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 sub-systems, e.g.
  - CICS
  - Db2
  - IMS/TM
  - IMS/DB
  - MQ
  - MVS Batch
  - Outbound REST APIs
  - 3270 screen based applications (HATS)
  - IBM DVM
  - IBM File Manager
- z/OS Connect Security

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

z/OS Connect EE

z/OS Connect EE

- CICS
- IMS/TM
- IMS/DB
- Db2
- MQ
- IBM File Manager
- 3270
- IBM DVM
- MVS
- WAS
- Custom*

* Other Vendors or your own implementation

/but_first, what_is_REST?

What makes an API “RESTful”?

REST is an Architectural Style

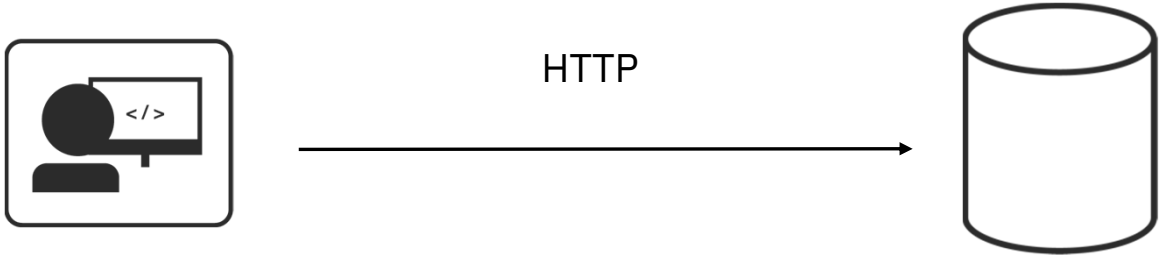
z/OS Connect EE

REST stands for **Re**presentational **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

z/OS Connect EE

Path and Query parameters are used for refinement of the request

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

GET
POST
PUT
DELETE

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

URIs represent things (or lists of things)

Request/Response Body is used to represent the data object

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

REST vs RESTful

- REST is an architectural style of development having these principles plus..
- It should be stateless
- It should access all the resources from the server using only URI
- For performing CRUD operations, it should use HTTP verbs such as get, post, put and delete
- It should return the result only in the form of JSON

- REST based services follow some of the above principles and not all, whereas RESTful means it follows all the above principles.

- Remember - Not all REST APIs are RESTful APIs
- The key is consistency, RESTful APIs are consistent, REST APIs are not

RESTful Examples

z/OS Connect Enterprise Edition:

POST `/account?name=Fred` +  *(JSON with Fred's information)*

GET `/account?number=1234`

PUT `/account?number=1234` +  *(JSON 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

z/OS Connect EE

(How to know if you are doing it wrong)

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

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

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

```
POST http://www.acme.com/customers
BODY { "firstName": "Joe",
       "lastName" : "Bloggs",
       "addr"     : "10 Old Street",
       "phoneNo"  : "01234 0123456" }
```



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

3. Operational data embedded in the request body

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



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

Why is REST popular?

z/OS Connect EE

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?

z/OS Connect EE

It is more than just an API framework

OPEN API INITIATIVE

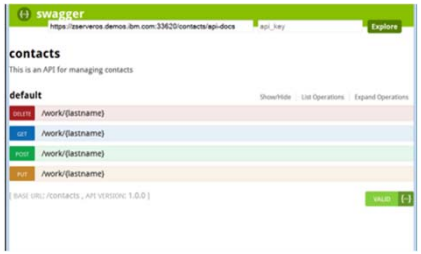
There are a number of tools available to aid consumption:

Consume Swagger

Swagger Codegen create stub code to consume APIs from various languages

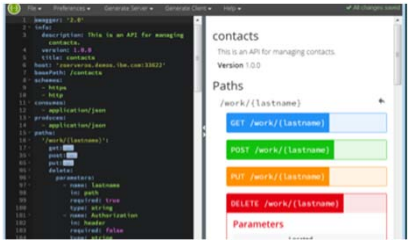
Read Swagger

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



Write Swagger

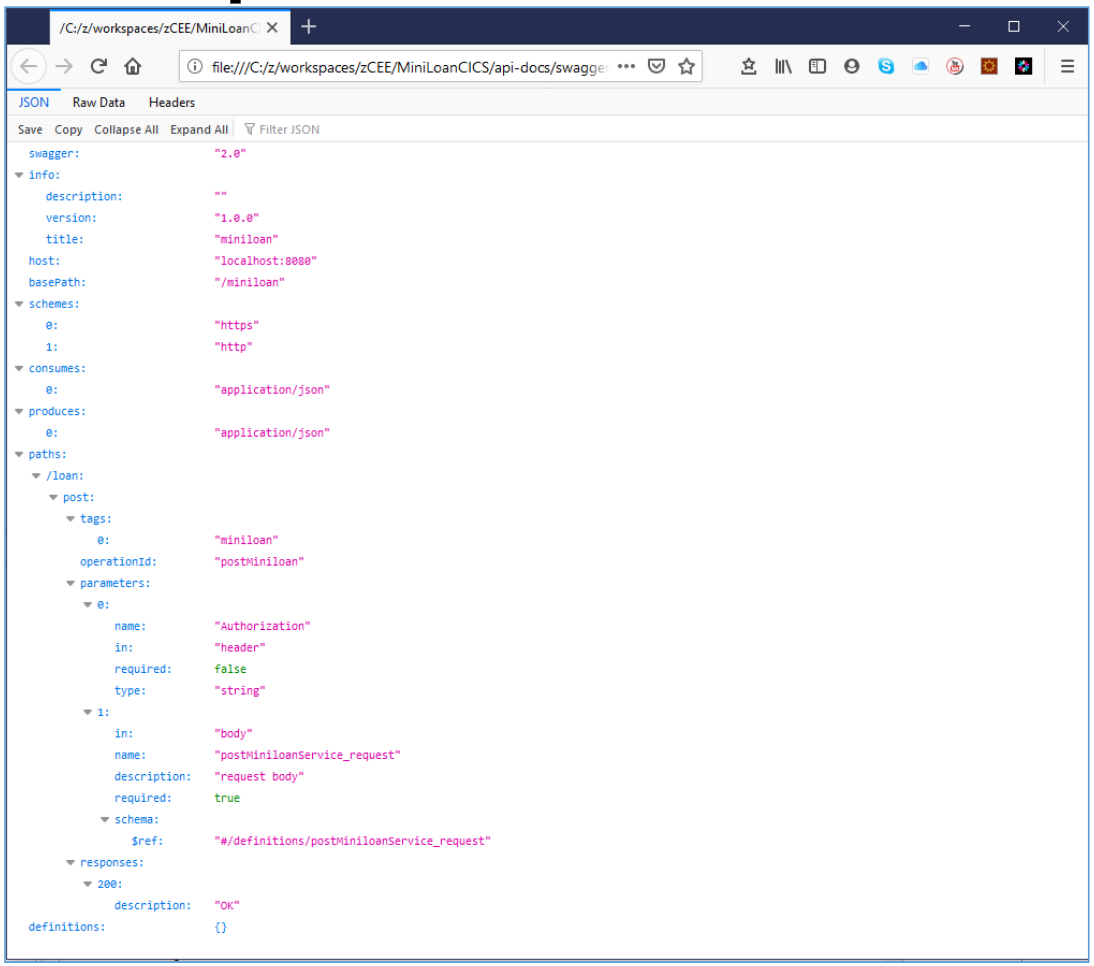
Swagger Editor allows API developers to design their swagger documents.



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

Example: https://developer.psa-peugeot-citroen.com/inc/

Swagger Example

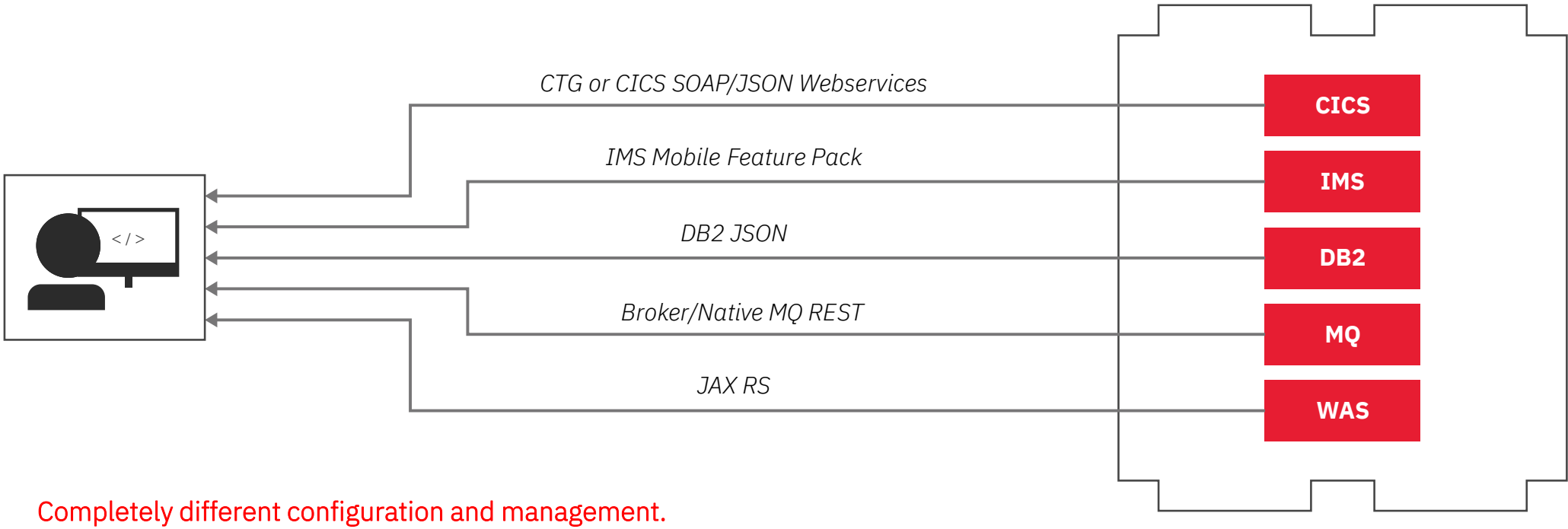


Why /zos_connect_ee?

Truly RESTful APIs to and from your mainframe.

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

z/OS Connect EE



Completely different configuration and management.

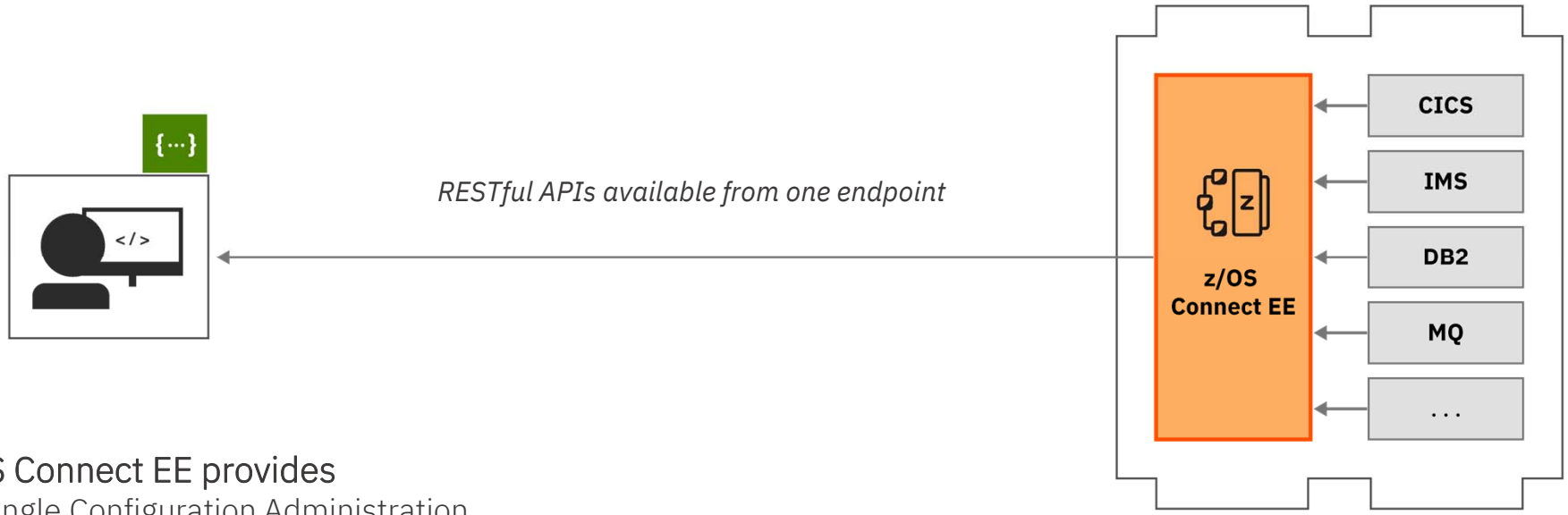
Multiple endpoints for developers to call/maintain access to.

These are typically not RESTful!

A single entry point was needed

z/OS Connect EE

Expose z/OS resources without writing any code.



z/OS Connect EE provides

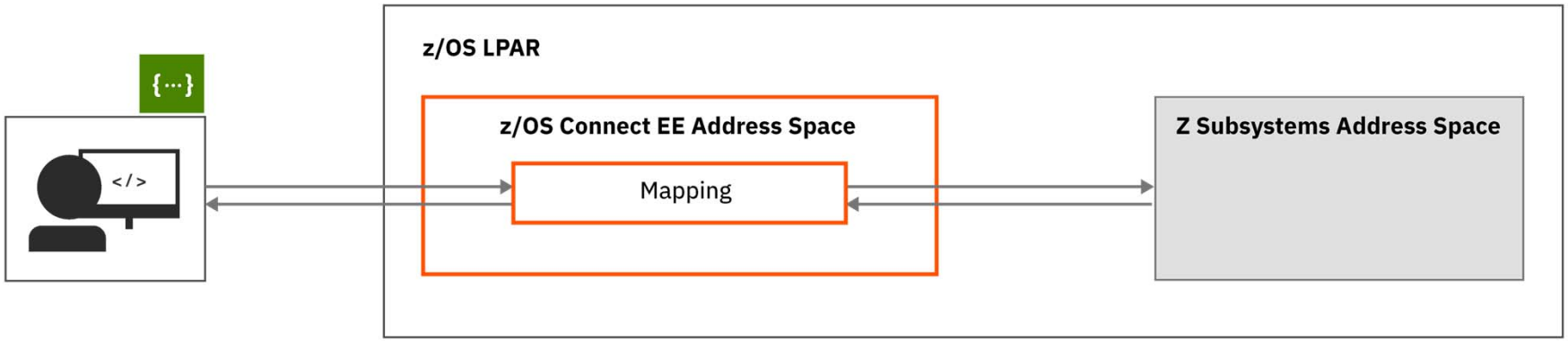
- ❑ Single Configuration Administration
- ❑ Single Security Administration
- ❑ With sophisticated mapping of truly RESTful APIs to existing mainframe and services data without writing any code.

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

Let’s Start with Data mapping

z/OS Connect EE

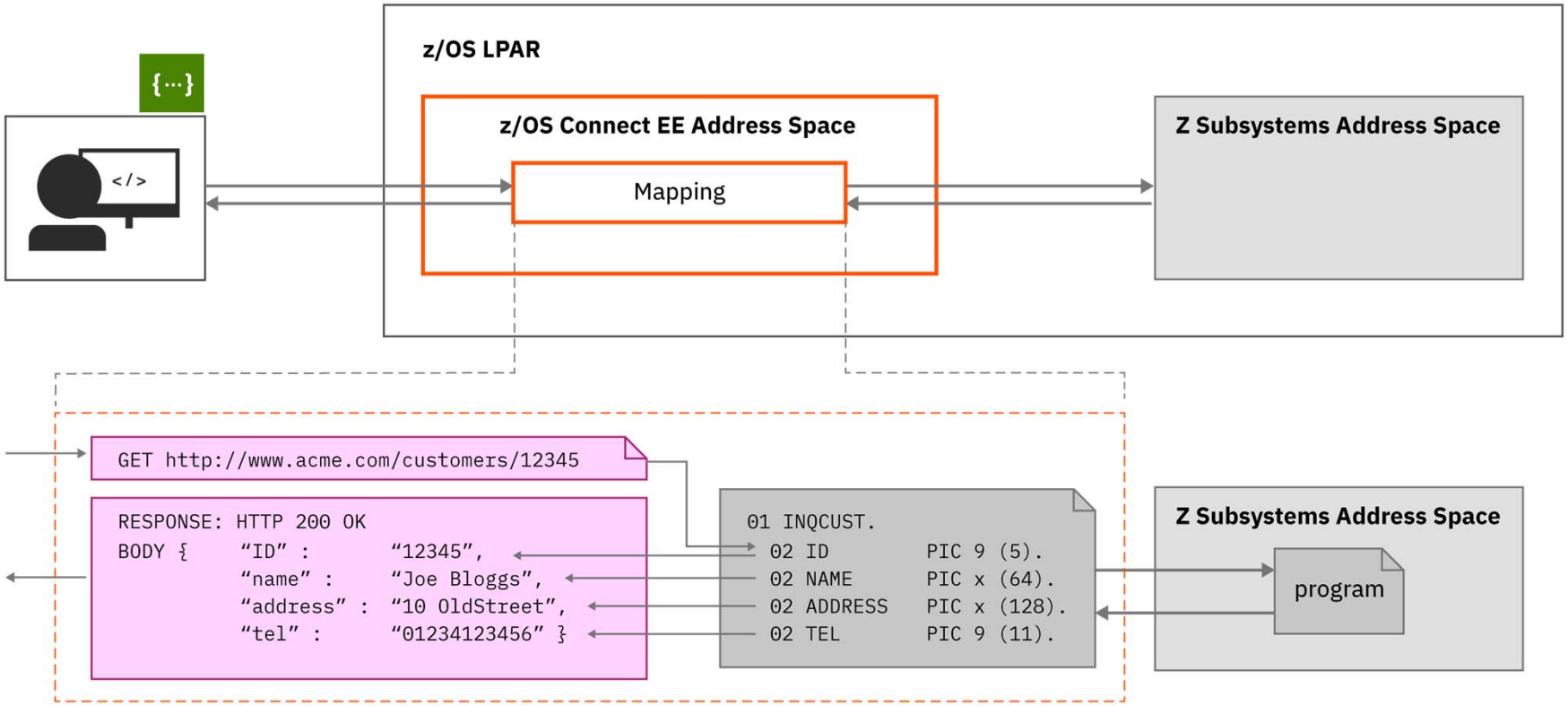
Converting JSON to the target’s subsystem’s normal format



Data mapping Example

z/OS Connect EE

A closer look



COBOL versus JSON Example

z/OS Connect EE

```
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":{
                    "type":"string",
                    "maxLength":20
                 },
                 "creditScore":{
                    "type":"number",
                    "format":"decimal",
                    "multipleOf":0.01,
                    "maximum":9999999999999999.99,
                    "minimum":0
                 },
```

"name":"Mitch Johnson",
"creditScore":"72000"

All data is sent as character strings and numeric precision and sign bit is removed as an issue

minilonaServiceResponse.si

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:message xmlns:ns2="http://www.ibm.com/ims/Transaction" transactionCode="" messageName="minilonaServiceResponse" dire
    <message id="1" name="minilonaServiceResponse">
        <segment id="1" name="COMMAREA" originalName="COMMAREA">
            <field name="MINILOAN_COMMAREA" originalName="MINILOAN_COMMAREA" included="Y" path="MINILOAN_COMMAREA">
                <startPos>1</startPos>
                <bytes>6073</bytes>
                <maxBytes>0</maxBytes>
                <applicationDatatype datatype="STRUCT"/>
                <field name="name" originalName="NAME" included="Y" defaultValue="" isHex="false" path="MINILOAN_COMMAREA
                    <startPos>1</startPos>
                    <bytes>20</bytes>
                    <maxBytes>20</maxBytes>
                    <remarks></remarks>
                    <applicationDatatype datatype="CHAR"/>
                </field>
                <field name="creditScore" originalName="CREDITSCORE" included="Y" defaultValue="" isHex="false" path="MINI
                    <startPos>21</startPos>
                    <bytes>18</bytes>
                    <maxBytes>18</maxBytes>
                    <remarks></remarks>
                    <marshaller isSigned="N" isSignLeading="N" isSignSeparate="N" isWCHAROnly="N">
                        <typeConverter>ZONEDDECIMAL</typeConverter>
                    </marshaller>
                    <applicationDatatype datatype="DECIMAL" precision="18" scale="0"/>
                </field>
```

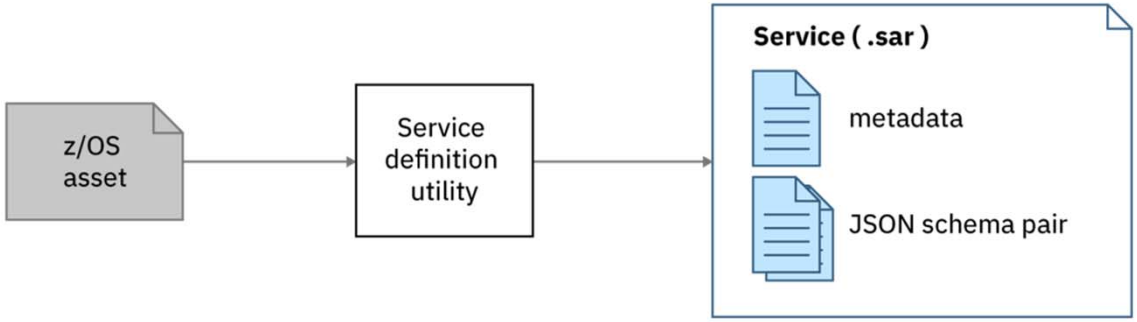
Steps to expose a z/OS application

z/OS Connect EE

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

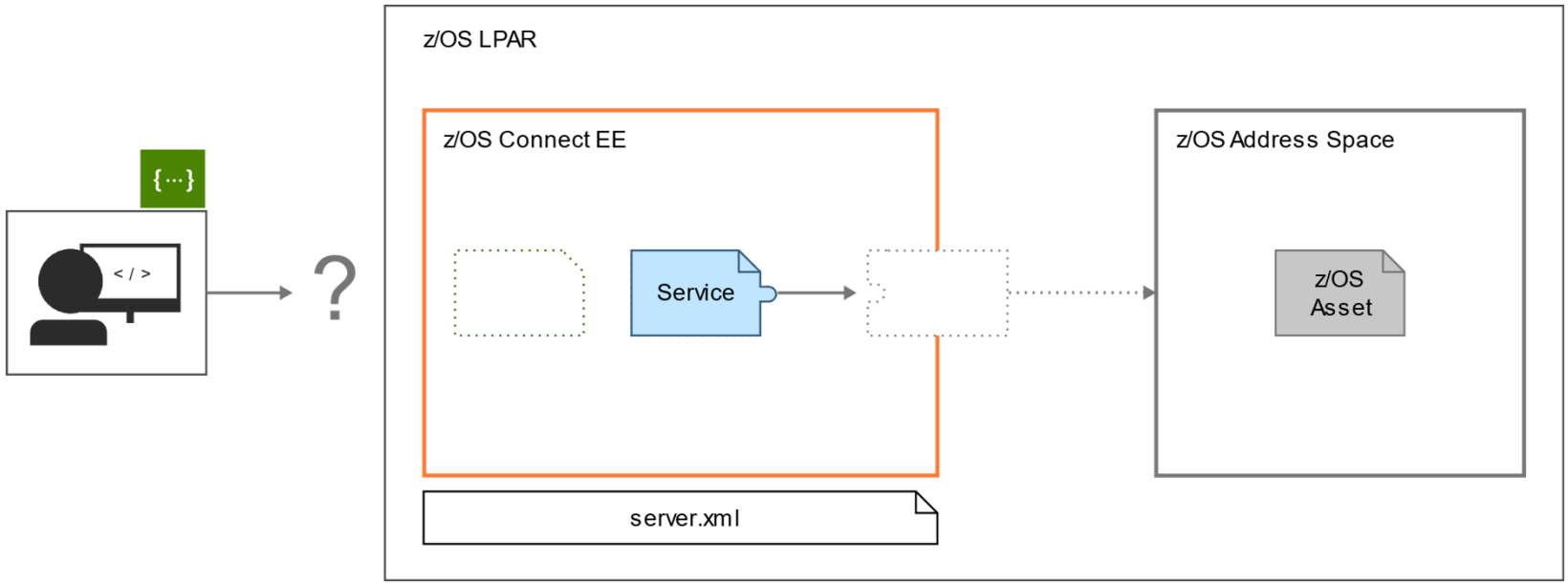
- API Toolkit (CICS, IMS TM, IMS DB, Db2 and MQ)
- z/OS Connect EE Build Toolkit (MVS Batch, IBM File Manager and HATS)
- DVM Toolkit

ibm.biz/zosconnect-sar-creation

Steps to expose a z/OS application

z/OS Connect EE

2. Deploy and export the service



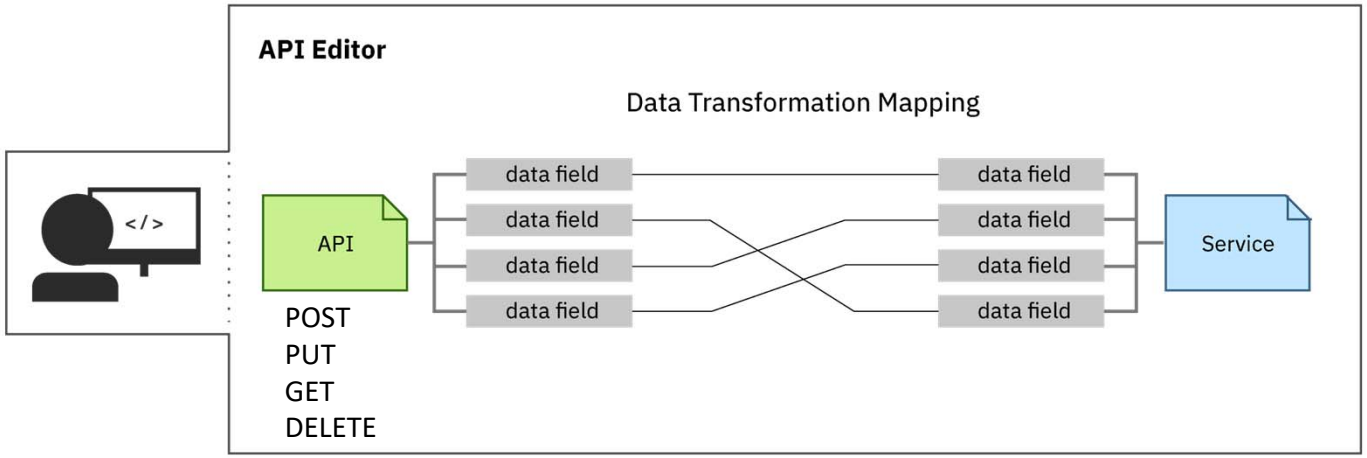
Deploy the `.sar` file generated in **Step 2** using the right-click deploy in **the API toolkit**, or by copying the `.sar` file to the services directory.

ibm.biz/zosconnect-define-services

Steps to expose a z/OS application

z/OS Connect EE

3. Create an API using exported services



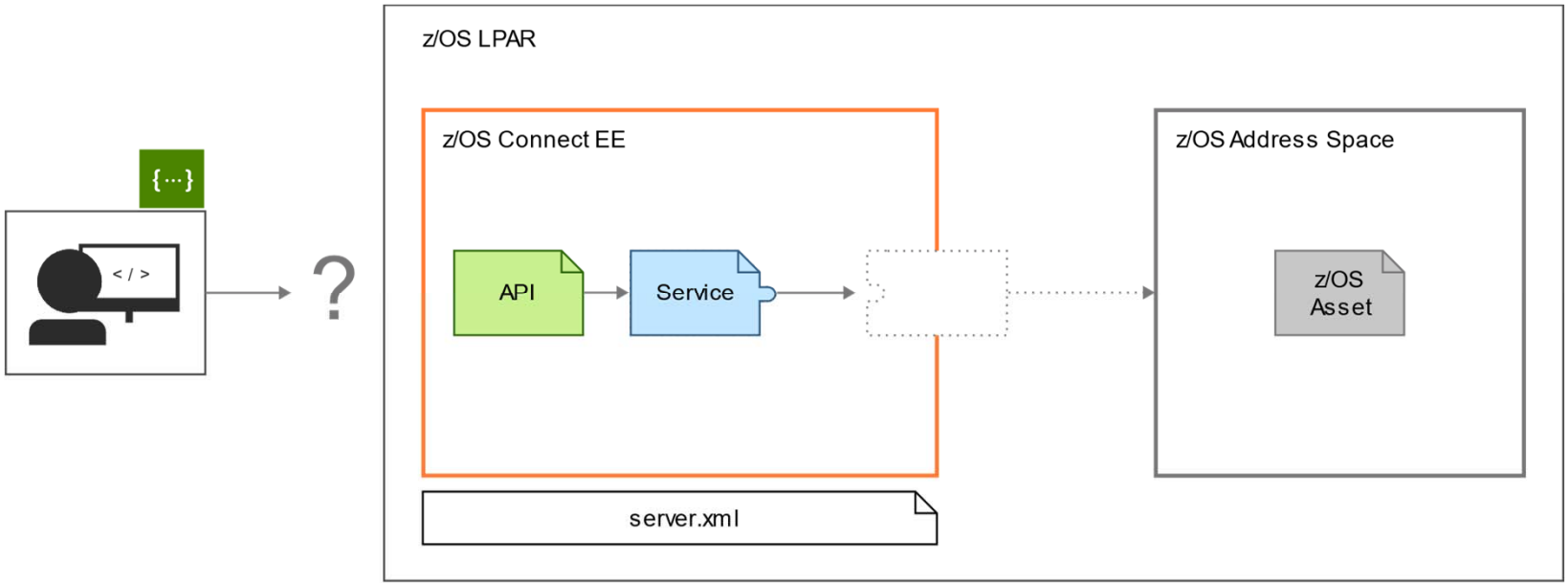
- Import your `.sar` file into the **API toolkit**, and start designing your RESTful API.
- Provides additional data mapping
- From the editor, create an **API Archive file** (`.aar`), which describes your API and how it maps to underlying services.

ibm.biz/zosconnect-create-api

Steps to expose a z/OS application

z/OS Connect EE

4. Deploy the API



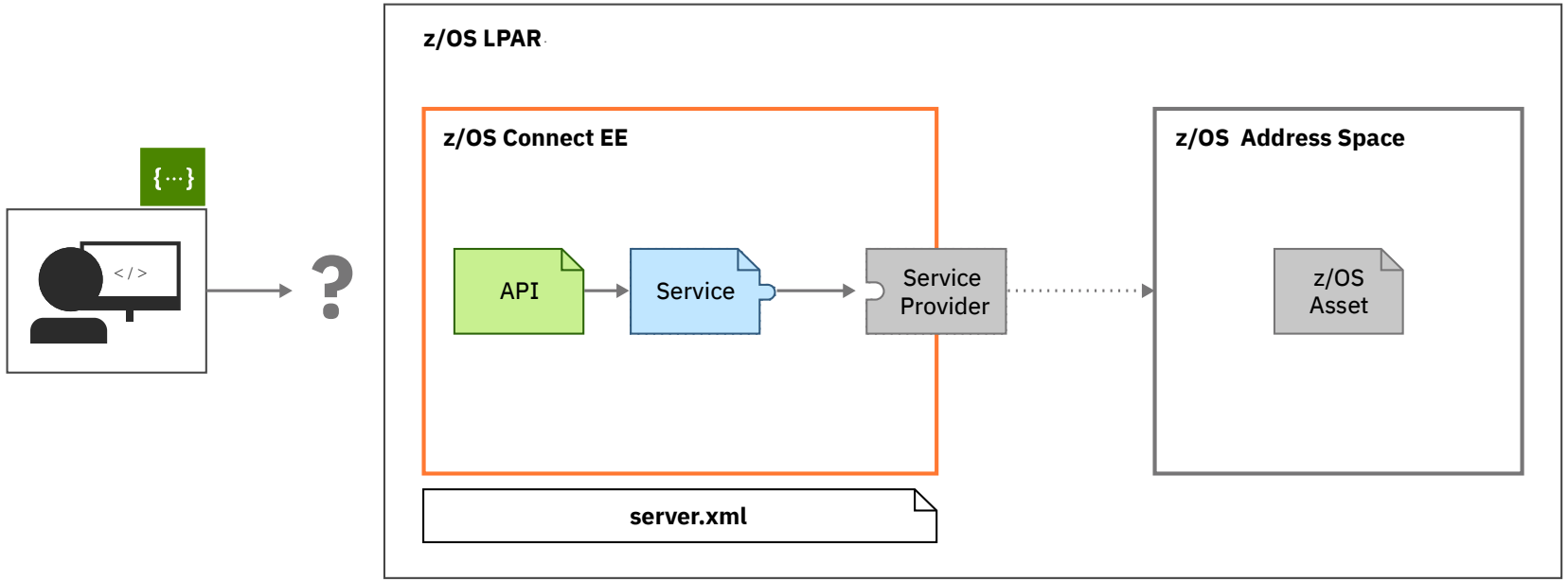
Deploy your API using the right-click deploy in **the API toolkit**, or by copying the `.aar` file to the *apis* directory.

ibm.biz/zosconnect-deploy-api

Steps to expose a z/OS application

z/OS Connect EE

5. Configure the service provider



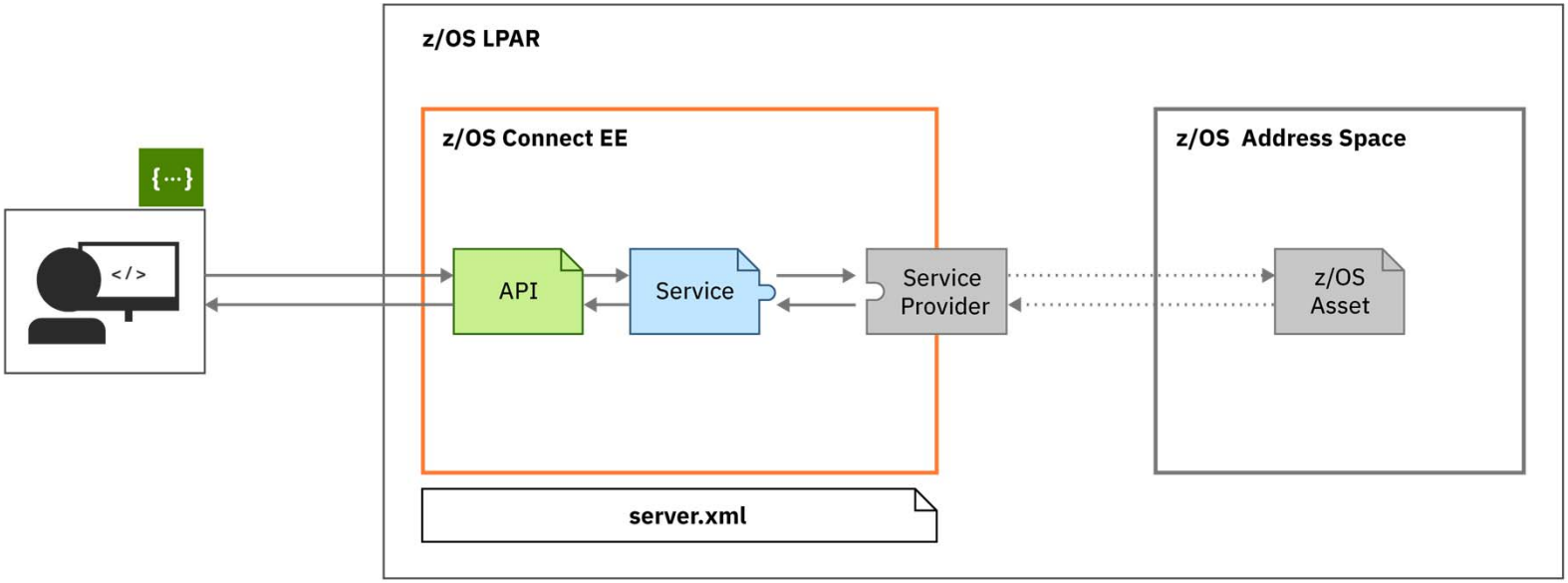
Configure the system-appropriate service provider to connect to your backend system in your `server.xml`.

ibm.biz/zosconnect-configuring

Steps to expose a z/OS application

z/OS Connect EE

6. Done



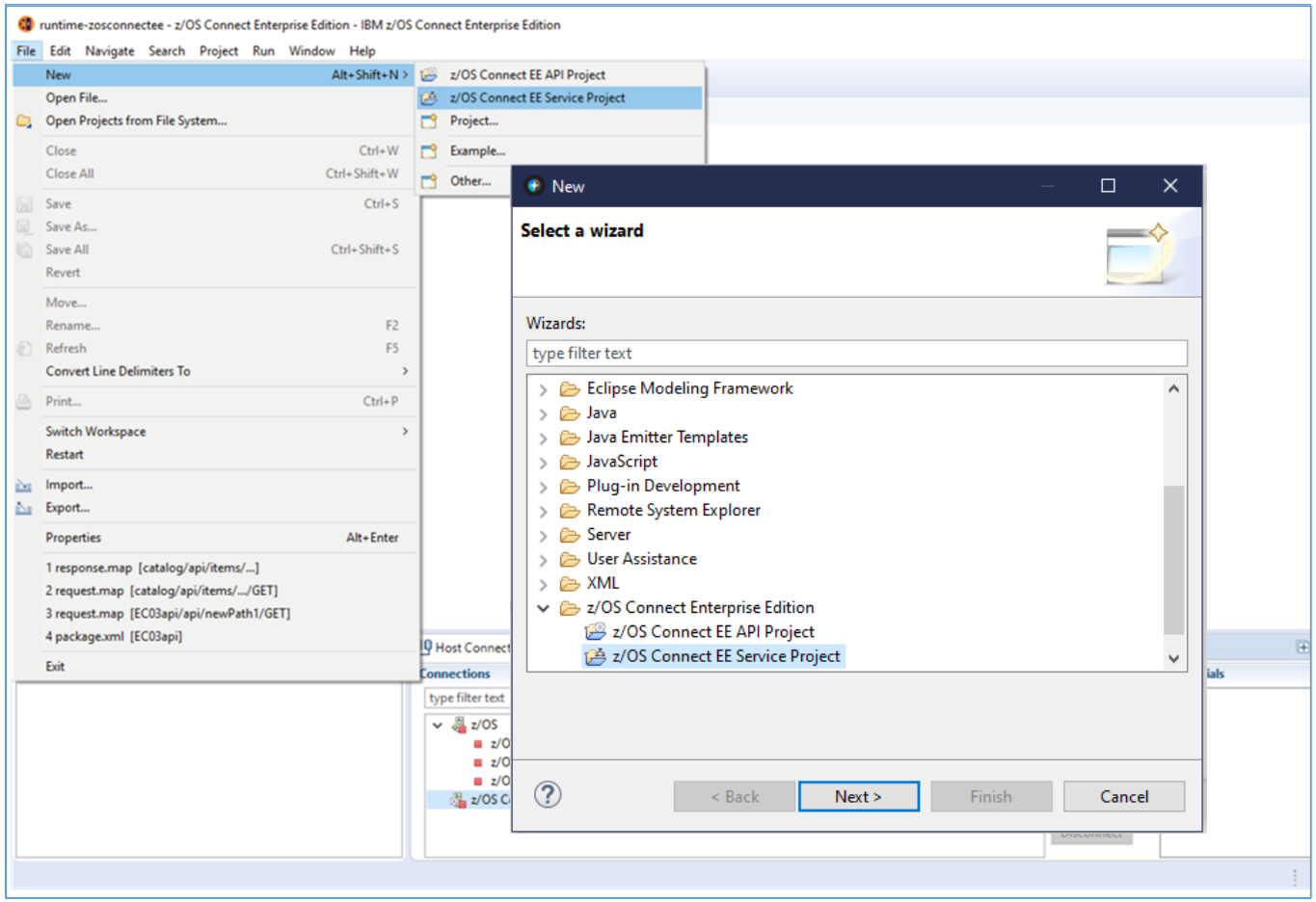
The API is ready to be consumed that requires no knowledge that a z/OS recourse is being accessed

/api_toolkit/services

Simple **service creation**.

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

z/OS Connect EE



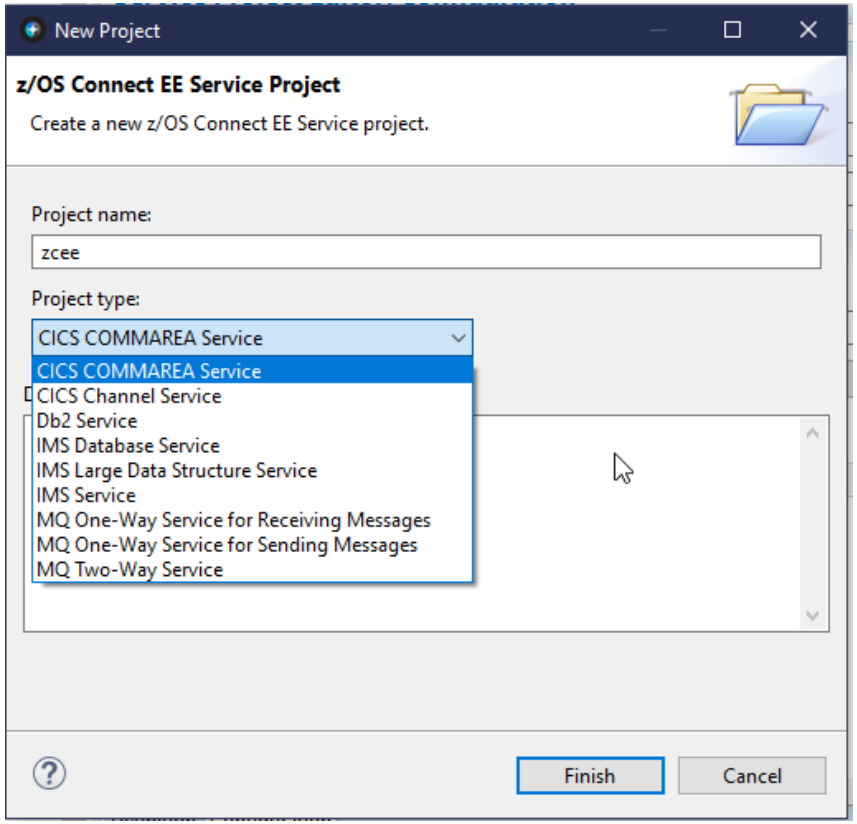
Use the **API toolkit** to create services through Eclipse-based tooling.

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

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

z/OS Connect EE

Service creation – a common interface

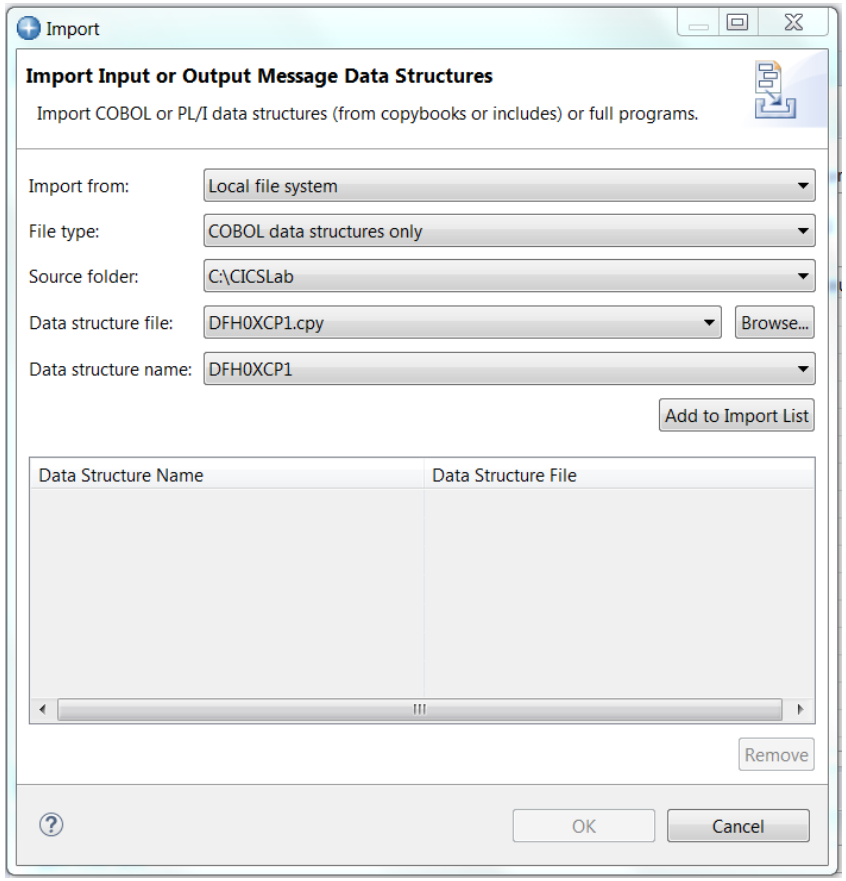


A common interface for service creation, agnostic of back end subsystem.

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

z/OS Connect EE

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



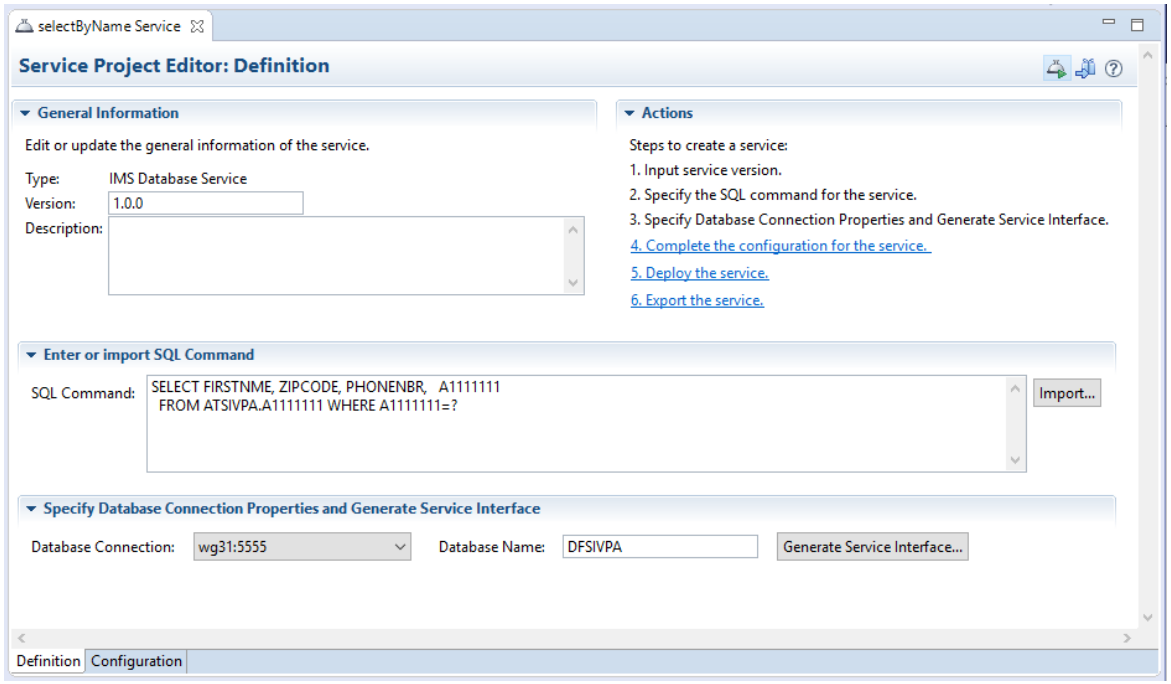
You start by importing data structures into the service interface from the local file system or the workspace.

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

API toolkit – Creating Services for IMS DB

z/OS Connect EE

Creating a service project from the IMS Catalog

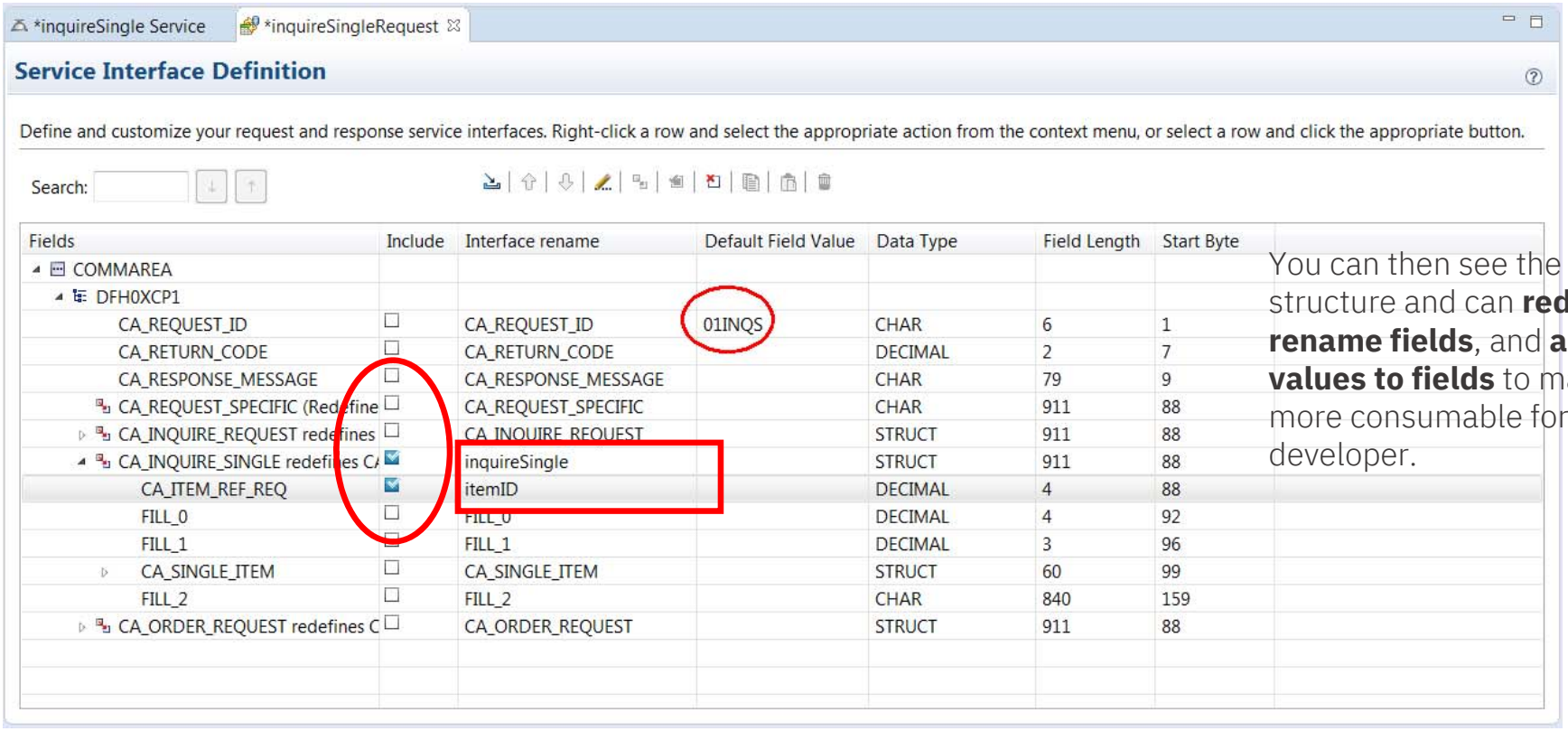


You start use the IMS Catalog to assist with developing and testing SQL SELECT commands used for accessing IMS databases.

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

z/OS Connect EE

Regardless, either allows editing a service interface definition

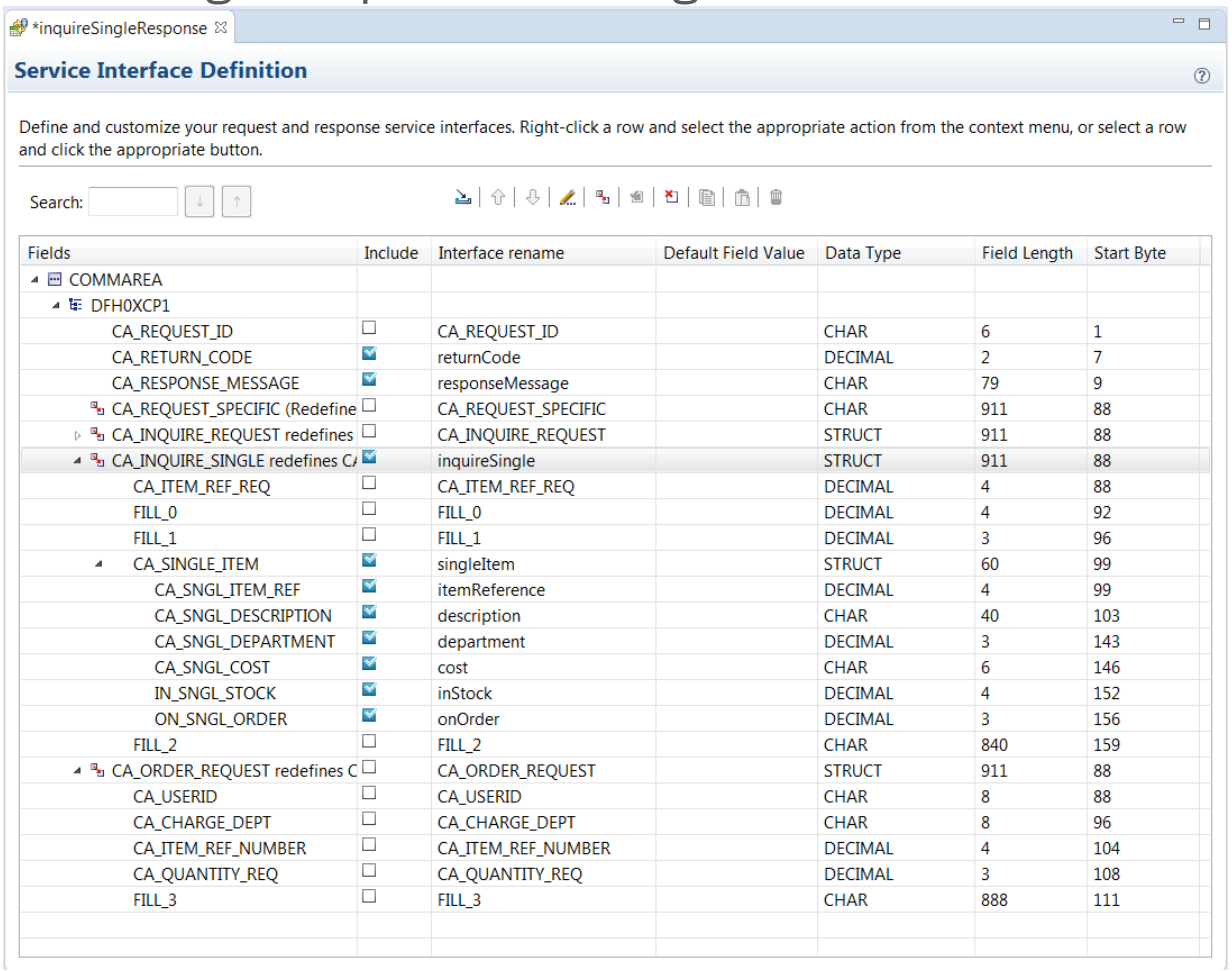


You can then see the imported data structure and can **redact fields**, **rename fields**, and **add default values to fields** to make the service more consumable for an API developer.

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

z/OS Connect EE

Editing a response message



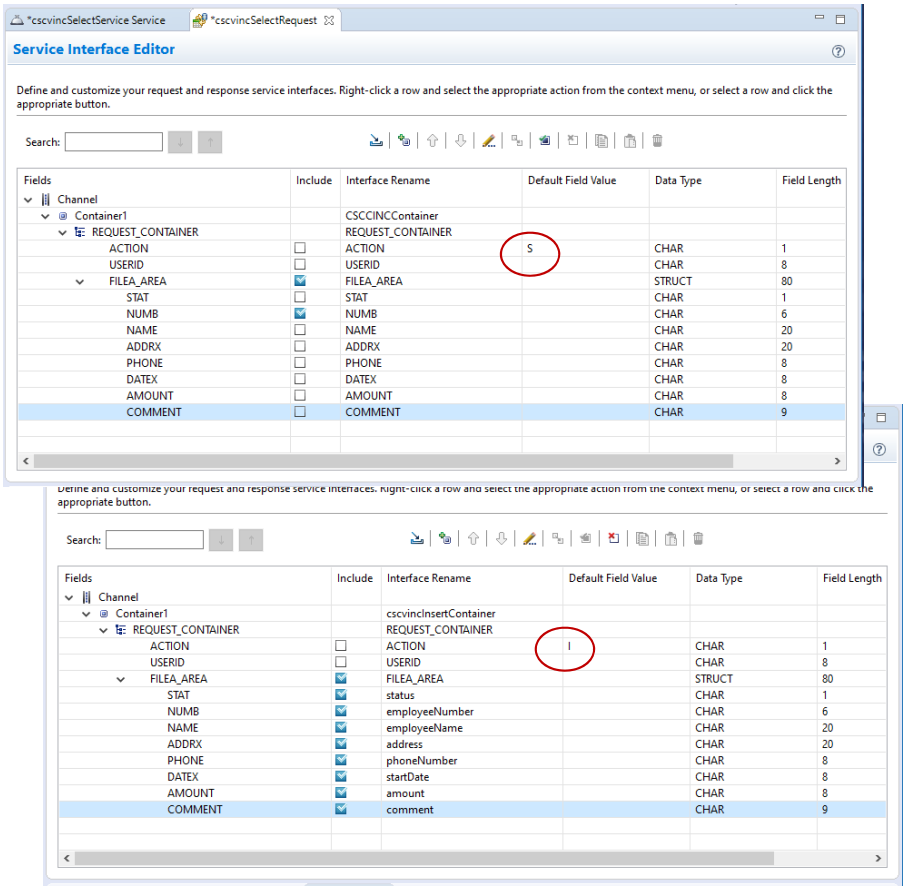
Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMAREA						
DFH0XCP1						
CA_REQUEST_ID	☐	CA_REQUEST_ID		CHAR	6	1
CA_RETURN_CODE	☑	returnCode		DECIMAL	2	7
CA_RESPONSE_MESSAGE	☑	responseMessage		CHAR	79	9
CA_REQUEST_SPECIFIC (Redefine	☐	CA_REQUEST_SPECIFIC		CHAR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

You can then see the imported data structure and can **redact fields** and **rename fields**

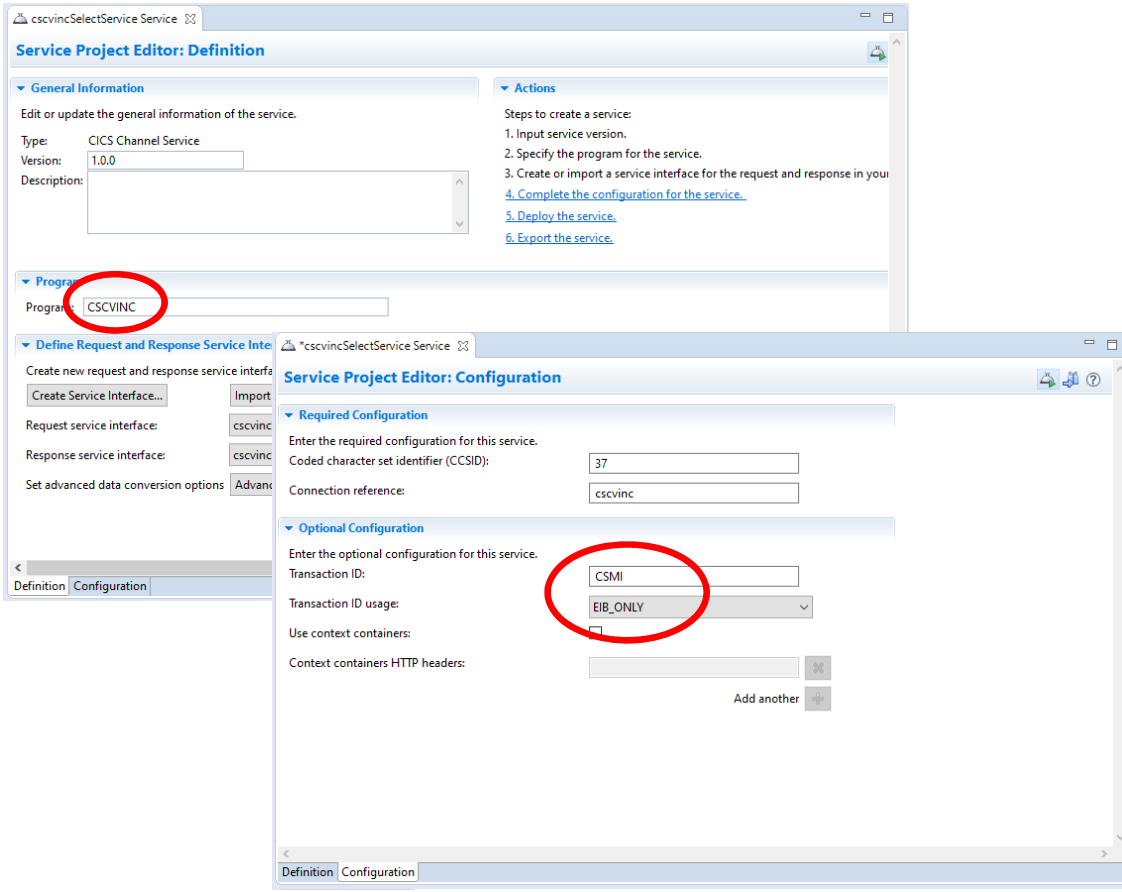
API toolkit – Creating Services for CICS

z/OS Connect EE

Creating multiple services to the same resource



CICS Meta Data



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

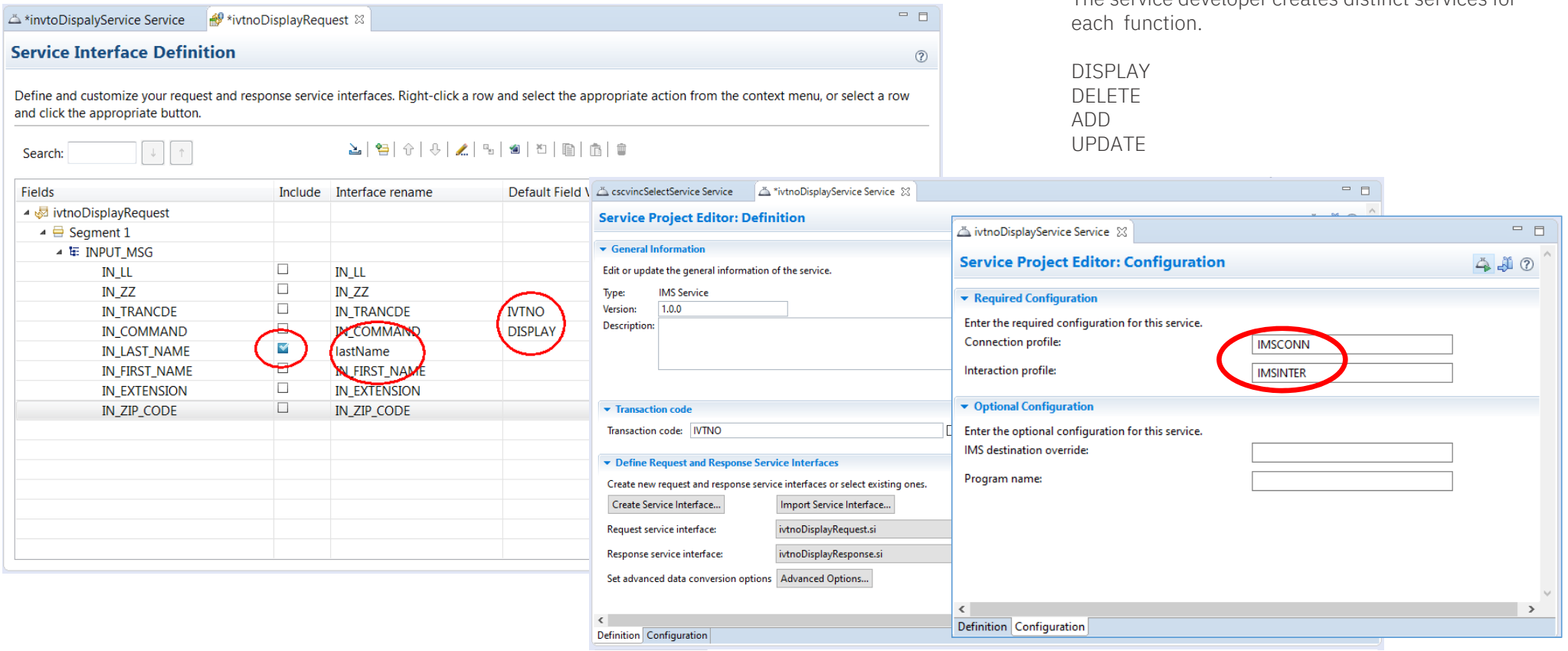
API toolkit – Creating Services for IMS

z/OS Connect EE

Creating a “GET” service interface request definition

The service developer creates distinct services for each function.

DISPLAY
DELETE
ADD
UPDATE



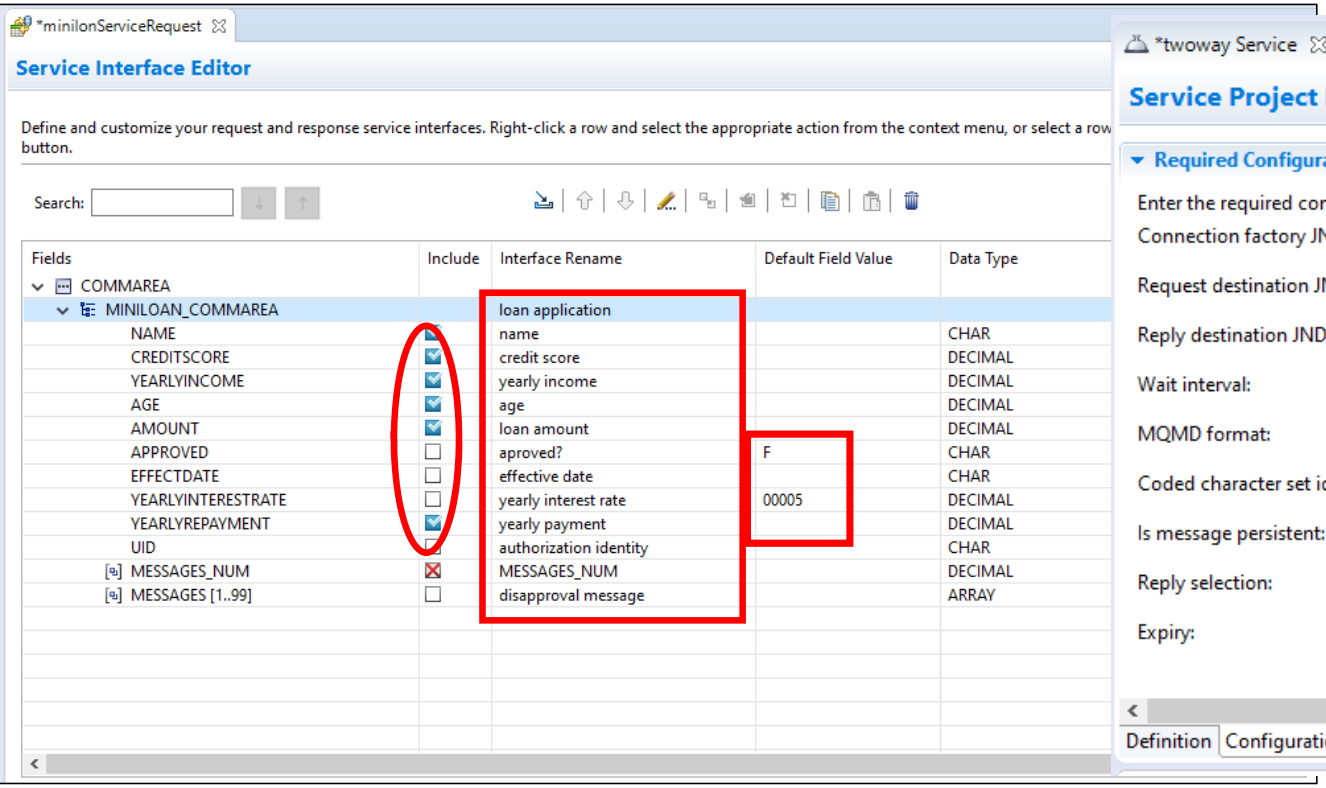
IMS/TM Meta Data

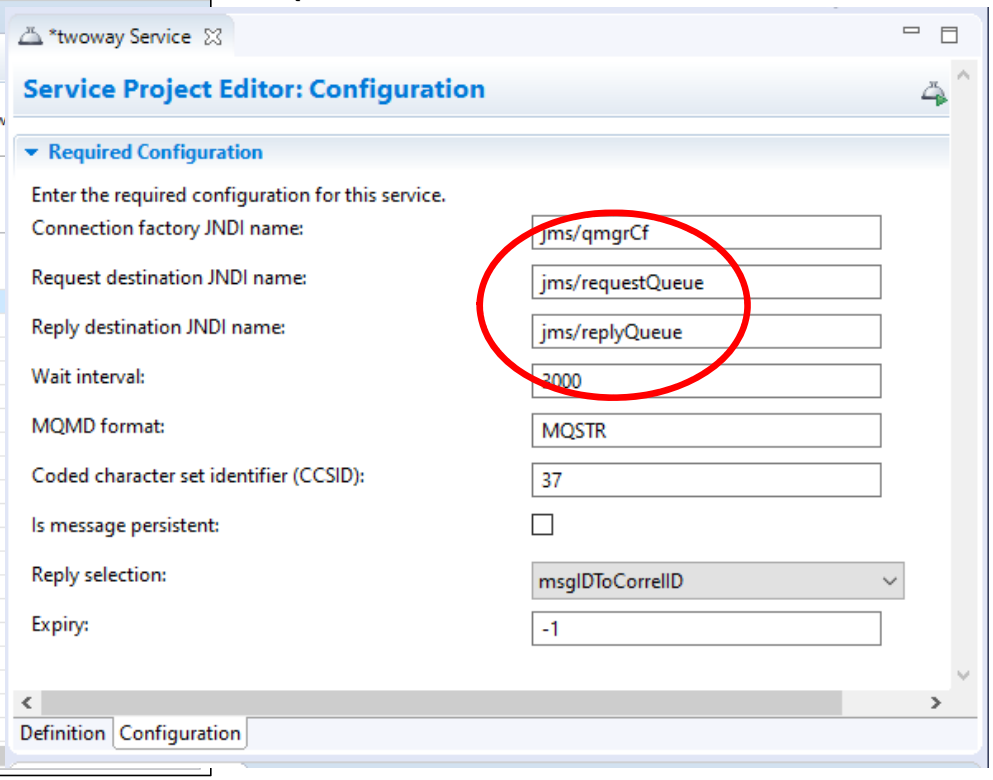
API toolkit – Creating Services for MQ

z/OS Connect EE

Creating a service interface definition

MQ Meta Data



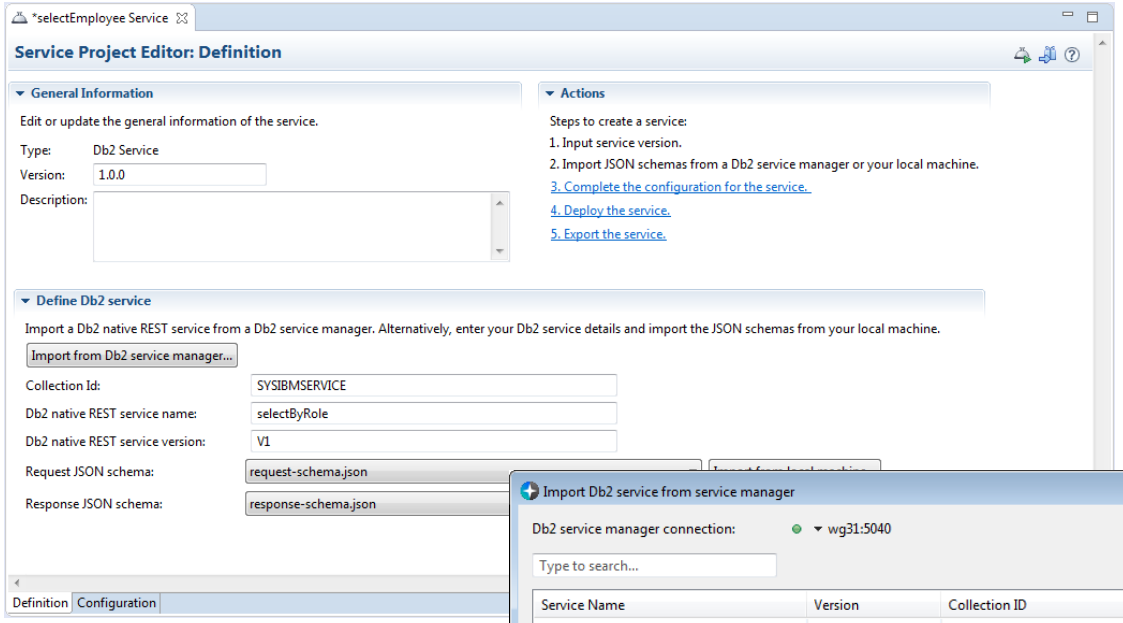


Again the service developer can then see the imported data structure and can **redact fields**, **rename fields**, and **add default values to fields** to make the service more consumable for an API developer.

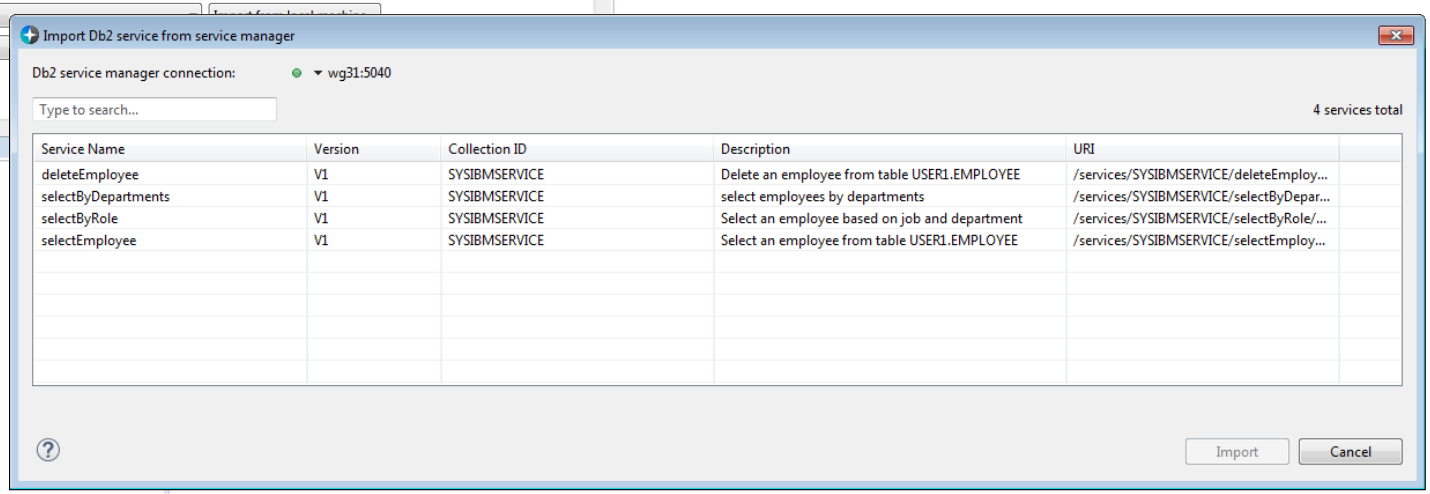
API toolkit – Creating Services for Db2

Creating a service project from Db2 REST service

z/OS Connect EE



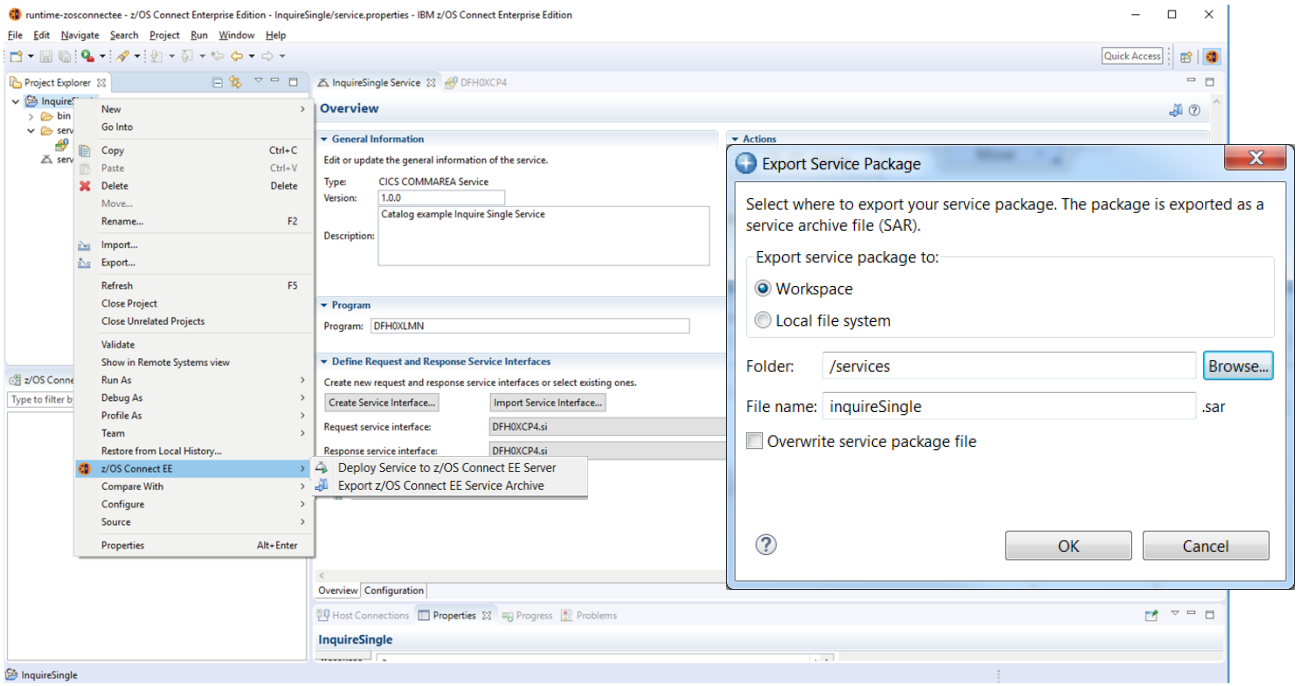

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT  DD *
  SELECT EMPNO AS "employeeNumber", FIRSTNME AS "firstName",
        MIDINIT AS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



The service developer retrieve the Db2 REST services

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

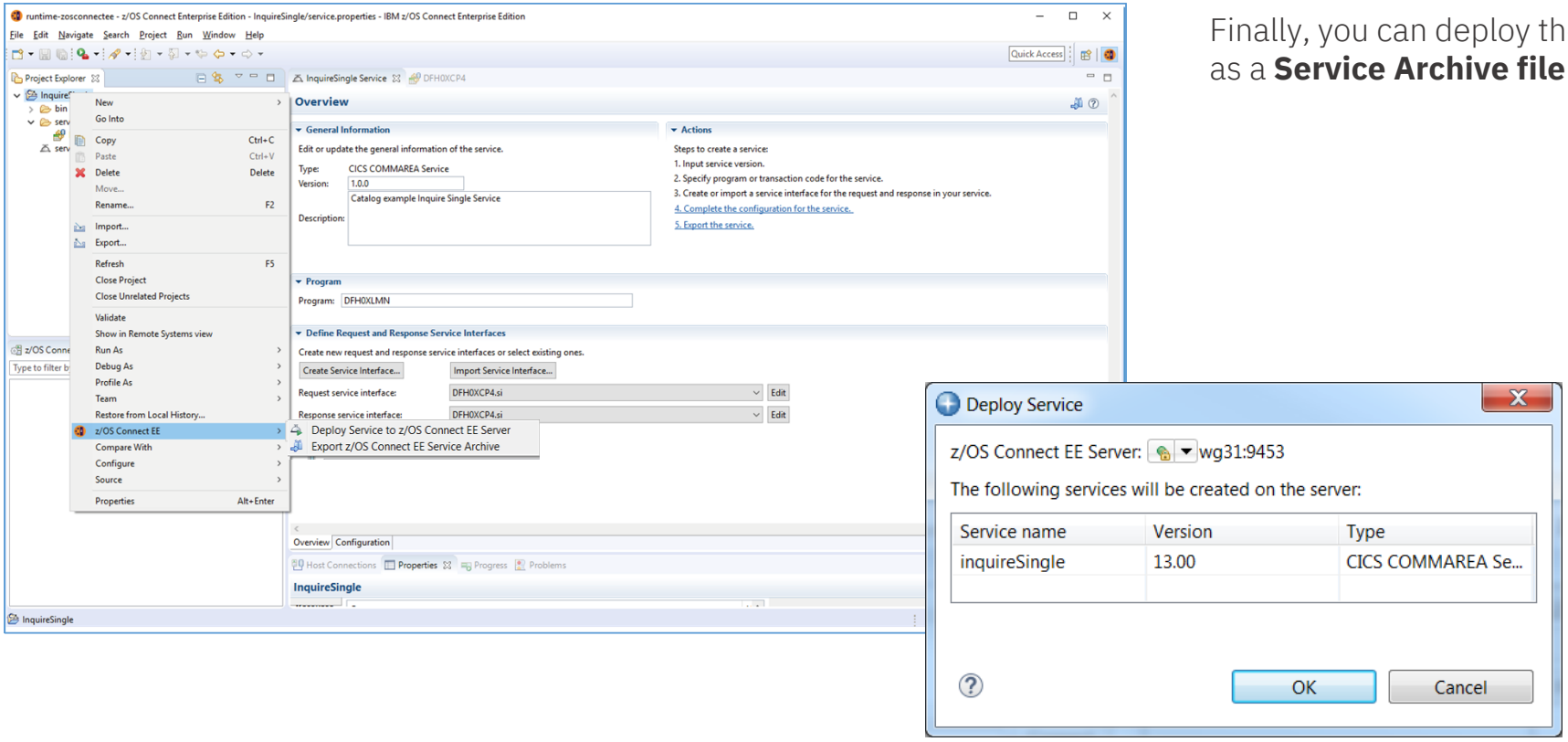
z/OS Connect EE



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

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

z/OS Connect EE

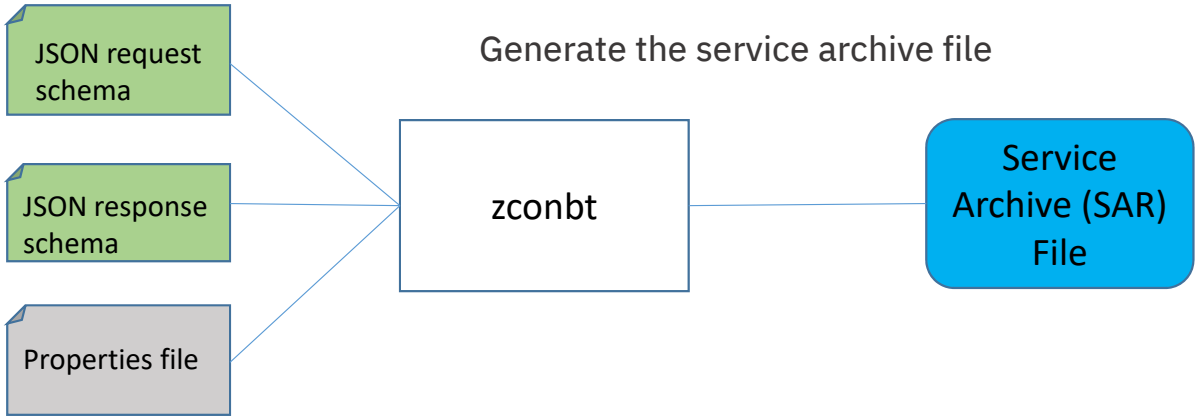


Finally, you can deploy the service project as a **Service Archive file** (`.sar`)

Creating Services without the Toolkit – REST

z/OS Connect EE

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



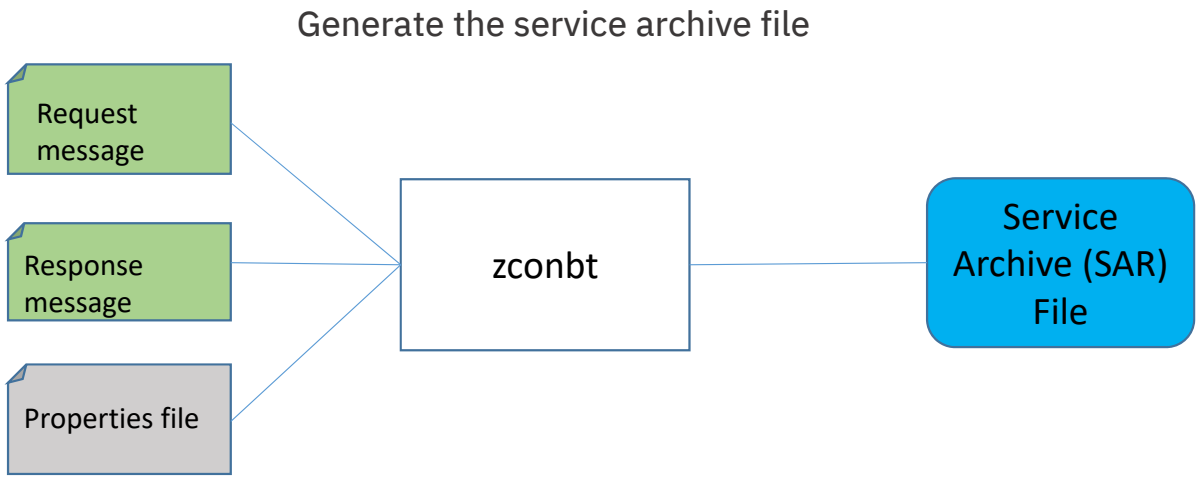

```
provider=rest
name=getCompany
version=1.0
description=Obtain a list of companies
requestSchemaFile=getCompanyRequest.json
responseSchemaFile=getCompanyResponse.json
verb=POST
uri=/Trader/rest/GetCompany
connectionRef=HatsConn
```

Creating Services without the Toolkit – Batch

z/OS Connect EE

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=filearеq.cpy
responseStructure=filearsp.cpy
```

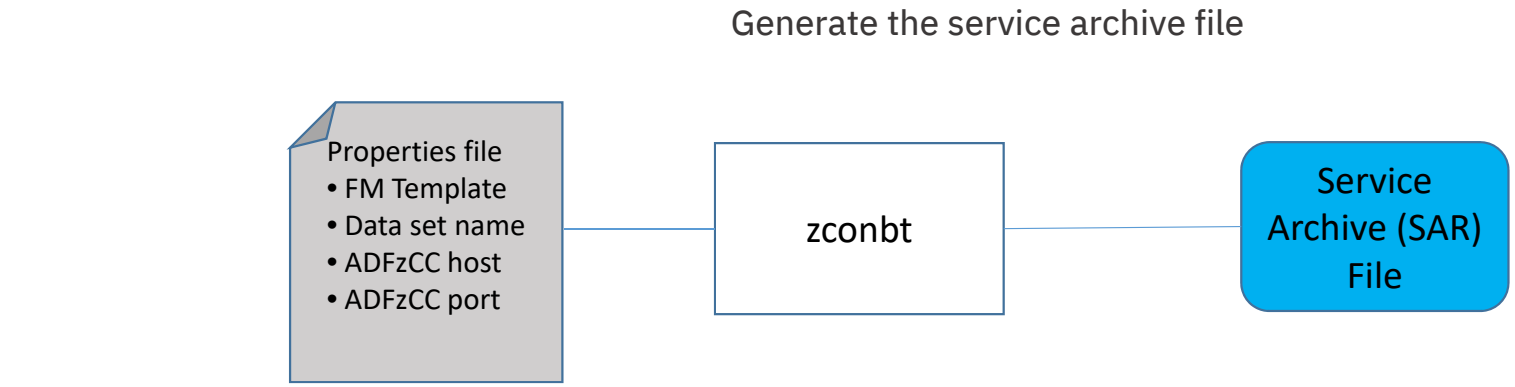


WebSphere **O**ptimized **L**ocal **A**dapter – a protocol for cross memory communications between address spaces

Creating Services without the Toolkit – FM

z/OS Connect EE

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

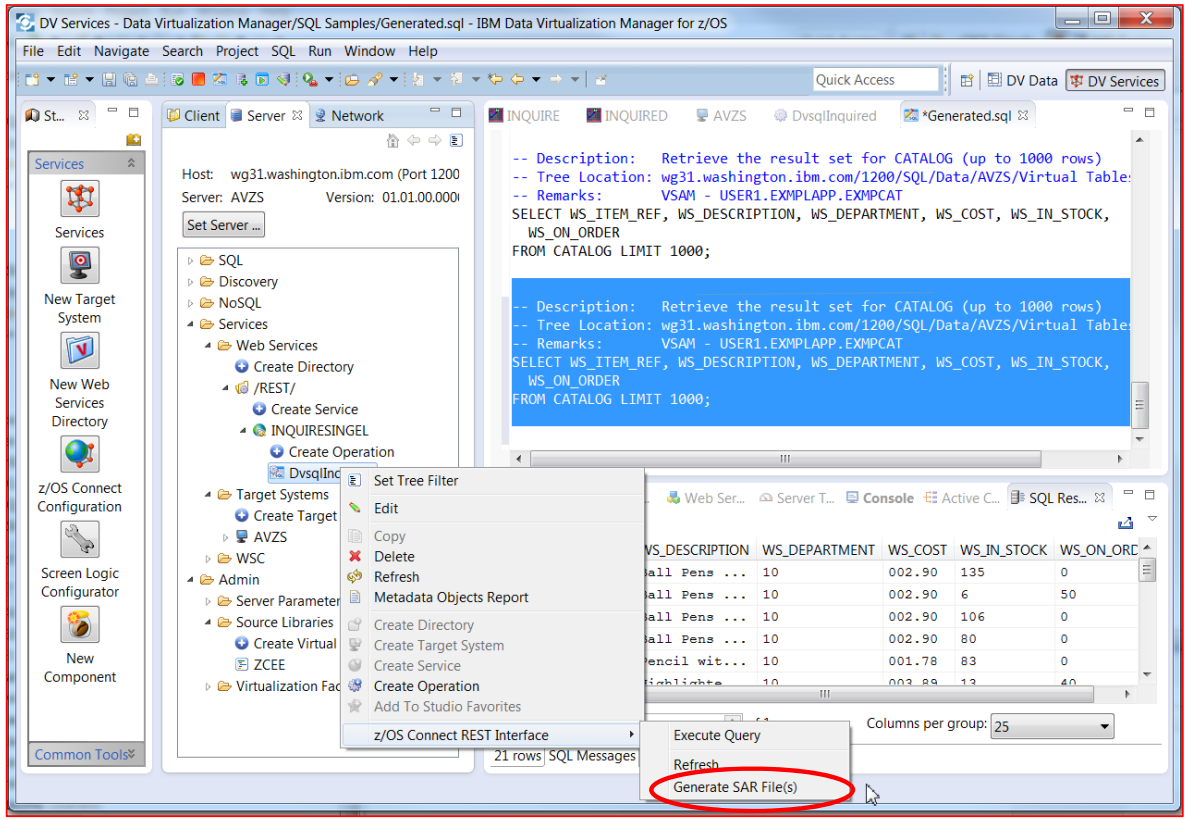



```
name=filea
provider=filemanager
host=wg31.washington.ibm.com
version=1.0
port=2800
file=USER1.ZCEE.FILEA
template=USER1.TEMPLATE(FILEA)
connid=default
userid=USER1
passwd=USER1
```

Creating Services without the Toolkit

z/OS Connect EE

For DVM use the DVM Studio



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

Quick and easy **API mapping.**

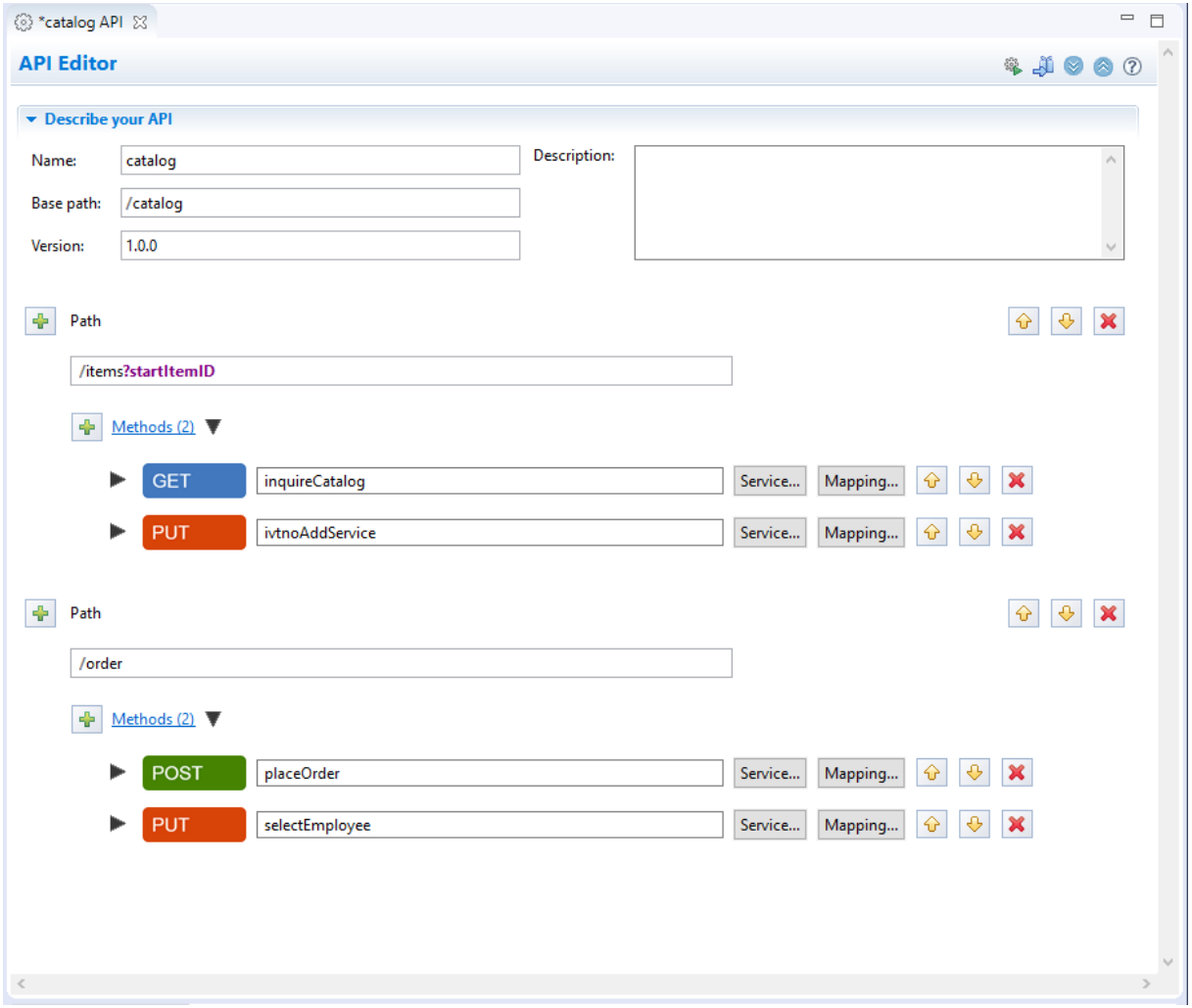
Remember: All service archives files are functionally equivalent regardless of how there are created

/api_toolkit/api_editor

Quick and easy **API mapping.**

API toolkit – API Editor

z/OS Connect EE



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

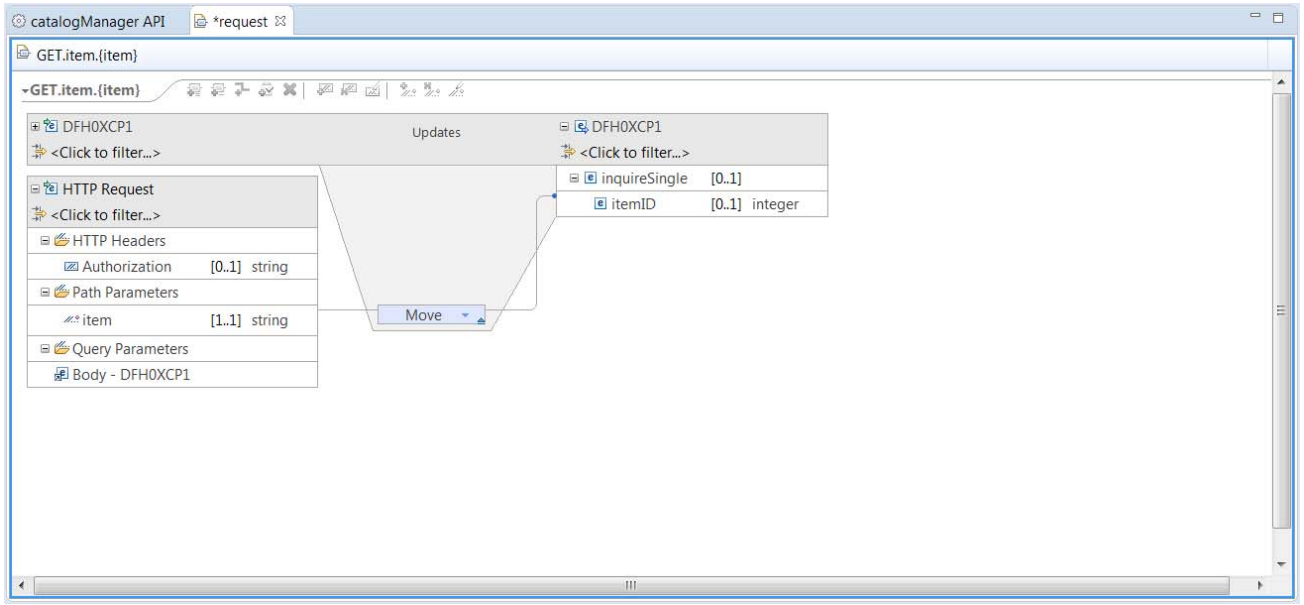
Once you define your API, you can map backend services to each request.

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

API toolkit – API Editor

z/OS Connect EE

API mapping: Point-and-click interface



Map both the request and response for each API.

Map path and query parameters to native data structures.

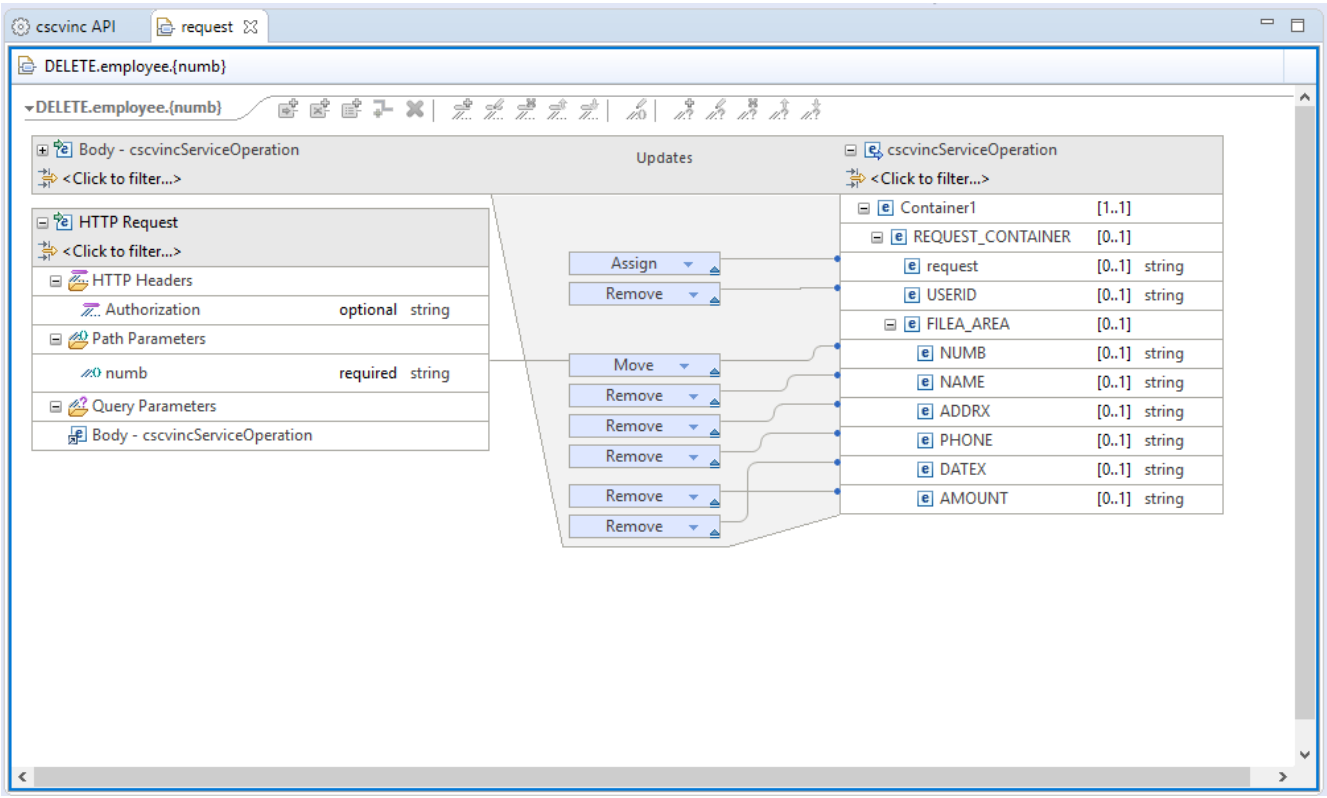
Assign static values to fields, useful for Op codes.

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

API toolkit – API Editor

z/OS Connect EE

API mapping: Point-and-click interface

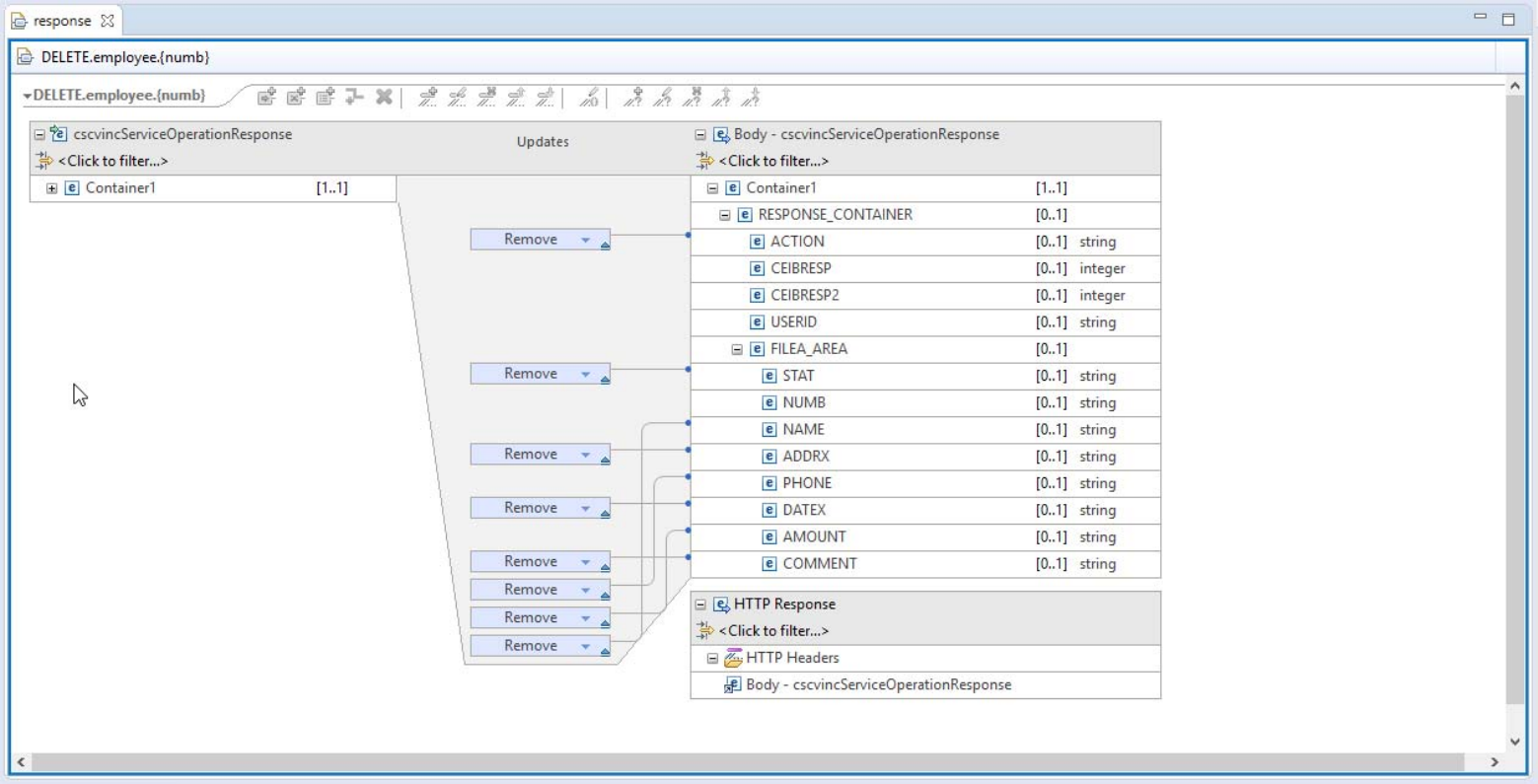


API toolkit – API Editor

z/OS Connect EE

API mapping: Point-and-click interface

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

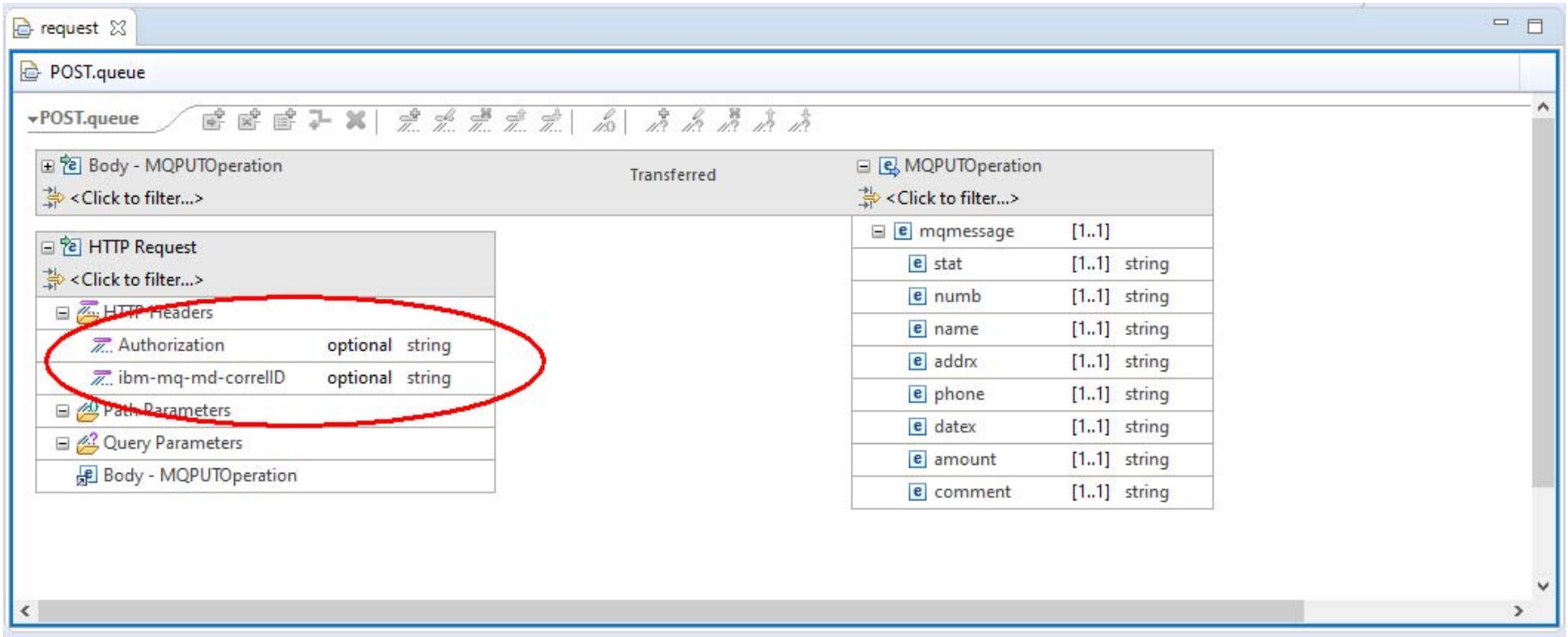


API toolkit – API Editor

z/OS Connect EE

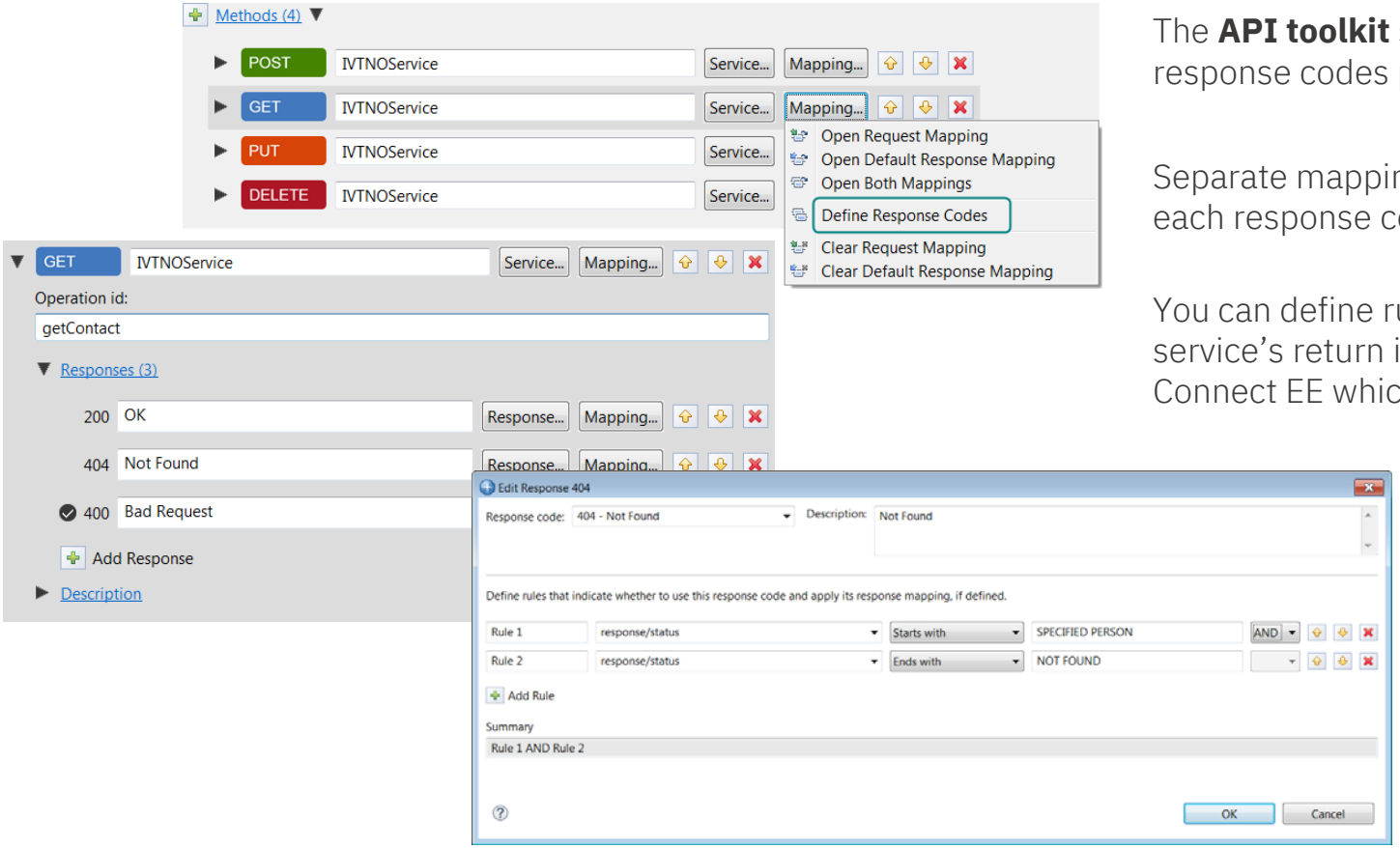
API mapping: Adding HTTP header properties

Allows the API Developer to remove fields from the response



API toolkit

API definition with multiple response codes



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

Separate mappings can be defined for each response code.

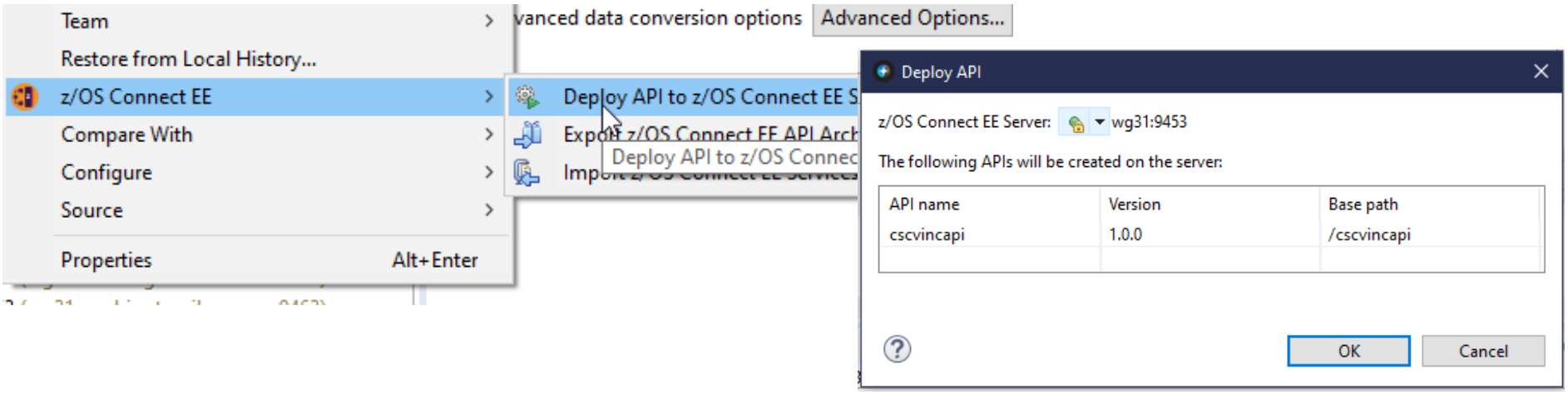
You can define rules based on fields in the service's return interface to tell z/OS Connect EE which response code to return

API toolkit – API Editor

z/OS Connect EE

Server connection and API deployment

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



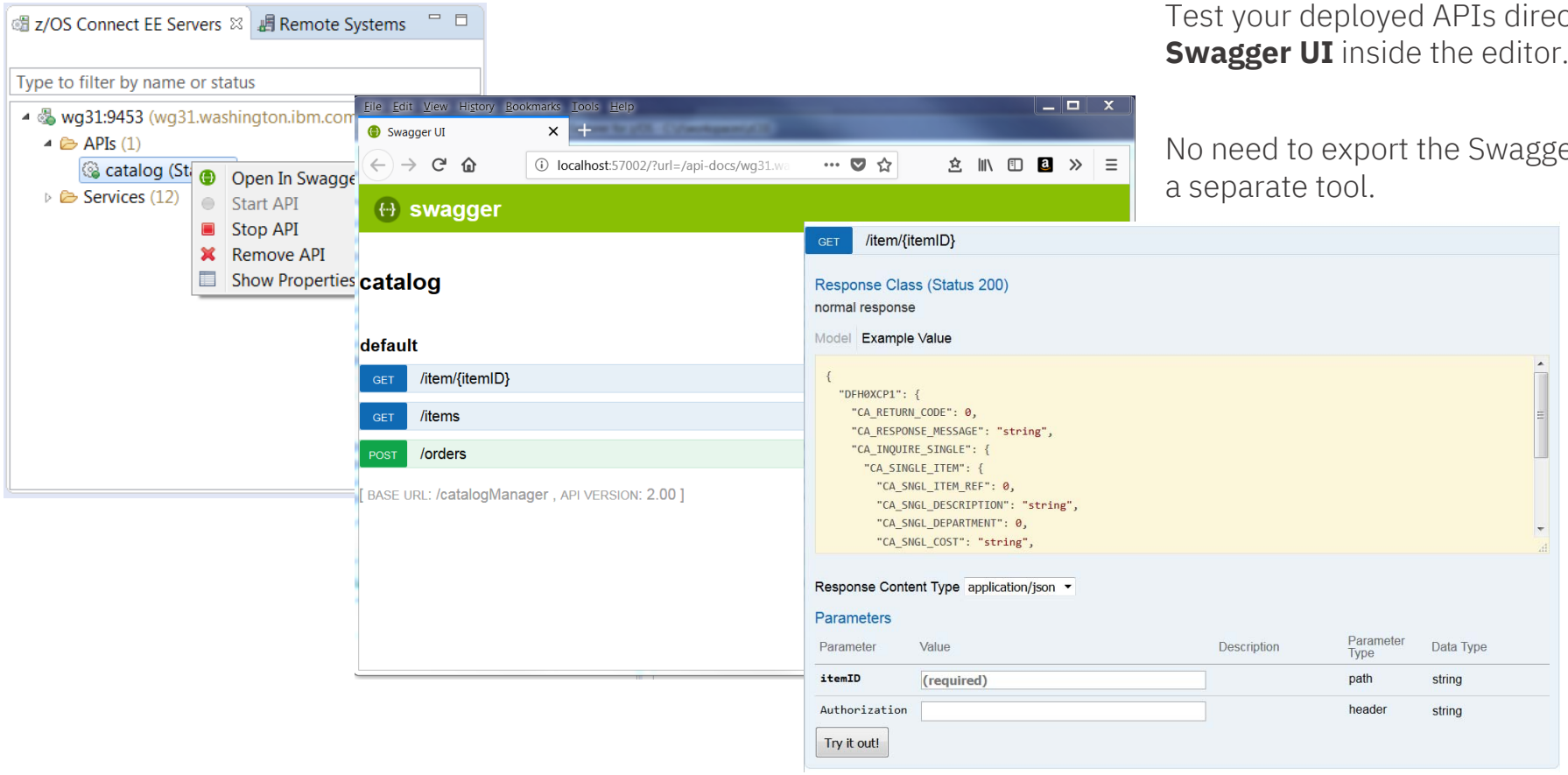
Right-click deploy to server enables developers to quickly deploy, test, and iterate on their APIs.

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

z/OS Connect EE



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

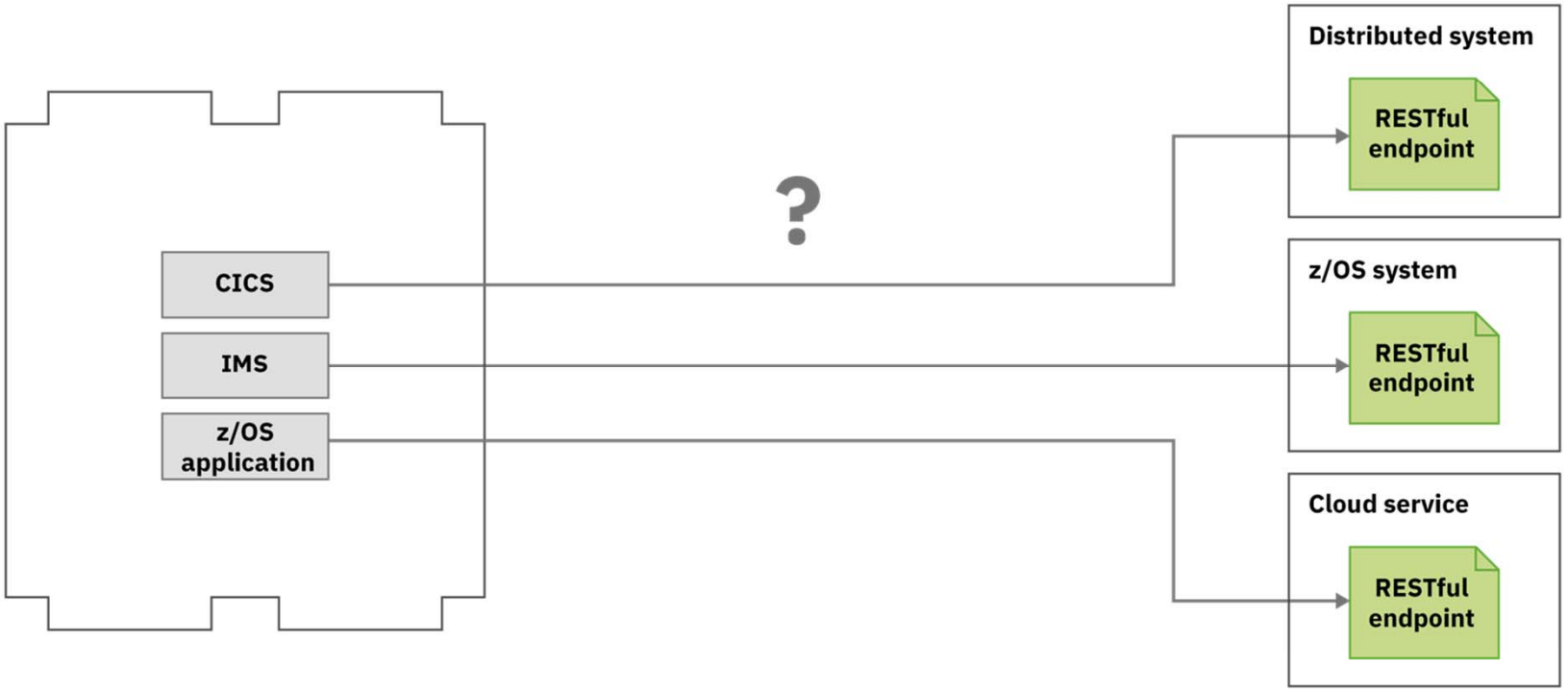
No need to export the Swagger doc to a separate tool.

/api_toolkit/apiRequesters

Quick and easy **API mapping.**

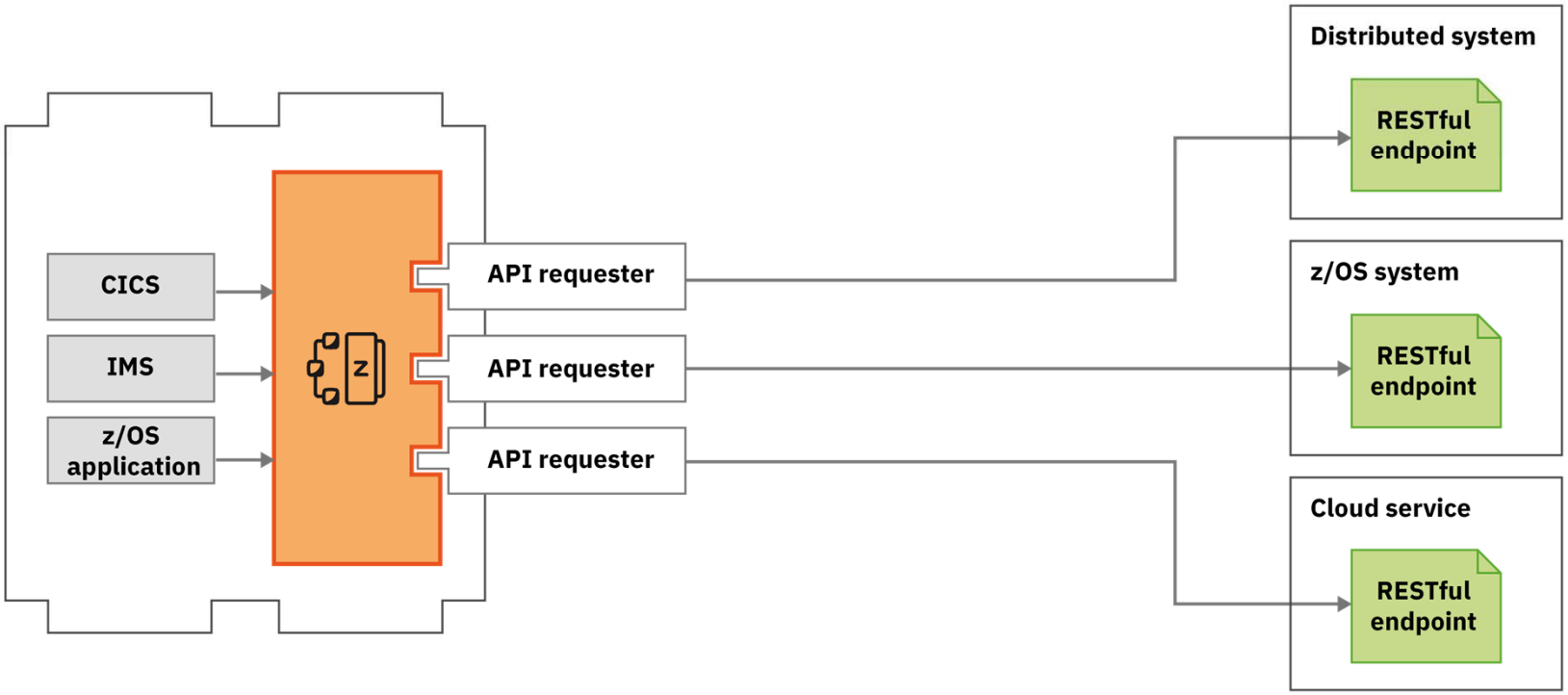
What about calling external APIs from my z/OS assets?

z/OS Connect EE



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

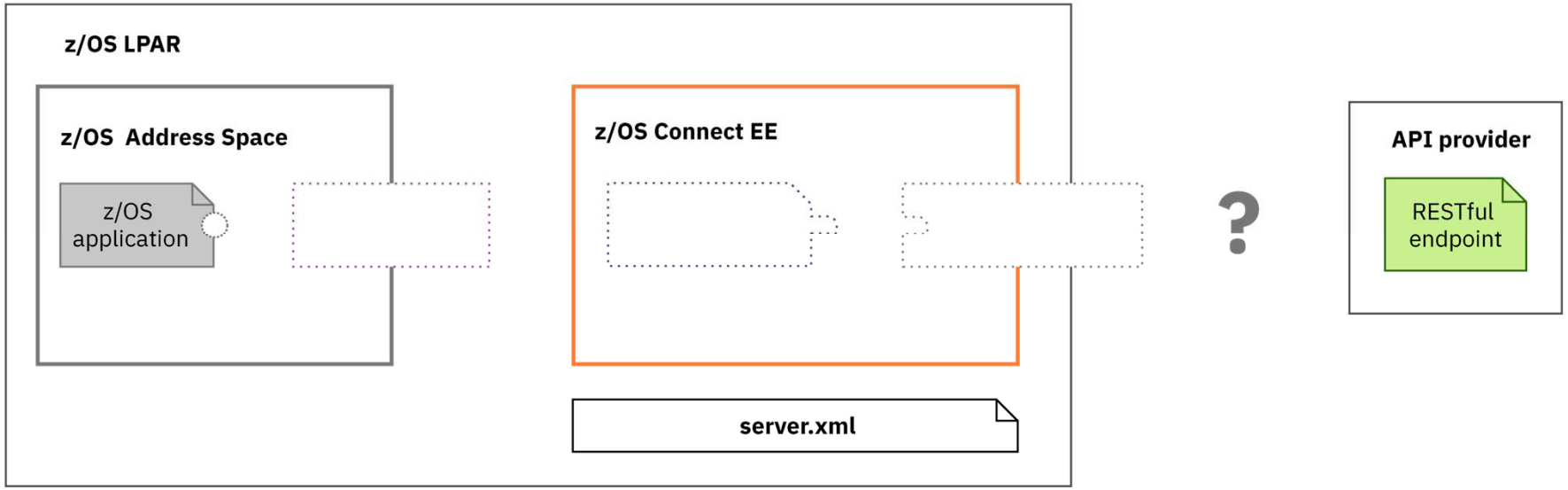
z/OS Connect EE



Steps to calling an external API

z/OS Connect EE

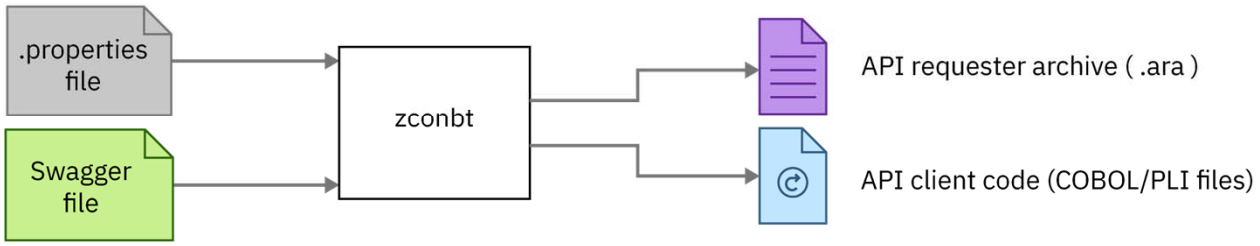
Starting point



Steps to calling an external API

z/OS Connect EE

Step 1. Generate API requester archive from Swagger



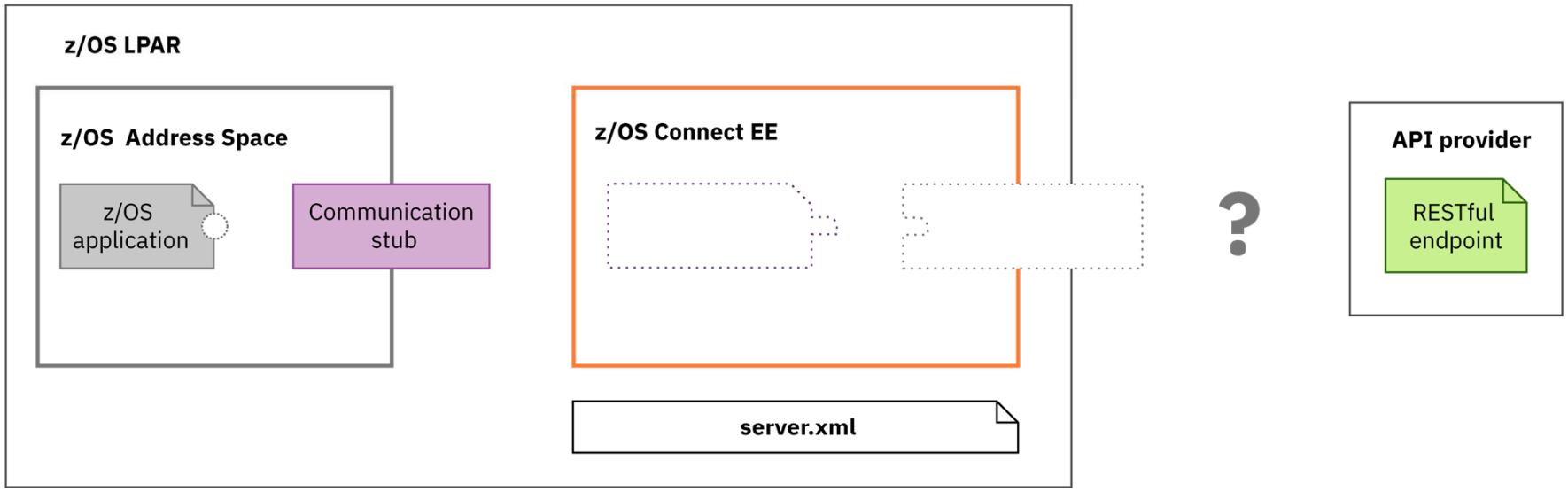
Generate your `.ara` file, and API client code.

ibm.biz/zosconnect-generate-ara

Steps to calling an external API

z/OS Connect EE

Step 2. Configure communication stub



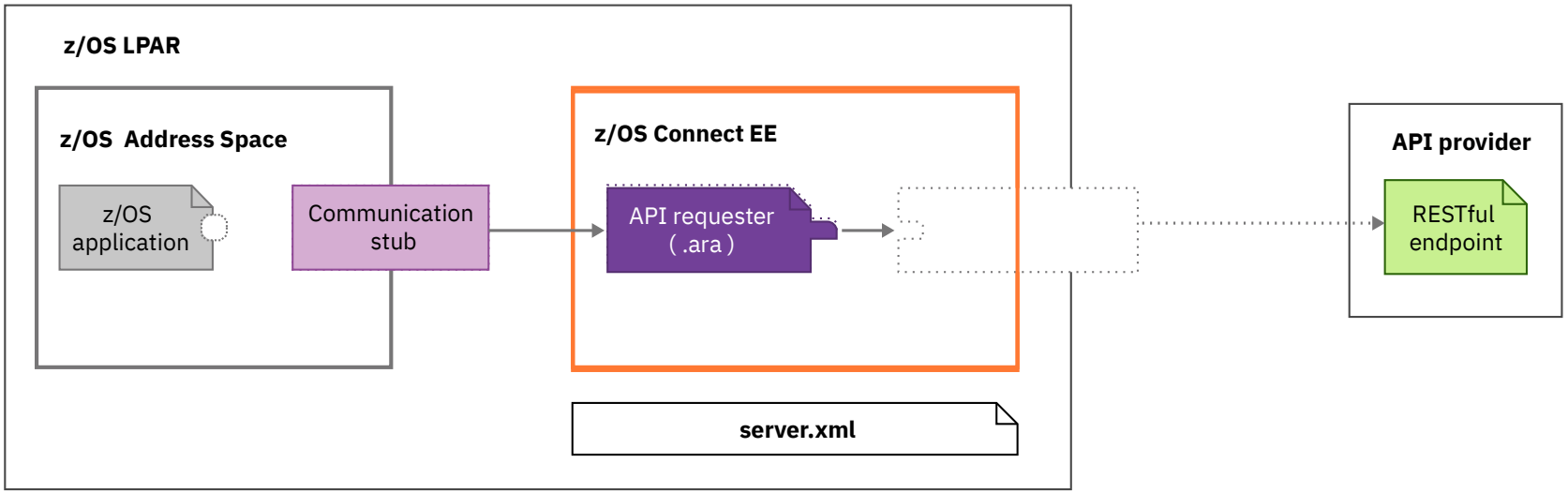
Configure a communication stub. You only need to do this once per system.

ibm.biz/zosconnect-configure-comms-stub

Steps to calling an external API

z/OS Connect EE

Step 3. Deploy API requester (.ara) archive

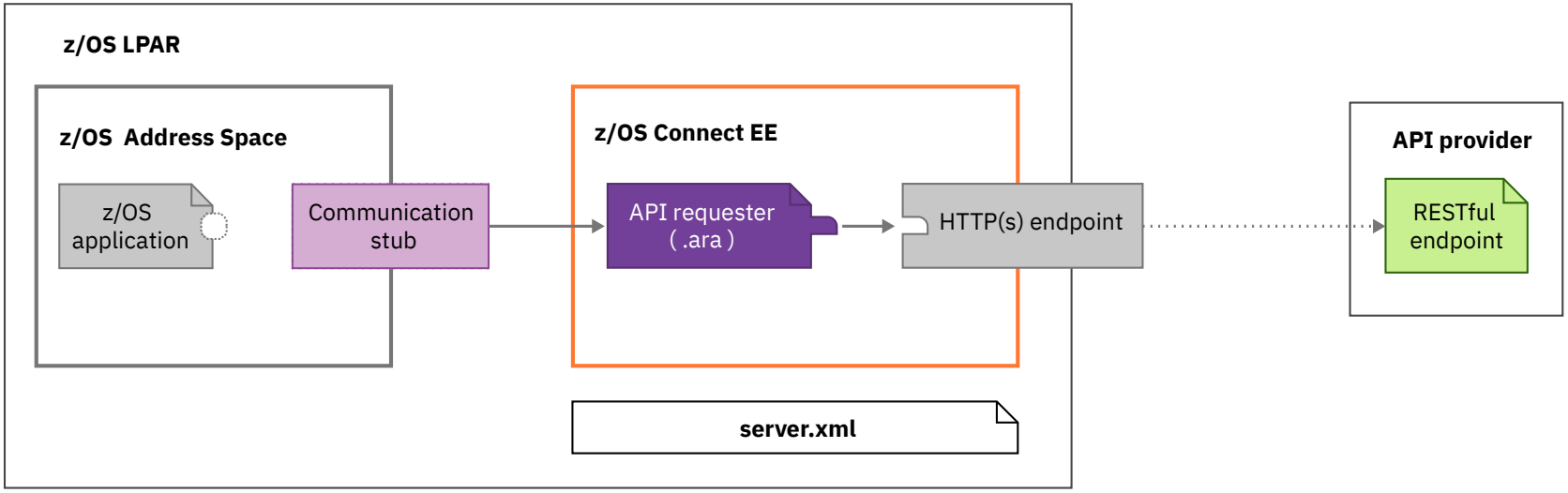


Deploy your API requester archive to the *apiRequester* directory.

Steps to calling an external API

z/OS Connect EE

Step 4. Configure HTTP(S) endpoint



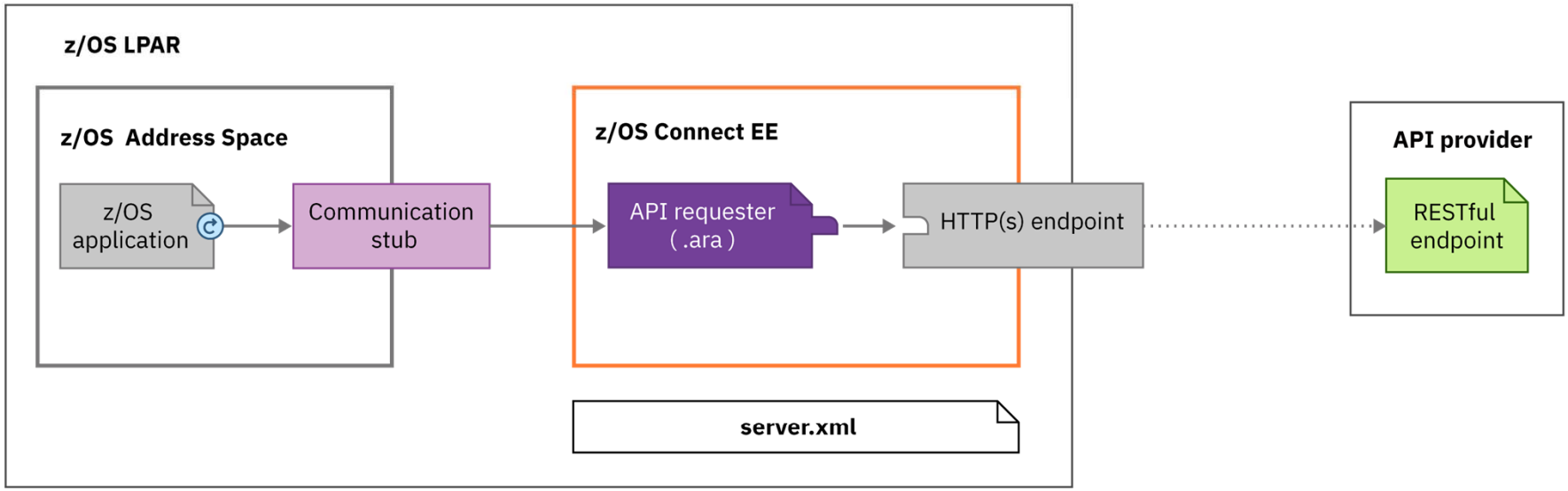
Configure the connection between z/OS Connect EE and the external API.

ibm.biz/zosconnect-configure-endpoint-connection

Steps to calling an external API

z/OS Connect EE

Step 5. Update z/OS application

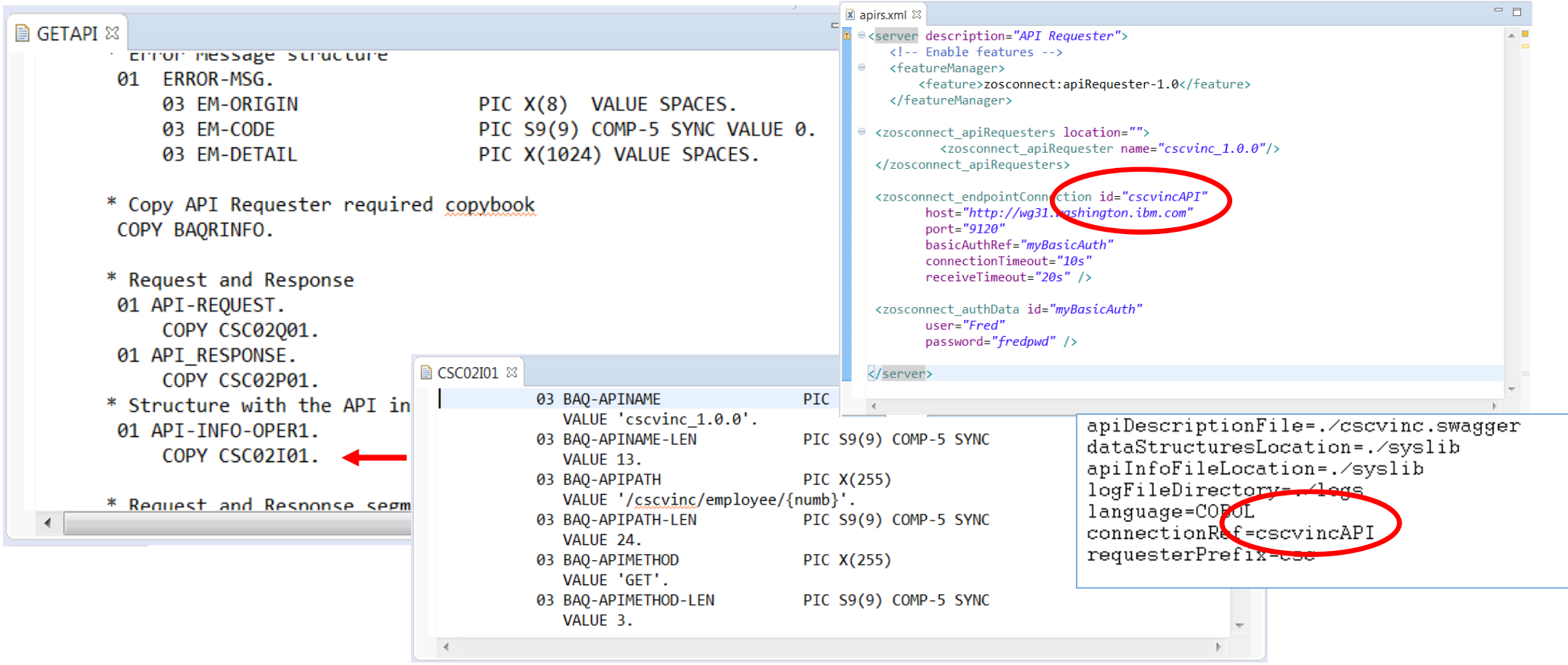


Finally, add the generated API client code to your existing application and use it to make the external API call.

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



Steps to calling an external API

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

GETAPI

```
      *---------------------------------------------------------------*
      * 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
```

Steps to calling an external API

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

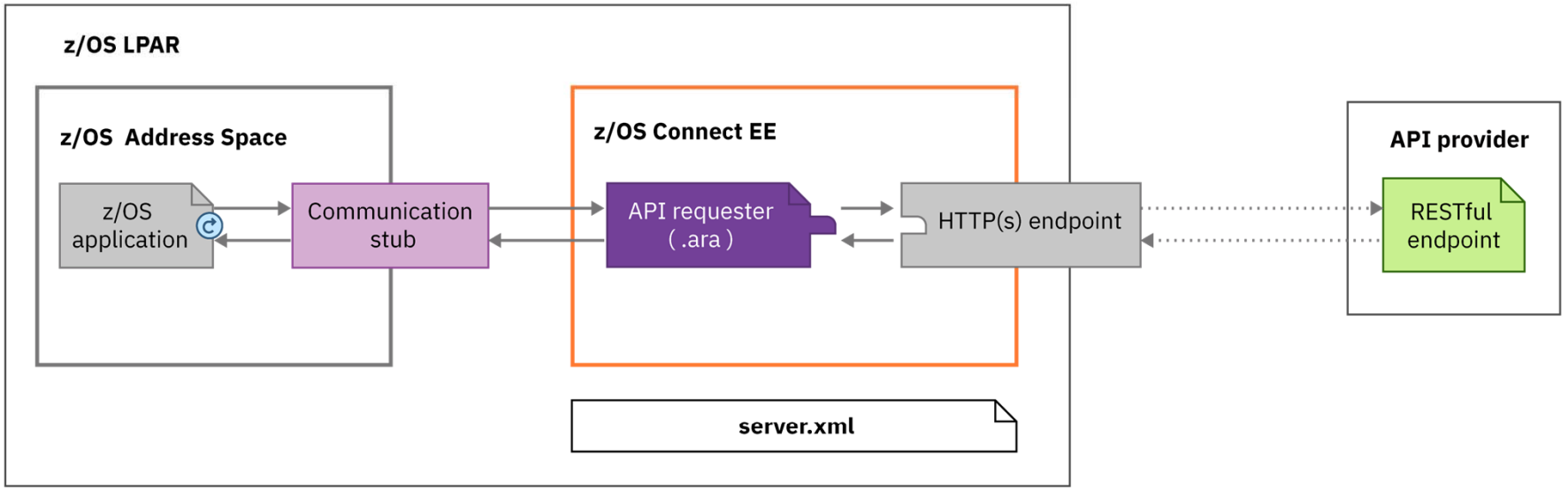
GETAPI

```
                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.
```

Steps to calling an external API

z/OS Connect EE

Done



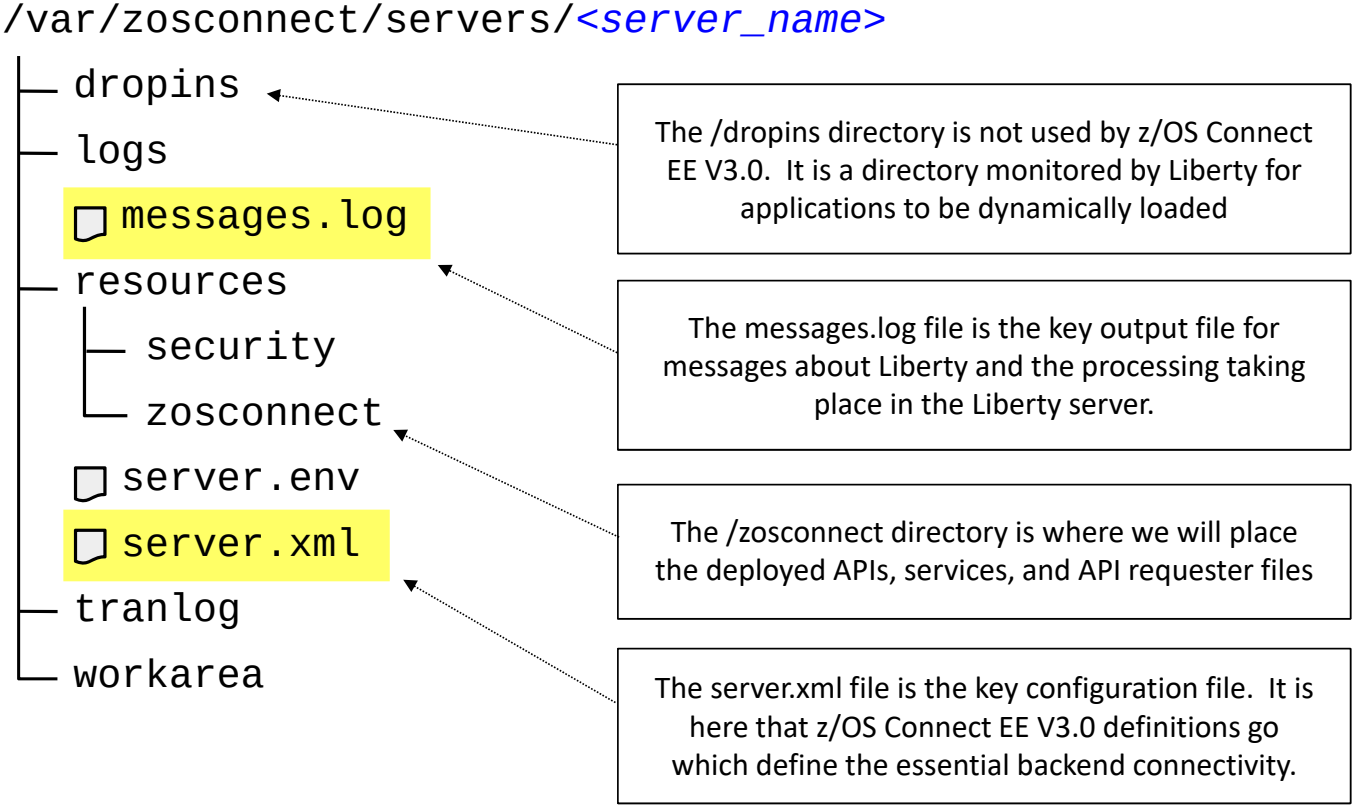
/common_scenarios

Typical connection patterns to different subsystems.

Tour of Server Configuration Directories and Files

z/OS Connect EE

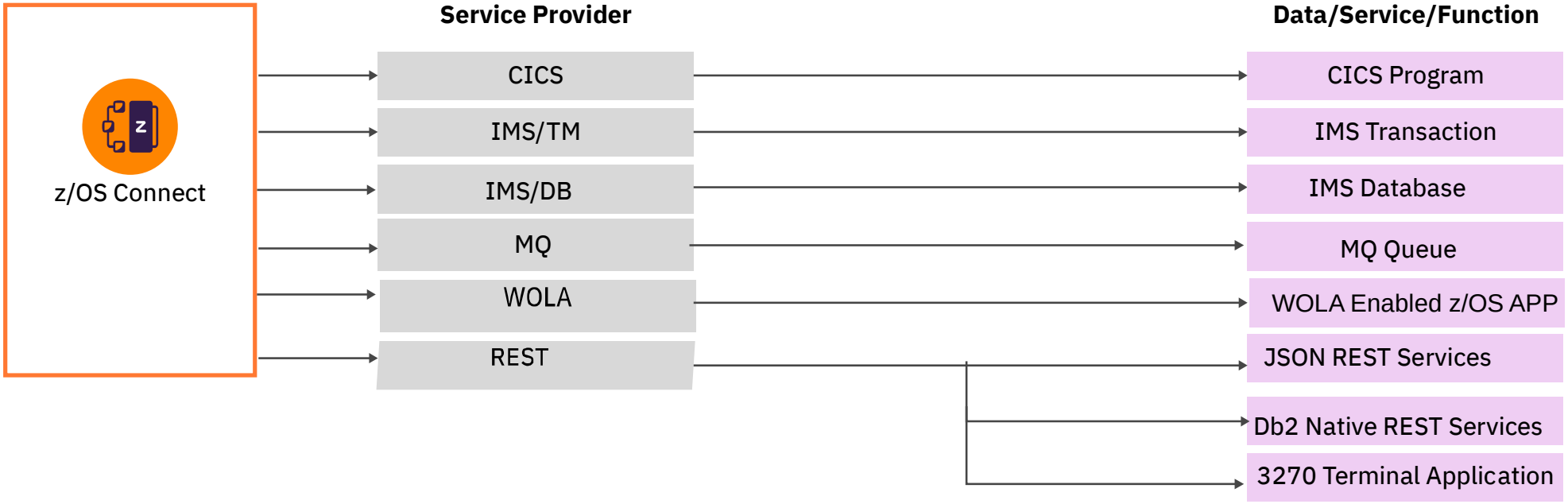
A z/OS Connect EE V3.0 server configuration structure looks like this:



What assets can z/OS Connect EE map to?

z/OS Connect EE

And which service provider could I use?

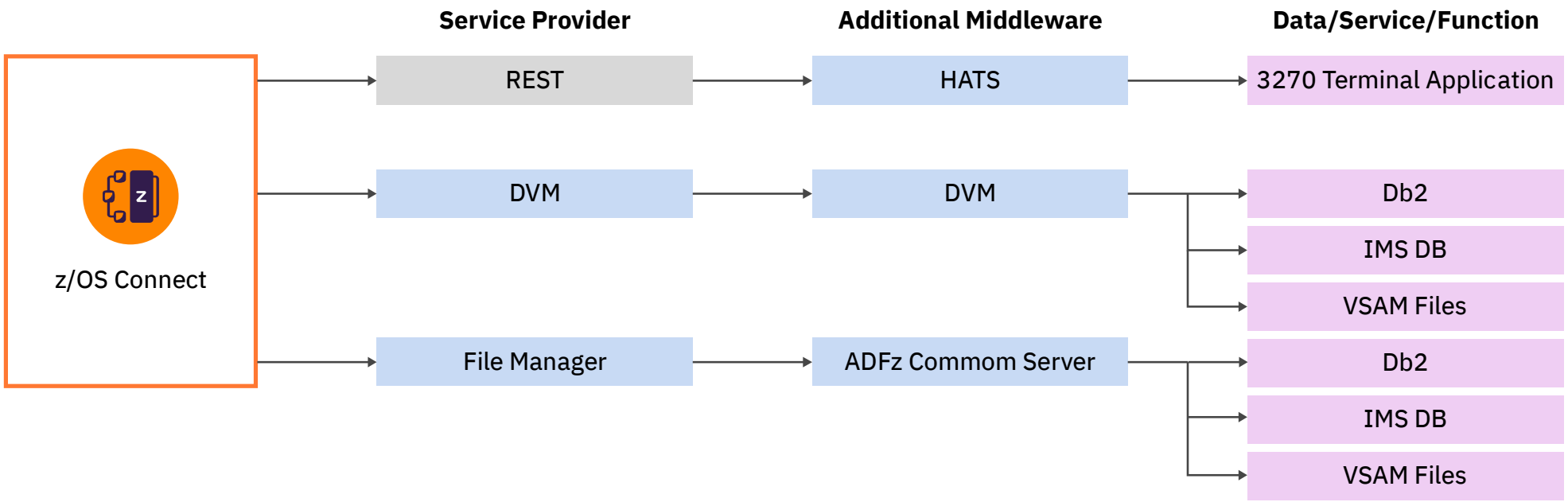


The core **service providers** included with z/OS Connect EE provide API access to a wide range of z/OS assets.

Additional Middleware

Additional value from the ecosystem

z/OS Connect EE

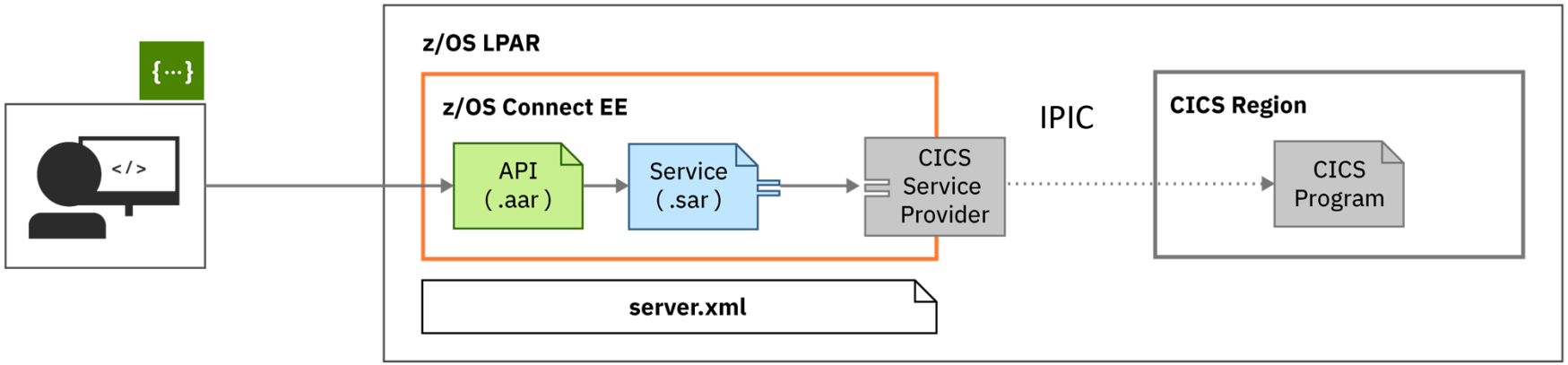


z/OS Connect EE is **pluggable** and **extensible** allowing the use of additional middleware to expand the list of z/OS assets you can expose as APIs

Connections to CICS

z/OS Connect EE

Topology



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

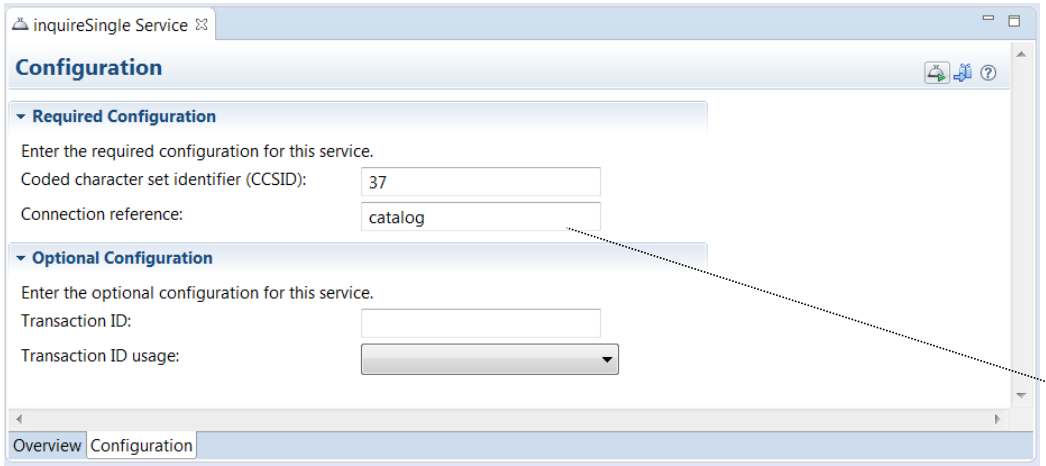
An IPIC connection must be configured in CICS.

ibm.biz/zosconnect-scenarios

CICS IPIC (server.xml)

z/OS Connect EE

The server.xml file is the key configuration file:



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

Define IPIC connection to CICS

catalog.xml

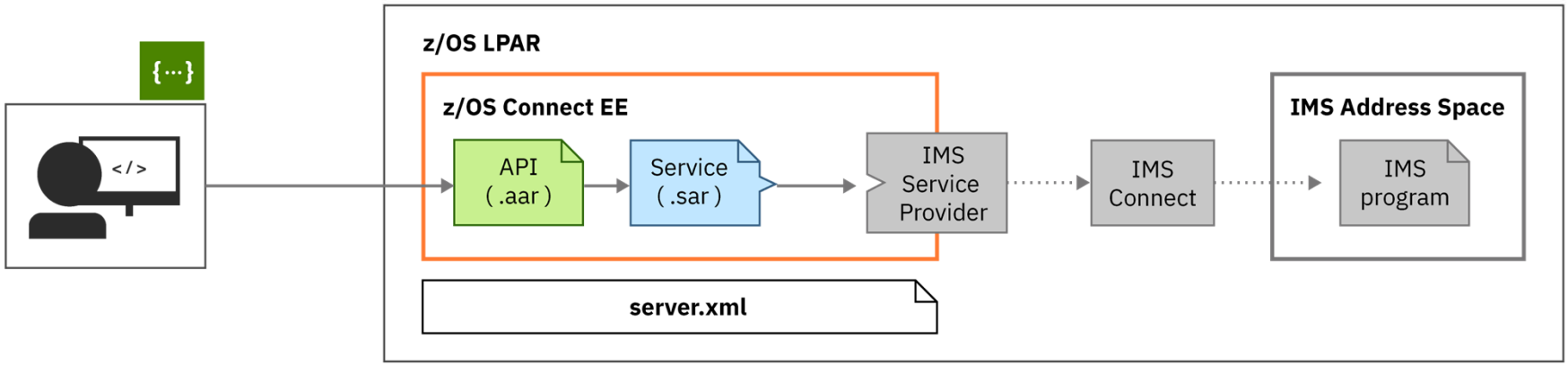
Design | Source

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

Connections to IMS TM

z/OS Connect EE

Topology

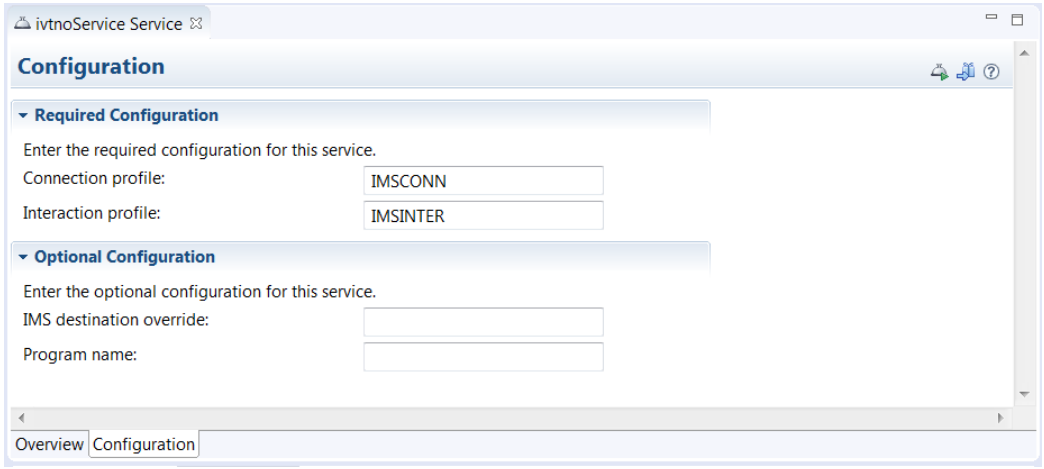


Configure the connection to IMS through `ims-connections.xml` and `ims-interactions.xml` in the IMS service registry.

ibm.biz/zosconnect-scenarios

IMS Connections and Interactions

z/OS Connect EE



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,TMEMBER=OTMAMEM)
IMSPLEX=(MEMBER=IMS14HWS,TMEMBER=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>
```

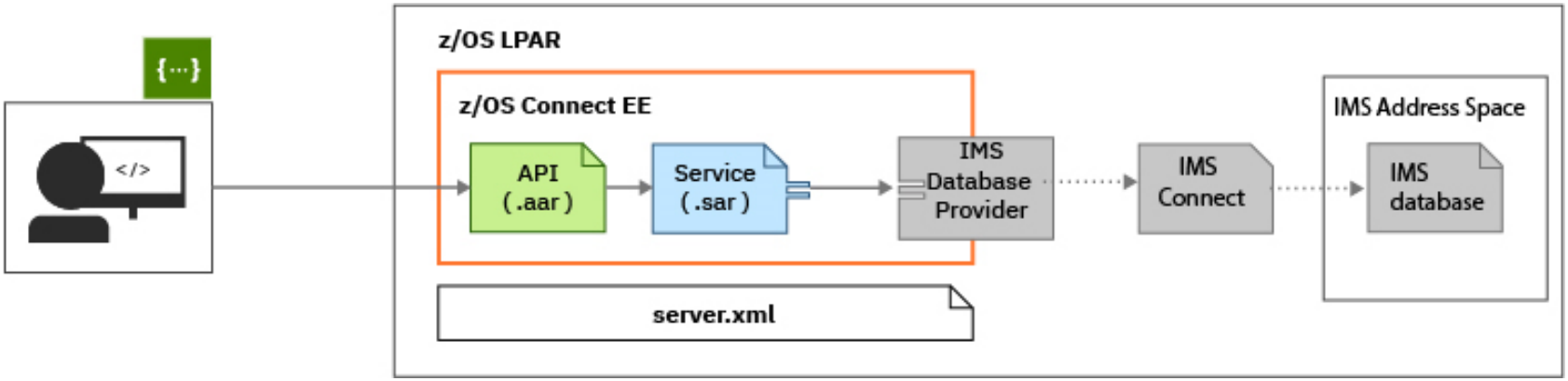
Interaction

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

Connections to IMS DB

z/OS Connect EE

Topology



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

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

ibm.biz/zosconnect-scenarios

IMS Connection Factory

z/OS Connect EE

ConnectionFactory

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

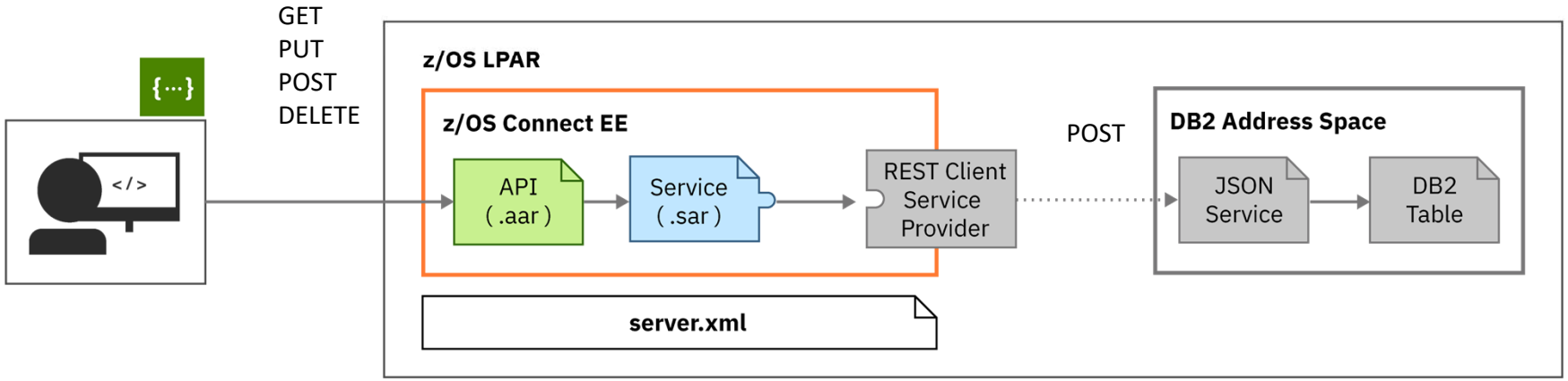
IMS Connect HWSCFG

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

Connections to Db2

z/OS Connect EE

Topology



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

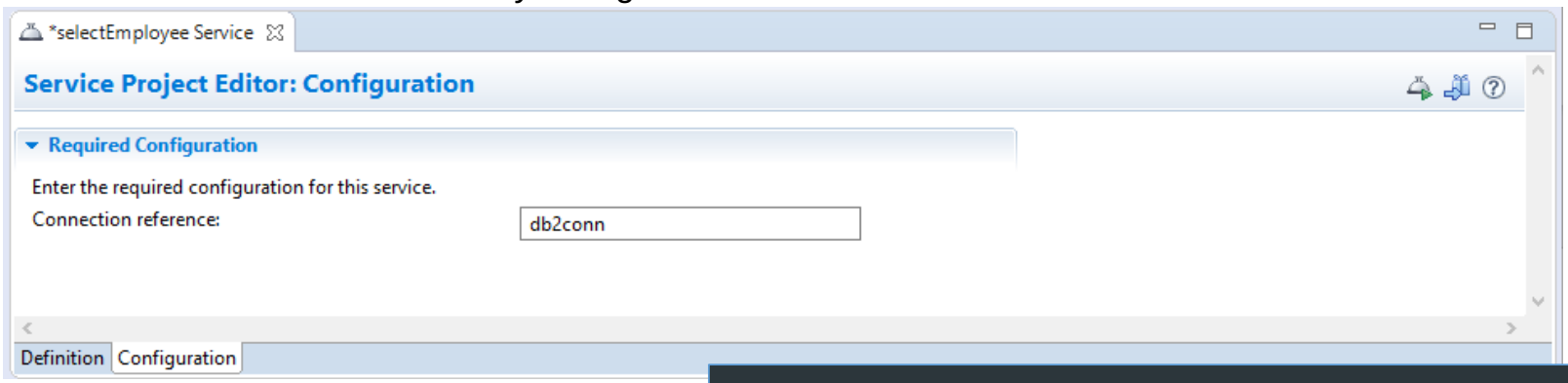
A Db2 REST Service must be configured in DB2.

ibm.biz/zosconnect-db2-rest-services

The server.xml File (Db2)

z/OS Connect EE

The server.xml file is the key configuration file:




```
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

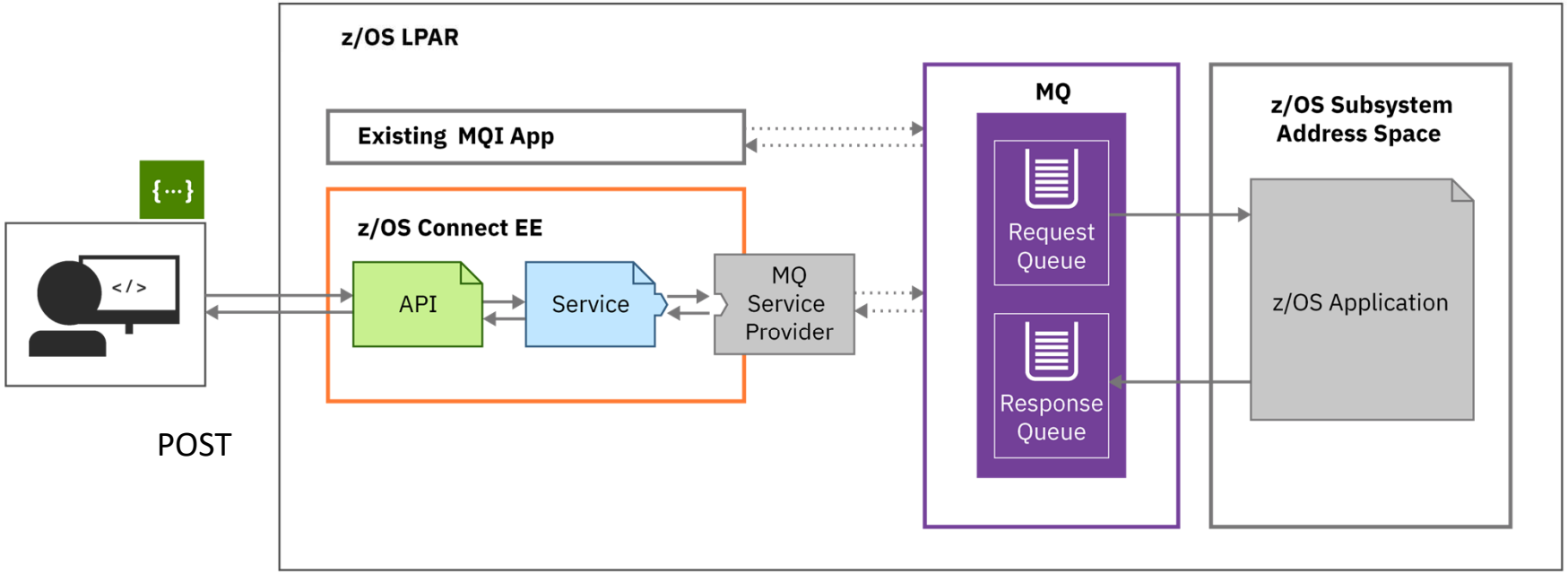
Design | Source

```
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

z/OS Connect EE

Topology (Two-way service example)



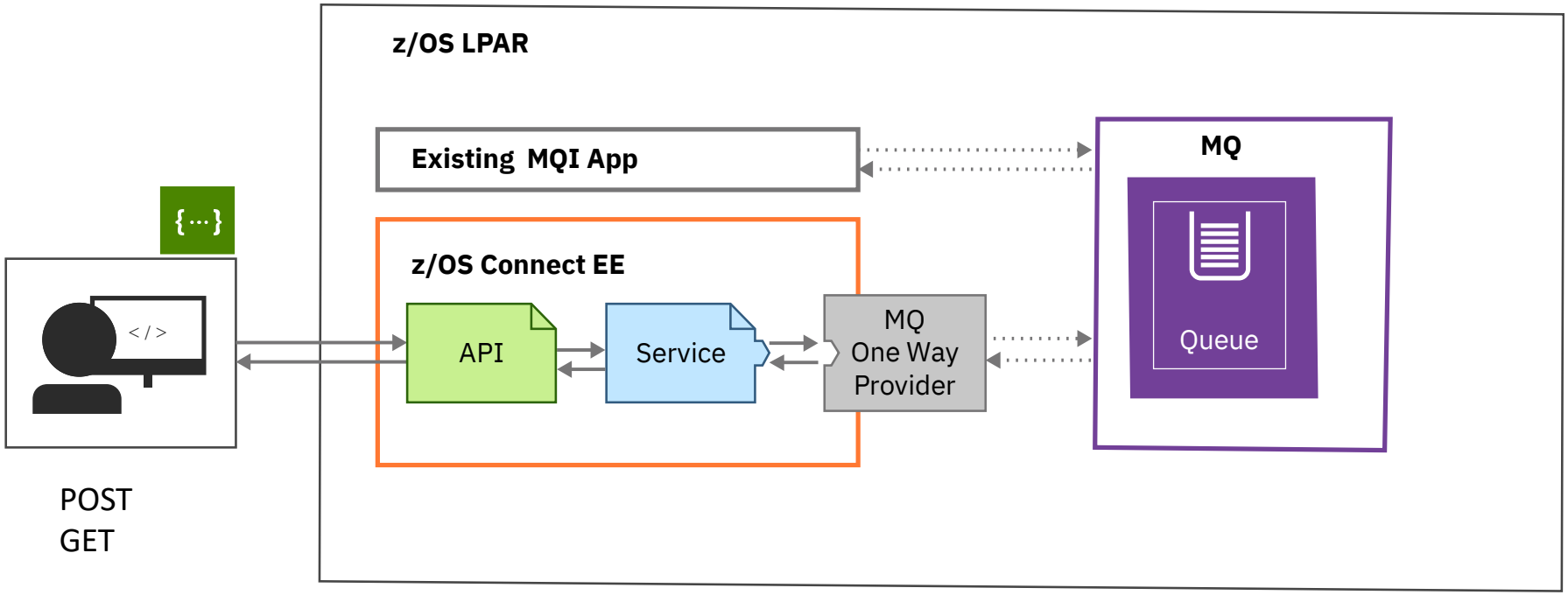
You can also configure one-way services.

ibm.biz/zosconnect-mq-service-provider

Connections to MQ

z/OS Connect EE

Topology (One-way service example)



ibm.biz/zosconnect-mq-service-provider

The server.xml File (MQ)

z/OS Connect EE

*twoway Service

Service Project Editor: Configuration

Required Configuration

Enter the required configuration for this service.

Connection factory JNDI name:	jms/qmgrCf
Request destination JNDI name:	jms/requestQueue
Reply destination JNDI name:	jms/replyQueue
Wait interval:	3000
MQMD format:	MQSTR
Coded character set identifier (CCSID):	37
Is message persistent:	
Reply selection:	msgIDToCorrelID
Expiry:	-1

Definition | Configuration

mq.xml — Read only

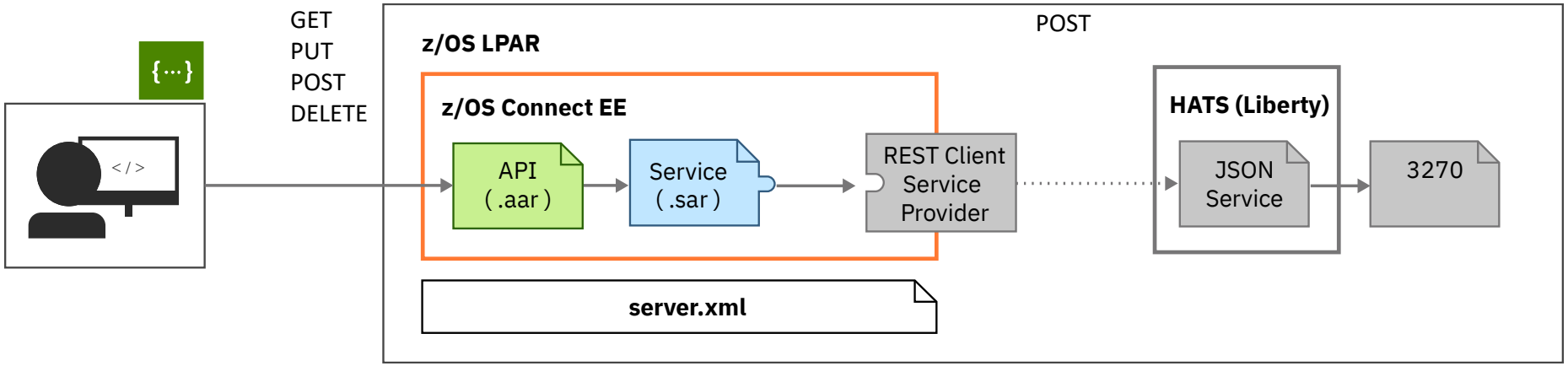
Design | Source

```
<server description="MQ Service Provider">

 <featureManager>
       <feature>zosconnect:mqService-1.0</feature>
 </featureManager>

 <variable name="wmqJmsClient.rar.location"
          value="/usr/lpp/mqm/V9R1M1/java/lib/jca/wmq.jmsra.rar"/>
 <wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>

  <connectionManager id="ConMgr1" maxPoolSize="5"/>

  <jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
        connectionManagerRef="ConMgr1">
        <properties.wmqJMS transportType="BINDINGS"
                  queueManager="QMZ1" />
  </jmsConnectionFactory>

  <jmsQueue id="default" jndiName="jms/default">
      <properties.wmqJms
          baseQueueName="ZCONN2.DEFAULT.MQZCEE.QUEUE"
          CCSID="37"/>
  </jmsQueue>

  <jmsQueue id="request" jndiName="jms/request">
      <properties.wmqJms
          baseQueueName="ZCONN2.TRIGGER.REQUEST"
          targetClient="MQ"
          CCSID="37"/>
  </jmsQueue>

  <jmsQueue id="response" jndiName="jms/response">
      <properties.wmqJms
          baseQueueName="ZCONN2.TRIGGER.RESPONSE"
          targetClient="MQ"
          CCSID="37"/>
  </jmsQueue>
```

Connection to HATS

z/OS Connect EE

Topology



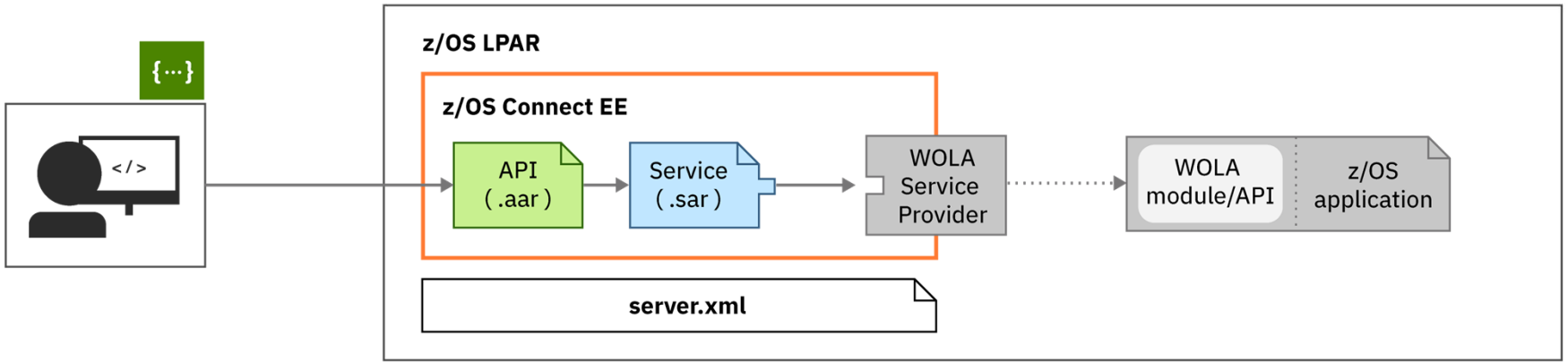
Connection to the HATS REST Service is configured in `server.xml`.

ibm.biz/zosconect-db2-rest-services

Connections to a MVS batch application

z/OS Connect EE

Topology



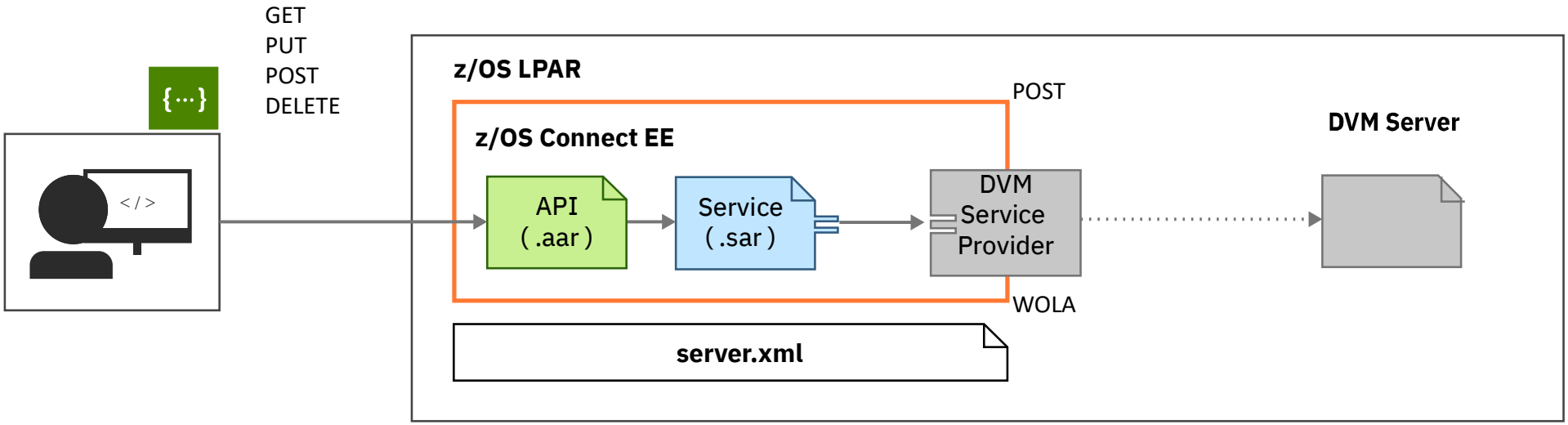
Connection to WOLA is configured in `server.xml`.

The z/OS application must be WOLA-enabled.

Connections to DVM

z/OS Connect EE

Topology



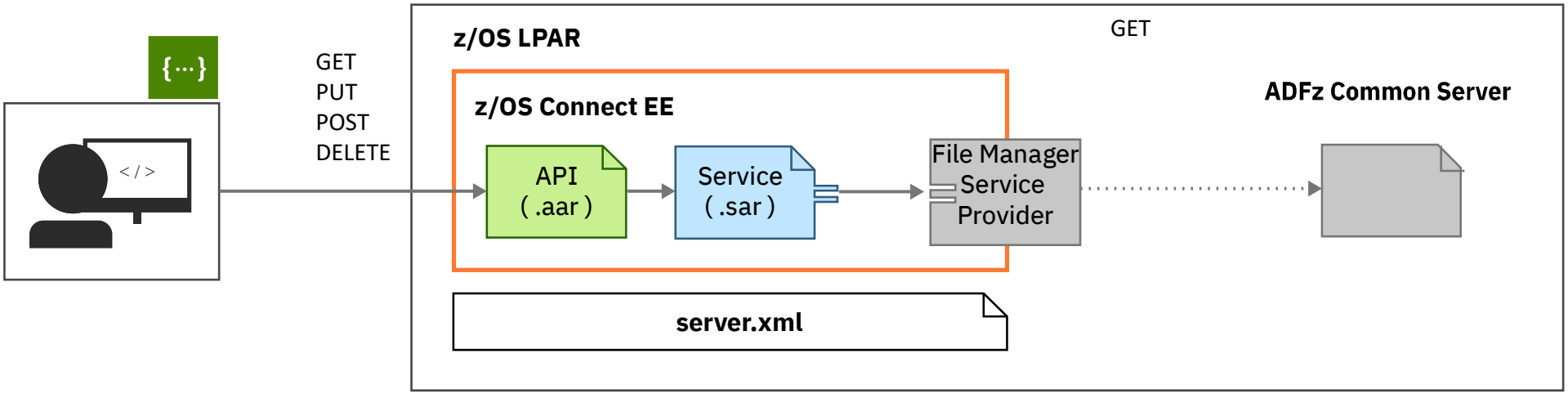
The DVM service provider uses WOLA

ibm.biz/zosconect-db2-rest-services

Connections to File Manager

z/OS Connect EE

Topology



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

A File Manager Template is required .

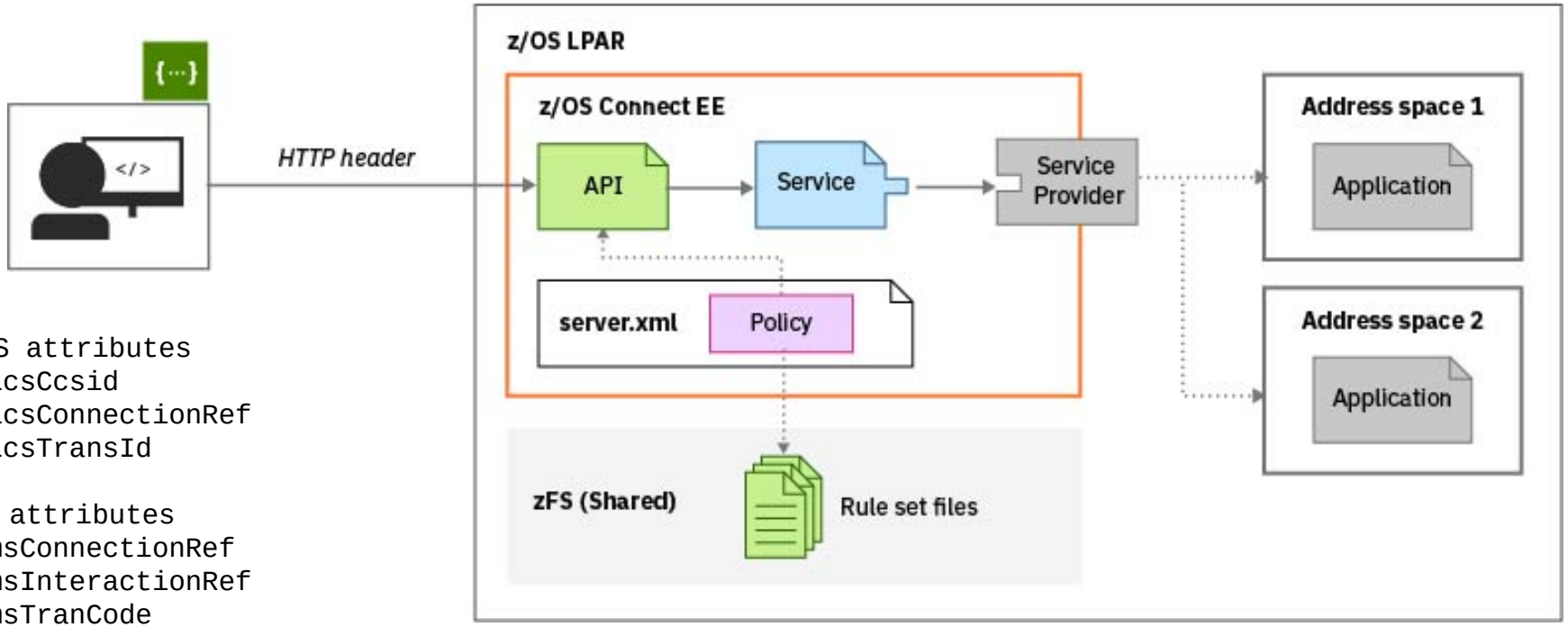
/miscellaneousTopics

performance, high availability, Liberty

API Policies

z/OS Connect EE

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



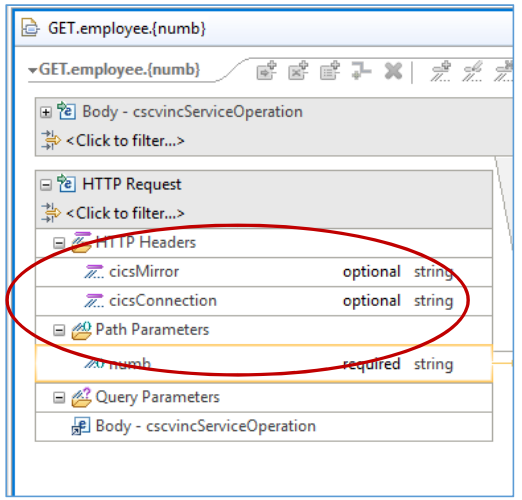
CICS attributes
- cicsCcsid
- cicsConnectionRef
- cicsTransId

IMS attributes
- imsConnectionRef
- imsInteractionRef
- imsTranCode

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"/> [1]
          </conditions>
          <actions>
              <set property="cicsTransId" value="${cicsMirror}"/>
          </actions>
      </rule>
      <rule name="connection-rule">
          <conditions>
              <header name="cicsConnection"
                      value="cscvinc,cics92,cics93"/>
          </conditions>
          <actions>
              <set property="cicsConnectionRef"
value="${cicsConnection}"
          </actions>
      </rule>
  </ruleset>
```



Curl

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

[1]Transaction MIJO needs to be a clone of CSMI (e.g. invoke program DFHMIRS)

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.
```

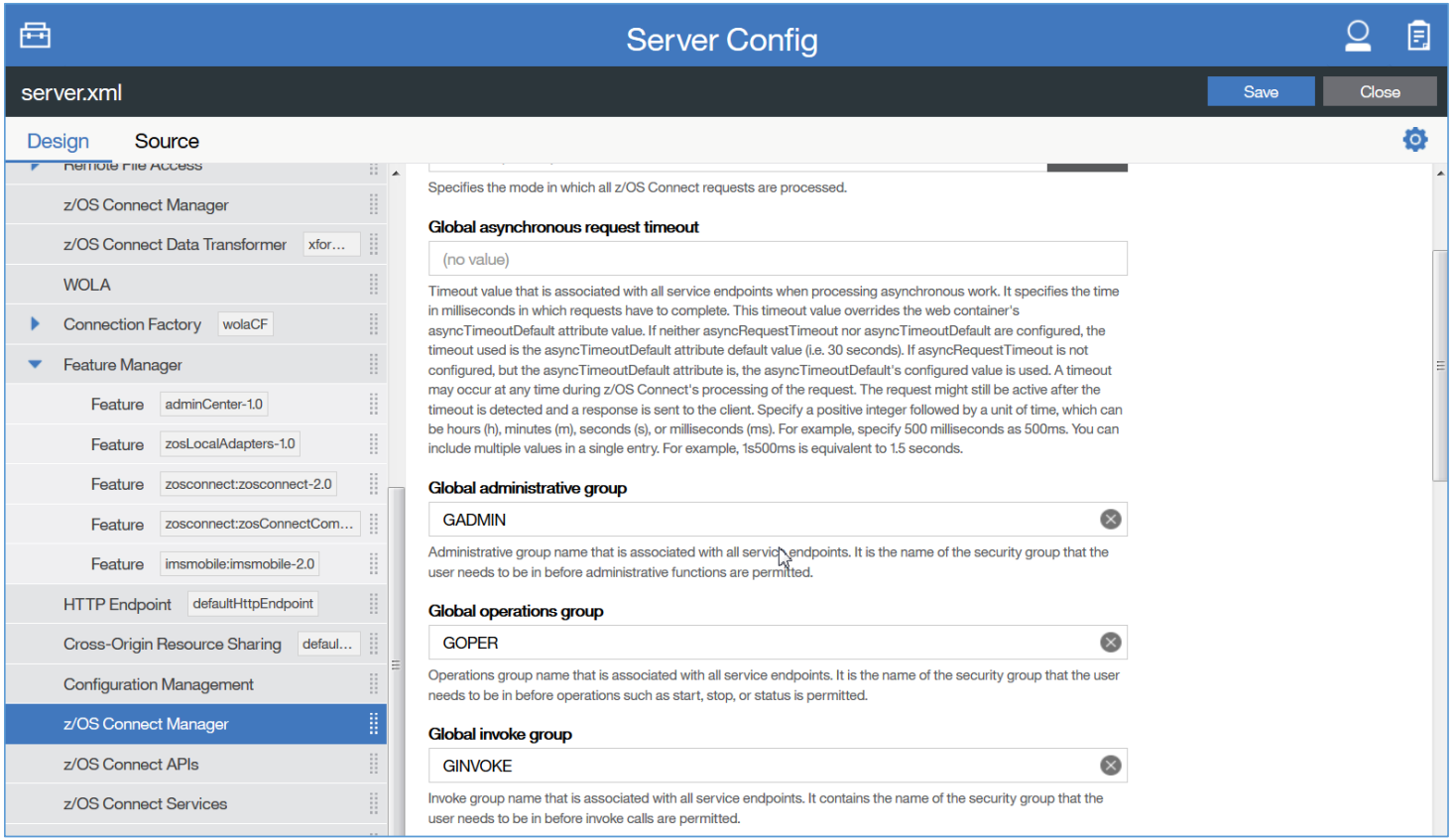
STDERR

```
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.
```

Liberty's "adminCenter" Feature

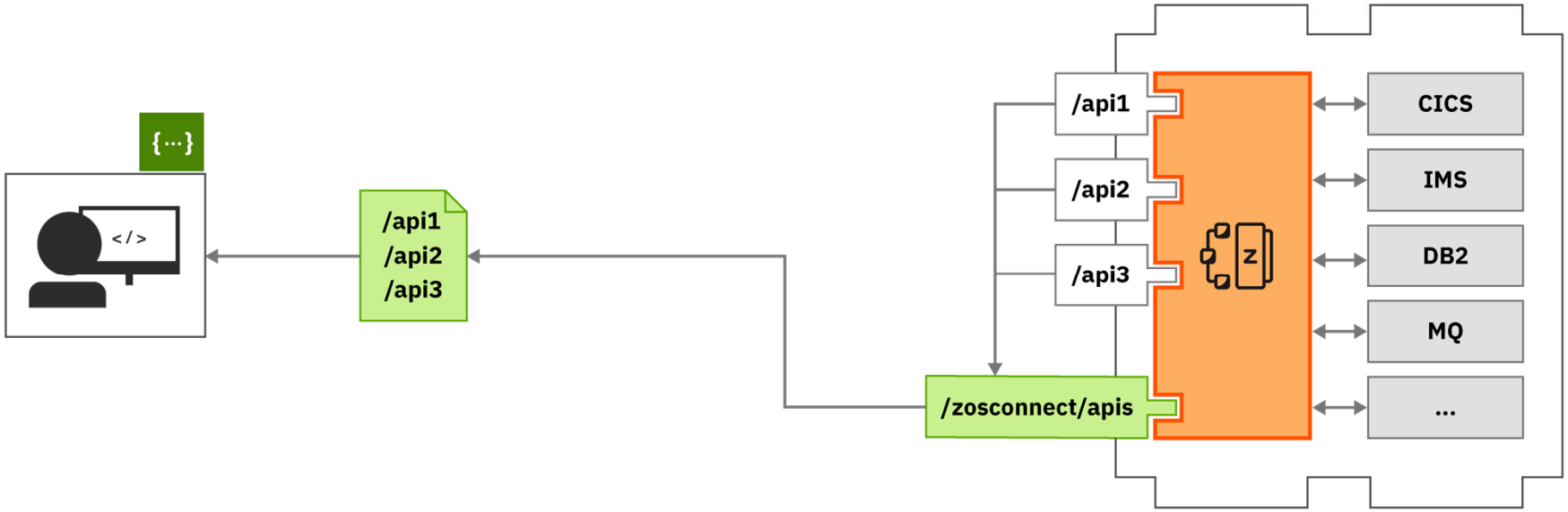
z/OS Connect EE

Web browser interface to the server's configuration files



API Documentation

z/OS Connect EE



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

RESTful Administrative Interface for Services

z/OS Connect EE

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

Most administration tasks are supported by the RESTful administration interface

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

POST /zosConnect/services inquireSingle.sar
PUT /zosConnect/services/{serviceName}?status=started|stopped
PUT /zosConnect/services inquireSingle.sar
GET /zosConnect/services
GET /zosConnect/services/{serviceName}
DELETE /zosConnect/services/{serviceName}

*Useful for deploying DB2 and HATS service archive files

RESTful Administrative Interface for APIs

z/OS Connect EE

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

Most administration tasks are supported by the RESTful administration interface

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

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

RESTful Administrative Interface for API Requesters

z/OS Connect EE

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

Most administration tasks are supported by the RESTful administration interface

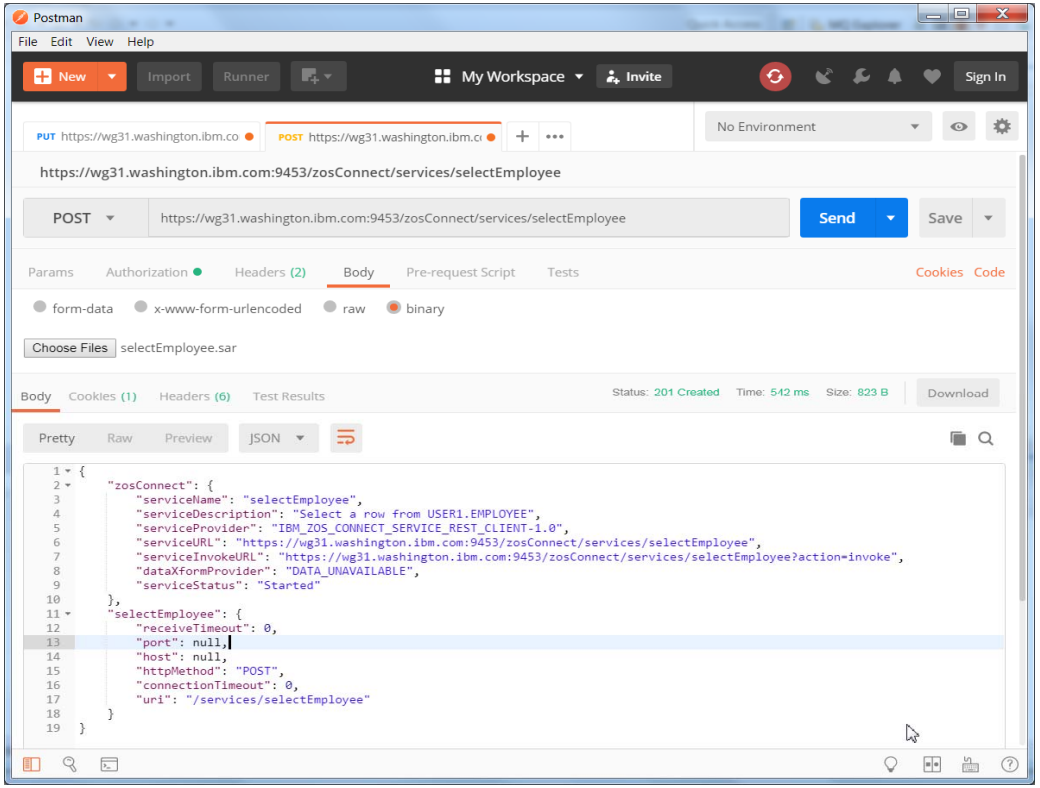
Method	Administrative Task
GET	Get a list of API Requesters
	Get the details of an API Requester
POST	Deploy an API Requester
PUT	Update an API Requester
	Change the status of an API Requester
DELETE	Delete an API Requester

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

Deploying Db2 Service Archive Options

z/OS Connect EE

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



Command:

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

Results:

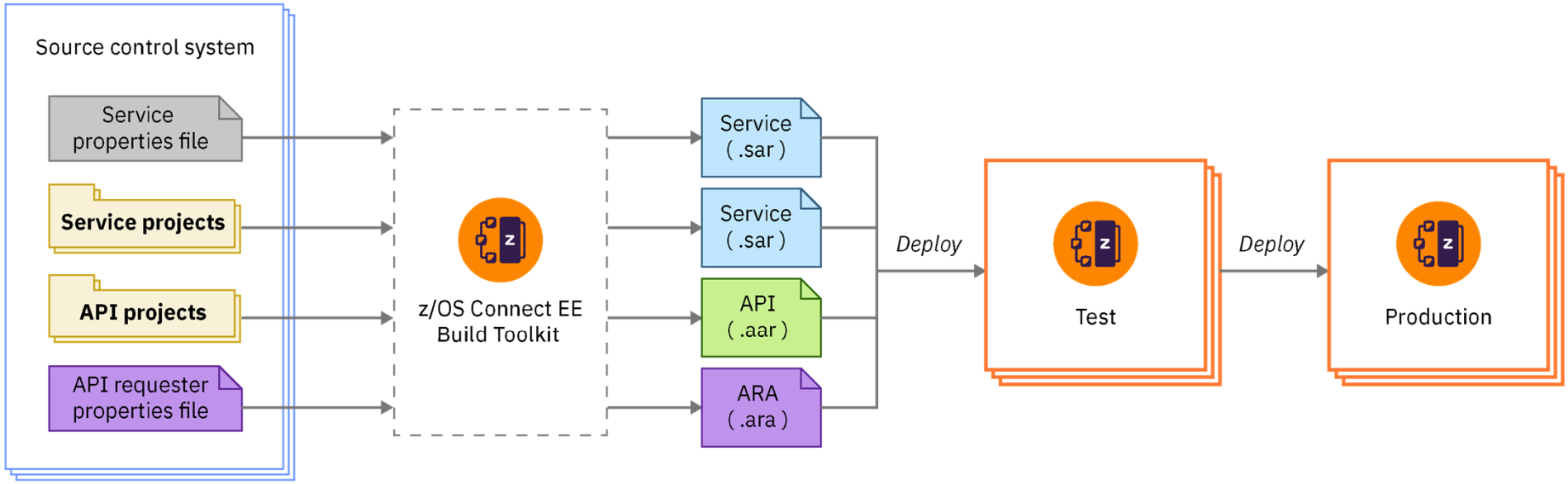

```
{"zosConnect":{"serviceName":"selectEmployee","serviceDescription":"Select a row from USER1.EMPLOYEE","serviceProvider":"IBM_ZOS_CONNECT_SERVICE_REST_CLIENT-1.0","serviceURL":"https://mpxm:9453/zosConnect/services/selectEmployee","serviceInvokeURL":"https://mpxm:9453/zosConnect/services/selectEmployee?action=invoke","dataXformProvider":"DATA_UNAVAILABLE","se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

DevOps using z/OS Connect EE

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

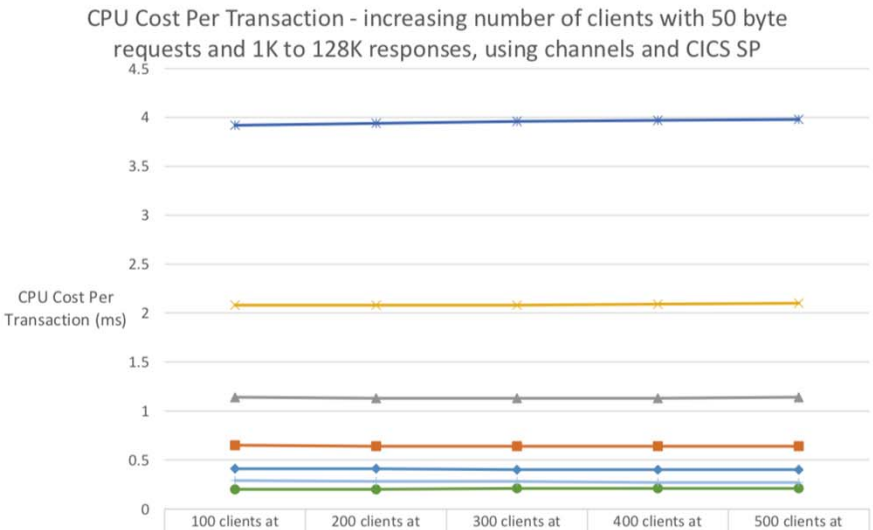


ibm.biz/zosconnect-devops

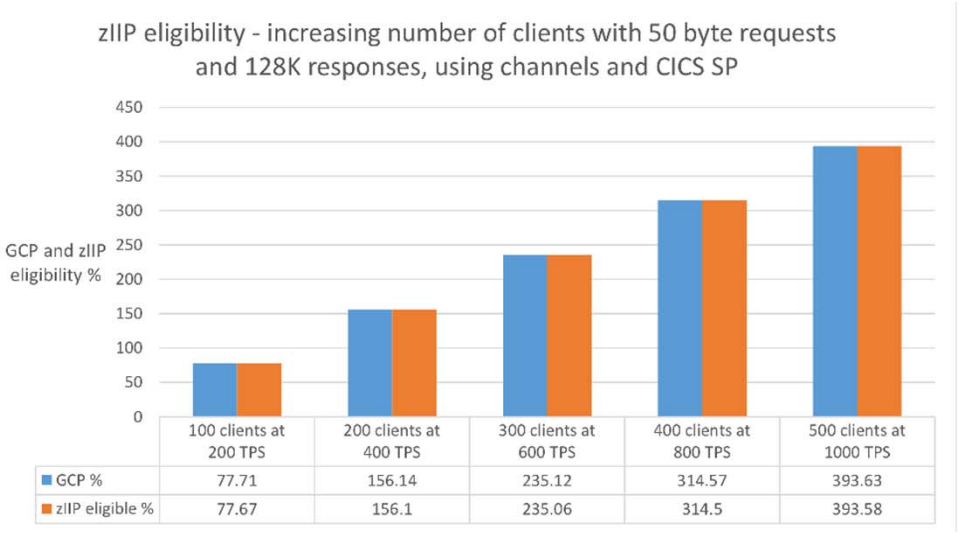
Performance: API Provider

z/OS Connect EE

High Speed, High Throughput, Low Cost



	100 clients at 200 TPS	200 clients at 400 TPS	300 clients at 600 TPS	400 clients at 800 TPS	500 clients at 1000 TPS
1K response	0.2	0.2	0.21	0.21	0.21
4K response	0.29	0.28	0.28	0.27	0.27
8K response	0.41	0.41	0.4	0.4	0.4
16K response	0.65	0.64	0.64	0.64	0.64
32K response	1.14	1.13	1.13	1.13	1.14
64K response	2.08	2.08	2.08	2.09	2.1
128K response	3.92	3.94	3.96	3.97	3.98



	100 clients at 200 TPS	200 clients at 400 TPS	300 clients at 600 TPS	400 clients at 800 TPS	500 clients at 1000 TPS
GCP %	77.71	156.14	235.12	314.57	393.63
zIIP eligible %	77.67	156.1	235.06	314.5	393.58

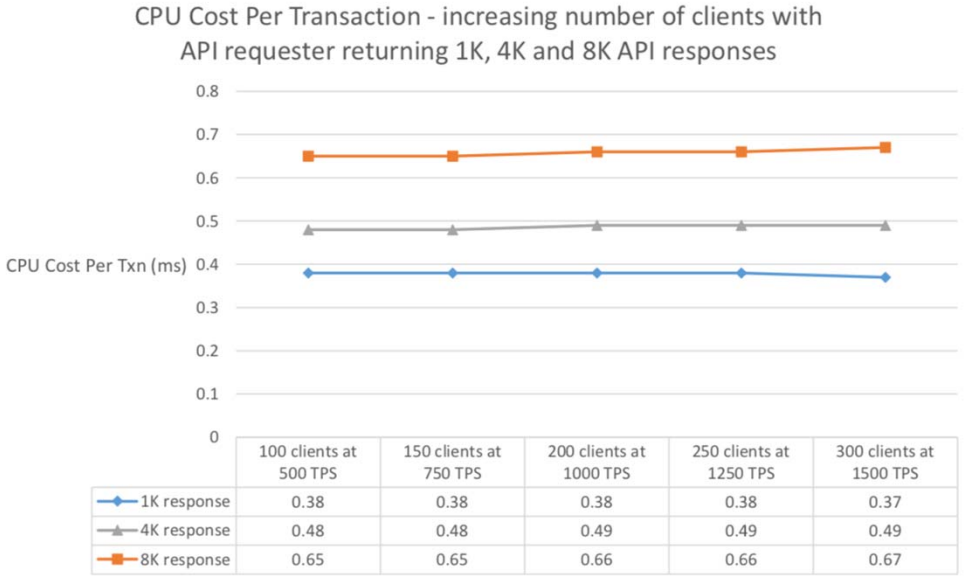
z/OS Connect EE is a Java-based product:
Over **99%** of its MIPs are **eligible for ZIIP offload**.

ibm.biz/zosconnect-performance-report

Performance: API Requester

High Speed, High Throughput, Low Cost

z/OS Connect EE



	100 clients at 500 TPS	150 clients at 750 TPS	200 clients at 1000 TPS	250 clients at 1250 TPS	300 clients at 1500 TPS
1K response	0.38	0.38	0.38	0.38	0.37
4K response	0.48	0.48	0.49	0.49	0.49
8K response	0.65	0.65	0.66	0.66	0.67

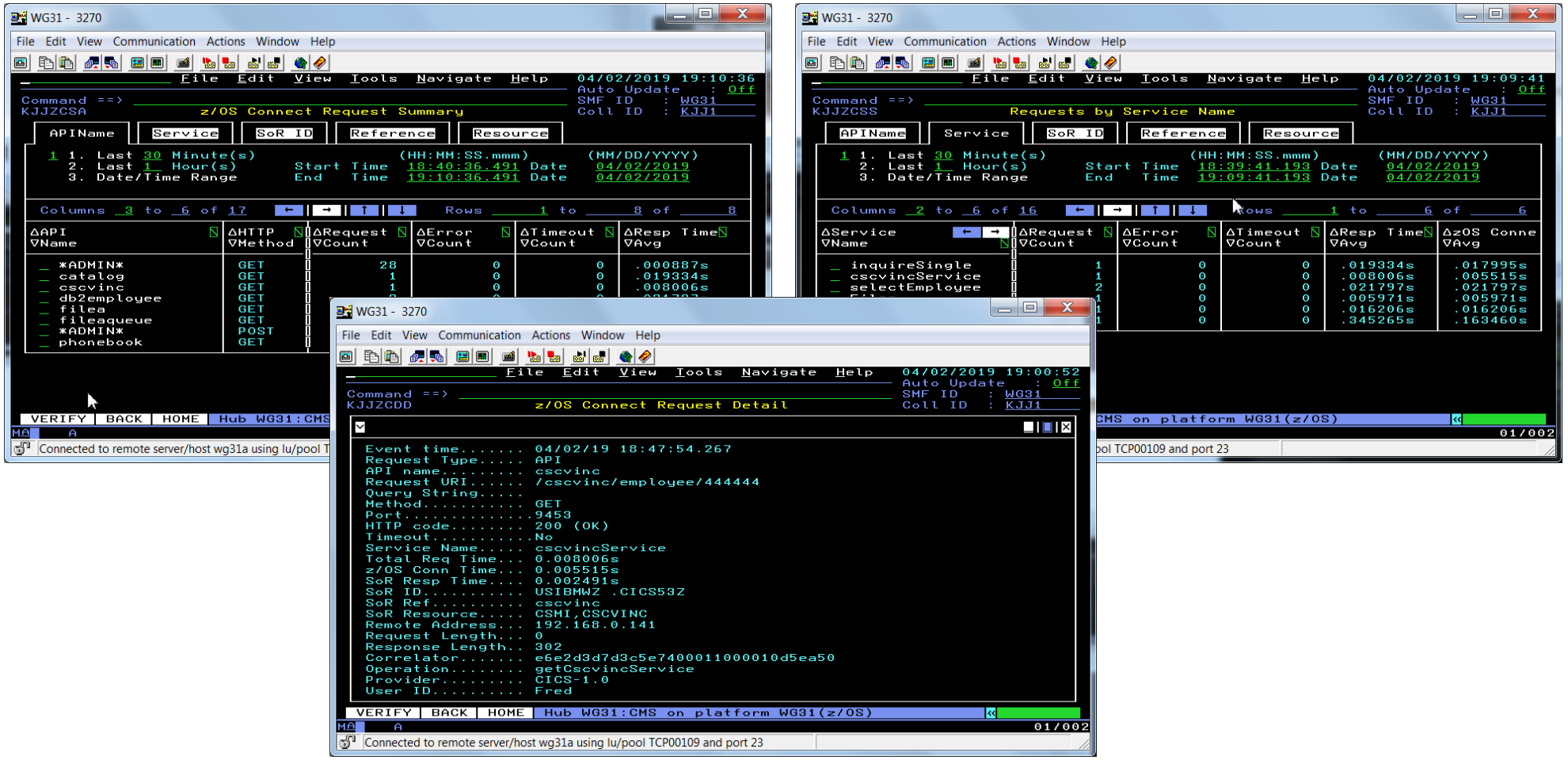
	100 clients at 500 TPS	150 clients at 750 TPS	200 clients at 1000 TPS	250 clients at 1250 TPS	300 clients at 1500 TPS
GCP % - 8K response	32.25	48.4	65.47	81.96	99.09
zIIP eligible % - 8K response	32.21	48.35	65.42	81.9	99.02

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

ibm.biz/zosconnect-performance-report

IBM z Omegamon for JVM

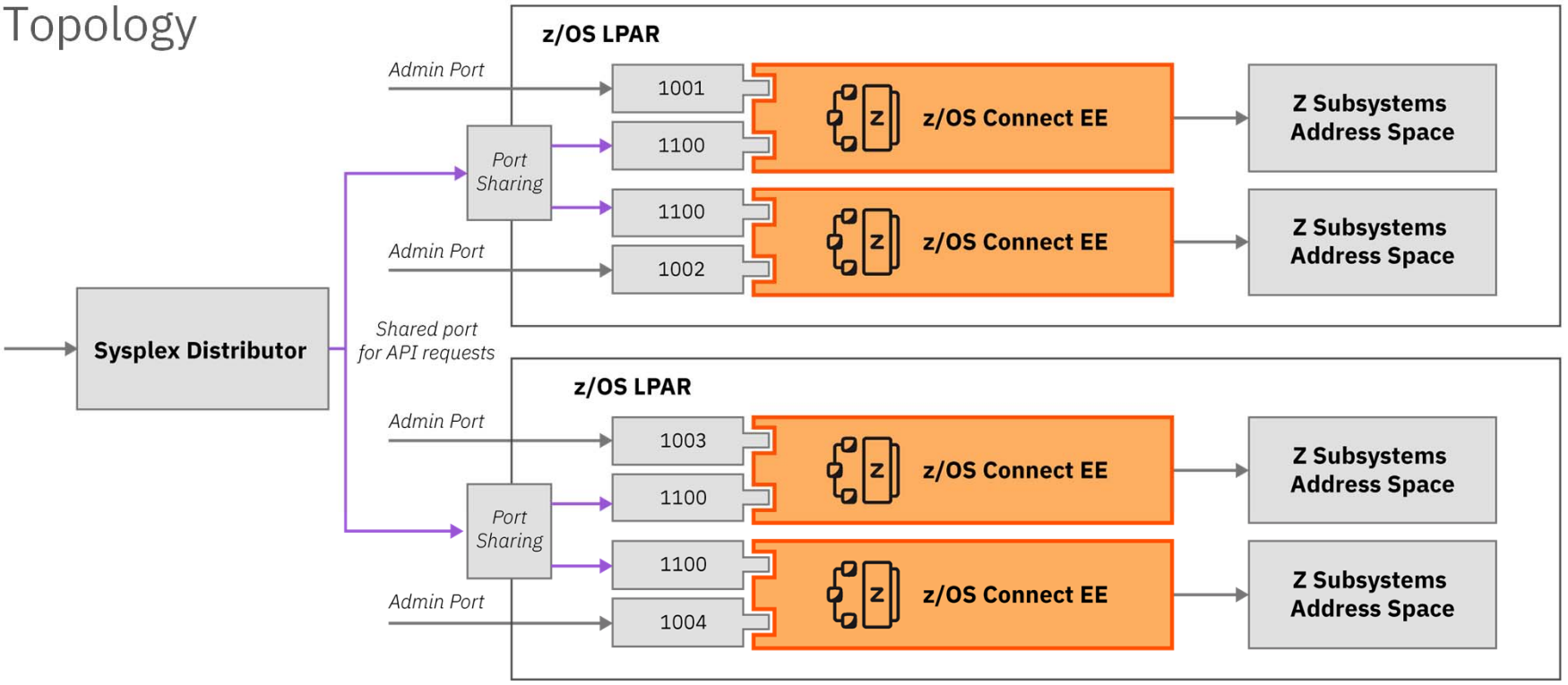
z/OS Connect EE



High Availability

z/OS Connect EE

Topology



ibm.biz/zosconnect-ha-concepts

ibm.biz/zosconnect-scenarios

/security

How is security implement?

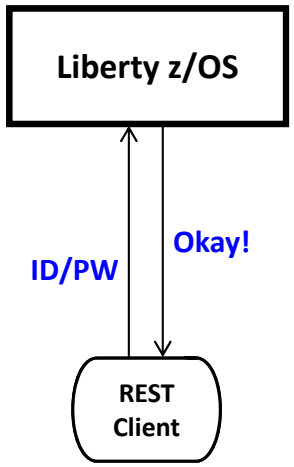
Common security challenges

- **End-to-end security** is hampered by the issue of how to provide secure access between middleware components that use disparate security technologies e.g. registries
  - This is a driver for implementing open security models like OAuth and OpenID Connect and standard tokens like JWT
- Security when using z/OS Connect is implemented in many products including z/OS Connect, WebSphere Liberty Profile on z/OS, SAF/RACF, CICS, IMS, Db2, MQ …
  - And these are all documented in different places
- Often security is at odds with **performance**, because the most secure techniques often involve the most processing overhead especially if not configured optimally

Authentication

Several different ways this can be accomplished:

Basic Authentication



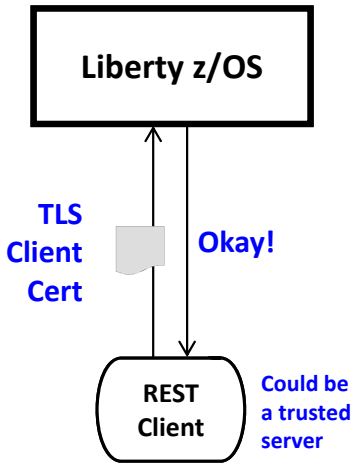
Server prompts for ID/PW

Client supplies ID/PW

Server checks registry:

- Basic (server.xml)
- LDAP
- SAF

Client Certificate



Server prompts for cert.

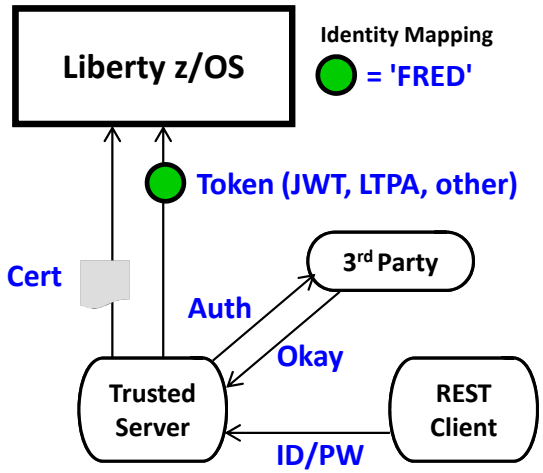
Client supplies certificate

Server validates cert and maps to an identity

Registry options:

- LDAP
- SAF

Third Party Authentication



Client authenticates to 3rd party sever

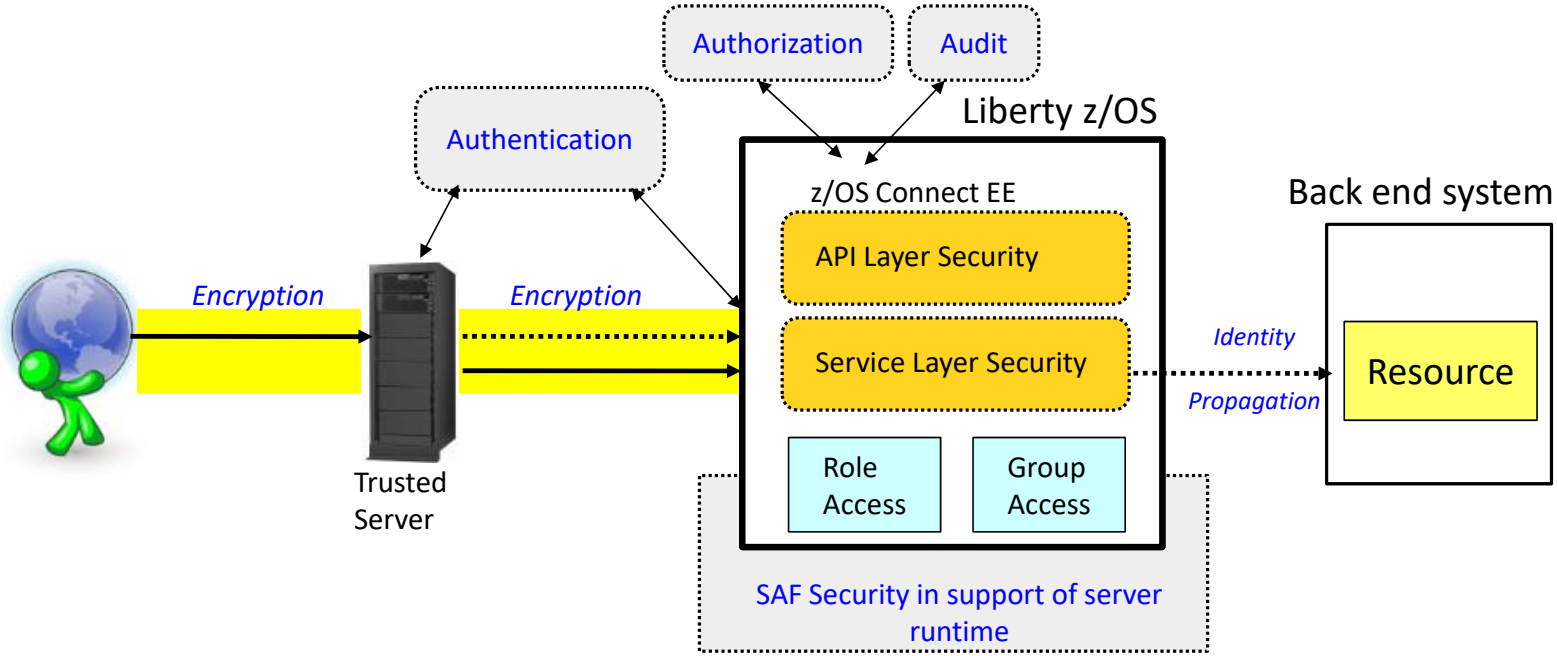
Client receives a trusted 3rd party token

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

Registry options:

- LDAP
- SAF

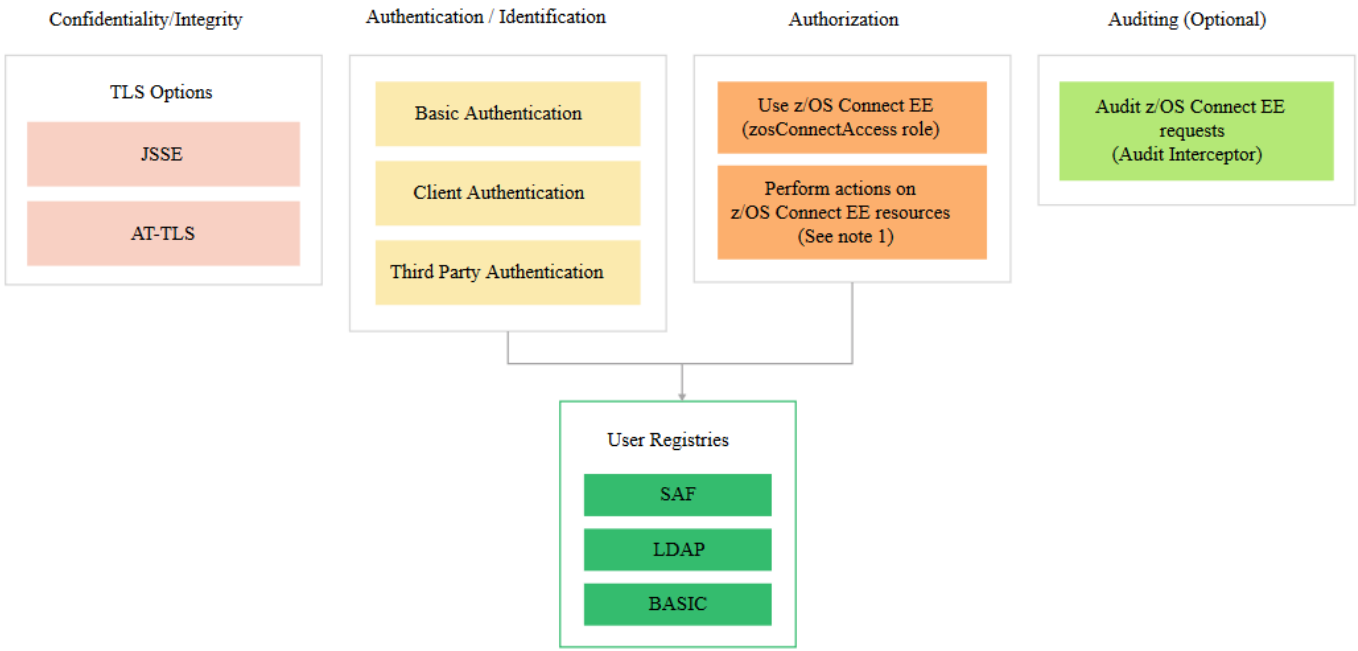
z/OS Connect EE API provider security overview



1. Authentication (basic, client certificates, 3rd party authentication)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (role and group access)
4. Audit
5. Configuring security with SAF
6. Back end identity propagation (CICS, IMS, Db2, MQ)

See Dev Center article "Securing APIs with z/OS Connect EE" overview of z/OS Connect EE security

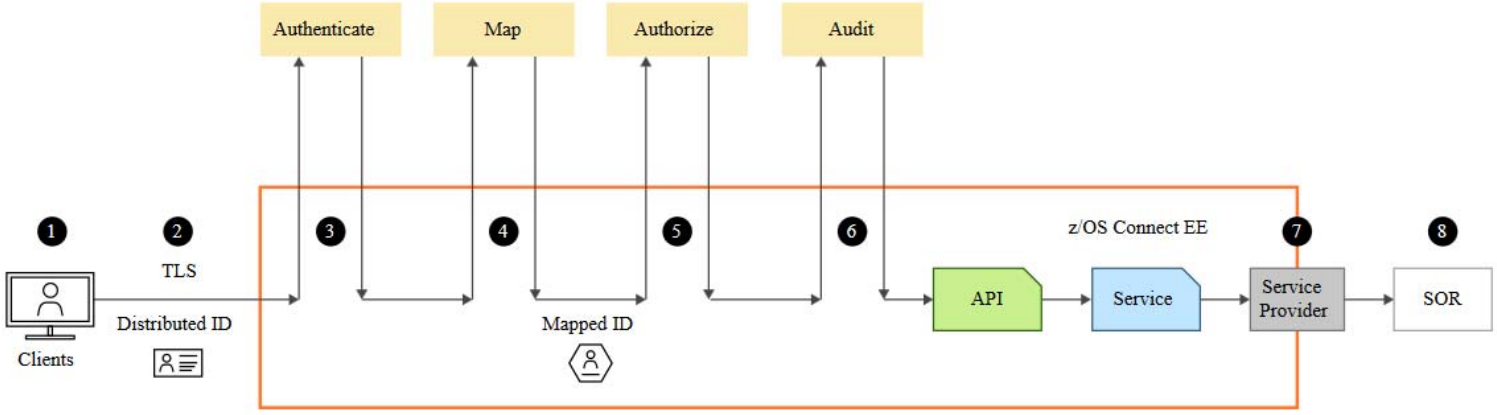
z/OS Connect EE security options



http://ibm.biz/zosconnect-security

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.

Typical z/OS Connect EE security flow



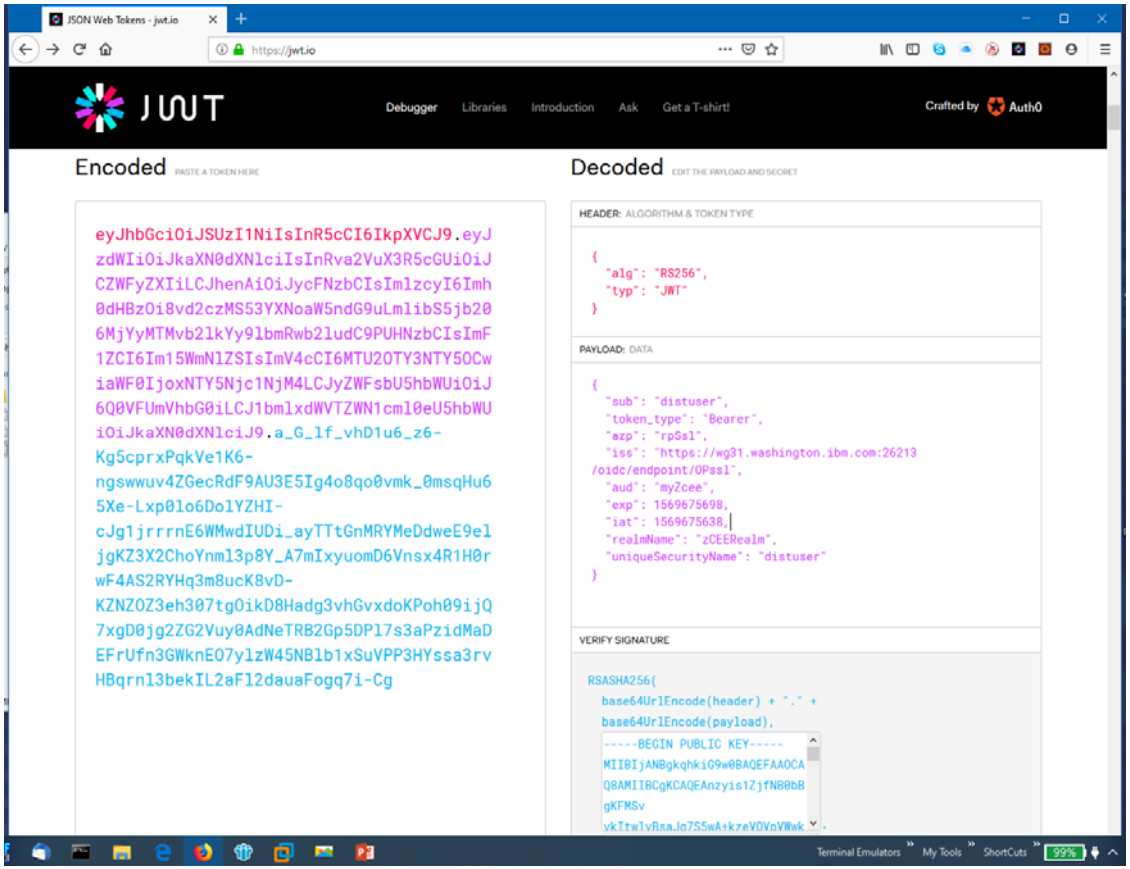
1. The credentials provided by the client
2. Secure the connection to the z/OS Connect EE server
3. Authenticate the client. This can be within the z/OS Connect EE server or by requesting verification from a third party server
4. Map the authenticated identity to a user ID in the user registry
5. Authorize the mapped user ID to connect to z/OS Connect EE and optionally authorize user to invoke actions on APIs
6. Audit the API request
7. 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
8. The program or database request may run in the SoR under the mapped ID

Security token types by z/OS Connect EE

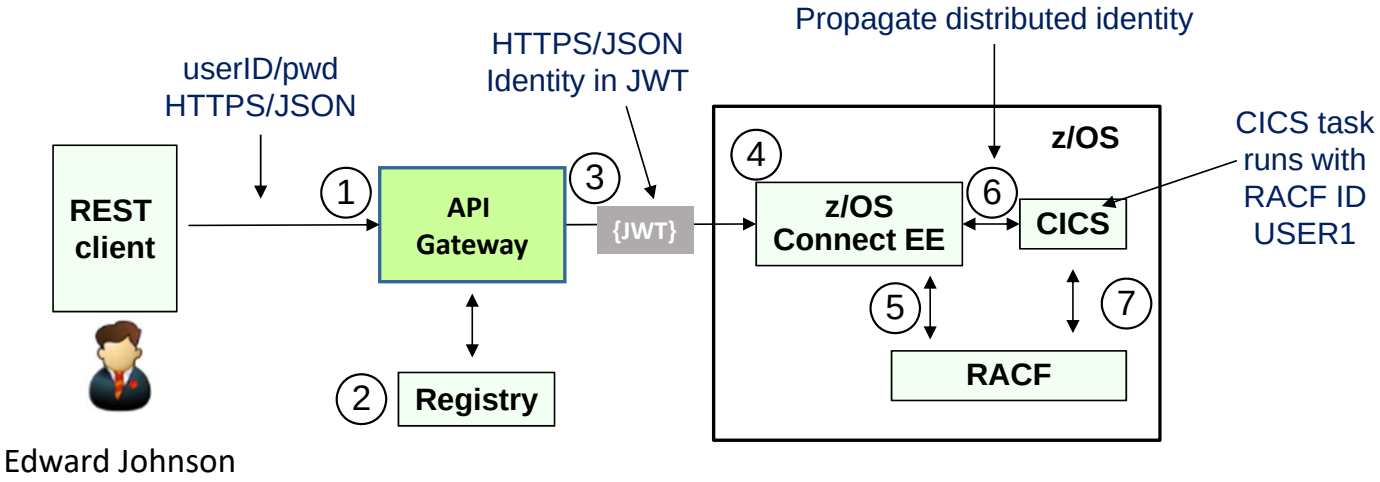
Token type	How used	Pros	Cons
LTPA	Authentication technology used in IBM WebSphere	• Easy to use with WebSphere and DataPower	• IBM Proprietary token
SAML	XML-based security token and set of profiles	• Token includes user id and claims • Used widely with SoR applications	• Tokens can be heavy to process • No refresh token
OAuth 2.0 access token	Facilitates the authorization of one site to access and use information related to the user's account on another site	• Used widely for SoE applications e.g with Google, Facebook, Microsoft, Twitter …	• Needs introspection endpoint to validate token
JWT	JSON security token format	• More compact than SAML • Ease of client-side processing especially mobile	

JWT (JSON Web Token)

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



Example scenario – security flow



RACMAP ID(USER1) MAP USERDIDFILTER(NAME('auser')) REGISTRY(NAME('*'))

1. User authenticates with the managed API using a "distributed" identity and a password
2. An external registry is used as the user registry for distributed users and groups
3. API Gateway generates a JWT and forwards the token with the request to z/OS Connect EE
4. z/OS Connect EE validates JWT
5. z/OS Connect EE calls RACF to map distributed ID to RACF user ID and authorizes access to API
6. z/OS Connect EE CICS service provider propagates distributed ID to CICS
7. CICS calls RACF to map distributed ID to RACF user ID and performs resource authorization checks

JWT used in scenario

```
{
 "alg": "RS256"
}
{
 "sub": auser",
 "token_type": "Bearer",
 "azp": "rpSsl",
 "iss": "https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl",
 "aud": "myZcee",,
 "realmName": "zCEERealm",
 "uniqueSecurityName": «auser"
}
RSASHA256(base64UrlEncode(header)+ base64UrlEncode(payload)
```

- The header contains an **alg** (algorithm) element value **RS256**
  - **RS256** (RSA Signature with SHA-256) is an asymmetric algorithm which uses a **public/private** key pair
  - **ES512** (Elliptic Curve Digital Signature Algorithm with SHA-512) link for more info
  - **HS256** (HMAC with SHA-256) is a symmetric algorithm with only one (**secret**) key
- The **iss** (issuer) claim identifies the principal that issued the JWT
- The **sub** (subject) claim **distuser** identifies the principal that is the subject of the JWT
- The **aud** (audience) claim **myZcee** identifies the recipients for which the JWT is intended

Configuring authentication with JWT

z/OS Connect EE can perform user authentication with JWT using the support that is provided by the *openidConnectClient-1.0* feature. The ***<openidConnectClient>*** element is used to accept a JWT token as an authentication token

```
<openidConnectClient id="RPssl" inboundPropagation="required"
    signatureAlgorithm="RS256" trustAliasName="JWT-Signer"
    trustStoreRef="jwtTrustStore"
     userIdentityToCreateSubject="sub" mapIdentityToRegistryUser="true"
    issuerIdentifier="https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl"
    authnSessionDisabled="true" audiences="myZcee"/>
```

- ***inboundPropagation*** is set to required to allow z/OS Connect EE to use the received JWT as an authentication token
- ***signatureAlgorithm*** specifies the algorithm to be used to verify the JWT signature
- ***trustStoreRef*** specifies the name of the keystore element that defines the location of the validating certificate
- ***trustAliasName*** gives the alias or label of the certificate to be used for signature validation
- ***userIdentityToCreateSubject*** indicates the claim to use to create the user subject
- ***mapIdentityToRegistryUser*** indicates whether to map the retrieved identity to the registry user
- ***issuerIdentifier*** defines the expected issuer
- ***authnSessionDisabled*** indicates whether a WebSphere custom cookie should be generated for the session
- ***audiences*** defines a list of target audiences

See Dev Center article "Using a JWT with z/OS Connect EE" for full description of scenario

Using authorization filters with z/OS Connect EE

Authentication filter can be used to filter criteria that are specified in the **authFilter** element to determine whether certain requests are processed by certain providers, such as OpenID Connect, for authentication.

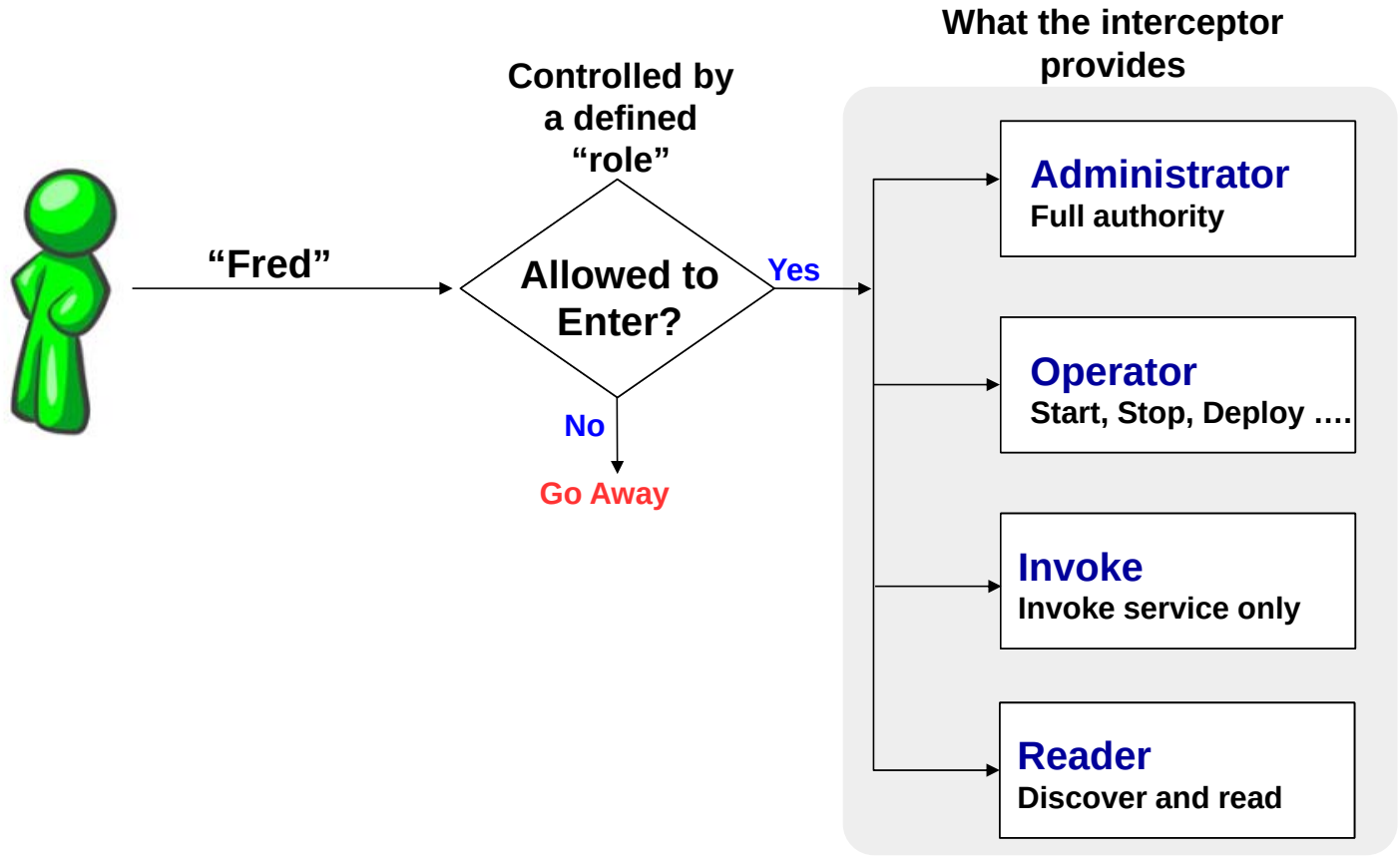
```
<openidConnectClient id="RPssl" inboundPropagation="required"
     signatureAlgorithm="RS256" trustAliasName="JWT-Signer"
     trustStoreRef="jwtTrustStore"
      userIdentityToCreateSubject="sub" mapIdentityToRegistryUser="true"
     issuerIdentifier="https://wg31.washington.ibm.com:26213/oidc/endpoint/OPssl"
     authnSessionDisabled="true" audiences="myZcee"
     authFilterRef="JwtAuthFilter"/>
<authFilter id="API Gateway">
     <remoteAddress id="ApiAddress" ip="10.7.1.*" matchType="equals"/>
</authFilter>
<authFilter id="PhoneBook">
      <requestUrl id="URL"  urlPattern="/phoneBook/*" matchType="equals"/> </authFilter>
<authFilter id="JwtAuthFilter" >
  <requestHeader id="authHeader" name="Authorization" value="Bearer" matchType="contains"/>
</authFilter>
```

Some alternative filter types

- A ***remoteAddress*** element is compared against the TCP/IP address of the client that sent the request.
- The ***host*** element is compared against the "Host" HTTP request header, which identifies the target host name of the request.
- The ***requestUrl*** element is compared against the URL that is used by the client application to make the request.

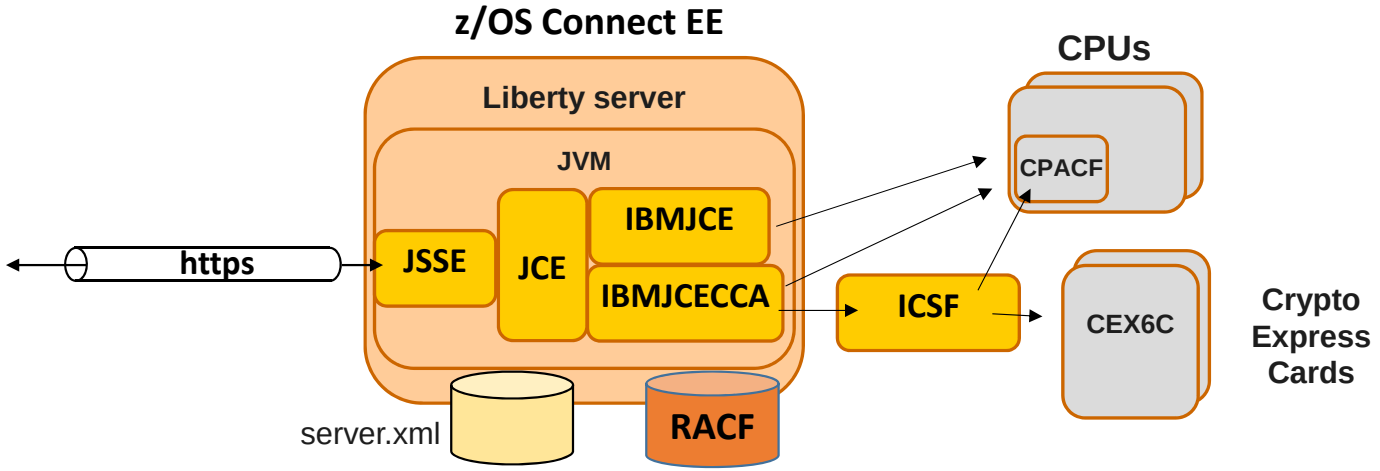
Authorization interceptor

The “authorization interceptor” is a supplied piece of interceptor code that will check to see if the user has the authority to perform the action requested:



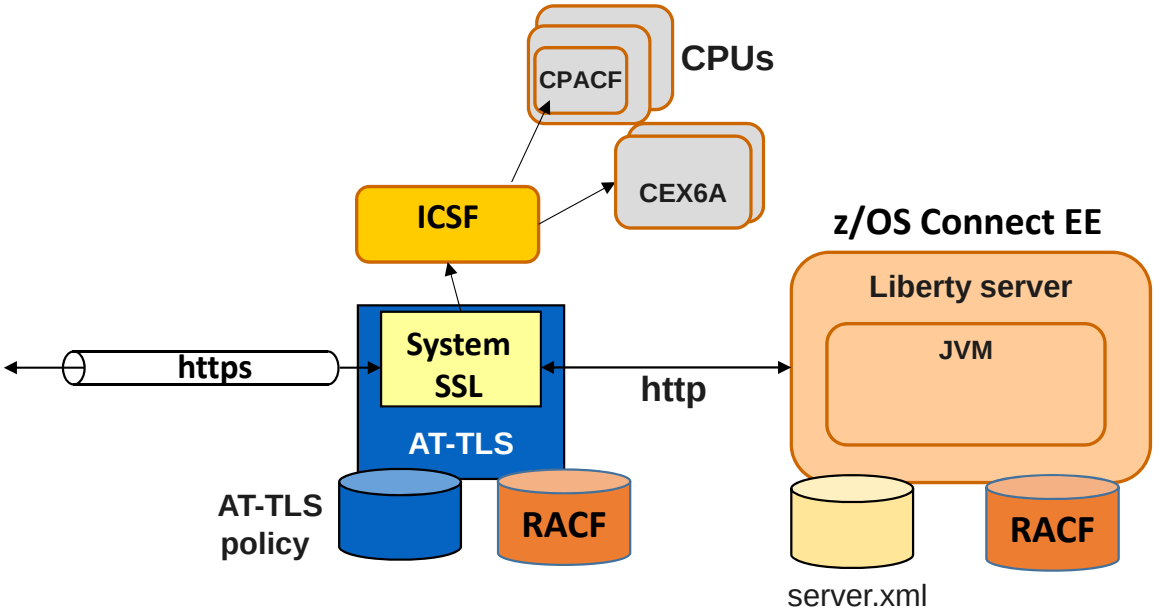
Encryption

Using JSSE with z/OS Connect EE



- z/OS Connect EE support for SSL/TLS is based on **Liberty server** support
- **Java Secure Socket Extension** (JSSE) API provides framework and Java implementation of SSL and TLS protocols used by Liberty HTTPS support
- **Java Cryptography Extension** (JCE) is standard extension to the Java Platform that provides implementation for cryptographic services
- **IBM Java SDK** for z/OS provides two different JCE providers, **IBMJCE** and **IBMJCECCA**

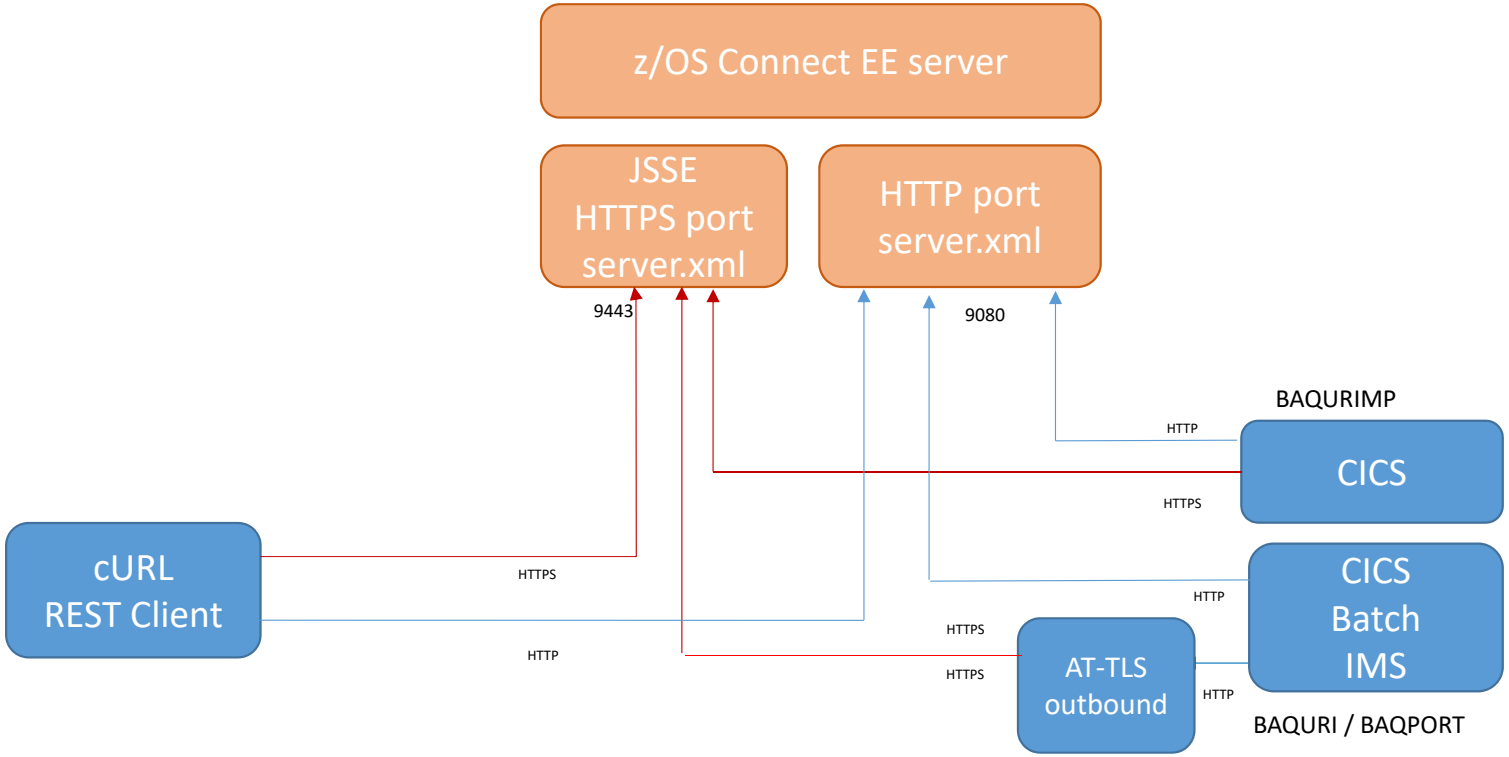
Using AT-TLS with z/OS Connect EE



- **Application Transparent TLS** (AT-TLS) creates a secure session on behalf of z/OS Connect
- Only define http ports in server.xml (z/OS Connect does not know that TLS session exists)
- Define TLS protection for all applications (including z/OS Connect) in **AT-TLS policy**
- AT-TLS uses **System SSL** which exploits the CPACF and Crypto Express cards via ICSF

AT-TLS Inbound to zCEE Scenarios

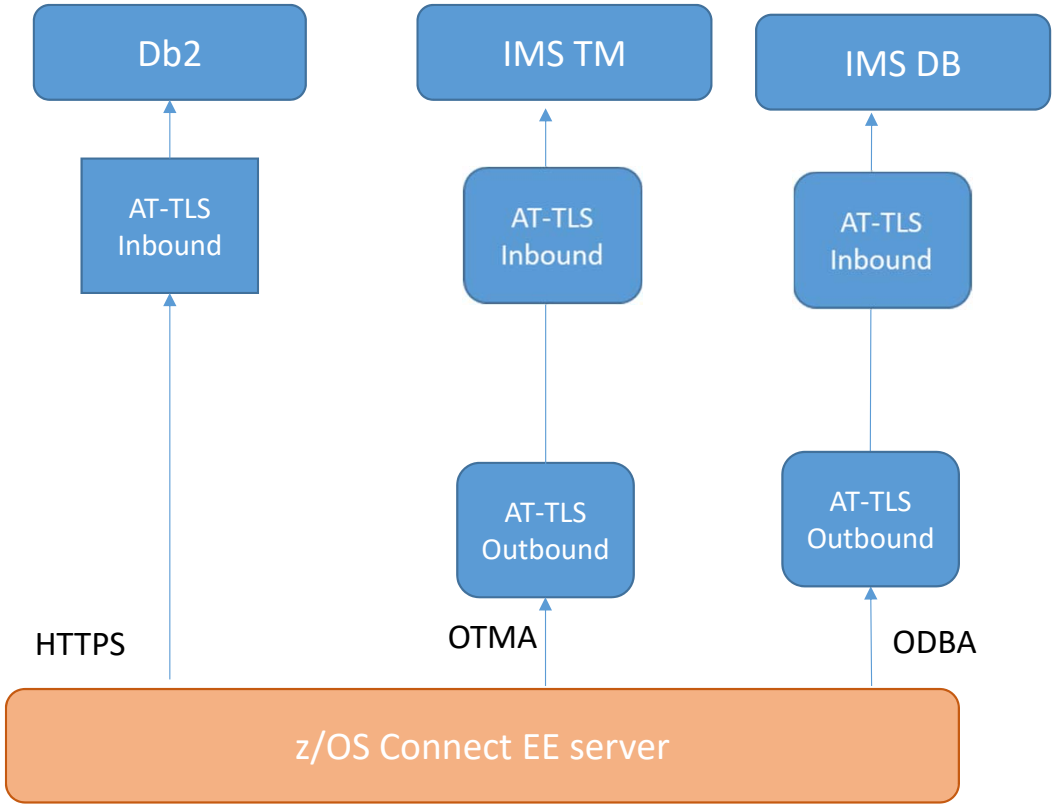
z/OS Connect EE



A Outbound Policy allows non-SSL clients to connection to HTTPS ports

AT-TLS Outbound from zCEE Scenarios (HTTPS/OTMA/ODBA)

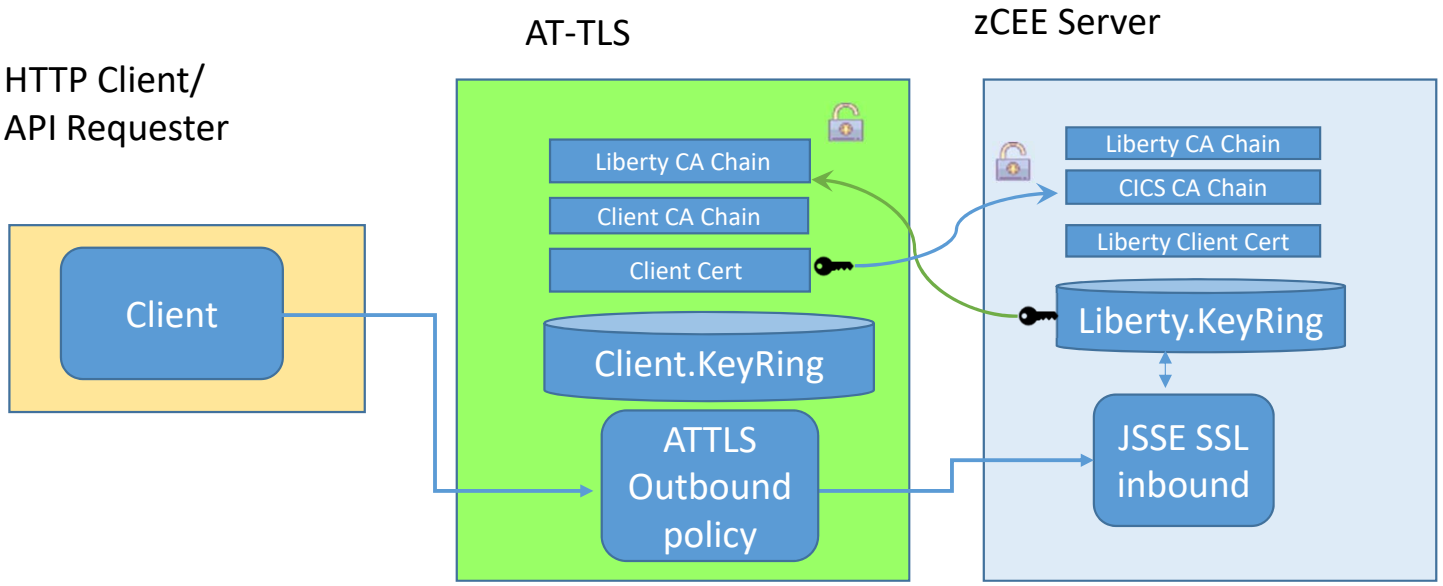
z/OS Connect EE



Inbound Policies Provide TLS support for incoming requests

A Outbound Policies provide SSL support for outgoing requests

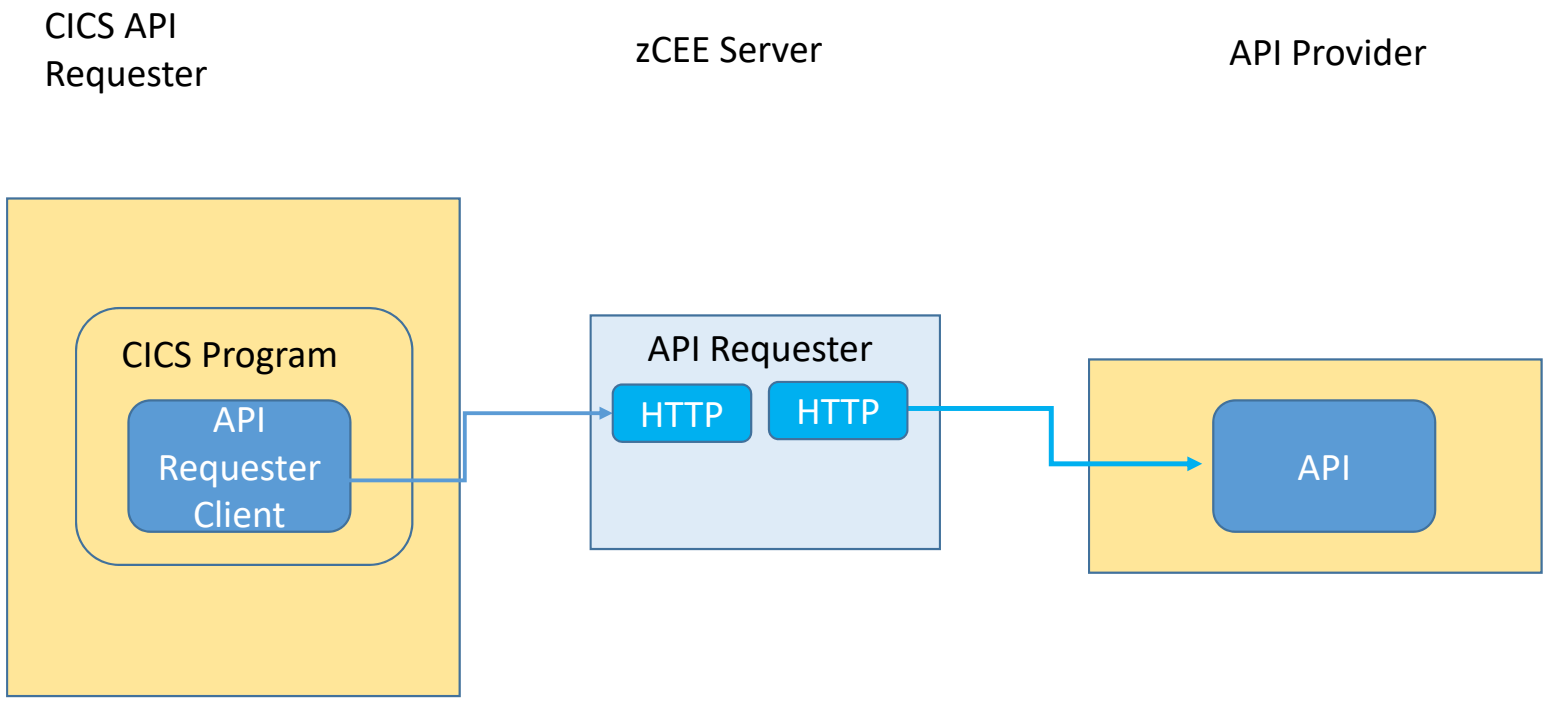
Client AT-TLS Handshake Flow



Outbound Policy acts a surrogate SSL client

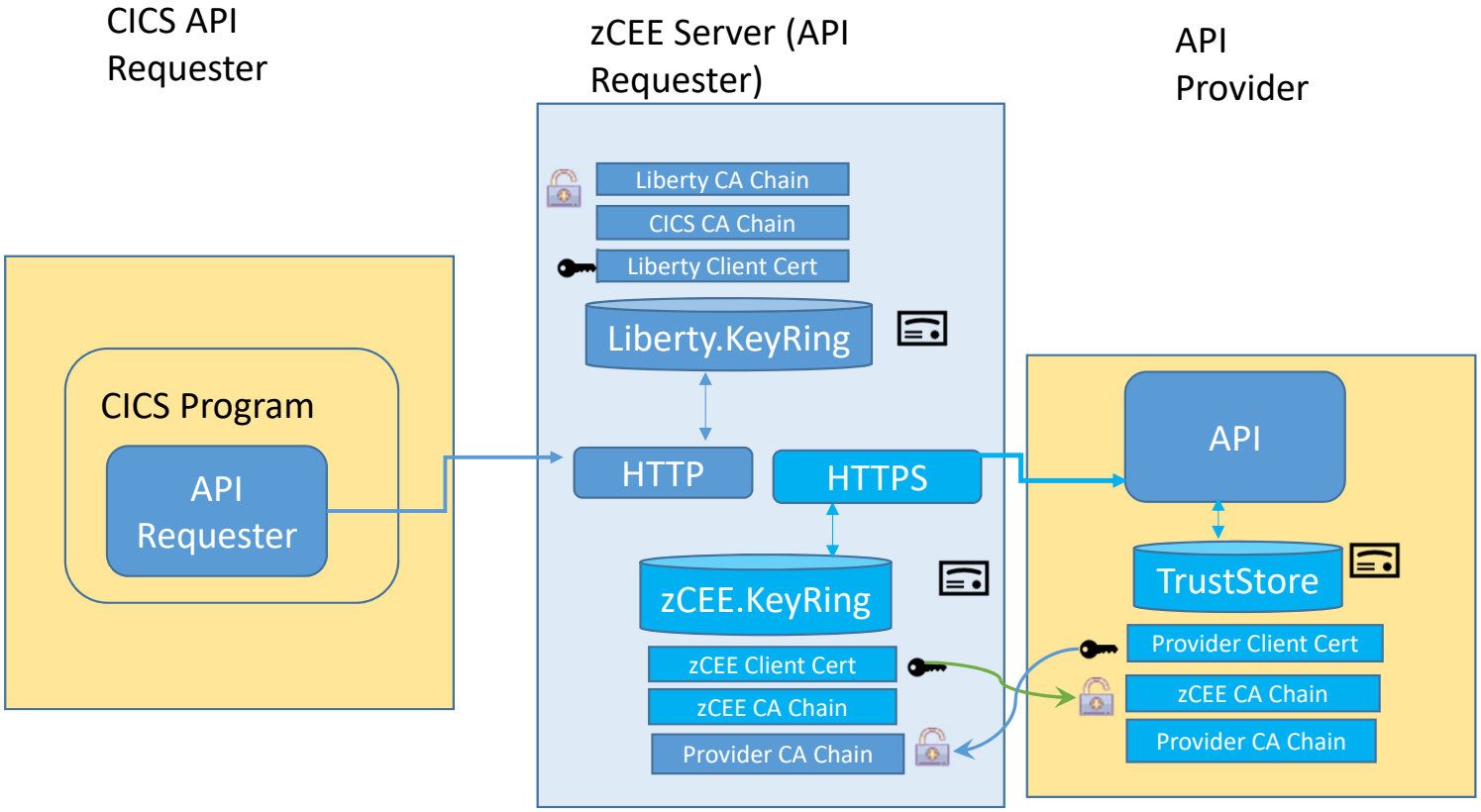
API Requester - Non-TLS Handshake Flow

z/OS Connect EE



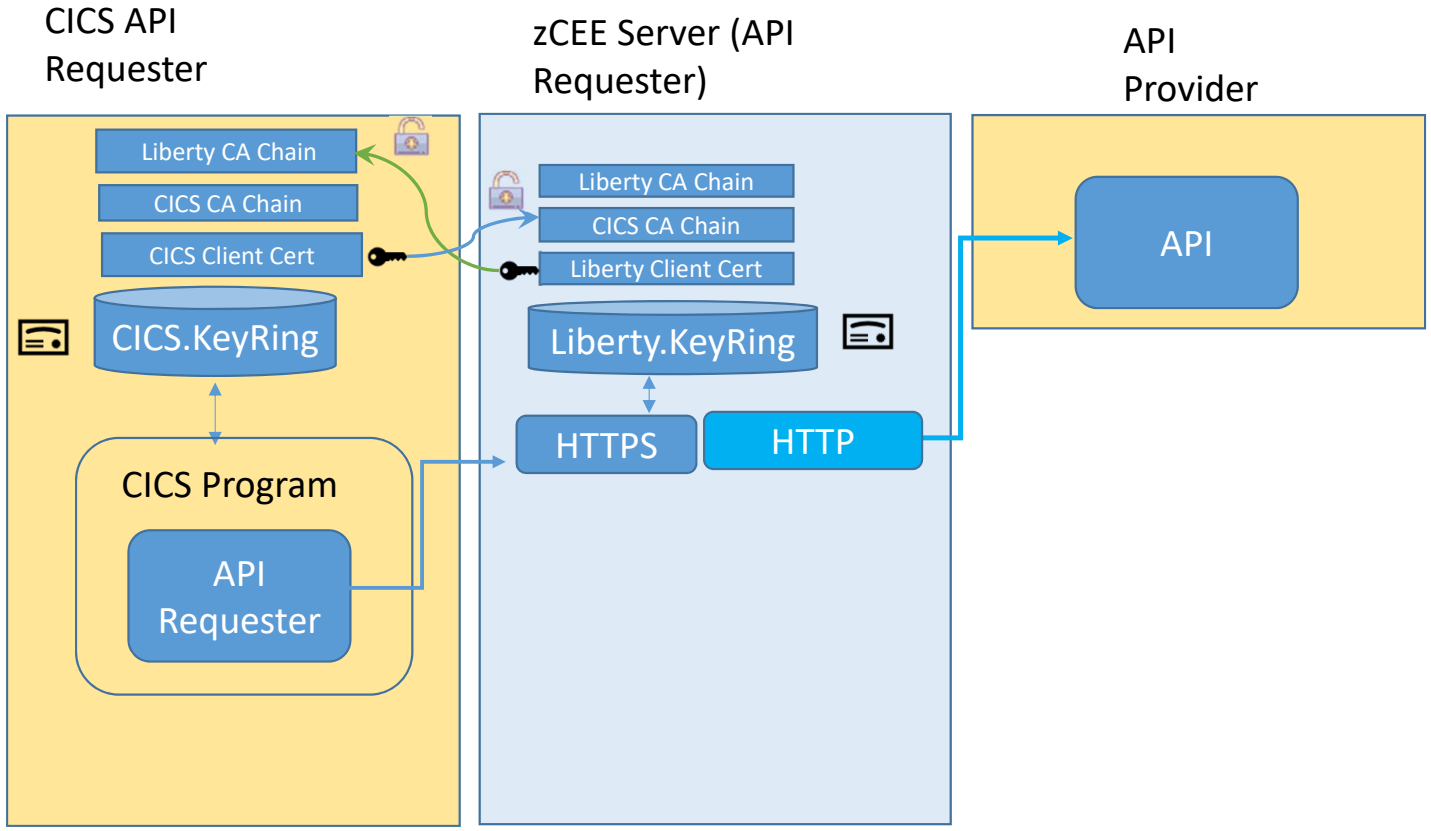
API Requester - TLS Handshake Flow

z/OS Connect EE



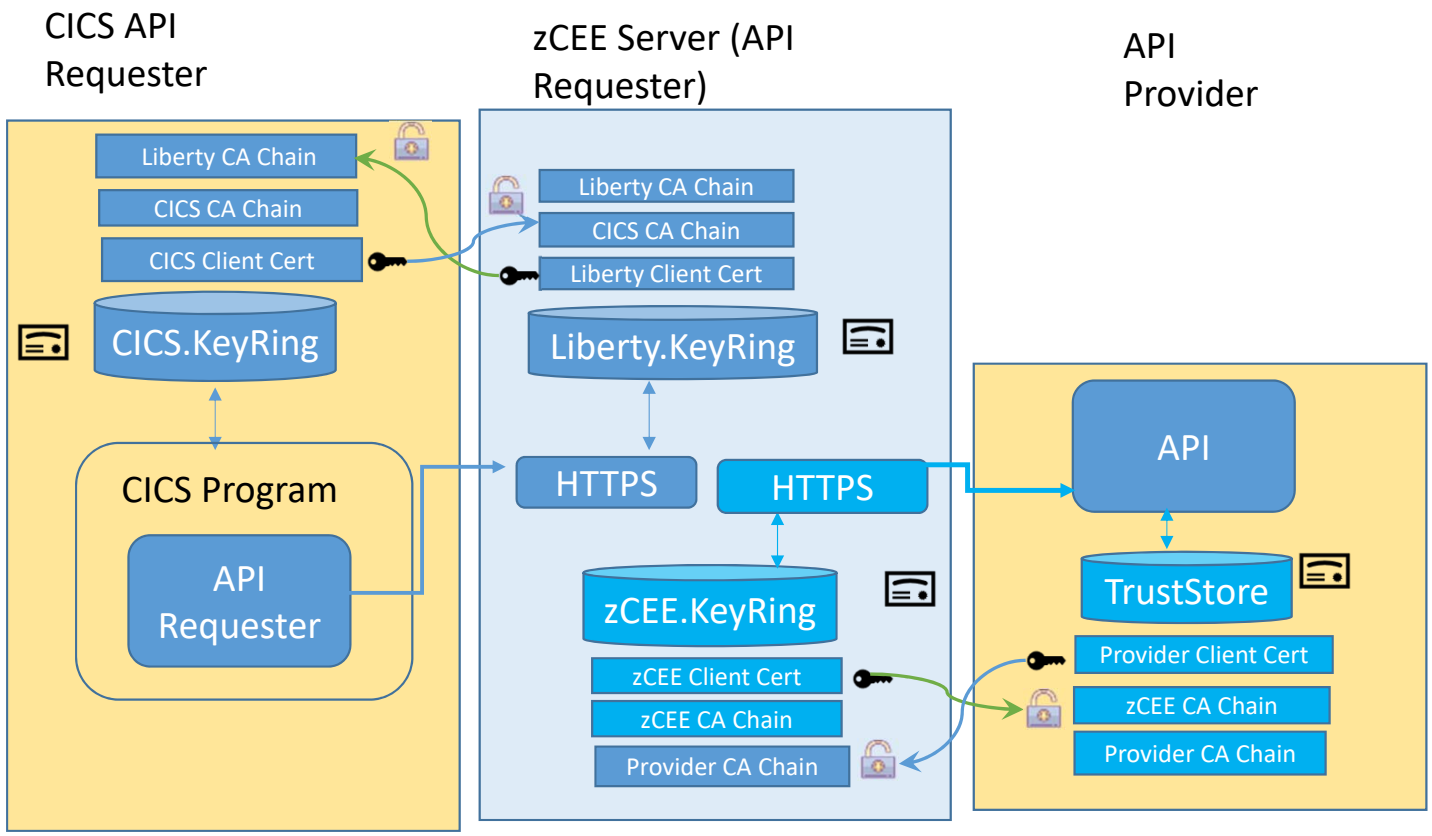
API Requester - TLS Handshake Flow

z/OS Connect EE



API Requester - TLS Handshake Flow

z/OS Connect EE



Cyphers

- During the TLS handshake, the TLS protocol and data exchange cipher are negotiated
- Choice of cipher and key length has an impact on performance
- You can restrict the protocol (SSL or TLS) and ciphers to be used
- Example setting server.xml file

```
<ssl id="DefaultSSLSettings"
keyStoreRef="defaultKeyStore" sslProtocol="TLSv1.2"
enabledCiphers="TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384"/>
```

- This configures use of TLS 1.2 and two supported ciphers
- It is recommended to control what ciphers can be used in the server rather than the client

Persistent connections

- Persistent connections can be used to avoid too many handshakes
- Configured by setting the `keepAliveEnabled` attribute on the httpOptions element to **true**
- Example setting server.xml file

```
<httpEndpoint host="*" httpPort="80" httpsPort="443"
id="defaultHttpEndpoint" httpOptionsRef="httpOpts"/>

<httpOptions id="httpOpts" keepAliveEnabled="true"
maxKeepAliveRequests="500" persistTimeout="1m"/>
```

- This sets the connection timeout to **1 minute** (default is 30 seconds) and sets the maximum number of persistent requests that are allowed on a single HTTP connection to **500**
- It is recommended to set a maximum number of persistent requests when connection workload balancing is configured
- It is also necessary to configure the client to support persistent connections

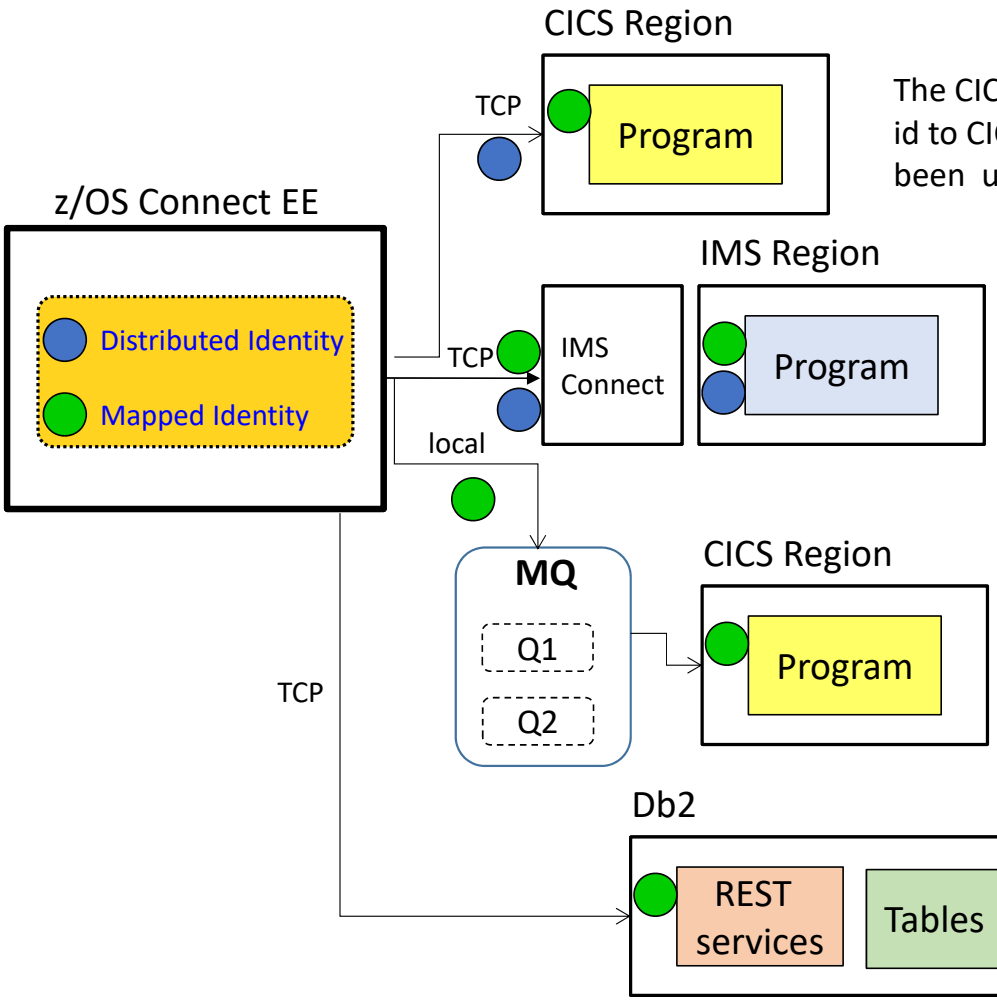
SSL sessions

- When connections timeout, it is still possible to avoid the impact of full handshakes by reusing the SSL session id
- Configured by setting the `sslSessionTimeout` attribute on the sslOptions element to an amount of time
- Example setting server.xml file

```
<httpEndpoint host="*" httpPort="80" httpsPort="443"
id="defaultHttpEndpoint" httpOptionsRef="httpOpts“
sslOptionsRef="mySSLOptions"/>

<httpOptions id="httpOpts" keepAliveEnabled="true"
maxKeepAliveRequests=“100" persistTimeout="1m"/>

<sslOptions id="mySSLOptions" sslRef="DefaultSSLSettings"
sslSessionTimeout=“10m”/>
```

- This sets the timeout limit of an SSL session to **10 minutes** (default is 8640ms)
- SSL session ids are not shared across z/OS Connect servers

Flowing an identity to the back end



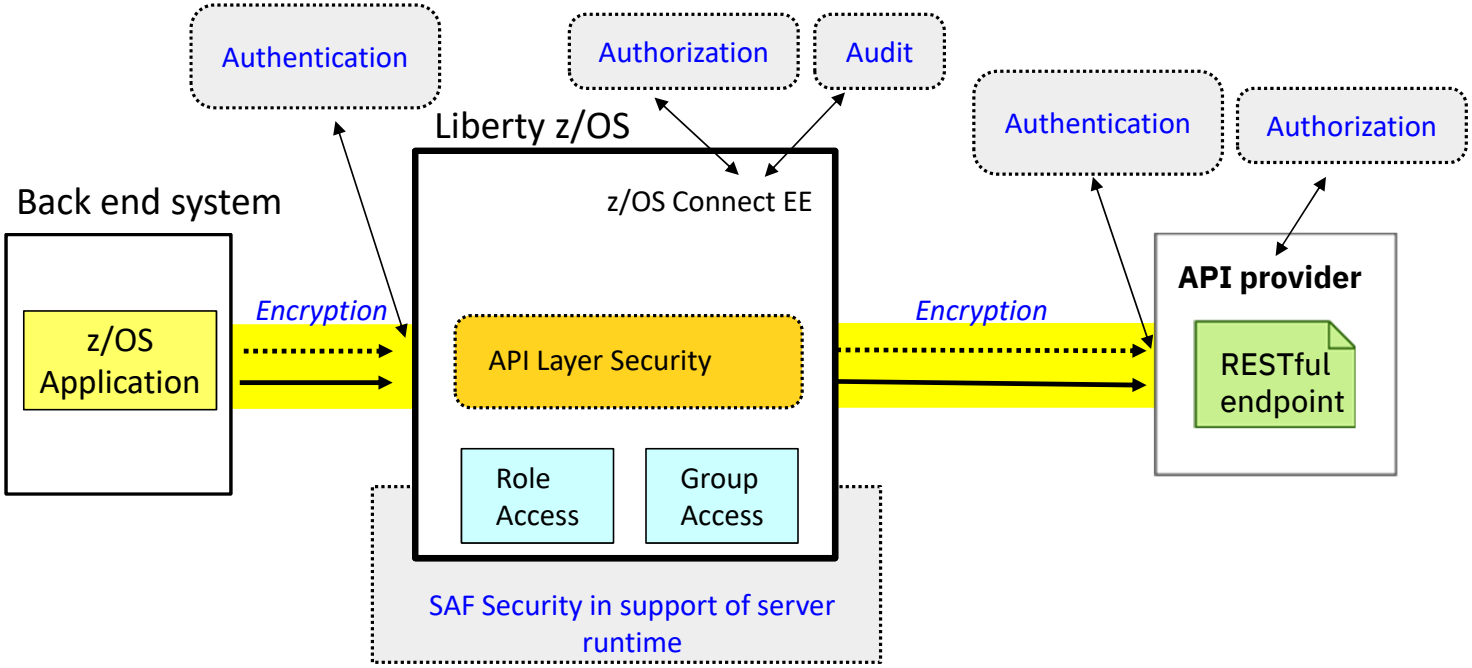
The CICS SP propagates the distributed id to CICS (or sends the SAF id if it has been used for authentication)

The IMS SP asserts the mapped id to IMS but also sends the distributed id for audit purposes (V3.0.33) added support for Passtickets.

The MQ SP asserts the mapped id to CICS

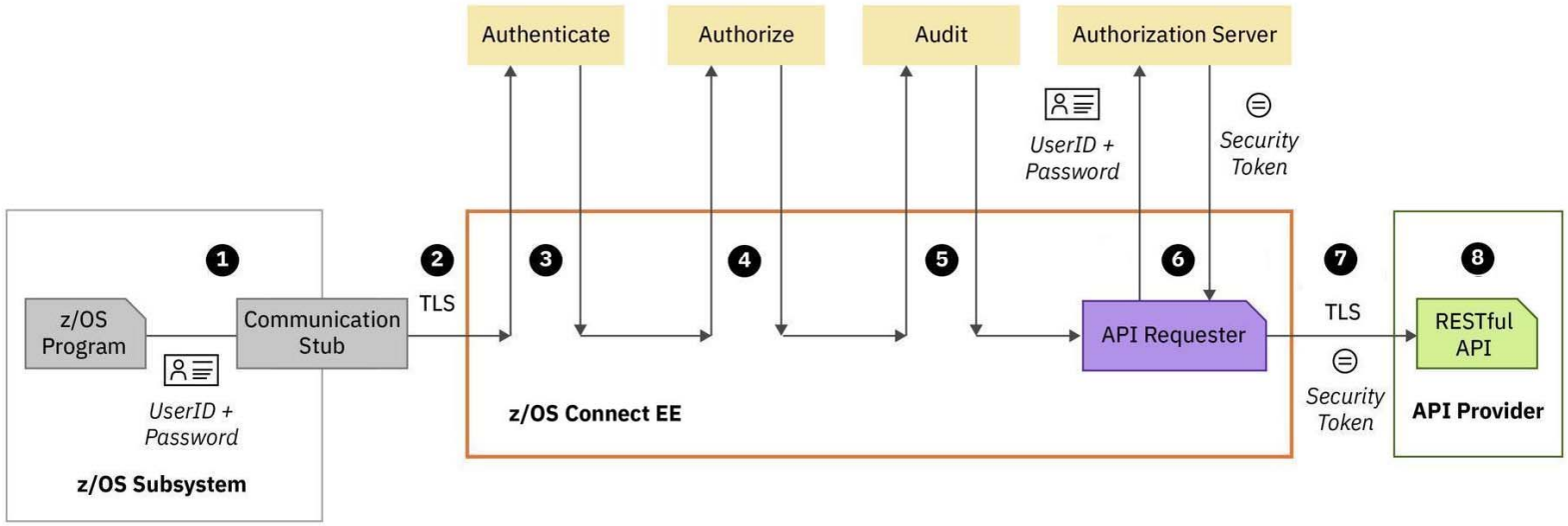
The REST client SP can request a Passticket and send with mapped ID to Db2 (available today in V3.0.15)

API requester security – overview



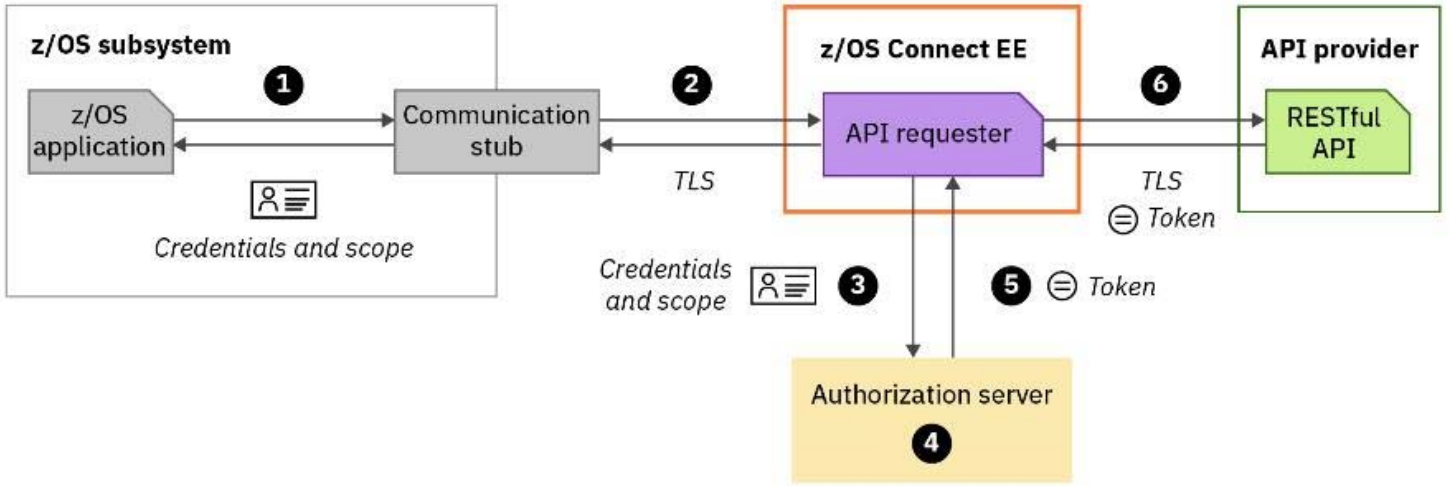
1. Authentication (basic, client certificate)
2. Encryption (aka "SSL" or "TLS")
3. Authorization (OAuth)
4. Audit
5. Configuring security with SAF

Typical z/OS Connect EE security flow



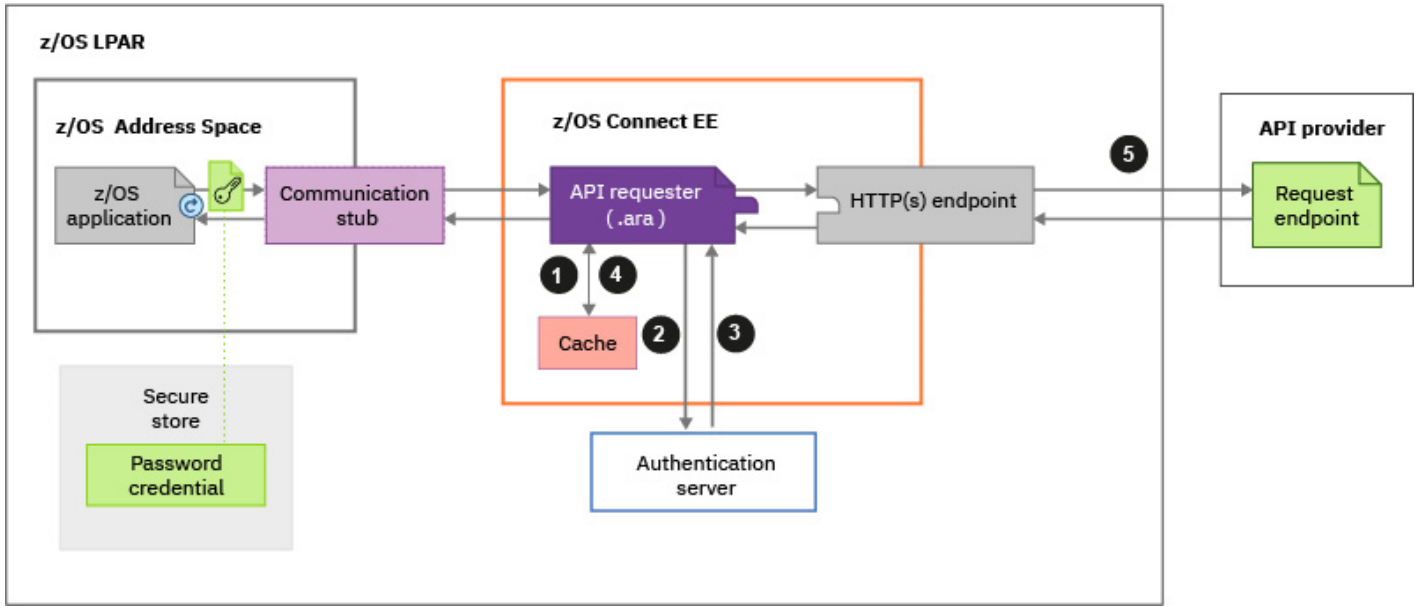
1. A user ID and password can be used for basic authentication by the z/OS Connect EE server
2. Connection between the CICS, IMS, or z/OS application and the z/OS Connect EE server can use TLS
3. Authenticate the CICS, IMS, or z/OS application.
4. Authorize the authenticated user ID to connect to z/OS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

Calling an API with OAuth 2.0 support



MOVE user TO BAQ-OAUTH-USERNAME
MOVE password TO BAQ-OAUTH-PASSWORD
MOVE clientId to BAQ-OAUTH-CLIENTID
MOVE clientSecret to BAQ-OAUTH-SECRET

Calling an API with JWT support



MOVE user TO BAQ-TOKEN-USERNAME
MOVE password TO BAQ-TOKEN-PASSWORD

/questions?thanks=true

Thank you for listening.

- Material can be downloaded from:
  http://tinyurl.com/y28fsezs
- z/OS Connect EE Users Group
  https://www.linkedin.com/groups/8731382/