



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

Mitch Johnson mitchj@us.ibm.com

John Brefach John.J.Brefach@ibm.com

Washington System Center



Agenda

- An Introduction and Overview of using REST API
- Enabling RESTful API to z/OS resources, e.g.
 - CICS
 - Db2
 - IMS/TM
 - IMS/DB
 - MQ
- A brief overview of z/OS Connect API Security

*For more on administration and security, contact your local IBM rep regarding the schedule of workshop *zCADMIN IBM z/OS Connect Administration Wildfire Workshop*

Notes and Disclaimers



- The information in this presentation was derived from various product documentation web sites.
- Additional information included in this presentation was distilled from years of experience implementing security using RACF with z/OS products like CICS, IMS, Db2, MQ, etc. as well as Java runtimes environments like WebSphere Application Server and WebSphere Application Server Liberty which is commonly called Liberty.
- There will be additional information on slides that will be designated as Tech/Tips. These contain information that at perhaps at least interesting and hopefully, useful to the reader.
- **IBM z/OS Connect (OpenAPI 2)** refers to the z/OS Connect EE product prior to service level V3.0.55. **IBM z/OS Connect (OpenAPI 3)** refers to the additional functions and features added with 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.
- The examples, tips, etc. present in this material are based on firsthand experiences and are not necessarily sanctioned by Liberty or z/OS Connect development.

This session is part of a series of z/OS Connect related workshops. . .



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

Mitch Johnson mitchj@us.ibm.com

John Brefach John.J.Brefach@ibm.com

Washington System Center

mitchj@us.ibm.com



z/OS Connect OpenAPI 3

Designer and z/OS Native server
Experiences and Observations

Mitch Johnson
mitchj@us.ibm.com
Washington Systems Center



mitchj@us.ibm.com

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

© 2017, 2023 IBM Corporation
Slide 4



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



© 2017, 2023 IBM Corporation
Slide 1

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

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

ZADMIN - 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 / 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.pdf Add files via upload

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

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

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

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

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

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

zCONNEE-Wildfire-Workshop / OpenAPI3 /

emitchj Delete admin

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

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

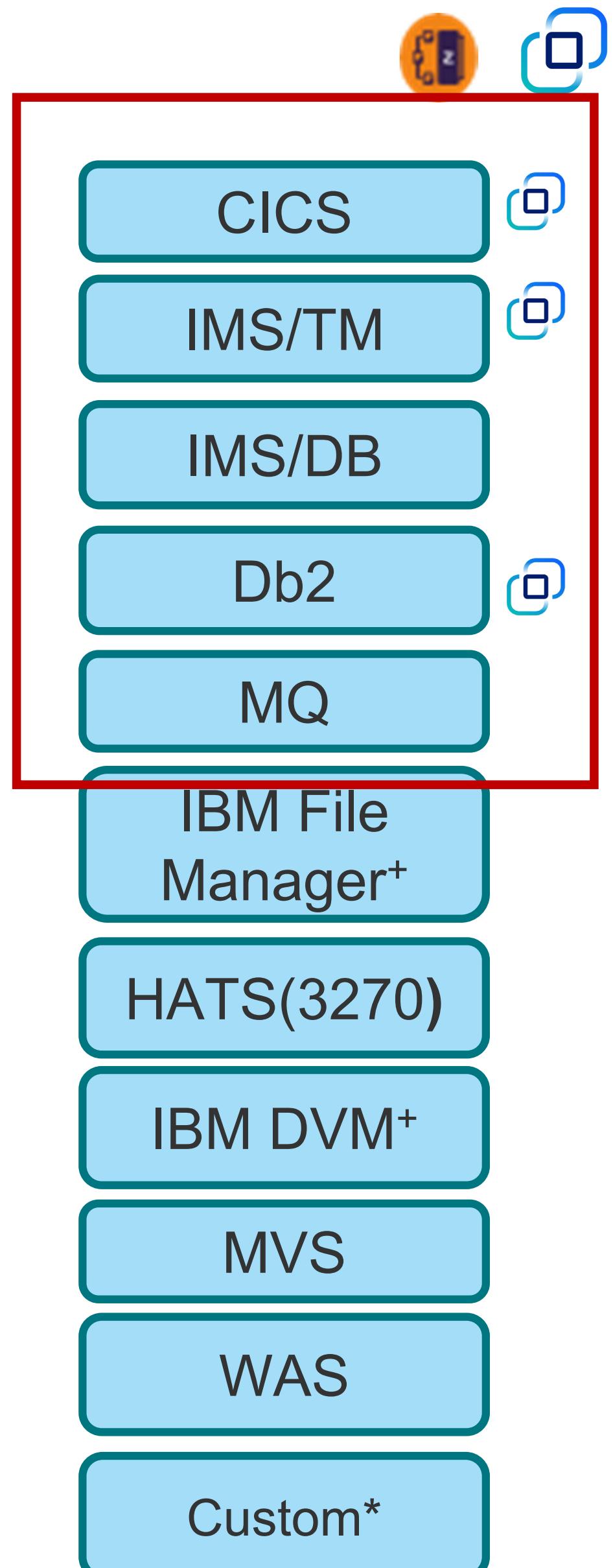
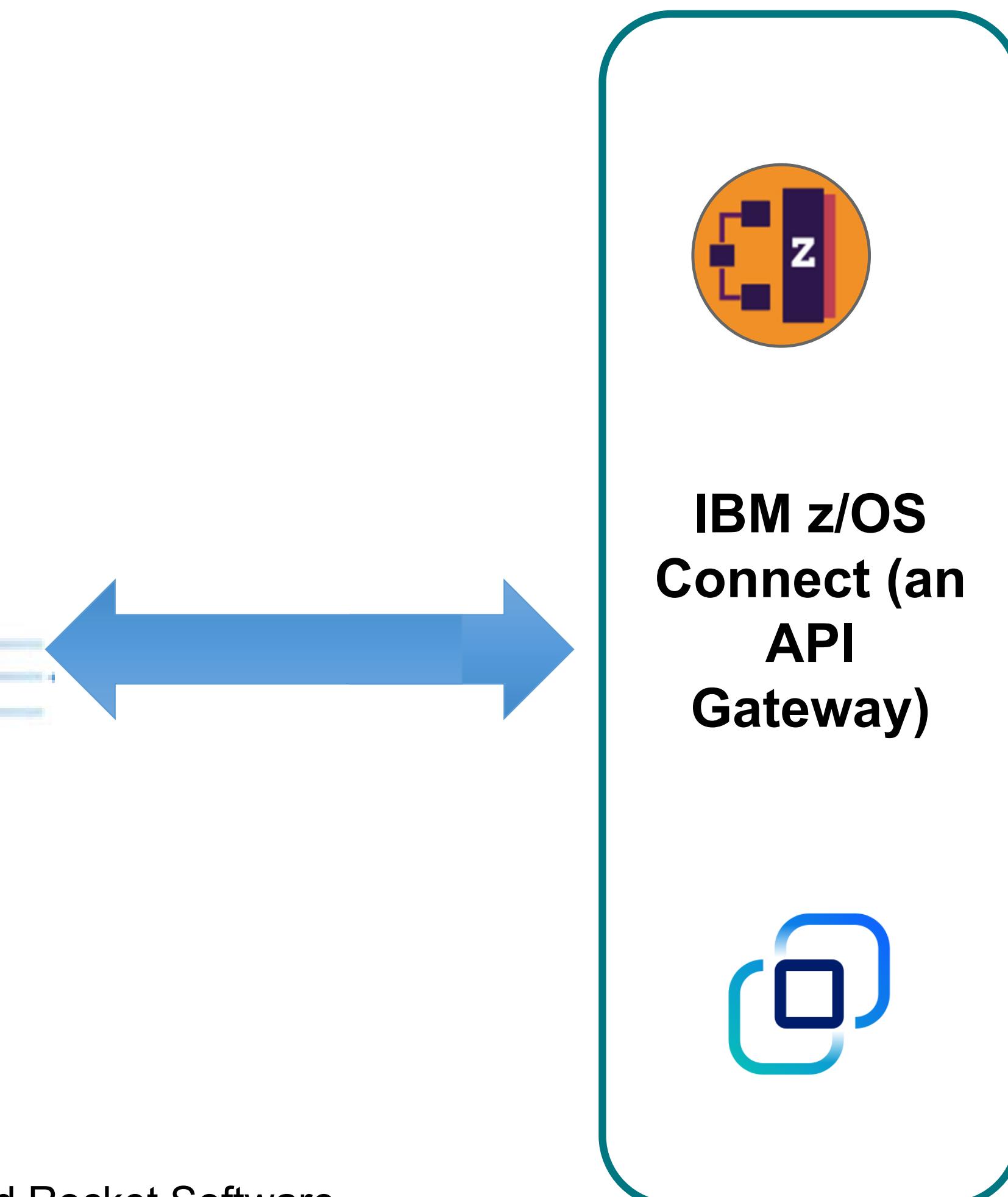
z/OS Connect EE exposes z/OS resources to clients in the “cloud” using RESTful APIs



mitchj@us.ibm.com

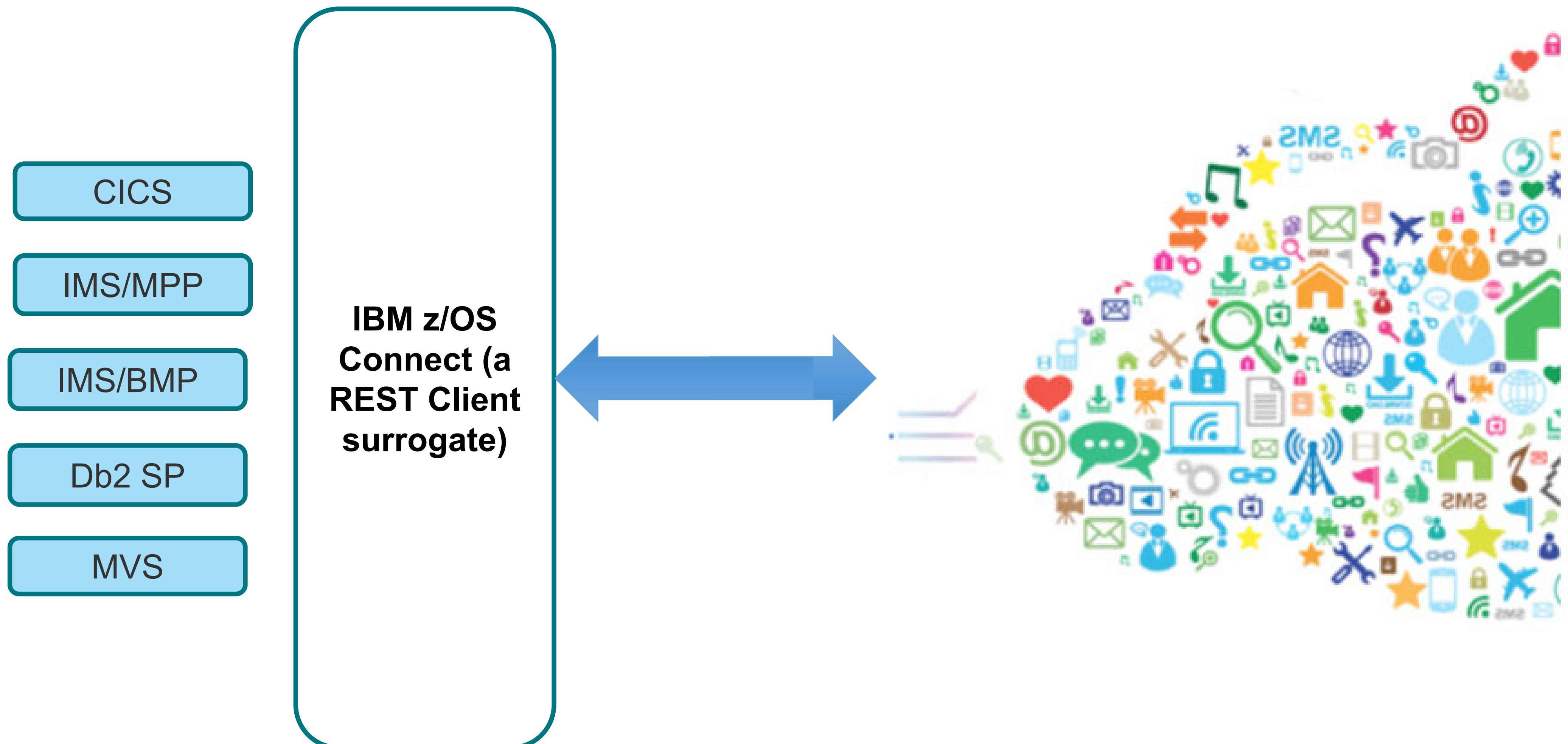
+ HCL and Rocket Software

*Other Vendors or your own implementation





Note that z/OS Connect also exposes external REST APIs
in the “cloud” to z/OS applications



Let's start by defining REST

/what_is_REST?

And what makes an API “RESTful”?



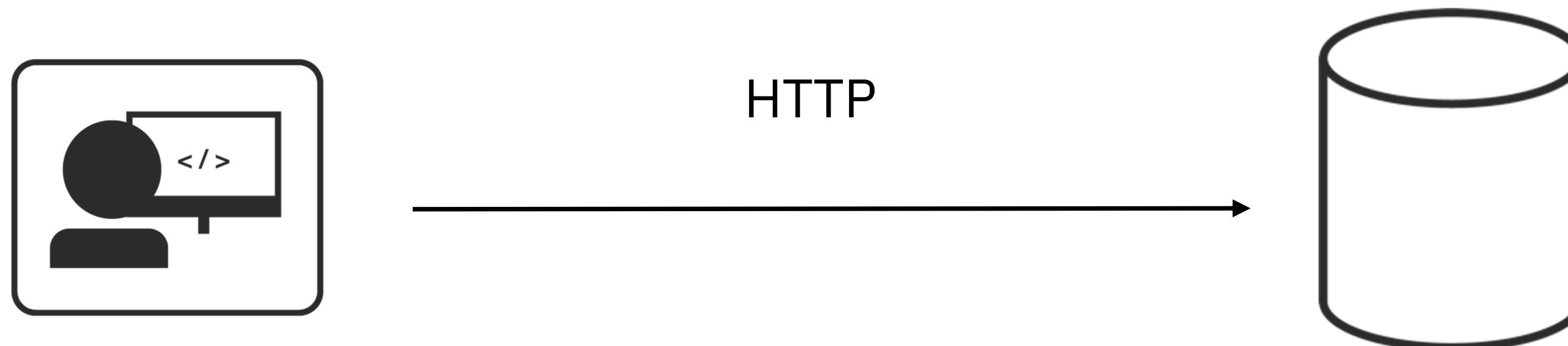
REST is architectural programming style

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”.

Simple and intuitive 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.



Key Principles of the REST API

Uses HTTP verbs to specify Create, Read, Update, Delete (CRUD) operations, for background on CURD, see URL https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

POST
GET
PUT
DELETE

Uniform Resource Identifier (URI) 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" }

Request/Response Body is used to represent the data object



REST vs RESTful

REST is an architectural style of development having these 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
- For performing CRUD operations, it should use HTTP verbs such as get, post, put and delete
- It should return the result only in the form of consistent and simple JSON

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

- Remember - Not all REST APIs are RESTful APIs
- The key is consistency, RESTful APIs are consistent with these basic principles, REST APIs are not



Why is REST 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.



RESTful Examples

POST /account/ + (a JSON request message with Fred's information)

GET /account?number=1234

PUT /account/1234 + (a JSON request message with dollar amount of deposit)

HTTP Verb conveys the method against the resources; i.e., POST is for create, GET is for balance, etc.

URI conveys the resource to be acted upon; i.e., Fred's account with number 1234

The JSON body carries the specific data for the action (verb) against the resource (URI)

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>



Not every REST API is a RESTful API

(How to know if an API is not RESTful)

1. Different URIs with the same method for operations on the same object

POST http://www.acme.com/customers/**GetCustomerDetails**/12345

POST http://www.acme.com/customers/**UpdateCustomerAddress**/12345?**address=**

2. Different representations of the same objects between request and response messages

POST http://www.acme.com/customers
BODY { ***/89:N\$3**": "J5)",
 "**2\$9:N\$3**" : "B25++9",
 "**\$((8**" : "^(O2(S:8)):",
 "**6.54)N5**" : "[^_`a [^_`abc" }

RESPONSE HTTP 201 CREATED
BODY { "id" : "12345",
 "**4\$3**" : "**J5) B25++9**",
 "**\$((8)99**" : "^(N)= S:8)):"
 ":) 2" : "[^_`a [^_`abcx" }

3. Operational data (update, etc.) embedded in the request body

POST http://www.acme.com/customers/12345
BODY { "**;6(\$:)F/2()**" : "**\$((8)99**",
 "**4)=V\$2;)**" : "^(N)= S:8)):" }

RESPONSE HTTP 200 OK
BODY { "id" : "12345",
 "name" : "Joe Bloggs",
 "address": "^(N)= S:8)):"
 "tel" : "01234 123456" }

One goal is to try to simplify the development of client applications



1. Use the same URIs for the same resource with the appropriate method for operations

GET http://www.acme.com/customers/12345

PUT http://www.acme.com/customers/12345?address=10%20New%20Street

2. Use the same JSON property names between request and response messages

POST http://www.acme.com/customers/12345

BODY { "4\$3)": "J5) B25++9",
 "\$((8)99)": "^ [02(S:8)):",
 "6.54)N5": "[^_`a [^_`abc" }



RESPONSE HTTP 201

BODY { "id" : "12345",
 "4\$3)": "J5) B25++9",
 "\$((8)99)": "^ [N)= S:8)):",
 "6.54)N5": "[^_`a [^_`abcx" }

3. Use JSON name/value pairs

PUT http://www.acme.com/customers/12345

BODY { "\$((8)99)": "^ [N)= S:8)):"}



RESPONSE HTTP 200 OK



Non-REST client code is usually tightly coupled with the resource being accessed

CICS client

```
*cscinq.java
24 COMMAREABUFFER commarea = new COMMAREABUFFER();
25 commarea.setNumb("111111");
26
27 //CICSConnection connection = new CICSConnection();
28 com.ibm.connector2.cics.ECIIInteractionSpec interSpec = new com.ibm.connector2.cics.ECIIInteractionSpec();
29
30 //      Create an interaction spec
31 interSpec.setFunctionName("CSCVINQ");
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.ECIIInteraction interaction =
57         (com.ibm.connector2.cics.ECIIInteraction) connection.createInteraction();
58
59     //      Execute the interaction
60     interaction.execute(interSpec, commarea, commarea);
61 }
```

MQ client

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



One goal for REST simplicity is to have client code that is unaware of the infrastructure

```
ZceeCICSGet.java
55
56
57
58
59
60
61
62
63
64
65
66
// 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());
    }
}
```

```
ZceeDb2Get.java
52
53
54
55
56
57
58
59
60
61
62
// 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());
    }
}
```

These Java clients have the same application programming pattern.

- They provide an URL and a method and provide a request message.
- Then 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.

```
URL: https://wg31.washington.ibm.com:9453/mqapi/
NAME: TINA J YOUNG
NUMB: 001781
ADDRX: SINDELFINGEN, GERMANY
PHONE: 70319990
DATEX: 21 06 77
AMOUNT: $0009.99
```

MQ

```
lastName: LAST1
firstName: FIRST1
zipCode: D01/R01
extension: 8-111-1111
message: ENTRY WAS DISPLAYED
HTTP code: 200
```

IMS



/oai/open_api_initiative

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.

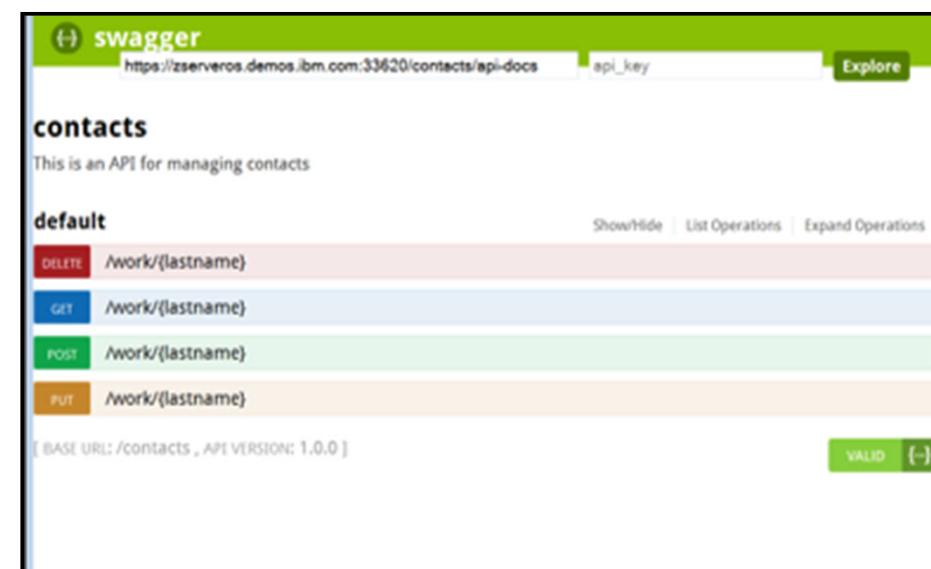


There are a number of tools available to aid in the creation and consumption of APIs

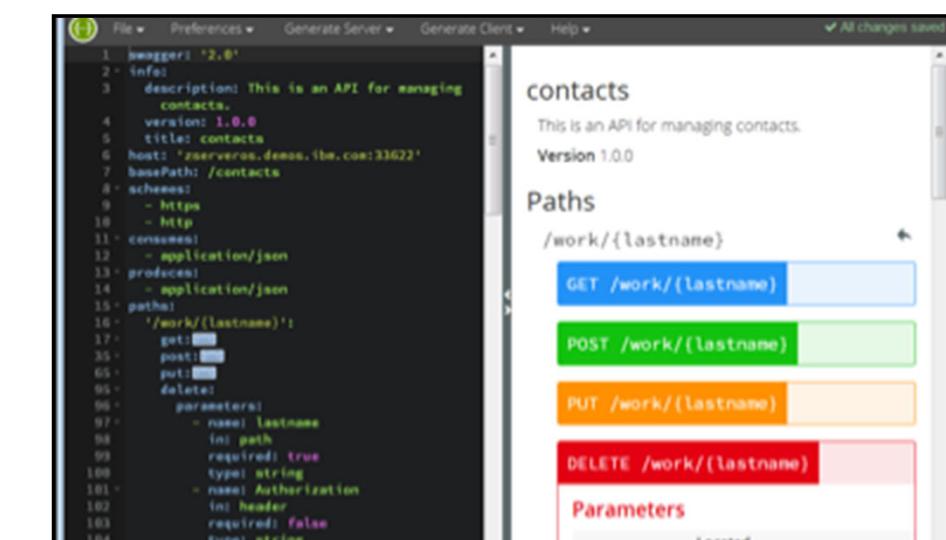
Code Generation* - create stub code to consume APIs from various languages



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



Editors - allows API developers to design their OpenAPI documents.



* z/OS Connect API Requester

+z/OS Connect, MQ REST support, Zowe

Important - You may have used or heard of the term Swagger with the use of APIs. As the use of APIs has grown this term has become in some respects misleading. To be more precise, OpenAPI refers to the API specifications (OpenAPI 2 and OpenAPI3) where Swagger refers to the tooling used to implement the specifications.



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

The industry standard framework for describing REST APIs



- **z/OS Connect and Swagger 2.0 (Open API Specification 2), 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**

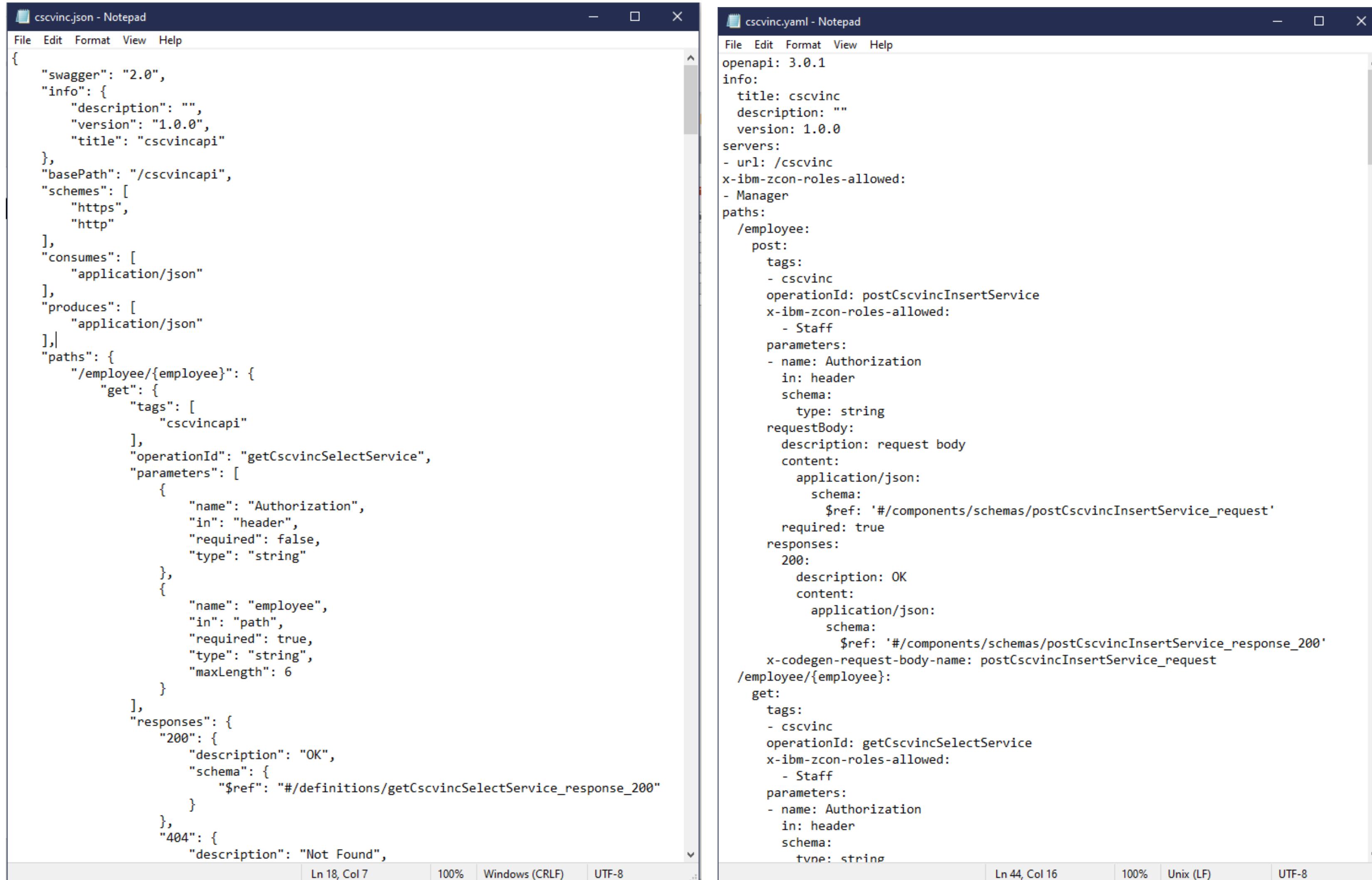
As companies mature their API strategy, they begin to introduce API governance boards to drive consistency in their API design. As more public APIs are created, government and industry standards bodies begin to regulate and drive for standardization. This drives the need for “API first” functional mapping 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**

Contrast the OpenAPI 2/OpenAPI 3 specification differences



z/OS Connect
OpenAPI2 tooling
produces an
OpenAPI 2
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).



The image shows two side-by-side Notepad windows. The left window, titled 'cscvinc.json - Notepad', contains the JSON representation of an OpenAPI 2 specification. It includes sections for swagger version (2.0), info (title: 'cscvincapi'), basePath ('/cscvincapi'), schemes ('https', 'http'), consumes ('application/json'), produces ('application/json'), and paths. The paths section details a 'get' method for '/employee/{employee}' with parameters for Authorization (header, required: false, type: string) and employee (path, required: true, type: string, maxLength: 6). It also defines responses for 200 and 404. The right window, titled 'cscvinc.yaml - Notepad', contains the YAML representation of an OpenAPI 3 specification. It includes openapi (3.0.1), info (title: 'cscvinc', description: '', version: 1.0.0), servers (url: '/cscvinc'), and paths. The paths section details a 'post' method for '/employee' with parameters for Authorization (header, type: string) and request body (application/json, schema: \$ref: '#/components/schemas/postCscvincInsertService_request', required: true). It also defines responses for 200 and 404.

z/OS Connect
OpenAPI3 tooling
consumes an
OpenAPI3 specification
document 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 asset. Also,
JSONata can be
used to augment
the API response



What value does zOS Connect Add?

Truly RESTful APIs to and from your mainframe.

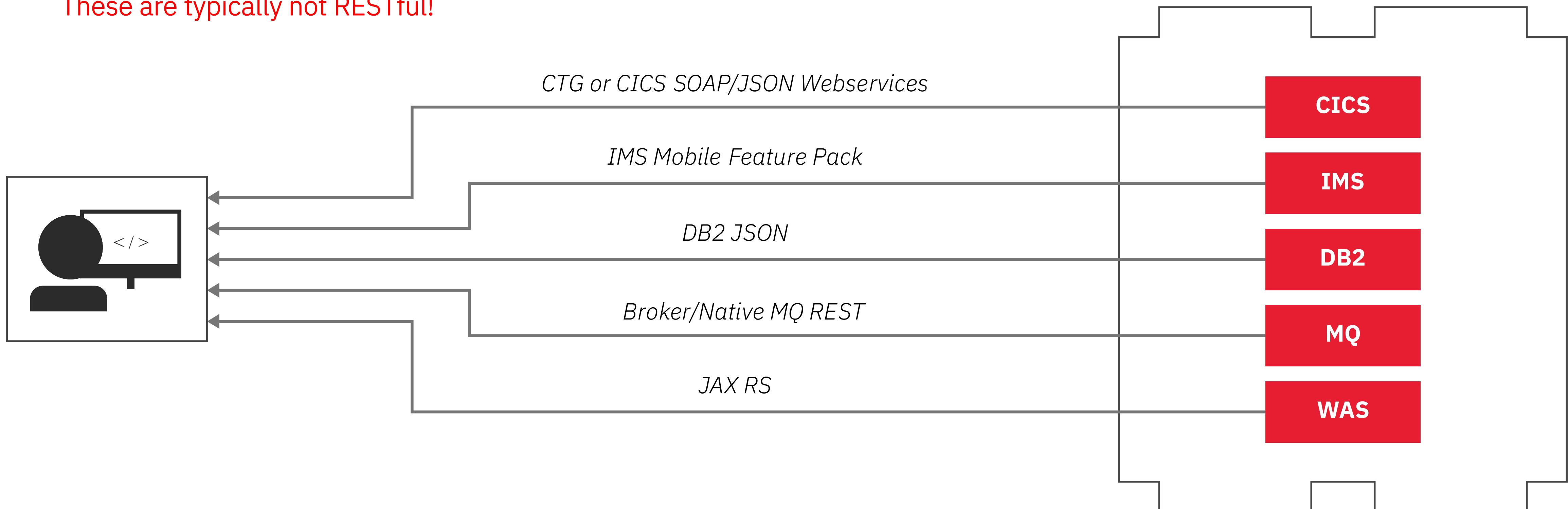


There was support for REST before z/OS Connect but..

Completely different configuration and management.

Multiple endpoints for developers to call/maintain access to.

These are typically not RESTful!





z/OS Connect provides a single API gateway entry point

- And exposes z/OS resources without writing or changing any code.

z/OS Connect EE provides

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



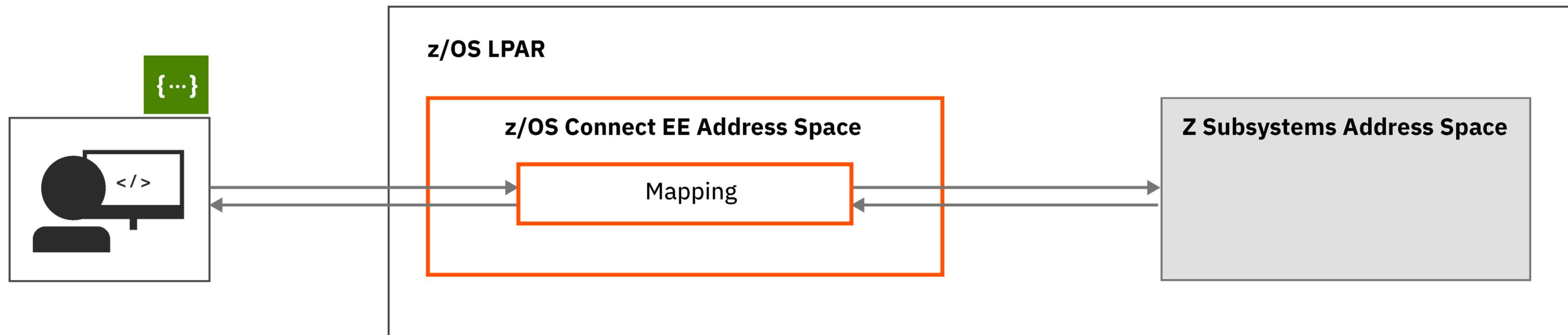


**Other than a RESTful interface,
what else does z/OS Connect provide?**



Data mapping/transformation

- Converts the JSON message to the format the target's subsystem expects*.

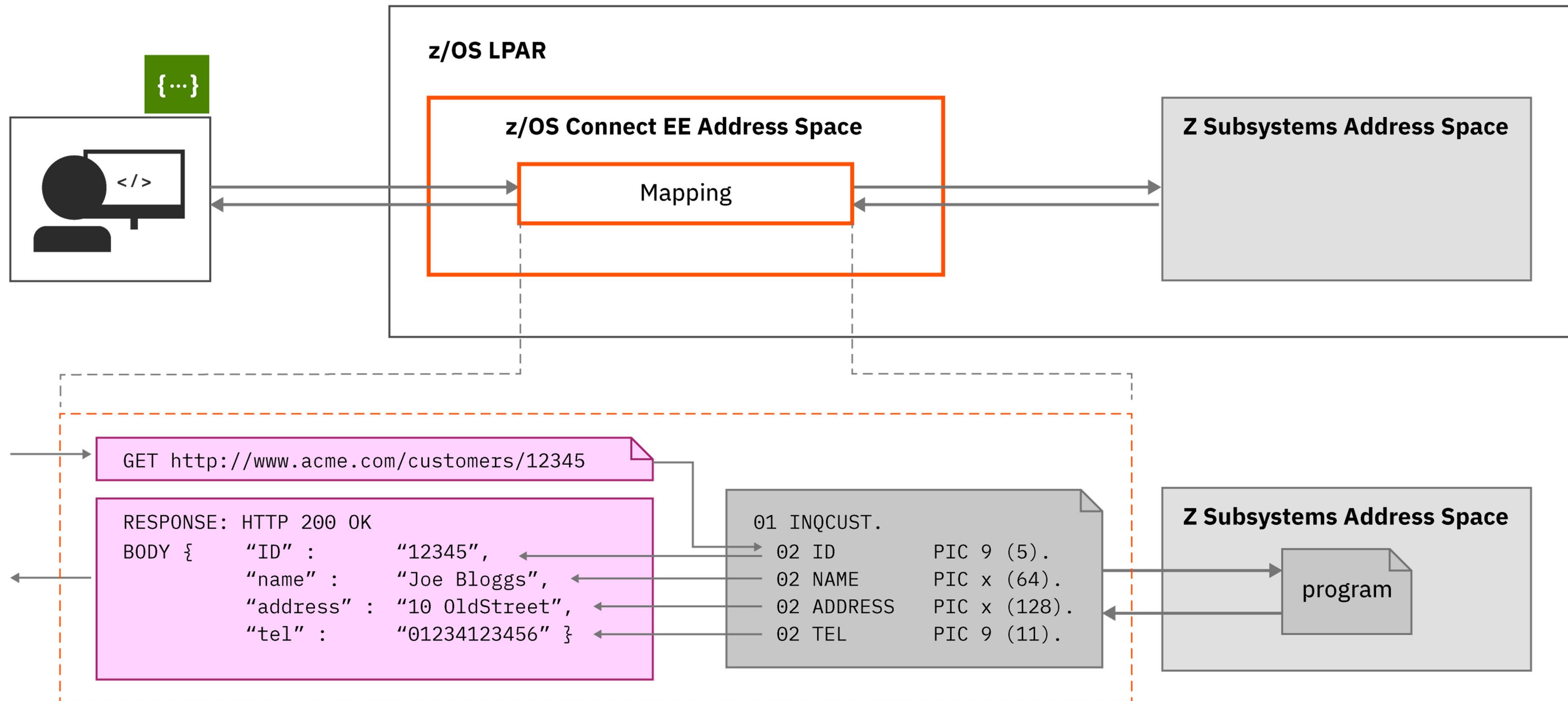


* Most z/OS subsystems depend on information in a serial data format and do not normally work with JSON request/response messages. Examples of serialized messages are CICS COMMAREAAs and CONTAINERS, IMS or MQ messages, or records stored in sequential or VSAM data sets. Data mapping and transformation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



A data mapping and transformation example

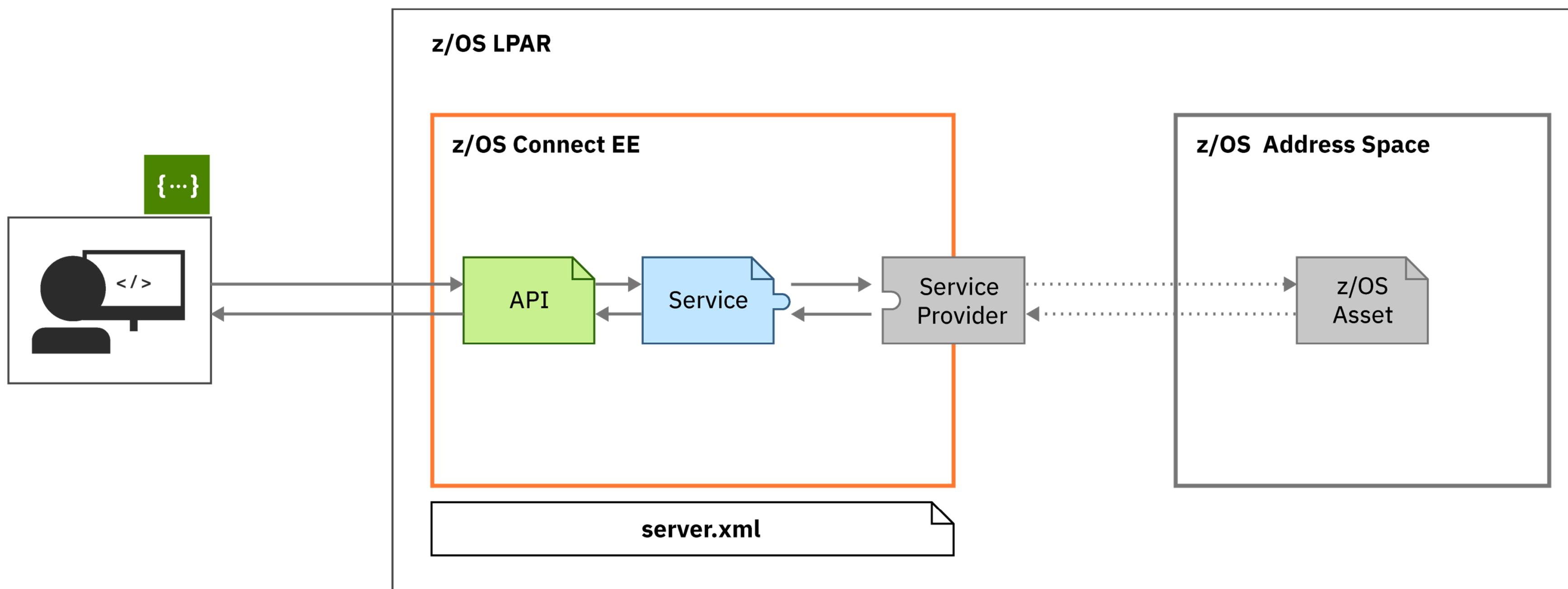
- A closer look at the transformation process





Accessing an z/OS Asset using an API with Open API 2

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

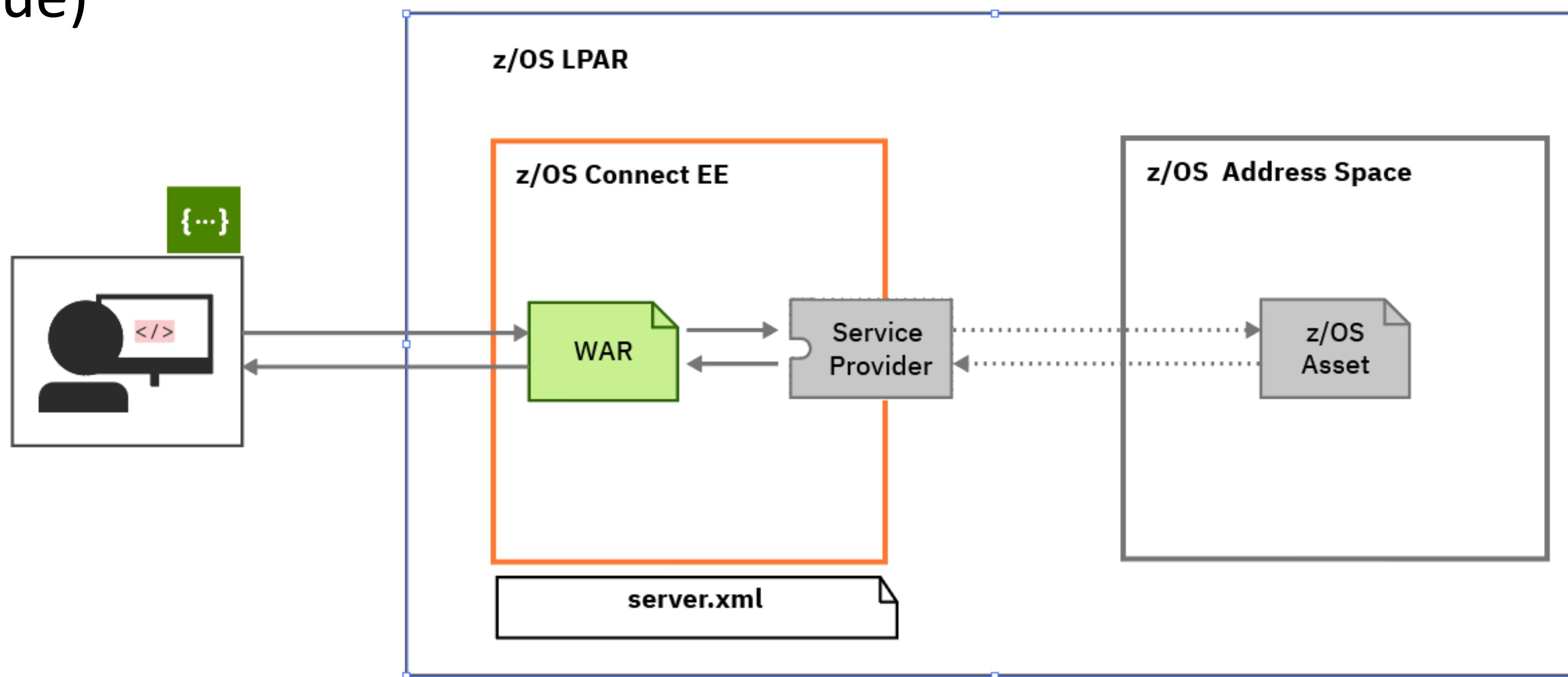


- The API is ready to be consumed and requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS Asset (e.g., CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific instance of a resource (e.g., host and port,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



Accessing a z/PS asset using an API with Open API 3

- z/OS Connect OpenAPI 3 APIs are developed using a z/OS Connect Designer web browser tool to develop 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)

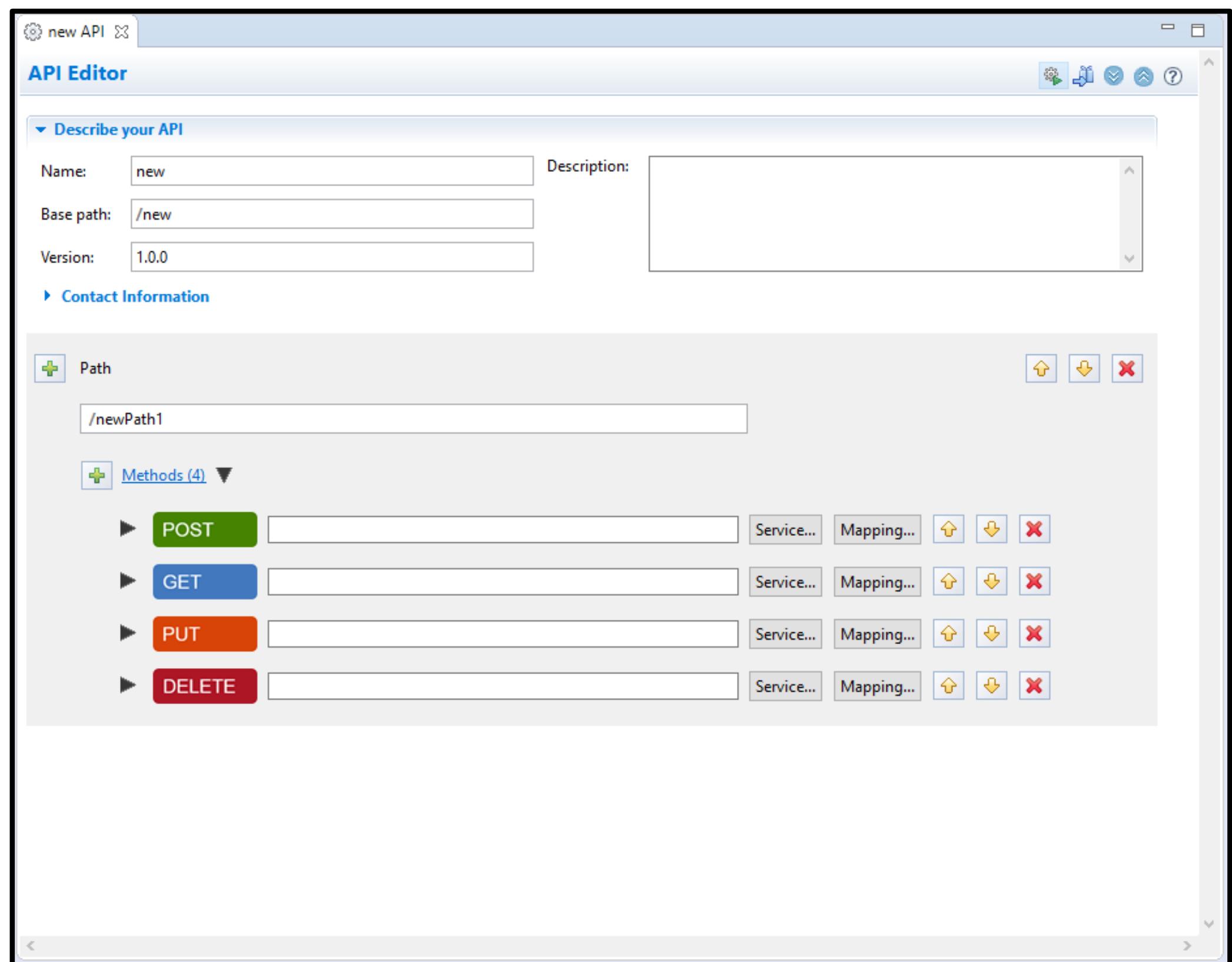


z/OS Connect OpenAPI Tooling

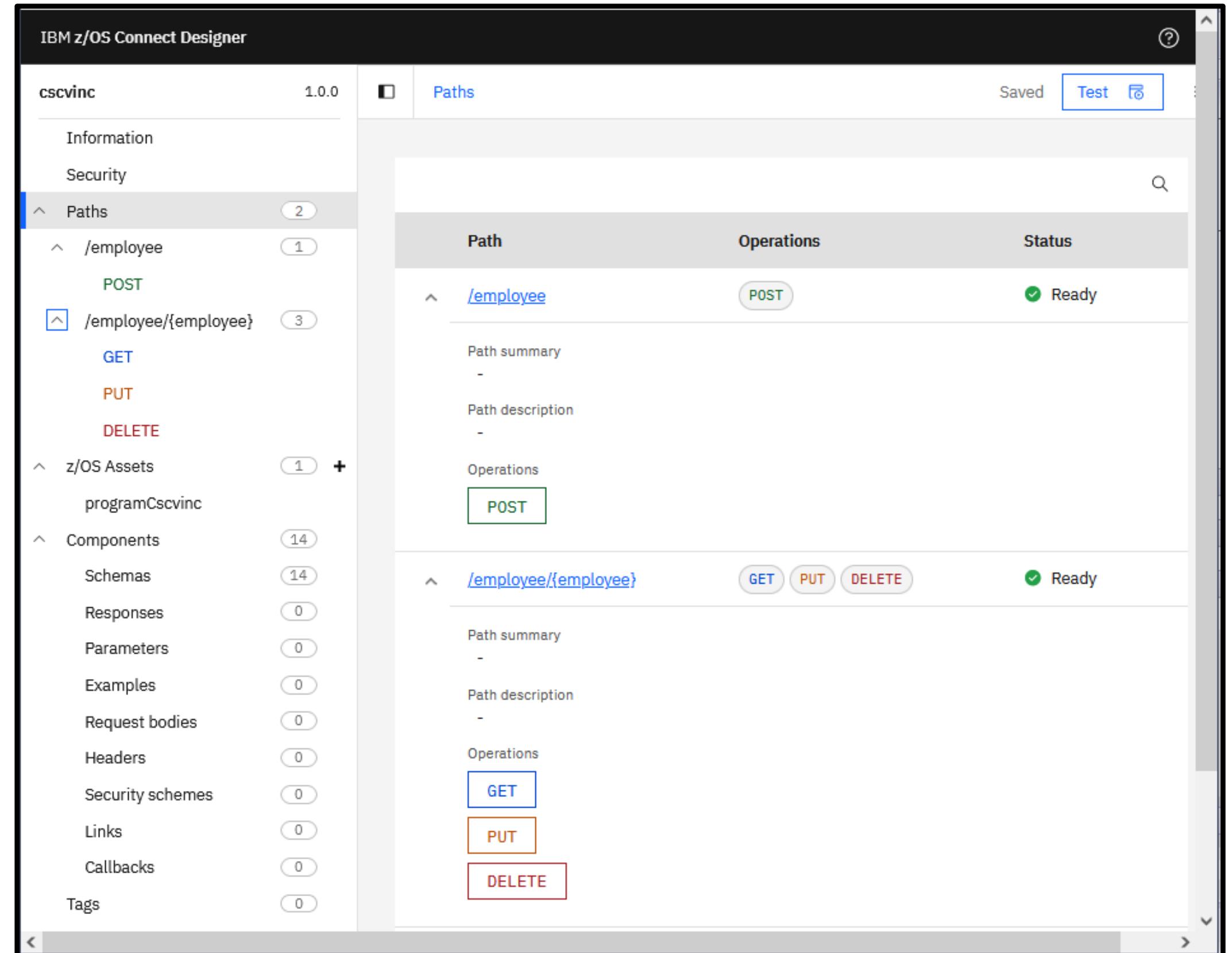
z/OS Connect's OpenAPI 2 Palette versus the OpenAPI 3 API Designer



z/OS Connect API Toolkit (Eclipse)



z/OS Connect Designer (Designer Container)



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

The API specification provides predefined URI Paths and methods.



An OpenAPI 2 versus an OpenAPI 3 Response Message

The screenshot shows the Eclipse-based z/OS Connect API Toolkit interface. On the left, there is a tree view of API definitions, with a red circle highlighting the 'Body - cscvincSelectServiceOperationResponse' node under 'GET.employee.{employee}'. On the right, the 'IBM z/OS Connect Designer' window displays the API definition. It shows an 'Employee' resource with a 'GET' method. The 'Paths' section lists '/employee' and '/employee/{employee}' with counts of 1 and 3 respectively. The 'Responses' section shows a '200 - OK' status with a detailed JSON schema. A red circle highlights the JSON schema for the 'response' field, which includes fields like 'employeeNumber', 'name', 'address', 'phoneNumber', 'date', 'amount', and 'comment'. The JSON schema is defined as:

```
response:
  type: object
  properties:
    employeeDetails:
      type: object
      properties:
        employeeNumber:
          type: string
        name:
          type: string
        address:
          type: string
        phoneNumber:
          type: string
        date:
          type: string
        amount:
          type: string
        comment:
          type: string
```

Eclipse based - z/OS Connect API Toolkit -
The contents of the API's response message are
directly derived from the z/OS asset's response

Web browser - z/OS Connect Designer –
The contents of the API's response
message are derived from the z/O asset's
response and JSONata coding, see URL
<https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=concepts-what-is-jsonata> for more information on
JSONata.



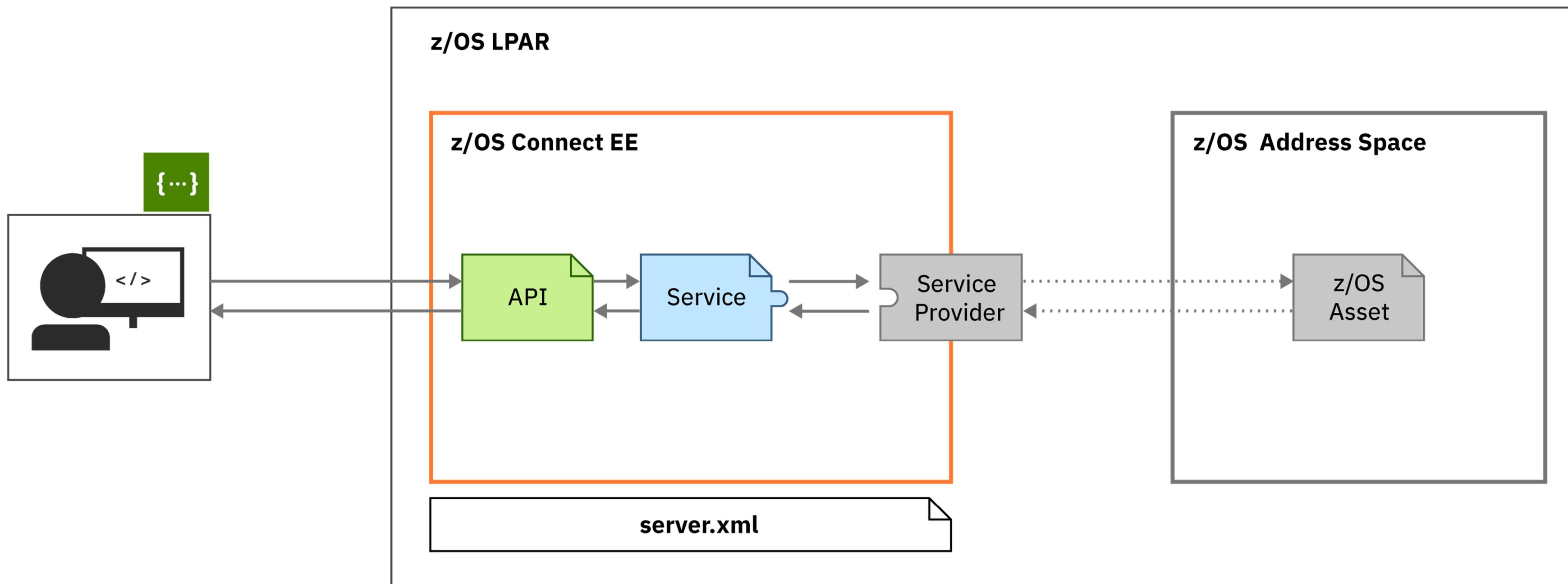
OpenAPI 2

Creating **Services** using the Eclipse Toolkit



Accessing a z/OS asset (Open API 2)

z/OS Connect OpenAPI 2 does not use a single monolithic interface
– but rather 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 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)

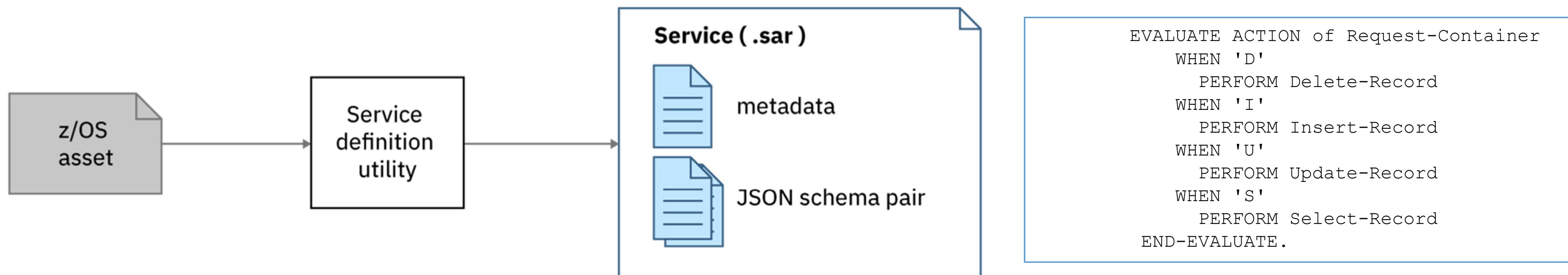
Describing the interaction (service) with the z/OS resource (OpenAPI 2)



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

To start mapping an API, z/OS Connect EE 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



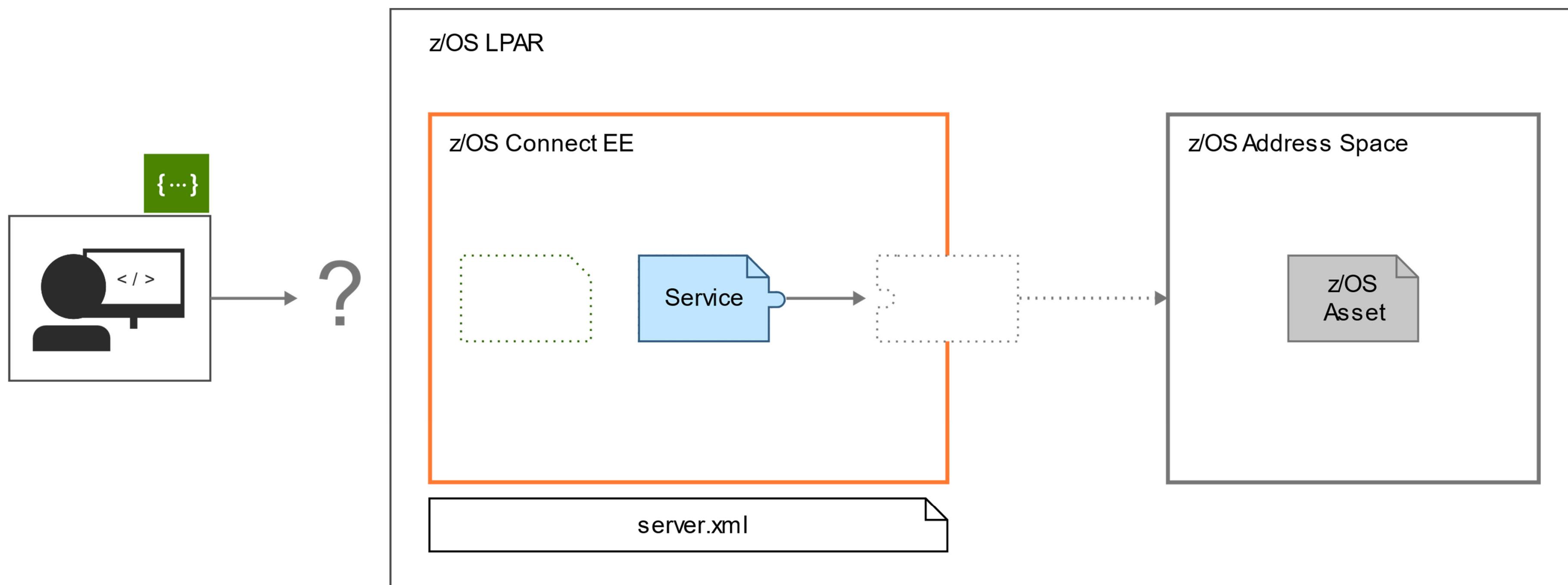
Use a system-appropriate utility to generate a service archive file for the z/OS application

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



Deploy and export the service archive (OpenAPI 2)

Deploy and export the service archive file

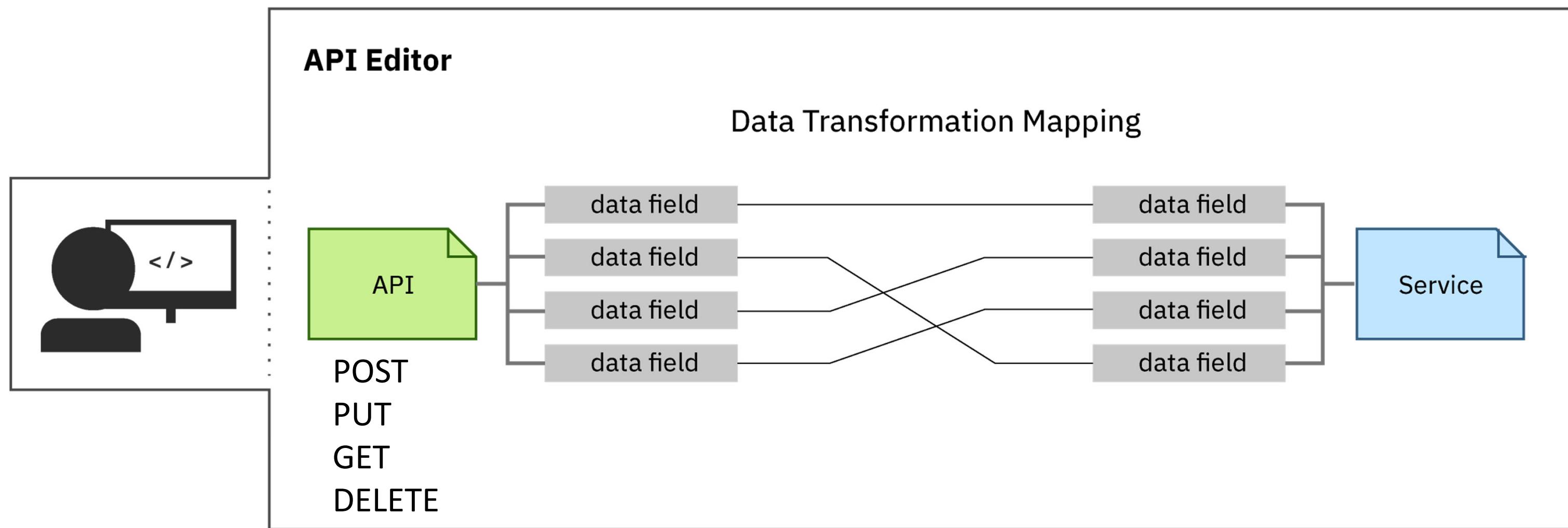


Deploy the service archive file generated in **Step 1** using the right-click deploy in **the API toolkit**.

Develop an API using the service interactions (Open API 2)



Export the service and then import it to create an API that consumes the service

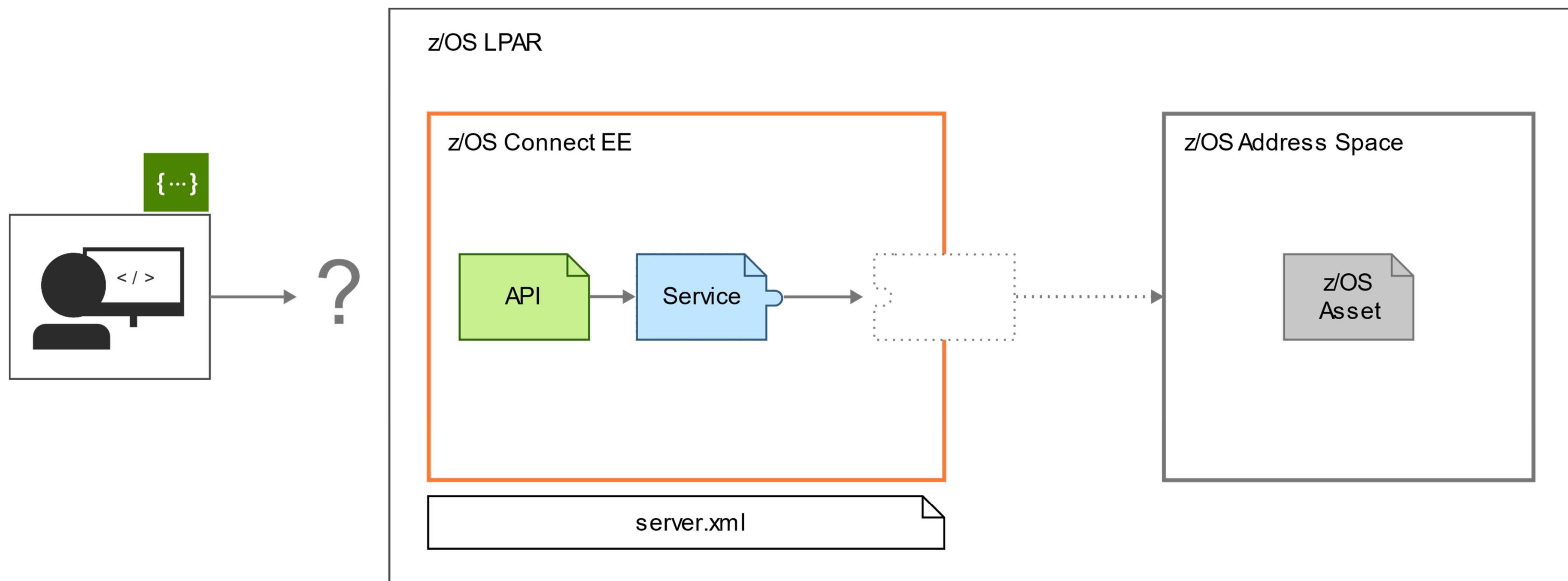


- Import the service archive file into the **API toolkit** and start designing the RESTful API.
- Provides additional data mapping
- Use the editor to describe the API and how it maps to underlying services.



Deploy the API (Open API 2)

Deploy the API archive file

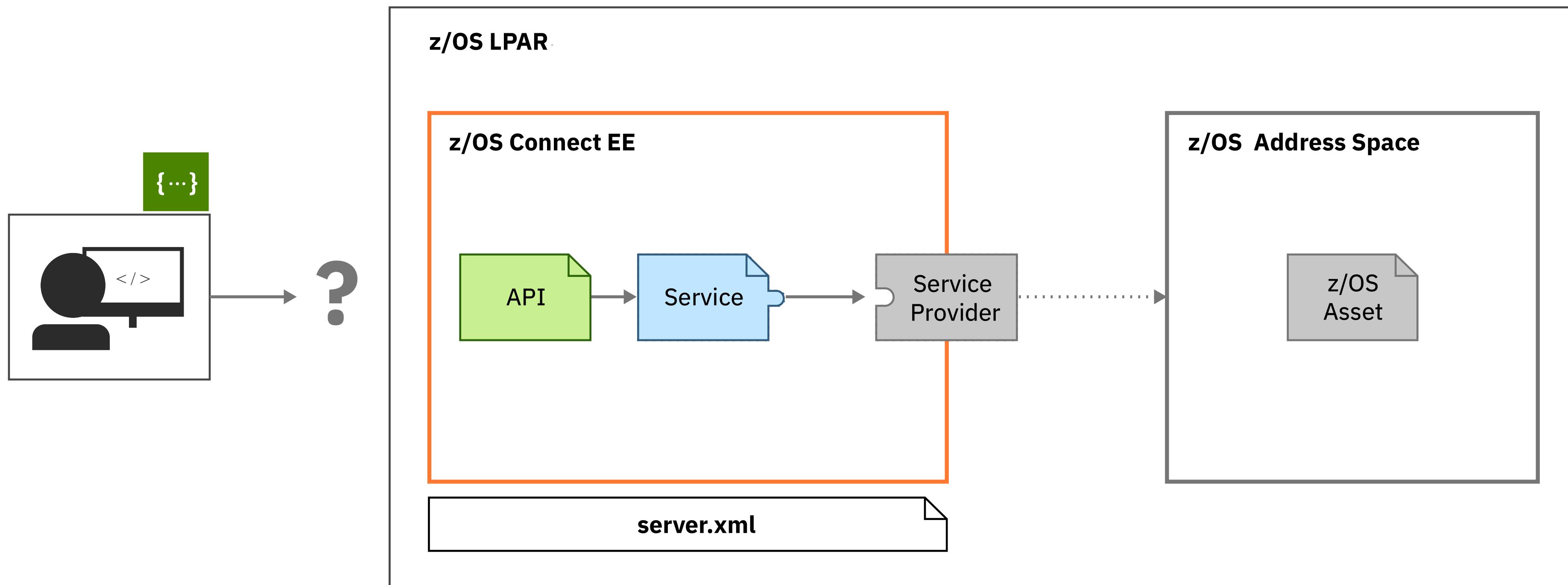


Deploy your API using the right-click deploy in **the API toolkit**



Configure access the z/OS sub system (Open API 2)

Configure the service provider

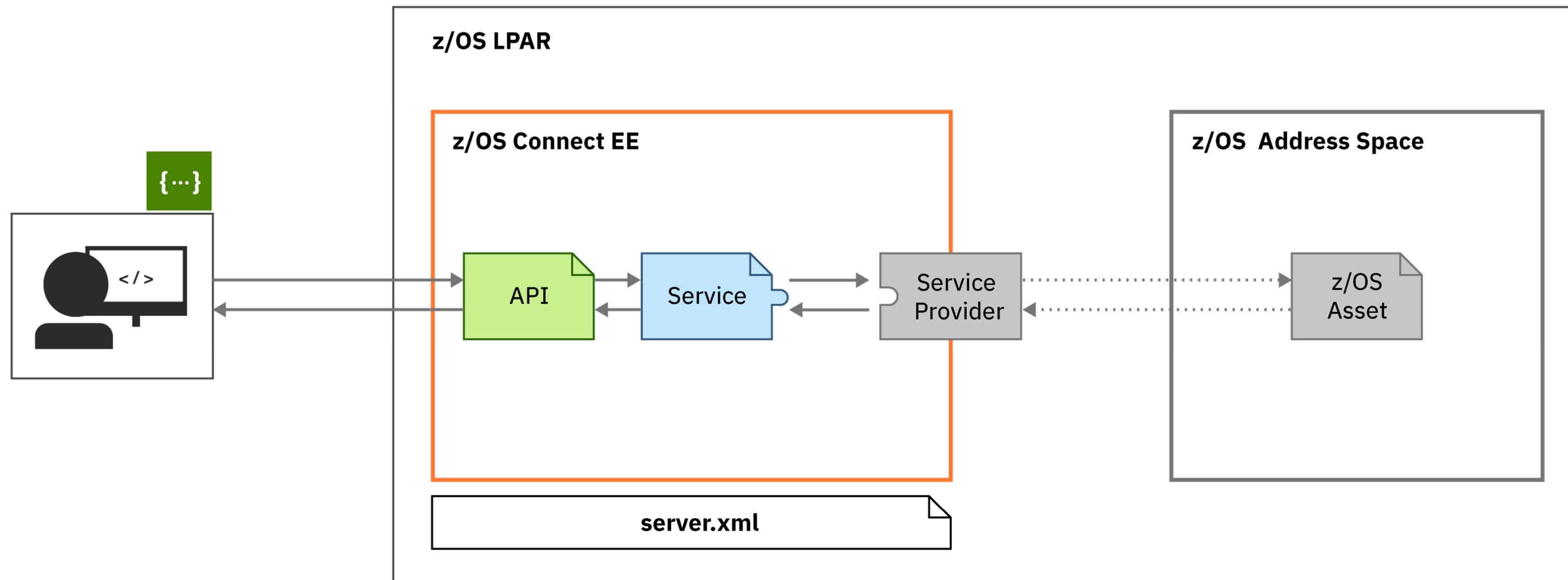


Configure the system-appropriate service provider to connect to your backend system in your `server.xml`.



Complete access to a z/OS Asset (Open API 2)

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)

Start by identifying the format of the messages (e.g., 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.

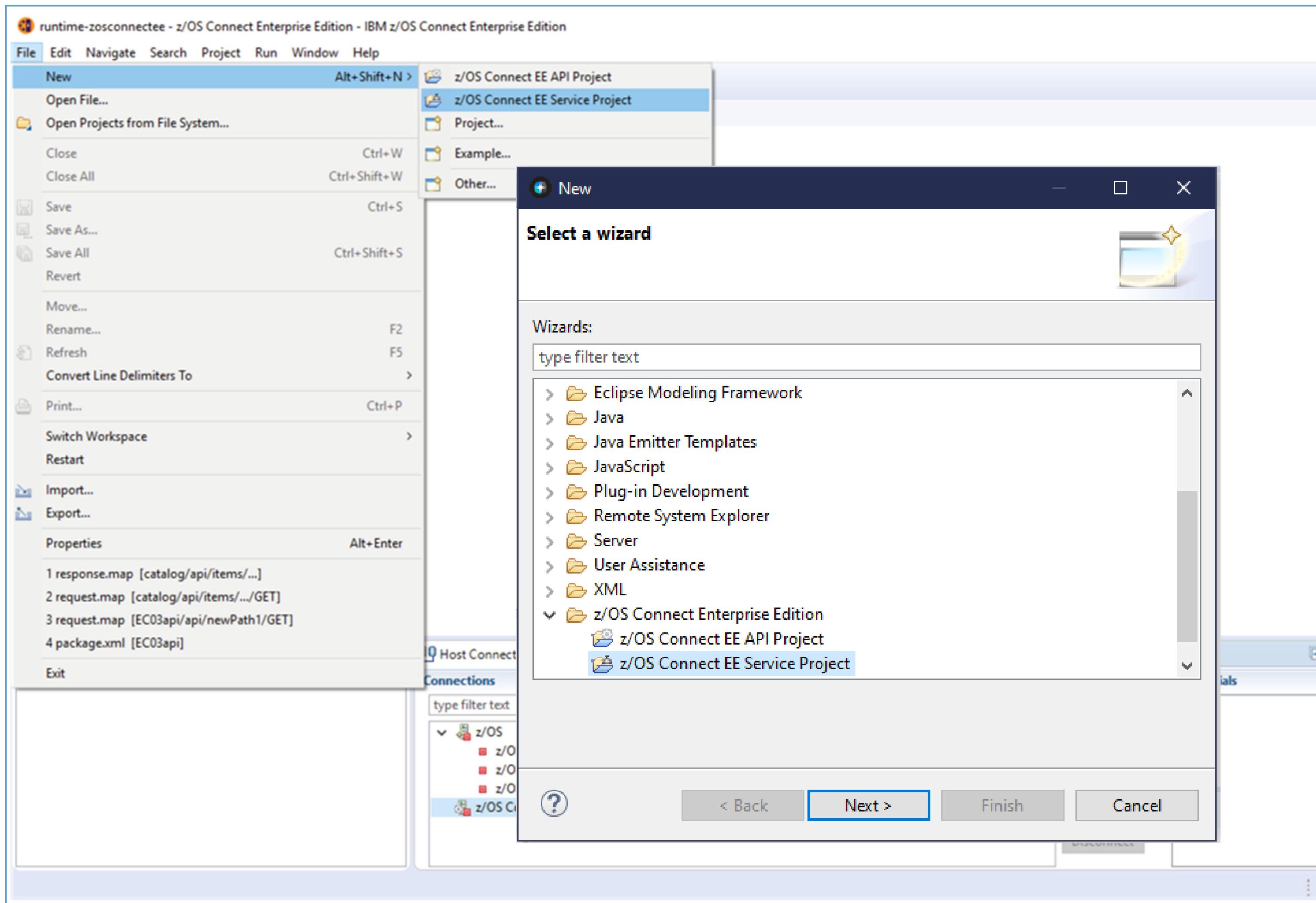
PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.
    INITIALIZE Request-Container.
    INITIALIZE Response-Container.
* Save current CICS userid
    EXEC CICS ASSIGN USERID(USERID of Response-Container)
        END-EXEC.

* Obtain name of channel
    EXEC CICS ASSIGN CHANNEL(Channel-Name)
        END-EXEC.

* If no channel passed in, terminate with abend code NOCN
MA A
Connected to remote server/host wg31 using lu/pool TCP00105 and port 23
Adobe PDF on Documents\*.pdf
04/015
```



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



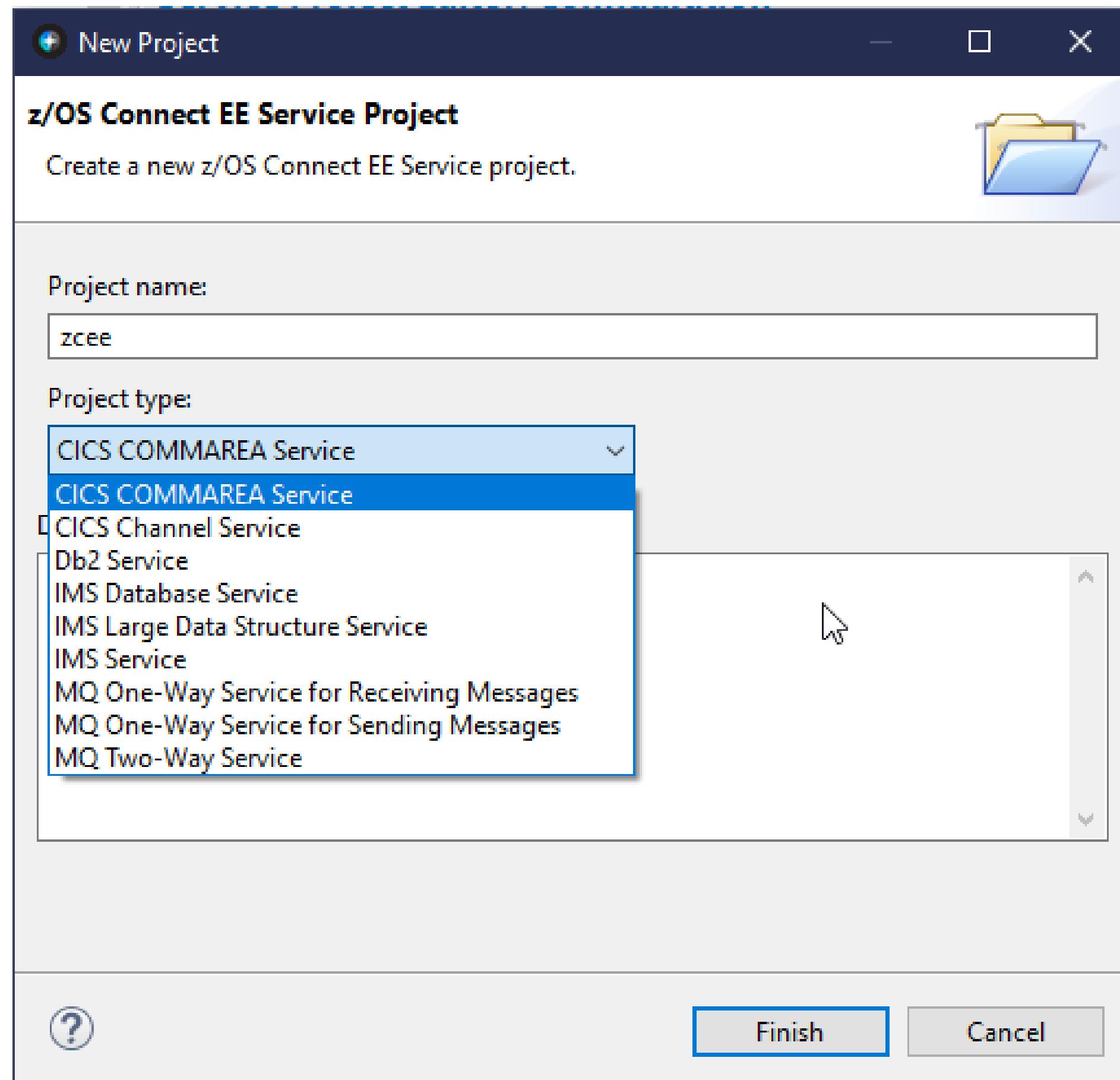
Use the **API toolkit** to create services through Eclipse-based tooling.

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



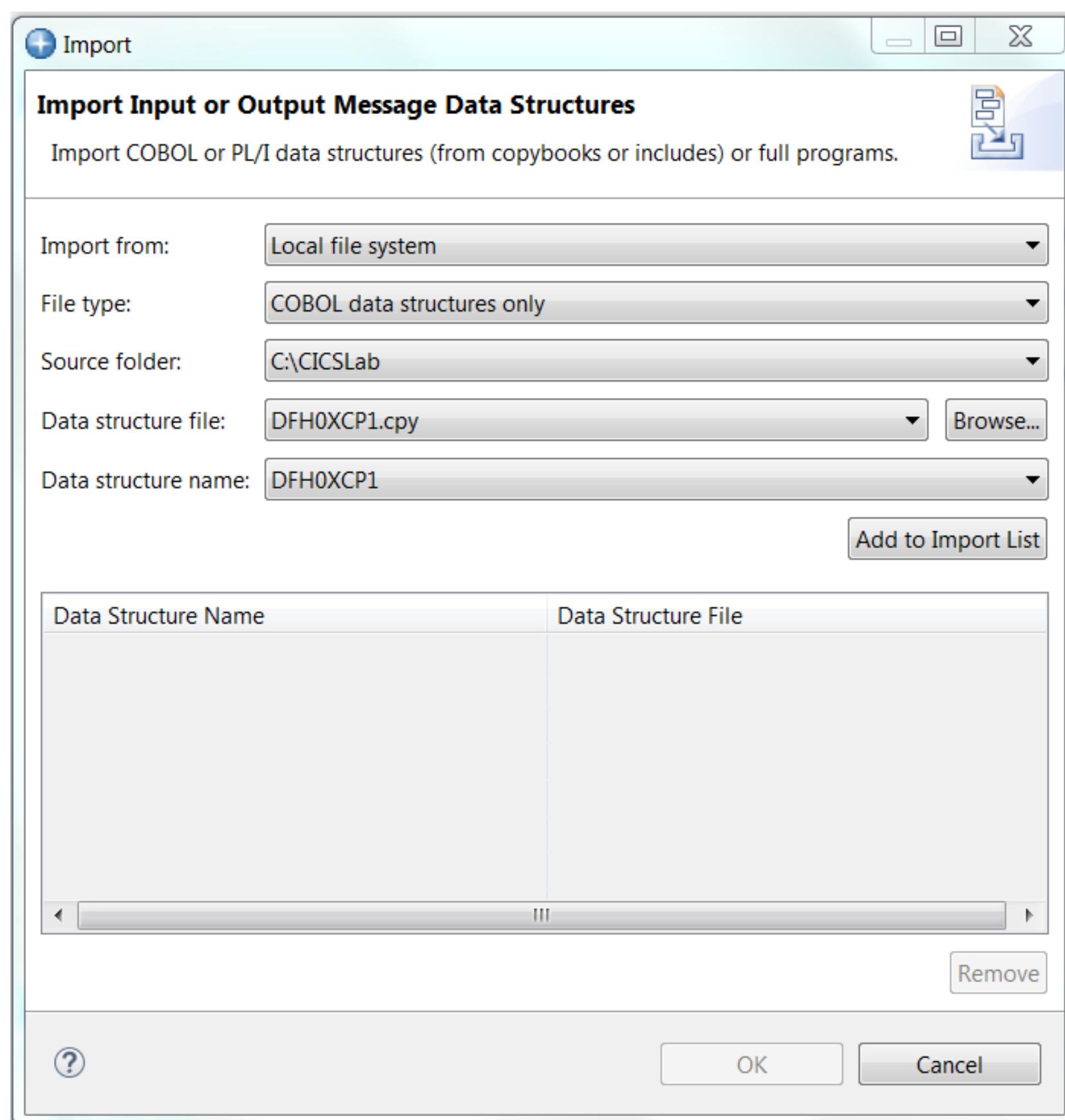
A common interface for service creation, irrespective of back-end subsystem.

- CICS services that can invoke almost any CICS programs accessed by EXEC CICS LINK request (COMMAREA or CHANNEL). See URL <http://www.ibm.com/docs/en/cics-ts/5.6?topic=link-exception-conditions-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



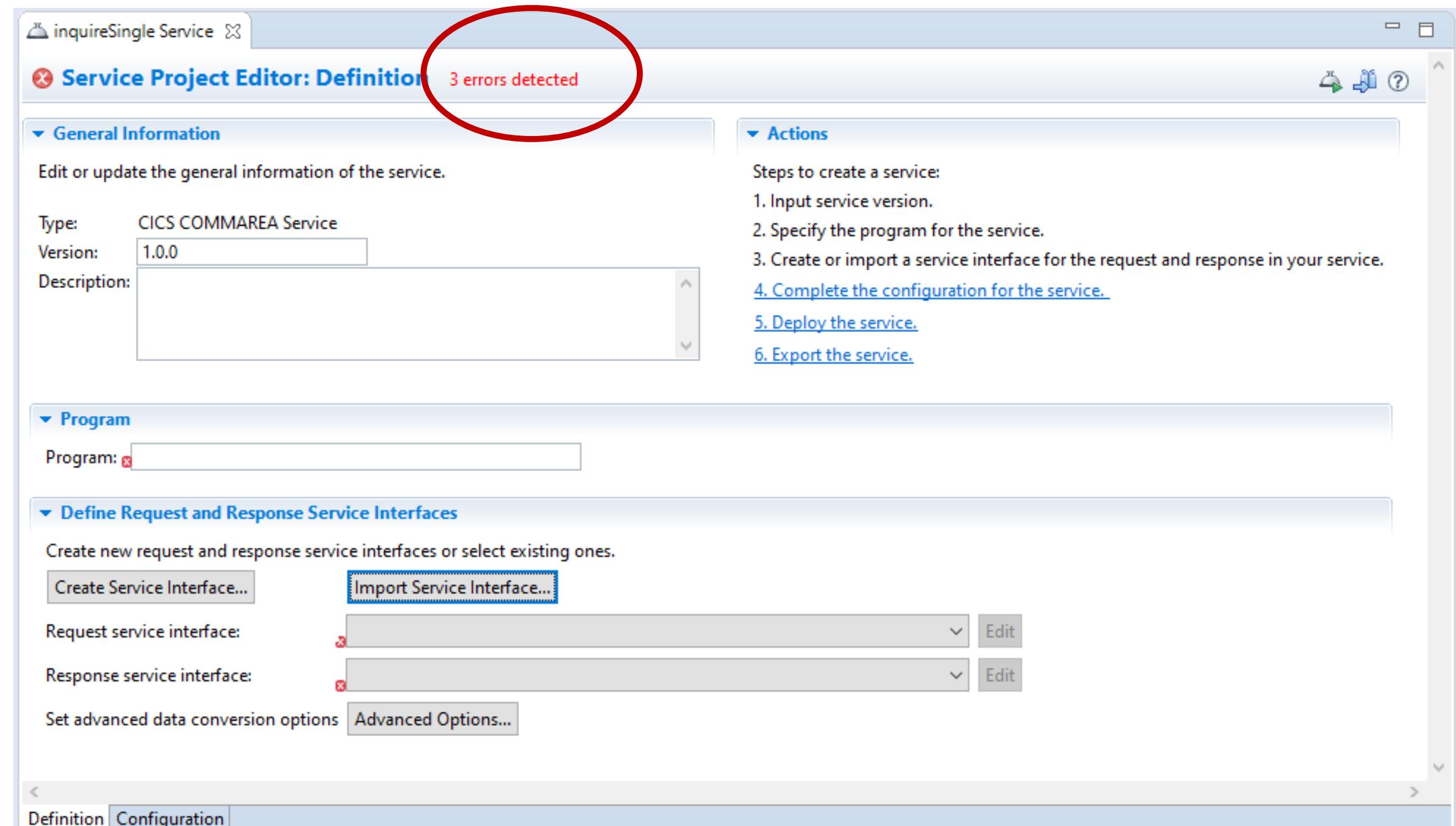
API toolkit – Creating Services for CICS, IMS TM and MQ

Creating a service project from source for a COMMAREA, Container or 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

Allows editing 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.

The screenshot shows a software interface titled "Service Interface Definition". It displays a table of fields with columns for "Fields", "Include", "Interface rename", "Default Field Value", "Data Type", "Field Length", and "Start Byte". A red box highlights the "Interface rename" column for the "CA_REQUEST_ID" row, which has the value "01INQS". Another red box highlights the entire row for "CA_INQUIRE_REQUEST redefines CA_INQUIRE_SINGLE redefines CA_ORDER_REQUEST". This row contains the "inquireSingle" value in the "Interface rename" column and has two checked checkboxes in the "Include" column. The "inquireSingle" value is also circled in red.

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFHOXCP1						
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)		CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines CA_INQUIRE_SINGLE redefines CA_ORDER_REQUEST	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	CA_INQUIRE_REQUEST inquireSingle CA_ORDER_REQUEST		STRUCT	911	88
CA_ITEM_REF_REQ	<input 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 CA_INQUIRE_REQUEST	<input type="checkbox"/>	CA_ORDER_REQUEST		STRUCT	911	88



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

And editing 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:

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA	<input type="checkbox"/>					
DFH0XCP1	<input type="checkbox"/>					
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 redefines	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_INQUIRE_SINGLE redefines CA_INQUIRE_REQUEST	<input 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 redefines CA_INQUIRE_SINGLE	<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

Creating multiple services definitions to the same resource

The Service Interface Editor displays two tables for defining request and response service interfaces:

- cscvincSelectService Service:** This table defines the request service interface. It includes fields for Channel, Container1, REQUEST_CONTAINER, and FILEA_AREA. The REQUEST_CONTAINER row has an ACTION field set to 'S' (Select), which is highlighted with a red circle.
- cscvincSelectRequest:** This table defines the response service interface. It includes fields for Channel, Container1, REQUEST_CONTAINER, and FILEA_AREA. The REQUEST_CONTAINER row has an ACTION field set to 'I' (Insert), which is highlighted with a red circle.

The Service Project Editor: Definition window shows the following details:

- General Information:** Type: CICS Channel Service, Version: 1.0.0, Description: [empty]
- Program:** Program: CSCVINC (highlighted with a red circle)
- Define Request and Response Service Interfaces:** Create new request and response service interfaces or select existing ones.
- Code Editor:** Contains the following CICS EVALUATE block:

```
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.
```

The service developer creates distinct services for each function by setting the ACTION field to S for select, I for insert, U for update or D for delete



Accessing a CICS program – The significance of Transaction ID Usage

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 (selected)

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
EIBDS = '.....'
+ EIBREQID = '.....'

ENTER: CONTINUE PF1 : UNDEFINED PF2 :
PF4 : SUPPRESS DISPLAYS PF5 :
PF7 : SCROLL BACK PF8 :
PF10: PREVIOUS DISPLAY PF11:
MA 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:
MA D
Connected to remote server/host wg31a using lu/pool TCP0013

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'OE02' LINK
EIBRCODE = X'000000000000
EIBDS = '.....'
+ 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 PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CON
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFIN
MA D
Connected to remote server/host wg31a using lu/pool TCP0014

```

EIB_AND_MIRROR

```

WG31 - 3270
File Edit Settings View Communication Actions Window Help
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
EIBDS = '.....'
+ EIBREQID = '.....'

ENTER: CONTINUE PF1 : UNDEFINED PF2 :
PF4 : SUPPRESS DISPLAYS PF5 :
PF7 : SCROLL BACK PF8 :
PF10: PREVIOUS DISPLAY PF11:
MA D
Connected to remote server/host wg31a using lu/pool TCP0015

WG31 - 3270
File Edit Settings View Communication Actions Window Help
TRANSACTION: MIJO PROGRAM: DFH
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS LINK PROGRAM
PROGRAM ('CSCVINC')
SYNCONRETURN
TRANSID ('MIJO')
CHANNEL ('Channel')
NOHANDLE

OFFSET:X'001CD6' LINE:
ENTER: CONTINUE PF1 : UNDEFINED PF2 :
PF4 : SUPPRESS DISPLAYS PF5 :
PF7 : SCROLL BACK PF8 :
PF10: PREVIOUS DISPLAY PF11:
MA D
Connected to remote server/host wg31a using lu/pool TCP0016

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

EIBTIME = 182613
EIBDATE = 0122051
EIBTRNID = 'MIJO'
EIBTASKN = 8949
EIBTRMID = '/ADB'

EIBCPSON = 0
EIBCALEN = 0
EIBAID = X'00'
EIBFN = X'OE02' LINK
EIBRCODE = X'000000000000
EIBDS = '.....'
+ 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 PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CON
PF10: PREVIOUS DISPLAY PF11: EIB DISPLAY PF12: UNDEFIN
MA C
Connected to remote server/host wg31a using lu/pool TCP0016 and port 23

```

- **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

mitchj@us.ibm.com

These attributes also be used in `zosconnect_cicsIpConnection` and `zosconnect_services>service configuration` elements.

© 2017, 2023 IBM Corporation

Slide 48



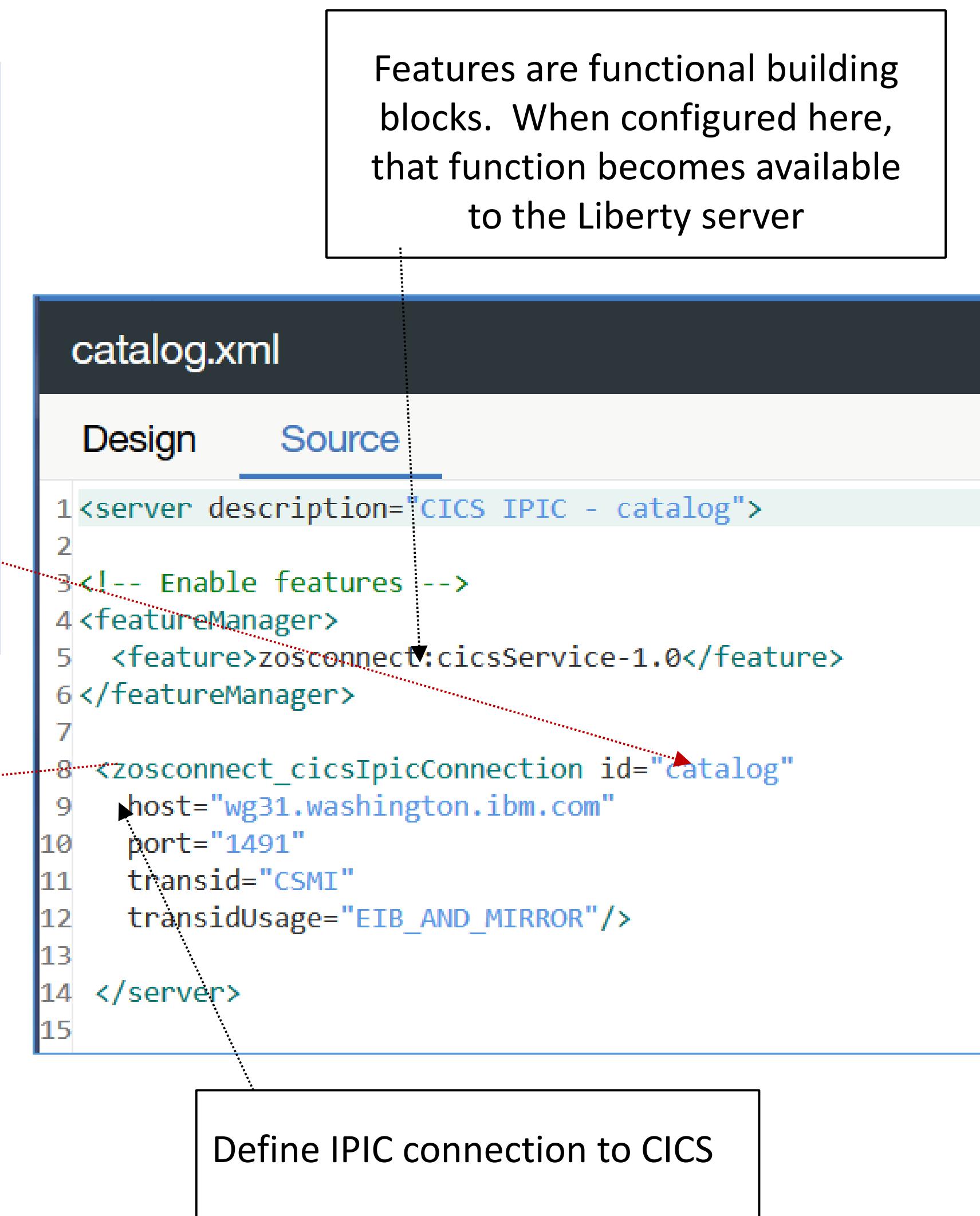
Accessing a CICS program uses CICS IP interconnectivity (IPIC)

The server.xml file is the key configuration file:

```
OVERTYPE TO MODIFY
CEDA ALter TCpipservice( IPIC      )
  TCpipservice : IPIC
  GROUp       : SYSPGRP
  DEscription  ==> DFHISAIPIPC
  Urm          ==> 01491
  POrtnumber   ==> 01491
  STatus       ==> Open
  PROtocol    ==> IPic
  TRansaction  ==> CISS
  Backlog      ==> 00000
  TSqprefix    :
  Host         ==> ANY
  (Mixed Case) ==>
  Ipaddress    ==> ANY
  Speciftcpss  ==>
  Socketclose  ==> No
  MAXPersist   ==> No
  + MAXDataLEN ==> 000032

  1-65535      Open | Closed
  0-32767      Http | Eci | User | IPic
  3-524288     No  | 0-240000 (HHMMSS)
                No  | 0-65535

  SYSID=CICS APPLID=CICS53Z
PF 1 HELP 2 COM 3 END      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
MA E
Connected to remote server/host wg31 using lu/pool TCP00104 and port 23
```





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	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 checked="" type="checkbox"/>	IN_COMMAND	IVTNO DISPLAY	CHAR	8
IN_LAST_NAME	<input type="checkbox"/>	lastName		CHAR	10
IN_FIRST_NAME	<input type="checkbox"/>	IN_FIRST_NAME		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: IMSINTER

Interaction profile:

▼ 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, X=BAREA=100, RACF=Y, RRS=N)
TCPIP= (HOSTNAME=TCPIP, PORTID=(4000, LOCAL), RACFID=JOHNSON,
TIMEOUT=5000)
DATASTORE= (GROUP=OTMAGRP, ID=IVP1, MEMBER=HWSMEM, TMEMBER=OT
MAMEM)
```



API toolkit – Creating Services for IMS DB

Creating a service project from the IMS Catalog

The screenshot shows the 'Service Project Editor: Definition' window. In the 'Actions' section, steps for creating a service are listed: 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, and 6. Export the service. The 'SQL Command' field contains the query: 'SELECT FIRSTNME, ZIPCODE, PHONENBR, A1111111 FROM ATSVPA.A1111111 WHERE A1111111=?'. This field is circled in red.

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=DFSIVD1,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=IDX,RULES=LAST
DBDGEN
FINISH
END
```



API toolkit – Creating Services for IMS DB

The Toolkit allows editing a service interface definitions*

The screenshot shows the Service Interface Editor window. The title bar says "response" and "Service Interface Editor". A message at the top reads: "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." Below is a search bar and a set of icons for sorting and filtering. The main area is a table with columns: Fields, Include, Interface Rename, Default Field Value, Data Type, and Field Length.

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

ConnectionFactory

```
<connectionFactory id="DFSIVPACConn">
<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 – Creating Services for MQ

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

MQ Meta Data

The screenshot shows two windows side-by-side:

Service Interface Editor: This window displays a table of fields from a data structure named "miniloanServiceRequest". The table has columns: Fields, Include, Interface Rename, Default Field Value, and Data Type. A red circle highlights the "Include" column for several fields under the "MINILOAN_COMMAREA" section. A red box highlights the "Interface Rename" column for the "loan application" row, which contains a list of fields: name, credit score, yearly income, age, loan amount, aproved?, effective date, yearly interest rate, yearly payment, authorization identity, MESSAGES_NUM, and disapproval message. The "Default Field Value" column for the "name" field contains "F", and for "MESSAGES_NUM" it contains "00005".

Service Project Editor: Configuration: This window shows configuration settings for a service named "twoWay Service". It includes sections for Required Configuration, MQMD format, and Reply selection. A red circle highlights the "jms/qmgrCf" and "jms/requestQueue" fields in the "Required Configuration" section. Another red circle highlights the "jms/replyQueue" and "2000" fields in the same section.

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

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">
    <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 X

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

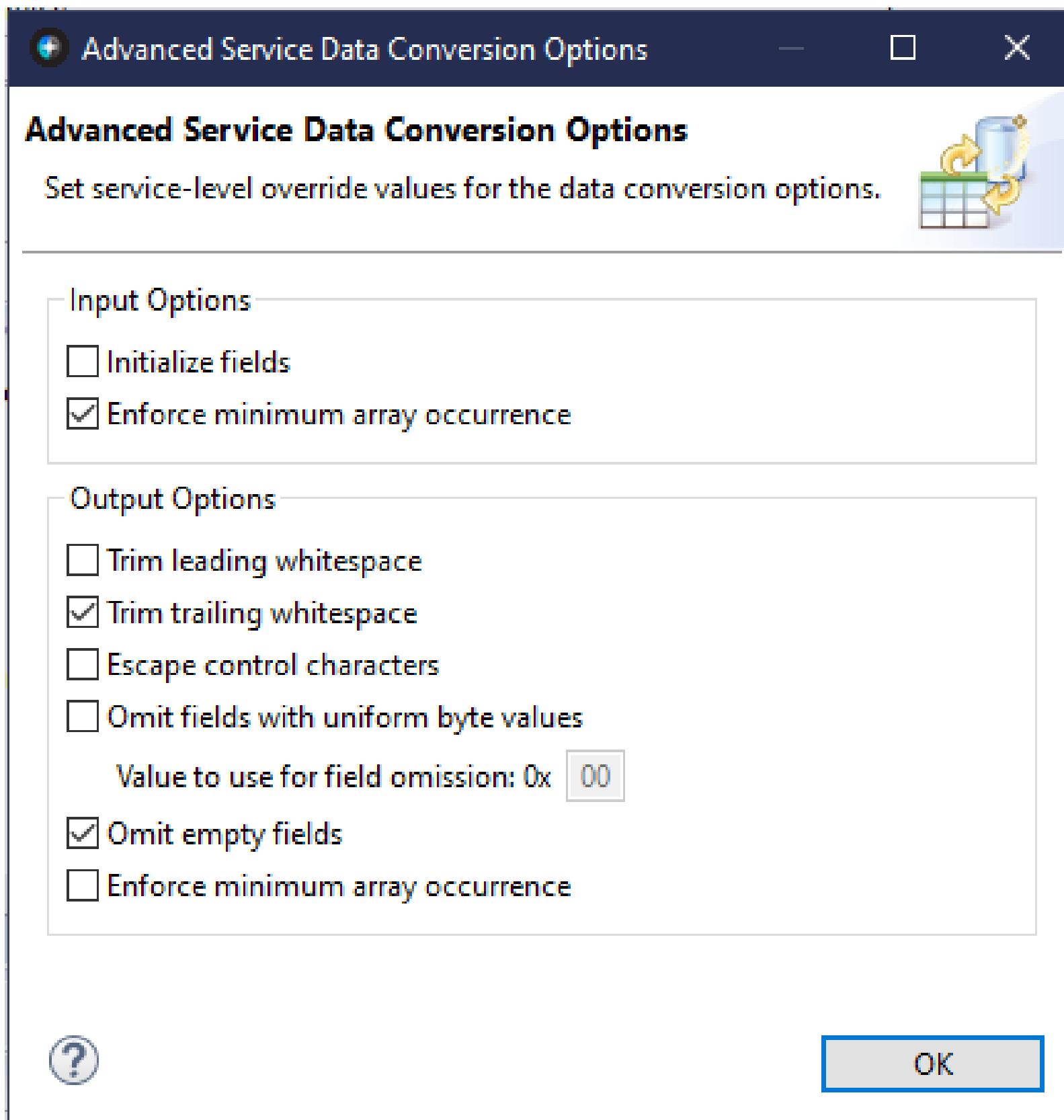
mq.xml

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="CLIENT"
15     queueManager="QMZ1" />
16 </jmsConnectionFactory>
17
18 <jmsConnectionFactory id="qmgrCF2" jndiName="jms/qmgrCF2">
19   connectionManagerRef="ConMgr1"
20   <properties.wmqJMS transportType="CLIENT"
21     queueManager="ZMQ1"
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
47
```



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

```
//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	V1	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	V2	zCEEService	Select an employee from table USER1.E...
selectEmployee	V2	zCEEService	Select an employee from table USER1.EM...
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 Import from local machine...
Response JSON schema: response-schema.json 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 interface for a project named "selectEmployee Service". The left pane displays various service definitions, including DSNL004I, USIBMWZ, and WG31.WASHINGTON.IBM.COM, each with specific parameters like LOCATION, LU, and TCPPORT. The right pane shows the configuration for "selectEmployee Service". A red arrow points from the "db2conn" connection reference in the "Required Configuration" section to the "zosconnect_zosConnectServiceRestClientConnection" element in the "db2pass.xml" XML file. Another red arrow points from the "TCPPORT" value 2446 in the service definition to the "port" attribute in the XML file.

*selectEmployee Service

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection reference: db2conn

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

db2pass.xml

Design Source

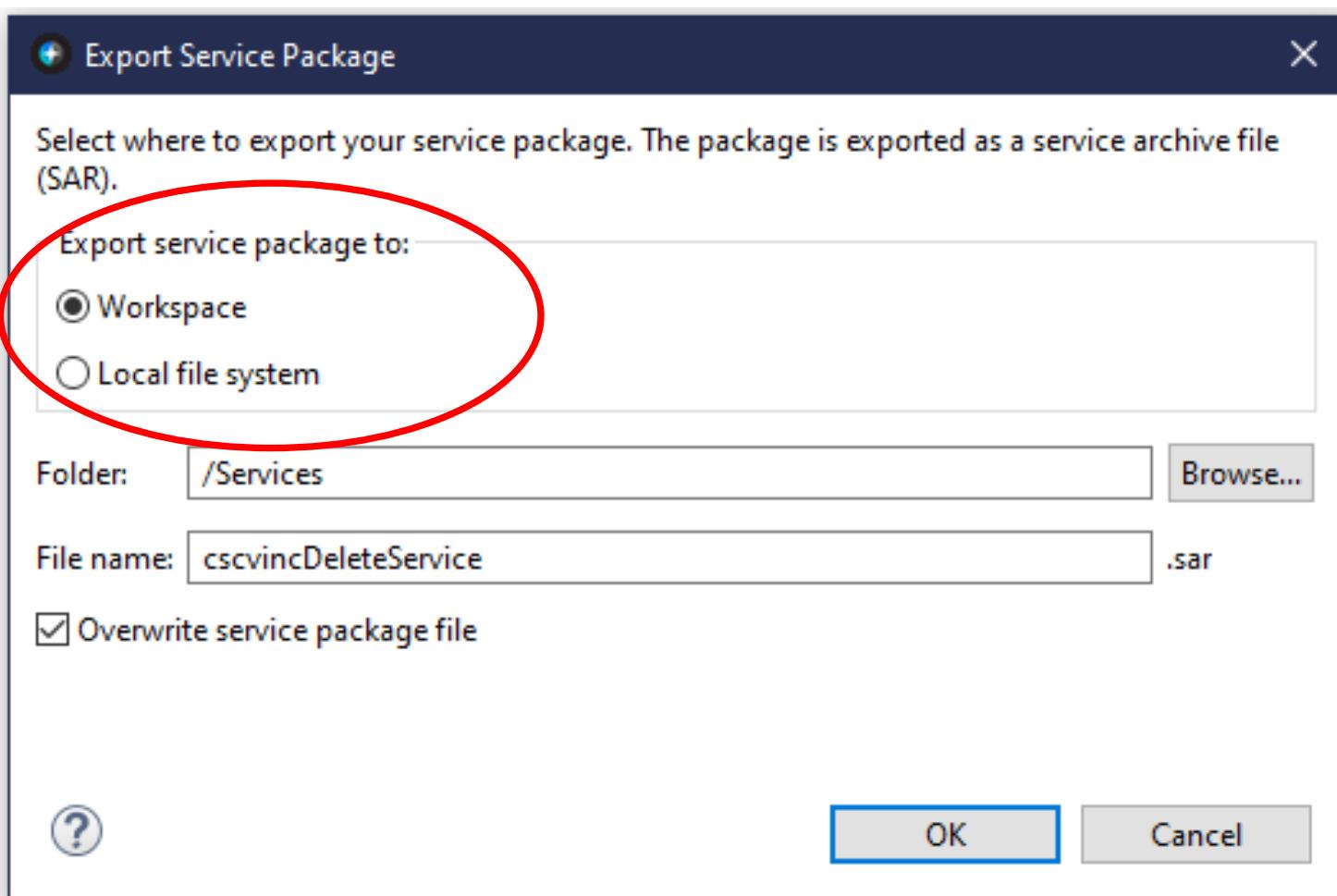
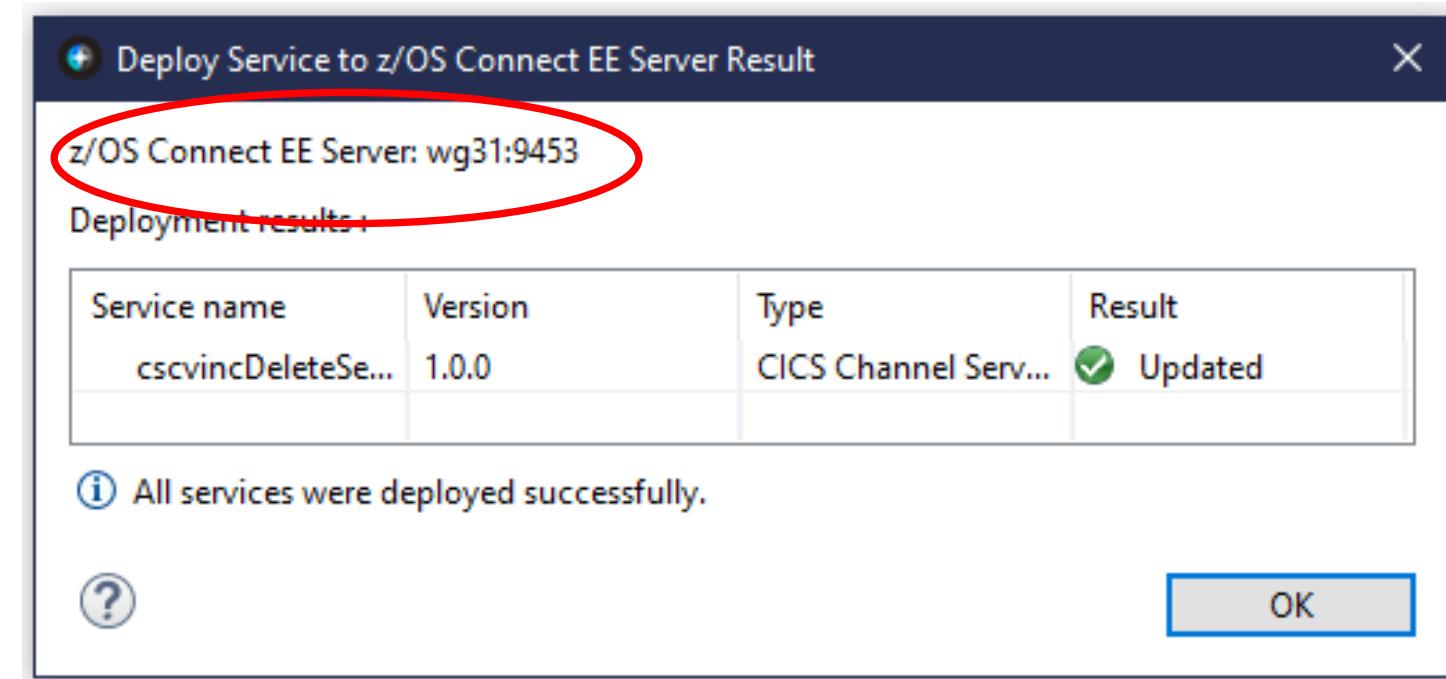
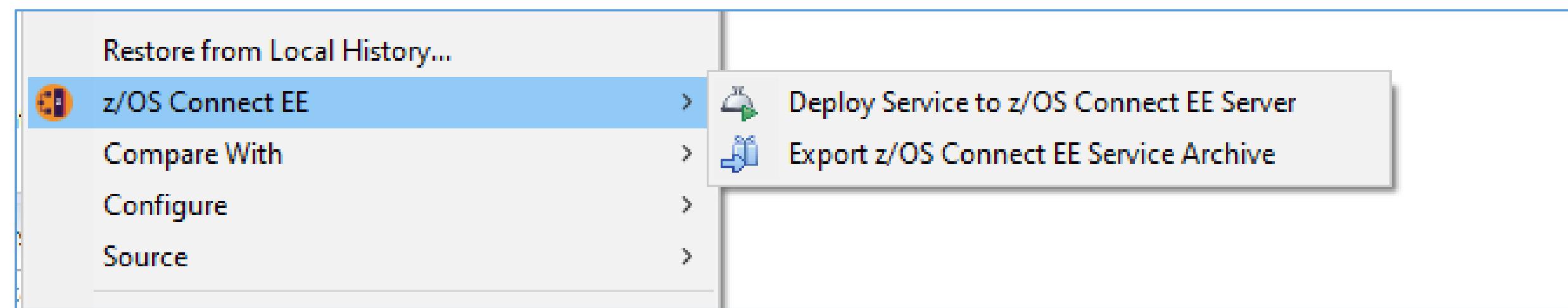
```
1 <server description="DB2 REST">
2
3   <zosconnect_zosConnectServiceRestClientConnection id="db2conn"
4     host="wg31.washington.ibm.com"
5     port="2446"
6     basicAuthRef="dsn2Auth" />
7
8   <zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
9     applName="DSN2APPL"/>
10
11 </server>
12
```



API toolkit – Services Editor

Server connection and Services deployment

Manage z/OS Connect EE 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

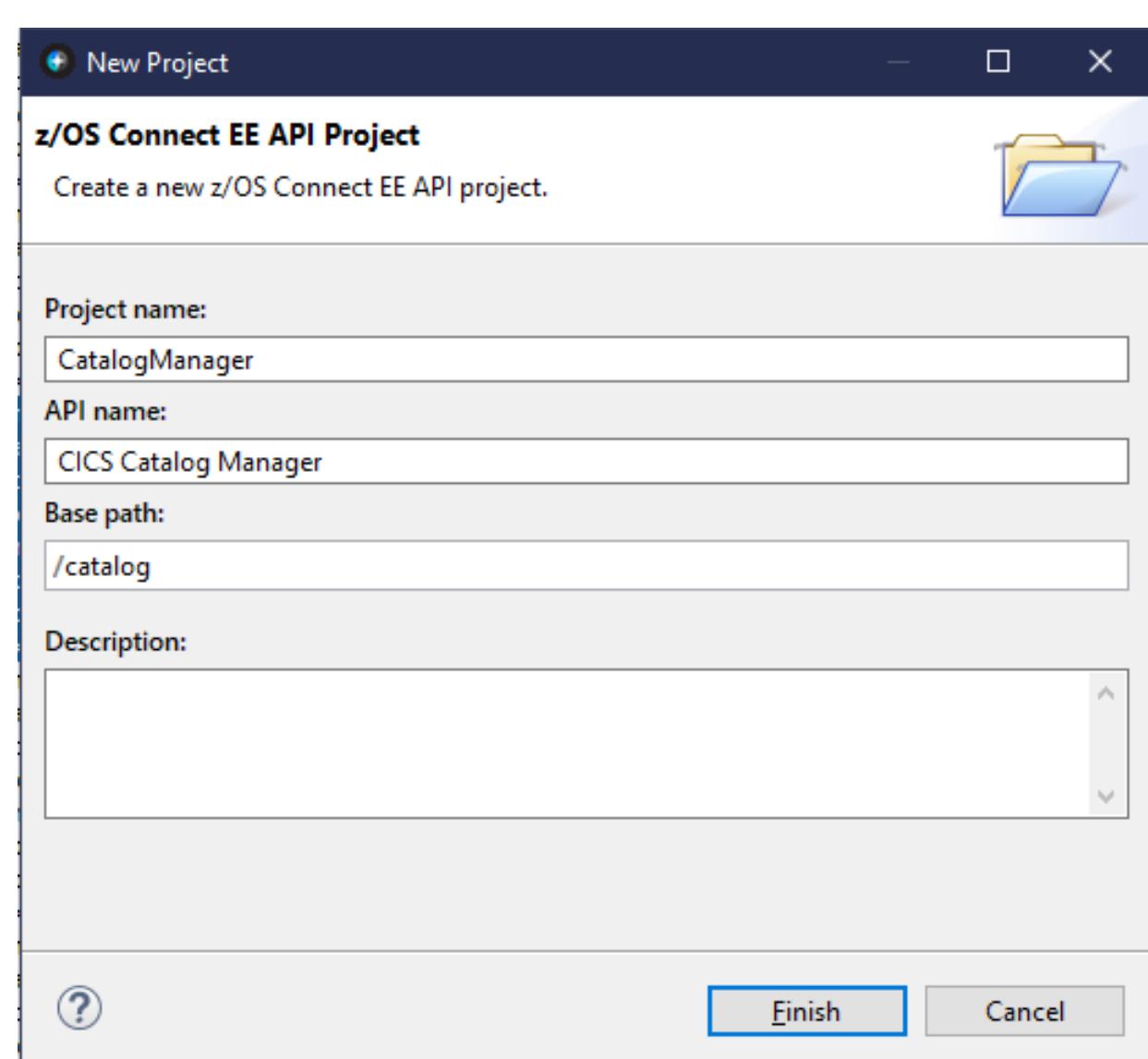


/api_toolkit/api_editor

Quick and easy **API mapping**.



API toolkit – API Editor



Describe your API

Name: CICS Catalog Manager
Base path: /catalog
Version: 1.0.0

Contact Information

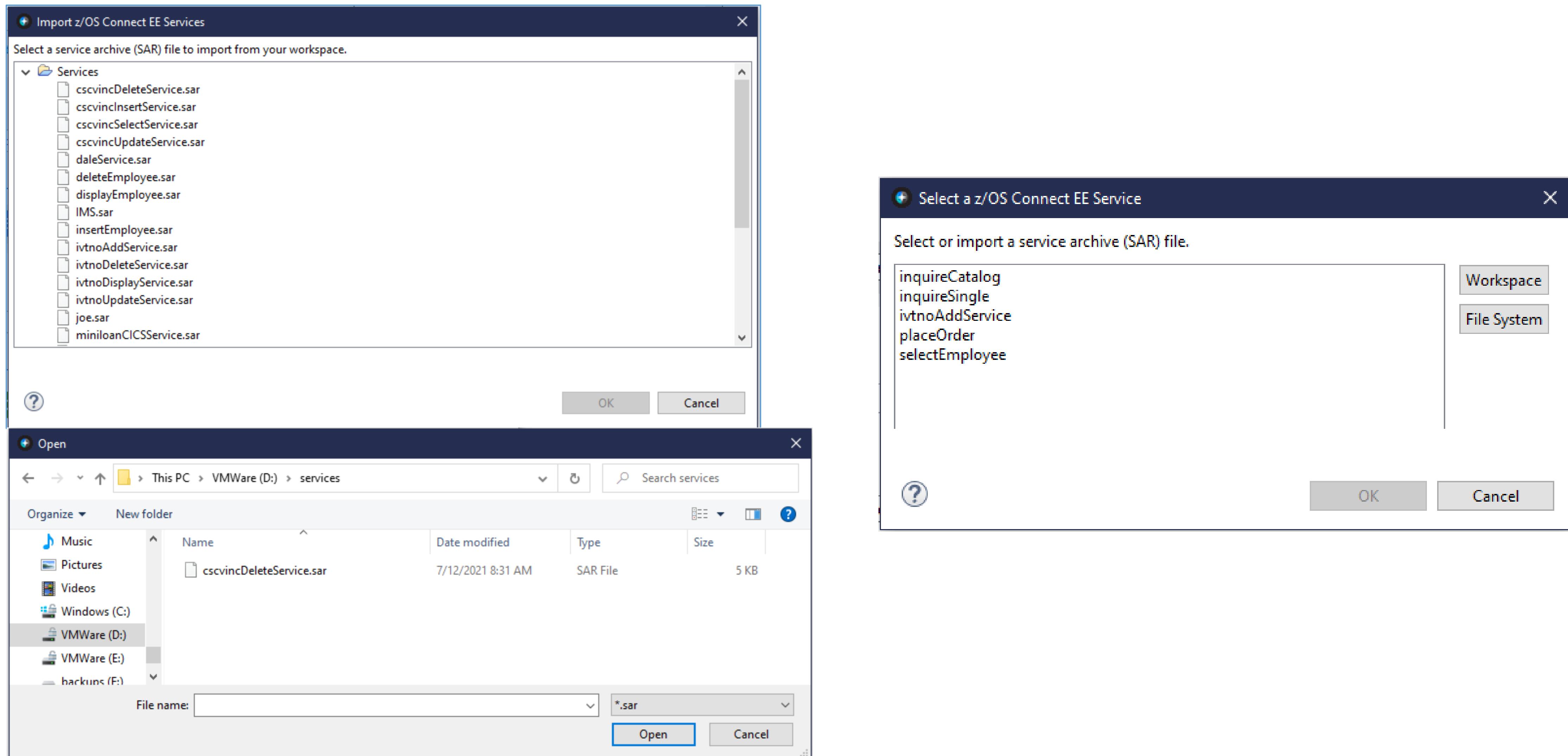
Path

/newPath1

Methods (4)

Method	Action	Service...	Mapping...	Up	Down	Delete
POST						
GET						
PUT						
DELETE						

Importing the service archive file from a filesystem or Eclipse project directory





API toolkit – API Editor

The screenshot shows the API Editor interface within the IBM Explorer for z/OS. It displays three API endpoints:

- catalog**: Path `/catalog`, Methods: GET `inquireCatalog`, PUT `ivtnoAddService`.
- order**: Path `/order`, Methods: POST `placeOrder`, PUT `selectEmployee`.
- itemID**: Path `/item/{itemID}`, Methods: GET `inquireSingle`.

A red oval highlights the `itemID` endpoint.

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

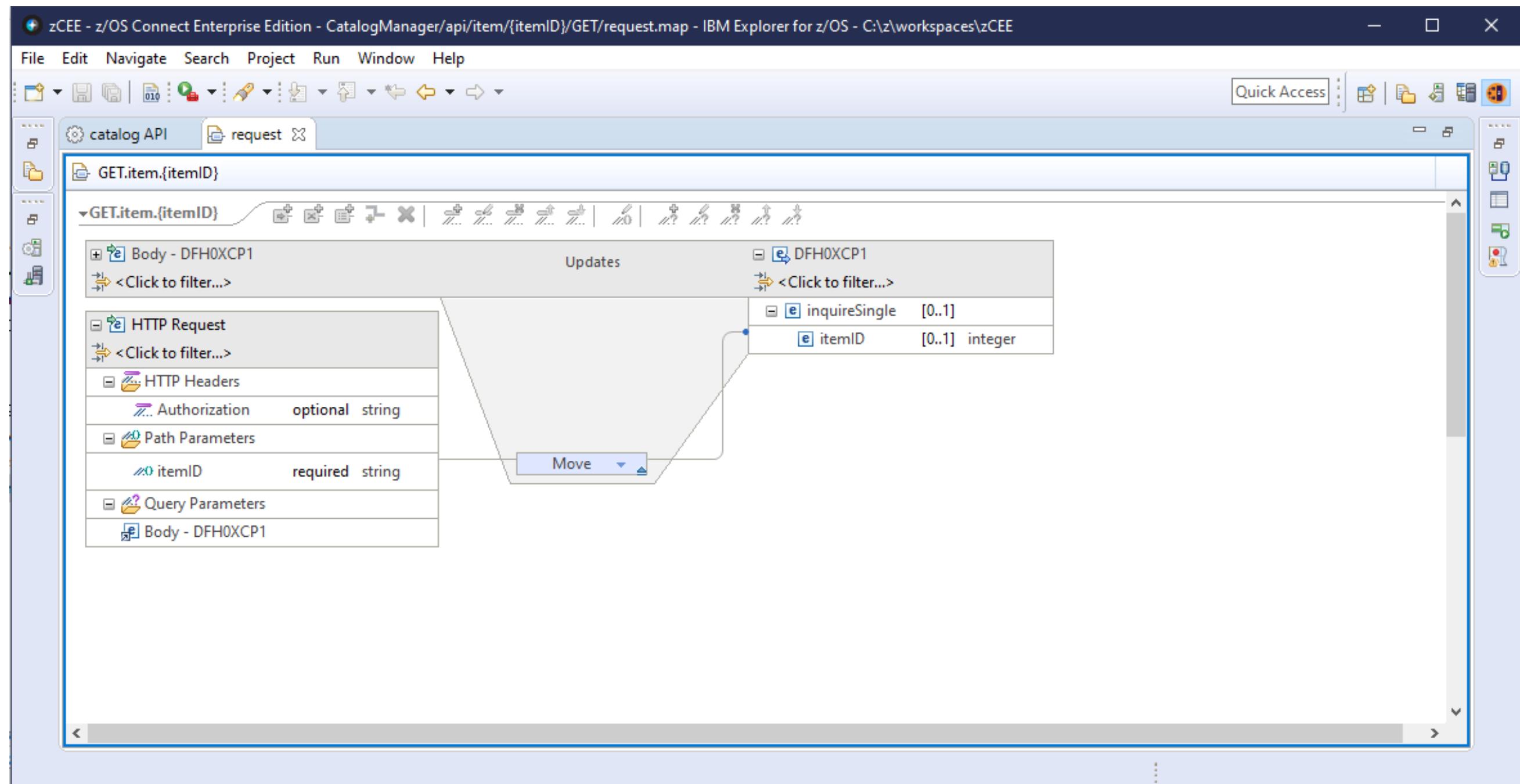
Once you define your API, you can map backend services to each request.

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.

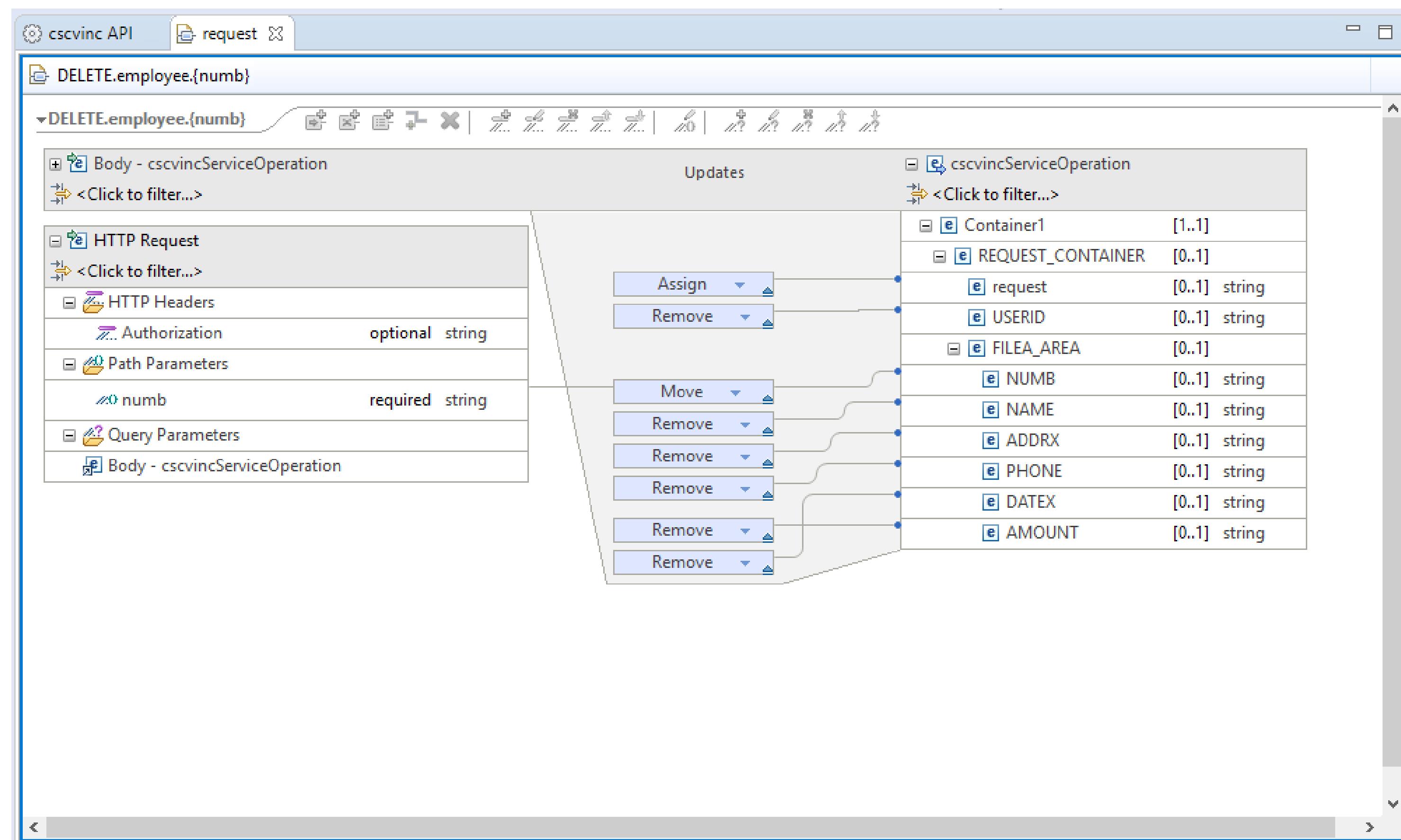
Assign static values to fields, useful for Op codes.

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



API toolkit – API Editor

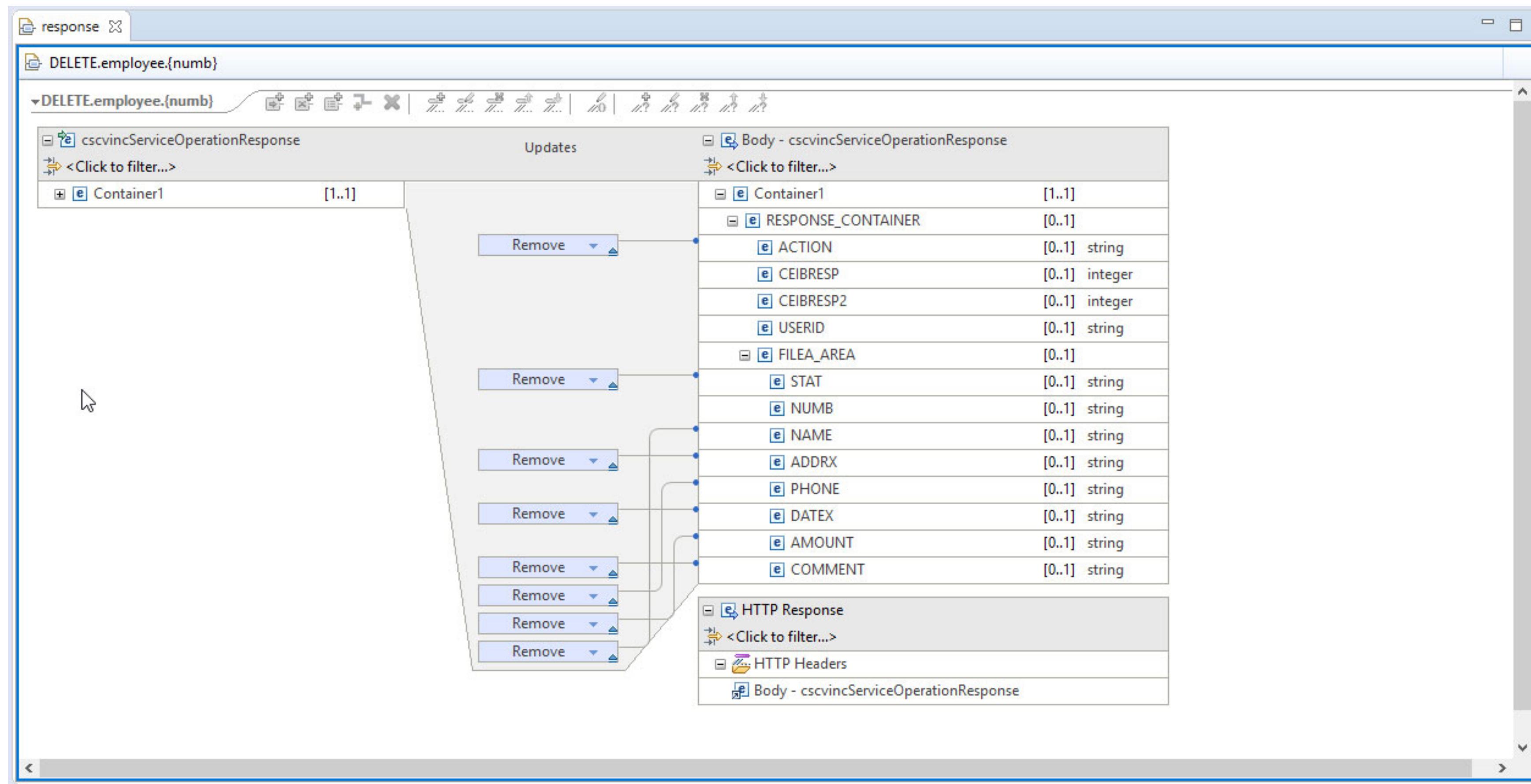
API mapping: Remove or assign values to the fields exposed by service developer





API toolkit – API Editor

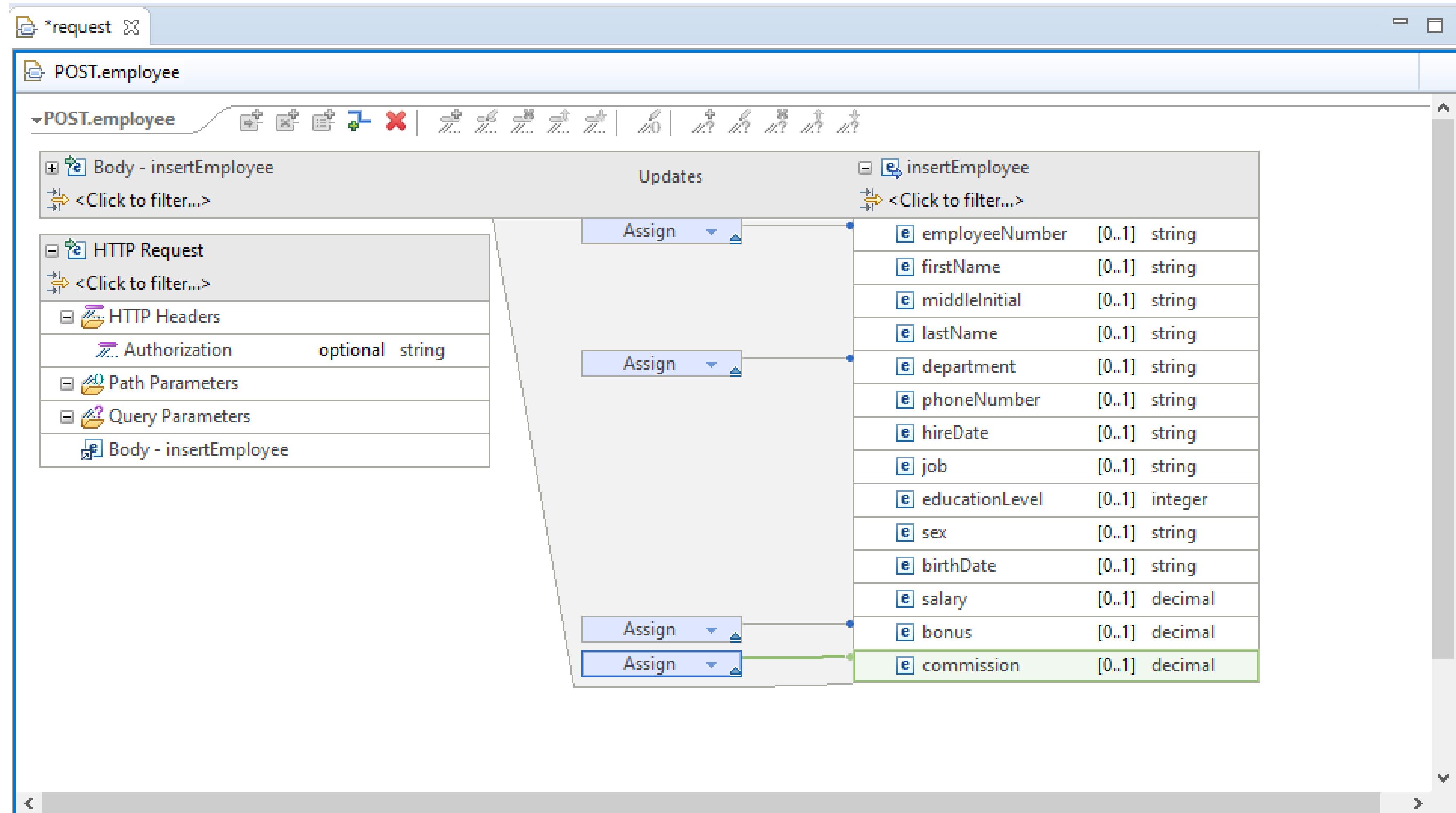
API mapping: Allows the API Developer to remove fields from the response to tailor the API





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 panes. The left pane, titled 'request' and 'POST.queue', contains an 'HTTP Request' section. This section includes an 'HTTP Headers' group, which is circled in red. Inside this group are two entries: 'Authorization' (optional string) and 'ibm-mq-md-correlID' (optional string). Below this group are sections for 'Path Parameters' and 'Query Parameters'. The right pane, also titled 'POST.queue', contains an 'MQPUTOperation' section. This section lists several fields: 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, and comment [1..1] string.



API toolkit

API mapping: API definition with multiple response codes

The screenshot shows the API toolkit interface for defining API mappings. On the left, the API path `/employee/{employee}` is defined with four methods: GET, POST, DELETE, and PUT. The GET method is mapped to the service `cscvincSelectService` with operation id `getCscvincSelectService`. The Responses section for this method shows two entries: 404 Not Found and 200 OK. 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: Rule 1 (using field `!e/Container1/RESPONSE_CONTAINER/CEIBRESP`) and Rule 2 (using field `!/Container1/RESPONSE_CONTAINER/CEIBRESP2`). Both rules map to value 13. 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.

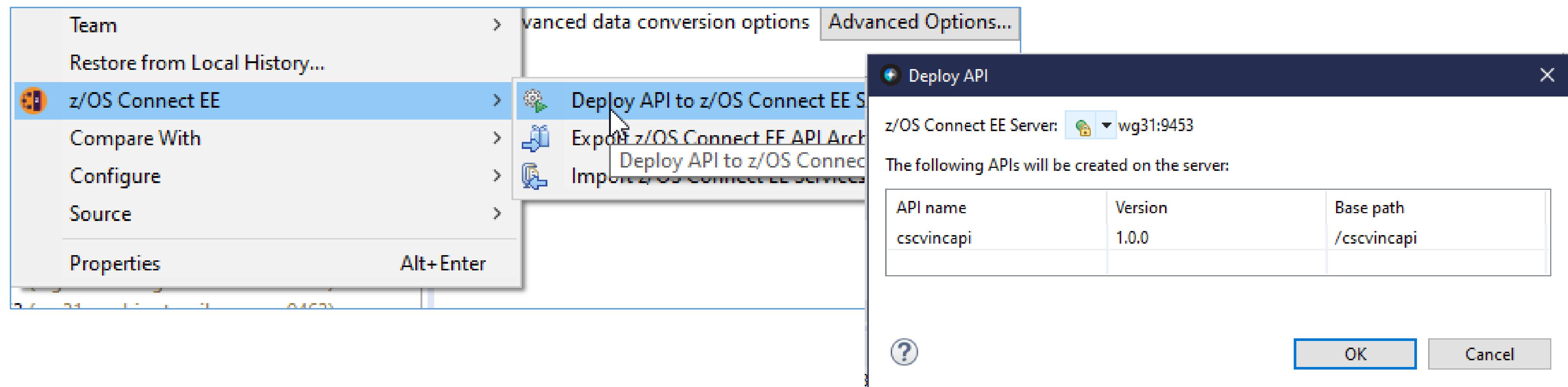
You can define rules based on fields in the service's return interface to tell z/OS Connect EE which response code to return



API toolkit – API Editor

Server connection and API deployment

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



API toolkit – API Editor

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 a selected server (wg31:9443) and its APIs. A context menu is open for the 'cscvinc' API, with options like 'Open In Swagger UI'. Two separate browser windows are displayed, both titled 'Swagger UI' and showing the Swagger UI interface for the 'cscvinc' API. The left window shows the API list with methods: POST /employee, DELETE /employee/{employee}, GET /employee/{employee}, and PUT /employee/{employee}. The right window is a detailed view of the GET /employee/{employee} endpoint, showing the Response Class (Status 200), Model (Example Value), Parameters (Authorization header, employee path), and Response Messages (HTTP Status Code 404, Reason Not Found). The response model is shown as JSON:

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



/api_toolkit/services

Creating **service creation** not using the Eclipse Toolkit



Creating 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.
```



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. A POST request is selected with the URL `https://wg31.washington.ibm.com:9453/zosConnect/services`. The 'Body' tab is active, displaying a JSON object representing a service archive file. The JSON content is as follows:

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

- 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": 60000, "port": "2446", "host": "wg31.washington.ibm.com", "basicAuthConfigId": "dsn2Auth", "id": "Db2Conn", "httpMethod": "POST", "connectionTimeout": 30000, "uri": "/services/selectEmployee"}}
```



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



Open API 3 z/OS Connect Designer

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

The specification description provides the details of the API (OpenAPI 3)



```
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: A lesson learned the hard way. . . .

z/OS Native Server Considerations – API Provider Deployment

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}/configDropins/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"/>
```



z/OS Server Considerations – API Context Root

<https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=devops-zos-connect-server-overview>

The drop-ins directory

The *drop-ins* directory, `/config/dropins` is a special directory that is supported by WebSphere® Application Server for Liberty. It allows `.war` 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 simplifies the 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 of `/`, 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 WebSphere Application Server for Liberty special subject `ALL_AUTHENTICATED_USERS`. For more information, see [How to define authorization roles](#).

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

• Server XML configuration required to define applications and add 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"/>
```

The API's specification required updates



The image shows two side-by-side Notepad windows. The left window is titled 'cscvinc.json - Notepad' and contains a JSON API specification. The right window is titled 'cscvinc.yaml - Notepad' and contains a YAML API specification. Both files include sections for 'servers', 'paths', and 'parameters'. Red ovals highlight several specific sections in both files: the 'servers' section, the 'tags' and 'operationId' section under a 'post' operation, the 'parameters' section under a 'get' operation, and the 'parameters' section under another 'post' operation. The 'parameters' sections are particularly circled in both files.

```
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",
          }
        }
      }
    }
  }
}
```

```
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:
            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: getEmployeeSelectService
      x-ibm-zcon-roles-allowed:
        - Staff
      parameters:
        - name: Authorization
          in: header
          schema:
            type: string

```

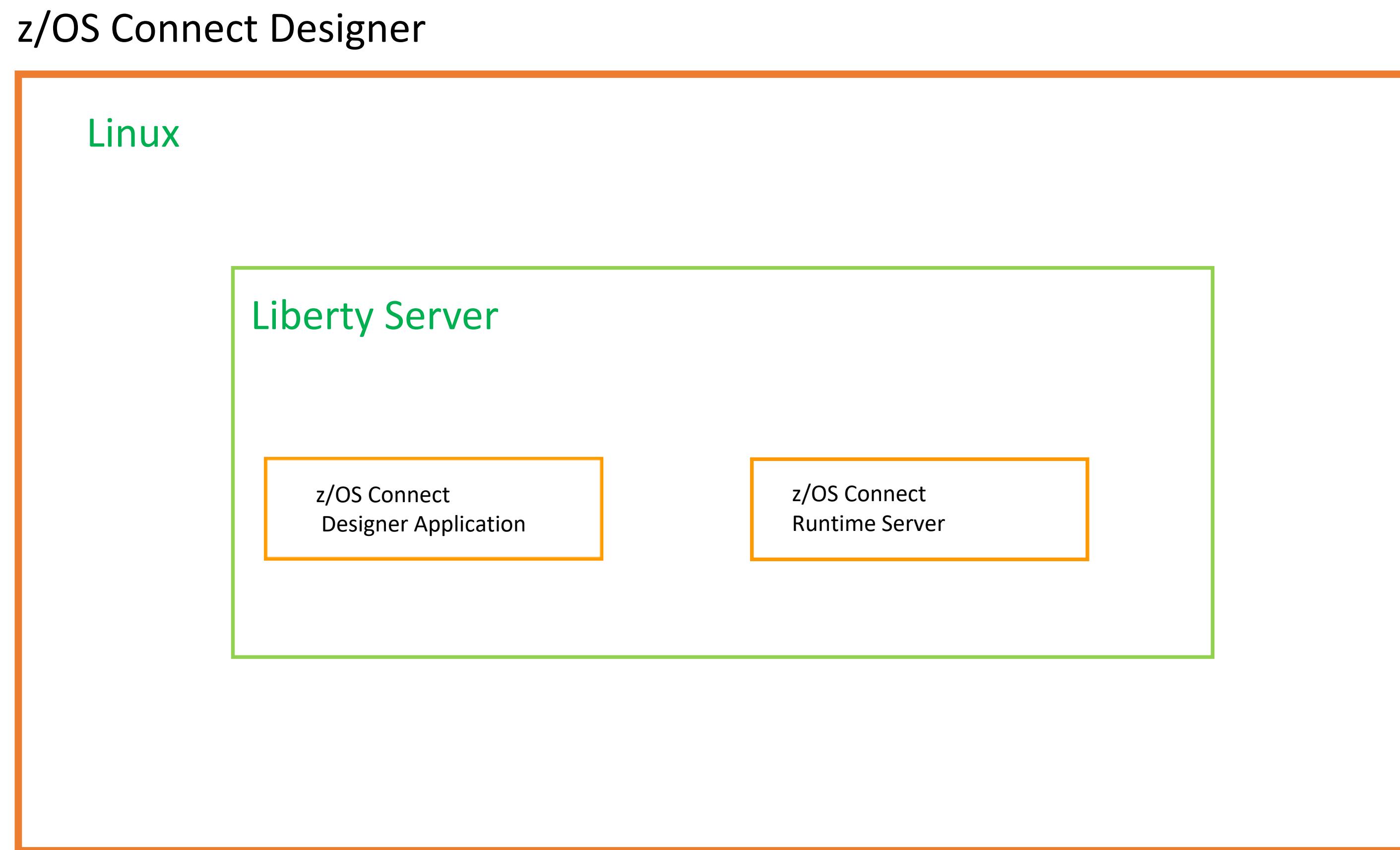
Where to make these
Is the question.

We recommend
making the changes in
the specification file
before it is imported
into the Designer.



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 runtime options

one API per container, multiple Designer containers may be required

A Windows, Mac OS# 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

Windows considerations



Windows 10

Windows Subsystem for Linux (WSL)

Ubuntu, openSUSE, etc.

z/OS Connect
Designer Container

z/OS Connect
Designer Container

z/OS Connect
Designer Container

z/OS Connect
Designer Container

A z/OS Connect Designer is an application running in Liberty running in Linux (the container) which is running Linux (the container runtime) running in WSL running in Windows.

Tech-Tip: Some suggested runtime environments that required no license one goal to show you why you don't need the GUI tools (or why I am considered a neanderthal)

The screenshot shows the Docker Engine documentation page for the 'Server' section. It includes a table of supported platforms across different architectures (x86_64/amd64, arm64/aarch64, arm (32-bit), s390x) for various Linux distributions like CentOS, Debian, Fedora, Raspbian, RHEL, SLES, Ubuntu, and Binaries.

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

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

The screenshot shows the Podman documentation 'Get Started' page. It features a large heading 'Get Started with Podman' and a brief introduction: '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.' There are three call-to-action buttons: 'Installation Instructions', 'Documentation', and 'Podman Troubleshooting Guide'.

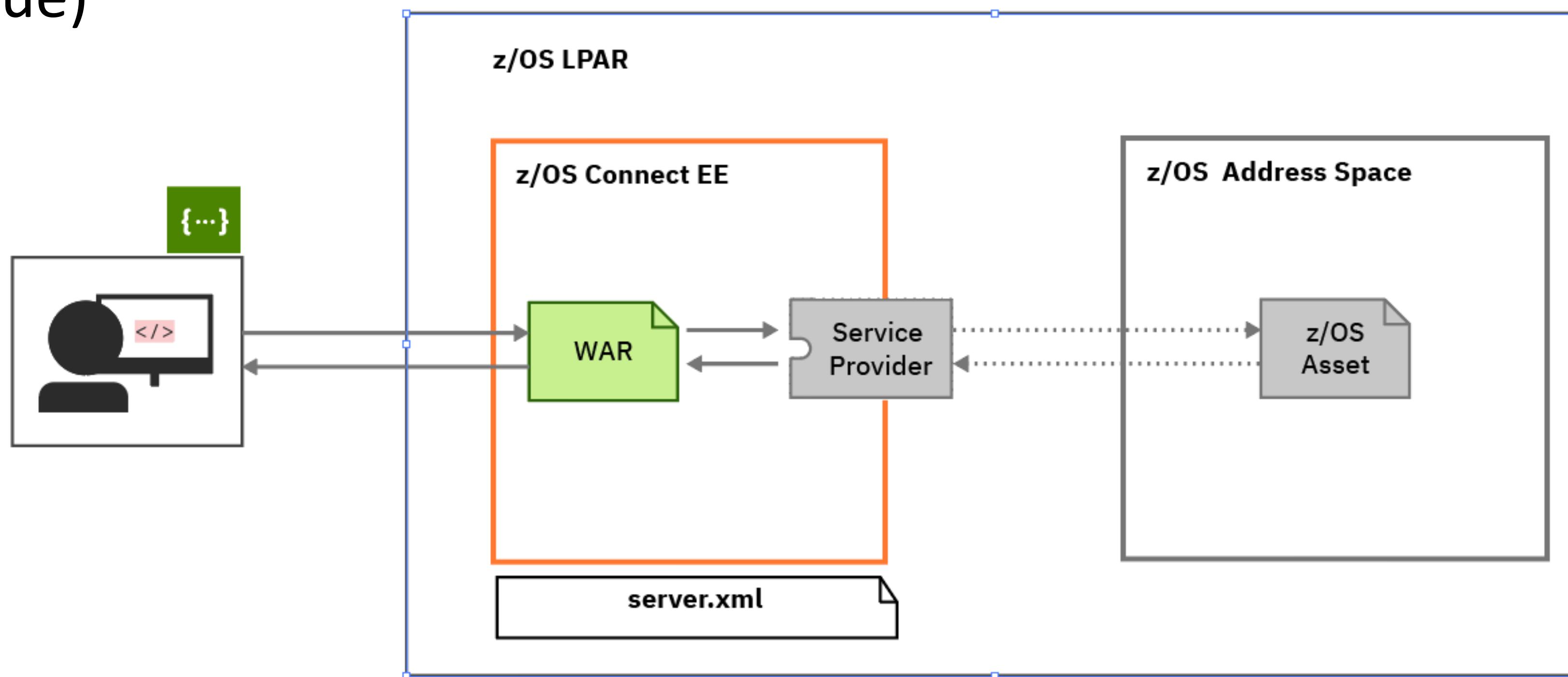
<https://podman.io/get-started>

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 asset (Open API 3)

- z/OS Connect OpenAPI 3 APIs are developed using a z/OS Connect Designer web browser tool to developer 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



mitchj@us.ibm.com

The screenshot shows the "YAML Viewer" section of the JSON formatter. On the left, there is a code editor window displaying the following YAML code:

```
1 openapi: 3.0.0
2 info:
3   description: "CICS Filea Sample VSAM Application"
4   version: 1.0.0
5   title: cscvinc
6 x-ibm-zcon-roles-allowed:
7 - Manager
8 paths:
9 /employee:
10 post:
11   tags:
12     - cscvinc
13   operationId: postCscvincInsertService
14   parameters:
15     - name: Authorization
16       in: header
17       required: false
18       schema:
19         type: string
20   requestBody:
21     content:
22       application/json:
23         schema:
24           $ref: "#/components/schemas/postCscvincInsertService_request"
25           description: request body
26           required: true
27   responses:
28     "200":
29       description: OK
30       content:
31         application/json:
```

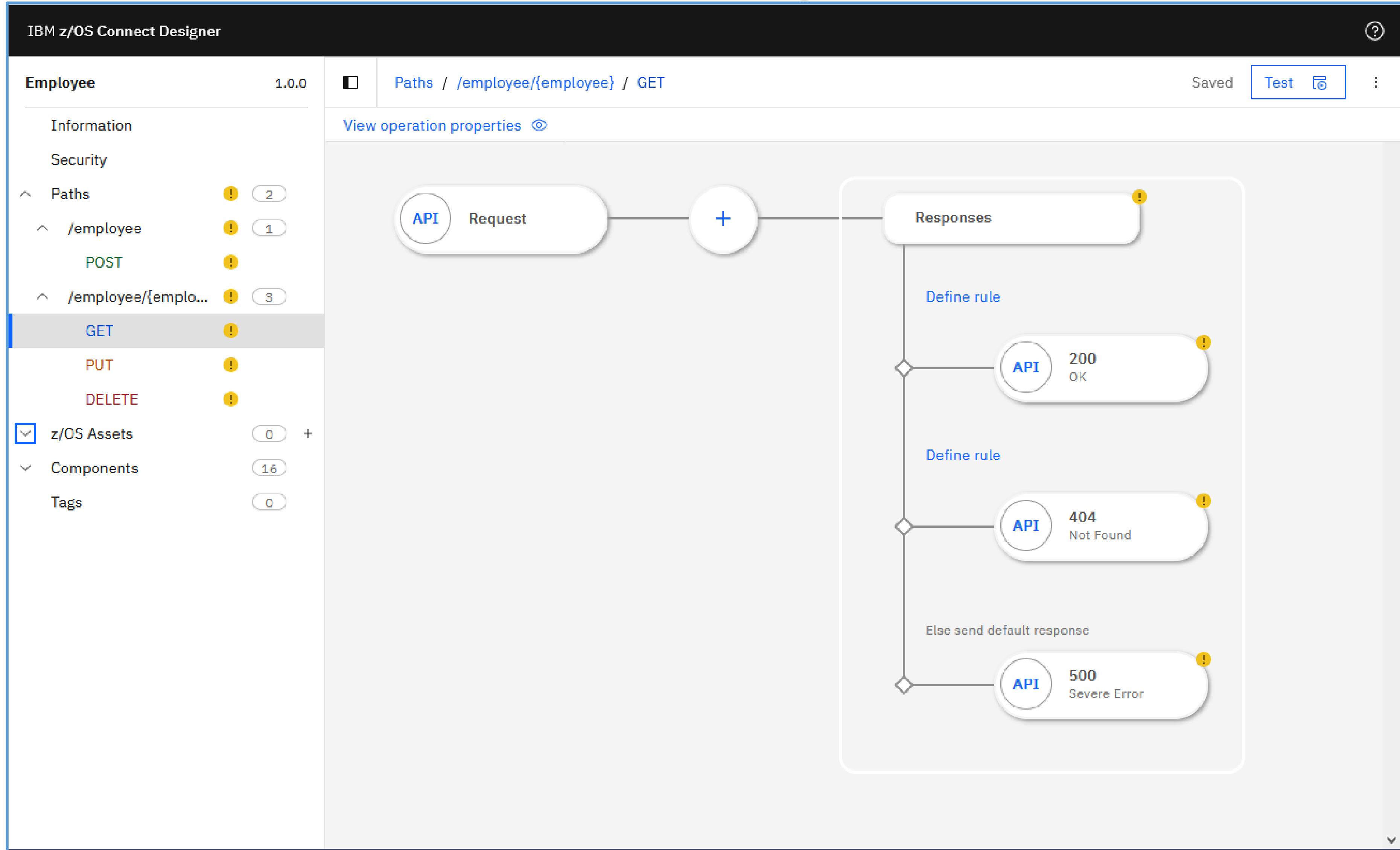
On the right, there are three buttons: "Load Data", "YAML Viewer", and "Download". Below these buttons is a "YAML Tree" panel showing the hierarchical structure of the API definition:

```
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}
```

At the bottom of the interface, there is a copyright notice: © 2017, 2023 IBM Corporation.



Import the description of an API into the Designer (OpenAPI 3)



Describe the asset, a CICS program or IMS transaction (OpenAPI 3)



The screenshot displays two side-by-side configuration interfaces for the **IBM z/OS Connect Designer**.

Left Screen (CICS Channel Program Configuration):

- Title:** IBM z/OS Connect Designer - cscvinc 1.0.0
- Section:** Add z/OS Asset (Step 2 of 5)
- Asset Type:** CICS channel program
- CICS Program Name:** CSCVINC
- Program Language:** COBOL
- CCSID:** 037
- CICS Connection:** cicsConn (selected)
- Optional Configuration:**
 - Transaction ID (optional): Input Transaction ID
 - Transaction ID usage (optional): Select usage
- Buttons:** Previous (disabled), Next

Right Screen (IMS Transaction Configuration):

- Title:** IBM z/OS Connect Designer - Employee 1.0.0
- Section:** Add z/OS Asset (Step 2 of 5)
- Asset Type:** IMS transaction
- Transaction Code:** IVTNO
- Program Language:** COBOL
- IMS Connection:** imsConn (selected)
- Buttons:** Previous, Next



Or a Db2 REST service (OpenAPI 3)

The screenshot shows the IBM z/OS Connect Designer interface. On the left, the 'EmployeesApi' API is defined with various HTTP methods and paths. The 'Paths' section includes a path for '/employees/{id}' with 'GET', 'PUT', and 'DELETE' methods. The 'z/OS Assets' section lists several assets like 'addEmployee', 'deleteEmployee', 'getEmployee', etc. On the right, the 'Add z/OS Asset' dialog is open, showing the 'Import from Db2 service manager' button. Below it, fields for 'Db2 native REST service collection ID' (e.g., SYSIBMSERVICE), 'Db2 native REST service name' (e.g., myService), and 'Db2 native REST service version (optional)' (e.g., V1) are filled. Under 'Import Db2 native REST service request schema', there's a URL input field with 'http://github.com/example/api-docs' and an 'Import file' button. A similar section for 'Import Db2 native REST service response schema' is also present. At the bottom, 'Previous' and 'Next' buttons are visible. To the right of the main window, a separate 'Import Db2 native REST service' dialog is displayed, listing 12 found services. The table has columns for Service name, Version, Collection ID, Path, Description, and Status. Services listed include 'addEmployee', 'deleteEmployee', 'getEmployee', 'getEmployees', 'updateEmployee', 'displayEmployee', 'insertEmployee', 'selectByDepartments', and 'selectByRole', all marked as 'Available'. The 'Import' button is at the bottom right of this dialog.

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



Map the API's method and request with the resource's input (OpenAPI 3)

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

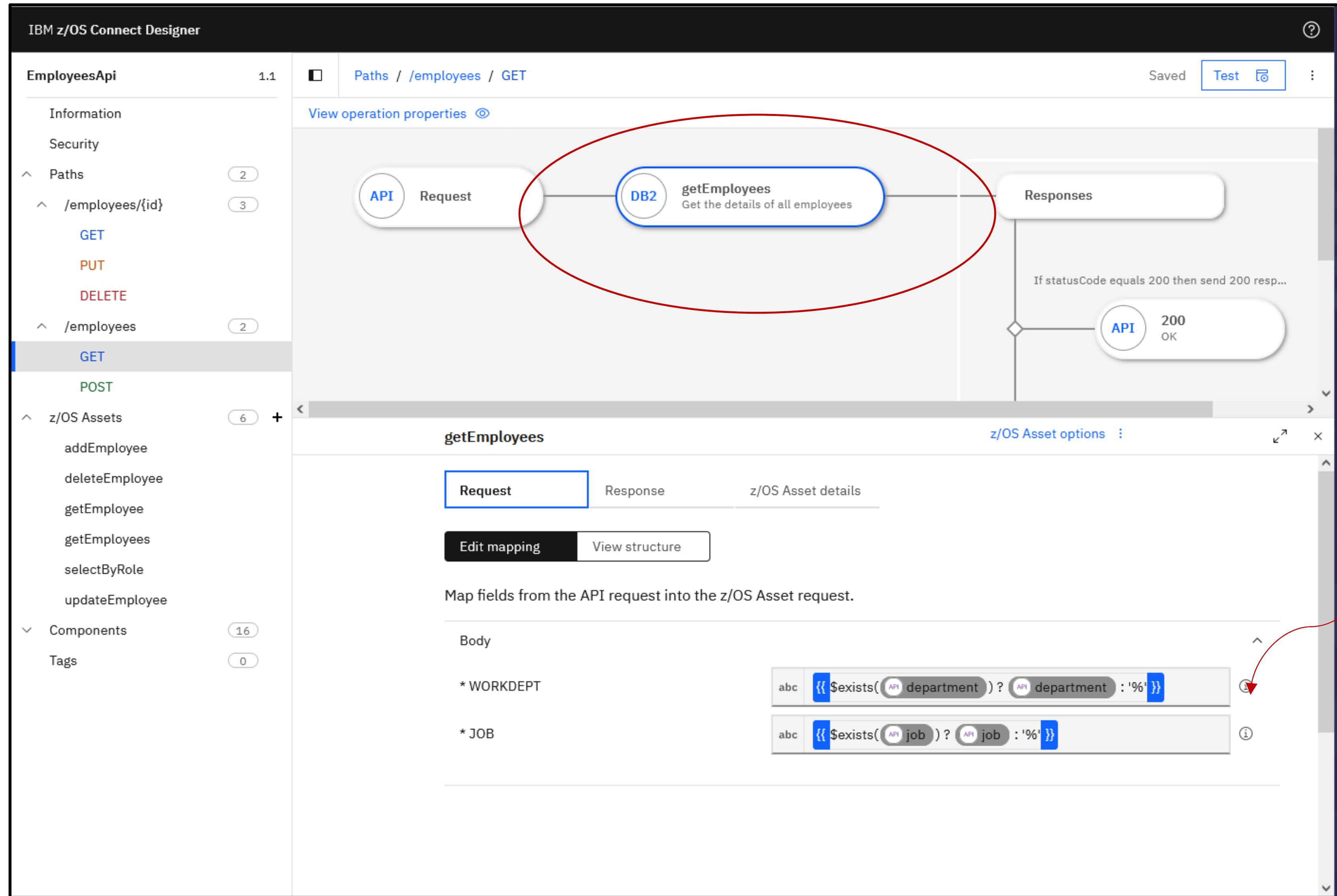
Request-Container

- ACTION abc S
- USERID abc
- FILEA-AREA

 - STAT abc
 - NUMB abc employee

mitchj@us.ibm.com

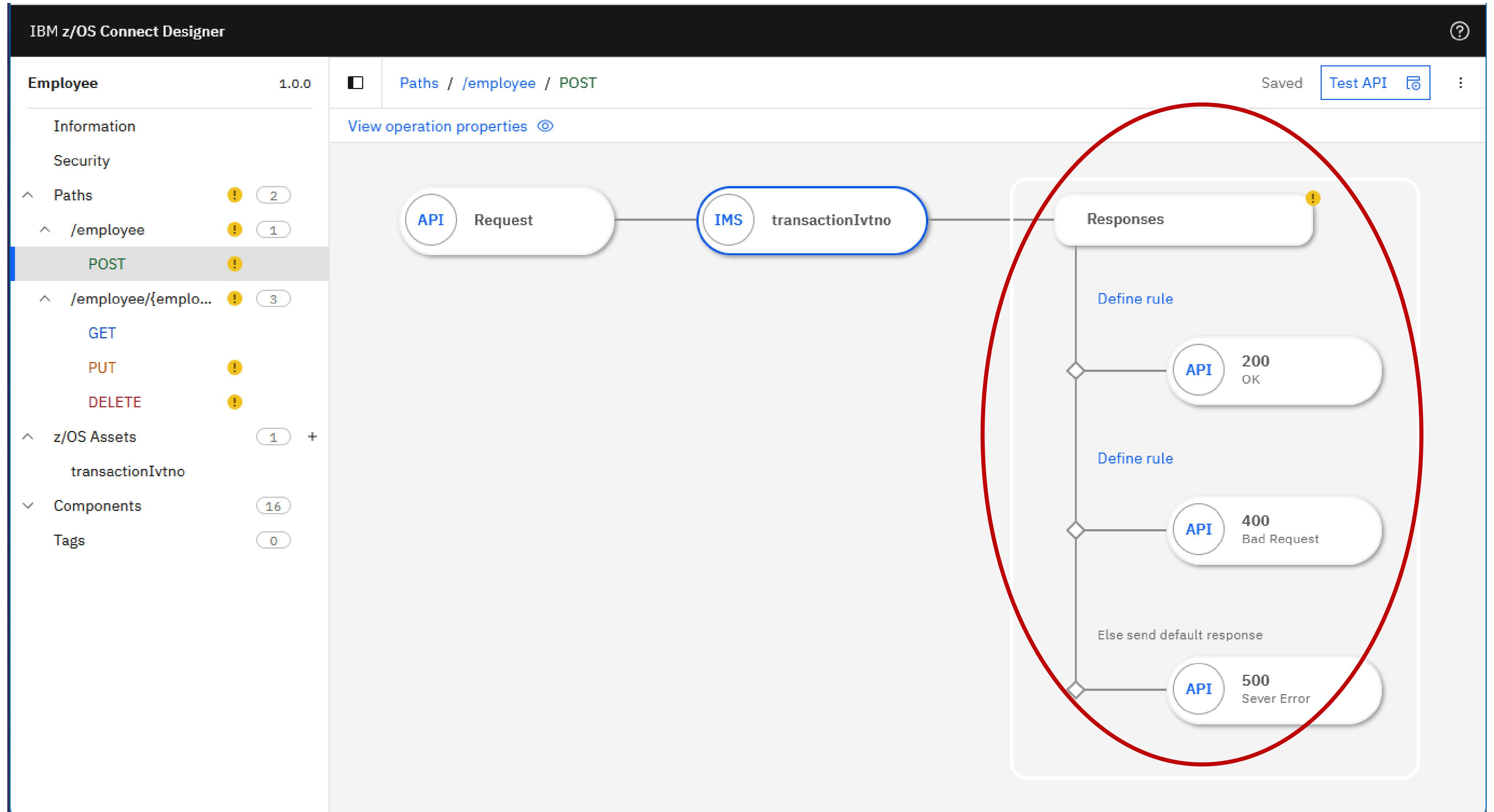
Map the API's request with a Db2 REST service's request message (OpenAPI 3)



JSONata example



Map the resource's results to the API's response message (OpenAPI 3)



Map the resource's response to the API's response (200)(OpenAPI 3)

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

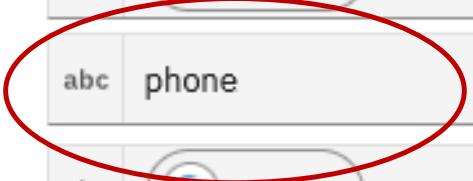
abc  ADDRX

abc  phone

abc  DATEX

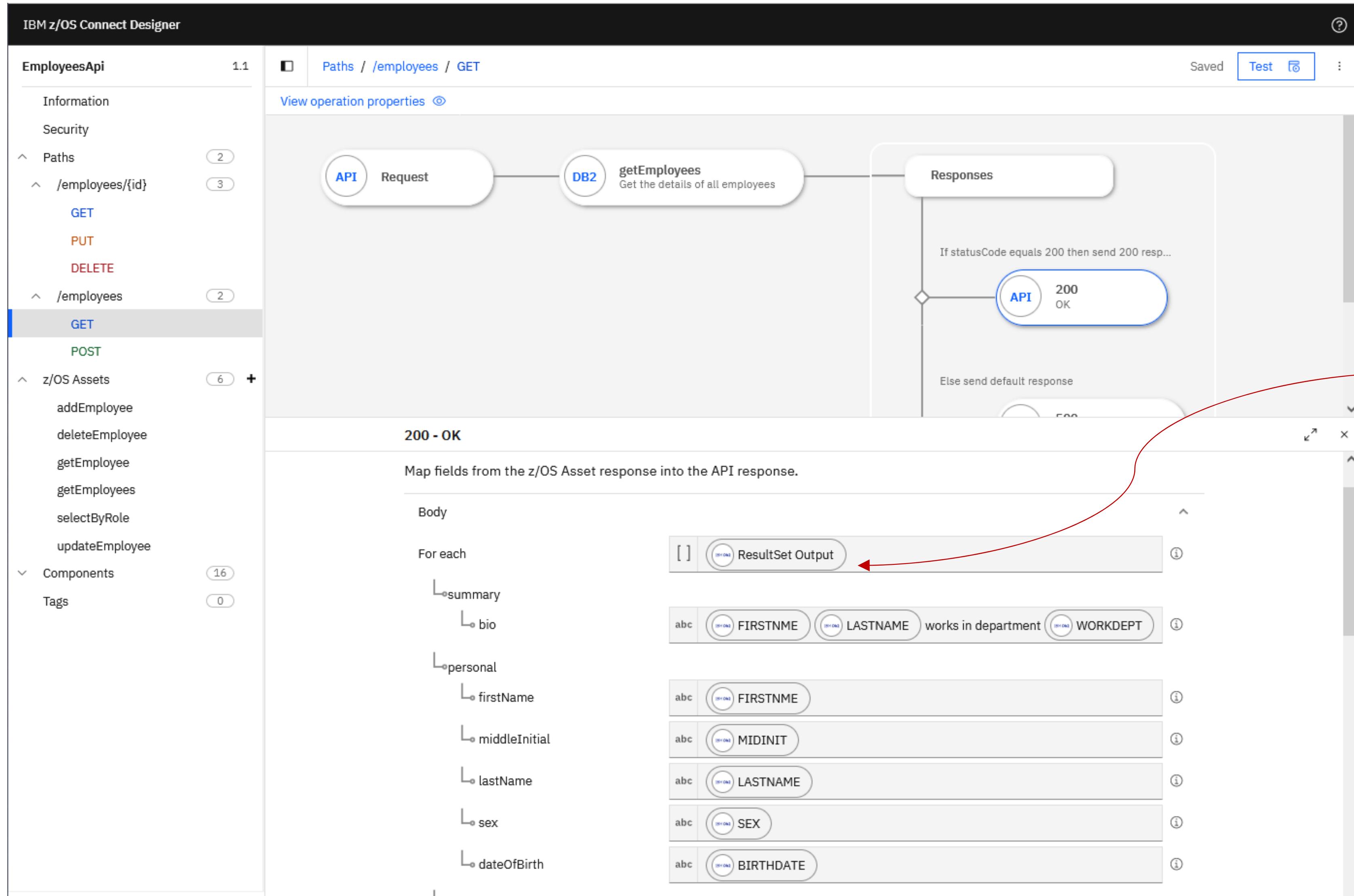
abc  AMOUNT

abc  COMMENT



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

Map the Db2's response to the API's response message (OpenAPI 3)



ResultSet Output is a list or array of 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 Saved Test : [View operation properties](#)

Information Security Paths /employees/{id} 2
/employees 3
GET PUT DELETE
/employees 2
z/OS Assets 6 +
Components 16
Tags 0

View operation properties

```
graph TD; Start(( )) --> API1((API))["200 Updated"]; API1 --> Decision1{ }; Decision1 --> API2((API))["404 Not Found"]; Decision1 --> API3((API))["500 Internal Server Error"]; Decision1 --> ElseText[Else send default response]; IfText[If "Update Count" equals 0 then send 404 res...]
```

404 - Not Found

Edit mapping View structure

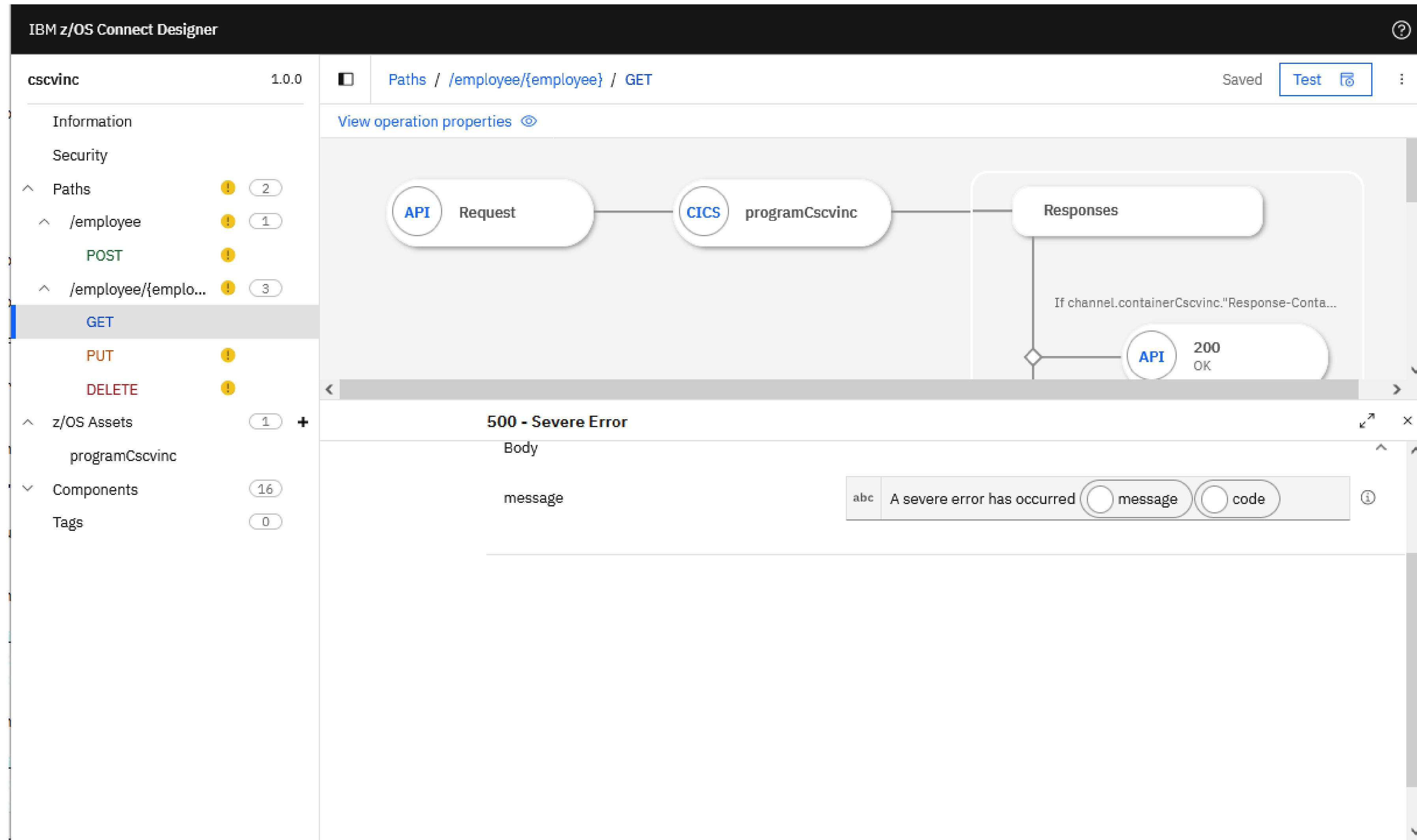
Map fields from the z/OS Asset response into the API response.

Body

message abc Employee not found ①



z/OS Connect Designer for OpenAPI 3 (500)

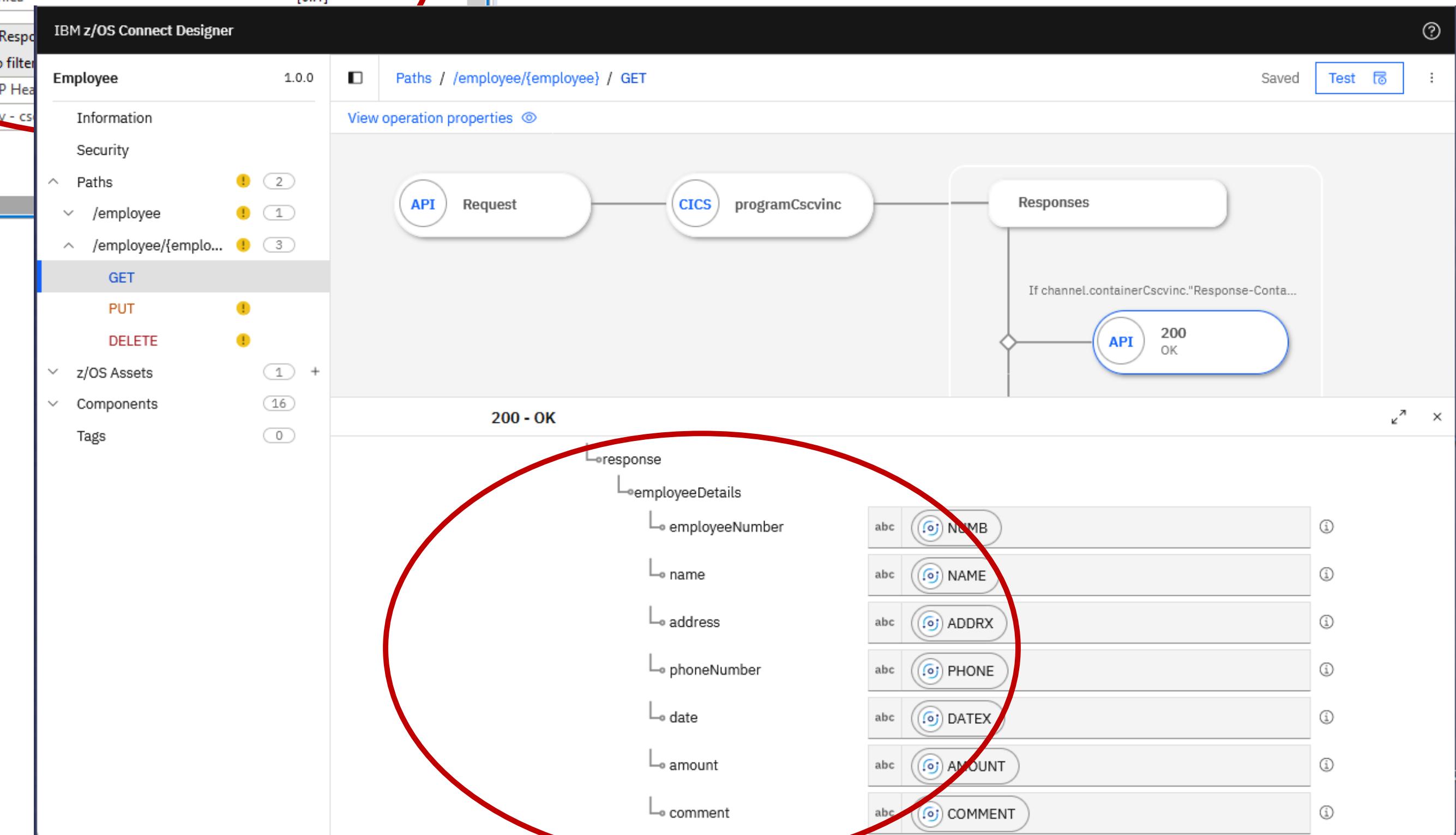




Contrast the OpenAPI 2 versus an OpenAPI 3 Response Message

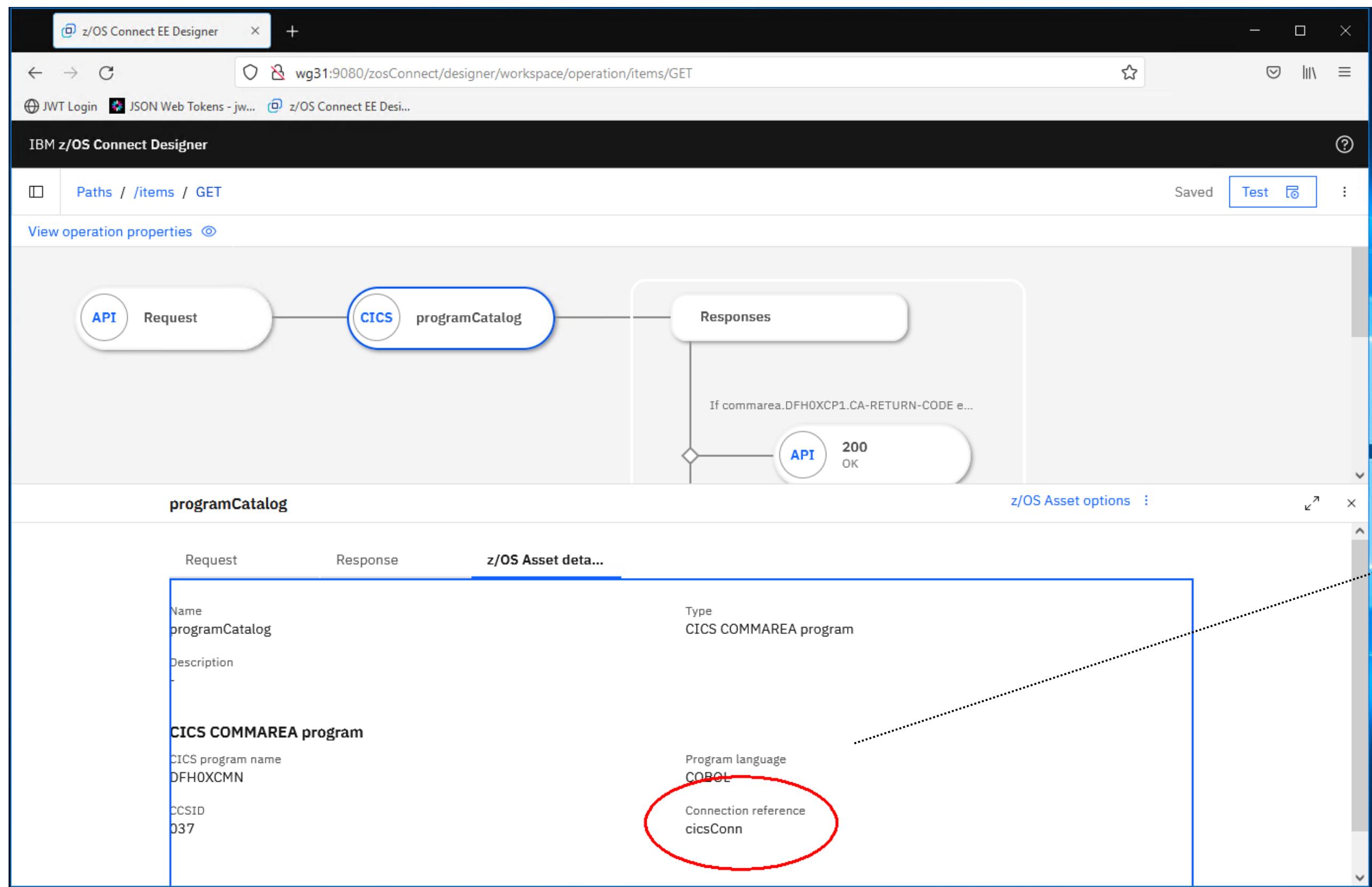
The screenshot shows two side-by-side response message structures. The left one is for an OpenAPI 2 asset, and the right one is for an OpenAPI 3 asset. Both structures include a 'Body' section with various fields like 'cscvincContainer', 'response', 'CEIBRESP', 'CEIBRESP2', 'USERID', and 'filea'. A red oval highlights the 'Body' sections in both.

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



Web browser - z/OS Connect Designer –
The contents of the API's response
message are must be mapped to the z/O
asset's response and JSONata coding, see
URL <https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=concepts-what-is-jsonata> for more information on
JSONata.

Server XML - Accessing a CICS program using IPIC (OpenAPI 3)



The screenshot shows the 'Server Config' interface with a tab for 'cics.xml'. It has two tabs: 'Design' (selected) and 'Source'. The 'Source' tab displays the following XML code:

```
1<server description="CICS IPIC connections">
2
3<!-- Enable features -->
4<featureManager>
5  <feature>zosconnect:cics-1.0</feature>
6</featureManager>
7
8<zosconnect_cicsIpicConnection id="cicsConn" host="${CICS_HOST}"
9  port="${CICS_PORT}" />
10
11</server>
12
```

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

mitchj@us.ibm.com

© 2017, 2023 IBM Corporation
Slide 101



Server XML - Accessing an IMS transaction(OpenAPI 3)

The screenshot shows the z/OS Connect Designer interface. On the left, the navigation pane lists 'Employee' resources: 'Information', 'Security', 'Paths' (with items '/employee', '/employee/{employee}', and '/employee/{employee}'), 'GET', 'PUT', 'DELETE', 'z/OS Assets' (with item 'transactionIvtno'), and 'Components' (with 16 items). The 'transactionIvtno' asset is selected. The main panel displays an API path: 'Paths / /employee/{employee} / GET'. This path consists of an 'API Request' node connected to an 'IMS transactionIvtno' node. The 'transactionIvtno' node has a 'z/OS Asset options' tab selected, showing details: Name 'transactionIvtno', Type 'IMS transaction', Description '-', Transaction code 'IVTNO', Program language 'COBOL', and Connection profile 'imsConn' (which is circled in red).

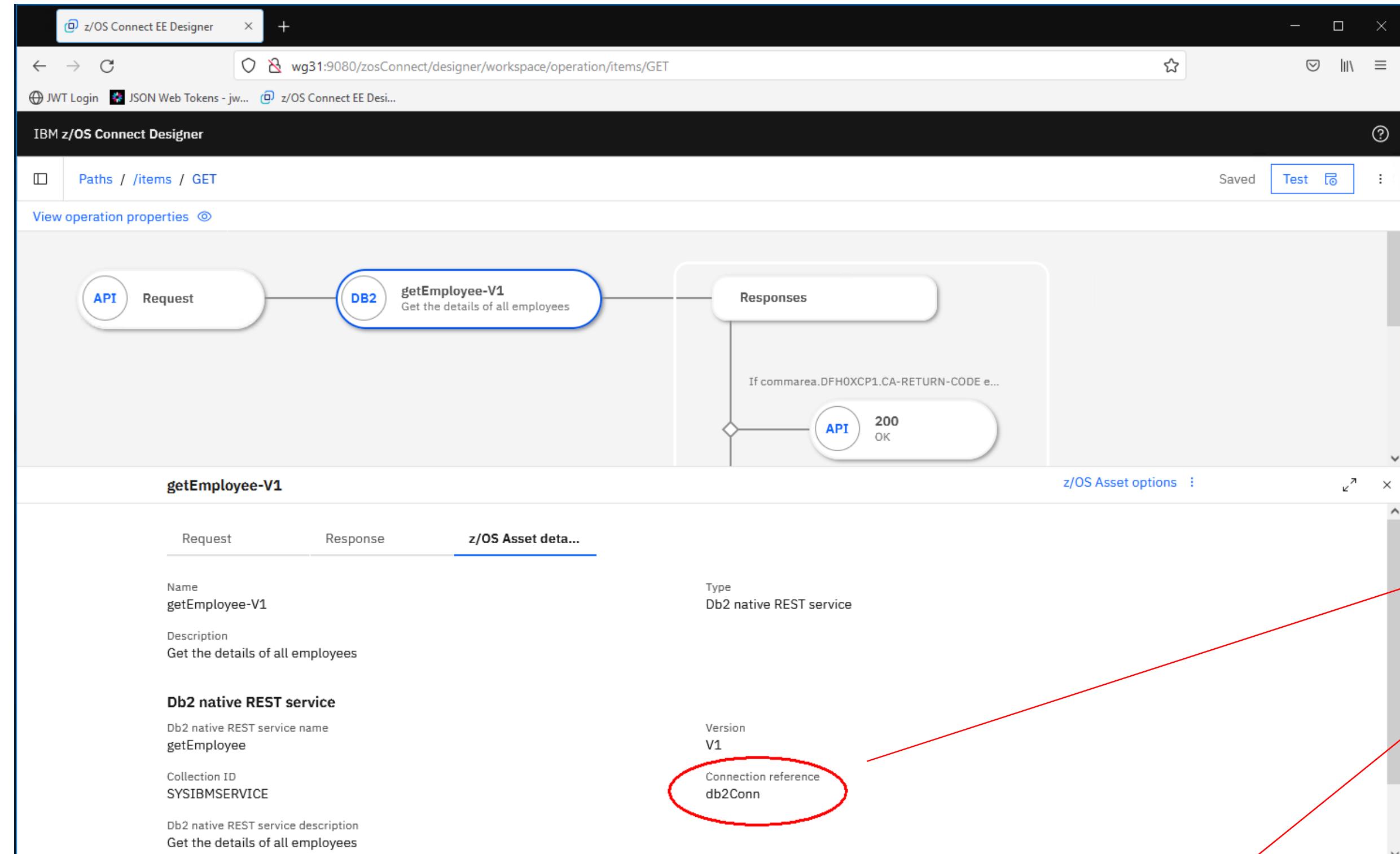
On the right, a modal window titled 'Server Config' is open for the file 'ims.xml'. It has tabs for 'Design' and 'Source'. The 'Source' tab shows the XML configuration:

```
<?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>
```

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



Server XML - Accessing a Db2 REST service (OpenAPI 3)



Server Config

db2.xml

Design **Source**

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <server description="Db2 Connections">
3   <featureManager>
4     <feature>zosconnect:db2-1.0</feature>
5   </featureManager>
6   <zosconnect_credential user="${DB2_USERNAME}">
7     password="${DB2_PASSWORD}" id="commonCredentials" />
8   <zosconnect_db2Connection id="db2Conn" host="${DB2_HOST}">
9     port="${DB2_PORT}" credentialRef="commonCredentials" />
10 </server>
11

```

Define connections to Db2 using variables defined in bootstrap.properties file

```

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.



EJB roles for z/OS Connect (OpenAPI 3)

```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />  
  
<webApplication id="CatalogManager" location="${server.config.dir}/apps/api.war" contextRoot="catalog" 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
```

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*

*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*.



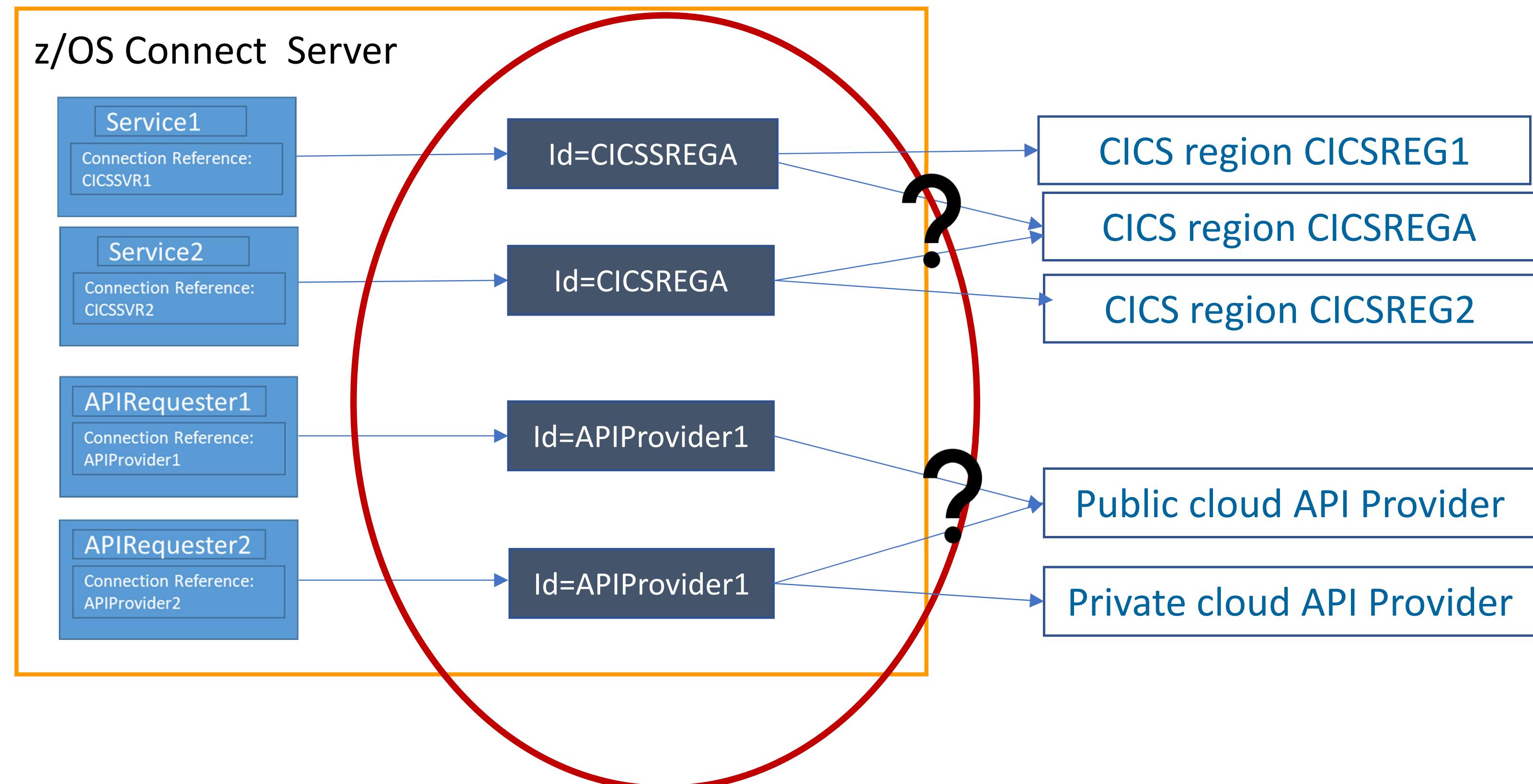
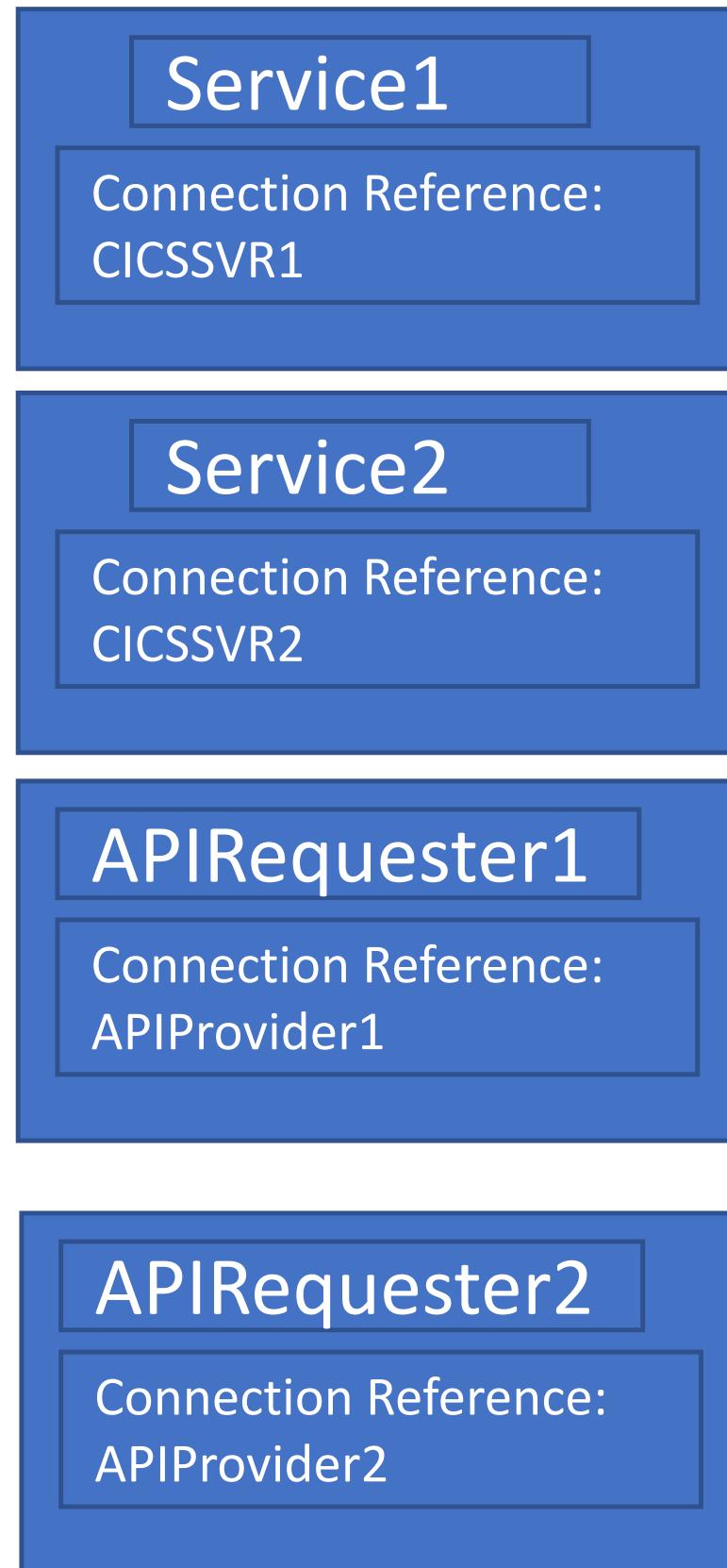
Connection Reference Consideration

Carefully consider the names used for connections



Use naming conventions for connection references

Use application meaningful names or an extendable convention for connection reference names





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, Help, Home, Workspaces, Reports, Explore, and a search bar. To the right of the search bar are icons for cloud, plus, gear, and a bell, followed by an Upgrade button. Below the navigation is a list of recent requests. One request is selected, showing a GET method to https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111. The request tab is active, showing Params, Authorization (with a green dot), Headers (10), Body, Pre-request Script, Tests, and Settings. The Body tab is selected, displaying a JSON response. The response body is as follows:

```
1  {
2     "cscvincSelectServiceOperationResponse": {
3         "cscvincContainer": {
4             "response": {
5                 "CEIBRESP": 0,
6                 "CEIBRESP2": 0,
7                 "USERID": "CICSUSER",
8                 "filea": {
9                     "employeeNumber": "111111",
10                    "name": "C. BAKER",
11                    "address": "OTTAWA, ONTARIO",
12                    "phoneNumber": "51212003",
13                    "date": "26 11 81",
14                    "amount": "$0011 .00"
15                }
16            }
17        }
18    }
19 }
```

Below the response, there are tabs for Body, Cookies (1), Headers (8), and Test Results. The status bar at the bottom shows Status: 200 OK, Time: 205 ms, Size: 899 B, and Save Response. The bottom navigation bar includes Find and Replace, Console, Bootcamp, Runner, Trash, and a settings icon.

mitchj@us.ibm.com

<https://www.postman.com/downloads/>



Testing an API with the API Explorer (zCEE V3.0.48)

The screenshot shows the IBM API Explorer interface. On the left, a sidebar lists various Liberty REST APIs: cscvinc, db2employee, filemgr, imsPhoneBook, jwtlvpDemoApi, 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 PUT operation is highlighted with a yellow background. On the right, a detailed view of the cscvinc API is shown. The 'Request URL' field contains the curl command: curl -X GET --header 'Accept: application/json' --header 'Authorization: Basic RnJlZDpmcmVk' 'https://mpz3.washington.ibm.com:9443/cscvinc/employee/111111'. The 'Response Body' field displays a JSON response for an employee with ID 111111, including details like name, address, and phone number. The 'Response Code' field shows 200, and the 'Response Headers' field shows content-language: en-US, content-length: 269, and content-type: application/json; charset=UTF-8.



Testing an API using the cURL tool

```
c:\ Command Prompt
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>



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=sto+  
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%.aar --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



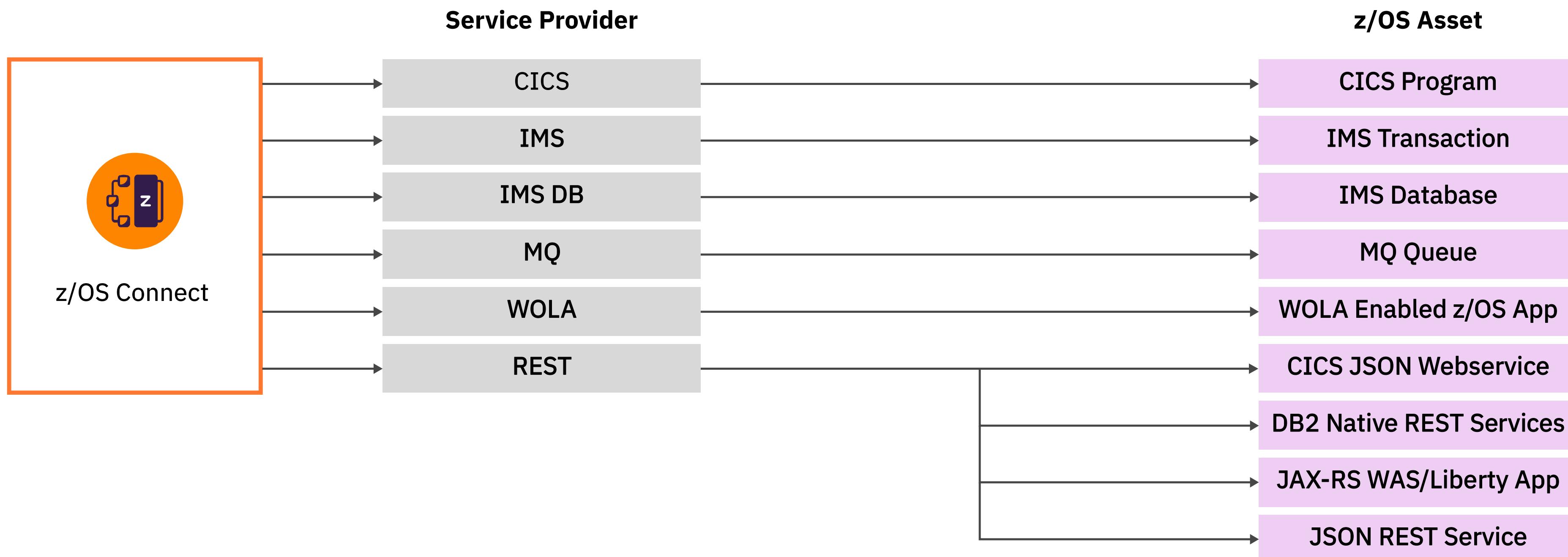
/miscellaneousTopics

performance, high availability, Liberty



What assets can z/OS Connect EE map to?

And which service provider could I use?

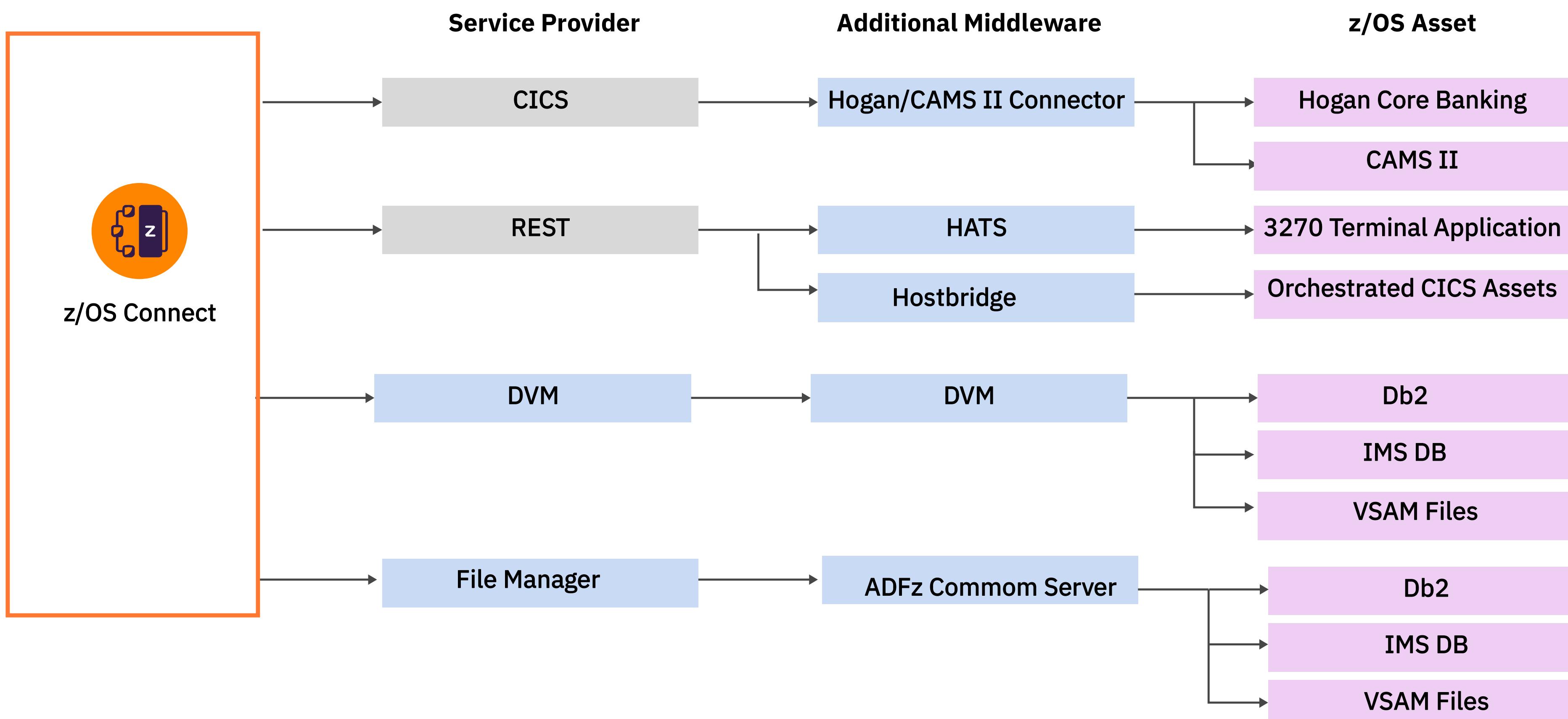


The core **service providers** included with z/OS Connect EE provide API access to a wide range of z/OS assets.



Additional Middleware

Additional value from the ecosystem



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



API Policies

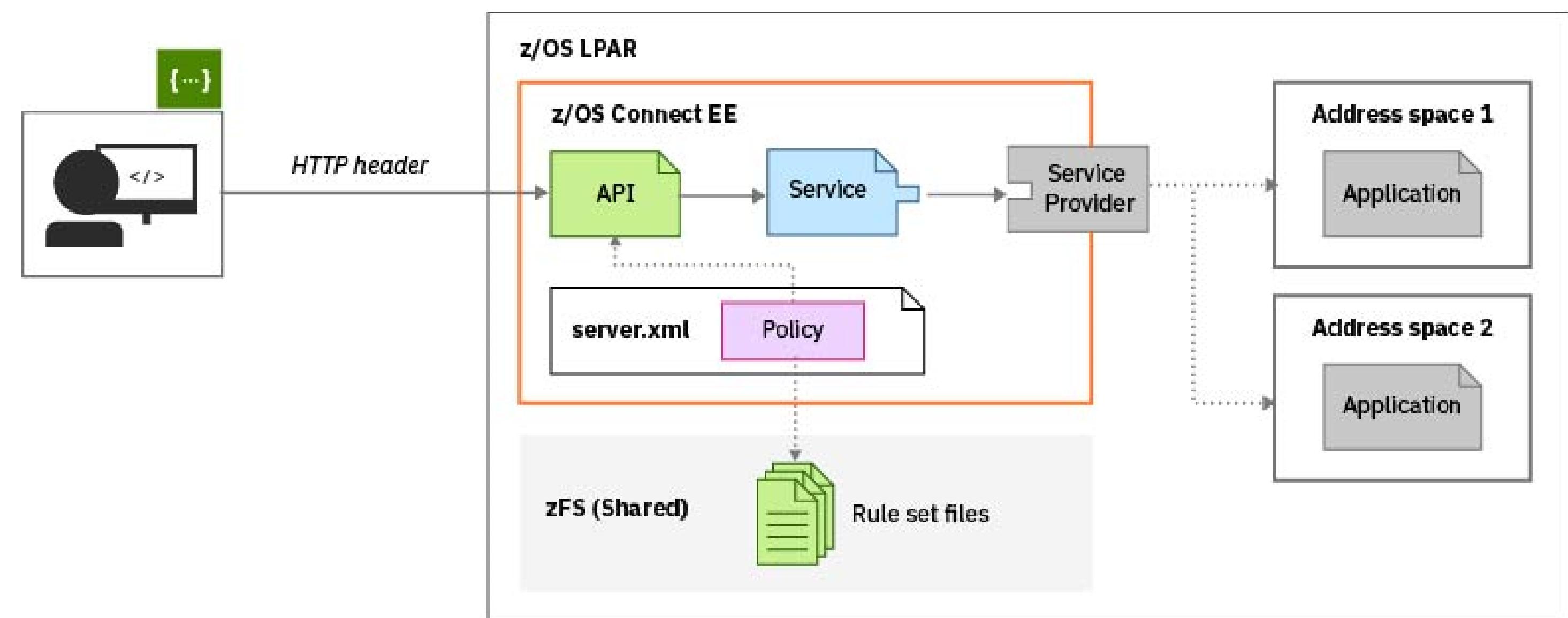
- HTTP header properties can be used to select alternative for IMS (V3.0.4) , CICS (V3.0.10), Db2 (V3.0.36) or MQ (V3.0.39)
- Policies can be configured globally for every API in the server or for individual APIs (V3.0.11)

CICS attributes
• cicsCcsid
• cicsConnectionRef
• cicsTransId

IMS attributes
• imsConnectionRef
• imsInteractionRef
• imsInteractionTimeout
• imsLtermOverrideName
• imsTranCode
• imsTranExpiration

Db2 attributes
• db2ConnectionRef
• db2CollectionID

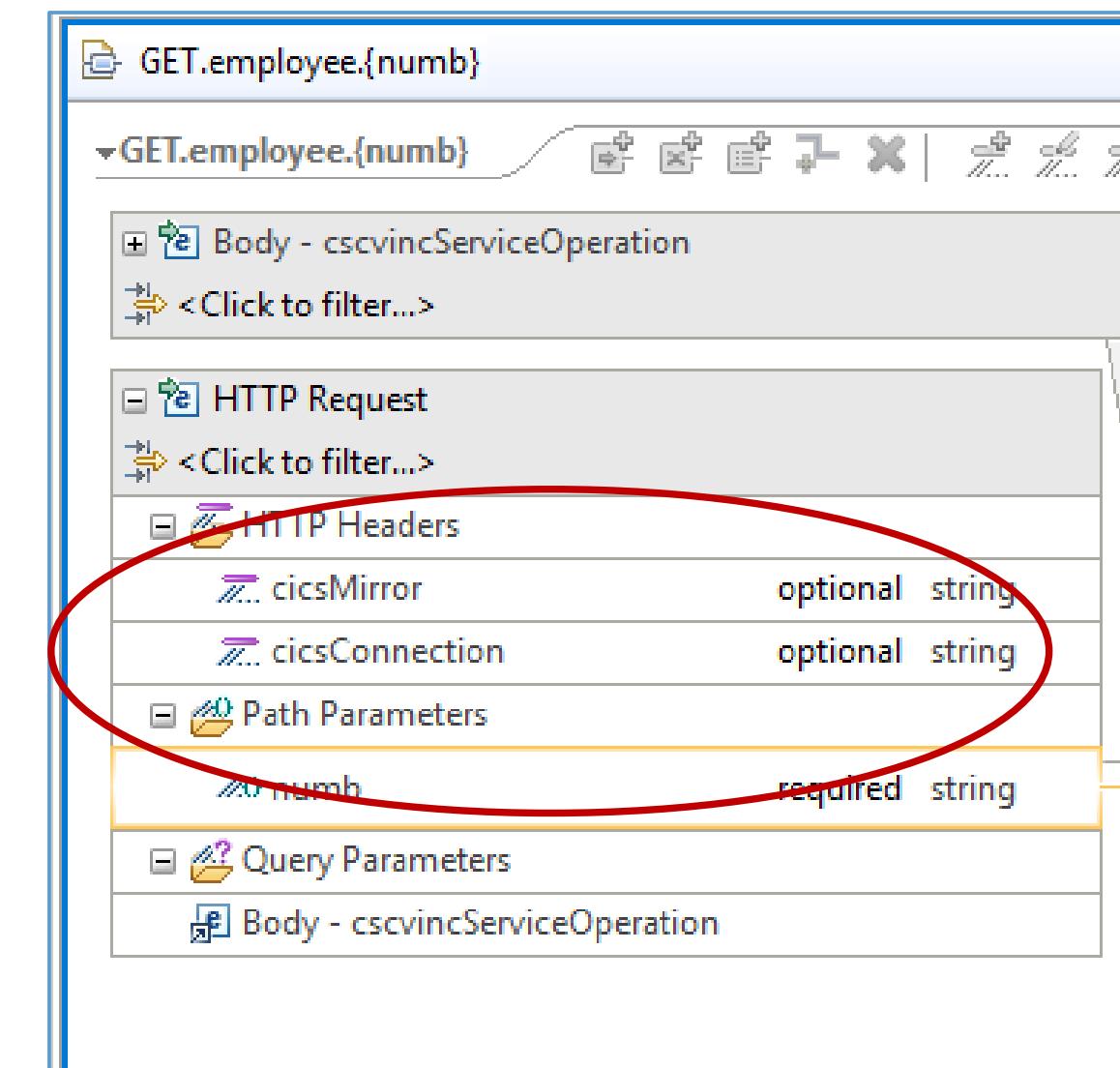
MQ attributes
• mqConnectionFactory
• mqDestination
• mqReplyDestination





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

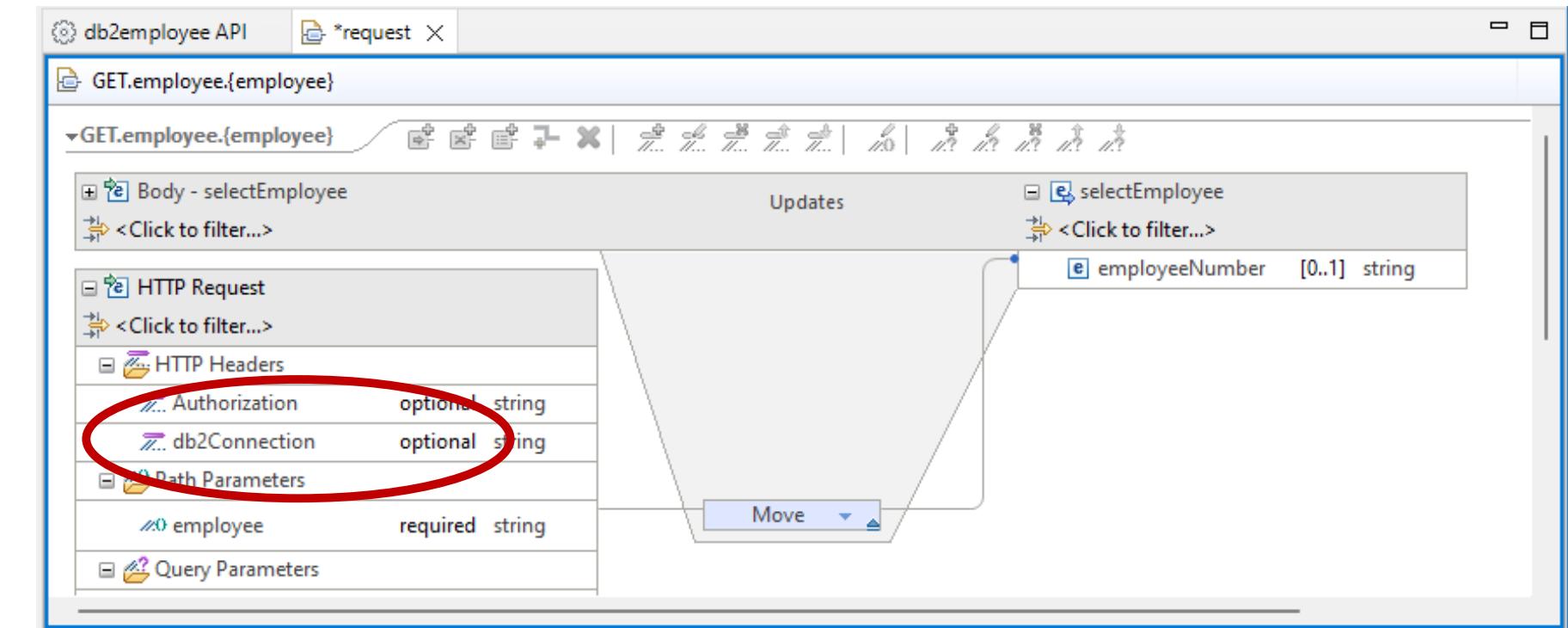
```
curl -X GET --header 'Accept: application/json' --header 'cicsMirror: MIJO' --header 'cicsConnection: cscvinc' 'https://m...
```

*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"/>

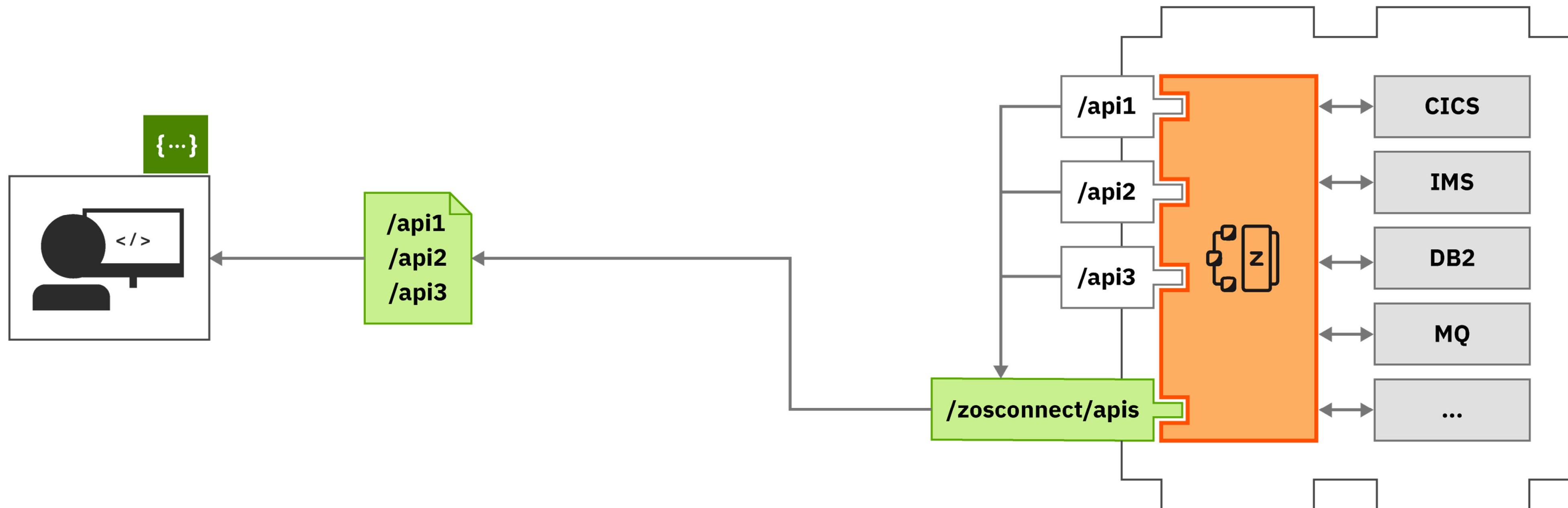
<zosconnect_zosConnectServiceRestClientConnection id="user1"
  sslCertsRef="user1SSLSettings" host="wg31.washington.ibm.com" port="2445" />
<ssl id="user1SSLSettings" keyStoreRef="Db2KeyStore" trustStoreRef="Db2KeyStore" clientKeyAlias="USER1"/>

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

Requires APAR PH53162



API Documentation



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



/security

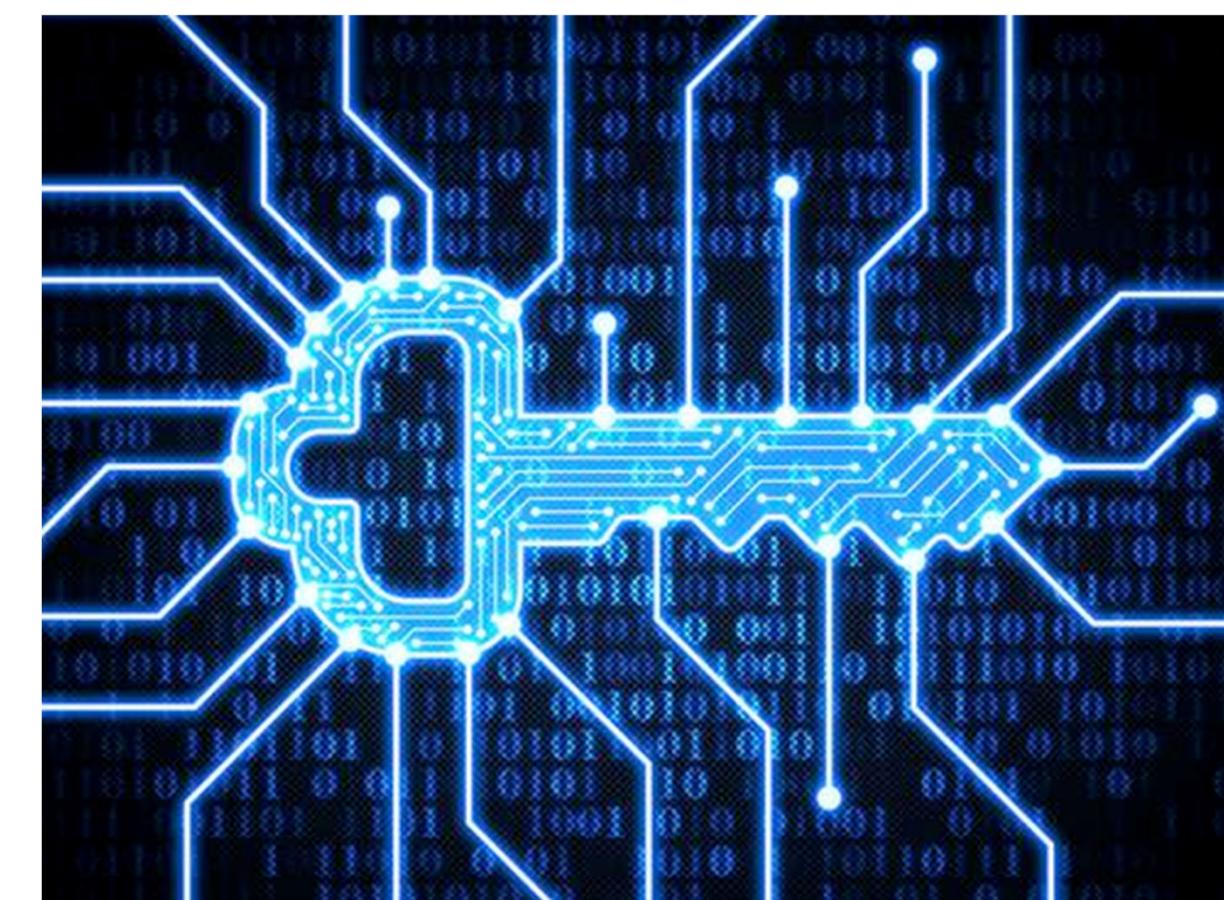
How is security implement?



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.

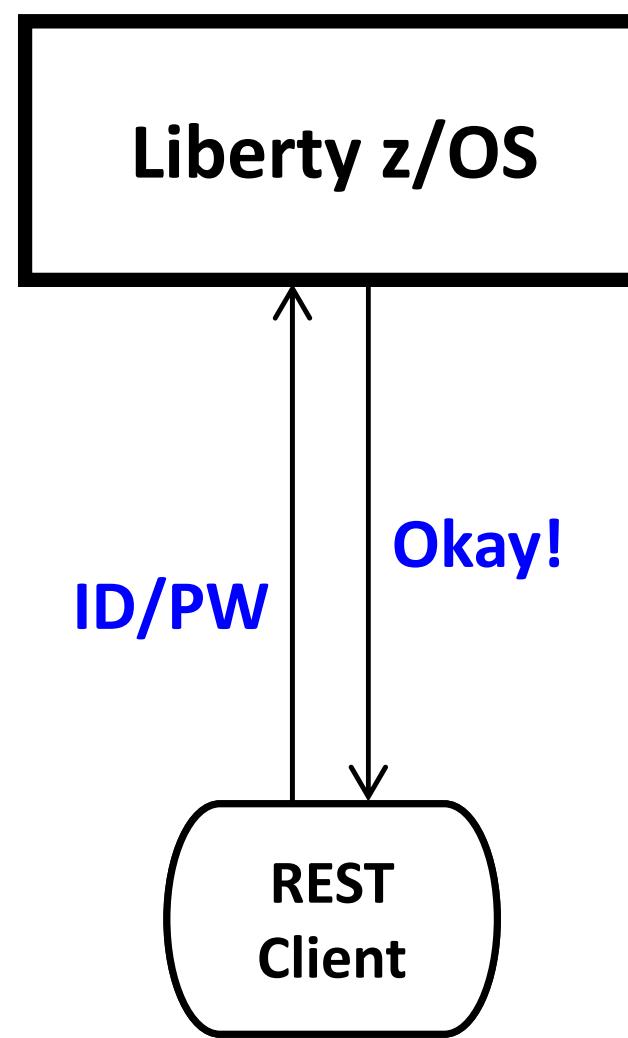




API Provider Authentication

Several different ways this can be accomplished:

Basic Authentication



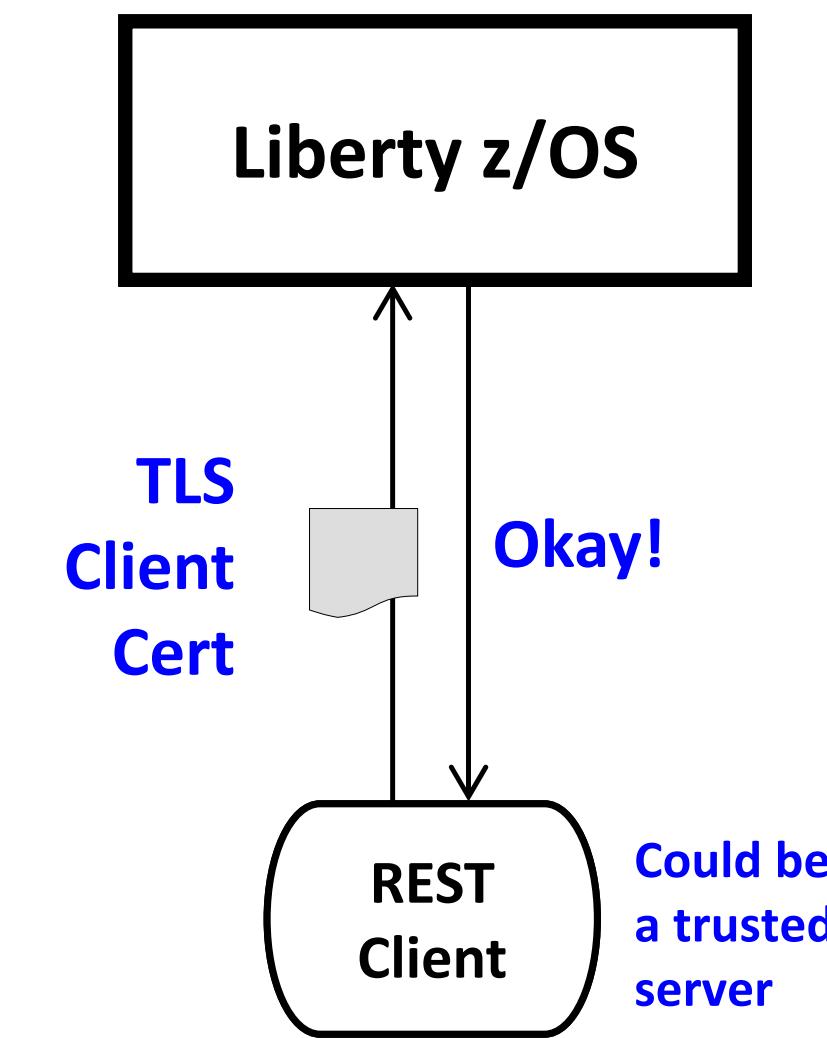
Server prompts for ID/PW

Client supplies ID/PW or
ID/Passticket

Server checks registry:

- Basic (server.xml)
- LDAP
- SAF

Client Certificate



Server prompts for cert.

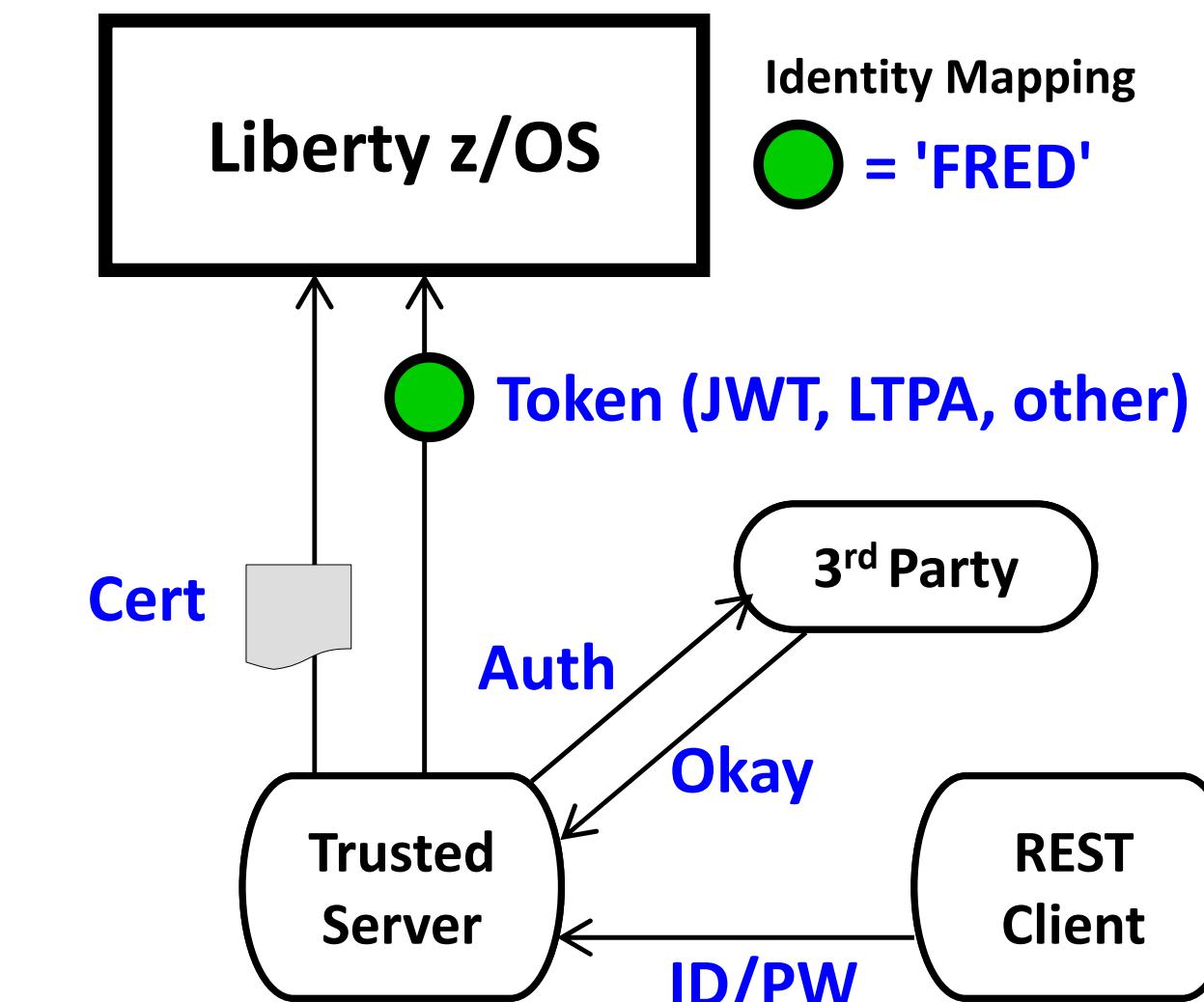
Client supplies certificate

Server validates cert and
maps to an identity

Registry options:

- LDAP
- SAF

Third Party Authentication



Client authenticates to 3rd party sever

Client receives a trusted 3rd party token

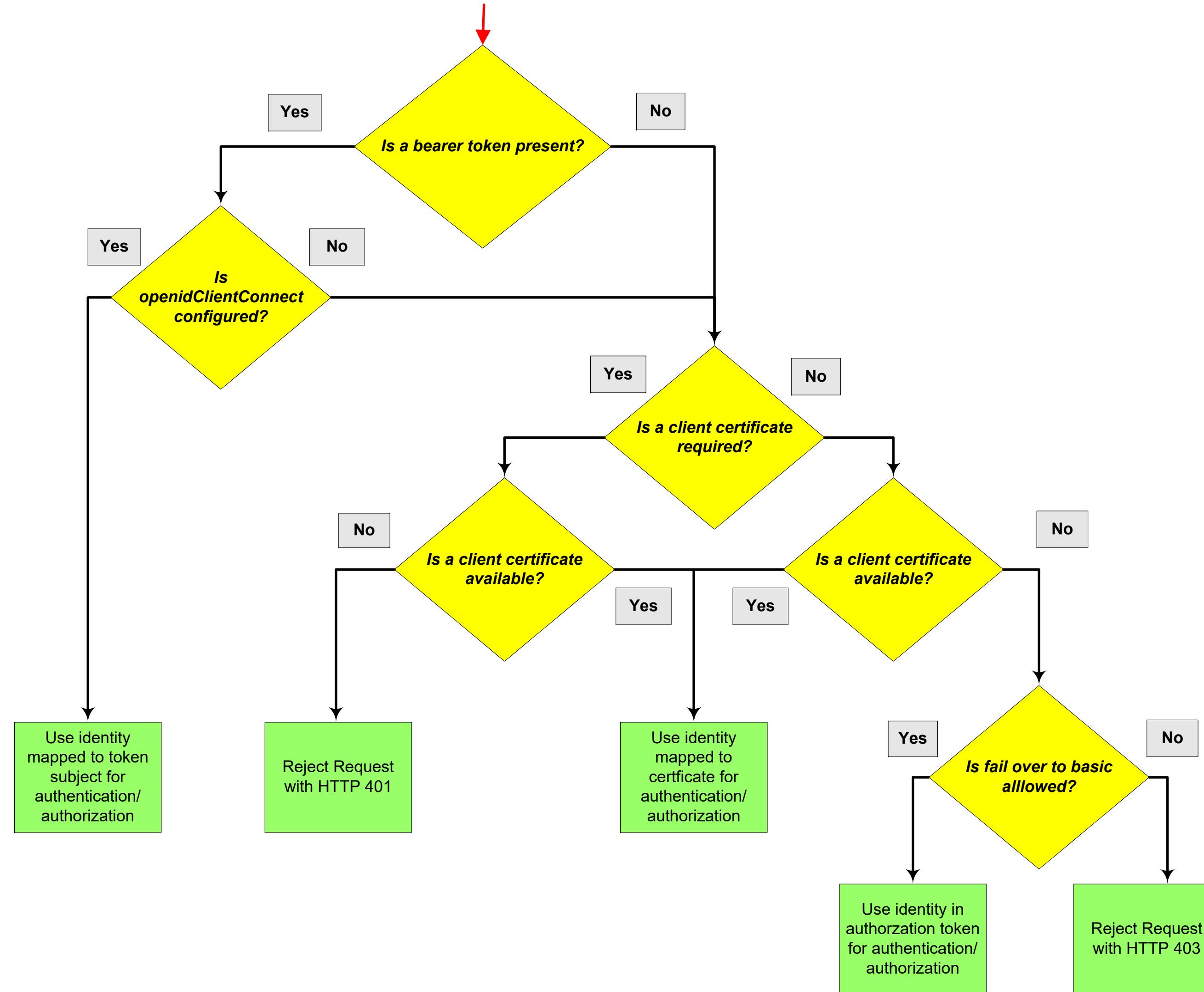
Token flows to Liberty z/OS and is
mapped to an identity

Registry options:

- LDAP
- SAF

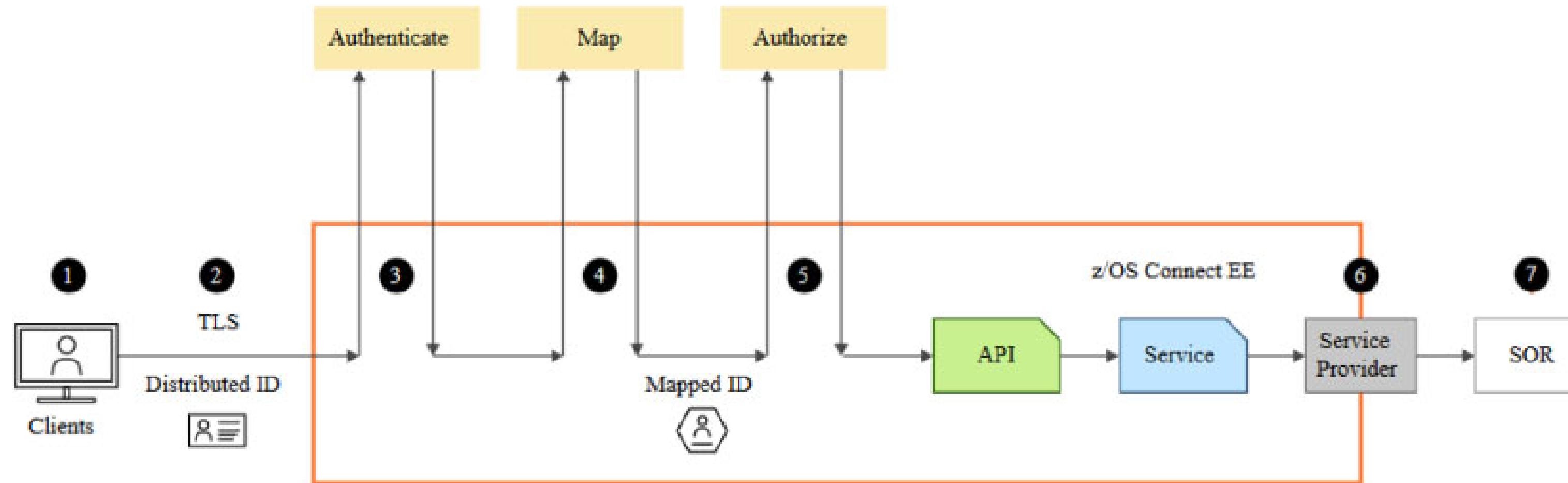


Authentication credential precedence order for determining authorization identity





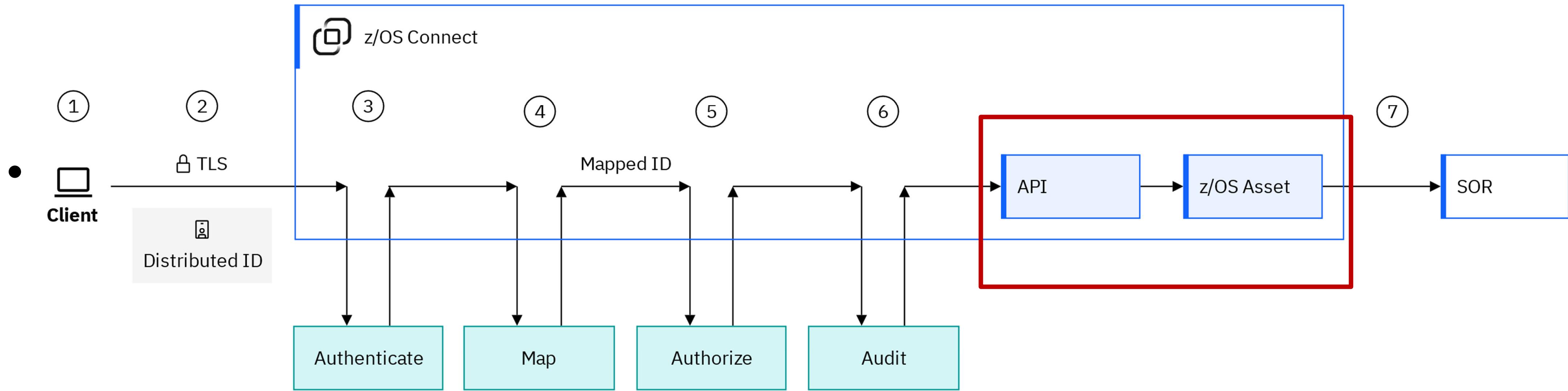
Typical z/OS Connect EE API Provider security flow



1. The credentials provided by the client
2. Secure the connection to the z/OS Connect EE server
3. Authenticate the client. This can be within the z/OS Connect EE 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 EE 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 a typical z/OS Connect EE API Provider security flow - inbound (OpenAPI 3)



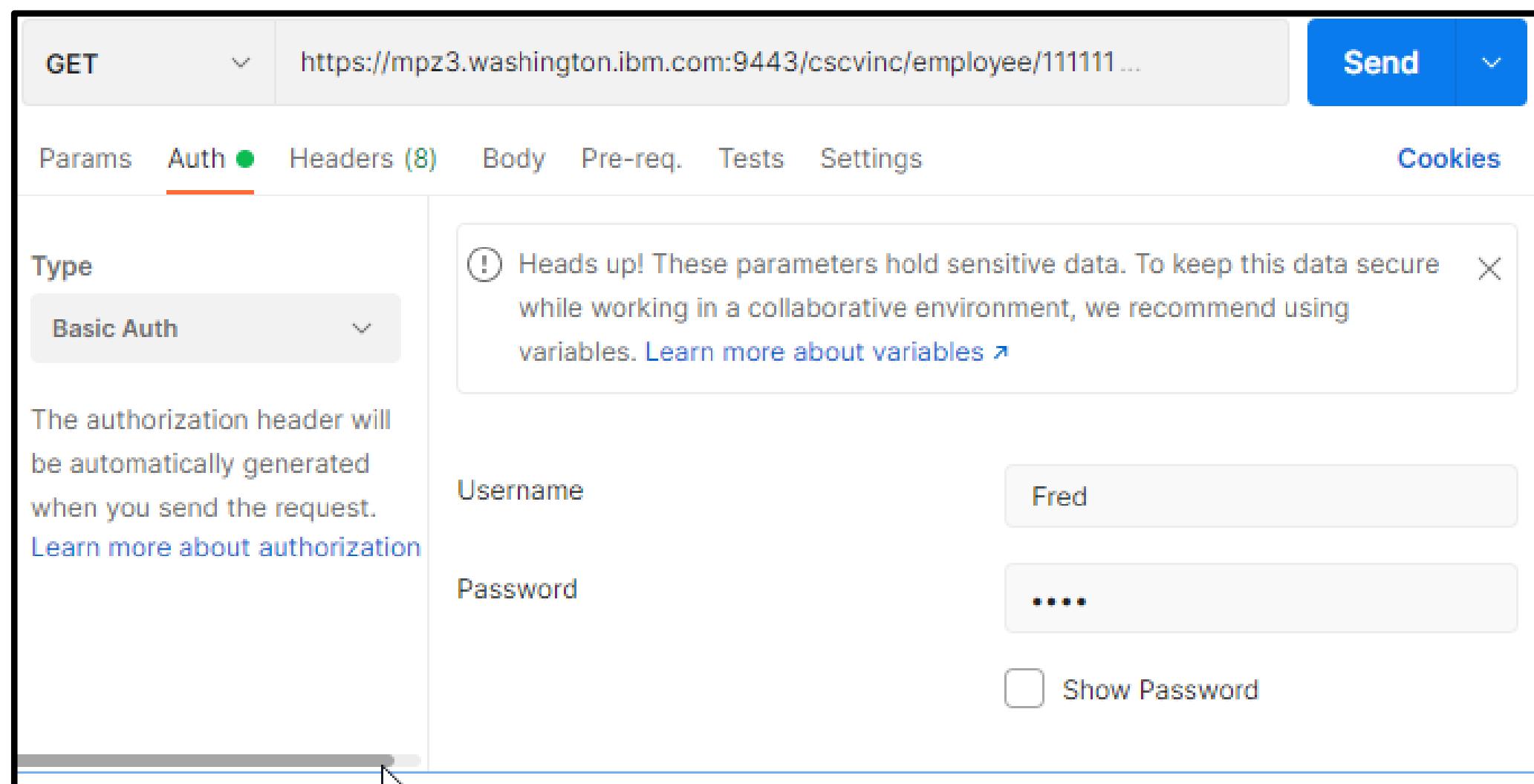
The flow includes the following security steps that can be performed by IBM z/OS Connect. The credentials are provided by the client. These can be a user ID and password, a JWT, or a TLS certificate.

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 the required 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.

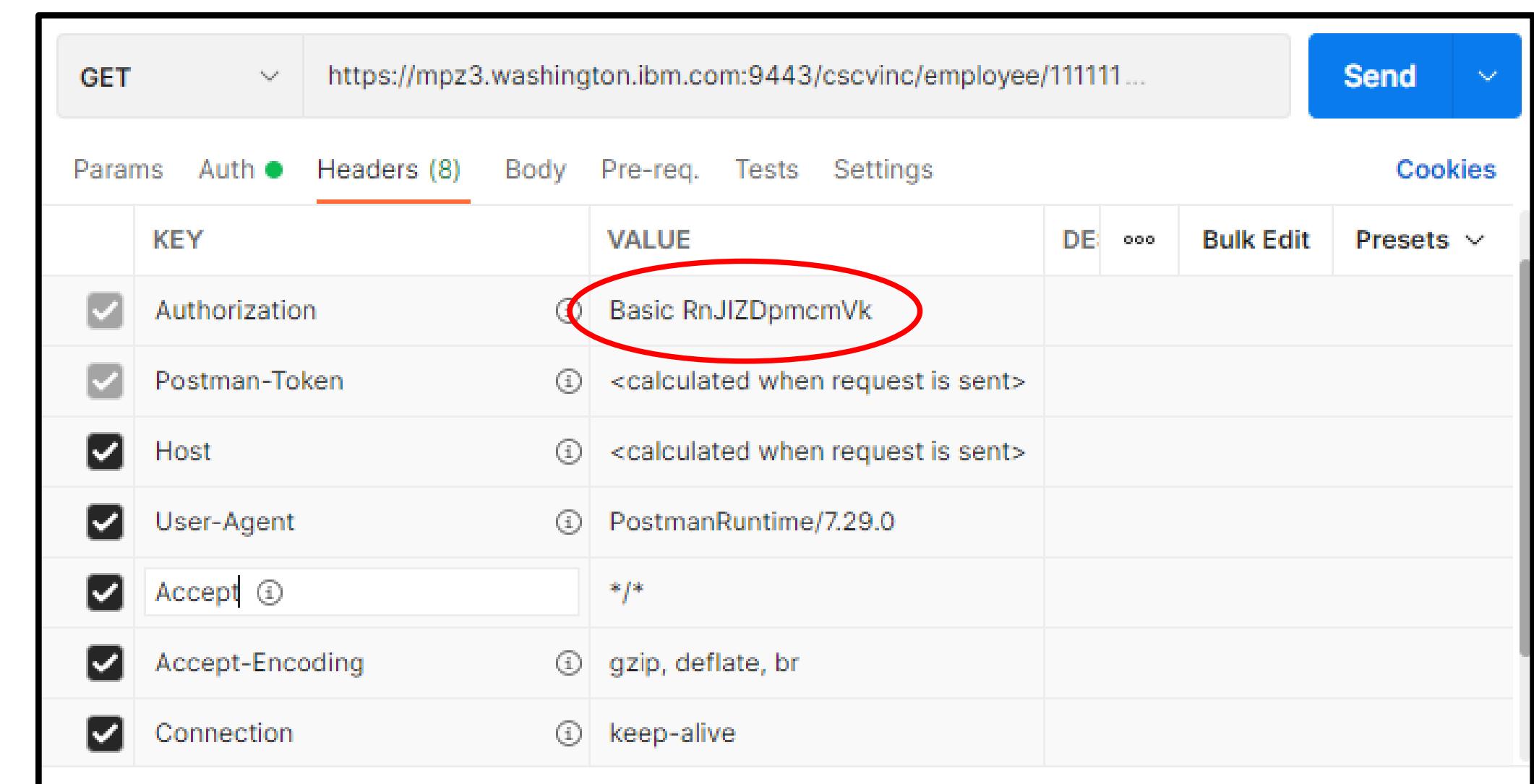
Basic authentication – Where the client provides an identity and password



- When sending a request to a Liberty server, 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.
 - 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. Under 'Type', 'Basic Auth' is chosen. A note says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' The 'Username' field contains 'Fred' and the 'Password' field contains '....'. There is a 'Show Password' checkbox.



The screenshot shows the Postman interface with the 'Headers' tab selected. It lists several headers: Authorization, Postman-Token, Host, User-Agent, Accept, Accept-Encoding, and Connection. The 'Authorization' header value is highlighted with a red oval: `Basic RnJIZDpmcmVk`.

KEY	VALUE
Authorization	Basic RnJIZDpmcmVk
Postman-Token	<calculated when request is sent>
Host	<calculated when request is sent>
User-Agent	PostmanRuntime/7.29.0
Accept	*/*
Accept-Encoding	gzip, deflate, br
Connection	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 the IBM API Explorer interface for the 'cscvinc' service. On the left, there's a list of operations: POST /cscvinc/employee, DELETE /cscvinc/employee/{employee}, and GET /cscvinc/employee/{employee}. A red box highlights the 'Authorization' field in the 'Parameters' section, which contains the value 'Basic dXNlcpwYXNzd29yZA=='. Another red box highlights the 'employee' parameter, which has the value '000050'. On the right, a modal window titled 'mpz3.washington.ibm.com:9443' asks for sign-in information, showing 'Username: user1' and 'Password:'. A red box highlights this modal window.

```
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", "file": {"employee": {"id": "111111", "name": "C. BAKER", "address": "OTTAWA, ONTARIO", "phoneNumber": "51212003", "date": "26 11 81", "amount": "$0011.00"}}}}}
```

```
URL url = new URL("https://wg31.washington.ibm.com:9453/db2/department?dept1=C01&dept2=C01");
System.out.println("URL: " + url);
HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Content-Type", "application/json");
byte[] bytesEncoded = Base64.encodeBase64("Fred:fredpwd".getBytes());
conn.addRequestProperty("Authorization", new String(bytesEncoded));

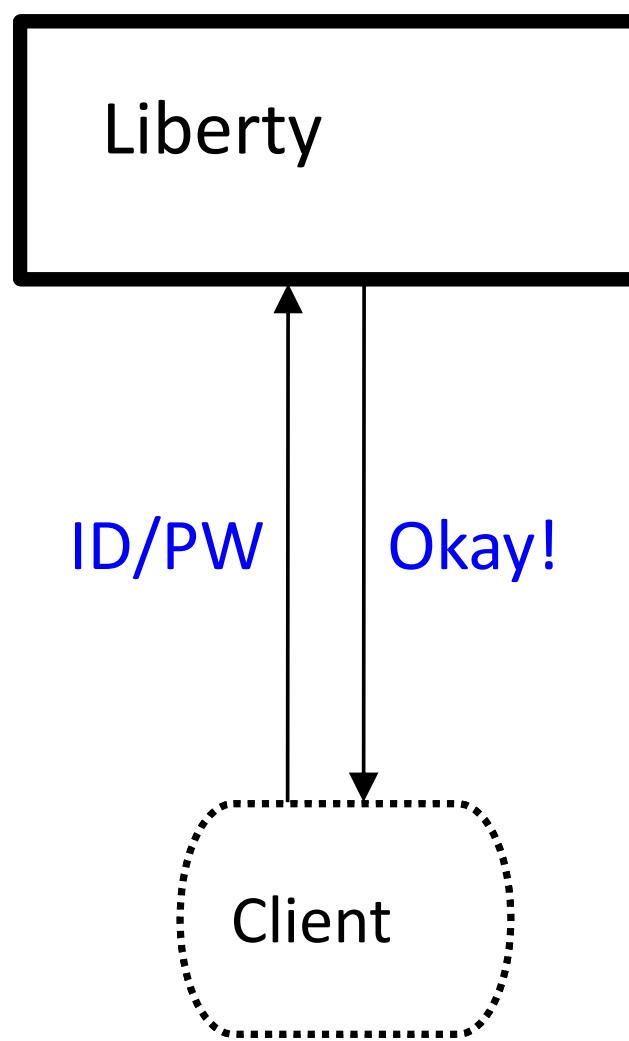
try {
    if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
    }
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader((conn.getInputStream())));
    String output;
    StringBuilder stringBuffer = new StringBuilder();
    while ((output = bufferedReader.readLine()) != null) {
        stringBuffer.append(output);
    }
    JSONObject json = new JSONObject(stringBuffer.toString());
    JSONArray jsonArray = json.getJSONArray("ResultSet 1 Output");
    JSONObject jsonEntry = new JSONObject();
    for (int index = 0; index < jsonArray.length(); index++) {
        jsonEntry = jsonArray.getJSONObject(index);
        if (jsonEntry.has("employeeNumber")){
```



Authentication - TLS Mutual Authentication

Several different ways this can be accomplished:

Basic Authentication



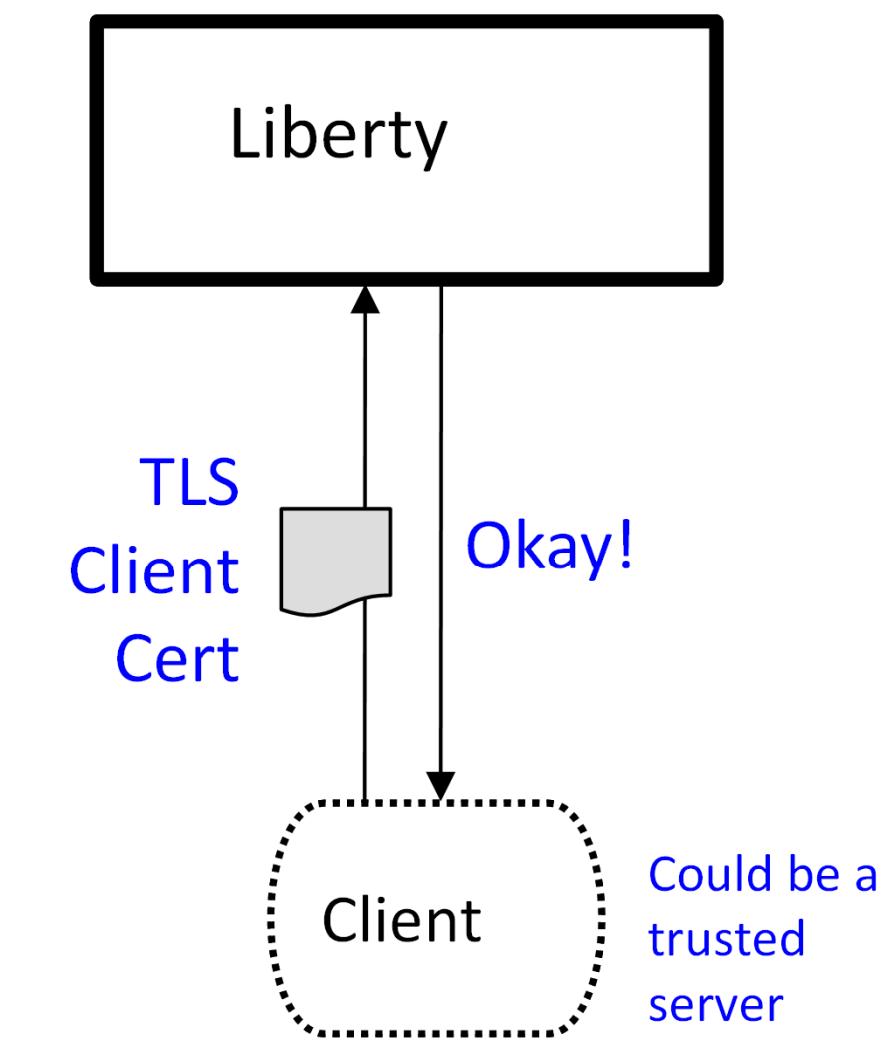
Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

Server checks registry:

- Basic (server.xml)
- SAF

Client Certificate



Server prompts for client certificate.

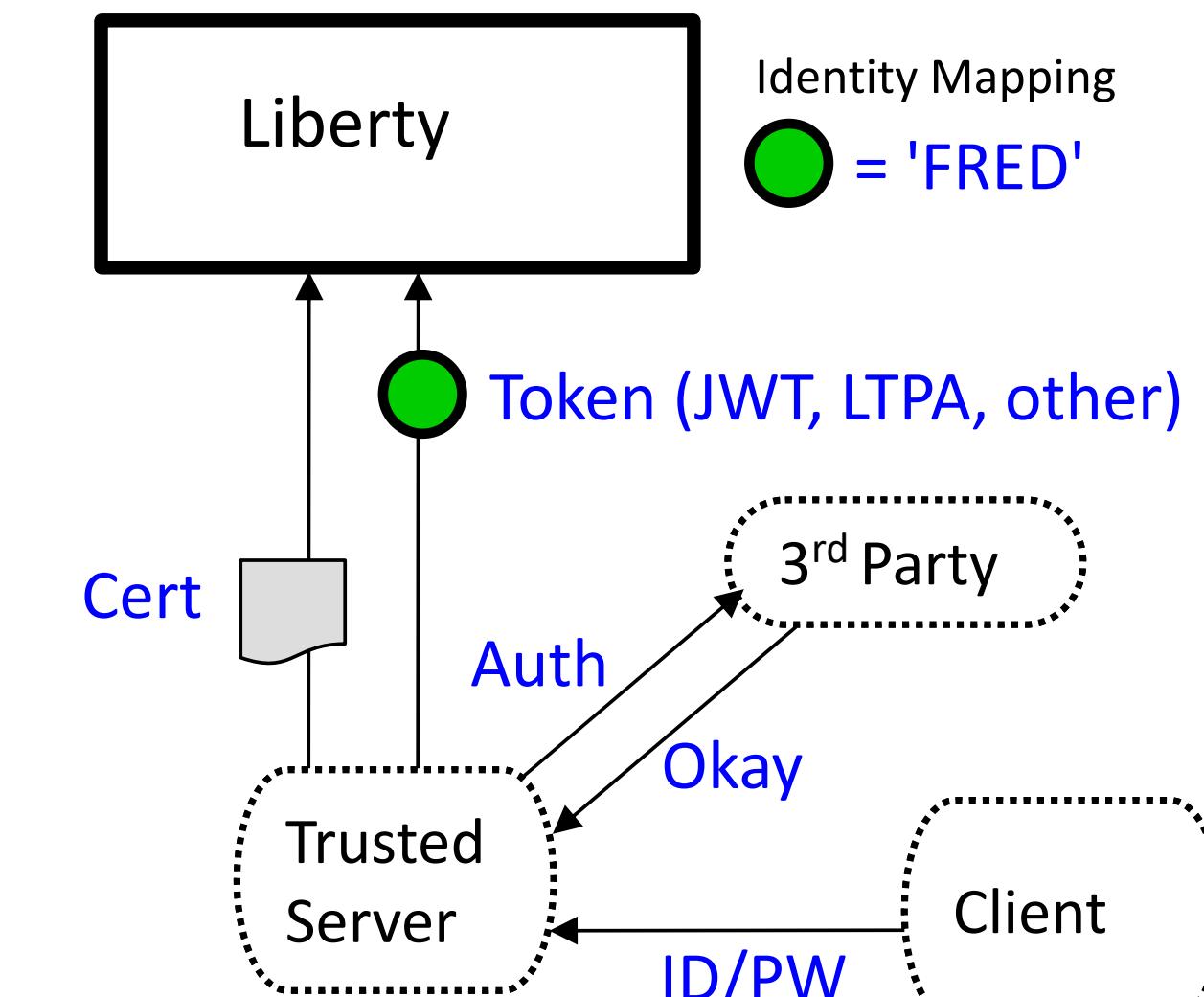
Client supplies personal certificate

Server validates client certificate and maps it to an identity

Registry options:

- SAF

Third Party Authentication



Client authenticates to 3rd party sever

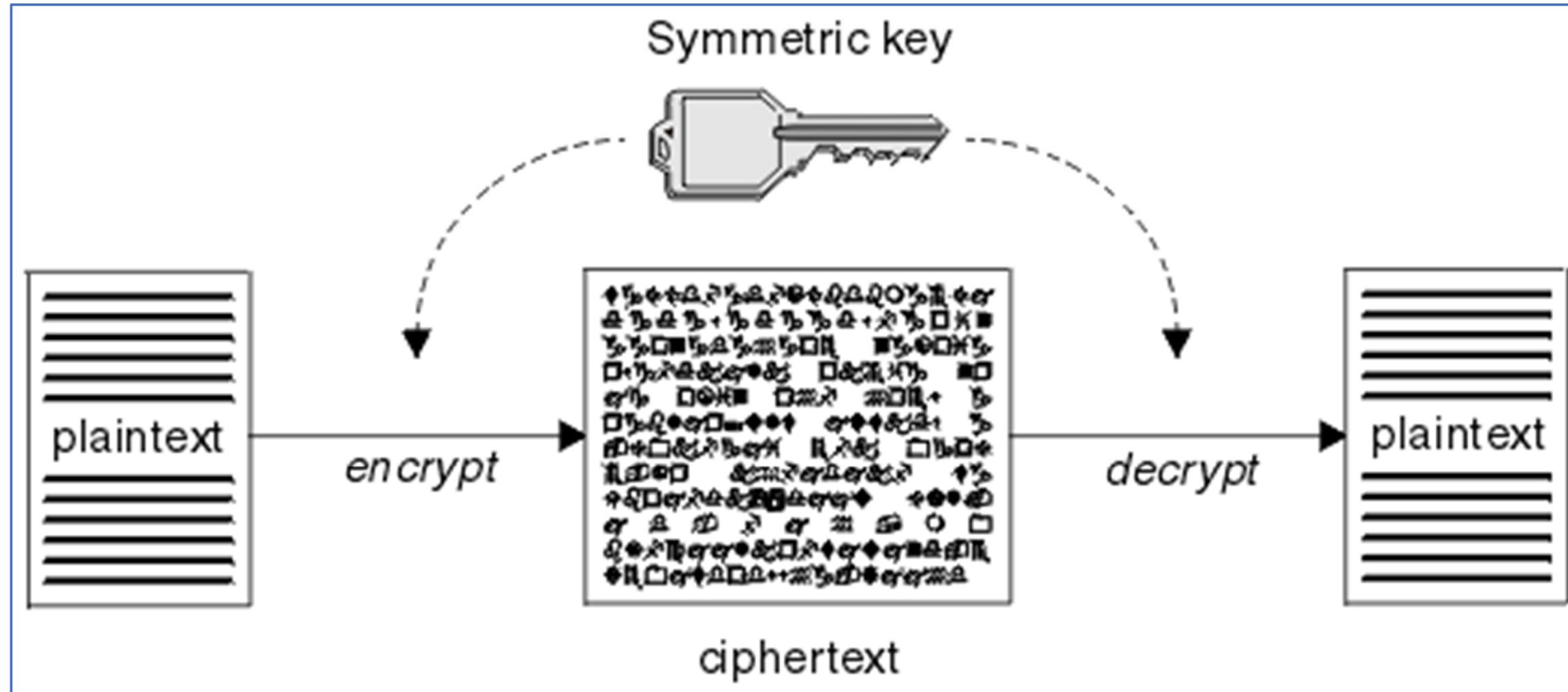
Client receives a trusted 3rd party token

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

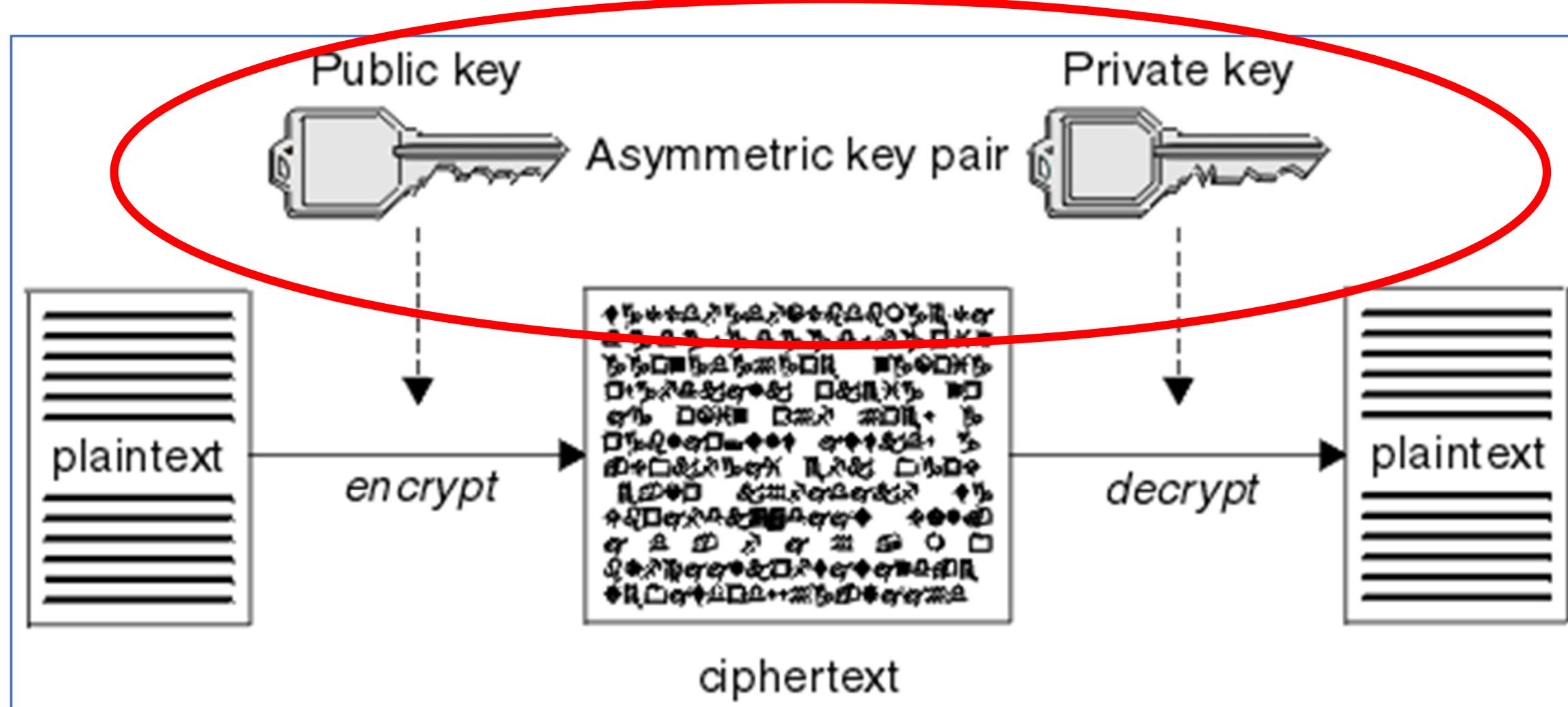
Registry options:

- 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.

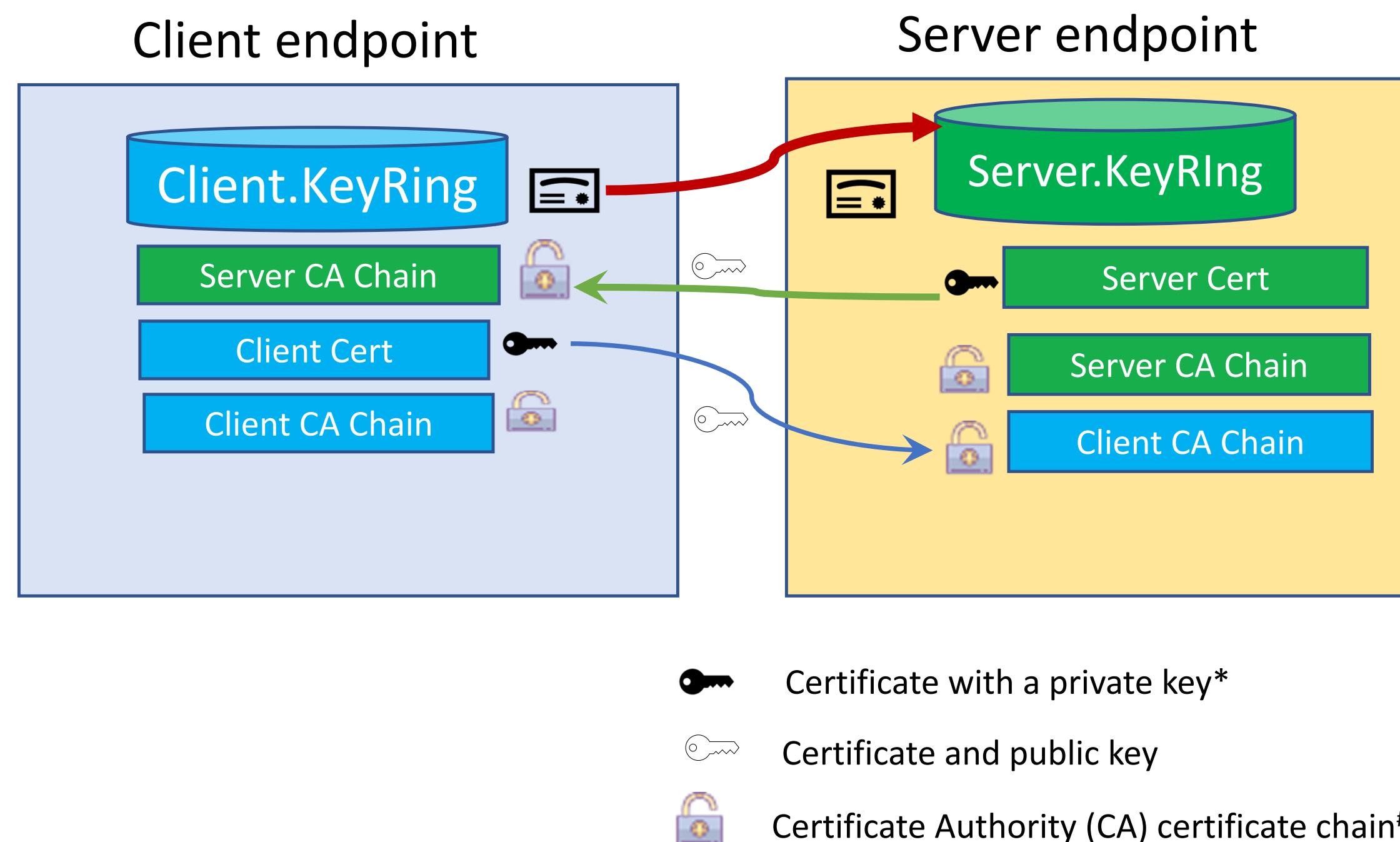
If an endpoint can successfully decrypt a message message encrypted received with a private key, the endpoint sending the message has successfully asserted its validity by proving it has the private used to encrypt the message.

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.

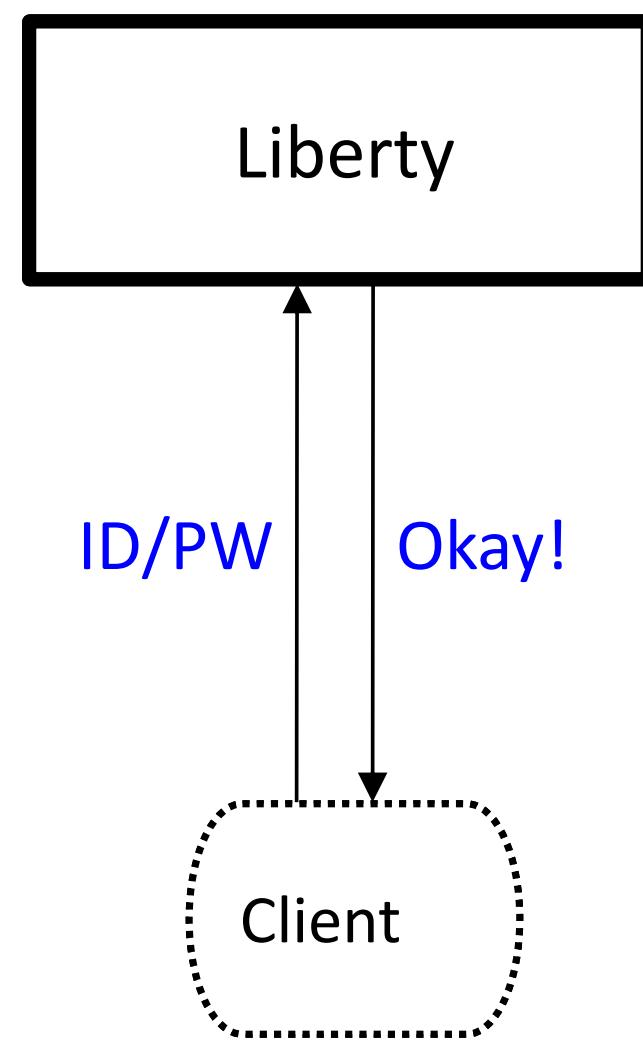
#Refers to the set or of certificates used to issue the server or client personal certificate including any intermediate certificates up to and including the root CA.



Authentication - Third Party Authentication

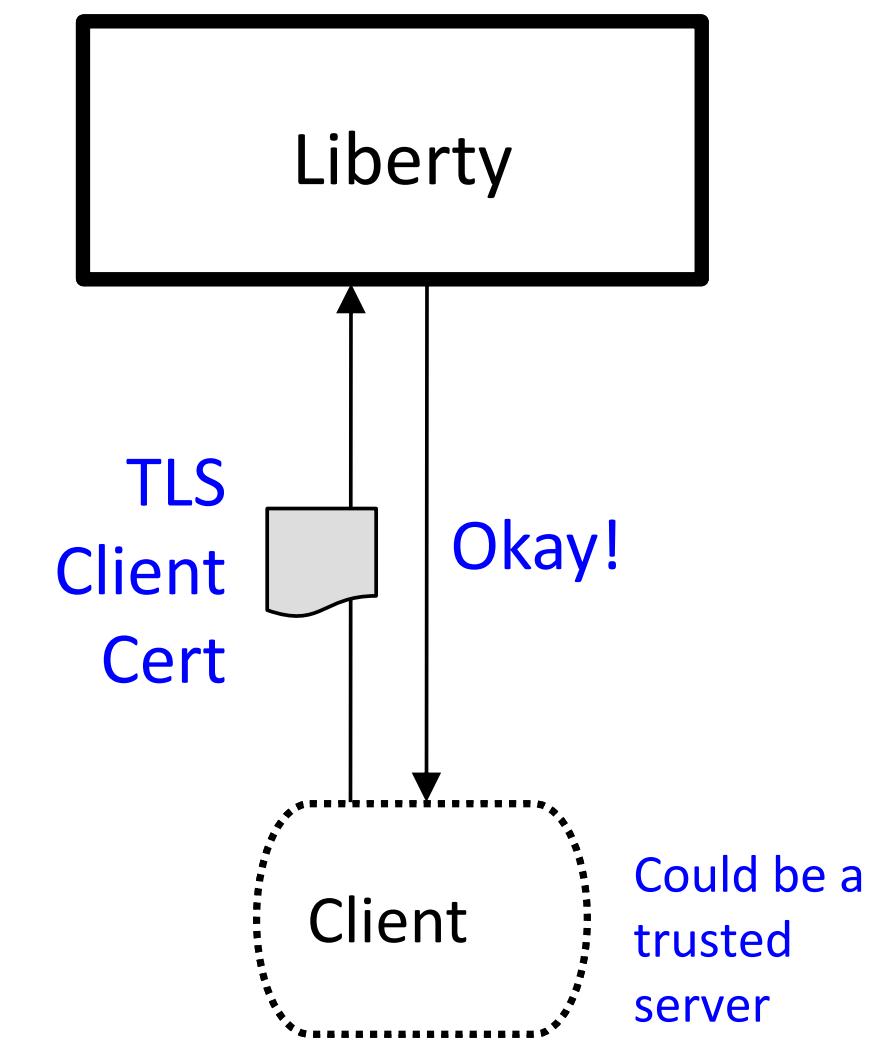
Several different ways this can be accomplished:

Basic Authentication



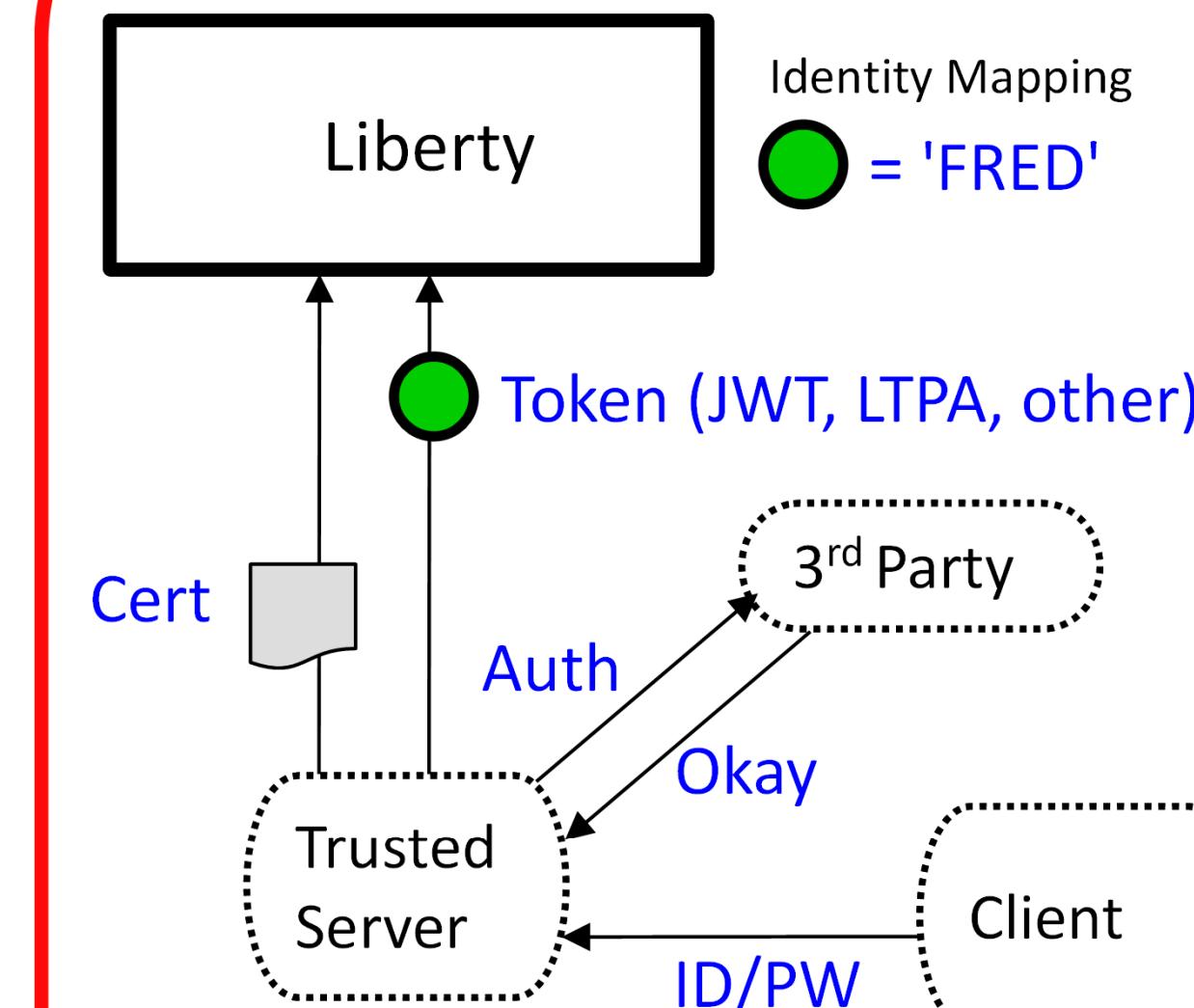
Server prompts for ID/PW
Client supplies ID/PW or ID/PassTicket
Server checks registry:
• Basic (server.xml)
• SAF

Client Certificate



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

Third Party Authentication



**Client authenticates to 3rd party sever
Client receives a trusted 3rd 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

The screenshot shows the UPS Sign Up page. At the top, there's a banner about UPS being open for business due to COVID-19. Below the banner, the UPS logo is displayed. A "Sign Up" button is prominent. Below it, a link to "Log in" is shown. There are sections for "Use one of these sites." with links to Google, Facebook, Amazon, and Apple. Another section for "Or enter your own information." contains fields for Name*, Email*, User ID*, Password*, and Phone. The "Password" field has a "Show" link next to it.

The screenshot shows the myNCDMV Sign In page. It features a "Log In" and "Sign Up" button at the top. The "Log In" section includes fields for "Email Address" (with "name@example.com" entered) and "Password". There's a "Remember Me" checkbox and "Log In" and "Forgot Password" buttons. Below this, there's a "Continue with" section for Apple, Facebook, and Google. A "Continue as Guest" link is also present. A notice for public computer users is at the bottom, and the page is powered by payit.



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

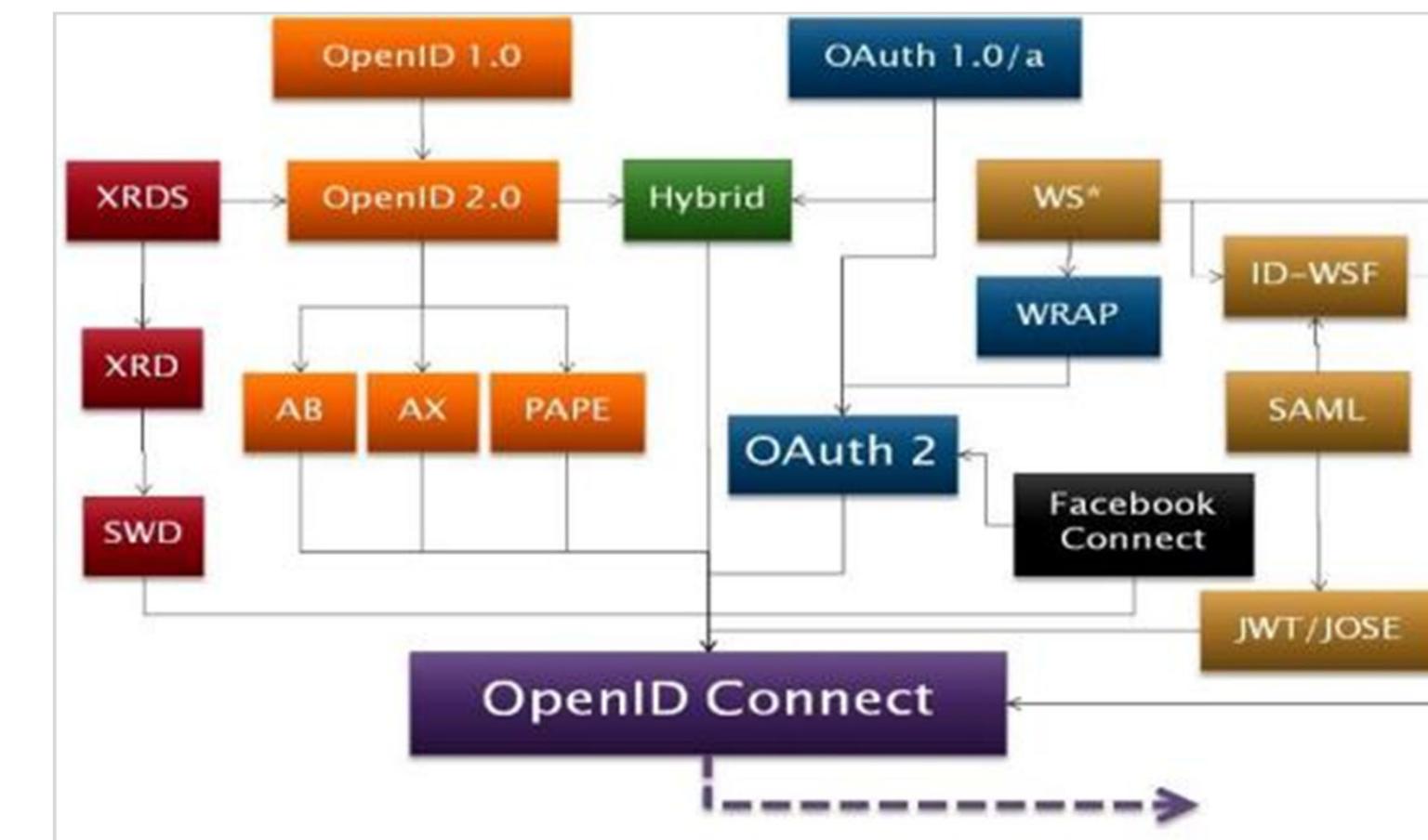
See the YouTube videos:

OAuth 2.0 and OpenID Connect (in plain English)

<https://www.youtube.com/watch?v=996OjexHze0>

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: eyJraWQiOiiI0cWpYLWJrWE9Vd19GX...vT_Ez0fD-. At the bottom of this string, there is a timestamp: "Mon Nov 02 2020 11:05:58 GMT-0500 (Eastern Standard Time)". A red oval highlights this timestamp and the preceding part of the string: DXC8fy6HoLD8gS5CX9Lqj7CcQsk. To the right, under "Decoded", the token is shown in JSON format:

```
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": "myZee",  
  "exp": 160433158,  
  "iat": 160433158,  
  "realmName": "zCEERealm",  
  "uniqueSecurityName": "Fred"  
}
```

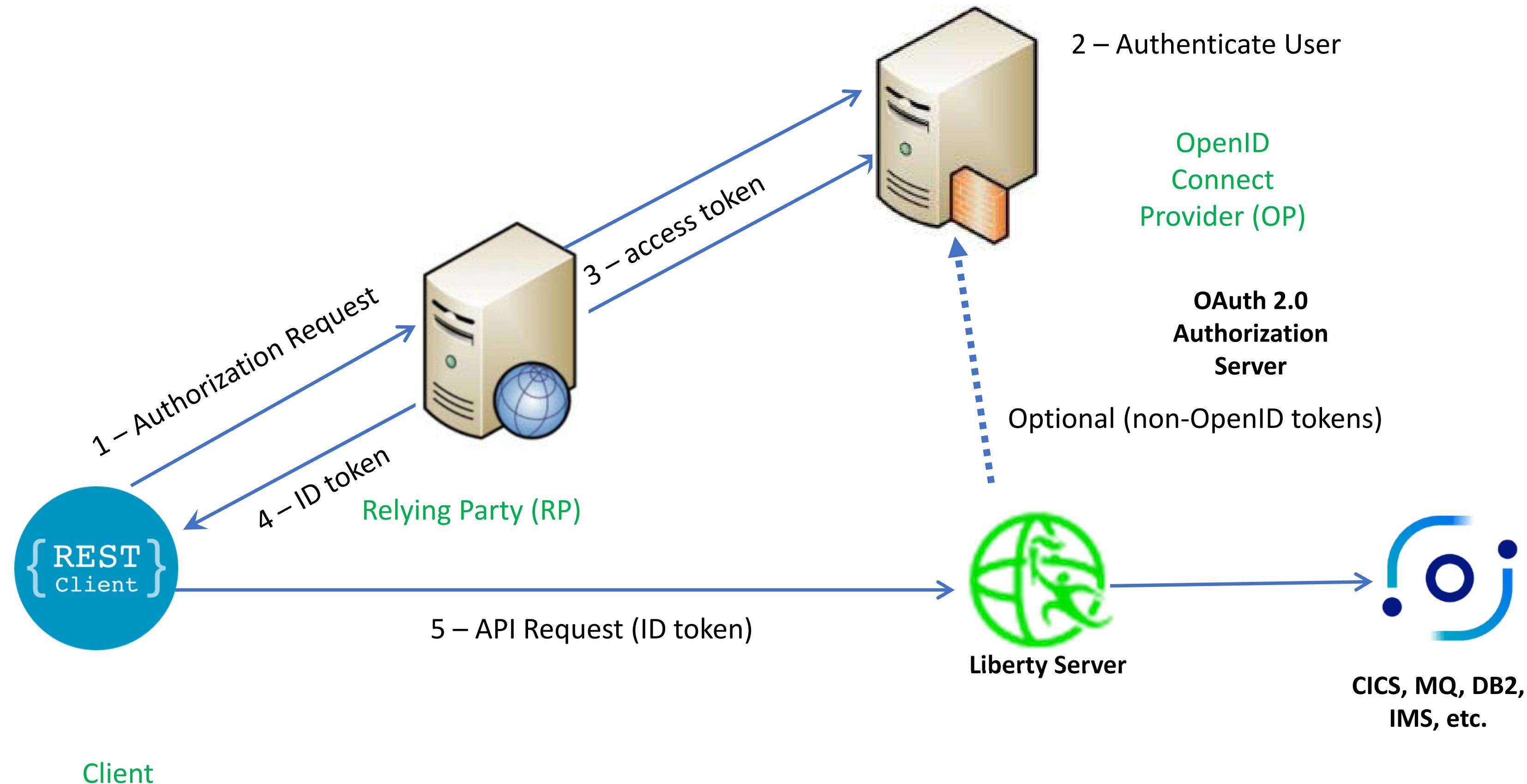
Values derived from the OAUTH configuration:

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

<https://jwt.io>



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



Agenda

- An Introduction and Overview of using REST API
- Enabling RESTful API to various z/OS resources, e.g.
 - CICS
 - Db2
 - IMS/TM
 - IMS/DB
 - MQ
- A brief overview of z/OS Connect Security*

*For more on security, contact your local IBM rep regarding the schedule of workshop *zOSSEC1 IBM z/OS Connect Administration/Security Wildfire Workshop*

z/OS Connect Wildfire Github Site

<https://ibm.biz/zCEEWorkshopMaterial>



The screenshot shows the GitHub repository 'ibm-wsc/zCONNEE-Wildfire-Workshop'. The master branch has 1 branch and 0 tags. The repository contains several folders and files, many of which have been updated via upload. A red oval highlights the 'OpenAPI2' folder.

File/Folder	Action	Last Commit
emitchj Delete xml directory	Add files via upload	12 seconds ago
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

The screenshot shows the GitHub repository 'zCONNEE-Wildfire-Workshop'. It displays two branches: 'master' and 'OpenAPI2'. The 'OpenAPI2' branch contains numerous PDF files related to developing APIs for different IBM products. The 'OpenAPI3' branch also contains some PDF files.

Branch	File	Action	Last Commit
OpenAPI2	Developing CICS API Requester Applications.pdf	Add files via upload	12 seconds ago
	Developing IMS API Requester Applications.pdf	Add files via upload	14 hours ago
	Developing MVS Batch API Requester Application.pdf	Add files via upload	6 days ago
	Developing RESTful APIs for Db2 DVM Services.pdf	Add files via upload	13 hours ago
	Developing RESTful APIs for Db2 REST Services.pdf	Add files via upload	2 months ago
	Developing RESTful APIs for HATS REST Service.pdf	Add files via upload	4 days ago
	Developing RESTful APIs for IMS DVM Services.pdf	Add files via upload	1 minute ago
	Developing RESTful APIs for IMS Database RES.pdf	Add files via upload	13 months ago
	Developing RESTful APIs for IMS Transactions.pdf	Add files via upload	2 months ago
	Developing RESTful APIs for a CICS Program.pdf	Add files via upload	3 months ago
OpenAPI3	emitchj Delete admin		13 hours ago
	Developing RESTful APIs for Db2 REST Services.pdf	Add files via upload	3 months ago

mitchj@us.ibm.com

- Contact your IBM representative to schedule access to these exercises

© 2017, 2023 IBM Corporation

Slide 142



Thank you for listening and your questions