



# IBM z/OS Connect Enterprise Edition

## Introduction and Overview

Mitch Johnson

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

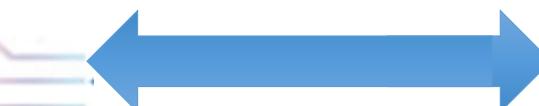
Washington System Center



# Agenda

- z/OS Connect Introduction and overview
- Discuss enabling RESTful API to various z/OS resources, e.g.
  - CICS
  - Db2
  - IMS/TM
  - IMS/DB
  - MQ
  - MVS Batch
  - Outbound REST APIs
  - IBM DVM
  - 3270 screen-based applications (HATS)
  - IBM File Manager
- An introduction to z/OS Connect Security, for more on security, contact your local IBM rep regarding the schedule of workshop *zOSSEC1 IBM z/OS Connect Security Wildfire Workshop*

# **z/OS Connect EE exposes z/OS resources to the “cloud” via RESTful APIs**



+ HCL and Rocket Software

\*Other Vendors or your own implementation

# **/but\_first, what\_is\_REST?**

What makes an API “RESTful”?

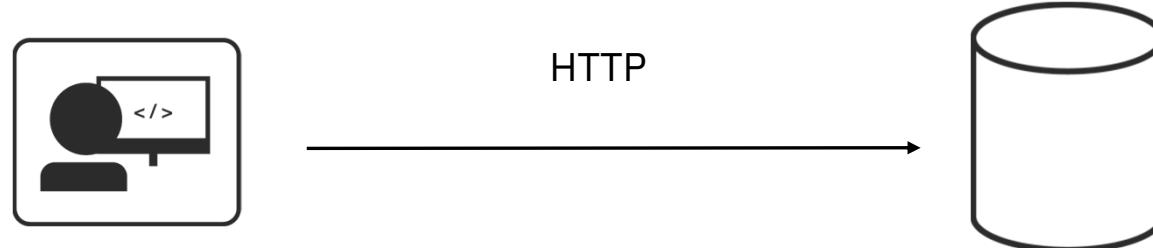
# REST is architectural programming style

**REST** stands 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 REST

Use HTTP verbs for Create, Read, Update, Delete (CRUD) operations

GET  
POST  
PUT  
DELETE

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

Path and Query parameters are used for refinement of the request

URI path identifies a resource (or lists of resources)

Request/Response Body is used to represent the data object

```
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" }
```



# REST vs RESTful

- REST is an architectural style of development having these principles plus..
- It should be stateless (transaction management should be considered by the client)
- It should access and/or identify all the resources from the server 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
- REST based services follow some of the above principles but not all, whereas RESTful means it follows 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

# RESTful Examples



z/OS Connect EE

**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 { "firstName": "Joe",  
 "lastName" : "Bloggs",  
 "addr" : "10 Old Street",  
 "phoneNo" : "01234 0123456" }



RESPONSE HTTP 201 CREATED  
BODY { "id" : "12345",  
 "name" : "Joe Bloggs",  
 "address" : "10 New Street"  
 "tel" : "01234 0123456" }

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

POST http://www.acme.com/customers/12345  
BODY { "updateField": "address",  
 "newValue" : "10 New Street" }



RESPONSE HTTP 200 OK  
BODY { "id" : "12345",  
 "name" : "Joe Bloggs",  
 "address" : "10 New Street"  
 "tel" : "01234 123456" }

# Why is REST popular?

<b>Ubiquitous Foundation</b>	It's based on HTTP, which operates on TCP/IP, which is a ubiquitous networking topology.
<b>Relatively Lightweight</b>	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.
<b>Relatively Easy Development</b>	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.
<b>Increasingly Common</b>	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.
<b>Stateless</b>	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.

# **How do we describe a REST API?**



## **/swagger/open\_api**

The industry standard framework for describing RESTful APIs is by a  
Swagger document

# Why use Swagger?

It is more than just an API framework



There are a number of tools available to aid consumption:

## Consume Swagger\*

**Swagger Codegen** create stub code to consume APIs from various languages



## Read Swagger<sup>+</sup>

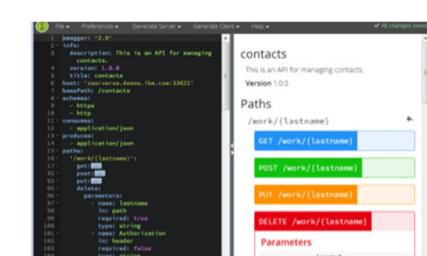
**Swagger UI** allows API consumers to easily browse and try APIs based on Swagger Doc.



The screenshot shows the Swagger UI interface for a 'contacts' API. It displays a list of operations under the 'default' path: GET /work/{lastname}, POST /work/{lastname}, PUT /work/{lastname}, and DELETE /work/{lastname}. The 'POST' operation is highlighted in green, indicating it is selected or being used.

## Write Swagger

**Swagger Editor** allows API developers to design their swagger documents.



The screenshot shows the Swagger Editor interface with a Swagger document for a 'contacts' API. The document includes a title 'This is an API for managing contacts.', version 'Version 1.0.0', and a single path '/work/{lastname}' with four operations: GET, POST, PUT, and DELETE. Each operation has its own configuration panel on the right.

<https://blog.readme.io/what-is-swagger-and-why-it-matters/>

+z/OS Connect, MQ REST support, Zowe

\* z/OS Connect API Requester

# Tech-Tip: Swagger Example



**z/OS Connect EE**

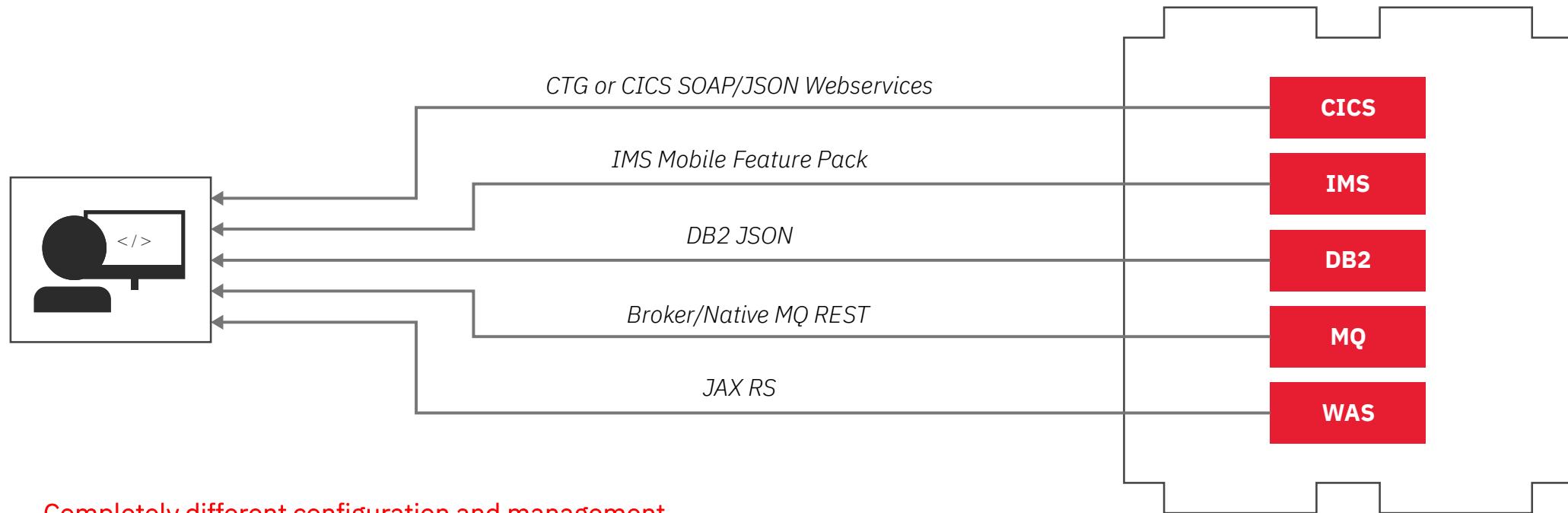
```
/C:/z/workspaces/zCEE/Miniloan/a X +  
file:///C:/z/workspaces/zCEE/Miniloan/api-do...  
JSON Raw Data Headers  
Save Copy Collapse All Expand All Filter JSON  
swagger: "2.0"  
info:  
  description: ""  
  version: "1.0.0"  
  title: "miniloan"  
  termsOfService: "Localhost case"  
  basePath: "/miniloan"  
schemes:  
  0: "https"  
  1: "http"  
consumes:  
  0: "application/json"  
produces:  
  0: "application/json"  
paths:  
  /loan:  
    post:  
      tags:  
        0:  
          name: "miniloan"  
          operationId: "postMiniloanService"  
      parameters:  
        0:  
          name: "Authorization"  
          in: "header"  
          required: false  
          type: "string"  
        1:  
          in: "body"  
          name: "postMiniloanService_request"  
          description: "request body"  
          required: true  
      schema:  
        $ref: "#/definitions/postMiniloanService_request"  
    responses:  
      200:  
        description: "OK"  
        schema:
```



## Why /zos\_connect\_ee?

Truly RESTful APIs to and from your mainframe.

# Could we not do REST before z/OS Connect? Yes, but....



Completely different configuration and management.

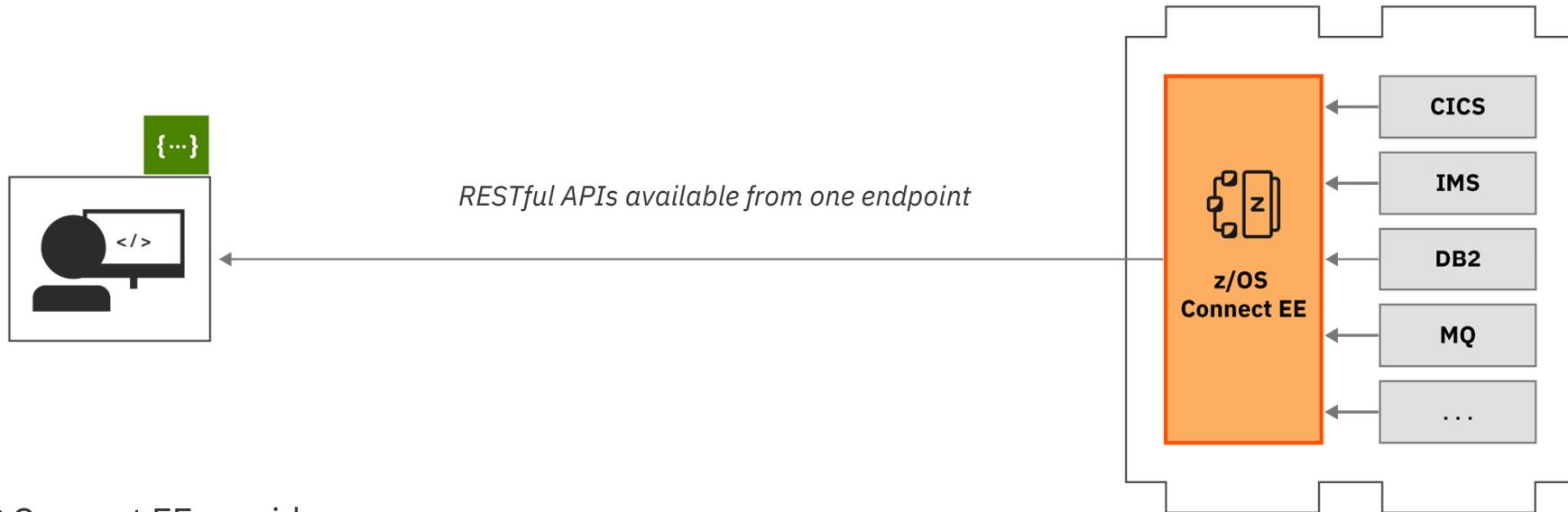
Multiple endpoints for developers to call/maintain access to.

These are typically not RESTful!



# z/OS Connect provides a single-entry point

To expose z/OS resources without writing any code.



## z/OS Connect EE provides

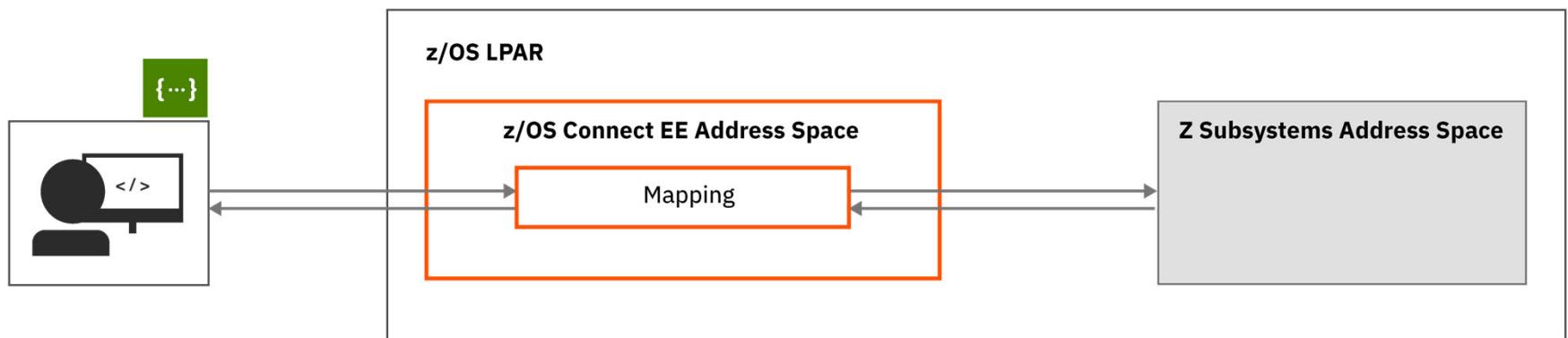
- Single Configuration Administration
- Single Security Administration
- With sophisticated mapping of truly RESTful APIs to existing mainframe and services data without writing any code.



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

# Let's Start with Data mapping

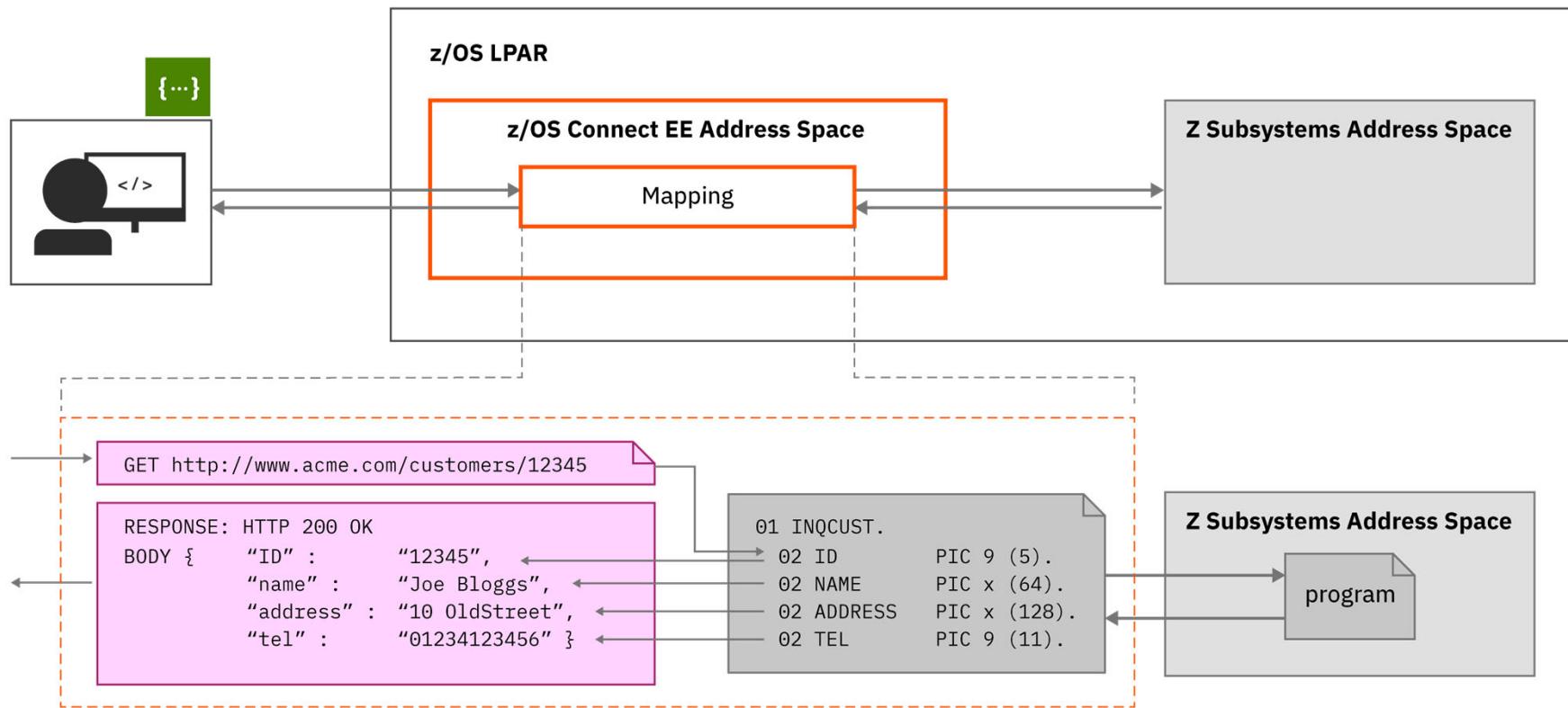
Converting JSON to the format the target's subsystem expects\*.



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

# Data mapping Example

A closer look



# Tech-Tip: COBOL versus JSON Schema Example



```
01 MINILOAN-COMMAREA.  
 10 name pic X(20).  
 10 creditScore pic 9(16)v99.  
 10 yearlyIncome pic 9(16)v99.  
 10 age pic 9(10).  
 10 amount pic 9999999v99.  
 10 approved pic X.  
     88 BoolValue value 'T'.  
 10 effectDate pic X(8).  
 10 yearlyInterestRate pic S9(5).  
 10 yearlyRepayment pic 9(18).  
 10 messages-Num pic 9(9).  
 10 messages pic X(60) occurs 1 to 10 times  
      depending on messages-Num.
```

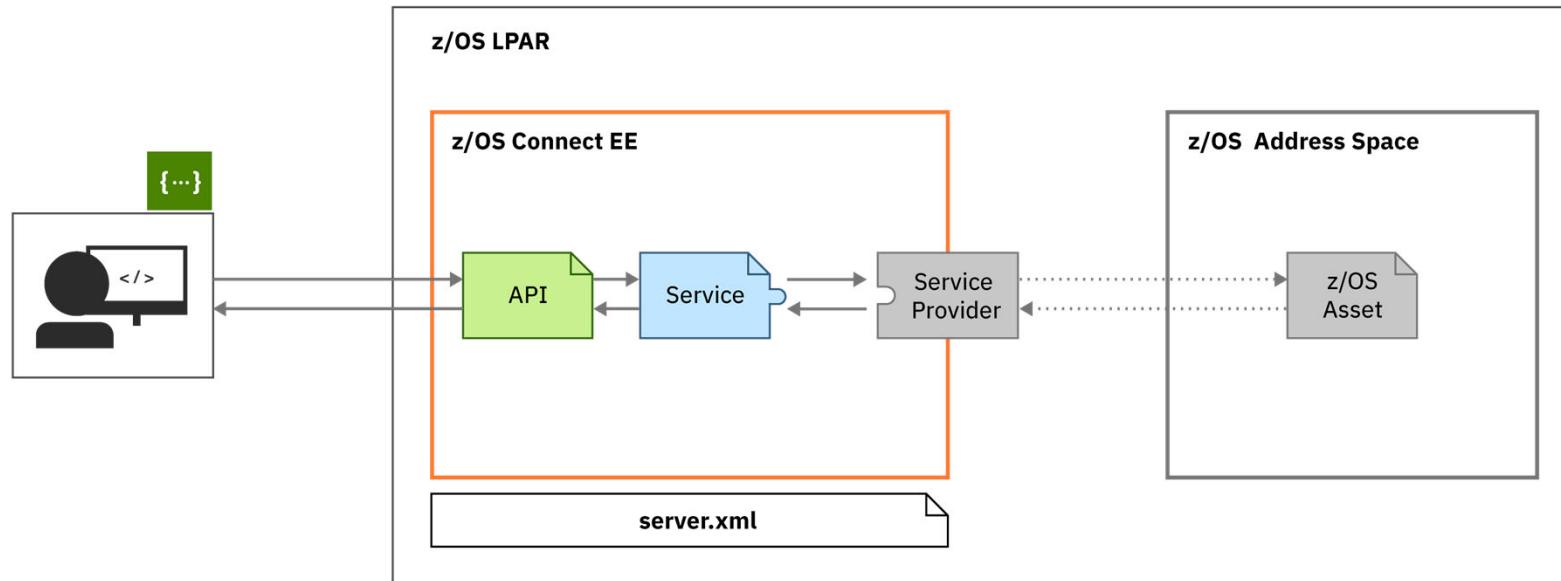
```
"MINILOAN_COMMAREA" : {  
    "type" : "object",  
    "properties" : {  
        "NAME" : {  
            "maxLength" : 20,  
            "type" : "string"  
        },  
        "CREDITSCORE" : {  
            "multipleOf" : 0.01,  
            "minimum" : 0,  
            "maximum" : 99999999999999.99,  
            "type" : "number",  
            "format" : "decimal"  
        },  
    }  
},
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<ns2:message xmlns:ns2="http://www.ibm.com/ims/Transaction" transactionCode="" messageName="miniloanRequest" direction="0" serviceType="0">  
    <message id="1" name="miniloanRequest">  
        <segment id="1" name="COMMAREA" originalName="COMMAREA" included="Y" path="MINILOAN_COMMAREA">  
            <field name="MINILOAN_COMMAREA" originalName="MINILOAN_COMMAREA" included="Y" path="MINILOAN_COMMAREA.NAME">  
                <startPos>1</startPos>  
                <bytes>736</bytes>  
                <maxBytes>0</maxBytes>  
                <applicationDatatype datatype="STRUCT"/>  
                <field name="NAME" originalName="NAME" included="Y" path="MINILOAN_COMMAREA.NAME">  
                    <startPos></startPos>  
                    <bytes>20</bytes>  
                    <maxBytes>20</maxBytes>  
                    <applicationDatatype datatype="CHAR"/>  
                </field>  
                <field name="CREDITSCORE" originalName="CREDITSCORE" included="Y" path="MINILOAN_COMMAREA.CREDITSCORE">  
                    <startPos>21</startPos>  
                    <bytes>18</bytes>  
                    <maxBytes>18</maxBytes>  
                    <marshaller isSigned="N" isSignLeading="N" isSignSeparate="N" isWCHAROnly="N">  
                        <typeConverter>ZONEDDECIMAL</typeConverter>  
                    </marshaller>  
                    <applicationDatatype datatype="DECIMAL" precision="18" scale="2"/>  
                </field>
```

[https://www.ibm.com/support/knowledgecenter/en/SSEPEK\\_10.0.0/char/src/tpc/db2z\\_endianness.html](https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/char/src/tpc/db2z_endianness.html)

# Steps to expose a z/OS application

It is not a single monolithic interface – separated into components



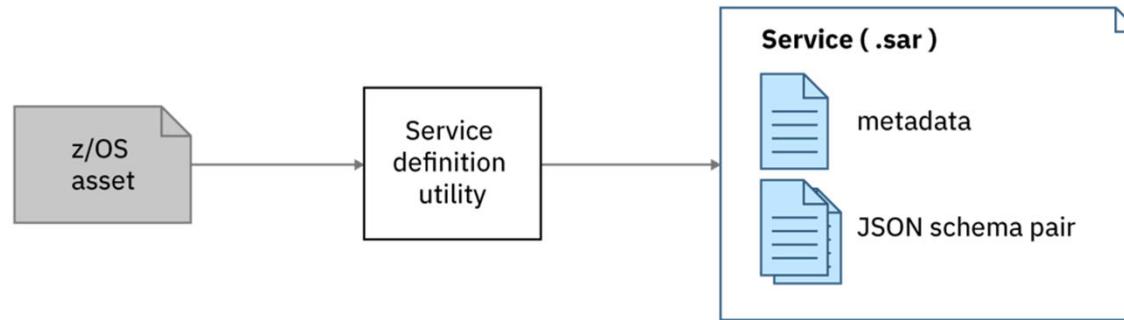
- The API provides the RESTful interface is ready to be consumed by a client and it requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS Asset (e.g., CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific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

# Steps to expose a z/OS application

## 1. Start by creating a service to represent an interaction with the 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

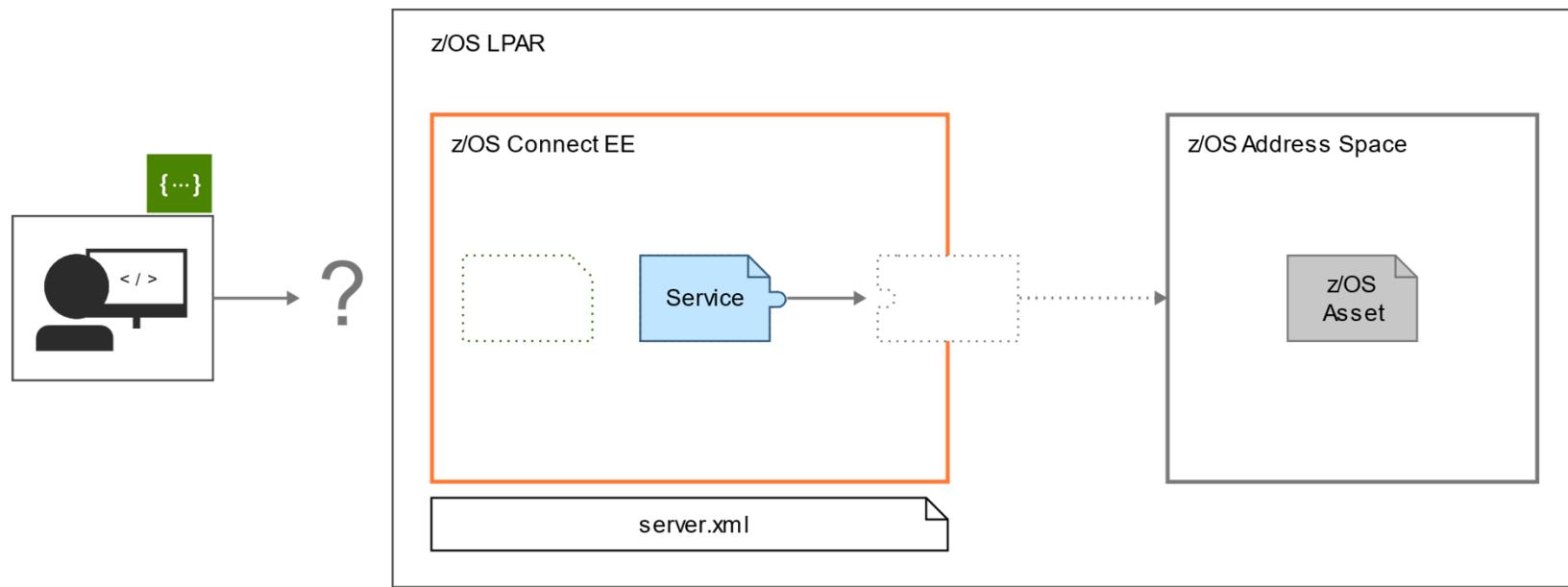


Use a system-appropriate utility to generate a service archive file for the z/OS application

- 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

# Steps to expose a z/OS application

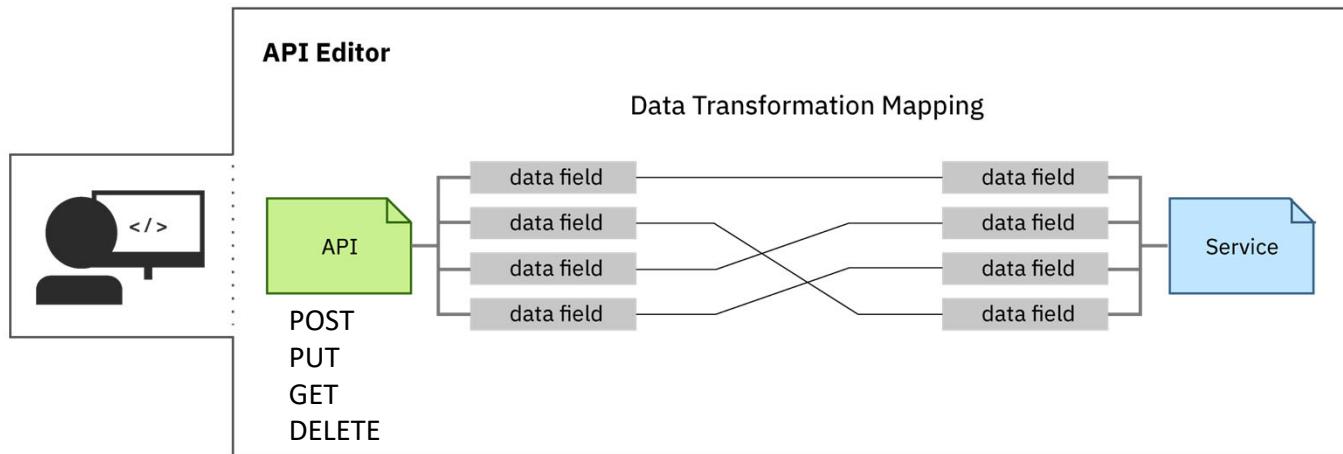
## 2. Deploy and export the service



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

# Steps to expose a z/OS application

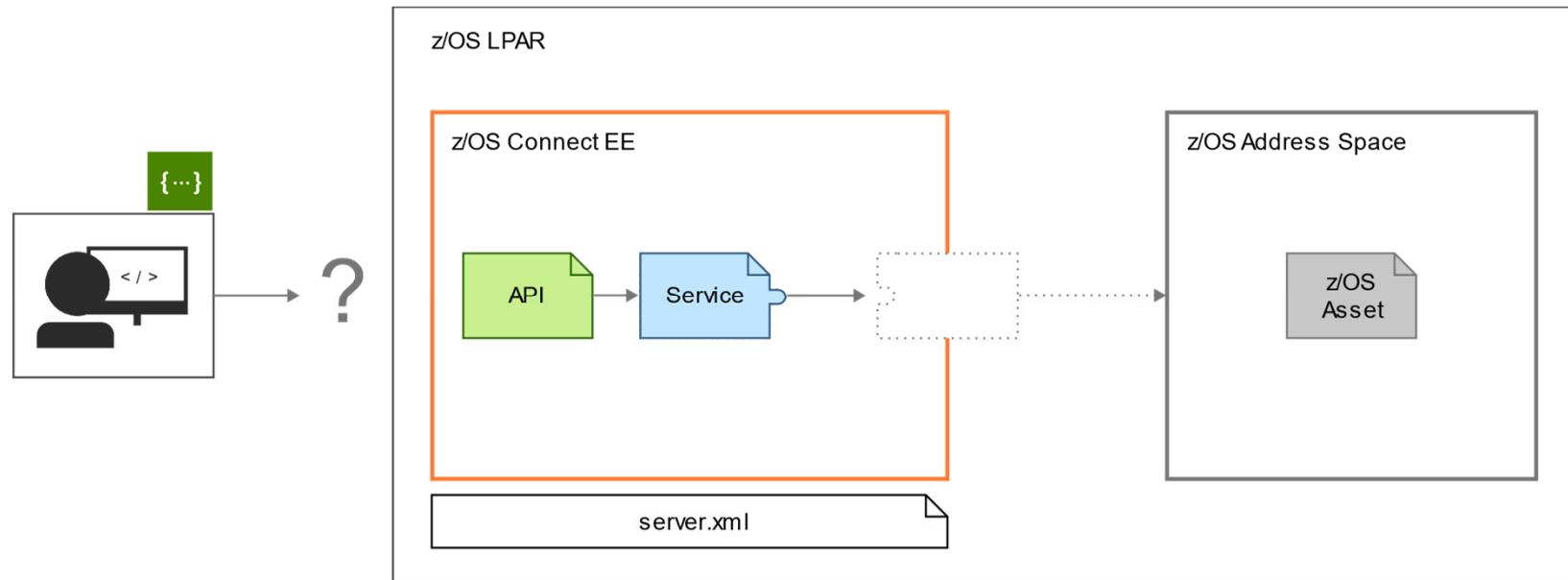
3. Export the service and then import it to create an API that consumes the service



- 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.

# Steps to expose a z/OS application

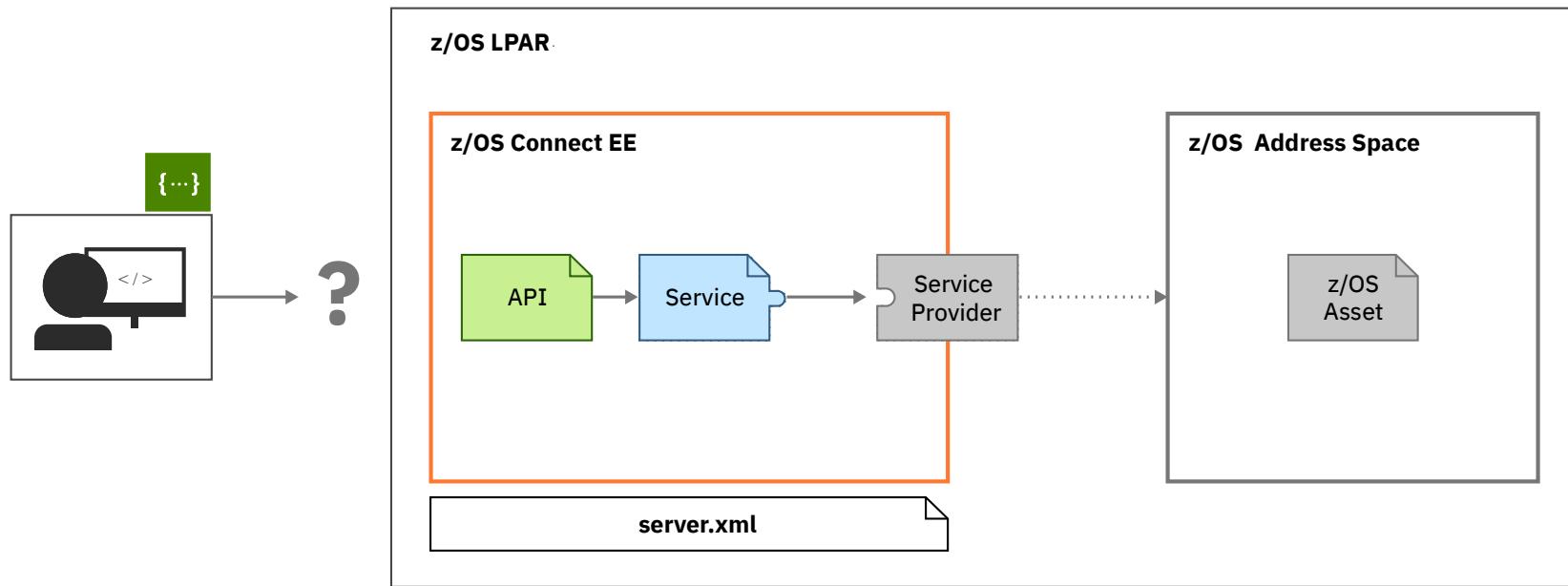
## 4. Deploy the API



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

# Steps to expose a z/OS application

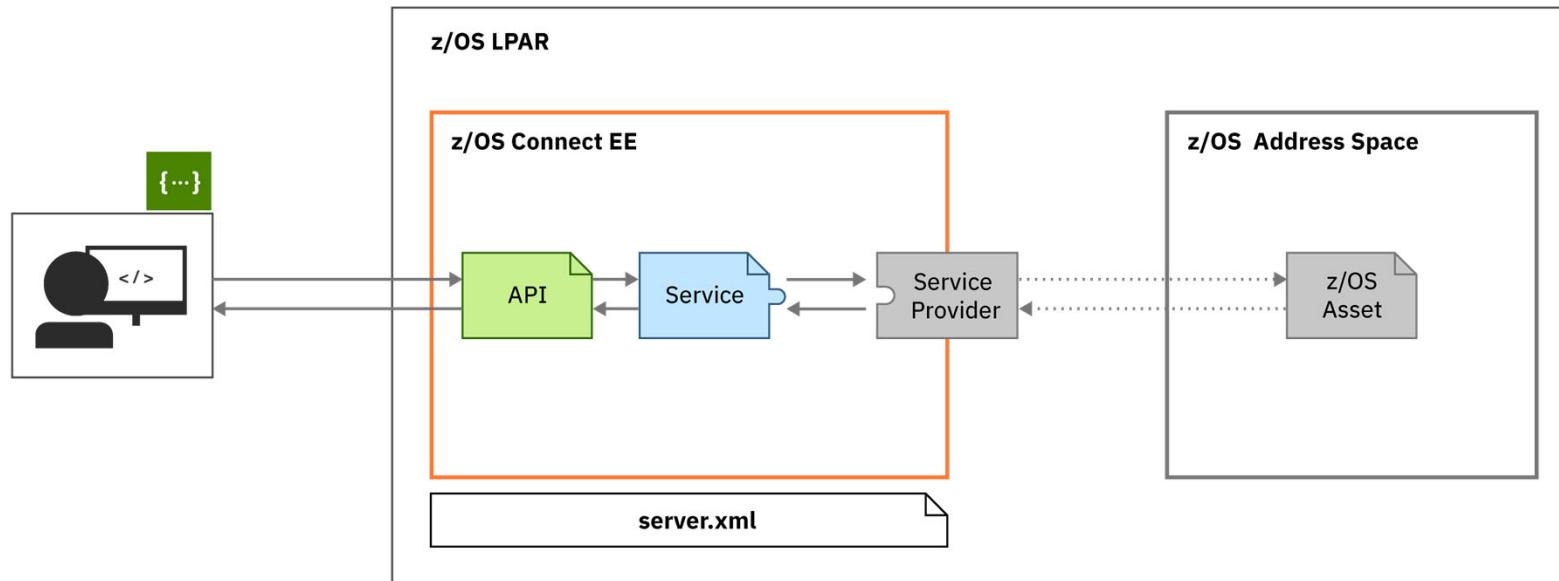
## 5. Configure the service provider



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

# Steps to expose a z/OS application

## 6. 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)

# Results: the client code is unaware of the z/OS infrastructure



**CICS**

```

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

```

Problems @ Javadoc Declaration Console Coverage

<terminated> ZceeCICSGet [Java Application] C:\Program Files\IBM\Java80\jre\bin\javaw.exe (May 7, 2021, 2:54:24 PM)

URL: https://wg31.washington.ibm.com:9453/cscvinc/employee/222222

USERID: CICSUSER

CEIBRESP: 0

CEIBRESP2: 0

name: DR E. GRIFFITHS

employeeNumber: 222222

amount: \$0022.00

address: FRANKFURT, GERMANY

phoneNumber: 20034151

date: 26 11 81

Response Message: {"cscvincSelectServiceOperationResponse":{"cscvincContainer":{"response":{"CEIBRESP2":0,"USERID":"CICSUSER"}}

**Db2**

```

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

```

Problems @ Javadoc Declaration Console Coverage

<terminated> ZceeMqPut [Java Application] C:\Program Files\IBM\Java80\jre\bin\javaw.exe (May 7, 2021, 2:56:06 PM)

URL: https://wg31.washington.ibm.com:9453/db2/employee/000010

Employee Number: 000010

First Name : CHRISTINE

Last Name: HAAS

Middle Initial: T

**MQ**

```

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

```

Problems @ Javadoc Declaration Console Coverage

<terminated> ZceeMqGet [Java Application] C:\Program Files\IBM\Java80\jre\bin\javaw.exe (May 7, 2021, 8:53:07 AM)

URL: https://wg31.washington.ibm.com:9453/mqapi/

NAME: TINA J YOUNG

NUMB: 001781

ADDRX: SINDELINGEN,GERMANY

PHONE: 70319990

DATEX: 21 06 77

AMOUNT: \$0009.99

**IMS**

```

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

```

Problems @ Javadoc Declaration Console Coverage

<terminated> ZceeMSGet [Java Application] C:\Program Files\IBM\Java80\jre\bin\javaw.exe (May 17, 2021, 8:48:25 AM)

URL: https://wg31.washington.ibm.com:9453/phonebook/contacts/LAST1

lastName: LAST1

firstName: FIRST1

zipCode: D01/R01

extension: 8-111-1111

message: ENTRY WAS DISPLAYED

HTTP code: 200



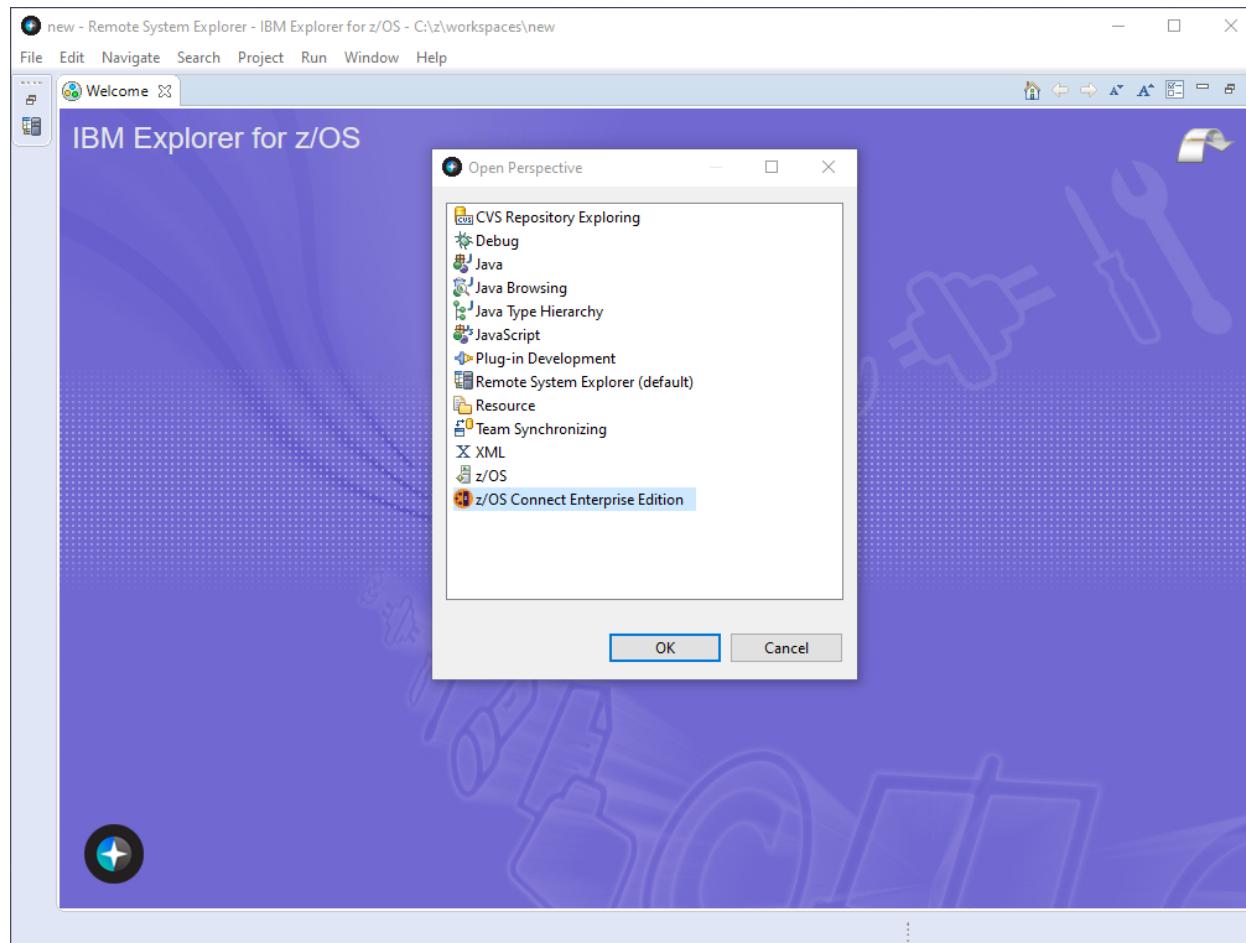
# /api\_toolkit/services

Simple **service creation.**

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



z/OS Connect EE



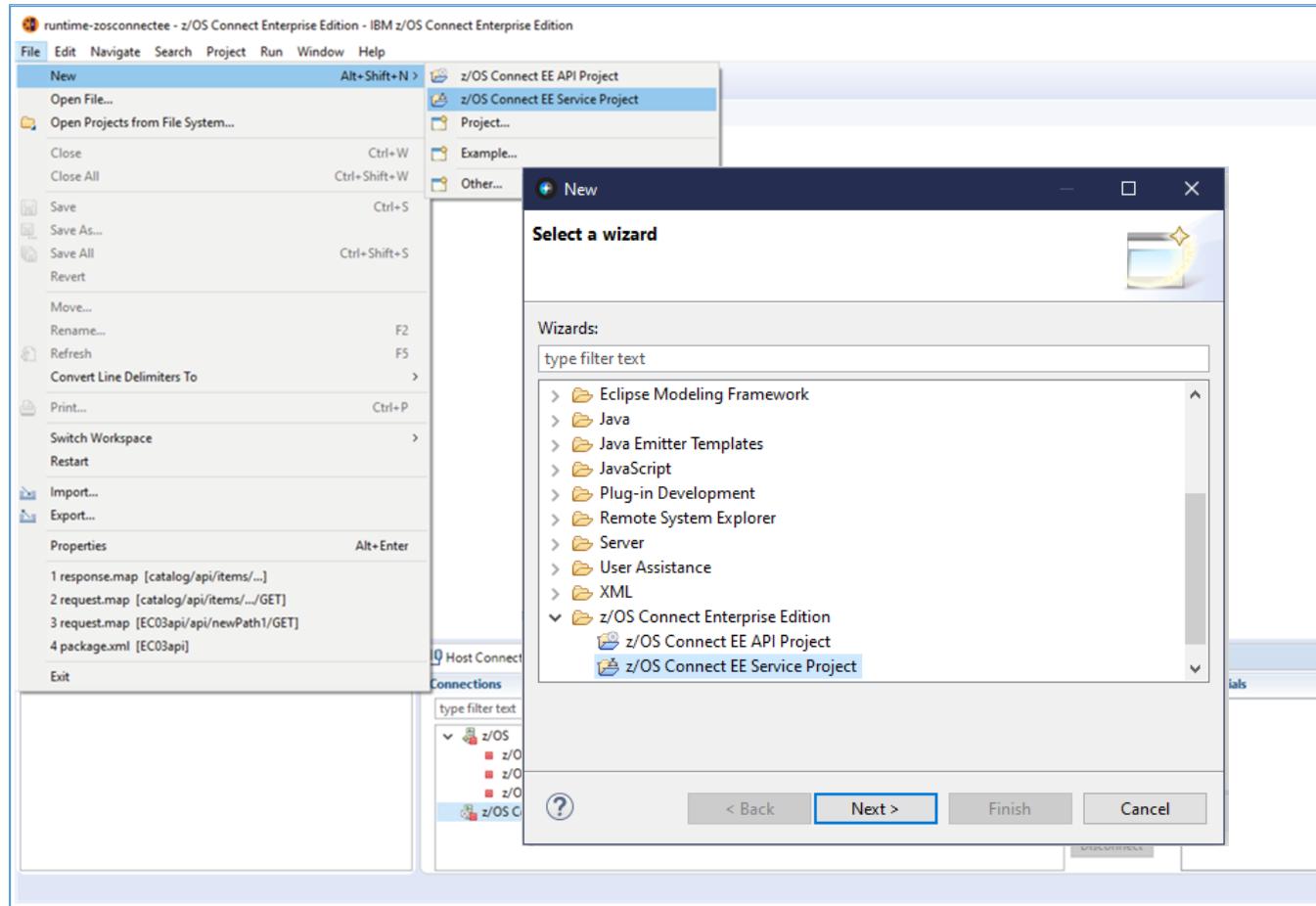
Use the **API toolkit** to create services through Eclipse-based tooling.

The API toolkit is available in the z/OS Connect Enterprise Edition Perspective in an Eclipse environment.

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



z/OS Connect EE

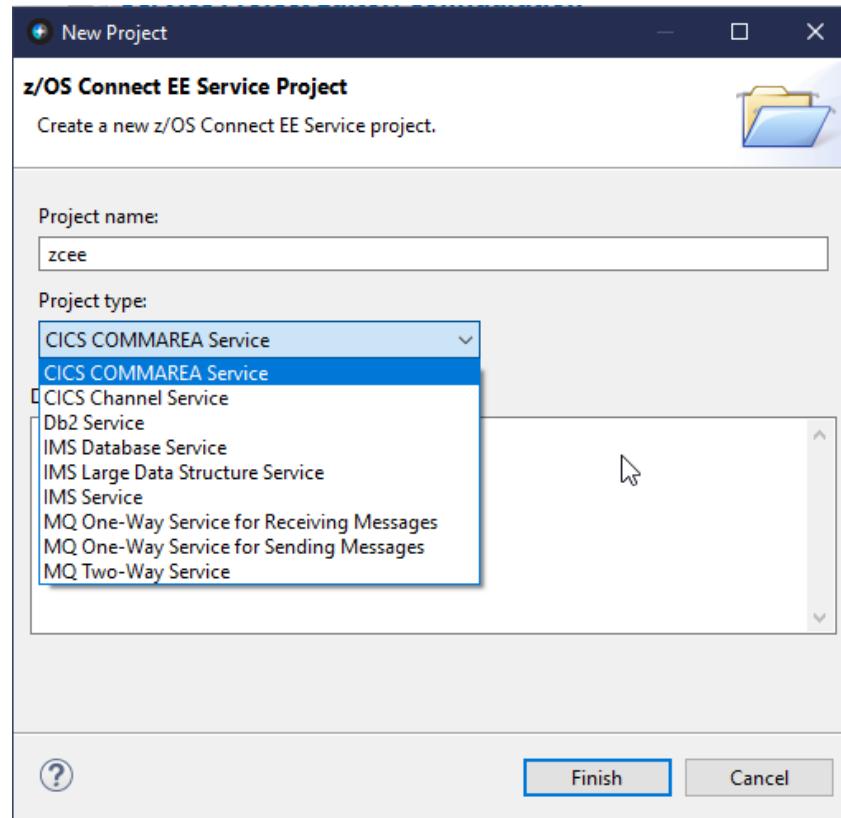


Use the **API toolkit** to create services through Eclipse-based tooling.

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

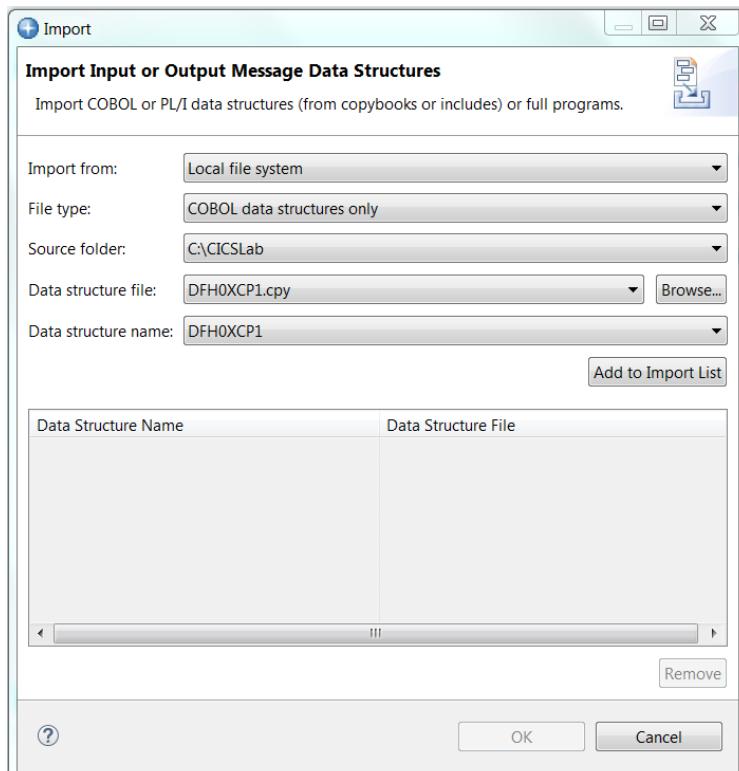
## Service creation – a common interface



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

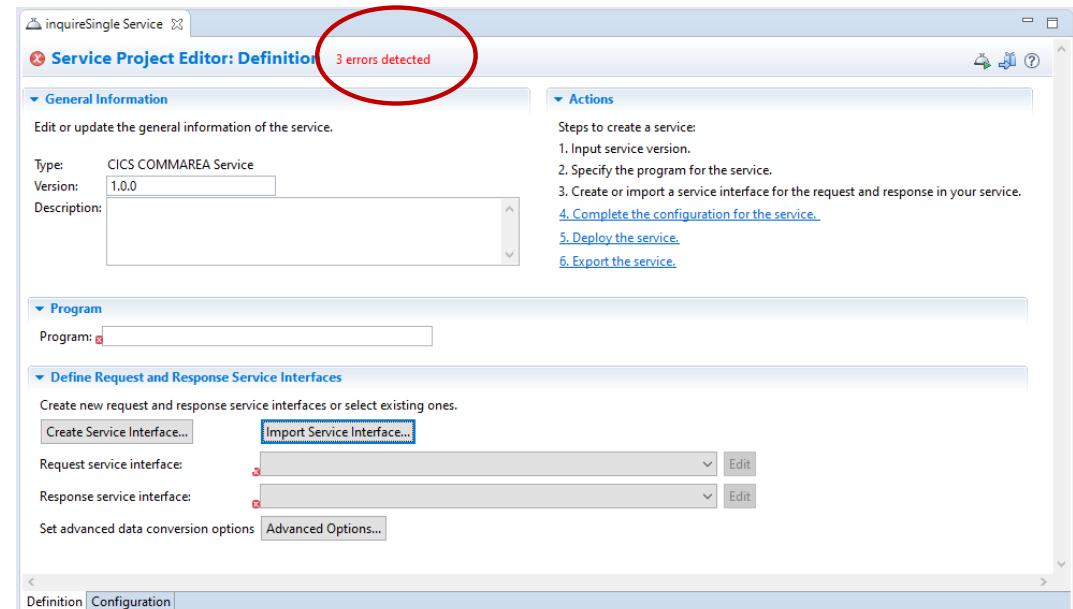
# API toolkit – Creating Services for CICS, IMS TM and MQ

## 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.

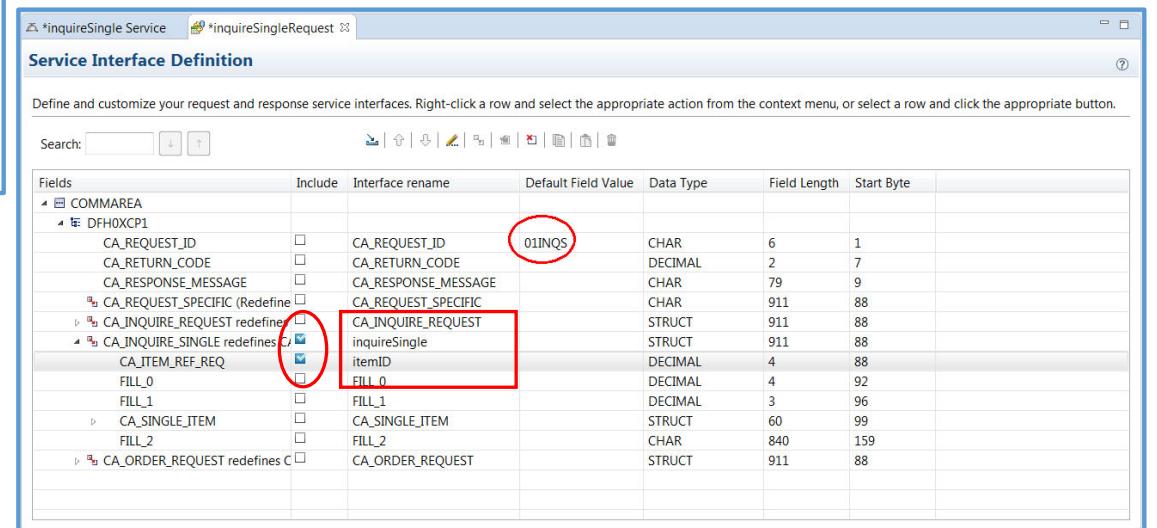


# API toolkit – Creating Services for CICS, IMS TM and MQ

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 for a service named "\*inquireSingle Service". The table has 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\_INQUIRE\_REQUEST field, which is set to "inquireSingle". Another red box highlights the "Default Field Value" column for the same field, which is set to "01INQS". The table also lists other fields like CA\_REQUEST\_ID, CA\_RETURN\_CODE, CA\_RESPONSE\_MESSAGE, CA\_REQUEST\_SPECIFIC, CA\_ITEM\_REF\_REQ, and CA\_ORDER\_REQUEST.

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFHXCPI						
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID	01INQS	CHAR	6	1
CA_RETURN_CODE	<input type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	<input type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefine)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefine	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_INQUIRE_SINGLE redefines CA_INQUIRE_REQUEST	<input checked="" type="checkbox"/>	inquireSingle		STRUCT	911	88
CA_ITEM_REF_REQ	<input type="checkbox"/>	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.

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA	<input type="checkbox"/>					
DFH0XCP1	<input checked="" type="checkbox"/>					
CA_REQUEST_ID	<input checked="" 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 (Redefines CA_INQUIRE_REQUEST)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911	88
CA_INQUIRE_REQUEST redefines CA_INQUIRE_SINGLE	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
CA_INQUIRE_SINGLE redefines CA_ITEM_REF_REQ	<input checked="" type="checkbox"/>	inquireSingle		STRUCT	911	88
CA_ITEM_REF_REQ	<input type="checkbox"/>	CA_ITEM_REF_REQ		DECIMAL	4	88
FILL_0	<input type="checkbox"/>	FILL_0		DECIMAL	4	92
FILL_1	<input type="checkbox"/>	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	<input checked="" type="checkbox"/>	singleItem		STRUCT	60	99
CA_SNGL_ITEM_REF	<input checked="" type="checkbox"/>	itemReference		DECIMAL	4	99
CA_SNGL_DESCRIPTION	<input checked="" type="checkbox"/>	description		CHAR	40	103
CA_SNGL_DEPARTMENT	<input checked="" type="checkbox"/>	department		DECIMAL	3	143
CA_SNGL_COST	<input checked="" type="checkbox"/>	cost		CHAR	6	146
IN_SNGL_STOCK	<input checked="" type="checkbox"/>	inStock		DECIMAL	4	152
ON_SNGL_ORDER	<input checked="" type="checkbox"/>	onOrder		DECIMAL	3	156
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840	159
CA_ORDER_REQUEST redefines CA_USERID	<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

\*cscvincSelectService Service \*cscvincSelectRequest

**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.

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length
Channel		CSCCINCContainer			
@ Container1		REQUEST_CONTAINER			
REQUEST_CONTAINER			S	CHAR	1
ACTION	<input type="checkbox"/>	ACTION		CHAR	1
USERID	<input type="checkbox"/>	USERID		CHAR	8
FILEA_AREA	<input checked="" type="checkbox"/>	FILEA_AREA		STRUCT	80
STAT	<input type="checkbox"/>	STAT		CHAR	1
NUMB	<input checked="" type="checkbox"/>	NUMB		CHAR	6
NAME	<input type="checkbox"/>	NAME		CHAR	20
ADDRX	<input type="checkbox"/>	ADDRX		CHAR	20
PHONE	<input type="checkbox"/>	PHONE		CHAR	8
DATEX	<input type="checkbox"/>	DATEX		CHAR	8
AMOUNT	<input type="checkbox"/>	AMOUNT		CHAR	8
COMMENT	<input type="checkbox"/>	COMMENT		CHAR	9

\*cscvincSelectService Service \*cscvincInsertRequest

**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.

Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length
Channel		cscvincInsertContainer			
@ Container1		REQUEST_CONTAINER			
REQUEST_CONTAINER			I	CHAR	1
ACTION	<input type="checkbox"/>	ACTION		CHAR	1
USERID	<input type="checkbox"/>	USERID		CHAR	8
FILEA_AREA	<input checked="" type="checkbox"/>	FILEA_AREA		STRUCT	80
STAT	<input checked="" type="checkbox"/>	status		CHAR	1
NUMB	<input checked="" type="checkbox"/>	employeeNumber		CHAR	6
NAME	<input checked="" type="checkbox"/>	employeeName		CHAR	20
ADDRX	<input checked="" type="checkbox"/>	address		CHAR	20
PHONE	<input checked="" type="checkbox"/>	phoneNumber		CHAR	8
DATEX	<input checked="" type="checkbox"/>	startDate		CHAR	8
AMOUNT	<input checked="" type="checkbox"/>	amount		CHAR	8
COMMENT	<input checked="" type="checkbox"/>	comment		CHAR	9

\*cscvincSelectService Service

**Service Project Editor: Definition**

**General Information**

Edit or update the general information of the service.

Type: CICS Channel Service  
Version: 1.0.0  
Description:

**Actions**

Steps to create a service:

1. Input service version.
2. Specify the program for the service.
3. Create or import a service interface for the request and response in your service.
4. Complete the configuration for the service.
5. Deploy the service.
6. Export the service.

**Program**

Program: CSCVINC

**Define Request and Response Service Interfaces**

Create new request and response service interfaces or select existing ones.

**Request service interface:** cscvincSelectRequest.si  
**Response service interface:** cscvincSelectResponse.si  
**Set advanced data conversion options:** Advanced Options...

```
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.
```

mitchj@us.ibm.com

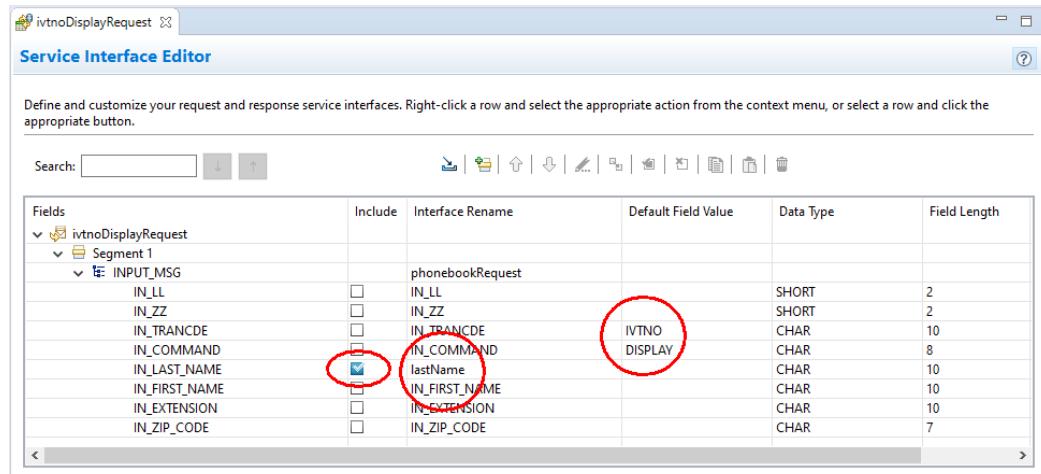
The service developer creates distinct services for each function by setting the ACTION field to S for select, I for insert, U for update or D for delete

© 2018, 2021 IBM Corporation

# 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 Service Interface Editor

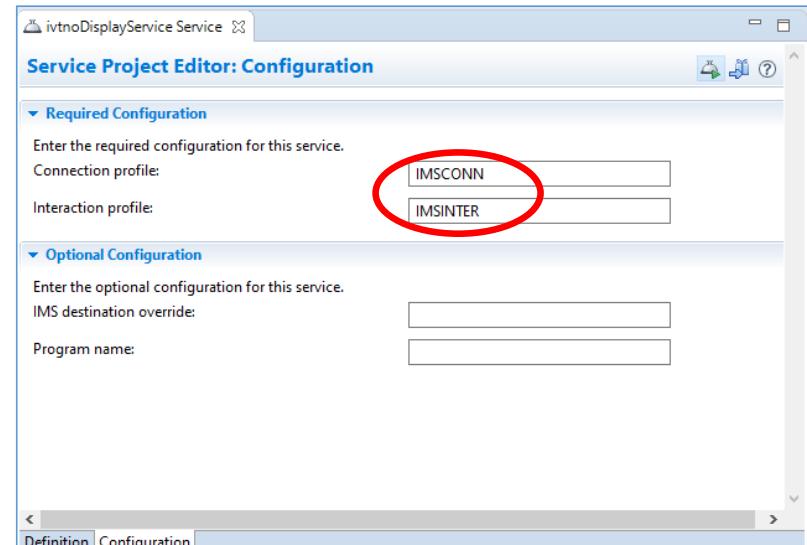
Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

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_TRANCDE	<input type="checkbox"/>	IN_TRANCDE		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

mitchj@us.ibm.com

The service developer creates distinct services for each function.

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

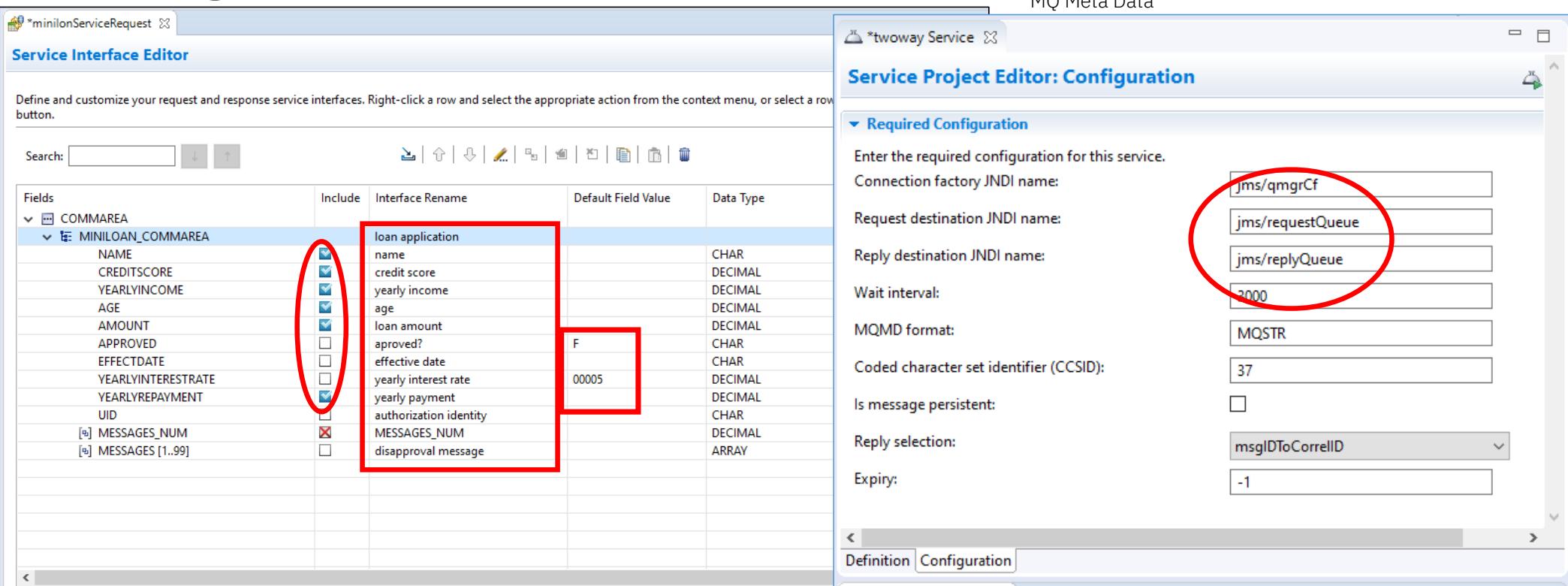


IMS/TM Meta Data

© 2018, 2021 IBM Corporation

# API toolkit – Creating Services for MQ

## Creating a “POST” service interface definition



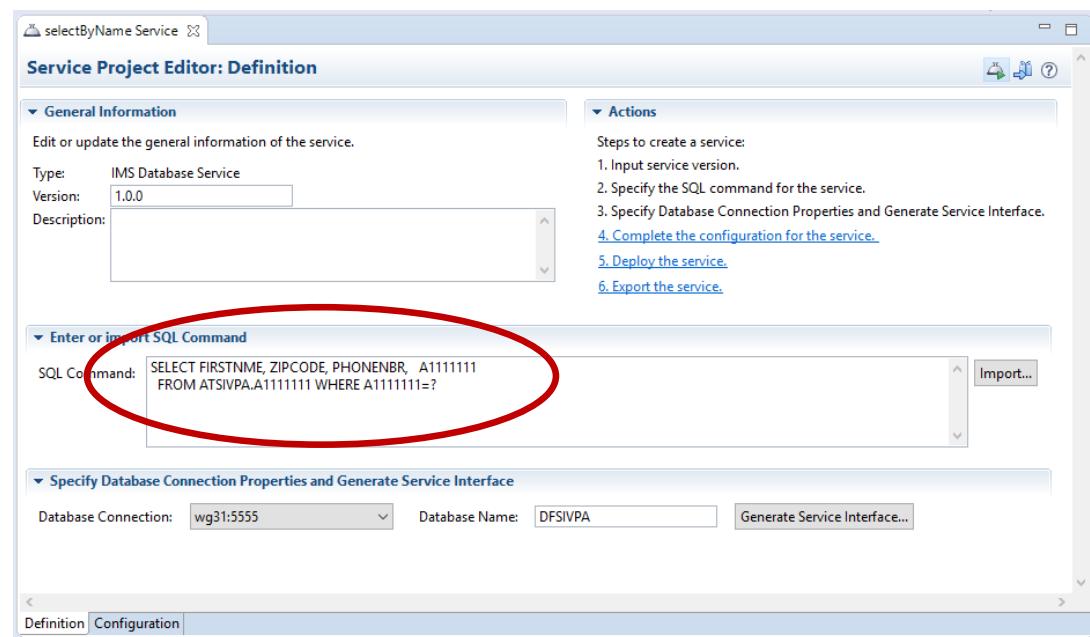
The screenshot displays two windows from the API toolkit:

- Service Interface Editor:** Shows a table of fields for a service named "minilnServiceRequest". A red box highlights the "Interface Rename" column for the "loan application" row, which contains the values "name", "credit score", "yearly income", "age", "loan amount", "aprov?", "effective date", "yearly interest rate", "yearly payment", "authorization identity", "MESSAGES\_NUM", and "disapproval message". A red circle highlights the "Include" checkbox for this row.
- Service Project Editor: Configuration:** Shows configuration settings for a service named "twoway Service". A red circle highlights the "jms/requestQueue" and "jms/replyQueue" fields, both of which have their JNDI names circled in red.

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.

# API toolkit – Creating Services for IMS DB

## Creating a service project from the IMS Catalog

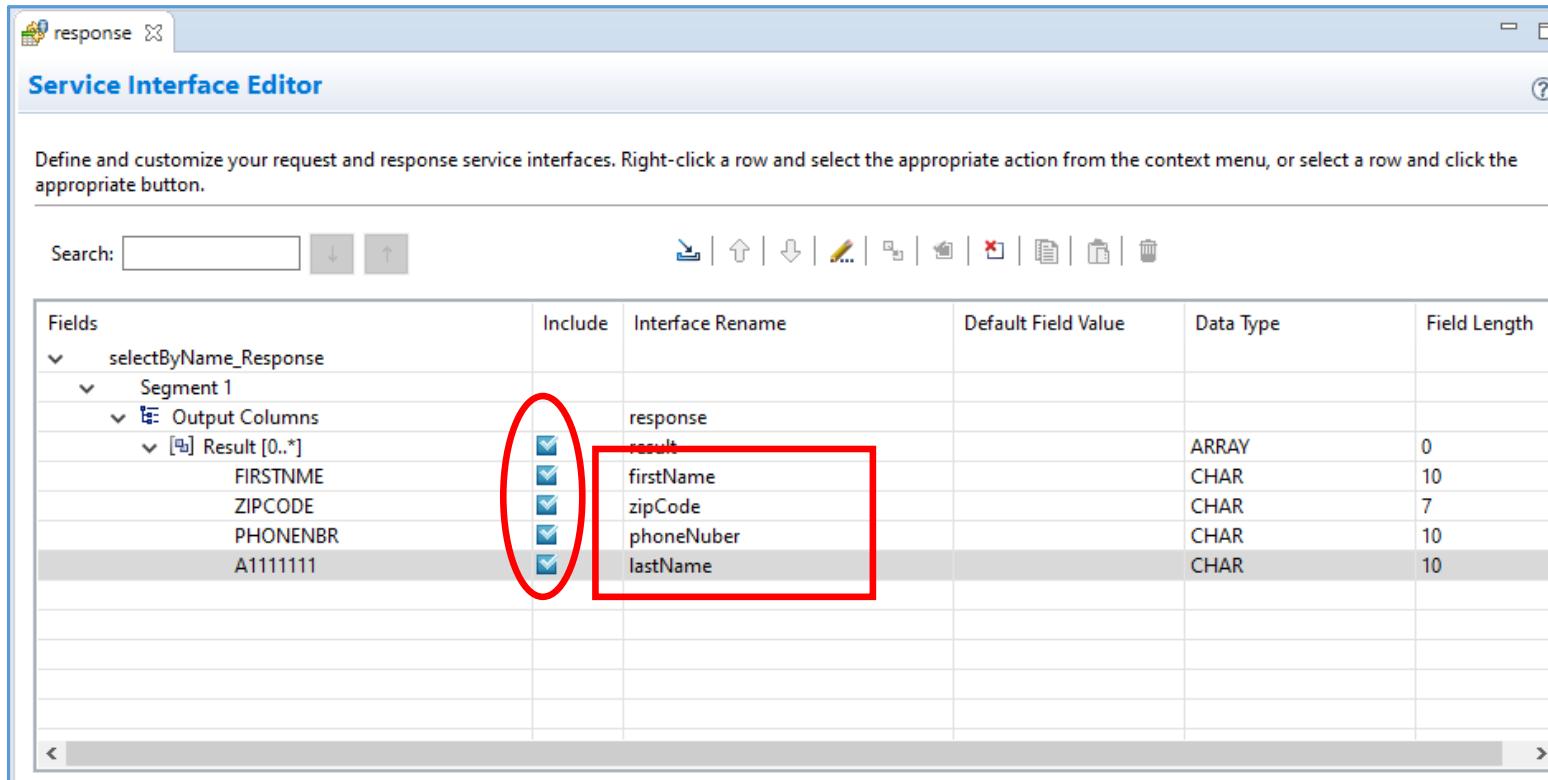


Use the IMS Catalog to assist with developing and testing SQL SELECT commands used for accessing IMS databases.

```
*-----*
* 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=DFSVVD1,DEVICE=3380,SIZE=2048
SEGM    NAME=A1111111,PARENT=0,BYTES=40,RULES=(LLV,LAST),
        PTR=(TB,CTR)
FIELD   NAME=(A1111111,SEQ,U),BYTES=010,START=00001,TYPE=C
FIELD   NAME=FIRSTNME,BYTES=010,START=00011,TYPE=C
FIELD   NAME=PHONENBR,BYTES=010,START=00021,TYPE=C
FIELD   NAME=ZIPCODE,BYTES=7,START=00031,TYPE=C
LCHILD  NAME=(A1,IVPDB1I),POINTER=INDX,RULES=LAST
DBDGEN
FINISH
END
```

## API toolkit – Creating Services for IMS DB

The Toolkit allows editing a service interface definitions\*



The screenshot shows the Service Interface Editor window. The title bar says "Service Interface Editor". The main area has a heading "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 this is a search bar and a set of toolbar icons. The main content is a table with the following columns: Fields, Include, Interface Rename, Default Field Value, Data Type, and Field Length. The table data is as follows:

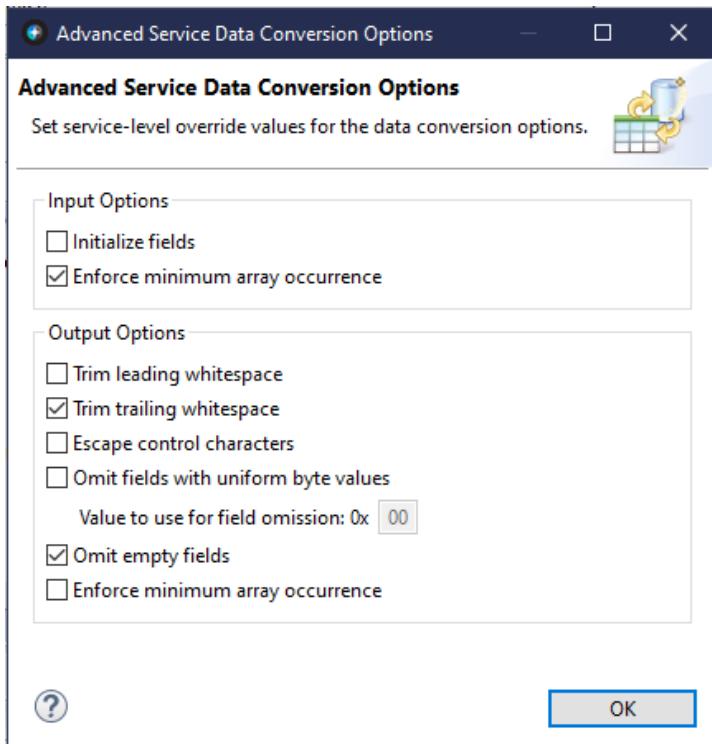
Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length
selectByName_Response					
Segment 1					
Output Columns					
Result [0..*]					
FIRSTMNAME	<input checked="" type="checkbox"/>	response			
ZIPCODE	<input checked="" type="checkbox"/>	result		ARRAY	0
PHONENBR	<input checked=""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

\*Using a slightly different process



z/OS Connect EE

# 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", FIRSTNAME AS "firstName",
         MIDINIT AS "middleInitial", LASTNAME AS "lastName",
         WORKDEPT AS "department", PHONENO AS "phoneNumber",
         JOB AS "job"
    FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN DD *
DSN SYSTEM(DSN2)
BIND SERVICE(SYSIBMSERVICE) -
NAME("selectEmployee") -
SQLENCODING(1047) -
DESCRIPTION('Select an employee from table USER1.EMPLOYEE')
```

 Import Db2 service from service manager

Db2 service manager connection:  wg31:2446

Type to search...

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

Definition Configuration

Import Cancel

 \*selectEmployee Service

### Service Project Editor: Definition

**General Information**  
Edit or update the general information of the service.

Type: Db2 Service  
Version: 1.0.0  
Description:

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

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

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

The service developer retrieves details about the Db2 REST services

Note there is no service interface editor available

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



z/OS Connect EE

The screenshot shows the IBM z/OS Connect Enterprise Edition interface. On the left is the Project Explorer, which contains a service named 'InquireSingle'. The main window displays the 'Overview' tab of the 'InquireSingle Service' configuration. Under 'General Information', the 'Type' is set to 'CICS COMMAREA Service' and 'Version' is '1.0.0'. The 'Program' field contains 'DFH0XLMN'. In the 'Actions' section, steps for creating a service are listed: 1. Input service version, 2. Specify program or transaction code for the service, 3. Create or import a service interface for the request and response in your service, 4. Complete the configuration for the service, and 5. Export the service. Below this, under 'Define Request and Response Service Interfaces', there are fields for 'Request service interface' (set to 'DFH0XCP4.si') and 'Response service interface' (set to 'DFH0XCPA.si'). A context menu is open over the 'Request service interface' field, with options 'Create Service Interface...' and 'Import Service Interface...'. To the right of the main window, a 'Deploy Service' dialog box is displayed. It shows the 'z/OS Connect EE Server' as 'wg31:9453'. The list of services to be created on the server includes:

Service name	Version	Type
inquireSingle	13.00	CICS COMMAREA Se...

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

Finally, deploy the service project as a  
**Service Archive file (.sar)**

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

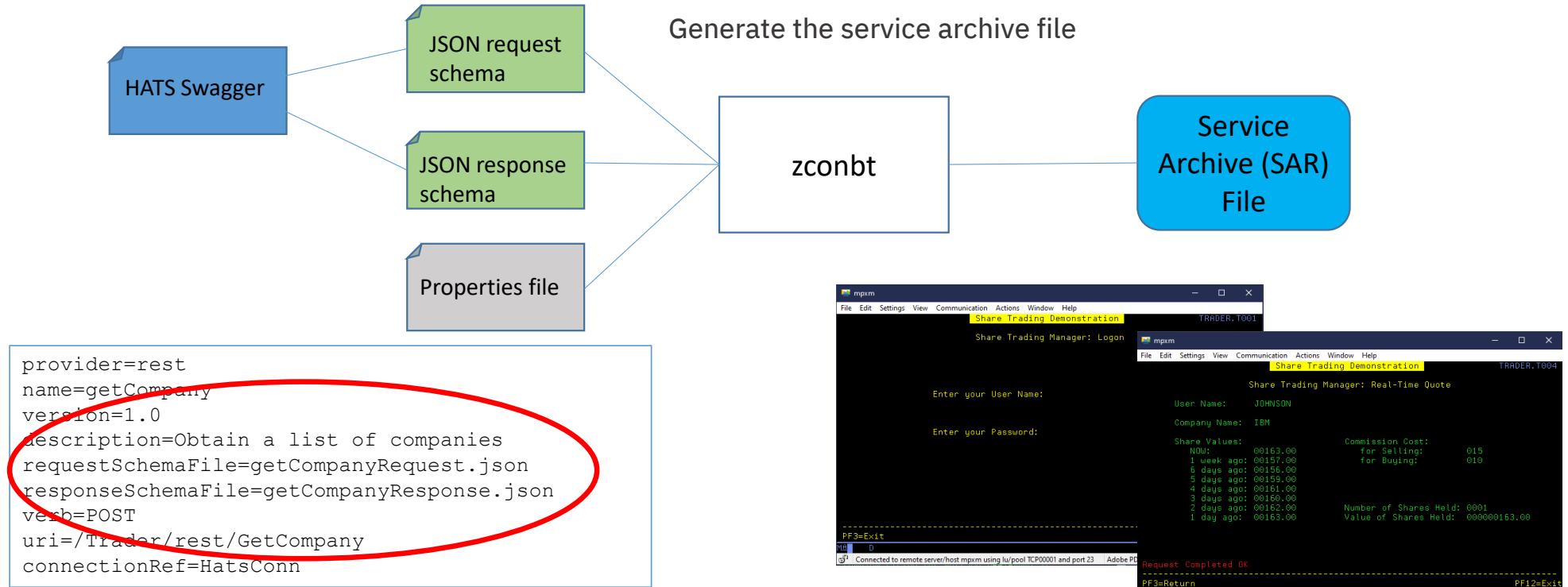


The screenshot shows the IBM z/OS Connect Enterprise Edition interface. On the left, the Project Explorer displays a service named 'InquireSingle'. The main window shows the 'Overview' tab of the 'InquireSingle Service' properties. Under 'General Information', the 'Type' is set to 'CICS COMMAREA Service' and 'Version' to '1.0.0'. The 'Program' field contains 'DFH0XLMN'. In the 'Actions' section, steps for creating a service are listed: 1. Input service version, 2. Specify program or transaction code for the service, 3. Create or import a service interface for the request and response in your service, 4. Complete the configuration for the service, and 5. Export the service. Below this, under 'Define Request and Response Service Interfaces', there are fields for 'Request service interface' (set to 'DFH0XCP4.si') and 'Response service interface' (set to 'DFH0XCPA.si'). A context menu is open over the 'Export...' option in the 'z/OS Connect EE' submenu, showing options like 'Deploy Service to z/OS Connect EE Server' and 'Export z/OS Connect EE Service Archive'. A separate 'Export Service Package' dialog box is overlaid on the interface, prompting the user to select where to export the service package. The dialog has fields for 'Export service package to:' (radio buttons for 'Workspace' and 'Local file system', with 'Workspace' selected), 'Folder:' (set to '/services'), 'File name:' (set to 'inquireSingle'), and 'Overwrite service package file' (checkbox). Buttons for '?', 'OK', and 'Cancel' are at the bottom.

And export the service project as a **Service Archive file (.sar)**.

# Creating Services using zconbt – REST

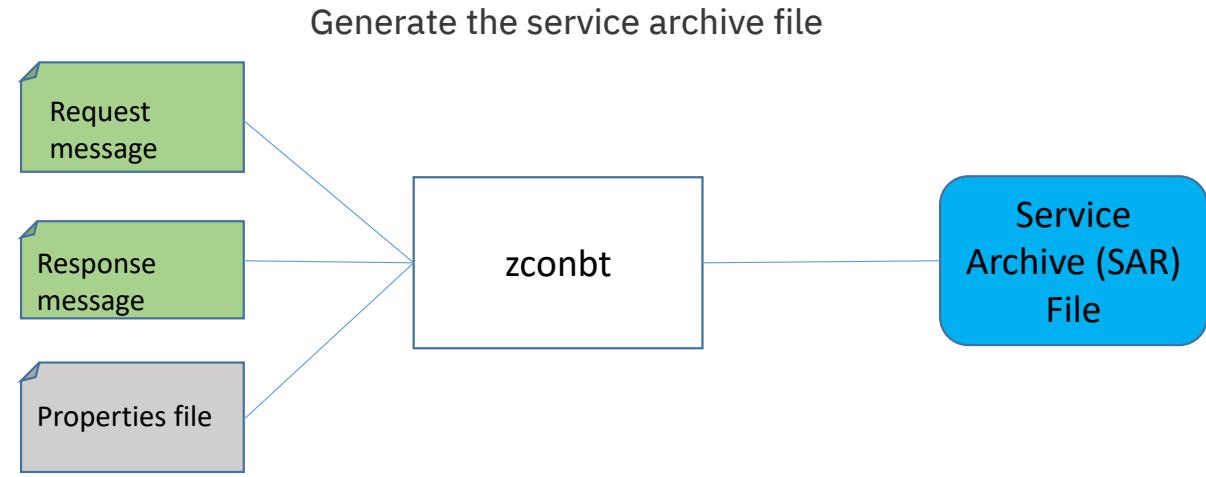
For HATS REST Services use the z/OS Connect Build toolkit (zconbt)



# Creating Services using zconbt – MVS Batch

For batch WOLA services use the z/OS Connect Build toolkit (zconbt)

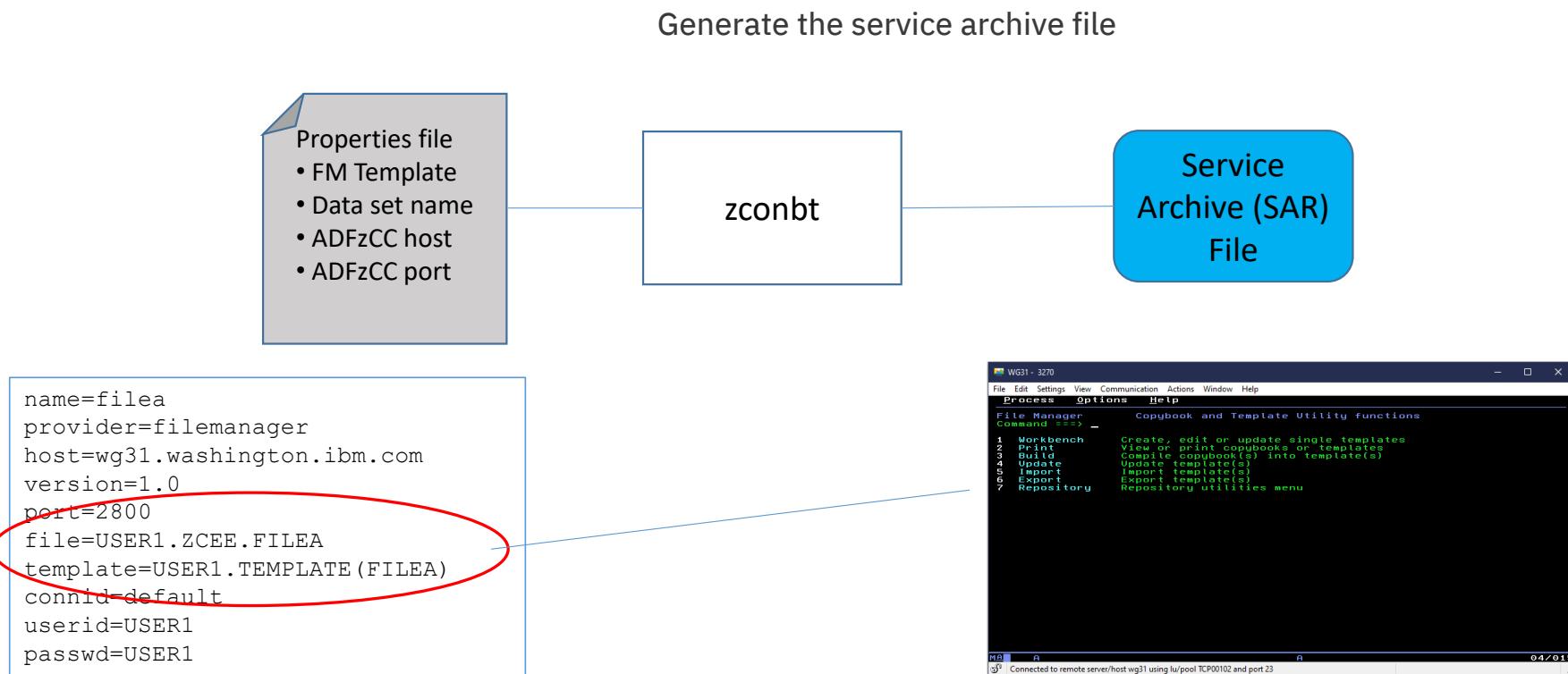
```
name=Filea
version=1.0
provider=wola
description=COBOL batch program
language=COBOL
program=ATSFILEA
register=FILEAZCON
connectionRef=wolaCF
requestStructure=fileareq.cpy
responseStructure=filearsp.cpy
```



**WebSphere Optimized Local Adapter** – a protocol for cross memory communications between address spaces

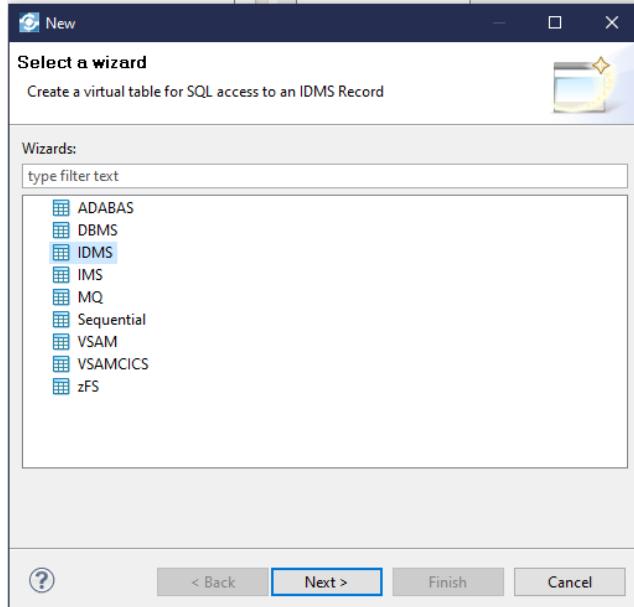
# Creating Services using zconbt – File Manager

For File Manager Services use the z/OS Connect Build toolkit (zconbt)



# Creating Services - DVM

For DVM use the DVM Studio



The screenshot shows the DVM Studio interface. The title bar says 'DV Data - Data Virtualization Manager/SQL Samples/Generated.sql - IBM Data Virtualization Manager for z/OS'. The top menu includes File, Edit, Navigate, Search, Project, SQL, Run, Window, Help. The left sidebar has sections for Data (Navigator, JDBC), Data Sources, Edit SQL, and Set Current Server. The main pane shows a tree view under 'Services' with 'Web Services' expanded, showing options like Create Directory, /REST/, INSERT, UPD, Set Tree Filter, Target System, WSC, and Admin. A context menu is open over the 'Web Services' node, with 'Generate SAR File(s)' highlighted with a red oval. To the right is a code editor window showing a SQL script named 'Generated.sql' with several INSERT and UPDATE statements. Below the code editor is a 'Server Trace' window showing a table with columns WS\_DESCRIPTION, WS\_DEPARTMENT, and WS\_C. The table has three rows: Mitch Johnson, 10, 002. The bottom right corner of the 'Generate SAR File(s)' button is also circled in red.

# **z/OS Resources accessible from DVM using SQL\***



**z/OS Connect EE**

Data Source	SELECT	INSERT	UPDATE	DELETE	CALL Statement
CA IDMS	Yes	No	No	No	N/A
CICS COMMAREA	N/A	N/A	N/A	N/A	Yes
DB2 for z/OS	Yes	Yes	Yes	Yes	Yes
Db2 Direct	Yes	No	No	No	N/A
Db2 other platforms	Yes	Yes	Yes	Yes	Yes
IMS DBCTL	Yes	Yes	Yes	Yes	N/A
IMS Direct	Yes	No	No	No	N/A
IMS OTMA	N/A	N/A	N/A	N/A	Yes
MQ	Yes	No	No	No	N/A
Natural	N/A	N/A	N/A	N/A	Yes
SMF	Yes	No	No	No	N/A
Sequential data set	Yes	Yes	No	No	N/A
VSAM data set*	Yes	Yes	Yes	Yes	N/A
z/OS Syslog	Yes	No	No	No	N/A
OMVS files	Yes	No	No	No	Yes

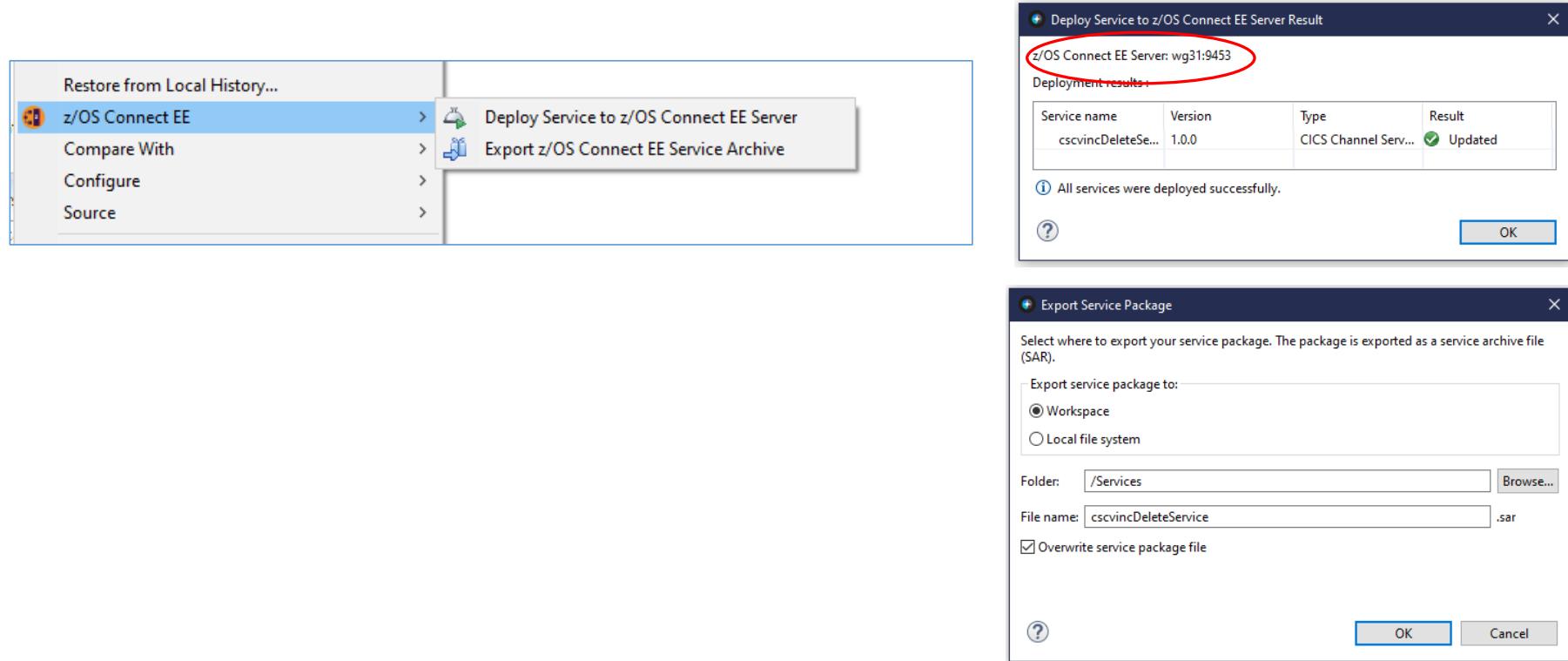
\*Administrating IBM DVM Manager, SC27-9303

© 2018, 2021 IBM Corporation

# API toolkit – Services Editor

## Server connection and Services deployment

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



The screenshot shows the API toolkit interface with the "z/OS Connect EE" server selected in the "Host Connections" view. A context menu is open, showing options like "Deploy Service to z/OS Connect EE Server" and "Export z/OS Connect EE Service Archive". To the right, two dialogs are displayed: "Deploy Service to z/OS Connect EE Server Result" and "Export Service Package".

**Deploy Service to z/OS Connect EE Server Result**

z/OS Connect EE Server: wg31:9453

Deployment results:

Service name	Version	Type	Result
cscvincDeleteSe...	1.0.0	CICS Channel Serv...	Updated

All services were deployed successfully.

**Export Service Package**

Select where to export your service package. The package is exported as a service archive file (SAR).

Export service package to:

Workspace

Local file system

Folder: /Services

File name: cscvincDeleteService.sar

Overwrite service package file



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

Quick and easy **API mapping**.

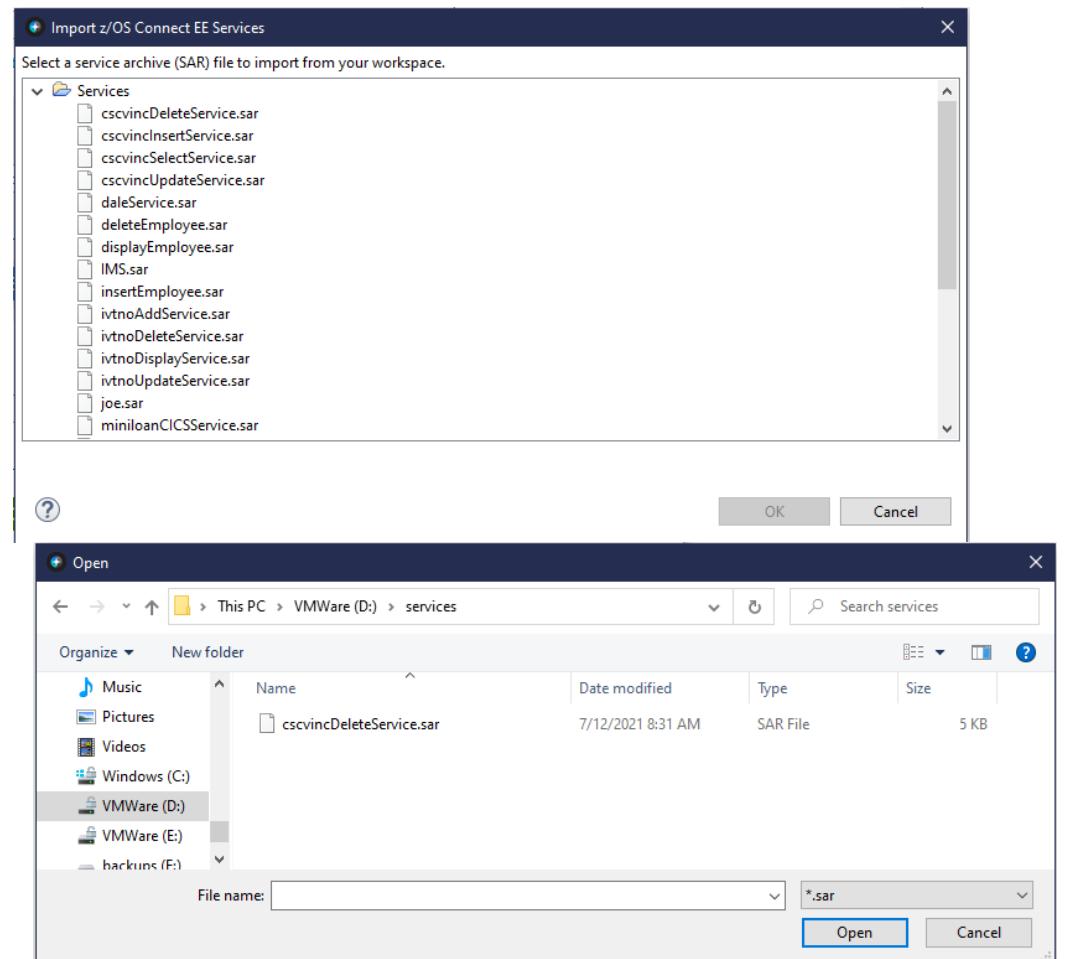
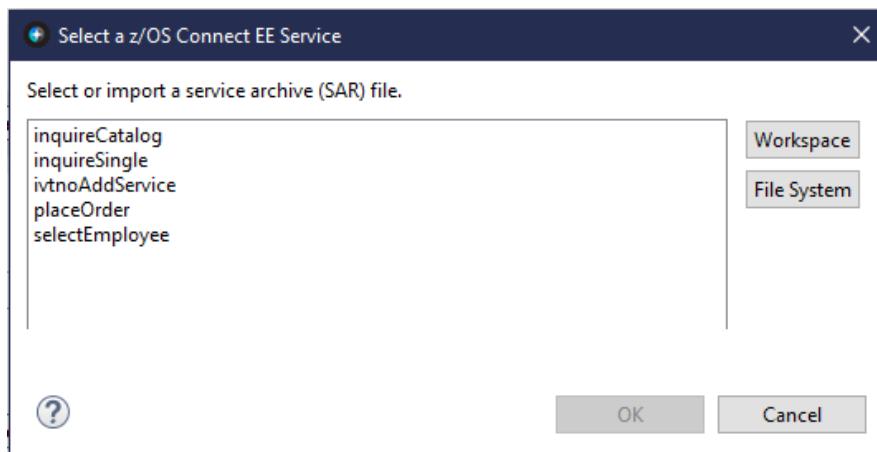
*Remember: All service archives files are functionally equivalent regardless of how they are created*



## /api\_toolkit/api\_editor

Quick and easy **API mapping**.

# Importing the service archives files



mitchj@us.ibm.com

© 2018, 2021 IBM Corporation



# API toolkit – API Editor

The screenshot shows the API Editor interface with two API definitions:

- catalog API**:
  - Name:** catalog
  - Description:** (empty)
  - Base path:** /catalog
  - Version:** 1.0.0
  - Path:** /items{startItemID}
  - Methods (2):**
    - ▶ **GET** inquireCatalog (Service... Mapping...) (with icons for up, down, and delete)
    - ▶ **PUT** ivtnoAddService (Service... Mapping...) (with icons for up, down, and delete)
- order API**:
  - Path:** /order
  - Methods (2):**
    - ▶ **POST** placeOrder (Service... Mapping...) (with icons for up, down, and delete)
    - ▶ **PUT** selectEmployee (Service... Mapping...) (with icons for up, down, and delete)

mitchj@us.ibm.com

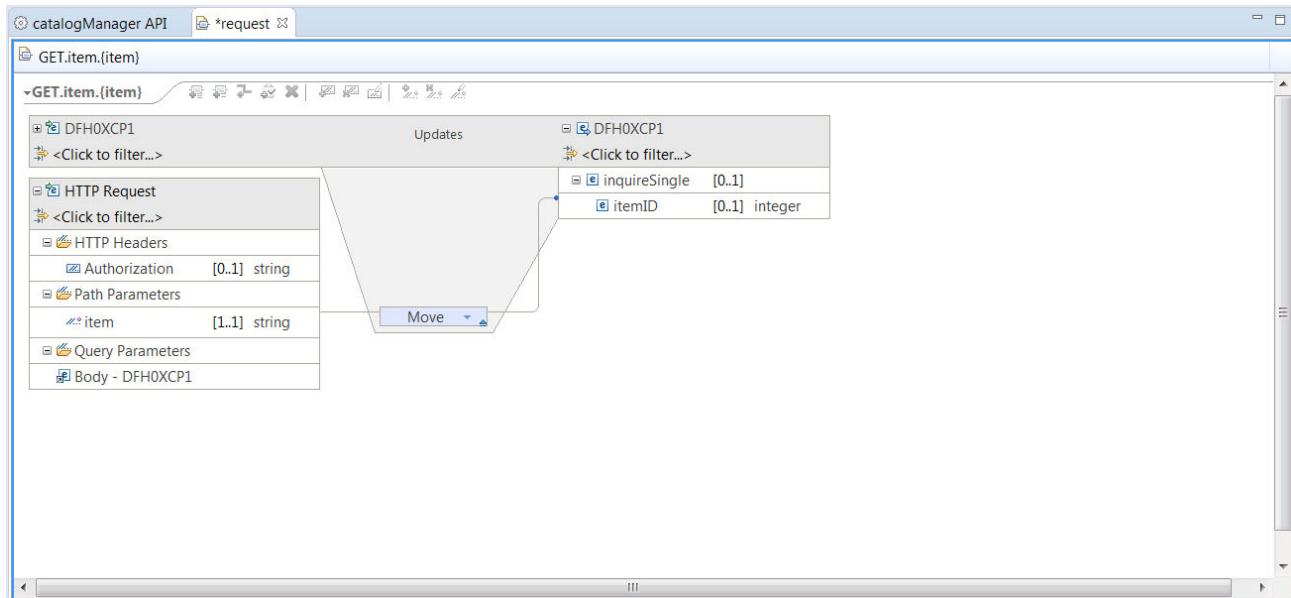
The **API toolkit** is designed to encourage RESTful API design.

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

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

# API toolkit – API Editor

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



Map both the request and response for each API.

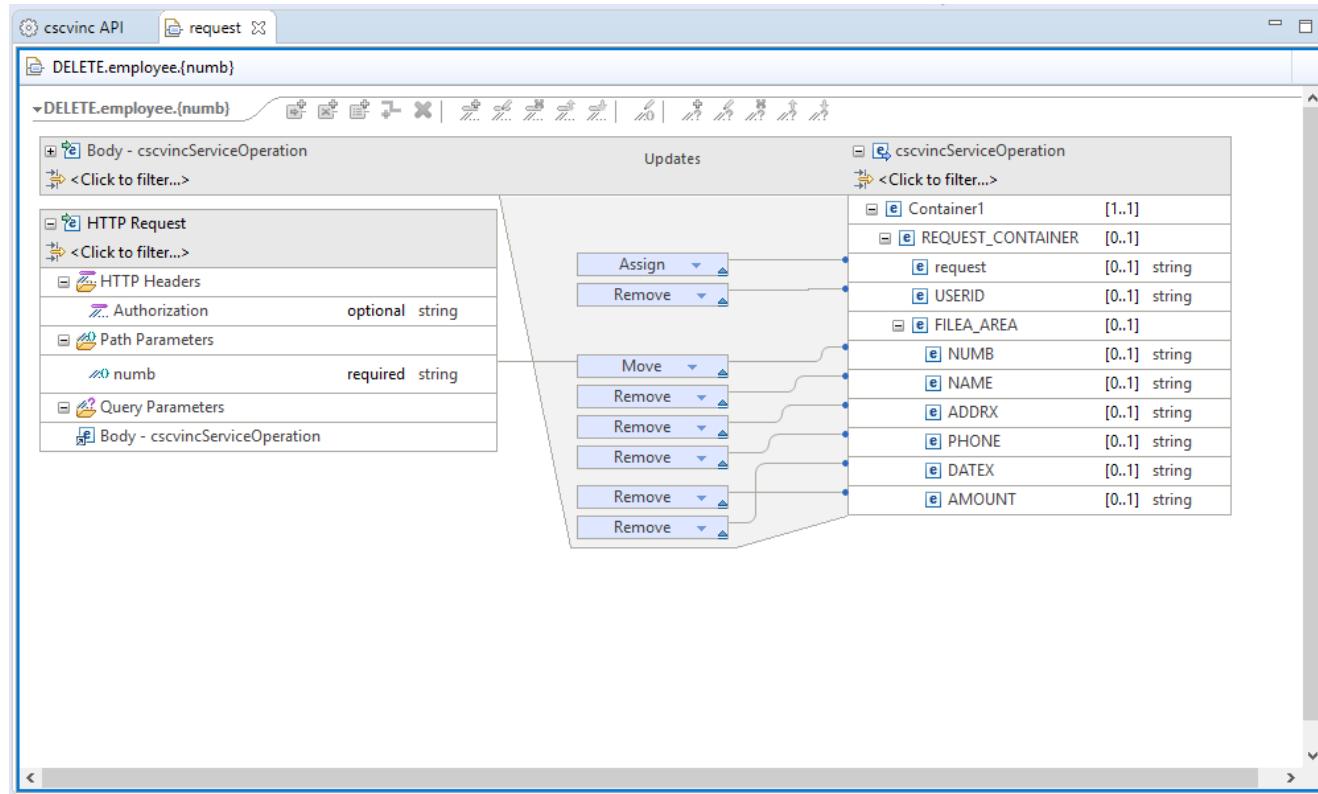
Map path and query parameters to native data structures.

Assign static values to fields, useful for Op codes.

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

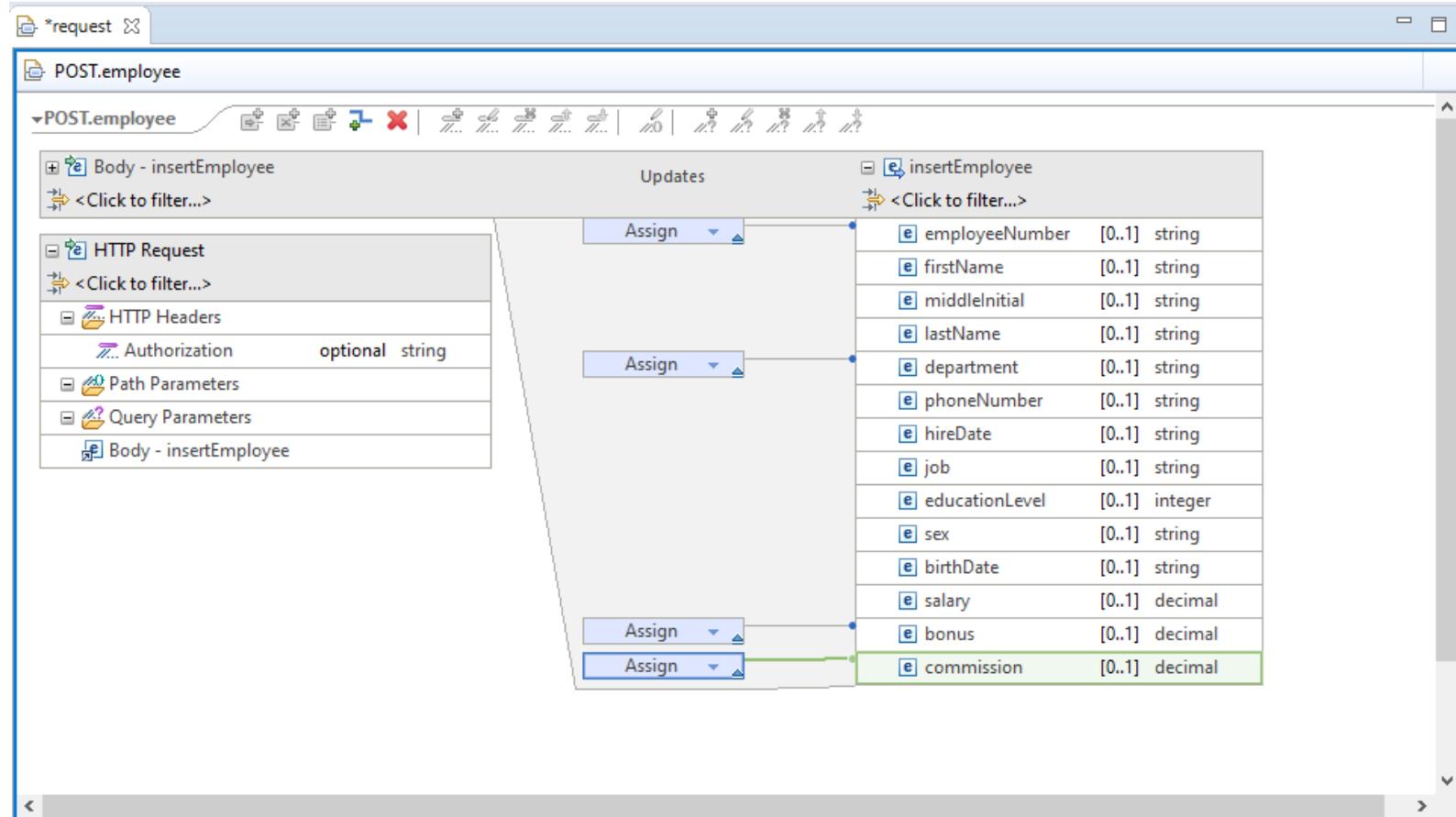
# API toolkit – API Editor

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



# API toolkit – API Editor and Db2 REST service

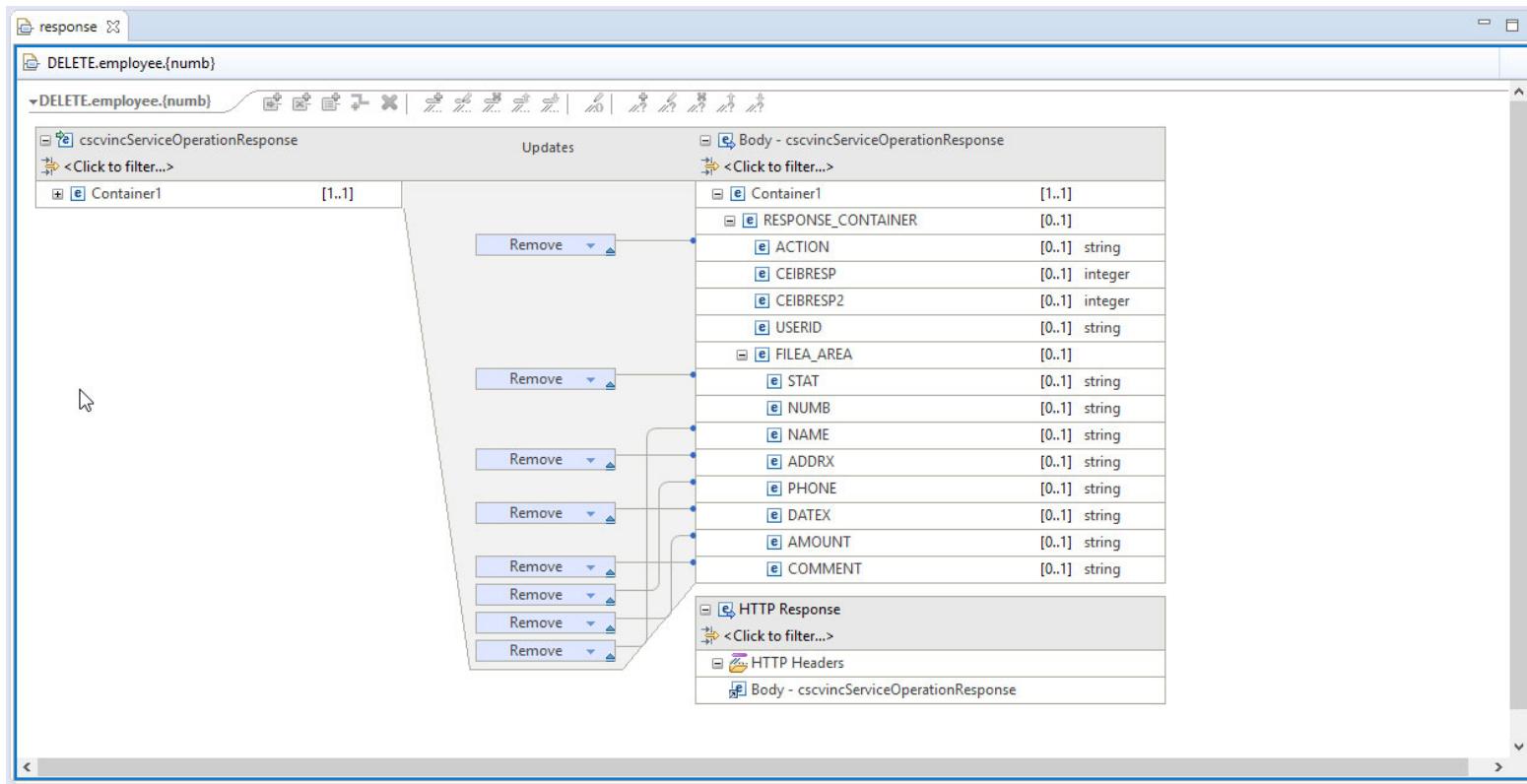
API mapping: Remove/Assign values columns exposed in Db2 REST service



# API toolkit – API Editor

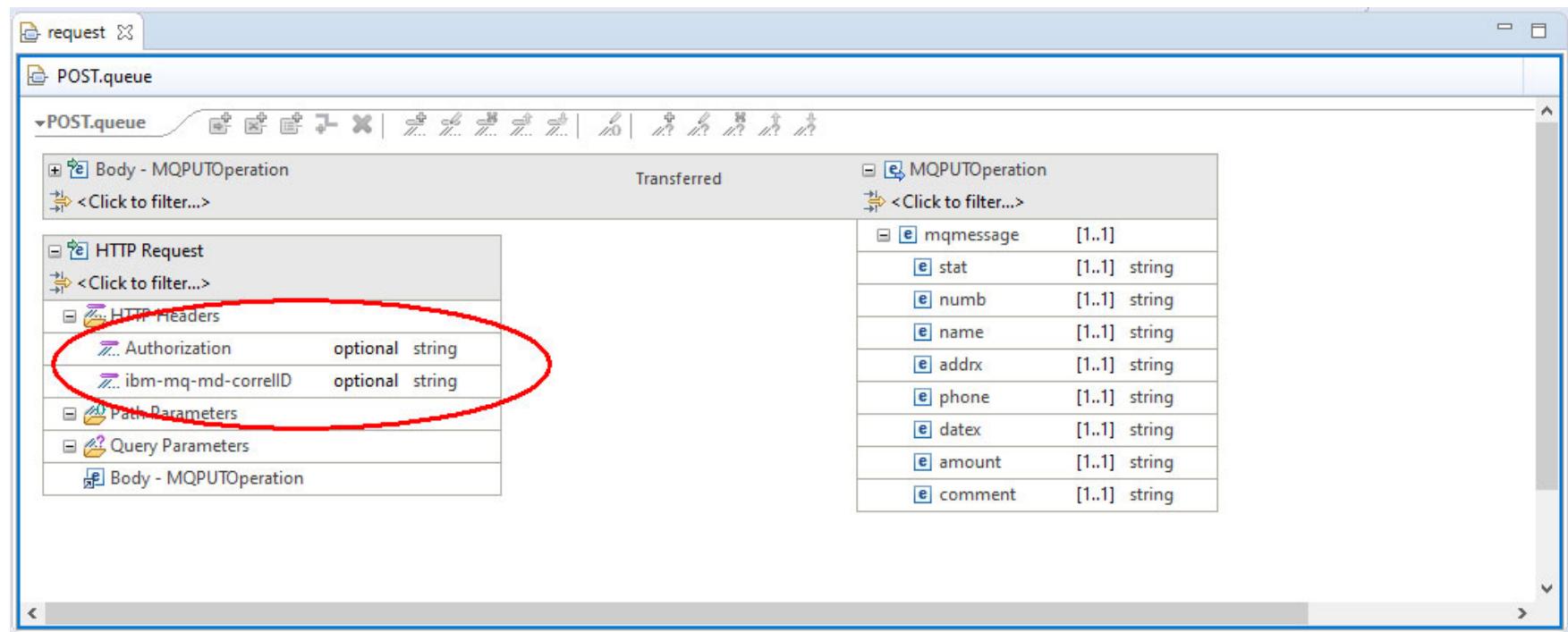


**API mapping:** Allows the API Developer to remove fields from the response to tailor the API



# API toolkit – API Editor

API mapping: Allows adding HTTP header properties



The screenshot shows the API Editor interface with two main sections: 'POST.queue' on the left and 'MQPUTOperation' on the right.

**POST.queue (Left):**

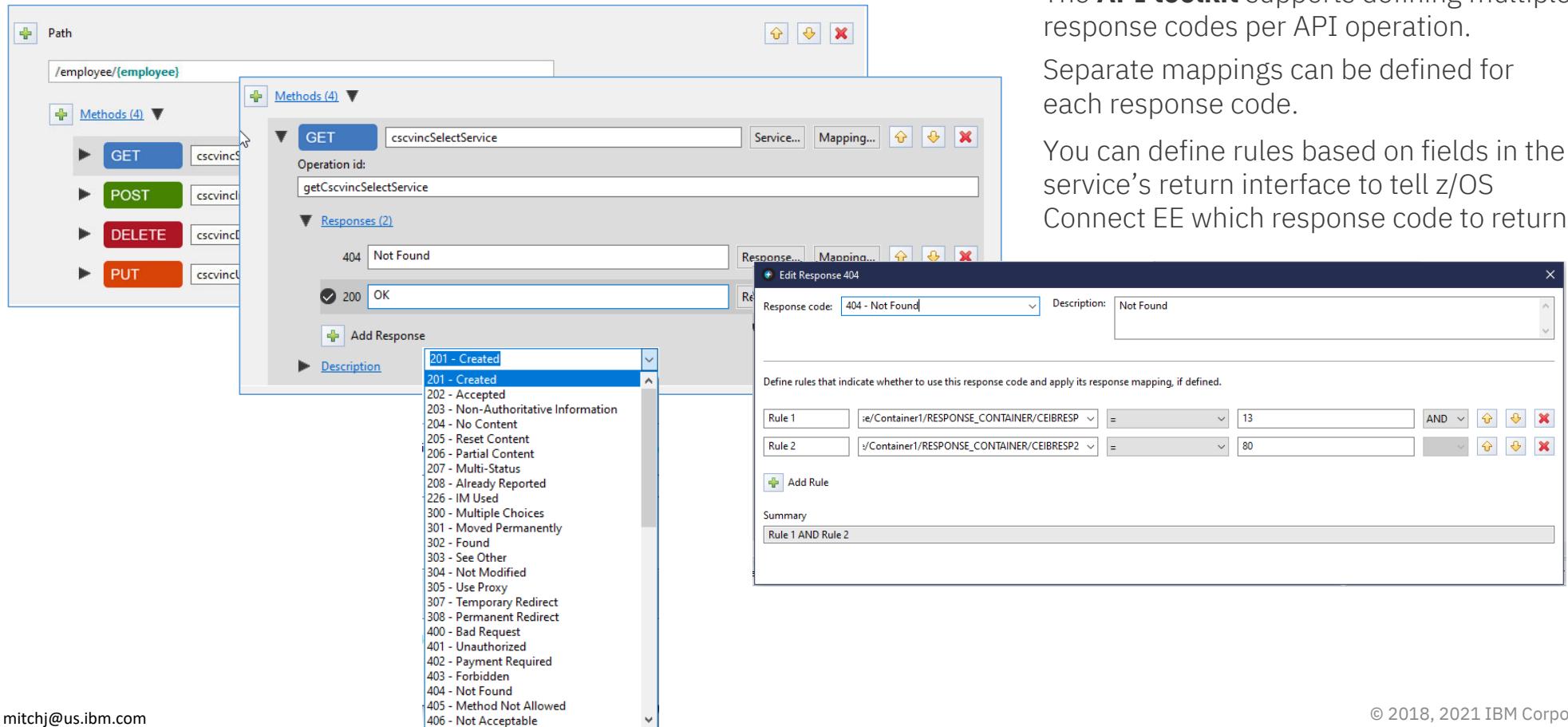
- Body - MQPUTOperation:** Contains a link to 'Click to filter...'.
- HTTP Request:** Contains a link to 'Click to filter...'.
  - HTTP Headers:** Contains two entries circled in red:
    - ... Authorization optional string
    - ... ibm-mq-md-correlID optional string
  - Path Parameters:**
  - Query Parameters:**
  - Body - MQPUTOperation:**

**MQPUTOperation (Right):**

- MQPUTOperation:** Contains a link to 'Click to filter...'.
- mqmessage:** [1..1] string
  - stat [1..1] string
  - numb [1..1] string
  - name [1..1] string
  - addrx [1..1] string
  - phone [1..1] string
  - datex [1..1] string
  - amount [1..1] string
  - comment [1..1] string

# 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 selected. Under the `Methods (4)` section, a `GET` method is highlighted, pointing to the service `cscvincSelectService`. The `getCscvincSelectService` operation is defined. In the `Responses (2)` section, two responses are listed: `404 Not Found` and `200 OK`. A context menu is open over the `404 Not Found` response, displaying the `Edit Response 404` dialog. This dialog shows the response code as `404 - Not Found` and the description as `Not Found`. It also contains a rule editor with two rules:

- `Rule 1`: `se/Container1/RESPONSE_CONTAINER/CEIBRESP` = 13
- `Rule 2`: `z/Container1/RESPONSE_CONTAINER/CEIBRESP2` = 80

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

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

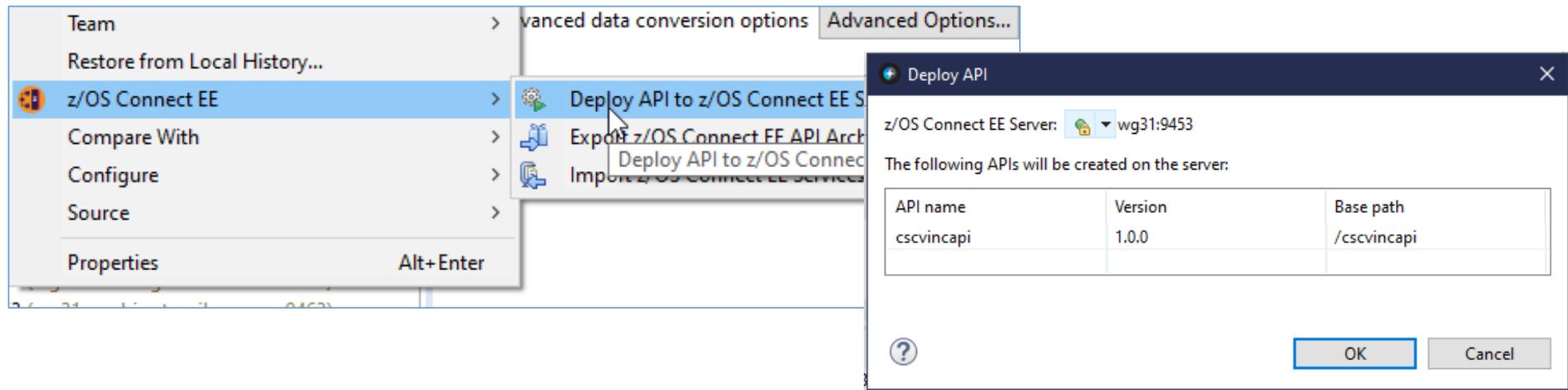
Separate mappings can be defined for each response code.

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



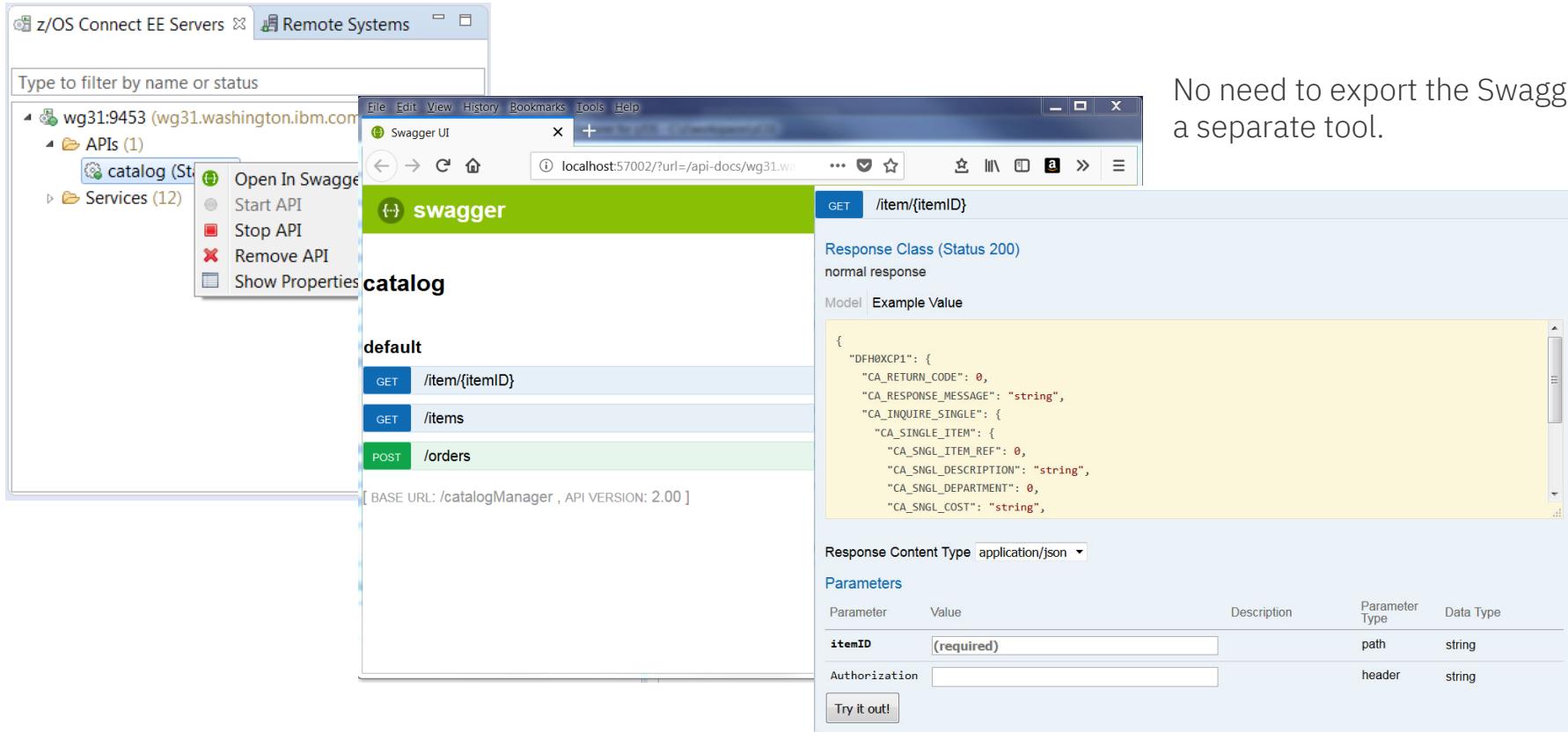
**Right-click deploy to server** enables developers to quickly deploy, test, and iterate on their APIs.

**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.



The screenshot shows the z/OS Connect EE interface with the "Remote Systems" tab selected. In the left sidebar, under "APIs (1)", there is a "catalog" entry with a context menu open, showing options like "Open In Swagger". A modal window titled "Swagger UI" is displayed over the main interface, showing the Swagger documentation for the "catalog" endpoint. The "catalog" section contains three operations: "GET /item/{itemID}", "GET /items", and "POST /orders". The "POST /orders" operation is highlighted with a green background. The "Model" tab is selected, displaying a JSON schema for the response:

```
{  
  "DFH0XCP1": {  
    "CA_RETURN_CODE": 0,  
    "CA_RESPONSE_MESSAGE": "string",  
    "CA_INQUIRE_SINGLE": {  
      "CA_SINGLE_ITEM": {  
        "CA_SNGL_ITEM_REF": 0,  
        "CA_SNGL_DESCRIPTION": "string",  
        "CA_SNGL_DEPARTMENT": 0,  
        "CA_SNGL_COST": "string",  
      }  
    }  
}
```

The "Parameters" section shows two parameters: "itemID" (required, path, string) and "Authorization" (header, string). A "Try it out!" button is present at the bottom of the modal.

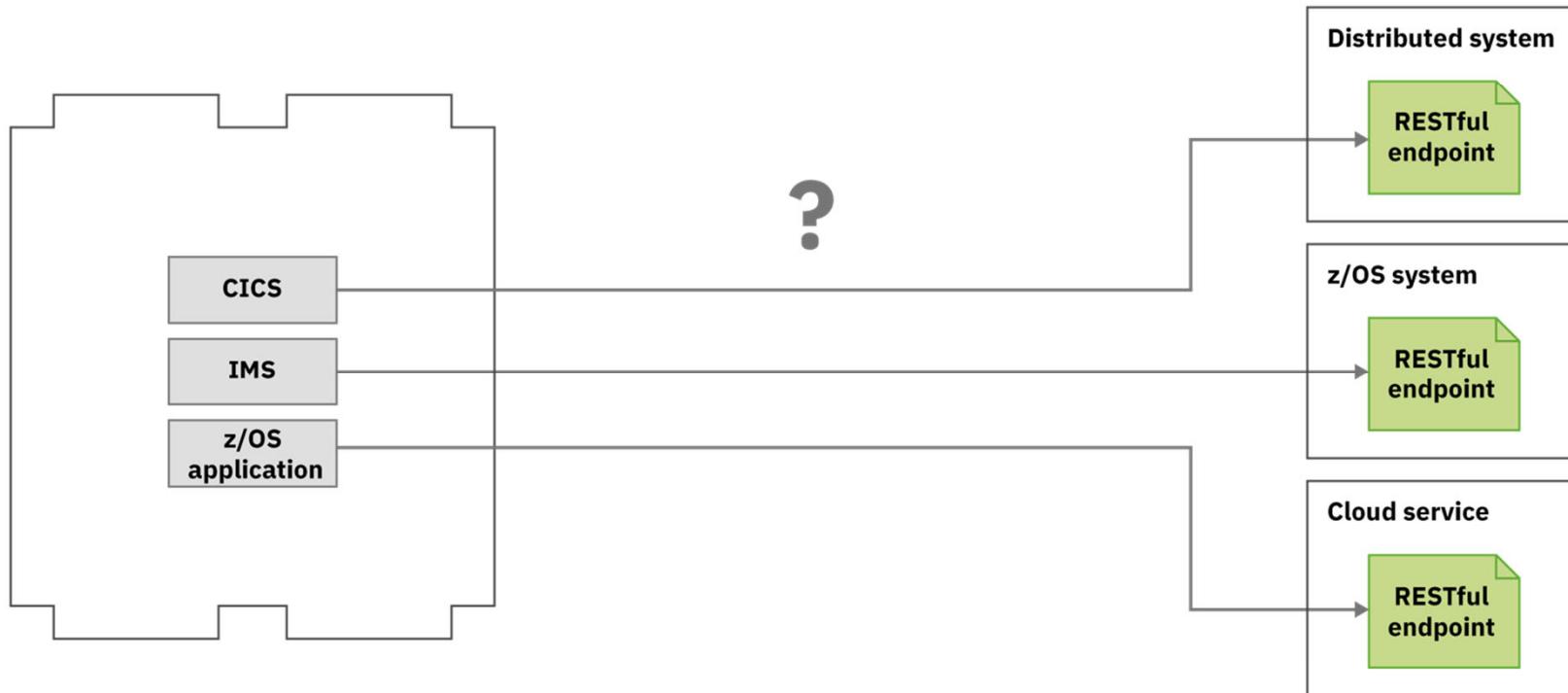
No need to export the Swagger doc to a separate tool.



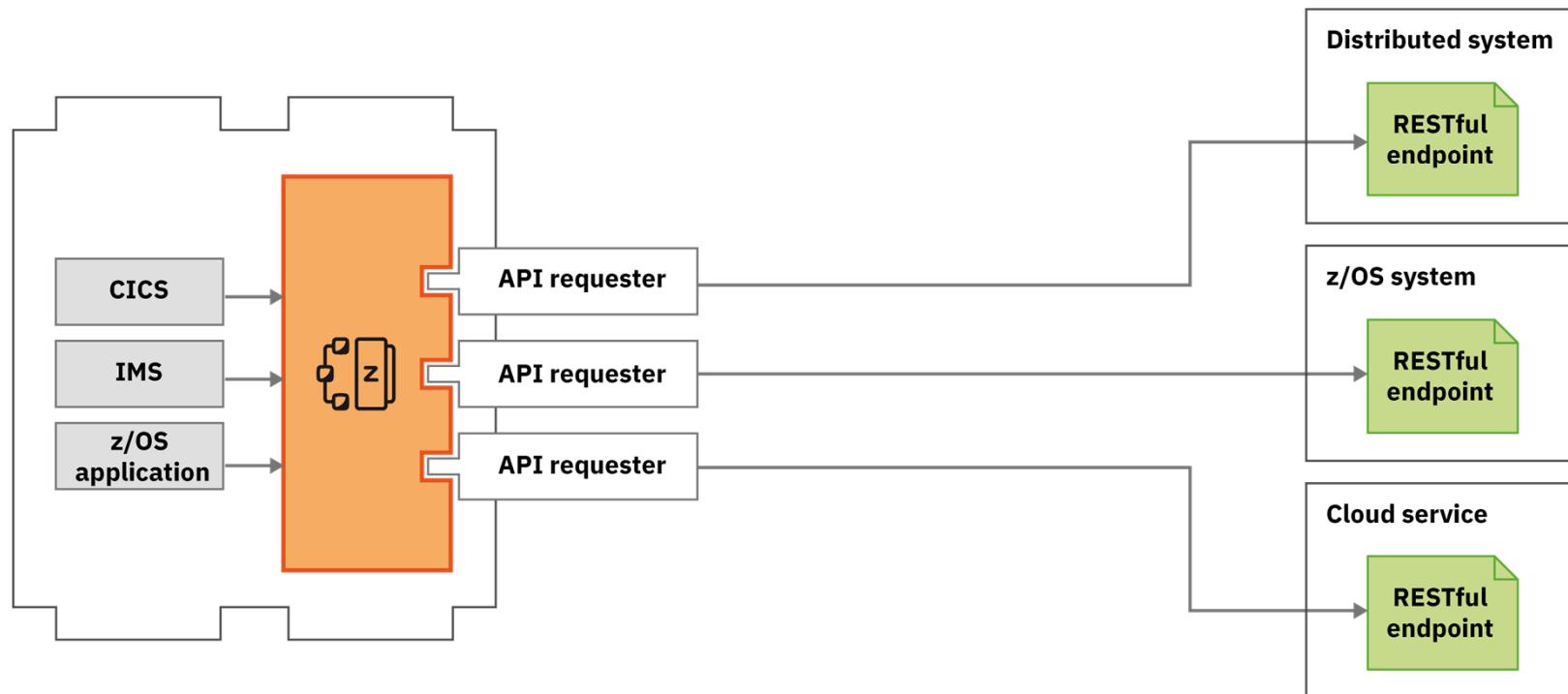
# **/api\_toolkit/apiRequesters**

Quick and easy **API mapping**.

# What about calling external APIs from my z/OS assets?



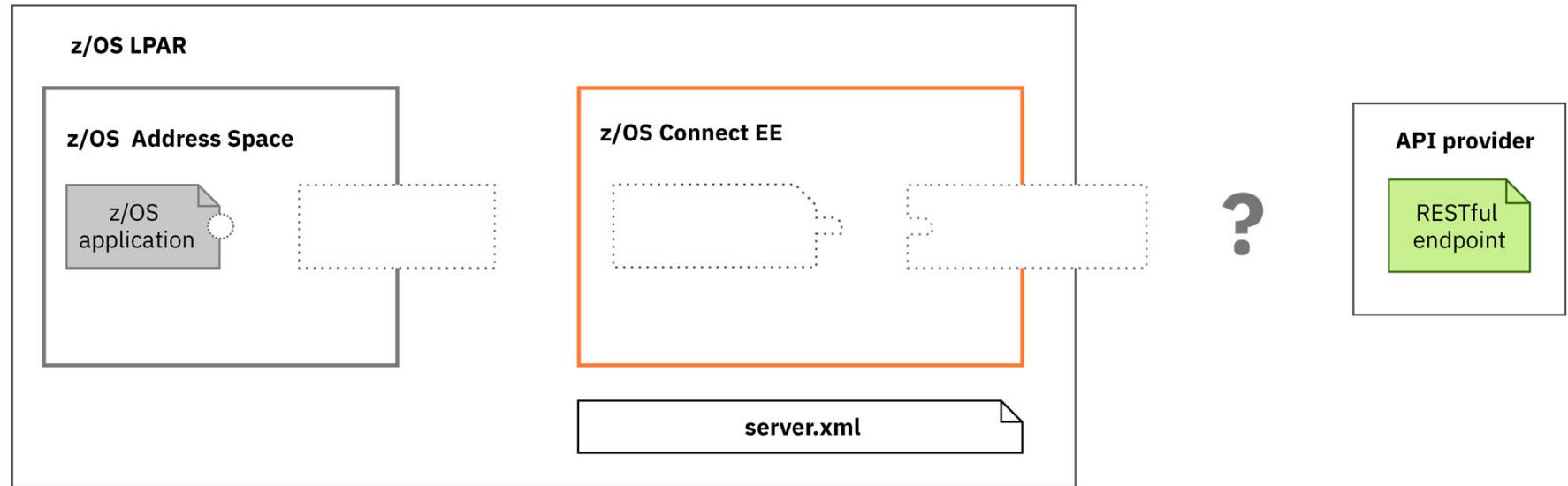
# Use API requester to call external APIs from z/OS assets





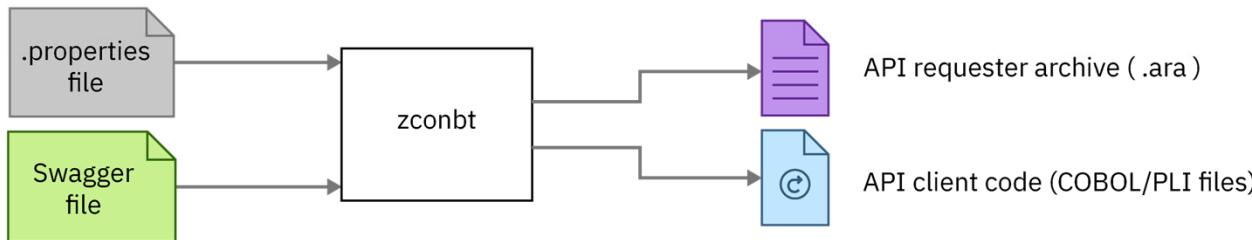
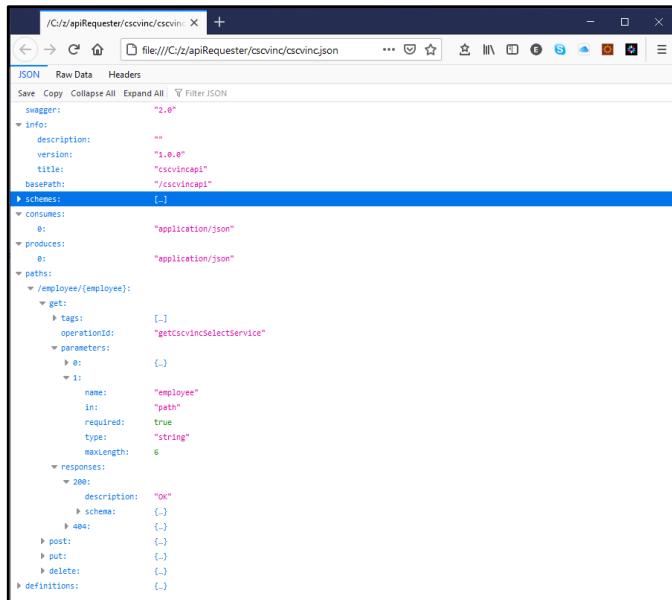
# Steps to calling an external API

Starting point



# Steps to calling an external API

## Step 1. Generate API requester archive and API client code from Swagger

A screenshot of a JSON editor window showing a Swagger API definition. The URL is `file:///C:/z/apiRequester/cscvinc/cscvinc.json`. The JSON structure includes:

```

{
  "swagger": "2.0",
  "info": {
    "description": "",
    "version": "1.0.0",
    "title": "cscvincapi",
    "basePath": "/cscvincapi"
  },
  "schemes": [...],
  "consumes": [...],
  "produces": [...],
  "paths": {
    "/employee/{employee}": {
      "get": {
        "tags": [...],
        "operationId": "getcscvincSelectService",
        "parameters": [
          {
            "name": "employees",
            "in": "path",
            "required": true,
            "type": "string",
            "maxLength": 6
          }
        ],
        "responses": {
          "200": {
            "description": "OK",
            "schema": {...}
          },
          "404": {...}
        }
      }
    }
  }
}
  
```

**.properties file#**

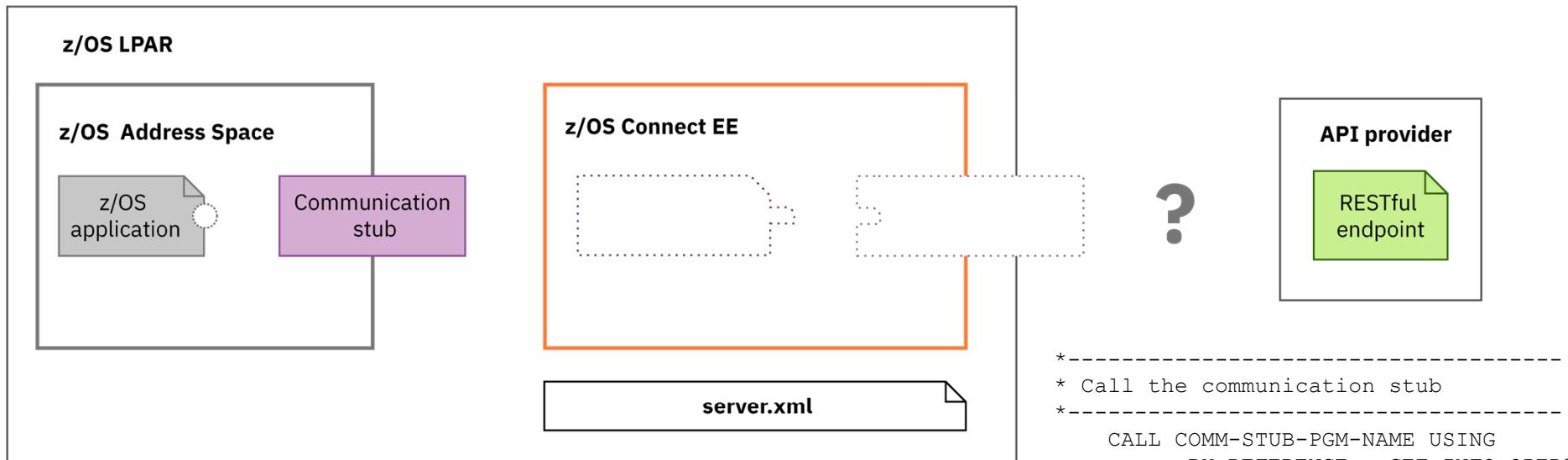
```

apiDescriptionFile=./cscvinc.json
dataStructuresLocation=./syslib
apiInfoFileLocation=./syslib
logFileDirectory=./logs
language=COBOL
connectionRef=cscvincAPI
requesterPrefix=csc
  
```

#Additional property file attributes, e.g., `defaultCharacterMaxLength`, etc. are described at  
**The build toolkit properties file** article at URL  
<https://www.ibm.com/docs/en/zosconnect/3.0?topic=toolkit-build-properties-file>

# Steps to calling an external API

## Step 2. Configure communication stub



Configure a communication stub.

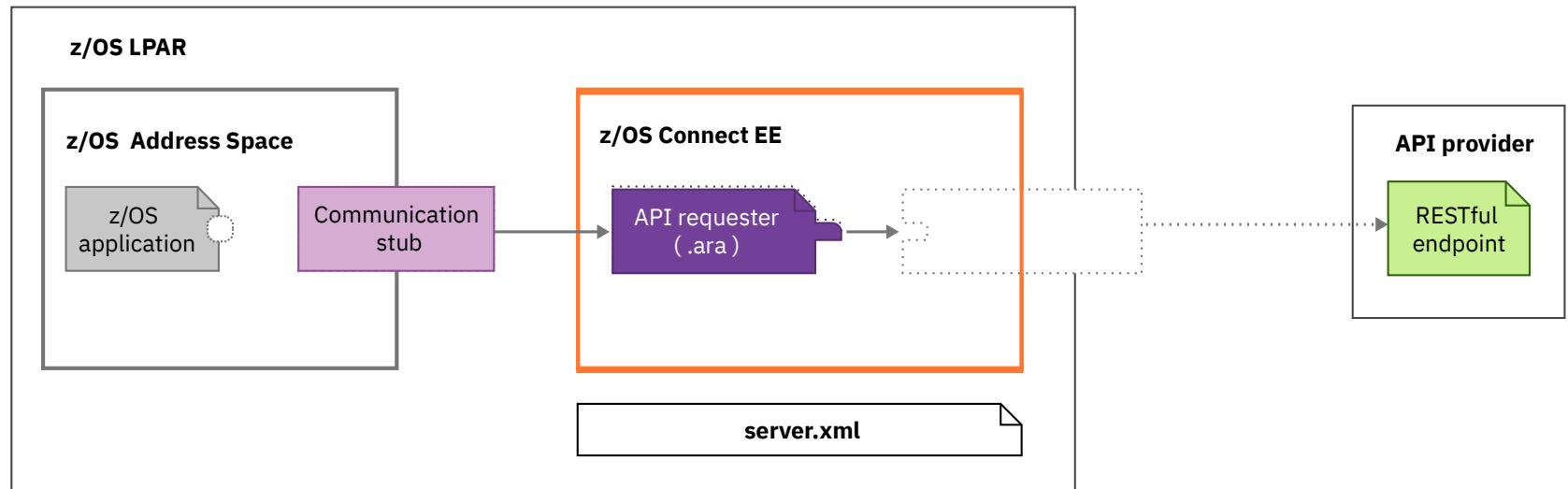
- For CICS region systems using URIMAP resources
- For non CICS client the configuration is done via environment variables

 [ibm.biz/zosconnect-configure-comms-stub](http://ibm.biz/zosconnect-configure-comms-stub)

```
*-----  
* Call the communication stub  
*-----  
CALL COMM-STUB-PGM-NAME USING  
      BY REFERENCE      GET-INFO-OPER1  
      BY REFERENCE      BAQ-REQUEST-INFO  
      BY REFERENCE      BAQ-REQUEST-PTR  
      BY REFERENCE      BAQ-REQUEST-LEN  
      BY REFERENCE      BAQ-RESPONSE-INFO  
      BY REFERENCE      BAQ-RESPONSE-PTR  
      BY REFERENCE      BAQ-RESPONSE-LEN.
```

# Steps to calling an external API

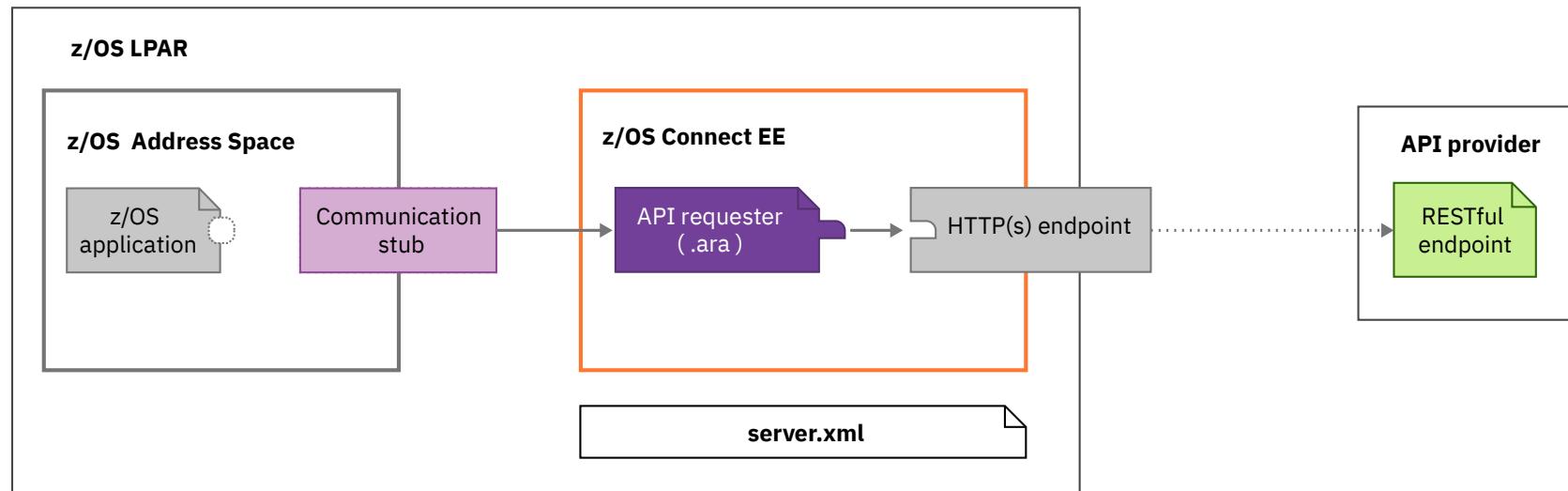
## Step 3. Deploy API requester (.ara) archive



Deploy your API requester archive to the *apiRequesters* directory.

# Steps to calling an external API

## Step 4. Configure HTTP(S) endpoint

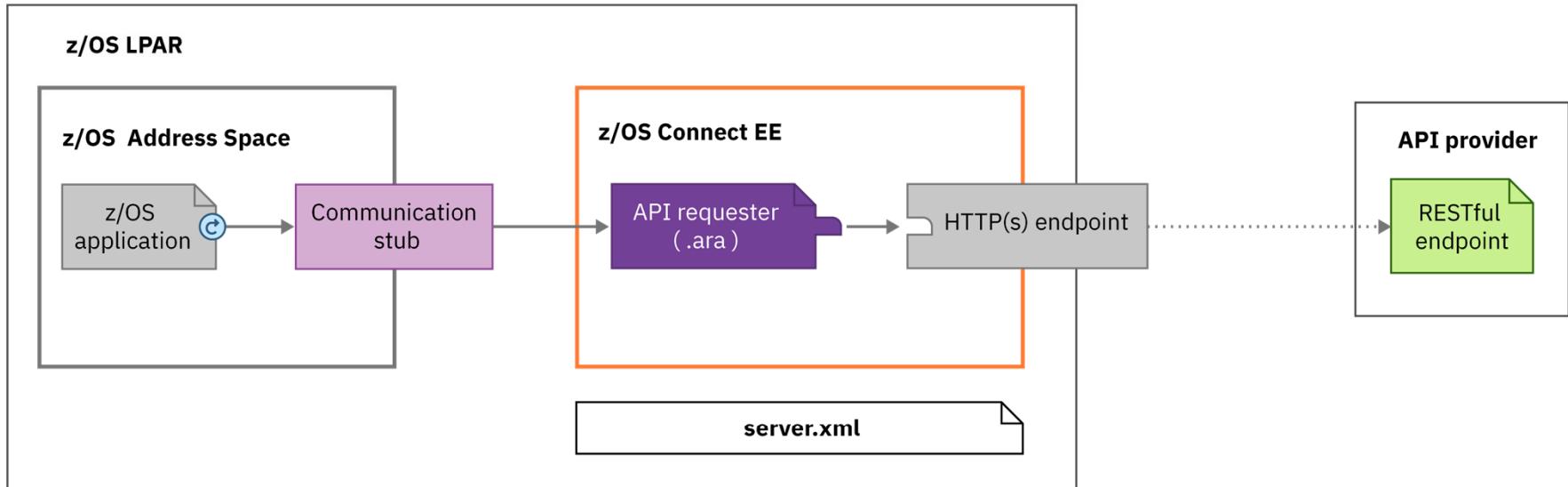


Configure the connection between z/OS Connect EE and the external API.

 [ibm.biz/zosconnect-configure-endpoint-connection](http://ibm.biz/zosconnect-configure-endpoint-connection)

# Steps to calling an external API

## Step 5. Update z/OS application



Finally, add the generated API client code to your existing application and use it to make the external API call.

 [ibm.biz/zosconnect-configure-requester-zos-application](https://ibm.biz/zosconnect-configure-requester-zos-application)

# Steps to calling an external API

Step 5a. Update the z/OS application to include new copy books

The screenshot shows the Rational Application Developer interface with three windows:

- apirs.xml**: An XML configuration file for the API Requester. It includes a server definition with a featureManager containing a zosconnect:apiRequester-1.0 feature, and a zosconnect\_endPointConnection section with a host set to "http://wg31.washington.ibm.com". This section is circled in red.
- GETAPI**: A window displaying an error message structure. It includes a note: "\* Copy API Requester required copybook COPY BAQRINFO." and a note: "\* Request and Response". It also shows a structure with the API in it, specifically COPY CSC02I01, which is circled in red.
- CSC02I01**: A window displaying a copybook structure. It lists fields such as BAQ-APINAME, BAQ-APINAME-LEN, BAQ-APIPATH, BAQ-APIPATH-LEN, BAQ-APIMETHOD, and BAQ-APIMETHOD-LEN. To the right of the copybook, there is a properties panel with several configuration options, some of which are circled in red: apiDescriptionFile, connectionRef, and requesterPrefix.

# Steps to calling an external API

Step 5b. Update the z/OS application to call the stub

The screenshot shows the RAD interface with four windows:

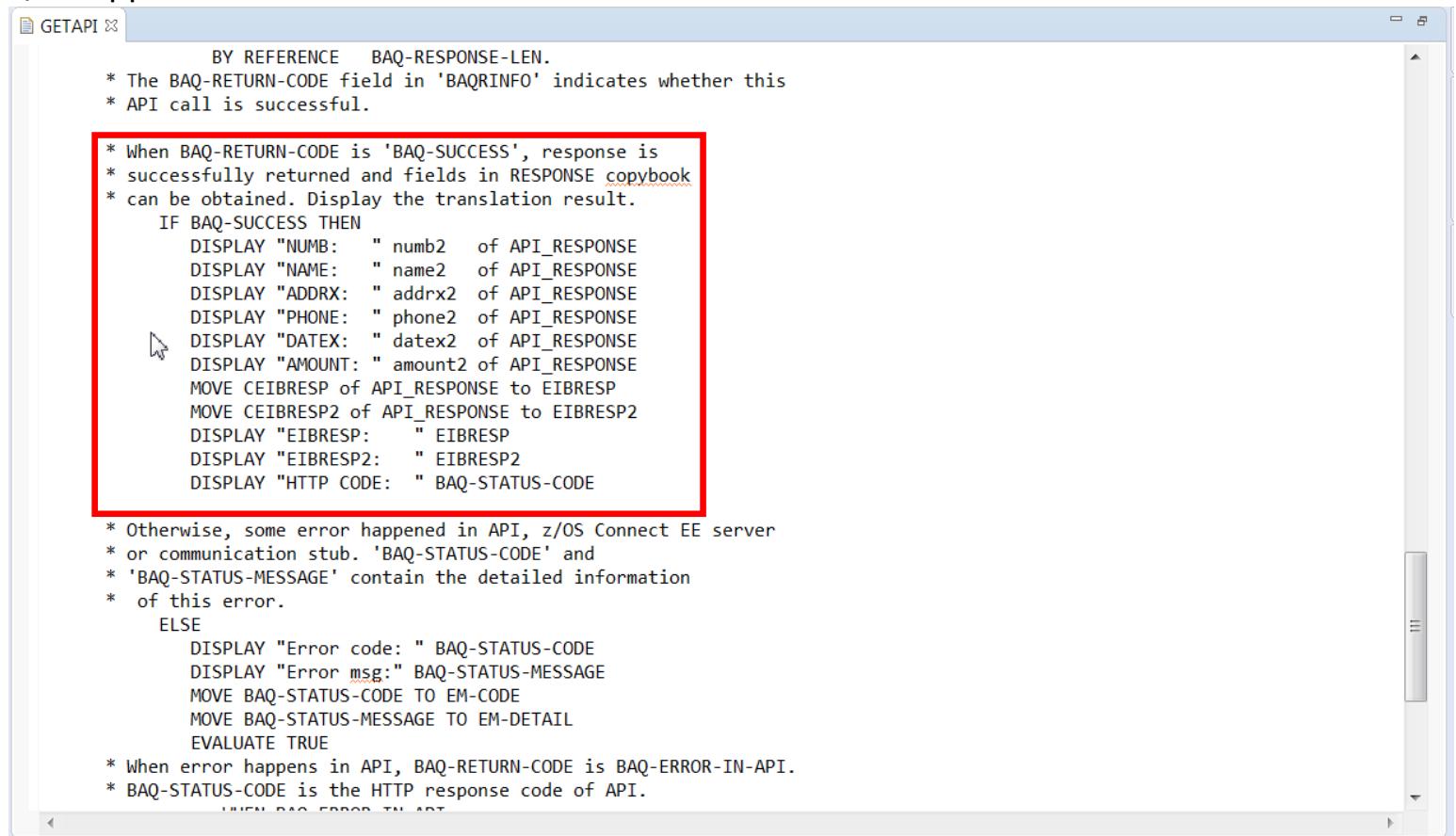
- GETAPI**: The main application window containing assembly code for calling an external API. A red box highlights the following line:

```
CALL COMM-STUB-PGM-NAME LISTNG
```

```
BY REFERENCE API-INFO-OPER1
BY REFERENCE BAQ-REQUEST-INFO
BY REFERENCE BAQ-REQUEST-PTR
BY REFERENCE BAQ-REQUEST-LEN
BY REFERENCE BAQ-RESPONSE-INFO
BY REFERENCE BAQ-RESPONSE-PTR
BY REFERENCE BAQ-RESPONSE-LEN.
```
- CSC00101**: A configuration file showing API parameters. It includes:
  - BAQ-APINAME: 'cscvincap1\_1.0.0'
  - BAQ-APINAME-LEN: 16
  - BAQ-APIPATH: '/v2/cscvincap1/employees/employee/%d'
  - BAQ-APIPATH-LEN: 41
  - BAQ-APIMETHOD: 'GET'
  - BAQ-APIMETHOD-LEN: 3
- CSC00Q01**: A configuration file showing JSON schema for the request body. It includes:
  - Employee length: 9999
  - Employee: 6
  - ReqPathParameters: 99 employee-length (9999), 99 employee (6)
- CSC00P01**: A configuration file showing the response structure. It includes:
  - RespBody: cscvincSelectServiceOp-num (9999)
  - Container1: 12
  - RESPONSE-CONTAINER2-num: 15

# Steps to calling an external API

## Step 5c. Update the z/OS application to access the results



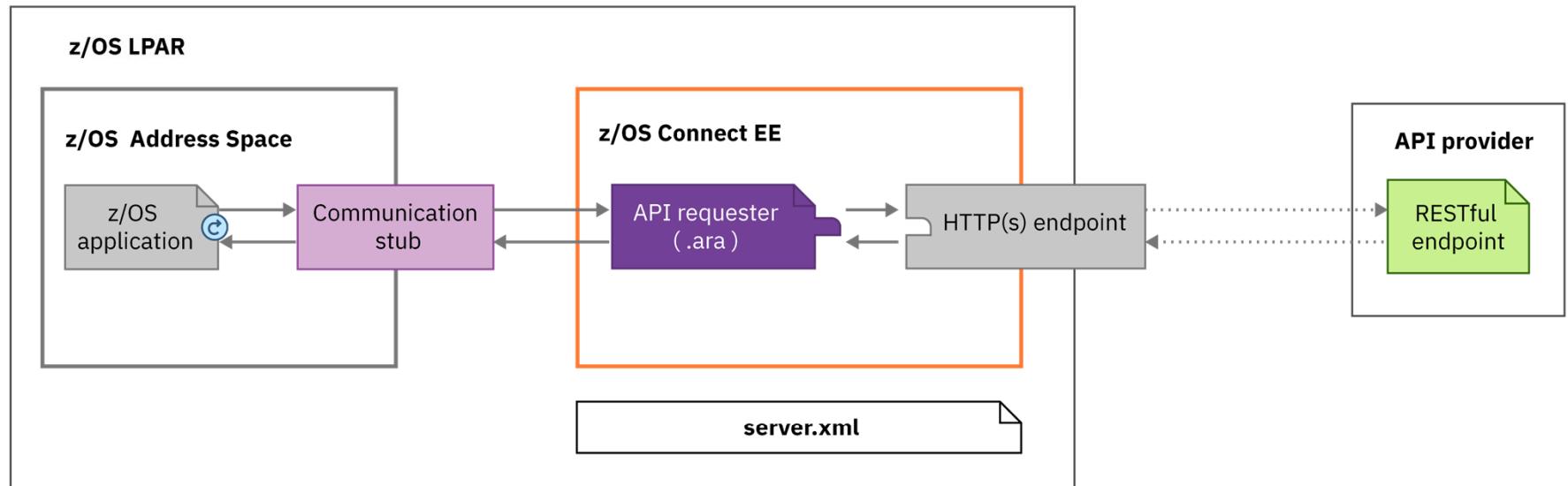
The screenshot shows the GETAPI editor window with the following AS/400 JCL code:

```
BY REFERENCE BAQ-RESPONSE-LEN.  
* The BAQ-RETURN-CODE field in 'BAQRINFO' indicates whether this  
* API call is successful.  
  
* When BAQ-RETURN-CODE is 'BAQ-SUCCESS', response is  
* successfully returned and fields in RESPONSE copybook  
* can be obtained. Display the translation result.  
IF BAQ-SUCCESS THEN  
  DISPLAY "NUMB: " numb2  of API_RESPONSE  
  DISPLAY "NAME: " name2  of API_RESPONSE  
  DISPLAY "ADDRX: " addrx2 of API_RESPONSE  
  DISPLAY "PHONE: " phone2 of API_RESPONSE  
  DISPLAY "DATEX: " datex2 of API_RESPONSE  
  DISPLAY "AMOUNT: " amount2 of API_RESPONSE  
  MOVE CEIBRESP of API_RESPONSE to EIBRESP  
  MOVE CEIBRESP2 of API_RESPONSE to EIBRESP2  
  DISPLAY "EIBRESP: " EIBRESP  
  DISPLAY "EIBRESP2: " EIBRESP2  
  DISPLAY "HTTP CODE: " BAQ-STATUS-CODE  
  
* Otherwise, some error happened in API, z/OS Connect EE server  
* or communication stub. 'BAQ-STATUS-CODE' and  
* 'BAQ-STATUS-MESSAGE' contain the detailed information  
* of this error.  
ELSE  
  DISPLAY "Error code: " BAQ-STATUS-CODE  
  DISPLAY "Error msg:" BAQ-STATUS-MESSAGE  
  MOVE BAQ-STATUS-CODE TO EM-CODE  
  MOVE BAQ-STATUS-MESSAGE TO EM-DETAIL  
  EVALUATE TRUE  
* When error happens in API, BAQ-RETURN-CODE is BAQ-ERROR-IN-API.  
* BAQ-STATUS-CODE is the HTTP response code of API.  
  WHEN BAQ-ERROR-IN-API
```

A red box highlights the section of code that handles the successful API call, specifically the IF BAQ-SUCCESS THEN block.

# Steps to calling an external API

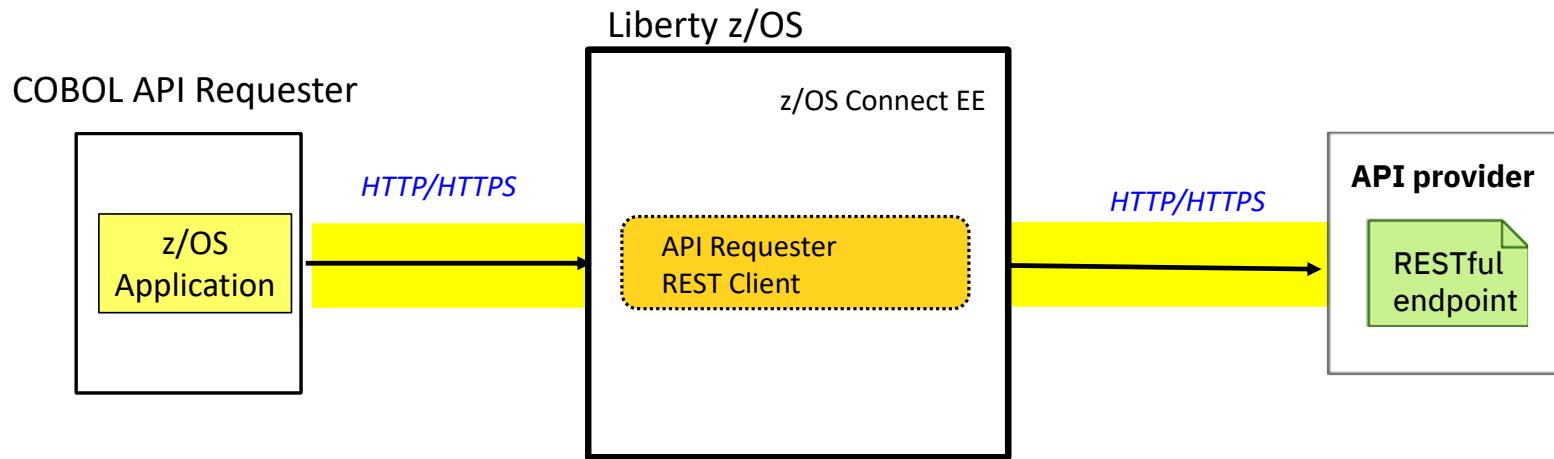
Done





z/OS Connect EE

## API requester to API Provider connection overview



MVS Batch and IMS HTTP connection details provided by:

- Environment Variables (BAQURI, BAQPORT)
  - Via JCL
  - LE Options (CEEROPTS)
  - Programmatically (CEEENV)
- HTTP or HTTPS

CICS HTTP connection details provided by:

- CICS URIMAP resource (default BAQURIMP)
  - HOST
  - PORT
  - SCHEME (HTTP/HTTPS)

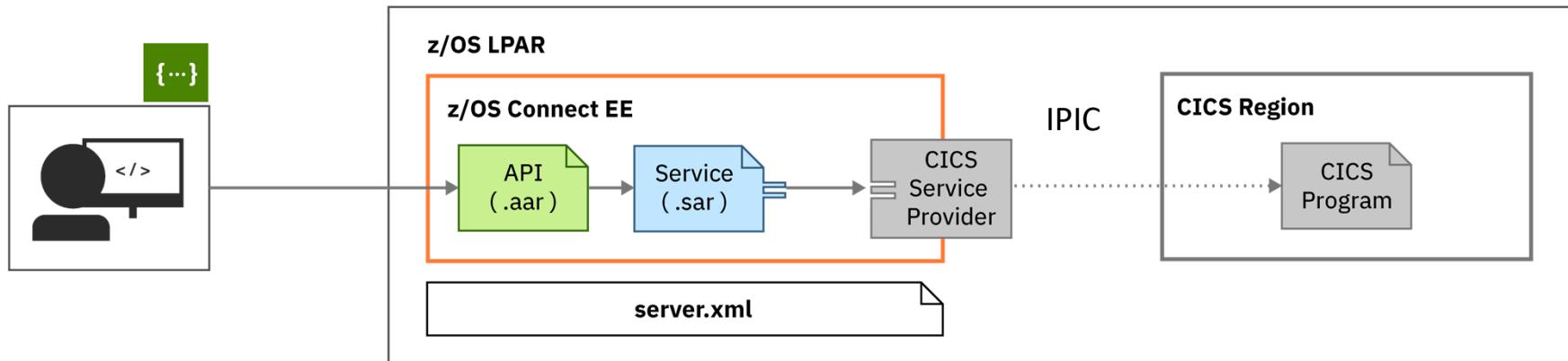


## **/common\_scenarios**

Typical connection patterns to different subsystems.

# Connections to CICS

## Topology



Connection to CICS is configured in `server.xml`.

An IPIC connection must be configured in CICS.

 [ibm.biz/zosconnect-scenarios](http://ibm.biz/zosconnect-scenarios)

# CICS IPIC (server.xml)



z/OS Connect EE

The server.xml file is the key configuration file:

The screenshot shows the 'inquireSingle Service' configuration dialog and a CICS transaction screen. The configuration dialog includes fields for 'Required Configuration' (Coded character set identifier (CCSID): 37, Connection reference: catalog) and 'Optional Configuration' (Transaction ID: [empty], Transaction ID usage: dropdown). The CICS transaction screen displays system parameters like CICS RELEASE = 0710 and various connection details.

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

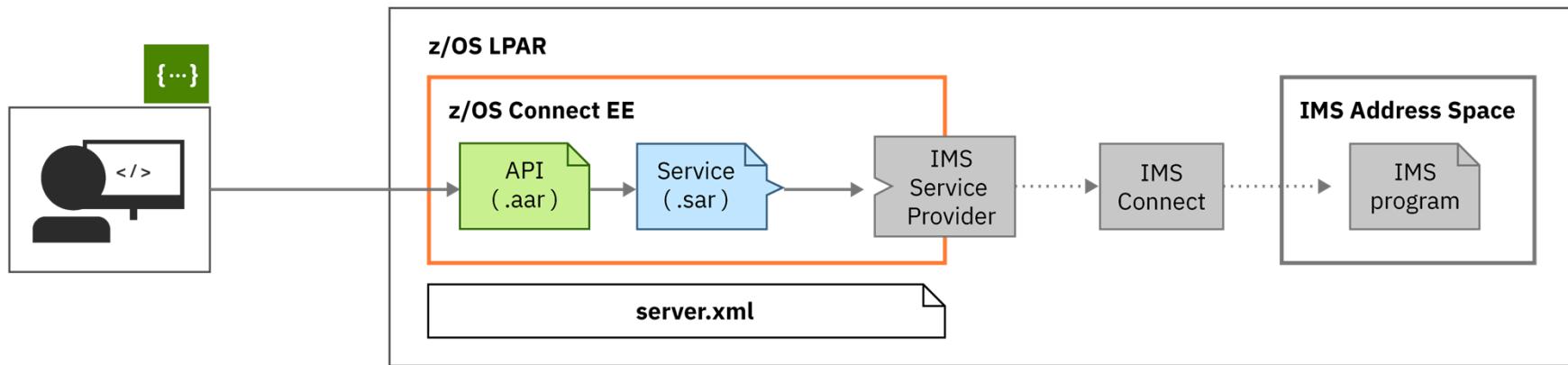
Features are functional building blocks. When configured here, that function becomes available to the Liberty server

Define IPIC connection to CICS

# Connections to IMS TM



## Topology



Configure the connection to IMS through `ims-connections.xml` and `ims-interactions.xml` in the IMS service registry.

[ibm.biz/zosconnect-scenarios](http://ibm.biz/zosconnect-scenarios)

# IMS Connections and Interactions



z/OS Connect EE

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

## IMS Connect HWSCFG

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

## Connection

```
<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>

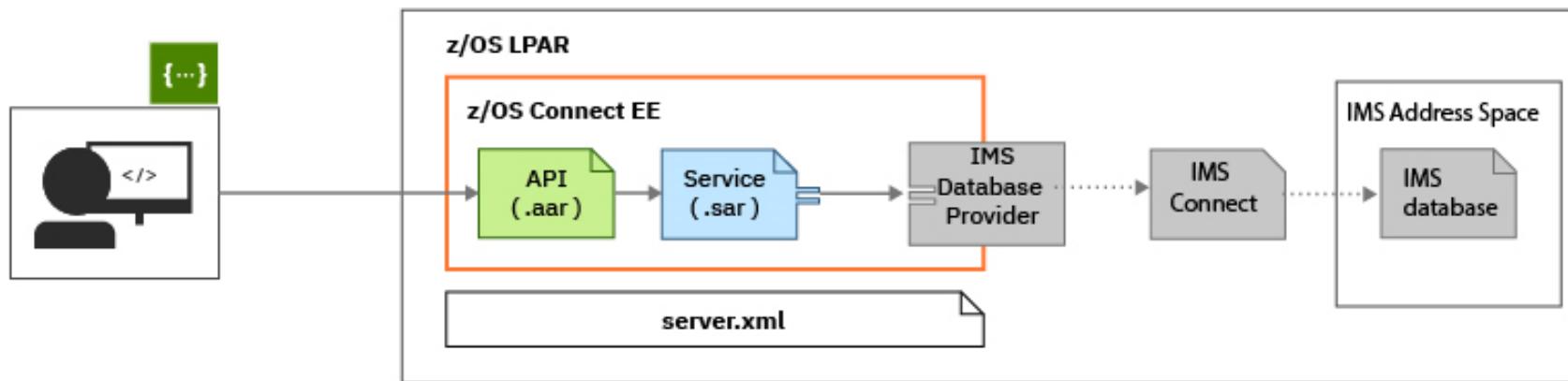
<authData id="Connection1_Auth" password="encryptedPassword1" user="userName1"/>
</server>
```

## Interaction

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

# Connections to IMS DB

## Topology



Configure the connection to IMS using a Connection Factory in server.xml

Use the **API toolkit** to configure the service.

# IMS Connection Factory



z/OS Connect EE

## Service Project Editor: Configuration



### Required Configuration

Enter the required configuration for this service.

Connection profile:

DFSIVPACConn

### ConnectionFactory

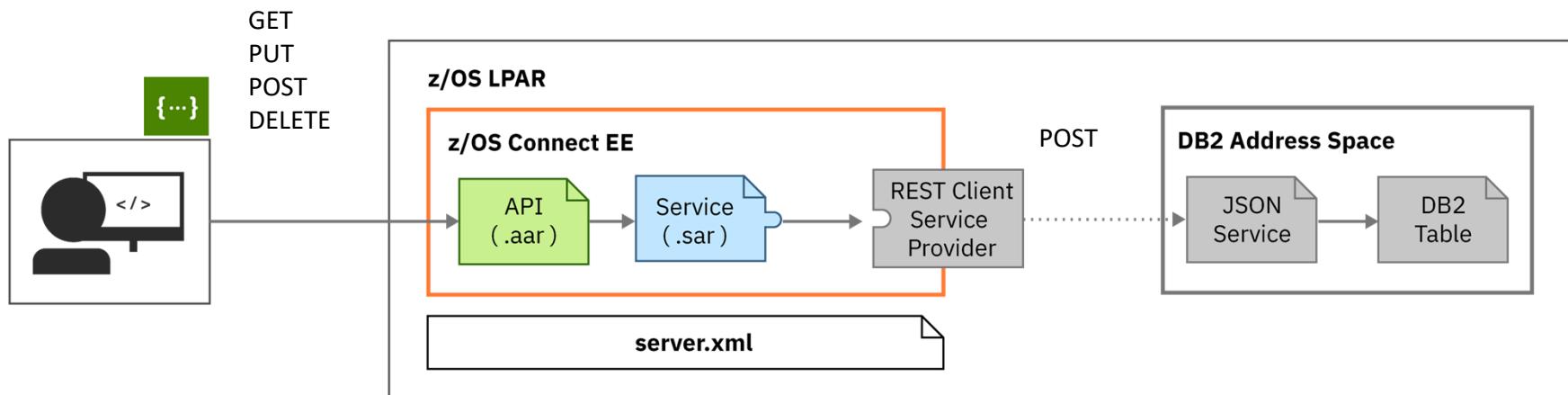
```
<connectionFactory id="DFSIVPACConn">
<properties.imsudbJLocal
  databaseName="DFSIVPA"
  datastoreName="IVP1"
  datastoreServer="wg31.washington.ibm.com"
  driverType="4"
  portNumber="5555"
  user="USER1"
  password="password"
  flattenTables="True"/>
</connectionFactory>
```

### IMS Connect HWSCFG

```
HWS=(ID=IMS14HWS,XIBAREA=100,RACE=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)
```

# Connections to Db2

## Topology



Connection to the JSON Service is configured in `server.xml`.

A Db2 REST Service must be configured in DB2.

# The server.xml File (Db2)



z/OS Connect EE

The screenshot shows the Service Project Editor interface with two main panes. The left pane, titled 'selectEmployee Service Configuration', displays service definitions and their properties. The right pane, titled 'db2pass.xml', shows the XML source code for the 'server' element.

**Service Project Editor: Configuration**

**Required Configuration**

Enter the required configuration for this service.

Connection reference: **db2conn**

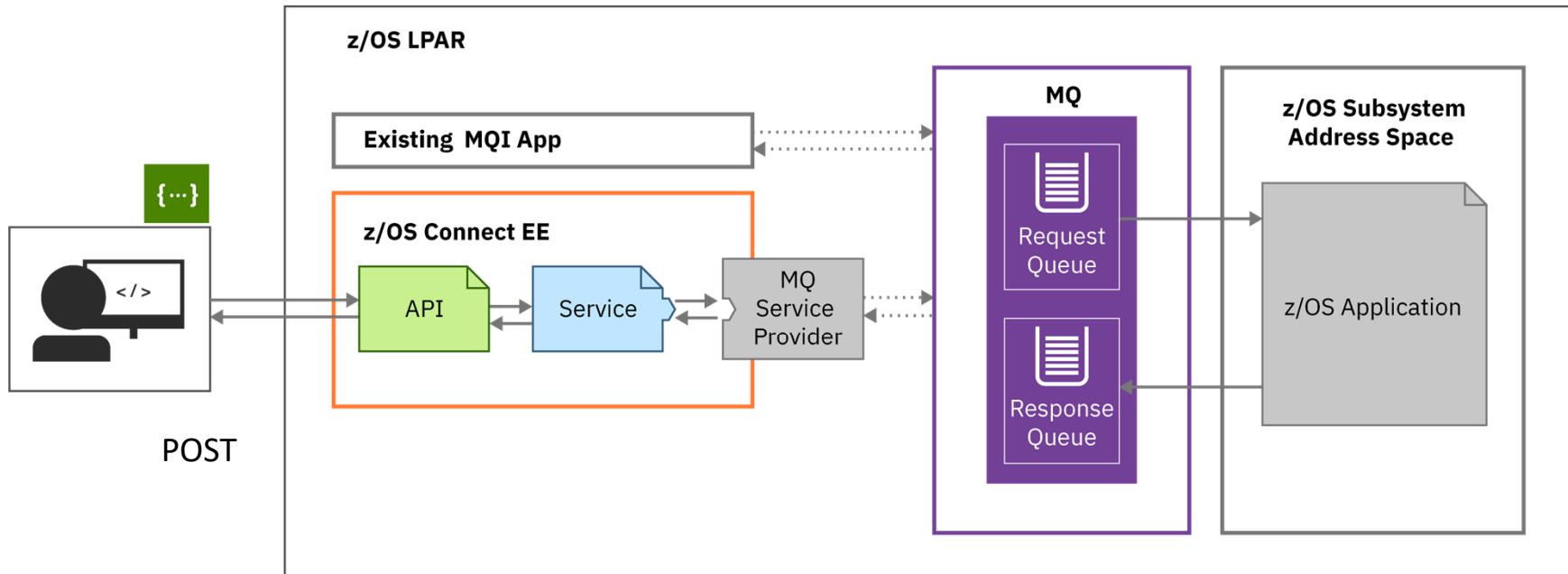
**db2pass.xml**

**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
```

# Connections to MQ

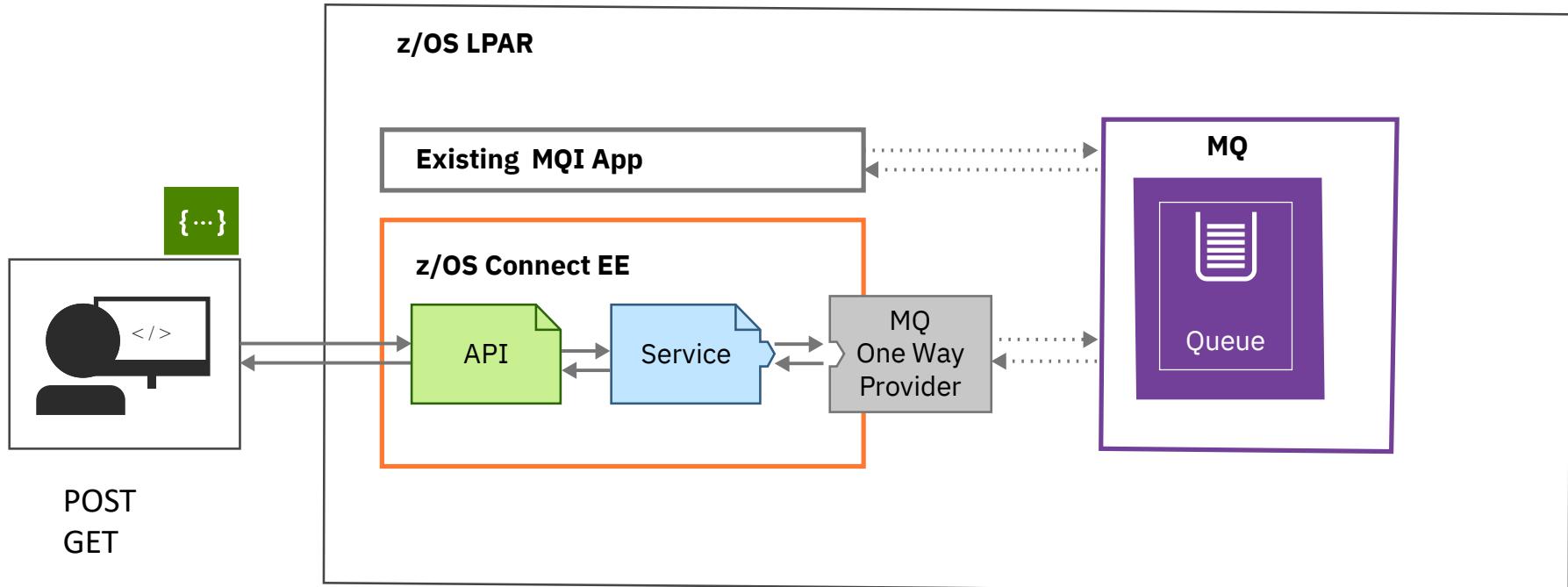
Topology (Two-way service example)



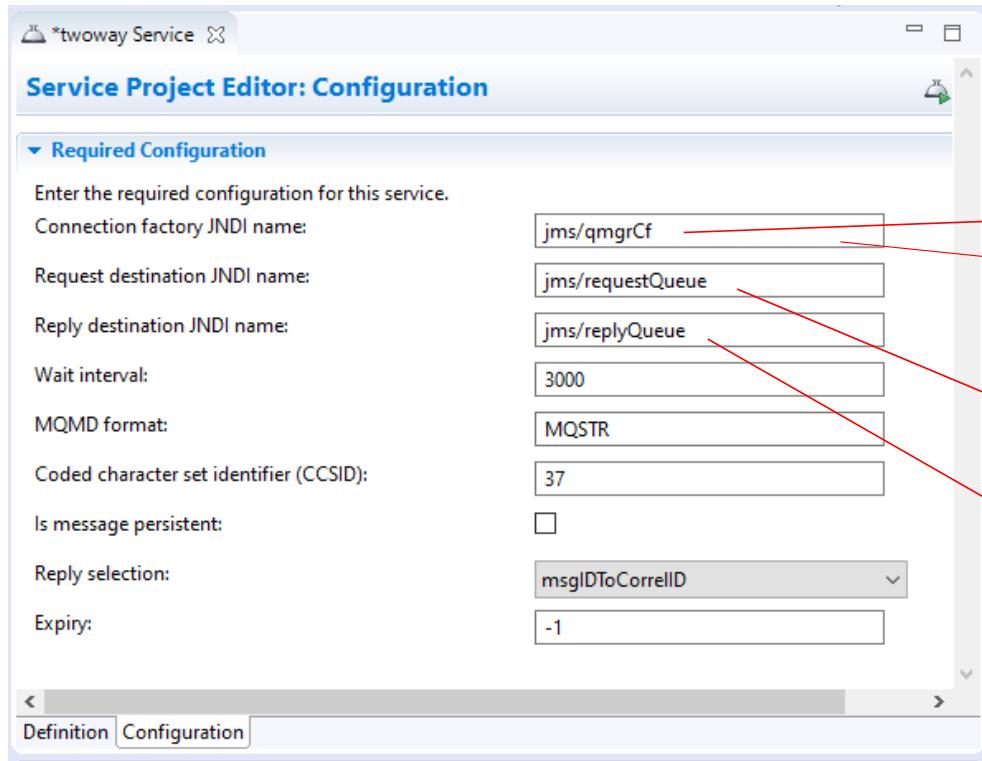
You can also configure one-way services.

# Connections to MQ

Topology (One-way service example)



# The server.xml File (MQ)



mq.xml

Read only Close

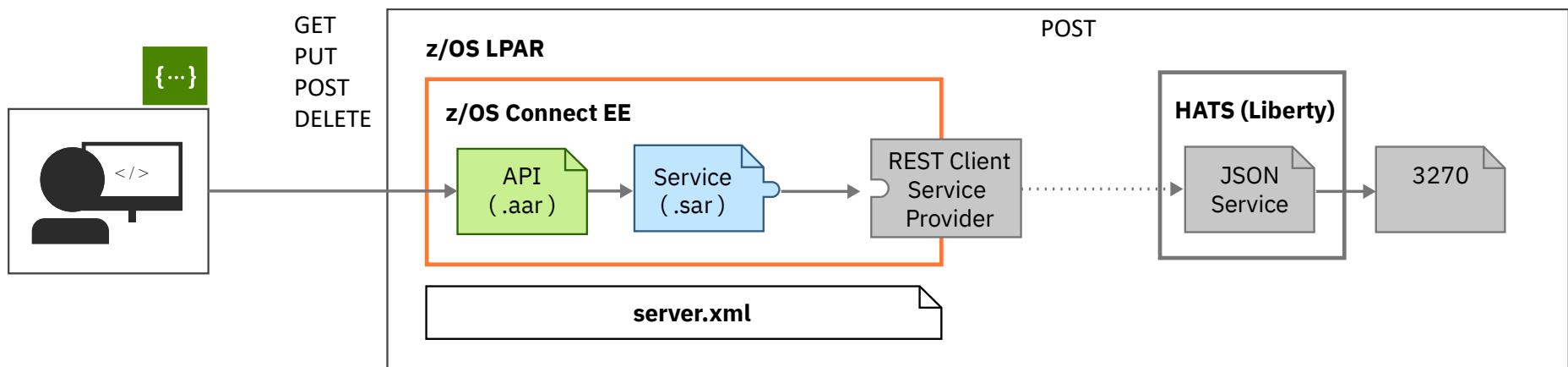
Design Source

```
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="BINDINGS"
15     queueManager="QM21" />
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
```

# Connection to HATS



## Topology



Connection to the HATS REST Service is configured in `server.xml`.

# HATS server XML configuration



```
getCompany.properties - Notepad
File Edit Format View Help
provider=rest
name=getCompany
version=1.0
description=Obtain a list of companies
requestSchemaFile=getCompanyRequest.json
responseSchemaFile=getCompanyResponse.json
verb=POST
uri=/Trader/rest/GetCompany
connectionRef=HatsConn
```

Server Config

hats.xml

Read only Close

Design Source

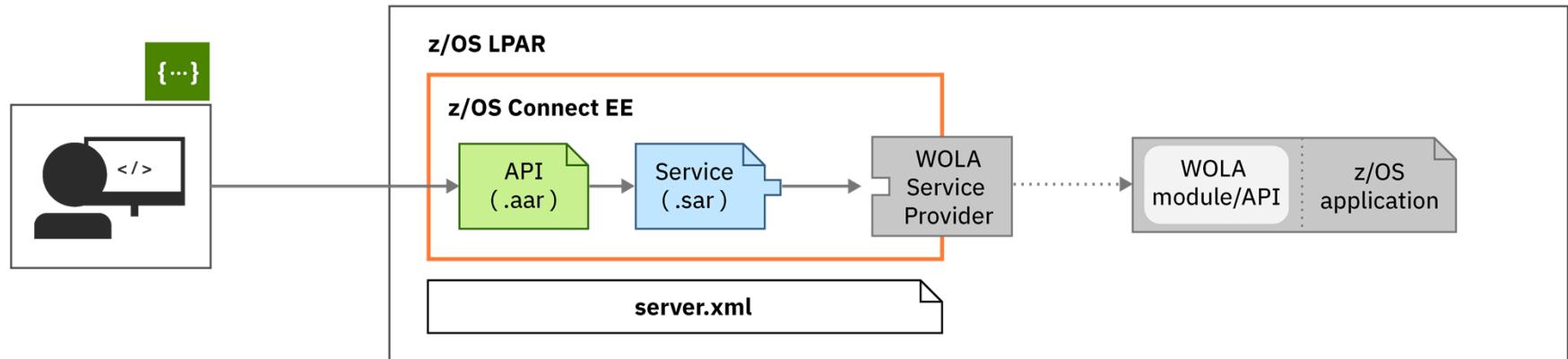
```
<server description="HATS">
  <zosconnect_zosConnectServiceRestClientConnection id="HatsConn">
    host="wg31.washington.ibm.com"
    port="29080" />
</server>
```

## HATS Liberty server.xml

```
<!-- To access this server from a remote client, add a host attribute to the following element, e.g. host="*" -->
<httpEndpoint id="defaultHttpEndpoint"
  httpPort="29080" host="*"
  httpsPort="29443" />
```

# Connections to a MVS batch application

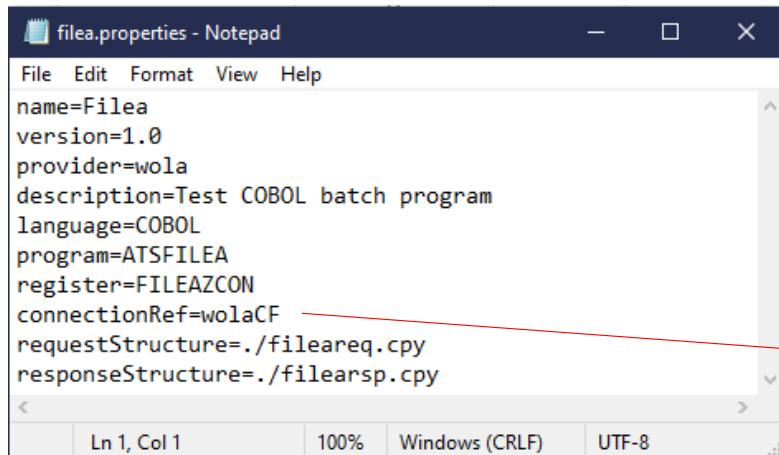
## Topology



Connection to WOLA is configured in `server.xml`.

The z/OS application must be WOLA-enabled.

# MVS batch server XML



```
filea.properties - Notepad
File Edit Format View Help
name=Filea
version=1.0
provider=wola
description=Test COBOL batch program
language=COBOL
program=ATSFIL
register=FILEAZCON
connectionRef=wolaCF
requestStructure=./fileareq.cpy
responseStructure=./filearsp.cpy
```

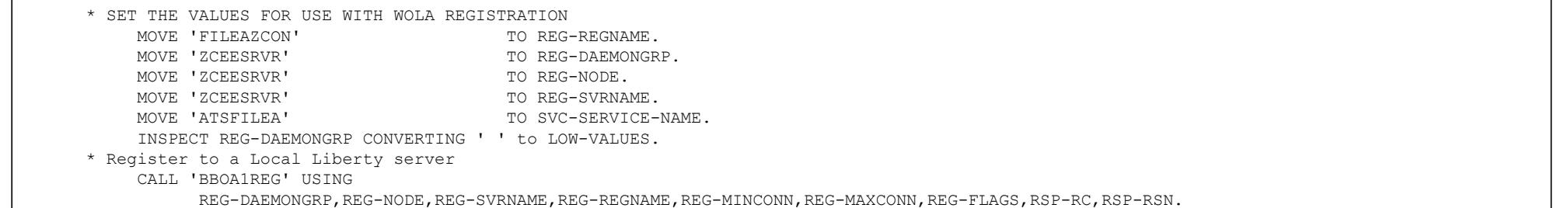


Server Config

wola.xml

Design      Source

```
<server description="WOLA">
  <featureManager>
    <feature>zosLocalAdapters-1.0</feature>
  </featureManager>
  <zosLocalAdapters wolaGroup="ZCEESRVR">
    wolaName2="ZCEESRVR"
    wolaName3="ZCEESRVR"/>
  <connectionFactory id="wolaCF">
    jndiName="eis/ola">
      <properties.ola/>
    </connectionFactory>
  </server>
```



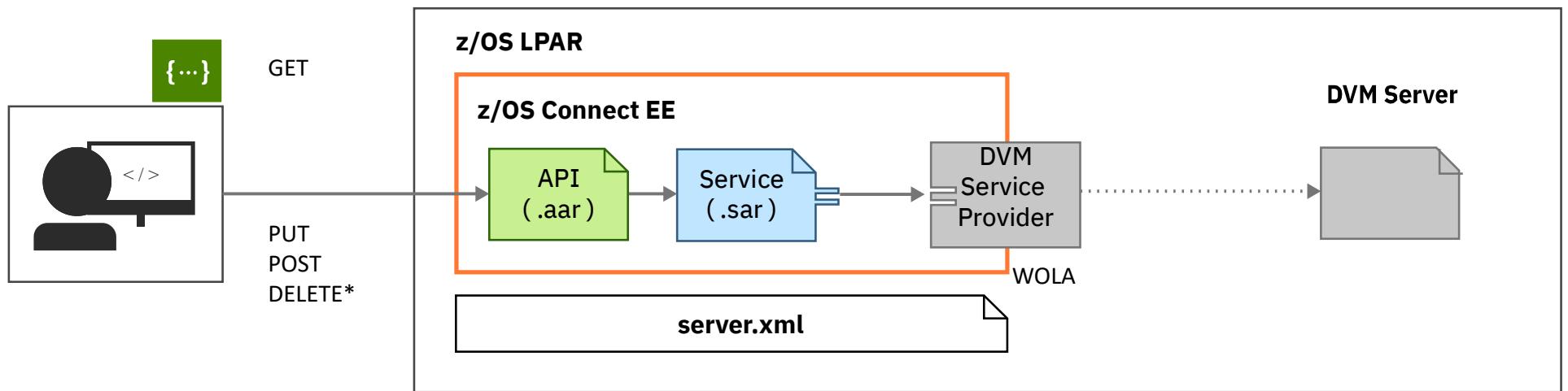
```
* SET THE VALUES FOR USE WITH WOLA REGISTRATION
MOVE 'FILEAZCON'          TO REG-REGNAME.
MOVE 'ZCEESRVR'            TO REG-DAEMONGRP.
MOVE 'ZCEESRVR'            TO REG-NODE.
MOVE 'ZCEESRVR'            TO REG-SVRNAME.
MOVE 'ATSFIL'              TO SVC-SERVICE-NAME.
INSPECT REG-DAEMONGRP CONVERTING ' ' to LOW-VALUES.
* Register to a Local Liberty server
CALL 'BBOA1REG' USING
  REG-DAEMONGRP,REG-NODE,REG-SVRNAME,REG-REGNAME,REG-MINCONN,REG-MAXCONN,REG-FLAGS,RSP-RC,RSP-RSN.
```

# Connections to DVM



z/OS Connect EE

## Topology



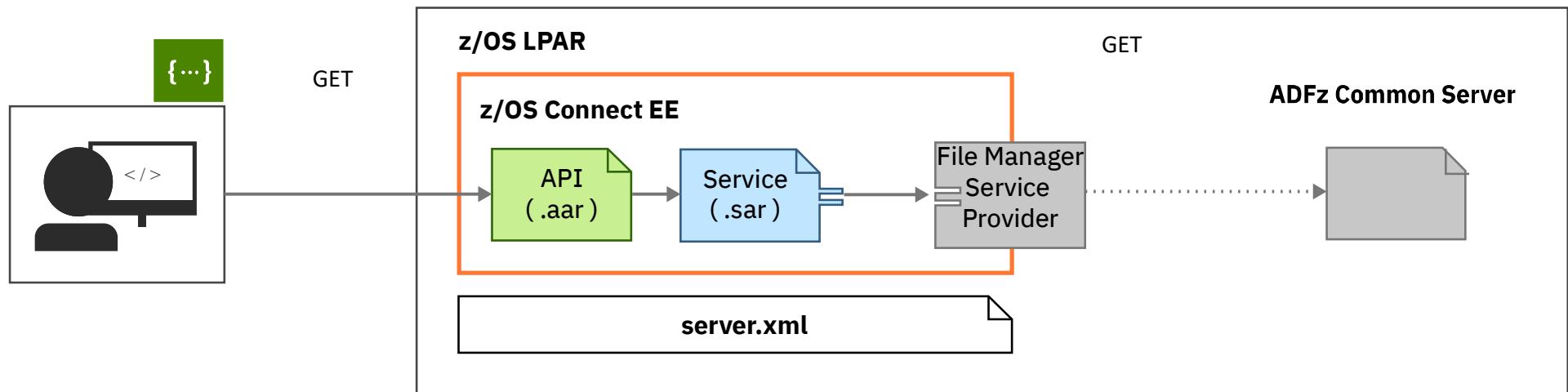
The DVM service provider uses WOLA

\* Requires a resource manager (e.g. RLS, VSAMCICS, etc.)

# Connections to File Manager



## Topology



Connection to the Application Delivery Foundation for z (ADFz) common server is over TCP/IP

A File Manager Template is required .

# File Manager server XML



```
filea.properties - Notepad
File Edit Format View Help
name=filea
provider=filemanager
host=wg31.washington.ibm.com
version=1.0
port=2800
file=USER1.ZCEE.FILEA
template=USER1.ZCEE.TEMPLATE(FILEA)
connid=default
userid=USER1
passwd=USER1

<
Ln 1, Col 1 100% Windows (CRLF) UTF-8 ..>
```

Server Config

filemgr.xml

Read only Close

Design Source

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">
  <!-- Enable features -->
  <featureManager>
    <feature>filemanager:fmProvider-2.0</feature>
  </featureManager>
  <FileManager_Connection id="default"
    runport="2800"
    max_timeout="1800"/>
</server>
```

SYS1.PROCLIB(IPVSRV1)

```
//IPVSRV1 PROC PORT=2800,FAMILY='AF_INET',TRACE=N
//      SET ENV=''
//RUN      EXEC PGM=IPVSRV,REGION=40M,
//          PARM=('&ENV/&PORT &FAMILY &TRACE')
// SET IPV=SYSP.ADFZ.JCL           <== Update HLQ
//** Common component authorised library
//STEPLIB  DD DISP=SHR,DSN=ADFZ.SIPVMODA      <== ADFzCC APF LIBRARY
//SYSPRINT DD SYSOUT=*
//IPVTRACE DD SYSOUT=*                  <== OUTPUT if Tracing
//STDOUT   DD SYSOUT=*
//** Server wide, then participating product configurations
//CONFIG   DD DISP=SHR,DSN=&IPV.(IPVCFG)
```

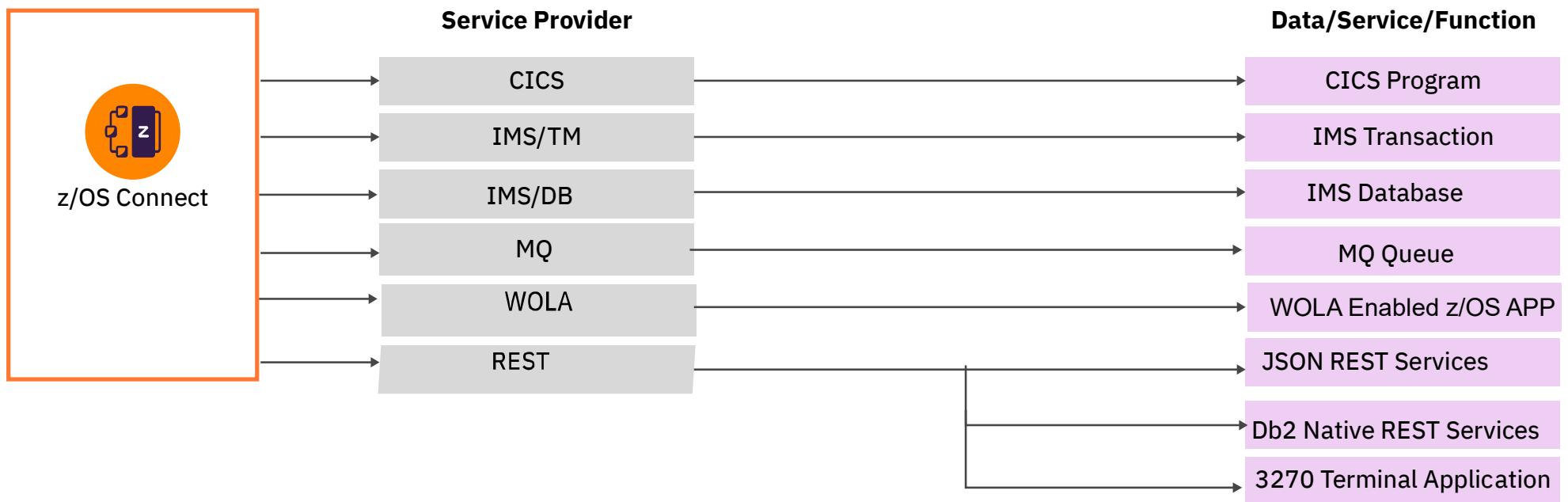


## /miscellaneousTopics

performance, high availability, Liberty

# What assets can z/OS Connect EE map to?

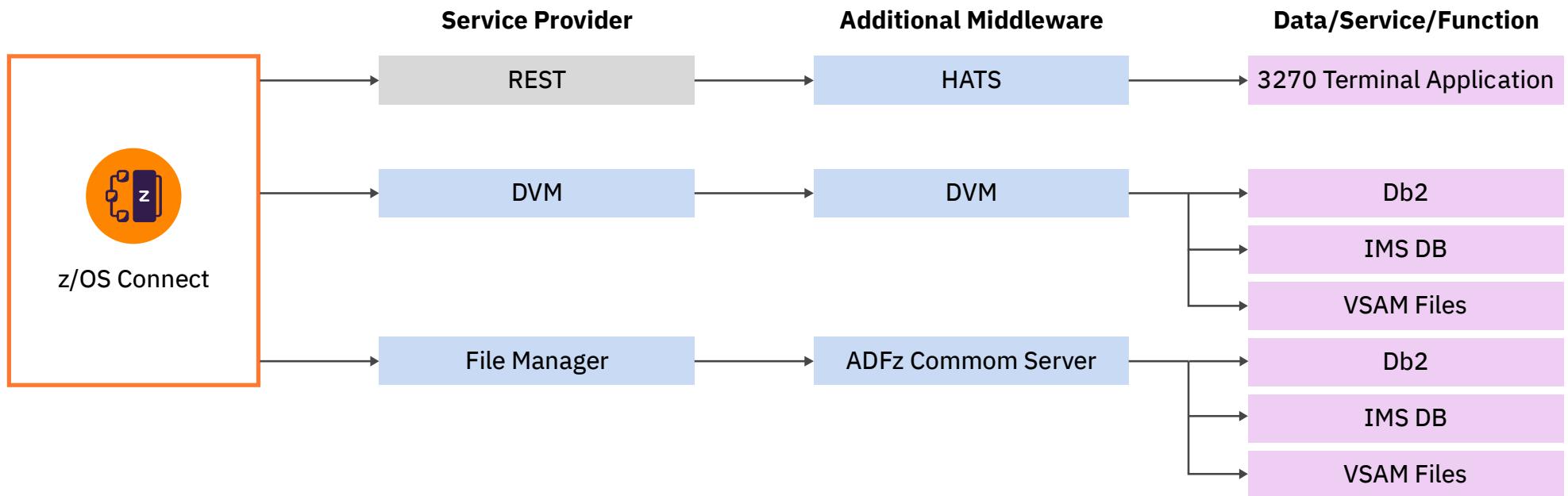
And which service provider could I use?



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

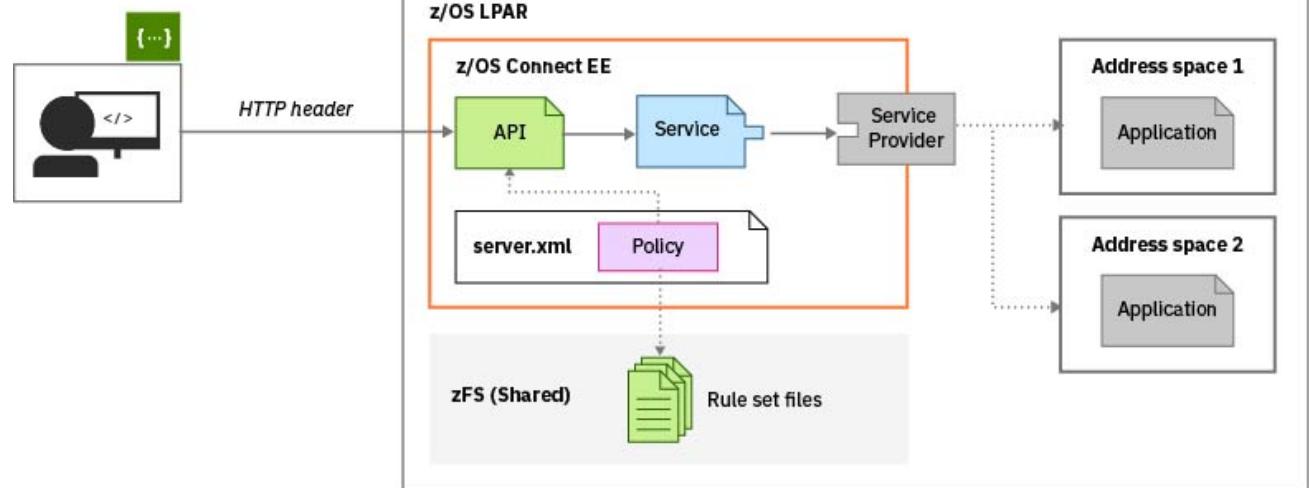
- imsConnectionRef
- imsInteractionRef
- imsInteractionTimeout
- imsLtermOverrideName
- imsTranCode
- imsTranExpiration

Db2 attributes

- db2ConnectionRef
- db2CollectionID

MQ attributes

- mqConnectionFactory
- mqDestination
- mqReplyDestination



# A sample API Policies for CICS



z/OS Connect EE

```
<ruleset name="CICS rules">
  <rule name="csmi-rule">
    <conditions>
      <header name="cicsMirror" value="CSMI,MIJO"/> *
    </conditions>
    <actions>
      <set property="cicsTransId" value="${cicsMirror}"/>
    </actions>
  </rule>
  <rule name="connection-rule">
    <conditions>
      <header name="cicsConnection"
             value="cscvinc,cics92,cics93"/>
    </conditions>
    <actions>
      <set property="cicsConnectionRef" value="${cicsConnection}">
    </actions>
  </rule>
</ruleset>
```

The screenshot shows the z/OS Connect API designer interface. A red oval highlights the 'Path Parameters' section, specifically the 'numb' entry which is labeled as 'Required string'. Other sections shown include 'HTTP Headers' (with 'cicsMirror' and 'cicsConnection' listed as 'optional string'), 'Query Parameters', and 'Body - cscvincServiceOperation'.

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)



z/OS Connect EE

# Displaying zCEE messages on the console and/or spool

## server.xml

```
<zosLogging wtoMessage=
  "BAQR0657E,BAQR0658E,BAQR0660E,BAQR0686E,BAQR0687E"
  hardCopyMessage=
  "BAQR0657E,BAQR0658E,BAQR0660E,BAQR0686E,BAQR0687E"/>
```

## MVS Console

```
18.12.02 STC00137 +BAQR0686E: Program CSCVINC is not available in the CICS region with
  811           connection ID cscvinc; service cscvincService failed.
18.12.02 STC00137 +BAQR0686E: Program CSCVINC is not available in the CICS region with
  812           connection ID cscvinc; service cscvincService failed.
19.07.12 STC00137 +BAQR0657E: Transaction abend MIJO occurred in CICS while using
  745           connection cscvinc and service cscvincService.
```

## STDERR

```
ÝERROR   " BAQR0686E: Program CSCVINC is not available in the CICS region with connection cscvinc and service cscvincService.
ÝERROR   " BAQR0686E: Program CSCVINC is not available in the CICS region with connection cscvinc and service cscvincService.
ÝERROR   " BAQR0657E: Transaction abend MIJO occurred in CICS while using CICS connection cscvinc and service cscvincService.
```



# Tech/Tip: Providing access to configuration/log information

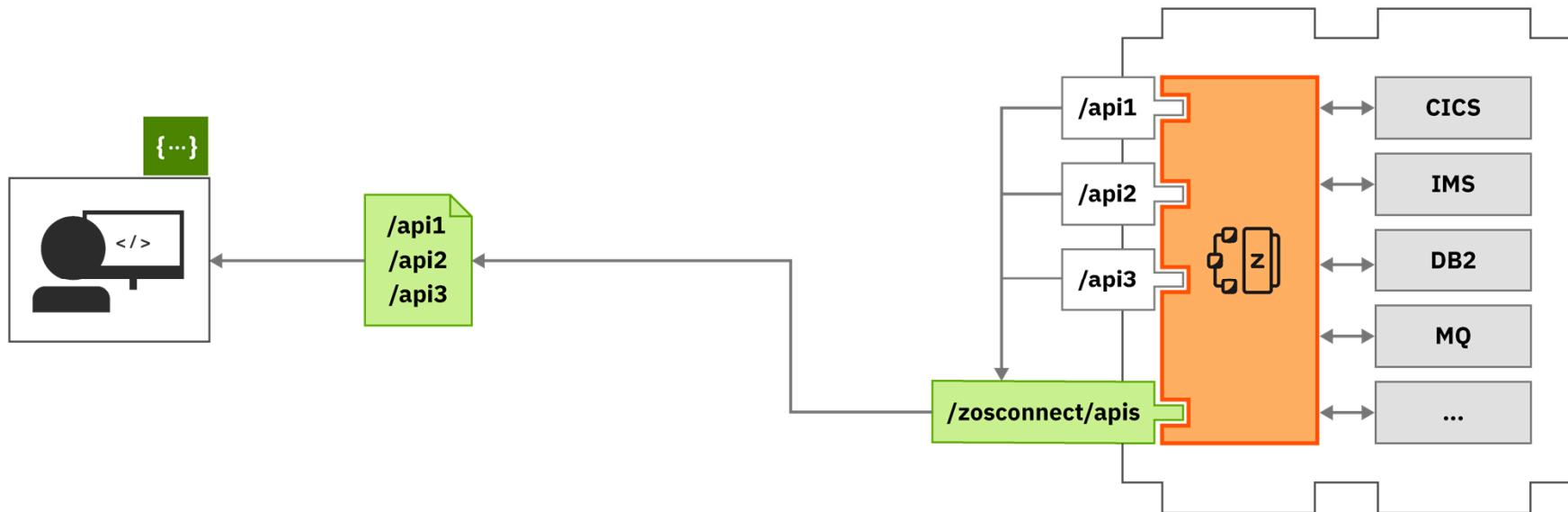
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--<server description="new server">
<include location="${server.config.dir}/includes/safSecurity.xml"/>
<include location="${server.config.dir}/includes/ipicSSLIDProp.xml"/>
<include location="${server.config.dir}/includes/keyringOutbound.xml"/>
<include location="${server.config.dir}/includes/groupAccess.xml"/>
<include location="${server.config.dir}/includes/shared.xml"/>
<include location="${server.config.dir}/includes/oauth.xml"/>
<include location="${server.config.dir}/includes/adminCenter.xml"/>
<include location="${server.config.dir}/includes/mqClientTLS.xml"/>
<include location="${server.config.dir}/includes/web.xml"/>
<!-- Enable features -->
<featureManager>
<feature>zosconnect:zosConnect-2.0</feature>
<feature>zosconnect:zosConnectCommands-1.0</feature>
</featureManager>
<!--<br>
To access this server from a remote client add a host at
-->
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" ht
<!--<br>
add cors to allow cross origin access, e.g. when using s
-->
```

```
<webApplication id="serverConfig-location" name="serverConfig"
location="${server.config.dir}">
<web-ext context-root="/server/config"
enable-file-serving="true" enable-directory-browsing="true">
<file-serving-attribute name="extendedDocumentRoot"
value="${server.config.dir}" />
</web-ext>
</webApplication>
```

\*\*\*\*\*  
product = WAS FOR z/OS 20.0.0.6, z/OS Connect 03.00.41 (wlp-1.0.41.cl200620200528-0414)  
wlp.install.dir = /shared/IBM/zosconnect/v3r0/wlp/  
server.config.dir = /var/zosconnect/servers/myServer/  
java.home = /shared/java/J8\_0\_64  
java.version = 1.8.0\_261  
java.runtime = Java(TM) SE Runtime Environment (8.0.6.15 - pmz6480sr6fp15-20200724\_01(SR6 FP15))  
os = z/OS (02.03.00; s390x) (en\_US)  
process = 16778879@wg31  
\*\*\*\*\*  
[2/19/21 15:48:18:901 GMT] 0000000b com.ibm.ws.kernel.launch.internal.FrameworkManager  
[2/19/21 15:48:19:869 GMT] 00000017 com.ibm.ws.config.xml.internal.XMLConfigParser  
/var/zosconnect/servers/myServer/includes/safSecurity.xml  
/var/zosconnect/servers/myServer/includes/ipicIDProp.xml  
/var/zosconnect/servers/myServer/includes/oauth.xml  
/var/zosconnect/servers/myServer/includes/test.xml  
/var/zosconnect/servers/myServer/includes/keyringOutbound.xml  
/var/zosconnect/servers/myServer/includes/groupAccess.xml  
[2/19/21 15:48:19:906 GMT] 00000017 com.ibm.ws.config.xml.internal.XMLConfigParser  
/var/zosconnect/servers/myServer/includes/shared.xml  
[2/19/21 15:48:19:907 GMT] 00000017 com.ibm.ws.config.xml.internal.XMLConfigParser  
/var/zosconnect/servers/myServer/includes/oauth.xml  
[2/19/21 15:48:19:911 GMT] 00000017 com.ibm.ws.config.xml.internal.XMLConfigParser  
/var/zosconnect/servers/myServer/includes/test.xml  
[2/19/21 15:48:19:994 GMT] 00000016 com.ibm.ws.zos.core.internal.NativeServiceTracker  
below-the-line storage limit is 8MB and the above-the-line storage limit is 1729MB.  
[2/19/21 15:48:19:997 GMT] 00000016 com.ibm.ws.zos.core.internal.NativeServiceTracker  
[2/19/21 15:48:20:012 GMT] 00000016 com.ibm.ws.zos.core.internal.NativeServiceTracker  
process.  
[2/19/21 15:48:20:089 GMT] 00000016 com.ibm.ws.zos.core.internal.NativeServiceTracker  
[2/19/21 15:48:20:091 GMT] 00000016 com.ibm.ws.zos.core.internal.NativeServiceTracker  
\*\*\*\*\*  
A CNWKE0001I: The server myServer has been launched.  
A CNWKG0028A: Processing included configuration resource:  
I CNWKB0125I: This server requested a REGION size of 0KB. The  
I CNWKB0126I: MEMLIMIT=1000. MEMLIMIT CONFIGURATION SOURCE=JCL.  
I CNWKB0122I: This server is connected to the default angel  
I CNWKB0103I: Authorized service group KERNEL is available.  
I CNWKB0103I: Authorized service group LOCALCOM is available

# API Documentation



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



## 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.

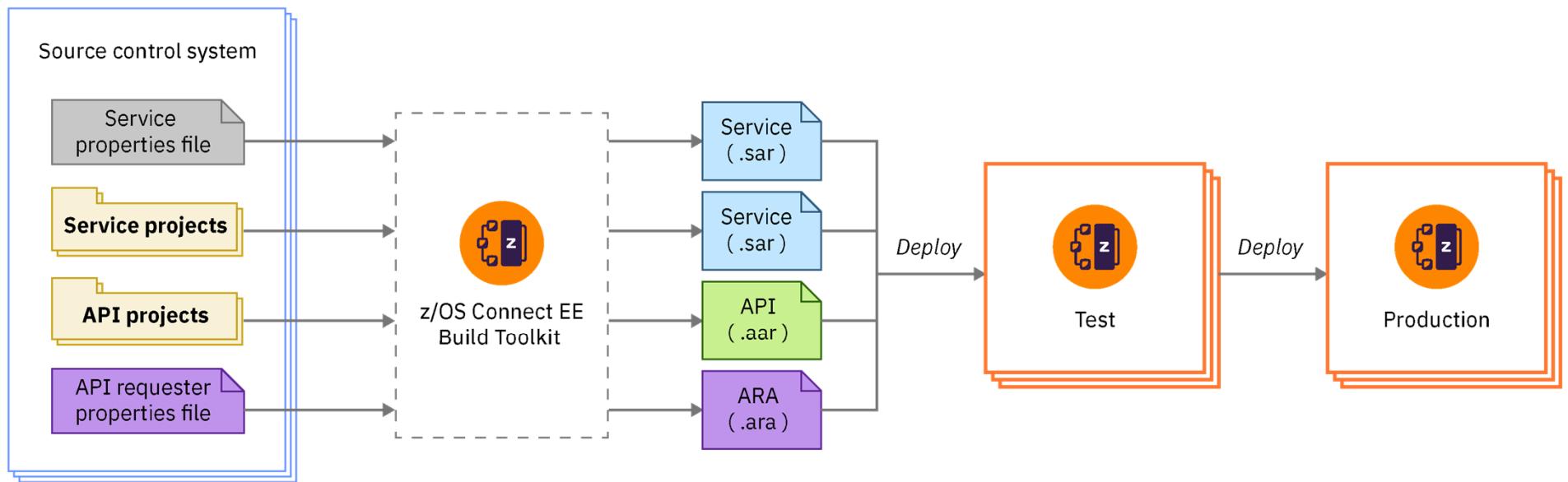
Show/Hide | List Operations | Expand Operations

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

# DevOps using z/OS Connect EE

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

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





# Deploying Service Archive – z/OS options

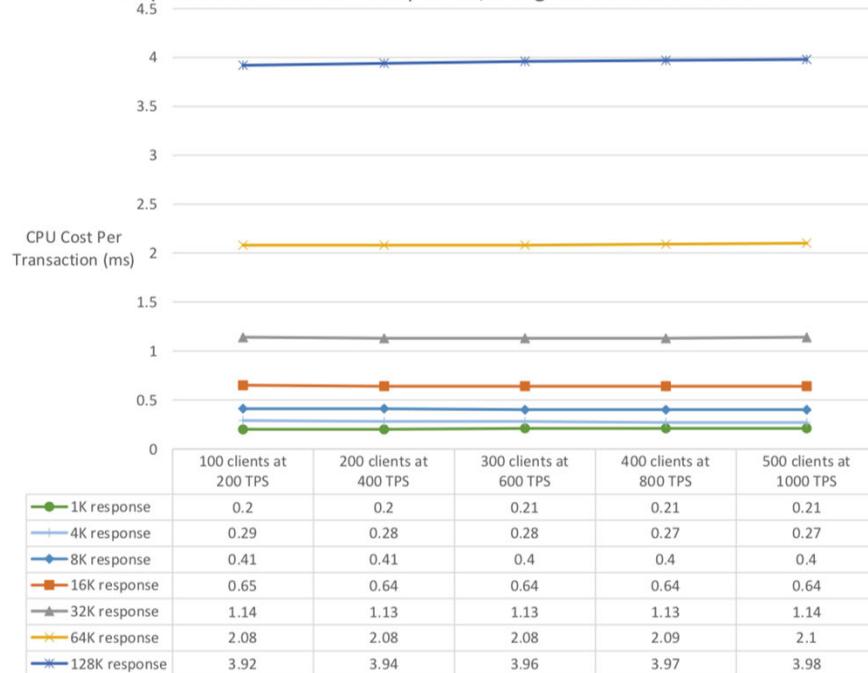
```
//*****
///* SET SYMBOLS
//*****
//EXPORT EXPORT SYMLIST=(*)
// SET CURL='/usr/lpp/rocket/curl'
//*****
///* CURL Procedure
//*****
//CURL PROC
//CURL EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR DD SYSOUT=*
//STDOUT DD SYSOUT=*
// PEND
//*****
///* STEP CURL - use cURL to stop API cscvinc
//*****
//LIST EXEC CURL
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X PUT +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9443/zosConnect/apis/cscvinc?status=stoped
//*****
///* STEP CURL - use cURL to delete the API cscvinc
//*****
//DELETE EXEC CURL
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X DELETE +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9443/zosConnect/apis/cscvinc
//*****
///* STEP CURL - use curl to deploy the API cscvinc
//*****
//DEPLOY EXEC CURL
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X POST +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
--data-binary @/u/johnson/cscvinc.aar +
--header "Content-Type: application/zip" +
https://wg31.washington.ibm.com:9443/zosConnect/apis
```

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

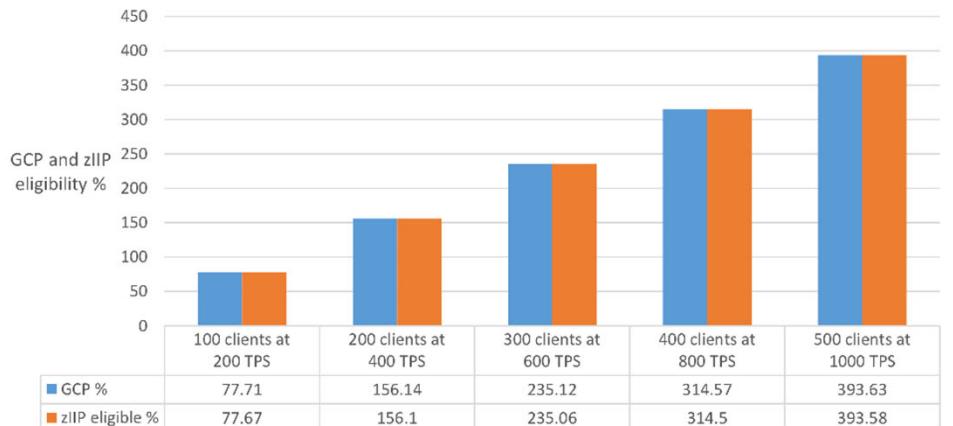
# Performance: API Provider

## High Speed, High Throughput, Low Cost

CPU Cost Per Transaction - increasing number of clients with 50 byte requests and 1K to 128K responses, using channels and CICS SP



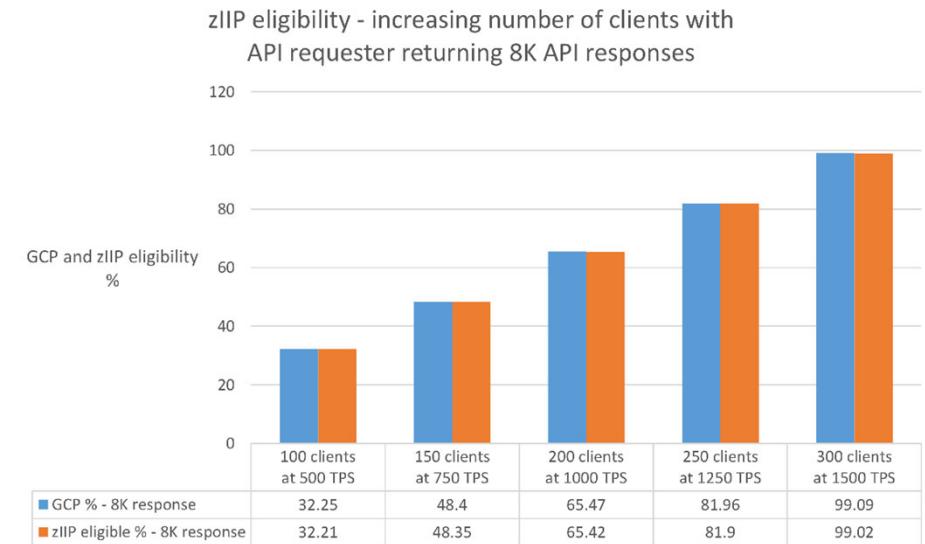
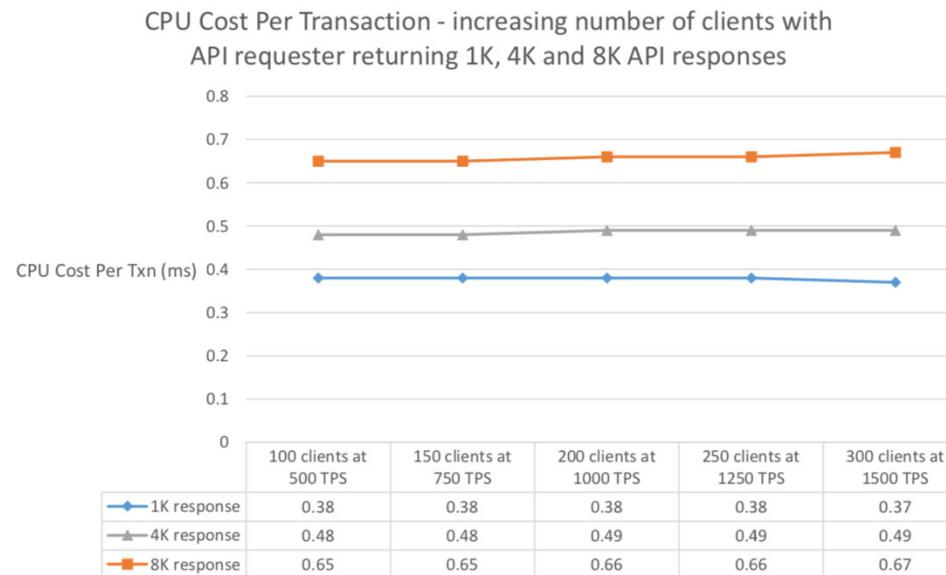
zIIP eligibility - increasing number of clients with 50 byte requests and 128K responses, using channels and CICS SP



z/OS Connect EE is a Java-based product:  
Over **99%** of its MIPs are **eligible for ZIIP offload**.

# Performance: API Requester

High Speed, High Throughput, Low Cost



z/OS Connect EE is a Java-based product:  
Over **99%** of its MIPs are **eligible for ZIIP offload**.



# /security

How is security implement?

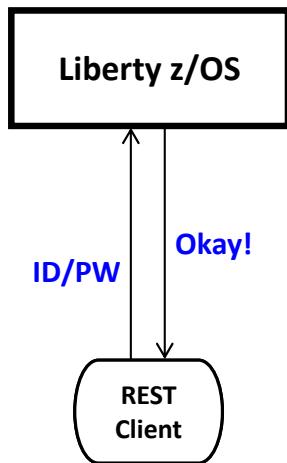
# API Provider Authentication



z/OS Connect EE

Several different ways this can be accomplished:

## Basic Authentication



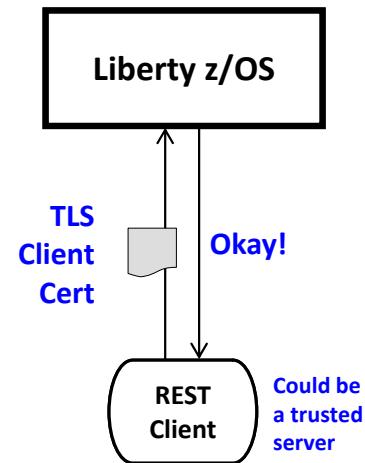
Server prompts for ID/PW

Client supplies ID/PW or  
ID/Passticket

Server checks registry:

- Basic (server.xml)
- LDAP
- SAF

## Client Certificate



Server prompts for cert.

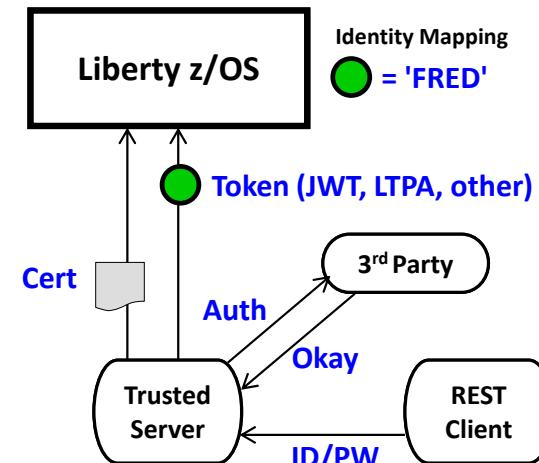
Client supplies certificate

Server validates cert and  
maps to an identity

Registry options:

- LDAP
- SAF

## Third Party Authentication



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

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

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

Registry options:

- LDAP
- SAF



# Third Party Authentication Examples

The image displays two side-by-side screenshots of web pages illustrating third-party authentication.

**Left Screenshot: UPS Sign Up**

This screenshot shows the UPS "Sign Up" page. At the top, there's a banner stating "UPS is open for business: Service impacts related to Coronavirus ...More". Below the banner, the UPS logo is displayed. A "Sign Up / Log in" link and a "Search or Track" input field are visible. The main section is titled "Sign Up" and includes a link for users who already have an ID. It provides several social media integration options: Google, Facebook, Amazon, Apple, and Twitter. Below these, there's a section for entering personal information: Name\*, Email\*, User ID\*, Password\*, and Phone. The password field has a "Show" link next to it. A "Feedback" button is located on the right side of the form.

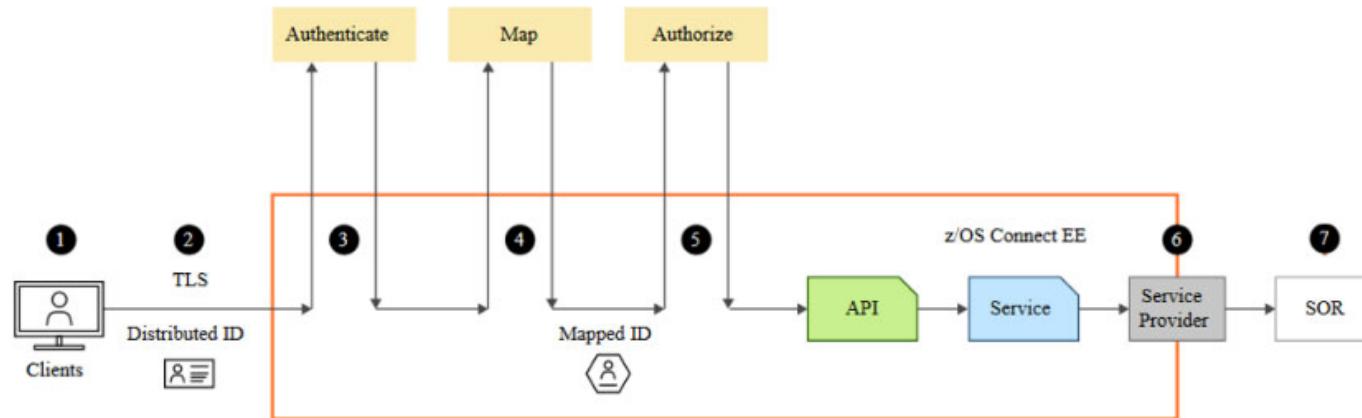
**Right Screenshot: myNCDMV Log In**

This screenshot shows the "Log In" page for myNCDMV. The background features a scenic view of autumn foliage. The page has "Log In" and "Sign Up" tabs at the top. The "Log In" tab is selected. It contains fields for "Email Address" (with placeholder "name@example.com") and "Password" (with placeholder "\*\*\*\*\*"). There's also a "Remember Me" checkbox. Below these are "Log In" and "Forgot Password" buttons. Further down, there are three social media login options: "Continue with Apple", "Continue with Facebook", and "Continue with Google". A "Continue as Guest" link is also present. At the bottom, a notice for public computer users states: "NOTICE FOR PUBLIC COMPUTER USERS - If you sign in with Google, Apple, or Facebook you are also signing into that account on this computer. Remember to sign out when you're done." The page is powered by "payit".

mitchj@us.ibm.com

© 2018, 2021 IBM Corporation

# 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

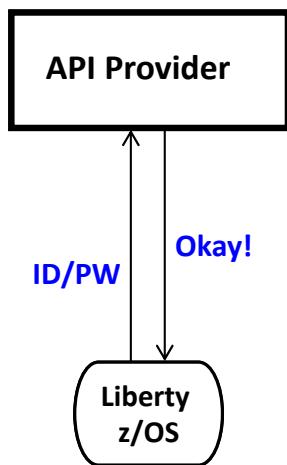
# API Requester Authentication



z/OS Connect EE

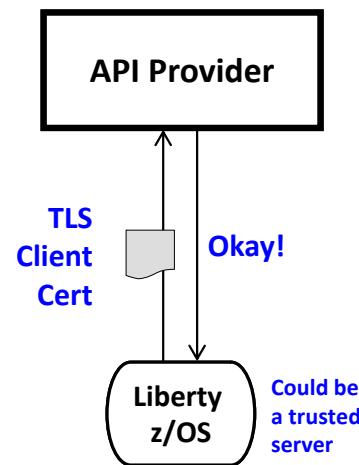
Several different ways this can be accomplished:

## Basic Authentication



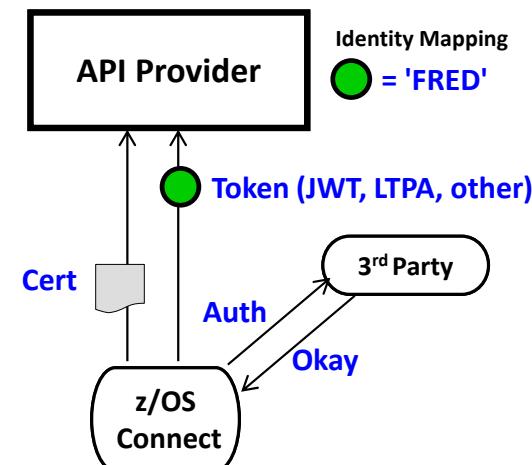
zCEE supplies ID/PW or  
ID/Passticket

## Client Certificate



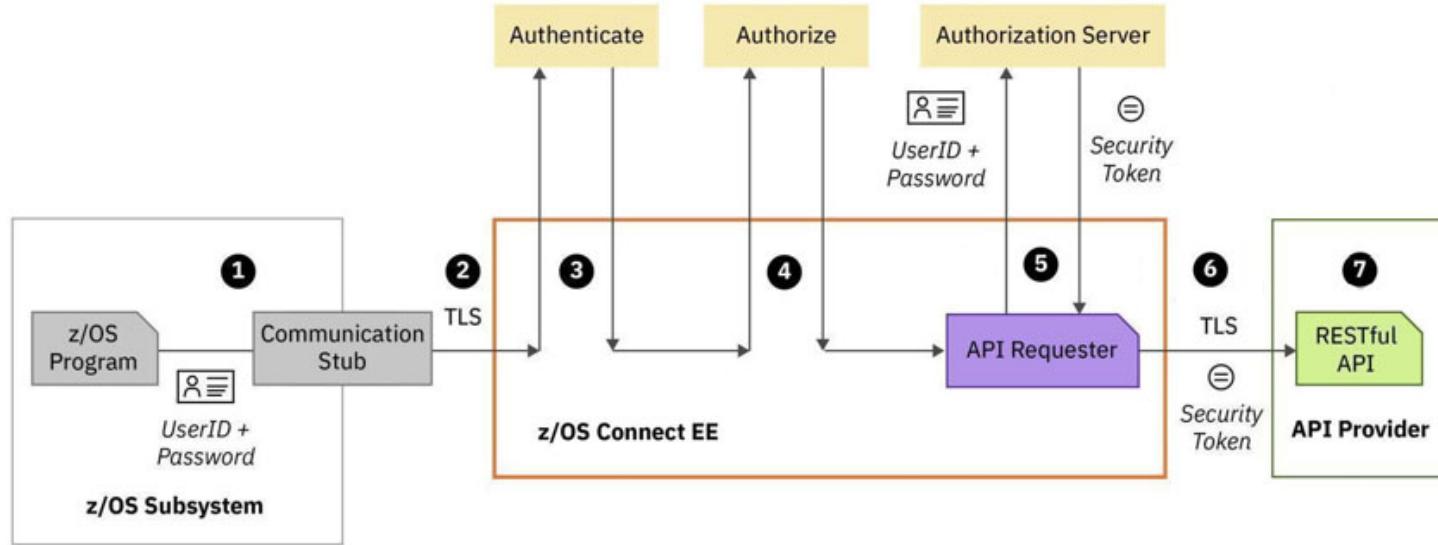
Server prompts for certificate  
zCEE supplies certificate

## Third Party Authentication



zCEE authenticates to 3<sup>rd</sup> party sever  
zCEE receives a trusted 3<sup>rd</sup> party token  
Token flows to API Provider

# Typical z/OS Connect EE API Requester security flow



1. A user ID and password can be used for basic authentication by the z/OS Connect EE server
2. Connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server can use TLS
3. Authenticate the CICS, IMS, or z/OS application.
4. Authorize the authenticated user ID to connect to z/OS Connect EE and to perform specific actions on z/OS Connect EE API requesters
5. Pass the user ID and password credentials to an authorization server to obtain a security token.
6. Secure the connection to the external API provider, and provide security credentials such as a security token to be used to invoke the RESTful API
7. The RESTful API runs in the external API provider

# z/OS Connect Wildfire Github Site

<http://tinyurl.com/y28fsezs>



The screenshot shows a GitHub repository page for 'ibm-wsc/zCONNEE-Wildfire-Workshop'. The left sidebar lists various branches and topics, with 'exercises' highlighted and circled in red. The main content area shows a list of files under the 'master' branch, all uploaded by user 'emitchj'. The files are primarily PDFs related to developing APIs for various IBM services like CICS, IMS, MVS, RESTful APIs, and MQ.

File Name	Description	Last Updated
Developing CICS API Requester Applications.pdf	Add files via upload	2 months ago
Developing IMS API Requester Applications.pdf	Add files via upload	2 months ago
Developing MVS Batch API Requester Applications.pdf	Add files via upload	2 months ago
Developing RESTful APIs for DVM VSAM Services.pdf	Add files via upload	20 days ago
Developing RESTful APIs for DVM VSAMCICS Services.pdf	Add files via upload	20 days ago
Developing RESTful APIs for Db2 REST Services.pdf	Add files via upload	2 months ago
Developing RESTful APIs for HATS REST Services.pdf	Add files via upload	2 months ago
Developing RESTful APIs for IMS Database REST Services....	Add files via upload	2 months ago
Developing RESTful APIs for IMS Transactions.pdf	Add files via upload	2 months ago
Developing RESTful APIs for MQ.pdf	Add files via upload	2 months ago
Developing RESTful APIs for MVS Batch.pdf	Add files via upload	2 months ago
Developing RESTful APIs for a CICS COMMAREA program.pdf	Add files via upload	2 months ago
Developing RESTful APIs for a CICS Container program.pdf	Add files via upload	2 months ago

- Contact your IBM representative to schedule access to these exercises

mitchj@us.ibm.com

© 2018, 2021 IBM Corporation



## **/questions?thanks=true**

Thank you for listening.

- z/OS Connect EE Users Group: <https://www.linkedin.com/groups/8731382/>