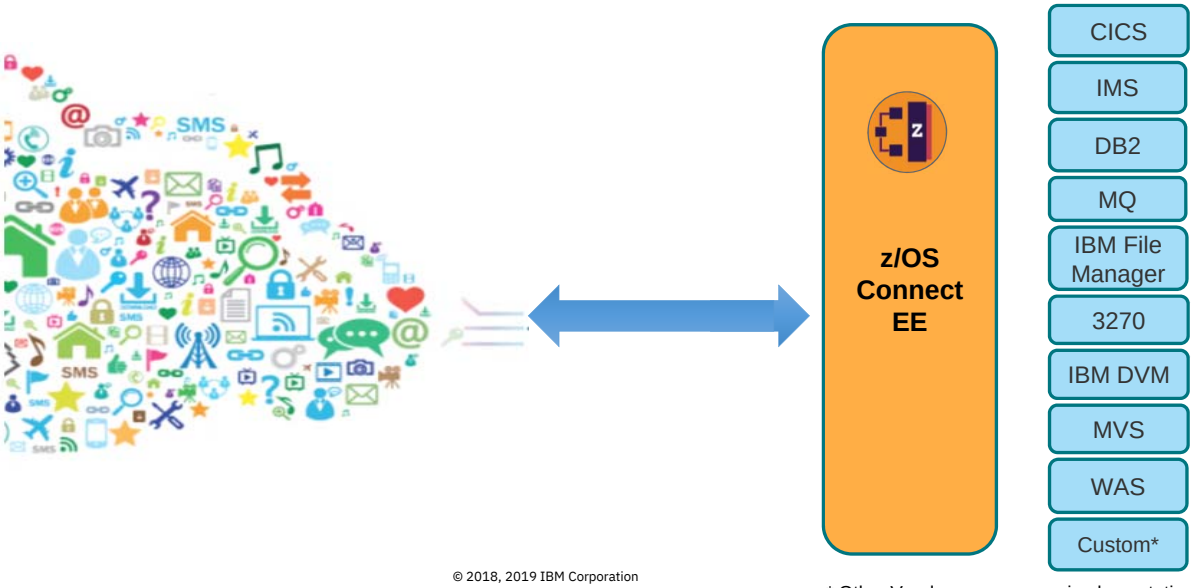
# Agenda

- z/OS Connect Introduction and overview
- Self paced, hands-on exercises to API enable z application from various sub-systems, e.g.
  - CICS
  - DB2
  - IMS/TM
  - MQ
  - IBM DVM
  - IBM File Manager
  - MVS Batch
  - Outbound REST APIs
  - 3270 screen based applications
- z/OS Connect Security



## z/OS Connect EE exposes z/OS resources to the "cloud" via RESTful APIs






* Other Vendors or your own implementation

# /what_is_REST?

What makes an API “RESTful”?



# REST is an Architectural Style

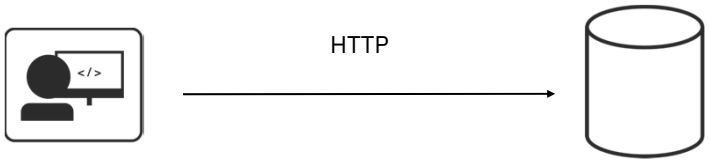
z/OS Connect EE

**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

z/OS Connect EE

- 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



7

# Roast API Recipe

z/OS Connect EE

(How not to do REST…)

1. Take a SOAP/XML web service name, add a "/" before it.
2. Choose randomly an HTTP method between GET, PUT, POST, DELETE.
3. Transform input/output data from XML to JSON.
4. If the method is GET or DELETE, put all parameters in query variables.
5. And be sure to always return HTTP status 200.



Source: apihandyman.io

# RESTful Examples

z/OS Connect EE

**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. Unique URIs for different 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**

```
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 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?

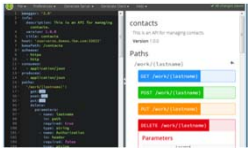
z/OS Connect EE

It is more than just an API framework

There are a number of tools available to aid consumption:

### Write Swagger

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



### Read Swagger

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

### Consume Swagger

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


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

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



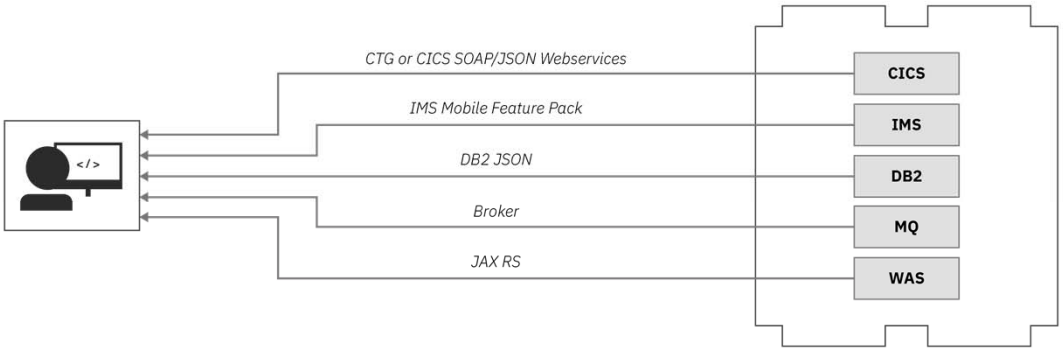


# Why /zos_connect_ee?

Truly RESTful APIs to and from your mainframe.

# Can't we do REST and JSON today?

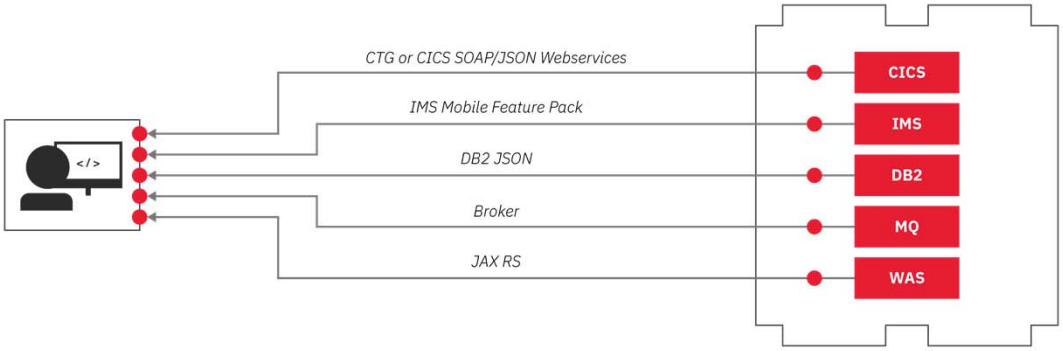
z/OS Connect EE






# 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 is needed!

z/OS Connect EE

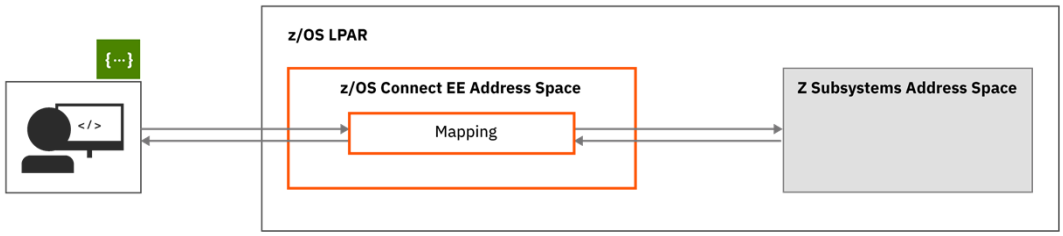

- Single Configuration Administration
- Single Security Administration
- With sophisticated mapping of truly RESTful APIs to existing mainframe and services data without writing any code.



# Let's Start with Data mapping

z/OS Connect EE

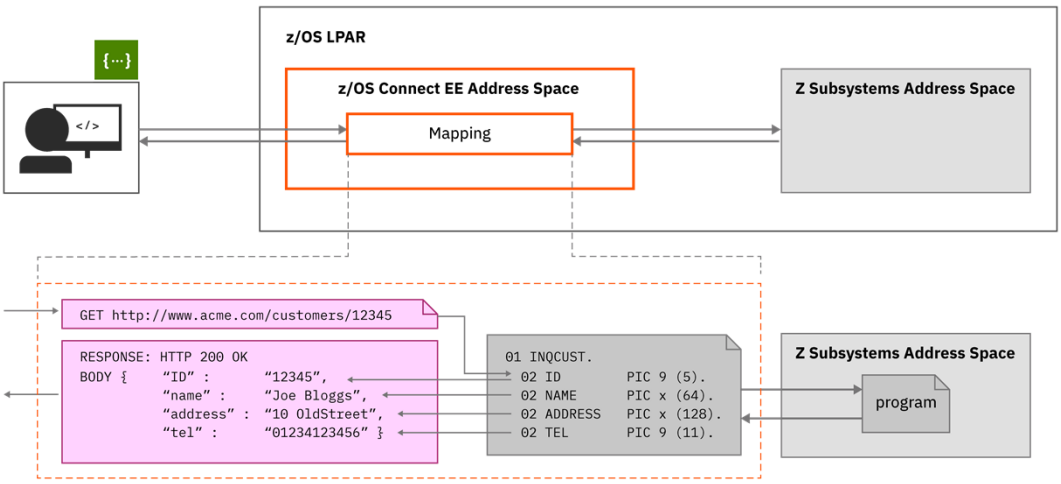
Converting from JSON to the target's subsystem's format



# 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 99 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
                },
```

COBOL Source v JSON

“name”:”Mitch Johnson”,
“creditScore”:100

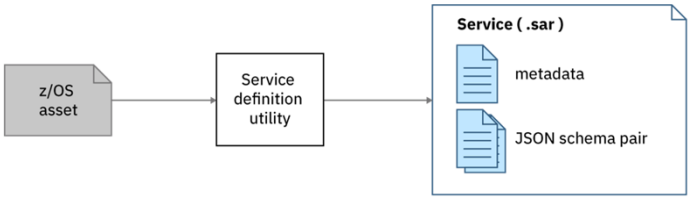
All data is sent as character strings and numeric precision and sign bit is removed as an issue

# Six Steps to expose a z/OS application

z/OS Connect EE

## 1. Create your service definition

To start mapping an API, z/OS Connect EE needs a representation of the underlying z/OS application: a **Service Archive file** (`.sar`).




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

- API Toolkit (CICS and IMS)
- BAQLS2JS (MQ and WOLA)
- z/OS Connect EE Build Toolkit (DB2 and HATS)
- DVM Toolkit

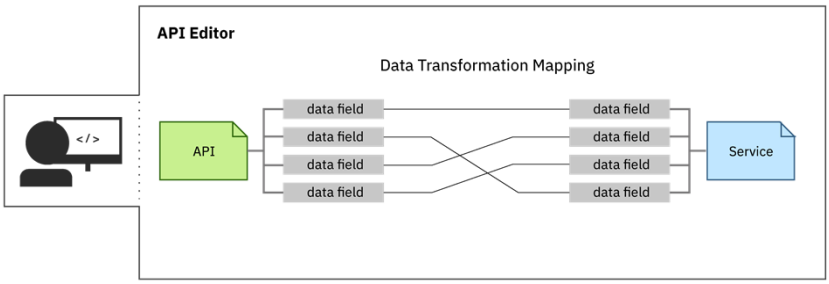
ibm.biz/zosconnect-sar-creation



# Six Steps to expose a z/OS application

z/OS Connect EE

## 2. Create your API




Import your `.sar` file into the **API toolkit**, and start designing your API.

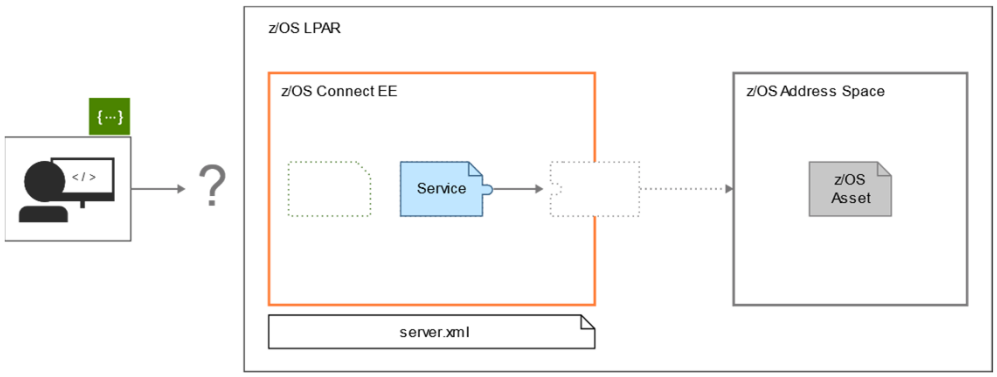
From the editor, create an **API Archive file** (`.aar`), which describes your API and how it maps to underlying services.

ibm.biz/zosconnect-create-api

# Six Steps to expose a z/OS application

z/OS Connect EE

## 3. Deploy your 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.

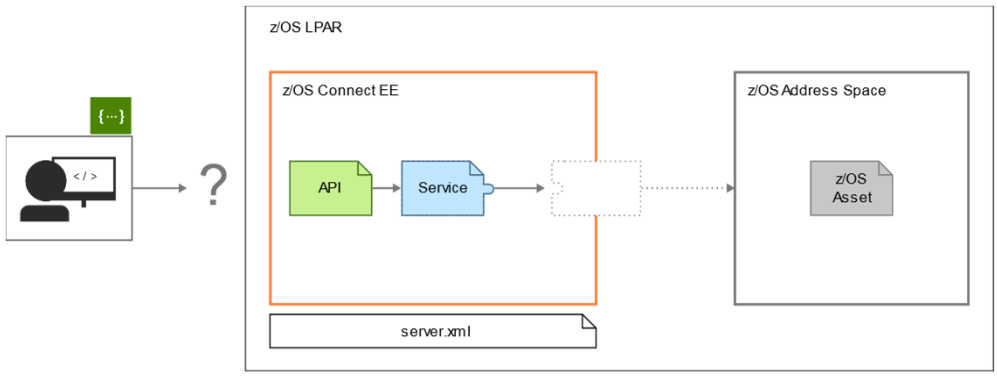
ibm.biz/zosconnect-define-services



# Six Steps to expose a z/OS application

z/OS Connect EE

## 4. Deploy your API



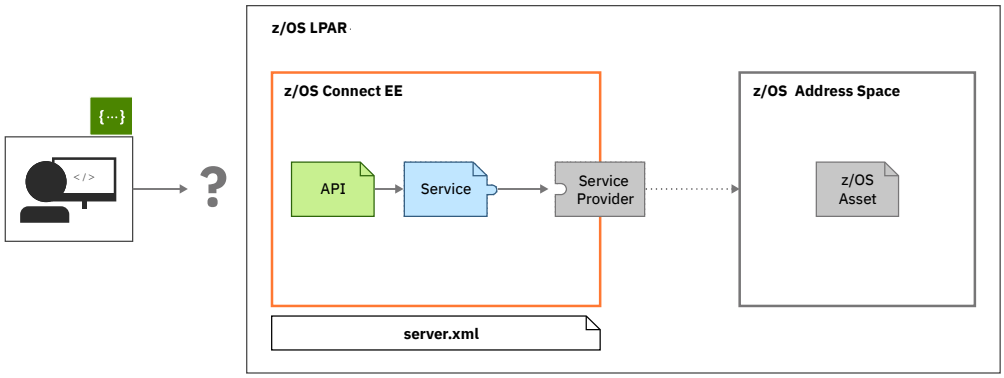

Deploy your API using the right-click deploy in **the API toolkit**, or by copying the `.aar` file to the *apis* directory.

ibm.biz/zosconnect-deploy-api

# Six Steps to expose a z/OS application

z/OS Connect EE

## 5. Configure your service provider




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

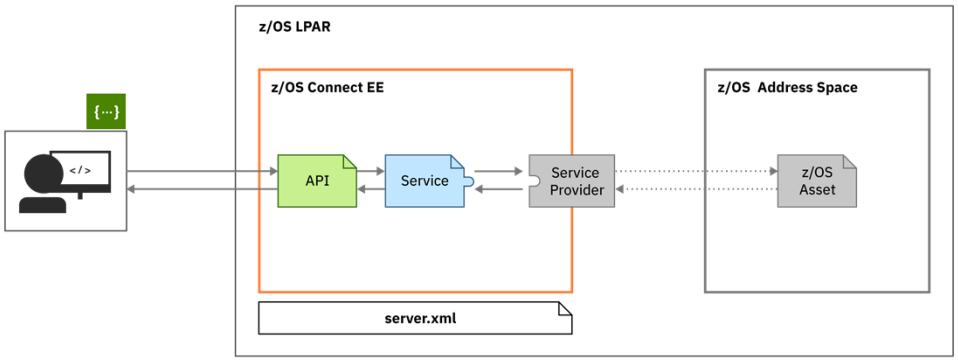
ibm.biz/zosconnect-configuring



# Six Steps to expose a z/OS application

z/OS Connect EE

## 6. Done




Your API is ready to be consumed: go tell your developers!

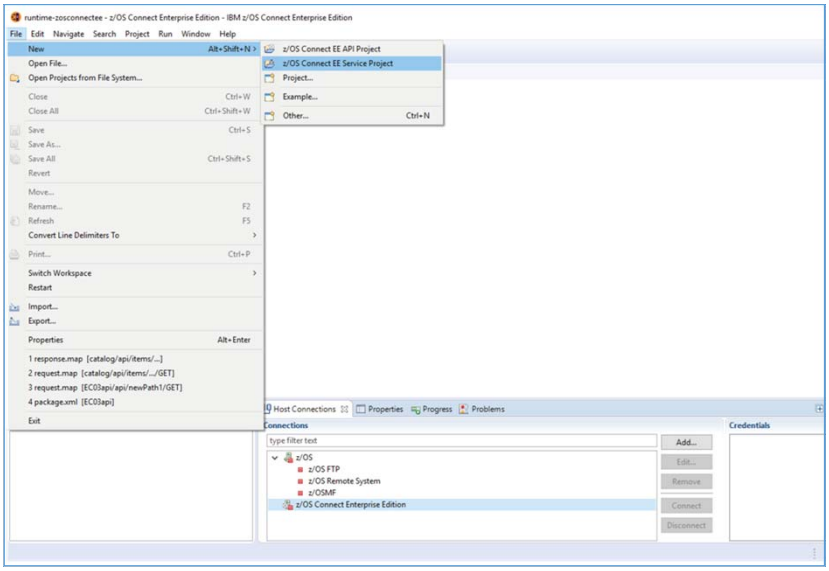
# /api_toolkit/services

Simple **service creation**.



## API toolkit – Creating Services for CICS and IMS

z/OS Connect EE



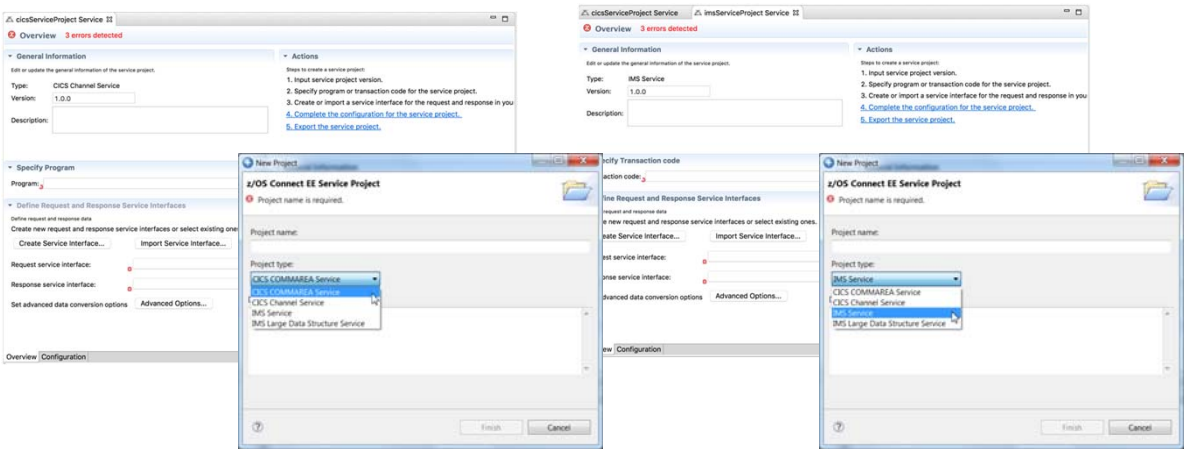

Use the **API toolkit** to create services through Eclipse-based tooling.

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

# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

## Service creation – a common interface



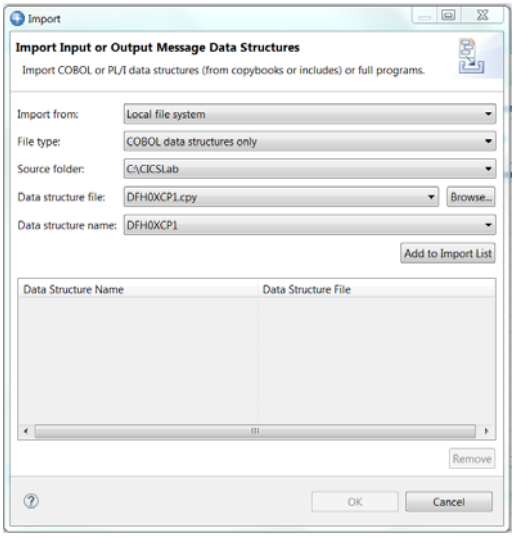

A common interface for service creation, agnostic of back end subsystem.



# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

## Creating a service project



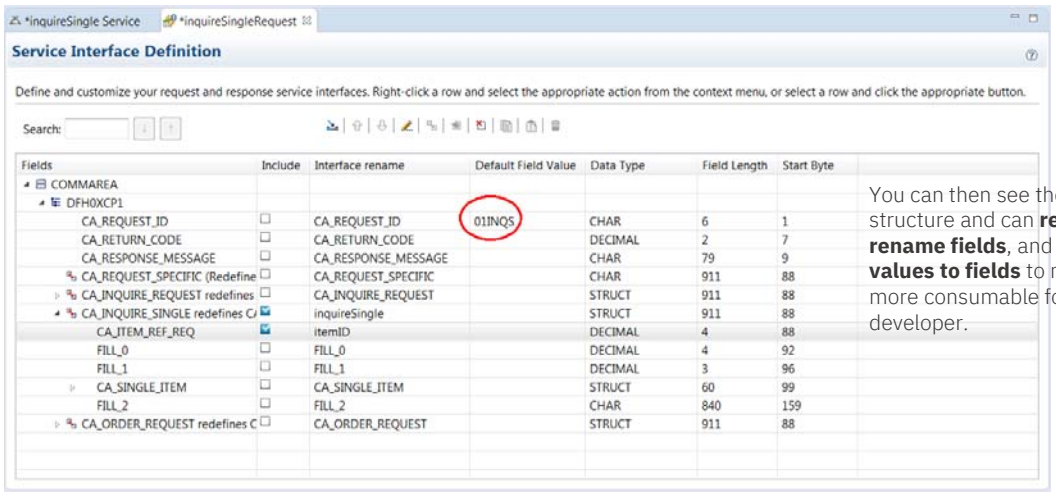

You start by importing data structures into the service interface from the local file system or the workspace.

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

# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

## Creating a service interface definition



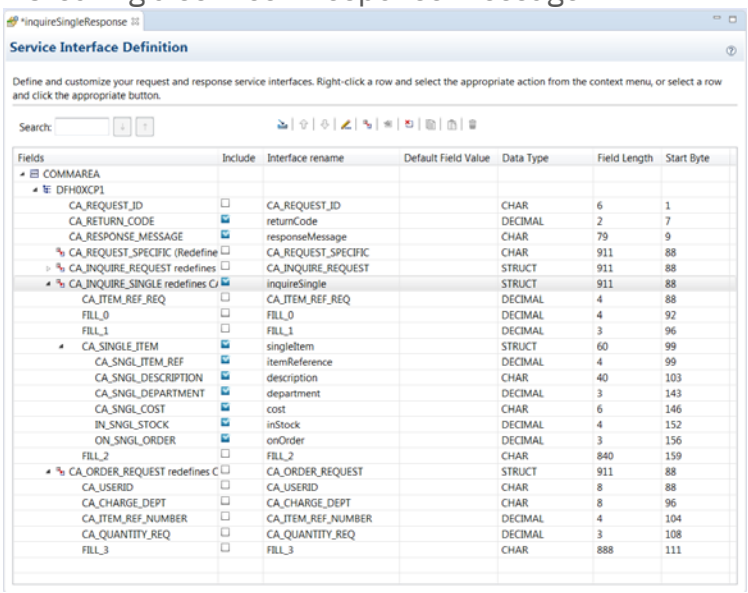

You can then see the imported data structure and can **redact fields**, **rename fields**, and **add default values to fields** to make the service more consumable for an API developer.



# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

## Creating a service – response message

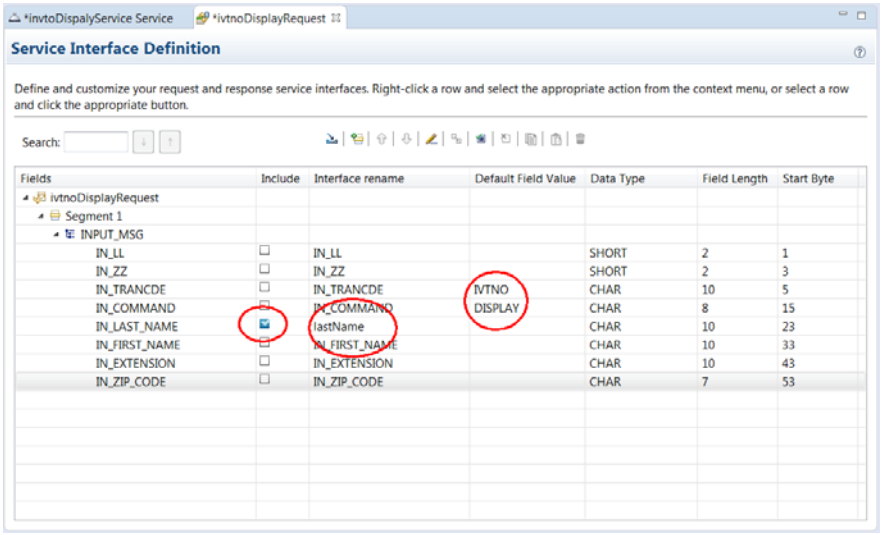



You can then see the imported data structure and can **redact fields** and **rename fields**

# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

## Creating a "GET" service interface request definition




The service developer creates distinct services for each function.

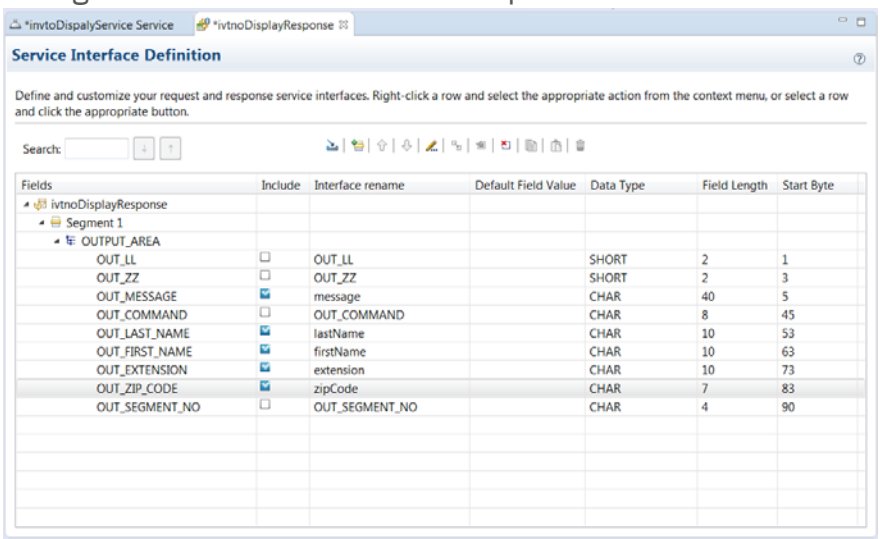
DISPLAY
DELETE
ADD
UPDATE



# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

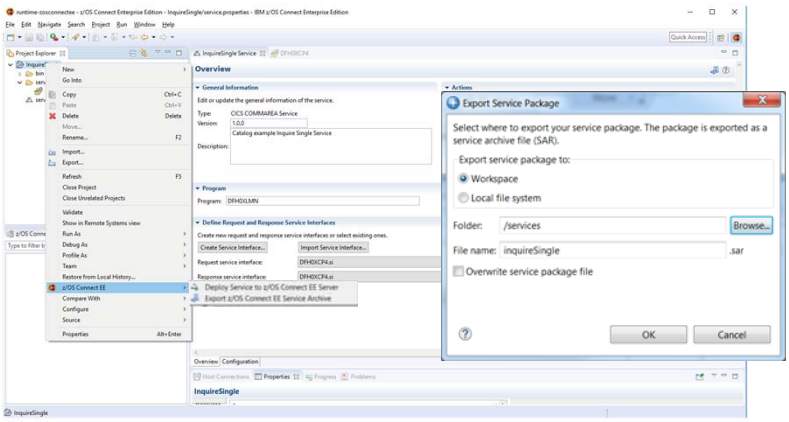
## Creating a "GET" service interface response definition



# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

## Creating a service for CICS and IMS



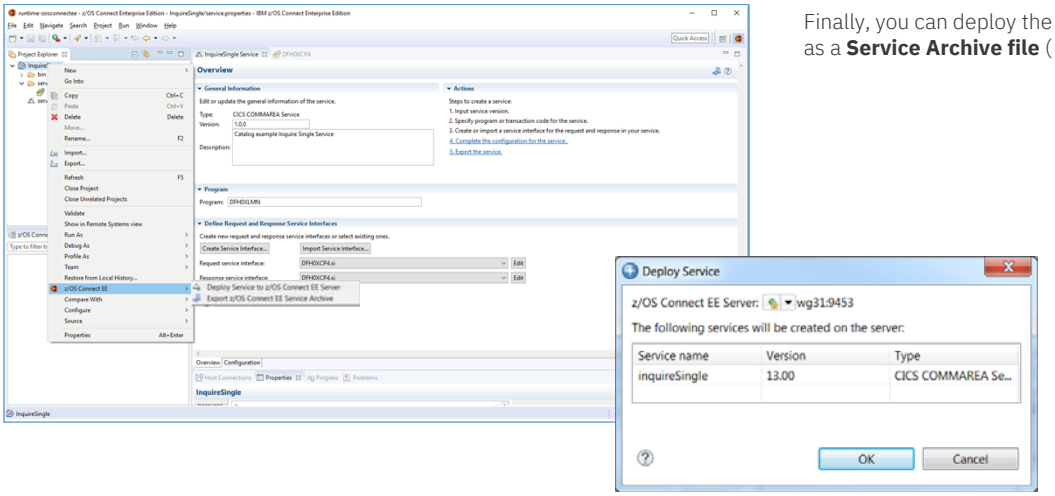

Finally, you can export the service project as a **Service Archive file** (`.sar`).



# API toolkit – Creating Services for CICS and IMS

z/OS Connect EE

## Creating a service for CICS and IMS

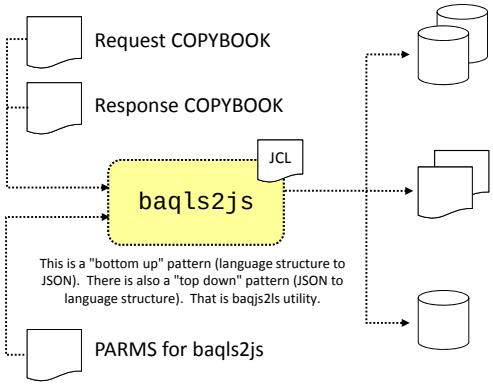



Finally, you can deploy the service project as a **Service Archive file** (`.sar`)

# Creating Services without the Toolkit - 1

z/OS Connect EE

For MQ and MVS Batch use the supplied conversion utility BAQLS2JS




### BIND Files

These are binary-format files that contain information about the field definitions and the data transformation requirements.

These are placed in a USS file system location based on input parms you specify.

### JSON Schema Files

These provide the JSON schema used to interact with the backend program based on the COPYBOOK data requirements.

These are placed in a USS file system location based on input parms you specify.

### Service Archive (SAR) File

This is a ZIP-format file that contains the JSON schema and some meta-data. This is input to the API Editor (next unit) to create the APIs.

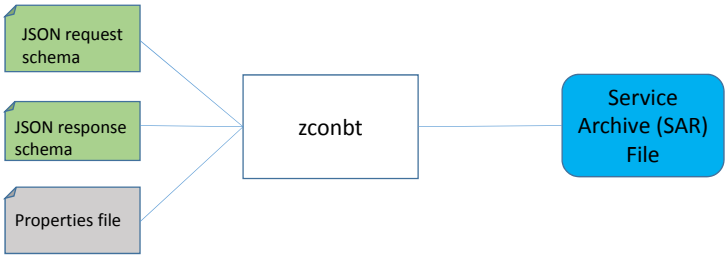
This is placed in a USS file system location based on input parms you specify.



# Creating Services without the Toolkit – 2a

z/OS Connect EE

For DB2 and HATS REST Services use the z/OS Connect Build toolkit (zconbt)

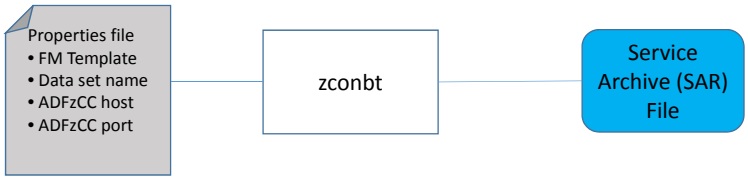



Generate the service archive file

# Creating Services without the Toolkit – 2b

z/OS Connect EE

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

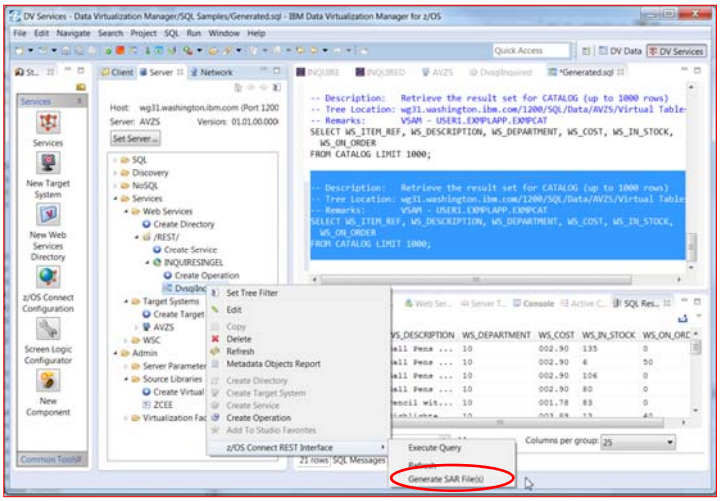



Generate the service archive file



## Creating Services without the Toolkit – Part 3

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*



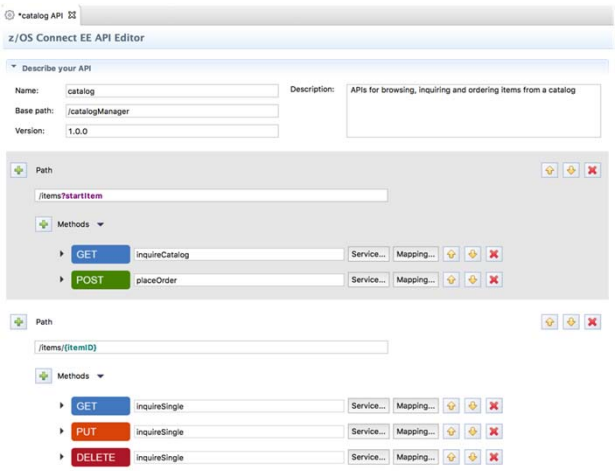
## /api_toolkit/api_editor

Quick and easy **API mapping.**

# API toolkit – API Editor

z/OS Connect EE

## API definition




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

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

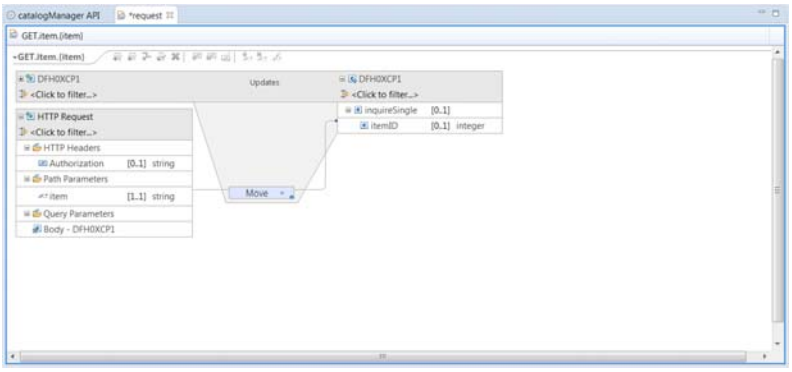
Your services are represented by `.sar` files, which you import into the **API toolkit**, regardless of how the .sar was generated.



# 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.

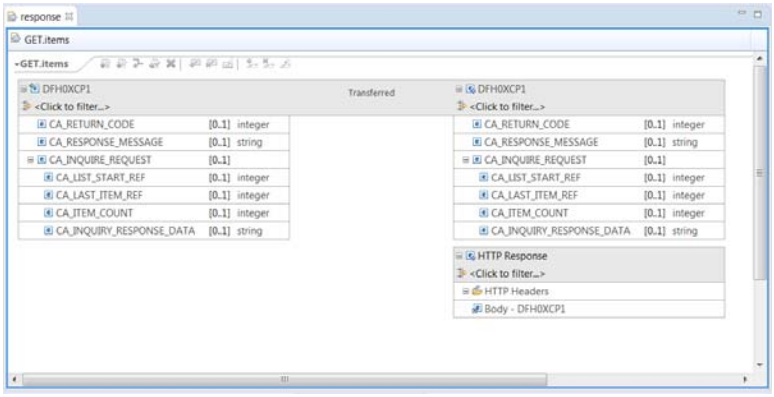
Assign static values to fields, useful for Op codes.

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




Allows the API Developer to remove fields to simplify the API

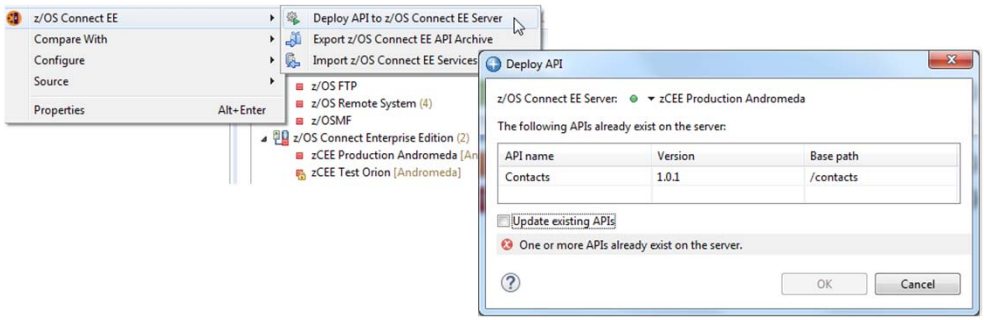


# API toolkit – API Editor

z/OS Connect EE

## Server connection and API deployment

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



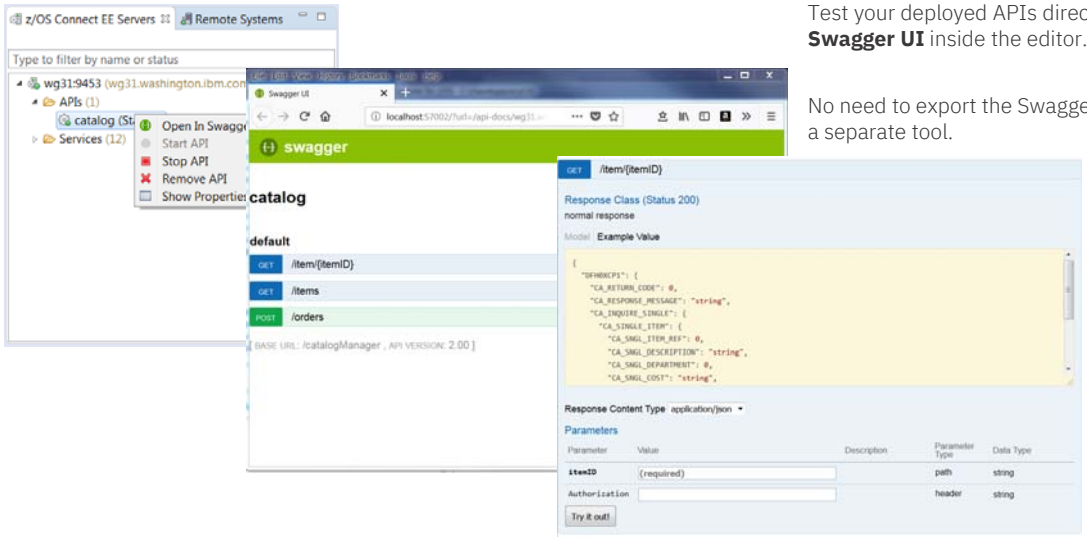

**Right-click deploy to server** enables developers to quickly deploy, test, and iterate on their APIs.

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

# API toolkit – API Editor

z/OS Connect EE

Testing with Swagger UI



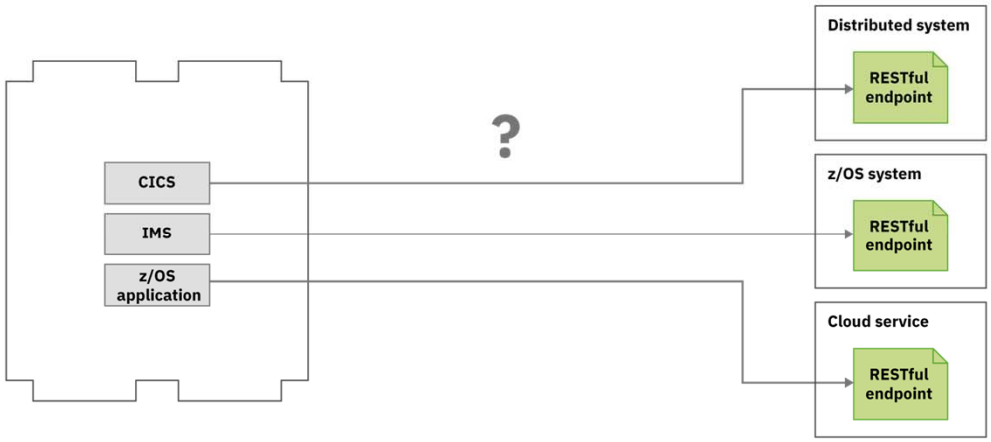

Test your deployed APIs directly with **Swagger UI** inside the editor.

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



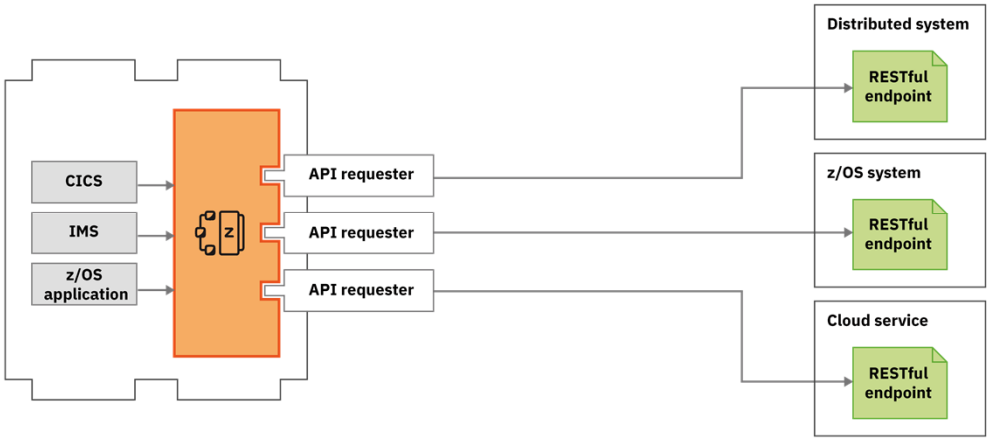
# What about calling external APIs from my z/OS assets?

z/OS Connect EE



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

z/OS Connect EE

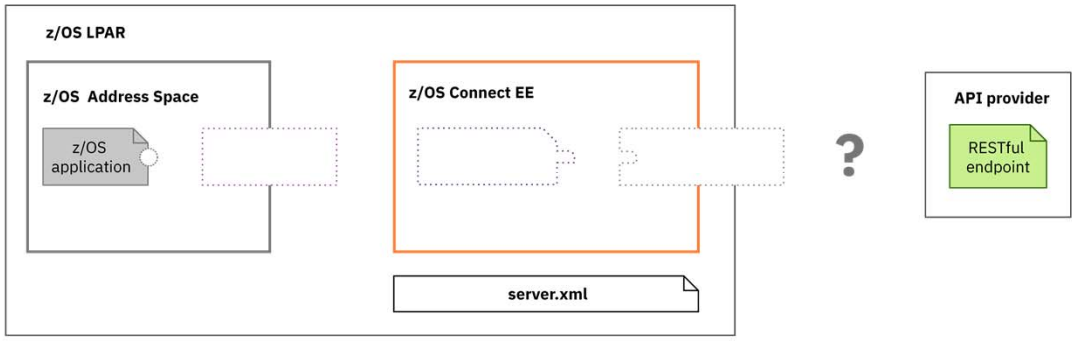





# Five steps to calling an external API

Starting point

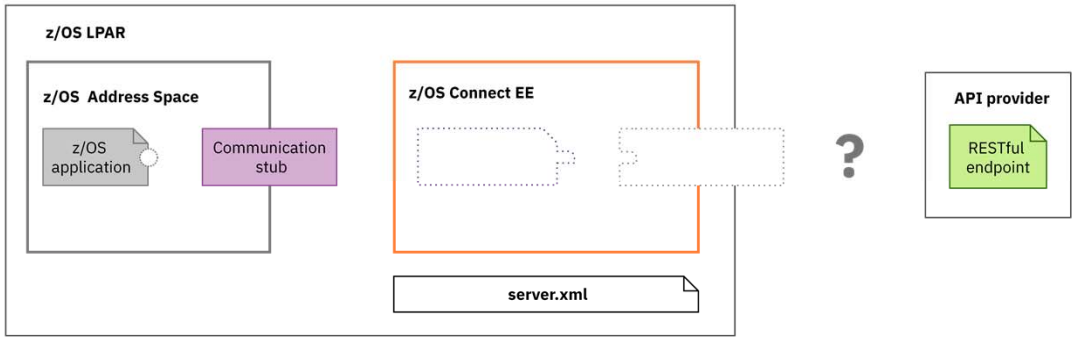
z/OS Connect EE



# Five steps to calling an external API

z/OS Connect EE

## Step 1. Configure communication stub




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

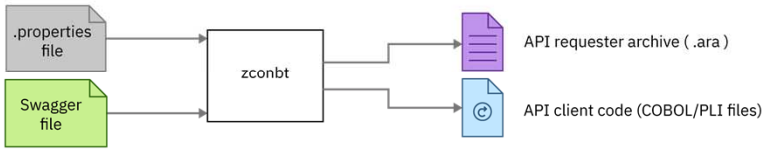
ibm.biz/zosconnect-configure-comms-stub



# Five steps to calling an external API

z/OS Connect EE

## Step 2. Generate API requester archive from Swagger



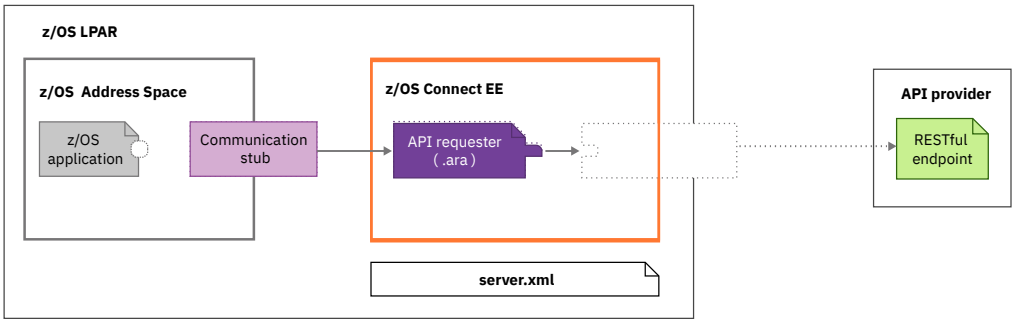

Generate your `.ara` file, and API client code.

ibm.biz/zosconnect-generate-ara

# Five steps to calling an external API

z/OS Connect EE

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



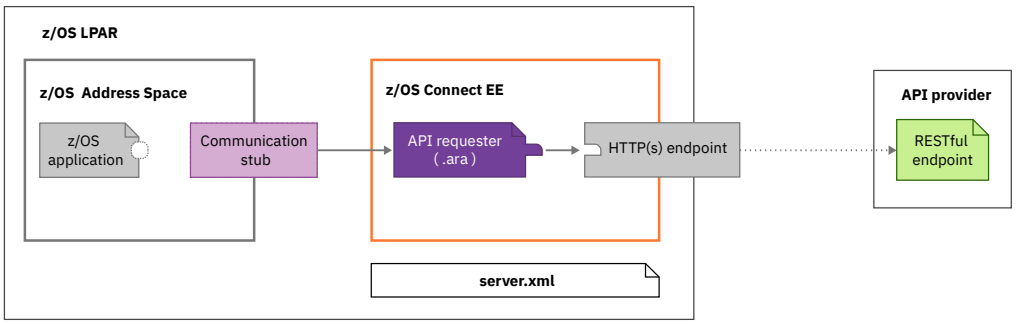

Deploy your API requester archive to the *apiRequester* directory.



# Five steps to calling an external API

z/OS Connect EE

## Step 4. Configure HTTP(S) endpoint



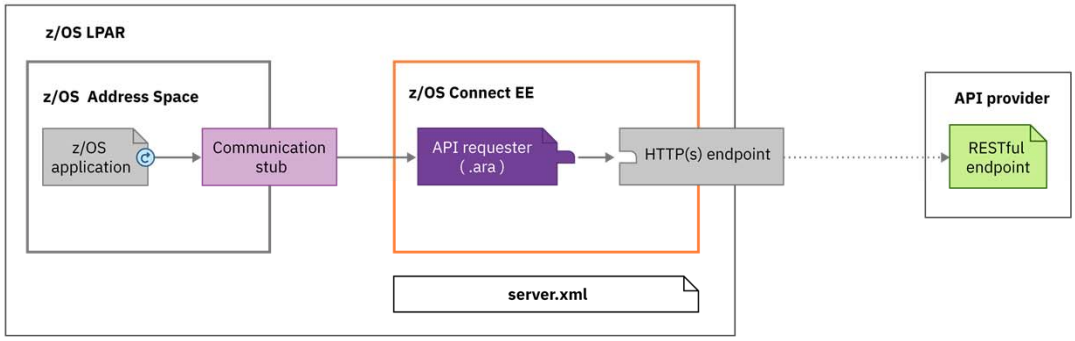

Configure the connection between z/OS Connect EE and the external API.

ibm.biz/zosconnect-configure-endpoint-connection

# Five 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



# Five steps to calling an external API

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

```
GETAPI
 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.

 * Copy API Requester required copybook
 COPY BAQRINFO.

 * Request and Response
 01 API-REQUEST.
     COPY CSC02Q01.
 01 API_RESPONSE.
     COPY CSC02P01.
 * Structure with the API in
 01 API-INFO-OPER1.
     COPY CSC02I01.

 * Request and Response segm
```

```
apis.xml
<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
           03 BAQ-APINAME              PIC
              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.
```

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

# Five 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
```



# Five steps to calling an external API

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

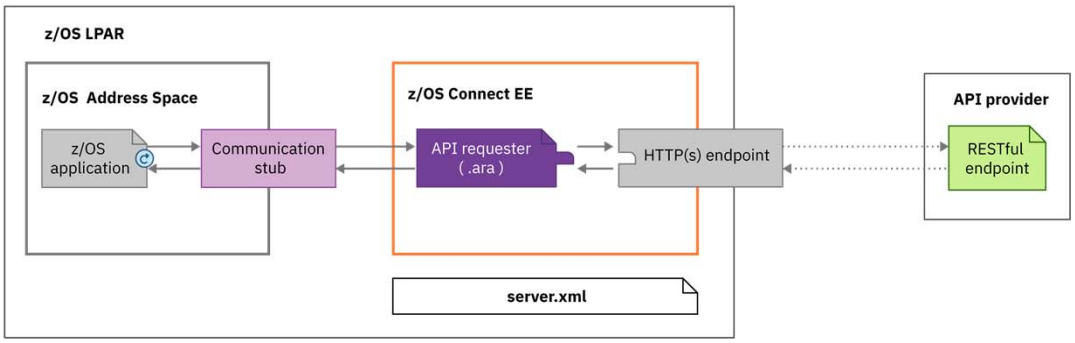
GETAPI

```
          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.
```

# Five steps to calling an external API

z/OS Connect EE

Done





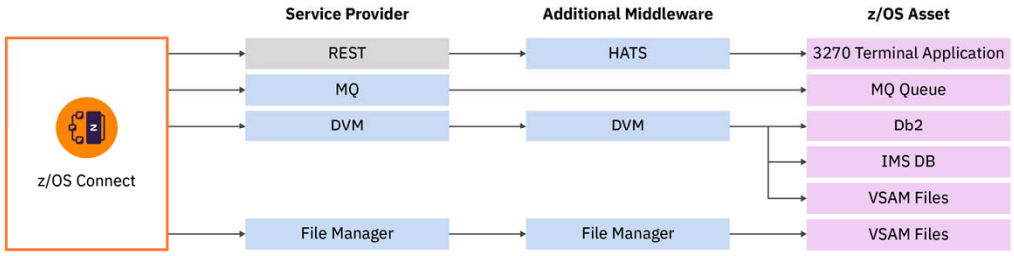

# /common_scenarios

Typical connection patterns to different subsystems.

# z/OS Connect EE 3rd party integrations

z/OS Connect EE

Additional value from the ecosystem




z/OS Connect EE is **pluggable** and **extensible** allowing 3rd Party Service Providers to expand the list of z/OS assets you can expose as APIs



# 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/<server_name>
├─ dropins
├─ logs
│  └ messages.log
├─ resources
│  ├─ security
│  └─ zosconnect
├ server.env
├ server.xml
├─ tranlog
└─ workarea
```

The /dropins directory is not used by z/OS Connect EE V3.0. It is a directory monitored by Liberty for applications to be dynamically loaded

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

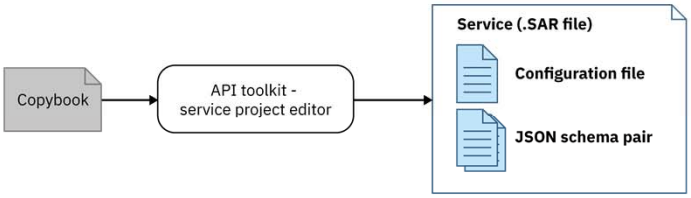
The /zosconnect directory is where we will place the deployed APIs, services, and API requester files

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.

# Connect to CICS

z/OS Connect EE

## Create the service definition




`.sar` file is created from a copybook using the Eclipse-based **API toolkit service project editor.**

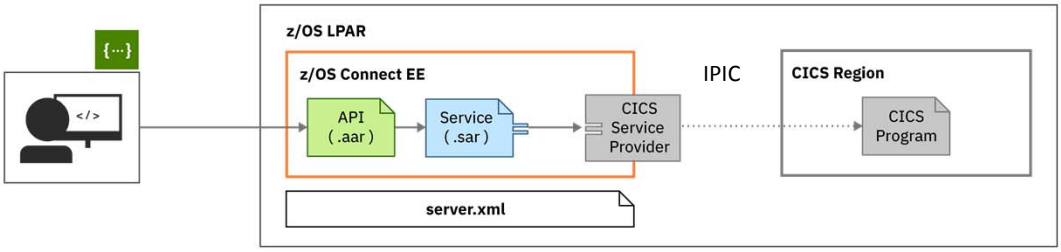
ibm.biz/zosconnect-sar-creation



# Connect to CICS

z/OS Connect EE

## Topology




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

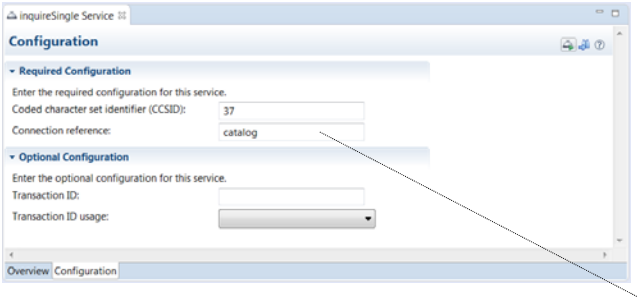
An IPIC connection must be configured in CICS.

ibm.biz/zosconnect-scenarios

# The server.xml File (CICS IPIC)

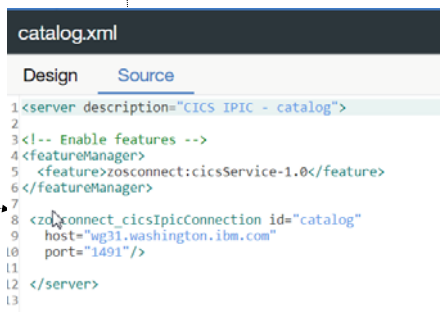
z/OS Connect EE

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



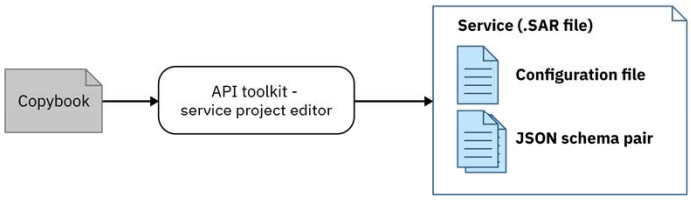



The IMS server.xml file ...

# Connect to IMS

z/OS Connect EE

## Create the service definition



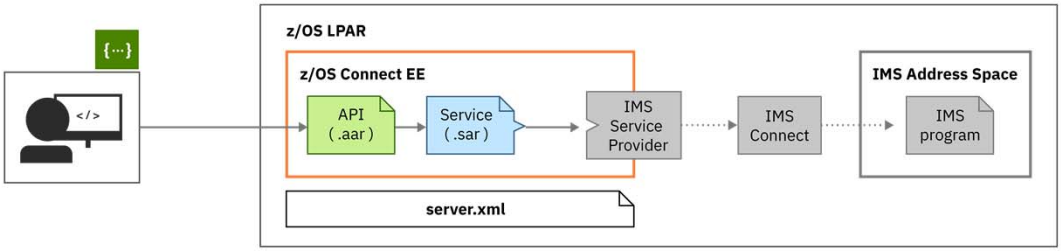

`.sar` file is created from a copybook using the Eclipse-based **API toolkit service project editor.**

ibm.biz/zosconnect-sar-creation

# Connect to IMS

z/OS Connect EE

## Topology




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

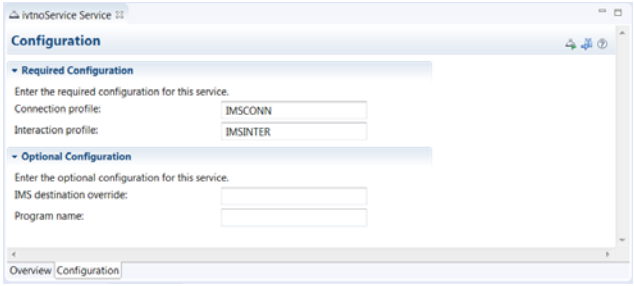
Use the **API toolkit** to configure the service.

ibm.biz/zosconnect-scenarios



# IMS Connections and Interactions

z/OS Connect EE




Connection

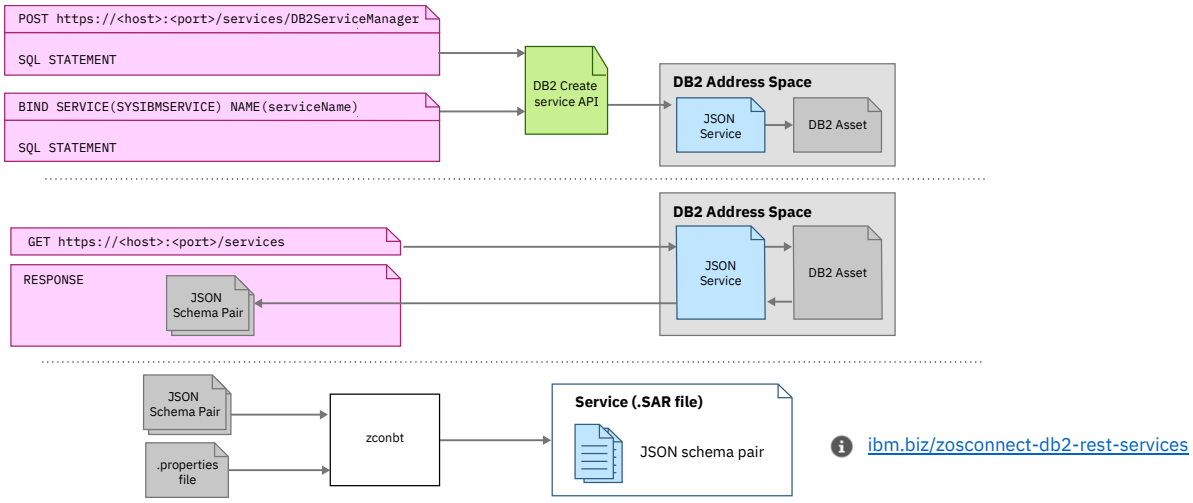
```
<server>
<imsmobile_imsConnection comment="" connectionFactoryRef="IVP1" connectionTimeout="-1" connectionType="IMSCONNECT" id="IMSCONN"/>
<connectionFactory containerAuthDataRef="Connection1_Auth" id="IVP1">
          <properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000"/>
</connectionFactory>

<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>
```

# Connect to Db2

## Create the service definition




ibm.biz/zosconnect-db2-rest-services

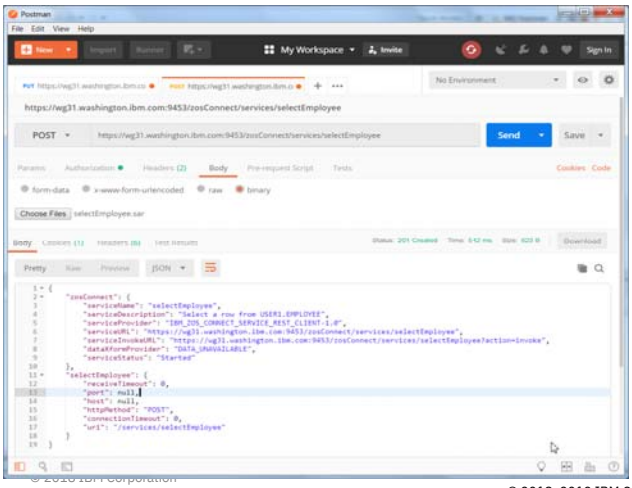
`.sar` file is created from the JSON schema of the DB2 service using the **zconbt utility**.



# Deploying Db2 Service Archive Options

z/OS Connect EE

- Use SAR as request message and use HTTP POST
- Use URI /zosConnect/services/{serviceName}
- Postman or curl://




**Command:**

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

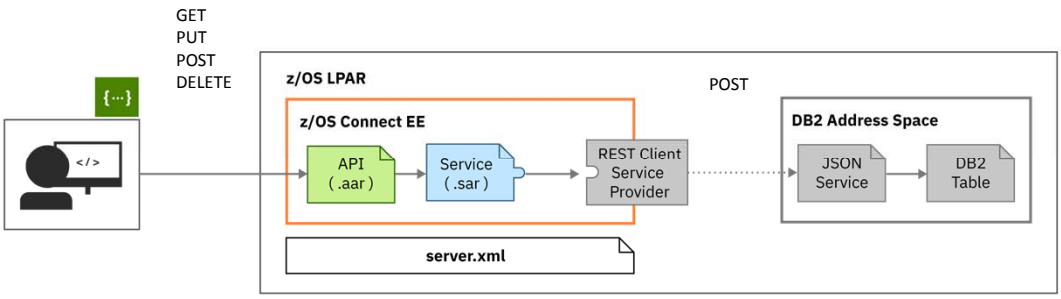
**Results:**

```
{"zosConnect":{"serviceName":"selectEmployee","serviceDescrip
tion":"Select a row from
USER1.EMPLOYEE","serviceProvider":"IBM_ZOS_CONNECT_SERVICE_RE
ST_CLIENT-1.0
","serviceURL":"https://mpxm:9453/zosConnect/services/selectE
mployee","serviceInvokeURL":"https://mpxm:9453/zosConnect/ser
vices/selectEmployee?action=invoke","dataXformProvider":"DATA
_UNAVAILABLE","serviceStatus":"Started"},"selectEmployee"
:{"receiveTimeout":0,"port":null,"host":null,"httpMethod":"PO
ST","connectionTimeout":0,"uri":"/services/selectEmployee"}}
```

# Connect to Db2

z/OS Connect EE

## Topology




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

A JSON 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:

db2pass.xml

Design | Source

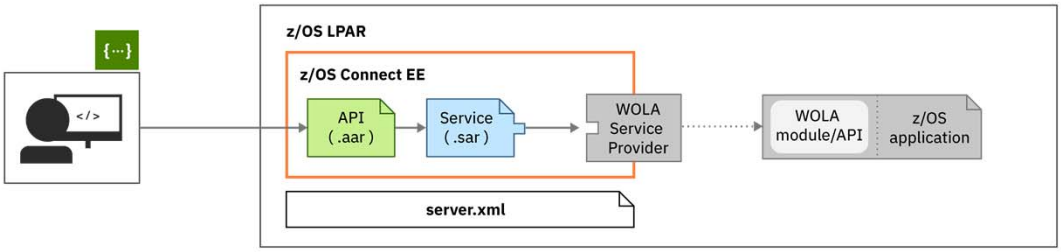
```
<server description="DB2 REST">

<zosconnect_zosConnectServiceRestClientConnection id="db2conn"
      host="wg31.washington.ibm.com"
      port="2446"
      basicAuthRef="dsn2Auth" />

<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
      applName="DSN2APPL"/>

</server>
```

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

# Connect to a WOLA-enabled z/OS application

z/OS Connect EE

## Topology




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

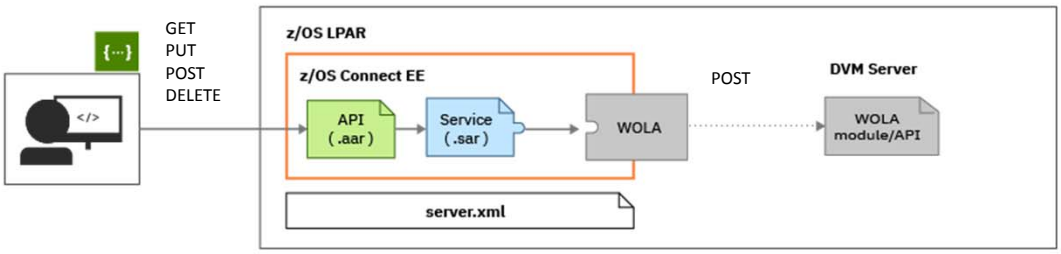
The z/OS application must be WOLA-enabled.



# Connect to DVM

z/OS Connect EE

## Topology




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

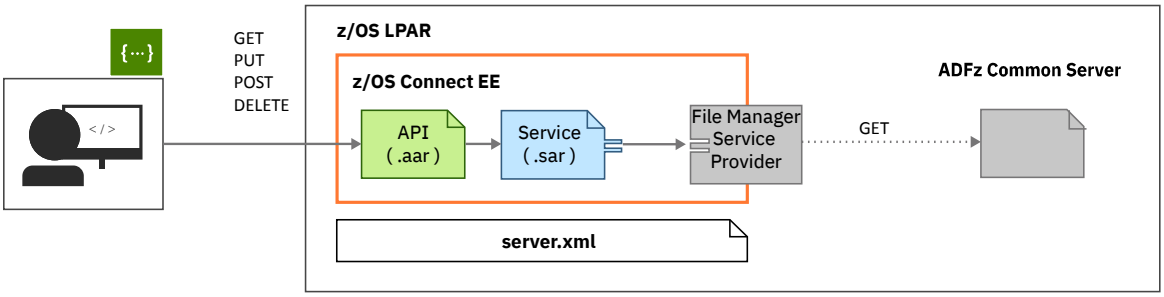
A REST service must be configured in the HATS.

ibm.biz/zosconect-db2-rest-services

# Connect to File Manager

z/OS Connect EE

## Topology




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

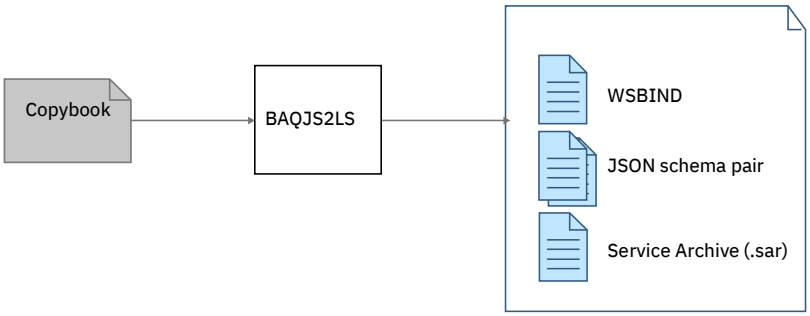
A File Manager Template is required .



# Connect to MQ

z/OS Connect EE

## Create the service definition



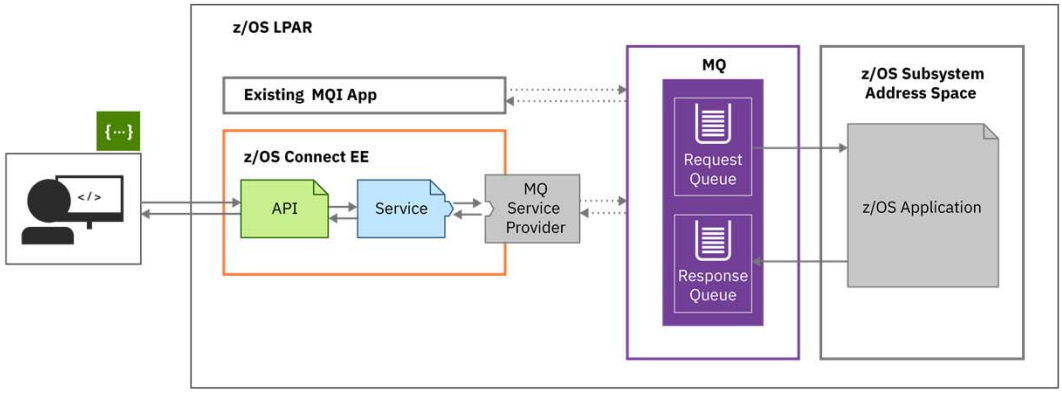

.sar is used to create the API using the API Editor in the Toolkit.

ibm.biz/zosconnect-sar-creation

# Connect to MQ

z/OS Connect EE

## Topology (Two-way service example)




You can also configure one-way services.

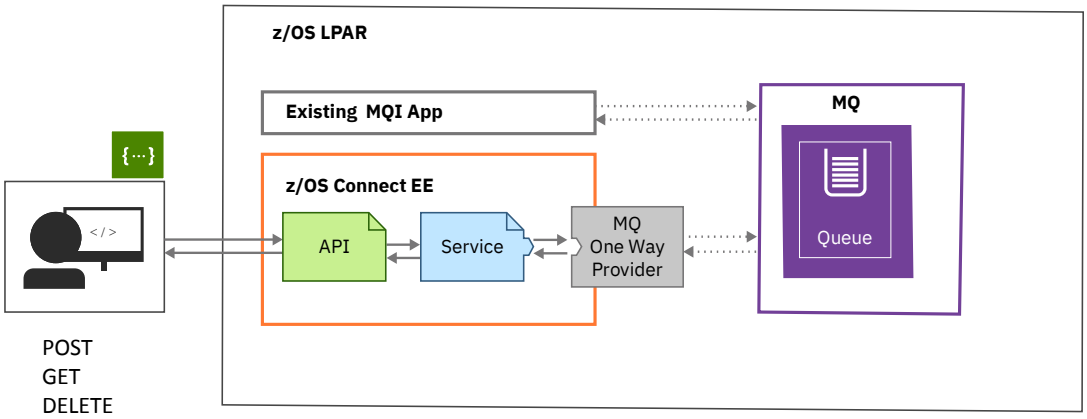
ibm.biz/zosconnect-mq-service-provider



# Connect to MQ

z/OS Connect EE

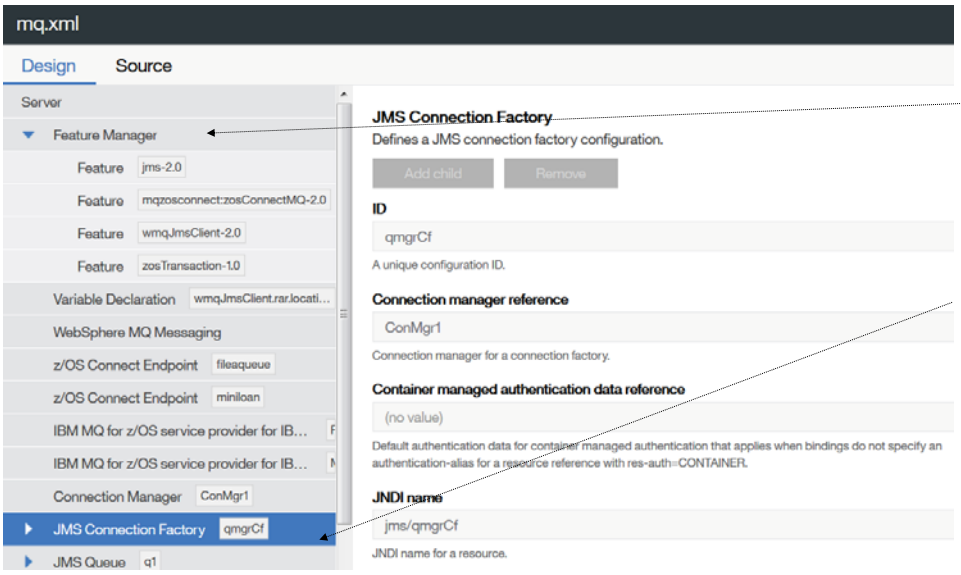
## Topology (One-way service example)




ibm.biz/zosconnect-mq-service-provider

# The server.xml File (MQ)

z/OS Connect EE




Features related to JMS Support

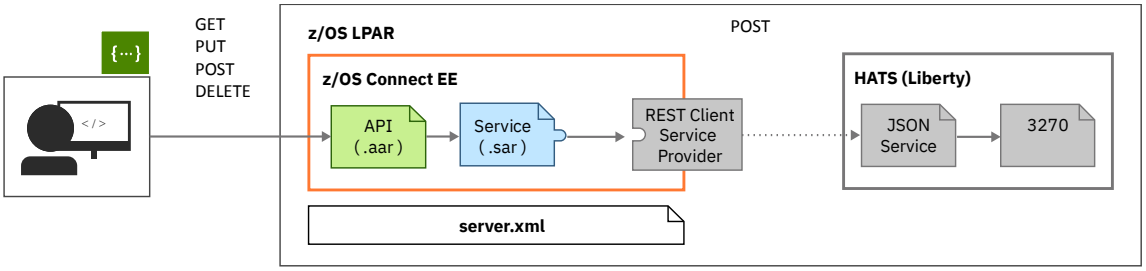
JMS Connection Factories,



# Connect to HATS

## Topology

z/OS Connect EE




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

A REST service must be configured in the HATS.

ibm.biz/zosconect-db2-rest-services

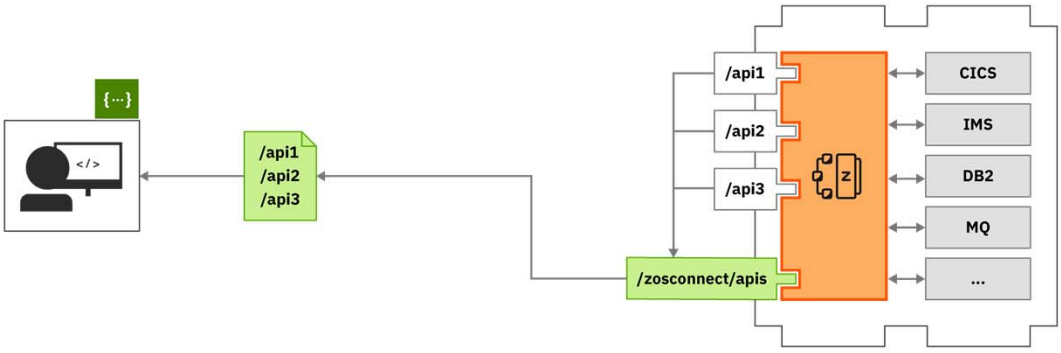
## /zosconnect/apidocs

Get the Swagger definitions for your APIs



## API Documentation

z/OS Connect EE




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

# /miscellaneousTopics

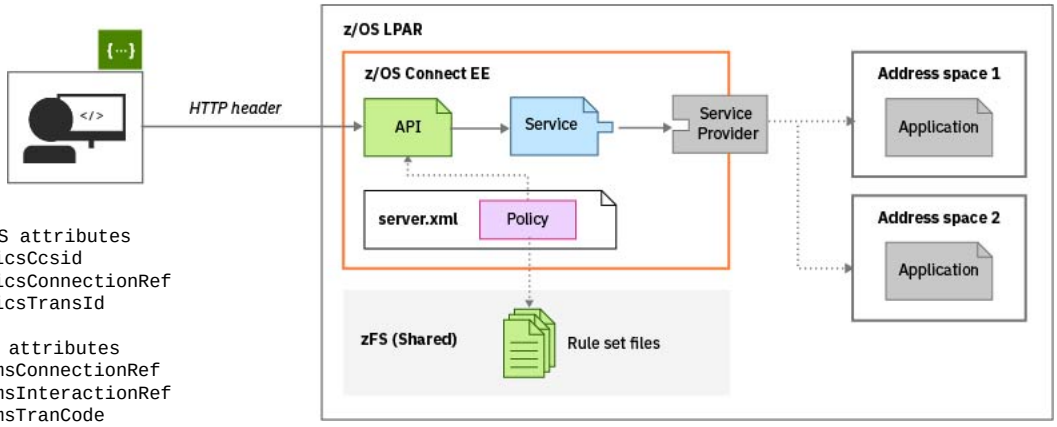
performance, high availability, Liberty



# 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)



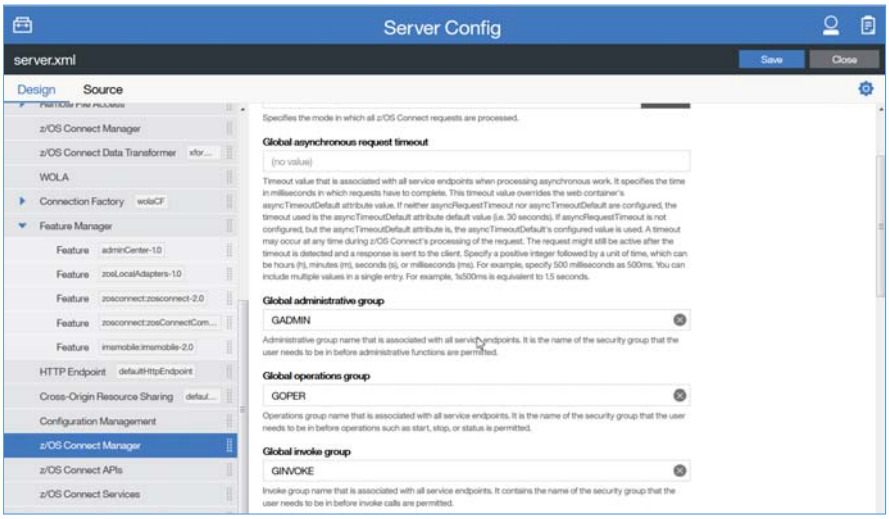

CICS attributes
- cicsCcsid
- cicsConnectionRef
- cicsTransId

IMS attributes
- imsConnectionRef
- imsInteractionRef
- imsTranCode

# Liberty's "adminCenter" Feature

z/OS Connect EE

Web browser interface to the server's configuration files






# 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 |

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

PUT /zosConnect/apis/{apiName}?status=started|stopped
GET /zosConnect/apis
GET /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

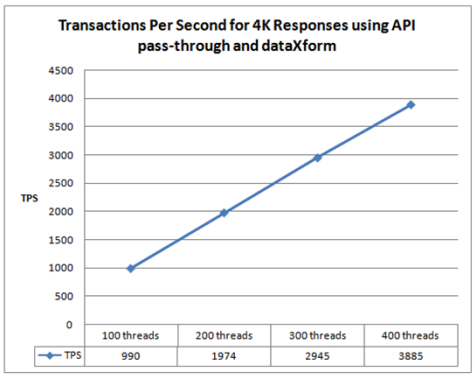
| Method | Administrative Task |
|---|---|
| GET | Get a list of API Requesters |
| | Get the details of an API Requester |
| POST | Deploy an API Requester |
| PUT | Update an API Requester |
| | Change the status of an API Requester |
| DELETE | Delete aa API Requester |

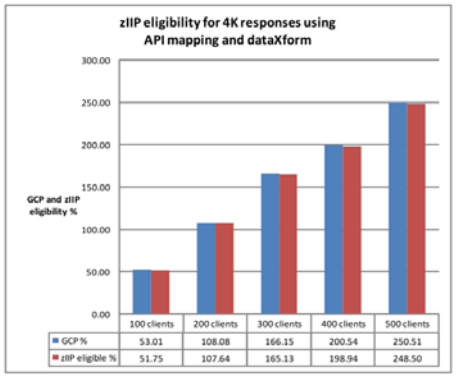
PUT /zosConnect/apiRequesters/{apiRequesterName}?status=started|stopped
GET /zosConnect/apiRequesters
GET /zosConnect/apiRequesters/{apRequesterName}

# Performance

z/OS Connect EE

High Speed, High Throughput, Low Cost



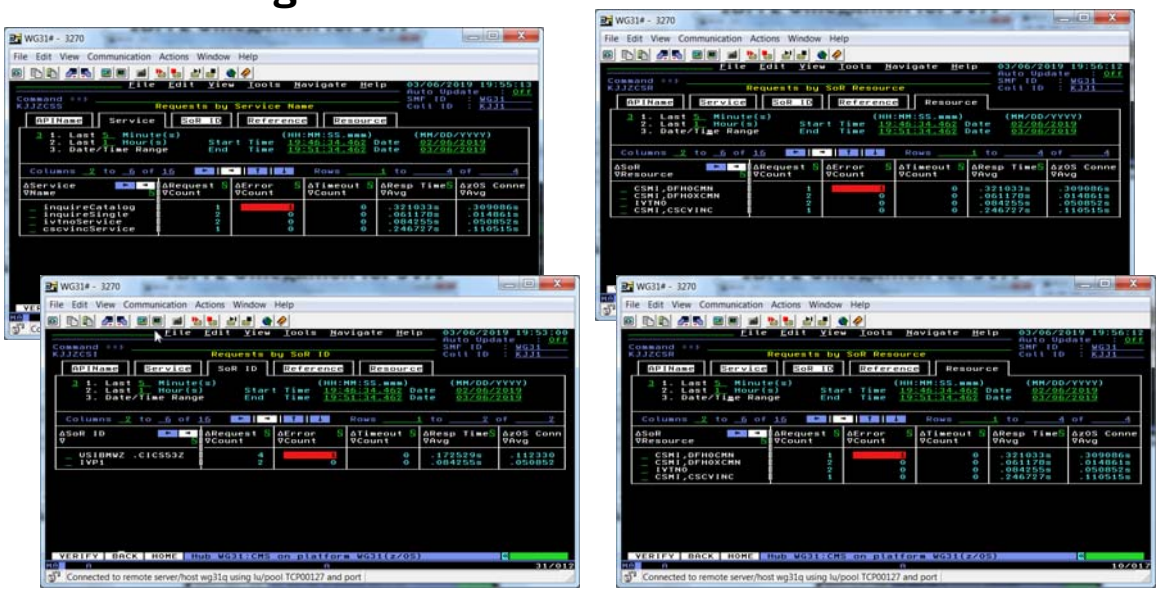

z/OS Connect EE is a Java-based product: Over **99%** of its MIPs are **eligible for ZIIP offload**.

ibm.biz/zosconnect-performance-report



# IBM z Omegamon for JVM

z/OS Connect EE

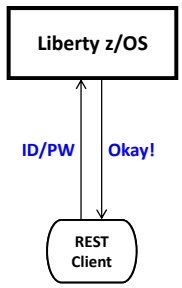


# Authentication

z/OS Connect EE

**Several different ways this can be accomplished:**

### Basic Authentication

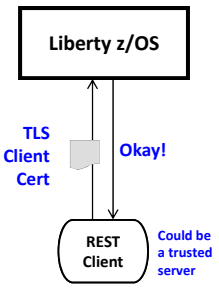



**Server prompts for ID/PW**

**Client supplies ID/PW**

**Server checks registry:**
- **Basic (server.xml)**
- **LDAP**
- **SAF**

### Client Certificate



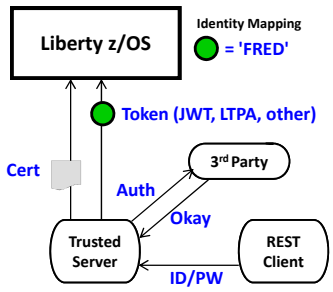

**Server prompts for cert.**

**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**

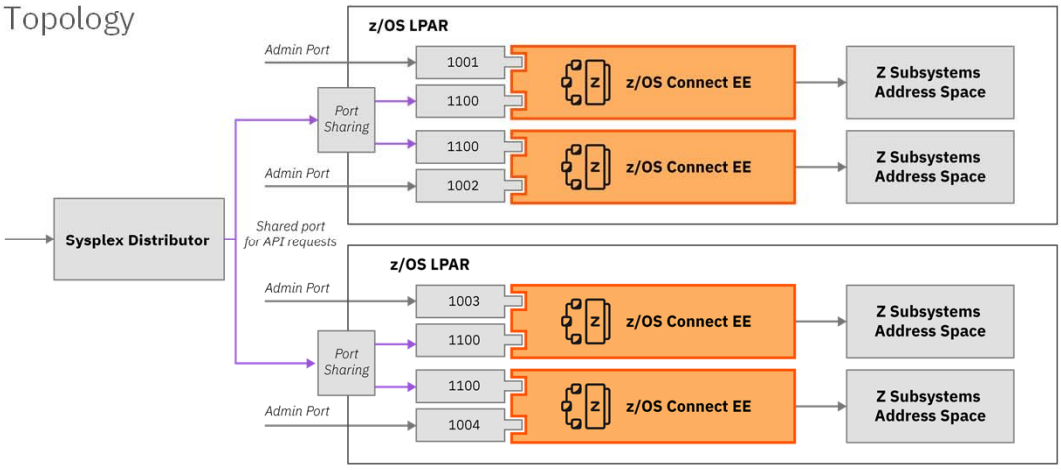
**Registry options:**
- **LDAP**
- **SAF**



# High Availability

z/OS Connect EE

Topology



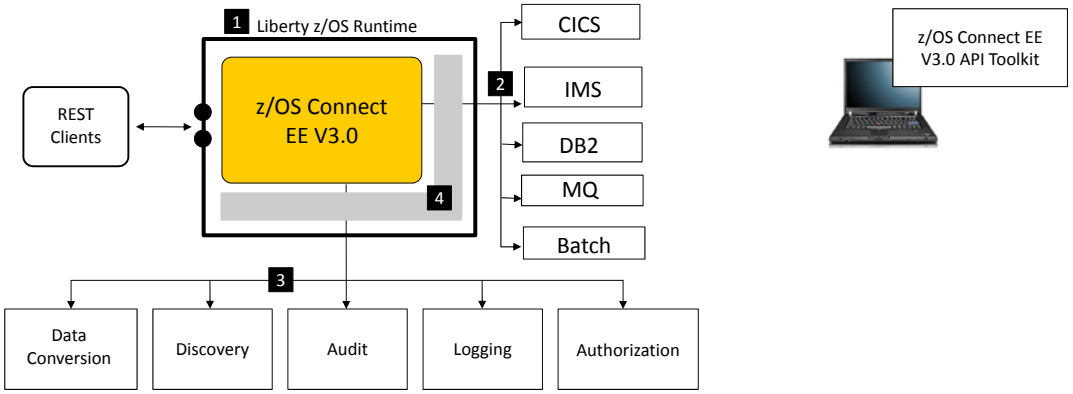

ibm.biz/zosconnect-ha-concepts

ibm.biz/zosconnect-scenarios

# What Does Liberty and z/OS Connect Provide?

z/OS Connect EE

A *framework* to secure and manage access to z/OS resources using REST:




1. Liberty is provided as a runtime. This is significant because the capabilities of Liberty are available to z/OS Connect EE V3.0 which runs inside, notably: security
2. Backend connectivity is provided with "service provider" code. The connectivity mechanism is a function of the backend system: CICS=IPIC; IMS=TCP;DB2=RESP; MQ=JMS;BATCH=WOLA
3. These are "interceptors" and provide function that is called for each request that arrives.
4. Both the "service provider" and "interceptor" interfaces are extensible, allowing you to write your own, or for third party vendors to write code and use z/OS Connect EE V3.0



## /questions?thanks=true

Thank you for listening.

# /exercises

basic security, exercise paths



## Exercises – Two paths or options

z/OS Connect EE

- ❑ Basic Configuration Hands-on Lab
  - ❑ Configure a z/OS Connect Server
  - ❑ Develop and deploy a Service
  - ❑ Develop and deploy an API
  - ❑ Test using Swagger UI
  - ❑ Enable Security (SAF and SSL)

Or one or more of the following:

- ❑ Developing APIs Hands-on Labs
  - ❑ CICS Container/COMMAREA
  - ❑ DB2
  - ❑ IMS Transaction
  - ❑ MQ
  - ❑ MVS Batch
  - ❑ HATS
  - ❑ IBM DVM
  - ❑ Outbound RESTful applications

- Material can be downloaded from:

https://github.com/ibm-wsc/zCONNEE-Wildfire-Workshop

- Copy/Paste files on desktop
  - ➢ Basic Configuration CopyPaste
  - ➢ Developing APIs CopyPaste

- Identities:
  - ➢ RACF: USER1/USER1
  - ➢ zCEE: Fred/fredpwd

- 3270 Key Sequences
  - ➢ Clear screen: Fn-P
  - ➢ Enter key: right CTRL