



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
- Overview of z/OS Connect Security, for more on security, see workshop
zOSSEC1 IBM z/OS Connect Security Wildfire Workshop

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
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”?

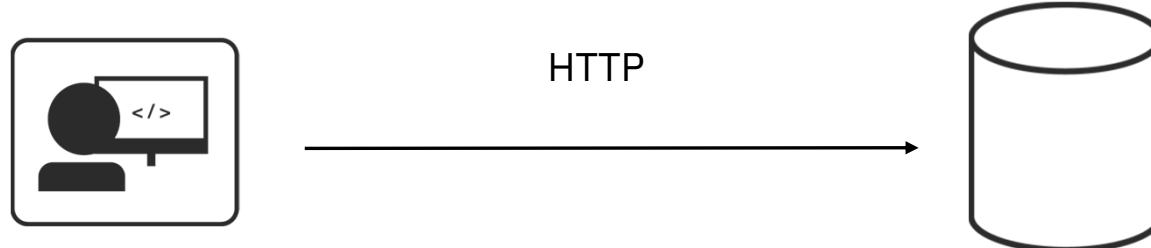
REST is an Architectural Style

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 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 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 with the same basic principles, REST APIs are not

RESTful Examples



z/OS Connect EE

z/OS Connect Enterprise Edition:

POST /account/ +  (*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 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 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



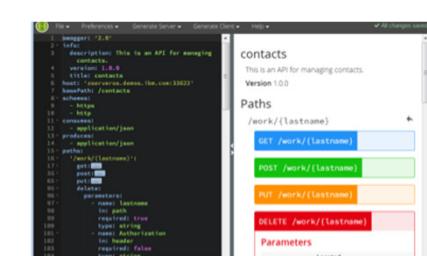
Read Swagger⁺

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



Write Swagger

Swagger Editor allows API developers to design their swagger documents.



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

Swagger Example



z/OS Connect EE

The image shows two side-by-side Swagger UI browser windows. Both windows have a header bar with back, forward, home, and search icons, and tabs for 'JSON', 'Raw Data', and 'Headers'. The left window displays the main API definition for 'miniloan' at `/C:/z/workspaces/zCEE/Miniloan/api-docs`. It includes sections for `swagger`, `info`, `schemes`, `consumes`, `produces`, and `path`. The `path` section contains a `/loan` endpoint with a `post` method. A red oval highlights the `basePath` field in the `info` section and the `post` method in the `path` section. Another red oval highlights the `schema` field within the `parameters` of the `post` method, which points to `#/definitions/postMiniloanService_request`. The right window also displays the same API definition at the same URL. A large red oval highlights the entire `definitions` section, which contains a `postMiniloanService_request` schema. This schema is defined as an object type with properties: `MINILoAN_COMMAREA` (string type, max length 20), `creditscore` (integer type, min 0, max 10000000000000000000), `yearlyIncome` (integer type, min 0, max 10000000000000000000), `age` (integer type, min 0, max 9999999999), `amount` (integer type, min 0, max 10000000000000000000), `effectiveDate` (string type, max length 8), `yearlyRepayment` (integer type, min 0, max 10000000000000000000), and a `postMiniloanService_response_200` schema (object type).

mitchj@us.ibm.com

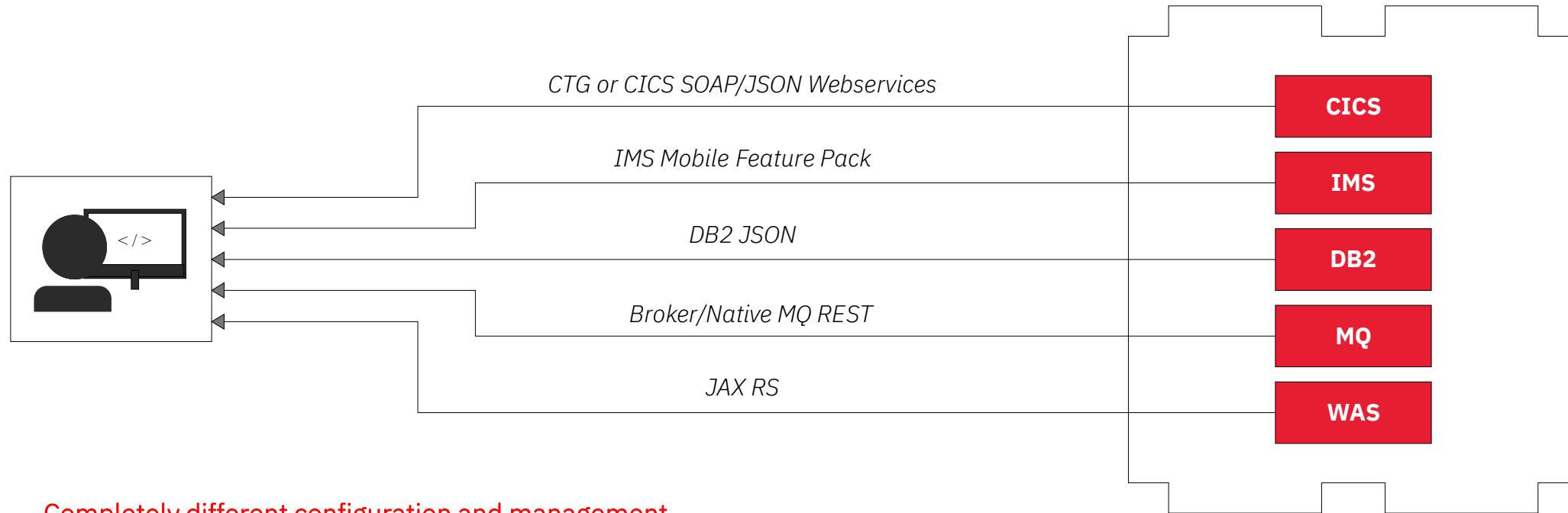
© 2018, 2021 IBM Corporation



Why **/zos_connect_ee?**

Truly RESTful APIs to and from your mainframe.

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



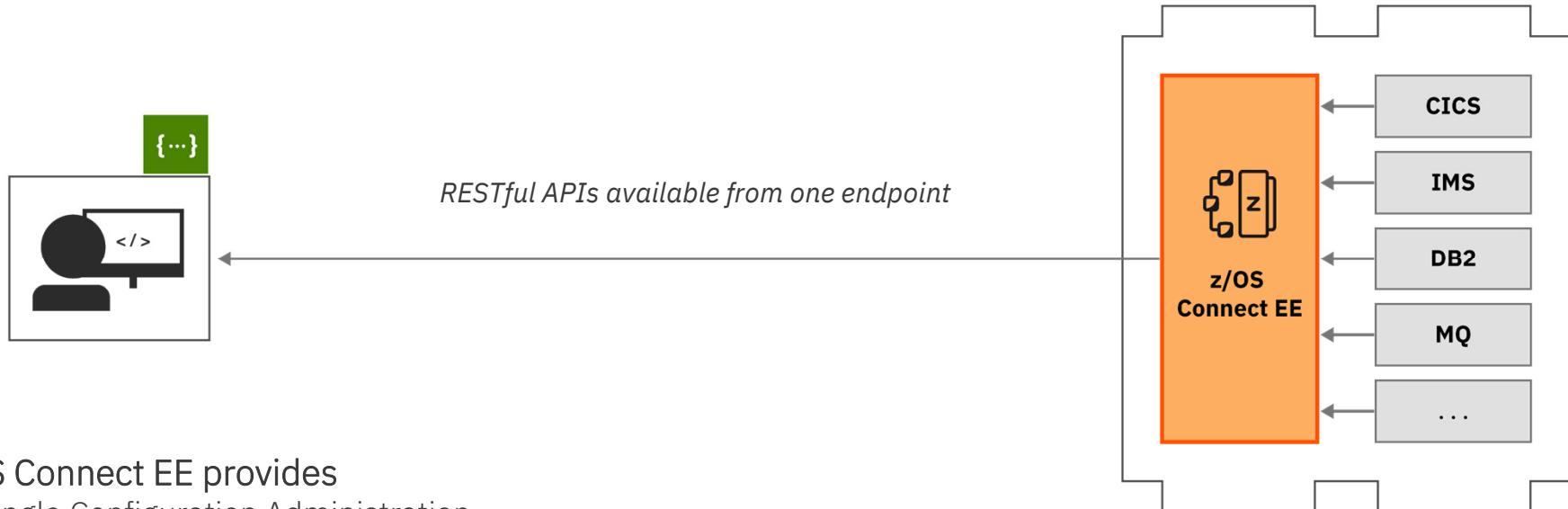
Completely different configuration and management.

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

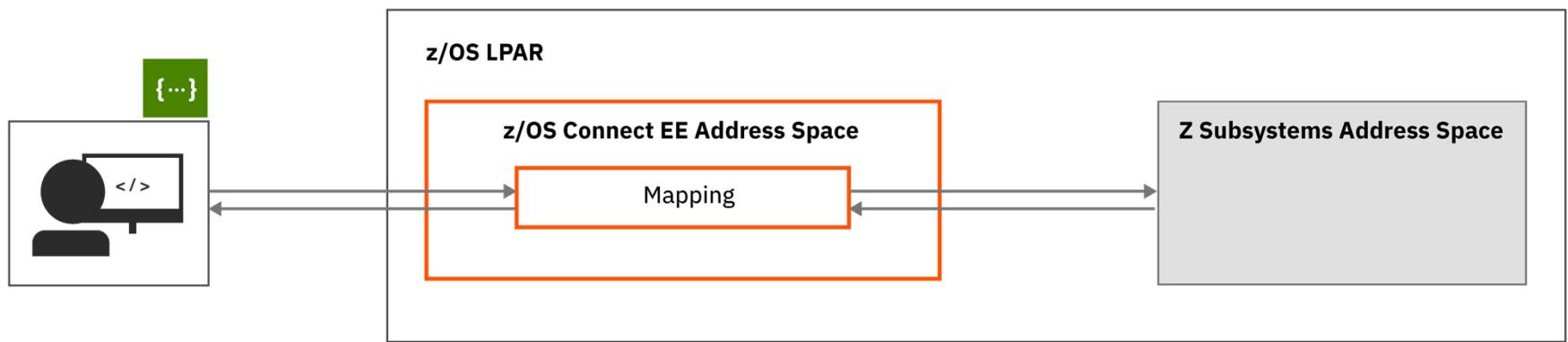
- Single Configuration Administration
- Single Security Administration
- With sophisticated mapping of truly RESTful APIs to existing mainframe and services data without writing any code.



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

Let's Start with Data mapping

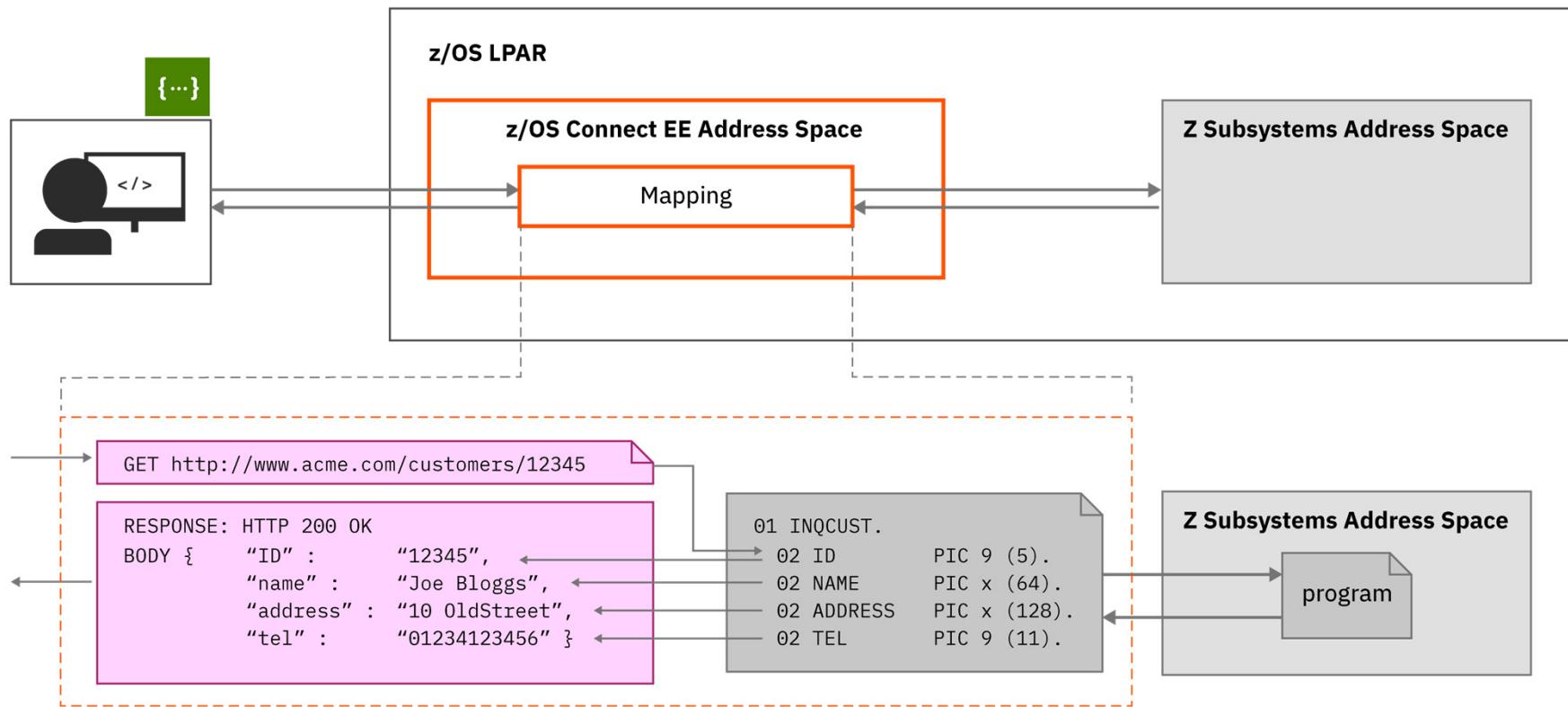
Converting JSON to the format the target's subsystem expects*.



* 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



COBOL versus JSON Schema Example

```

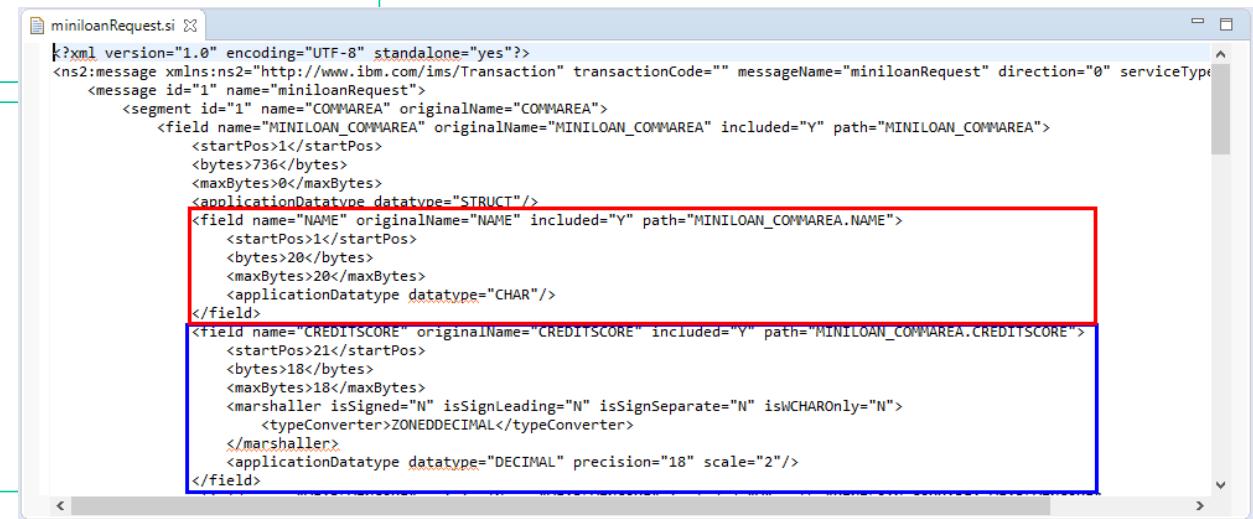
01 MINILOAN-COMMAREA.
 10 name pic X(20).
 10 creditScore pic 9(16)v99.
 10 yearlyIncome pic 9(16)v99.
 10 age pic 9(10).
 10 amount pic 9999999v99.
 10 approved pic X.
    88 BoolValue value 'T'.
 10 effectDate pic X(8).
 10 yearlyInterestRate pic S9(5).
 10 yearlyRepayment pic 9(18).
 10 messages-Num 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" : 9999999999999999.99,
      "type" : "number",
      "format" : "decimal"
    }
  }
},

```



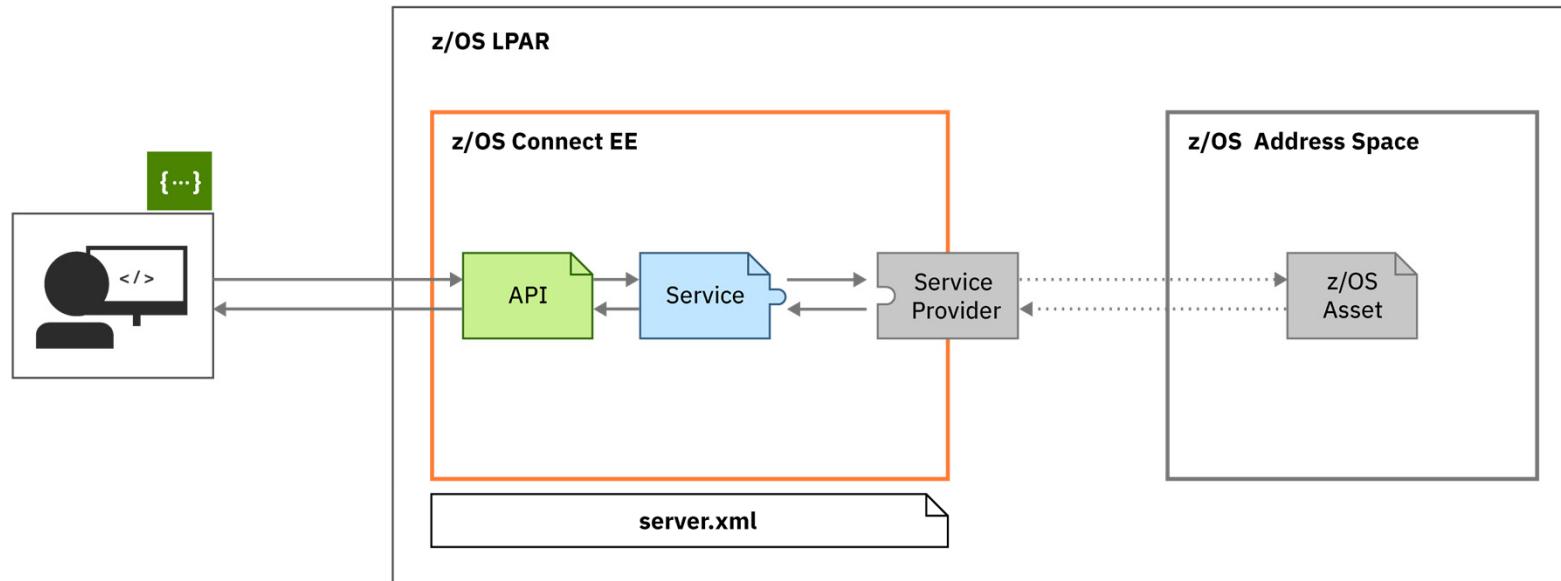
```

<?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

Steps to expose a z/OS application



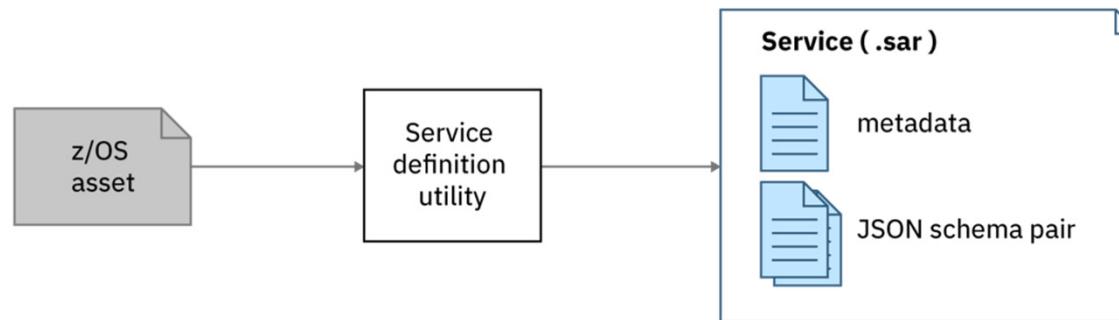
- The API provides the RESTful interface is ready to be consumed by a client and it requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS Asset (e.g., CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific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

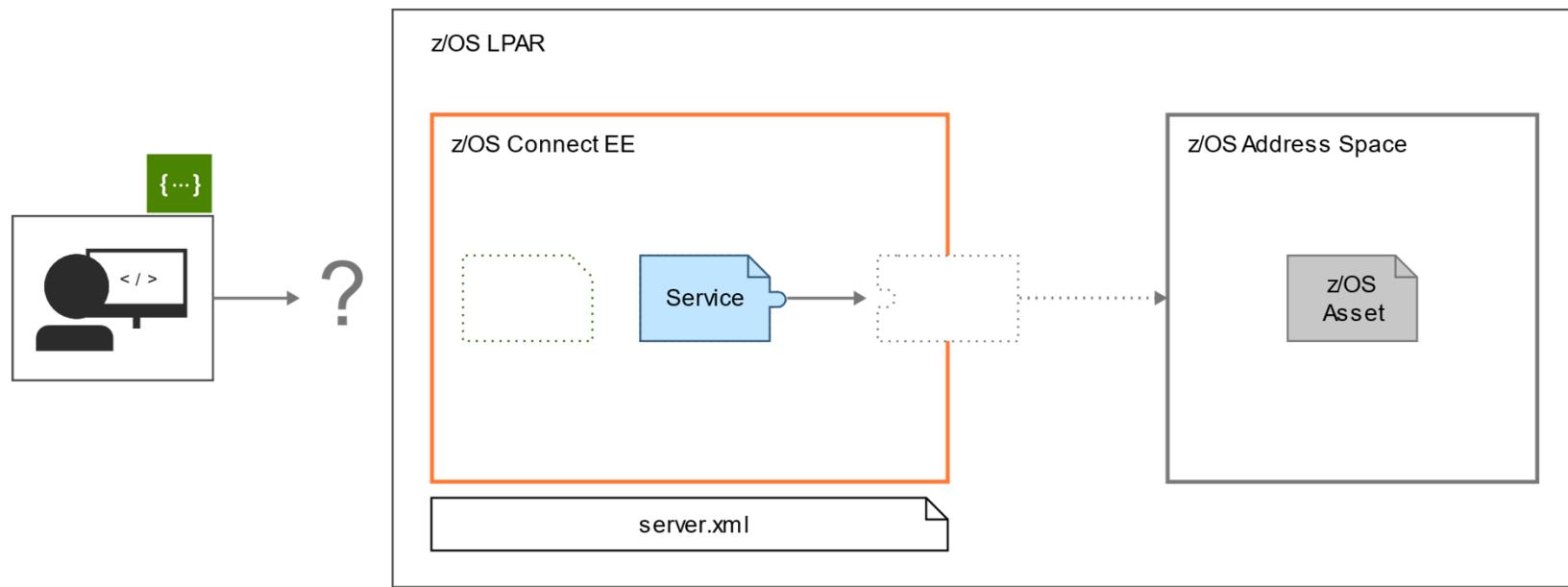


Use a system-appropriate utility to generate a `.sar` file for the z/OS application

- 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

Steps to expose a z/OS application

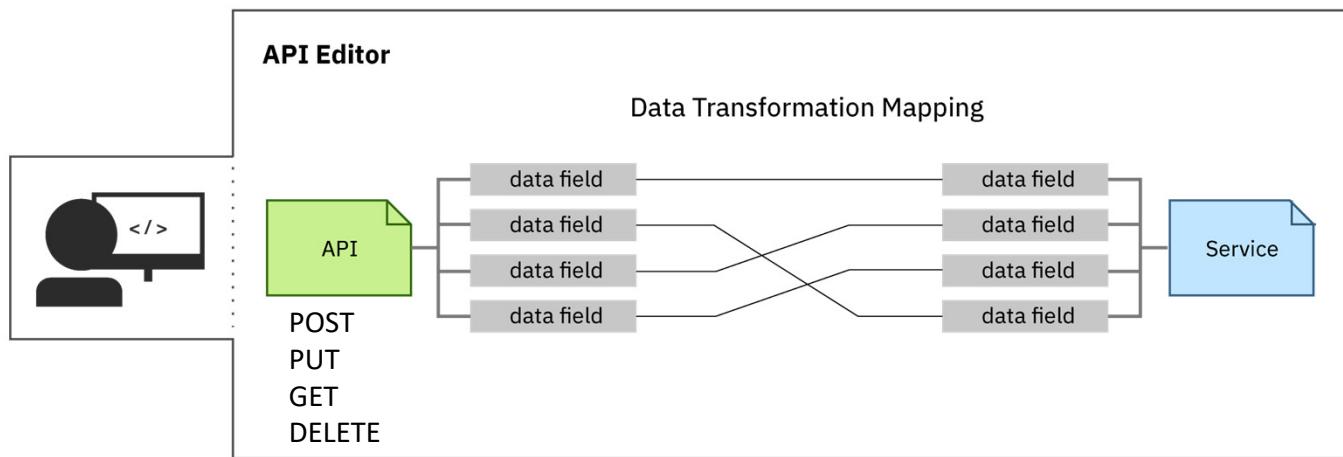
2. Deploy and export the service



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

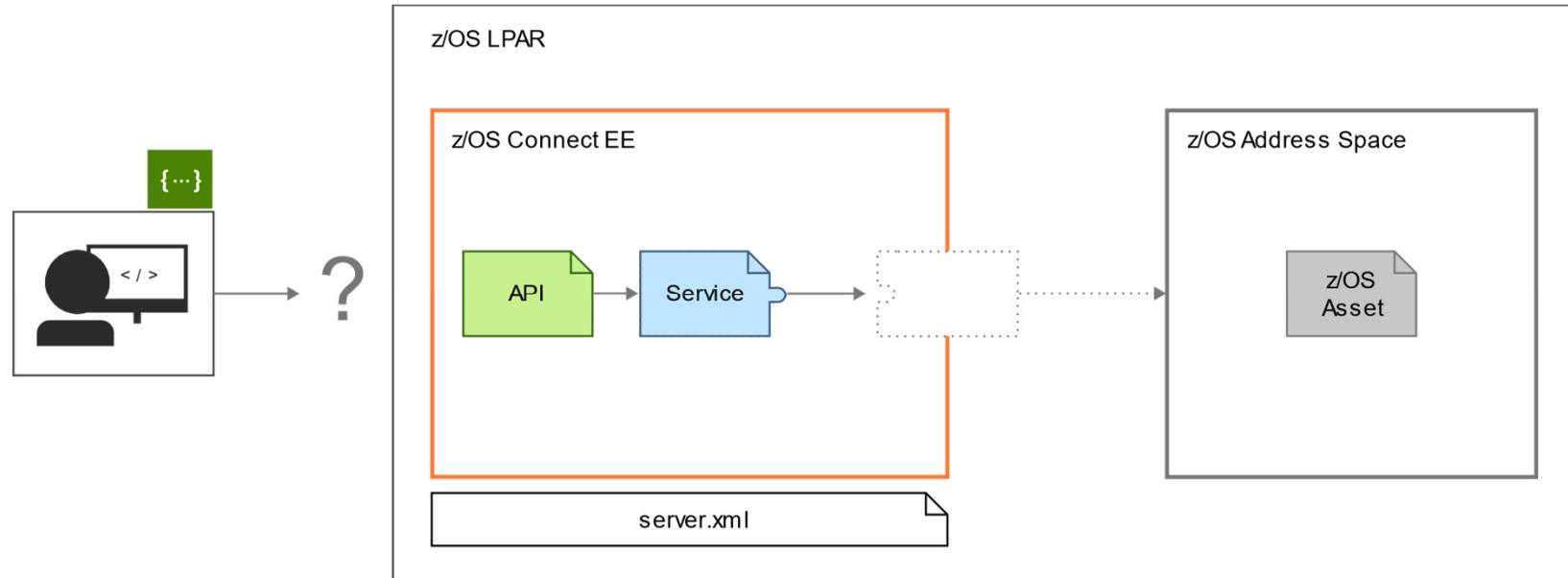
3. Create an API using exported services



- 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.

Steps to expose a z/OS application

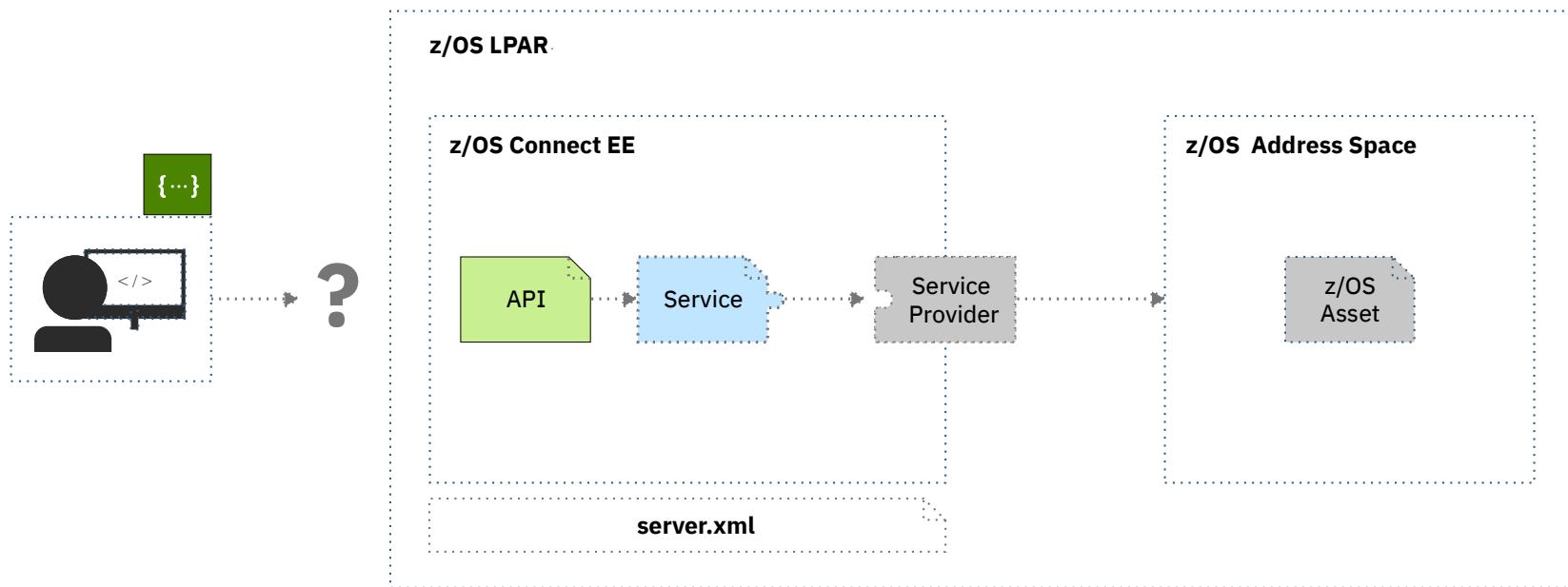
4. Deploy the API



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

Steps to expose a z/OS application

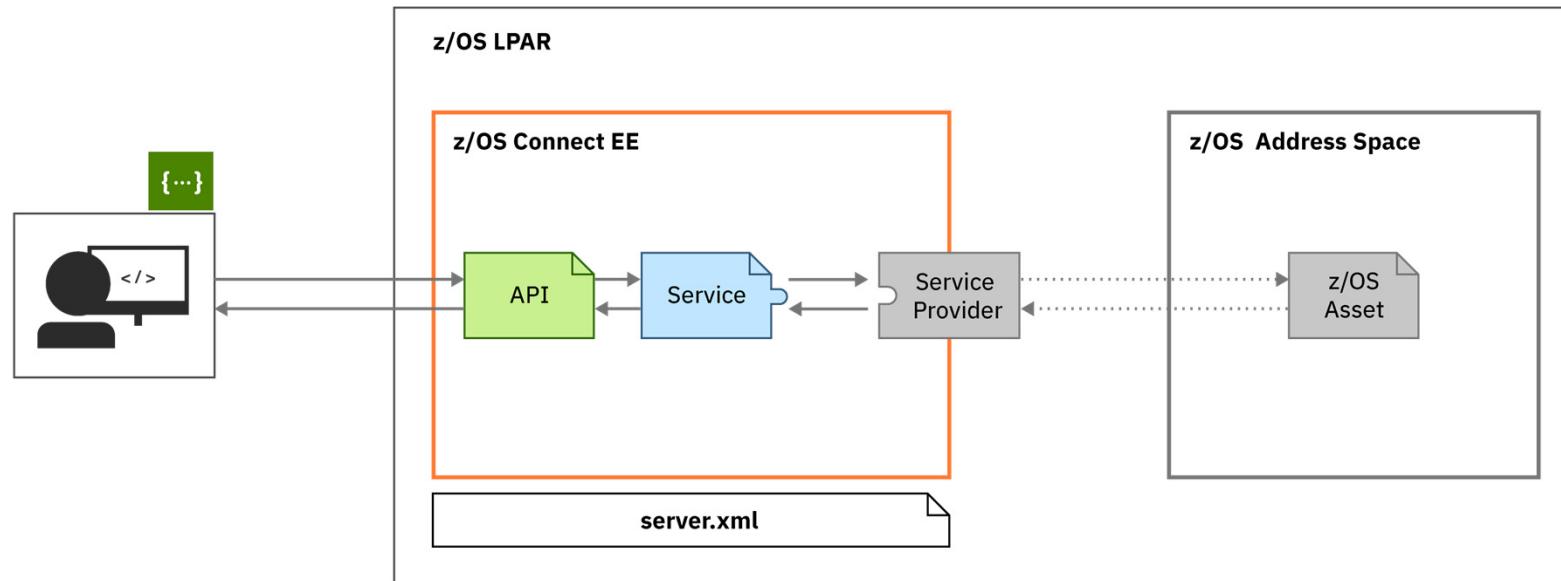
5. Configure the service provider



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)

The Client code is unaware of the z/OS infrastructure



z/OS Connect EE

```
ZceeCICSGetJava
```

```

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: \$0002.00

address: FRANKFURT, GERMANY

phoneNumber: 20034151

date: 26 11 81

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

```
ZceeMqGetJava
```

```

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: SINDELFLINGEN, GERMANY

PHONE: 70319990

DATEX: 21 06 77

AMOUNT: \$0009.99

MQ

```
ZceeDb2GetJava
```

```

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

```
ZceelMSGetJava
```

```

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
61 try {
62     if (conn.getResponseCode() != 200) {
63         throw new RuntimeException("Failed : HTTP error code : " + conn.getResponseCode());
64     }

```

Problems @ Javadoc Declaration Console Coverage

<terminated> ZceelMSGet [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

IMS



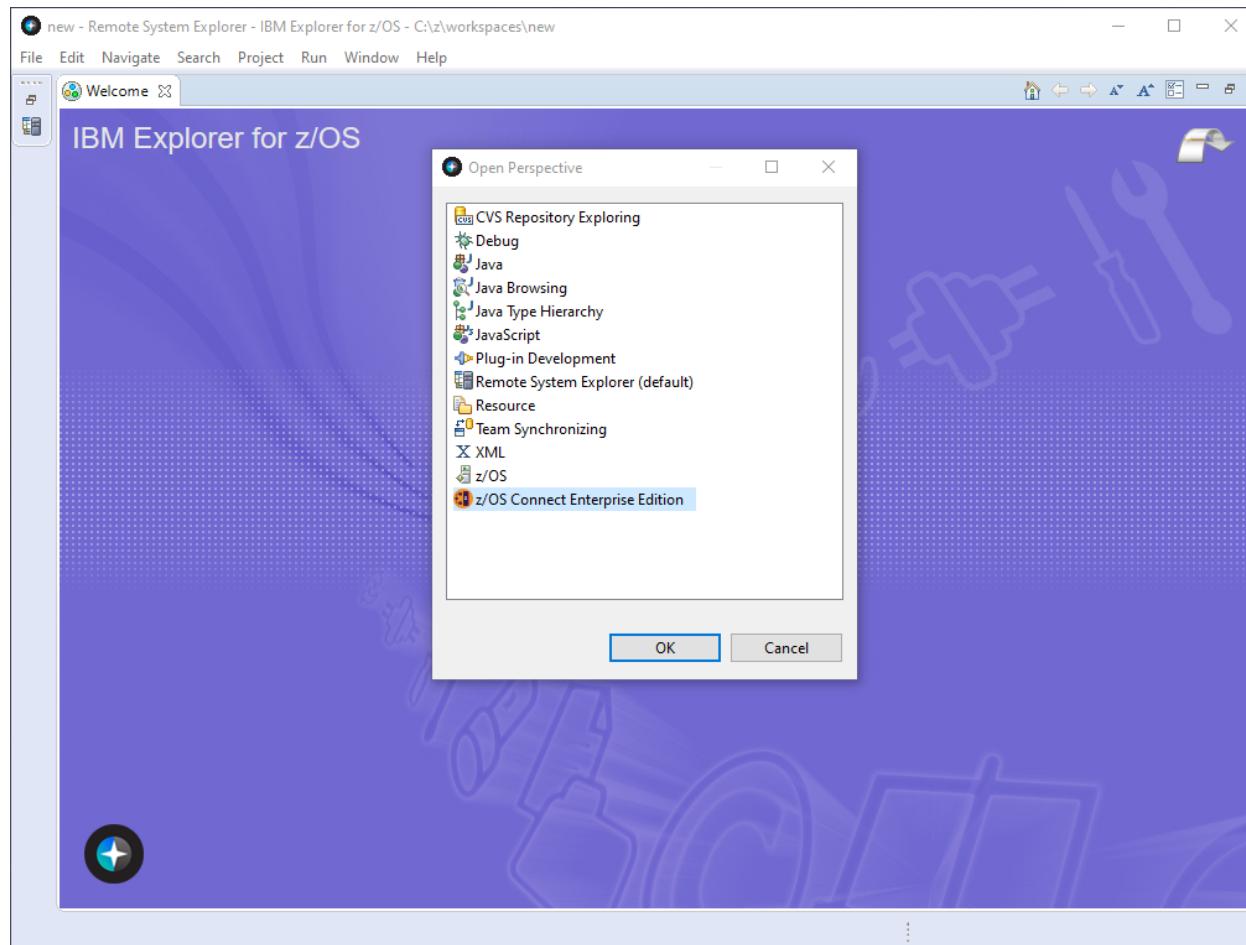
/api_toolkit/services

Simple **service creation.**

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



z/OS Connect EE



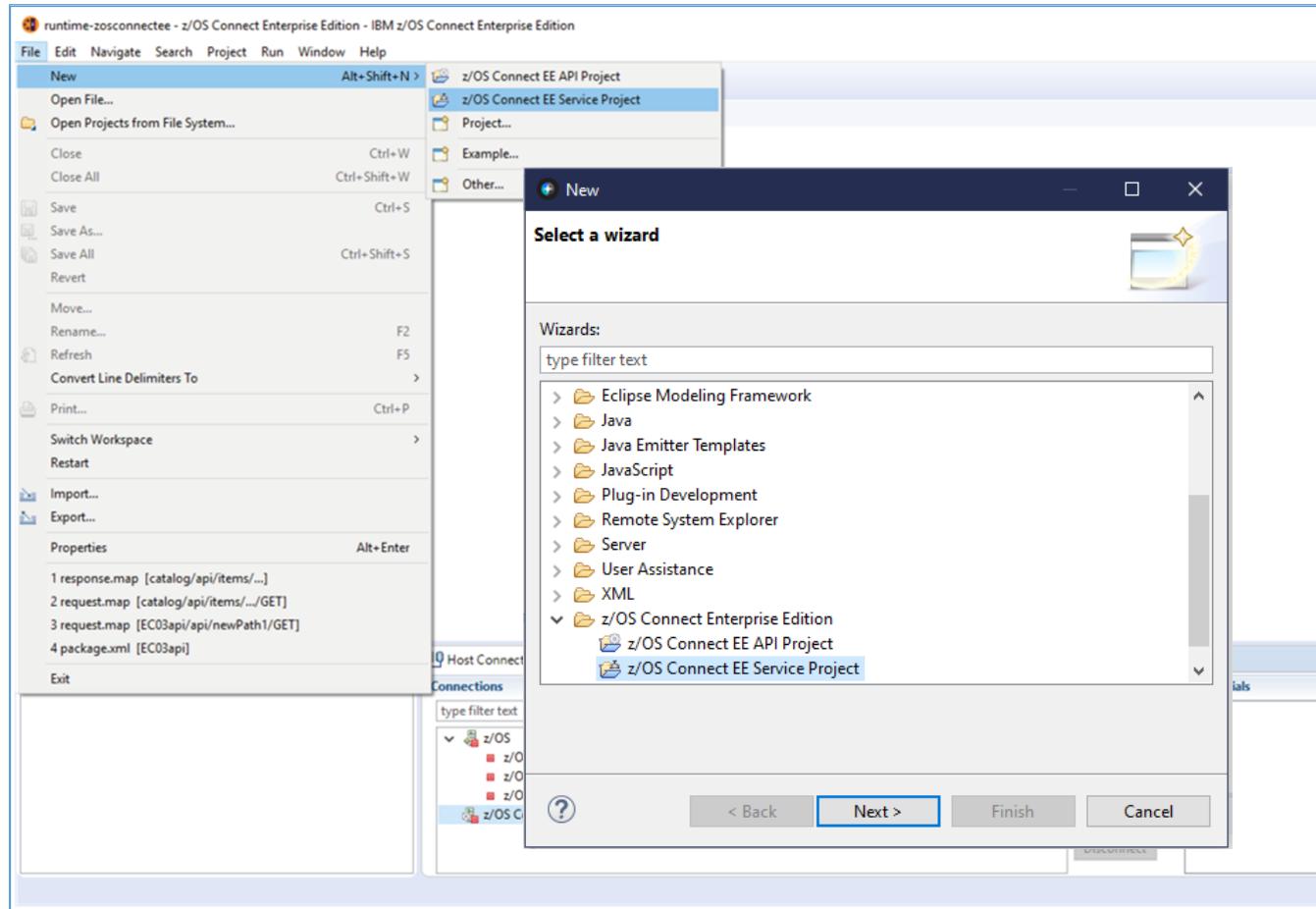
Use the **API toolkit** to create services through Eclipse-based tooling.

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

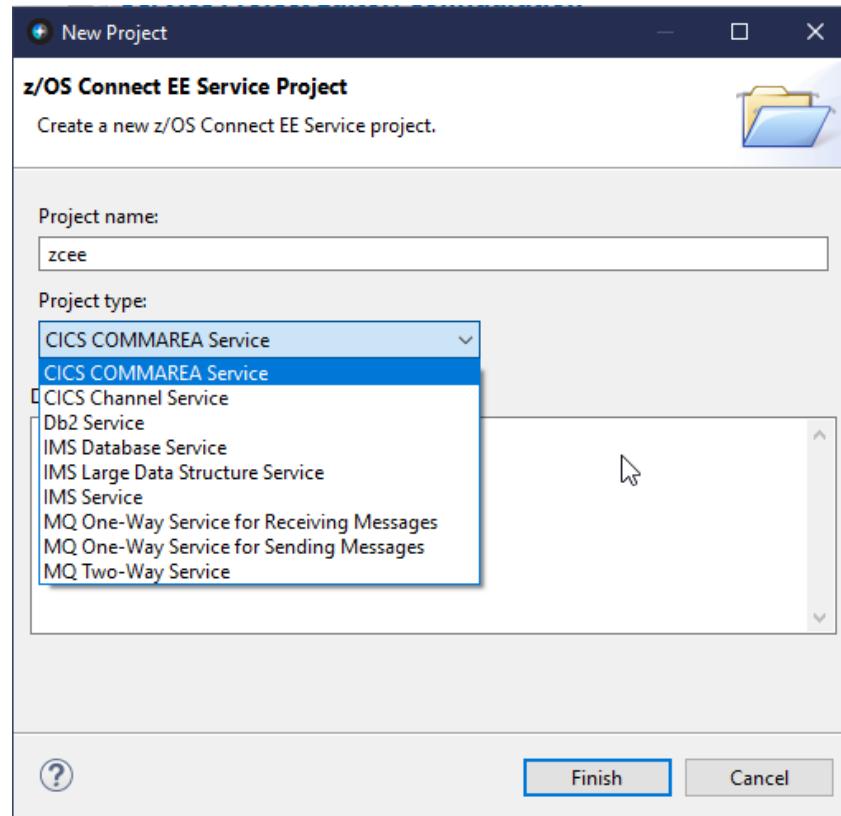


Use the **API toolkit** to create services through Eclipse-based tooling.

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

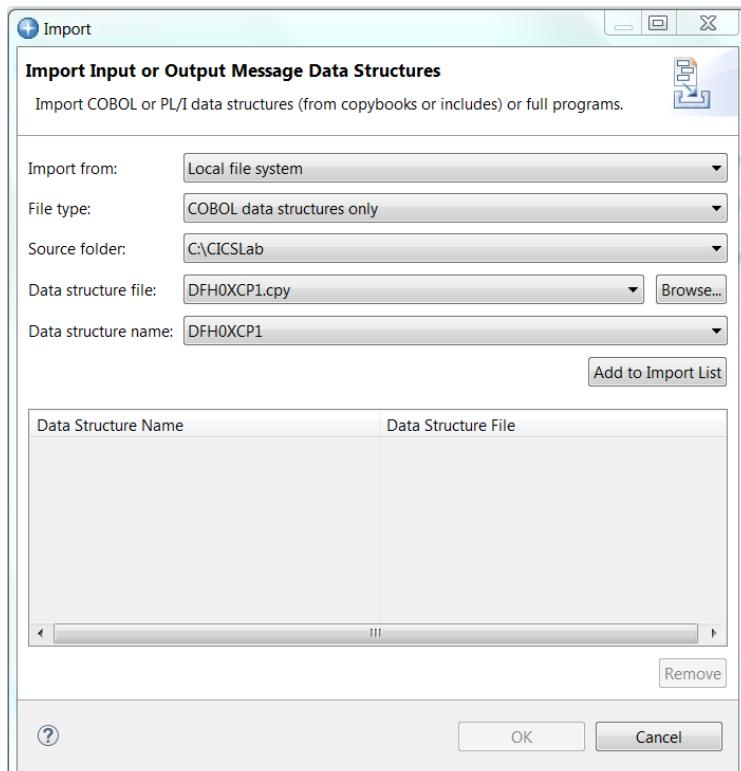
Service creation – a common interface



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

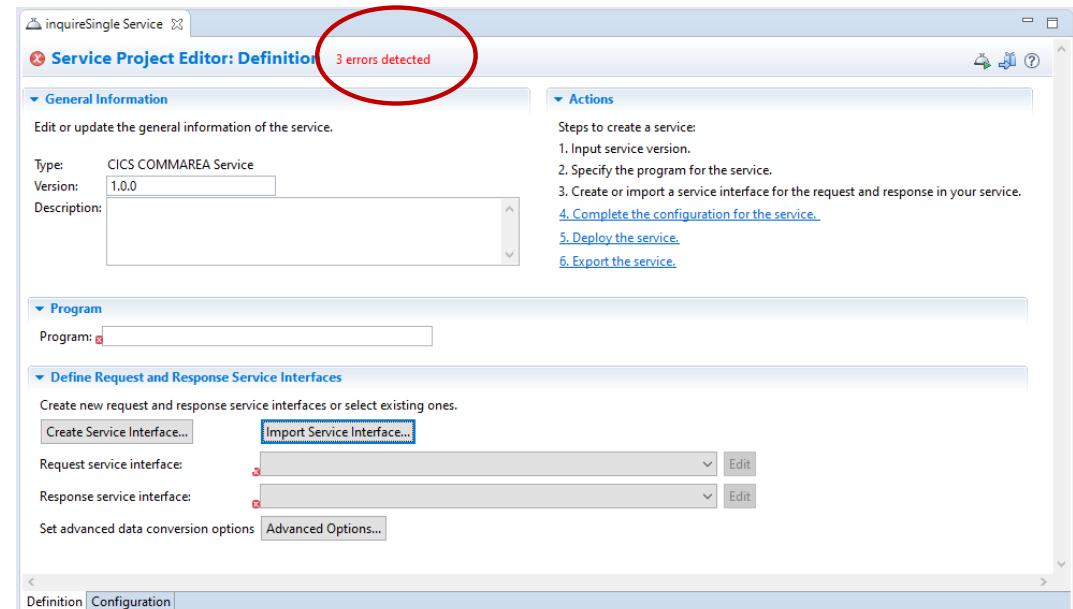
API toolkit – Creating Services for CICS, IMS TM and MQ

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 to create the request and response service interfaces.

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

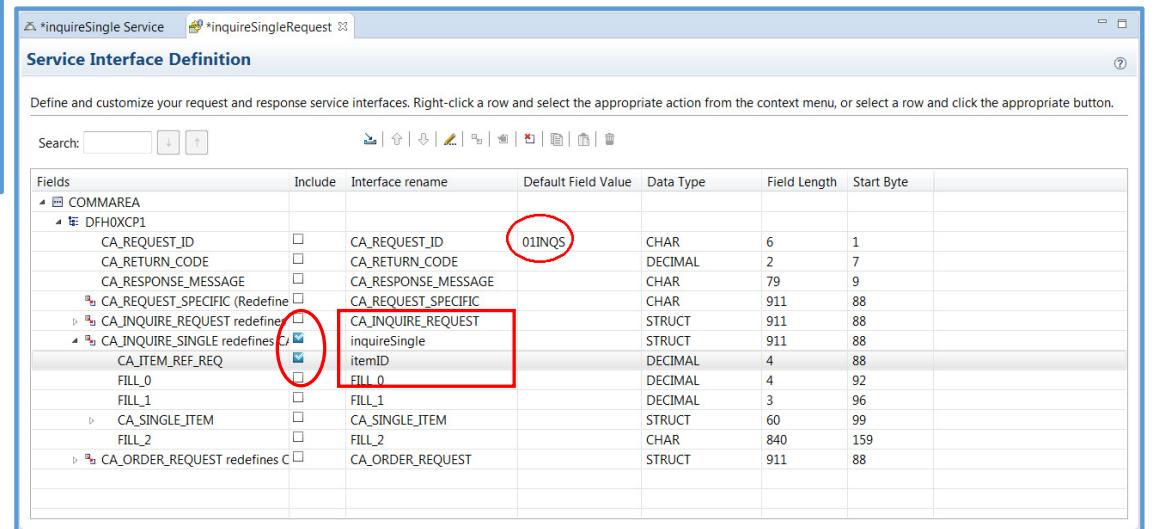


API toolkit – Creating Services for CICS, IMS TM and MQ

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 'Service Interface Definition' window with a table of fields. The table has columns: Fields, Include, Interface rename, Default Field Value, Data Type, Field Length, and Start Byte. A red circle highlights the 'Default Field Value' column for the CA_REQUEST_ID field, which is set to '01INQS'. A red box highlights the 'Interface rename' column for the CA_INQUIRE_REQUEST field, which is set to 'inquireSingle'. The table also lists other fields like 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 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 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: CSCVINC

Program: CSCVINC

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: cscvincSelectRequest.si Edit

Response service interface: cscvincSelectResponse.si Edit

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

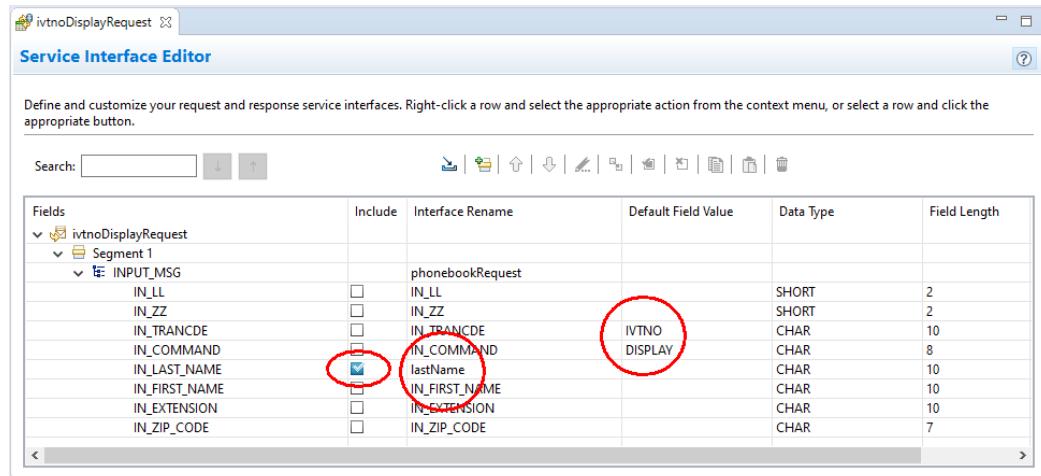
The service developer creates distinct services for each function by setting the ACTION field to S for select, I for insert, U for update or D for delete

© 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

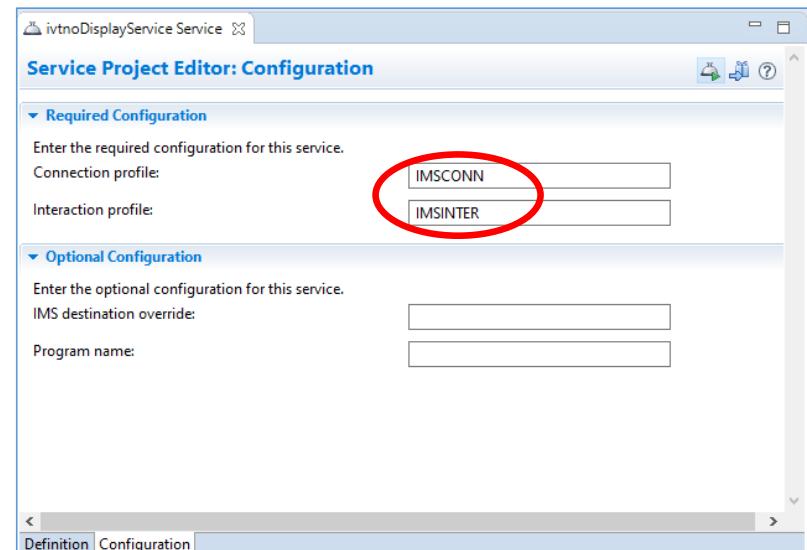
Define and customize your request and response service interfaces. Right-click a row and select the appropriate action from the context menu, or select a row and click the appropriate button.

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_TRANCODE	<input type="checkbox"/>	IN_TRANCODE		CHAR	10
IN_COMMAND	<input checked="" type="checkbox"/>	IN_COMMAND	IVTNO DISPLAY	CHAR	8
IN_LAST_NAME	<input type="checkbox"/>	lastName		CHAR	10
IN_FIRST_NAME	<input type="checkbox"/>	IN_FIRST_NAME		CHAR	10
IN_EXTENSION	<input type="checkbox"/>	IN_EXTENSION		CHAR	10
IN_ZIP_CODE	<input type="checkbox"/>	IN_ZIP_CODE		CHAR	7

mitchj@us.ibm.com

The service developer creates distinct services for each function.

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

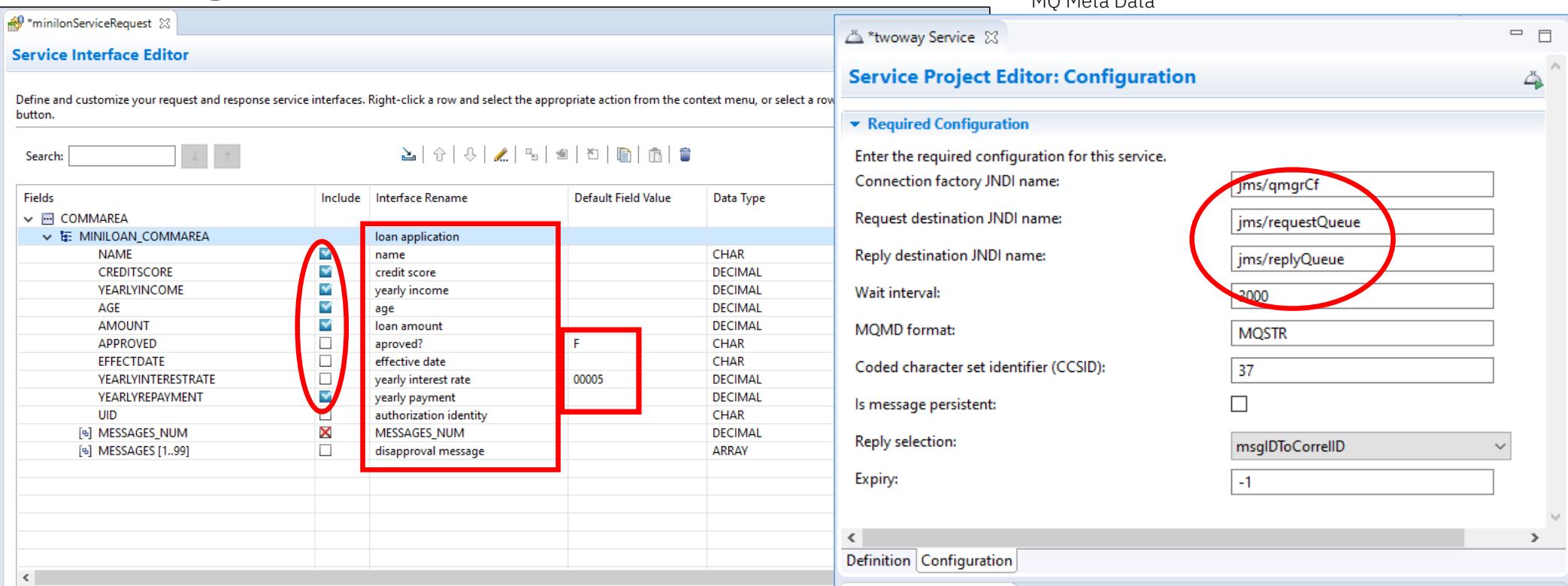


IMS/TM Meta Data

© 2018, 2021 IBM Corporation

API toolkit – Creating Services for MQ

Creating a “POST” service interface definition



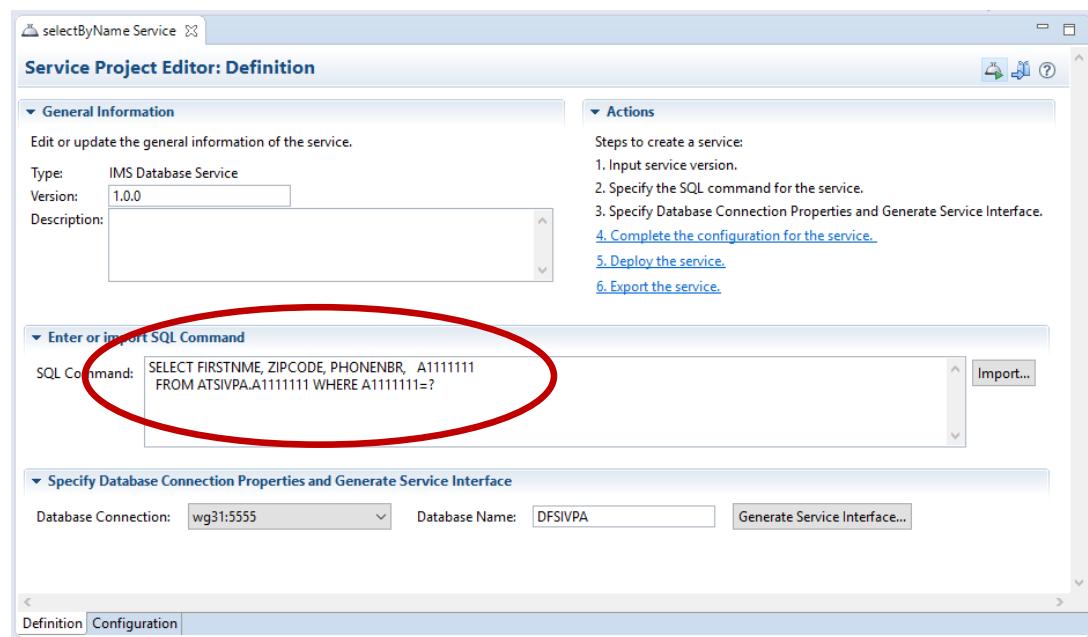
The screenshot shows two windows side-by-side:

- Service Interface Editor:** This window displays a table of fields for a service interface named "minilnServiceRequest". The columns are "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 "F". A red circle highlights the "Include" checkbox for the "MESSAGES_NUM" field.
- Service Project Editor: Configuration:** This window shows configuration settings for a service named "twoway Service". It includes sections for "Required Configuration" and "Optional Configuration". A red circle highlights the "jms/requestQueue" and "jms/replyQueue" fields, both of which have the value "jms/qmgrCf".

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

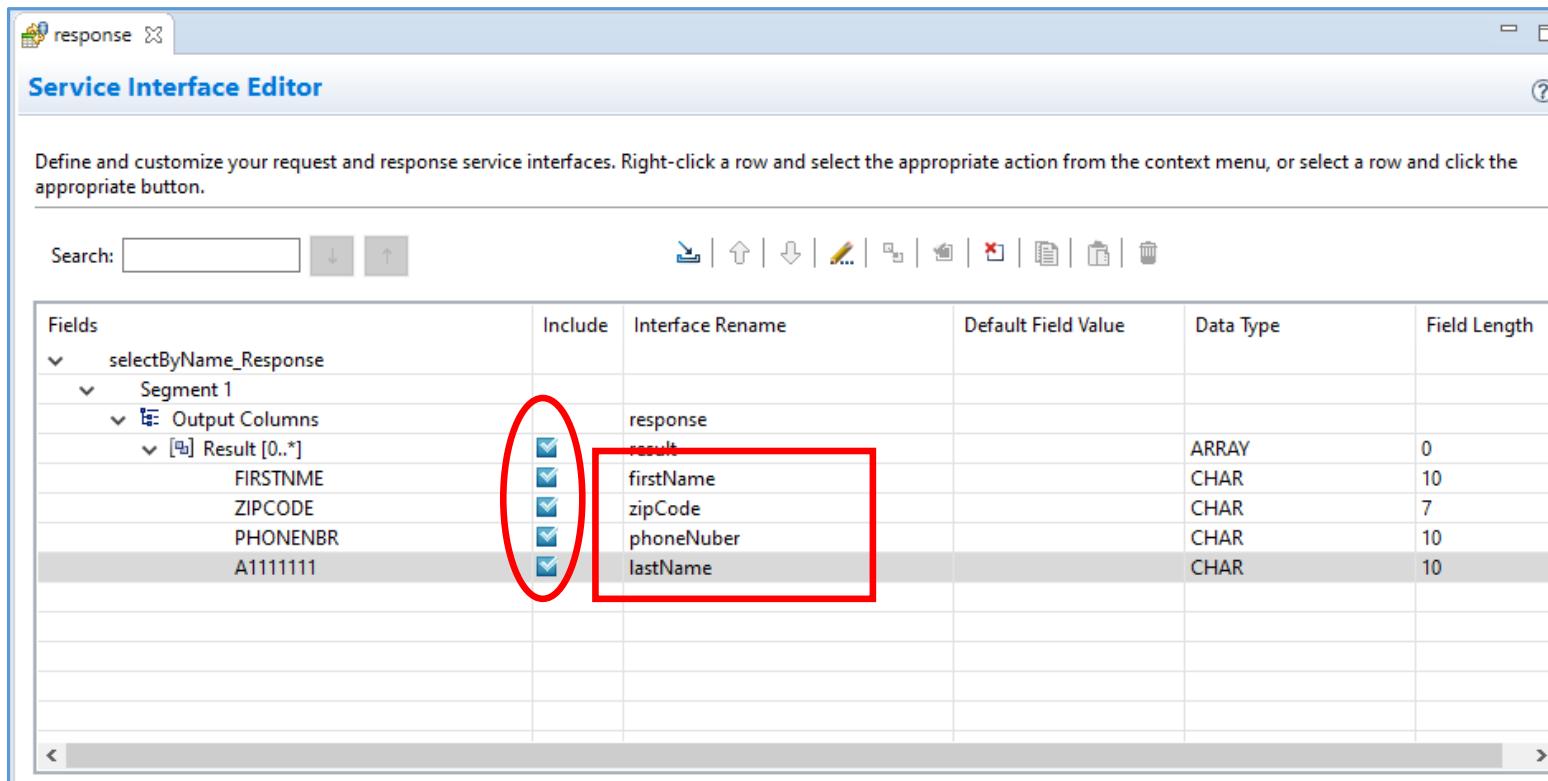


Use the IMS Catalog to assist with developing and testing SQL SELECT commands used for accessing IMS databases.

```
*-----*
* 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 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:

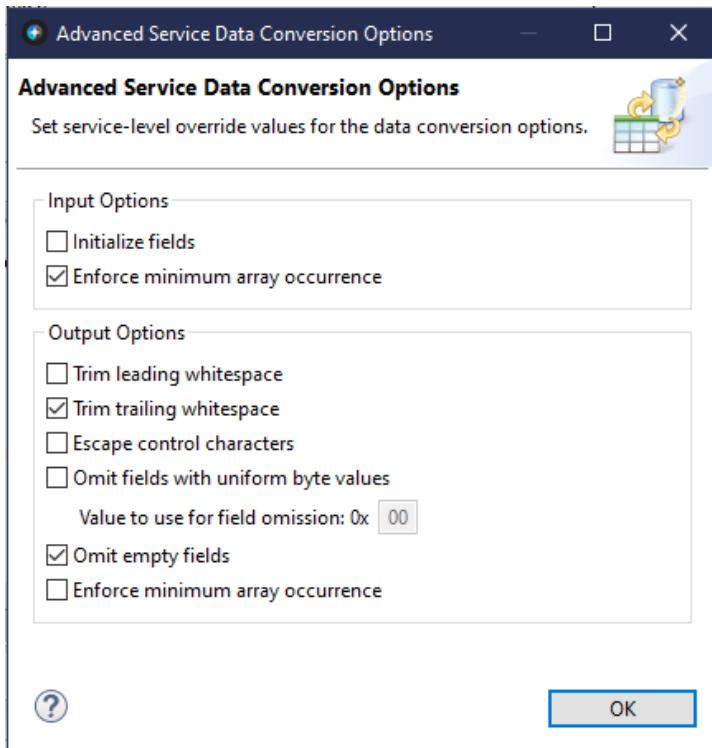
Fields	Include	Interface Rename	Default Field Value	Data Type	Field Length
selectByName_Response					
Segment 1					
Output Columns					
Result [0..*]	<input checked="" type="checkbox"/>	response			
FIRSTNAME	<input checked="" type="checkbox"/>	result		ARRAY	0
ZIPCODE	<input checked="" type="checkbox"/>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

*Using a slightly different process



z/OS Connect EE

API toolkit – Advanced Data Conversion Options



Request Messages:

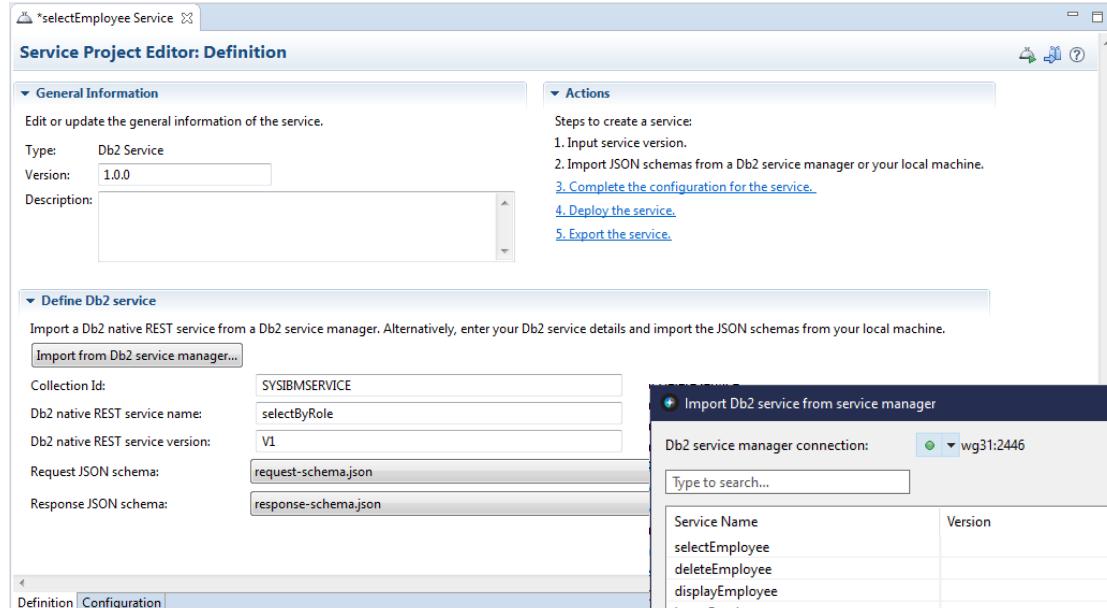
- Initialize fields
- Enforce minimum array occurrence

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

 *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...](#)

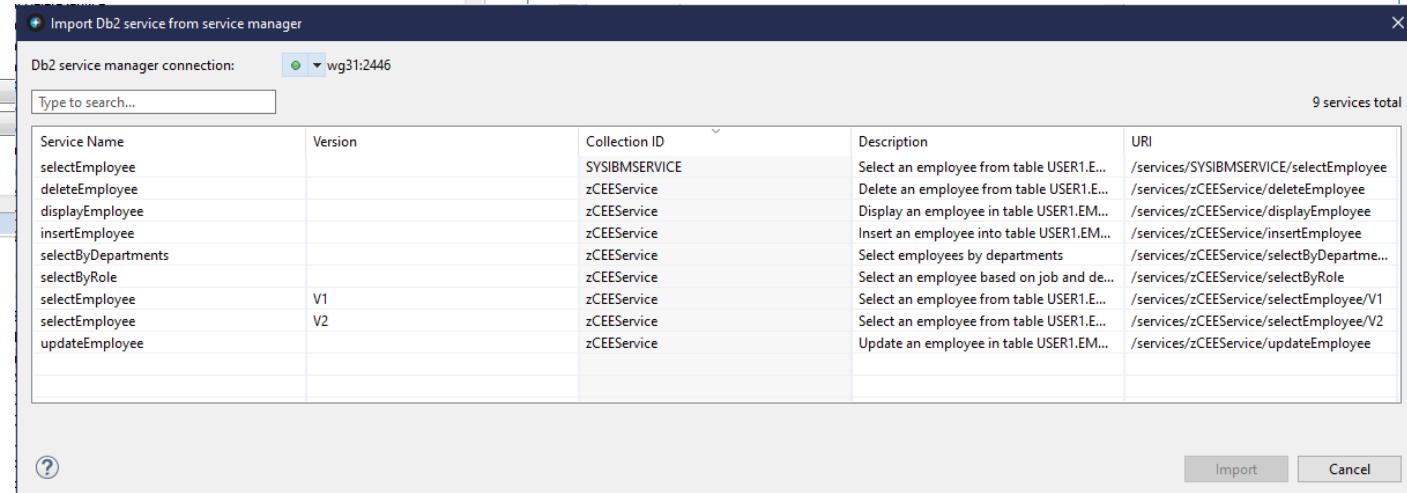
Collection Id: SYSIBMSERVICE
Db2 native REST service name: selectByRole
Db2 native REST service version: V1
Request JSON schema: request-schema.json
Response JSON schema: response-schema.json

[Definition](#) | [Configuration](#)

The service developer retrieves details about the Db2 REST services

Note there is no service interface editor available

mitchj@us.ibm.com



Import Db2 service from service manager

Db2 service manager connection: wg31:2446

Type to search... 9 services total

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.EM...	/services/zCEEService/displayEmployee
insertEmployee		zCEEService	Insert an employee into table USER1.EM...	/services/zCEEService/insertEmployee
selectByDepartments		zCEEService	Select employees by departments	/services/zCEEService/selectByDepartments
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

Import Cancel

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 single project named 'InquireSingle'. The main window displays the 'Overview' tab of the 'InquireSingle Service' configuration. Under 'General Information', the service is defined as a 'CICS COMMAREA Service' with version 1.0.0. The 'Program' field is set to '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...'. The right side of the interface shows tabs for 'Overview' and 'Configuration', along with 'Host Connections', 'Properties', 'Progress', and 'Problems'. A separate 'Deploy Service' dialog box is overlaid on the main interface. 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: 'inquireSingle' with 'Version 13.00' and 'Type CICS COMMAREA Se...'. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

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

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

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

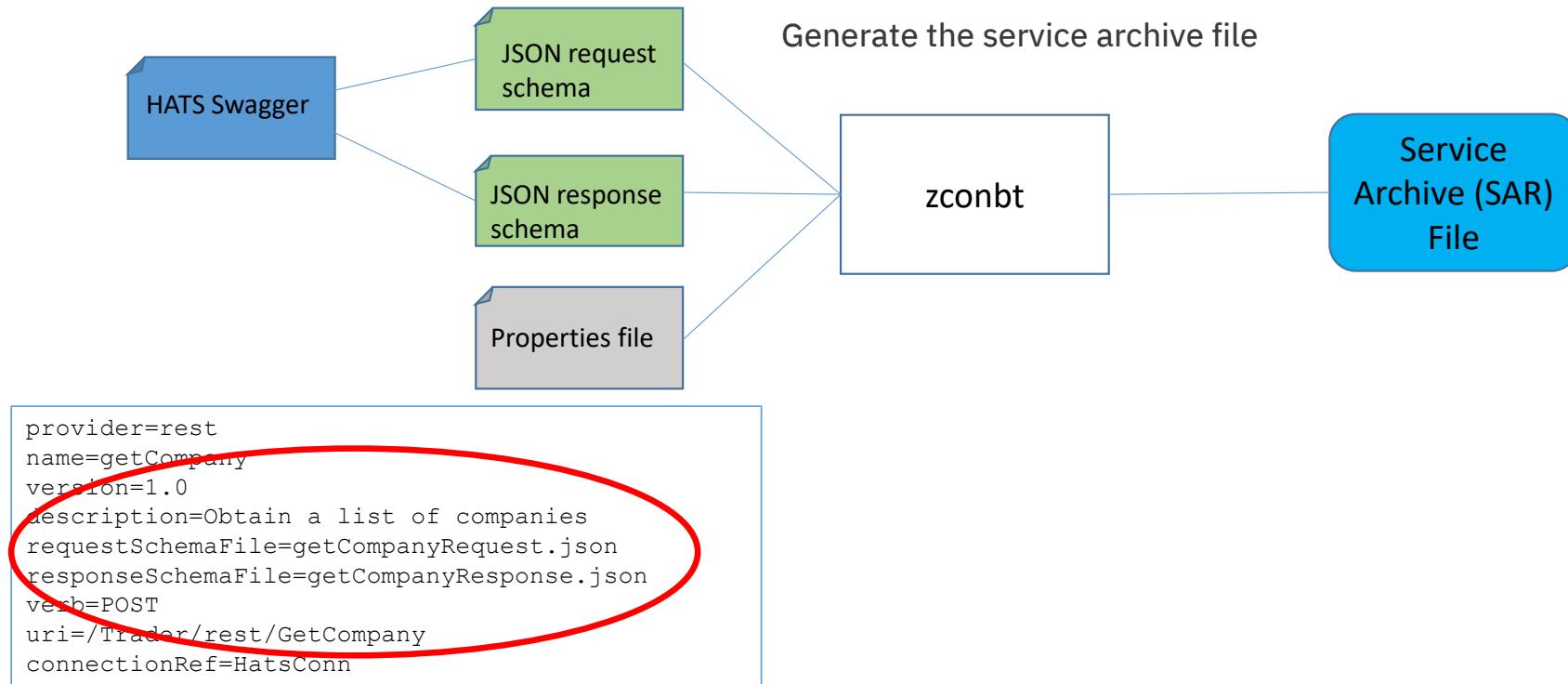


The screenshot shows the IBM z/OS Connect Enterprise Edition interface. On the left, the Project Explorer displays a service named 'InquireSingle'. The main window shows the 'Overview' tab of the 'InquireSingle Service' properties. Under 'General Information', the 'Type' is set to 'CICS COMMAREA Service' and 'Version' 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 '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 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.

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

Creating Services using zconbt – REST

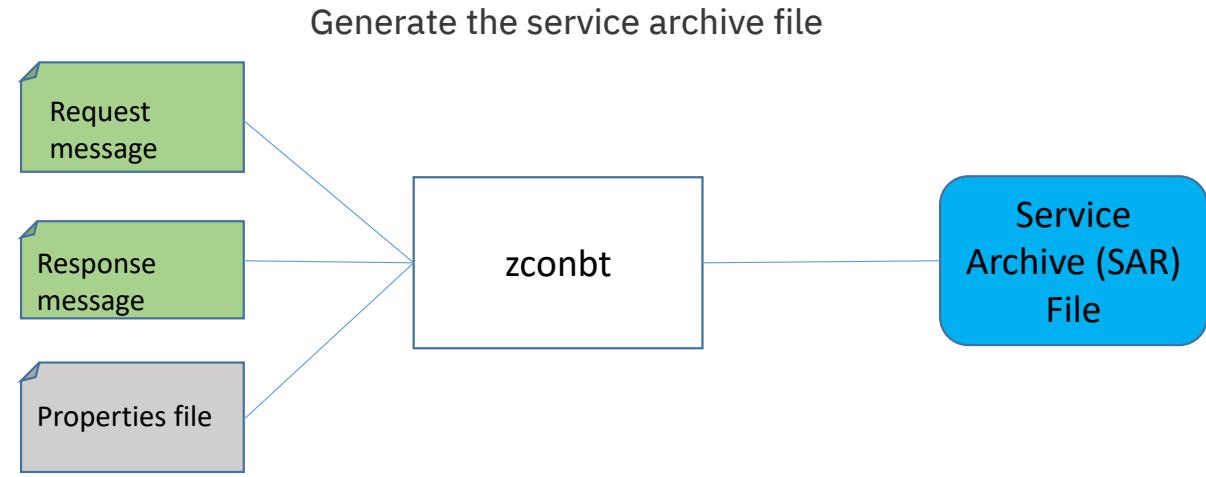
For HATS REST Services use the z/OS Connect Build toolkit (zconbt)



Creating Services using zconbt – MVS Batch

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

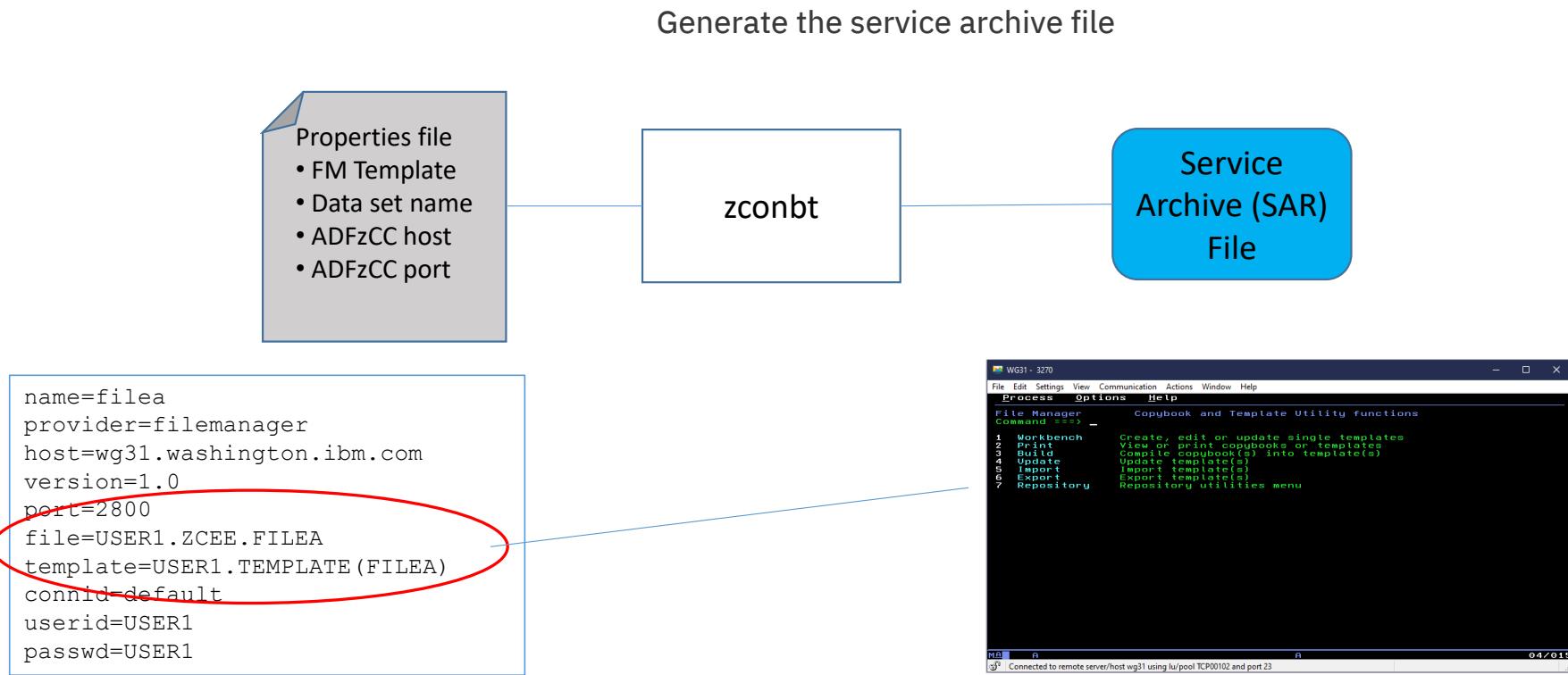
```
name=Filea
version=1.0
provider=wola
description=COBOL batch program
language=COBOL
program=ATSFILEA
register=FILEAZCON
connectionRef=wolaCF
requestStructure=fileareq.cpy
responseStructure=filearsp.cpy
```



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

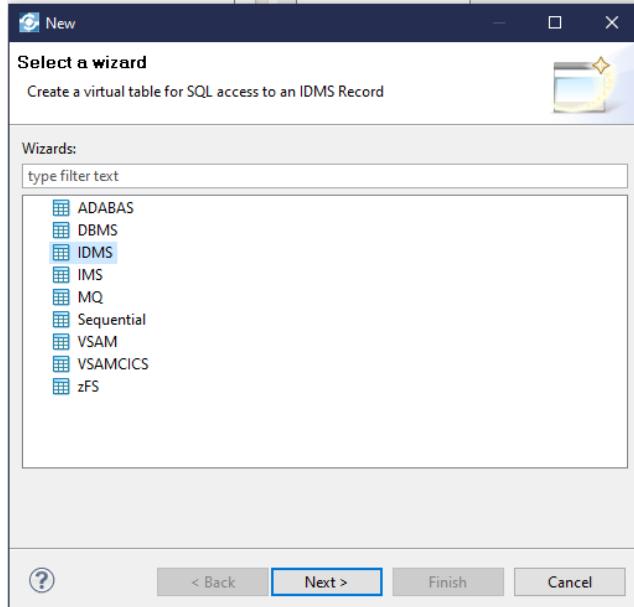
Creating Services using zconbt – File Manager

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. At the bottom is a table viewer showing data for columns WS_DESCRIPTION, WS_DEPARTMENT, and WS_C, with rows for Mitch Johnson (values 10, 002, 002) and a summary row (values 10, 002, 002). A 'Server Trace' and 'Console' tab are also visible.



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

mitchj@us.ibm.com

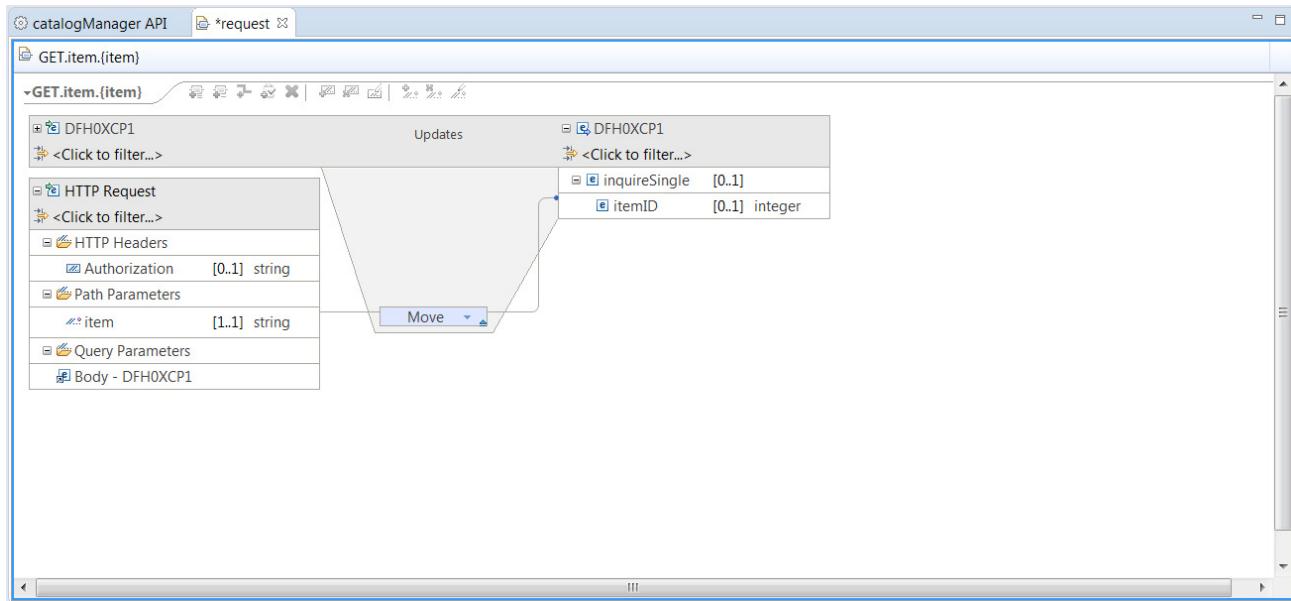
The **API toolkit** is designed to encourage RESTful API design.

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.

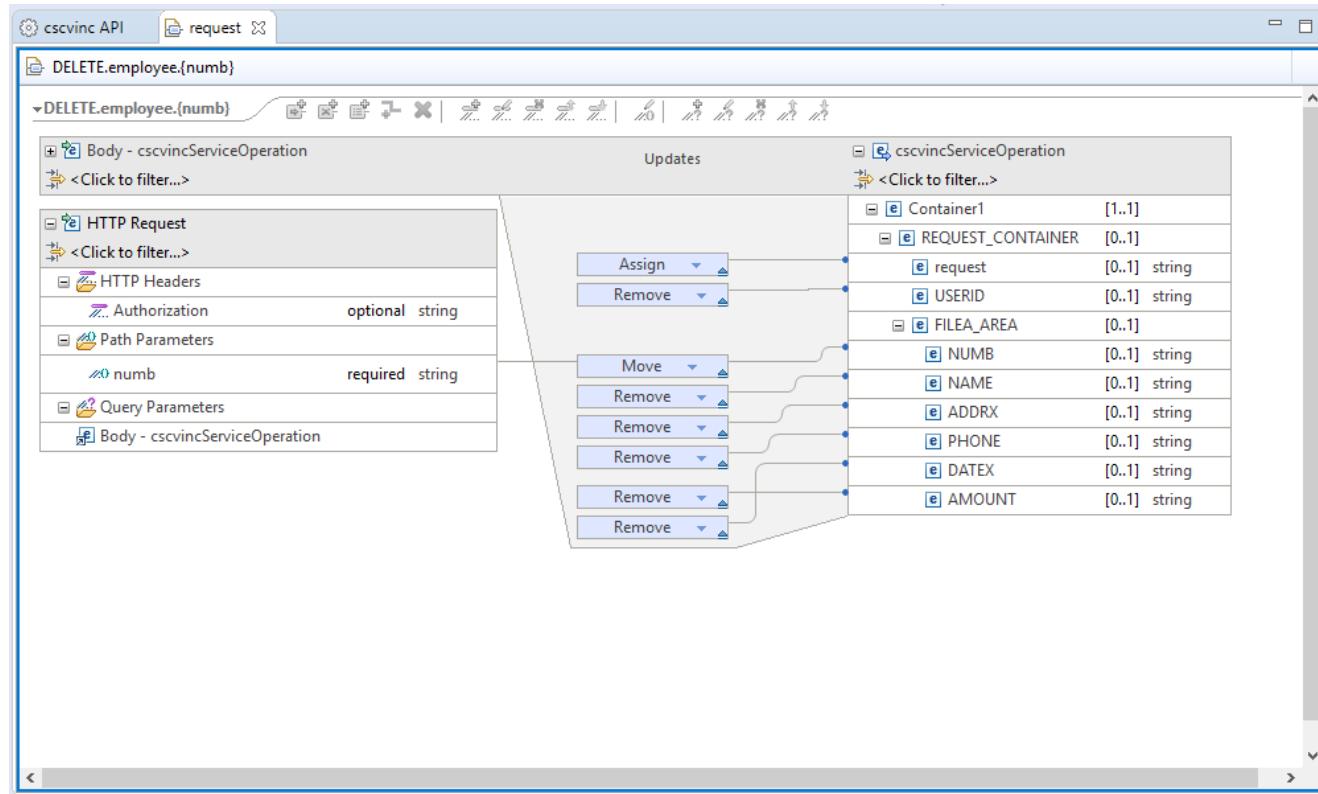
Map path and query parameters to native data structures.

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).

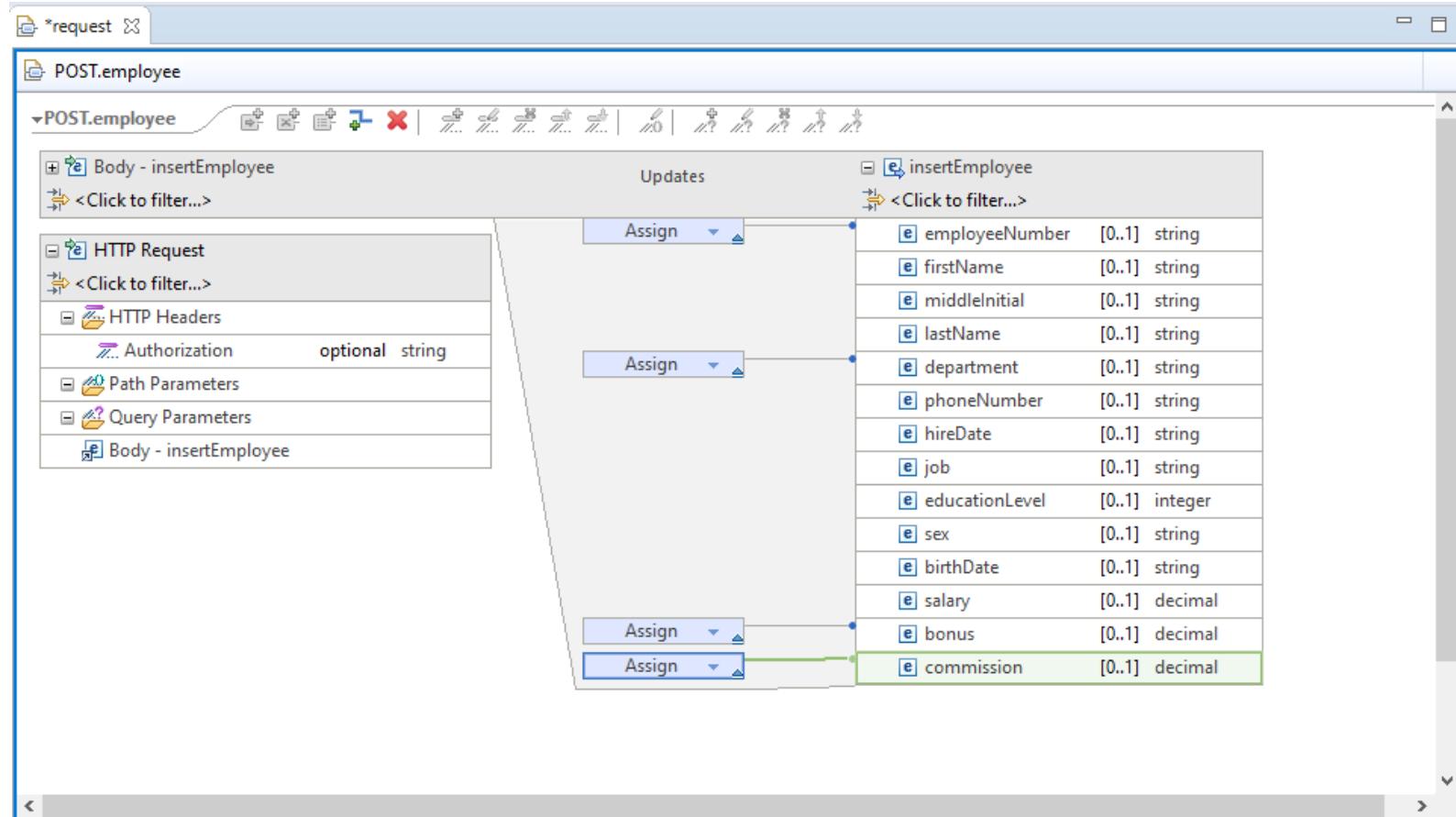
API toolkit – API Editor

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



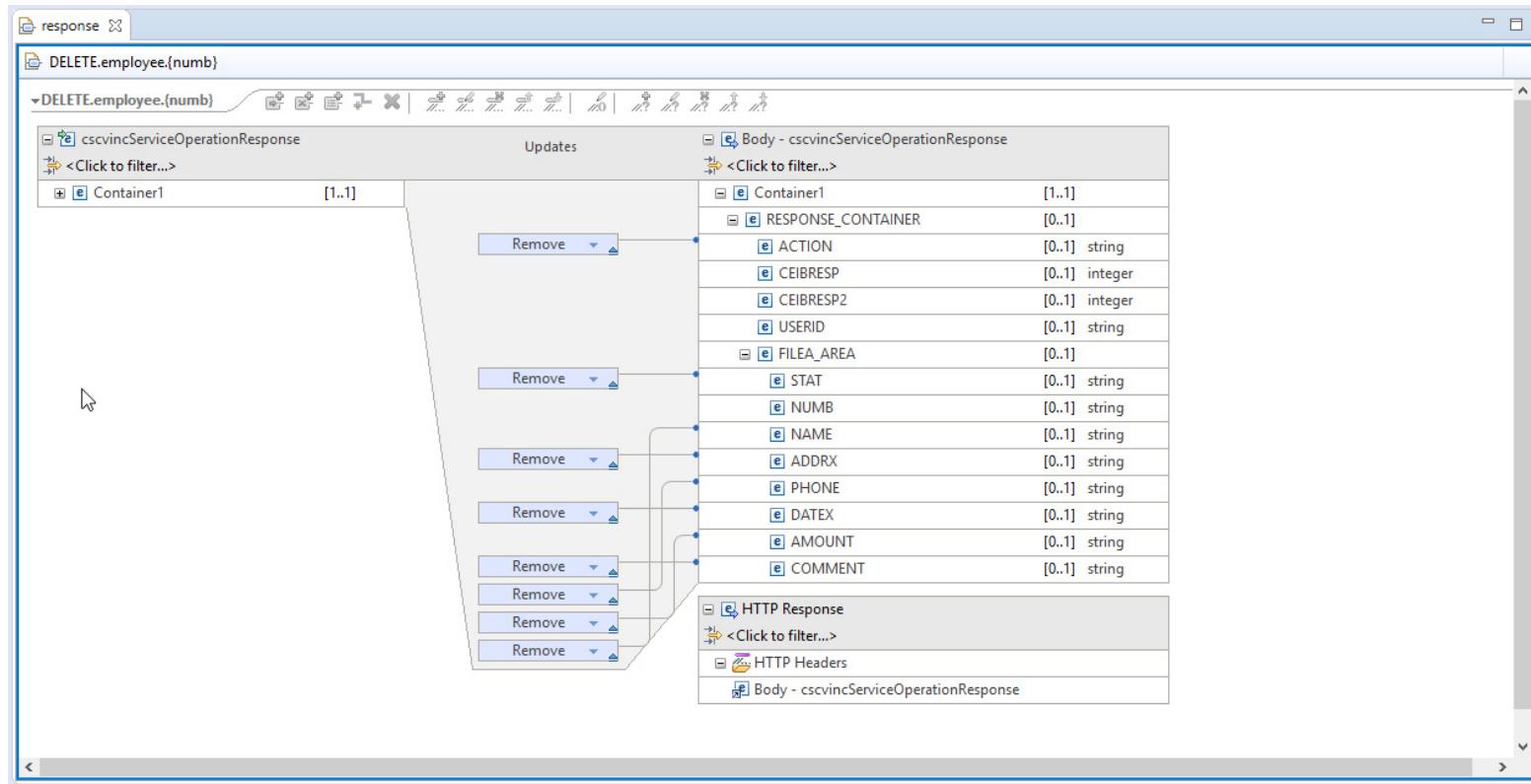
API toolkit – API Editor and Db2 REST service

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



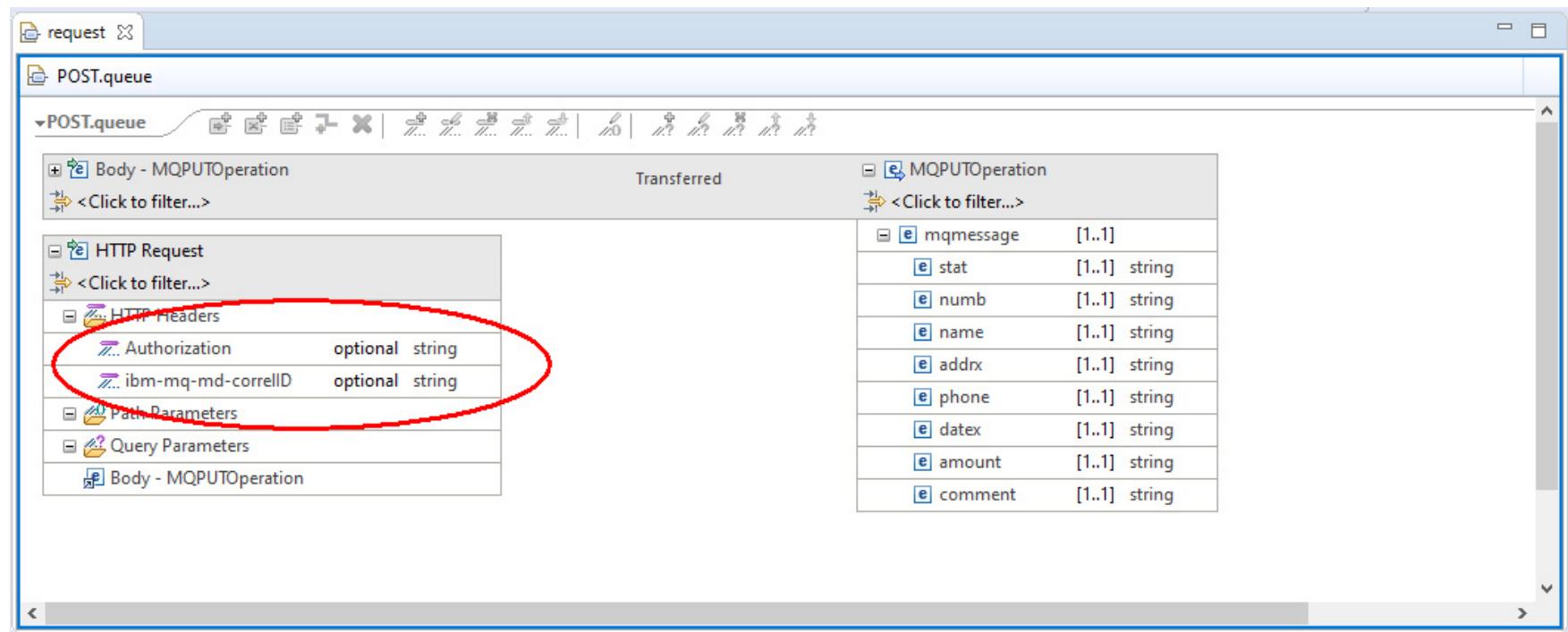
API toolkit – API Editor

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):

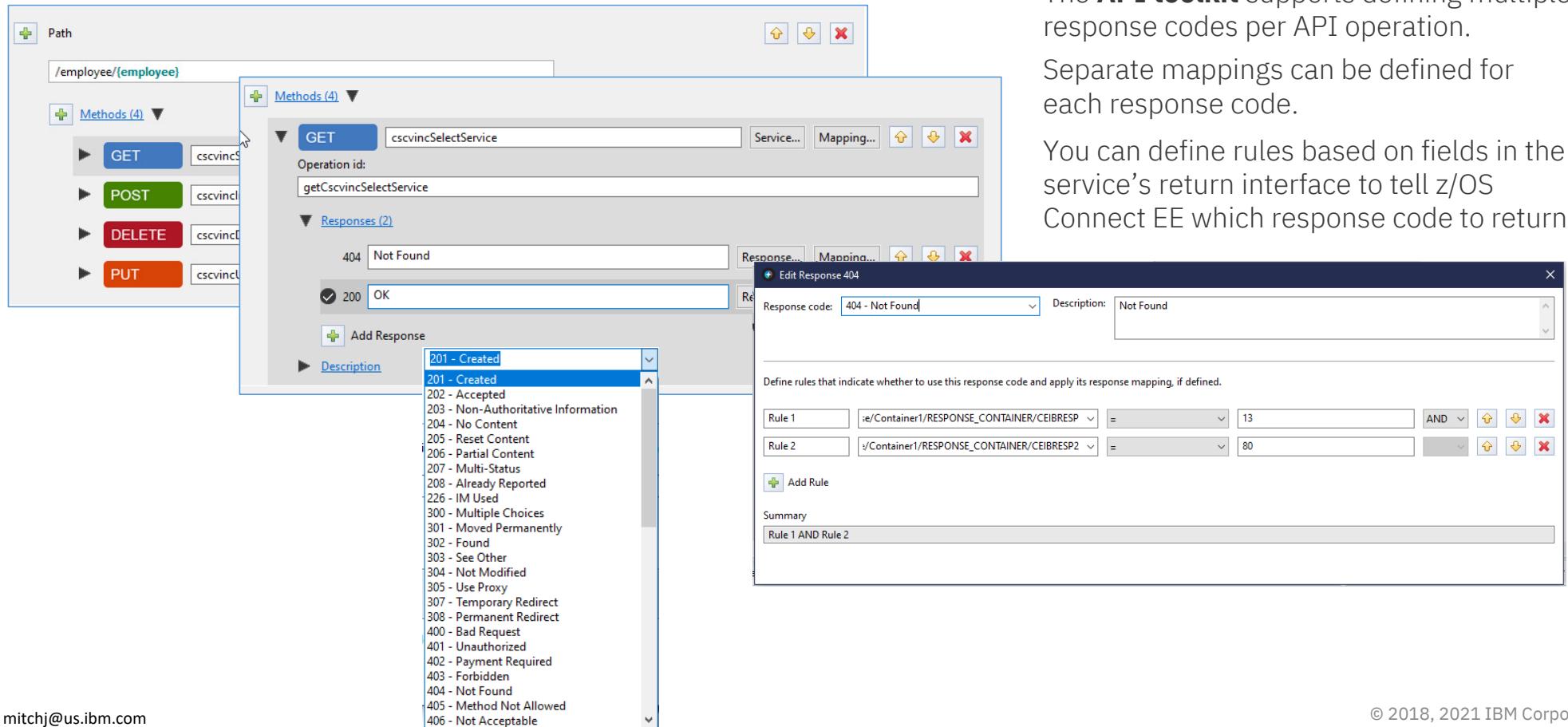
- Body - MQPUTOperation:** Contains a link to 'Click to filter...'.
- HTTP Request:** Contains a link to 'Click to filter...'.
 - HTTP Headers:** Contains:
 - Authorization (optional string)
 - ibm-mq-md-correlID (optional string)
 - Path Parameters:**
 - Query Parameters:**
 - Body - MQPUTOperation:**

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 is set to `/employee/{employee}`. The main area displays the `Methods (4)` section for a `GET` operation named `cscvincSelectService`. Under the `Responses (2)` section, two responses are defined: `404 Not Found` and `200 OK`. A context menu is open over the `200 OK` response, showing the `Edit Response 404` option. This menu includes fields for `Response code` (`404 - Not Found`) and `Description` (`Not Found`). Below these fields is a section for defining rules: `Define rules that indicate whether to use this response code and apply its response mapping, if defined.`. Two rules are listed: `Rule 1` and `Rule 2`. Rule 1 is based on the expression `se/Container1/RESPONSE_CONTAINER/CEIBRESP` being equal to `13`, and Rule 2 is based on the expression `sl/Container1/RESPONSE_CONTAINER/CEIBRESP2` being equal to `80`. The overall summary is `Rule 1 AND Rule 2`.

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

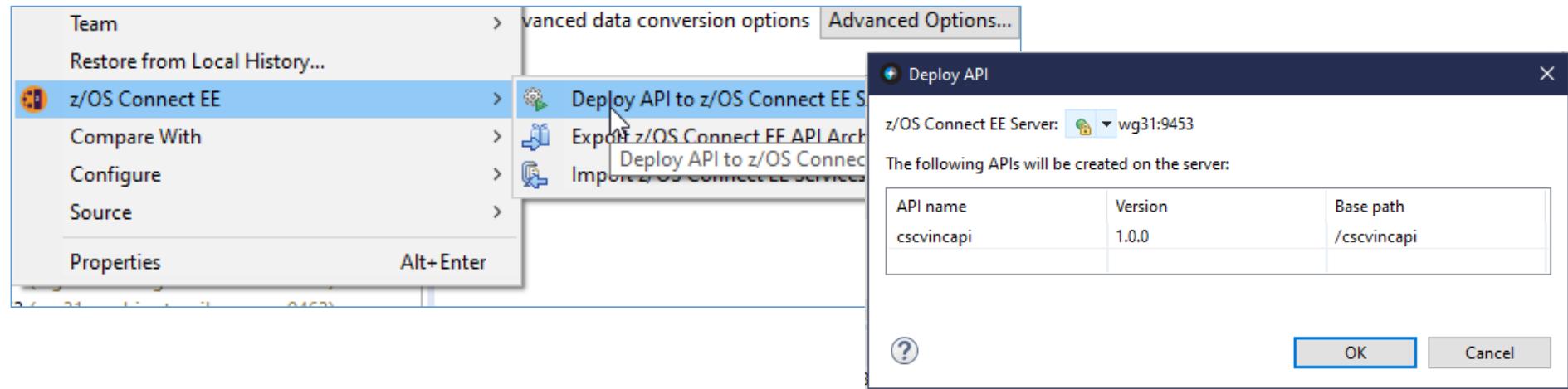
Separate mappings can be defined for each response code.

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:

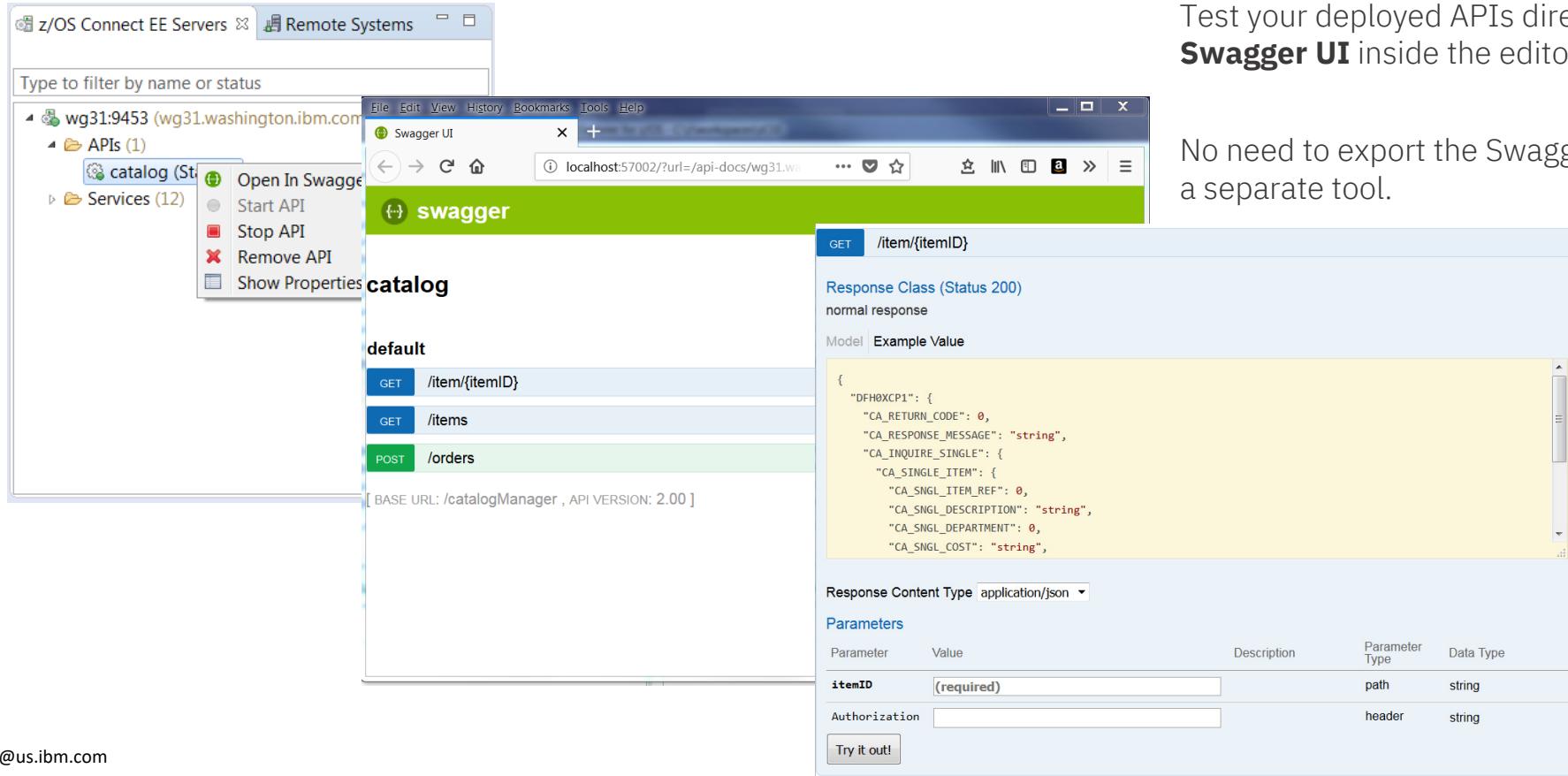


Right-click deploy to server enables developers to quickly deploy, test, and iterate on their APIs.

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



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 and 12 services. A context menu for the API is open, showing options like 'Open In Swagger UI'. 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. The right side of the screen shows the detailed view for the 'GET /item/{itemID}' operation. It includes a 'Model' tab with a JSON schema example:

```
{  
  "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  
      }  
    }  
}
```

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

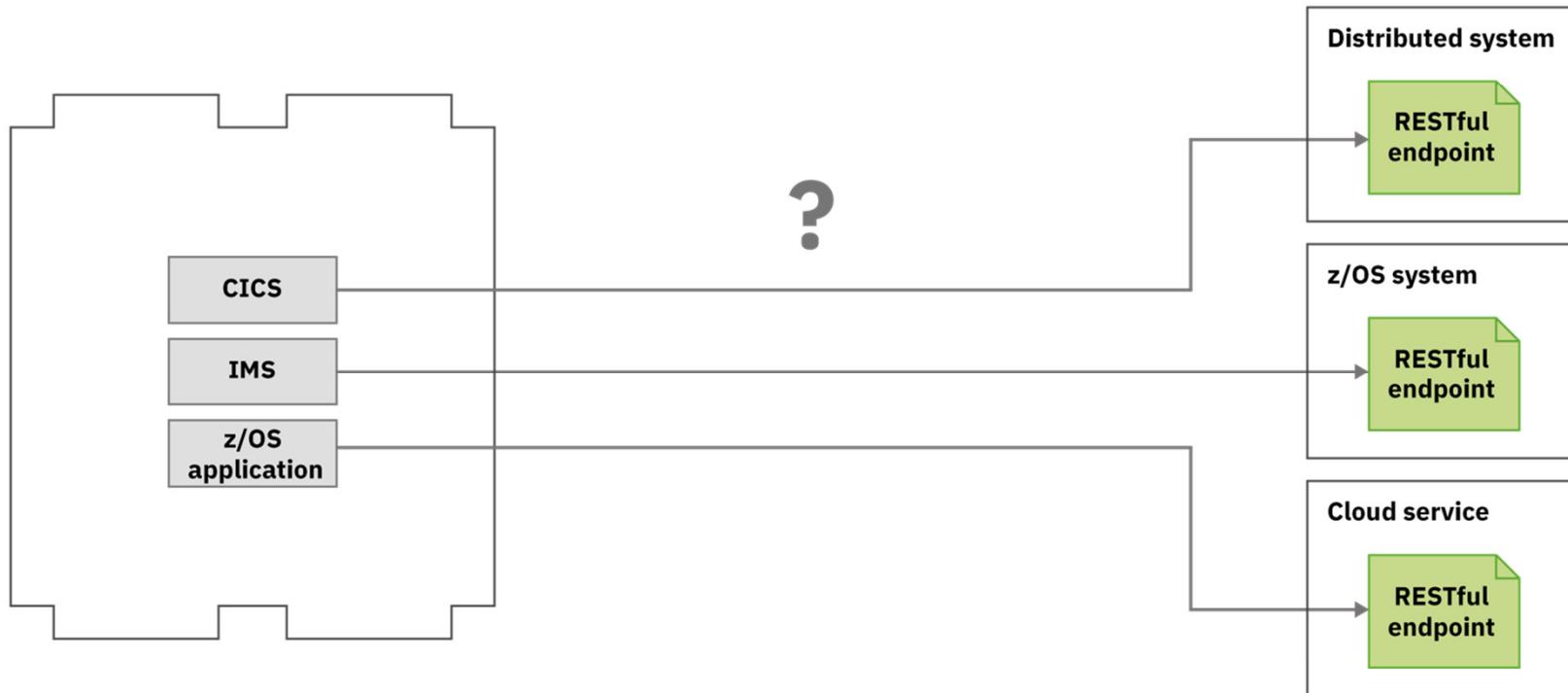
Test your deployed APIs directly with **Swagger UI** inside the editor.
No need to export the Swagger doc to a separate tool.



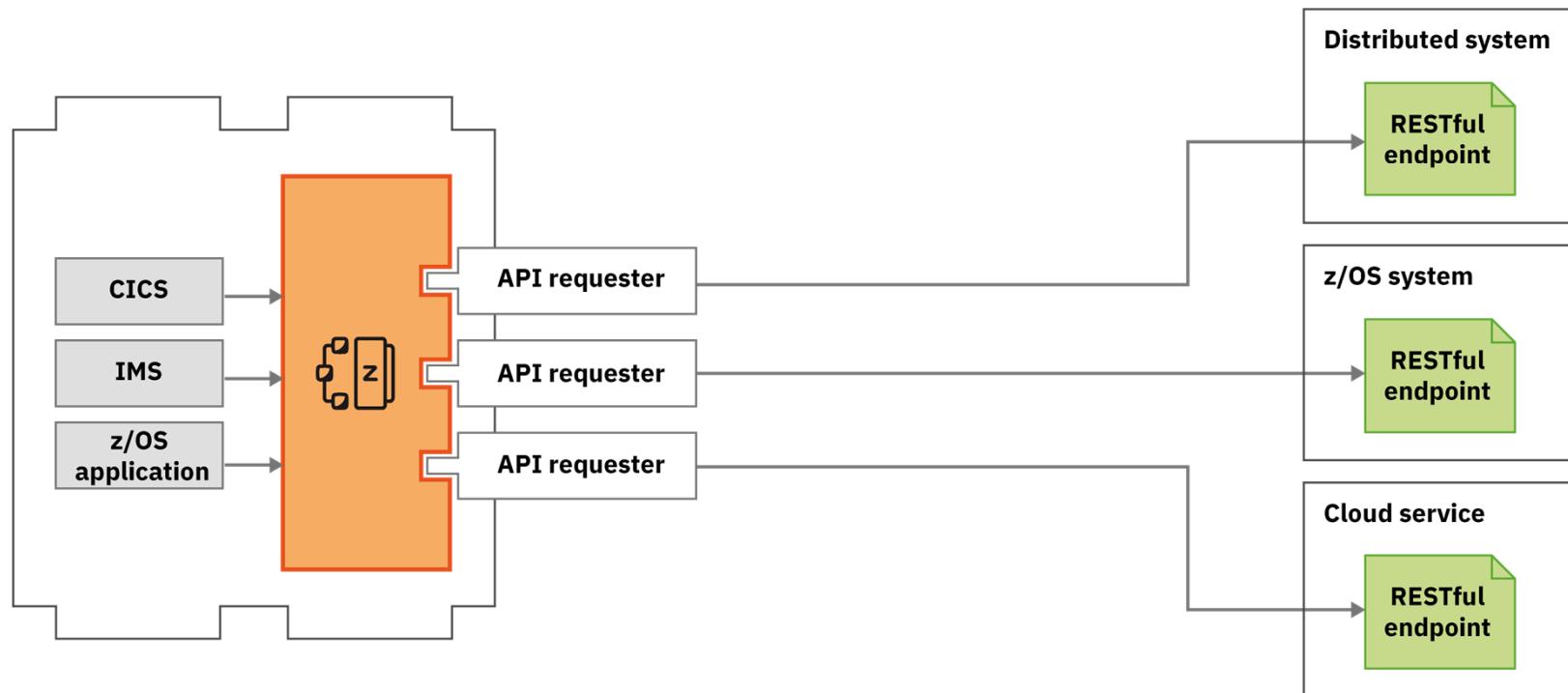
/api_toolkit/apiRequesters

Quick and easy **API mapping**.

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

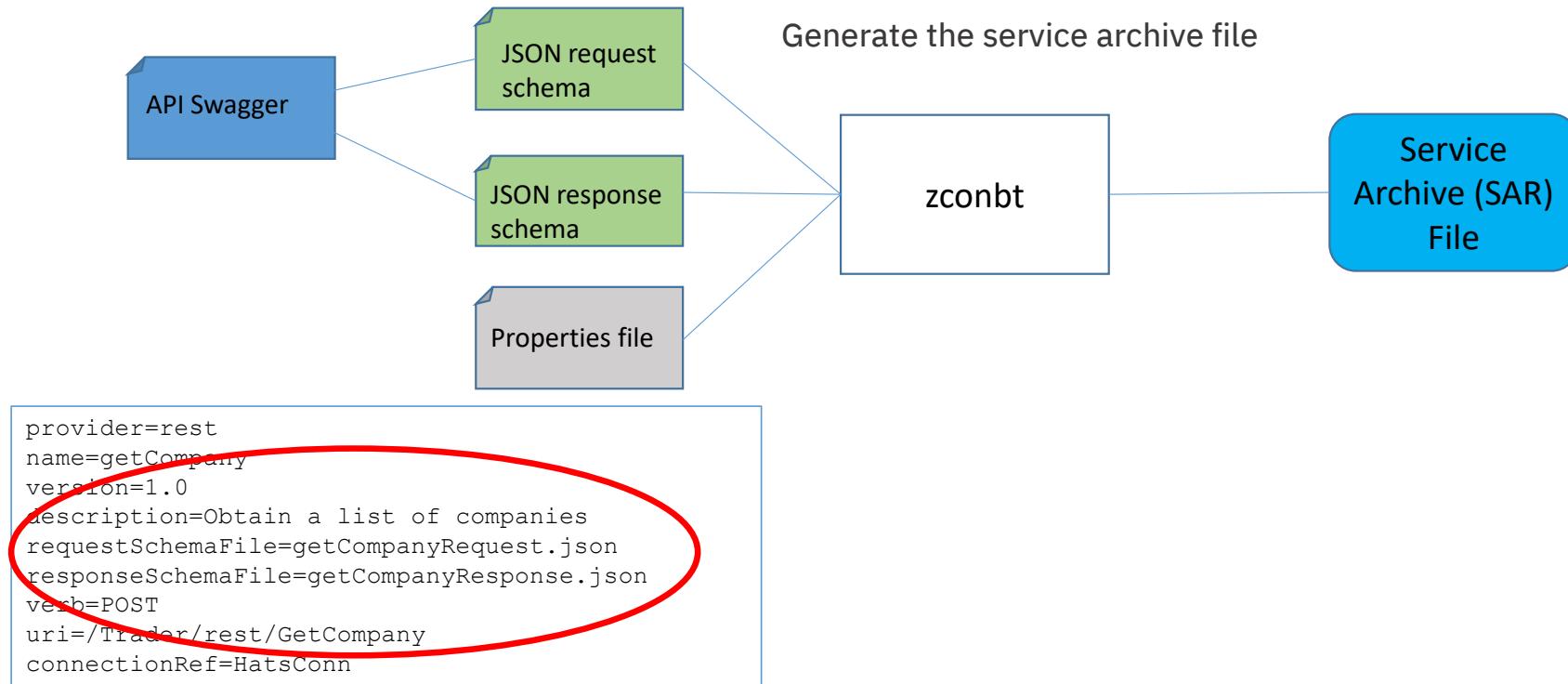


Use API requester to call external APIs from z/OS assets



Creating Services using zconbt – REST

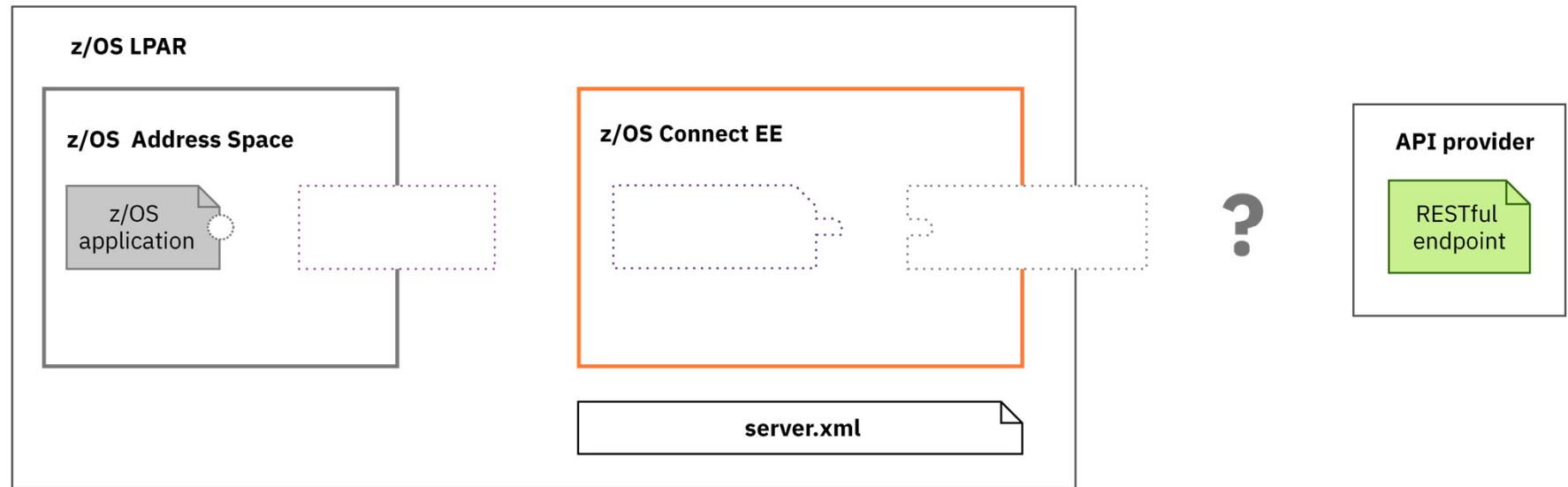
For HATS REST Services use the z/OS Connect Build toolkit (zconbt)





Steps to calling an external API

Starting point

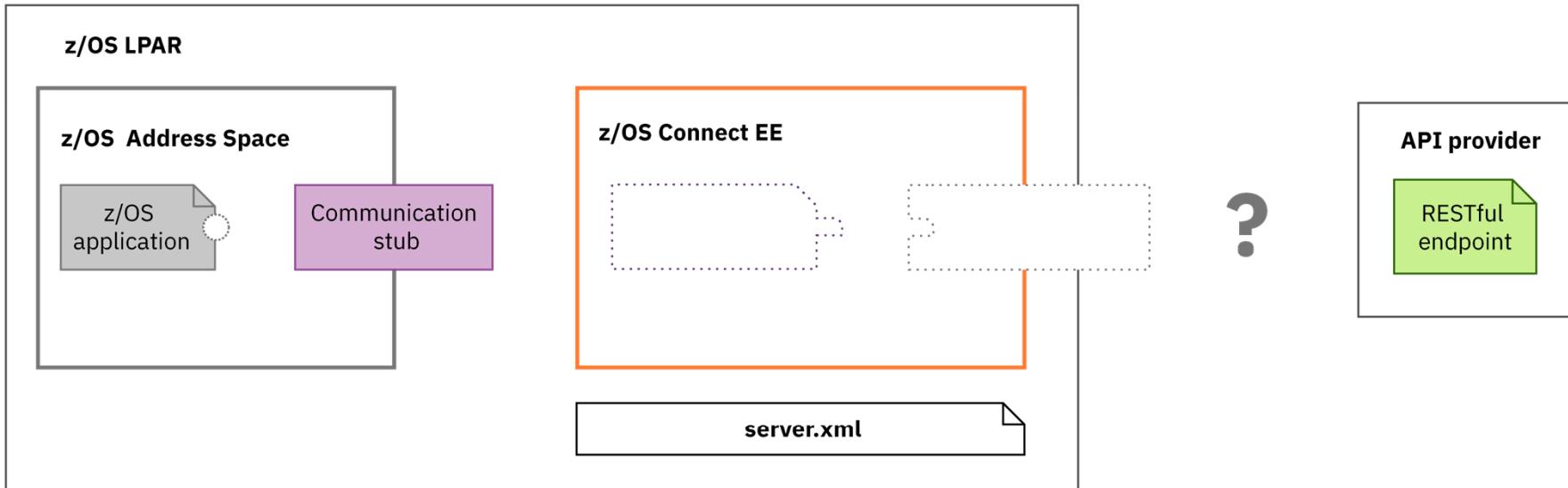




z/OS Connect EE

Steps to calling an external API

Step 1. Configure communication stub



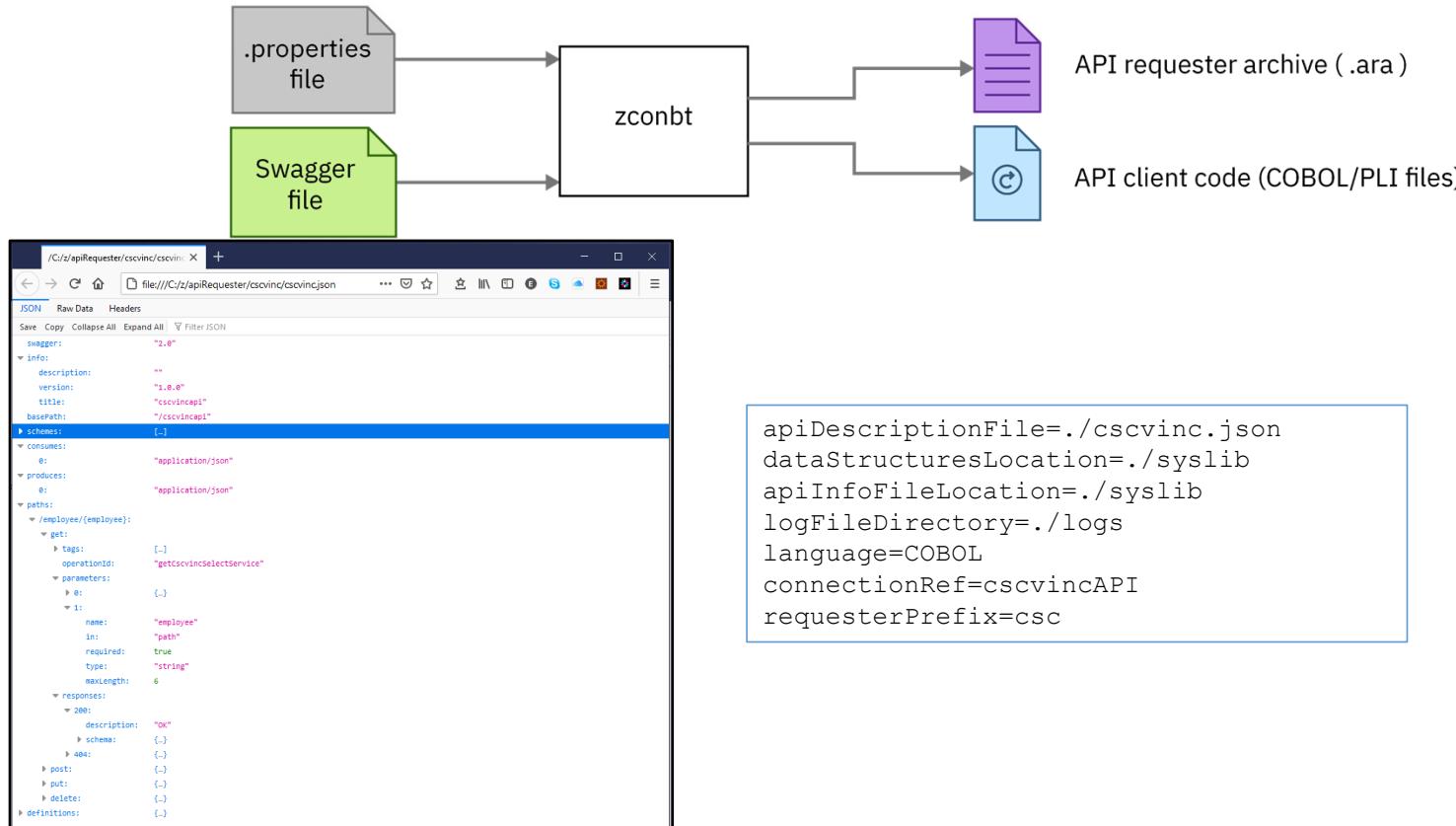
Configure a communication stub.

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

 ibm.biz/zosconnect-configure-comms-stub

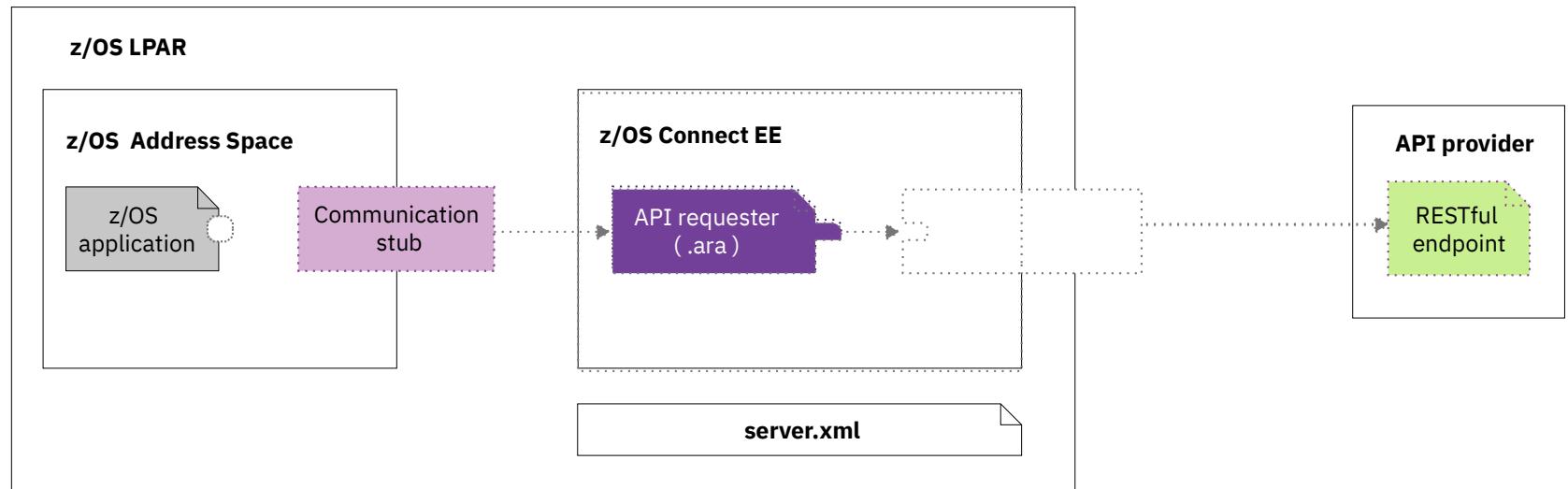
Steps to calling an external API

Step 2. Generate API requester archive and API client code from Swagger



Steps to calling an external API

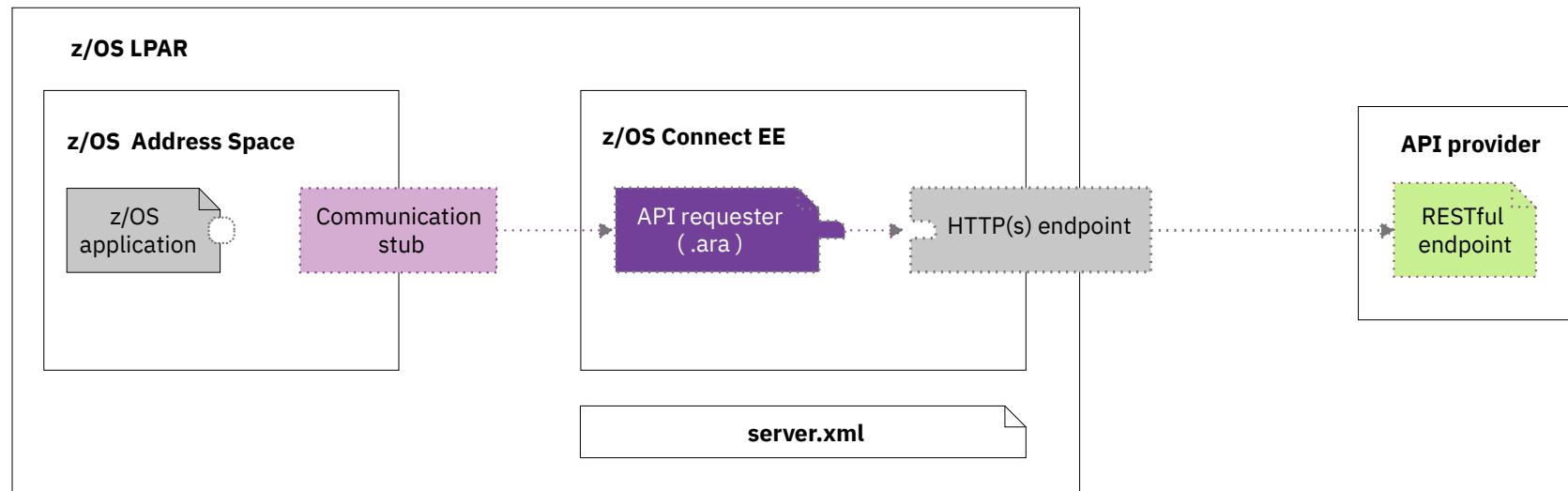
Step 3. Deploy API requester (.ara) archive



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

Steps to calling an external API

Step 4. Configure HTTP(S) endpoint

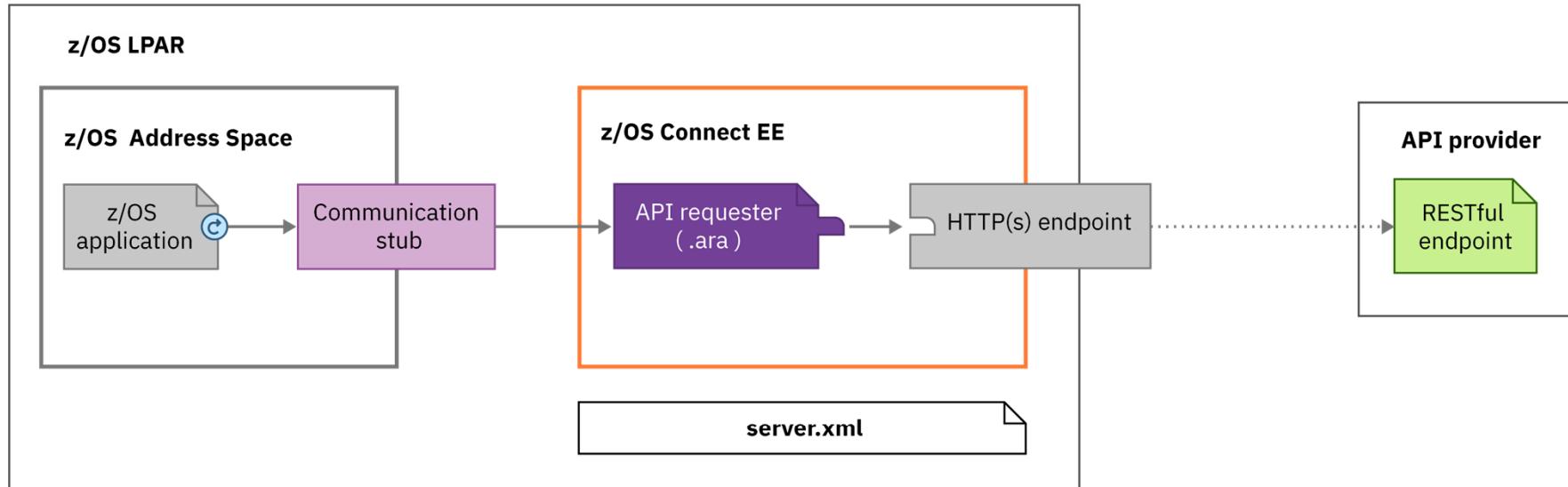


Configure the connection between z/OS Connect EE and the external API.

 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 the Rational Application Developer interface with three windows:

- GETAPI**: Shows the `ERROR MESSAGE STRUCTURE` with fields:
 - 01 ERROR-MSG.
 - 03 EM-ORIGIN PIC X(8) VALUE SPACES.
 - 03 EM-CODE PIC S9(9) COMP-5 SYNC VALUE 0.
 - 03 EM-DETAIL PIC X(1024) VALUE SPACES.
- apis.xml**: Shows XML configuration for an API requester:

```
<server description="API Requester">
  <!-- Enable features -->
  <featureManager>
    <feature>zosconnect:apiRequester-1.0</feature>
  </featureManager>

  <zosconnect_apiRequesters location="">
    <zosconnect_apiRequester name="cscvinc_1.0.0"/>
  </zosconnect_apiRequesters>

  <zosconnect_endpointConnection id="cscvincAPI">
    host="http://wg31.washington.ibm.com"
    port="9120"
    basicAuthRef="myBasicAuth"
    connectionTimeout="10s"
    receiveTimeout="20s" />

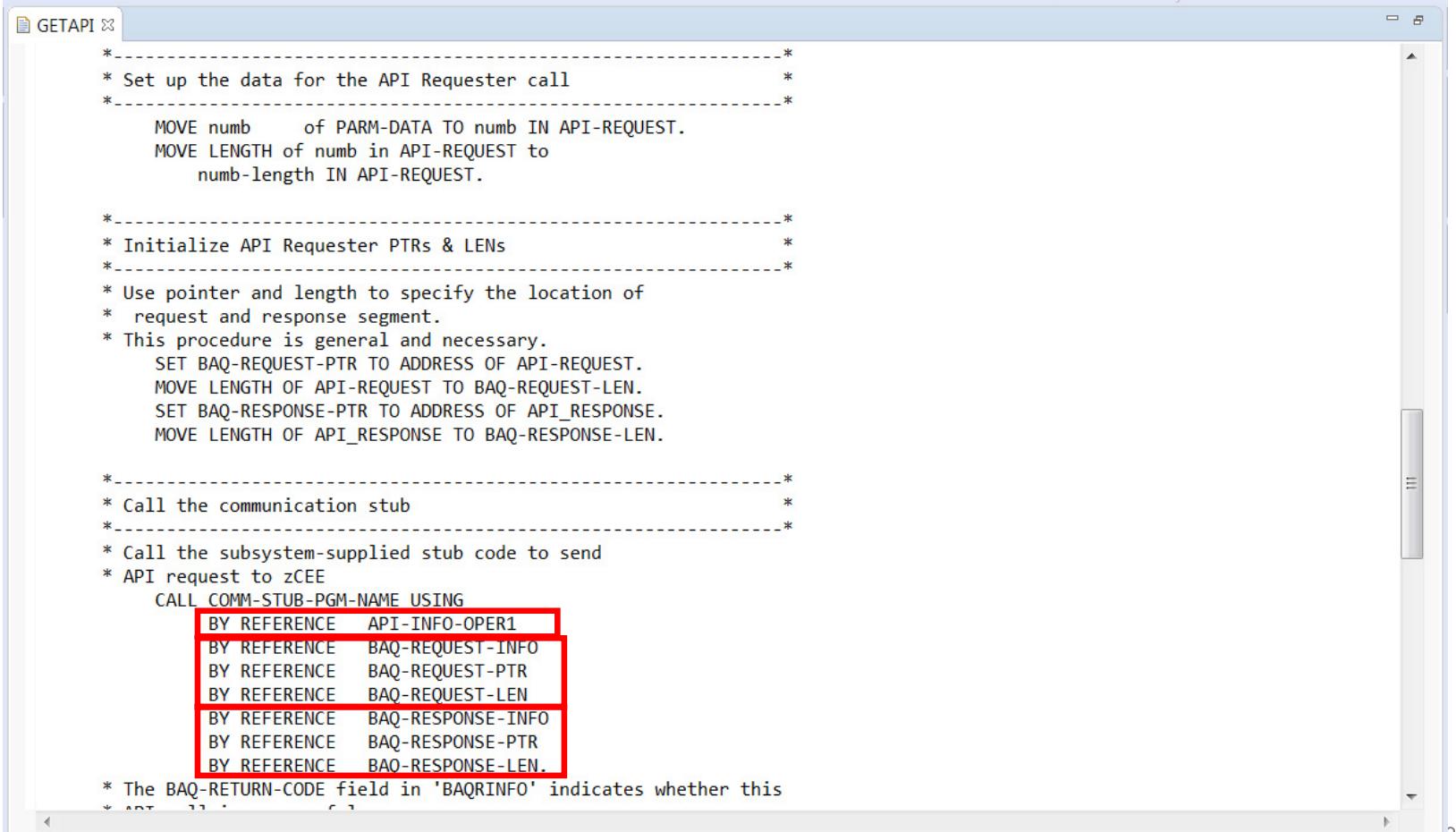
  <zosconnect_authData id="myBasicAuth">
    user="Fred"
    password="fredpwd" />
</server>
```
- CSC02I01**: Shows the `Structure with the API in API-INFO-OPER1` with fields:

03 BAQ-APINAME	PIC X(255)
VALUE 'cscvinc_1.0.0'.	
03 BAQ-APINAME-LEN	PIC S9(9) COMP-5 SYNC
VALUE 13.	
03 BAQ-APIPATH	PIC X(255)
VALUE '/cscvinc/employee/{numb}'.	
03 BAQ-APIPATH-LEN	PIC S9(9) COMP-5 SYNC
VALUE 24.	
03 BAQ-APIMETHOD	PIC X(255)
VALUE 'GET'.	
03 BAQ-APIMETHOD-LEN	PIC S9(9) COMP-5 SYNC
VALUE 3.	

Annotations in red circles highlight the `host` and `basicAuthRef` values in the `apis.xml` configuration, and the `connectionRef` and `requesterPrefix` values in the `CSC02I01` properties panel.

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 starting with asterisks (*). It includes instructions for setting up API REQUEST and REQUEST PTRs, initializing lengths, and finally calling the communication stub. A red box highlights the parameters passed to the stub call:

```
*-----*
* 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.

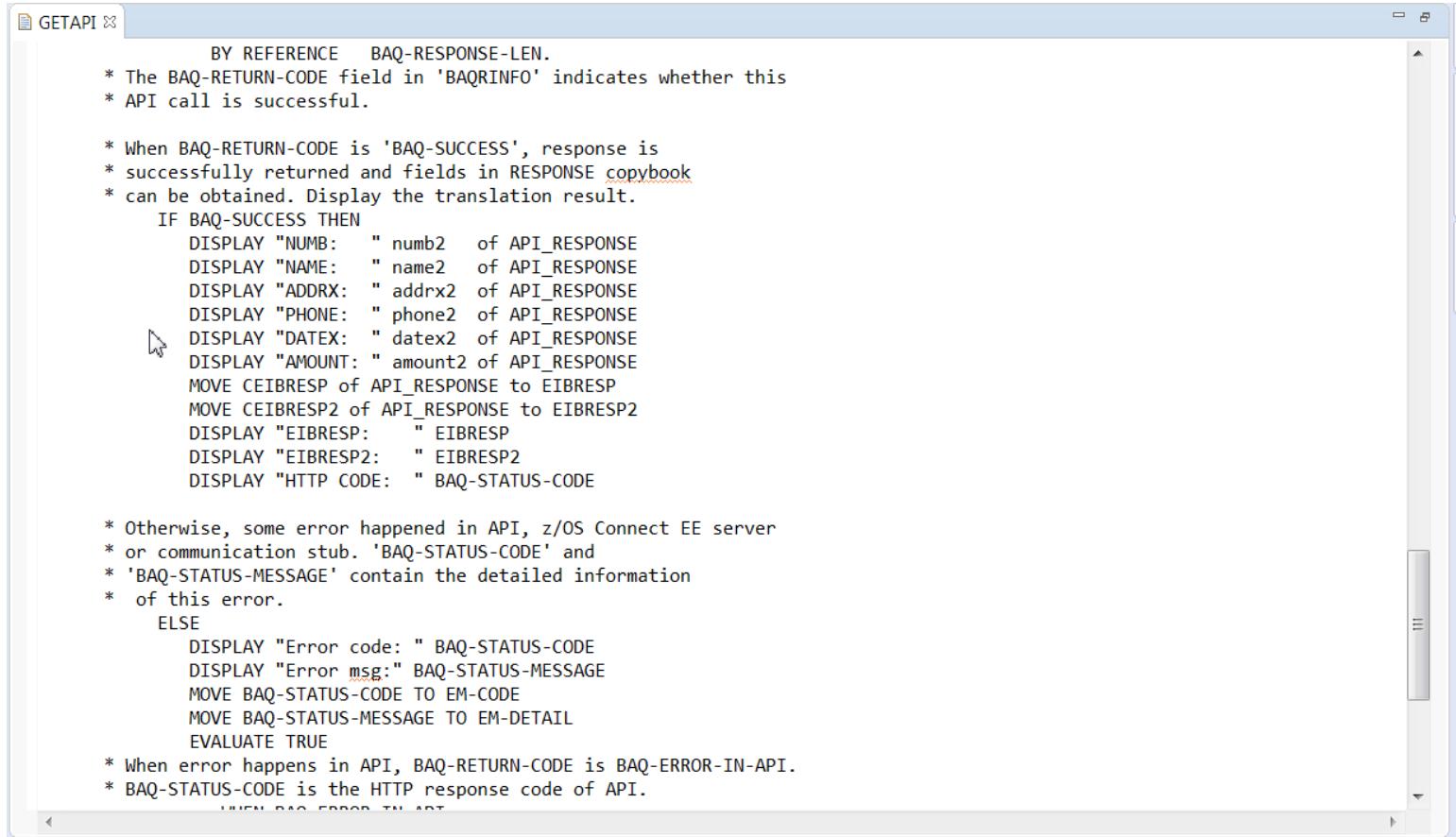
*-----*
* Initialize API Requester PTRs & LENs          *
*-----*
* Use pointer and length to specify the location of
* request and response segment.
* This procedure is general and necessary.
        SET BAQ-REQUEST-PTR TO ADDRESS OF API-REQUEST.
        MOVE LENGTH OF API-REQUEST TO BAQ-REQUEST-LEN.
        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
* API call was successful.
```

Steps to calling an external API

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

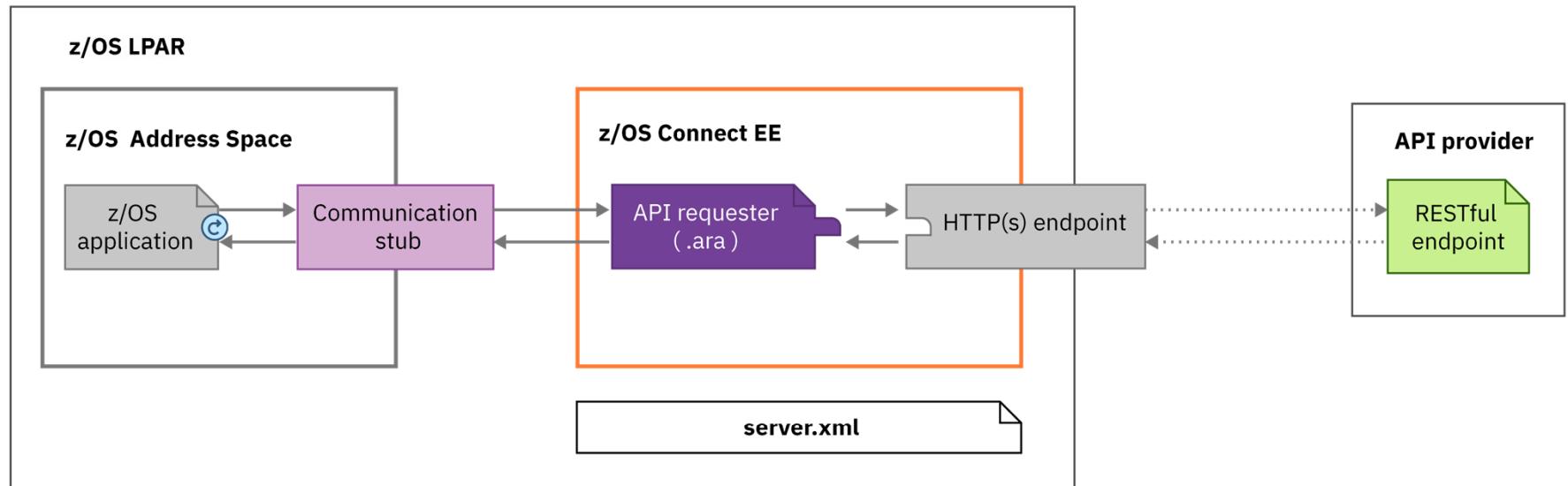


The screenshot shows a window titled "GETAPI" containing AS/400 JCL code. The code handles the response from an API call, specifically dealing with BAQ-RESPONSE-LEN and BAQ-RETURN-CODE fields. It includes logic to display various response fields if the call was successful ("BAQ-SUCCESS") or to handle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

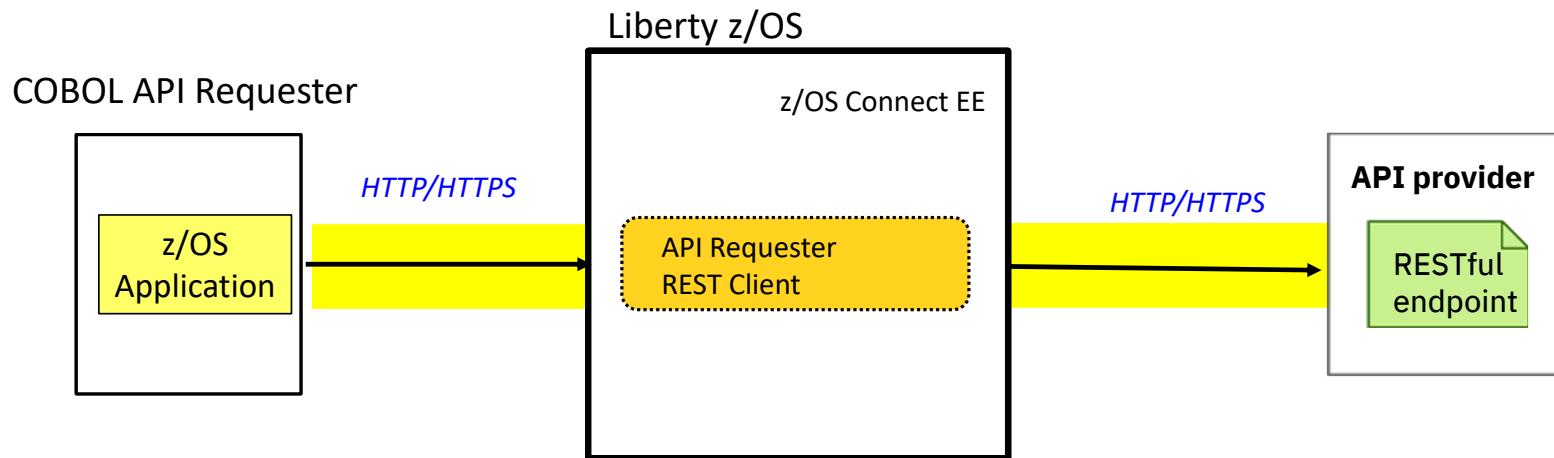
Done





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:

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

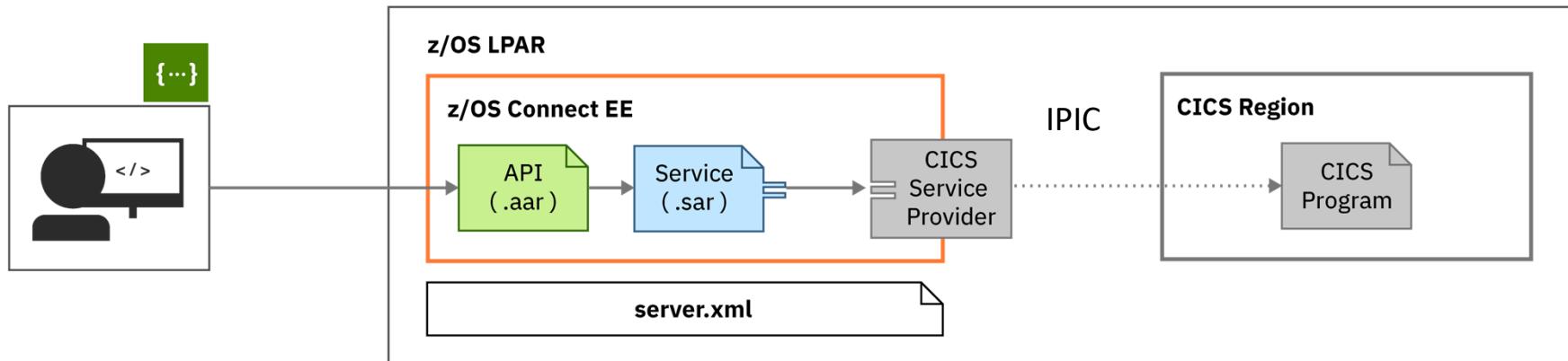


/common_scenarios

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

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 statistics and configuration details.

```
OVERTYPE TO MODIFY
CEDA ALTER TCpipservice( IPIC      )                                     CICS RELEASE = 0710
TCpipservice : IPIC
GRUop : SYSGRP
DEscription ==> DFHISAIPIPC
UrM ==> 01491
PORTnumber ==> 1-65535
STATUS ==> Open
PROtocol ==> IPic
TRANsaction ==> CISS
Backlog ==> 00000
TSqprefix :
Host ==> ANY
(Mixed Case) ==>
Ipaddress ==> ANY
SPECifTCPs ==>
SOCKETclose ==> No
MAXPersist ==> No
MAXDataLen ==> 000032
                                         No | 0-240000 (HHMMSS)
                                         No | 0-65535
                                         3-524288
SYSID=CICS APPLID=CICS53Z
PF 1 HELP 2 COM 3 END          6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
M A E
Connected to remote server/host wg31 using lu/pool TCP00104 and port 23
06/022
```

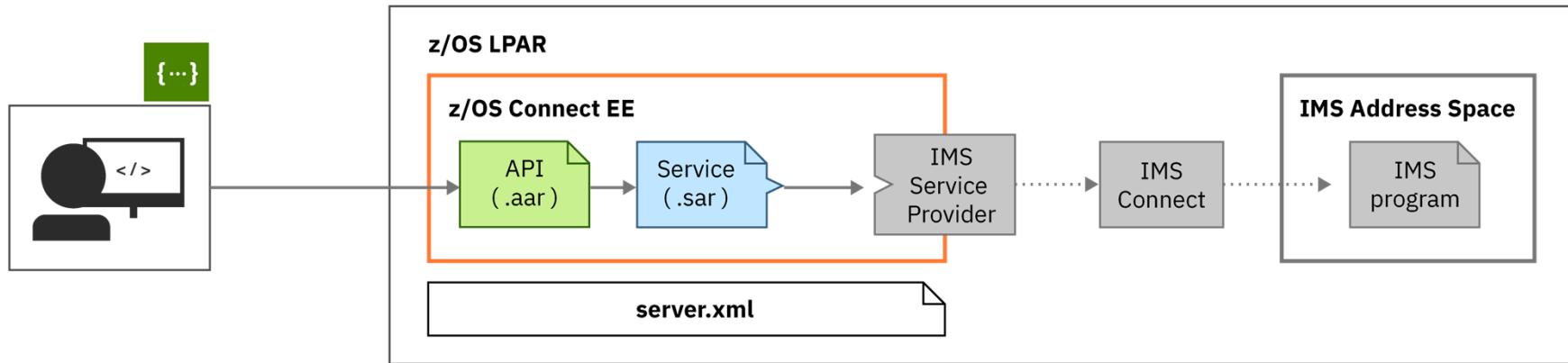
The screenshot shows the 'catalog.xml' configuration file. It contains XML code defining a server and its connection to CICS. A callout box points to the 'featureManager' section, stating: 'Features are functional building blocks. When configured here, that function becomes available to the Liberty server'. Another callout box points to the 'zosconnect_cicsIpicConnection' section, stating: 'Define IPIC connection to CICS'.

```
<server description="CICS IPIC - catalog">
<!-- Enable features -->
<featureManager>
<feature>zosconnect:cicsService-1.0</feature>
</featureManager>
<zosconnect_cicsIpicConnection id="catalog">
<host>wg31.washington.ibm.com</host>
<port>1491</port>
<transid>CSMI</transid>
<transidUsage>EIB_AND_MIRROR</transidUsage>
</zosconnect_cicsIpicConnection>
</server>
```

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>

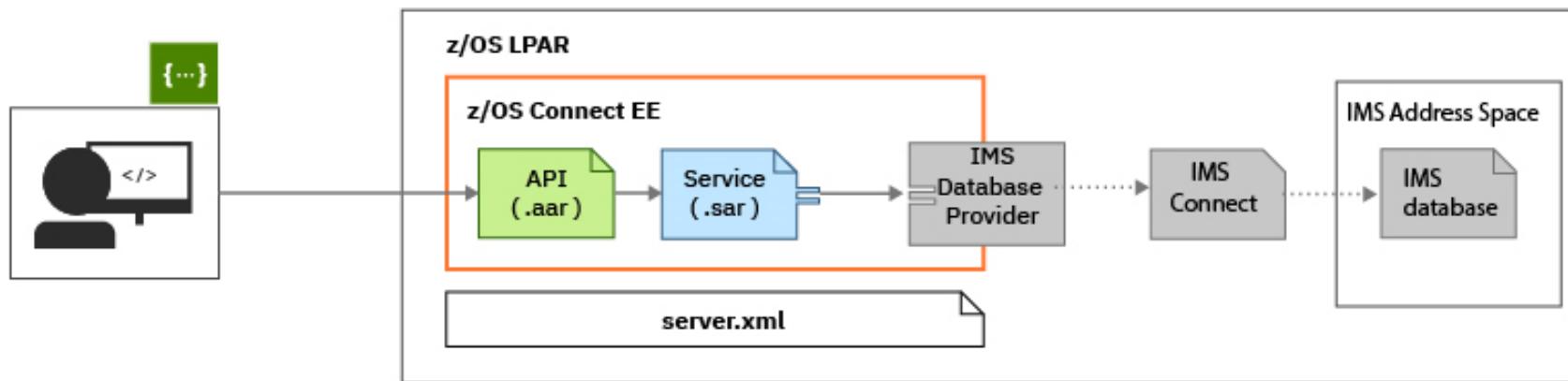
<authData id="Connection1_Auth" password="encryptedPassword1" user="userName1"/>
</server>
```

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.

IMS Connection Factory



z/OS Connect EE

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection profile: DFSIVPACConn

ConnectionFactory

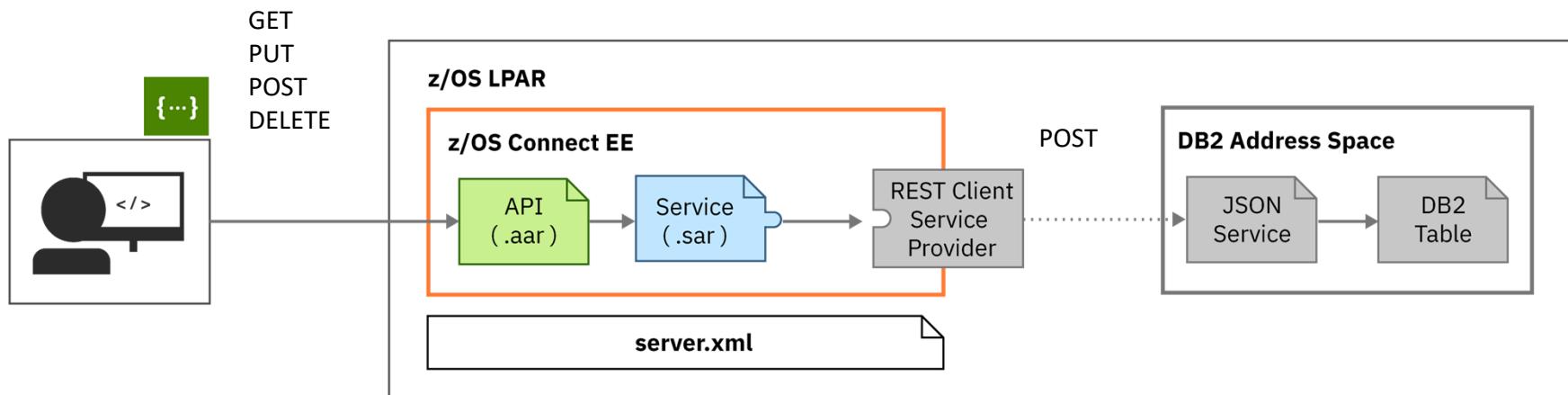
```
<connectionFactory id="DFSIVPACConn">
<properties.imsudbJLocal
  databaseName="DFSIVPA"
  datastoreName="IVP1"
  datastoreServer="wg31.washington.ibm.com"
  driverType="4"
  portNumber="5555"
  user="USER1"
  password="password"
  flattenTables="True"/>
</connectionFactory>
```

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 server.xml file is the key configuration file:

The screenshot shows the Service Project Editor interface with the 'Configuration' tab selected. On the left, the 'Required Configuration' section displays service parameters like DSNL004I, LOCATION, LU, and TCPPORT. A red arrow points from the 'db2conn' connection reference in the configuration to the 'zosconnect_zosConnectServiceRestClientConnection' element in the db2pass.xml code. Another red arrow points from the 'zosconnect_zosConnectServiceRestClientBasicAuth' element to the 'applName' attribute.

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

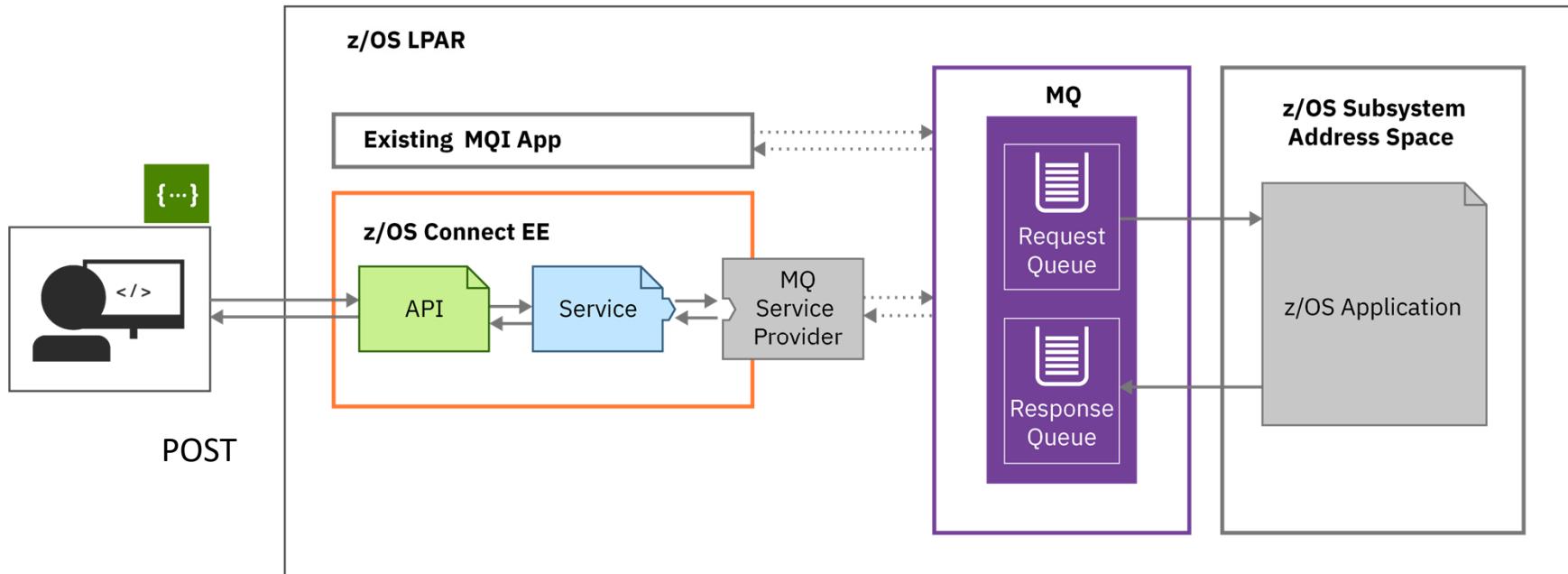
db2pass.xml

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

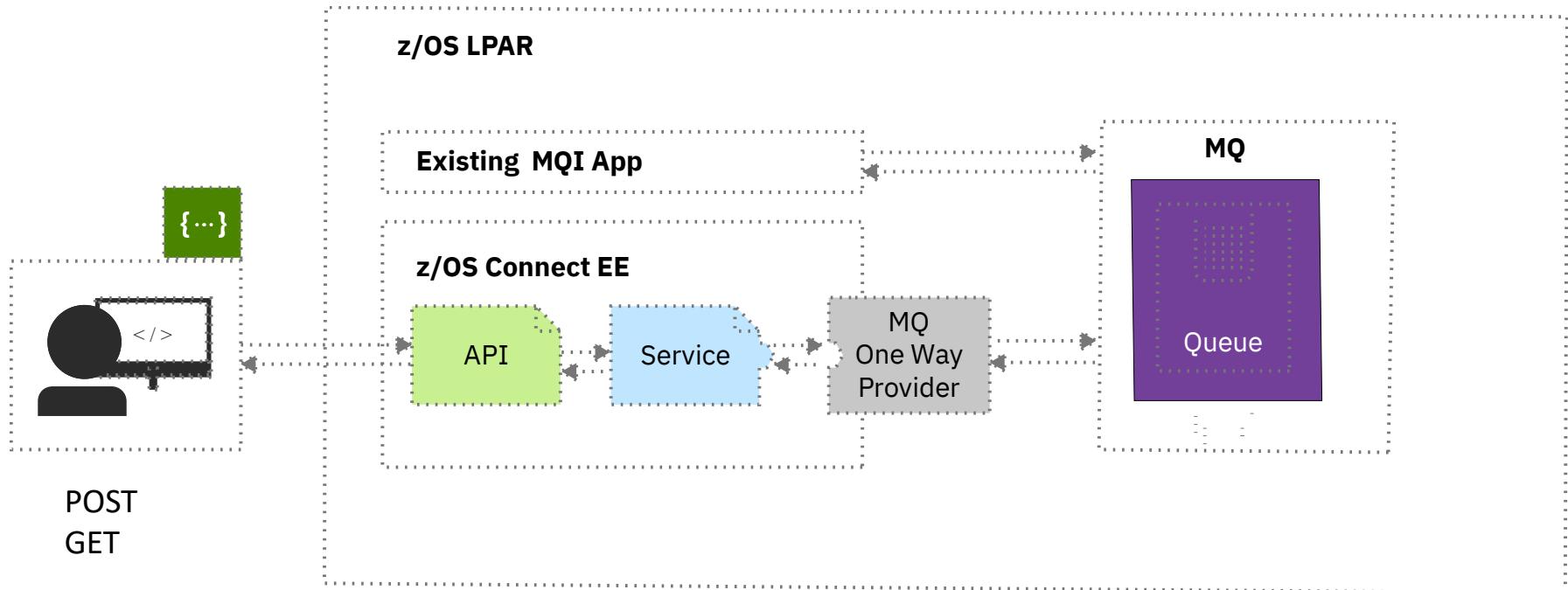
Topology (Two-way service example)



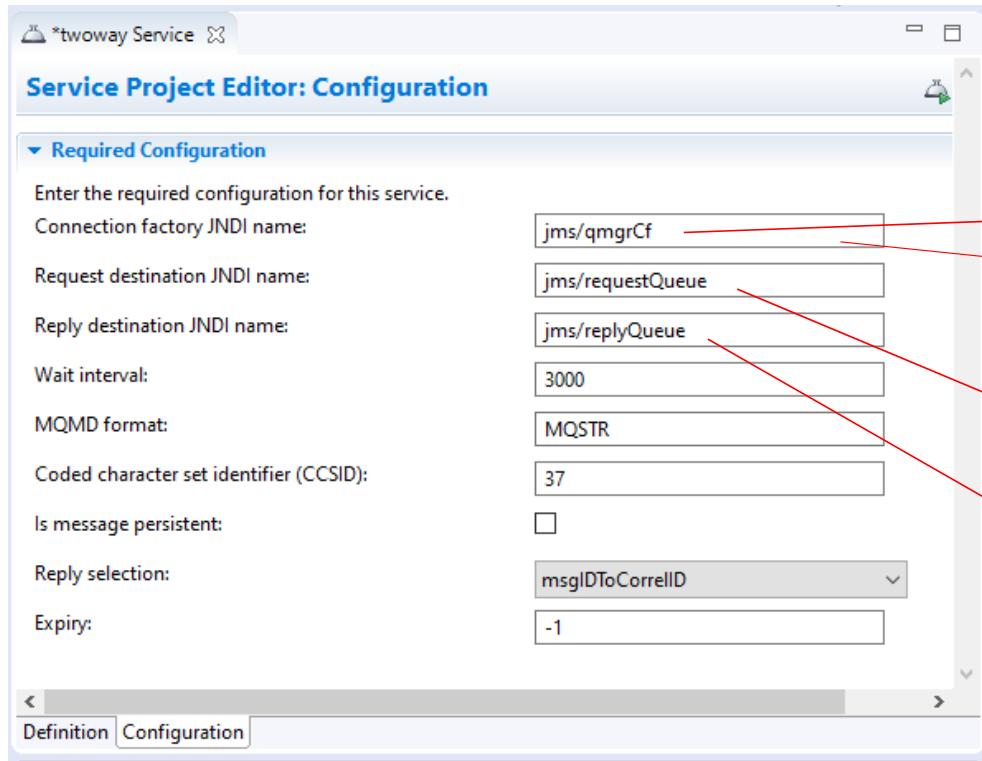
You can also configure one-way services.

Connections to MQ

Topology (One-way service example)



The server.xml File (MQ)



mq.xml

Read only Close

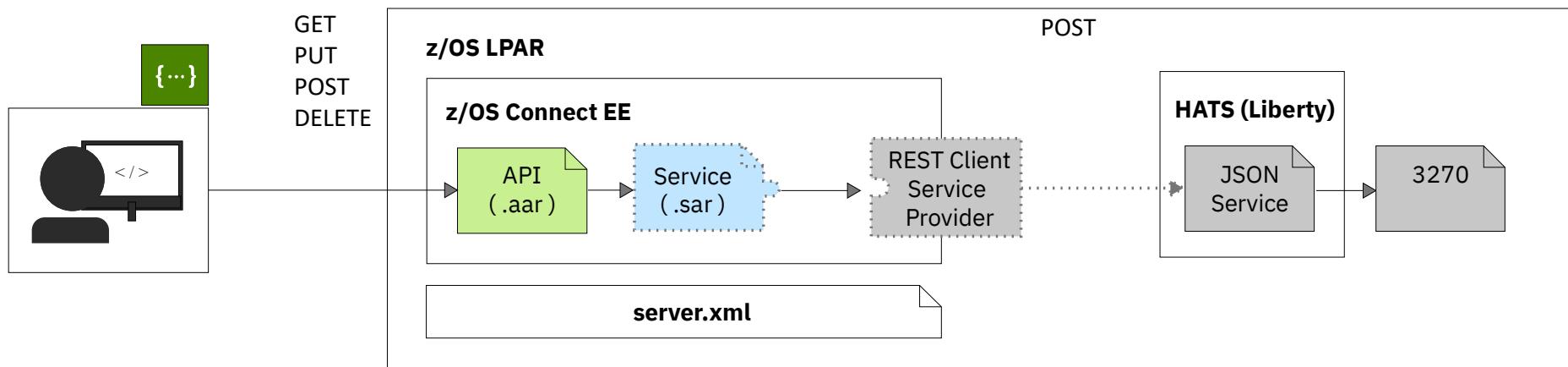
Design Source

```
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



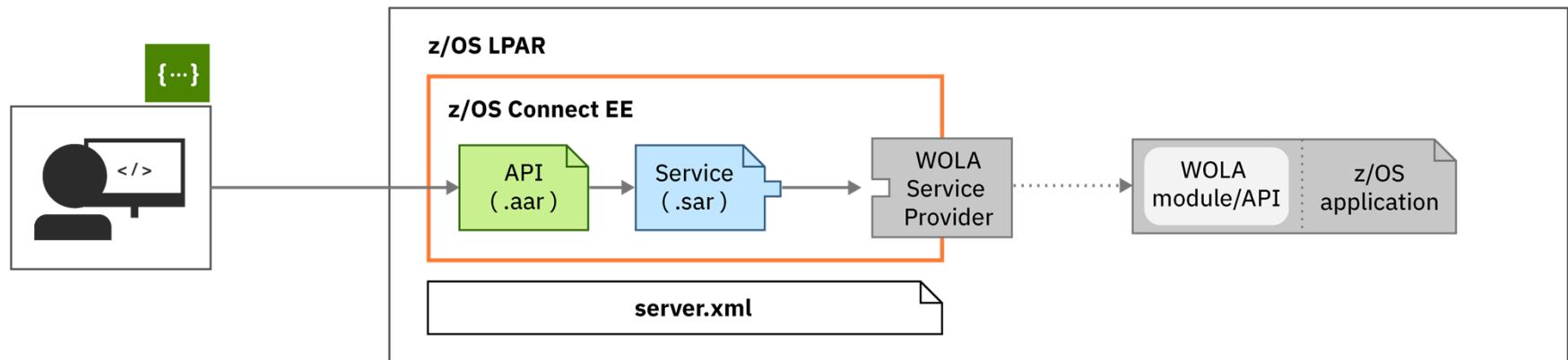
Topology



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

Connections to a MVS batch application

Topology



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

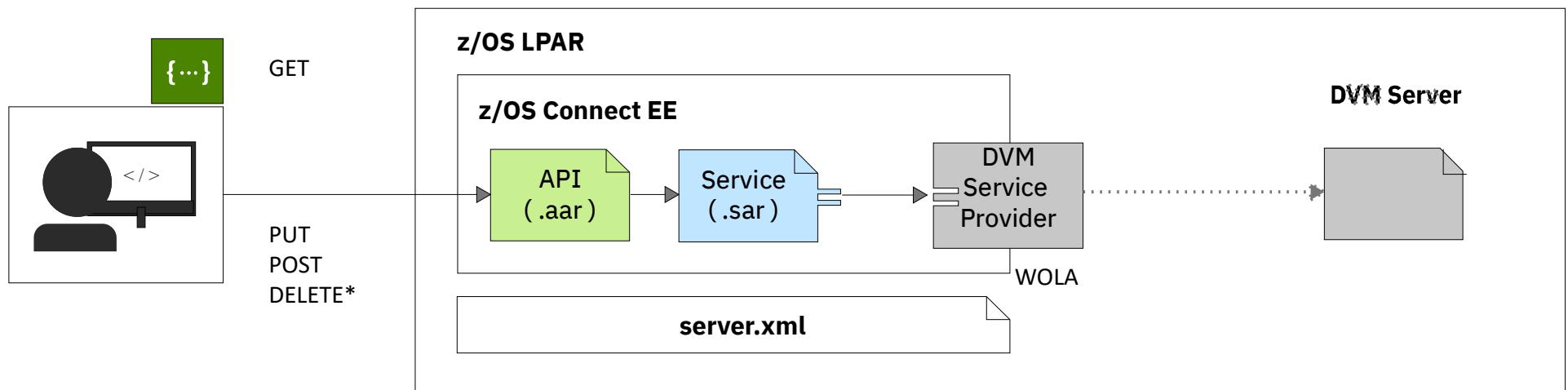
The z/OS application must be WOLA-enabled.

Connections to DVM



z/OS Connect EE

Topology



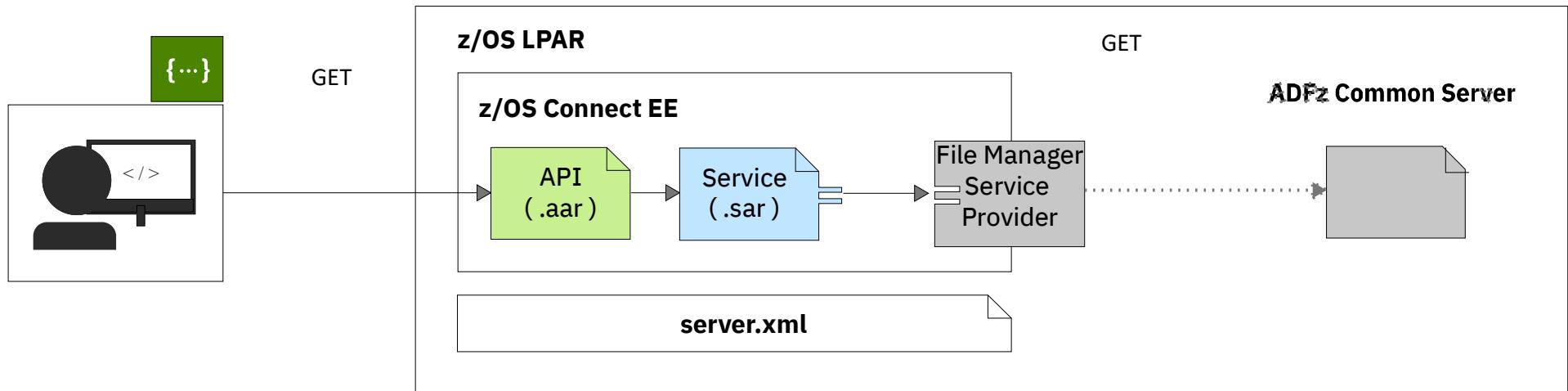
The DVM service provider uses WOLA

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

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.

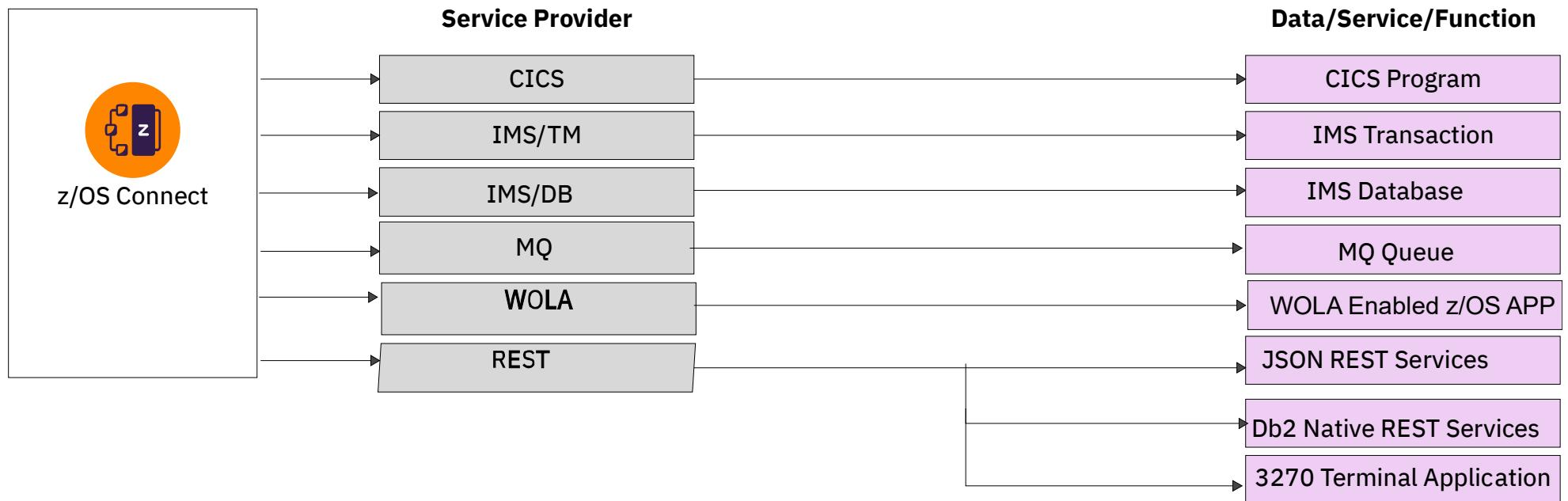
server.xml

```
<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?

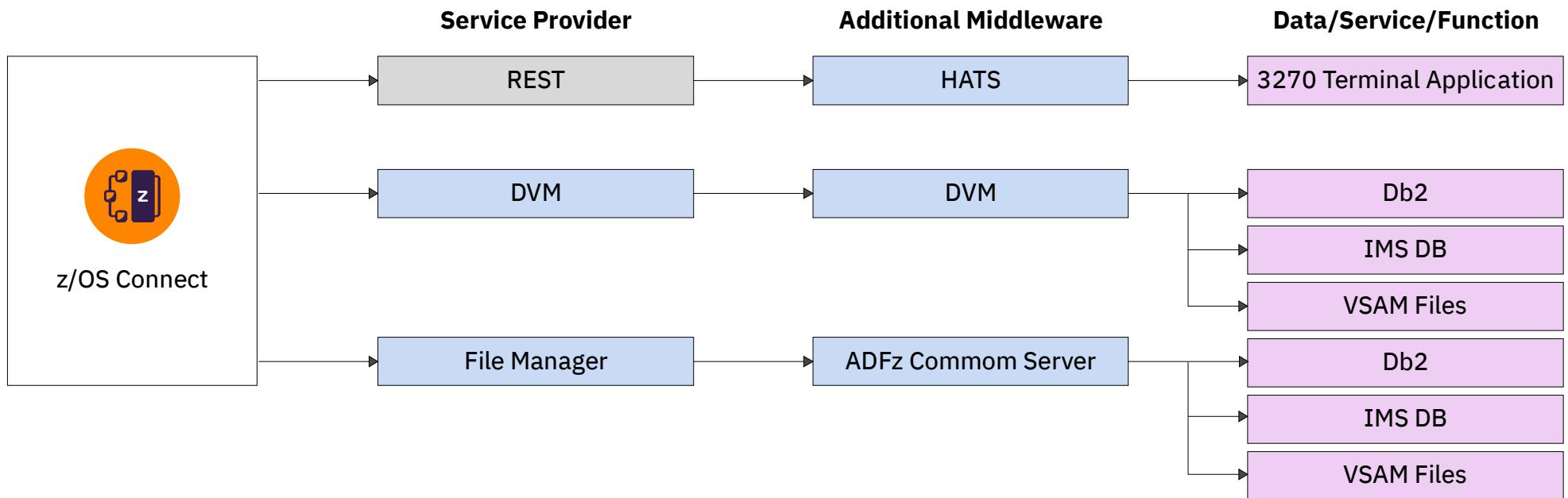
And which service provider could I use?



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

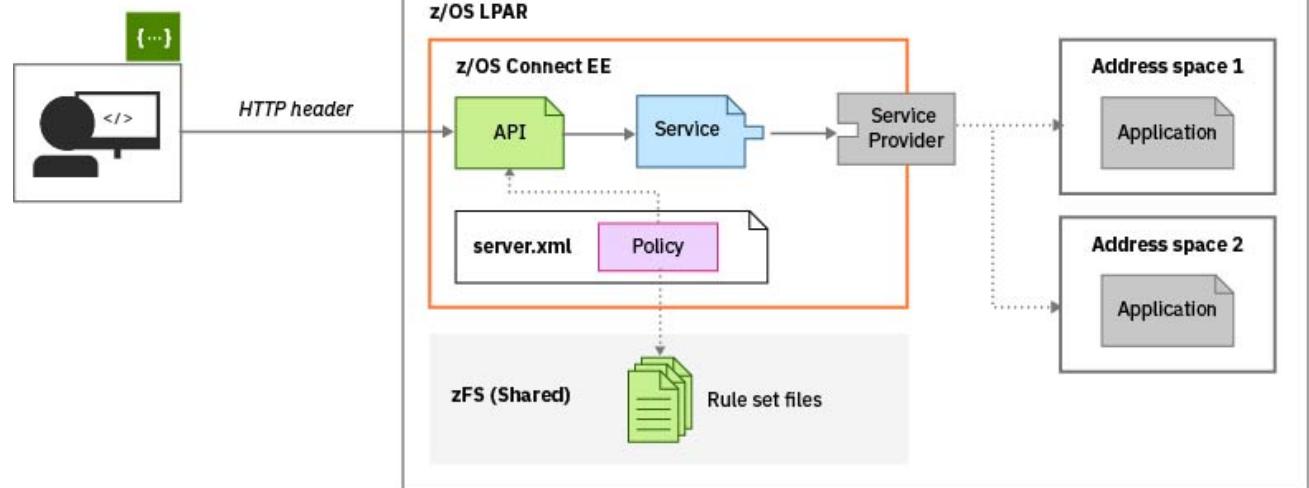
- imsConnectionRef
- imsInteractionRef
- imsInteractionTimeout
- imsLtermOverrideName
- imsTranCode
- imsTranExpiration

Db2 attributes

- db2ConnectionRef
- db2CollectionID

MQ attributes

- mqConnectionFactory
- mqDestination
- mqReplyDestination



A sample API Policies for CICS



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. It displays a configuration for a GET endpoint named "GET.employee.{numb}". The interface is organized into several sections:

- Body - cscvincServiceOperation**: Contains a link "[Click to filter...](#)".
- HTTP Request**: Contains a link "[Click to filter...](#)". Under this section, there are two header entries:
 - cicsMirror**: optional string
 - cicsConnection**: optional string
- Path Parameters**: Contains a link "[Click to filter...](#)".
 - {numb}**: Required string
- Query Parameters**: Contains a link "[Click to filter...](#)".
- Body - cscvincServiceOperation**: Contains a link "[Click to filter...](#)".

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
-->
```

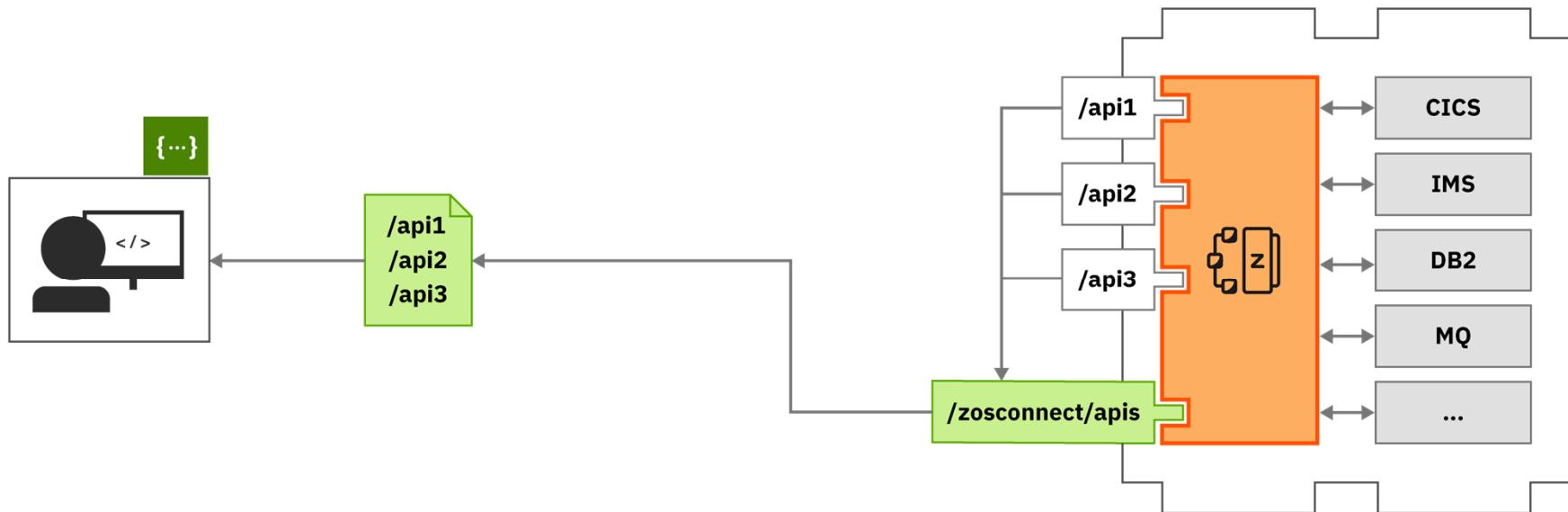
```
<webApplication id="serverConfig-location" name="serverConfig"
location="${server.config.dir}">
<web-ext context-root="/server/config"
enable-file-serving="true" enable-directory-browsing="true">
<file-serving-attribute name="extendedDocumentRoot"
value="${server.config.dir}" />
</web-ext>
</webApplication>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

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 15:48:18:869 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

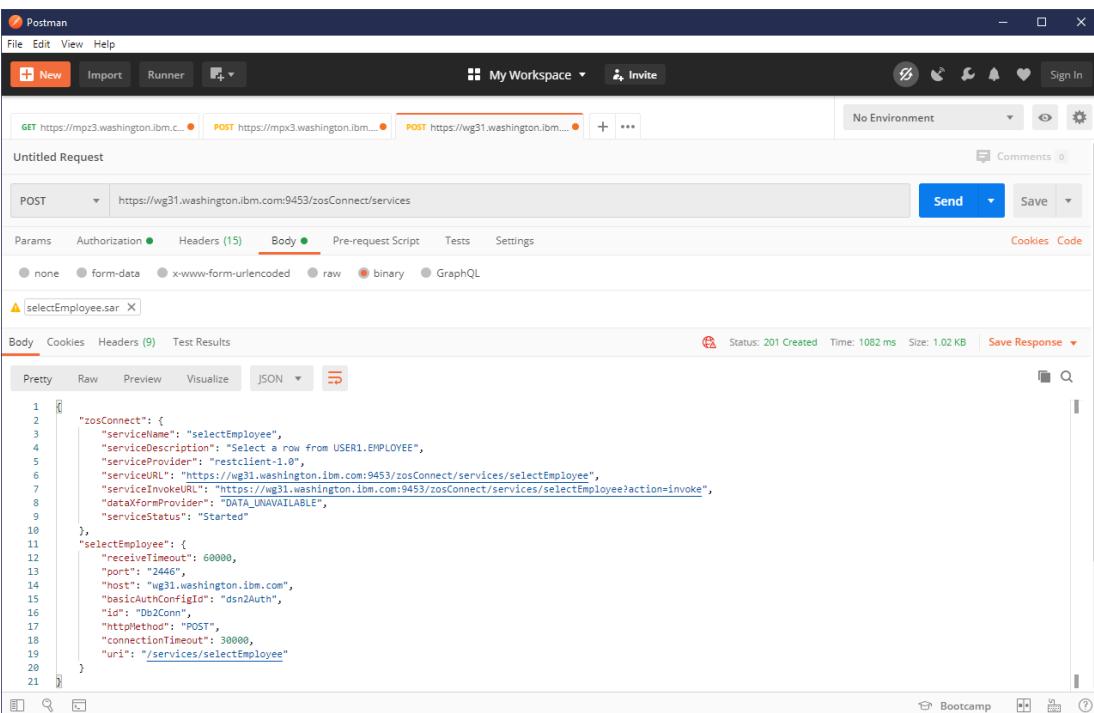
API Requesters : Operations that work with API Requesters.

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

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 application interface. A POST request is being made to <https://wg31.washington.ibm.com:9453/zosConnect/services>. The 'Body' tab is selected, and a file named 'selectEmployee.sar' is attached. The file content is displayed in a code editor:

```

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:

```

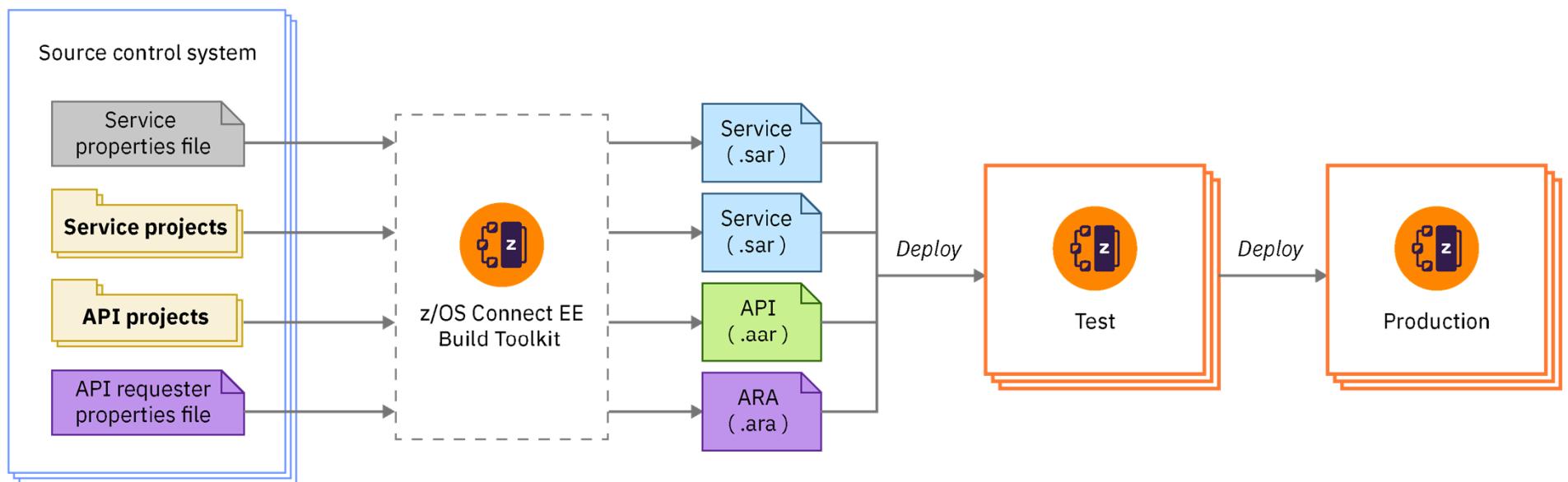
{
  "zosConnect": {
    "serviceName": "selectEmployee",
    "serviceDescription": "Select a row from USER1.EMPLOYEE",
    "serviceProvider": "IBM_ZOS_CONNECT_SERVICE_REST_CLIENT-1.0",
    "serviceURL": "https://mpxm:9453/zosConnect/services/selectEmployee",
    "serviceInvokeURL": "https://mpxm:9453/zosConnect/services/selectEmployee?action=invoke",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

```

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

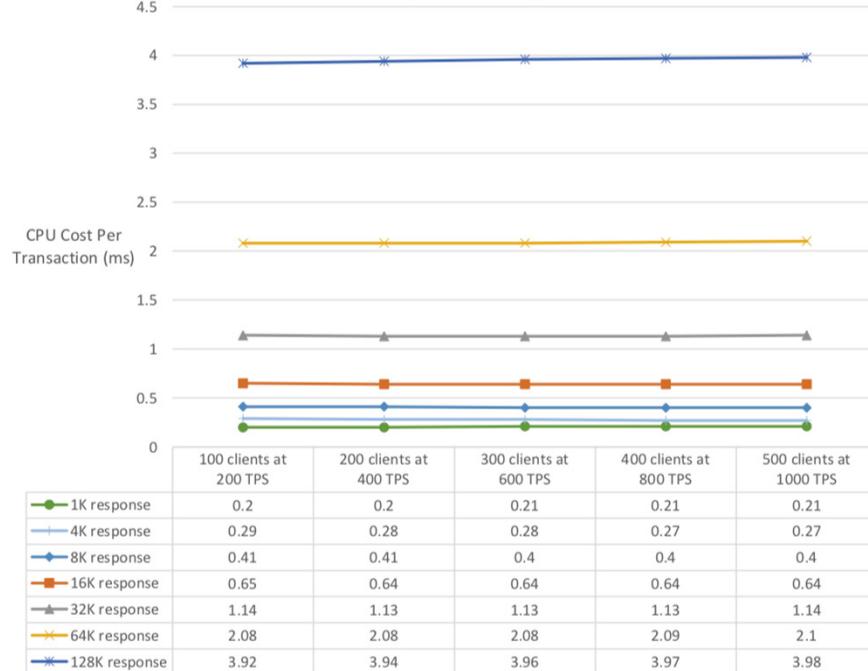
```
//*****
///* SET SYMBOLS
//*****
//EXPORT EXPORT SYMLIST=(*)
// SET CURL='/usr/lpp/rocket/curl'
//*****
///* CURL Procedure
//*****
//CURL PROC
//CURL EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR DD SYSOUT=*
//STDOUT DD SYSOUT=*
// PEND
//*****
///* STEP CURL - use cURL to stop API 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](https://www.rocketsoftware.com/platforms/ibm-z(curl-for-zos)

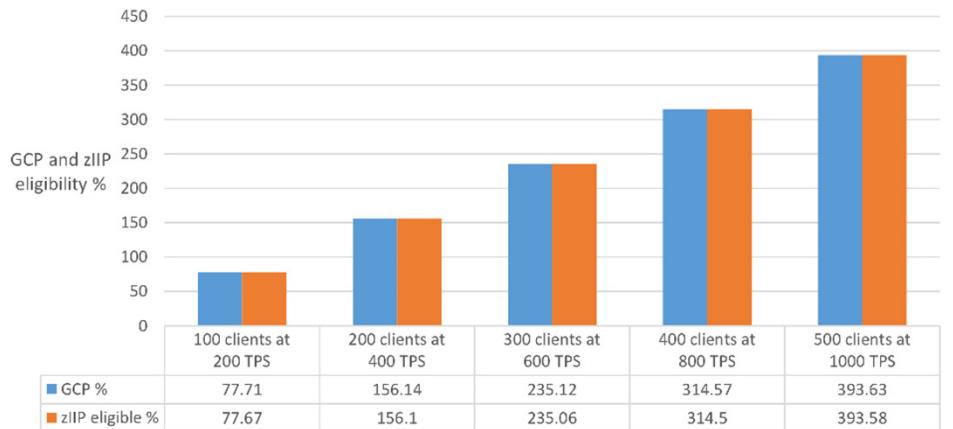
Performance: API Provider

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



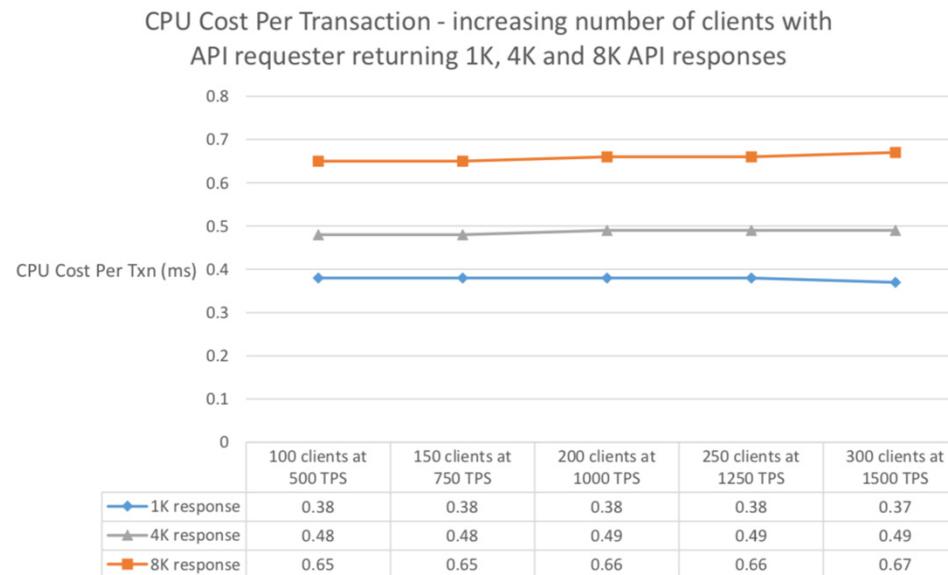
zIIP eligibility - increasing number of clients with 50 byte requests and 128K responses, using channels and CICS SP



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

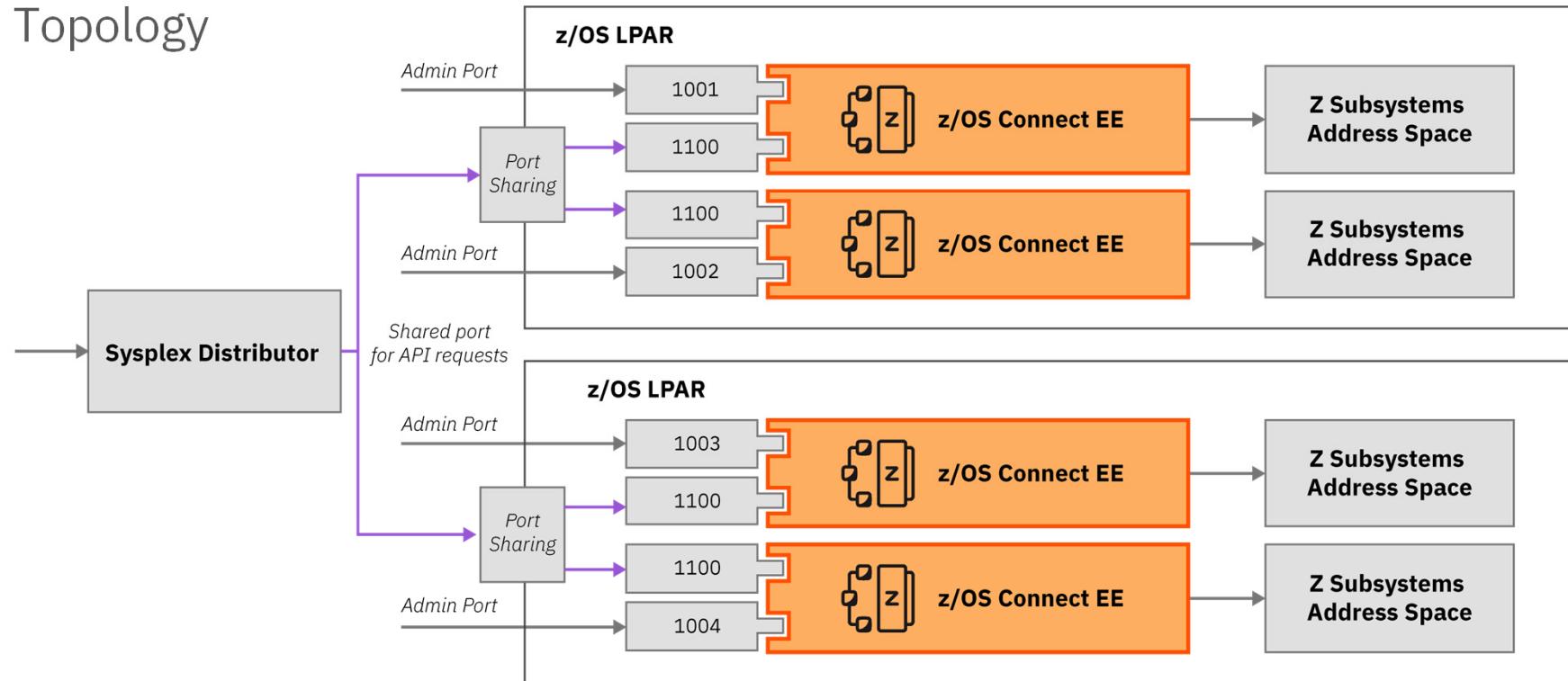


z/OS Connect EE is a Java-based product:
Over **99%** of its MIPs are **eligible for ZIIP offload**.



High Availability

Topology



ibm.biz/zosconnect-ha-concepts

ibm.biz/zosconnect-scenarios



/security

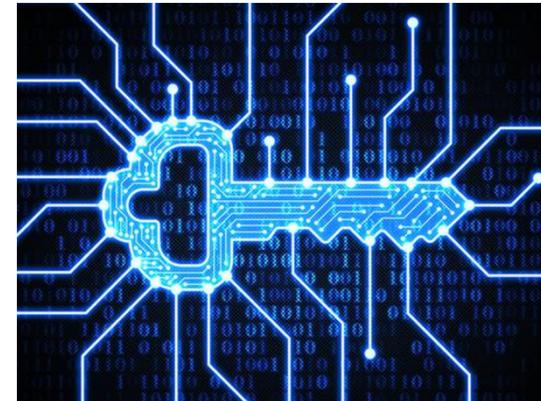
How is security implement?

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.



We need to understand the challenges



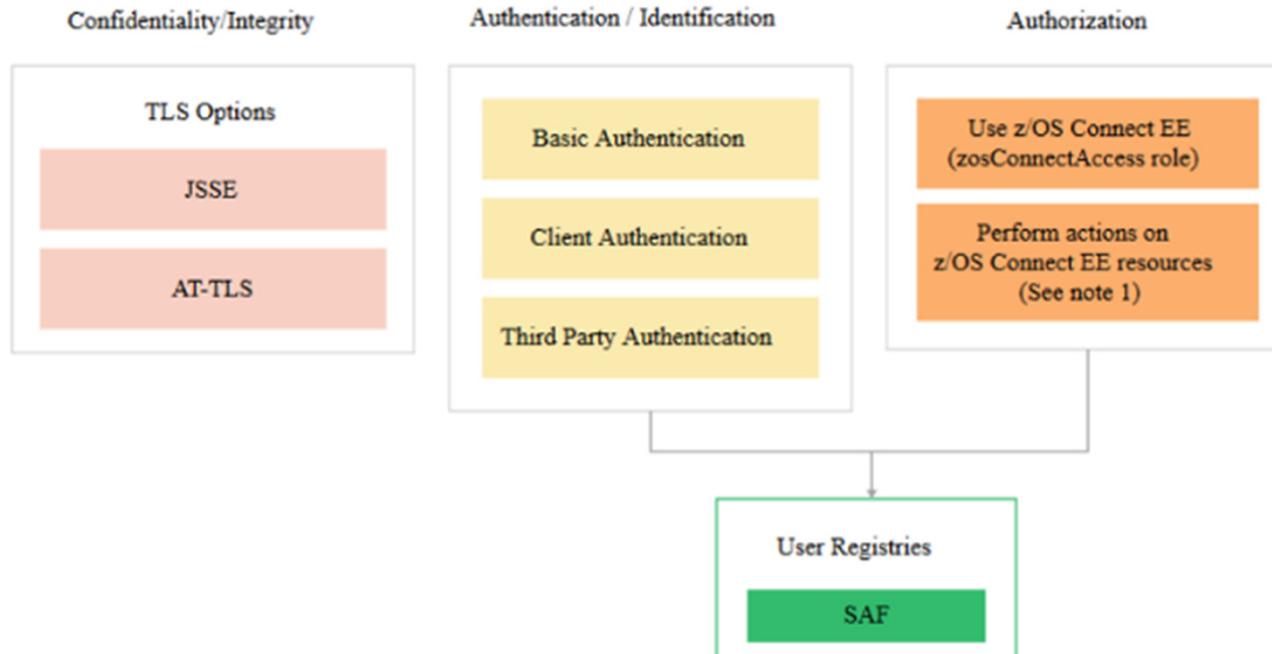
z/OS Connect EE

- Providing secure access between middleware components that use disparate security technologies e.g., registries like LDAP, SAF, TLS etc. in order to propagate security credentials from a client all the way to the targeted resource.
 - This is a driver for implementing open security models like OAuth and OpenID Connect and standard tokens like JWT
 - Integrating security involves different products including z/OS Connect, WebSphere Liberty Profile on z/OS with CICS, IMS, Db2, MQ,... probably for the first time in your environment.
-
- Security for of these components are all documented in different places
 - Considering that security is often at odds with **performance**, the more secure techniques often mean more processing overhead, especially if not configured optimally
-
- Remember security is usually not a choice but a requirement.

z/OS Connect EE security options



z/OS Connect EE



The actions which can be controlled by authorization (see Note 1 in the diagram above) are: deploying, querying, updating, starting, stopping and deleting of APIs, services and API requesters.

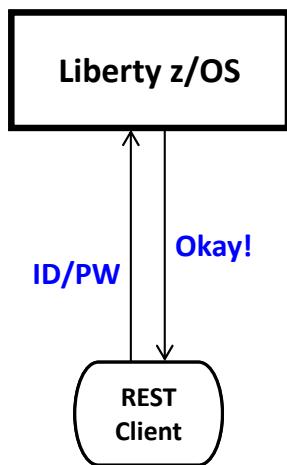
API Provider Authentication



z/OS Connect EE

Several different ways this can be accomplished:

Basic Authentication



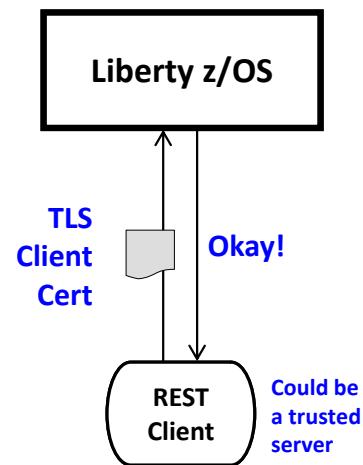
Server prompts for ID/PW

Client supplies ID/PW or
ID/Passticket

Server checks registry:

- Basic (server.xml)
- LDAP
- SAF

Client Certificate



Server prompts for cert.

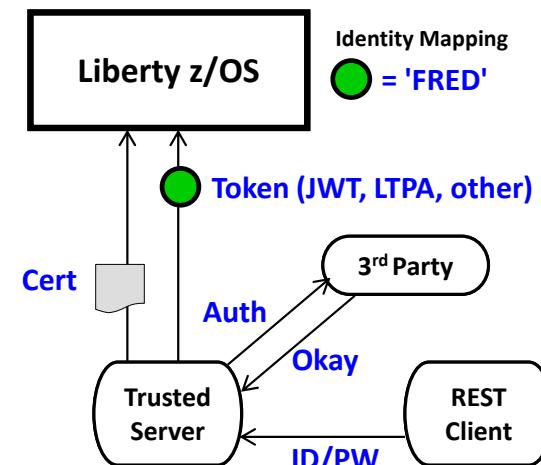
Client supplies certificate

Server validates cert and
maps to an identity

Registry options:

- LDAP
- SAF

Third Party Authentication



Client authenticates to 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



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 navigation bar includes links for "Sign up / Log in" and "Search or Track". The main section is titled "Sign Up" and contains a sub-instruction "Already have an ID? Log in". It provides several options for creating an account: "Use one of these sites." with links for Google, Facebook, Amazon, Apple, and Twitter; and "Or enter your own information." with fields for 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 active. The login form requires "Email Address" and "Password", with a "Remember Me" checkbox. Below the form are "Forgot Password" and "Continue as Guest" buttons. There are also links for "Continue with Apple", "Continue with Facebook", and "Continue with Google". A small notice at the bottom left reads: "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".

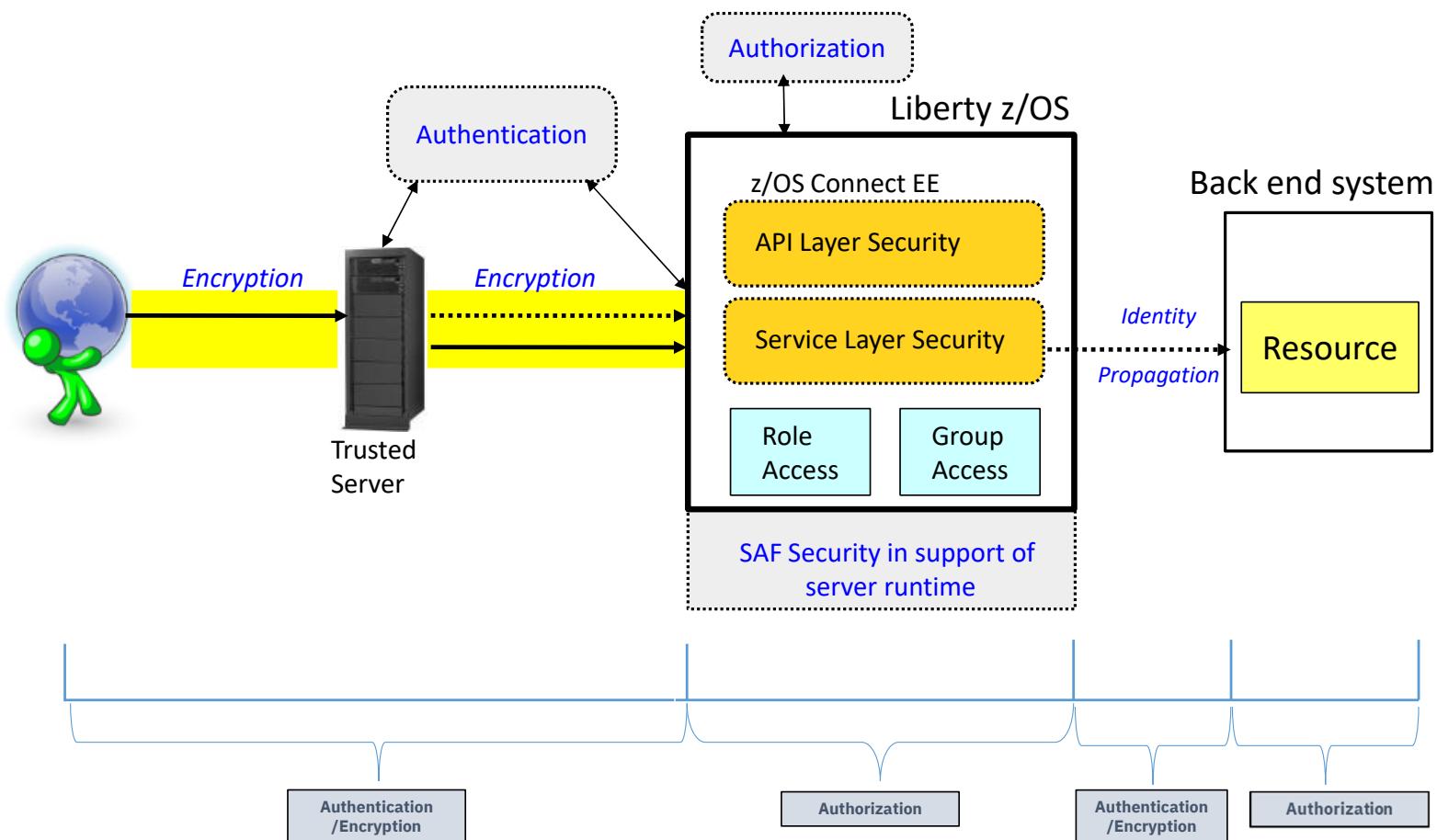
mitchj@us.ibm.com

© 2018, 2021 IBM Corporation

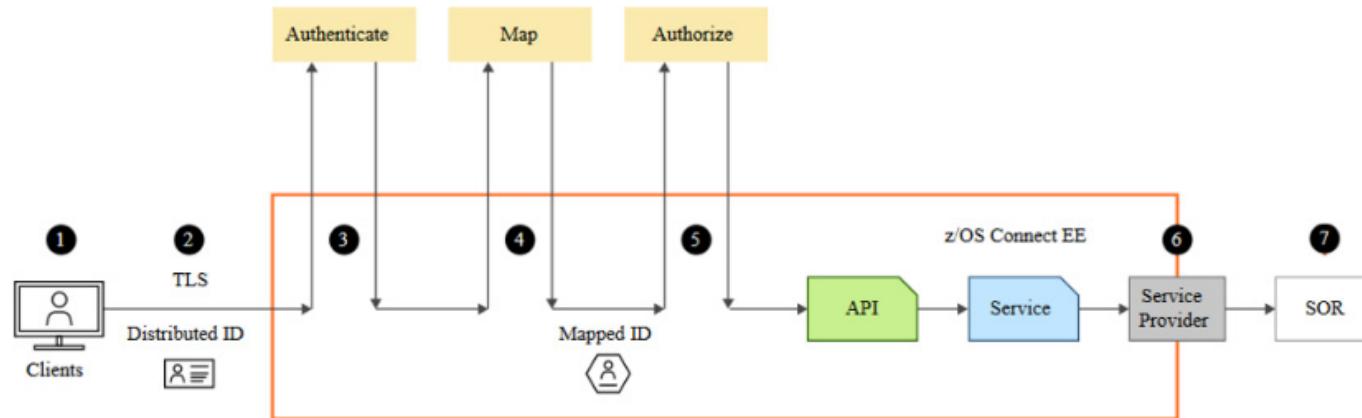
z/OS Connect EE API provider security overview



z/OS Connect EE



Typical z/OS Connect EE API Provider security flow

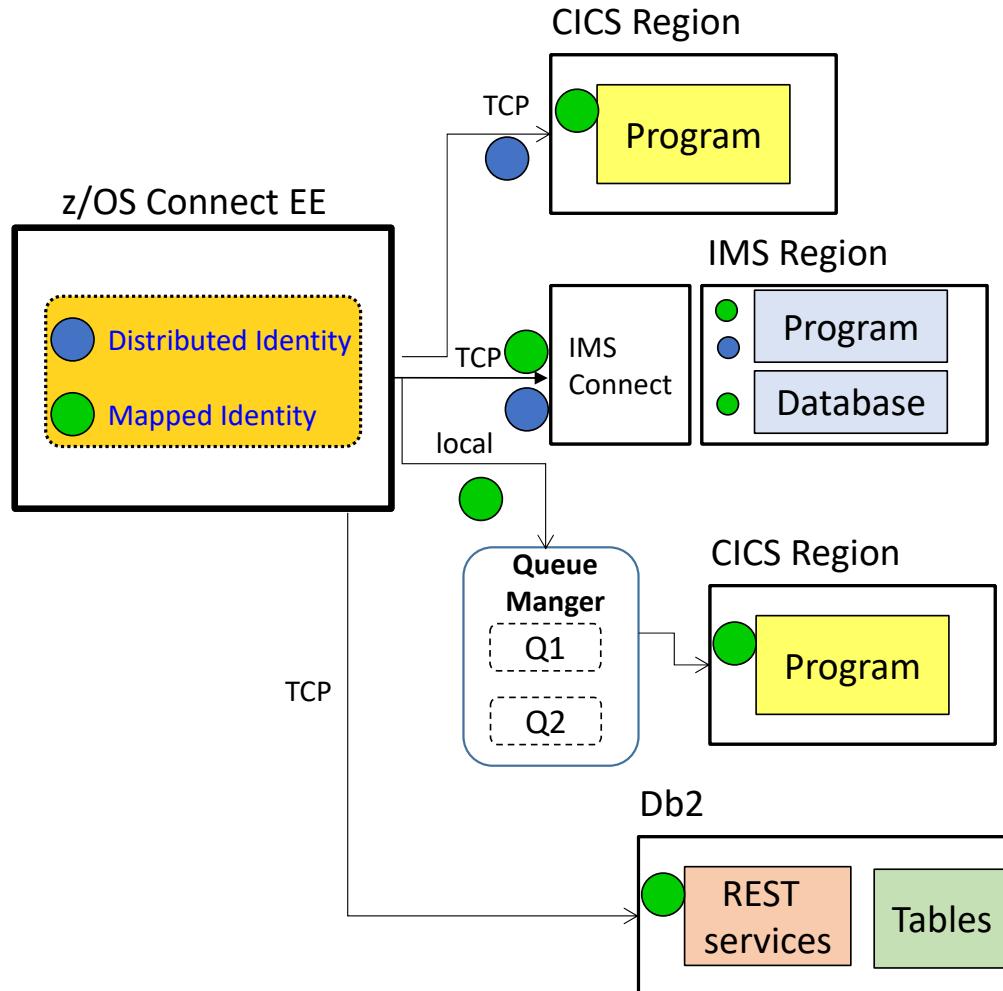


1. The credentials provided by the client
2. Secure the connection to the z/OS Connect EE server
3. Authenticate the client. This can be within the z/OS Connect EE server or by requesting verification from a third-party server
4. Map the authenticated identity to a user ID in the user registry
5. Authorize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

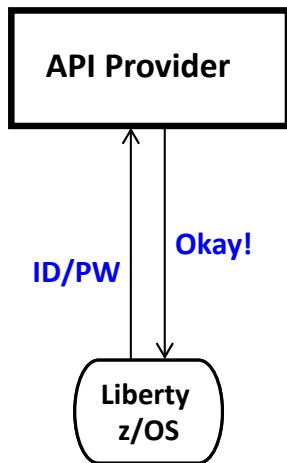
API Requester Authentication



z/OS Connect EE

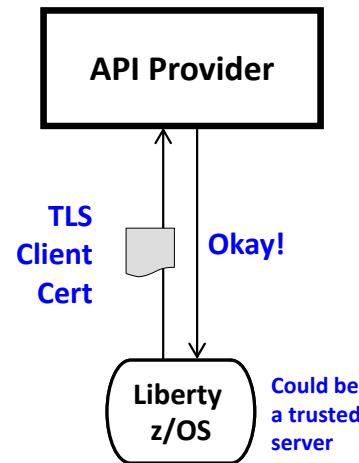
Several different ways this can be accomplished:

Basic Authentication



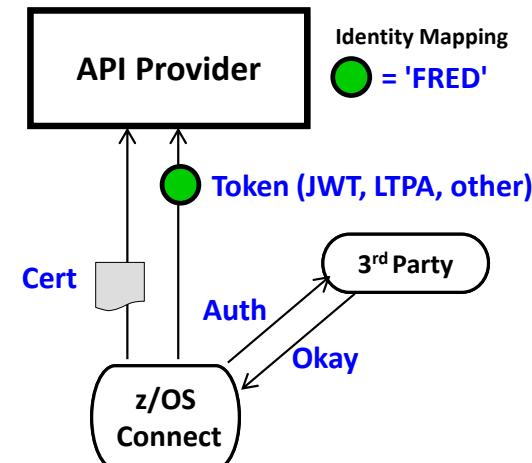
zCEE supplies ID/PW or
ID/Passticket

Client Certificate



Server prompts for certificate
zCEE supplies certificate

Third Party Authentication

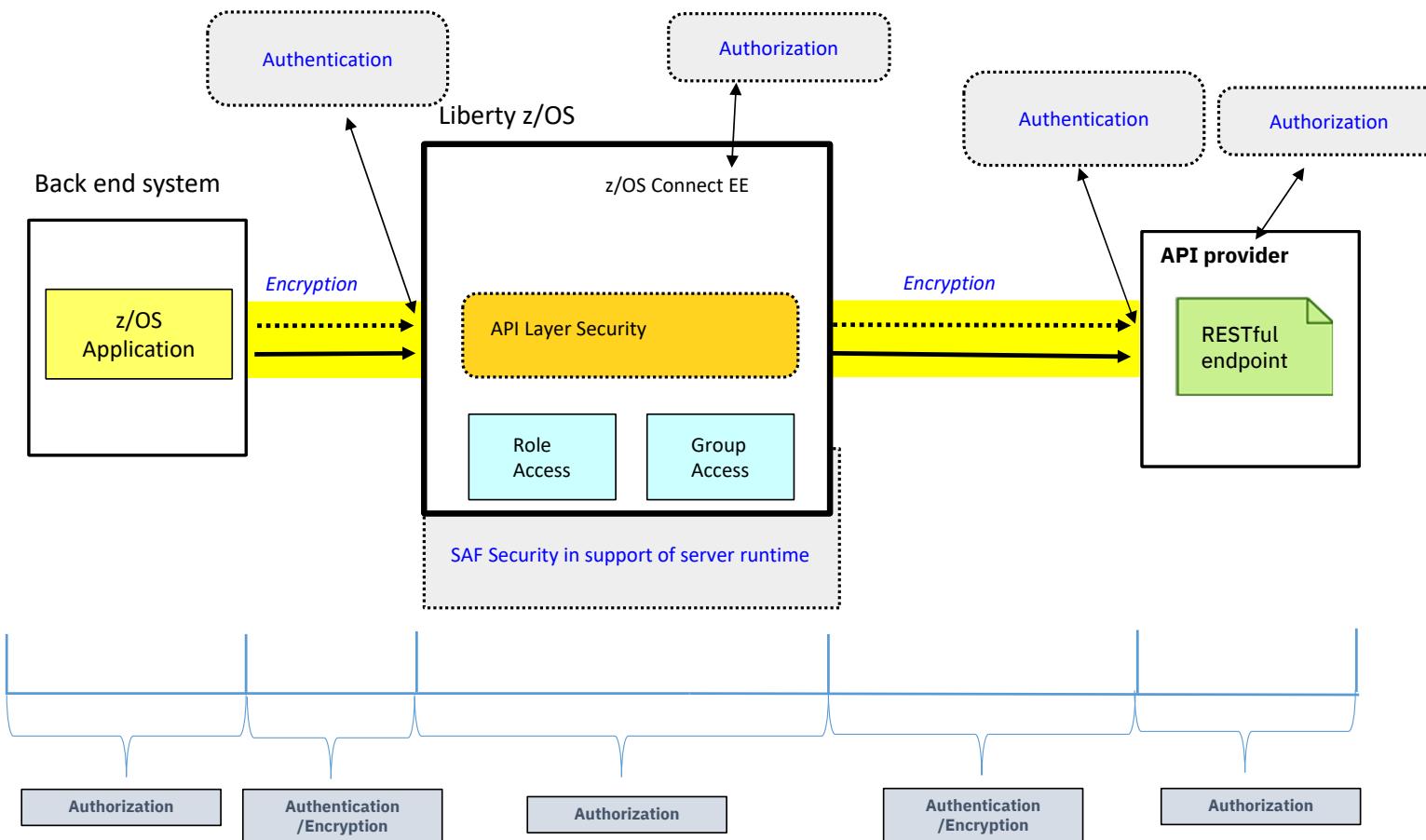


zCEE authenticates to 3rd party sever
zCEE receives a trusted 3rd party token
Token flows to API Provider

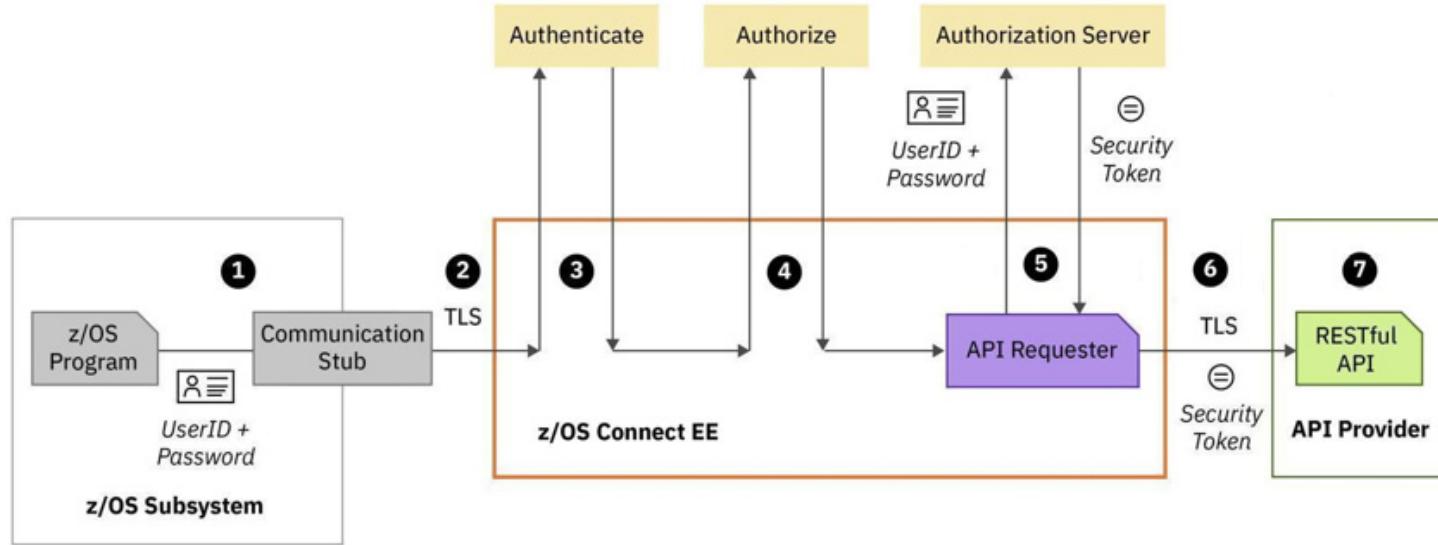
API requester security – overview



z/OS Connect EE



Typical z/OS Connect EE API Requester security flow



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	Action	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 local 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/>