincapi_1.0.0"
        runGlobalInterceptorsRef="false"
        interceptorsRef="interceptorList_a"
        adminGroup="aaprgrp1,aaprgrp2"
        operationsGroup="oaprgrp1,oaprgrp2"
        invokeGroup="iaprgrp1,oaprgrp2"
        readerGroup="raprgrp1,raprgrp2"/>
</zosconnect_apiRequesters>

<zosconnect_services>
    <service id="selectByEmployee" name="selectEmployee"
        runGlobalInterceptorsRef="false"
        interceptorsRef="interceptorList_a"
        adminGroup="asrvgrp1,asrvgrp2"
        operationsGroup="osrvgrp1,osrvgrp2"
        invokeGroup="isrvgrp1,isrvgrp2"
        readerGroup="rsrvrgrp1,rsrvgrp2"/>
</zosconnect_services>
```

Global interceptor list –  
authorization  
interceptor only

Alternative interceptor  
list – authorization and  
audit interceptors

This avoids duplication  
of interceptors

Note that these are z/OS  
Connect configuration  
elements. Documented in the  
z/OS Connect KC

# Tech/Tip: Server XML example – combining TLS/AUTH interceptor (OpenAPI 2)



```
<zosconnect_zosConnectManager  
    requireAuth="true"  
    requireSecure="true"  
    globalInterceptorsRef="interceptorList_g"  
    globalAdminGroup="SYSPGRP"  
    globalOperationsGroup="GBLOPERS"  
    globalInvokeGroup="GBLINVKE"  
    globalReaderGroup="GBLRDR"/>  
  
<zosconnect_authorizationInterceptor id="auth"/>  
<zosconnect_zosConnectInterceptors id="interceptorList_g"  
    interceptorRef="auth"/>  
  
<zosconnect_apiRequesters>  
    <apiRequester name="cscvincapi_1.0.0"  
        requireSecure="false"  
        invokeGroup="iaprgrp1"/>  
    </zosconnect_apiRequesters>
```

Global TLS security and authentication are enabled.

TLS security is disabled for this API requester archive artifact. Avoiding the HTTP 302 REDIRECT error.

This configuration would allow a MVS batch job to authenticate to z/OS Connect and use HTTP for the protocol (when an AT-TLS outbound policy is not available). Only authorization identities which are members of groups identified as administrators or invokers would be authorized to invoke this API requester.

**F BAQSTRT,ZCON,CLEARSAFCACHE**



# Tech-Tip: Solution for z/OS Connect Authorization Levels (OpenAPI 2)

```
<zosconnect_zosConnectManager  
    globalInterceptorsRef="interceptorList_g"  
    globalAdminGroup="SYSPGRP" globalOperationsGroup="GBLOPERS, CSCOPERS, DB2OPERS"  
    globalInvokeGroup="GBLINVKE" globalReaderGroup="GBLRDR"/>  
  
<zosconnect_zosConnectAPIs>  
    <zosConnectAPI name="cscvinc" operationsGroup="CSCOPERS" invokeGroup="CSCINV" />  
    <zosConnectAPI name="db2employee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE" />  
</zosconnect_zosConnectAPIs>  
  
<zosconnect_services>  
    <service name="cscvincSelectService" operationsGroup="CSCOPERS" invokeGroup="CSCINV" />  
    <service name="selectEmployee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE" />  
</zosconnect_services>  
  
<zosconnect_apiRequesters>  
    <apiRequester name="cscvincSelectService" operationsGroup="CSCOPERS" invokeGroup="CSCINV" />  
    <apiRequester name="selectEmployee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE" />  
</zosconnect_apiRequesters>
```

- Now members of groups SYSPGRP, GBLOPERS, **CSCOPERS**, **DB2OPERS** and GBLRDR can connect to a z/OS server from the API toolkit.
- Members of groups SYSPGRP, GBLOPERS, **CSCOPERS**, and **DB2OPERS** can deploy new artifacts.
- Only members of group **CSCOPERS** and **DB2OPERS** can manage artifacts after they are deployed.

# Tech-Tip: z/OS Toolkit and authorization status (OpenAPI 2)



Members of CSCOPERS and DB2OPERS can now connect to a server from the API Toolkit

CSCOPERS

The screenshot shows the 'z/OS Connect EE Servers' interface with the 'APIs' section expanded. Under 'APIs (9)', two items are circled in red: 'db2employee (Not Authorized)' and 'selectEmployee (Not Authorized)'. Other listed APIs include cscvinc, filemgr, imsPhoneBook, jwtlpvDemoApi, miniloancics, mqapi, phonebook, and restadmin.

- cscvinc (Started)
- db2employee (Not Authorized)
- filemgr (Started)
- imsPhoneBook (Started)
- jwtlpvDemoApi (Started)
- miniloancics (Started)
- mqapi (Started)
- phonebook (Started)
- restadmin (Started)

- cscvincDeleteService (Started)
- cscvincInsertService (Started)
- cscvincSelectService (Started)
- cscvincService (Started)
- cscvincUpdateService (Started)
- deleteEmployee (Started)
- displayEmployee (Started)
- inquireCatalog (Started)
- inquireSingle (Started)
- insertEmployee (Started)
- jwtlpvDemoService (Started)
- miniloanCICSService (Started)
- miniloanService (Started)
- mqGetService (Started)
- mqPutService (Started)
- placeOrder (Started)
- selectByDepartments (Started)
- selectByRole (Started)
- selectEmployee (Not Authorized)

DB2OPERS

The screenshot shows the 'z/OS Connect EE Servers' interface with the 'APIs' section expanded. Under 'APIs (9)', three items are circled in red: 'cscvinc (Not Authorized)', 'cscvincSelectService (Not Authorized)', and 'selectEmployee (Not Authorized)'. Other listed APIs are identical to the CSCOPERS list.

- cscvinc (Not Authorized)
- db2employee (Started)
- filemgr (Started)
- imsPhoneBook (Started)
- jwtlpvDemoApi (Started)
- miniloancics (Started)
- mqapi (Started)
- phonebook (Started)
- restadmin (Started)

- cscvincDeleteService (Started)
- cscvincInsertService (Started)
- cscvincSelectService (Not Authorized)
- cscvincService (Started)
- cscvincUpdateService (Started)
- deleteEmployee (Started)
- displayEmployee (Started)
- inquireCatalog (Started)
- inquireSingle (Started)
- insertEmployee (Started)
- jwtlpvDemoService (Started)
- miniloanCICSService (Started)
- miniloanService (Started)
- mqGetService (Started)
- mqPutService (Started)
- placeOrder (Started)
- selectByDepartments (Started)
- selectByRole (Started)
- selectEmployee (Started)



# EJB roles for z/OS Connect (OpenAPI 3)

```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />  
  
<webApplication id="CatalogManager" location="${server.config.dir}/apps/api.war" name="CatalogManager"/>  
  
<safRoleMapper profilePattern=%profilePrefix%.%resourceName%.%role%
```

```
openapi: 3.0.0  
...  
servers:  
- url: /  
x-ibm-zcon-roles-allowed:  
- Manager  
...  
paths:  
/items:  
  get:  
    operationId: itemsGet  
    ...  
  /items/{id}:  
    get:  
    ...  
    operationId: itemsIdGet  
    x-ibm-zcon-roles-allowed:  
      - Staff  
orders:  
  post:  
  ...  
  operationId: ordersPost  
  x-ibm-zcon-roles-allowed:  
    - Staff
```

*The value for %role% would be either **Manager** or **Staff**.*

So, the required SAF EJB roles to be defined would be:

- *BBGZDFLT.CatalogManager.Manager*
- *BBGZDFLT.CatalogManager.Staff*

Access to use the GET method to invoke /items would require read access to EJB role *BBGZDFLT.CatalogManager.Manager*. Access to use the GET method to invoke /items/{id} and the POST method to invoke /orders would require read access to EJB role *BBGZDFLT.CatalogManager.Staff*.

## **Flowing identities to back-end z/OS systems**



# Basic authentication - Identity and Password

Server XML Configuration elements where basic authentication can be provided.

```
<connectionFactory id="imsTM"> containerAuthDataRef="IMScredentials">
<authData id="IMScredentials" user= "identity" password= "password"/>

<connectionFactory id="imsDB">
<properties.imsudbJLocal databaseName="DFSIVPA" user="identity" password="password"/>
</connectionFactory>

<zosconnect_cicsIpicConnection id="CICS" authDataRef="CICScredentials"/>
<zosconnect_authData id="CICScredentials" user= "identity" password= "password"/>

<zosconnect_zosConnectServiceRestClientConnection id="Db2" basicAuthRef="db2Auth"/>
<zosconnect_zosConnectServiceRestClientBasicAuth id="db2Auth"
    userName="identity" password="password"/>

<jmsQueueConnectionFactory jndiName="MQ">
    <properties.wasJms userName="identity" password="password" />
</jmsQueueConnectionFactory>
```

The value of the password can be encoded in the server XML configuration file. Using the **securityUtility** shipped with WebSphere Liberty Profile.



# Using securityUtility to encrypt passwords

Best practice : use encryption for passwords instead of base64 encoding

- **securityUtility** – located in <wlp\_install\_dir>/wlp/bin Usage: securityUtility {encode|createSSLCertificate|help} [options]

- For encryption, use encode --key=encryption\_key
  - Specifies the key to be used when encoding using AES encryption. This string is hashed to produce an encryption key that is used to encrypt and decrypt the password. The key can be provided to the server by defining the variable **wlp.password.encryption.key** whose value is the key. If this option is not provided, a default key is used.

```
./securityUtility encode --encoding=aes --key=myKey myPassWord  
{aes}AHO0aXdiVD96u4oMRhoKeYH3U7aDqtFXTuHFBsO98WIb
```

- Support was added at Liberty 22.0.0.1 for storing an AES password encryption key in a SAF key ring, see URL  
<https://www.ibm.com/docs/en/was-liberty/zos?topic=slia-storing-aes-password-encryption-key-in-saf-key-ring>

```
./securityUtility encode --encoding=aes --keyring=safkeyring://JOHNSON/Liberty.KeyRing --keyringType=JCERACFKS  
--keyLabel="Johnson Client Cert" myPassWord
```

- Also supports 1-way hash encoding – for passwords in server.xml with basicRegistry

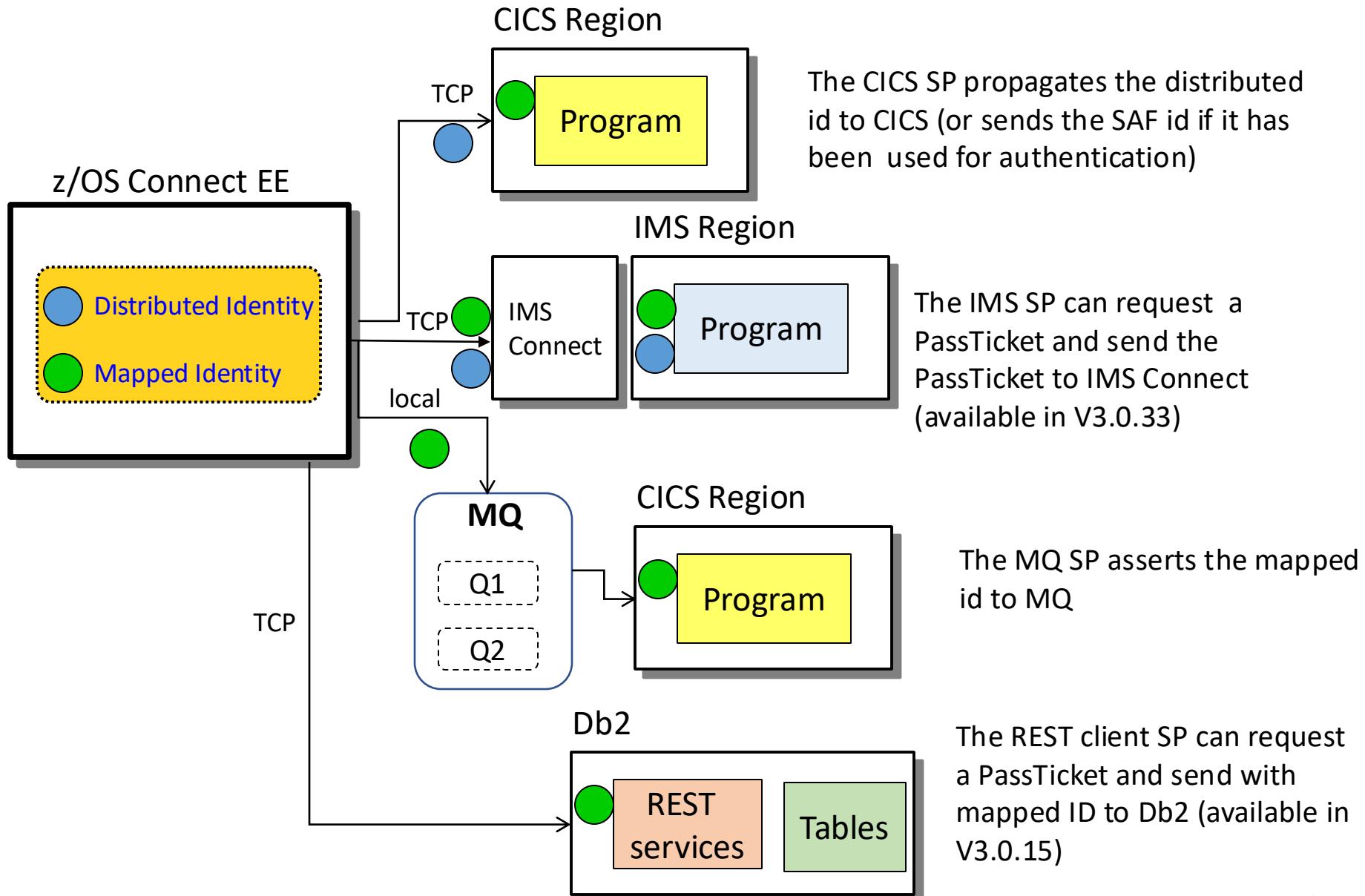
- For hash, use encode --encoding=hash

```
./securityUtility encode --encoding=hash XXXXXXXX  
{hash}ATAAAAAIcqTmHn5qZahAAAAAIMjzy+hP8YFaI06LiCreVe4etRLUS9a25eVuYtx6WKiv
```

See the WebSphere Application Server for z/OS Liberty *securityUtility* command at URL:

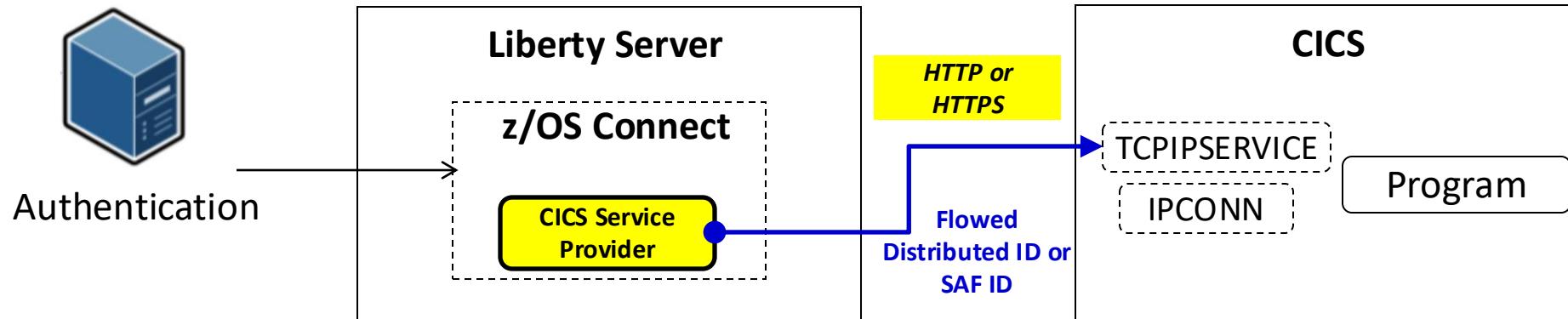
<https://www.ibm.com/docs/en/was-liberty/zos?topic=applications-securityutility-command>

# Flowing an identity to a back-end subsystem





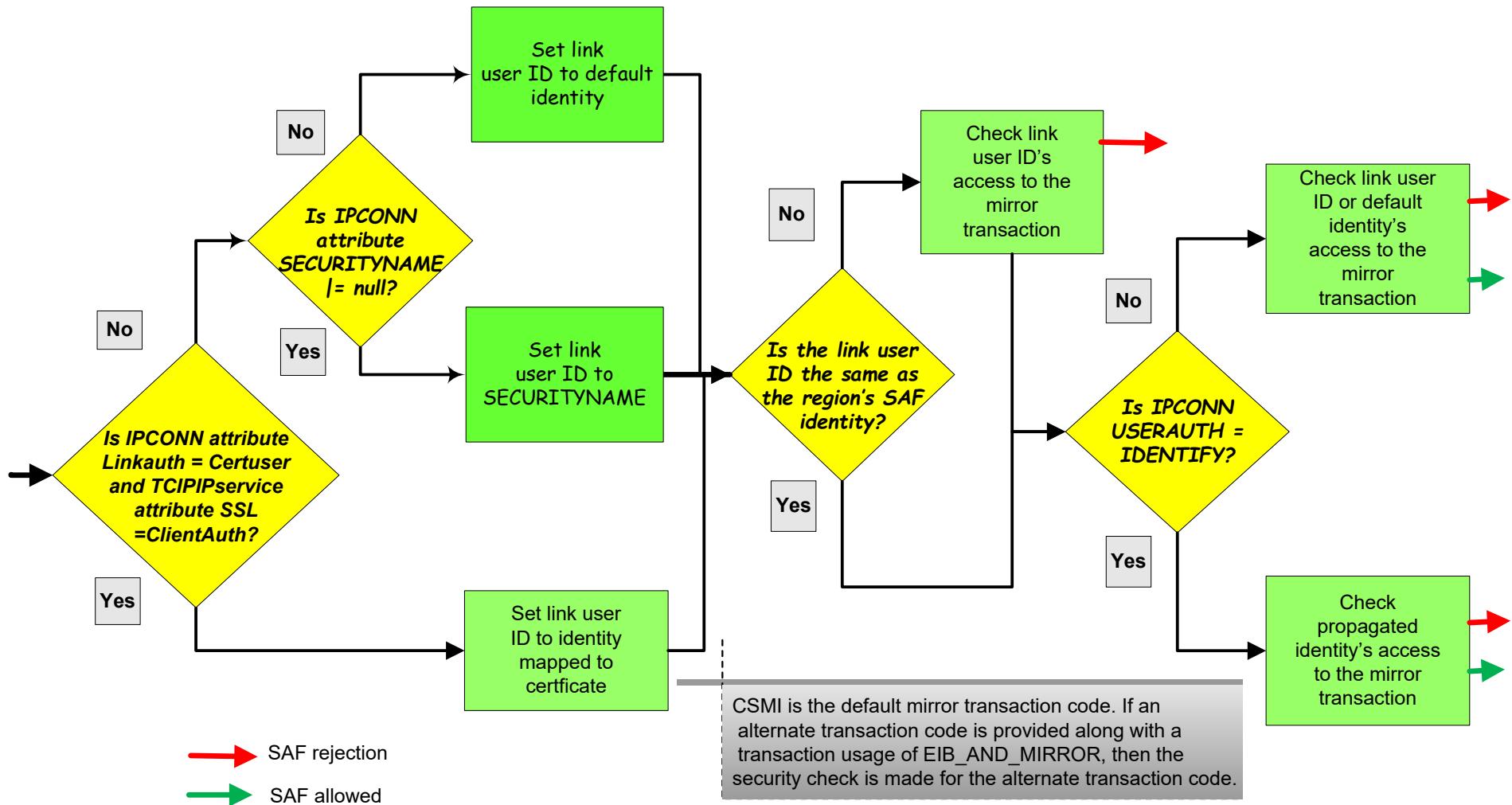
# Flowing a user ID with CICS service provider



Distributed identities can be propagated to CICS and then mapped to a RACF user ID by CICS. You can then view the distinguished name and realm for a distributed identity in the association data of the CICS task. **Important:** If the z/OS Connect server is not in the same Sysplex as the CICS system, you must use an IPIC TLS (JSSE) connection that is configured with client authentication.

If a SAF ID is used for authentication (e.g., basic authentication with a SAF registry) then the SAF ID is passed to CICS.

# Tech/Tip: CICS IPIC Security with USERAUTH(VERIFY)





# Flowing an identity to CICS

The zosconnect\_cicsIpicConnection element is the key :

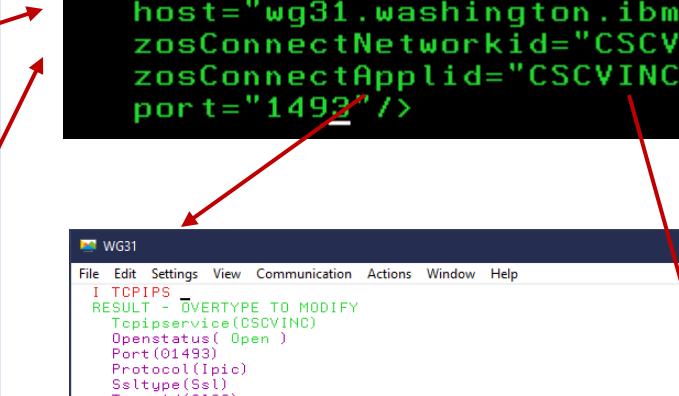
inquireSingle Service

**Configuration**

**Required Configuration**

Enter the required configuration for this service.

Coded character set identifier (CCSID):

Connection reference:  

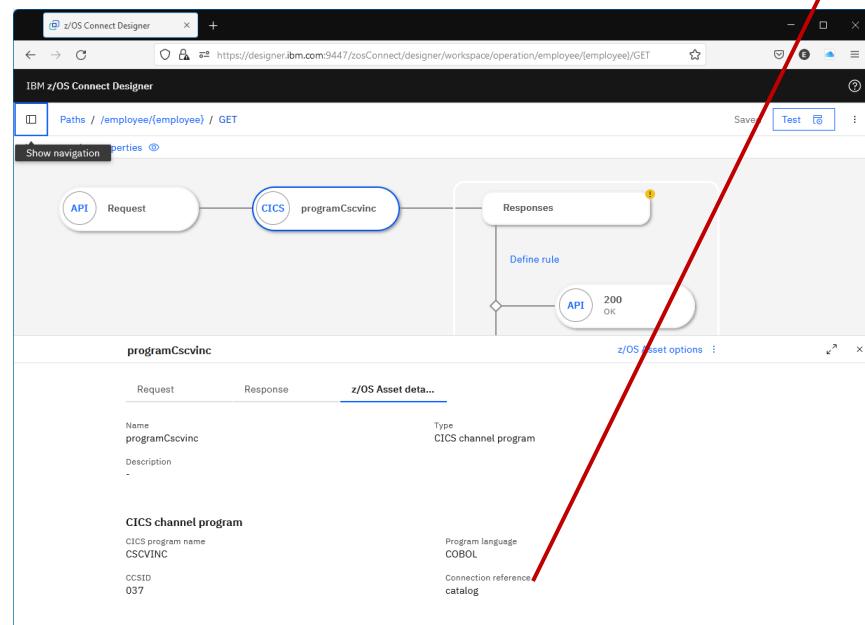
**Optional Configuration**

Enter the optional configuration for this service.

Transaction ID:

Transaction ID usage:

**Overview** **Configuration**



```
<zosconnect_cicsIpicConnection id="catalog">
  host="wg31.washington.ibm.com"
  zosConnectNetworkid="CSCVINC"
  zosConnectApplid="CSCVINC"
  port="1493" />
```

WG31

File Edit Settings View Communication Actions Window Help

I TCPIPS

RESULT - OVERTYPE TO MODIFY

Tcpipservice(CSCVINC)

Openstatus( Open )

Port(01493)

Protocol(Ipic)

Ssltype(Ssl)

Transid(CIIS)

Authenticate(Noauthentic)

Connections(00000)

Backlog( 01024 )

Maxdatalen( 000000 )

Urm( DFHISAIP )

Privacy(Supported)

Ciphers(3538392F3233)

Host(ANY)

Ipaddress(192.168.17.201)

Hosttype(Any)

Ipresolved(192.168.17.201)

+ Ipfamily(Ipv4family)

SYSID PF 1 HELP 2 HEX 3 END 5 VAR 7 SBH 8 SFH TIME: 13.1

MB D

Connected to remote server/host wg31 using lu/pool TCP00137 and port 23

## CICS TCPIPService

WG31

File Edit Settings View Communication Actions Window Help

I IPConn

RESULT - OVERTYPE TO MODIFY

Ipconn(CSCVINC)

Applid(CSCVINC)

Networkid(CSCVINC)

Servstatus( Inservice )

Connstatus( Released )

Ssltype( Nossi )

Purgeinterval( )

Ha(Notrequired)

Receivecount(001)

Sendcount(000)

Tcpipservice(CSCVINC)

Port()

Host()

Hosttype()

Ipresolved(0.0.0.0)

Ipfamily(Unknown)

Pendstatus( Notpending )

+ Recovstatus( Norecovdata )

SYSID=CICS APPLID=CICS53Z TIME: 12.36.15 DATE: 02/22/21

PF 1 HELP 2 HEX 3 END 5 VAR 7 SBH 8 SFH 10 SB 11 SF

MB D

Connected to remote server/host wg31 using lu/pool TCP00135 and port 23

## CICS IPCONN

# CICS IPCONN Resource



```
DEFINE IPCONN (ZOSCONN)
  GROUP (SYSPGRP)
  APPLID (ZCAPPL)
  NETWORKID (ZCNETID)
  TCPIPSERVICE (ZOSCONN)
  LINKAUTH (SECUSER | CERTUSER)
  USERAUTH (IDENTIFY)
  IDPROP (REQUIRED | OPTIONAL)
```

Must match `zosConnectApplid` set in  
`zosconnect_cicsIpicConnection`

Must match `zosConnectNetworkid` set in  
`zosconnect_cicsIpicConnection`

```
<zoscconnect_cicsIpicConnection
  id="cscvinc"
  host="wg31.washington.ibm.com"
  zosConnectApplid="ZOSAPPL"
  zosConnectNetworkid="ZCNETID"
  port="1491"/>
```

**LINKAUTH** Determines the user identity to be used for link security. The value is either **CERTUSER** or **SECUSER**. A value of **CERTUSER** sets the link identity to the identity associated with the client certificate received from the client endpoint (TLS mutual authentication is required). A value of **SECUSER** sets the link identity to the value of the *SECURITYNAME* attribute as defined in the IPConn resource.

**USERAUTH** Identifies how the identity under which the attached transaction attach security will run. Since a password is not available, a value of **VERIFY** is not possible. A value of **LOCAL** means the current link identity is used. A value of **DEFAULTUSER** means the CICS default identity is used. For identity propagation purposes, the value of **USERAUTH** should be **IDENTIFY** (no password will be required) so the identity provided by the client is used for executing the attached transaction. TLS must be used if the client is in a different Sysplex.

**IDPROP** Determines whether the original distributed identity authenticated by the z/OS Connect server is also propagated to CICS in addition to the mapped identity used for z/OS Connect authorization checks. A value of **NOTALLOWED** does not propagate the original distributed identity. A value of **OPTIONAL** will propagate to CICS the original distributed identity, if available. A value of **REQUIRED** requires that the original distributed identity be propagated to CICS. TLS must be used if the client is in a different Sysplex.

**CERTIFICATE** Provides the label of the certificate connected to the CICS key ring to be used for server endpoint certificate during a TLS handshake.



# Identity Propagation and CICS High Availability

Assume the service installed in a server files use the following *Connection reference* values:

- cscvinc
- catalog
- miniloan

If identity propagation is required for all connection, then the CICS IPCONN resources defined in the CICs that correspond to a zosconnect\_cicsIpicConnection configuration elements must be dedicated to that z/OS Connect server and connection reference can not be reused.

Simplify administration by still sharing a common cicsIpicConnection XML configuration element by using variables and a bootstrap properties file or “variables” XML file

Server baqsvr1's bootstrap.properties

```
ipicPort=1491  
cicsHost=dvipa.washington.ibm.com  
serverPrefix=baqsvr1
```

Server baqsvr2's bootstrap.properties

```
cicsHost=dvipa.washington.ibm.com  
ipicPort=1491  
serverPrefix=baqsvr2
```

Server baqsvr3's bootstrap.properties

```
cicsHost=dvipa.washington.ibm.com  
ipicPort=1491  
serverPrefix=baqsvr3
```

ipicIDProp.xml

```
<zosconnect_cicsIpicConnection id="cscvinc"  
host="${cicsHost}"  
zosConnectNetworkid="${wlp.server.name}"  
zosConnectApplid="${wlp.server.name}"  
sharedPort="true" port="${ipicPort}"/>  
<zosconnect_cicsIpicConnection id="catalog"  
host="${cicsHost}"  
zosConnectNetworkid="${serverPrefix}C"  
zosConnectApplid="${serverPrefix}C"  
sharedPort="true" port="${ipicPort}"/>  
<zosconnect_cicsIpicConnection id="miniloan"  
host="${cicsHost}"  
zosConnectNetworkid="${serverPrefix}M"  
zosConnectApplid="${serverPrefix}M"  
sharedPort="true" port="${ipicPort}"/>
```

→ baqsvr1 or baqsvr2

→ baqsvr1C or baqsvr2C

→ baqsvr1M or baqsvr2M



# CICS IPCONN and TCPIPSERVICE resources for HA

## CICS Specific TCPIPSERVICE - IPIC

```
TCpipservice : IPIC1
GROup       : SYSPGRP
Urm         ==> DFHISAIP
POrtnumber  ==> 01492
STatus      ==> Open
PROtocol    ==> IPic
TRansaction ==> CISS
Host        ==> ANY
Ipaddress   ==> ANY
SPeciftcps ==>
```

## CICS Generic TCPIPSERVICE - IPICG

```
TCpipservice : IPICG1
GROup       : SYSPGRP
Urm         ==> DFHISAIP
POrtnumber  ==> 01491
STatus      ==> Open
PROtocol    ==> IPic
TRansaction ==> CISS
Host        ==> ANY
Ipaddress   ==> ANY
SPeciftcps ==> IPIC
```

A client connects first to the CICS region's generic port (1491) and then the CICS region redirects the client to the region's specific port (1492).

## I IPCONN ACQ

STATUS: RESULTS - OVERTYPE TO MODIFY

```
Ipc(BAQSVR1 ) App(BAQSVR1) Net(BAQSVR1) Ins Acq Nos
      Rece(001) Sen(000) Tcp(IPIC )
Ipc(BAQSVR1C) App(BAQSVR1C) Net(BAQSVR1C) Ins Acq Nos
      Rece(001) Sen(000) Tcp(IPIC )
Ipc(BAQSVR1M) App(BAQSVR1M) Net(BAQSVR1M) Ins Acq Nos
      Rece(001) Sen(000) Tcp(IPIC )
Ipc(BAQSVR2 ) App(BAQSVR2) Net(BAQSVR2) Ins Acq Nos
      Rece(001) Sen(000) Tcp(IPIC )
Ipc(BAQSVR2C) App(BAQSVR2C) Net(BAQSVR2C) Ins Acq Nos
      Rece(001) Sen(000) Tcp(IPIC )
Ipc(BAQSVR2M) App(BAQSVR2M) Net(BAQSVR2M) Ins Acq Nos
      Rece(001) Sen(000) Tcp(IPIC )
```

Number of  
IPCONN resources  
equals the number  
of zCEE server  
times the number of  
unique connection  
references

<sup>1</sup>CICS requires the specific TCPIPSERVICE be installed before the corresponding generic TCPIPSERVICE resource. TCPIPServices are installed in alphabetically order, so the name of specific service must be alphabetically prior to the name of the generic TCPIPSERVICE.



# CICS IPIC connection processing for high availability load balancing\*

If the *reconnectInterval* attribute is set, at the specified time interval, a check is made to see if a new connection attempt should be attempted

A new connection is established if the current connection properties are not the preferred connection properties:

- If *reconnectInterval*, *preferredSpecificHost* and *preferredSpecificPort* are not set,
  - New connection attempts are disabled (this is the default behavior).
- If *reconnectInterval* is set and *preferredSpecificHost* and *preferredSpecificPort* are not set,
  - A new connection is attempted at the interval specified by the *reconnectInterval* time. Use this to enable regular connection rebalancing.
- If *reconnectInterval* and *preferredSpecificPort* are set and *preferredSpecificHost* is not set,
  - A new connection is attempted at the expiration time interval and if the current connected port in use does not match the preferred port
  - Relevant when shared port is for a single LPAR
  - Specific CICS region is preferred
- If *reconnectInterval* and *preferredSpecificHost* are set and *preferredSpecificPort* is not set
  - A new connection is attempted at the expiration time interval and if the current host in use does not match the preferred port
  - Relevant when shared port is across Sysplex
  - Any CICS region on a specific LPAR is preferred
- If *reconnectInterval*, *preferredSpecificHost* and *preferredSpecificPort* are all set
  - A new connection is attempted at the expiration time interval time and if both the current host and port in use do not match the preferred host and port
  - Relevant when shared port is on a single LPAR or across a Sysplex
  - Specific CICS region is preferred.

When the reconnection attempt results in a new connection to a CICS region, new requests are sent over the new connection. Previous connections will continue and when all requests have completed processing, the previous or old connection will be closed.



## Tech/Tip: A PassTicket provides an alternative to a password

- A PassTicket is generated by or for a client by using a secured sign-on key (whose value is masked or encrypted) to encrypt a valid *RACF identity* combined with the *application name* of the targeted resource. Also embedded in the PassTicket is a time stamp (based on the current Universal Coordinated Time (UCT)) which sets the time when the PassTicket will expire (usually 10 minutes).
- Access to PassTickets is managed using the RACF PTKTDATA class.
- For z/OS Connect, a RACF PassTicket can be used for basic authentication when connecting from any REST client on any platform to a z/OS Liberty server and for requests from a z/OS Connect server accessing IMS and Db2.
- ***PassTickets do not have to be generated on z/OS using RACF services.*** IBM has published the algorithm used to generate a PassTickets, see manual *z/OS Security Server RACF Macros and Interfaces, SA23-2288-40*. *Github has examples using Java, Python and other example are available on other sites.*

# PassTickets and IMS

- Basic authentication to IMS Connect using a PassTicket depends on the APPL parameters configured in IMS Connect.

```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1, MEMBER=HWSMEM, DRU=HWSYDRU0,
TMEMBER=OTMAMEM, APPL=IMSTMAPL)
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1),
DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000, APPL=IMSDBAPL)
```

```
RDEFINE PTKTDATA IMSTMAPL SSIGNON(0123456789ABCDEF) APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
```

```
RDEFINE PTKTDATA IRRPTAAUTH.IMSTMAPL.* UACC(NONE)
```

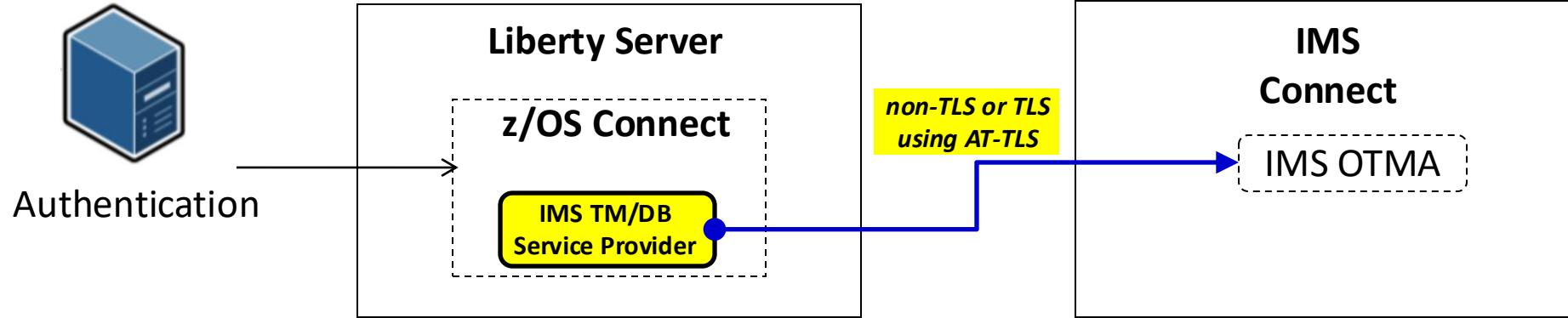
```
PERMIT IRRPTAAUTH.IMSTMAPL.* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
```

```
RDEFINE PTKTDATA IMSDBAPL SSIGNON(0123456789ABCDEF) APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
```

```
RDEFINE PTKTDATA IRRPTAAUTH.IMSDBAPL.* UACC(NONE)
```

```
PERMIT IRRPTAAUTH.IMSDBAPL.* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
```

# Flowing an identity to IMS Connect (TM)



```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)  
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)  
DATASTORE=(GROUP=OTMAGRP,ID=IVP1, MEMBER=HWSMEM, DRU=HWSYDRU0,  
TMEMBER=OTMAMEM, APPL=IMSTMPL)
```

## Authentication options:

1. User ID / password
2. PassTicket support

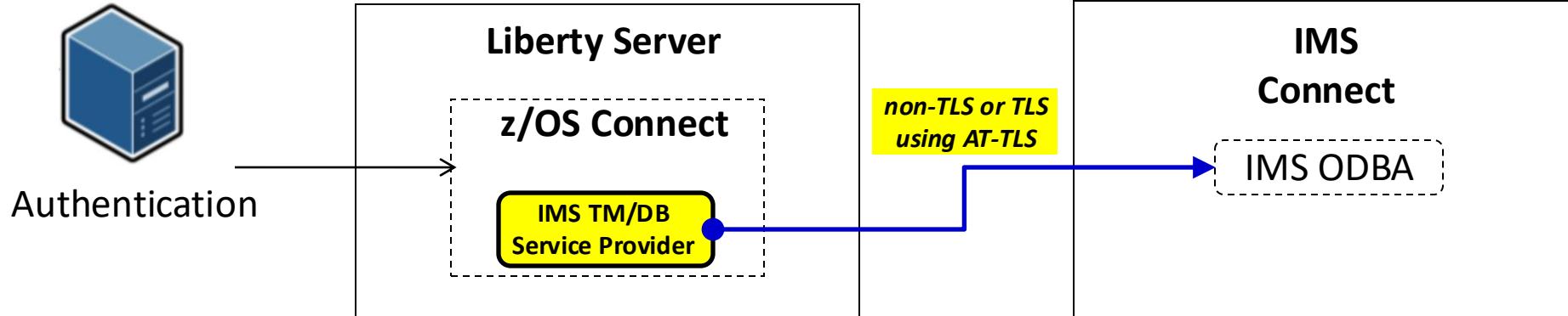
```
<connectionFactory containerAuthDataRef="Connection1_Auth" id="IVP1">  
<properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000"/>  
</connectionFactory>  
<authData id="Connection1_Auth" user="USER1" password="{xor}GhIPExAGDwg="/>
```

Specify a user identity and password to be used in the request to IMS Connect

```
<connectionFactory containerAuthDataRef="Connection1_Auth" id="IVP1">  
<properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000  
applicationName="IMSTMPL"/>  
</connectionFactory>
```

Request a PassTicket  
And use it in the request to IMS Connect

# Flowing an identity to IMS Connect (DB)



```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)  
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)  
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1),  
DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSDBAPL)
```

## Authentication options:

1. User ID / password
2. PassTicket support

```
<connectionFactory id="DFSIVPACConn"> <properties.imsudbJLocal  
databaseName="DFSIVPA" datastoreName="IVP1" portNumber="5555" driverType="4"  
datastoreServer="wg31.washington.ibm.com" flattenTables="True"  
user="USER1" password="USER1" />  
</connectionFactory>
```

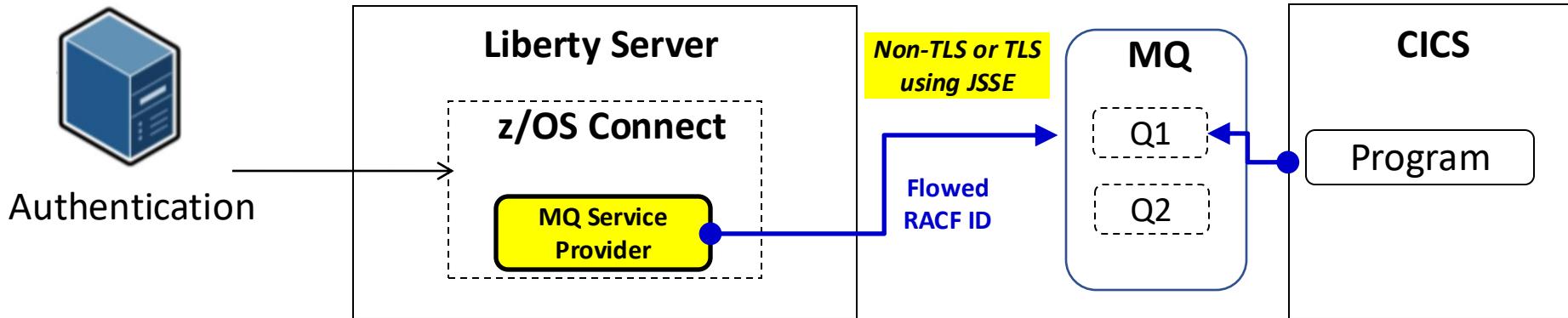
Specify a user identity and password to be used in the request to IMS Connect

```
<connectionFactory id="DFSIVPACConn"> <properties.imsudbJLocal  
databaseName="DFSIVPA" datastoreName="IVP1" portNumber="5555"  
datastoreServer="wg31.washington.ibm.com<< driverType="4" flattenTables="True"  
applicationName="IMSDBAPL" "/>  
</connectionFactory>
```

Request a PassTicket  
And use it in the request to IMS Connect



# Flowing a user ID with MQ service provider



Set **useCallerPrincipal=true** to flow the authenticated RACF user ID

```
<zosconnect_services>
  <service name="mqPut">
    <property name="destination" value="jms/default"/>
    <property name="useCallerPrincipal" value="true"/>
  </service>
</zosconnect_services>
```

Define identity propagation to MQ

# PassTickets and Db2

- ☐ Basic authentication Db2 using a PassTicket depends on the Db2 configuration.

```
DSNL080I -DSN2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:  
DSNL081I STATUS=STARTD  
DSNL082I LOCATION          LUNAME          GENERICCLU  
DSNL083I DSN2LOC           USIBMWZ.DSN2APPL  USIBMWZ.DSN0APPL  
DSNL084I TCPPORT=2446    SECPORT=2445   RESPORT=2447  IPNAME--NONE  
DSNL085I IPADDR=:192.168.17.201  
DSNL086I SQL      DOMAIN=WG31.WASHINGTON.IBM.COM  
DSNL105I CURRENT DDF OPTIONS ARE:  
DSNL106I PKGREL = COMMIT  
DSNL106I SESSIDLE = 001440  
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

```
DSNL080I -DSNC DSNLTDDF DISPLAY DDF REPORT FOLLOWS:  
DSNL081I STATUS=STARTD  
DSNL082I LOCATION          LUNAME          GENERICCLU  
DSNL083I DSN2LOC           -NONE          -NONE  
DSNL084I TCPPORT=2446    SECPORT=2445   RESPORT=2447  IPNAME=DB2IPNM  
DSNL085I IPADDR=:192.168.17.252  
DSNL086I SQL      DOMAIN=WG31.WASHINGTON.IBM.COM  
DSNL086I RESYNC DOMAIN=WG31.WASHINGTON.IBM.COM  
DSNL089I MEMBER IPADDR=:192.168.17.252  
DSNL105I CURRENT DDF OPTIONS ARE:  
DSNL106I PKGREL = COMMIT  
DSNL106I SESSIDLE = 001440  
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

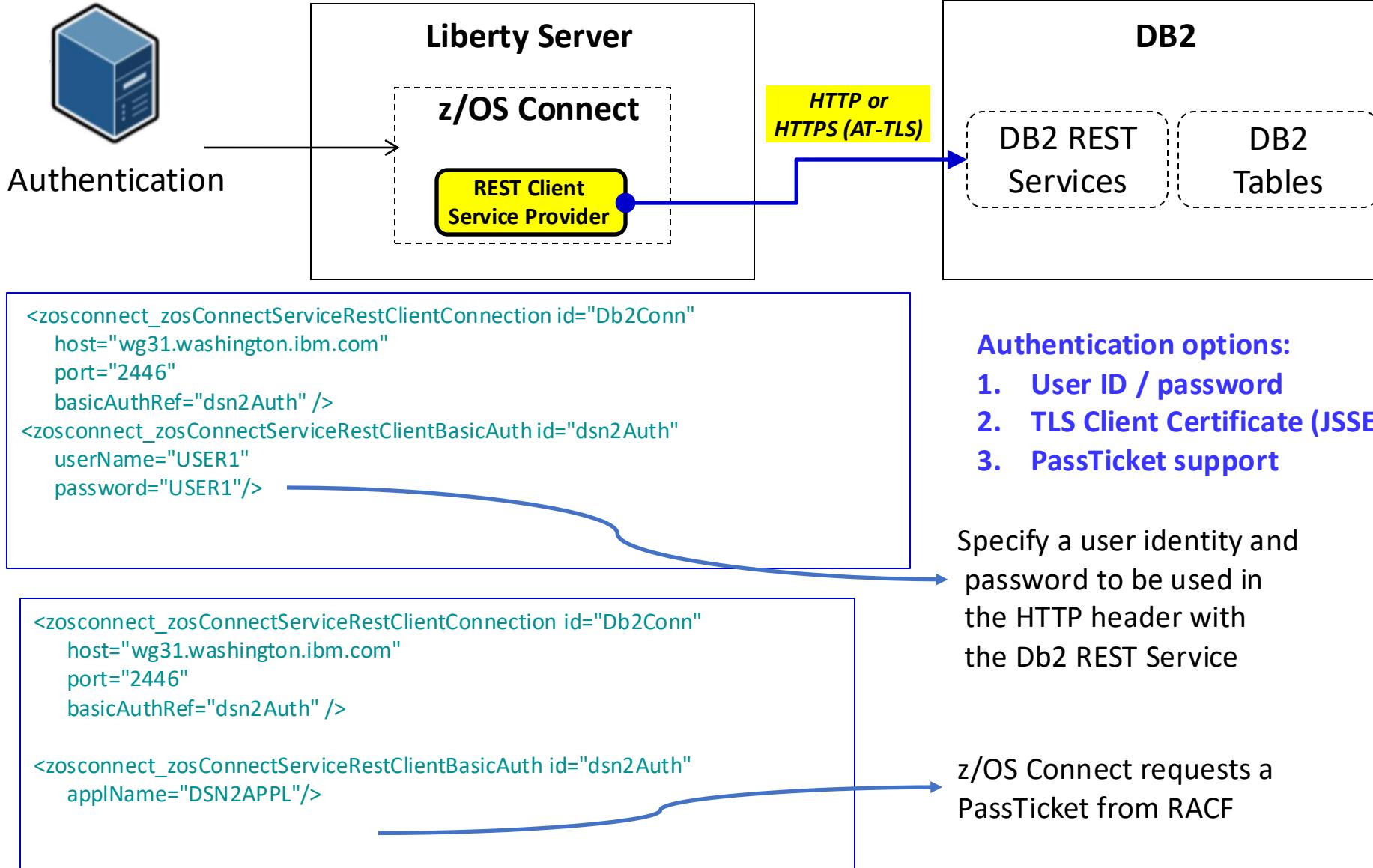
Which value should be used for *appName* is determined for use in RACF resources is determined as shown below.

- ☐ If *GENERICLU* is defined, use the second part of *GENERICLU* for *appName*, e.g., ***DSN0APPL***
- ☐ If *GENERICLU* is not defined, use the second part of *LUNAME* for *appName*, e.g., ***DSN2APPL***
- ☐ If neither *GENERICLU* or *LUNAME* is defined, use the value of the *IPNAME* for *appName*, e.g., ***DB2IPNM***

```
RDEFINE PTKTDATA DSN2APPL SSIGNON(0123456789ABCDEF)  APPLDATA('NO REPLAY PROTECTION') UACC(NONE)  
RDEFINE PTKTDATA IRRPTAAUTH.DSN2APPL.* UACC(NONE)  
PERMIT IRRPTAAUTH.DSN2APPL.* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
```



# Flowing the identity for the REST client SP (Db2)



# Tech/Tip: Db2 REST Security

- ❑ Access to Db2 REST services requires READ access to the Db2 subsystem DSNR REST resource. i.e., permit READ access to this resource to the identity in question, for example

```
PERMIT DSN2.REST CLASS(DSNR) ID(USER2) ACC(READ) where DSN2 is the Db2 subsystem ID  
SETROPTS RACLIST(DSNR) REFRESH
```

- ❑ Db2 package access is also required. If a user is not able to display a valid Db2 REST services in the z/OS Connect Db2 services development tooling or by using a **POST** to the Db2 provided REST interface URL of <http://wg31.washington.ibm.com:2446/services/DB2ServiceDiscover>, then they may not have sufficient access to the package containing the service.

For example, if service *zCEEService.selectEmployee* is defined to Db2 but not visible in the z/OS Connect tooling or if a **GET** request to URL <http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee> fails with message:

```
{  
  "StatusCode": 500,  
  "StatusDescription": "Service zCEEService.selectEmployee discovery failed due to  
  SQLCODE=-551 SQLSTATE=42501, USER2 DOES NOT HAVE THE PRIVILEGE TO PERFORM OPERATION EXECUTE  
  PACKAGE ON OBJECT zCEEService.selectEmployee. Error Location:DSNLJACC:35"  
}
```

The user needs to be granted execute authority on package *zCEEService.selectEmployee* with command:

```
GRANT EXECUTE ON PACKAGE "zCEEService"."selectEmployee" TO USER2 or  
GRANT EXECUTE ON PACKAGE "zCEEService"."*" TO USER2
```



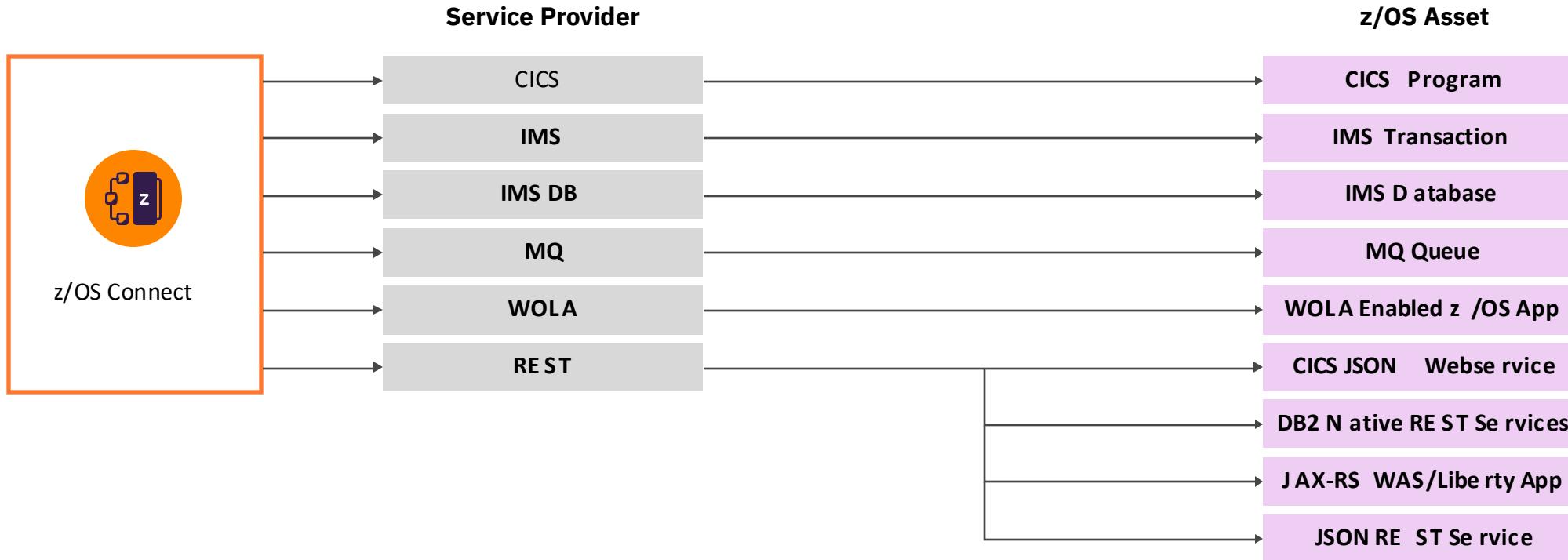
## /miscellaneousTopics

performance, high availability, Liberty



# What resources can z/OS Connect map to?

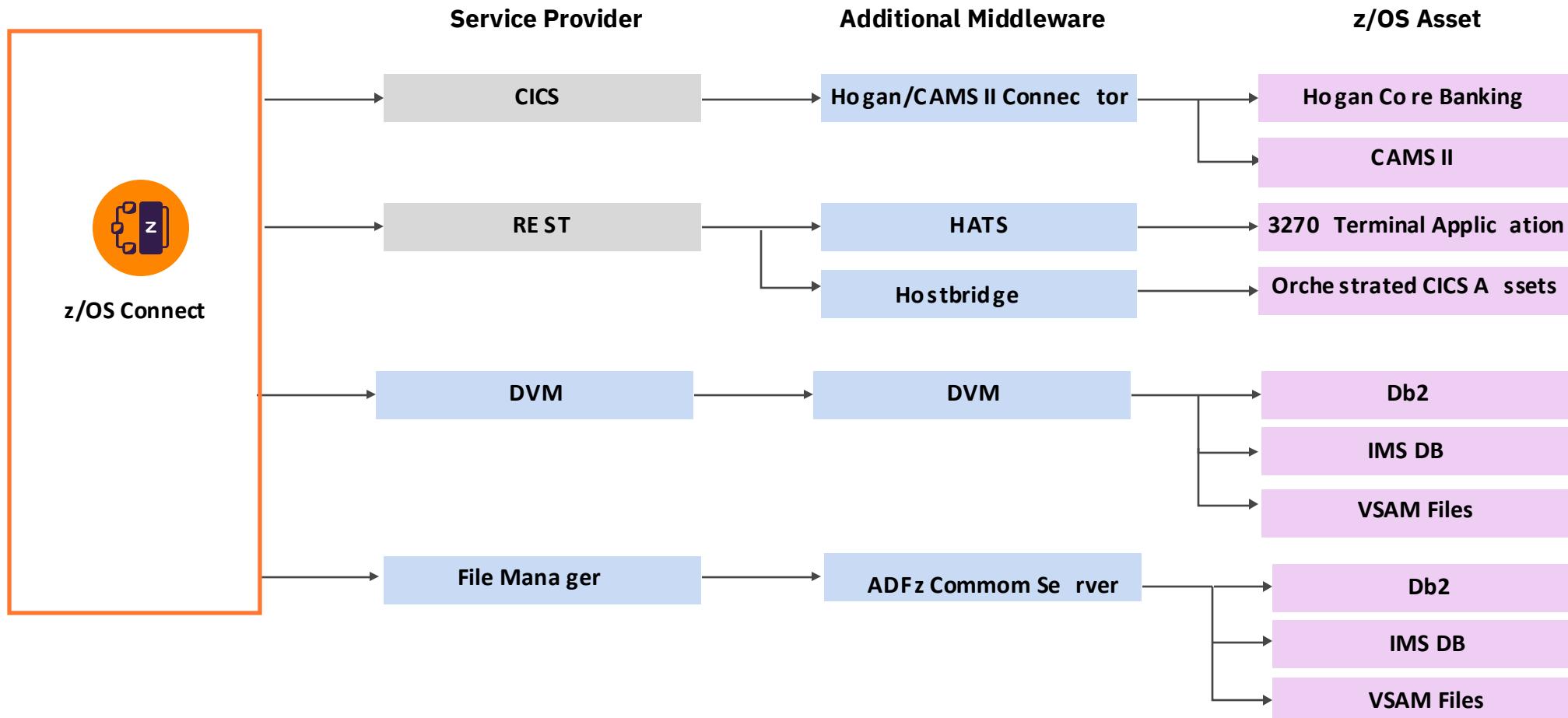
And which service provider could I use?



The core **service providers** included with z/OS Connect provide API access to a wide range of z/OS resources.

# Additional Middleware

Additional value from the ecosystem



z/OS Connect is **pluggable** and **extensible** allowing the use of additional middleware to expand the list of z/OS resources you can expose as APIs

# API Policies

- HTTP header properties can be used to select alternative for IMS (V3.0.4) , CICS (V3.0.10), Db2 (V3.0.36) or MQ (V3.0.39)
- Policies can be configured globally for every API in the server or for individual APIs (V3.0.11)

CICS attributes

- cicsCcsid
- cicsConnectionRef
- cicsTransId

IMS attributes

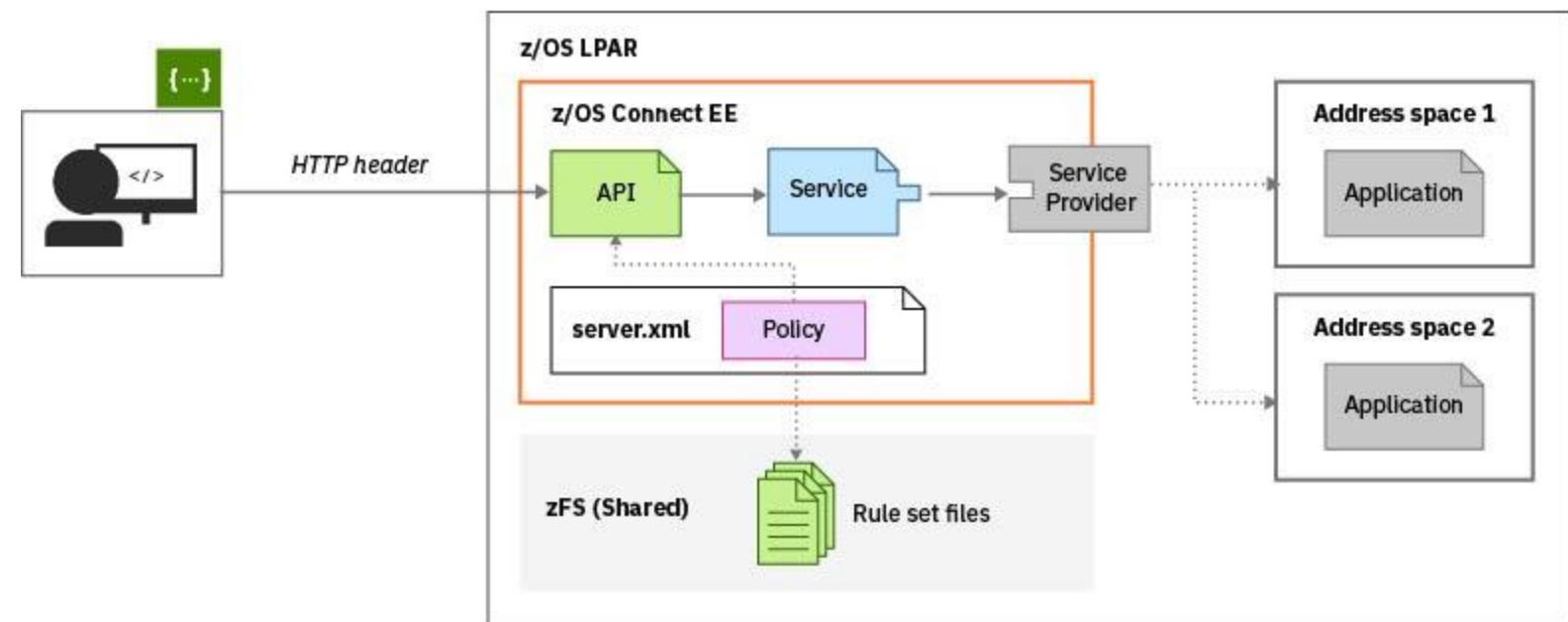
- imsConnectionRef
- imsInteractionRef
- imsInteractionTimeout
- imsITermOverrideName
- imsTranCode
- imsTranExpiration

Db2 attributes

- db2ConnectionRef
- db2CollectionID

MQ attributes

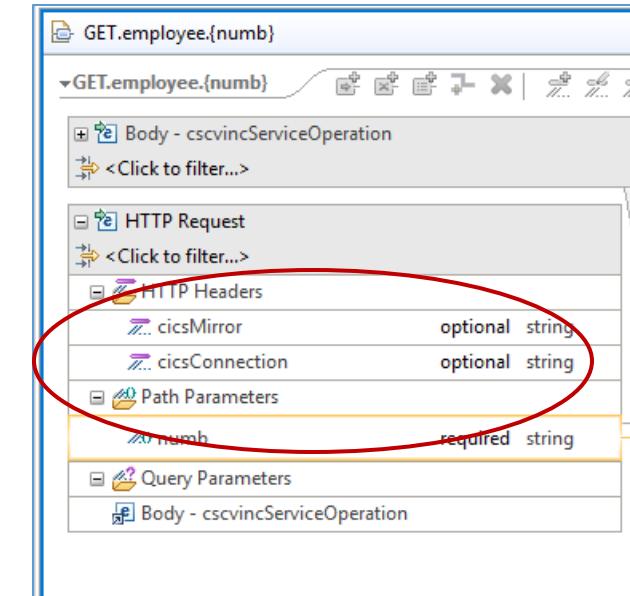
- mqConnectionFactory
- mqDestination
- mqReplyDestination





# A sample API Policies for CICS

```
<ruleset name="CICS rules">
  <rule name="csmi-rule">
    <conditions>
      <header name="cicsMirror" match="ANY_VALUE" /> *
    </conditions>
    <actions>
      <set property="cicsTransId" value="${cicsMirror}" />
    </actions>
  </rule>
  <rule name="connection-rule">
    <conditions>
      <header name="cicsConnection" value="" match="ANY_VALUE" />
    </conditions>
    <actions>
      <set property="cicsConnectionRef" value="${cicsConnection}" />
    </actions>
  </rule>
</ruleset>
```



## Curl

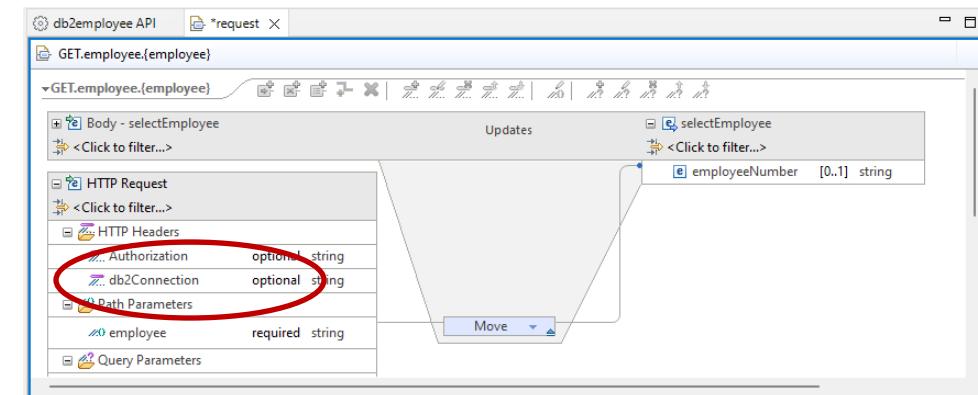
```
curl -X GET --header 'Accept: application/json' --header 'cicsMirror: MIJO' --header 'cicsConnection: cscvinc' 'https://m...
```

\*Transaction MIJO needs to be a clone of CSMI (e.g., invoke program DFHMIRS)



# A sample API Policies for Db2

```
<ruleset name="Db2 rules">
  <rule name="connection-rule">
    <conditions>
      <header name="db2Connection" value="" match="ANY_VALUE"/>
    </conditions>
    <actions>
      <set property="db2ConnectionRef" value="${db2Connection}"/>
    </actions>
  </rule>
</ruleset>
```



```
<zosconnect_zosConnectServiceRestClientConnection id="Db2Conn"
  sslCertsRef="db2SSLSettings" host="wg31.washington.ibm.com" port="2445" basicAuthRef="dsn2Auth" />
<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth" applName="DSN2APPL"/>
<ssl id="db2SSLSettings" keyStoreRef="Db2KeyStore" trustStoreRef="Db2KeyStore"/>

<zosconnect_zosConnectServiceRestClientConnection id="fred"
  sslCertsRef="fredSSLSettings" host="wg31.washington.ibm.com" port="2445" />
<ssl id="fredSSLSettings" keyStoreRef="Db2KeyStore" trustStoreRef="Db2KeyStore" clientKeyAlias="FRED"/>

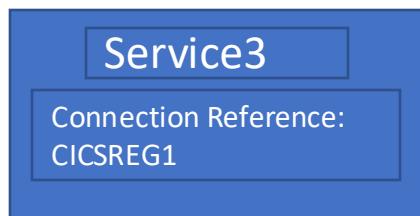
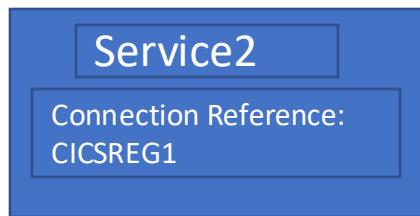
<zosconnect_zosConnectServiceRestClientConnection id="user1"
  sslCertsRef="user1SSLSettings" host="wg31.washington.ibm.com" port="2445" />
<ssl id="user1SSLSettings" keyStoreRef="Db2KeyStore" trustStoreRef="Db2KeyStore" clientKeyAlias="USER1"/>

<keyStore id="Db2KeyStore" location="safkeyring:///zCEE.Db2.KeyRing"
  password="password" type="JCERACFKS" fileBased="false" readOnly="true" />
```

Requires APAR PH53162

# Use naming conventions for service/endpoint connection references

Don't couple service and API requester connection names to specific systems or endpoints

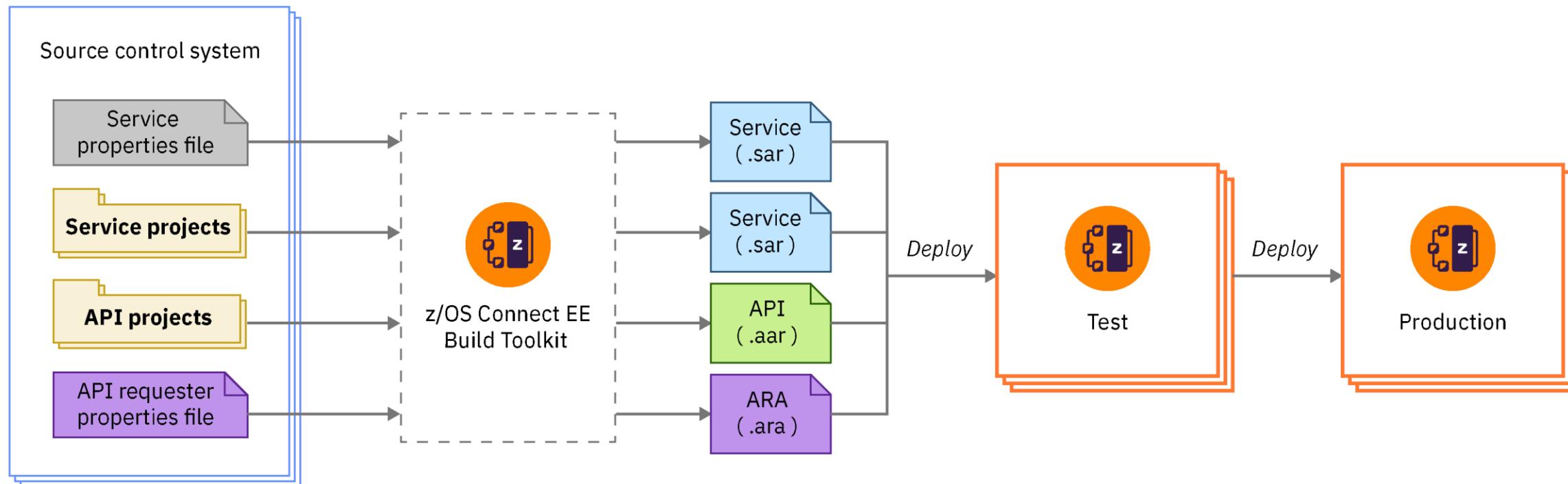


# DevOps using z/OS Connect RESTful Administrative APIs



Automate the development and deployment of services, APIs, and API requesters for continuous integration and delivery.

- The build toolkit supports the generation of service archives and API archives from projects created in the z/OS Connect API toolkit
- The build toolkit also supports the use of properties files to generate API requester archives
- Run the build toolkit from a build script to generate these archive files
- Deploy them to z/OS Connect servers by copying them to their deployment directories or by using the REST Admin API



# Agenda

- An Introduction to REST API
- Enabling RESTful API to z/OS resources, e.g.
  - CICS
  - Db2
  - IMS/TM
  - IMS/DB
  - MQ
- An overview of z/OS Connect API Provider Security
- Miscellaneous Topics

\*

# z/OS Connect Wildfire Github Site

<https://ibm.biz/zCEEWorkshopMaterial>



ibm-wsc/zCONNEE-Wildfire-Workshop

master 1 branch 0 tags

Go to file Add file Code

Commit	Message	Time
emitchj Delete xml directory	ea7ebdf 12 seconds ago	457 commits
APIRequesters	Add files via upload	14 hours ago
AdminSecurity	Add files via upload	6 days ago
COBOL Samples	Add files via upload	13 hours ago
OpenAPI2	Add files via upload	2 months ago
OpenAPI3	Add files via upload	4 days ago
XML Samples	Add files via upload	1 minute ago
README.md	Update README.md	13 months ago
ZCADMIN - zOS Connect ...	Add files via upload	2 months ago
ZCEESEC - zOS Connect Se...	Add files via upload	3 months ago
ZCINTRO - Introduction to...	Add files via upload	13 hours ago
ZCREQUEST - Introduction...	Add files via upload	3 months ago
zOS Connect EE V3 Advan...	Add files via upload	12 months ago
zOS Connect EE V3 Gettin...	Add files via upload	14 months ago

zCONNEE-Wildfire-Workshop

master zCONNEE-Wildfire-Workshop / OpenAPI2 /

emitchj Add files via upload

Developing CICS API Requester Applications.pdf Add files via upload

Developing IMS API Requester Applications.pdf Add files via upload

Developing MVS Batch API Requester Application... Add files via upload

Developing RESTful APIs for Db2 DVM Services... Add files via upload

Developing RESTful APIs for Db2 REST Services.... Add files via upload

Developing RESTful APIs for HATS REST Service... Add files via upload

Developing RESTful APIs for IMS DVM Services... Add files via upload

Developing RESTful APIs for IMS Database RES... Add files via upload

Developing RESTful APIs for IMS Transactions.p... Add files via upload

master zCONNEE-Wildfire-Workshop / OpenAPI3 /

emitchj Delete admin

Developing RESTful APIs for Db2 REST Services.... Add files via upload

Developing RESTful APIs for a CICS Program.pdf Add files via upload



The WSC Requests your Feedback!

Please scan the QR Code or go to this link:  
<https://tinyurl.com/bdey6tmv>

For Event Code, Enter: **ZOSCON** + the current date  
**(DDMMYY)**



Thank you for listening and your questions