



IBM z/OS Connect Enterprise Edition

Introduction and Overview

Mitch Johnson

mitchj@us.ibm.com

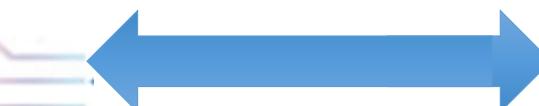
Washington System Center



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
- z/OS Connect Security

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



z/OS Connect EE

CICS

IMS/TM

IMS/DB

Db2

MQ

IBM File Manager⁺

3270

DVM⁺

MVS

WAS

Custom*

+ HCL and Rocket Software

*Other Vendors or your own implementation

/but_first, what_is_REST?

What makes an API “RESTful”?

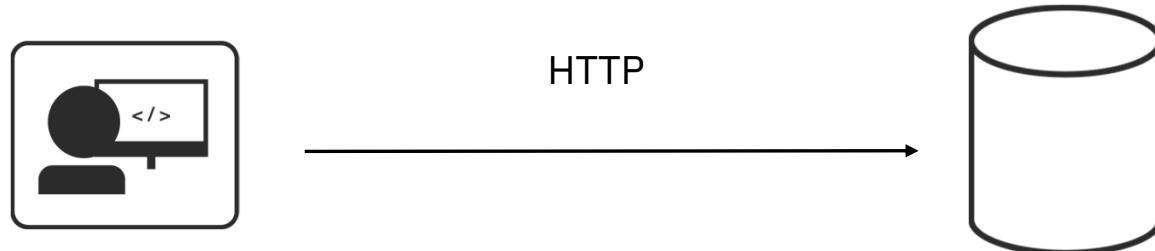
REST is an Architectural Style

REST stands for **R**epresentational **S**tate **T**ransfer.

An architectural style for **accessing** and **updating** data.

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
- It should access all the resources from the server using only 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 JSON
- REST based services follow some of the above principles and 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, REST APIs are not

RESTful Examples



z/OS Connect EE

z/OS Connect Enterprise Edition:

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

GET /account?number=1234

PUT /account/1234 +  (*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 you are doing it wrong)

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 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?

Ubiquitous Foundation	It's based on HTTP, which operates on TCP/IP, which is a ubiquitous networking topology.
Relatively Lightweight	Compared to other technologies (for example, SOAP/WSDL), the REST/JSON pattern is relatively light protocol and data model, which maps well to resource-limited devices.
Relatively Easy Development	Since the REST interface is so simple, developing the client involves very few things: an understanding of the URI requirements (path, parameters) and any JSON data schema.
Increasingly Common	REST/JSON is becoming more and more a de facto "standard" for exposing APIs and Microservices. As more adopt the integration pattern, the more others become interested.
Stateless	REST is by definition a stateless protocol, which implies greater simplicity in topology design. There's no need to maintain, replicate or route based on state.

How do we describe a REST API?



/swagger/open_api

The industry standard framework for describing RESTful APIs.

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⁺

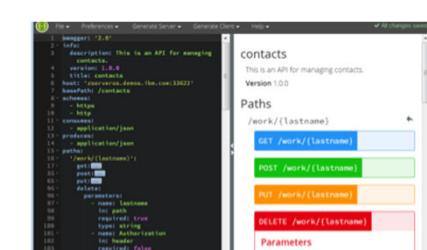
Swagger UI allows API consumers to easily browse and try APIs based on Swagger Doc.



The screenshot shows the Swagger UI interface for a 'contacts' API. It displays a list of operations under the 'default' path: 'DELETE /work/{lastname}', 'GET /work/{lastname}', 'POST /work/{lastname}', and 'PUT /work/{lastname}'. The 'POST' operation is highlighted with a green background. At the bottom, there is a note: 'BASE URL: http://server:8080/api-docs, API VERSION: 1.0.0'.

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 definitions for 'Contact' and 'Work', paths for 'GET /work/{lastname}', 'POST /work/{lastname}', 'PUT /work/{lastname}', and 'DELETE /work/{lastname}', and parameters for 'lastname'. The 'Paths' section is expanded, showing the four methods listed above.

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

Swagger Example

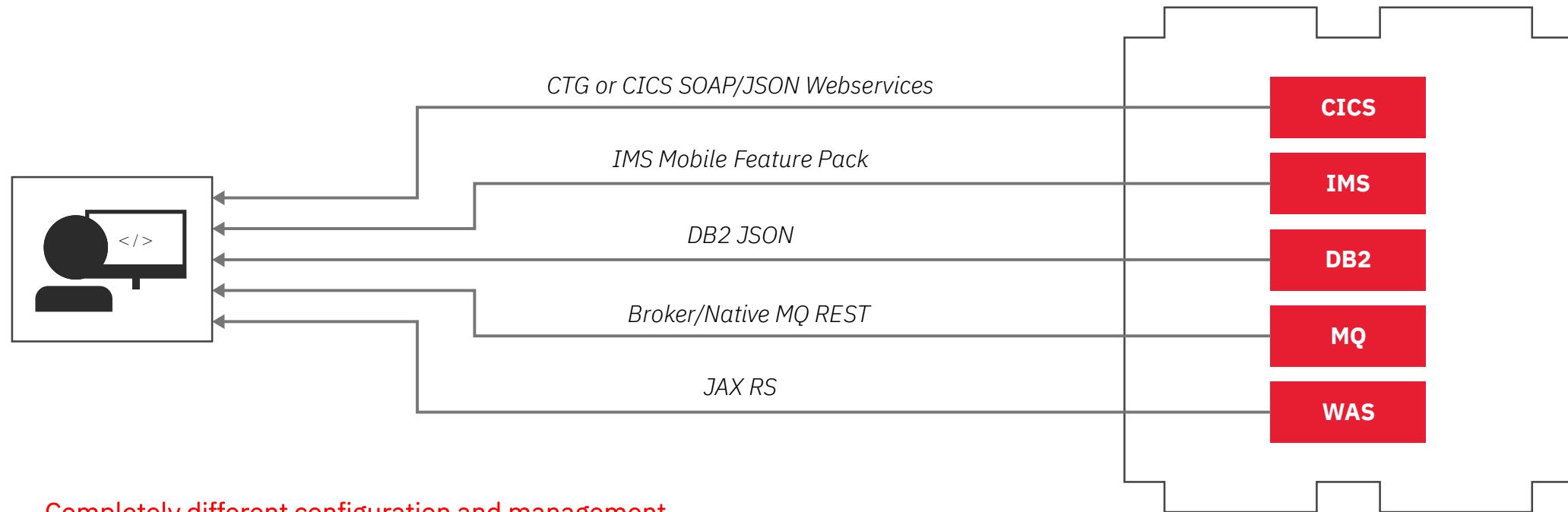




Why /zos_connect_ee?

Truly RESTful APIs to and from your mainframe.

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



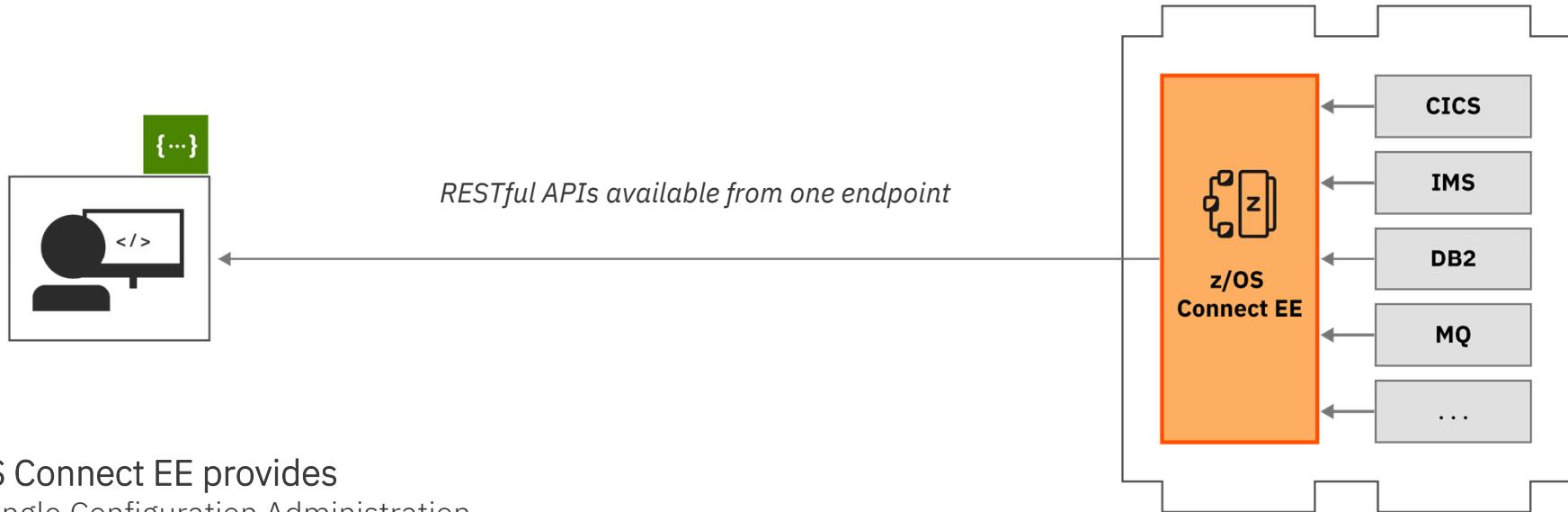
Completely different configuration and management.

Multiple endpoints for developers to call/maintain access to.

These are typically not RESTful!

A single entry point was needed

Expose z/OS resources without writing any code.



z/OS Connect EE provides

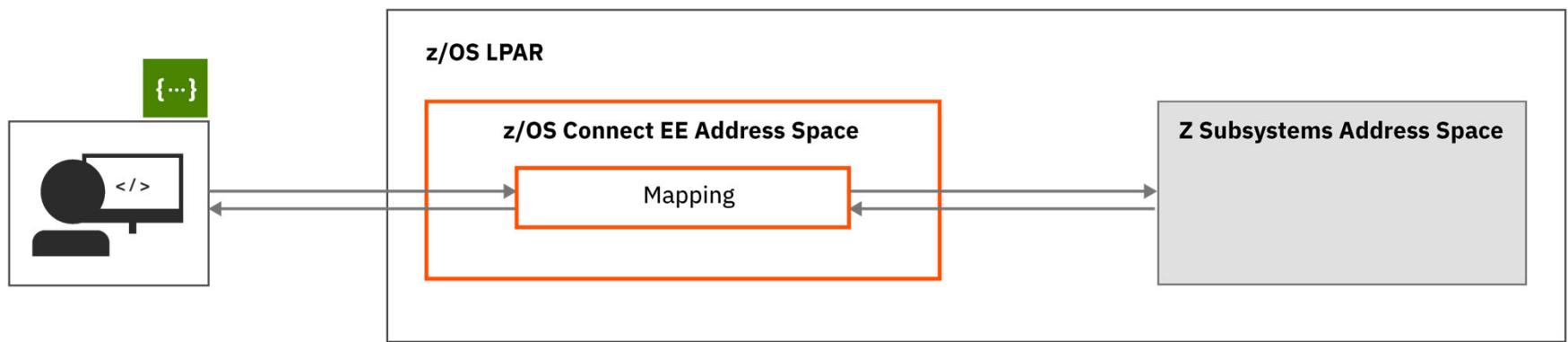
- Single Configuration Administration
- Single Security Administration
- With sophisticated mapping of truly RESTful APIs to existing mainframe and services data without writing any code.



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

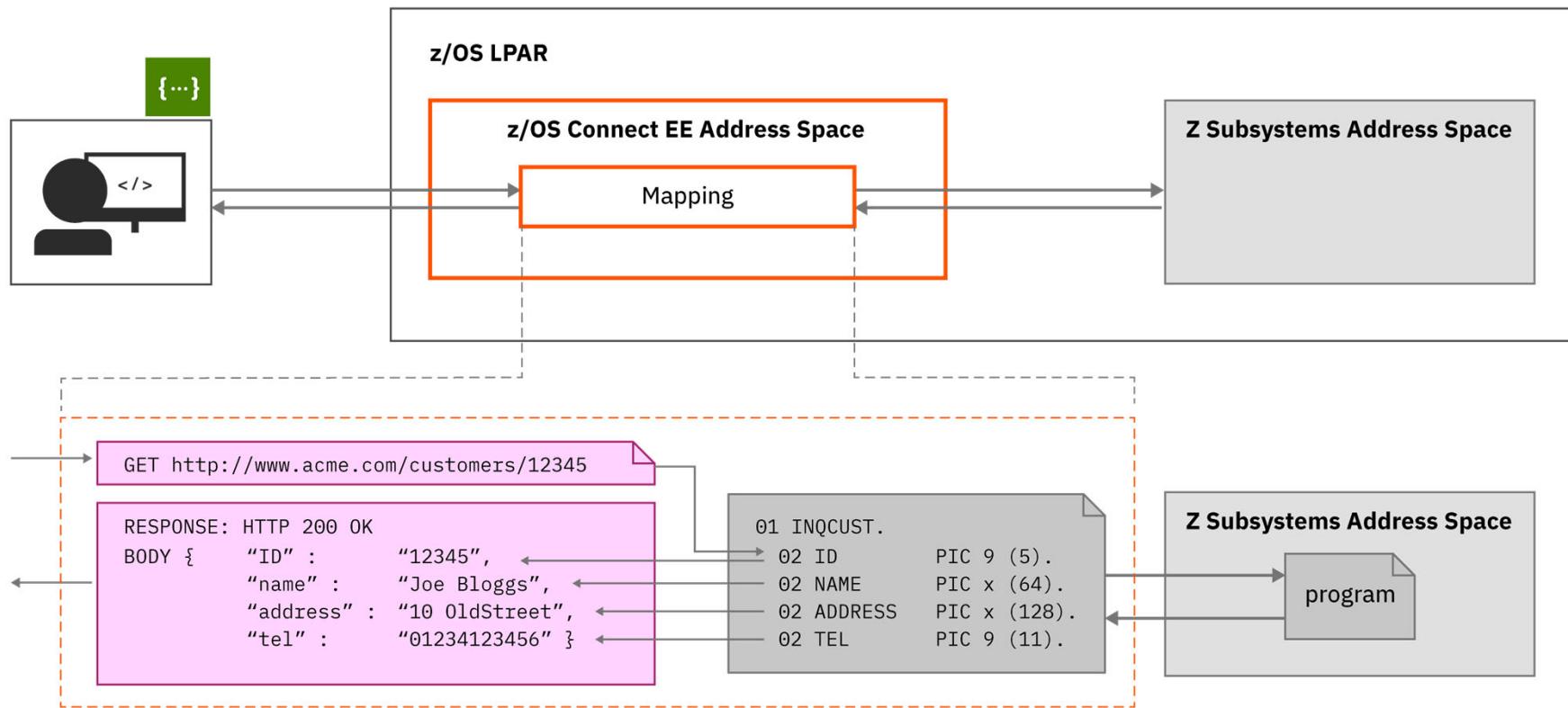
Let's Start with Data mapping

Converting JSON to the target's subsystem's normal format



Data mapping Example

A closer look



COBOL versus JSON 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" : 9999999999999999.99,
      "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>
      </field>
    </segment>
  </message>
</ns2:message>

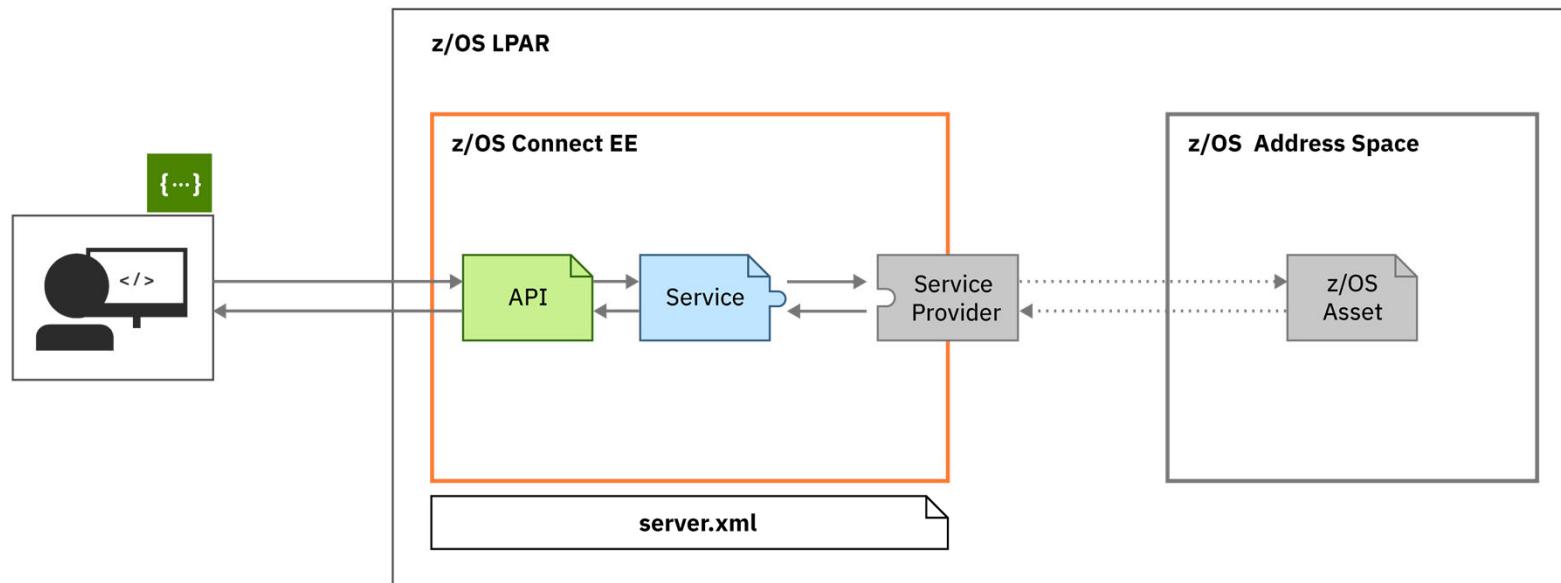
```

https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/char/src/tpc/db2z_endianness.html

Slide 21

MJ1 Mitch Johnson, 6/16/2020

Steps to expose a z/OS application



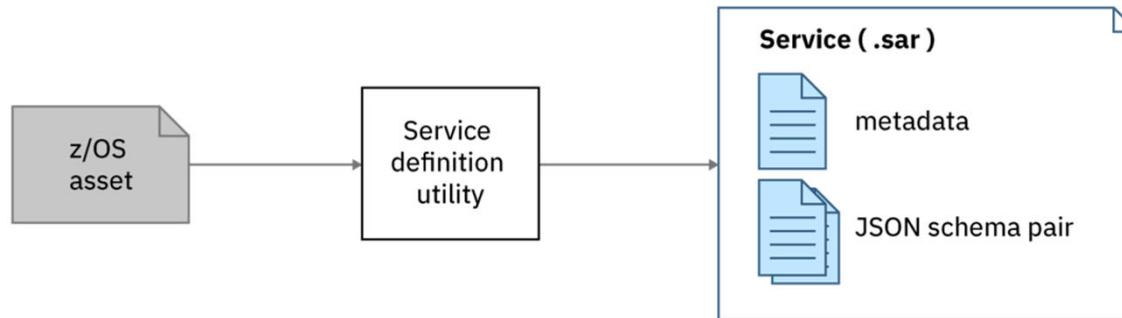
- The API provides the RESTful interface is ready to be consumed by a client and it requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS Asset (e.g. CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific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

Steps to expose a z/OS application

1. Create a service

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 `.sar` file for the z/OS application

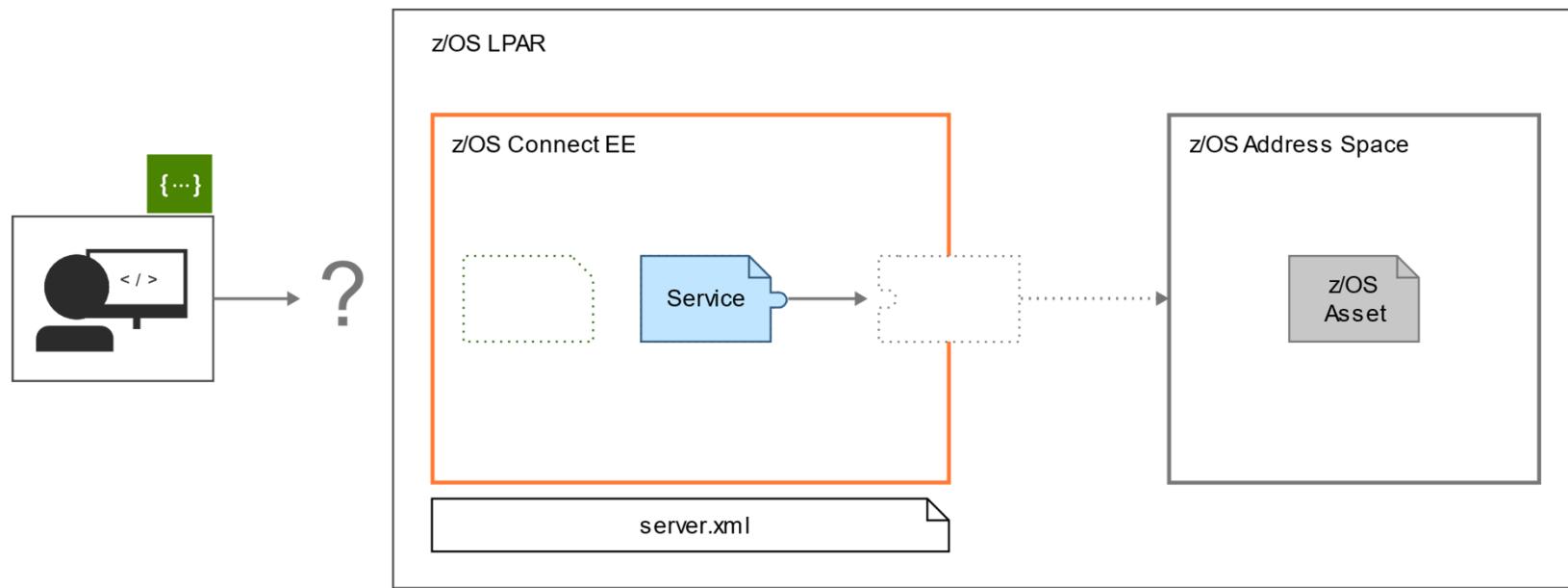
- Eclipse based - API Toolkit (CICS, IMS TM, IMS DB, Db2 and MQ)
- Command line - z/OS Connect EE Build Toolkit (MVS Batch, IBM File Manager and HATS)
- Eclipse based - DVM Toolkit

 ibm.biz/zosconnect-sar-creation



Steps to expose a z/OS application

2. Deploy and export the service

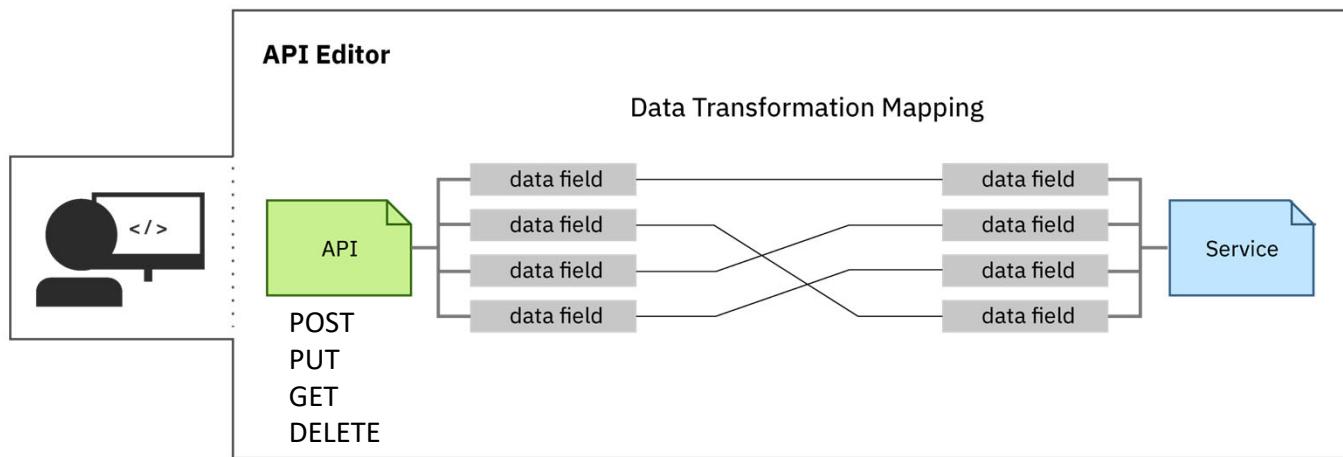


Deploy the service archive file generated in **Step 1** using the right-click deploy in **the API toolkit**.

ibm.biz/zosconnect-define-services

Steps to expose a z/OS application

3. Create an API using exported services



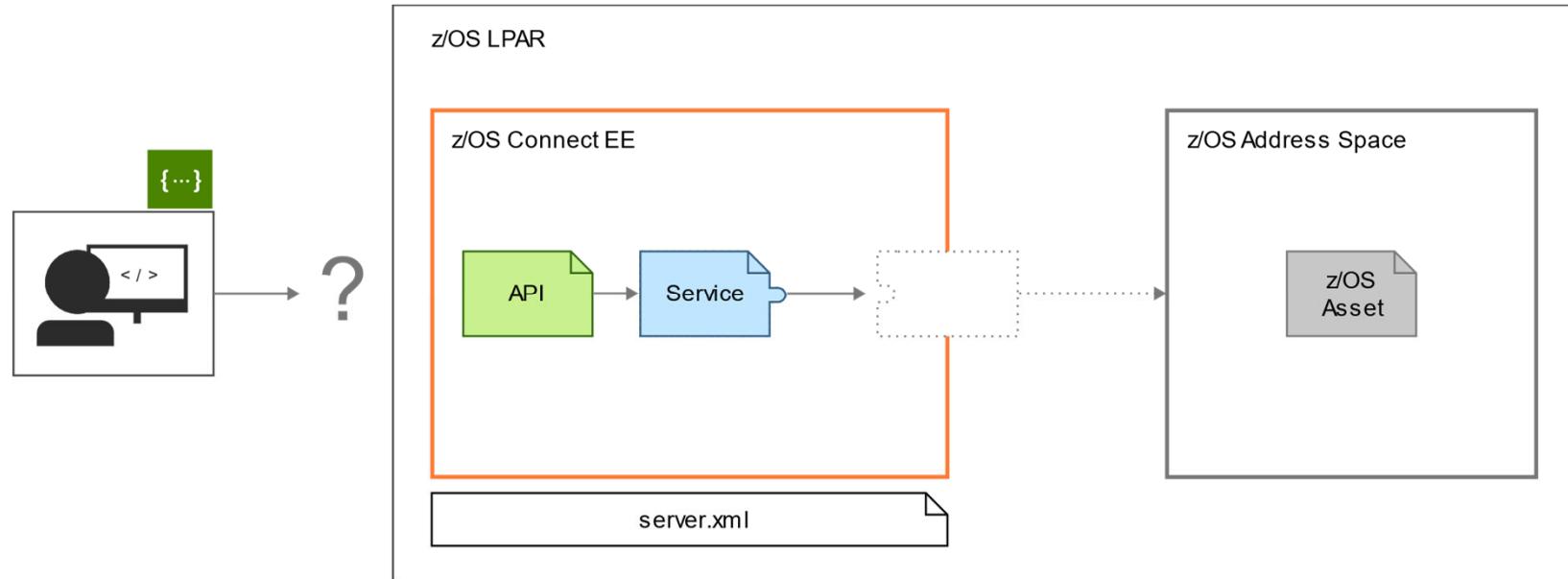
- Import the service archive file into the **API toolkit**, and start designing the RESTful API.
- Provides additional data mapping
- Use the editor to describe the API and how it maps to underlying services.

 ibm.biz/zosconnect-create-api



Steps to expose a z/OS application

4. Deploy the API

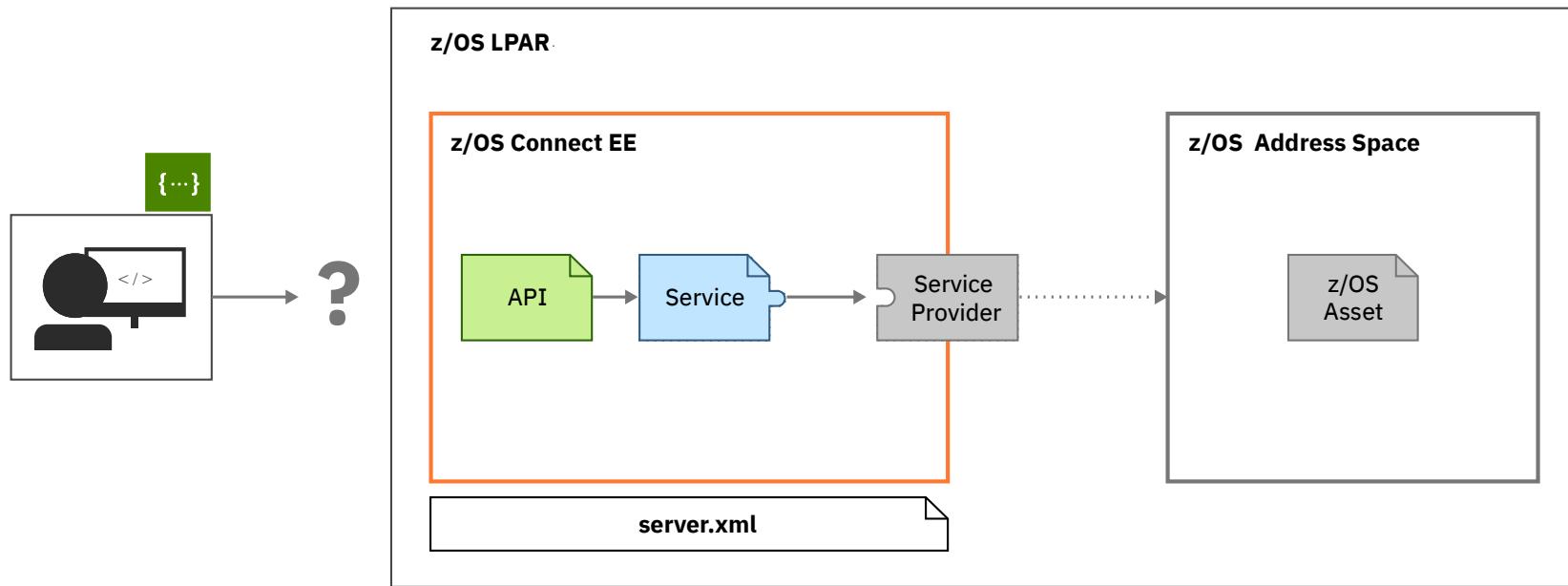


Deploy your API using the right-click deploy in **the API toolkit**

ibm.biz/zosconnect-deploy-api

Steps to expose a z/OS application

5. Configure the service provider

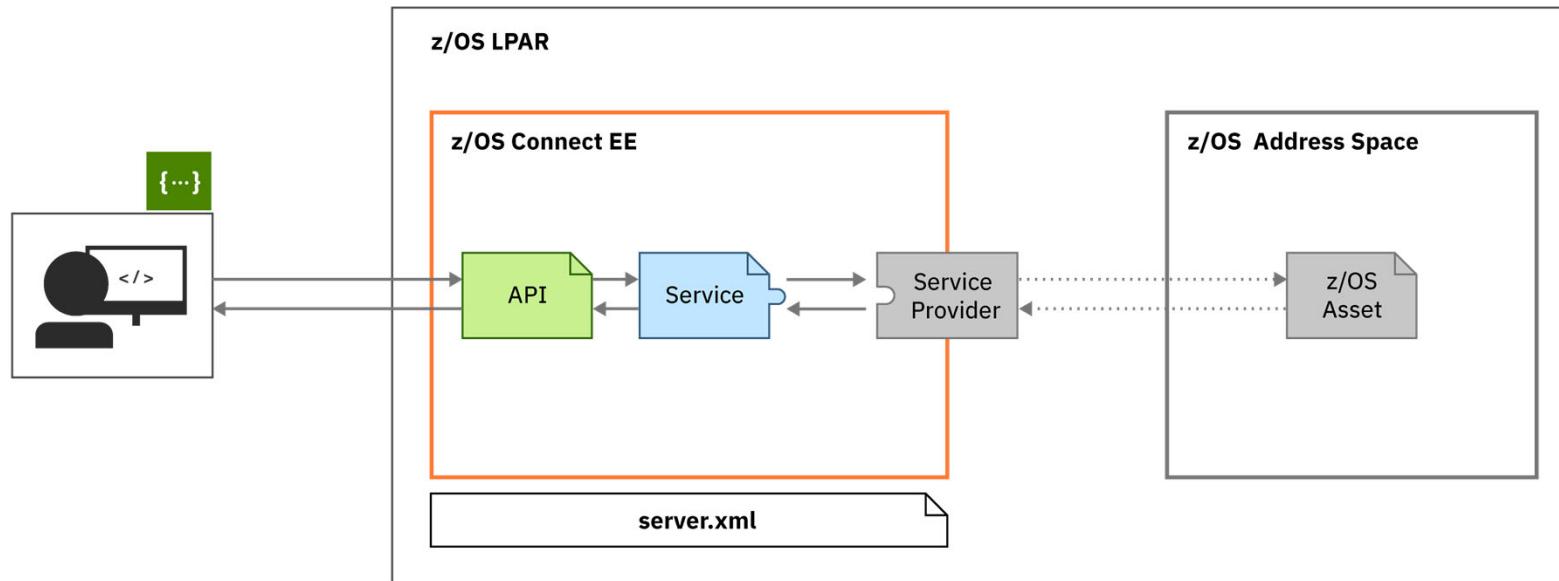


Configure the system-appropriate service provider to connect to your backend system in your `server.xml`.

 ibm.biz/zosconnect-configuring

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)



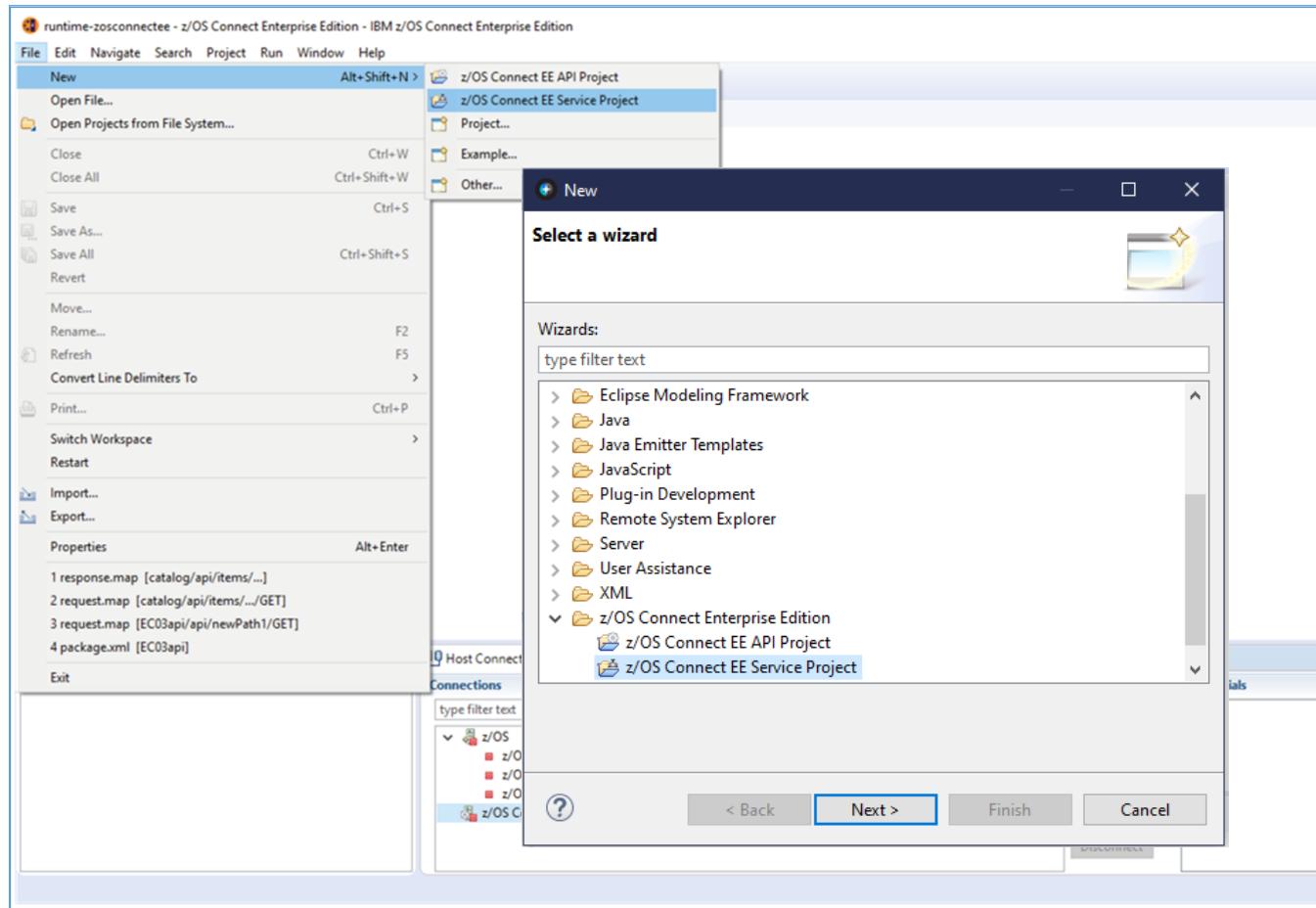
/api_toolkit/services

Simple **service creation.**

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



z/OS Connect EE

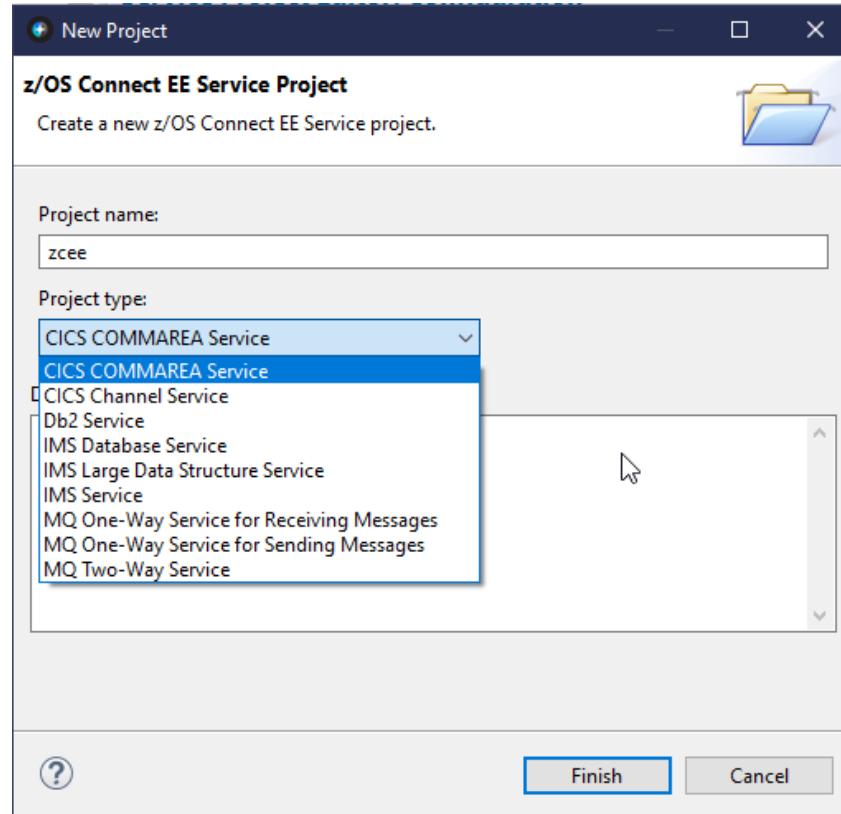


Use the **API toolkit** to create services through Eclipse-based tooling.

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

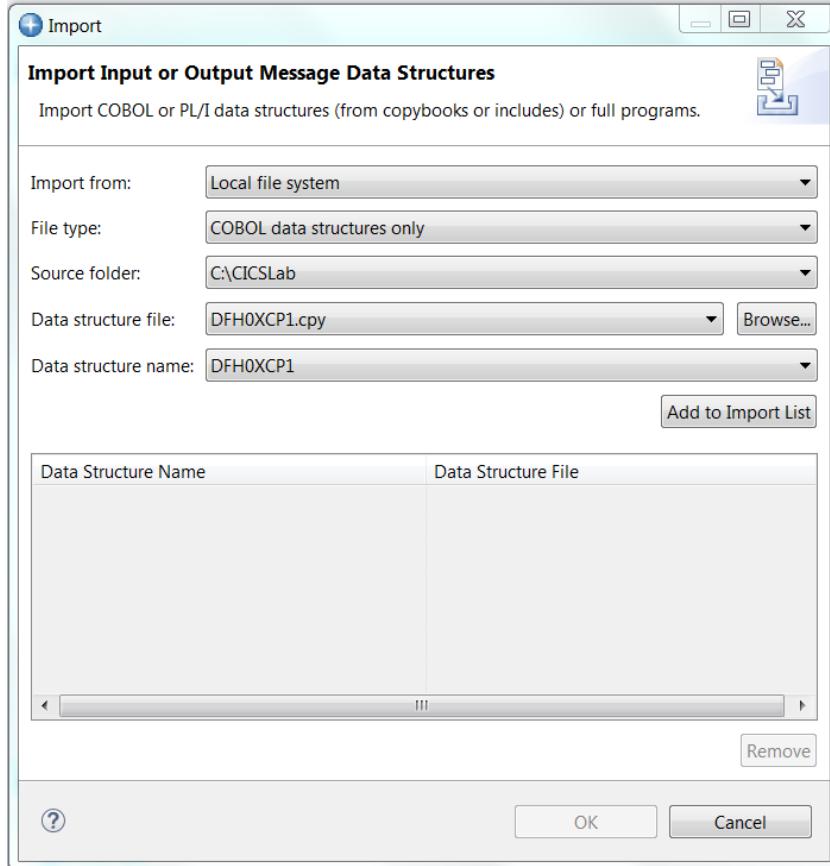
Service creation – a common interface



A common interface for service creation, agnostic 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.

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 service interface definition

*inquireSingle Service *inquireSingleRequest

Service Interface Definition

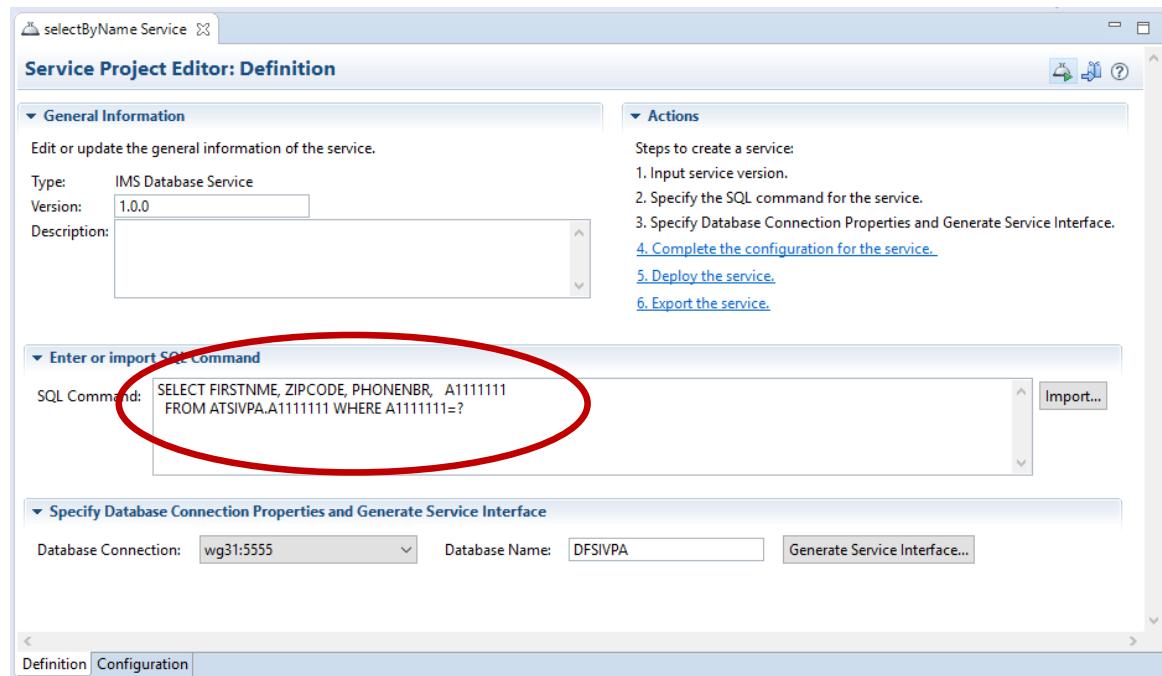
Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

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

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.

API toolkit – Creating Services for IMS DB

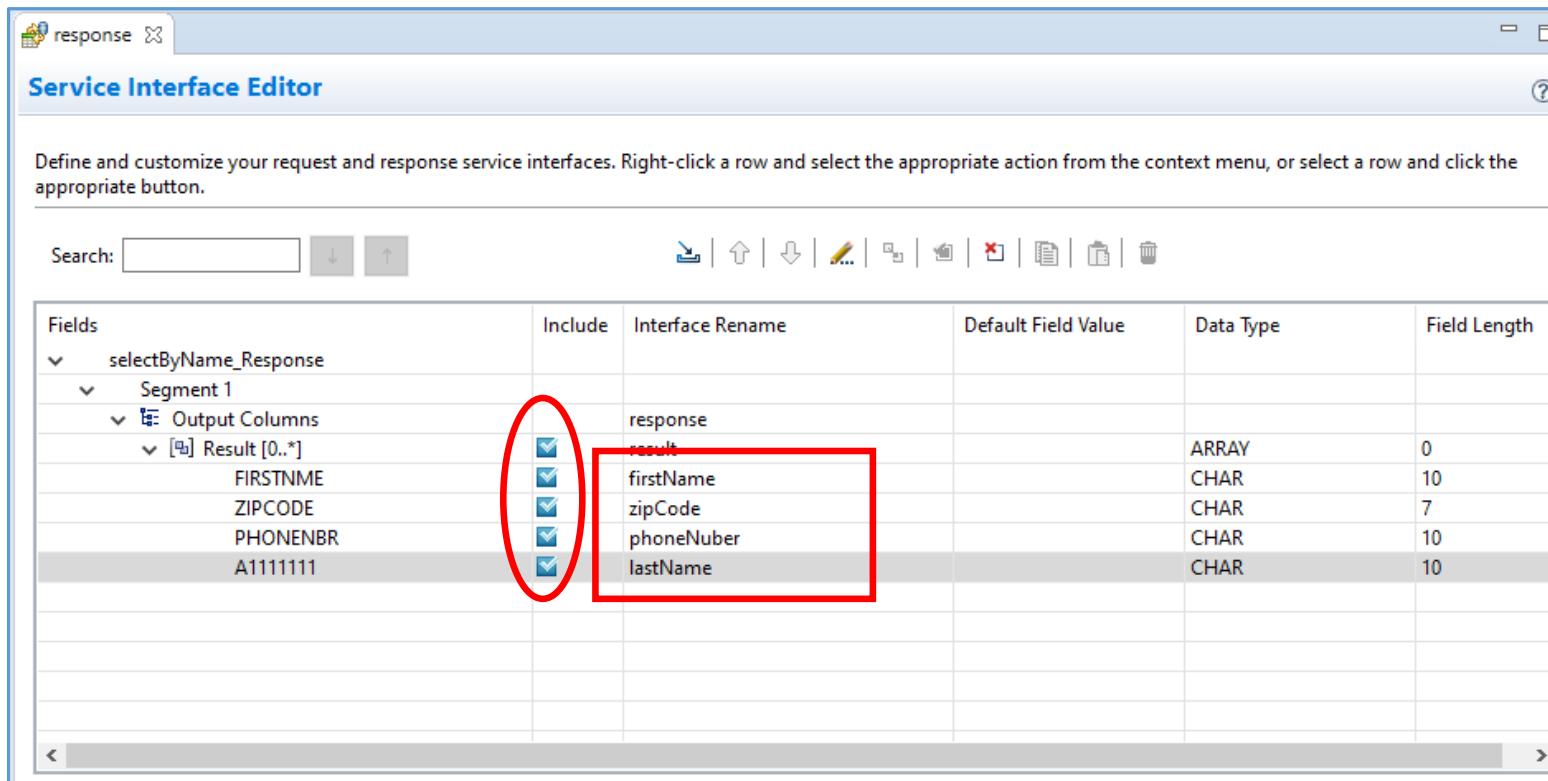
Creating a service project from the IMS Catalog



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

API toolkit – Creating Services for IMS DB

Again allows editing a service interface definition*



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 toolbar with icons for copy, paste, up, down, edit, etc. The main table has 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..*]					
FIRSTNAME	<input checked="" type="checkbox"/>	response		ARRAY	0
ZIPCODE	<input checked="" type="checkbox"/>	result		CHAR	10
PHONENBR	<input checked="" type="checkbox"/>	firstName		CHAR	7
A1111111	<input checked="" type="checkbox"/>	zipCode		CHAR	10
	<input checked="" type="checkbox"/>	phoneNuber		CHAR	10
	<input checked="" type="checkbox"/>	lastName		CHAR	10

*Using a slightly different process

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

Editing a response message

*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 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 to the same resource

The image shows two side-by-side screenshots of the Service Interface Editor. Both screenshots display a table of fields for a service interface named 'cscvincSelectService Service'.

Top Screenshot: This screenshot shows a row for 'REQUEST_CONTAINER' with the 'ACTION' field set to 'S'. A red circle highlights the value 'S' in the 'Default Field Value' column.

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

Bottom Screenshot: This screenshot shows a row for 'REQUEST_CONTAINER' with the 'ACTION' field set to 'I'. A red circle highlights the value 'I' in the 'Default Field Value' column.

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

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

mitchj@us.ibm.com

This screenshot shows the Service Project Editor with the 'Definition' tab selected. It displays the configuration for a 'CICS Channel Service' version 1.0.0.

General Information: The 'Program' field is set to 'CSCVINC', highlighted with a red circle.

Actions: A list of steps to create a service, including 'Complete the configuration for the service' (also highlighted with a red circle).

Program: CSCVINC

Define Request and Response Service Interface: Create Service Interface... Import

Request service interface: cscvinc

Response service interface: cscvinc

Set advanced data conversion options: Advanced

Required Configuration: Enter the required configuration for this service. Coded character set identifier (CCSID): 37, Connection reference: cscvinc

Optional Configuration: Transaction ID: CSMI, Transaction ID usage: EIB_ONLY, Use context containers: Context containers HTTP headers: Add another

© 2018, 2021 IBM Corporation

API toolkit – Creating Services for IMS

Creating a “GET” service interface request definition

*ivtnoDisplayService Service *ivtnoDisplayRequest

Service Interface Definition

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

Search:

Fields	Include	Interface rename	Default Field
ivtnoDisplayRequest			cscvincSelectService Service
Segment 1			*ivtnoDisplayService Service
INPUT_MSG			
IN_LL	<input type="checkbox"/>	IN_LL	
IN_ZZ	<input type="checkbox"/>	IN_ZZ	
IN_TRANCODE	<input type="checkbox"/>	IN_TRANCODE	
IN_COMMAND	<input type="checkbox"/>	IN_COMMAND	IVTNO
IN_LAST_NAME	<input checked="" type="checkbox"/>	lastName	DISPLAY
IN_FIRST_NAME	<input type="checkbox"/>	IN_FIRST_NAME	
IN_EXTENSION	<input type="checkbox"/>	IN_EXTENSION	
IN_ZIP_CODE	<input type="checkbox"/>	IN_ZIP_CODE	

ivtnoDisplayService Service

Service Project Editor: Definition

General Information
Edit or update the general information of the service.
Type: IMS Service
Version: 1.0.0
Description:

Transaction code
Transaction code: IVTNO

Define Request and Response Service Interfaces
Create new request and response service interfaces or select existing ones.
Create Service Interface... Import Service Interface...
Request service interface: ivtnoDisplayRequest.si
Response service interface: ivtnoDisplayResponse.si
Set advanced data conversion options Advanced Options...

Service Project Editor: Configuration

Required Configuration
Enter the required configuration for this service.
Connection profile: IMSCONN
Interaction profile: IMSINTER

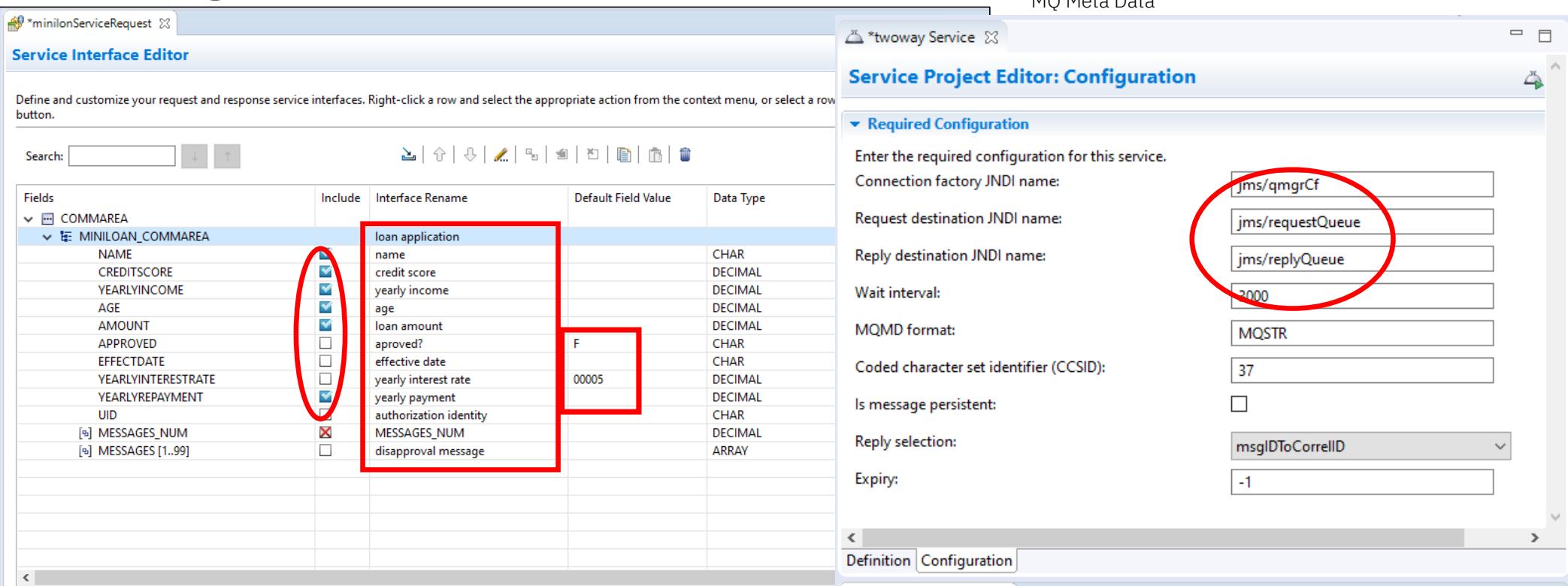
Optional Configuration
Enter the optional configuration for this service.
IMS destination override:
Program name:

The service developer creates distinct services for each function.

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

API toolkit – Creating Services for MQ

Creating a service interface definition



The screenshot shows two windows from the API toolkit:

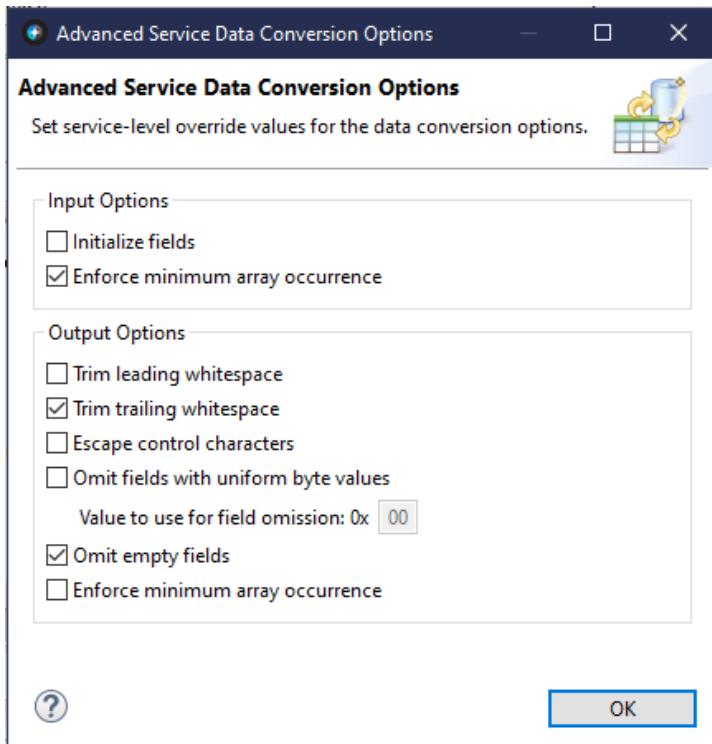
- Service Interface Editor:** This window displays a table of fields for a service interface named "minilonServiceRequest". The table has columns: Fields, Include, Interface Rename, Default Field Value, and Data Type. A red box highlights the "Interface Rename" column for the "loan application" row, which contains the value "loan application". Another red box highlights the "Default Field Value" column for the same row, which contains the value "F".
- Service Project Editor: Configuration:** This window shows configuration settings for a service named "twoWay Service". A red circle highlights the "Request destination JNDI name" field, which contains "jms/requestQueue".

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.



z/OS Connect EE

API toolkit – Advanced Data Conversion Options



Request Messages:

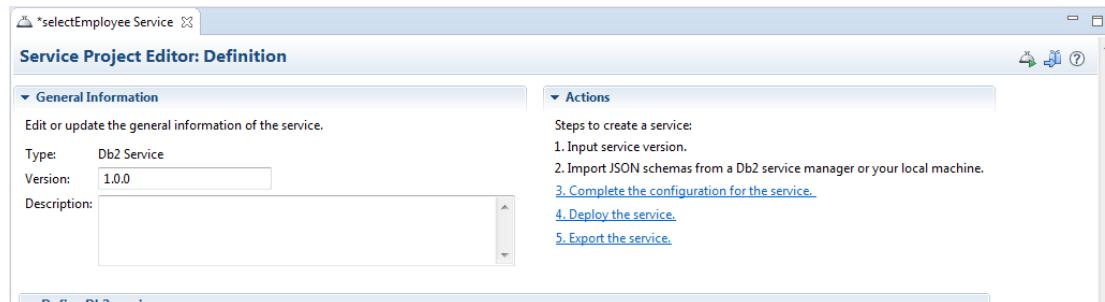
- Initialize fields
- Enforce minimum array occurrence

Response Messages:

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

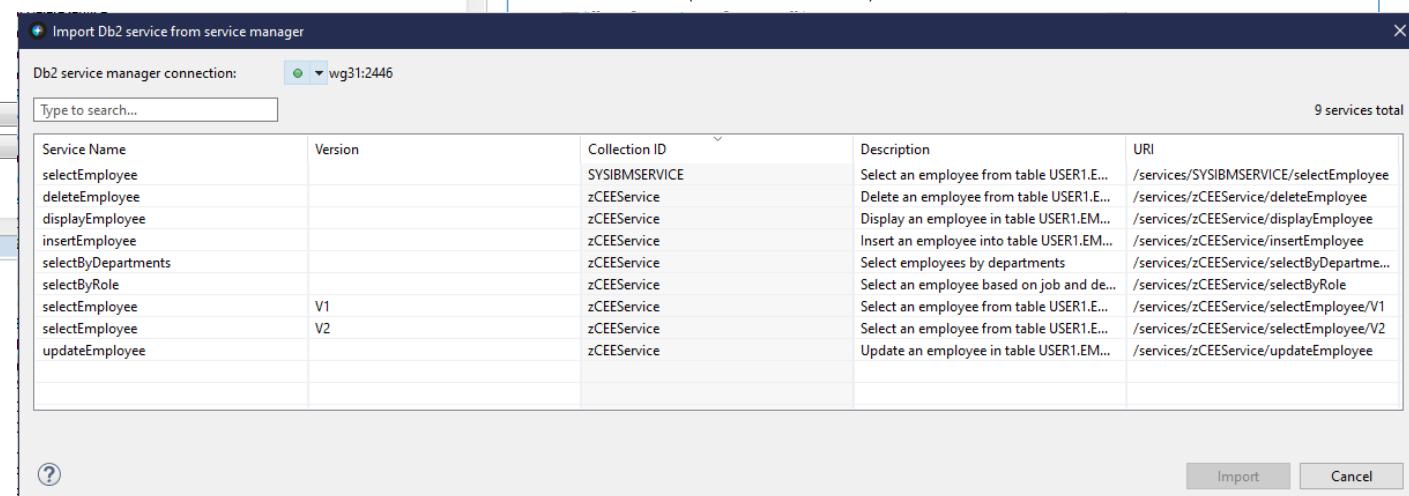
API toolkit – Creating Services for Db2

Creating a service project from Db2 REST service



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

- General Information:**
 - Type: Db2 Service
 - Version: 1.0.0
 - Description: [Empty]
- Actions:**
 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.](#)



The Import Db2 service from service manager dialog shows the following table of services:

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

Actions:

- Import
- Cancel

The service developer retrieves the Db2 REST services

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 with a service named 'InquireSingle'. The main window displays 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 'Request service interface' field, showing options like 'Create Service Interface...', 'Import Service Interface...', 'Edit', and 'Delete'. At the bottom of the main window, there are tabs for 'Overview' and 'Configuration', along with 'Properties', 'Progress', and 'Problems' buttons. A separate 'Deploy Service' dialog box is overlaid on the main window. It shows the 'z/OS Connect EE Server' dropdown set to 'wg31:9453'. The message 'The following services will be created on the server:' is displayed above a table. The table lists one service: 'Service name: inquireSingle, Version: 13.00, Type: CICS COMMAREA Se...'. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

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

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

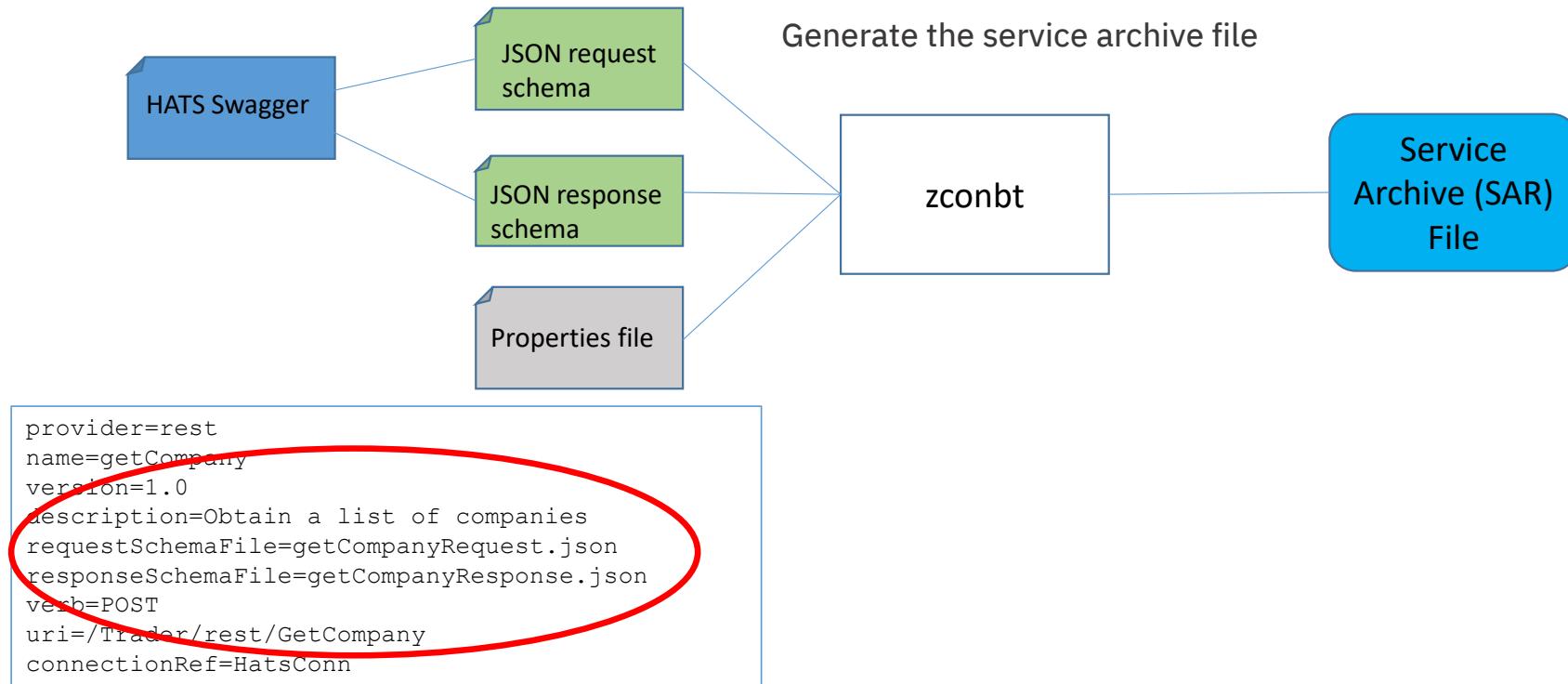


The screenshot shows the IBM z/OS Connect Enterprise Edition interface. On the left, the Project Explorer displays a service named 'InquireSingle'. The main window shows the 'Overview' tab of the 'InquireSingle Service' properties. Under 'General Information', the 'Type' is set to 'CICS COMMAREA Service' and 'Version' 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 'Export...' option in the 'Actions' section, 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 radio buttons for 'Workspace' (selected) and 'Local file system', a 'Folder' field set to '/services', a 'File name' field set to 'inquireSingle.sar', and an unchecked 'Overwrite service package file' checkbox. Buttons for '?', 'OK', and 'Cancel' are at the bottom.

Finally, you can export the service project as a **Service Archive file (.sar)**.

Creating Services without the Toolkit – REST

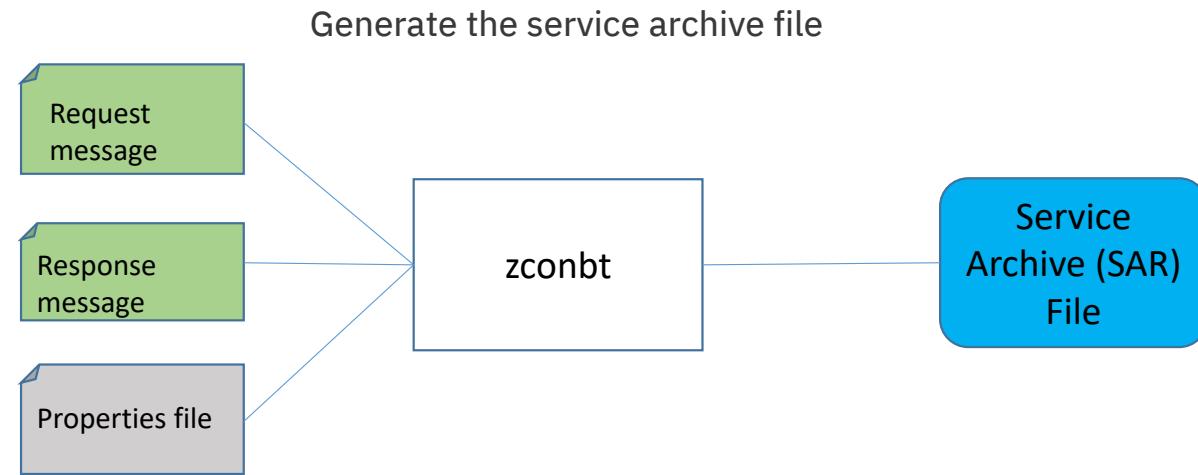
For HATS REST Services use the z/OS Connect Build toolkit (zconbt)



Creating Services without the Toolkit – MVS Batch

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

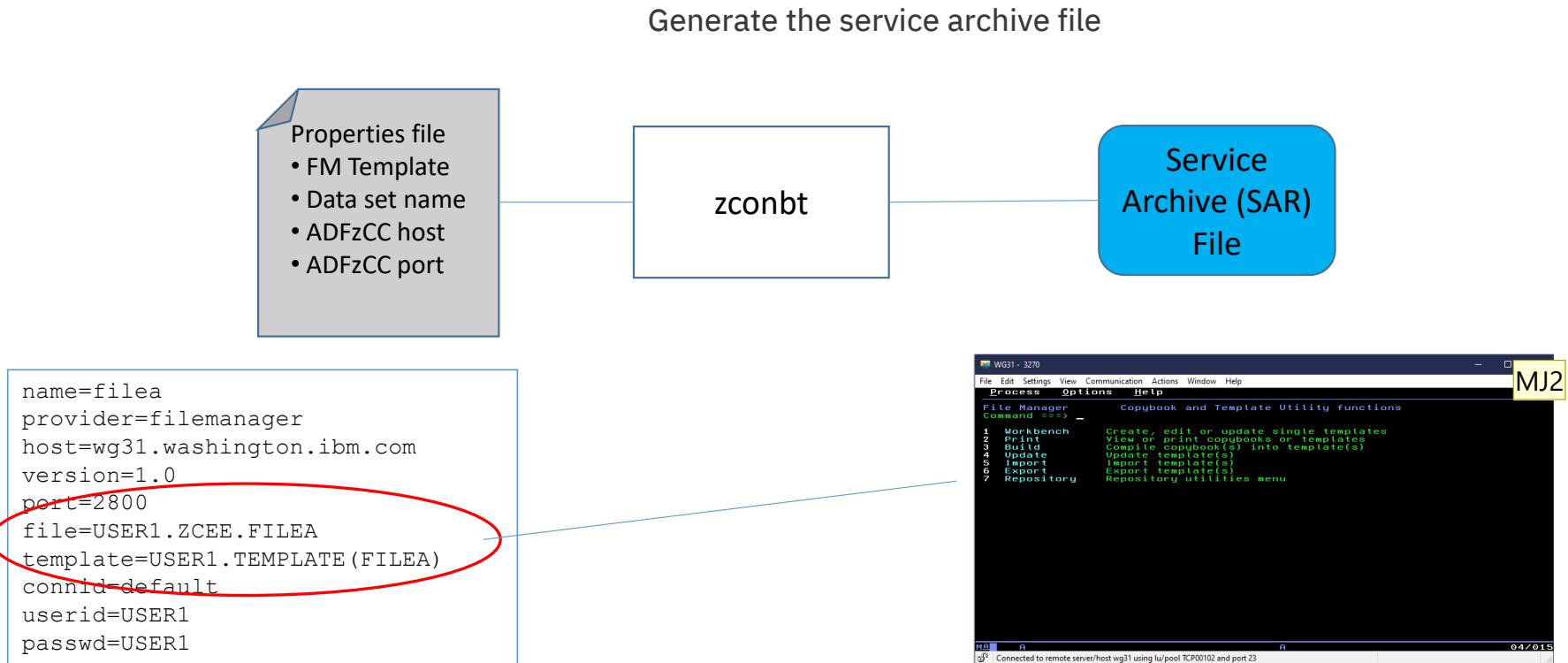
```
name=Filea
version=1.0
provider=wola
description=COBOL batch program
language=COBOL
program=ATSFILEA
register=FILEAZCON
connectionRef=wolaCF
requestStructure=fileareq.cpy
responseStructure=filearsp.cpy
```



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

Creating Services without the Toolkit – FM

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



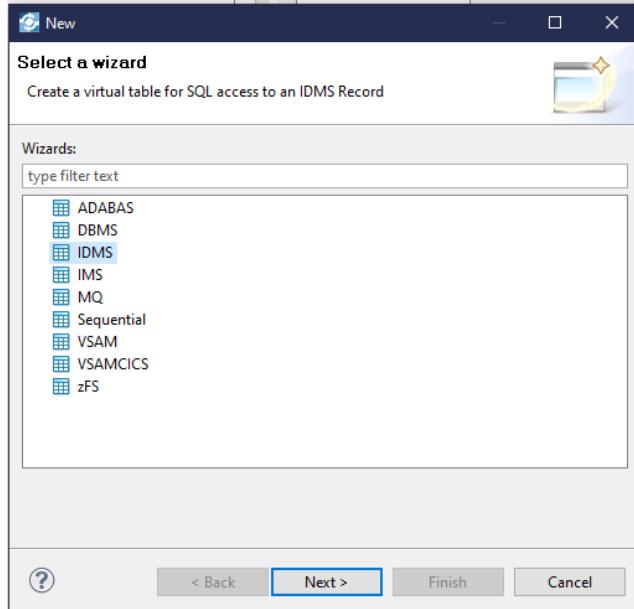
Slide 46

MJ2 Mitch Johnson, 9/2/2020

Creating Services without the Toolkit



For DVM use the DVM Studio



```
-- IEEE LOCATION: wg31.washington.ibm.com:1200/SUB/Data/AVZS/V1LU
-- Remarks: VSAMCICS - USER1.OFFICE.SUPPLIES
SELECT WS_ITEM_REF, WS_DESCRIPTION, WS_DEPARTMENT, WS_COST, WS_IN_
WS_ON_ORDER
FROM EXMPCAT LIMIT 1000;

INSERT INTO EXMPCAT WHERE (WS_ITEM_REF, WS_DESCRIPTION, WS_DEPARTM
WS_ON_ORDER) VALUES('9997','Mitch Johnson','010','002.90','0106'
INSERT INTO EXMPCAT WHERE (WS_ITEM_REF, WS_DESCRIPTION, WS_DEPARTM
WS_ON_ORDER) VALUES('9998','Mitch Johnson','010','002.90','0106'
INSERT INTO EXMPCAT WHERE (WS_ITEM_REF, WS_DESCRIPTION, WS_DEPARTM
WS_ON_ORDER) VALUES('9999','Mitch Johnson','010','002.90','0106'

UPDATE EXMPCAT SET WS_COST = '003.90' WHERE WS_ITEM_REF = 9999

MPCAT WHERE WS_ITEM_REF = 9999

Server Trace Console
WS_DESCRIPTION WS_DEPARTMENT WS_C
Mitch Johnson 10 002.
Mitch Johnson 10 002.
10 002.

Active Connections z/OS Connect REST Interface
DSN Name AVZS
Common Tools
Refresh
Generate SAR File(s)
Columns Group: 1 or 1
3 rows SQL Messages
Columns per group: 25
```



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

Quick and easy **API mapping**.

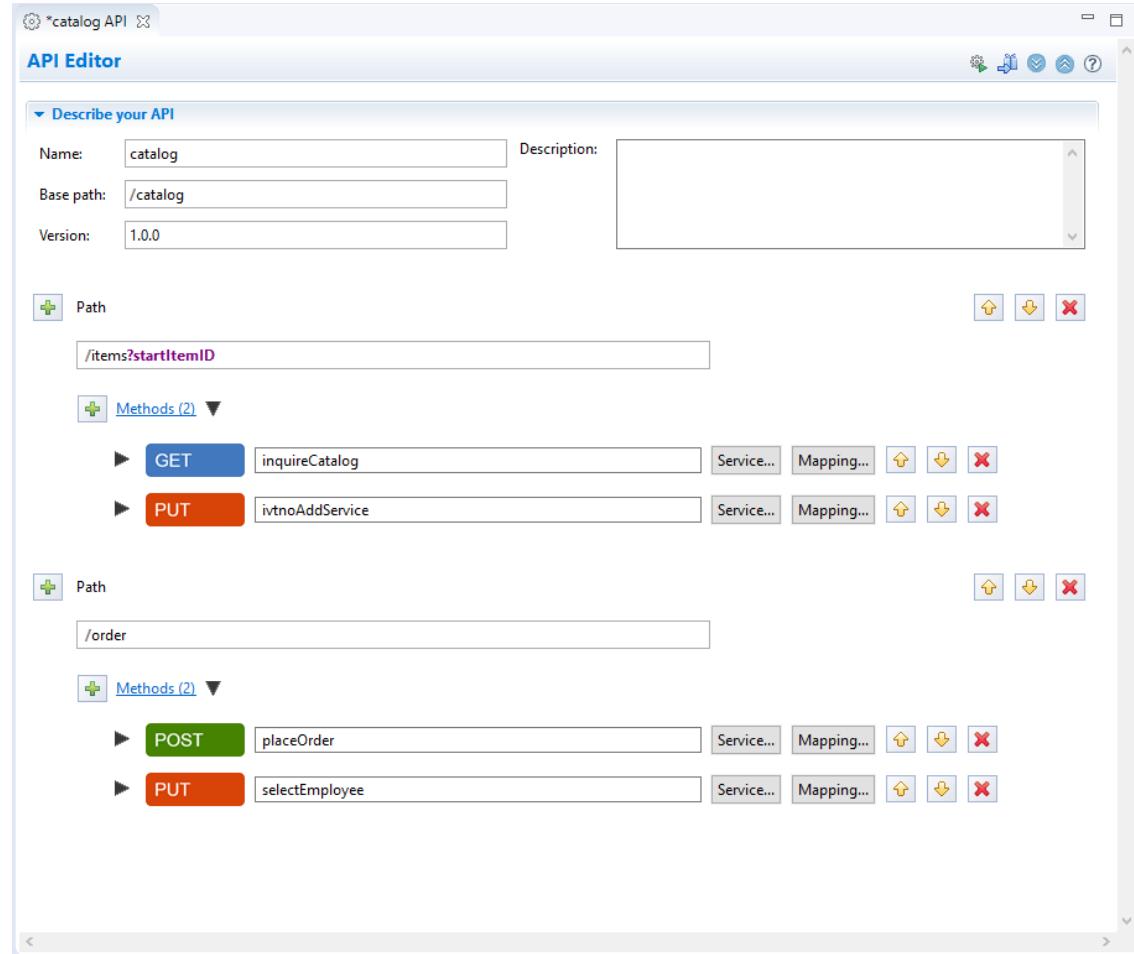
Remember: All service archives files are functionally equivalent regardless of how they are created



/api_toolkit/api_editor

Quick and easy **API mapping**.

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...)
 - ▶ PUT ivtnoAddService (Service... Mapping...)
- order API**:
 - Path**: /order
 - Methods (2)**
 - ▶ POST placeOrder (Service... Mapping...)
 - ▶ PUT selectEmployee (Service... Mapping...)

mitchj@us.ibm.com

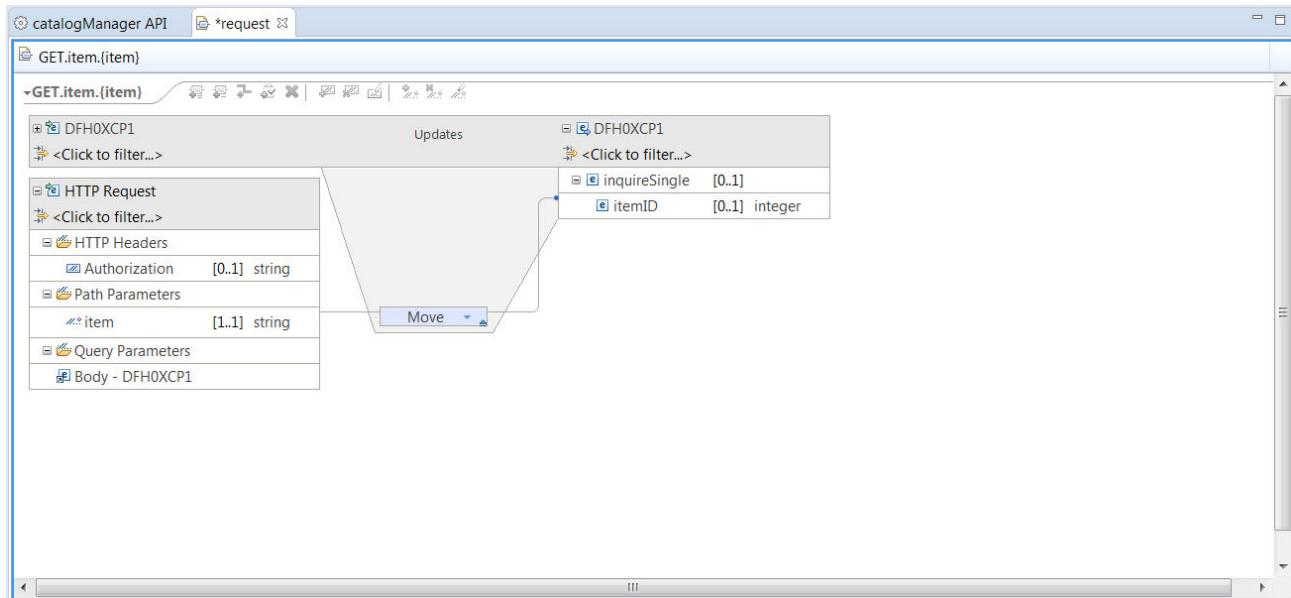
The **API toolkit** is designed to encourage RESTful API design.

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

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

API toolkit – API Editor

API mapping: Point-and-click interface



Map both the request and response for each API.

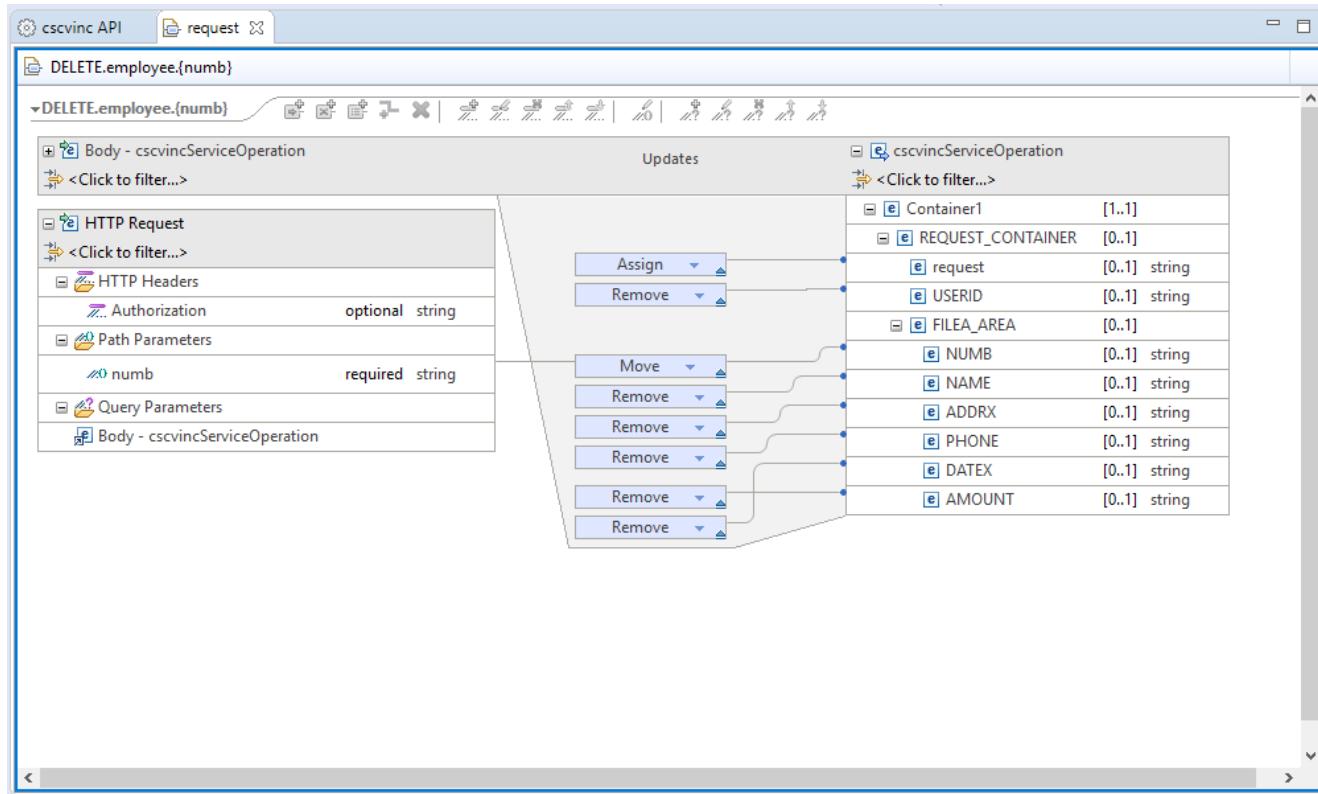
Map path and query parameters to native data structures.

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

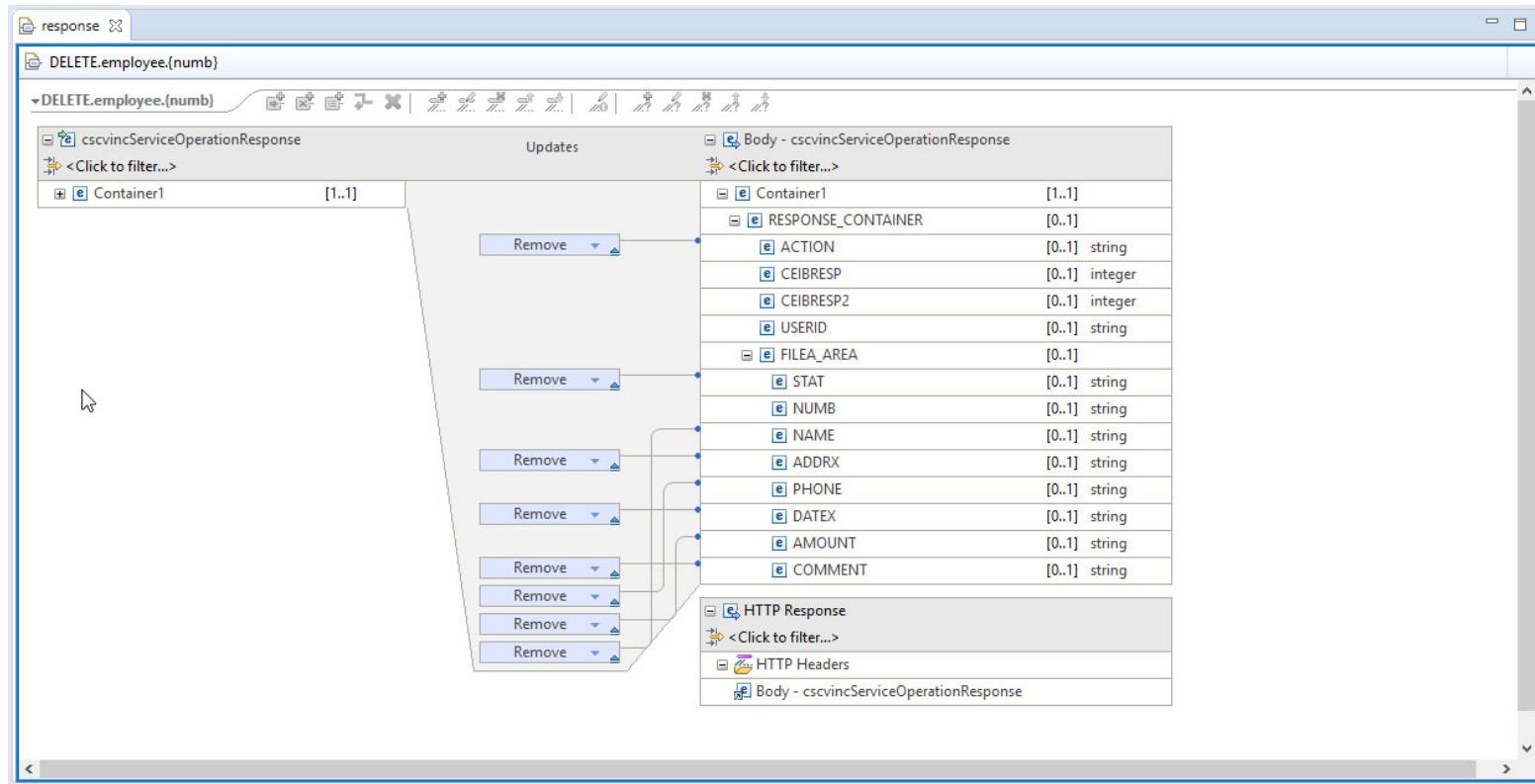
API mapping: Point-and-click interface



API toolkit – API Editor

API mapping: Point-and-click interface

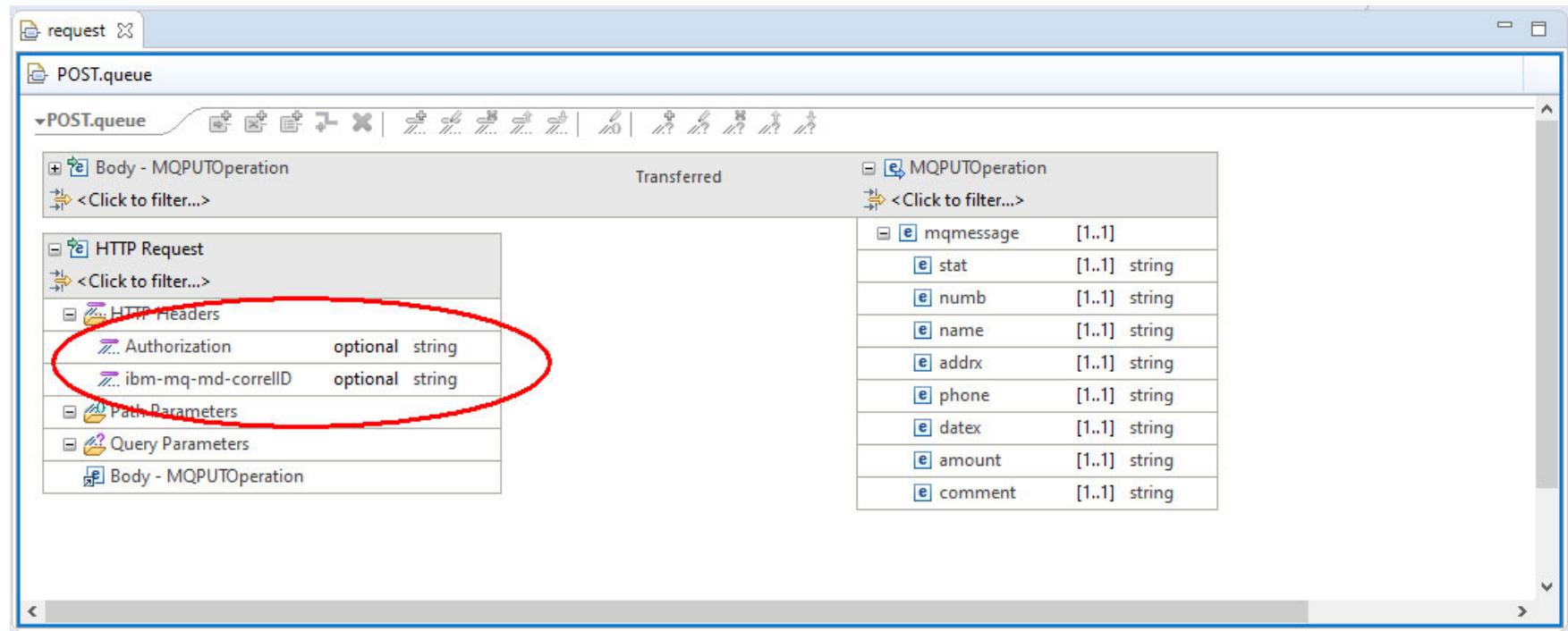
Allows the API Developer to remove fields from the response to simplify the API



API toolkit – API Editor

API mapping: Adding HTTP header properties

Allows the API Developer to remove fields to the request.

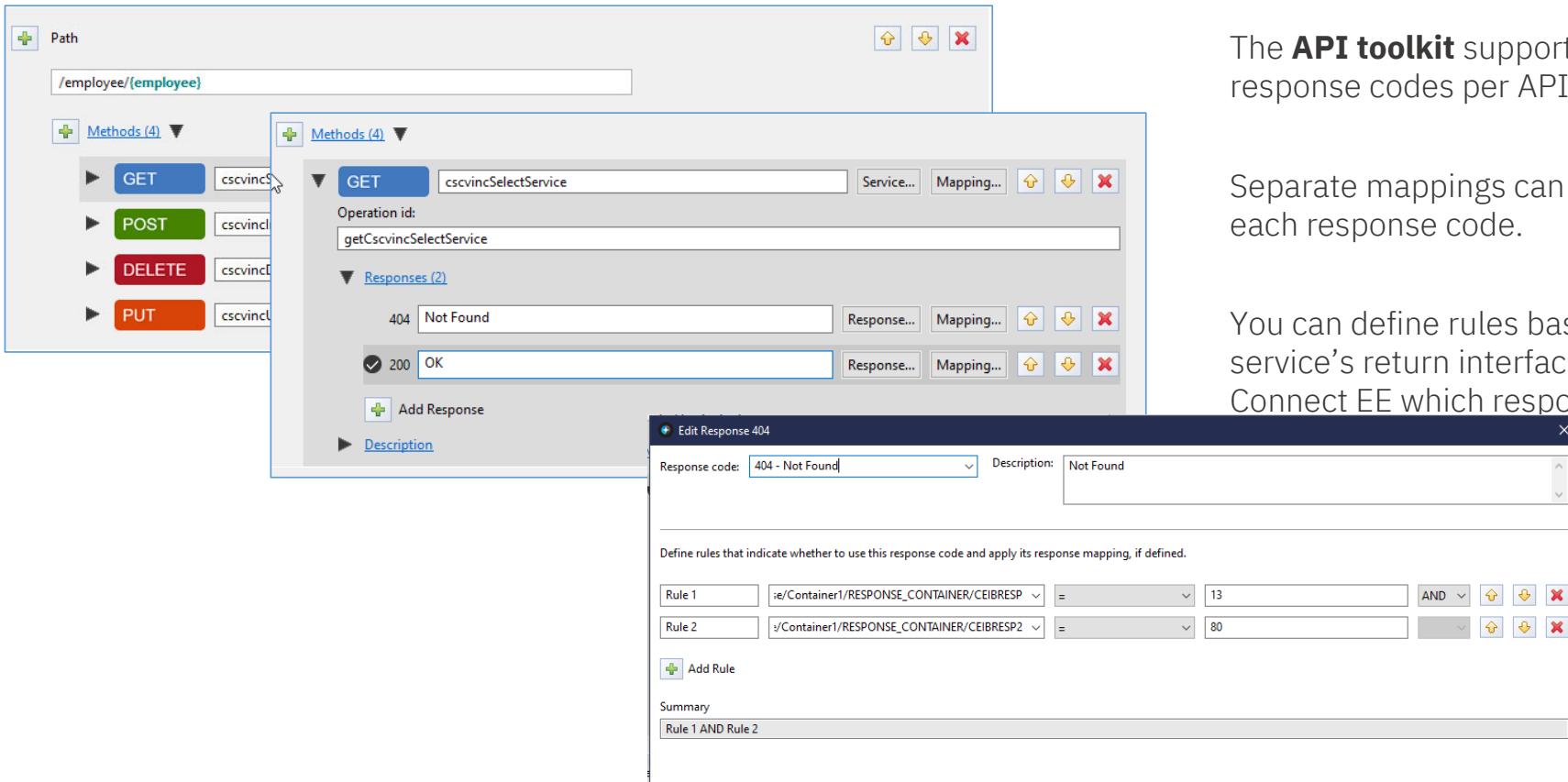


The screenshot shows the API toolkit - API Editor interface. On the left, under the 'POST.queue' tab, there is a tree view of the API structure. A red oval highlights the 'HTTP Headers' section, which contains two fields: 'Authorization' (optional string) and 'ibm-mq-md-correlID' (optional string). On the right, there is a detailed view of the 'MQPUTOperation' message structure, listing fields such as mqmessage, stat, numb, name, addrx, phone, datex, amount, and comment, each with its data type and cardinality.

Field	Type	Cardinality
mqmessage	[1..1]	
stat	[1..1]	string
numb	[1..1]	string
name	[1..1]	string
addrx	[1..1]	string
phone	[1..1]	string
datex	[1..1]	string
amount	[1..1]	string
comment	[1..1]	string

API toolkit

API definition with multiple response codes



The screenshot shows the API toolkit interface for defining an API operation. The path is set to `/employee/{employee}`. The Methods section lists four methods: GET, POST, DELETE, and PUT. The GET method is expanded, showing its configuration details. The Operation id is `getCscvincSelectService`. The Responses section contains two entries: a 404 response with the description "Not Found" and a 200 response with the description "OK". A modal window titled "Edit Response 404" is open, showing the response code "404 - Not Found" and the description "Not Found". Below this, there is a section for defining rules to apply this response code, with two rules defined:

Rule 1	Condition	Value	Operator	Value	Logical Operator	Rule 2	Condition	Value	Operator	Value	
Rule 1	<code>ie/Container1/RESPONSE_CONTAINER/CEIBRESP</code>	<code>=</code>	<code>13</code>	<code>AND</code>		Rule 2	<code>y/Container1/RESPONSE_CONTAINER/CEIBRESP2</code>	<code>=</code>	<code>80</code>		

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

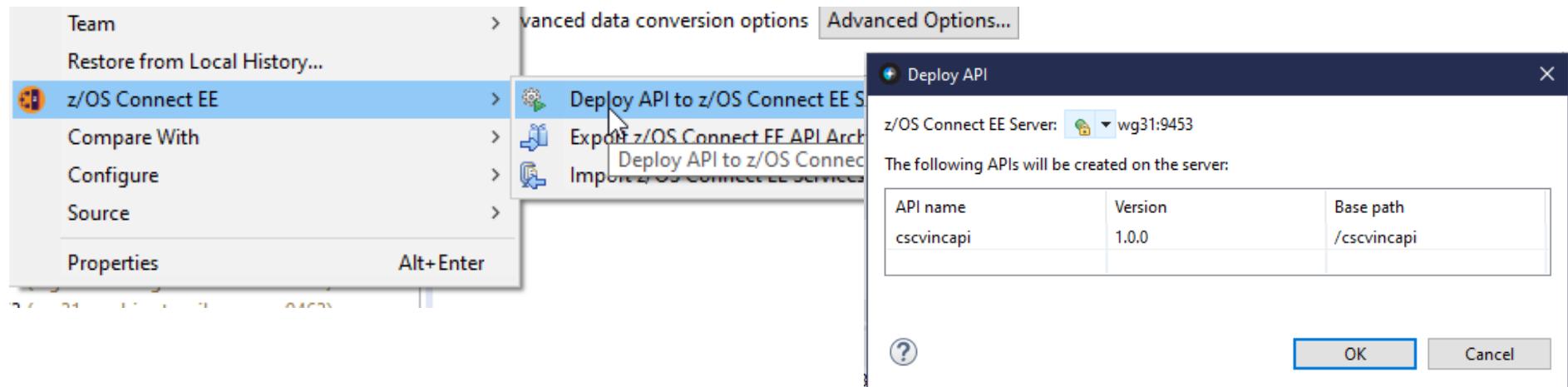
Separate mappings can be defined for each response code.

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:

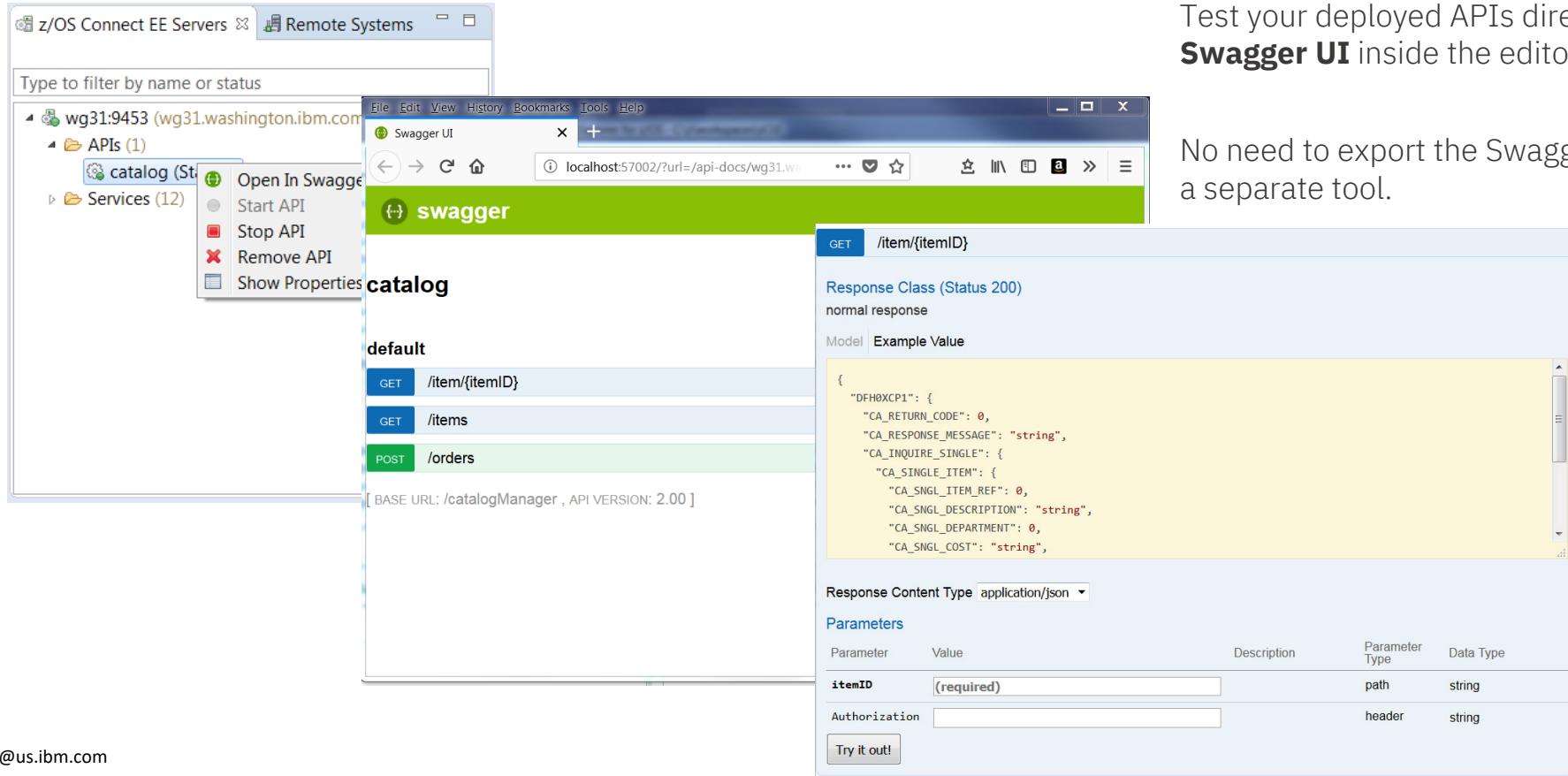


Right-click deploy to server enables developers to quickly deploy, test, and iterate on their APIs.

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



The screenshot shows the z/OS Connect EE interface. On the left, the 'Remote Systems' panel lists a server 'wg31:9453 (wg31.washington.ibm.com)' with one API listed under 'catalog'. A context menu is open over this API entry, showing options: 'Open In Swagger', 'Start API', 'Stop API', 'Remove API', and 'Show Properties'. The main area displays the Swagger UI for the 'catalog' API. The 'catalog' endpoint is selected. The 'default' section shows three operations: 'GET /item/{itemID}', 'GET /items', and 'POST /orders'. The 'POST /orders' operation is highlighted with a green background. Below the operations, it says '[BASE URL: /catalogManager , API VERSION: 2.00]'. The right side of the screen shows the Swagger UI interface. It has tabs for 'Model' and 'Example Value'. The 'Example Value' tab displays 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",  
        "CA_SNGL_QTY": 0  
      }  
    }  
  }  
}
```

Below the schema, there is a 'Response Content Type' dropdown set to 'application/json'. The 'Parameters' section contains two entries: 'itemID' (required, path, string) and 'Authorization' (header, string). At the bottom is a 'Try it out!' button.

Test your deployed APIs directly with **Swagger UI** inside the editor.

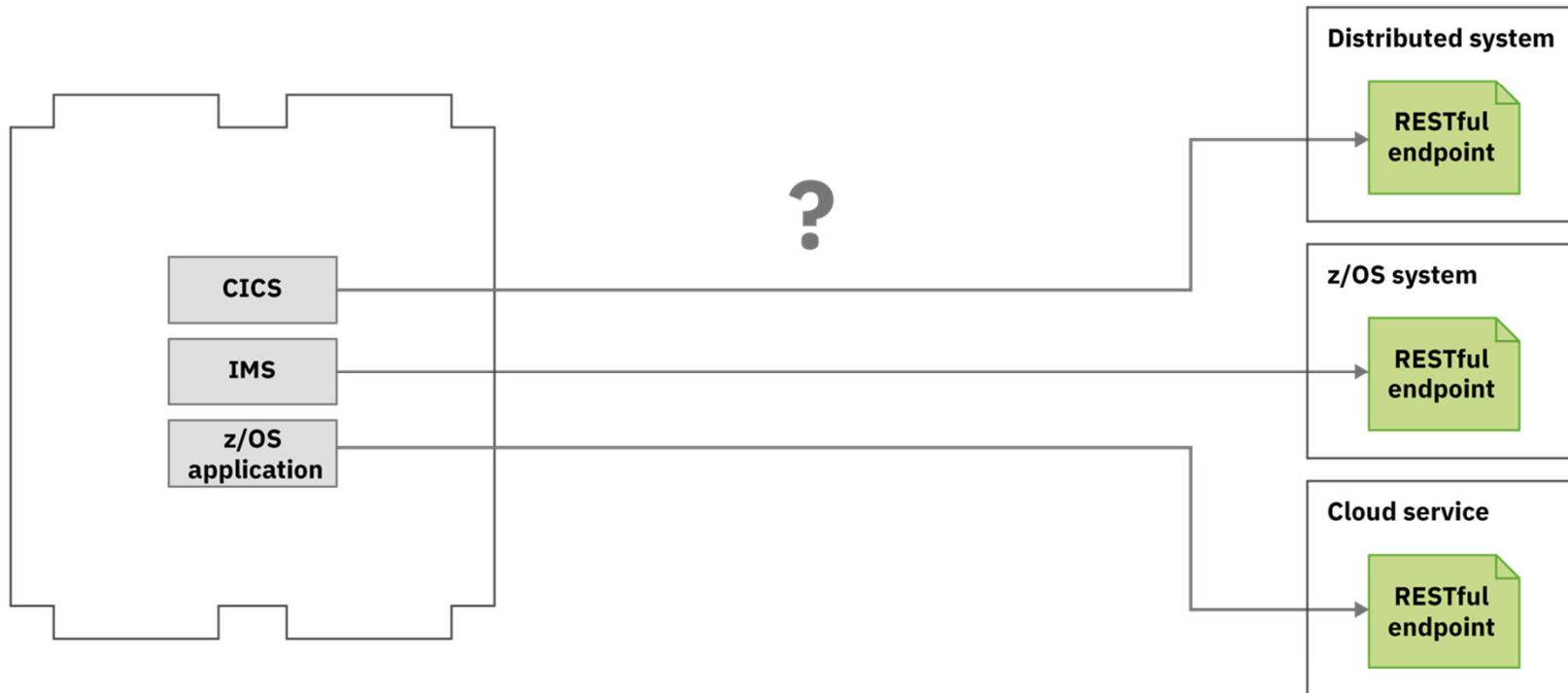
No need to export the Swagger doc to a separate tool.



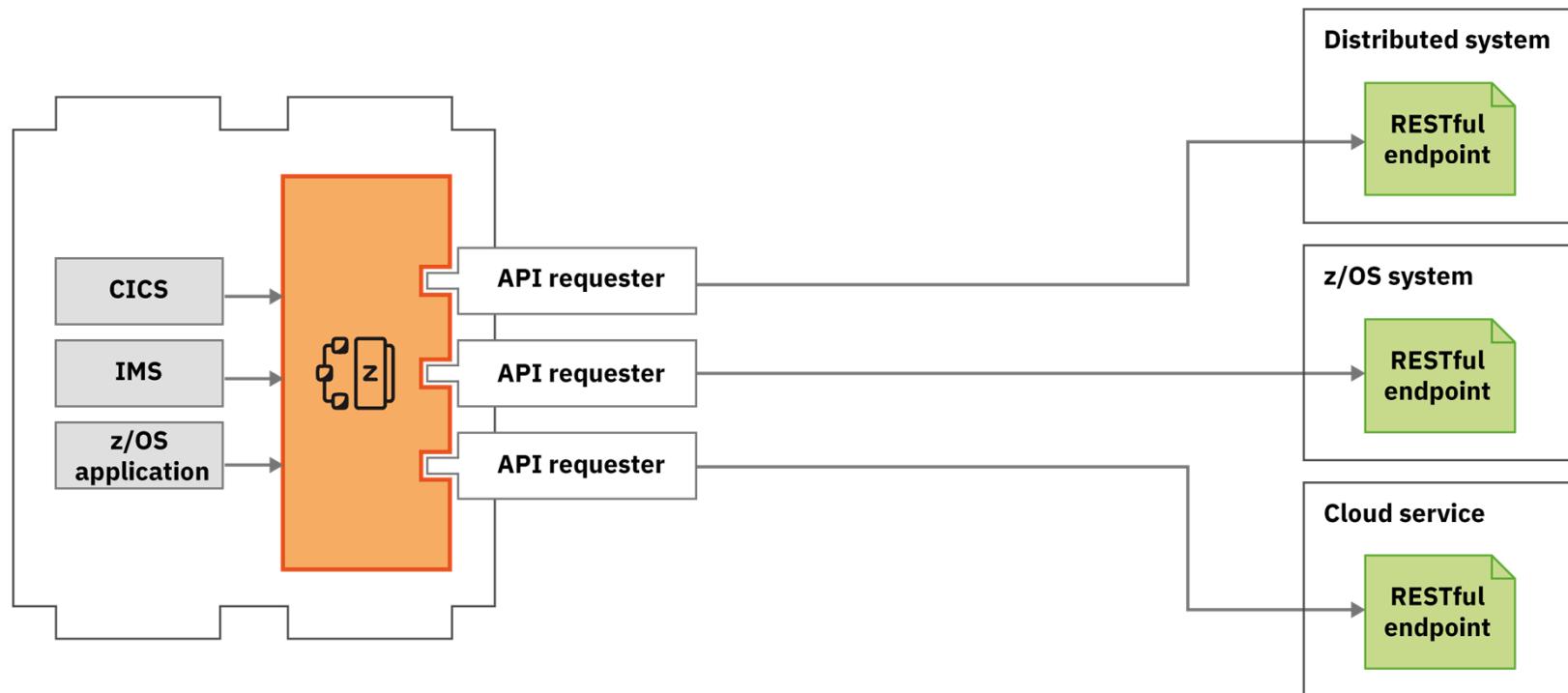
/api_toolkit/apiRequesters

Quick and easy **API mapping**.

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



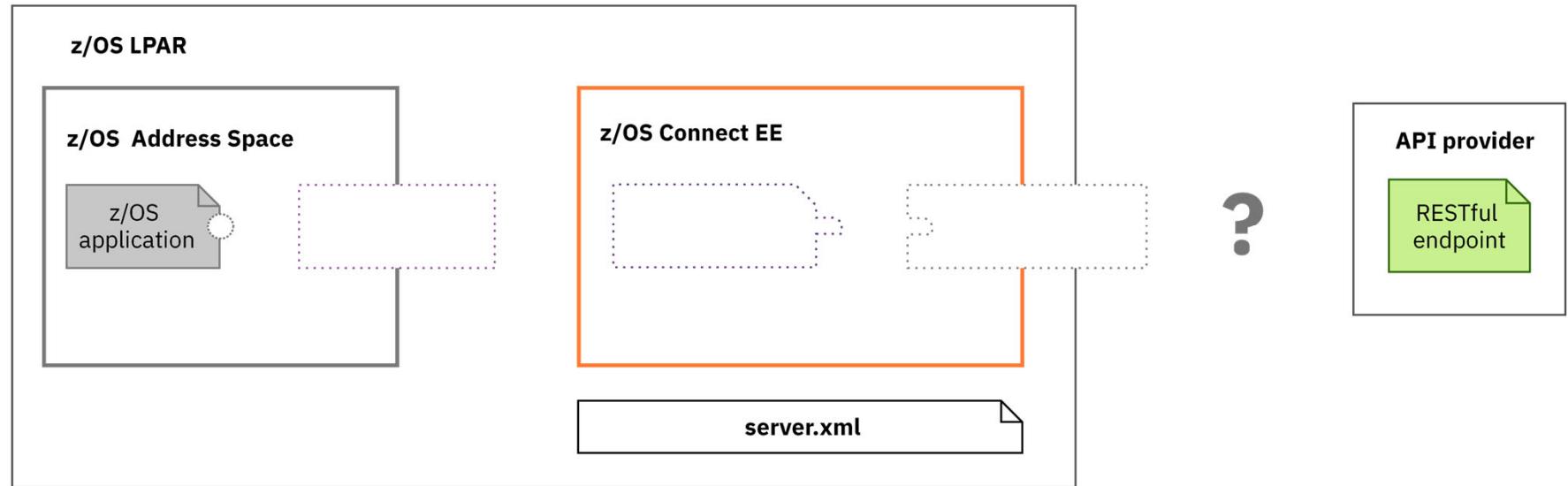
Use API requester to call external APIs from z/OS assets





Steps to calling an external API

Starting point

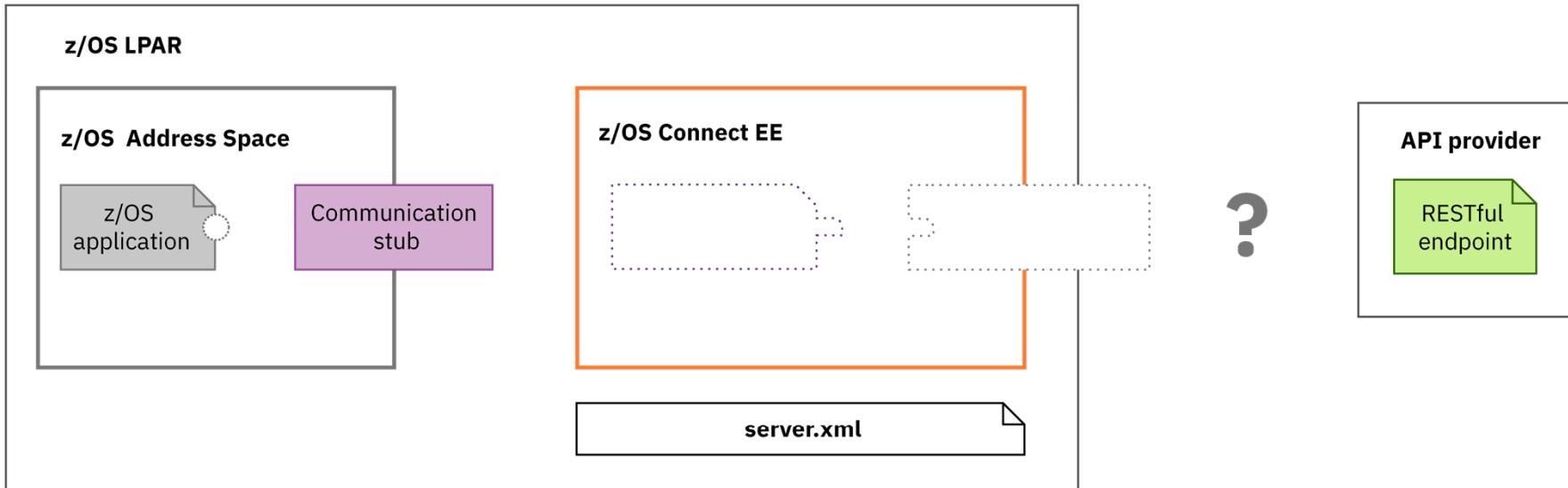




z/OS Connect EE

Steps to calling an external API

Step 1. Configure communication stub



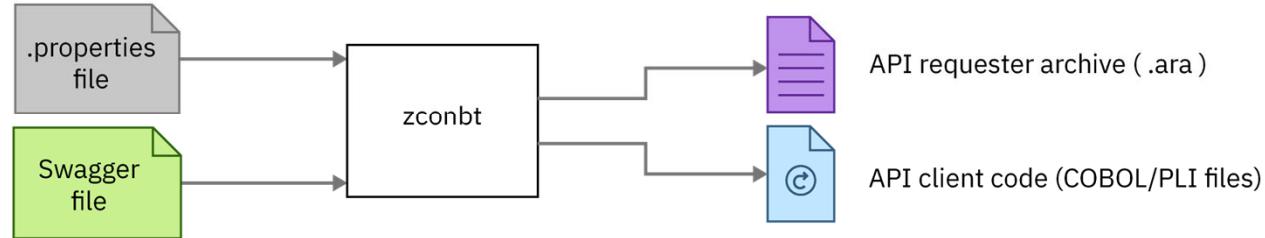
Configure a communication stub.

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

 ibm.biz/zosconnect-configure-comms-stub

Steps to calling an external API

Step 2. Generate API requester archive from Swagger

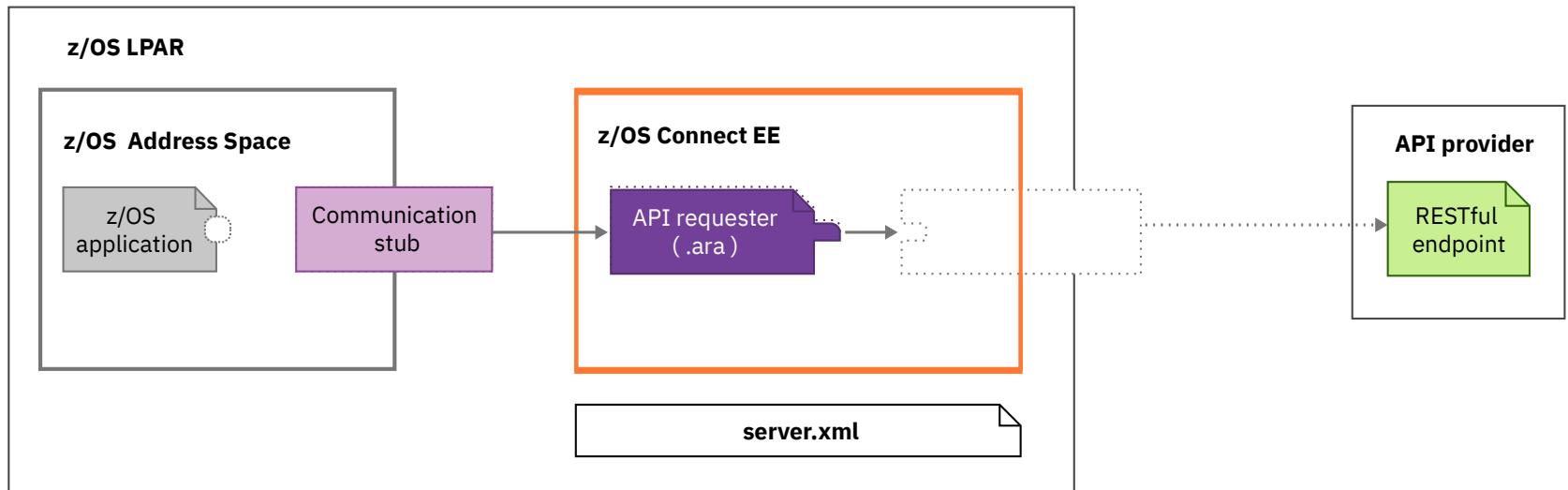


Generate the API requester archive file, and API client code.

 ibm.biz/zosconnect-generate-ara

Steps to calling an external API

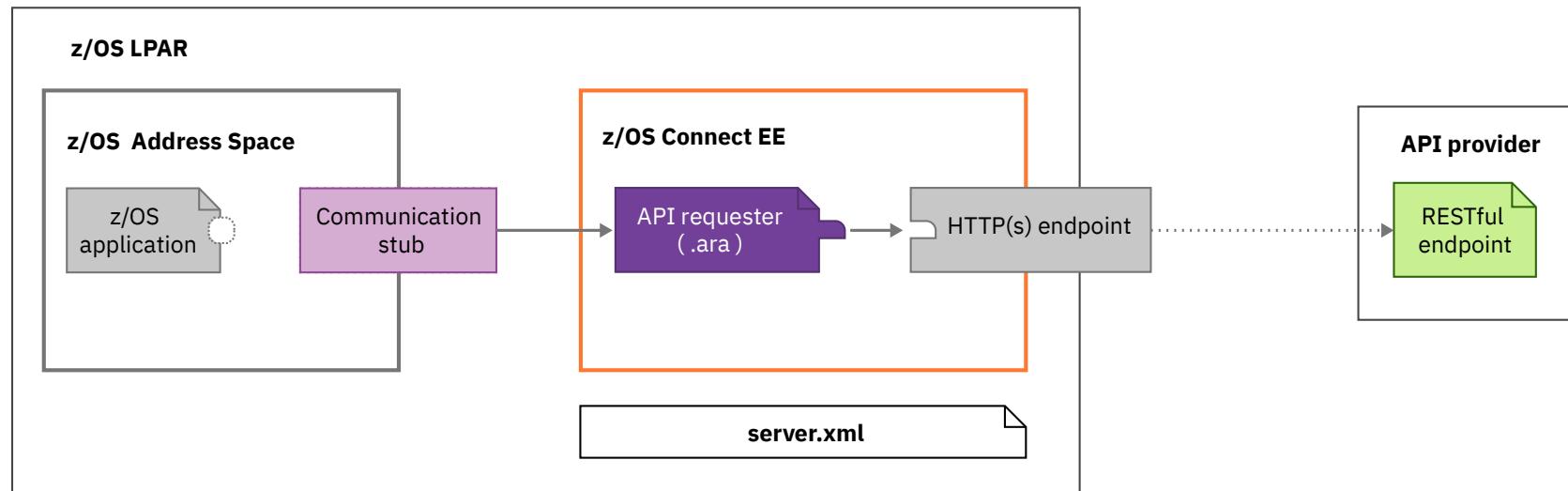
Step 3. Deploy API requester (.ara) archive



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

Steps to calling an external API

Step 4. Configure HTTP(S) endpoint

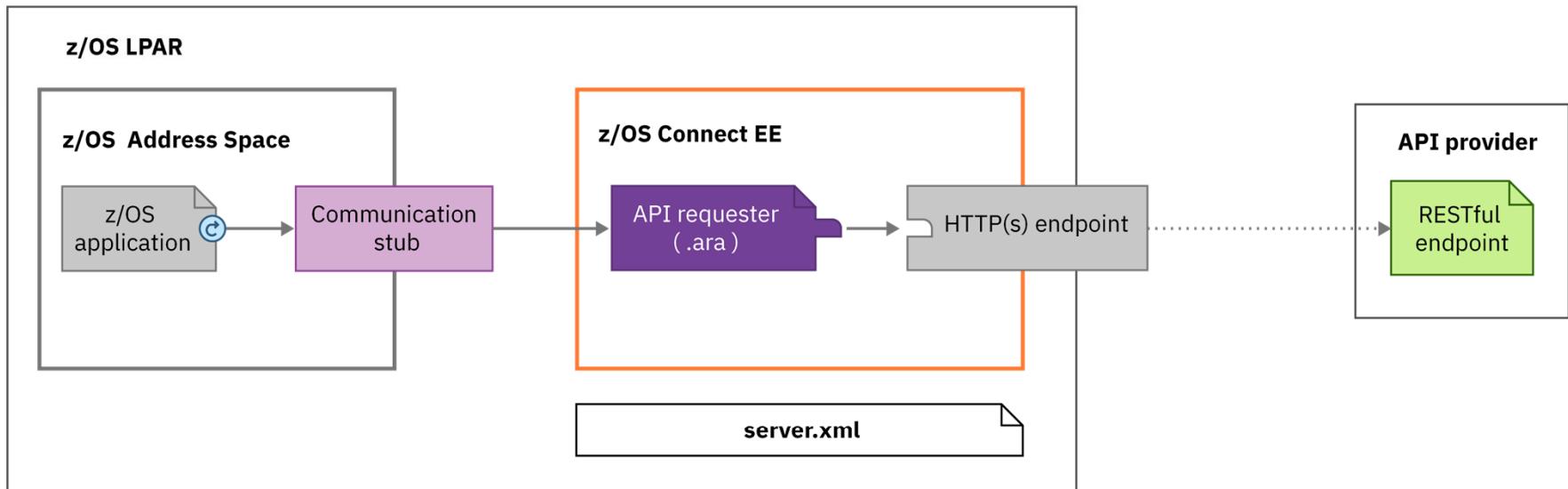


Configure the connection between z/OS Connect EE and the external API.

 ibm.biz/zosconnect-configure-endpoint-connection

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

Steps to calling an external API

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

The screenshot shows three windows in IBM Rational Application Developer:

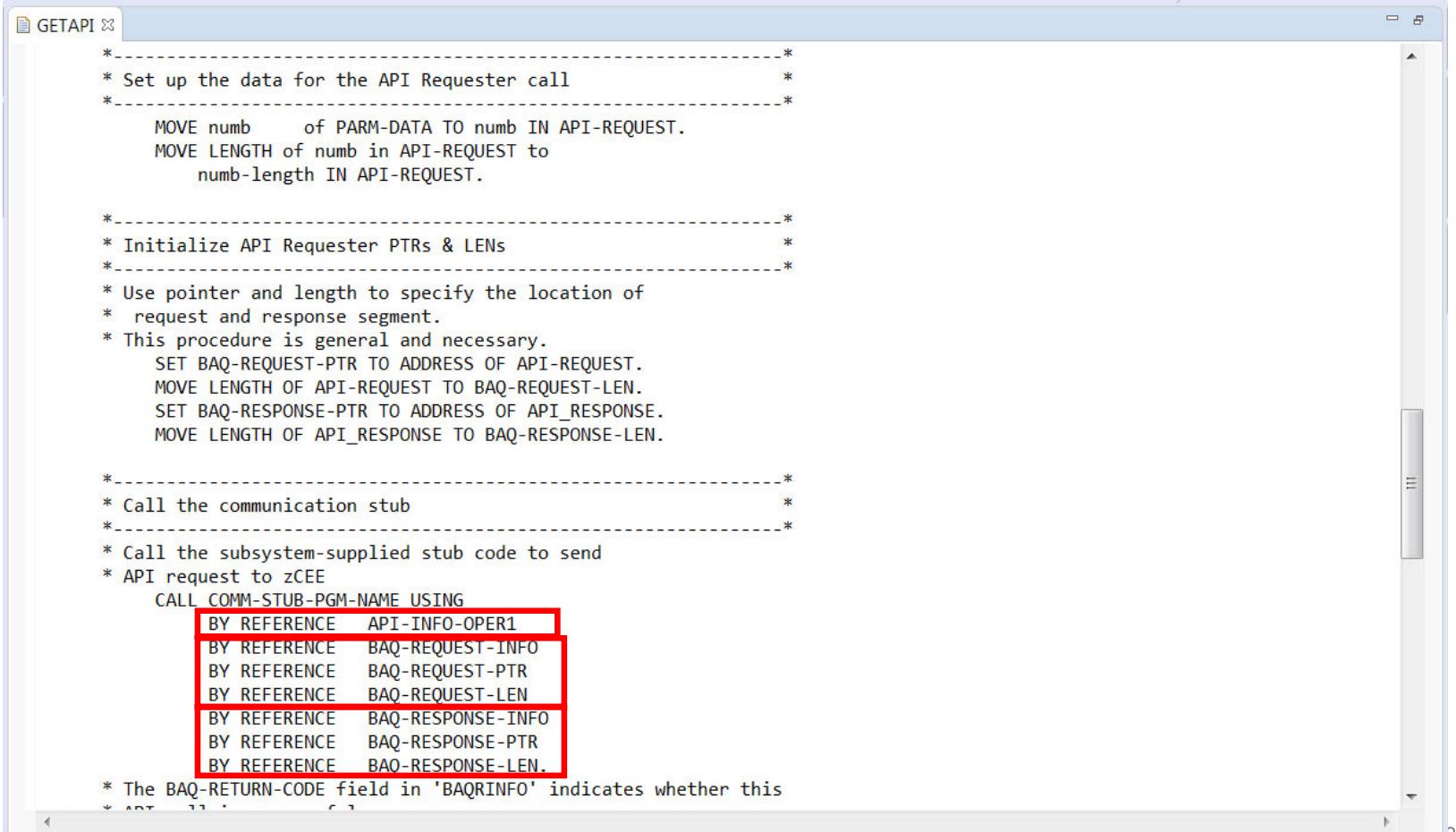
- GETAPI**: A code editor window showing an **ERROR MESSAGE STRUCTURE**. It includes fields for **EM-ORIGIN**, **EM-CODE**, and **EM-DETAIL**, and a note about a required copybook: *** Copy API Requester required copybook COPY BAQRINFO.**
- apis.xml**: An XML configuration file for the API requester. It defines a **server** with a **zosconnect_endpointConnection** block. The host is set to **http://wg31.washington.ibm.com** and the port to **9120**. A basic auth reference **myBasicAuth** is used with user **Fred** and password **fredpwd**. The connection timeout is **10s** and receive timeout is **20s**. A red oval highlights this section.
- CSC02I01**: A code editor window showing a structure definition. It includes fields for **BAQ-APINAME** (value: **'cscvinc_1.0.0'**), **BAQ-APINAME-LEN** (value: **13**), **BAQ-APIPATH** (value: **'/cscvinc/employee/{numb}'**), **BAQ-APIPATH-LEN** (value: **24**), **BAQ-APIMETHOD** (value: **'GET'**), and **BAQ-APIMETHOD-LEN** (value: **3**). A red arrow points from the **CSC02I01** window to the **COPY CSC02I01.** line in the **GETAPI** window.

On the right side of the **apis.xml** window, there is a preview pane showing configuration properties:

- apiDescriptionFile**: **./cscvinc.swagger**
- dataStructuresLocation**: **./syslib**
- apiInfoFileLocation**: **./syslib**
- logFileDirectory**: **./logs**
- language**: **COBOL**
- connectionRef**: **=cscvincAPI** (highlighted by a red oval)
- requesterPrefix**: **csc** (highlighted by a red oval)

Steps to calling an external API

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



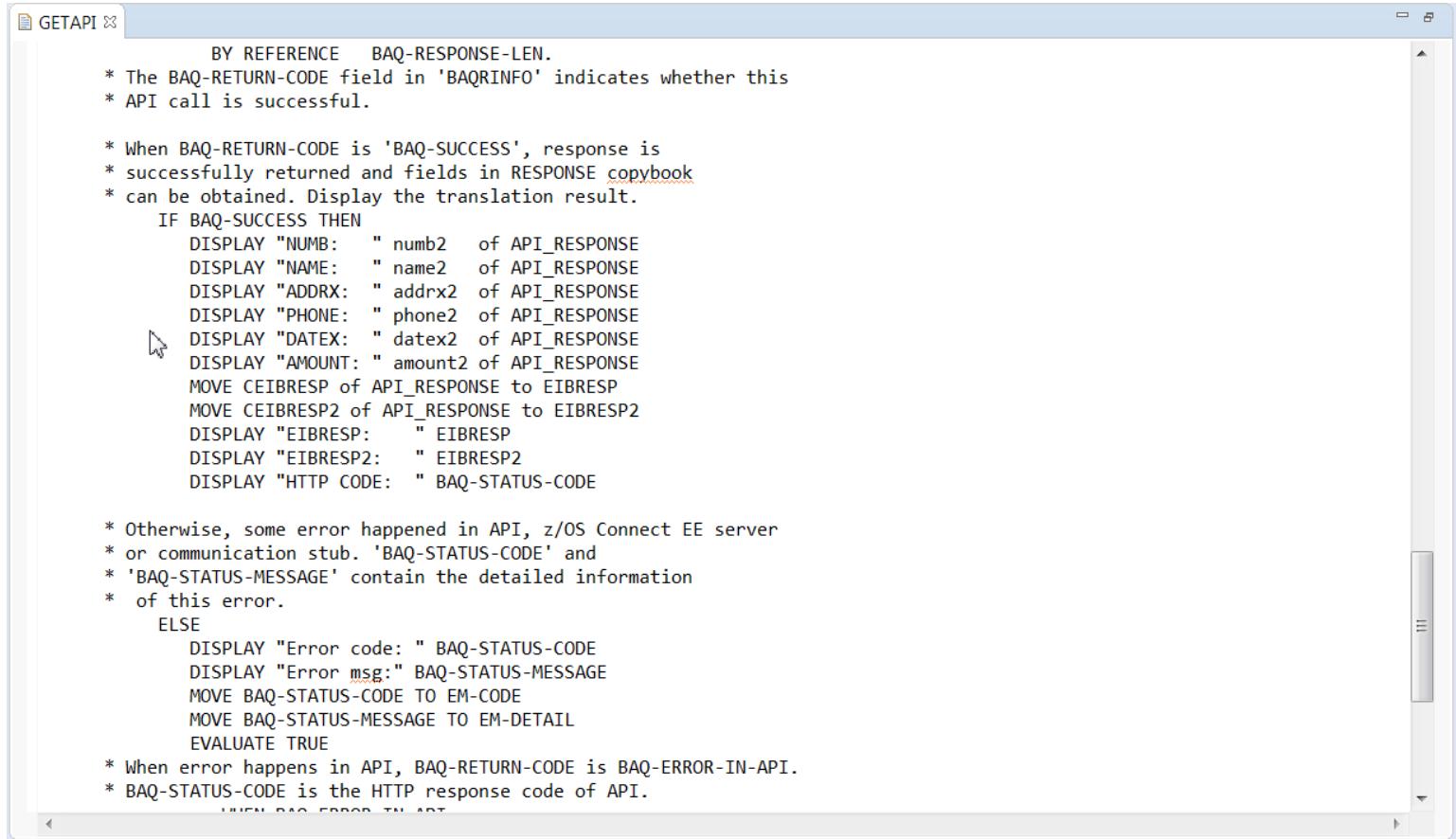
The screenshot shows the GETAPI editor window with assembly code. The code is organized into several sections with comments:

- *-----
* Set up the data for the API Requester call
*-----
- MOVE numb of PARM-DATA TO numb IN API-REQUEST.
MOVE LENGTH of numb in API-REQUEST to
numb-length IN API-REQUEST.
- *-----
* Initialize API Requester PTRs & LENs
*-----
- * Use pointer and length to specify the location of
* request and response segment.
* This procedure is general and necessary.
SET BAQ-REQUEST-PTR TO ADDRESS OF API-REQUEST.
MOVE LENGTH OF API-REQUEST TO BAQ-REQUEST-LEN.
SET BAQ-RESPONSE-PTR TO ADDRESS OF API_RESPONSE.
MOVE LENGTH OF API_RESPONSE TO BAQ-RESPONSE-LEN.
- *-----
* Call the communication stub
*-----
- * Call the subsystem-supplied stub code to send
* API request to zCEE
CALL COMM-STUB-PGM-NAME USING
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.
- * The BAQ-RETURN-CODE field in 'BAQRINFO' indicates whether this

A red rectangular box highlights the parameters passed to the COMM-STUB-PGM-NAME call, specifically the BY REFERENCE lines for API-INFO-OPER1, BAQ-REQUEST-INFO, BAQ-REQUEST-PTR, BAQ-REQUEST-LEN, BAQ-RESPONSE-INFO, BAQ-RESPONSE-PTR, and BAQ-RESPONSE-LEN.

Steps to calling an external API

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

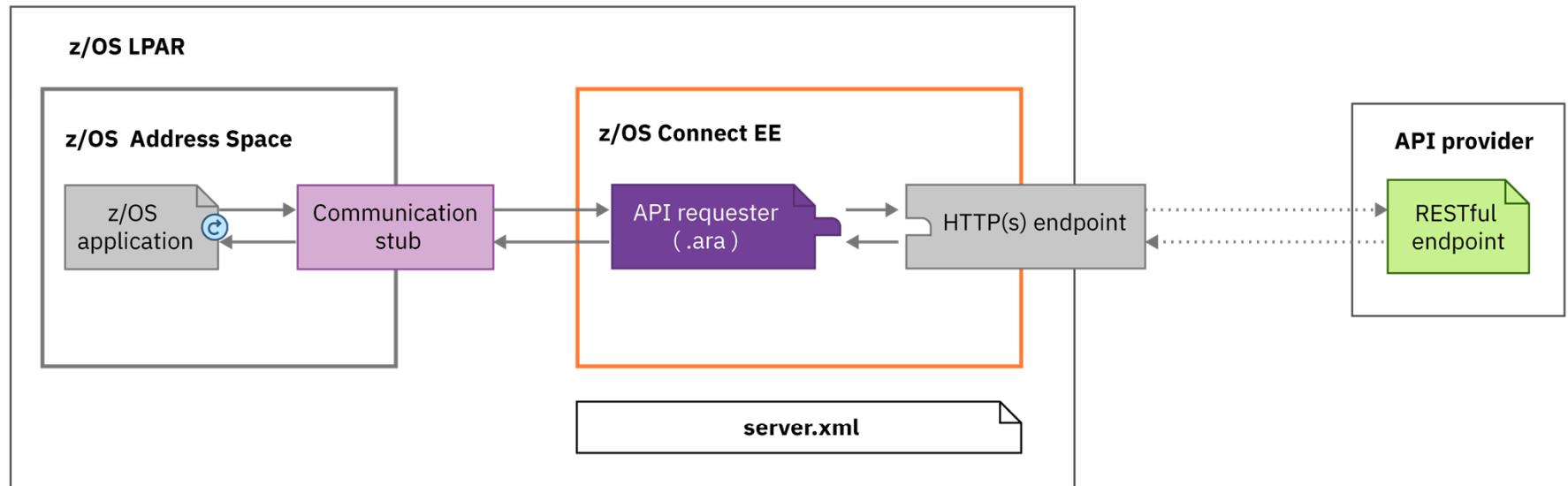


The screenshot shows a window titled "GETAPI" containing AS/400 JCL code. The code handles the response from an API call, specifically dealing with BAQ-RESPONSE-LEN and BAQ-RETURN-CODE fields. It includes logic to display various response fields if the call was successful ("BAQ-SUCCESS") or to handle errors by displaying error codes and messages.

```
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
```

Steps to calling an external API

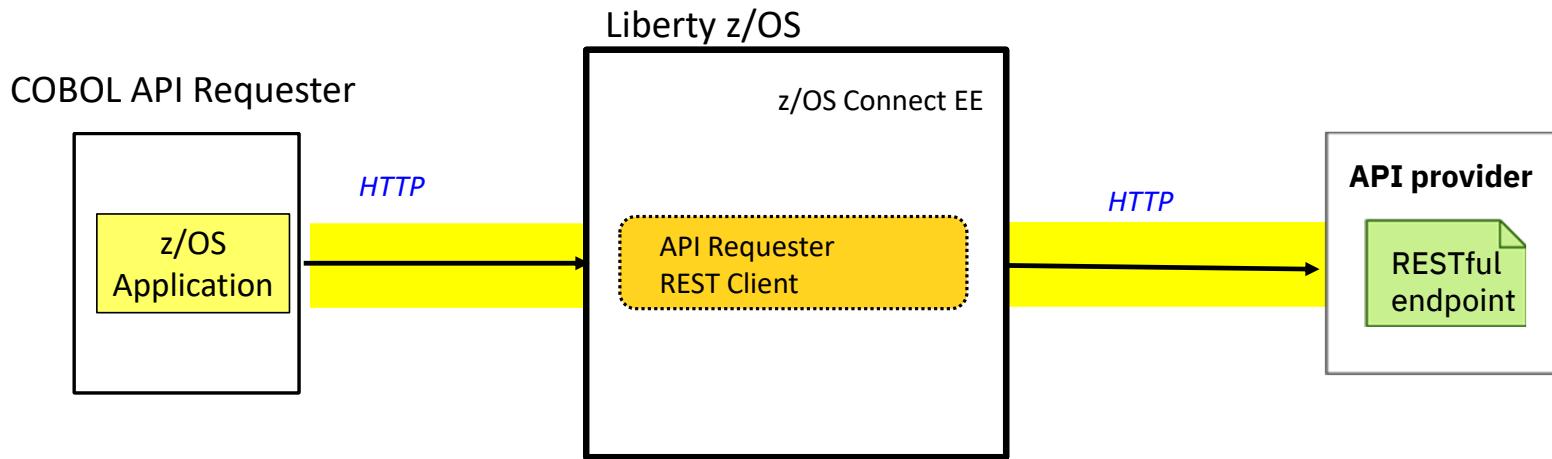
Done





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:

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

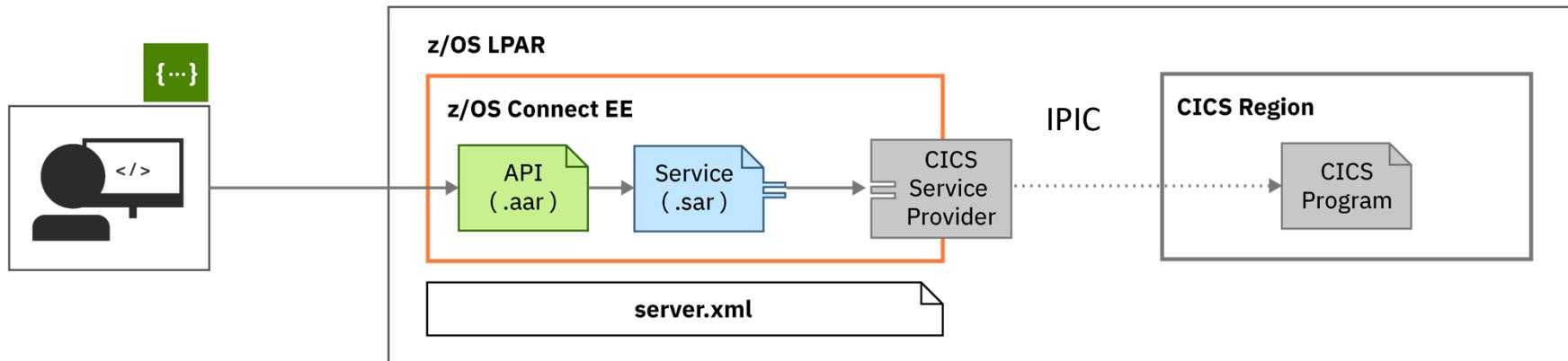


/common_scenarios

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

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
```

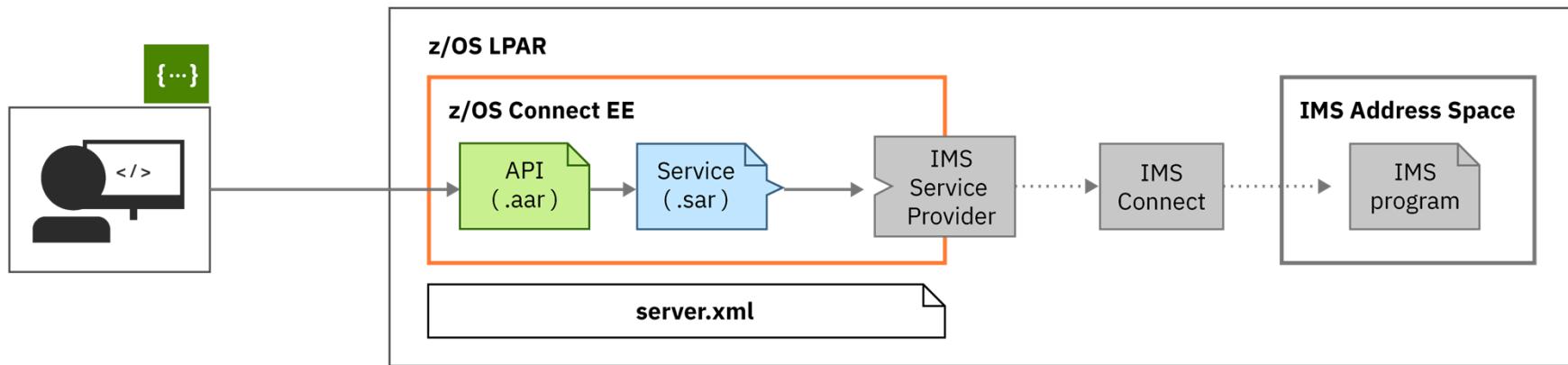
Features are functional building blocks. When configured here, that function becomes available to the Liberty server

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

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>

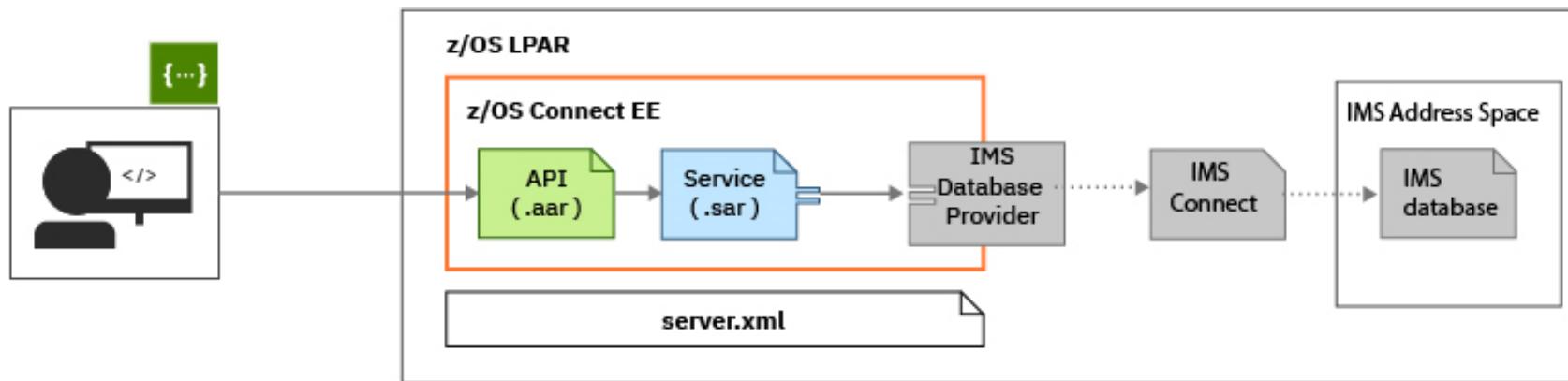
<authData id="Connection1_Auth" password="encryptedPassword1" user="userName1"/>
</server>
```

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.

 ibm.biz/zosconnect-scenarios

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

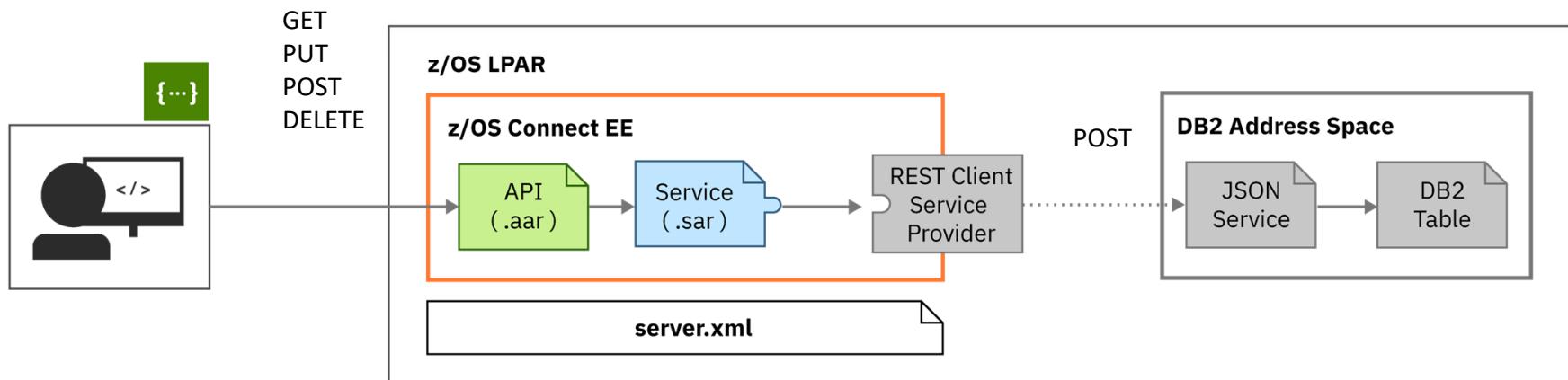
```
HWS= (ID=IMS14HWS,XIBAREA=100,RACF=N,RRS=N)
TCPIP= (HOSTNAME=TCPIP,PORTID= (4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE= (GROUP=OTMAGRP, ID=IVP1, MEMBER=HWSMEM, TMEMBER=OTMAMEM)
IMSPLEX= (MEMBER=IMS14HWS, TMEMBER=PLEX1)
ODACCESS= (ODBMAUTOCONN=Y,
DRDAPORT= (ID=5555,PORTTMOT=6000), ODBMTMOT=6000)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

mitc

© 2018, 2021 IBM Corporation

Connections to Db2

Topology



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

A Db2 REST Service must be configured in DB2.

 ibm.biz/zosconnect-db2-rest-services

The server.xml File (Db2)



z/OS Connect EE

The server.xml file is the key configuration file:

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection reference: db2conn

Definition Configuration

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

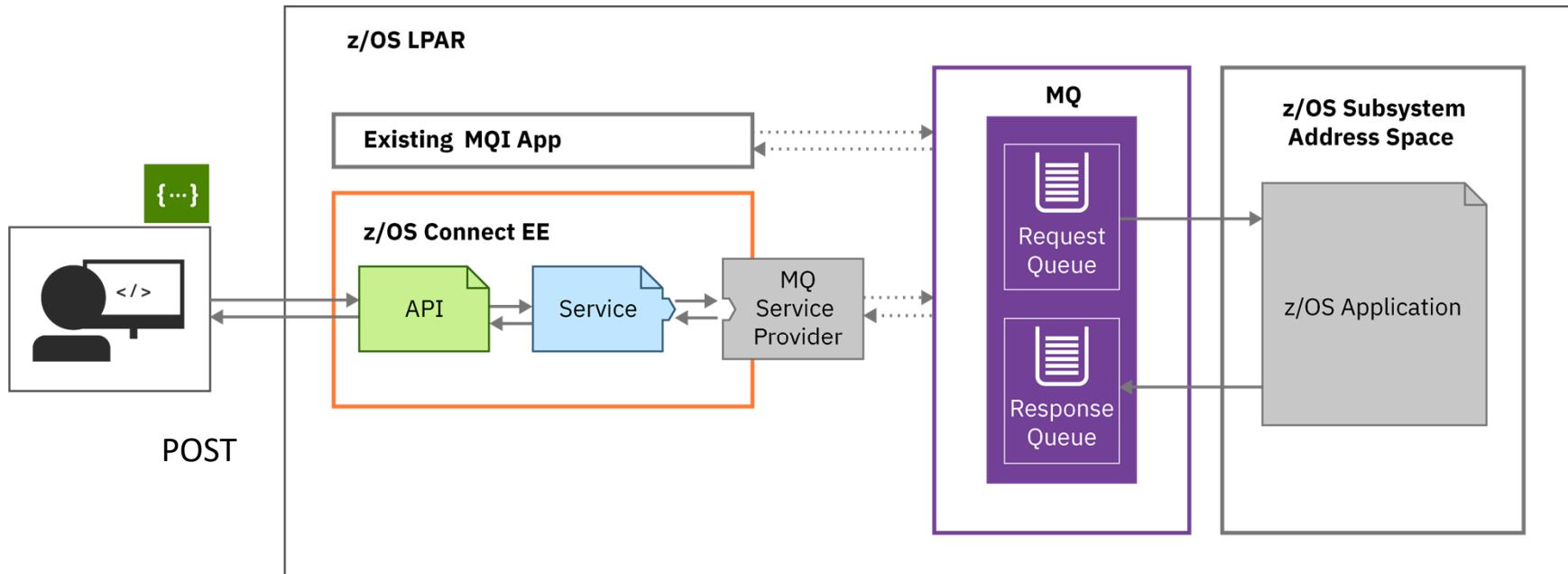
db2pass.xml

Design Source

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

Connections to MQ

Topology (Two-way service example)

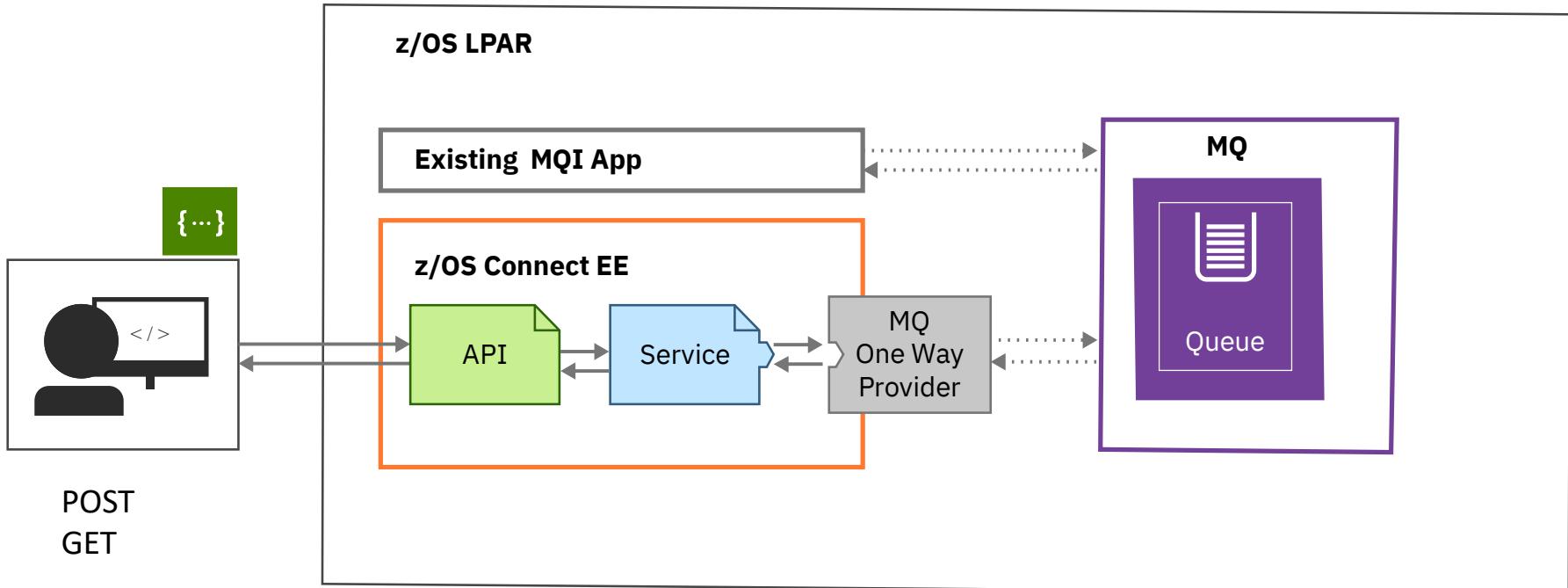


You can also configure one-way services.

 ibm.biz/zosconnect-mq-service-provider

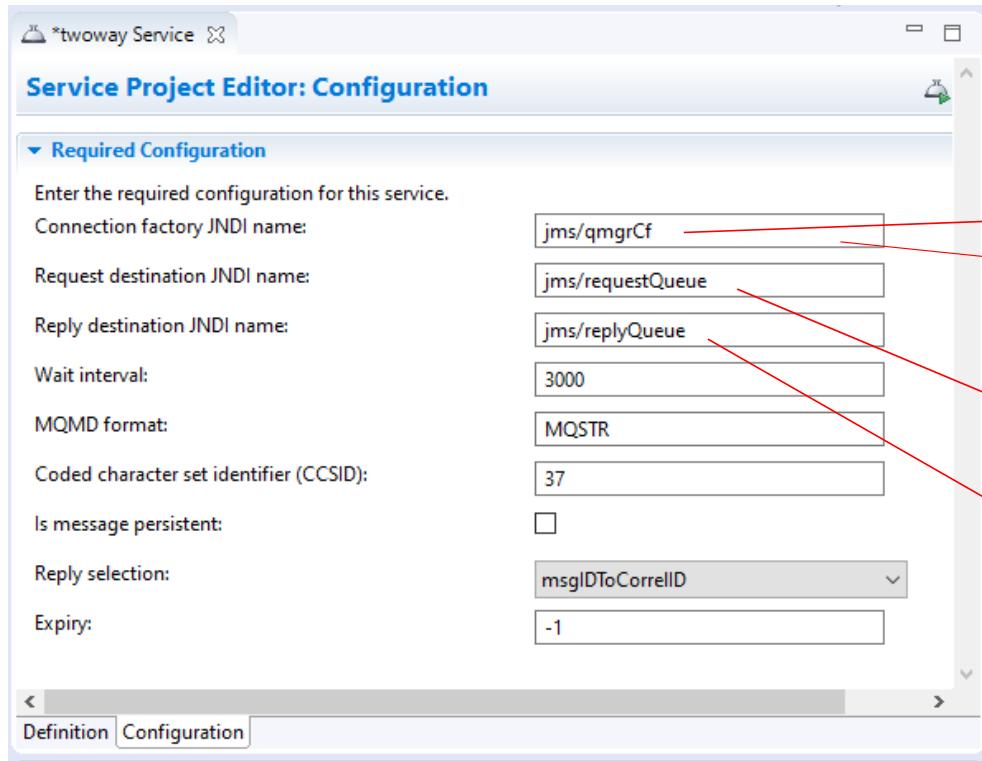
Connections to MQ

Topology (One-way service example)



 ibm.biz/zosconnect-mq-service-provider

The server.xml File (MQ)



```
mq.xml
Read only Close
Design Source
2
3 <featureManager>
4   <feature>zosconnect:mqService-1.0</feature>
5 </featureManager>
6
7 <variable name="wmqJmsClient.rar.location"
8   value="/usr/lpp/mqm/V9R1M1/java/lib/jca/wmq.jmsra.rar"/>
9 <wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>
10
11 <connectionManager id="ConMgr1" maxPoolSize="5"/>
12
13 <jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
14   connectionManagerRef="ConMgr1">
15   <properties.wmqJMS transportType="BINDINGS"
16     queueManager="QM21" />
17 </jmsConnectionFactory>
18
19 <jmsConnectionFactory id="qmgrCf2" jndiName="jms/qmgrCf2"
20   connectionManagerRef="ConMgr1">
21   <properties.wmqJMS transportType="CLIENT"
22     queueManager="ZMQ1"
23     channel="LIBERTY.DEF.SVRCONN"
24     hostName="wg31.washington.ibm.com"
25     port="1422" />
26 </jmsConnectionFactory>
27
28 <jmsQueue id="q1" jndiName="jms/default">
29   <properties.wmqJMS
30     baseQueueName="ZCONN2.DEFAULT.MQZCEE.QUEUE"
31     CCSID="37"/>
32 </jmsQueue>
33
34 <jmsQueue id="requestQueue" jndiName="jms/request">
35   <properties.wmqJMS
36     baseQueueName="ZCONN2.TRIGGER.REQUEST"
37     targetClient="MQ"
38     CCSID="37"/>
39 </jmsQueue>
40
41 <jmsQueue id="replyQueue" jndiName="jms/replyQueue">
42   <properties.wmqJMS
43     baseQueueName="ZCONN2.TRIGGER.RESPONSE"
44     targetClient="MQ"
45     CCSID="37"/>
46 </jmsQueue>
47
```

MQ V9.1.1 Added support for
remote queue managers.

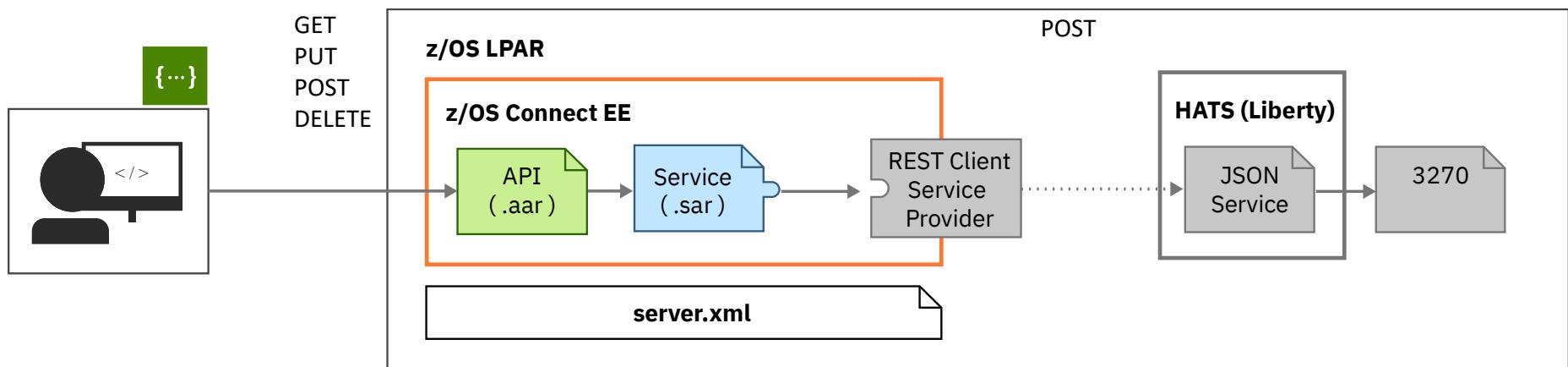
mitchj@us.ibm.com

© 2018, 2021 IBM Corporation

Connection to HATS



Topology

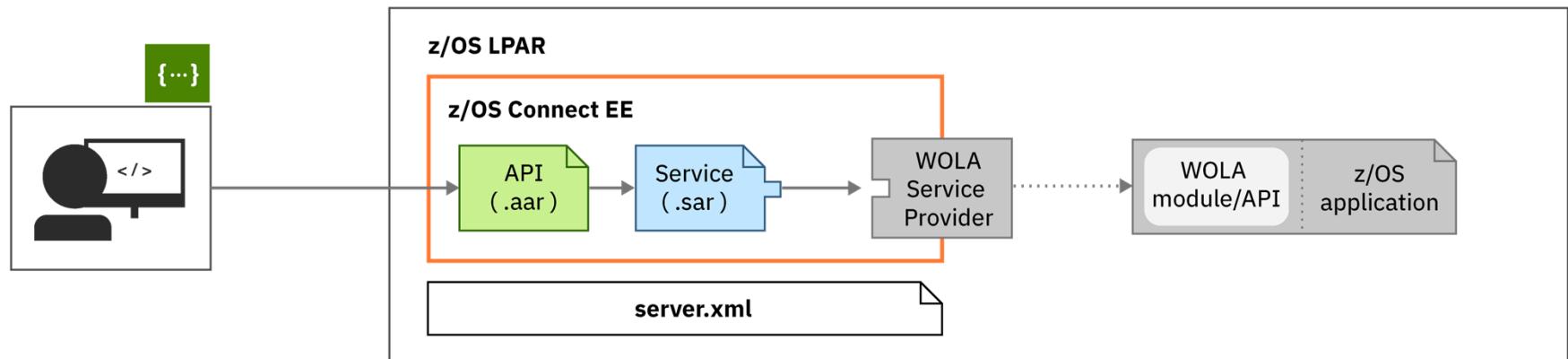


Connection to the HATS REST Service is configured in `server.xml`.

ibm.biz/zosconnect-db2-rest-services

Connections to a MVS batch application

Topology



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

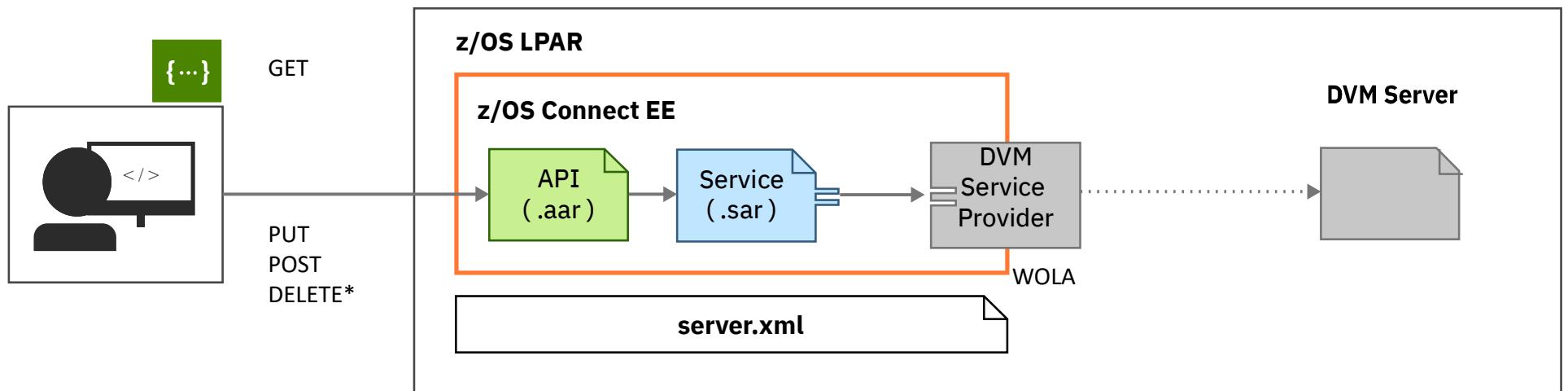
The z/OS application must be WOLA-enabled.

Connections to DVM



z/OS Connect EE

Topology



The DVM service provider uses WOLA

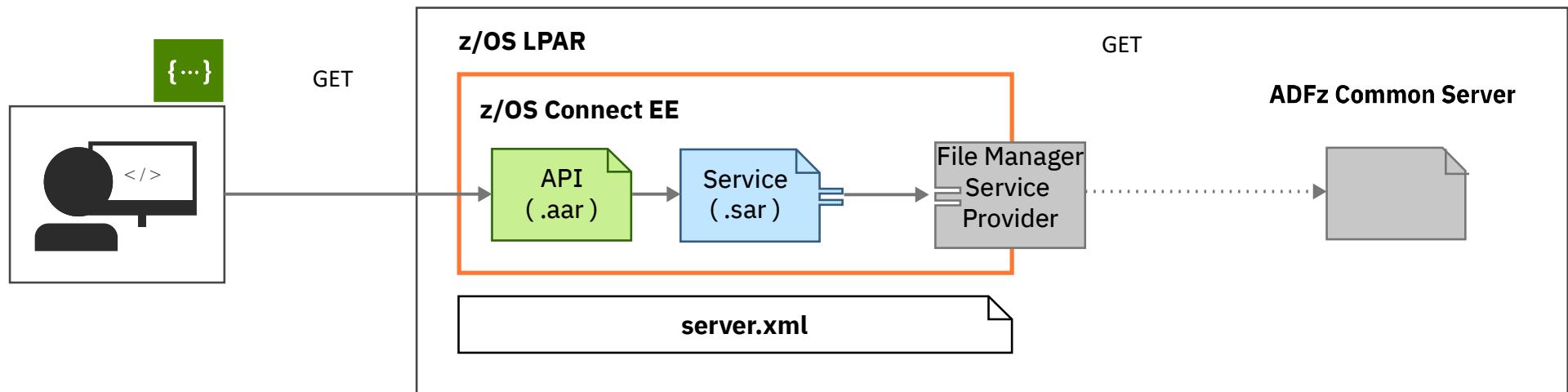
* Requires a resource manager (e.g. RLS, VSAMCICS, etc.)

 ibm.biz/zosconnect-db2-rest-services

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 .



/miscellaneousTopics

performance, high availability, Liberty

A Tour of Server Configuration Directories and Files



z/OS Connect EE

A z/OS Connect EE V3.0 server configuration structure looks like this:

```
/var/zosconnect
  /servers
    /zceesrv1
      /logs
        messages.log
  /resources
    /zosconnect
      /apis
      /apiRequesters
      /rules
      /services
        server.xml
        server.env
    /workarea
```

The messages.log file is the key output file for messages about Liberty and the processing taking place in the Liberty server.

The /zosconnect directory is where we will place the deployed APIs, services, and API requester files

The server.xml file is the key configuration file. It is here that z/OS Connect EE V3.0 definitions go which define the essential backend connectivity.

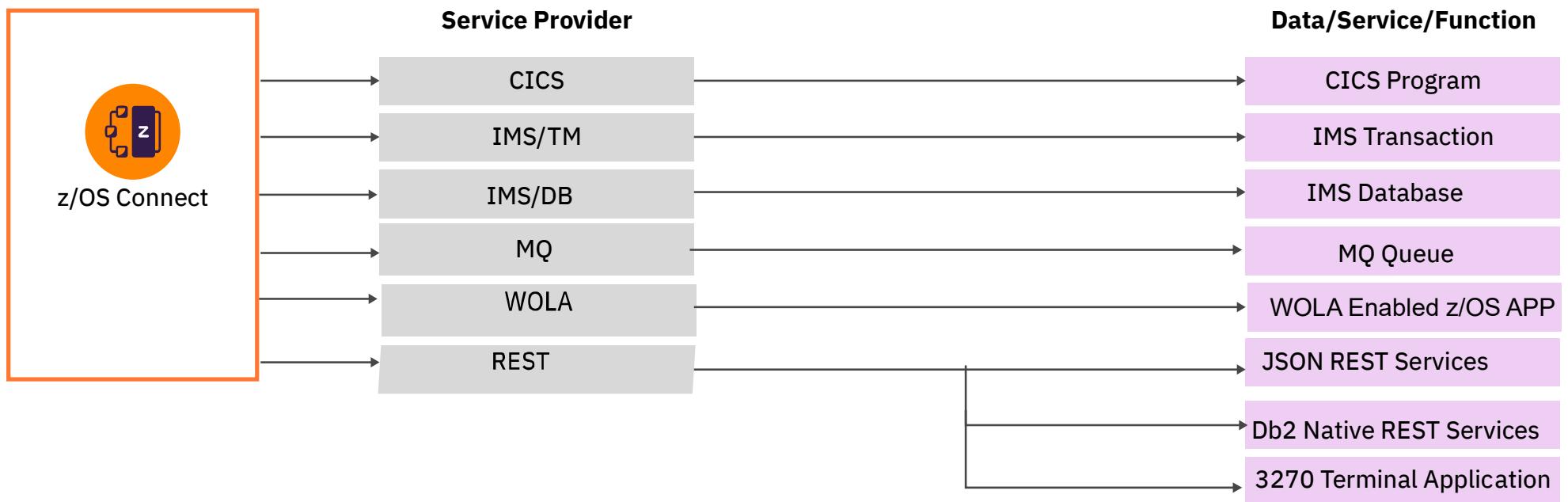
server.xml

```
<server description="zCEE Server">
<include location="${server.config.dir}/includes/safSecurity.xml"/>
<include location="${server.config.dir}/includes/ipicIDProp.xml"/>
<include location="${server.config.dir}/includes/keyringOutboundMutual.xml"/>
<include location="${server.config.dir}/includes/groupAccess.xml"/>
<include location="${server.config.dir}/includes/shared.xml"/>
<include location="${server.config.dir}/includes/apiRequesterHTTPS.xml"/>
<include location="${server.config.dir}/includes/imsDatabase.xml"/>
```

-Dcom.ibm.ws.logging.log.directory=/u/johnson/logs

What assets can z/OS Connect EE map to?

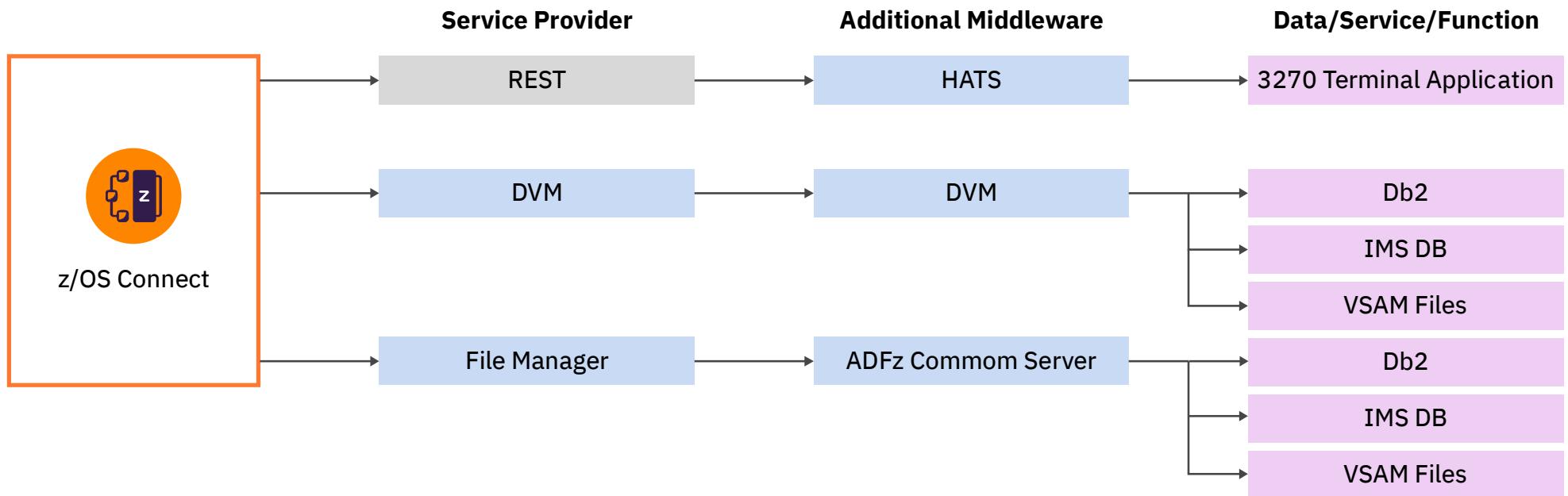
And which service provider could I use?



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

Additional Middleware

Additional value from the ecosystem



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

API Policies

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

CICS attributes

- cicsCcsid
- cicsConnectionRef
- cicsTransId

IMS attributes

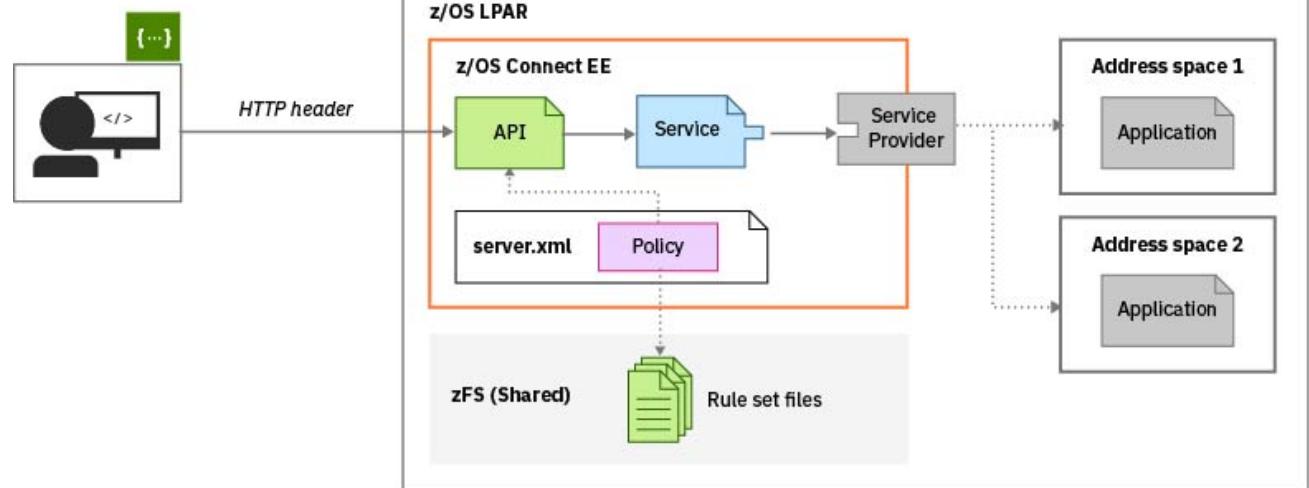
- imsConnectionRef
- imsInteractionRef
- imsInteractionTimeout
- imsLtermOverrideName
- imsTranCode
- imsTranExpiration

Db2 attributes

- db2ConnectionRef
- db2CollectionID

MQ attributes

- mqConnectionFactory
- mqDestination
- mqReplyDestination



A sample API Policies for CICS



```
<ruleset name="CICS rules">
  <rule name="csmi-rule">
    <conditions>
      <header name="cicsMirror" value="CSMI,MIJO"/>1
    </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>
```

GET.employee.{numb}

GET.employee.{numb}

Body - cscvincServiceOperation

HTTP Request

HTTP Headers

cicsMirror optional string

cicsConnection optional string

Path Parameters

{numb} Required string

Query Parameters

Body - cscvincServiceOperation

Curl

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

¹Transaction 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"/>
```

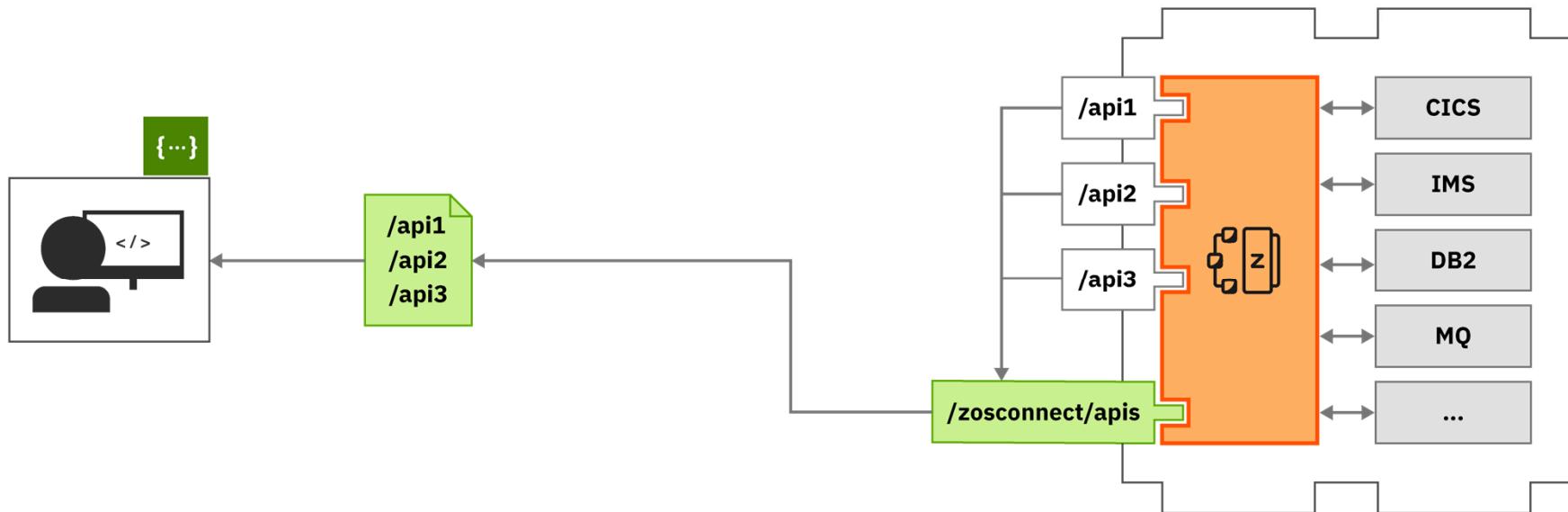
MVS Console

```
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.
```

API Documentation



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



RESTful Administrative Interface for Services

The administration interface for services is available in paths under /zosConnect/services.

Most administration tasks are supported by the RESTful administration interface

Method	Administrative Task
GET	Get details of a service
	Get the status of a service
	Get the request schema of a service
	Get the response schema of a service
POST	Deploy a service*
PUT	Update a service
	Change the status of a service
DELETE	Delete a service

POST /zosConnect/services inquireSingle.sar

PUT /zosConnect/services/{serviceName}?status=started|stopped

PUT /zosConnect/services inquireSingle.sar

GET /zosConnect/services

GET /zosConnect/services/{serviceName}

DELETE /zosConnect/services/{serviceName}

*Useful for deploying service archive files, service archives generated by zconbt, e.g. HATS

RESTful Administrative Interface for APIs

The administration interface for services is available in paths under /zosConnect/apis.

Most administration tasks are supported by the RESTful administration interface

Method	Administrative Task
GET	Get a list of APIs
	Get the details of an API
POST	Deploy an API
PUT	Update an API
	Change the status of an API
DELETE	Delete an API

```
POST  /zosConnect/apis CatalogManager.aar
PUT   /zosConnect/apis/{apiName}?status=started|stopped
PUT   /zosConnect/apis CatalogManager.aar
GET   /zosConnect/apis
GET   /zosConnect/apis/{apiName}
DELETE /zosConnect/apis/{apiName}
```

RESTful Administrative Interface for API Requesters

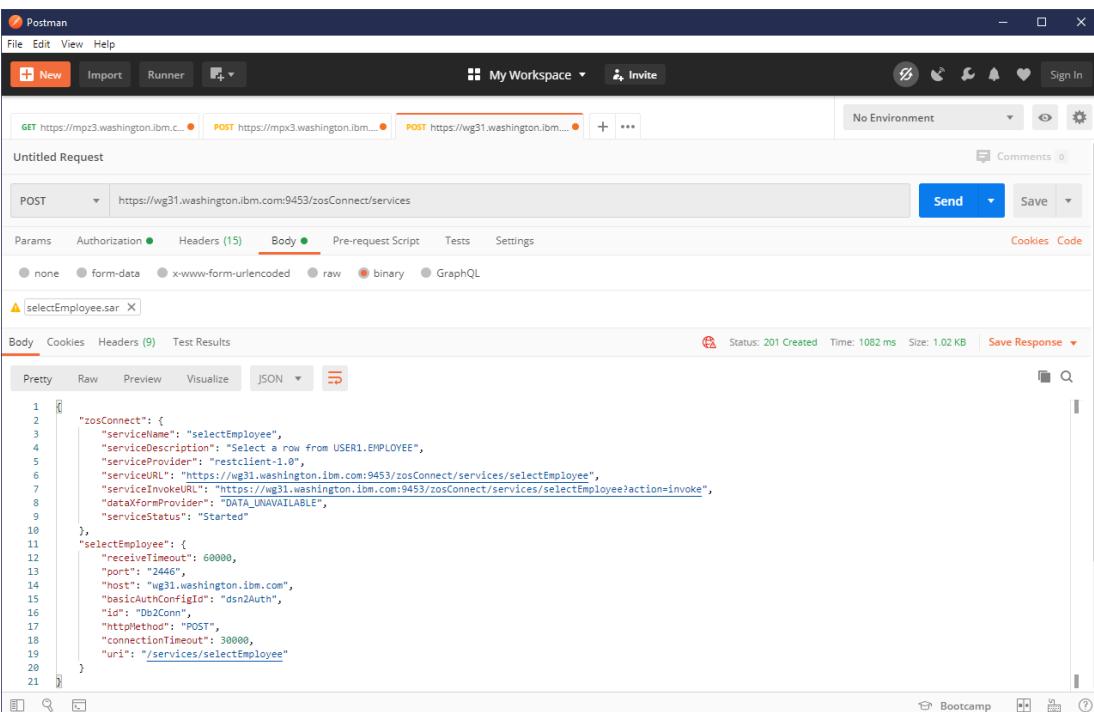
The administration interface for services is available in paths under `/zosConnect/apisRequesters`.
Most administration tasks are supported by the RESTful administration interface

Method	Administrative Task
GET	Get a list of API Requesters
	Get the details of an API Requester
POST	Deploy an API Requester
PUT	Update an API Requester
	Change the status of an API Requester
DELETE	Delete an API Requester

```
GET /zosConnect/apiRequesters cscvinc.aar
PUT /zosConnect/apiRequesters/{apiRequesterName}?status=started|stopped
PUT /zosConnect/apiRequesters cscvinc.aar
GET /zosConnect/apiRequesters
GET /zosConnect/apiRequesters/{apRequesterName}
DELETE /zosConnect/apiRequesters
```

Deploying Service Archive options

- Use SAR as request message and use HTTP POST
- Use URI path /zosConnect/services
- Postman or cURL



The screenshot shows the Postman interface with a POST request to <https://wg31.washington.ibm.com:9453/zosConnect/services>. The 'Body' tab is selected, showing the SAR file content:

```

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

```

Command:

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

Results:

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



Deploying Service Archive – z/OS options

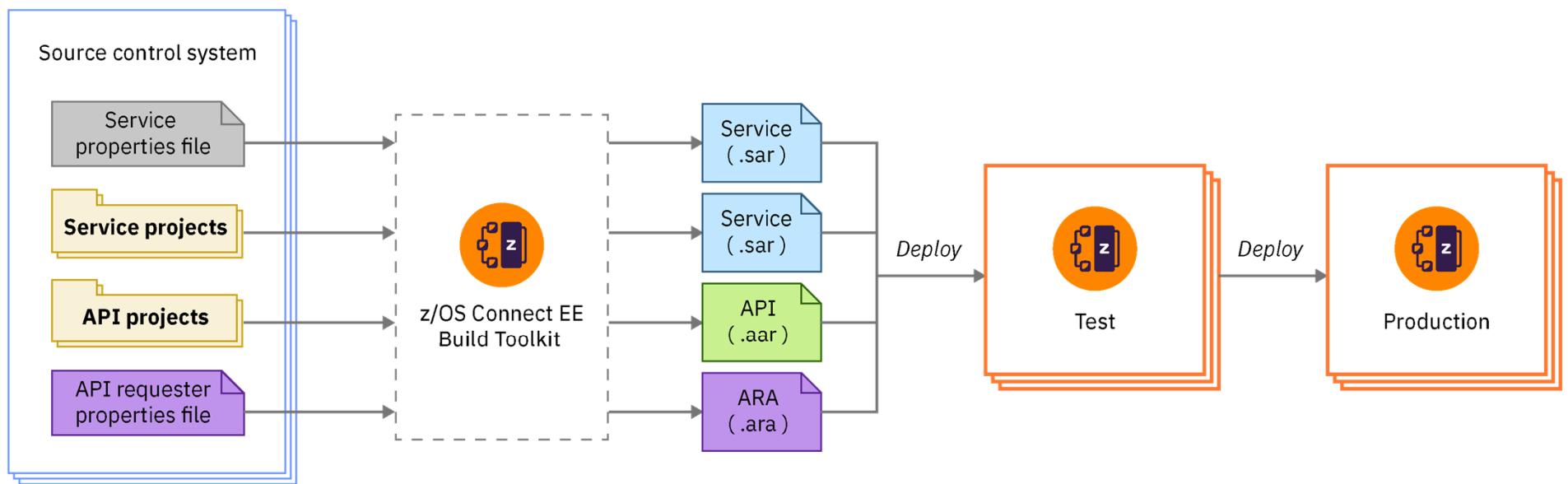
```
//*****
///* SET SYMBOLS
//*****
//EXPORT EXPORT SYMLIST=(*)
// SET CURL='/usr/lpp/rocket/curl'
//*****
///* CURL Procedure
//*****
//CURL PROC
//CURL EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR DD SYSOUT=*
//STDOUT DD SYSOUT=*
// PEND
//*****
///* STEP CURL - use cURL to stop API 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/cscvinc
```

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

DevOps using z/OS Connect EE

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

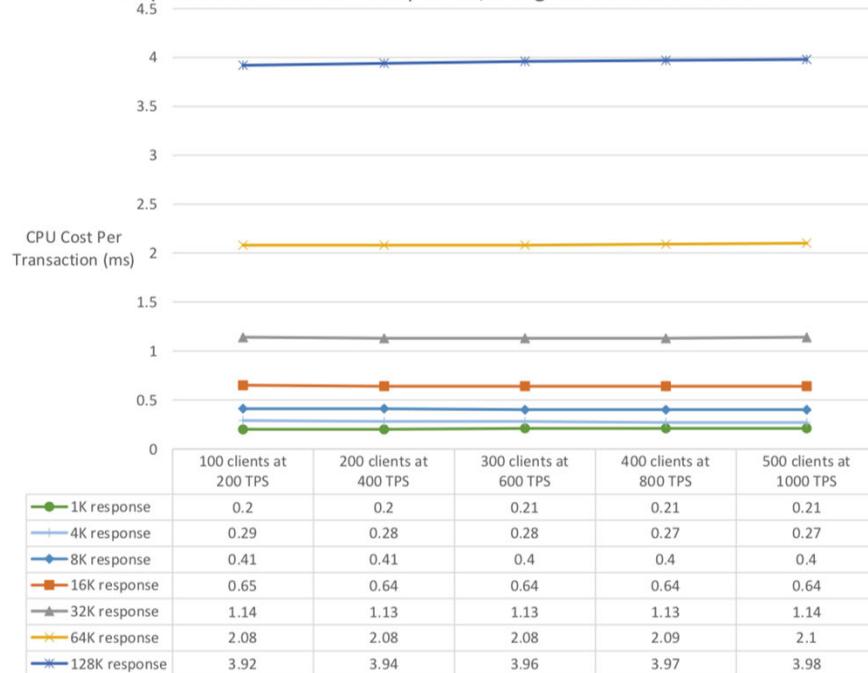
- The build toolkit supports the generation of service archives and API archives from projects created in the z/OS Connect EE API toolkit
- The build toolkit also supports the use of properties files to generate API requester archives
- Run the build toolkit from a build script to generate these archive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



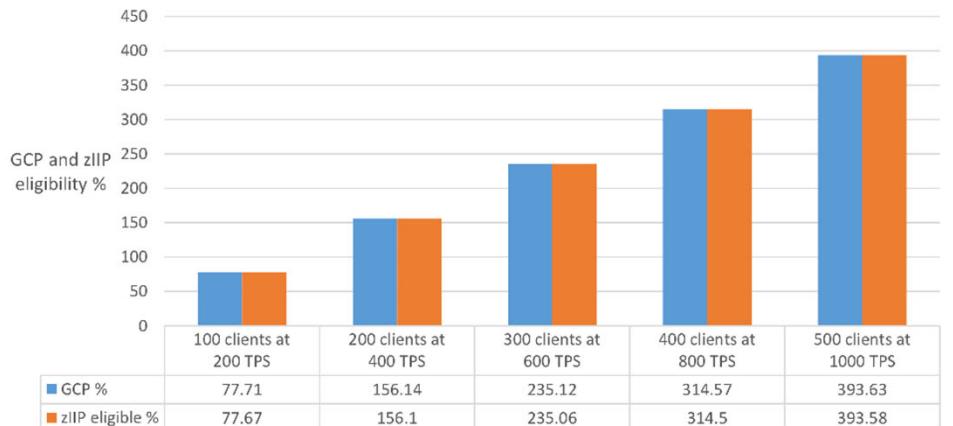
Performance: API Provider

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



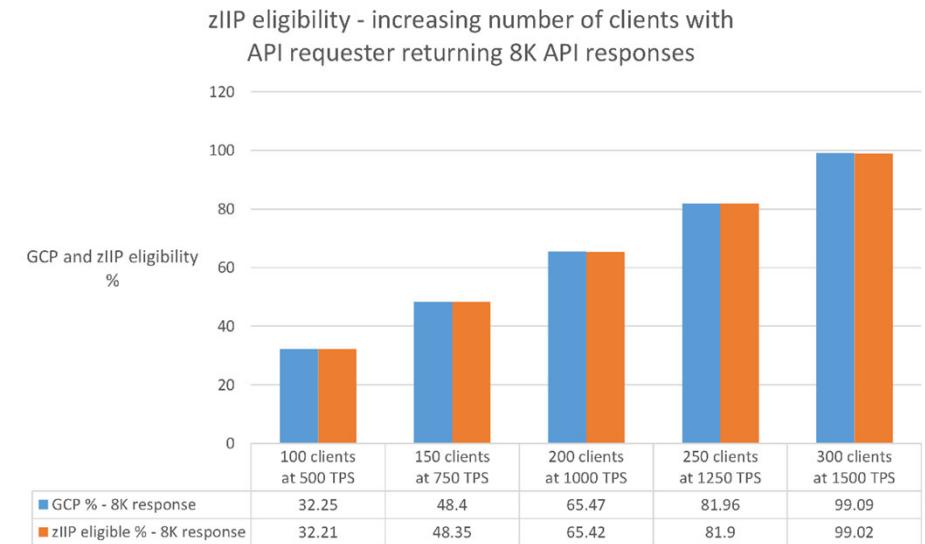
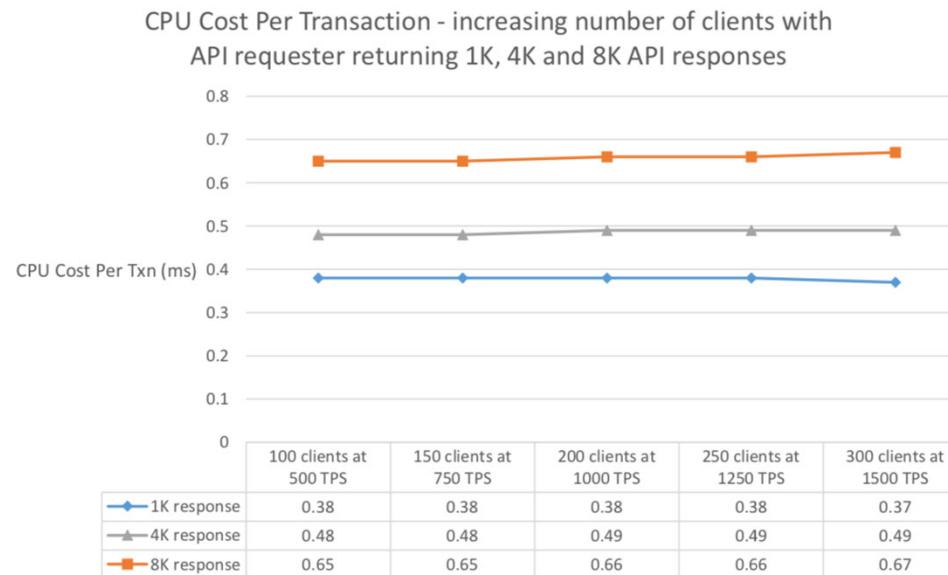
zIIP eligibility - increasing number of clients with 50 byte requests and 128K responses, using channels and CICS SP



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

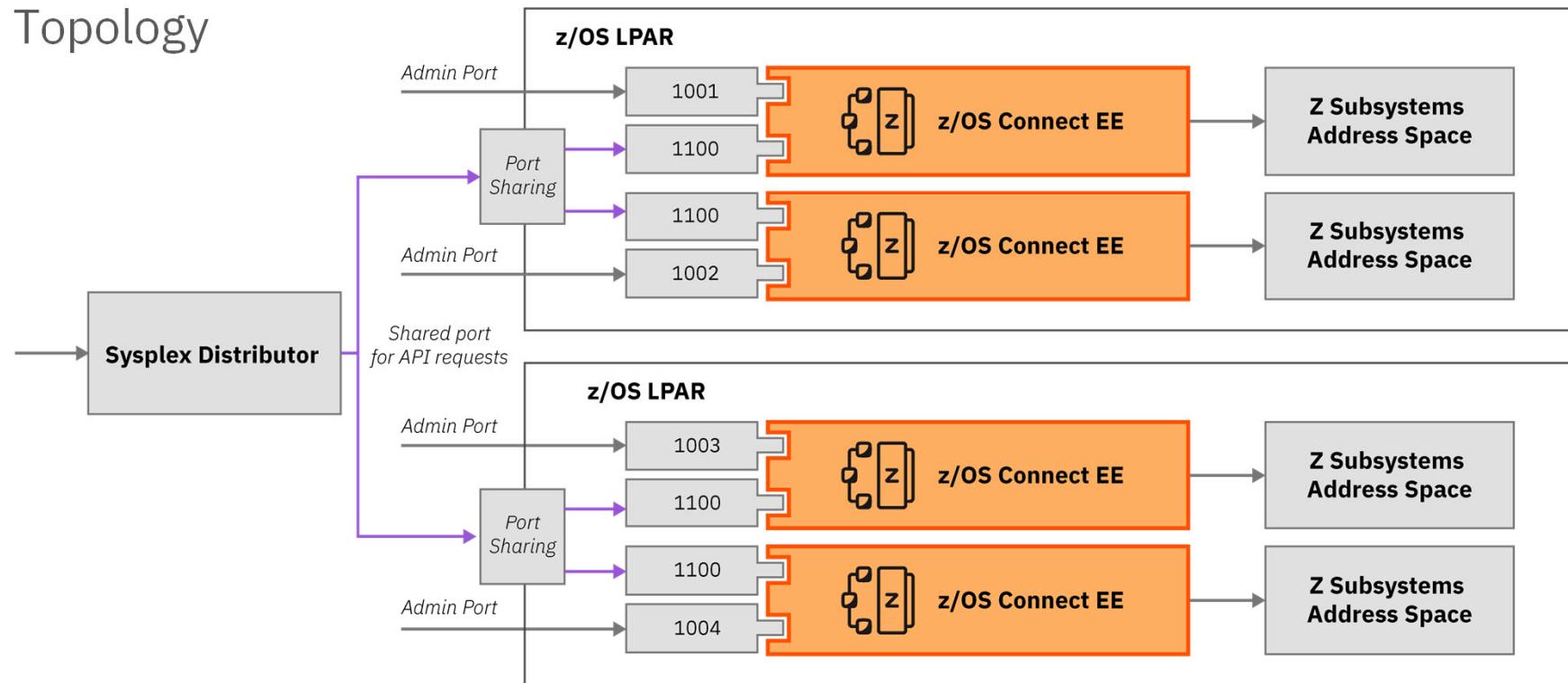


z/OS Connect EE is a Java-based product:
Over **99%** of its MIPs are **eligible for ZIIP offload**.



High Availability

Topology



ibm.biz/zosconnect-ha-concepts

ibm.biz/zosconnect-scenarios



/security

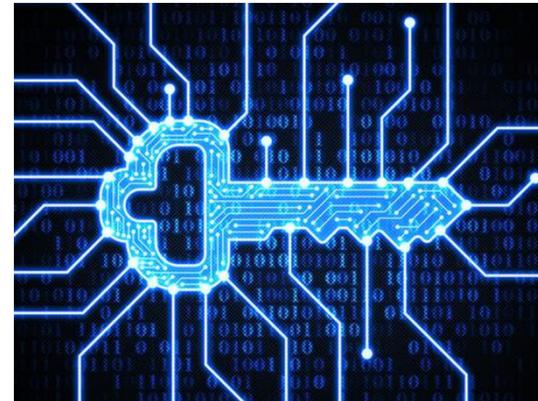
How is security implement?

General considerations for securing REST APIs



z/OS Connect EE

- Know who is invoking the API (**Authentication**)
 - Basic Authentication
 - TLS (mutual authentication)
 - Third Party Tokens
- Ensure that the data has not been altered in transit (**Data Integrity**) and ensure confidentiality of data in transit (**Encryption**)
 - TLS – (encrypted and signed messages)
- Control access (**Authorization**)
 - End user access to z/OS Connect
 - Access to APIs, etc.



Security Challenges



z/OS Connect EE

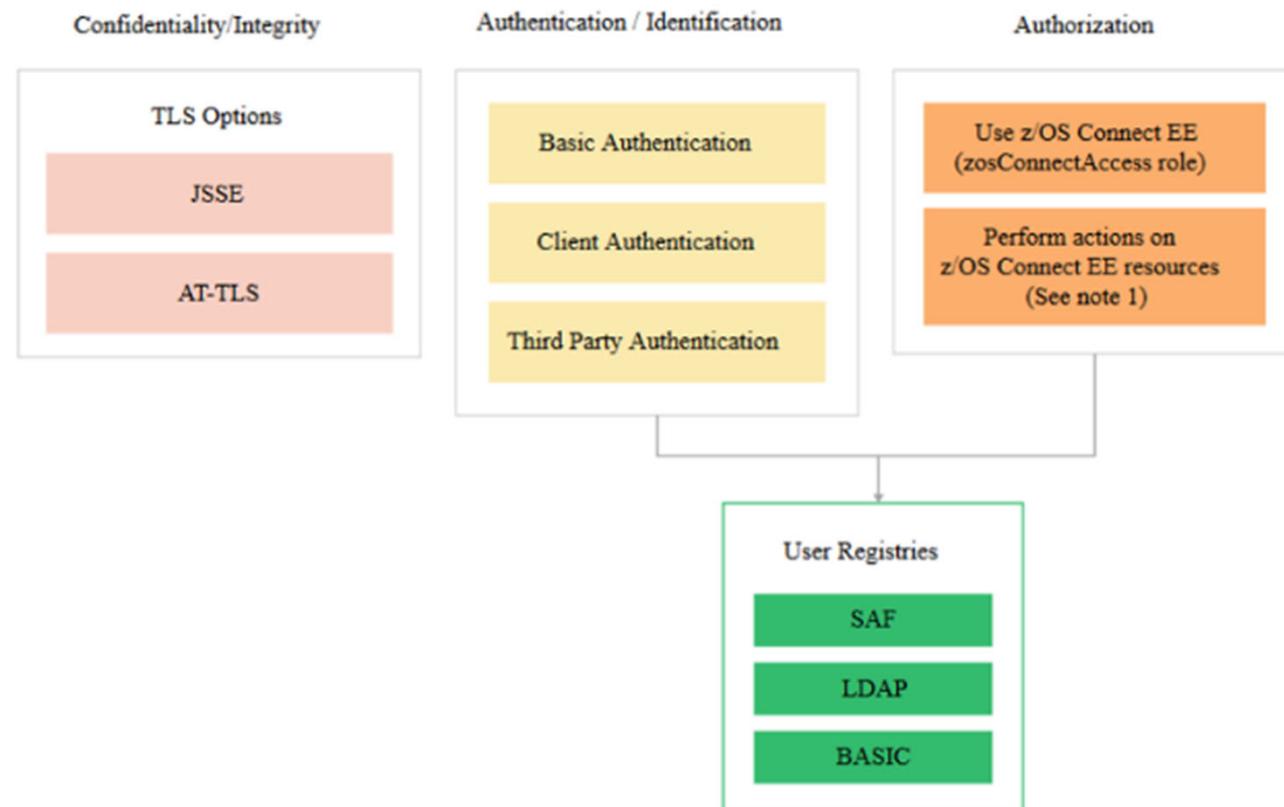
- providing secure access between middleware components that use disparate security technologies e.g., registries like LDAP, SAF, TLS etc.
 - This is a driver for implementing open security models like OAuth and OpenID Connect and standard tokens like JWT
- integrating security involving different products including z/OS Connect, WebSphere Liberty Profile on z/OS, CICS, IMS, Db2, MQ,... probably for the first time in an environment.
 - And these are all documented in different places
- often at odds with **performance**, because the most secure techniques often involve the most processing overhead especially if not configured optimally
 - Very few want to implement security, they are usually directed to implement security.

A single step-by-step map is not possible, the approach is building a solution using components, one step at a time based on way points. These way points are what this presentation focuses on.

z/OS Connect EE security options



z/OS Connect EE



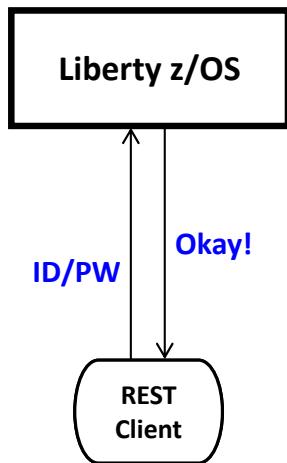
API Provider Authentication



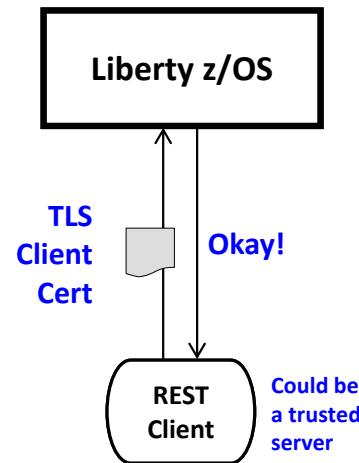
z/OS Connect EE

Several different ways this can be accomplished:

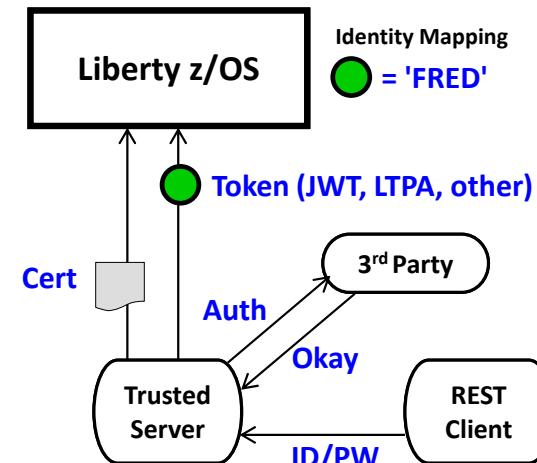
Basic Authentication



Client Certificate



Third Party Authentication



Server prompts for ID/PW

Client supplies ID/PW or
ID/Passticket

Server checks registry:

- Basic (server.xml)
- LDAP
- SAF

Server prompts for cert.

Client supplies certificate

Server validates cert and
maps to an identity

Registry options:

- LDAP
- SAF

Client authenticates to 3rd party sever

Client receives a trusted 3rd party token

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

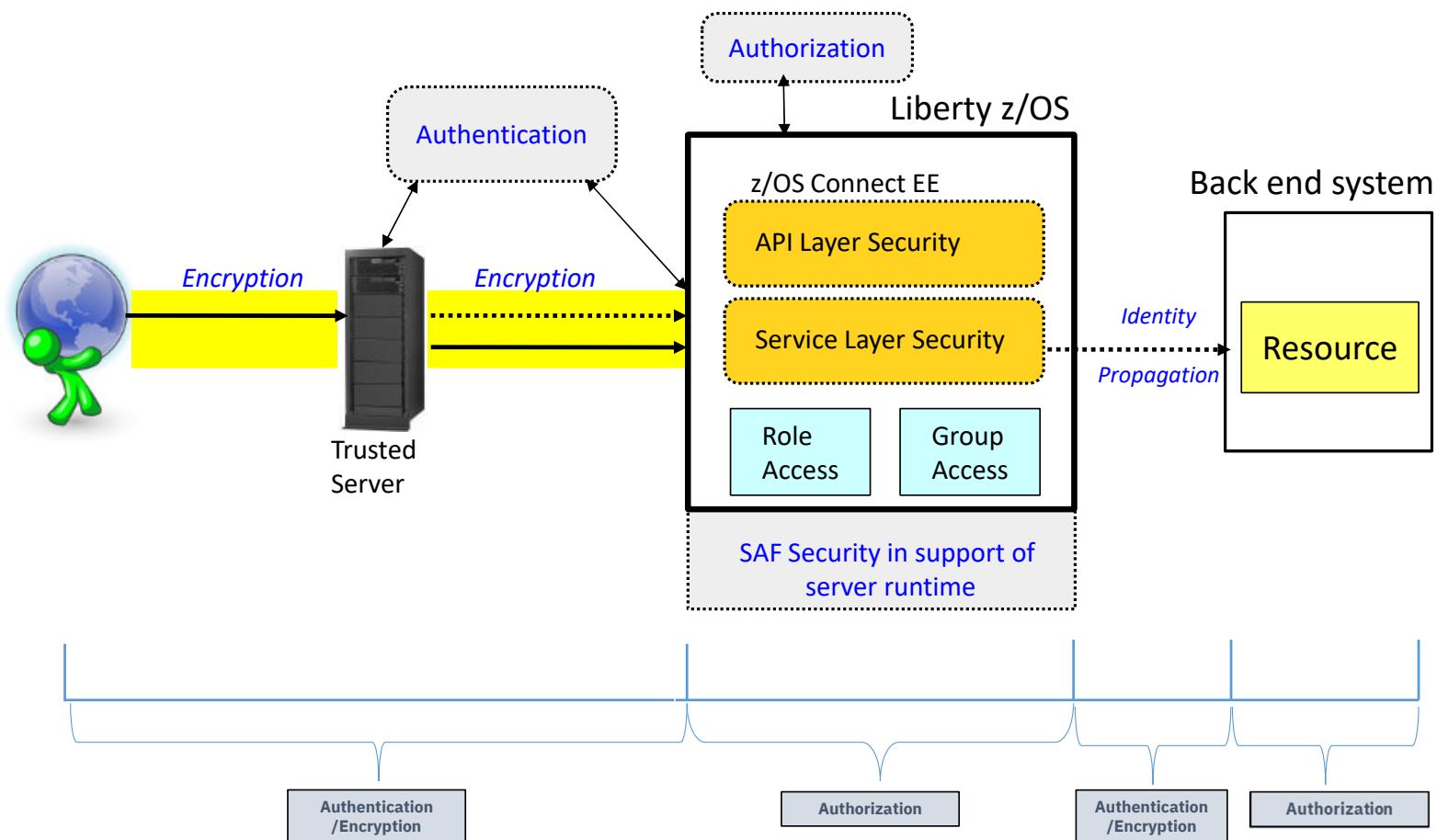
Registry options:

- LDAP
- SAF

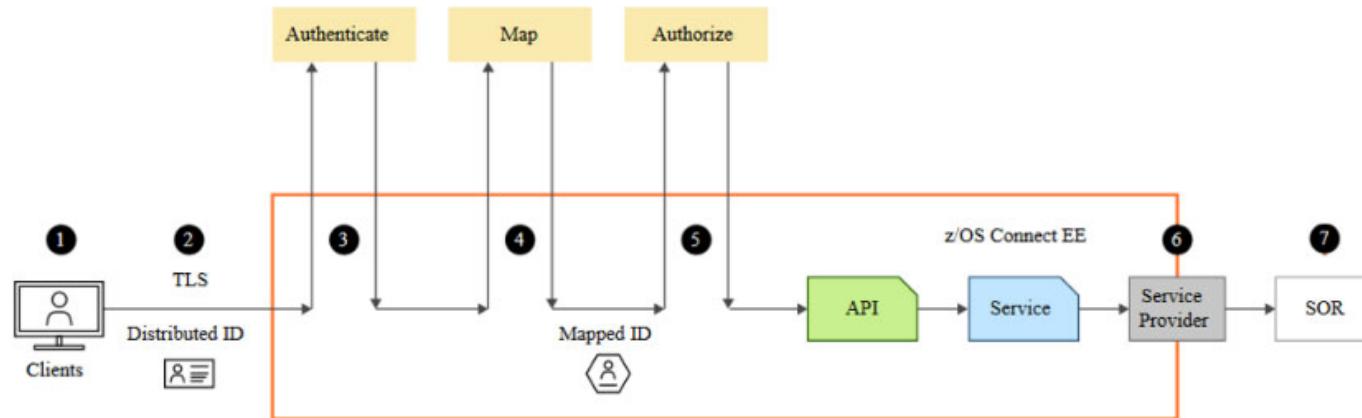
z/OS Connect EE API provider security overview



z/OS Connect EE



Typical z/OS Connect EE API Provider security flow

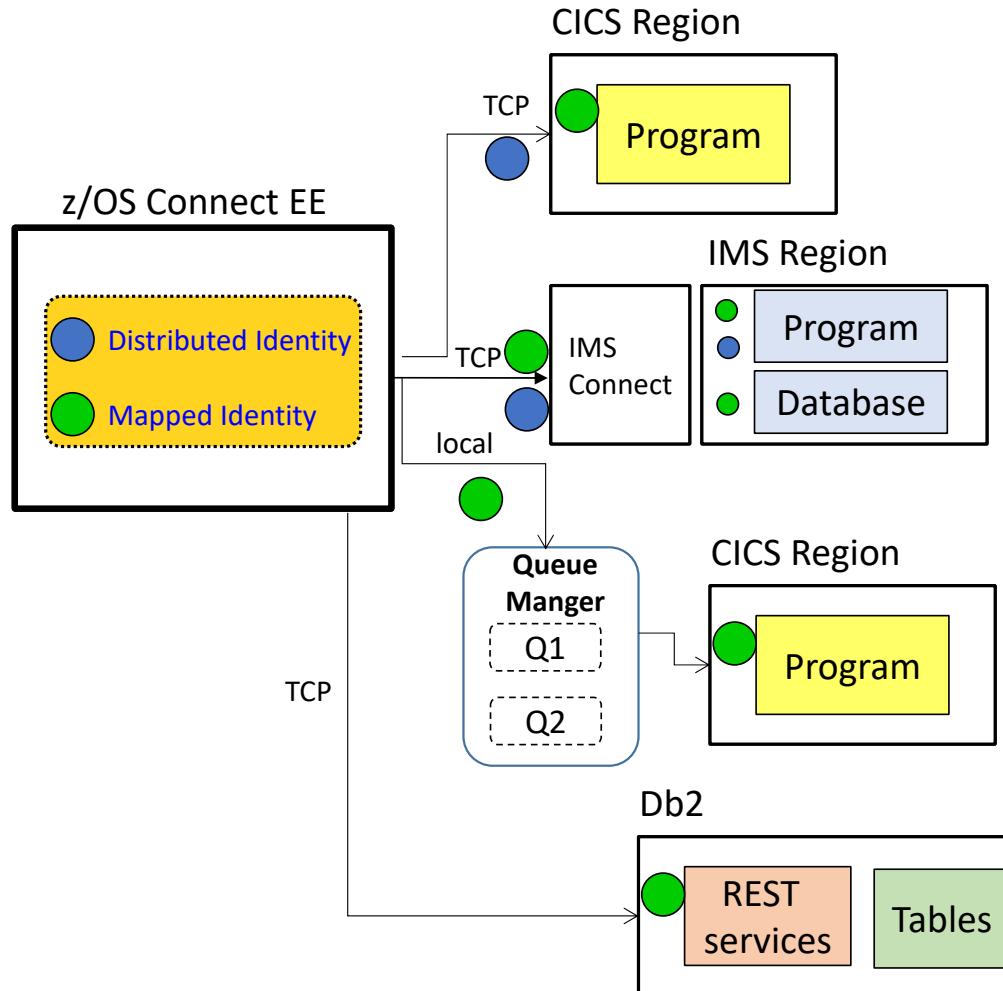


1. The credentials provided by the client
2. Secure the connection to the z/OS Connect EE server
3. Authenticate the client. This can be within the z/OS Connect EE server or by requesting verification from a third-party server
4. Map the authenticated identity to a user ID in the user registry
5. Authorize 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

Flowing an identity to the back end



z/OS Connect EE



The CICS SP propagates the distributed id to CICS (or sends the SAF id if it has been used for authentication)

The IMS TM and DB SPs asserts the mapped identity to IMS but also sends the distributed id for audit purposes (V3.0.33) added support for Passtickets.

The MQ SP asserts the mapped identity to the queue manager and onto CICS

The REST client SP can request a Passticket and send with mapped ID to Db2 (V3.0.15)

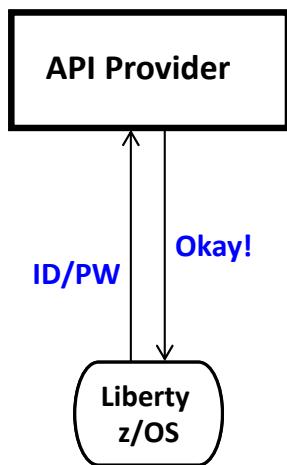
API Requester Authentication



z/OS Connect EE

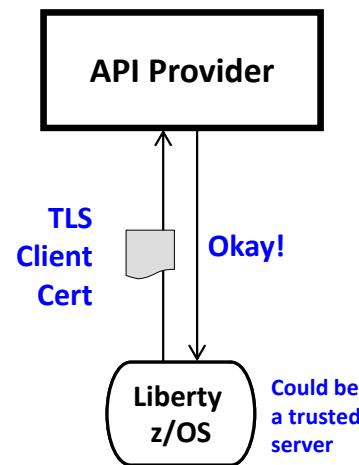
Several different ways this can be accomplished:

Basic Authentication



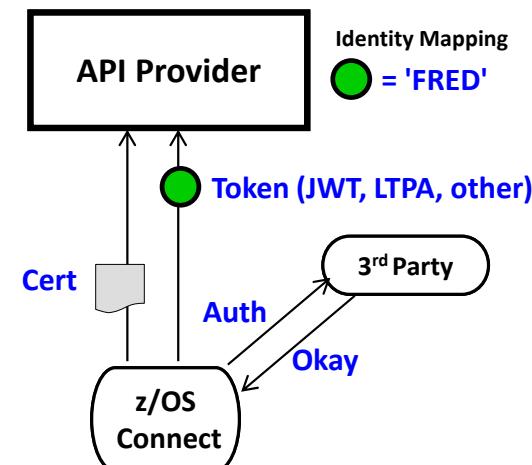
zCEE supplies ID/PW or
ID/Passticket

Client Certificate



Server prompts for certificate
zCEE supplies certificate

Third Party Authentication

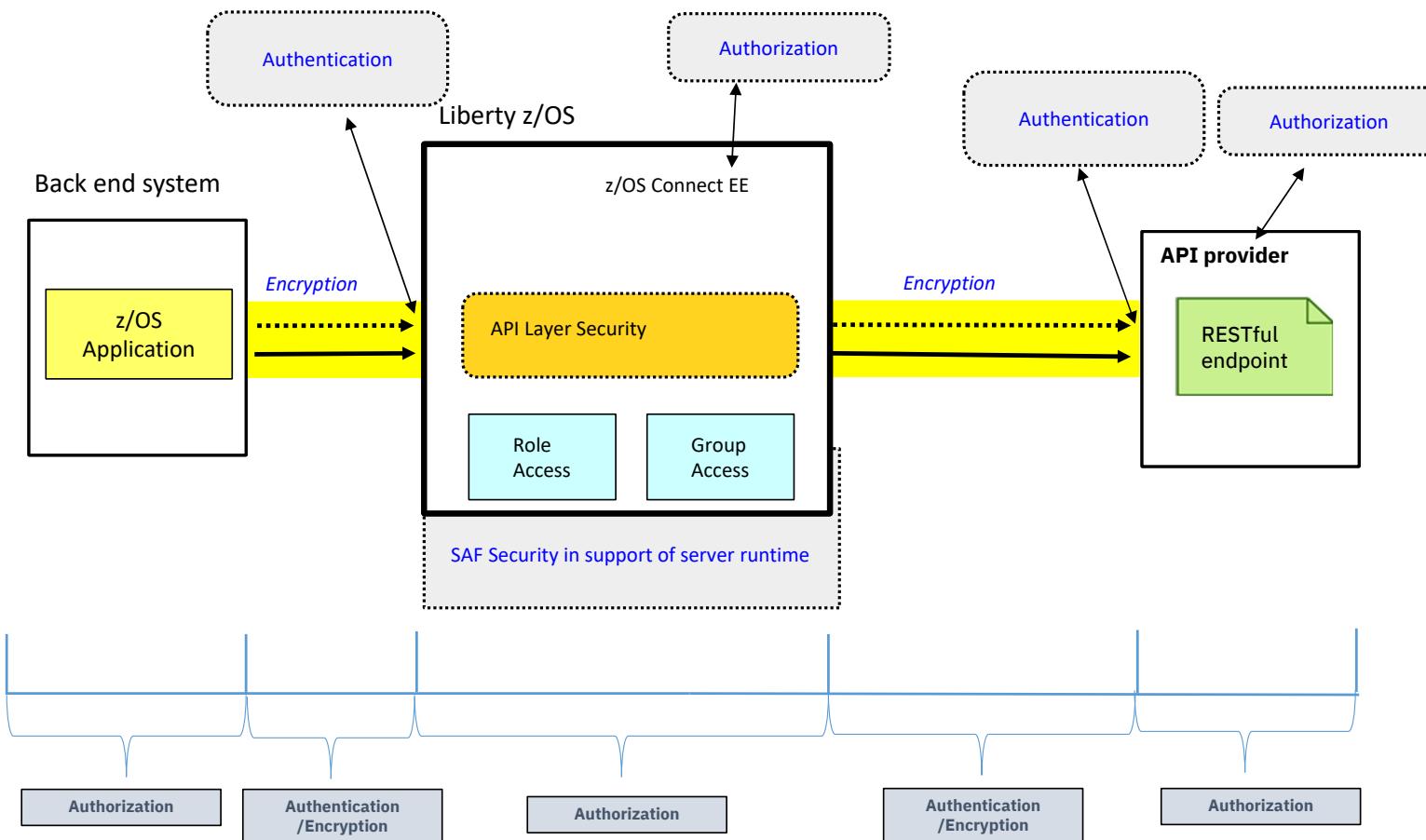


zCEE authenticates to 3rd party sever
zCEE receives a trusted 3rd party token
Token flows to API Provider

API requester security – overview



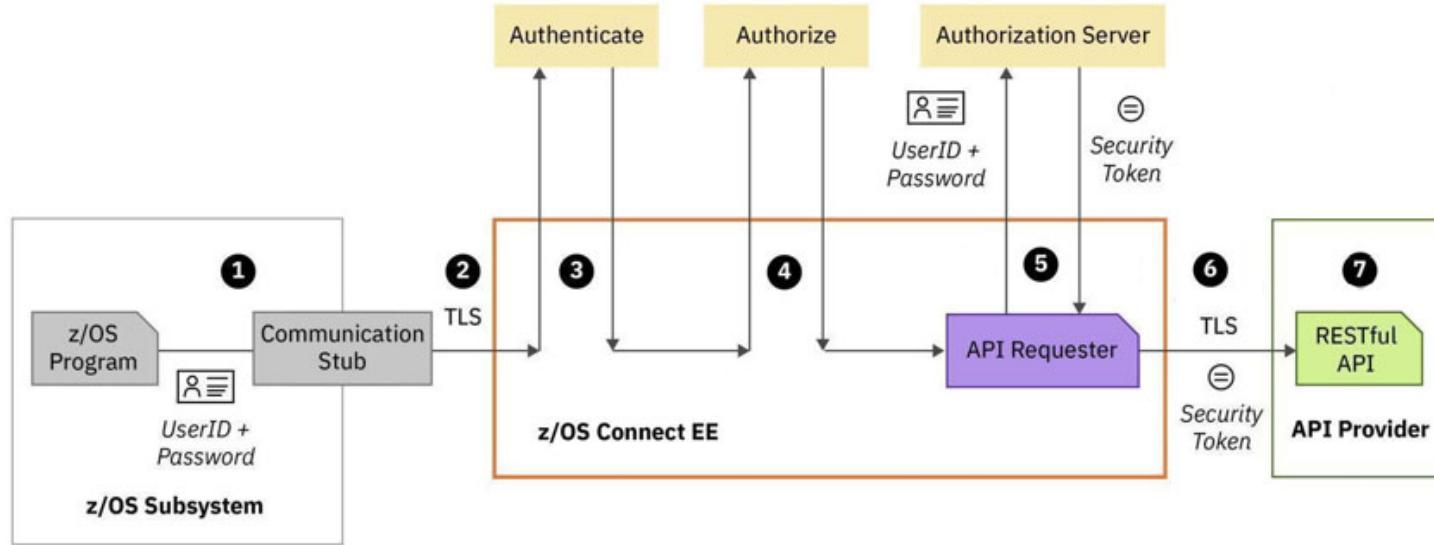
z/OS Connect EE



Typical z/OS Connect EE API Requester security flow



z/OS Connect EE



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

Github Site



Screenshot of a GitHub repository page for 'ibm-wsc/zCONNEE-Wildfire-Workshop'. The repository has 162 commits, 2 branches, 0 packages, 0 releases, and 1 contributor. It contains files like 'Misc Presentations', 'cobol', 'exercises', 'security', 'Introduction to zOS Connect EE.pdf', 'README.md', 'WP102724 - zOS Connect EE V3 Getting Started.pdf', and 'WSC Wildfire zOS Primer.pdf'. A note at the bottom states: 'This repository contains material from the z/OS Connect EE Wildfire workshops run by the IBM Washington Systems Center.'

The repository has 7 stars, 4 forks, and 2 issues. The 'exercises' folder contains several PDF files related to developing RESTful APIs for various z/OS Connect services. The 'security' folder also contains PDF files related to customization security.

File	Description	Last Commit
Developing Outbound APIs Requesters Applications.pdf	Add files via upload	7 days ago
Developing RESTful APIs for a CICS COMMAREA program.pdf	Add files via upload	6 months ago
Developing RESTful APIs for DVM Services.pdf	Add files via upload	2 months 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.pdf	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	last month
Developing RESTful APIs for a CICS program.pdf	Add files via upload	7 days ago
zCEE Customization Basic Configuration.pdf	Add files via upload	2 months ago
zCEE Customization Basic Security.pdf	Add files via upload	last month

File	Description	Last Commit
zCEE Customization Basic Configuration.pdf	Add files via upload	2 months ago
zCEE Customization Basic Security.pdf	Add files via upload	last month
zCEE Customization Security and CICS.pdf	Add files via upload	4 days ago
zCEE Customization Security and DB2.pdf	Add files via upload	4 days ago
zCEE Customization Security when accessing an IMS Database.pdf	Add files via upload	2 days ago
zCEE Customization Security when accessing an IMS Transaction.pdf	Add files via upload	2 days ago
zCEE Customization Security with MVS Batch.pdf	Add files via upload	4 days ago

mitchj@us.ibm.com

• <http://tinyurl.com/y28fsezs>

© 2018, 2021 IBM Corporation



/questions?thanks=true

Thank you for listening.

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