Accessing z/OS resources with REST APIs using IBM z/OS Connect

Mitch Johnson mitchj@us.ibm.com
John Brefach John.J.Brefach@ibm.com
Washington System Center

Agenda

- An Introduction to REST API

- Enabling RESTful API to z/OS resources,
  - CICS programs
  - Db2 tables and stored procedures
  - IMS transactions
  - IMS data bases
  - MQ queues

- An overview of z/OS Connect API Provider Security

- Miscellaneous Topics

mitchj@us.ibm.com

Notes and Disclaimers

- **IBM z/OS Connect (OpenAPI 2)** refers to the types of APIs supported by the z/OS Connect product prior to service level V3.0.55. **IBM z/OS Connect (OpenAPI 3)** refers to support for an additional type of API as well as additional functions and features added at service level V3.0.55. Important - servers configured for OpenAPI 2 can will continue to operate as is with service level V3.0.55 and later.

- A z/OS Connect OpenAPI 2, or a z/OS Connect OpenAPI 3 icon will appear on slides where the information is specific to these products. Don't hesitate to ask questions as to why the icon does or does not appear on certain slides

- A Liberty icon will appear on slides where the information both products. Don't hesitate to ask questions as to why the icon does or does not appear on certain slides.

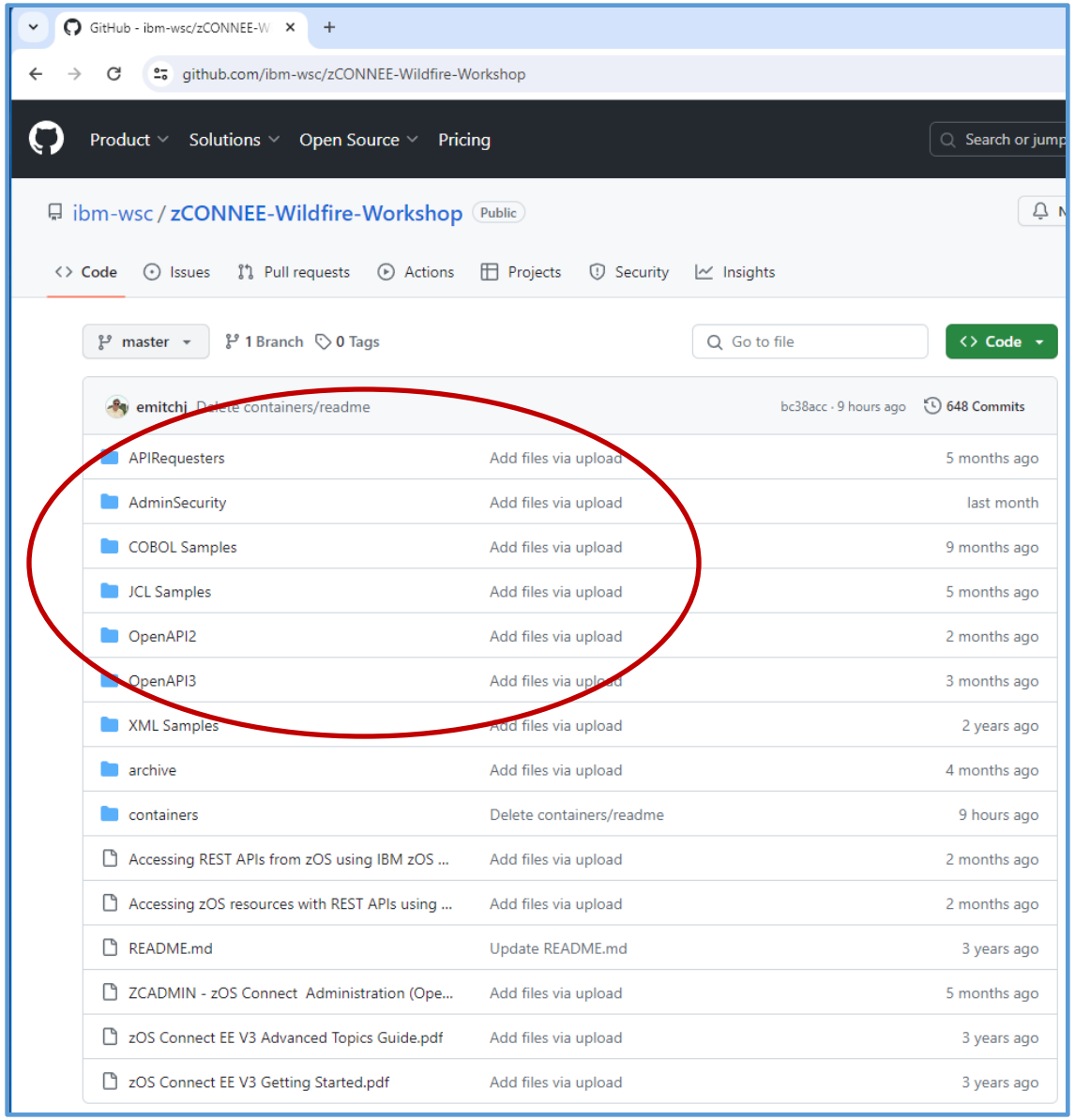
- There will be additional information on slides that will be designated as Tech/Tips. These contain information that hopefully will be useful to the reader.

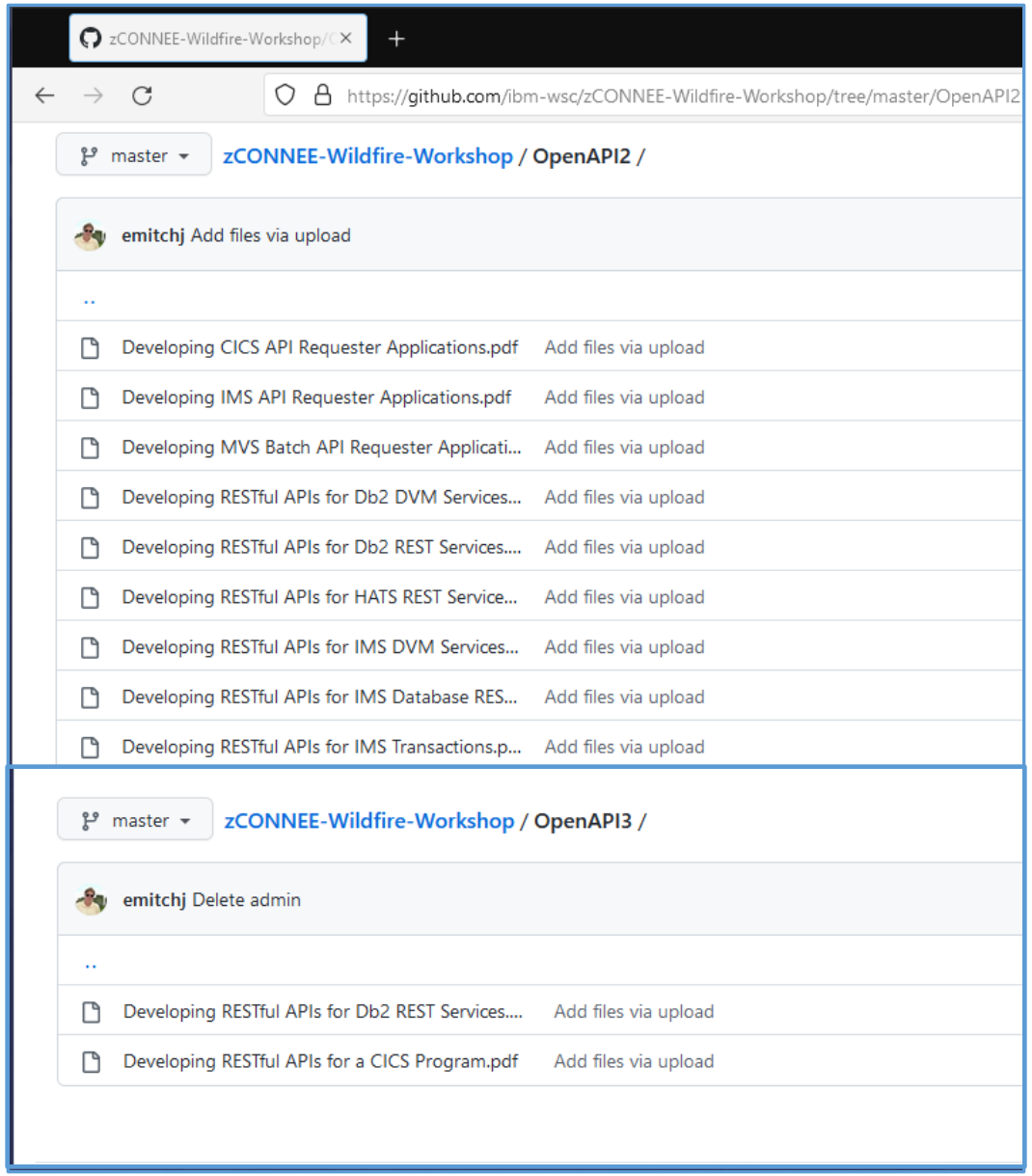
mitchj@us.ibm.com

This session is part of a series of Liberty and z/OS Connect related workshops. . .

z/OS Connect Wildfire Github Site

https://ibm.biz/zCeeWorkshopMaterial

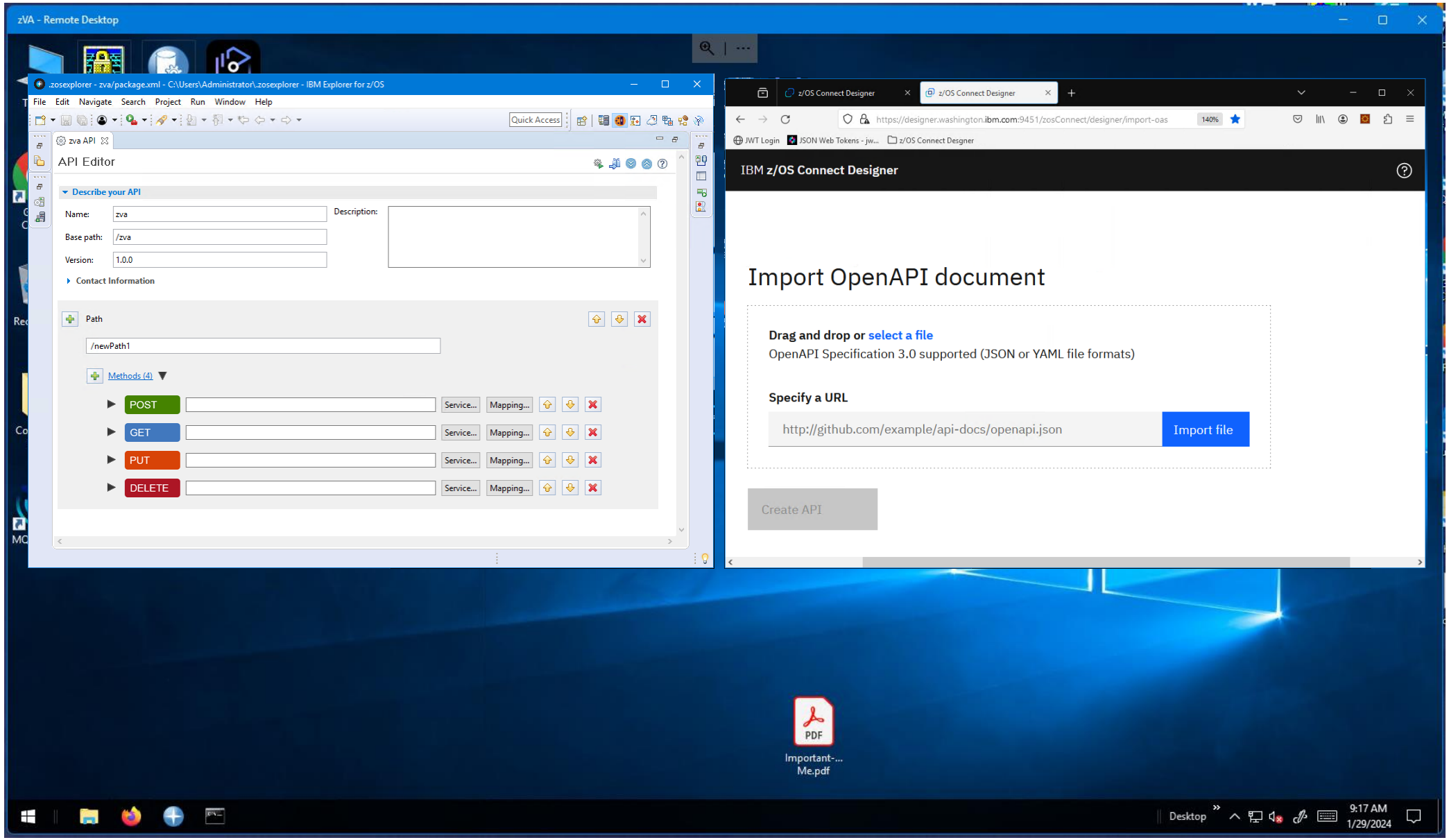




mitchj@us.ibm.com

Contact your local IBM representative to request hands-on access to these exercises

IBM's Z Virtual Access (zVA) provides access to hand-on exercises



mitchj@us.ibm.com

Contact your local IBM representative to request hands-on access to these exercises

z/OS Connect API provider support exposes z/OS resources to clients in the “cloud” via RESTful APIs

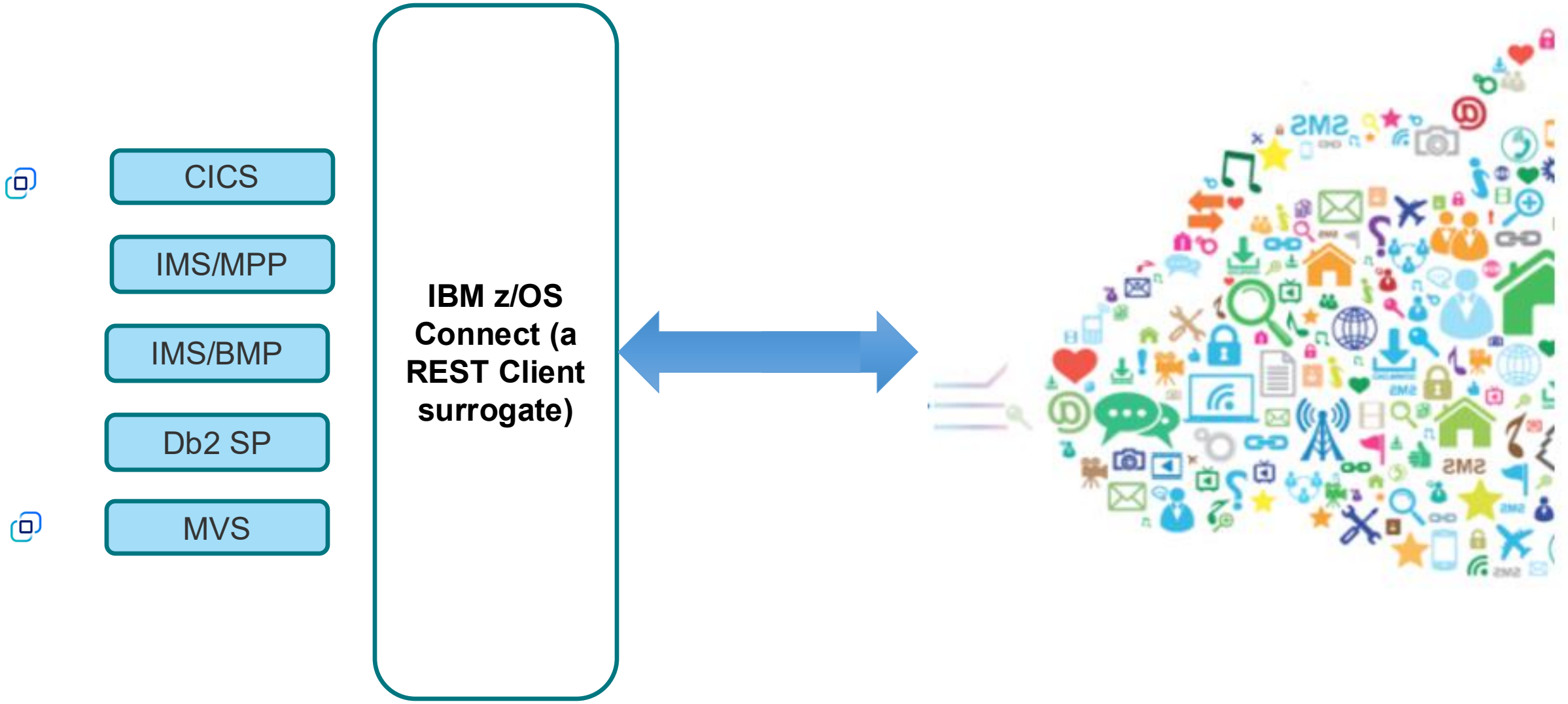
IBM z/OS Connect (an API Gateway)

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

+ HCL and Rocket Software
*Other Vendors or your own implementation

mitchj@us.ibm.com

Note: z/OS Connect API requester exposes external REST APIs in the “cloud” to z/OS applications



mitchj@us.ibm.com

Let’s start by defining REST and REST APIs

And what makes an API “RESTful”?

mitchj@us.ibm.com

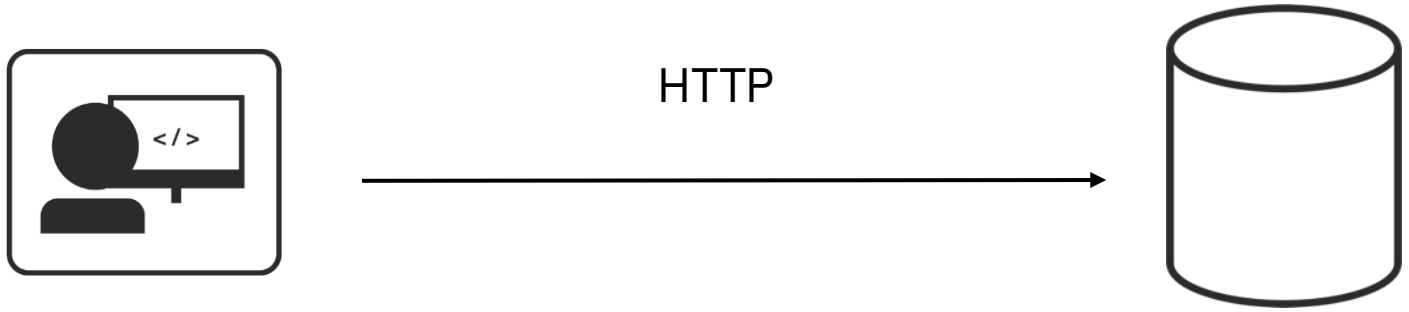
REST is architectural programming style

REST is an acronym standing for **Re**presentational **S**tate **T**ransfer*

An architectural programming style for **accessing** and **updating** data over the internet.

Typically using HTTP… but not all HTTP interfaces are “RESTful”.

Providing a simple and intuitive programming style 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.

mitchj@us.ibm.com

All REST APIs follow some basic key principles

Uniform Resource Identifier (URI) path identifies the resource (s)

They use HTTP verbs to specify Create, Read, Update, Delete (CRUD) operations, for more information on CURD, see URL https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

POST
GET
PUT
DELETE

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

Use Path and Query parameters to refine the request

Uniform Resource Locator (URL) identifies the protocol, host and port and includes the URI path

The use JSON (Java Script Object Notation) to represent the data object in Request/Response message bodies

```
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” }
```

mitchj@us.ibm.com

What makes a REST API a RESTful API?

REST is an architectural style of development having these basic principles plus..

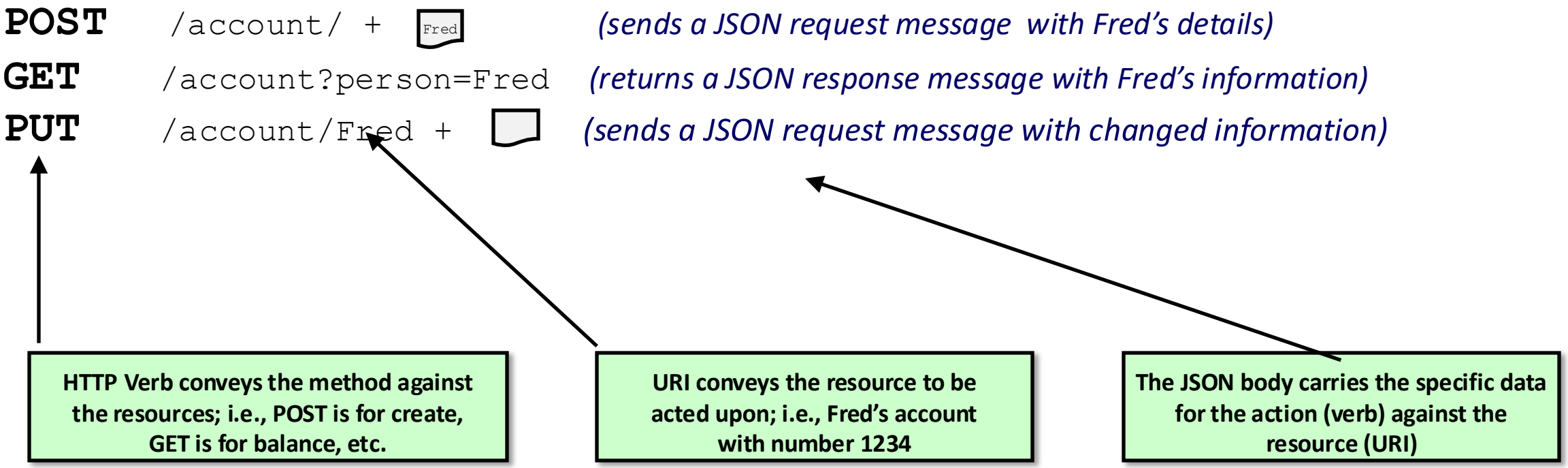
- It should be stateless (transaction management should be managed by the client)
- It should access and/or identify all server resources using only a **single** URI
- When performing the CRUD operations, the HTTP methods, GET, POST, PUT, DELETE should be used **as intended**
- It should return the result only in the form of **consistent and simple** JSON

When a REST API follows these basic principles, it is considered a RESTful API, whereas a REST API only follows some but not all the above principles

- The key is consistency, RESTful APIs follow these basic principles, REST APIs deviate from one or more of these principles

mitchj@us.ibm.com

RESTful Examples



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

mitchj@us.ibm.com

Let’s see one reason why,

Other solutions required that client code be tightly coupled with the resource since the resource's client code had to be integrated into the client application.

CICS client

*cscinq.java

```
COMMAREABUFFER commarea = new COMMAREABUFFER();
commarea.setNumb("111111");

//CICSConnection connection = new CICSConnection();
com.ibm.connector2.cics.ECIInteractionSpec interSpec = new com.ibm.connector2.cics.ECIInteractionSpec();

//       Create an interaction spec
interSpec.setFunctionName("CSCVINQ");
interSpec.setCommareaLength(commarea.getSize());

//       Create a Connection Factory (non-Managed)
com.ibm.connector2.cics.ECIManagedConnectionFactory mcf = new com.ibm.connector2.cics.ECIManagedConnectionFactory();
mcf.setConnectionURL("tcp://ctfmvs09.rtp.raleigh.ibm.com");
//mcf.setConnectionURL("local:");
mcf.setServerName("NQA11C00");
mcf.setPortNumber("2006");

try {
    //       Create an ECI connection factory using the nonMCF
    com.ibm.connector2.cics.ECIConnectionFactory cf =
        (com.ibm.connector2.cics.ECIConnectionFactory)mcf.createConnectionFactory();

    //       Create a connection spec
    com.ibm.connector2.cics.ECIConnectionSpec connectionSpec = new com.ibm.connector2.cics.ECIConnectionSpec();
    connectionSpec.setUserName("userid");
    connectionSpec.setPassword("p0ssword");

    //       Obtain a connection from the connection factory
    com.ibm.connector2.cics.ECIConnection connection =
        (com.ibm.connector2.cics.ECIConnection)cf.getConnection(connectionSpec);

    //       Create a interaction for the connection
    com.ibm.connector2.cics.ECIInteraction interaction =
        (com.ibm.connector2.cics.ECIInteraction) connection.createInteraction();

    //       Execute the interaction
    interaction.execute(interSpec, commarea, commarea);
```

MQ client

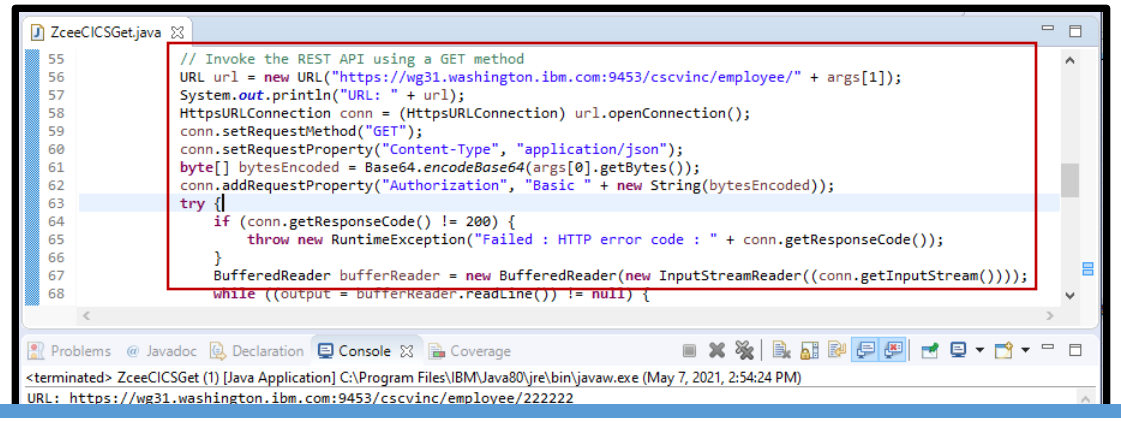
PutMessage.java

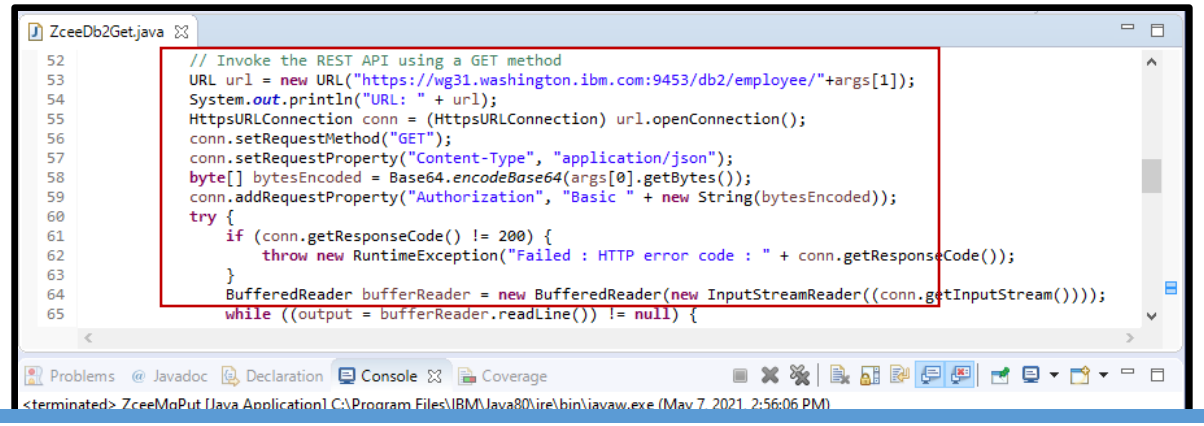
```
        // Create a connection to the queue manager
        if (bindingType.equals("client")) {
            // Set MQ Environment variables/parameters
            // qManager = "QMZB";
            if (qManager.equals("QMZB")) {
                com.ibm.mq.MQEnvironment.hostname = "mpx3";
                com.ibm.mq.MQEnvironment.channel = "Client.to.QMZB";
                com.ibm.mq.MQEnvironment.port = 1416;
            }
            if (qManager.equals("WMQ8")) {
                com.ibm.mq.MQEnvironment.hostname = "localzos";
                com.ibm.mq.MQEnvironment.port = 1436;
            }
        }
        com.ibm.mq.MQEnvironment.userID = userName;
        com.ibm.mq.MQEnvironment.password = passWord;

        qMgr = new MQQueueManager(qManager);
        System.out.println("Success:  Obtained a " + bindingType + " reference to the Queue manager: "+ qManager);

        // Set up the options on the queue we wish to open...
        // Note. All WebSphere MQ Options are prefixed with MQC in Java.

        int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

        // Now specify the queue that we wish to open,
        // and the open options...

        system_default_local_queue = qMgr.accessQueue(queueName,openOptions);
        System.out.println("Success:  Obtained a reference to the Queue: "  + queueName);

        sendMessage();
    }
    catch (MQException ex)
     {
       System.out.println("A WebSphere MQ error occurred : Completion code " +
                          ex.completionCode + " Reason code " + ex.reasonCode);
       if (ex.reasonCode == EMPTY_QUEUE)
        {
```

The CICS client uses CICS provided Java classes (com.ibm.connector2.cics.*) and the MQ client uses MQ provided Java classes (com.ibm.mq.*)

mitchj@us.ibm.com

A key goal for REST is to have client code that is unaware of the resource's infrastructure





These Java clients have the same application programming pattern.

- They provide an URL and a method and provide a request message.
- Then they use HTTP protocol to send a request to the API provider.
- A response message is returned after accessing a remote resource, e.g., a CICS program, a Db2 table, an IMS transaction or a MQ queue. The client application is unaware of the underlying infrastructure. No dependencies on coding for ECI, JDBC, OTMA, JMS, J2C, etc.

mitchj@us.ibm.com

Just a FYI: This same pattern applies to the z/OS API requester solution

ZceeCICSGet.java

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

ZceeDb2Get.java

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

z/OS Connect API requester support provides the same pattern for COBOL applications running on z/OS. The application simply provides a URI path, a method and a request message in a working storage structure. Then passes the addresses of the working storage areas as a parameters in a COBOL CALL to the z/OS Connect server. The interaction with the remote API endpoint is encapsulated in the z/OS Connect server, therefore there is a minimum need for new or changed COBOL code.

mitchj@us.ibm.com

These are the some of reasons why REST is popular today

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.

mitchj@us.ibm.com

Let’s start with how are REST API described

By using an OpenAPI specification document

The industry standard framework for describing REST APIs

The OpenAPI Initiative (OAI) was created by a consortium of forward-looking industry experts who recognize the immense value of standardizing on how APIs are described. As an open governance structure under the Linux Foundation, the OAI is focused on creating, evolving and promoting a vendor neutral description format. The OpenAPI Specification was originally based on the Swagger Specification, donated by SmartBear Software.

For more information see, https://en.wikipedia.org/wiki/OpenAPI_Specification

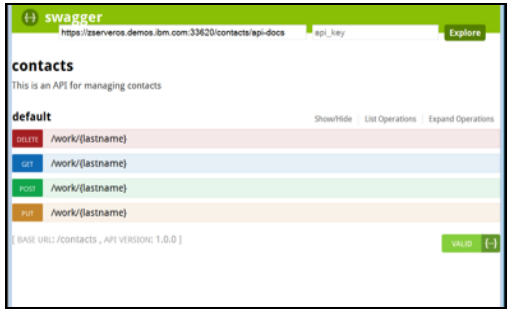
mitchj@us.ibm.com

There are a number of tools available to aid in the creation and consumption of APIs

Code Generation* - create stub code, to consume APIs from various languages, *e.g., Java data beans, COBOL copy books that describe the request and response messages.*

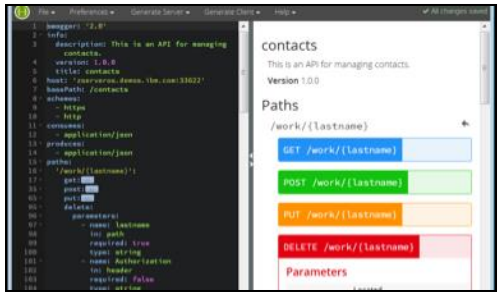
* z/OS Connect API Requester

Test UIs - allows API consumers to easily browse and try APIs based on an OpenAPI document.



+z/OS Connect, MQ REST support, Liberty, etc.

Editors - allows API developers to design their OpenAPI documents.



https://swagger.io/
https://jsonformatter.org/

Important - You may have used or heard of the term Swagger with the use of APIs. Swagger was the original name of the specification but in 2016 the Swagger specification was renamed the OpenAPI Specification (OAS).

mitchj@us.ibm.com

What is the significance of the OpenAPI Specification to z/OS Connect?

The OpenAPI specification is the industry standard framework for describing REST APIs

- **z/OS Connect and Open API Specification 2, (formally known as Swagger 2.0), was supported initially by z/OS Connect**

  Initially, accessing z/OS resources was the only desire for developing APIs .The interactions with the z/OS resources was driven by the layout of the CICS COMMAREA or CONTAINER, the IMS or MQ messages or the Db2 REST service.
  - The details of the interactions with the z/OS resource determined the contents of the API request and response messages and the subsequent specification document.
  - **z/OS Connect produces the specification document that describes the methods and request and response messages.**

- **z/OS Connect and Open API Specification 3, supported by z/OS Connect starting in March 2022 service, V3.0.55**

  As companies mature their API strategy, they begin to introduce API governance boards to drive consistency in their API design. As more public APIs are created, government and industry standards bodies begin to regulate and drive for standardization. This drives the need for “API first” functional mapping capabilities within the integration platform. The external API design determined the layouts of the API request and response messages provided by the specification documents which was consumes by z/OS Connect to describe the z/OS resource interactions.
  - The API details of the methods and layouts of request and response messages are provided in advance and access to the z/OS resource is driven by the API design
  - **z/OS Connect consumes the specification document that describes the methods and request and response messages**

Note: For our purposes, the terms OpenAPI 2.0 and Swagger 2.0 are interchangeable.

mitchj@us.ibm.com

Contrast the Swagger 2 versus OpenAPI 3 specification differences

z/OS Connect OpenAPI2 tooling **produces** an OpenAPI 2 specification document (aka OpenAPI 2.0), where the details of the methods and request/response messages in the API specification are driven by the nature of the z/OS resource (JSON format).

cscvinc.json - Notepad

File Edit Format View Help

```
{
    "swagger": "2.0",
    "info": {
        "description": "",
        "version": "1.0.0",
        "title": "cscvincapi"
    },
    "basePath": "/cscvincapi",
    "schemes": [
        "https",
        "http"
    ],
    "consumes": [
        "application/json"
    ],
    "produces": [
        "application/json"
    ],
    "paths": {
        "/employee/{employee}": {
            "get": {
                "tags": [
                    "cscvincapi"
                ],
                "operationId": "getCscvincSelectService",
                "parameters": [
                    {
                        "name": "Authorization",
                        "in": "header",
                        "required": false,
                        "type": "string"
                    },
                    {
                        "name": "employee",
                        "in": "path",
                        "required": true,
                        "type": "string",
                        "maxLength": 6
                    }
                ],
                "responses": {
                    "200": {
                        "description": "OK",
                        "schema": {
                            "$ref": "#/definitions/getCscvincSelectService_response_200"
                        }
                    },
                    "404": {
                        "description": "Not Found",
```

Ln 18, Col 7 100% Windows (CRLF) UTF-8

cscvinc.yaml - Notepad

File Edit Format View Help

```
openapi: 3.0.1
info:
  title: cscvinc
  description: ""
  version: 1.0.0
servers:
- url: /cscvinc
x-ibm-zcon-roles-allowed:
- Manager
paths:
  /employee:
    post:
      tags:
      - cscvinc
      operationId: postCscvincInsertService
      x-ibm-zcon-roles-allowed:
        - Staff
      parameters:
      - name: Authorization
        in: header
        schema:
          type: string
      requestBody:
        description: request body
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/postCscvincInsertService_request'
        required: true
      responses:
        200:
          description: OK
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/postCscvincInsertService_response_200'
      x-codegen-request-body-name: postCscvincInsertService_request
  /employee/{employee}:
    get:
      tags:
      - cscvinc
      operationId: getCscvincSelectService
      x-ibm-zcon-roles-allowed:
        - Staff
      parameters:
      - name: Authorization
        in: header
        schema:
          type: string
```

Ln 44, Col 16 100% Unix (LF) UTF-8

z/OS Connect OpenAPI3 tooling **consumes** an OpenAPI3 specification document (aka OSA3) and the details of the methods and request/response messages are driven by the API specification (YAML format*) and not the nature of the z/OS resource. Also, JSONata can be used to augment the API response

*Yet Another Markup Language

mitchj@us.ibm.com

https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=migrating-openapi-2-openapi-3

What value does zOS Connect Add for REST API clients?

Truly RESTful APIs to and from your mainframe.

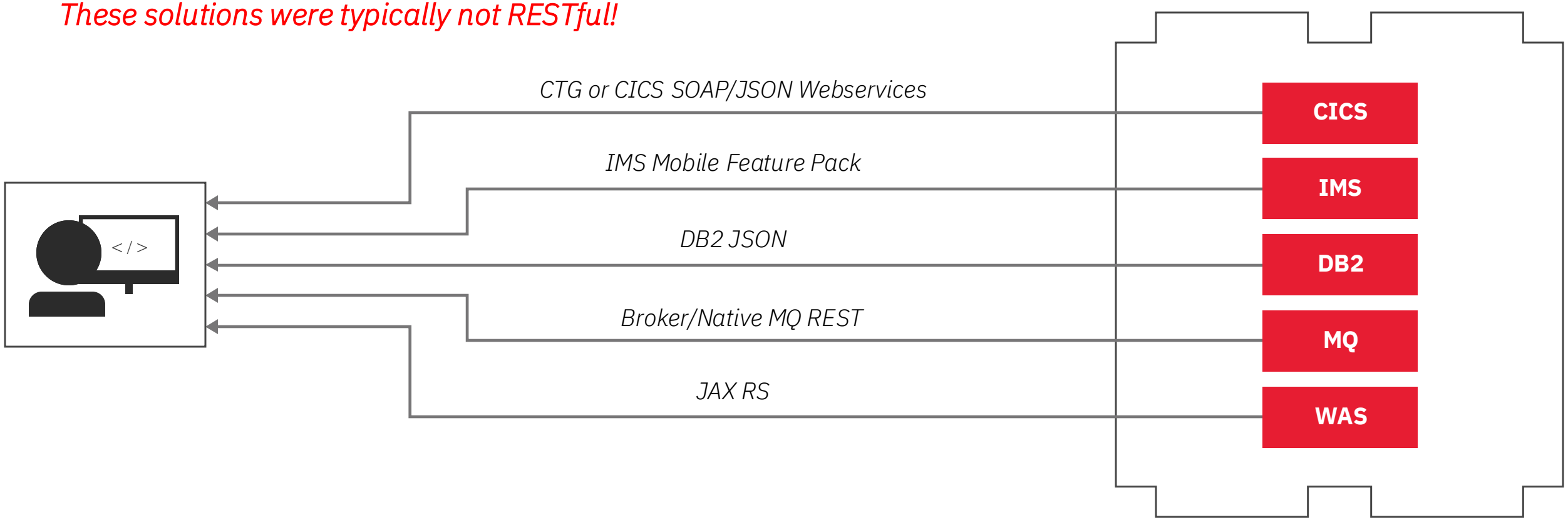
mitchj@us.ibm.com

There was support for REST to z/OS before z/OS Connect but..

Completely different configuration and management for each subsystem

Multiple endpoints for developers to call/maintain for access

These solutions were typically not RESTful!



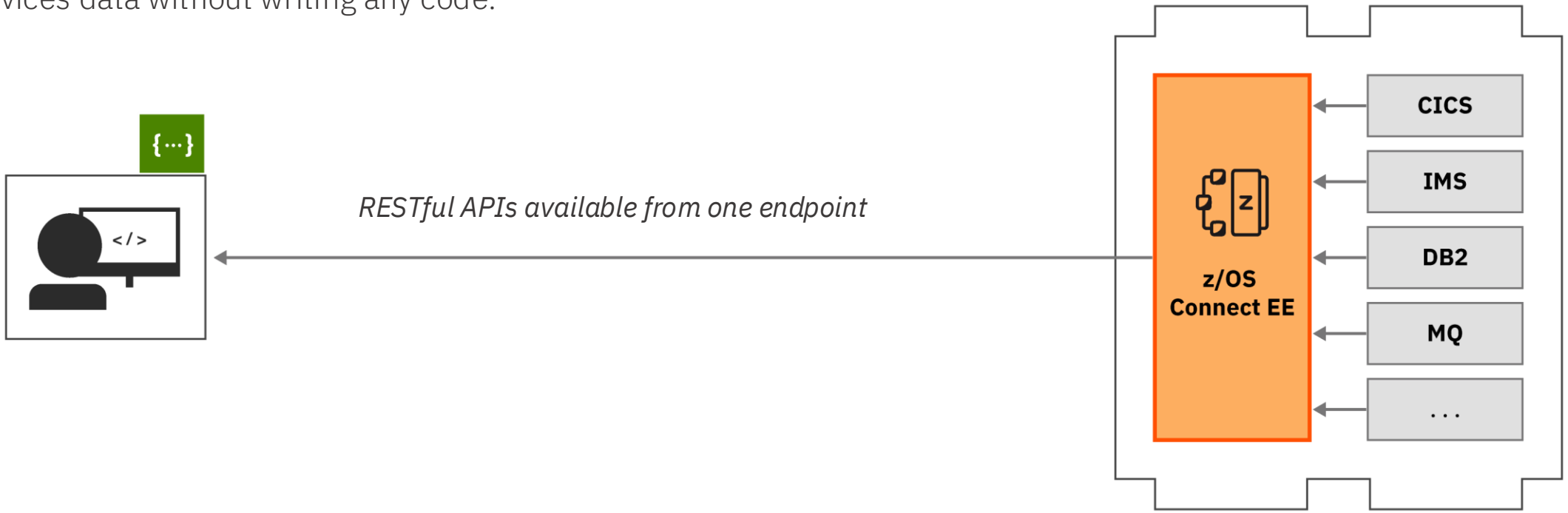
mitchj@us.ibm.com

z/OS Connect provided a single-entry point to z/OS resources

Without the need to write new code or change any existing code.

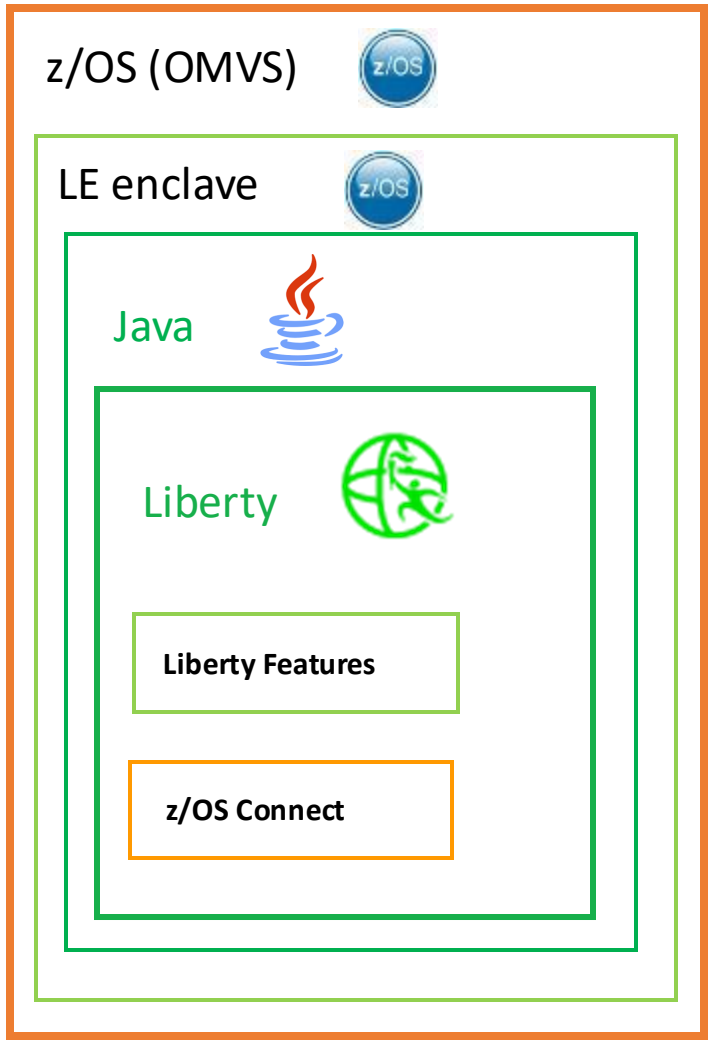
z/OS Connect provides

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



mitchj@us.ibm.com

A z/OS Connect instance executes in a Liberty environment on z/OS



- That is started by an OMVS script that starts the Java environment (in an OMVS process).
- Runs in a Language Environment (LE) enclave configured to support OMVS and Java processes.
- Liberty is a Java application and on z/OS, there are features that provide access to z/OS services and facilities (e.g., SAF, WLM, RRS, SMF, JCL, started tasks, etc.).
- The Liberty Java application provides an exeuction environment for multiple concurrent Java applications.

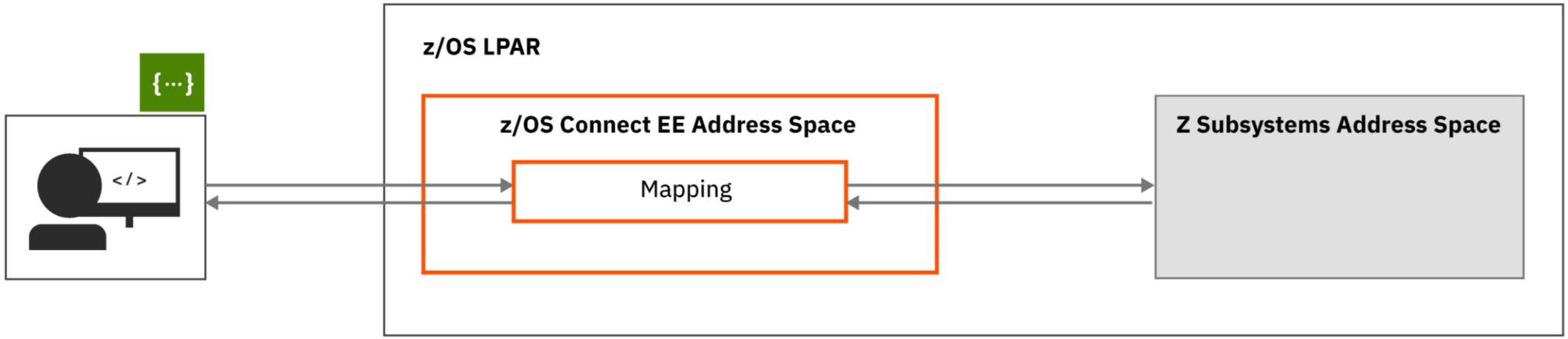
Knowing the different layers and their relationship is important regarding

- Understanding which layer a configuration options, e.g., environment variables, etc., applies.
- Monitoring and understanding the health of the server
- Performing problem determination and performance tuning

mitchj@us.ibm.com

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

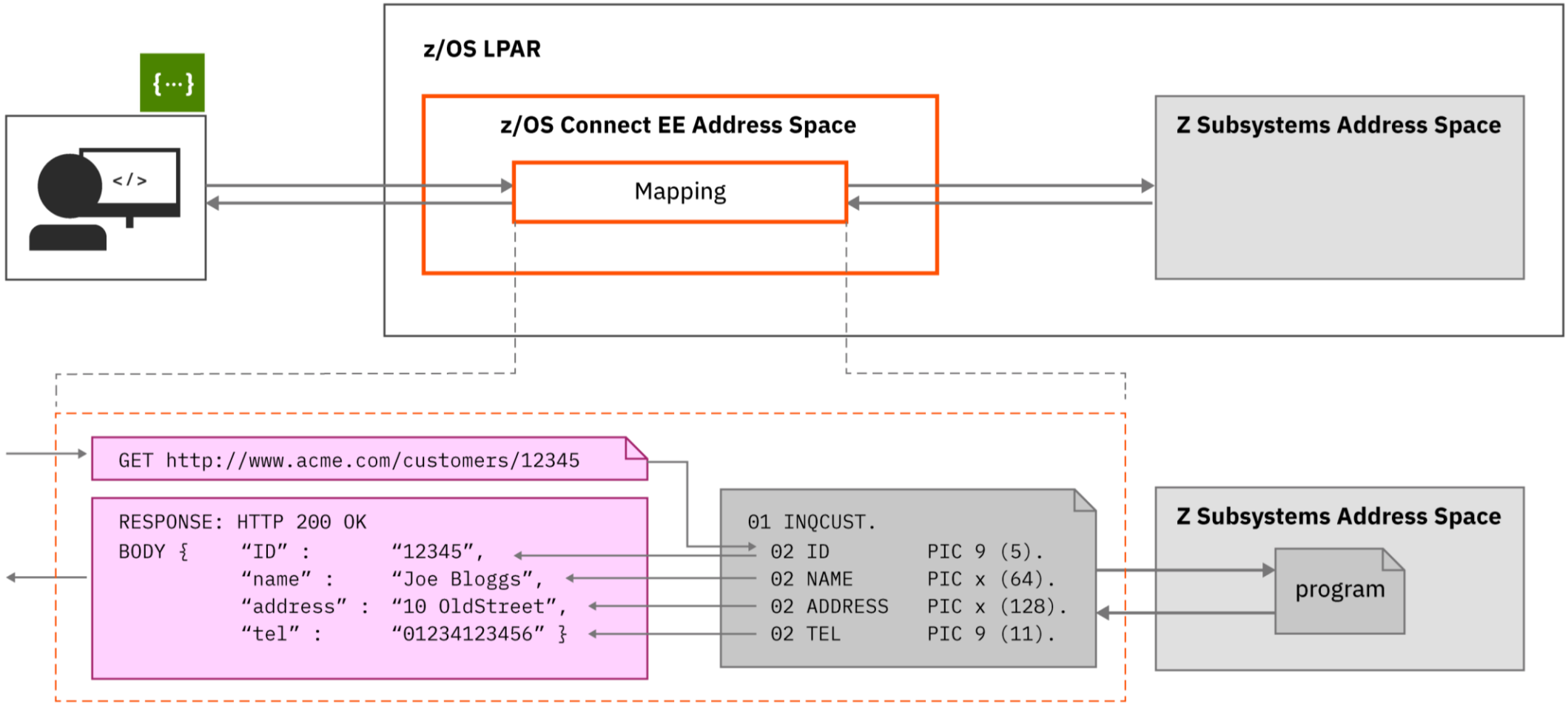
Automatic data mapping/transformation of the JSON message to a format the target's subsystem expects*



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

mitchj@us.ibm.com

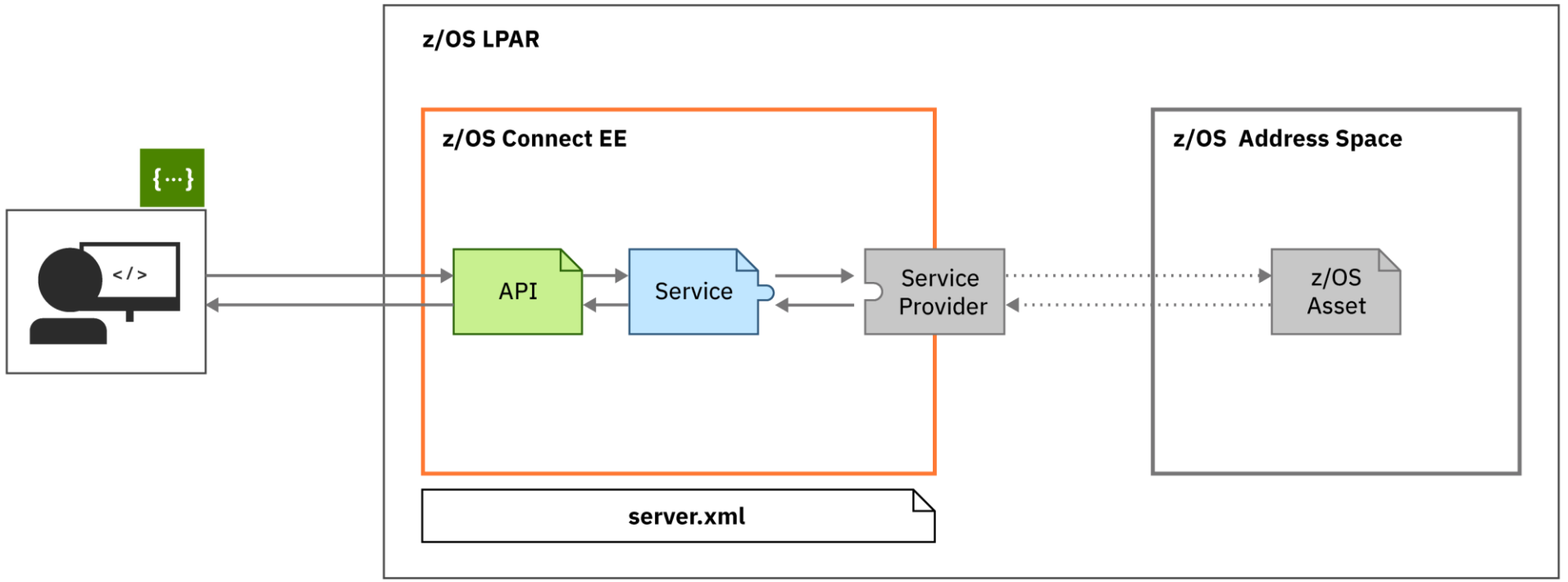
Data mapping and transformation – a closer look



mitchj@us.ibm.com

Accessing an z/OS resource using an Open API 2 API

- z/OS Connect OpenAPI 2 APIs are developed using an Eclipse based API Toolkit to create Service Archive (SAR) and API ARchive (AAR) files.

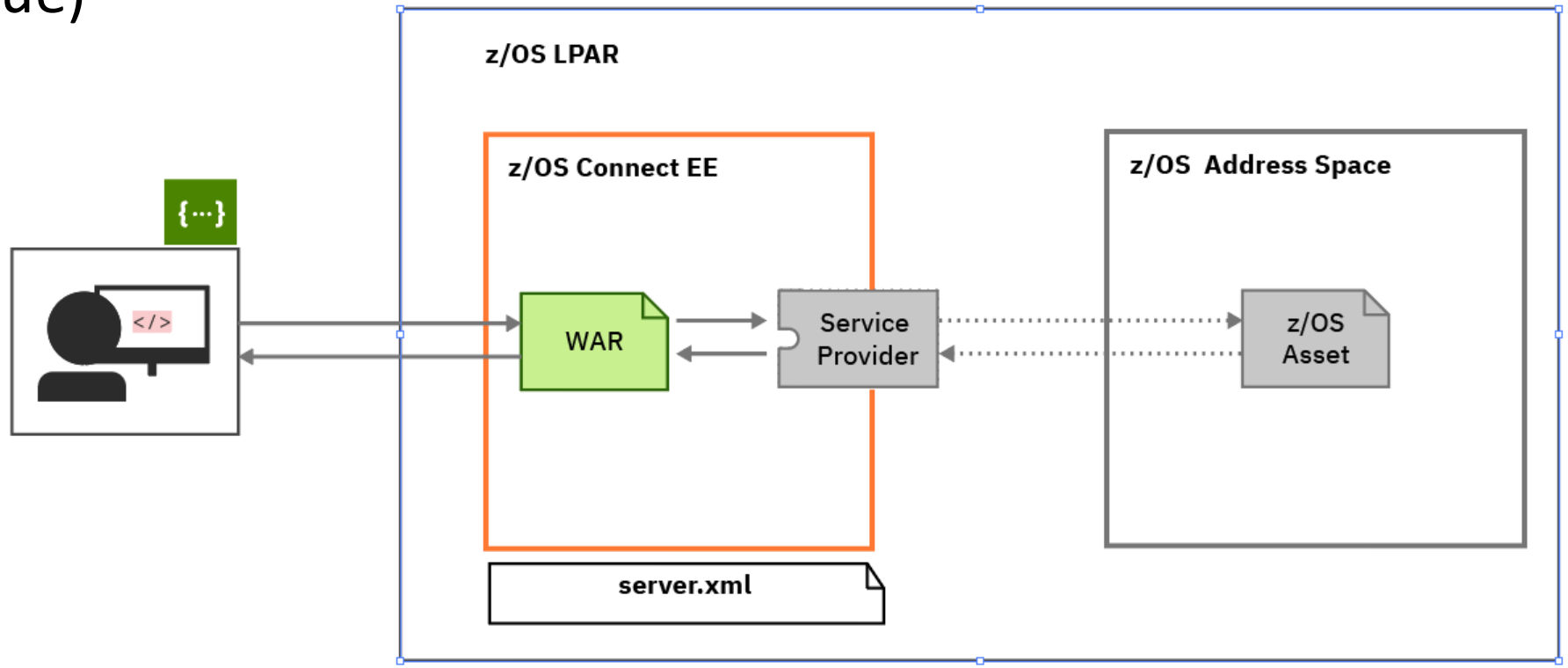


- The API is ready to be consumed and requires no knowledge that a z/OS resource is being accessed
- The Service provides meta data specific to the z/OS resource (e.g., CICS program, MQ queue manager, etc.)
- The Service Provider is tightly coupled to a specific instance of a resource (e.g., host and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

mitchj@us.ibm.com

Accessing a z/OS resource using an Open API 3 API

- z/OS Connect OpenAPI 3 APIs are developed using the z/OS Connect Designer web browser tool to create Web ARchive (WAR) files (a traditional Java packaging technique)



- The WAR provides the RESTful interface is ready to be consumed by a client and it requires the client to have no knowledge that a z/OS resource is being accessed as well as the mapping and transformation for accessing the resource.
- The Service Provider configuration is tightly coupled to a specific instance of a resource (e.g., host and port)

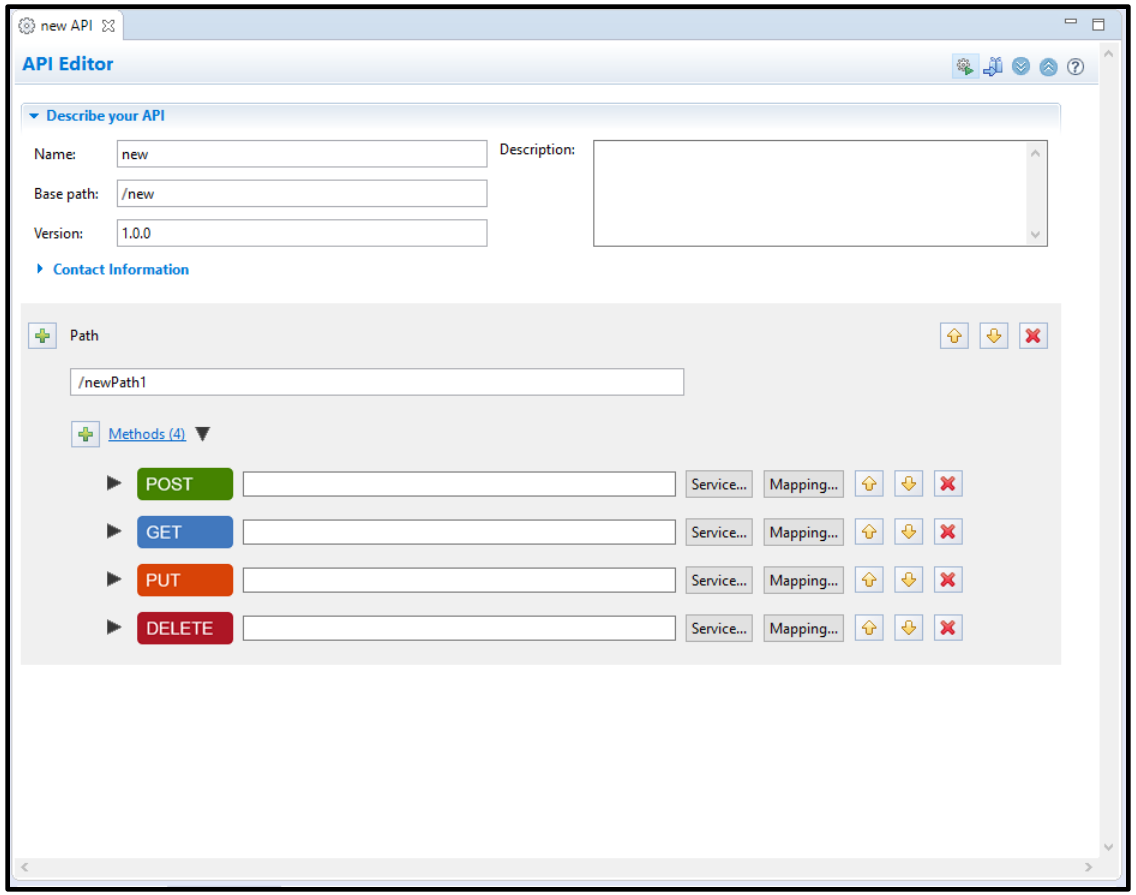
mitchj@us.ibm.com

z/OS Connect OpenAPI Tooling

mitchj@us.ibm.com

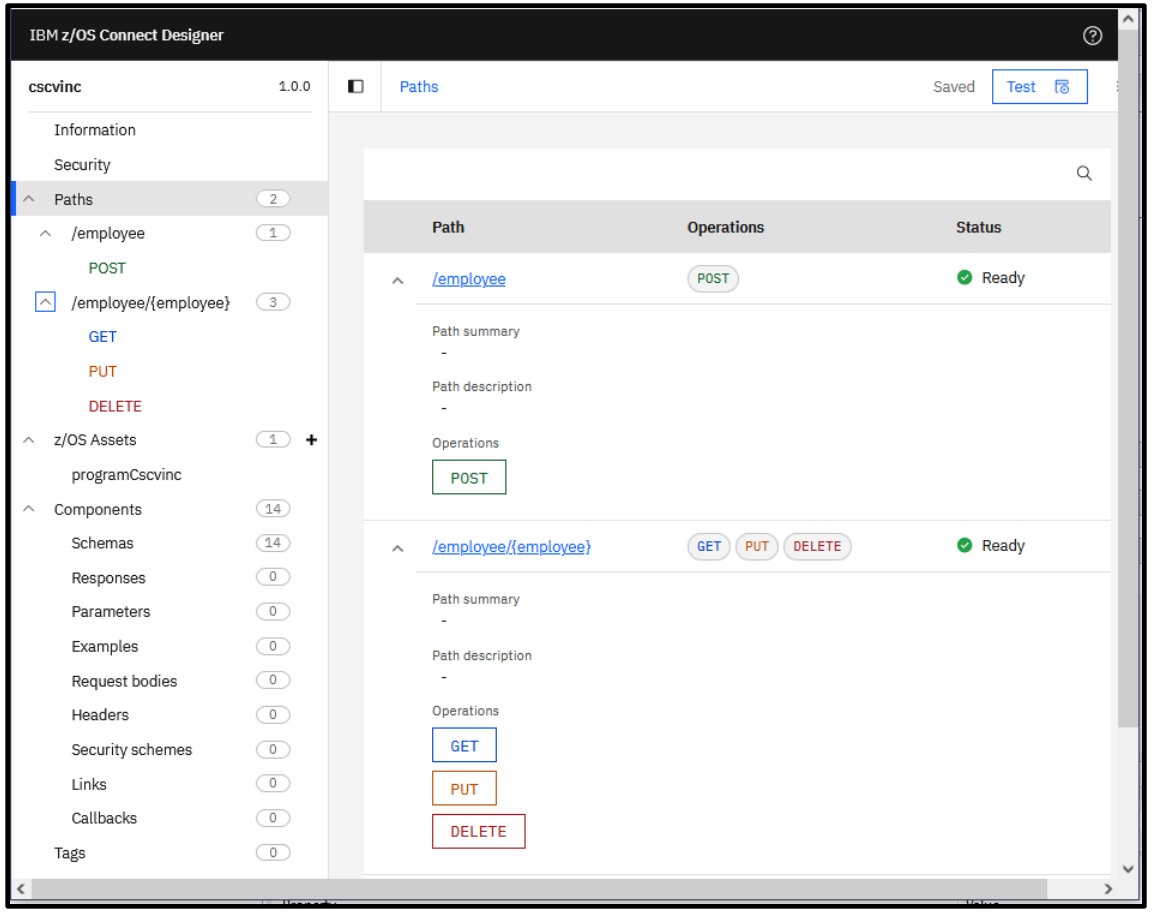
z/OS Connect's OpenAPI 2 Palette versus the OpenAPI 3 API Designer

z/OS Connect API Toolkit (Eclipse)



The API toolkit is used to define the URI paths and methods.

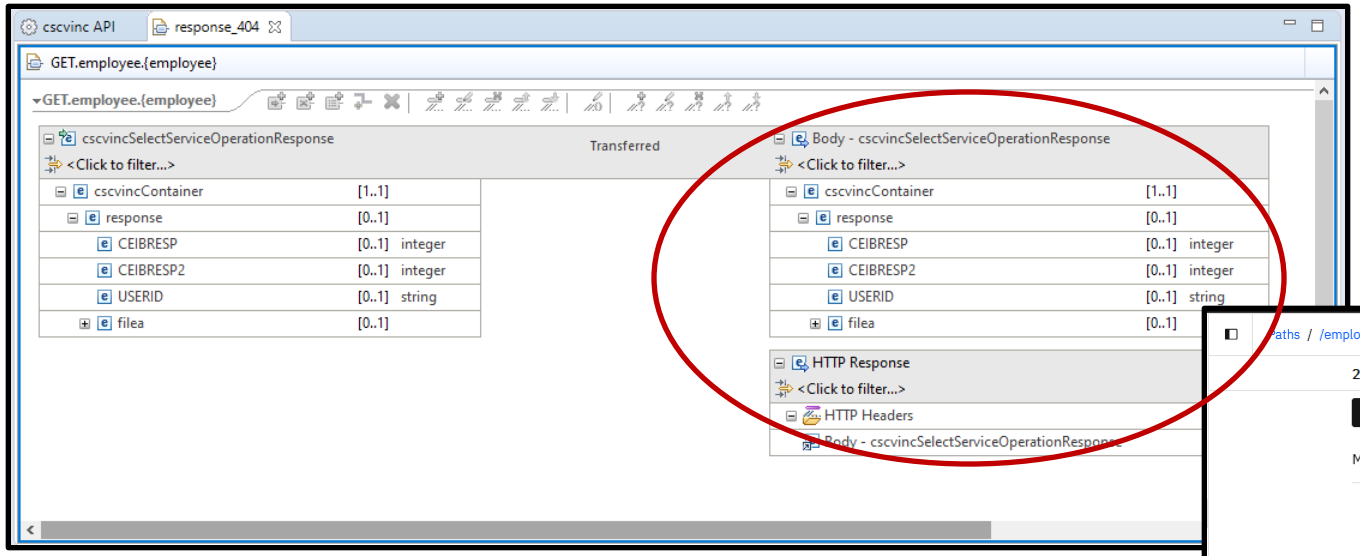
z/OS Connect Designer (Designer Container)



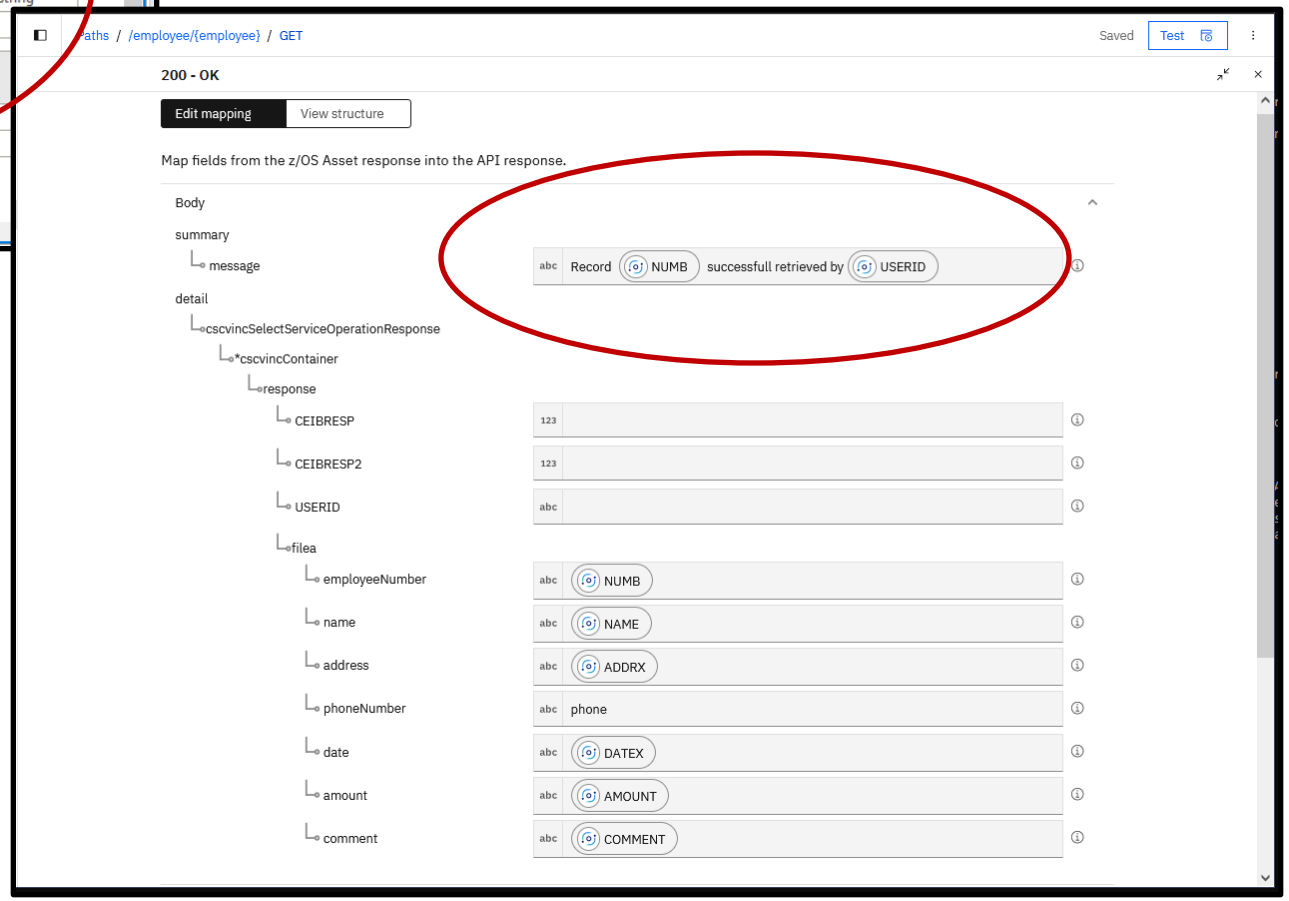
The API specification provides the predefined URI Paths and methods.

mitchj@us.ibm.com

A distinction between OpenAPI 2 and OpenAPI 3 response message



Eclipse based - z/OS Connect API Toolkit -
The contents of the API's response message are directly derived from the fields exposed in the z/OS resource's response.



Web browser - z/OS Connect Designer –
The contents of the API's response message are derived from the z/O resource's response and JSONata coding, see URL https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=connect-what-is-jsonata for more information on JSONata. Note that there maybe additional fields, e.g., *message* that are not in the z/OS resource's response message.

mitchj@us.ibm.com

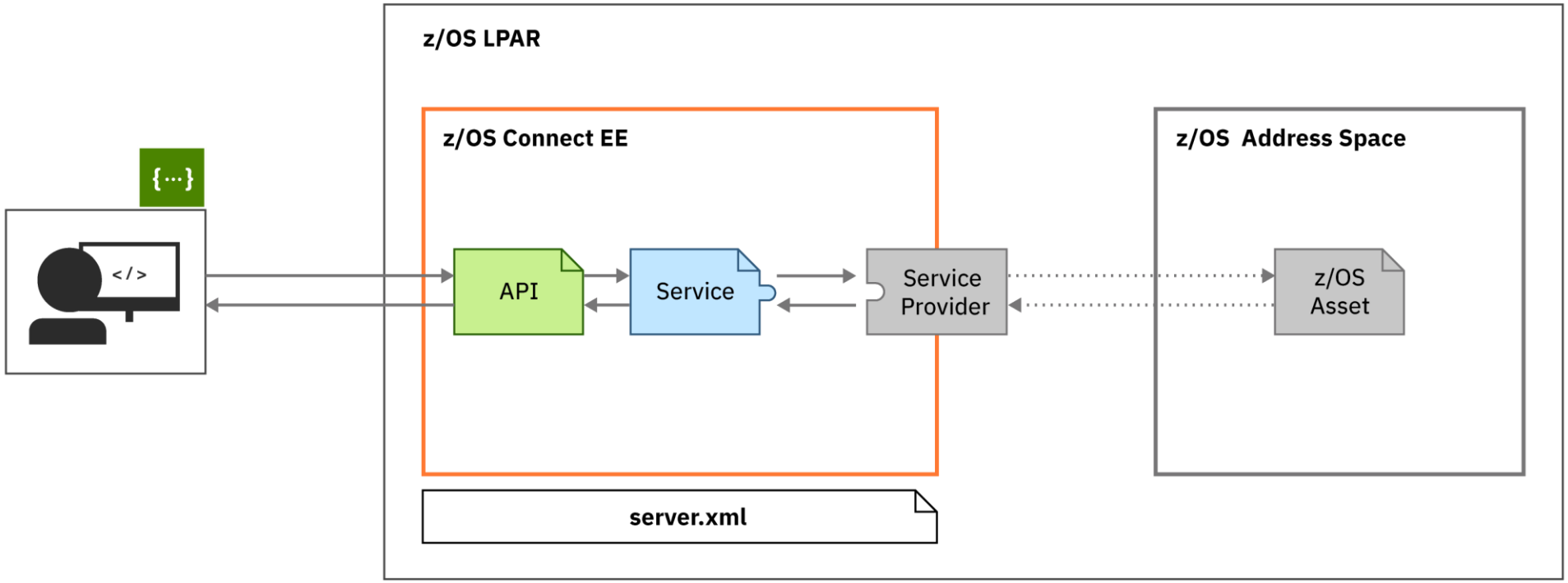
OpenAPI 2

Creating **Services** using the Eclipse Toolkit

mitchj@us.ibm.com

Accessing a z/OS resource with an Open API 2 API

z/OS Connect OpenAPI 2 uses separate plug-and-play components



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

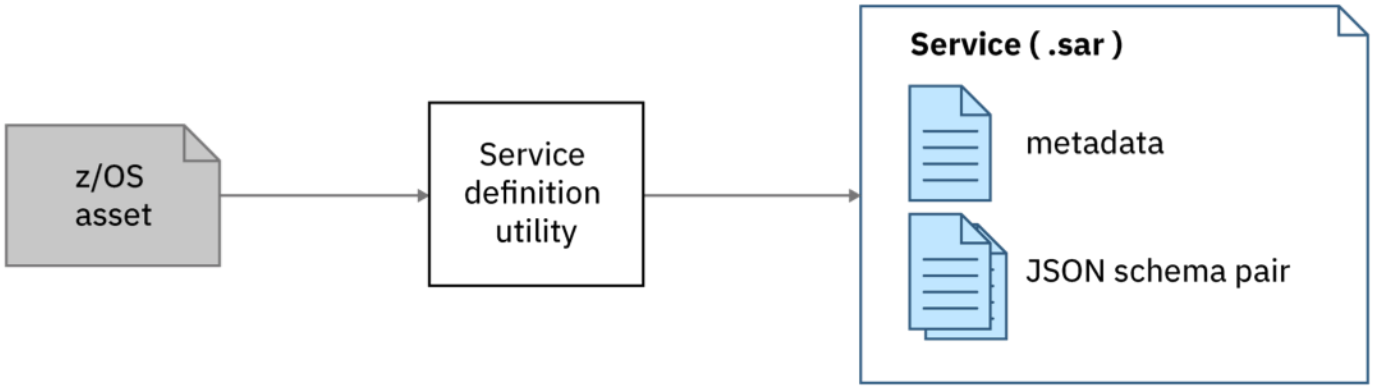
mitchj@us.ibm.com

Describing the interaction (service) with the z/OS resource

Start by creating a service to represent an interaction with the z/OS resource

To start mapping an API, z/OS Connect 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




```
EVALUATE ACTION of Request-Container
    WHEN 'D'
      PERFORM Delete-Record
    WHEN 'I'
      PERFORM Insert-Record
    WHEN 'U'
      PERFORM Update-Record
    WHEN 'S'
      PERFORM Select-Record
END-EVALUATE.
```

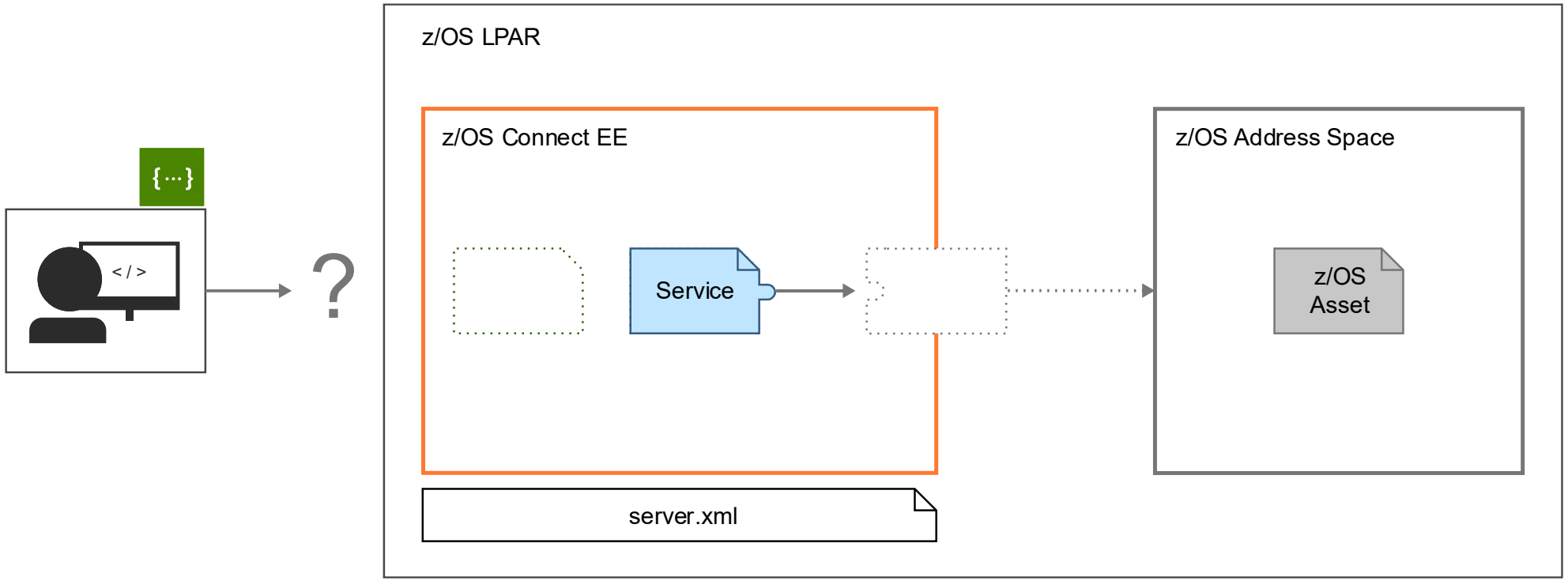
Use a system-appropriate utility to generate a service archive file for the z/OS application

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

mitchj@us.ibm.com

Deploy and export the service archive

Deploy to the server and export the service archive file for use in the API



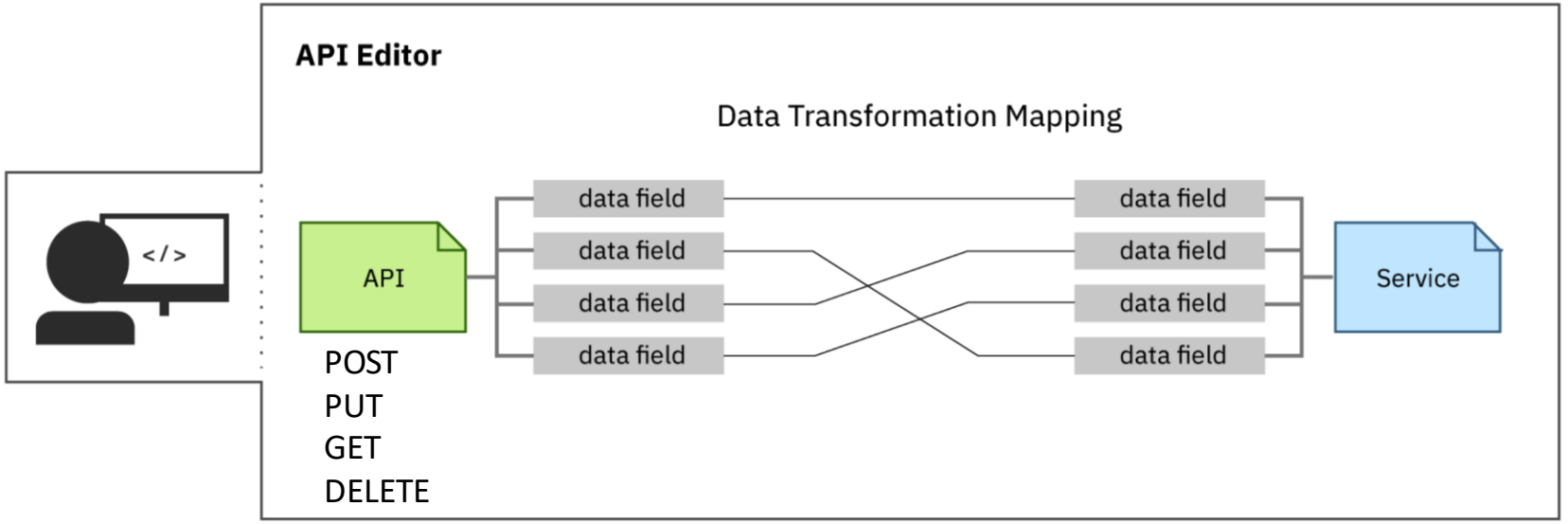
Use the right-click deploy in **the API toolkit**

- Deploy the service archive to the z/OS environment.
- Export the service archive to either another Eclipse project or a file system.

mitchj@us.ibm.com

Develop an API using the service interactions

Import the service archive file it to create an API that consumes the service

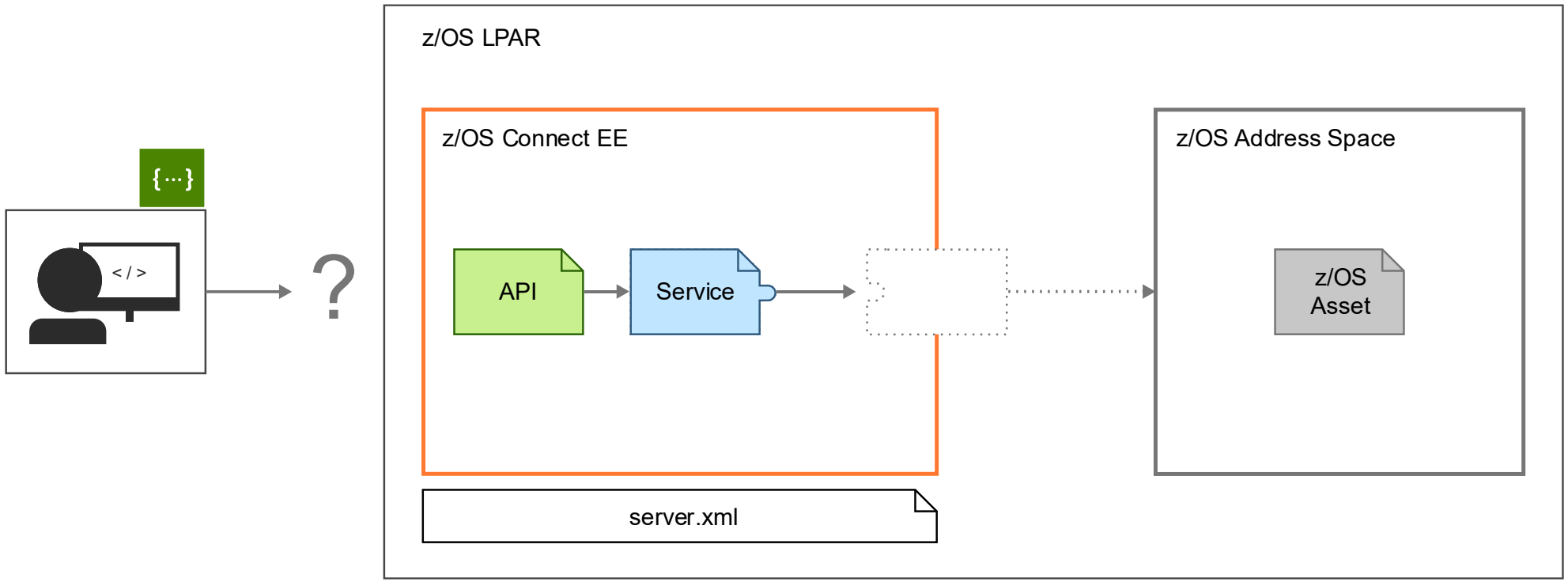


- Import the service archive into the **API toolkit** and start designing the RESTful API.
- Provides additional data mapping
- Use the editor to describe the API and how it maps to underlying services.
- Add logic to interpret the results to add support for multiple HTTP response code

mitchj@us.ibm.com

Deploy the Open API 2 API

Deploy the API archive file to the server

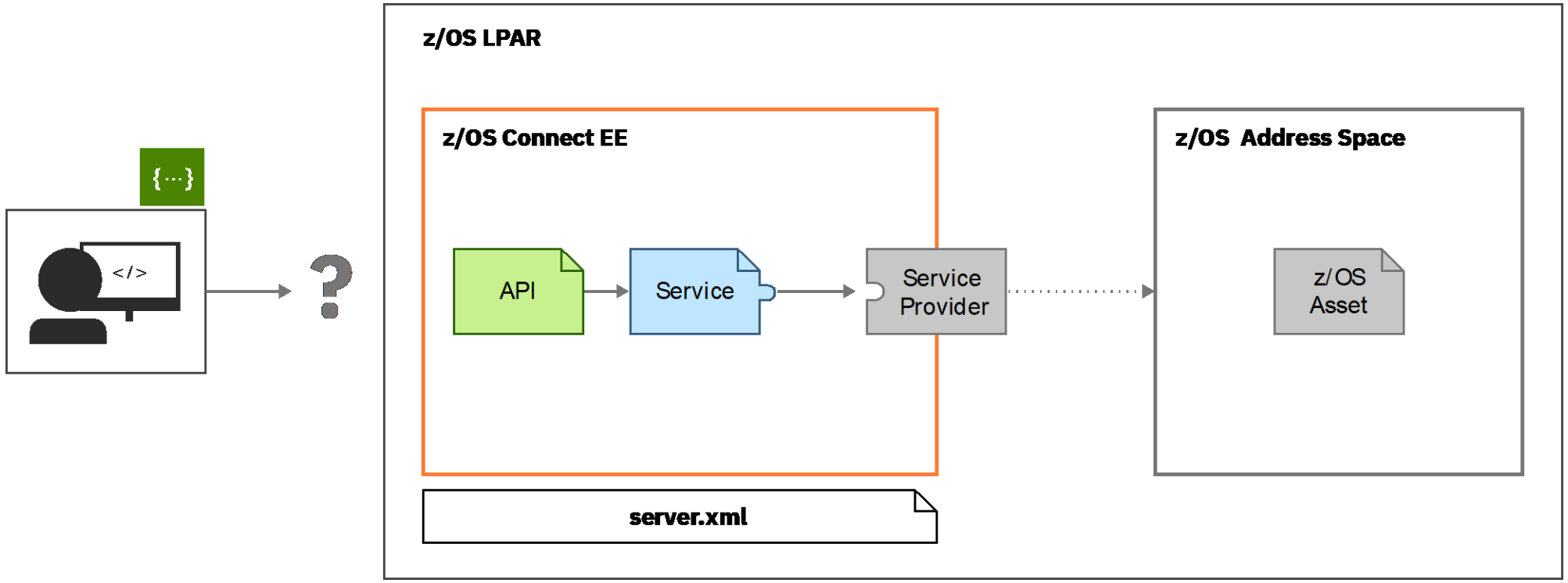


Deploy your API using the right-click deploy in **the API toolkit**

mitchj@us.ibm.com

Configure access to the z/OS subsystem

Configure the service provider

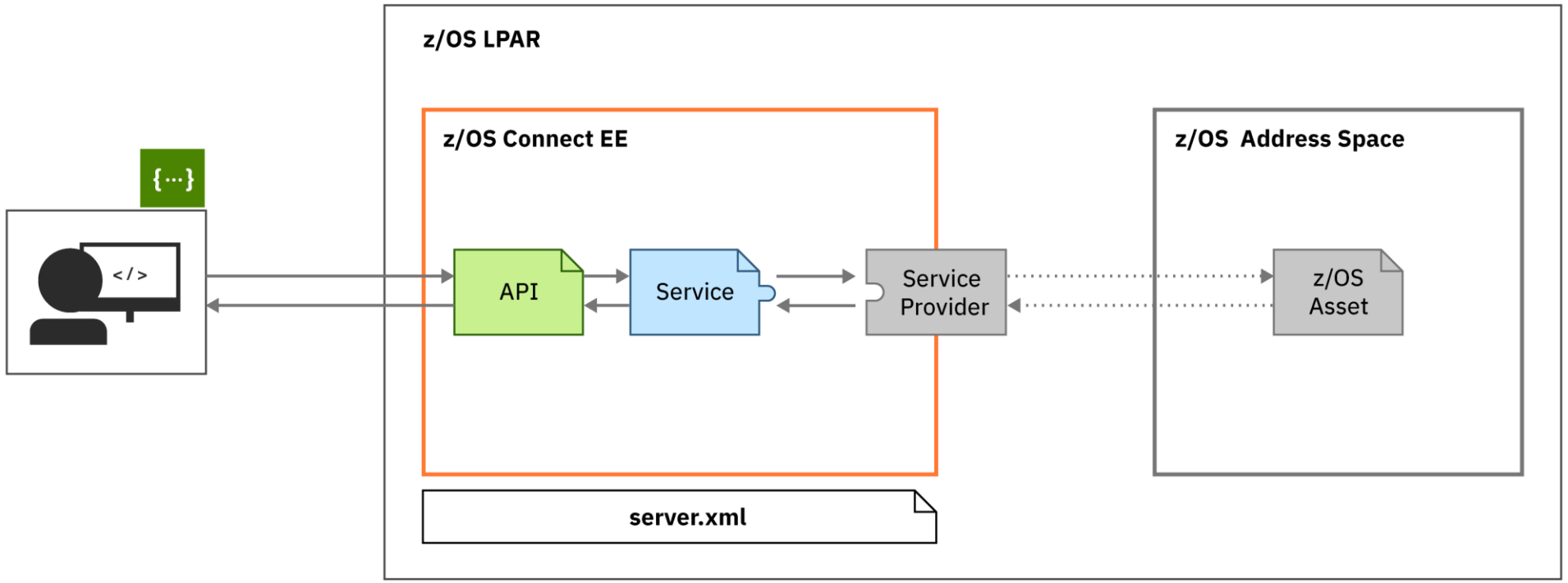


Configure the system-appropriate service provider to connect to your backend system in your `server.xml`.

mitchj@us.ibm.com

Results: complete RESTful access to a z/OS resource

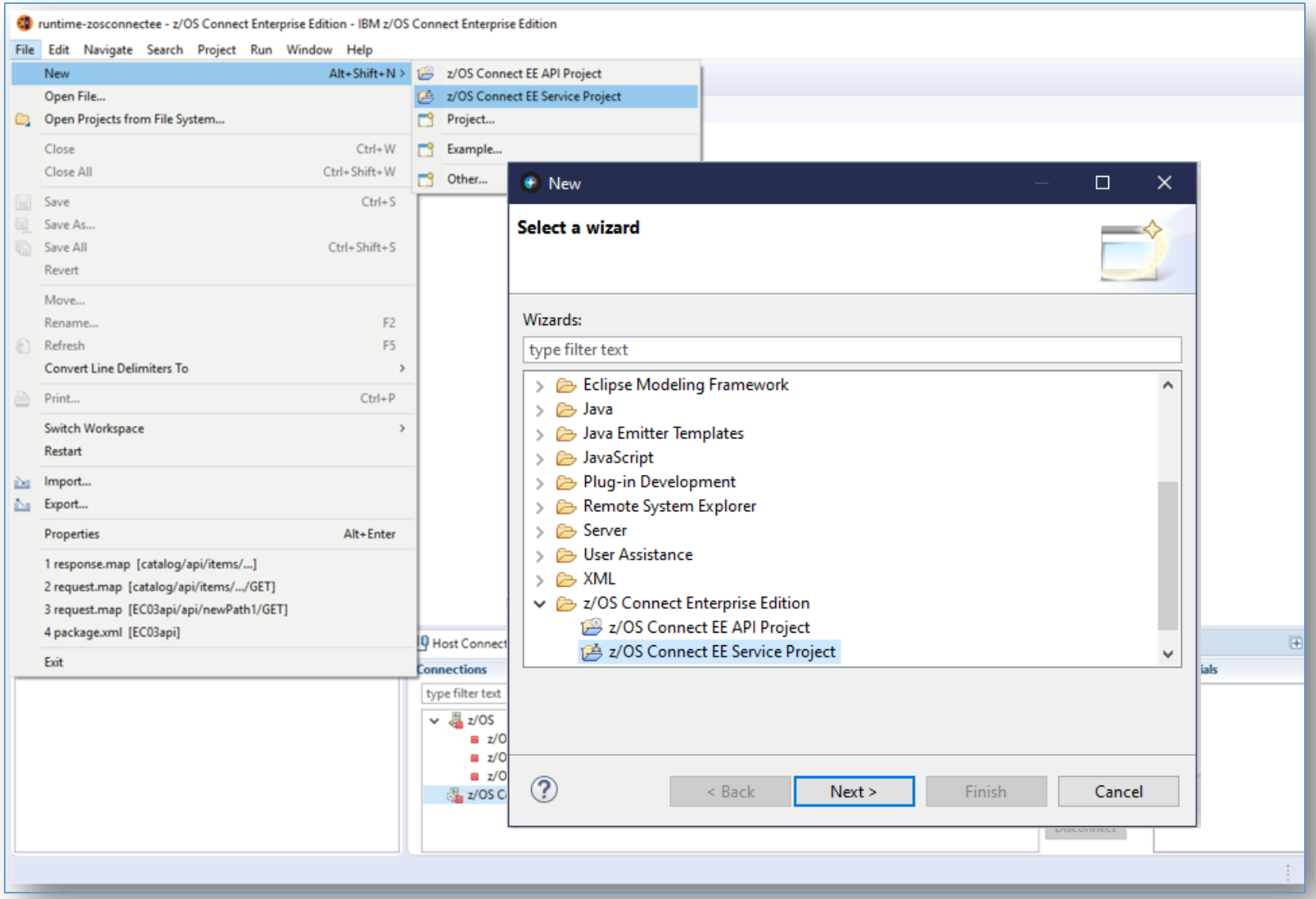
Done



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

mitchj@us.ibm.com

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



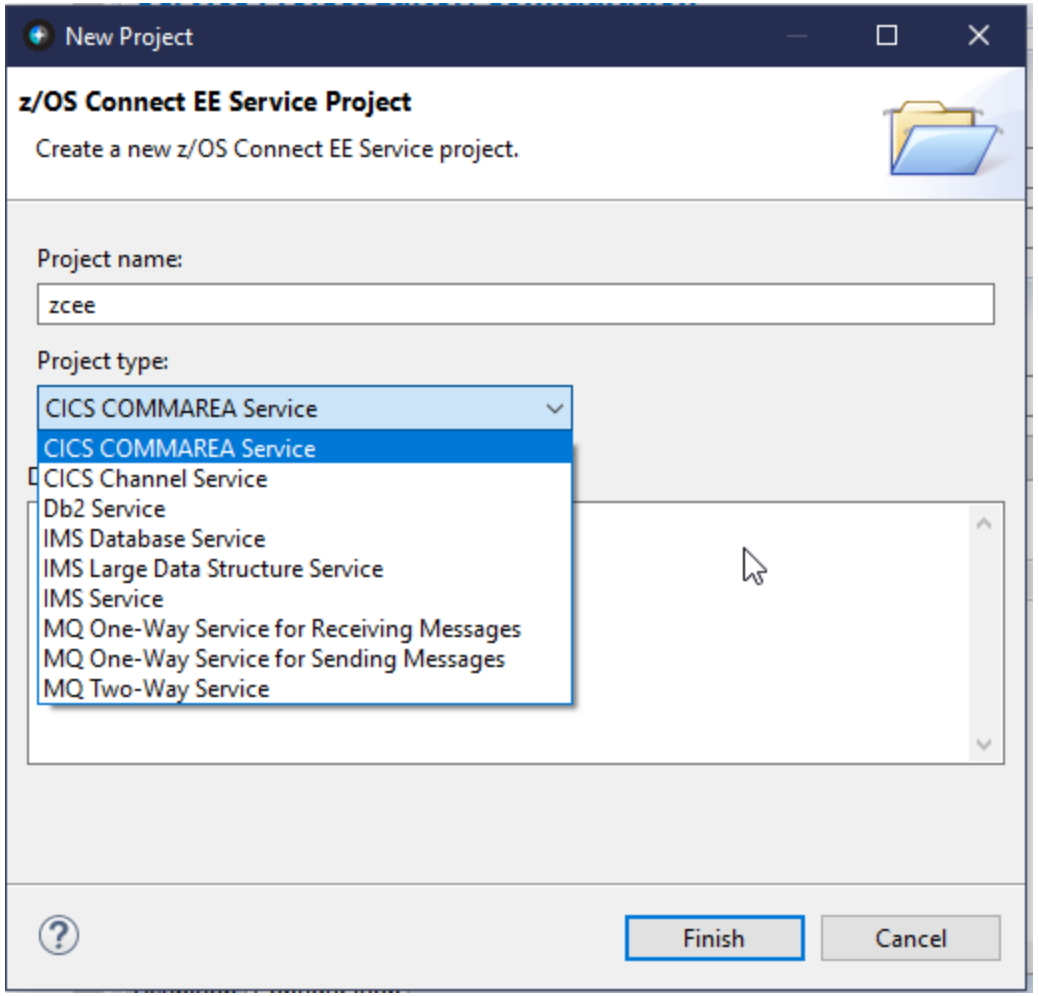
Use the **API toolkit** to create services through Eclipse-based tooling.

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

mitchj@us.ibm.com

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

Service creation – a common interface

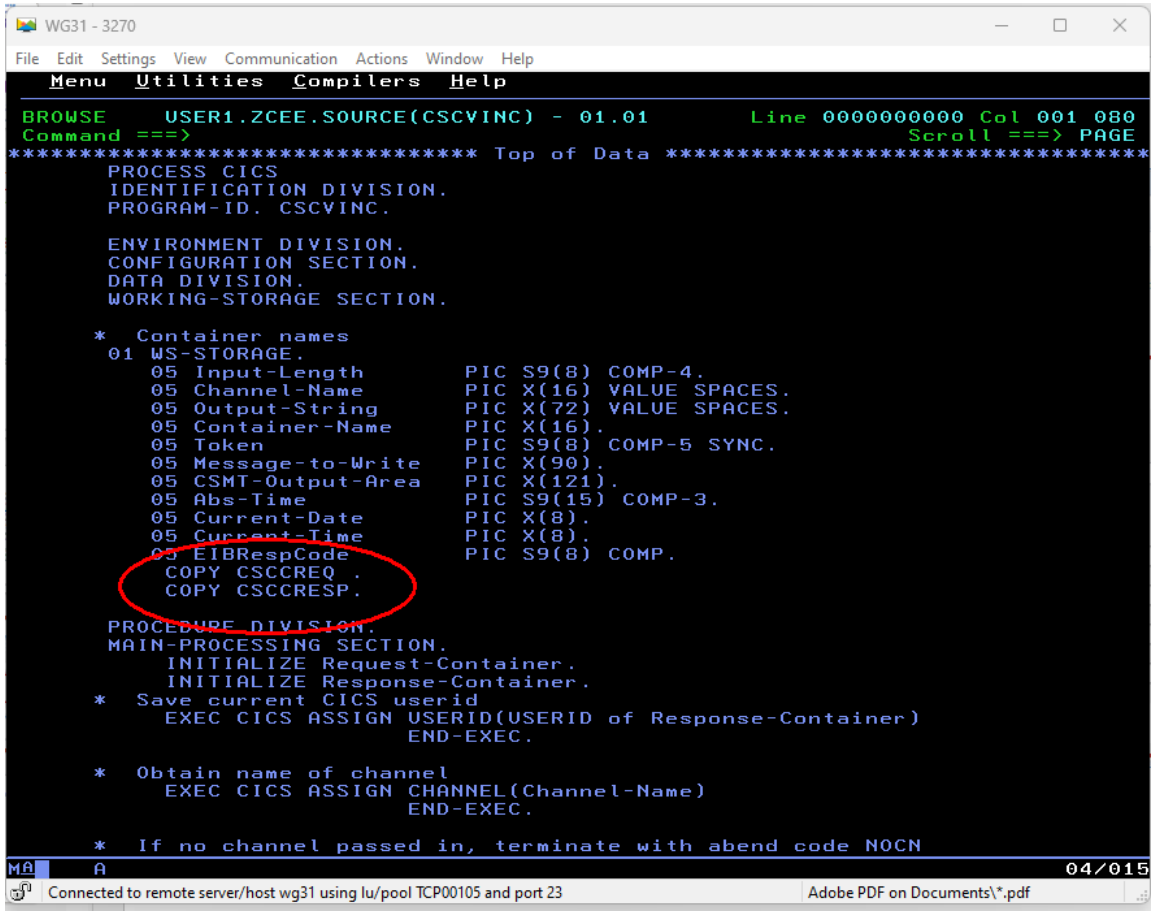


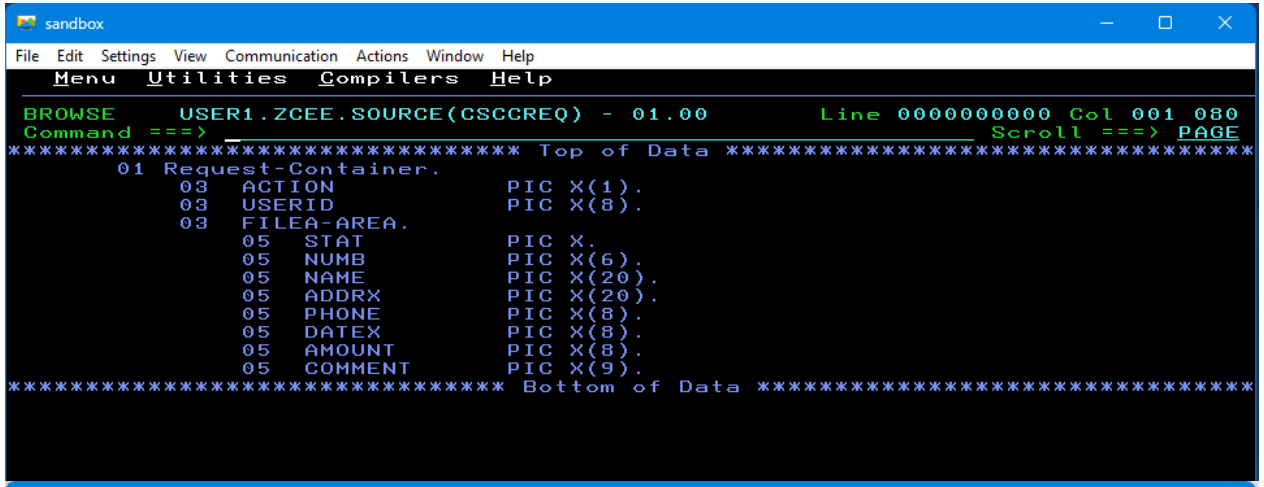
- CICS services that can invoke almost any CICS programs accessed by EXEC CICS LINK request (COMMAREA or CHANNEL). See URL https://www.ibm.com/docs/en/cics-ts/6.x?topic=dpl-exception-conditions-link-command for a list of EXEC CICS APIs not allowed in a program when invoked using a CICS Dynamic Program Link request.
- Db2 services that invoke a Db2 REST service.
- IMS DB services that access an IMS database.
- IMS TM services that sends a messages on an IMS message processing region.
- MQ services that use MQ request/reply queues for two-way services or access a single queue for MQ PUTs and MQ GETs on a either a local or remote queue manager

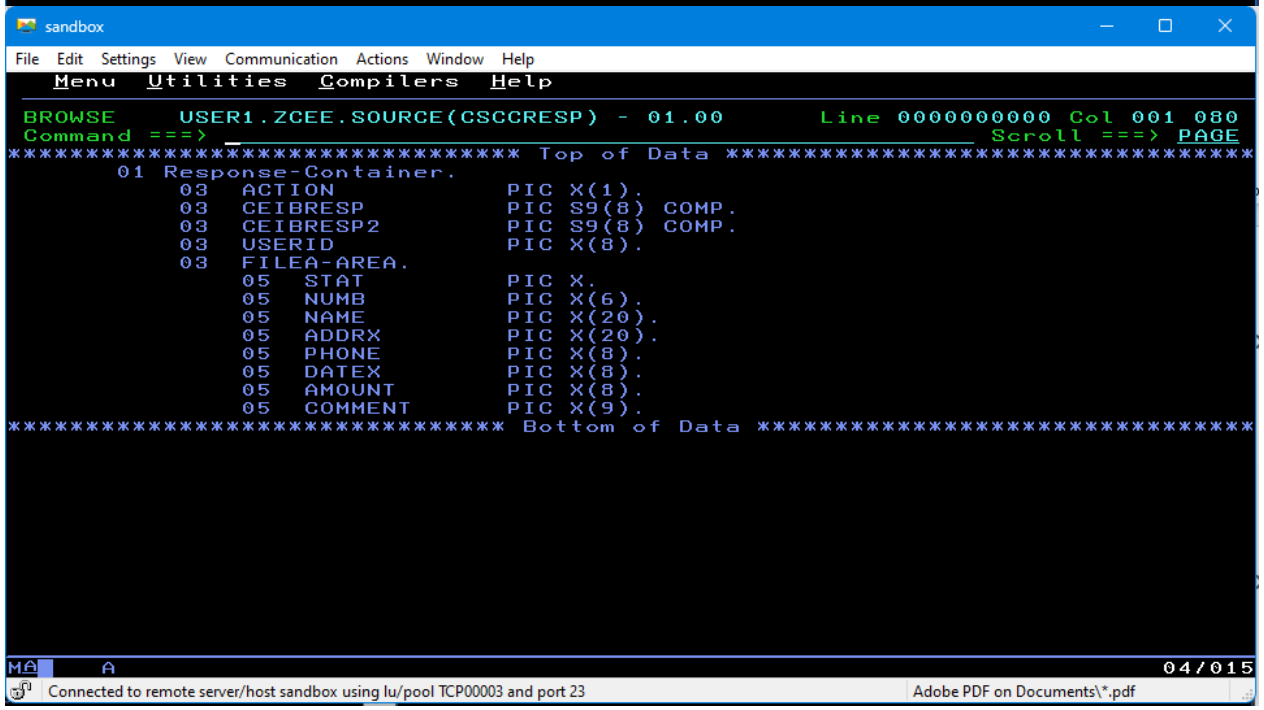
A common interface for service creation, irrespective of back-end subsystem.

mitchj@us.ibm.com

You start by identifying the format of the messages (copy books) used by the application



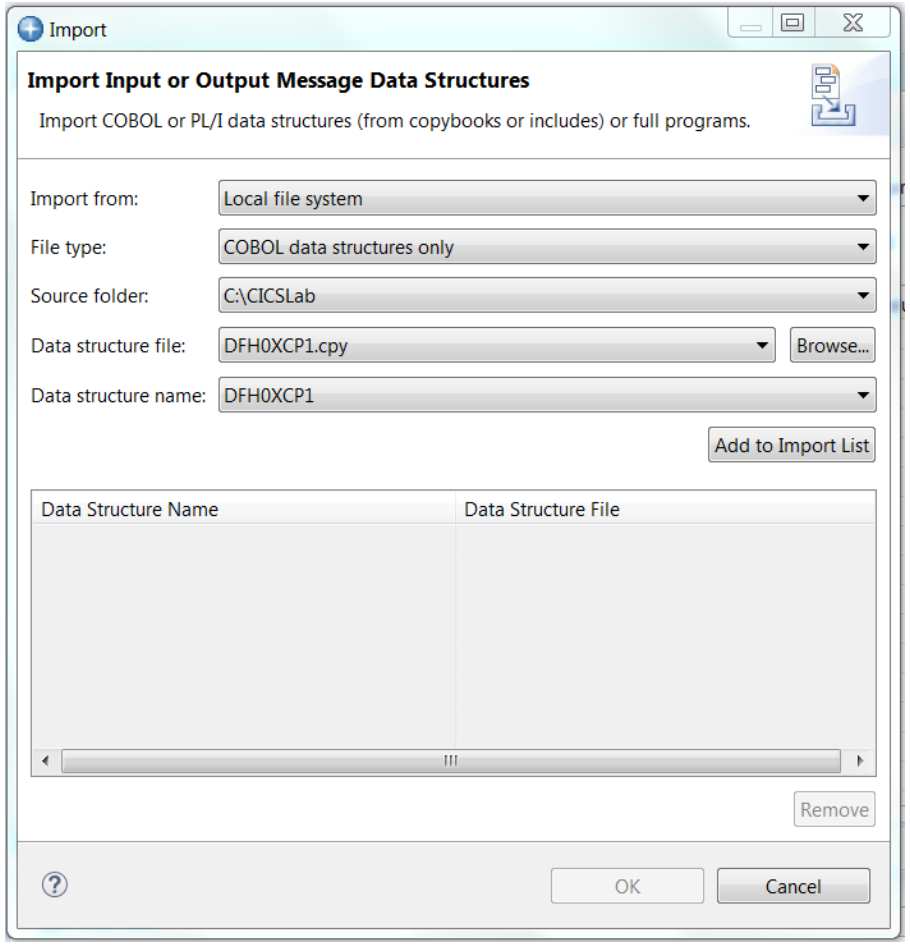




mitchj@us.ibm.com

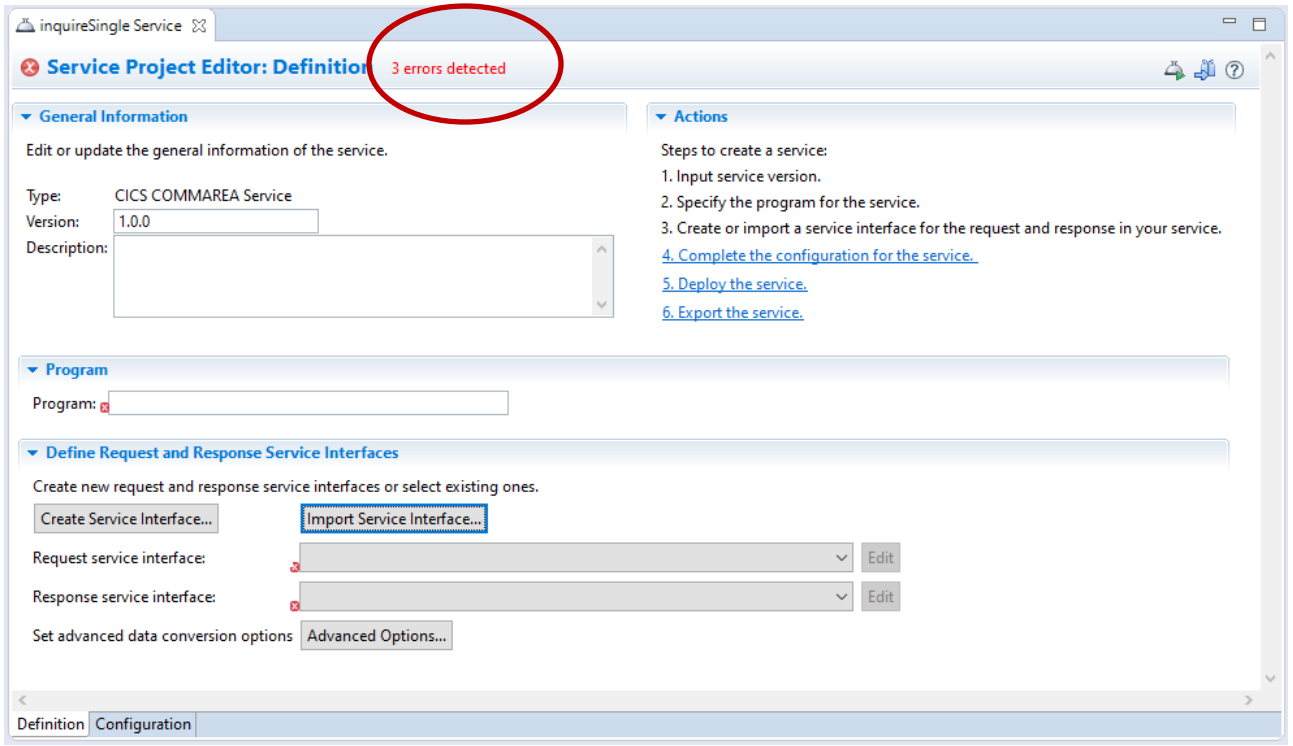
API toolkit – Creating Services for CICS, IMS TM and MQ

for a message driven resource, e.g., a CICS COMMAREA or Container or IMS or MQ Message



Start by importing data structures into the service interface from the local file system or the workspace to create the request and response service interfaces.

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



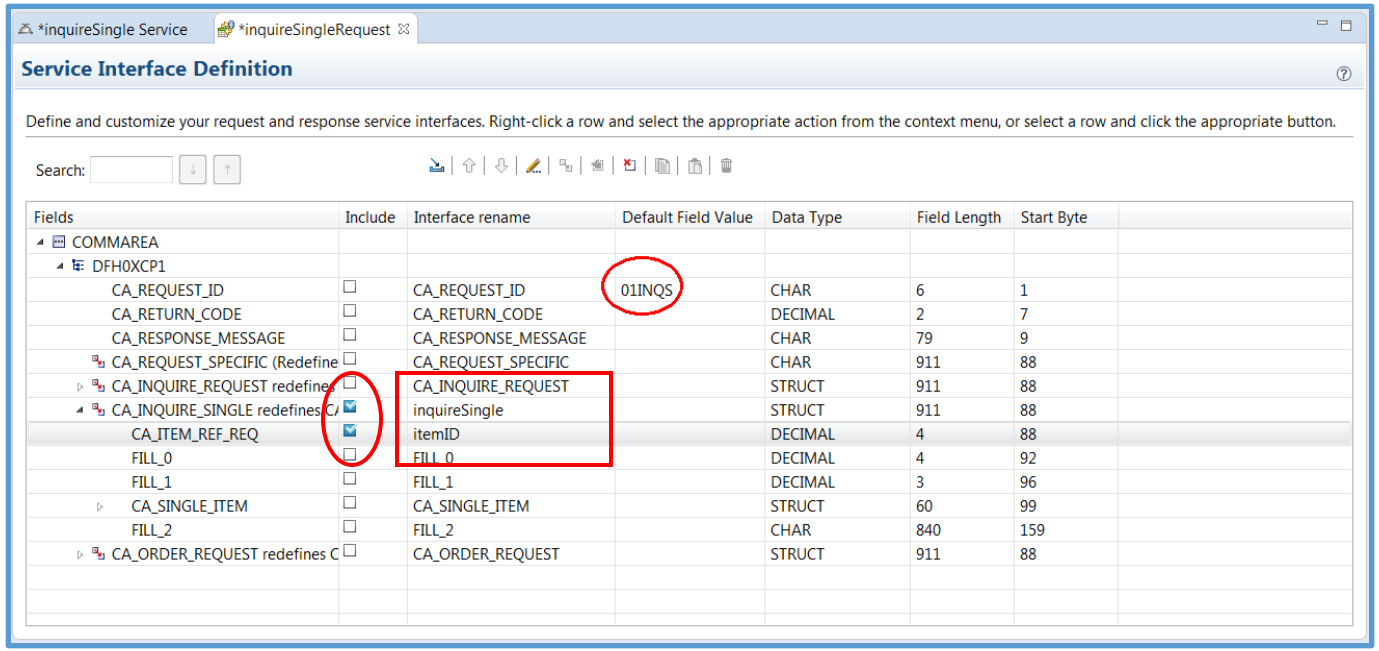
mitchj@us.ibm.com

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

The *service interface editor* allows creating and editing of a request service interface definition

```
*---------------------------------------------------------------*
* Check which operation in being requested
*---------------------------------------------------------------*
* Uppercase the value passed in the Request Id field
     MOVE FUNCTION UPPER-CASE(CA-REQUEST-ID) TO CA-REQUEST-ID
     EVALUATE CA-REQUEST-ID
         WHEN '01INQC'
*        Call routine to perform for inquire
             PERFORM CATALOG-INQUIRE
         WHEN '01INQS'
*        Call routine to perform for inquire for single item
             PERFORM CATALOG-INQUIRE-SINGLE
         WHEN '01ORDR'
*        Call routine to place order
             PERFORM PLACE-ORDER
         WHEN OTHER
*        Request is not recognised or supported
             PERFORM REQUEST-NOT-RECOGNISED
     END-EVALUATE
```

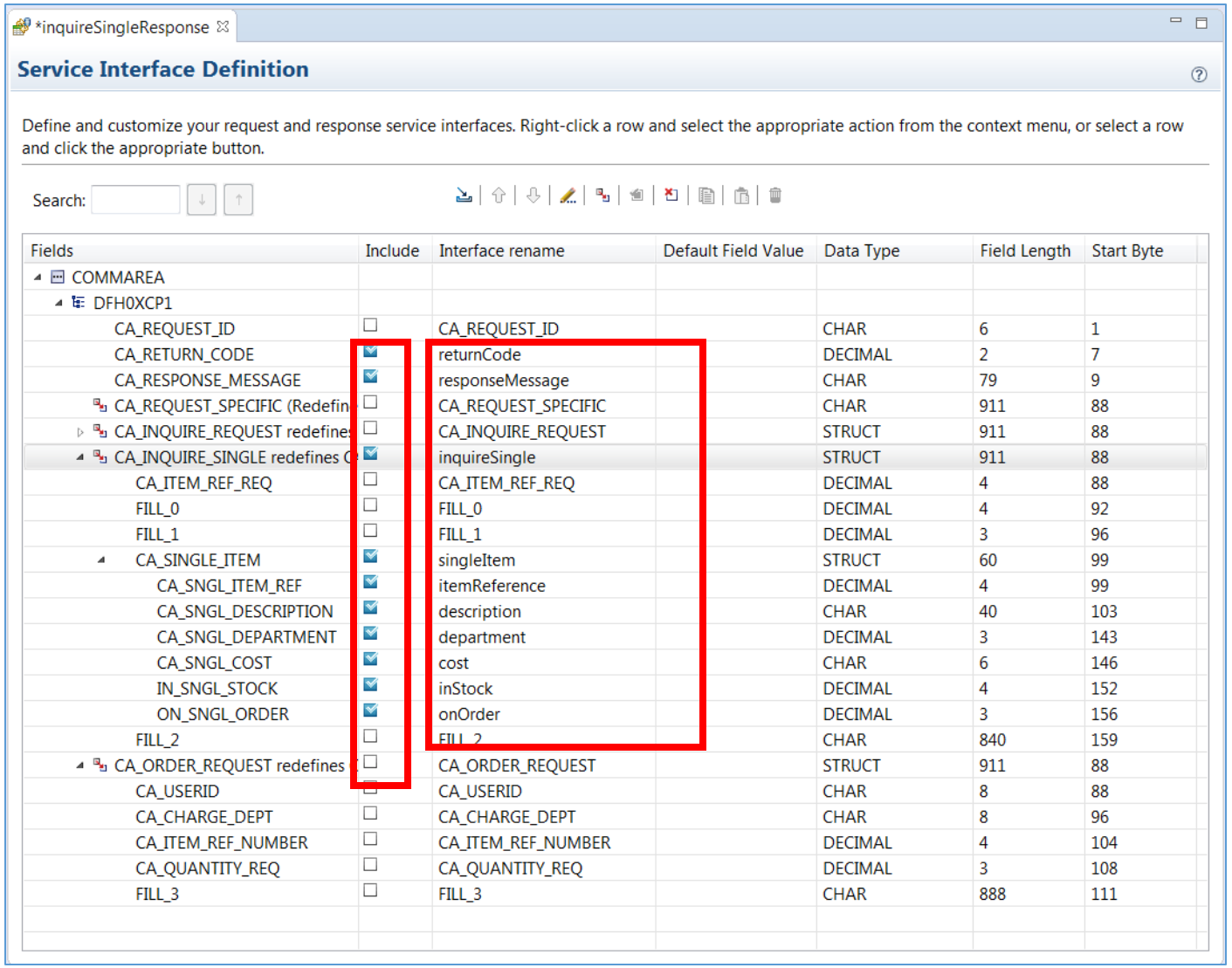
See the imported data structure and then can **redact fields**, **rename fields**, and **add default values to fields** to make the service more consumable for an API developer.



mitchj@us.ibm.com

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

And creating and editing of a response message service interface definition

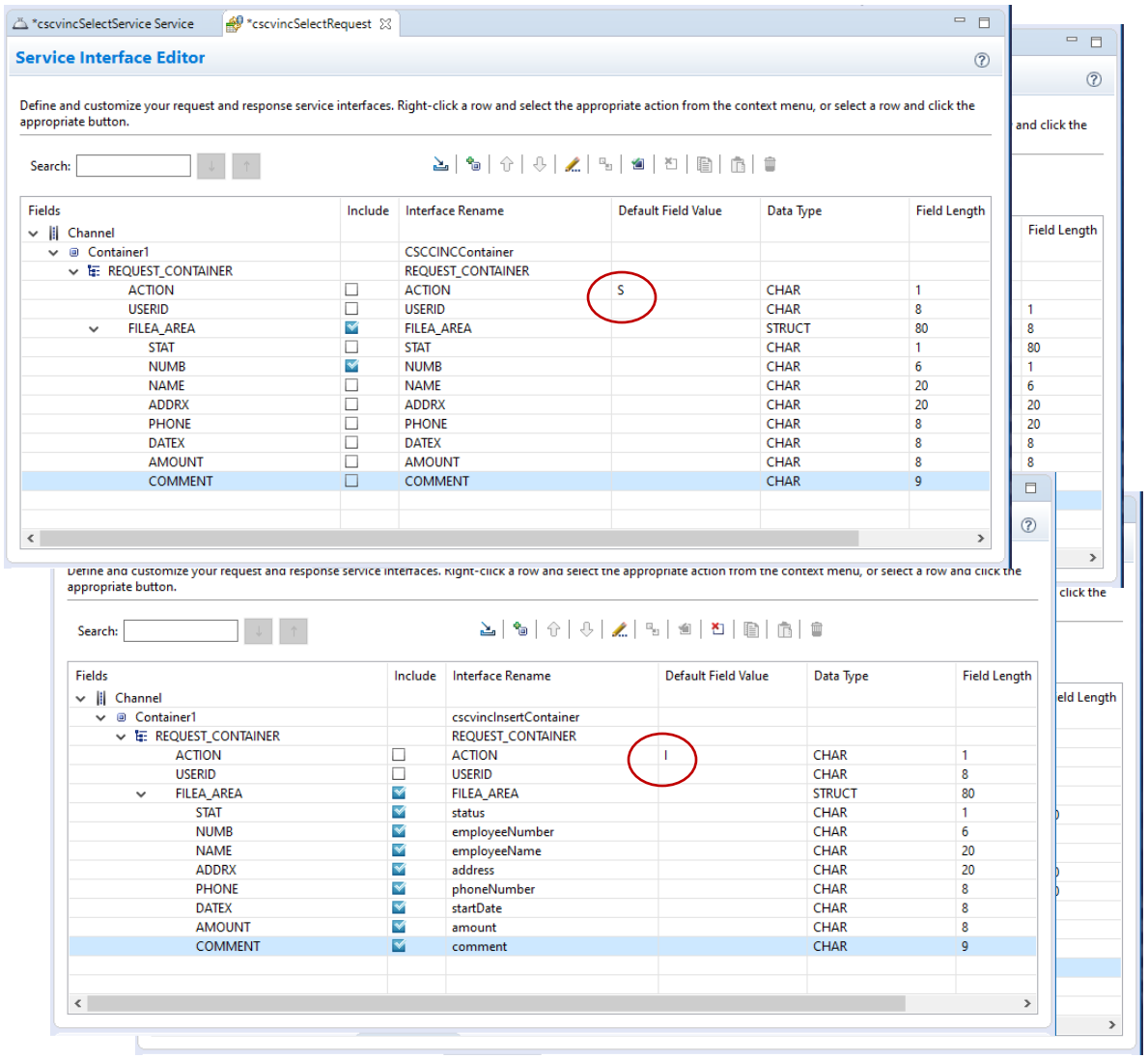


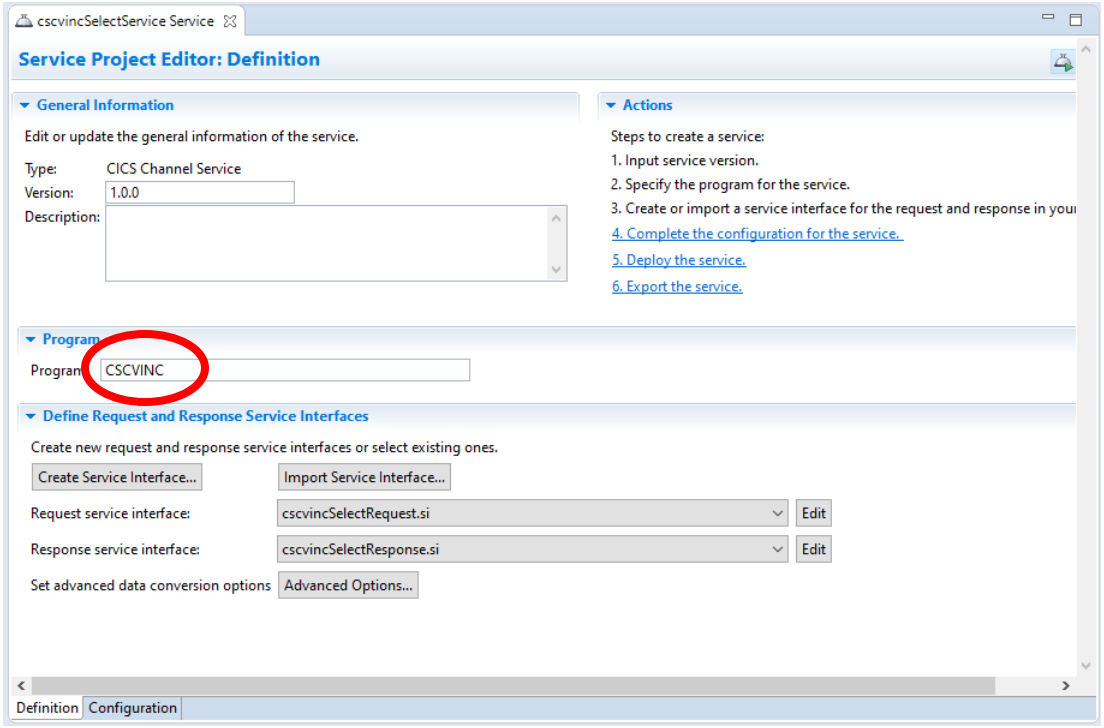
See the imported data structure and can **redact fields** and **rename fields**

mitchj@us.ibm.com

API toolkit – Creating Services for CICS

You can create multiple services definitions to the same resource





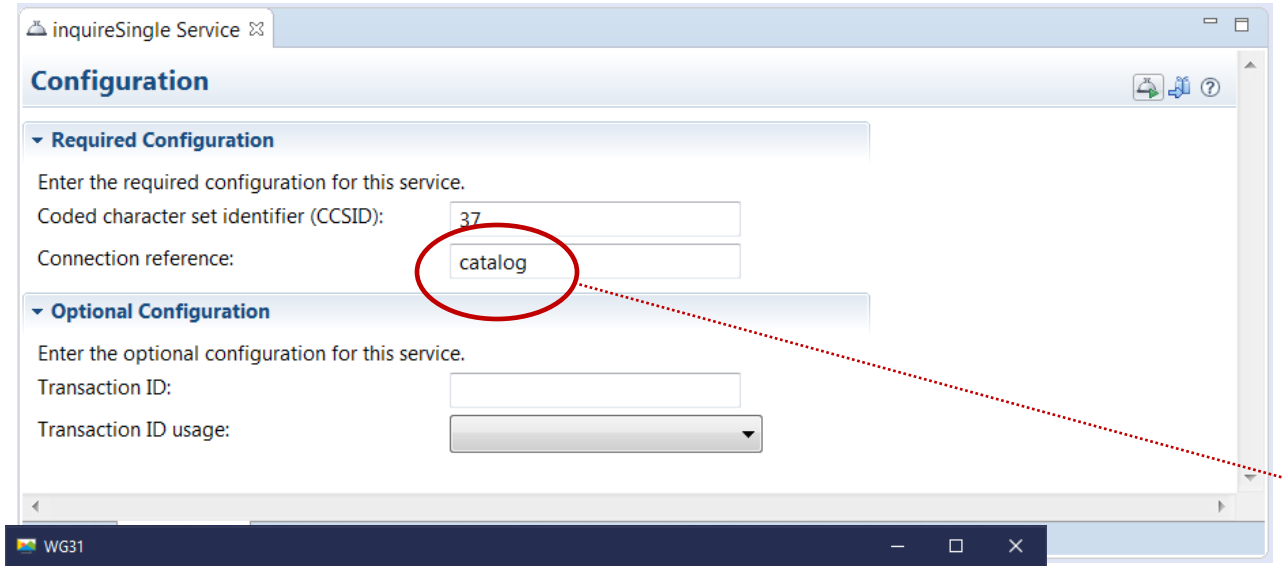

```
EVALUATE ACTION of Request-Container
    WHEN 'D'
      PERFORM Delete-Record
    WHEN 'I'
      PERFORM Insert-Record
    WHEN 'U'
      PERFORM Update-Record
    WHEN 'S'
      PERFORM Select-Record
 END-EVALUATE.
```

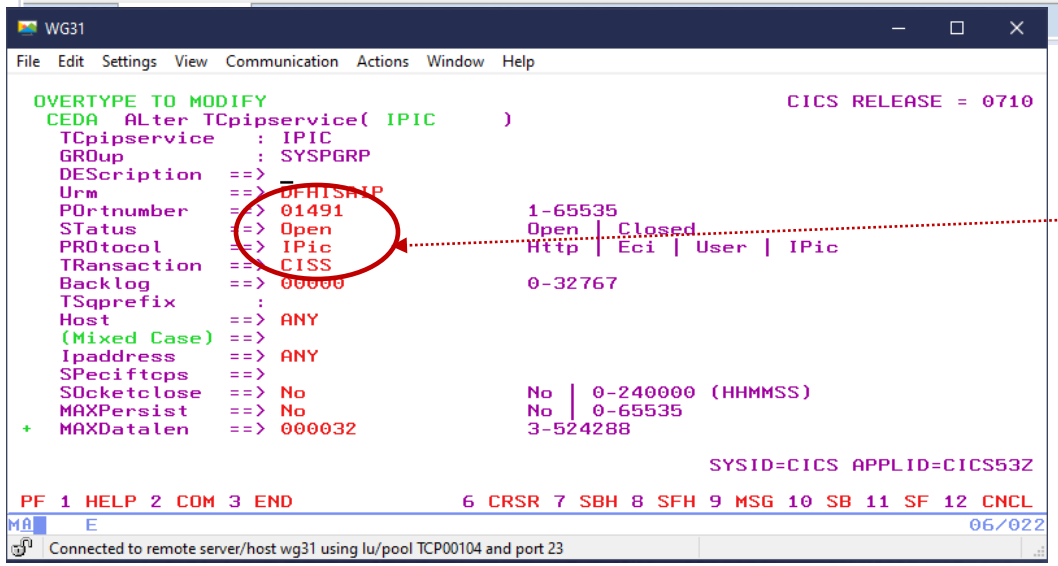
mitchj@us.ibm.com

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

Accessing CICS uses a connection reference that uses CICS IP interconnectivy (IPIC)

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

catalog.xml

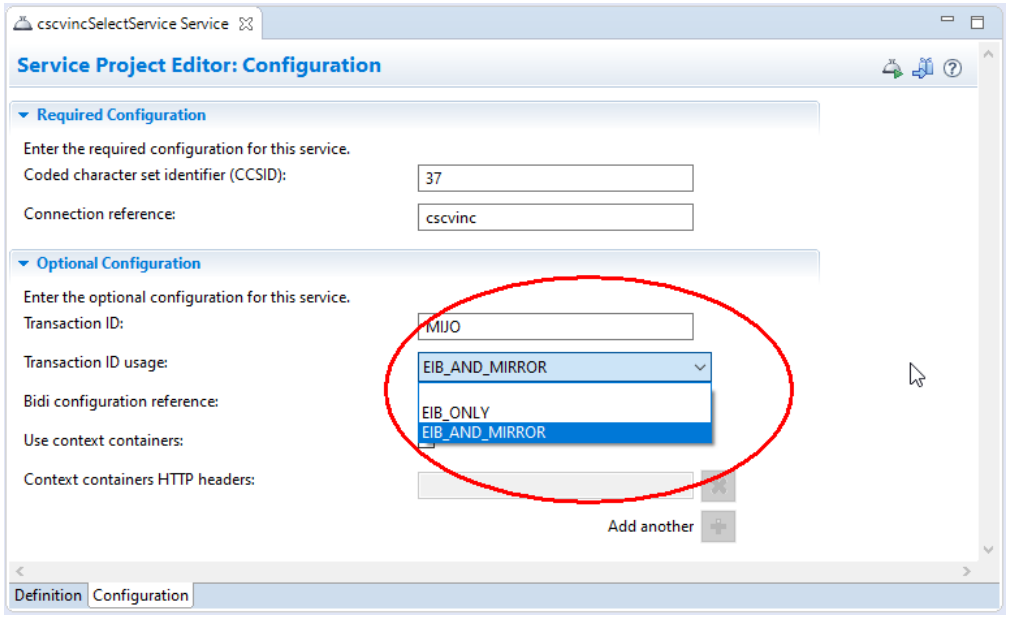
Design Source

```
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
```

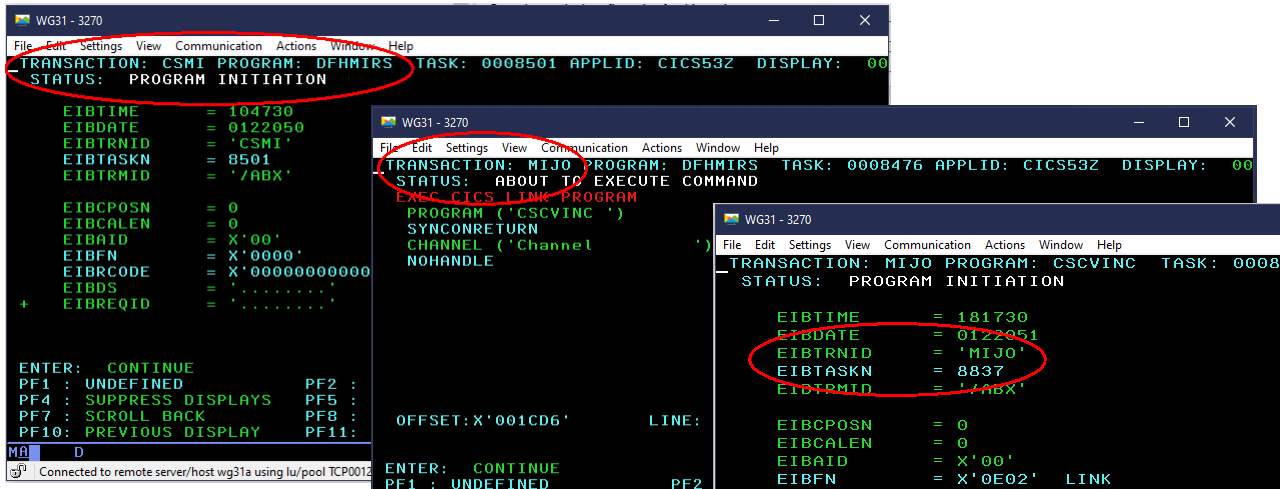
Define IPIC connection to CICS

mitchj@us.ibm.com

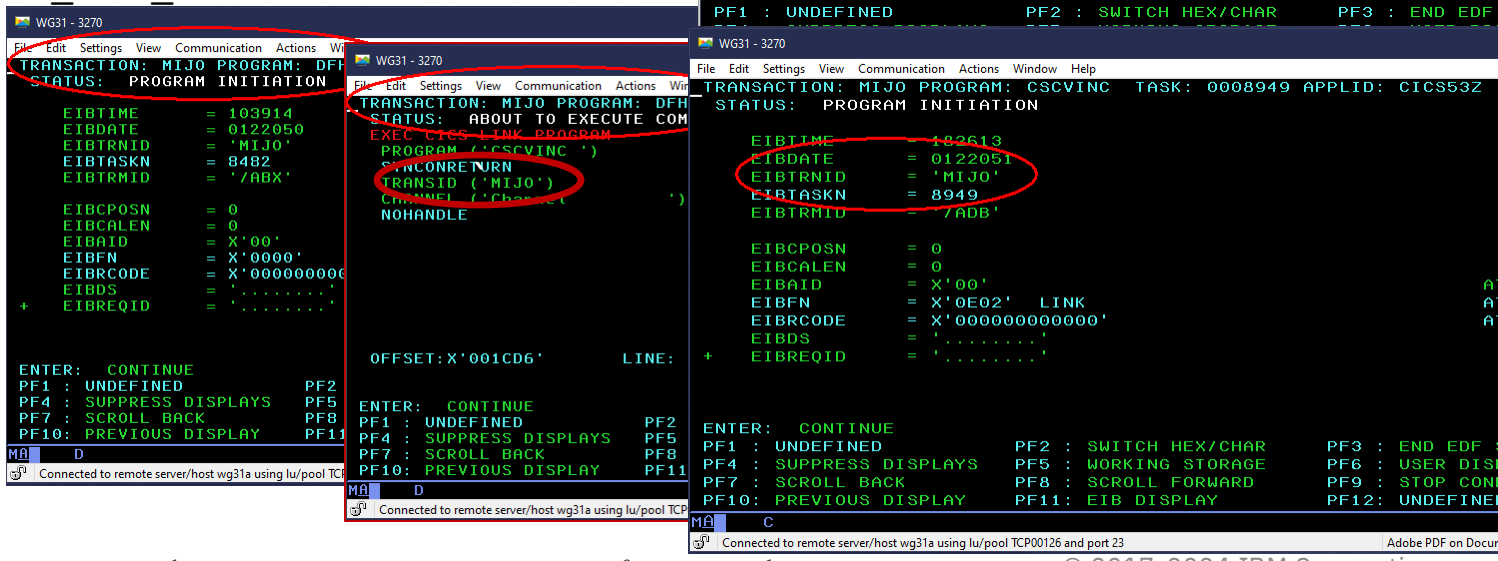
Tech-Tip : Accessing a CICS program – Transaction ID Usage



EIB_ONLY



EIB_AND_MIRROR



- **Transaction ID** attaches a CICS transaction (CSMI is the default) that starts the CICS DFHMIRS program.

- **Transaction ID Usage** attribute useful for:
  - Transaction security requirements
  - Db2 plan selection
  - Transaction classification and reporting

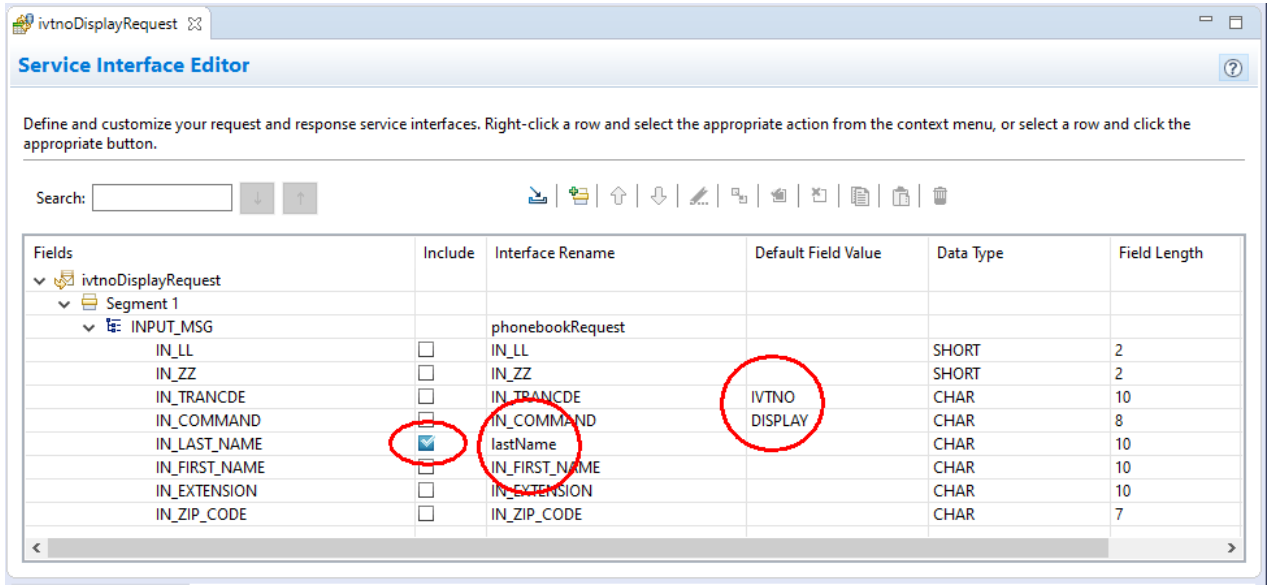
These attributes also be used in *zosconnect_cicsIpicConnection* and *zosconnect_services>service configuration* elements.

mitchj@us.ibm.com

API toolkit – Creating Services for IMS

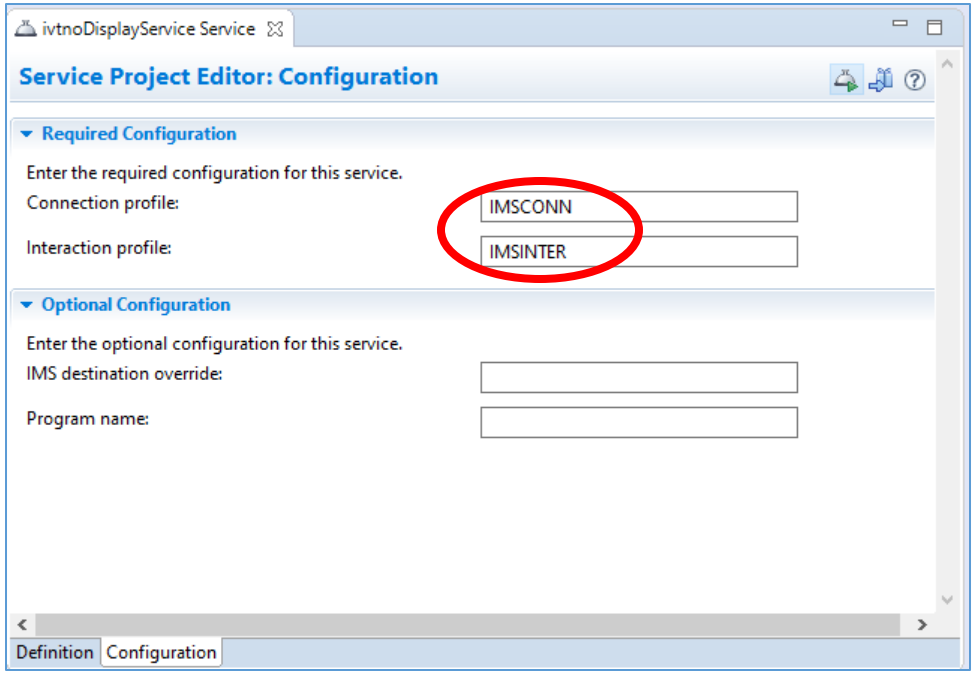
Creating a “GET” service interface request definition

```
*-------------------------------------------------------------------*
*       ROUTE TO REQUEST HANDLER                                    *
*-------------------------------------------------------------------*
          SPACE 1
          CLC    KADD,IOCMD      IF COMMAND ADD ENTERED ?
          BE     TOADD           ...THEN, GOTO INSERT ENTRY
          CLC    KUPD,IOCMD      IF COMMAND UPDATE ENTERED ?
          BE     TOUPD           ...THEN, GOTO UPDATE ENTRY
          CLC    KDEL,IOCMD      IF COMMAND DEL ENTERED ?
          BE     TODEL           ...THEN, GOTO DELETE ENTRY
          CLC    KDIS,IOCMD      IF COMMAND DIS ENTERED ?
          BE     TODIS           ...THEN, GOTO DISPLAY ENTRY
          CLC    KTAD,IOCMD      IF TEST ADD WITH REPLY ?
          BE     TOTAD           ...THEN,
          B      INVREQ1         INVALID REQUEST
```

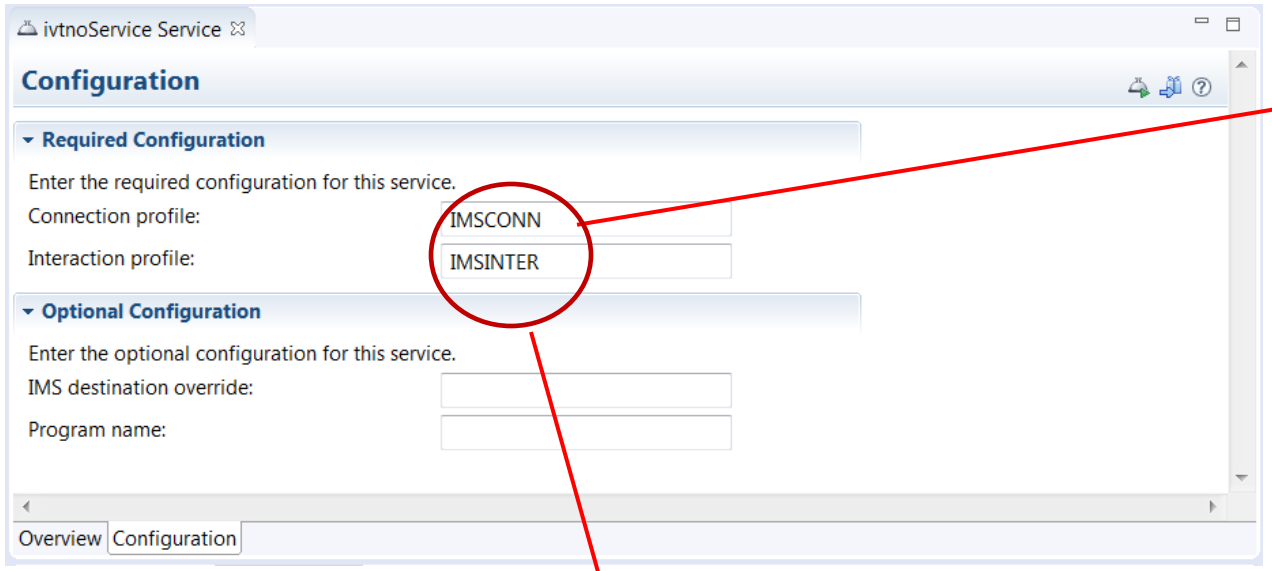


The service developer creates distinct services for each function.

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



IMS Connections and Interactions



Connection Profile

```
<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 Profile

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

IMS Connect HWSCFG

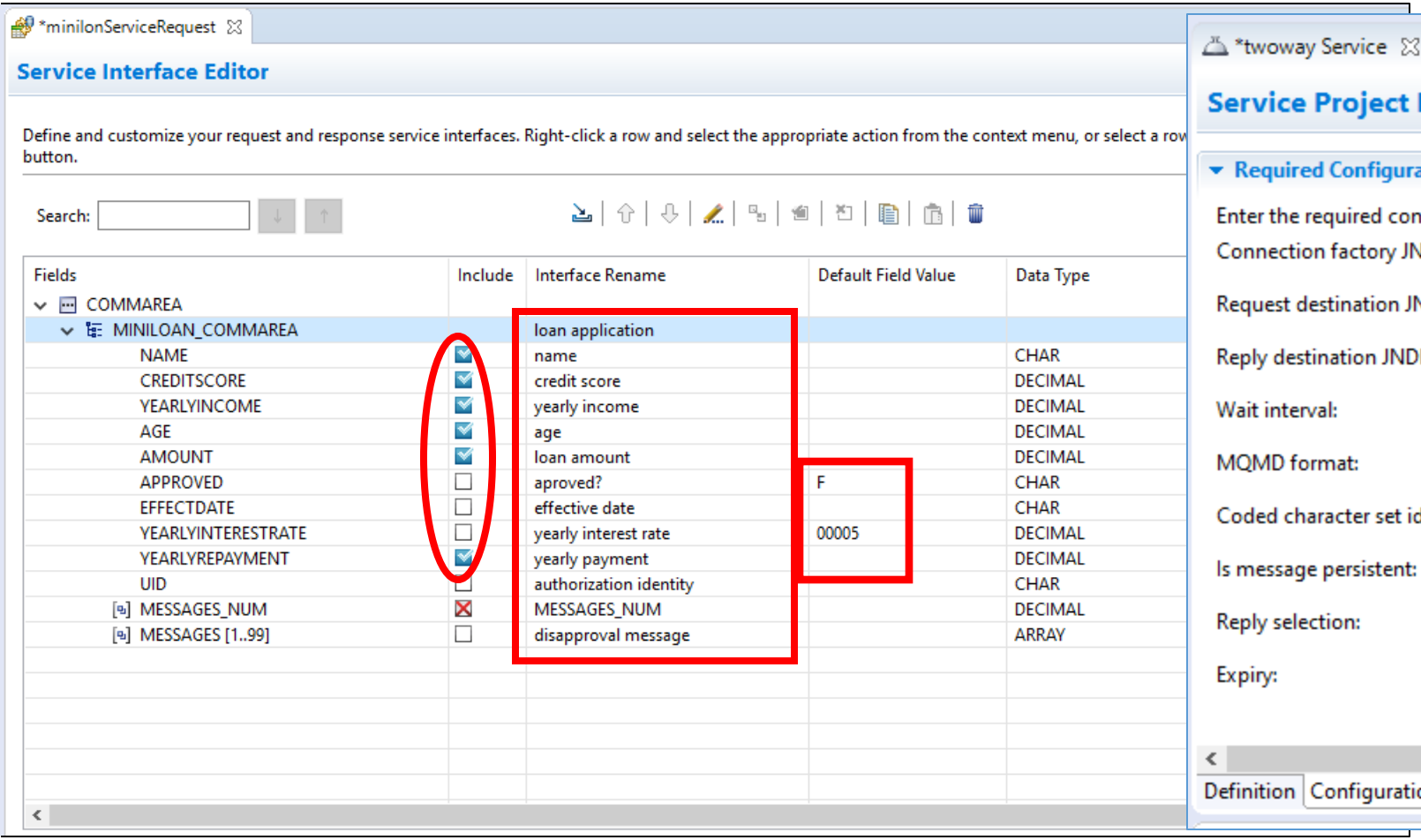
```
HWS=(ID=IMS14HWS,XIBAREA=100,RACF=Y,RRS=N)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,
TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,TMEMBER=OT
MAMEM)
```

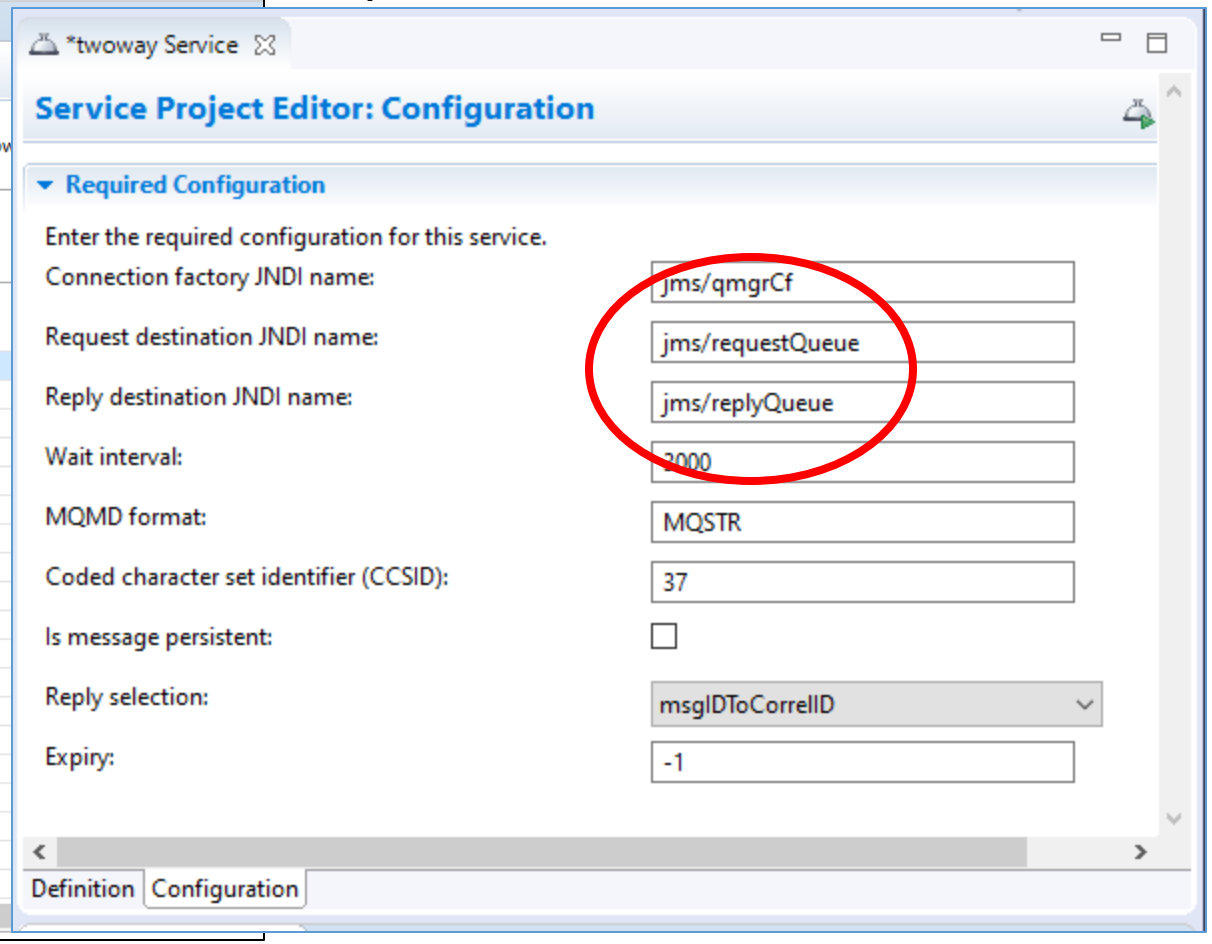
mitchj@us.ibm.com

API toolkit – Creating Services for MQ

Creating a MQ PUT (“POST”) service interface definition

MQ Meta Data

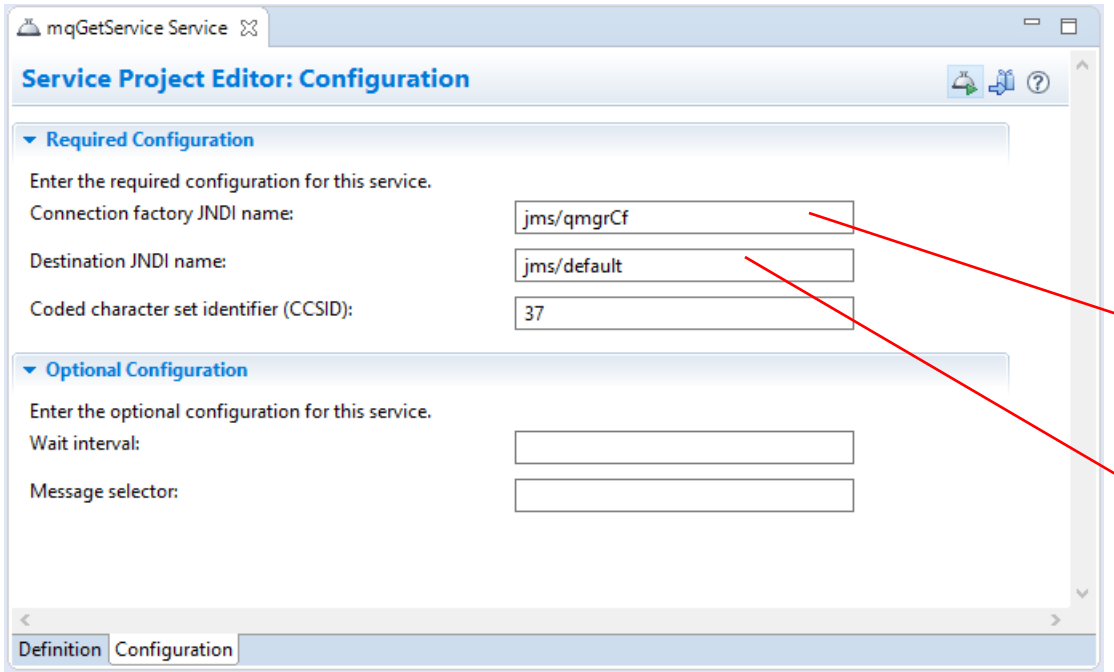




Again the service developer can then see the imported data structure and can **redact fields**, **rename fields**, and **add default values to fields** to make the service more consumable for an API developer.

mitchj@us.ibm.com

Using JMS to access MQ (One-Way)



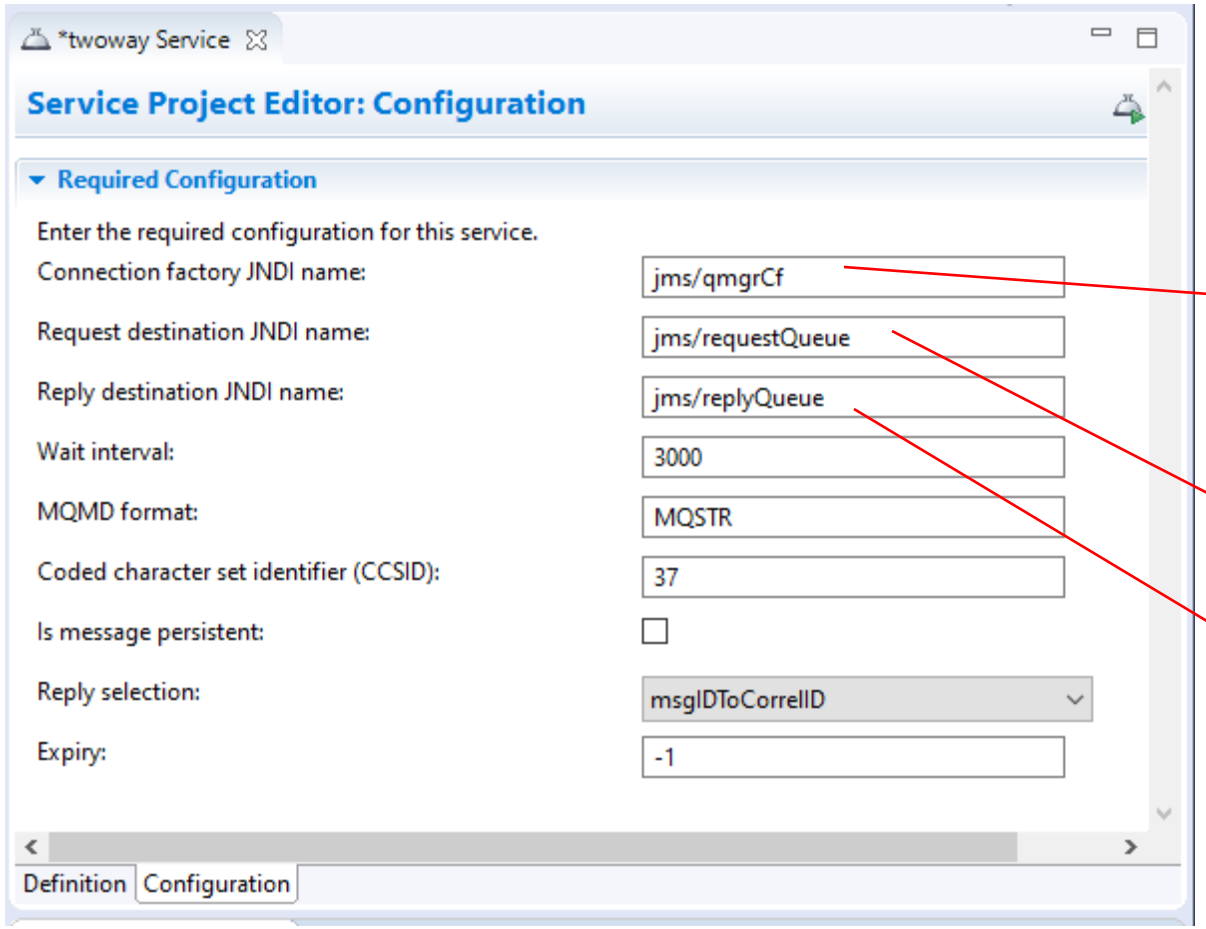
mqClient.xml — Read only — Close

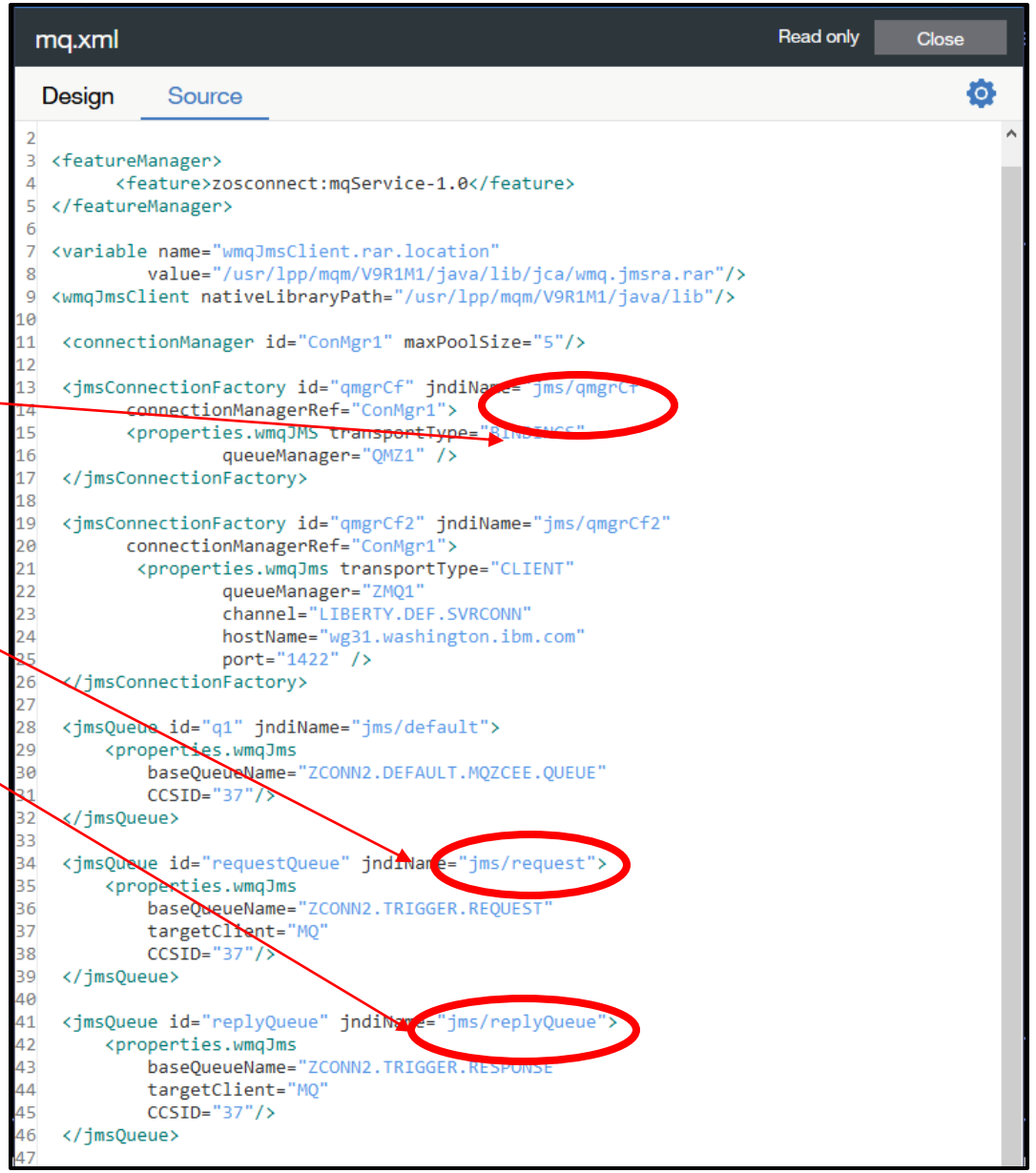
Design | Source

```
1  <server description="MQ Service Provider">
2
3   <featureManager>
4         <feature>zosconnect:mqService-1.0</feature>
5   </featureManager>
6
7   <variable name="wmqJmsClient.rar.location"
8             value="/usr/lpp/mqm/V9R1M1/java/lib/jca/wmq.jmsra.rar"/>
9   <wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R1M1/java/lib"/>
10
11  <zosconnect_services>
12        <service name="mqPutService">
13              <property name="useCallerPrincipal" value="false"/>
14        </service>
15  </zosconnect_services>
16
17   <connectionManager id="ConMgr1" maxPoolSize="5"/>
18
19   <jmsConnectionFactory id="qmgrCf" jndiName="jms/qmgrCf"
20         connectionManagerRef="ConMgr1">
21          <properties.wmqJMS transportType="CLIENT"
22                   queueManager="ZMQ1"
23                   channel="LIBERTY.DEF.SVRCONN"
24                   hostName="wg31.washington.ibm.com"
25                   port="1422" />
26   </jmsConnectionFactory>
27
28   <jmsQueue id="q1" jndiName="jms/default">
29       <properties.wmqJms
30           baseQueueName="ZCEE.DEFAULT.MQZCEE.QUEUE"
31           CCSID="37"/>
32   </jmsQueue>
33
34 </server>
35
```

mitchj@us.ibm.com

Using JMS to access MQ (Two-Way)

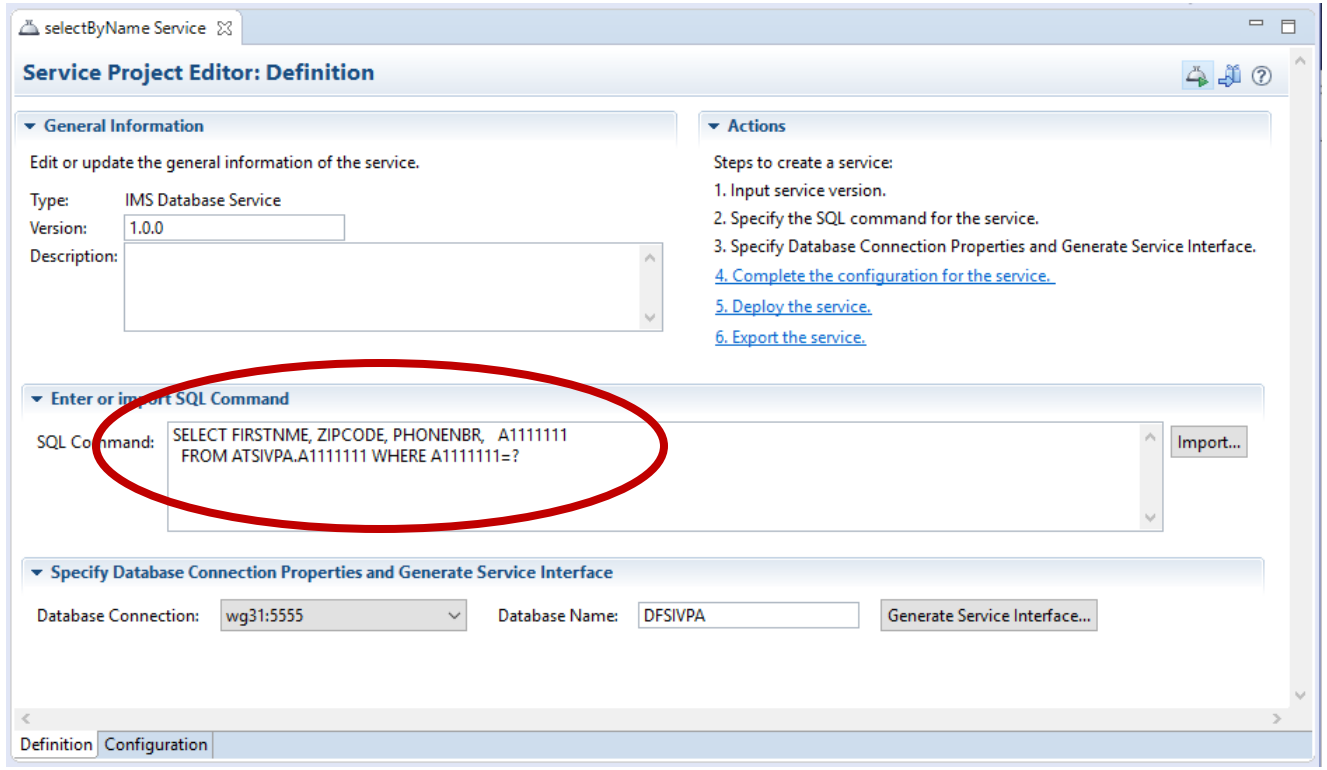




mitchj@us.ibm.com

API toolkit – Creating Services for IMS DB

Creating a service project from the IMS Catalog



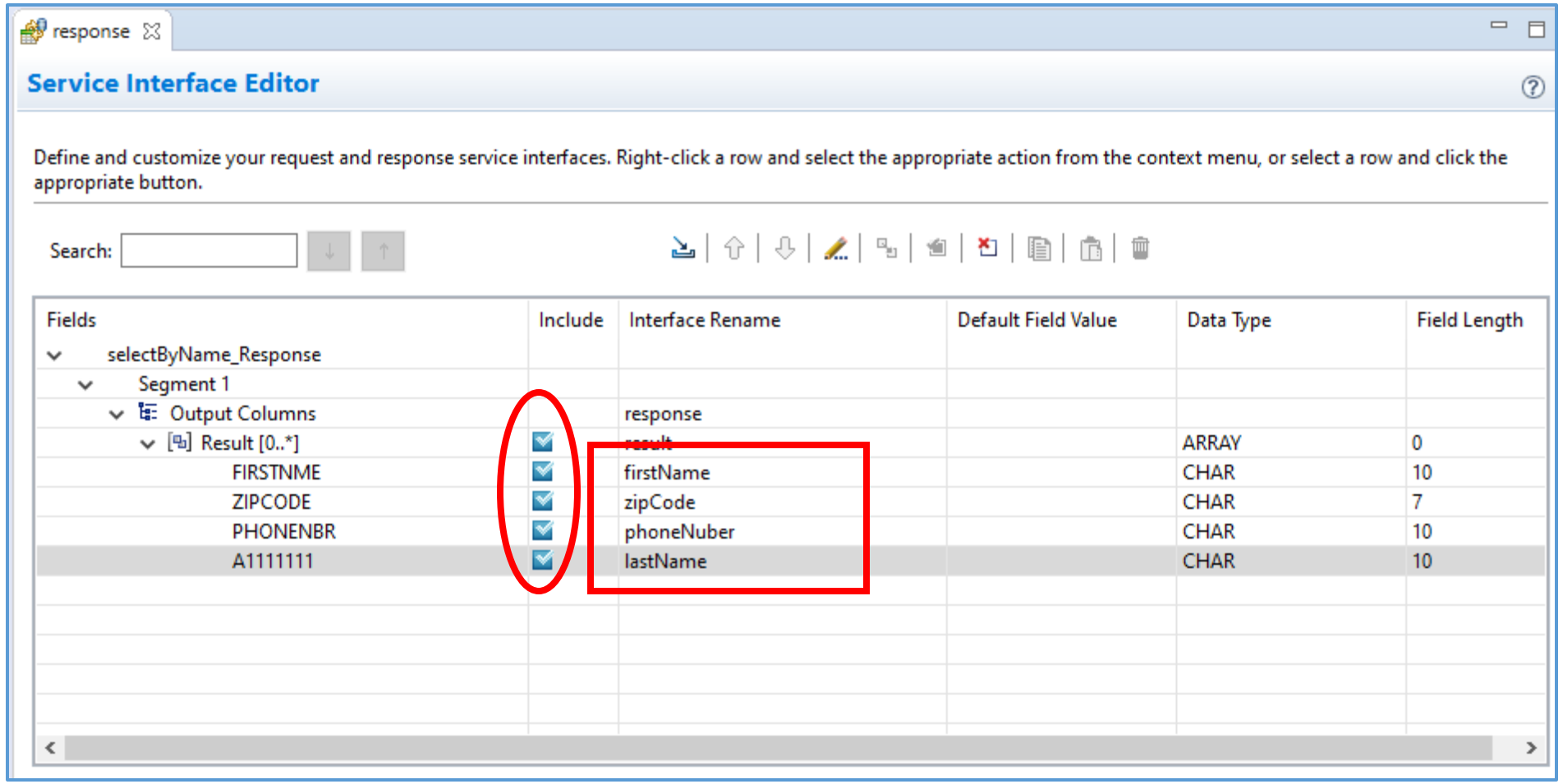
Use the IMS Catalog to assist with developing and testing SQL SELECT commands used for accessing IMS databases.

```
*-------------------------------------------------------------------*
*  SEGMENT DESCRIPTION                                              *
*    ROOT ONLY DATABASE                                             *
*      BYTES  1-10  LAST NAME (CHARACTER) - KEY                     *
*      BYTES  11-20 FIRST NAME (CHARACTER)                          *
*      BYTES  21-30 INTERNAL PHONE NUMBER (NUMERIC)                 *
*      BYTES  31-37 INTERNAL ZIP (CHARACTER)                        *
*      BYTES  38-40 RESERVED                                        *
*                                                                   *
*-------------------------------------------------------------------*
         DBD    NAME=IVPDB1,ACCESS=(HIDAM,OSAM)
         DATASET DD1=DFSIVD1,DEVICE=3380,SIZE=2048
         SEGM   NAME=A1111111,PARENT=0,BYTES=40,RULES=(LLV,LAST),      x
               PTR=(TB,CTR)
         FIELD  NAME=(A1111111,SEQ,U),BYTES=010,START=00001,TYPE=C
         FIELD  NAME=FIRSTNME,BYTES=010,START=00011,TYPE=C
         FIELD  NAME=PHONENBR,BYTES=010,START=00021,TYPE=C
         FIELD  NAME=ZIPCODE,BYTES=7,START=00031,TYPE=C
         LCHILD NAME=(A1,IVPDB1I),POINTER=INDX,RULES=LAST
         DBDGEN
         FINISH
         END
```

mitchj@us.ibm.com

API toolkit – Creating Services for IMS DB

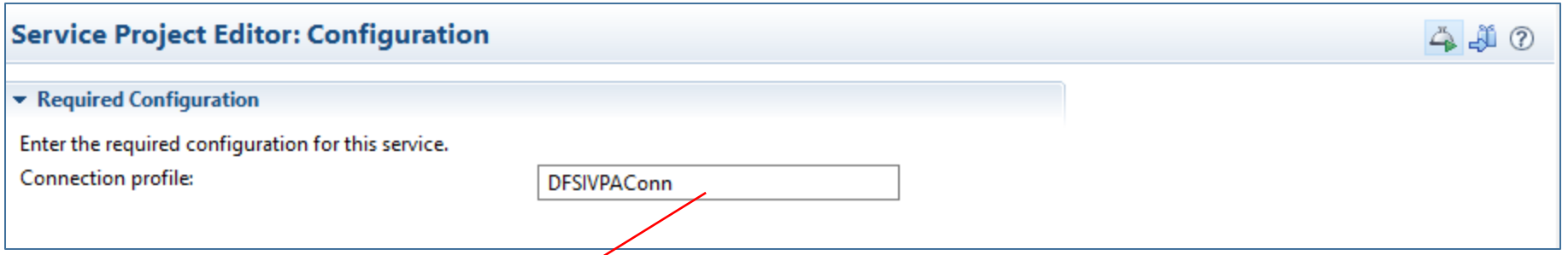
The Toolkit allows editing a service interface definitions for IM DB access*



*Using a slightly different process

mitchj@us.ibm.com

IMS Connection Factory in the server XML



ConnectionFactory

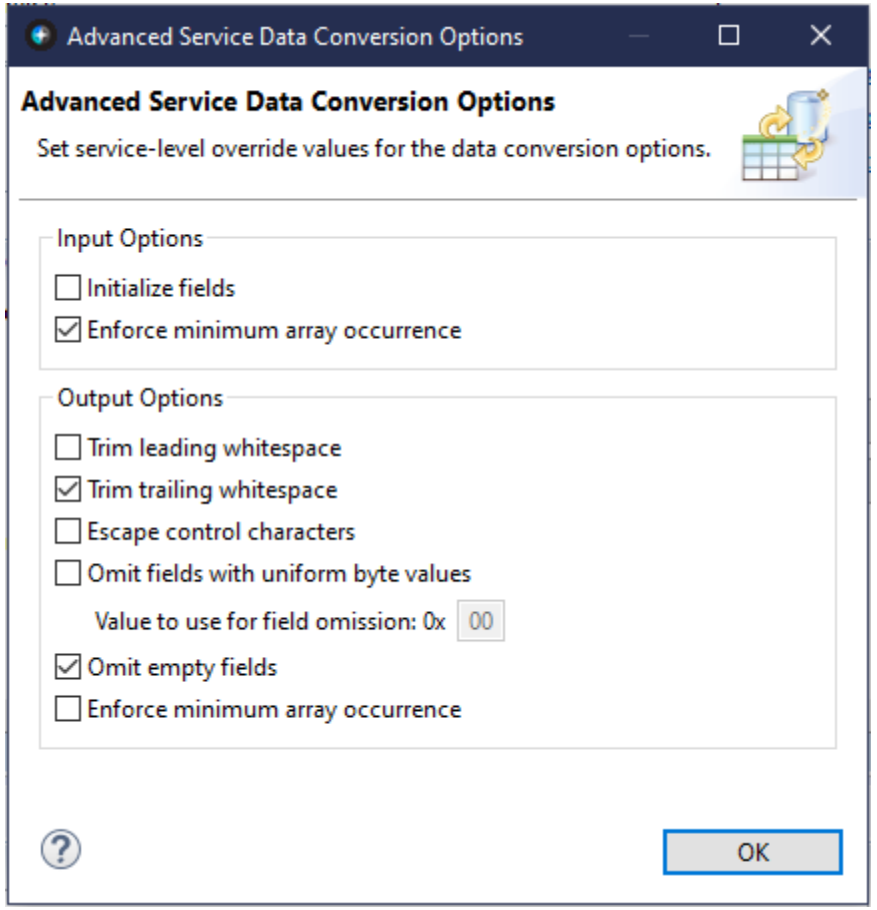
```
<connectionFactory id="DFSIVPAConn">
<properties.imsudbJLocal
   databaseName="DFSIVPA"
   datastoreName="IVP1"
   datastoreServer="wg31.washington.ibm.com"
   driverType="4"
   portNumber="5555"
   user="USER1"
   password="password"
   flattenTables="True"/>
</connectionFactory>
```

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)
```

mitchj@us.ibm.com

API toolkit – Advanced Data Conversion Options



Request Messages:

- Initialize fields
- Enforce minimum array occurrence

Response Messages:

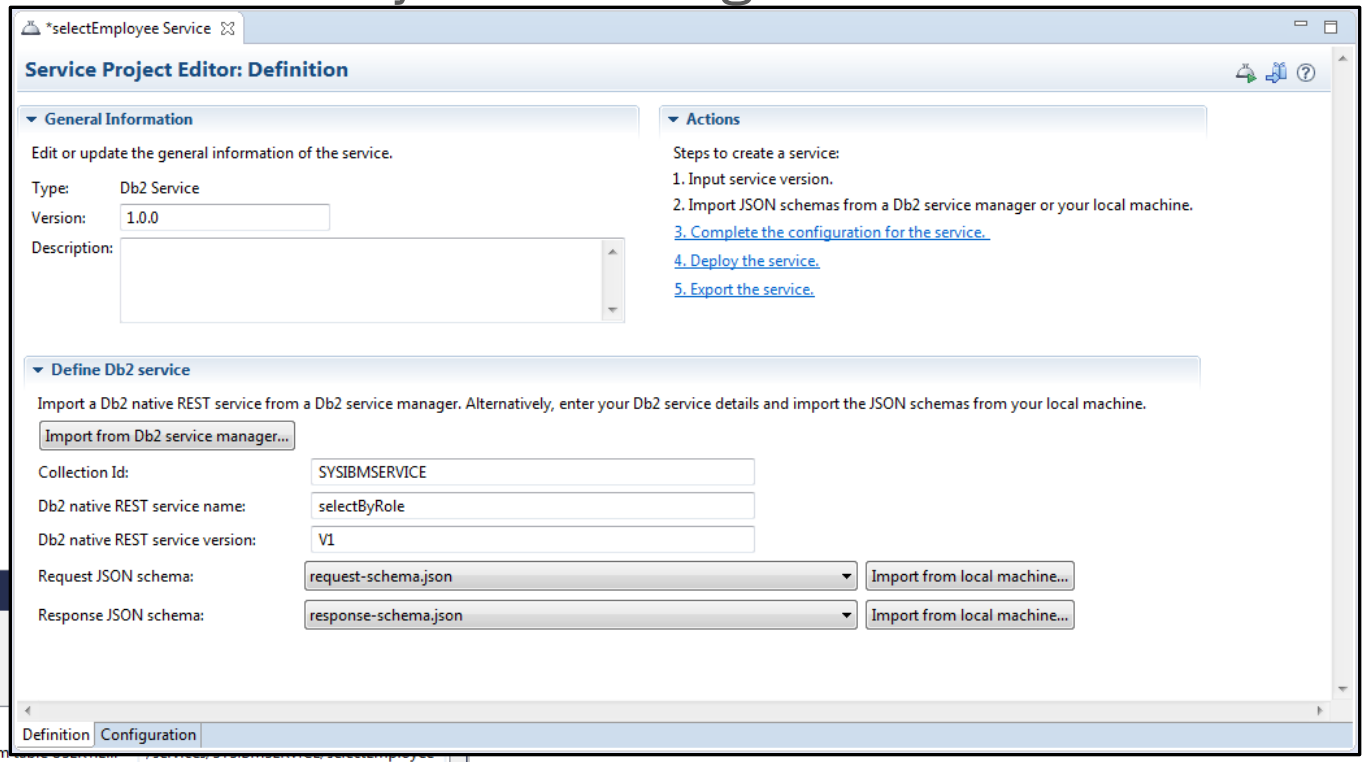
- Trim leading whitespace
- Trim trailing whitespace
- Escape control characters
- Omit fields with uniform byte values
- Omit empty fields
- Enforce minimum array occurrence

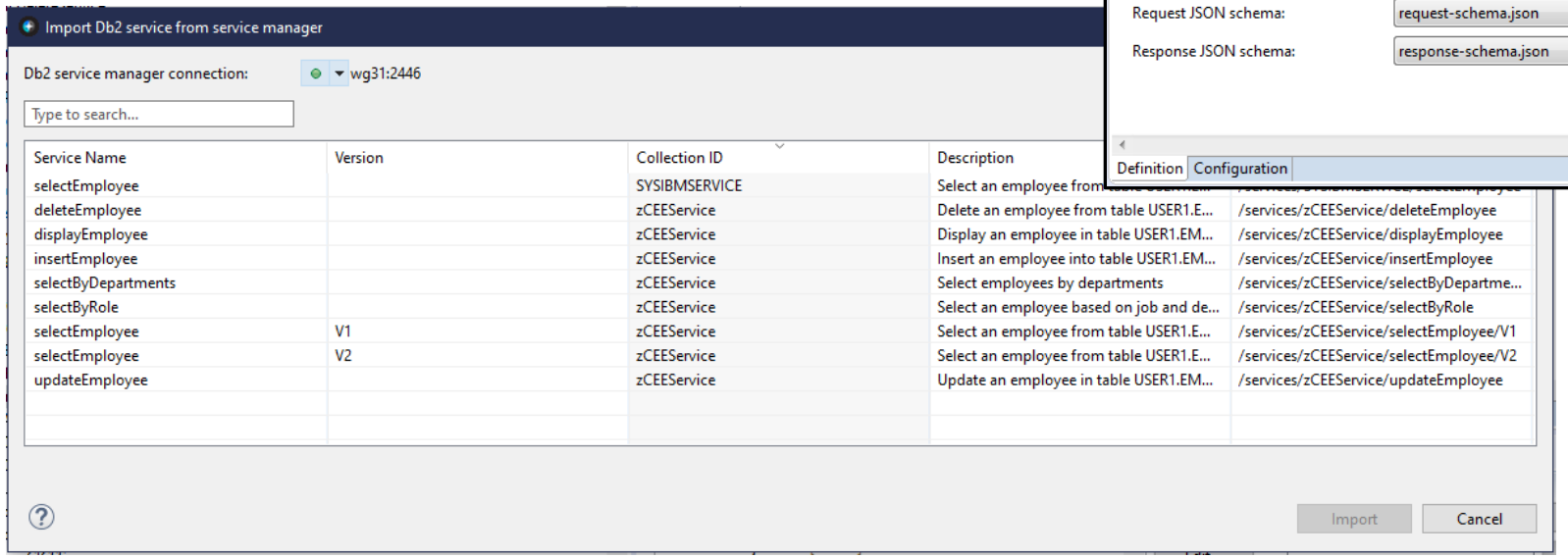
mitchj@us.ibm.com

API toolkit – Creating Services for Db2

Creating a service project from Db2 REST service by connecting to Db2

```
//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')
```



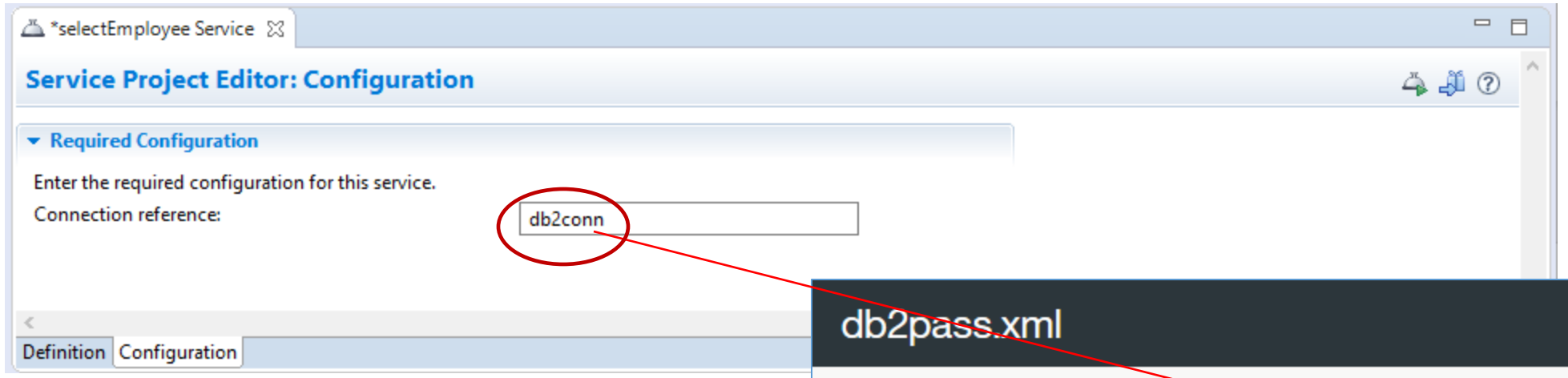


The service developer retrieves details about the Db2 REST services

Note there is no service interface editor available

mitchj@us.ibm.com

Accessing a Db2 REST service resource




```
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

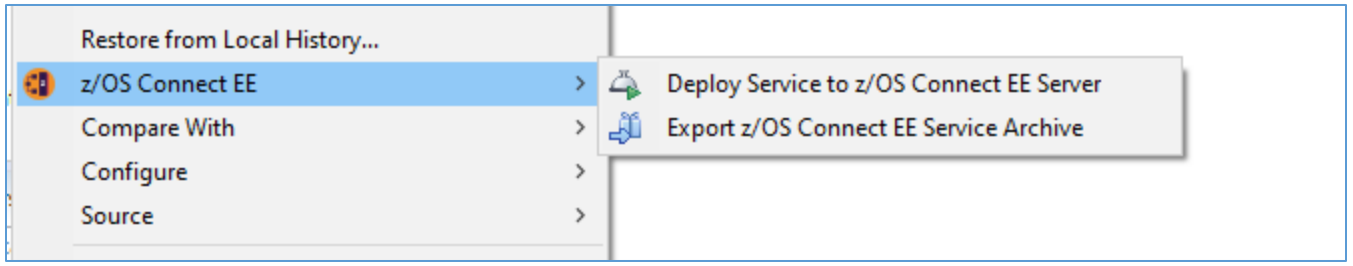
```
1  <server description="DB2 REST">
2
3   <zosconnect_zosConnectServiceRestClientConnection id="db2conn"
4         host="wg31.washington.ibm.com"
5         port="2446"
6         basicAuthRef="dsn2Auth" />
7
8   <zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
9         applName="DSN2APPL"/>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

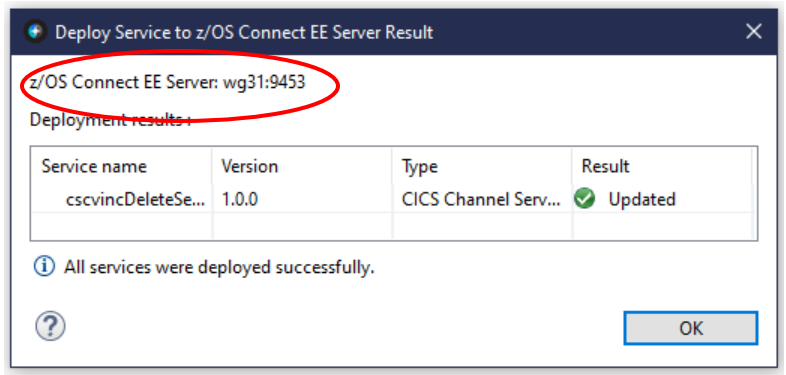
mitchj@us.ibm.com

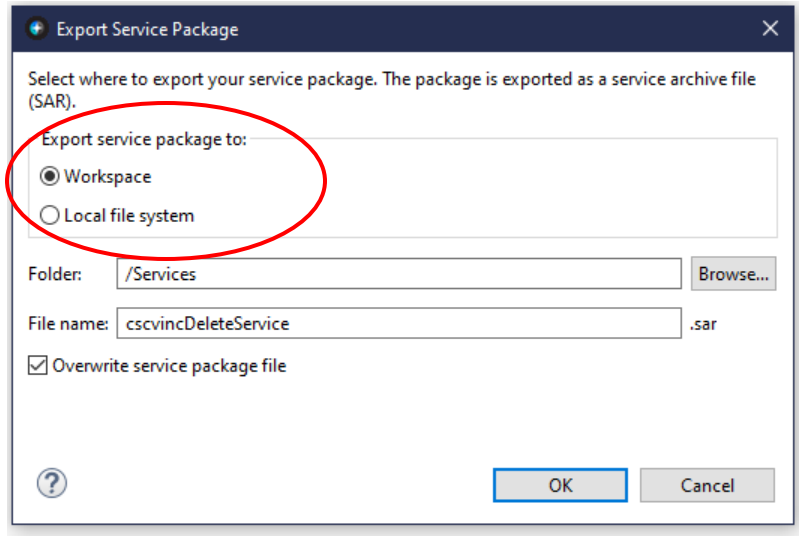
API toolkit – Services Editor

All services now be deployed to the server or exported for API development

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







mitchj@us.ibm.com

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

Creating APIs using the Eclipse Tooling

Remember: All service archives files are functionally equivalent regardless of how there are created or which resources they are intended to access

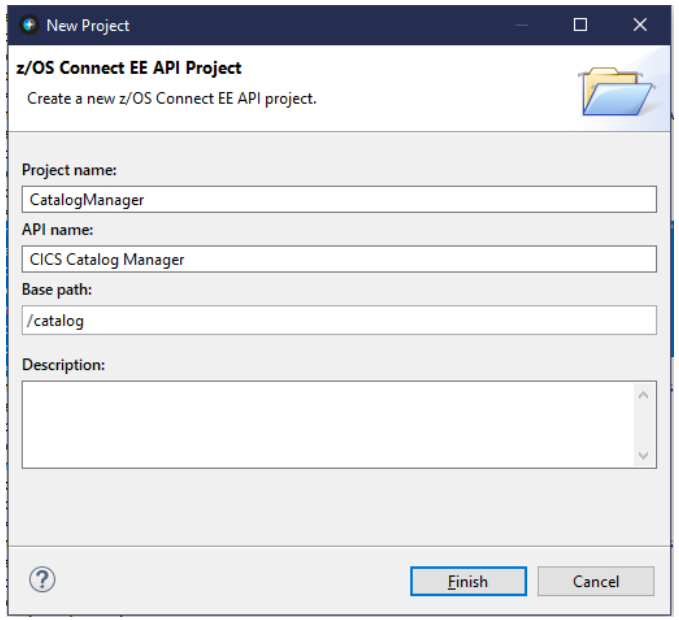
mitchj@us.ibm.com

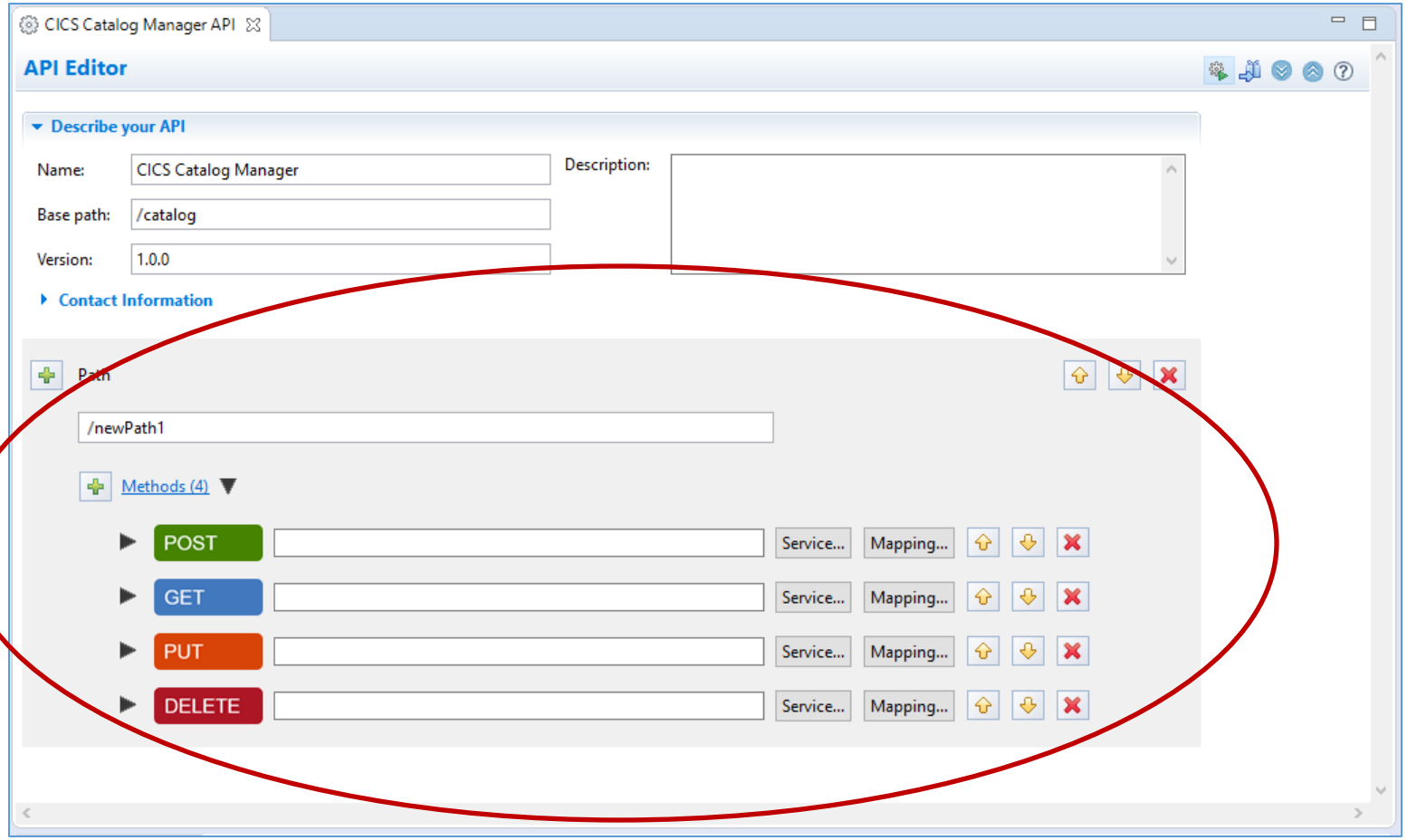
/api_toolkit/api_editor

Quick and easy **API mapping.**

mitchj@us.ibm.com

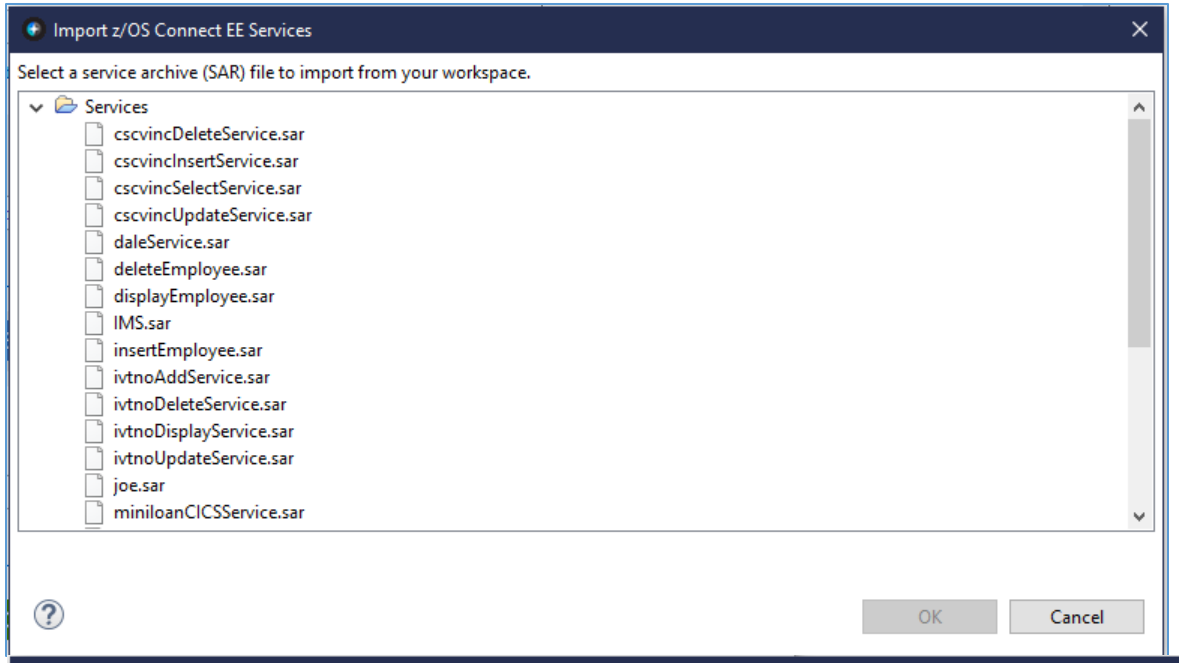
API toolkit – API Editor

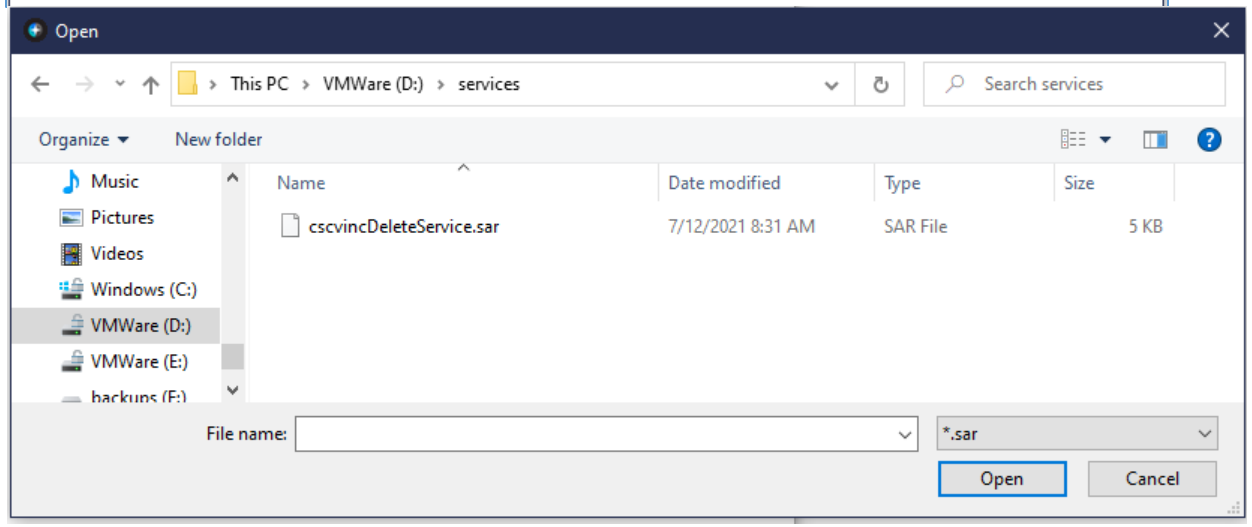


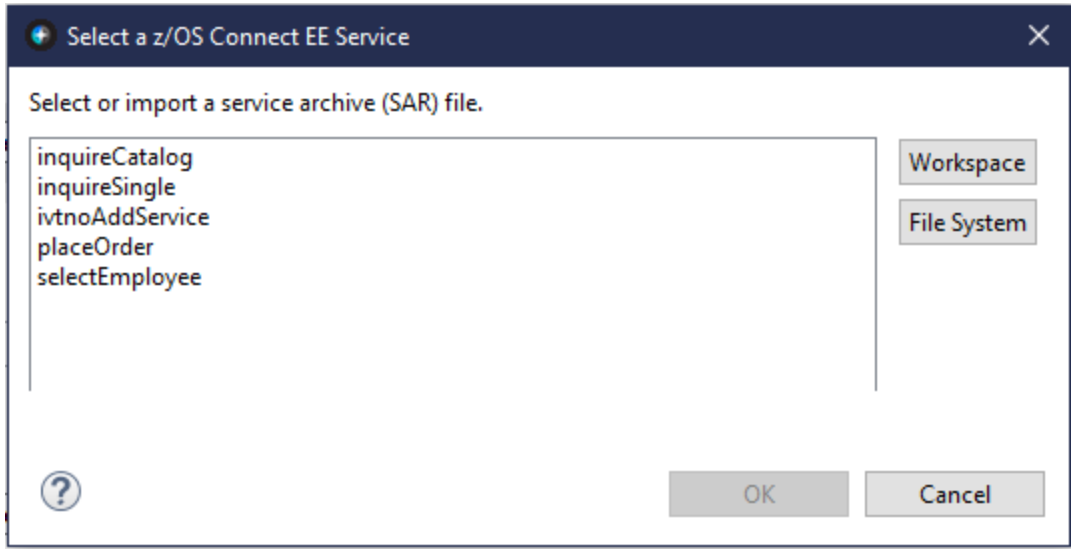


mitchj@us.ibm.com

Importing the service archive file from a filesystem or an Eclipse project

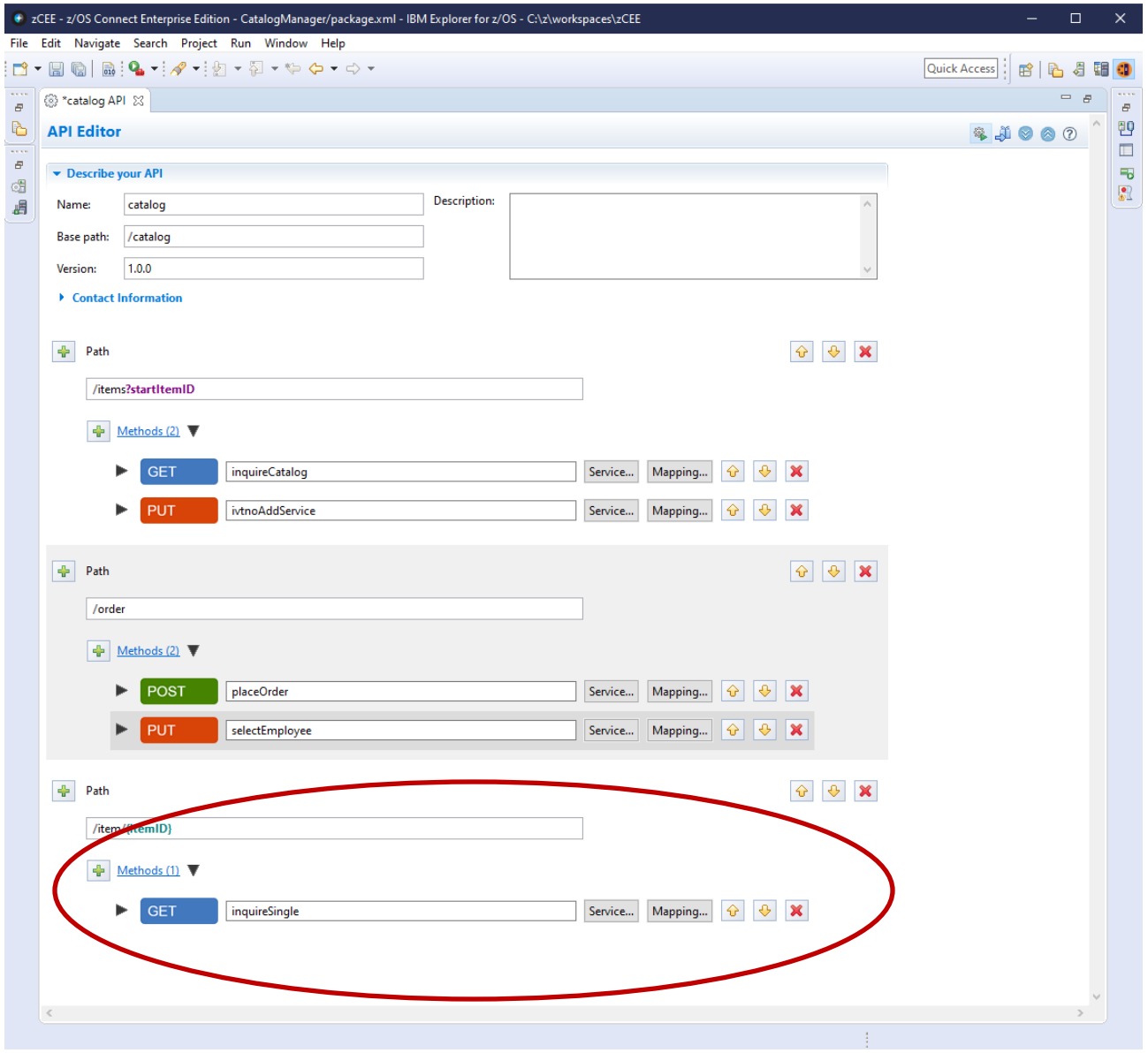






mitchj@us.ibm.com

API toolkit – API Editor



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

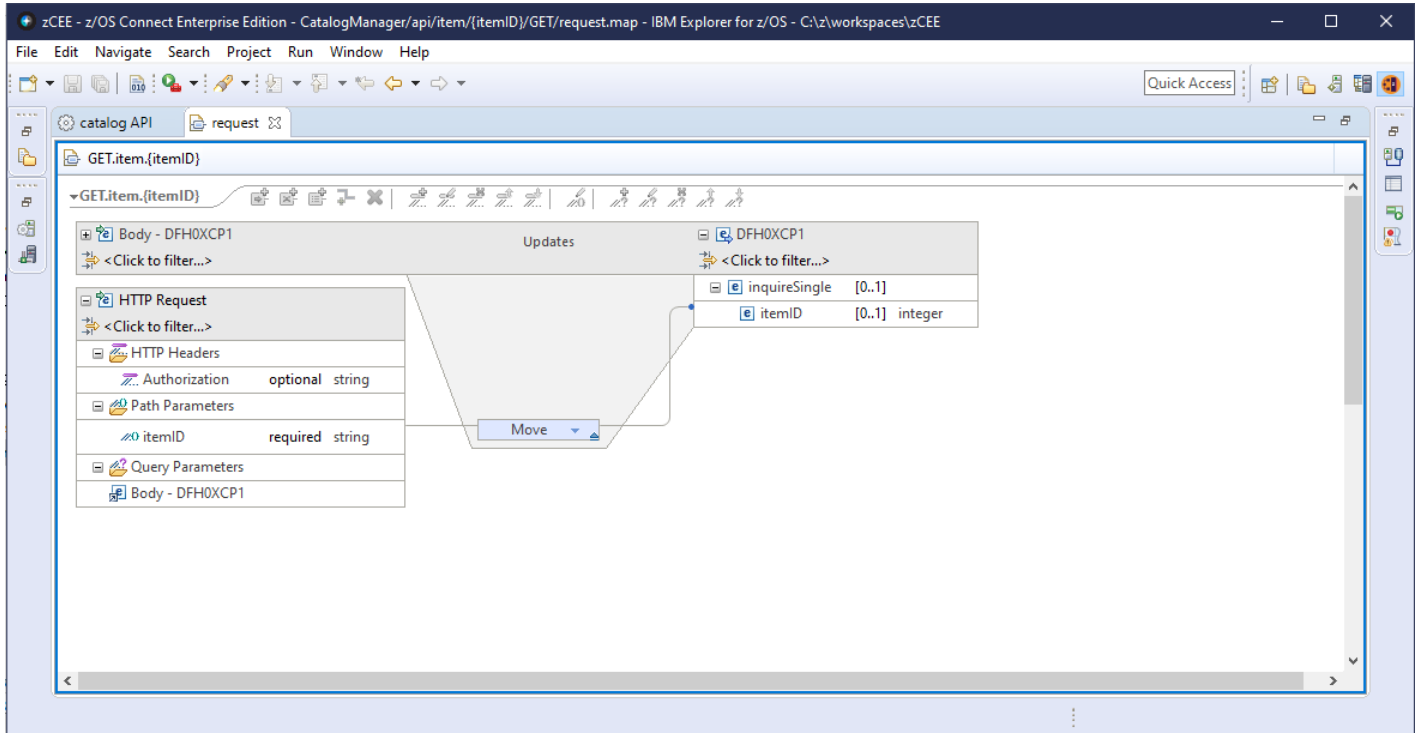
Once you define your API, you can map backend services to each request.

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

mitchj@us.ibm.com

API toolkit – API Editor

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



Map both the request and response for each API.

Map path and query parameters to native data structures.

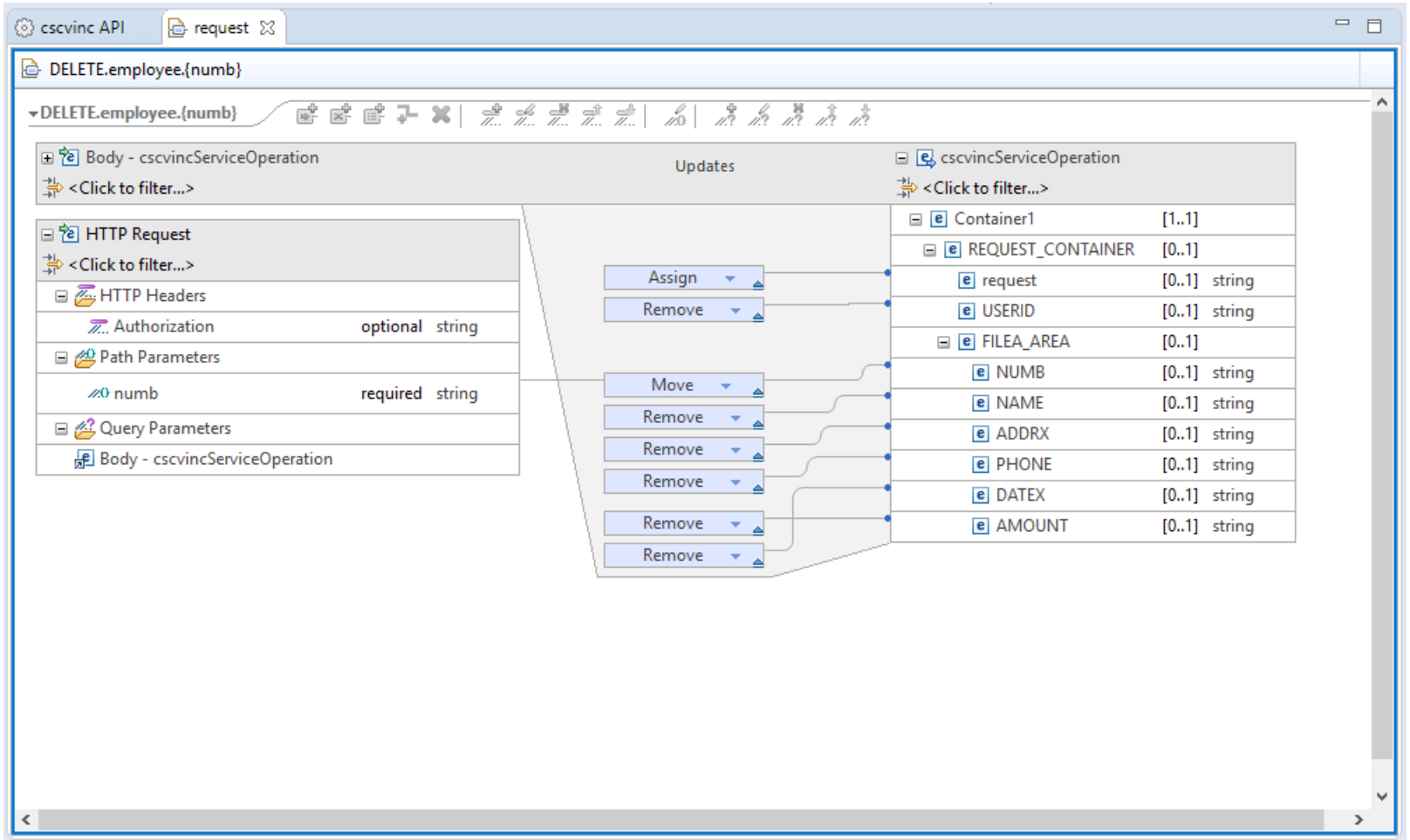
Assign static values to fields, useful for Op codes.

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

mitchj@us.ibm.com

API toolkit – API Editor

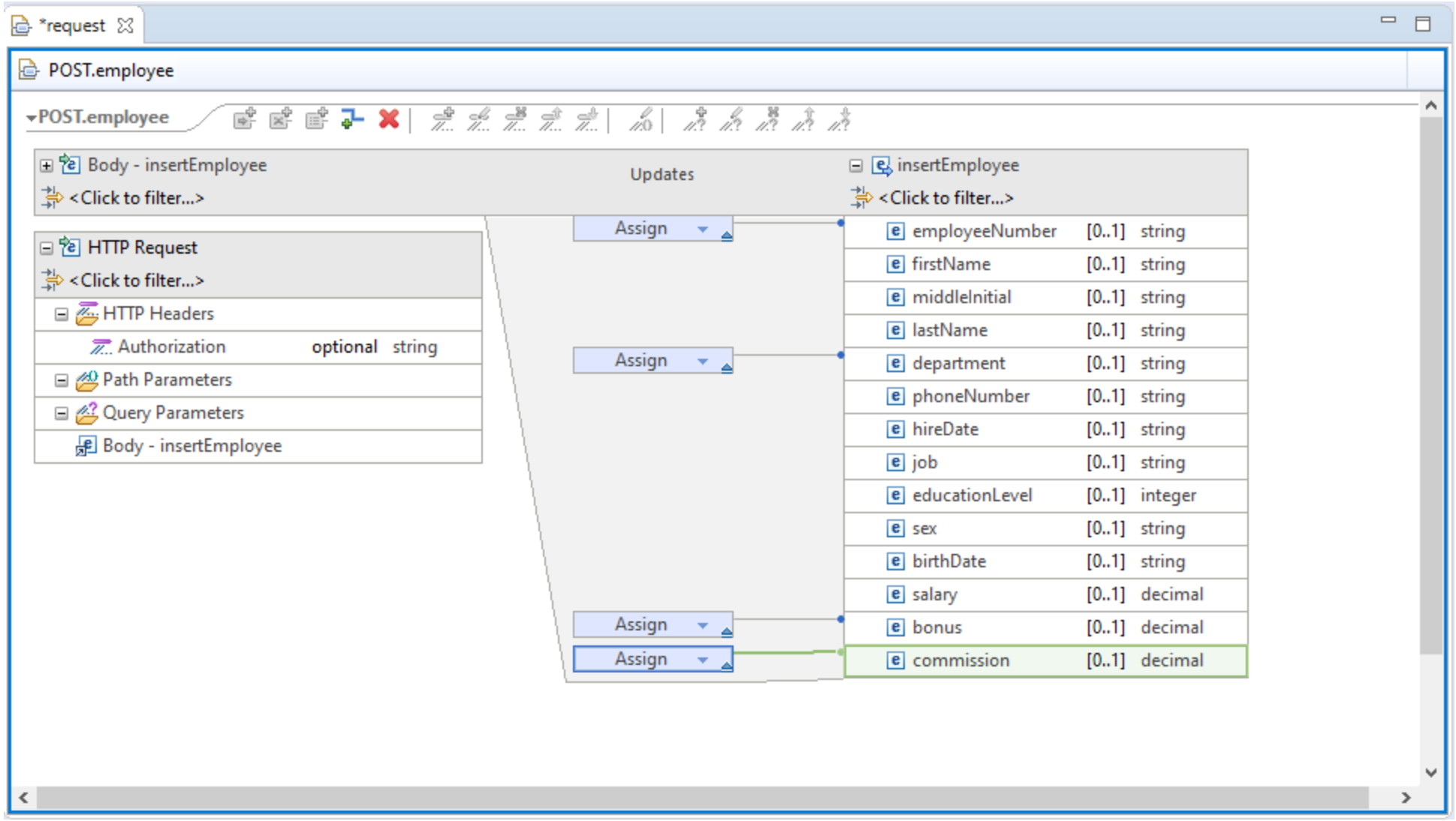
API mapping: Remove or assign values to the fields exposed by service developer



mitchj@us.ibm.com

API toolkit – API Editor and Db2 REST service

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

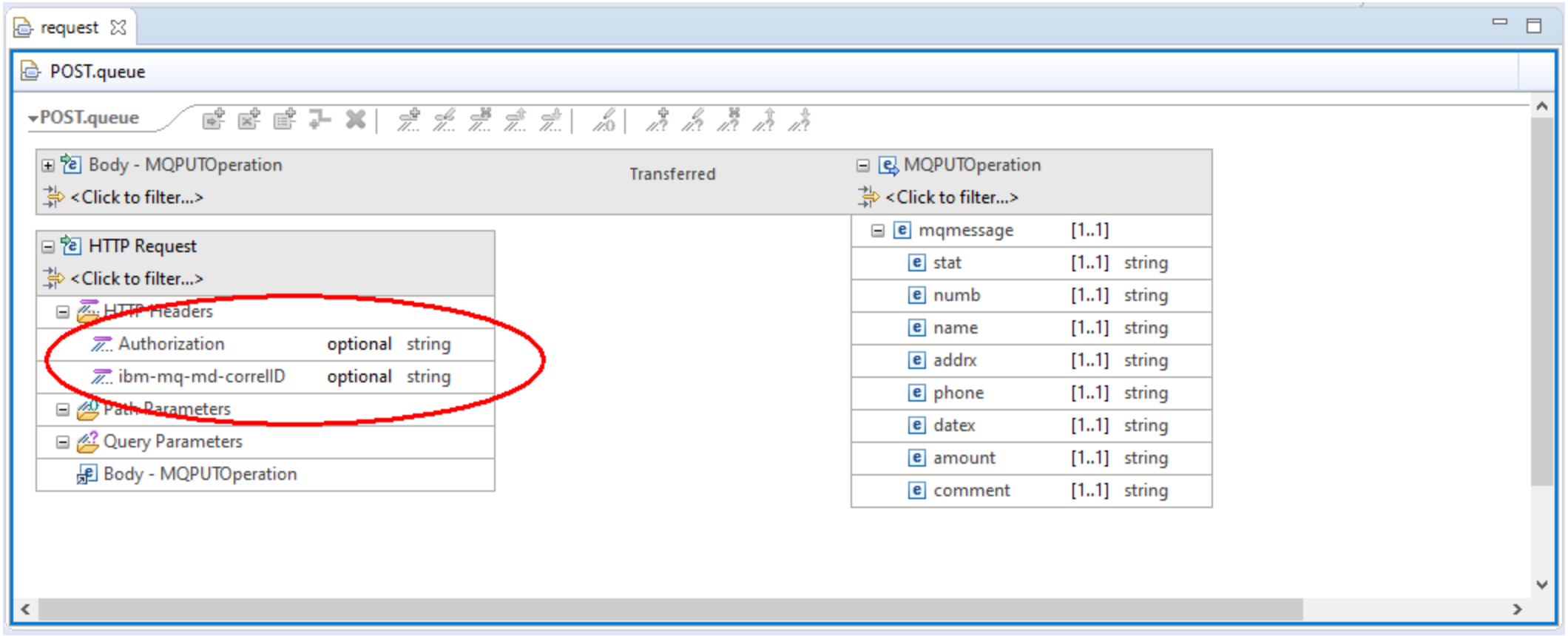


Set default values for a subset of table columns

mitchj@us.ibm.com

API toolkit – Header properties

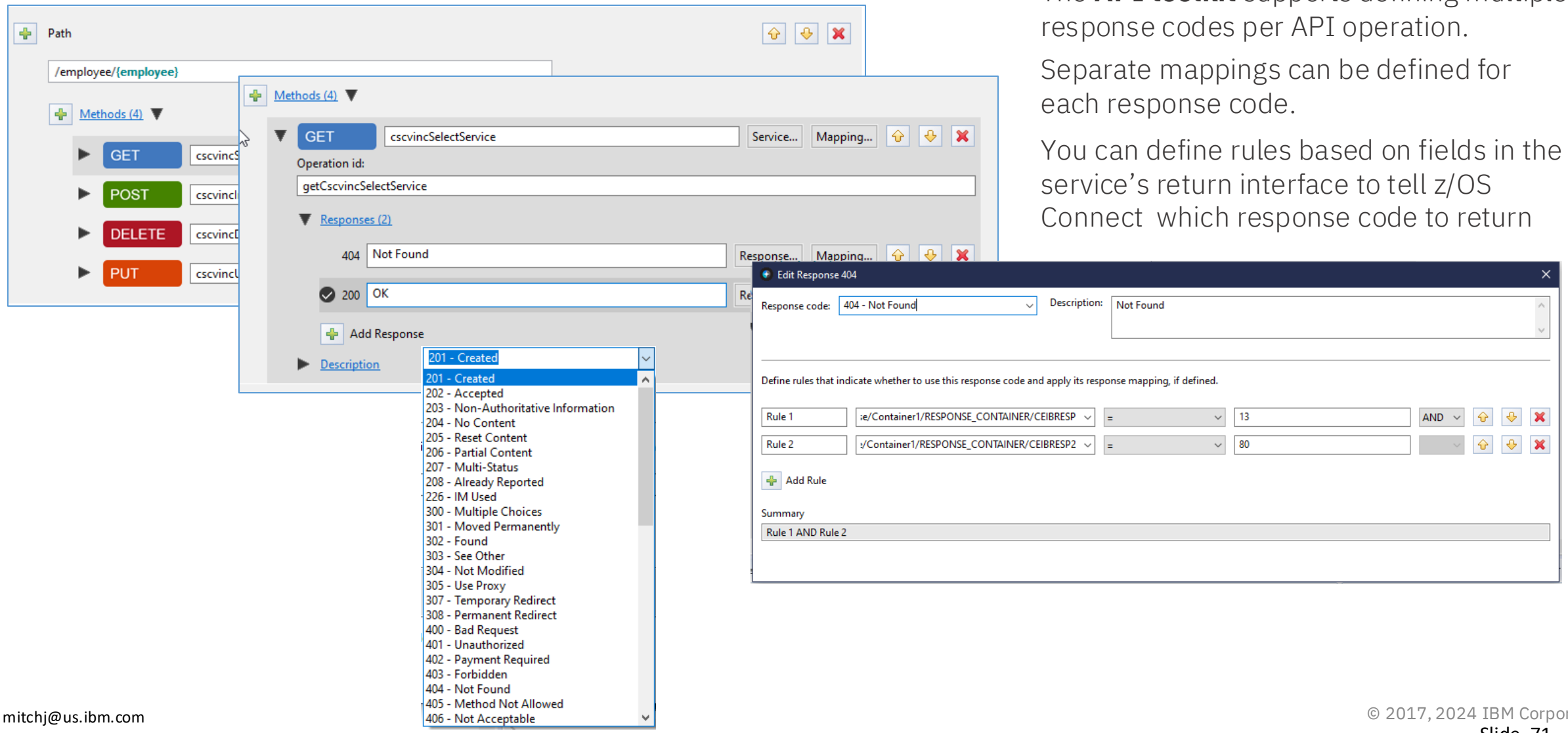
API mapping: Allows adding HTTP header properties



Retrieve specific message from queue based on correlation identity

mitchj@us.ibm.com

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.

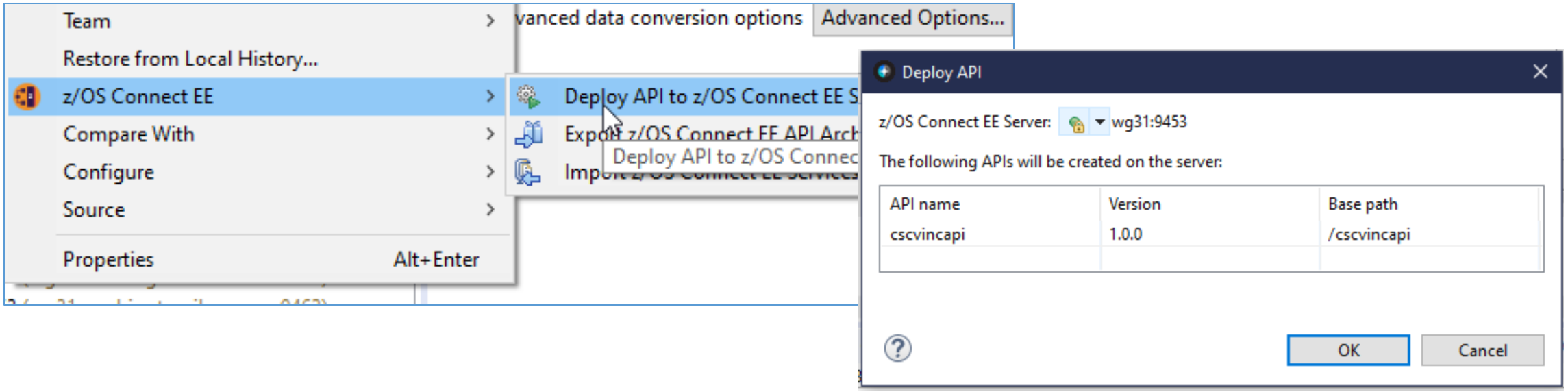
You can define rules based on fields in the service's return interface to tell z/OS Connect which response code to return

mitchj@us.ibm.com

API toolkit – API Editor

API deployment

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

mitchj@us.ibm.com

This process “creates” and deploys the OpenAPI2 specification document

z/OS Connect OpenAPI2 tooling **produces** an OpenAPI 2 (Swagger 2.0) specification document, where the details of the methods and request/response messages in the API specification are driven by the nature of the z/OS asset (JSON format).

mitchj@us.ibm.com

Tech-Tip: You can create Service and API Archive files outside of the Eclipse tooling

- **Use zconbt to create a service archive file**

```
C:\z\workspaces\zCEE>..\..\software\zconbt\bin\zconbt --
projectDirectory=C:\z\workspaces\zcee\miniloanService --file=miniloan.sar
BAQB0000I: z/OS Connect Enterprise Edition 3.0 Build Toolkit Version 1.10 (20220817-1609).
BAQB0001I: Creating service archive from configuration file
C:\z\workspaces\zcee\miniloanService\service.properties.
BAQB1022I: IBM MQ for z/OS plugin for IBM z/OS Connect EE V3.0 build toolkit, code level is
3.0.60.0(20220817-1609).
BAQB0002I: Successfully created service archive file miniloan.sar
```

- **Using zconbt to create an API archive file**

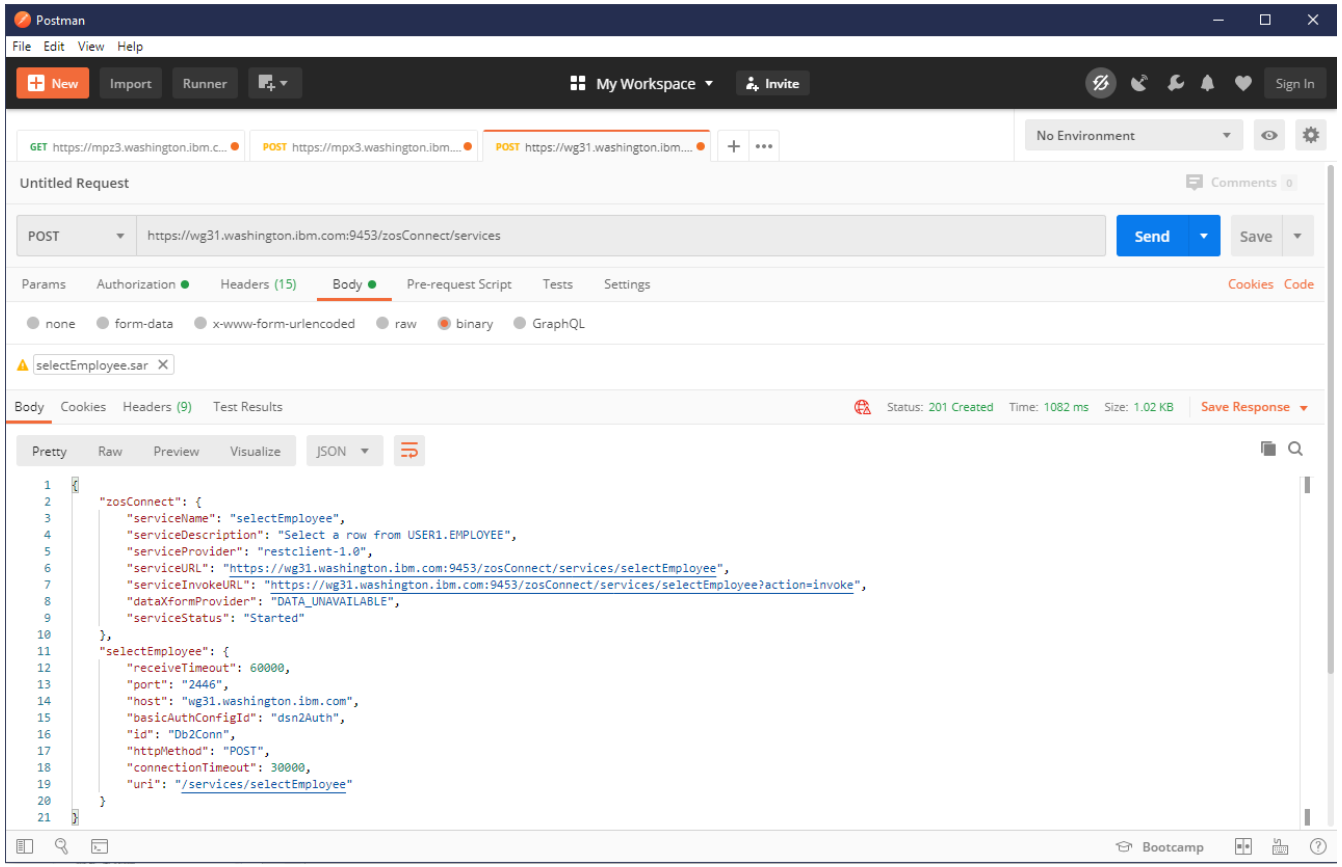
```
C:\z\workspaces\zCEE>..\..\software\zconbt\bin\zconbt --
projectDirectory=C:/z/workspaces/zcee/miniloan --file miniloan.aar
BAQB0000I: z/OS Connect Enterprise Edition 3.0 Build Toolkit Version 1.10 (20220817-1609).
BAQB0028I: Creating API archive file from API project directory
C:/z/workspaces/zcee/miniloan.
BAQB0029I: Successfully created API archive file miniloan.aar.
```

The Eclipse tool kit is still required to develop the service or API.

mitchj@us.ibm.com

Deploying Service and API Archive files outside of the Eclipse tooling

- Use SAR or AAR files as request message and use HTTP POST
- Use the z/OS Connect administrative RESTful APIs
- Use Postman or cURL (client for URL)



- Deploy a service archive file

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

- Deploy an API archive file

curl --data-binary @Catalog.aar --header "Content-Type: application/zip" https://mpxm:9453/zosConnect/apis

Results:

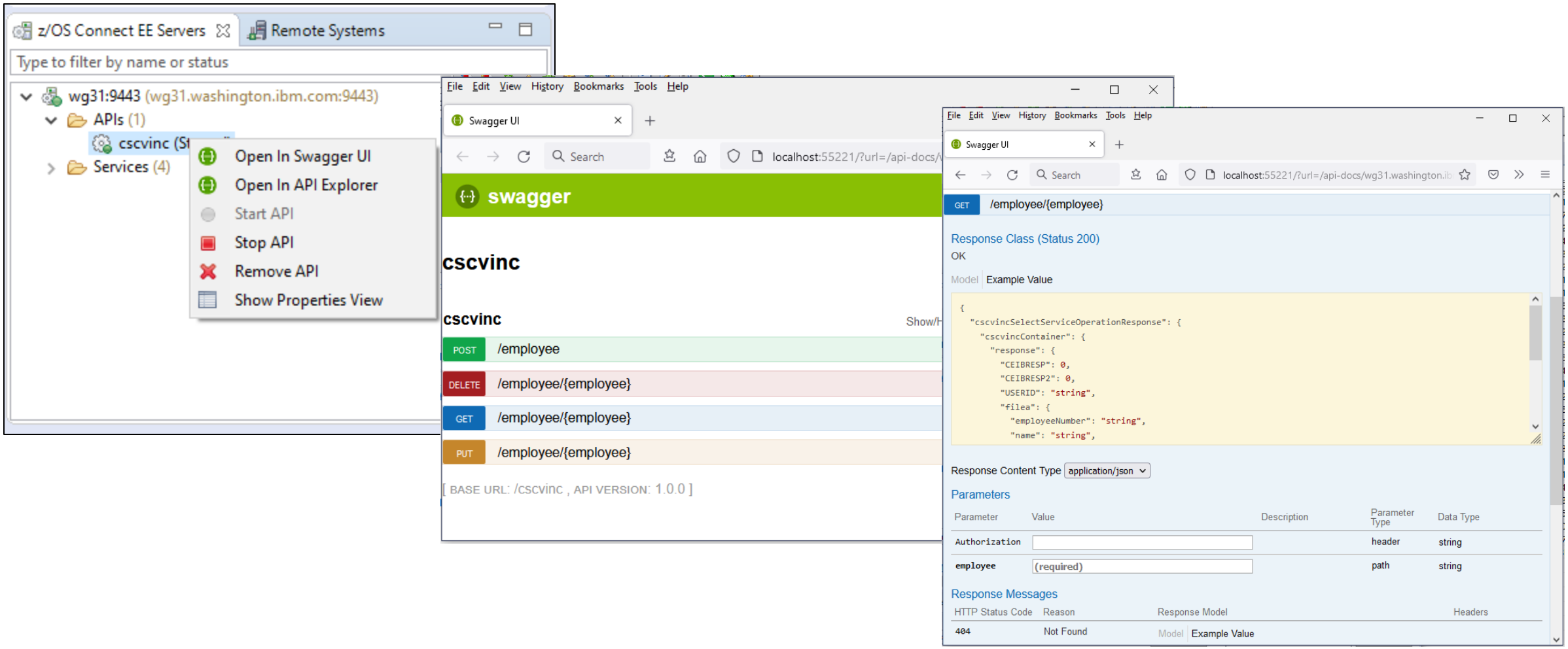

```
{"zosConnect":{"serviceName":"selectEmployee","serviceDescrip
tion":"Select a row from
USER1.EMPLOYEE","serviceProvider":"IBM_ZOS_CONNECT_SERVICE_RE
ST_CLIENT-1.0
","serviceURL":"https://mpxm:9453/zosConnect/services/selectE
mployee","serviceInvokeURL":"https://mpxm:9453/zosConnect/ser
vices/selectEmployee?action=invoke","dataXformProvider":"DATA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


mitchj@us.ibm.com

Testing an API using the API toolkit

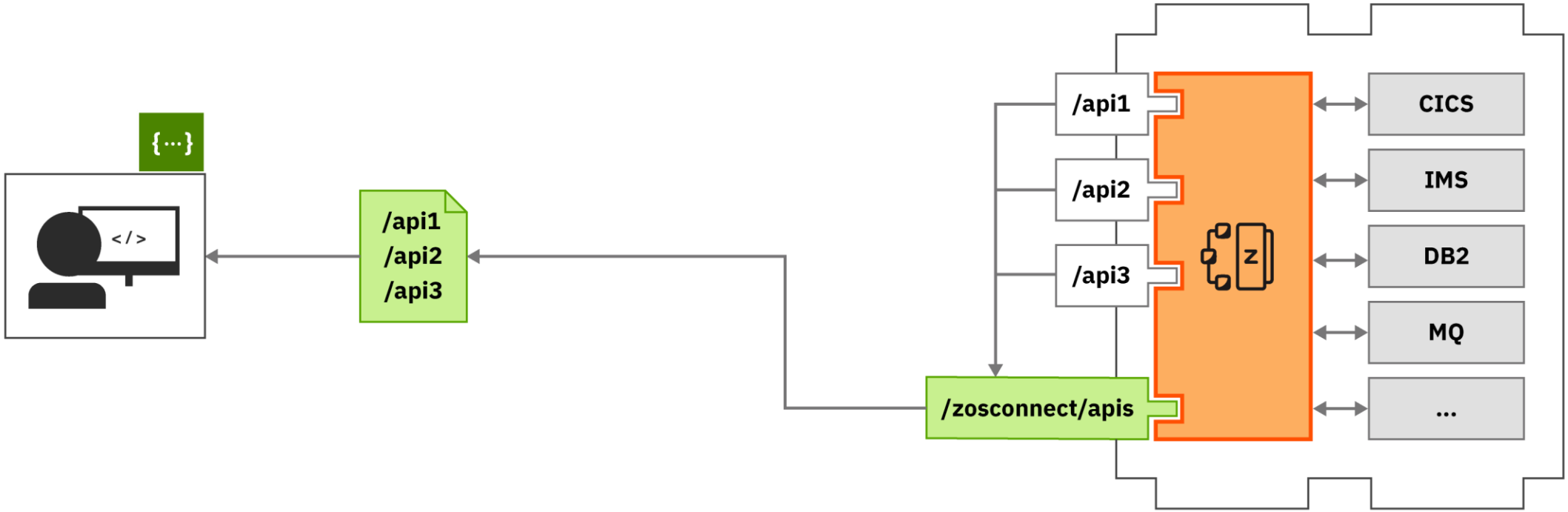
Testing with Swagger UI

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



mitchj@us.ibm.com

API Documentation



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

mitchj@us.ibm.com

Open API 3
z/OS Connect Designer

Accessing z/OS resources from API using the z/OS Connect Designer

mitchj@us.ibm.com

With OpenAPI3, the specification description provides the details of the API

cscvinc.yaml

File Edit View

```yaml
  "/employee/{employee}":
    get:
      tags:
        - Employee
      operationId: getEmployeeSelectService
      x-ibm-zcon-roles-allowed:
        - Staff
      parameters:
        - name: Authorization
          in: header
          required: false
          schema:
            type: string
        - name: employee
          in: path
          required: true
          schema:
            type: string
            maxLength: 6
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/getEmployeeSelectService_response_200"
        "404":
          description: Not Found
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/getEmployeeSelectService_response_404"
        "500":
          description: Severe Error
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/getEmployeeSelectService_response_500"
    put:
      tags:
        - Employee
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

cscvinc.yaml

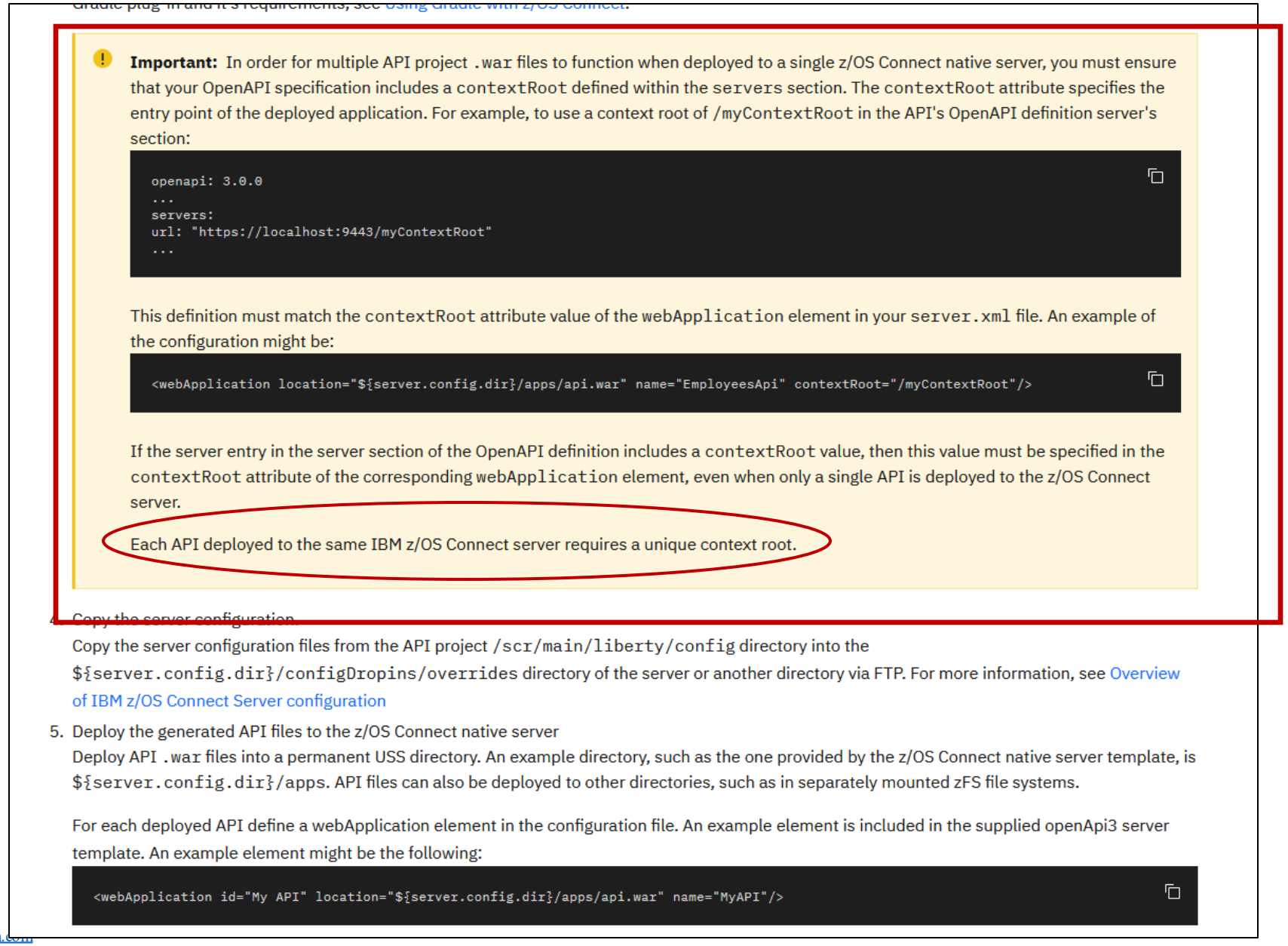
File Edit View

```yaml
getEmployeeSelectService_response_200:
  type: object
  properties:
    summary:
      $ref: '#/components/schemas/getEmployeeSelectService_response_200_message'
    detail:
      $ref: '#/components/schemas/getEmployeeSelectService_response_200_detail'
getEmployeeSelectService_response_200_message:
  type: object
  properties:
    message:
      type: string
  example:
    message: record retrieved
getEmployeeSelectService_response_200_detail:
  type: object
  properties:
    EmployeeSelectServiceOperationResponse:
      type: object
      properties:
        employeeData:
          type: object
          properties:
            response:
              type: object
              properties:
                employeeDetails:
                  type: object
                  properties:
                    employeeNumber:
                      type: string
                      maxLength: 6
                    name:
                      type: string
                      maxLength: 20
                    address:
                      type: string
                      maxLength: 20
                    phoneNumber:
                      type: string
                      maxLength: 8
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

mitchj@us.ibm.com

Let's stop for a moment:
z/OS Native Server Considerations – API Provider Deployment



Important: In order for multiple API project `.war` files to function when deployed to a single z/OS Connect native server, you must ensure that your OpenAPI specification includes a `contextRoot` defined within the `servers` section. The `contextRoot` attribute specifies the entry point of the deployed application. For example, to use a context root of `/myContextRoot` in the API's OpenAPI definition server's section:

```
openapi: 3.0.0
...
servers:
url: "https://localhost:9443/myContextRoot"
...
```

This definition must match the `contextRoot` attribute value of the `webApplication` element in your `server.xml` file. An example of the configuration might be:

```
<webApplication location="${server.config.dir}/apps/api.war" name="EmployeesApi" contextRoot="/myContextRoot"/>
```

If the server entry in the server section of the OpenAPI definition includes a `contextRoot` value, then this value must be specified in the `contextRoot` attribute of the corresponding `webApplication` element, even when only a single API is deployed to the z/OS Connect server.

Each API deployed to the same IBM z/OS Connect server requires a unique context root.

4. Copy the server configuration.
   Copy the server configuration files from the API project `/scr/main/liberty/config` directory into the `${server.config.dir}/configDropins/overrides` directory of the server or another directory via FTP. For more information, see Overview of IBM z/OS Connect Server configuration
5. Deploy the generated API files to the z/OS Connect native server
   Deploy API `.war` files into a permanent USS directory. An example directory, such as the one provided by the z/OS Connect native server template, is `${server.config.dir}/apps`. API files can also be deployed to other directories, such as in separately mounted zFS file systems.

   For each deployed API define a webApplication element in the configuration file. An example element is included in the supplied openApi3 server template. An example element might be the following:

```
<webApplication id="My API" location="${server.config.dir}/apps/api.war" name="MyAPI"/>
```

The samples as provided by z/OS Connect are not suitable as is for deployment to a z/OS Connect Native server.

https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=building-zos-connect-apis-gradle

mitchj@us.ibm.com

z/OS Server Considerations – API Context Root

https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=building-zos-connect-apis-gradle

The drop-ins directory

The *drop-ins* directory, `/config/dropins` is a special directory that is supported by WebSphere® Application Server for Liberty. It
files to be deployed and dynamically loaded into the running IBM z/OS Connect with no additional definitions that are required in th
configuration file.

By default, z/OS Connect Designer deploys the API `.war` file to this directory. Using the same directory in your API container image
creation of that image because the configuration remains the same.

A directory other than drop-ins

This is required in any of the following situations:

- The API's OpenAPI definition server's section contains server entry that includes a context root value, which is not just /.
- Multiple APIs are to be deployed to the same IBM z/OS Connect container. Because the API `.war` file will be generated with a conte
  and multiple API `.war` files in the same server must have unique context root values.
  You need to include a context root value (not /) in the API's OpenAPI definition server's section, for example to use a context root o

```
openapi: 3.0.0
   ...
   servers:
      url: https://localhost:9443/MyCompany
   ...
```

- Requests to start an API require authentication only, without authorization, so the authorization roles need to be mapped to the We
  Application Server for Liberty special subject ALL_AUTHENTICATED_USERS. For more information, see How to define authorization

If you choose not to use the drop-ins directory, you must alter the configuration that is used in z/OS Connect Designer during the creation of the API container image.

The question is when and where to make these updates?

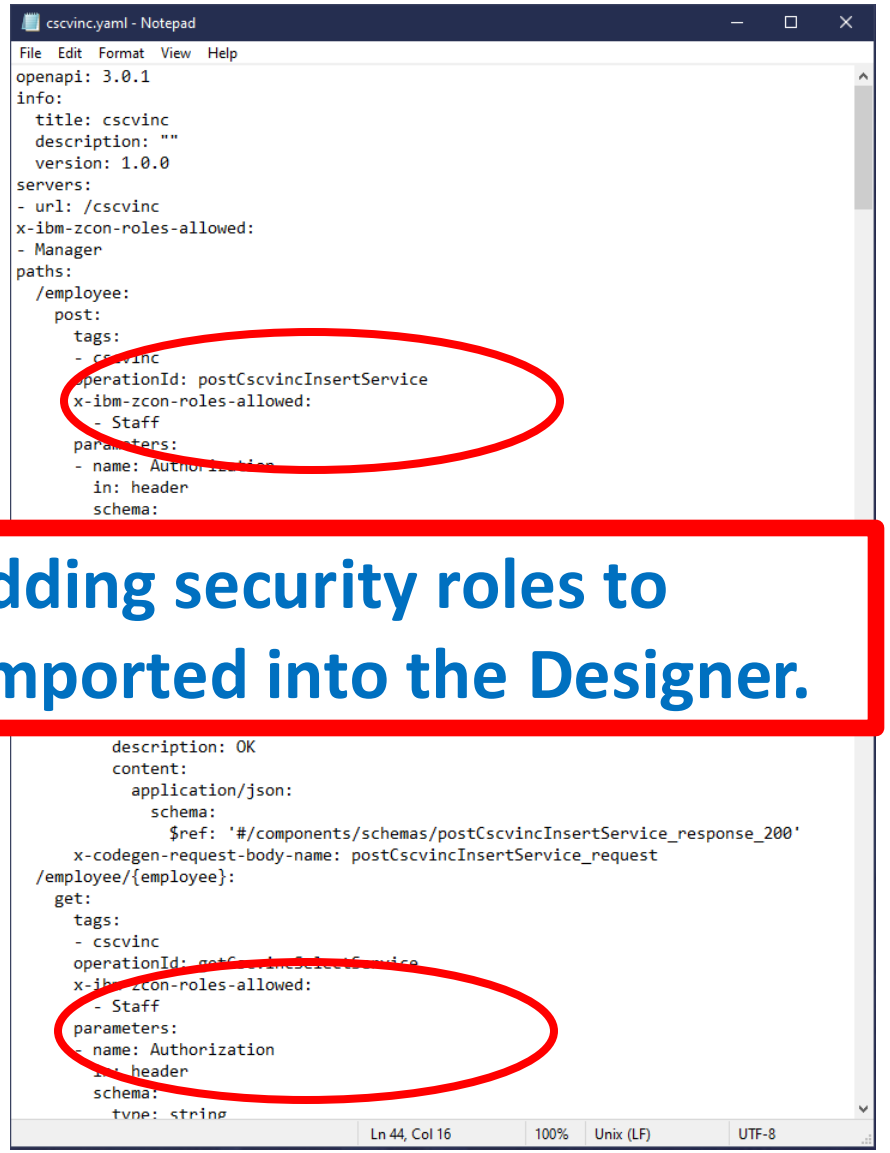
The WSC recommends making the changes in the specification file before it is imported into the Designer.

- **Explicit server XML configuration is *required* to define applications and add the context root**

```
<webApplication id="cics" contextRoot="/cics"  name="cicsAPI"
      location="${server.config.dir}apps/cscvinc.war"/>
<webApplication id="db2" contextRoot="/db2"  name="db2API"
      location="${server.config.dir}apps/employees.war"/>
```

mitchj@us.ibm.com

Also, the API's specification may require new or updated security roles

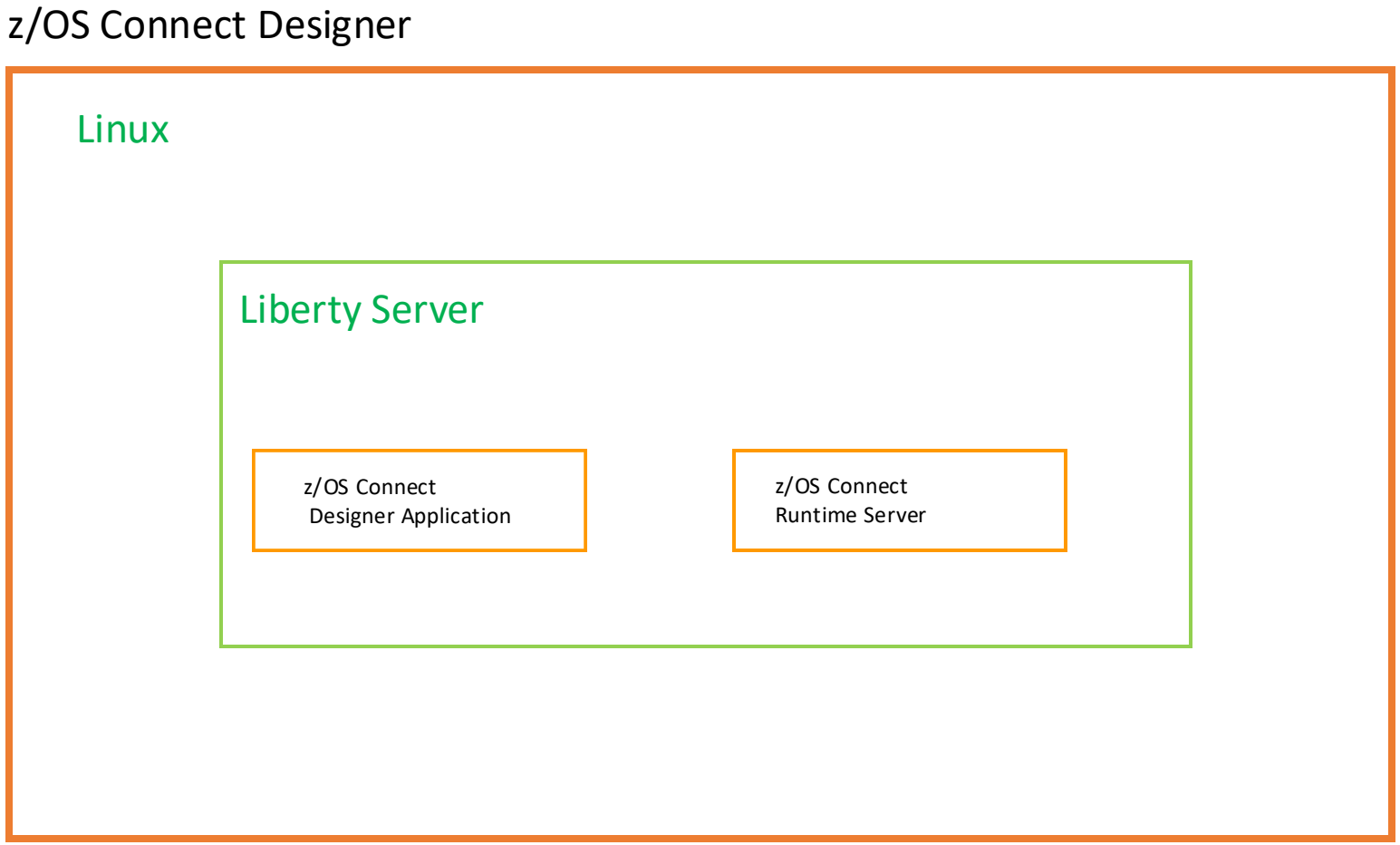


The WSC also recommends adding security roles to specification file before it is imported into the Designer.

mitchj@us.ibm.com

Let’s explore the z/OS Connect Designer Topology

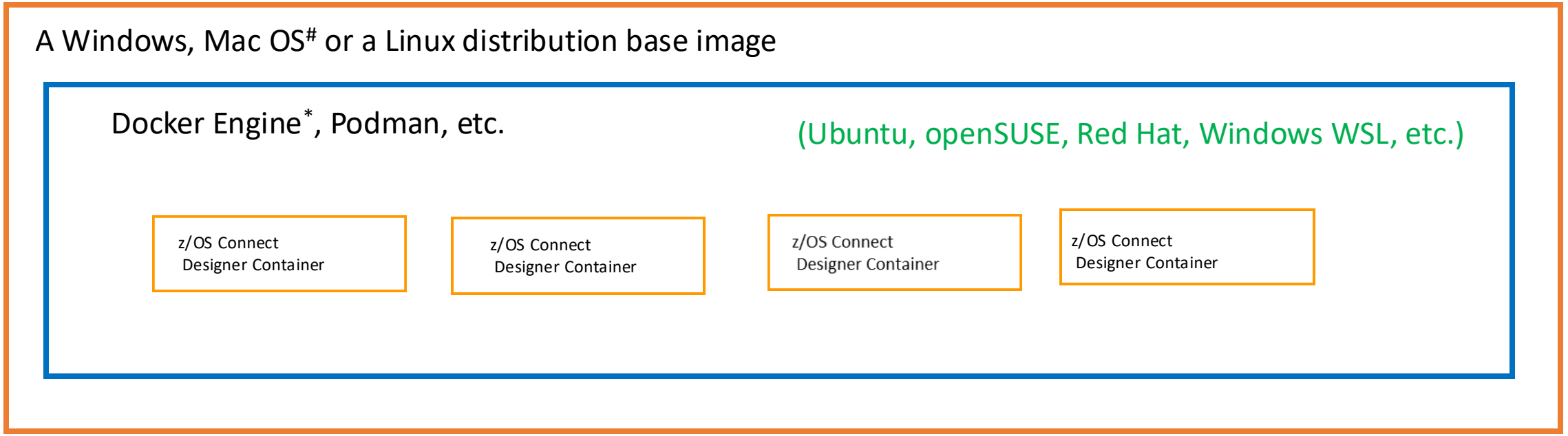
- A z/OS Connect Designer is composed of a self-contained Linux environment configured with a Liberty server running a z/OS Connect Designer application ***and*** a z/OS Connect runtime server, in total, known as a “container”.
- Running a Designer container requires a container runtime environment.



mitchj@us.ibm.com

z/OS Connect Designer runtime options

one API per container, i.e., multiple Designer containers may be required



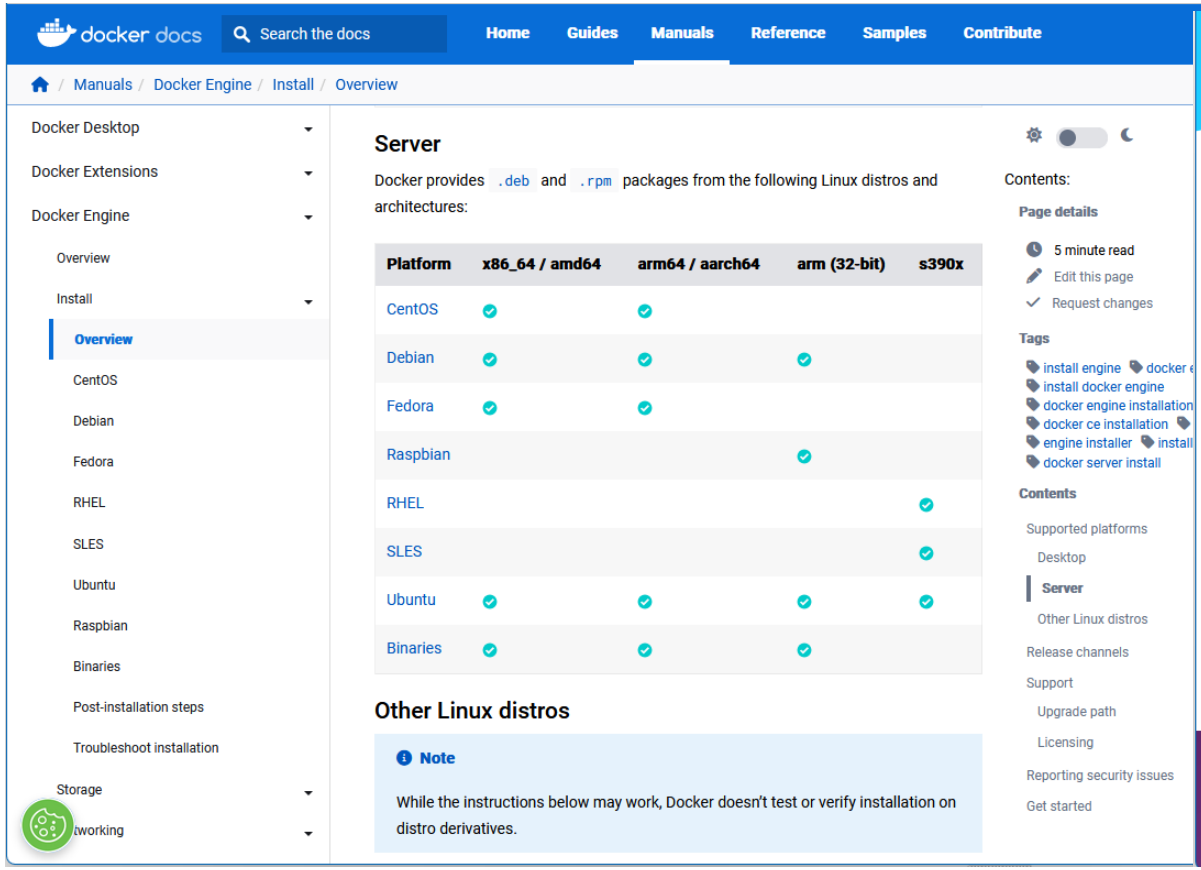
The Washington Systems Center provides our recommendations for best practices for configuring and managing Designer containers at URL https://ibm.biz/BdvrZh

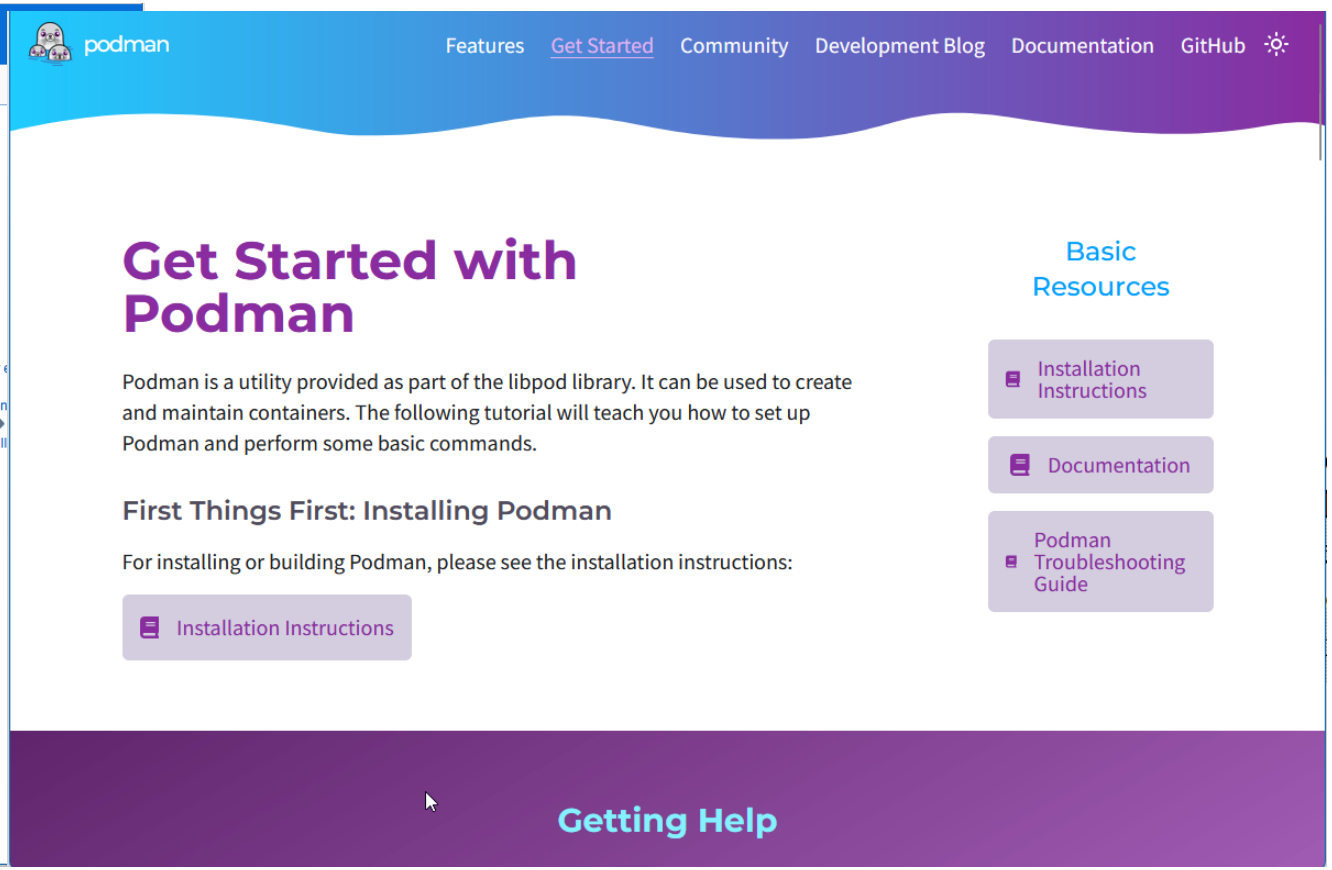
- For basic setup and management of the Designer environment, see *z/OS Connect Designer and Native Experiences*
- For a Podman example, see *Developing and Administering RESTful APIs for CICS REST Services*
- For a Docker engine example, *Developing and Administering RESTful APIs for Db2 REST Services*

mitchj@us.ibm.com

#Warning, Machines with Mac M1 Pro processors may be problematic.
*Docker Engine is not an option on Macs, use Podman

Tech-Tip: Some suggested runtime environments that required no license





https://docs.docker.com/engine/install/#server

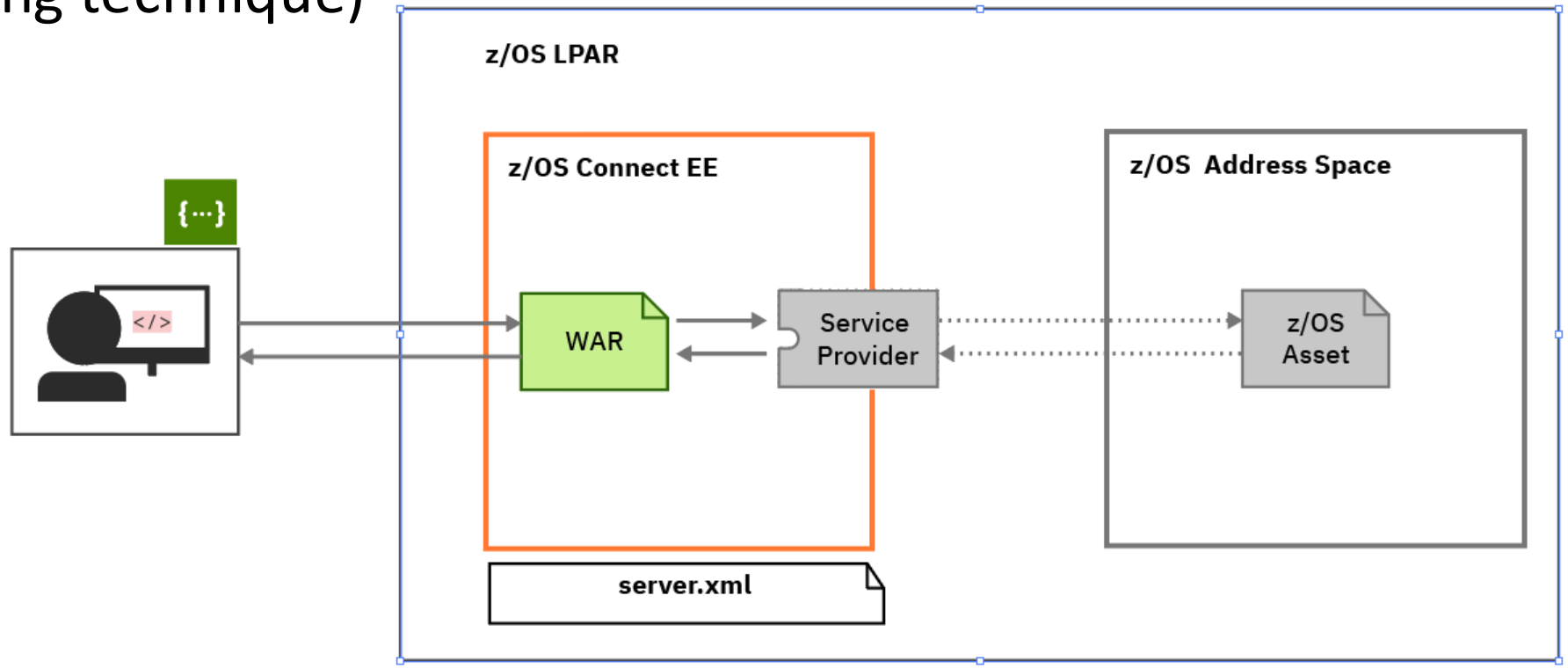
https://podman.io/get-started

Important: The command line interface (CLI) syntax is the same between the Docker Engine and Podman. Just change a Docker command from using the ***docker*** command to the ***podman*** command, e.g., ***docker ps -a*** when using Docker Engine becomes ***podman ps –a*** when using Podman.

mitchj@us.ibm.com

Accessing the CICS, Db2 or IMS resource using an Open API 3 API

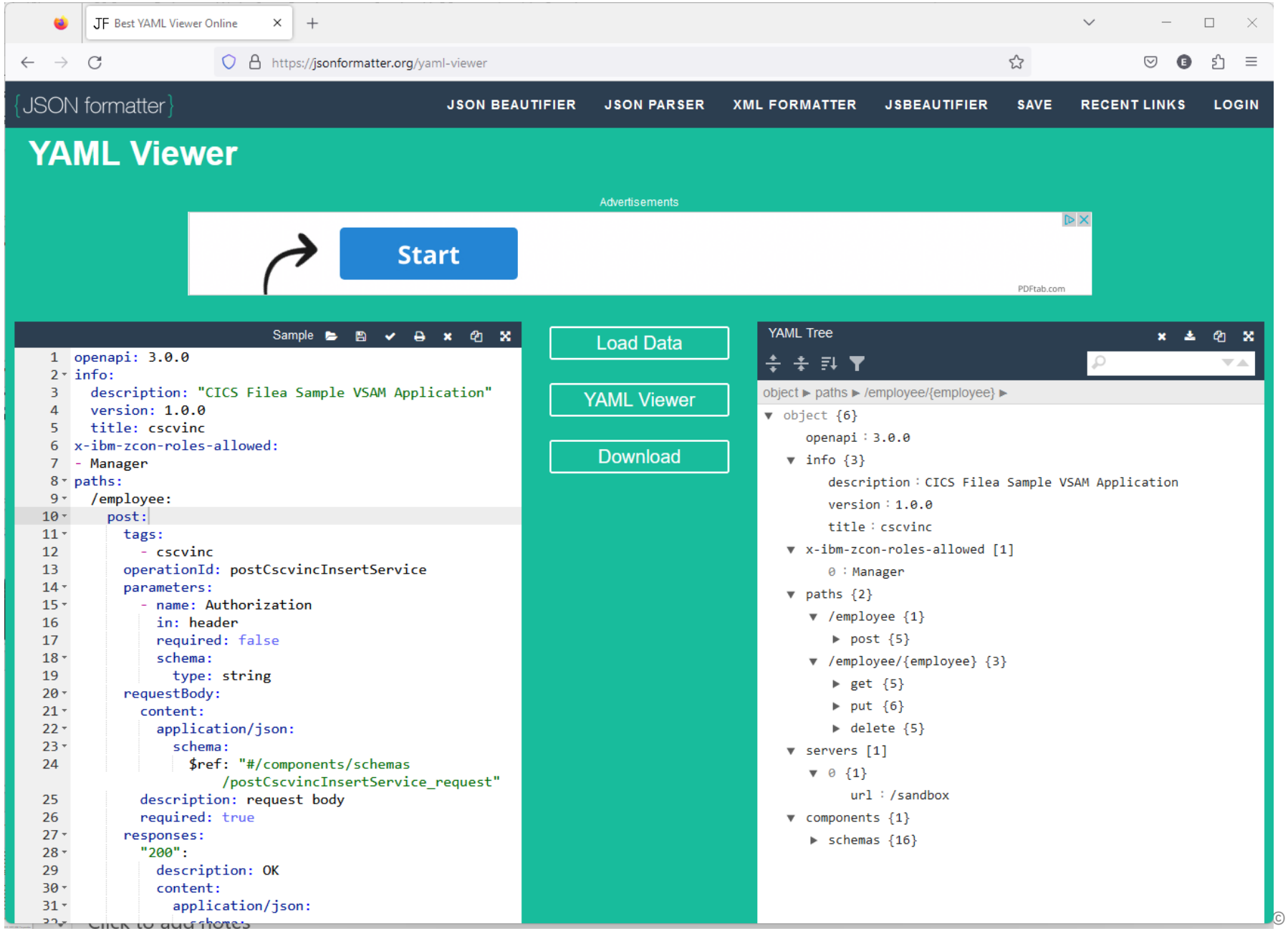
- z/OS Connect OpenAPI 3 APIs are developed using a z/OS Connect Designer web browser tool. This tool generates Web ARchive (WAR) files (a traditional Java packaging technique)



- The WAR provides the RESTful interface is ready to be consumed by a client and it requires the client to have no knowledge that a z/OS resource is being accessed as well as the mapping and transformation for accessingg the resource.
- The Service Provider is tightly coupled to a specific instance of a resource (e.g., host and port)

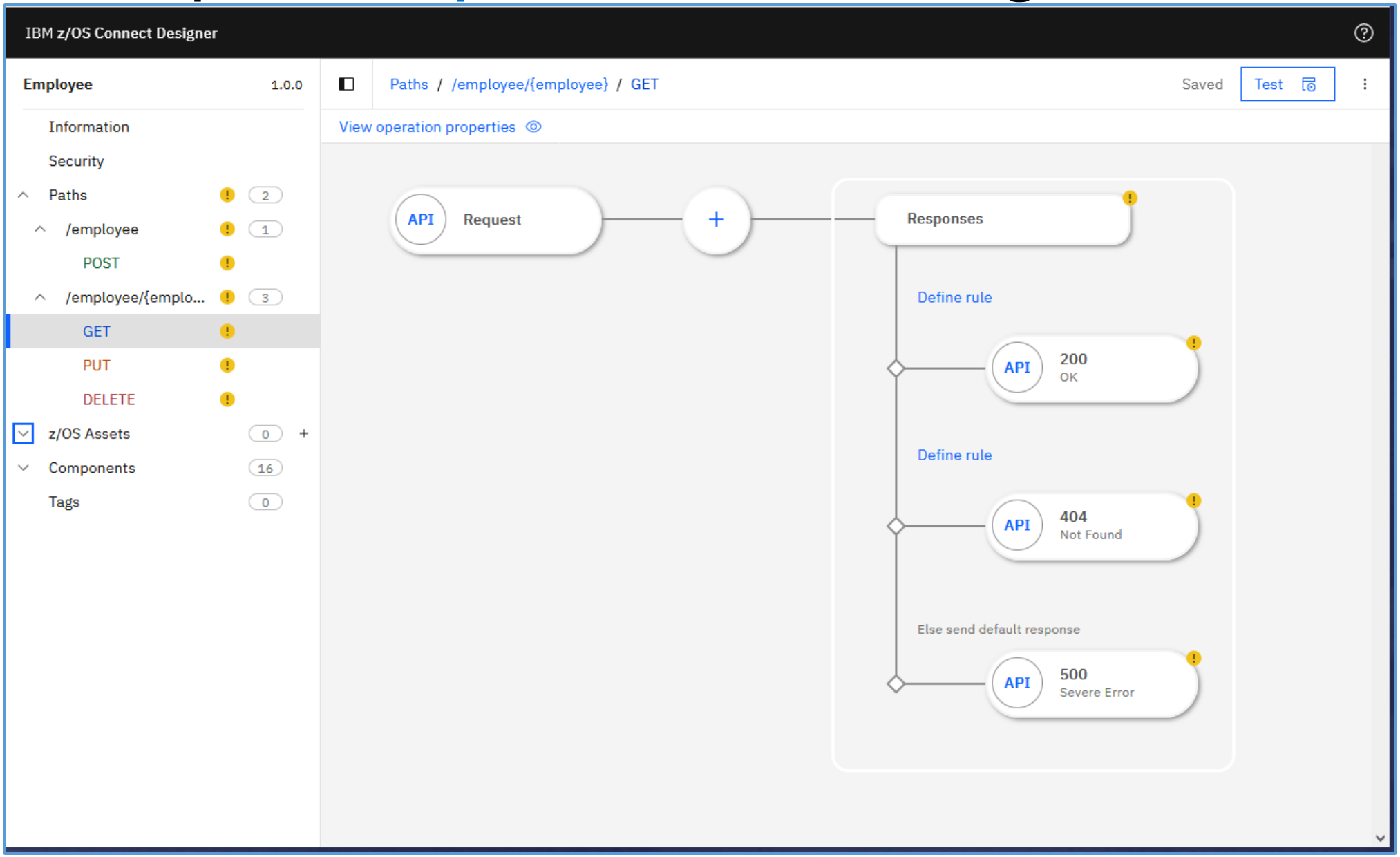
mitchj@us.ibm.com

With OpenAPI 3 you start with an YAML file description of the API



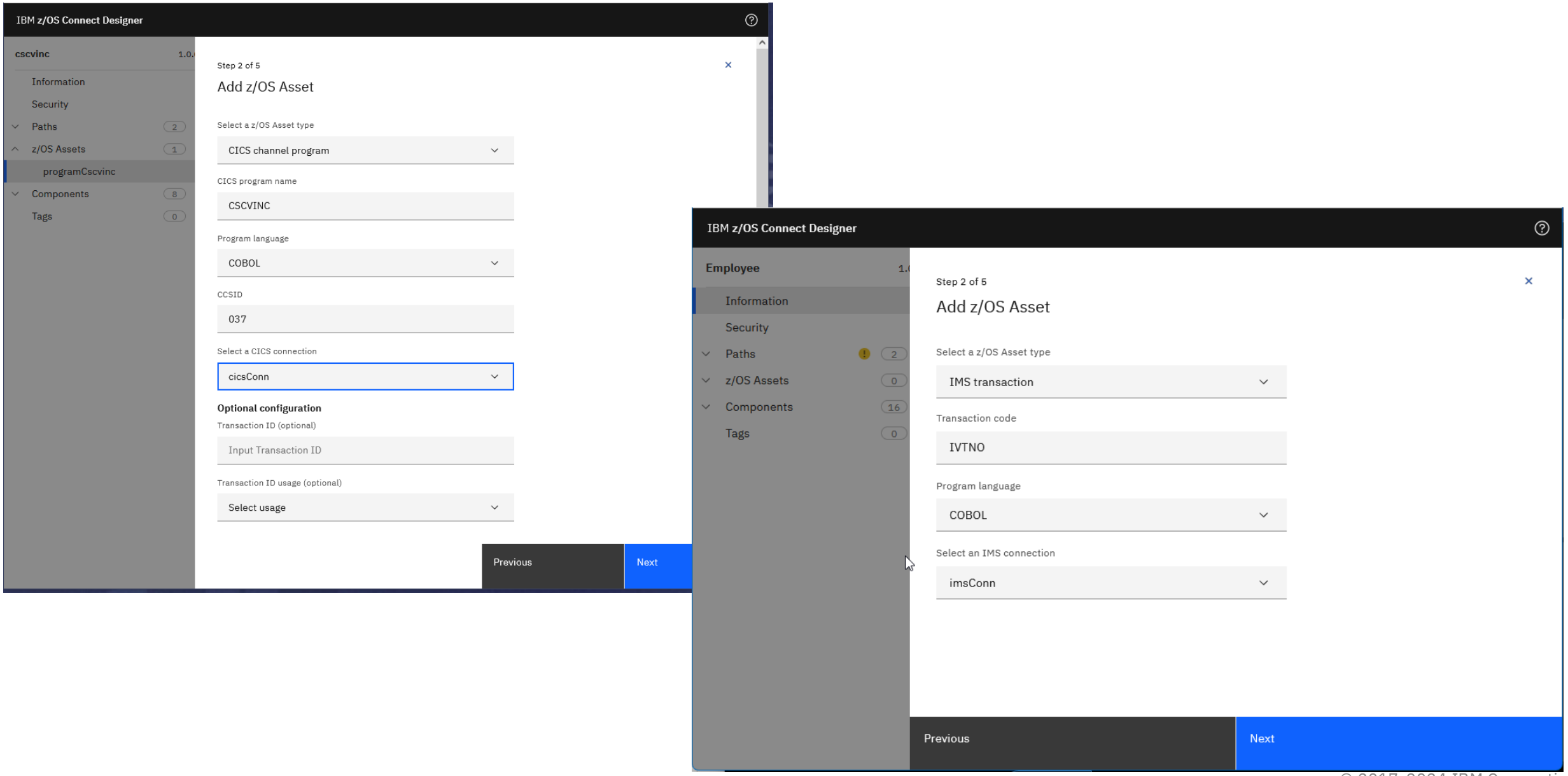
mitchj@us.ibm.com

Import the description of an OpenAPI 3 API into the Designer



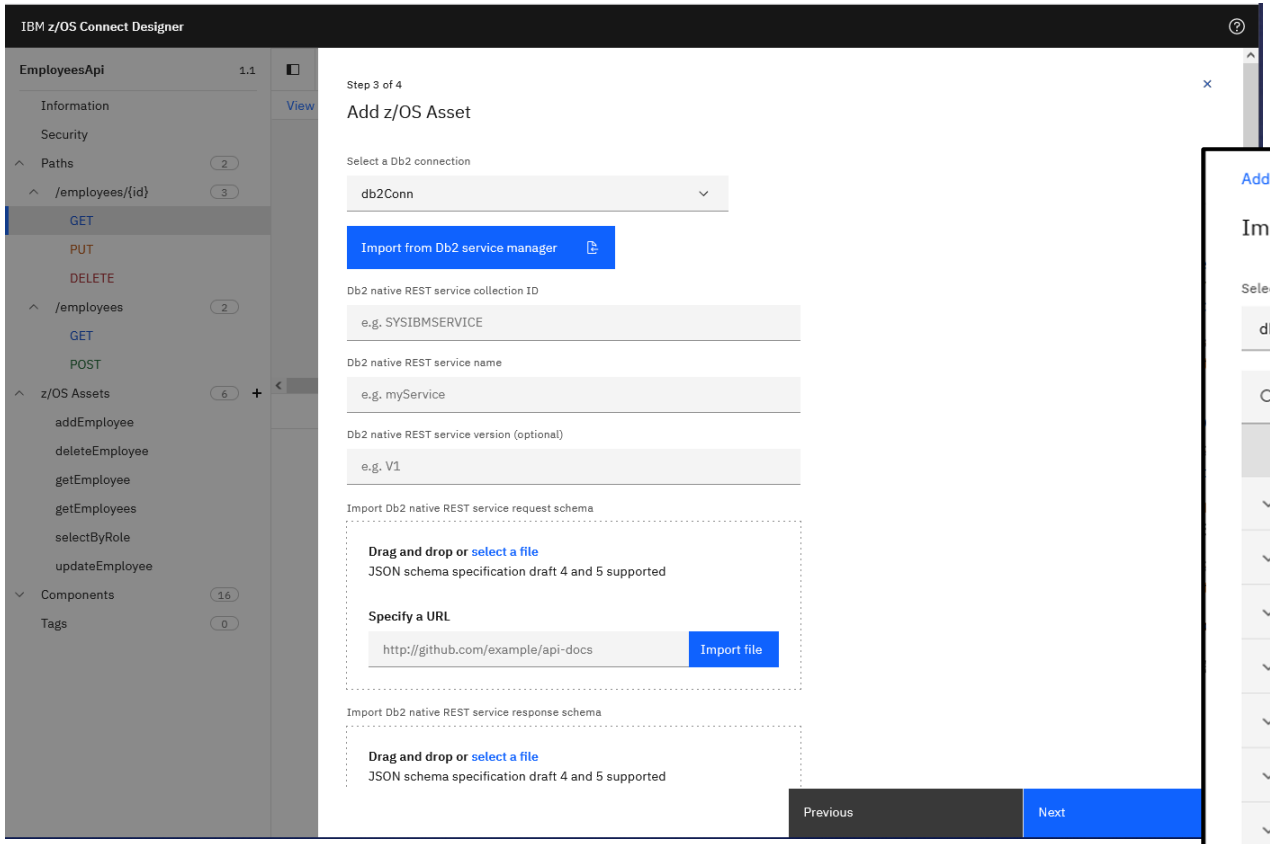
mitchj@us.ibm.com

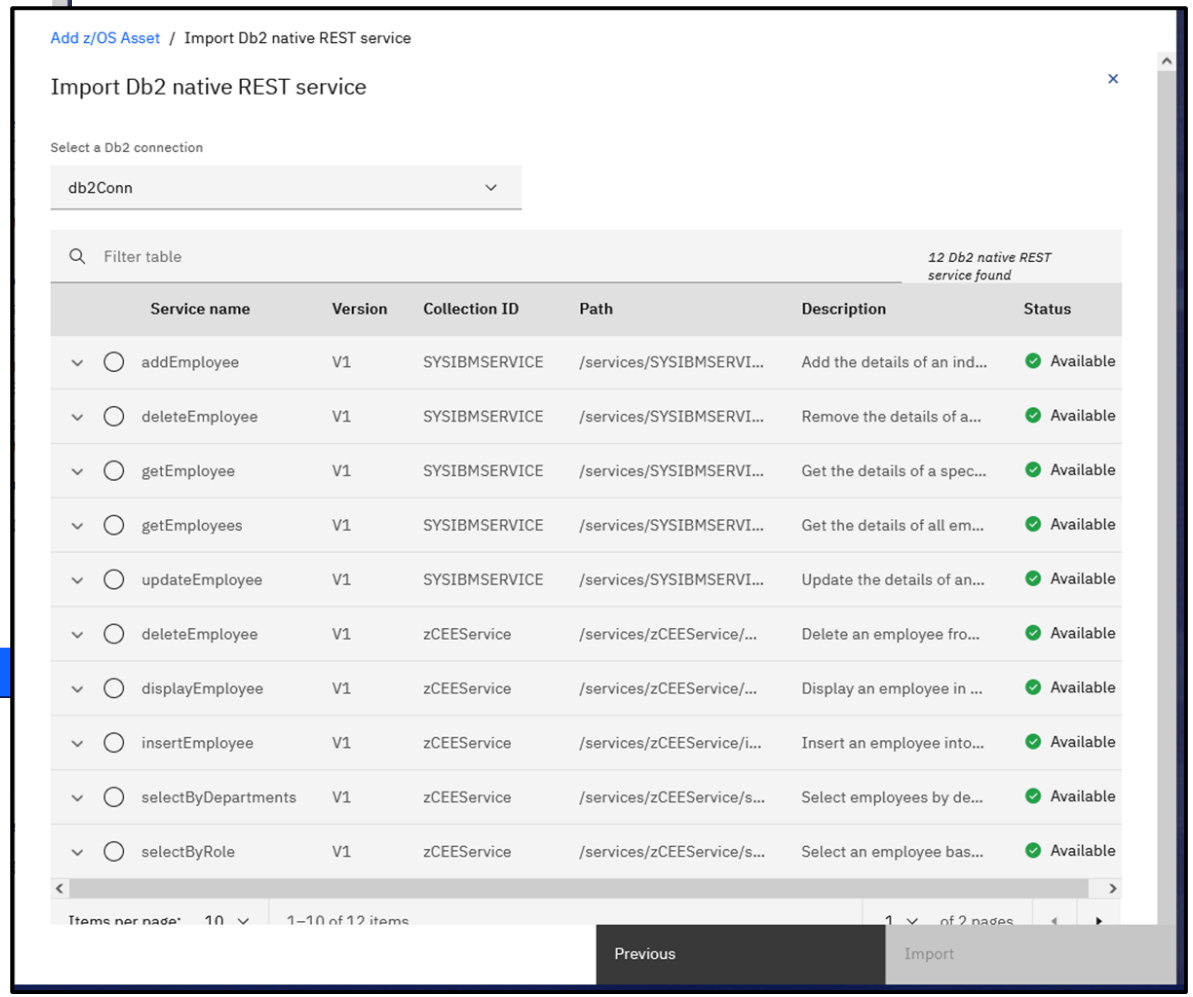
Describe the resource, a CICS program or IMS transaction



mitchj@us.ibm.com

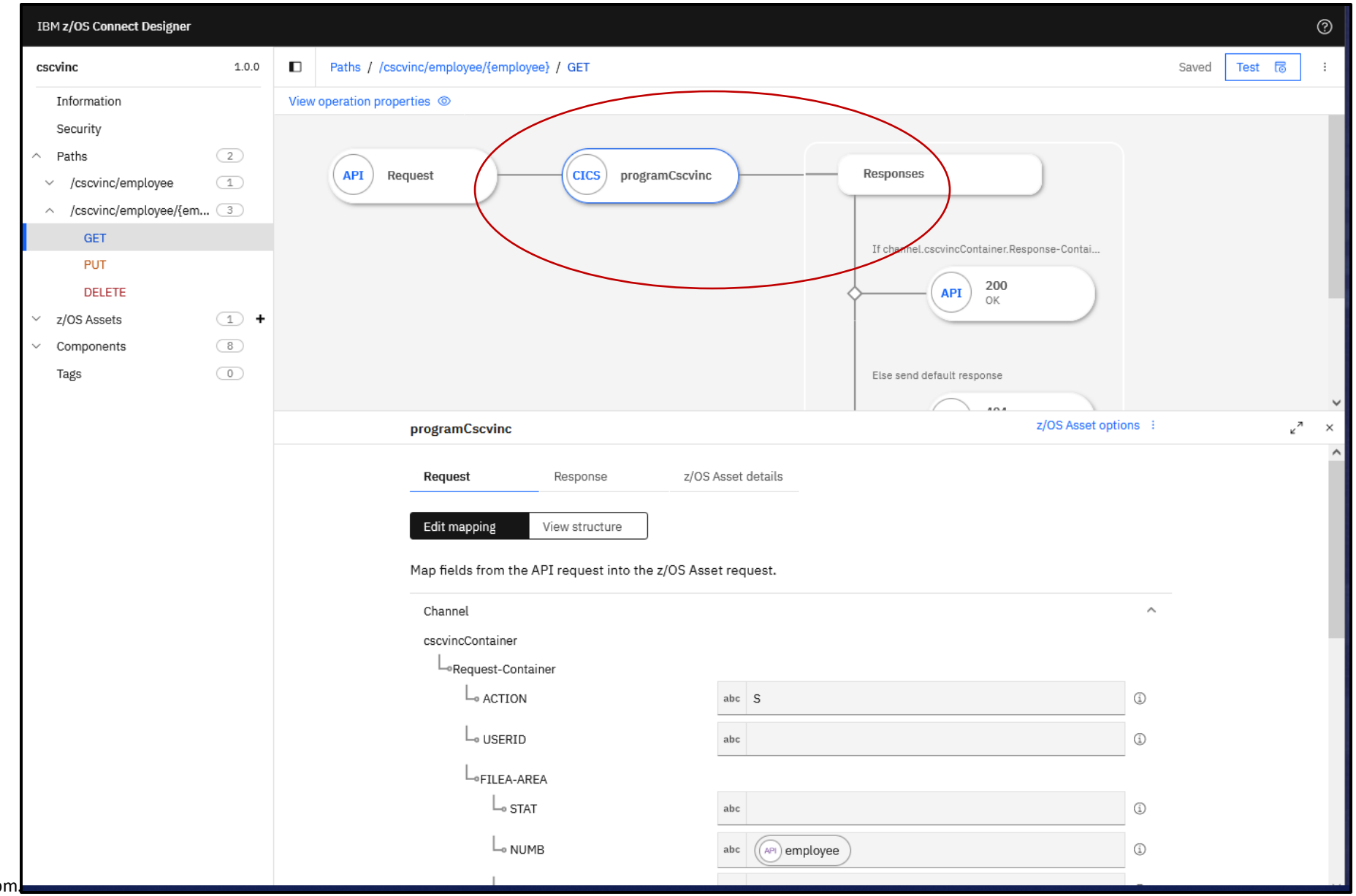
Or a Db2 REST service





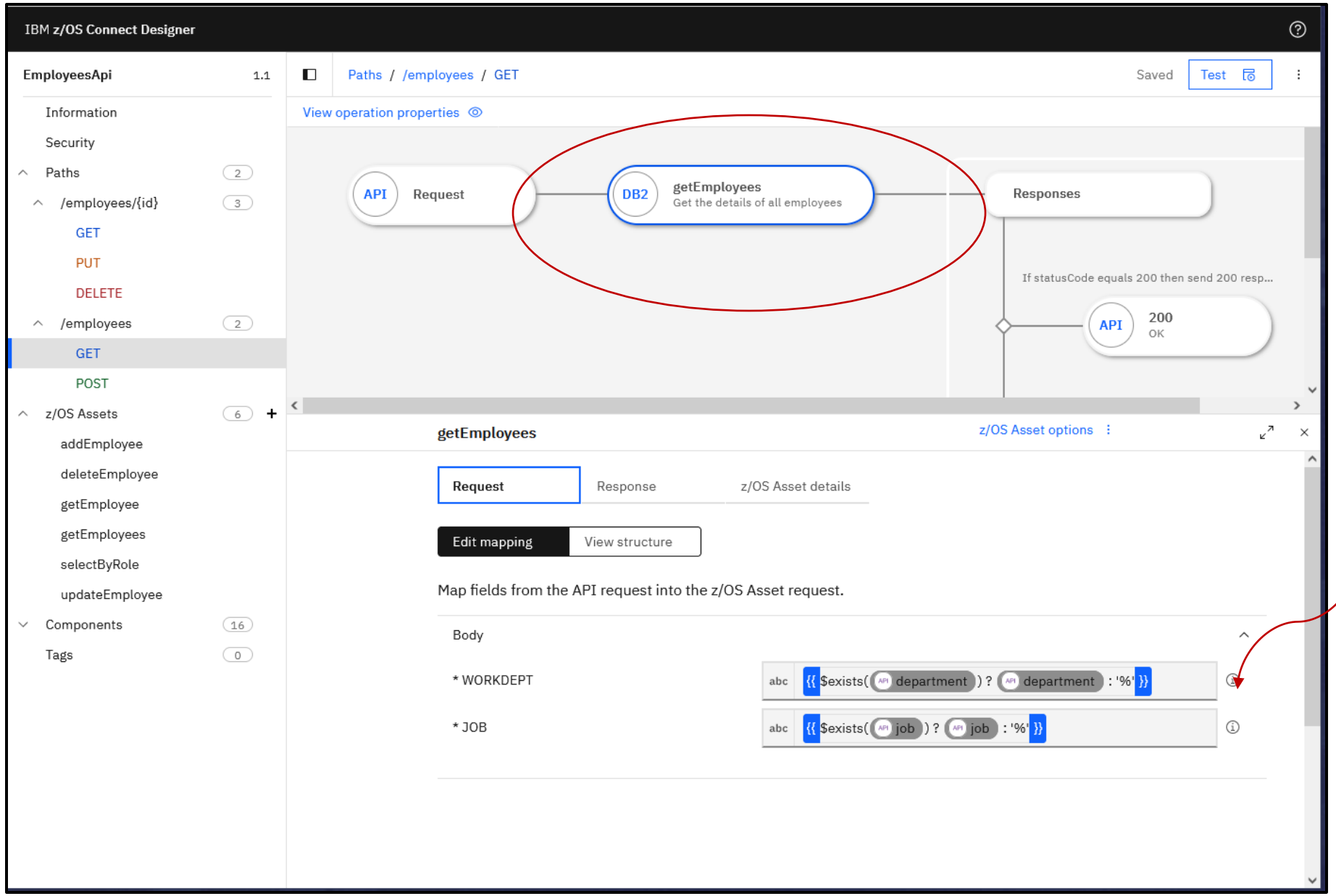
mitchj@us.ibm.com

Map the API's method and request with the resource's input



mitchj@us.ibm.

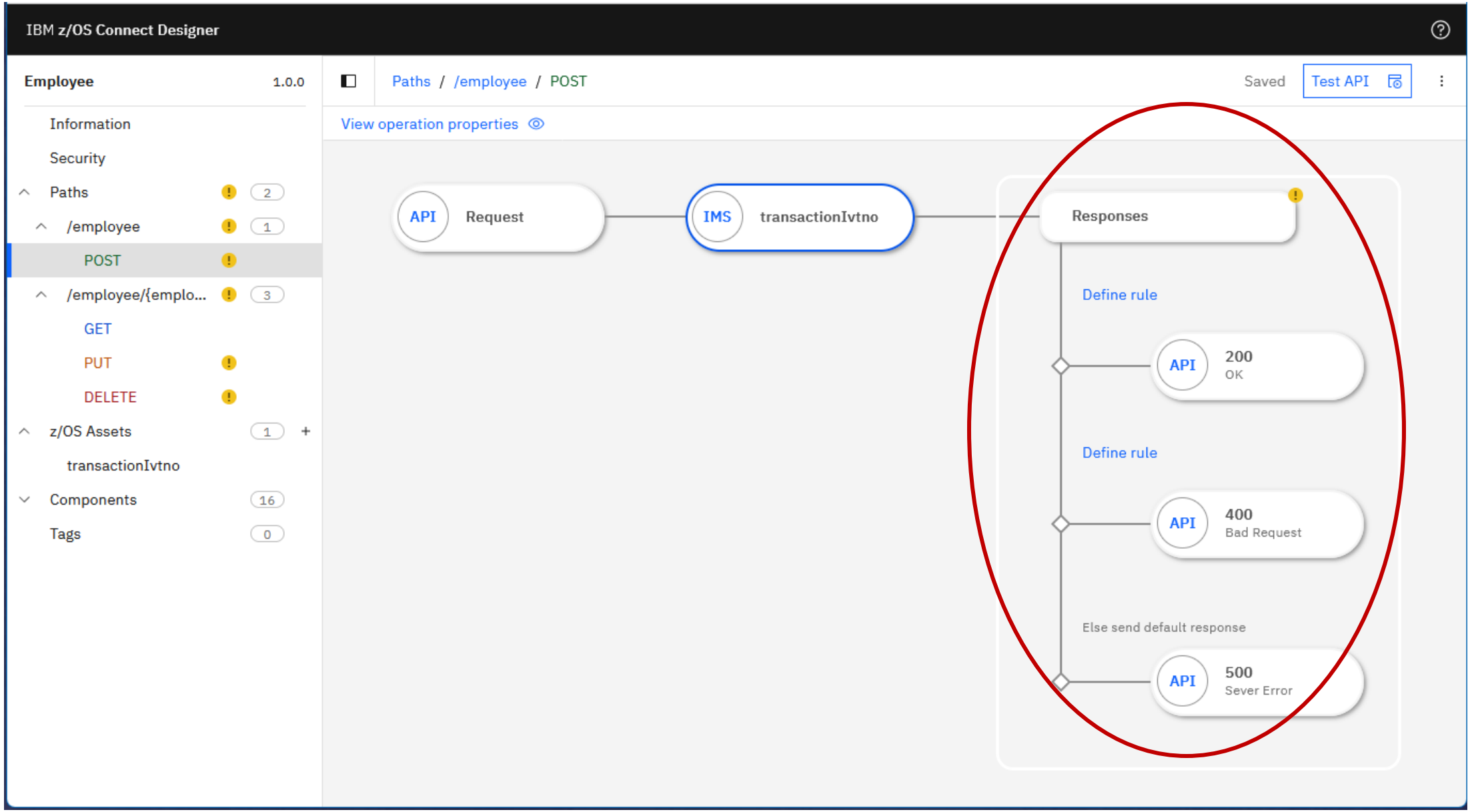
Map the API's request with the z/OS resource's input message



JSONata example

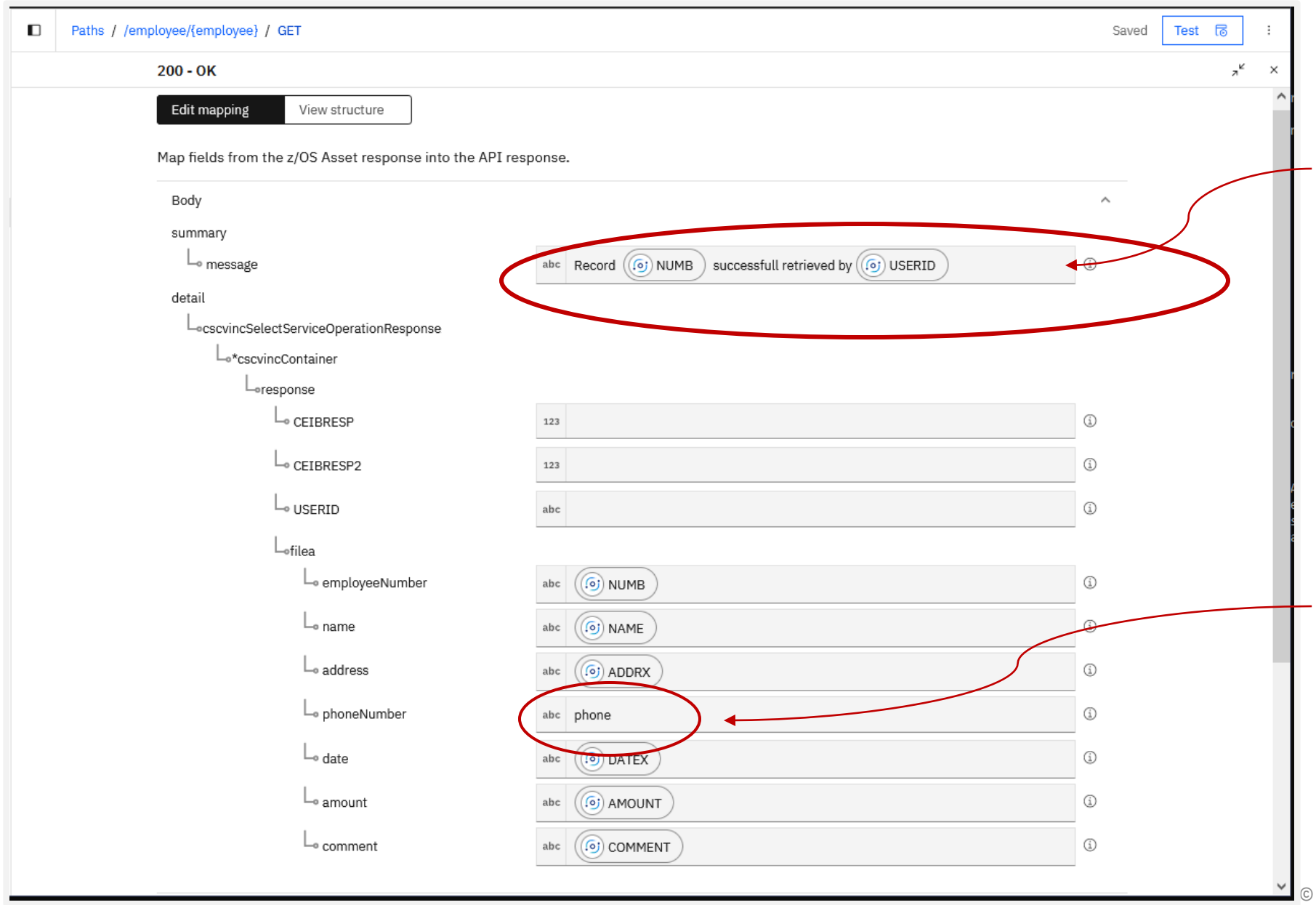
mitchj@us.ibm.com

Map the resource's results to the API's response messages



mitchj@us.ibm.com

Map the z/OS resource's response to the API's response (200)

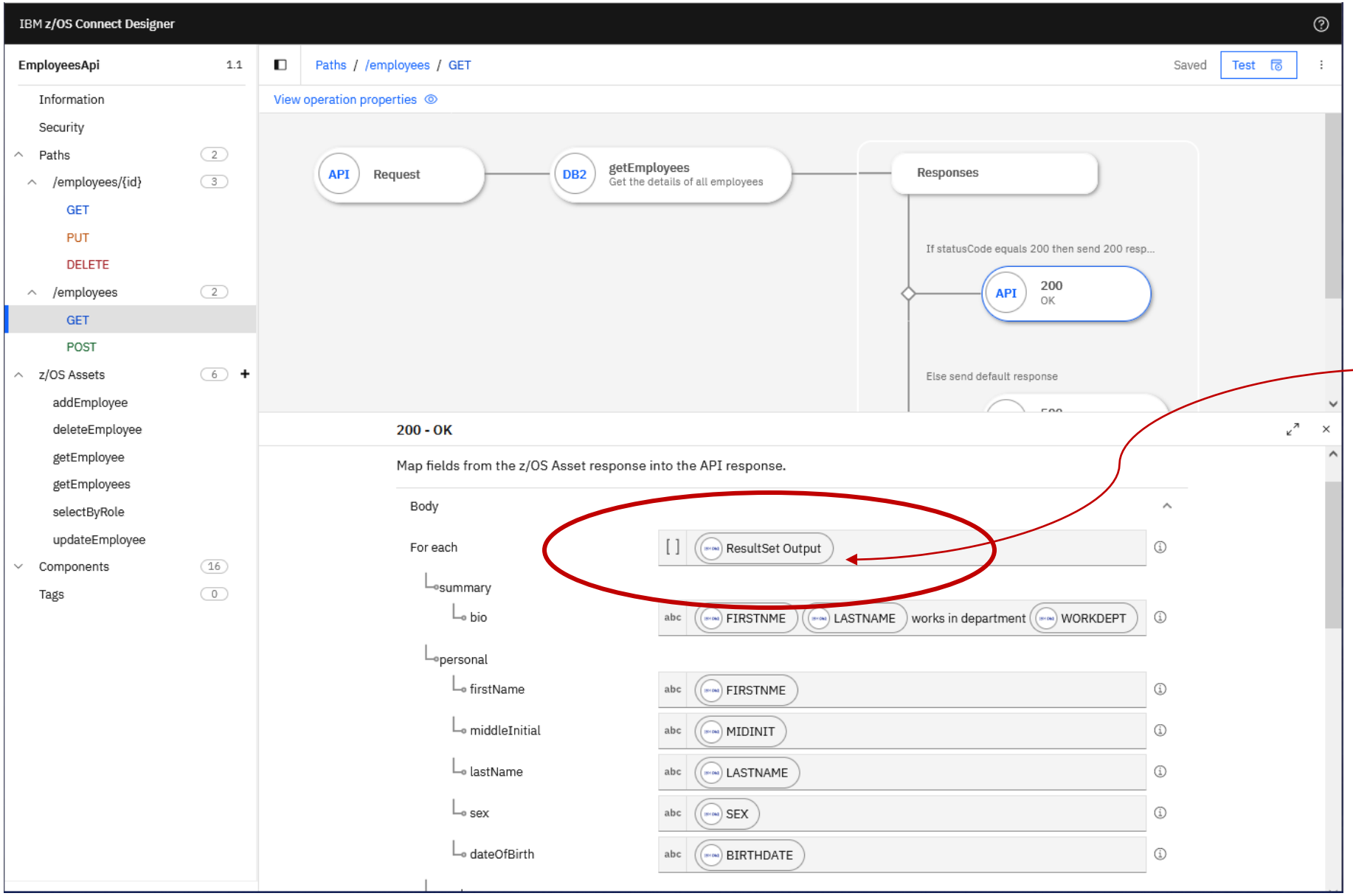


JSONata example showing the building of a complex message from the response.

Not recognized as a response message field. It is simply a text field..

mitchj@us.ibm.com

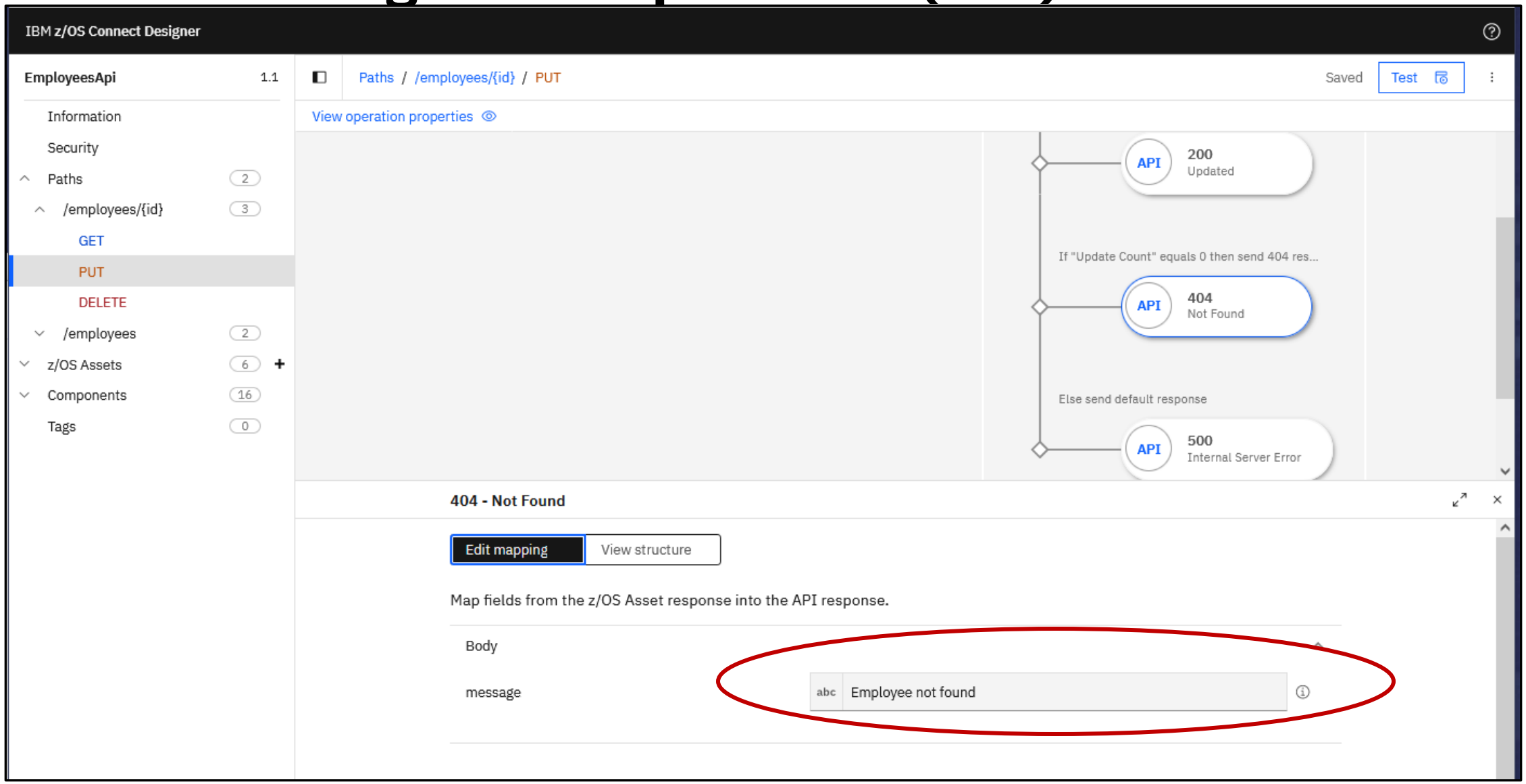
Map the z/OS resource response to the API's response message



ResutSet Output is a list or array of the results and the "For each" processes each element in the list.

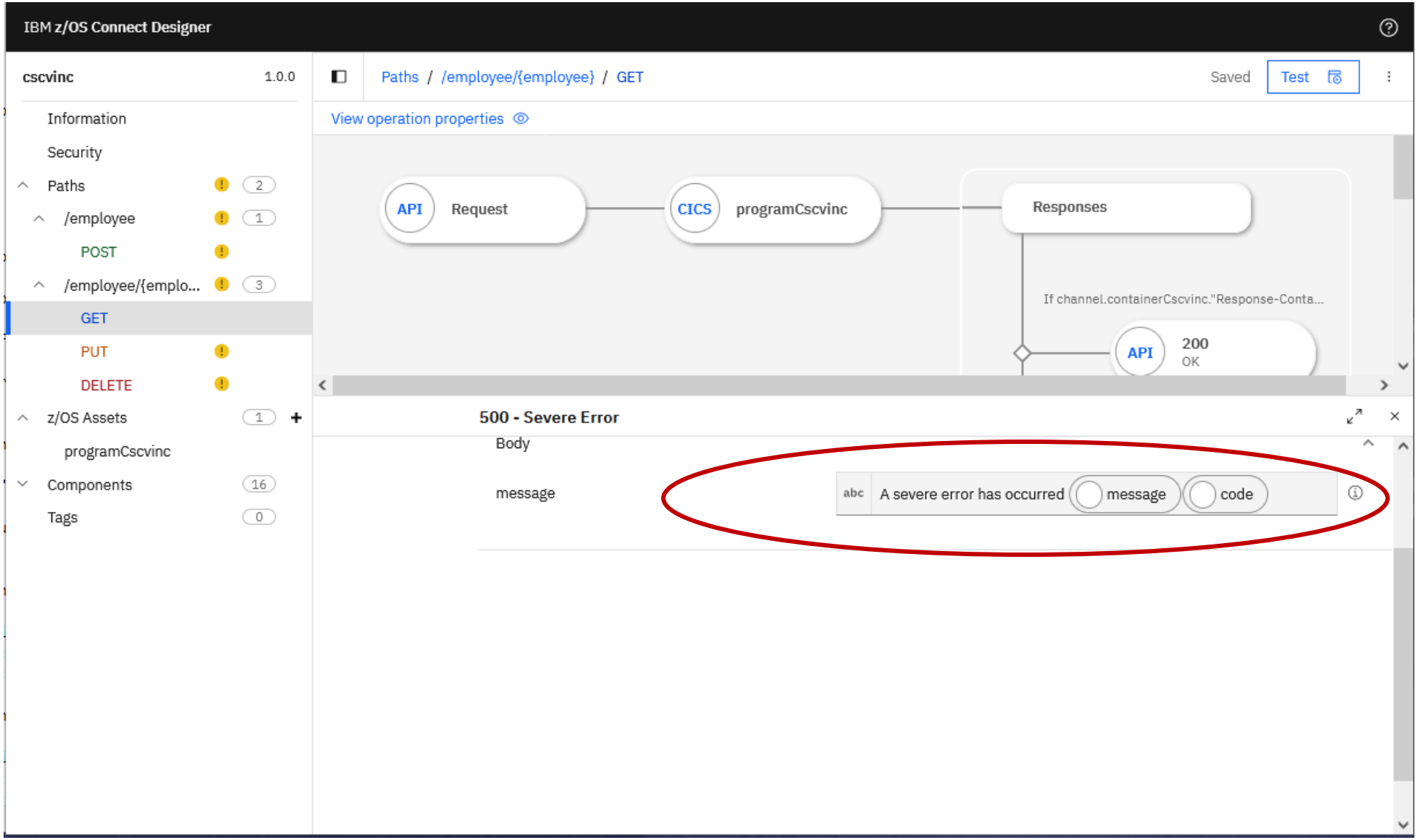
mitchj@us.ibm.com

z/OS Connect Designer for OpenAPI 3 (404)



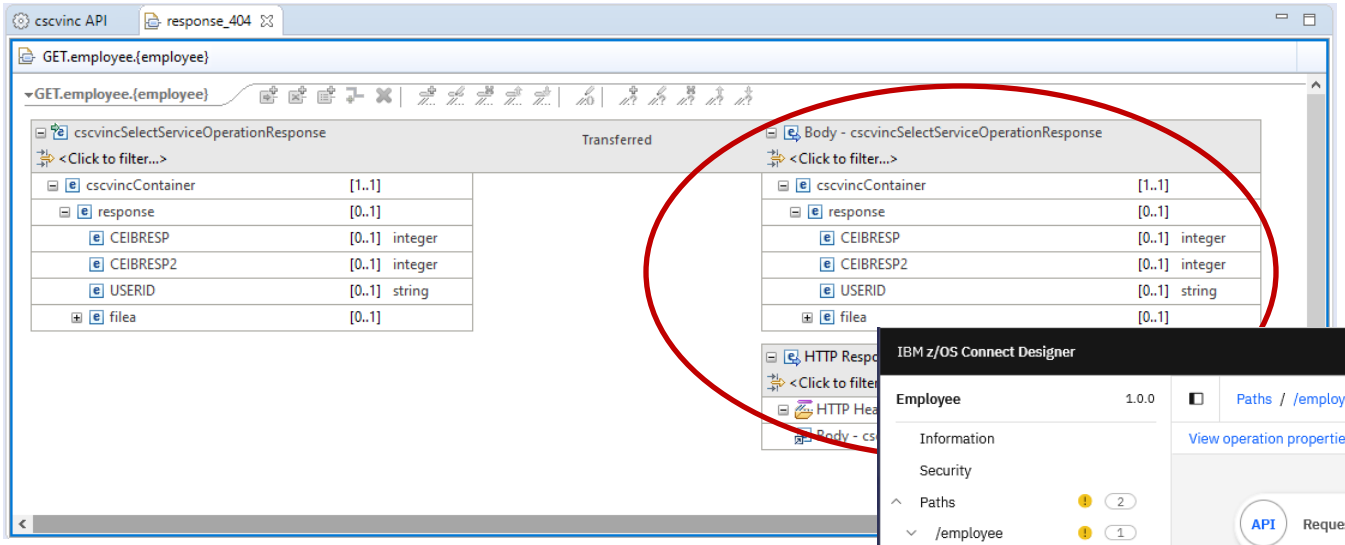
mitchj@us.ibm.com

z/OS Connect Designer for OpenAPI 3 (500)

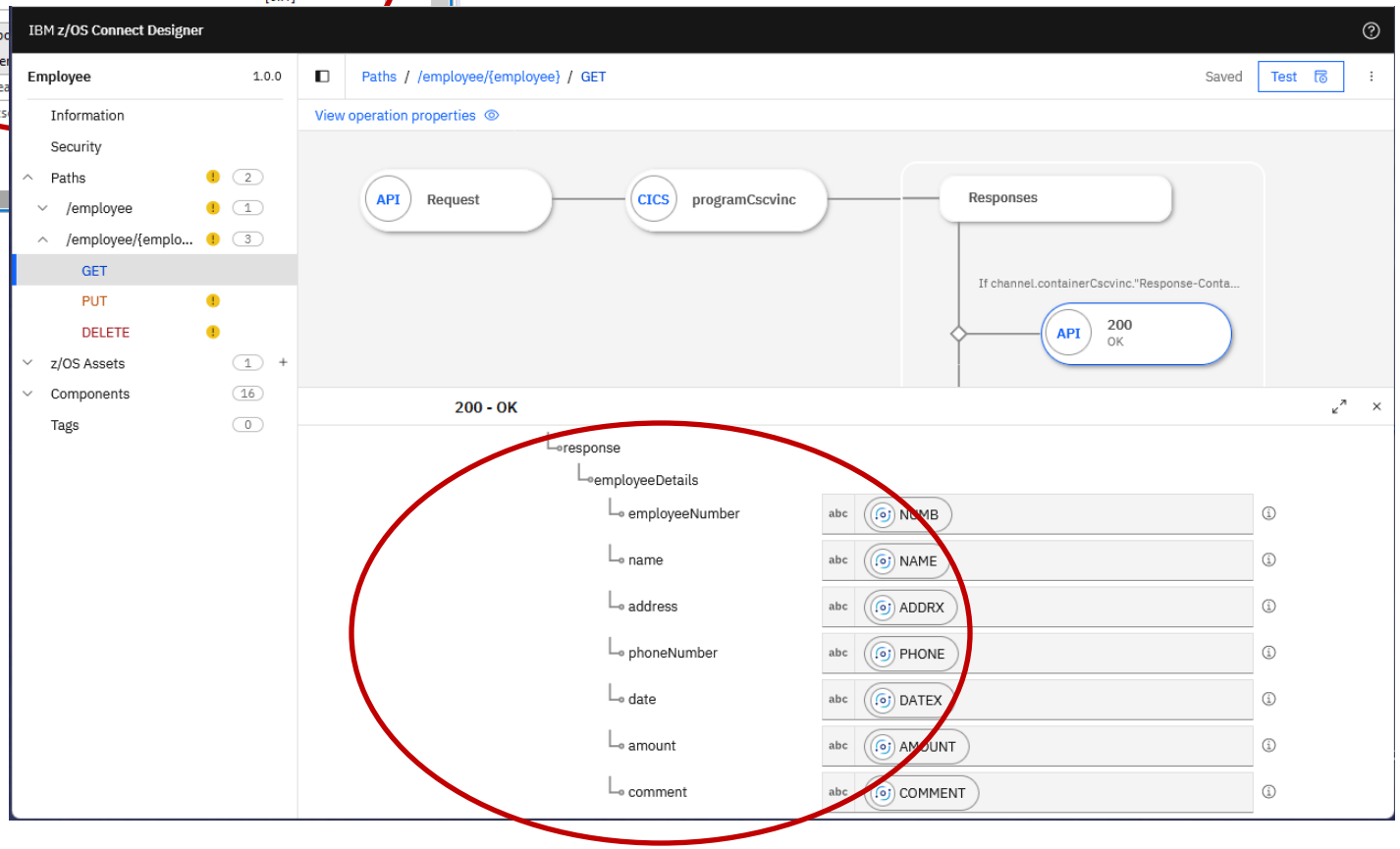


mitchj@us.ibm.com

Contrast the OpenAPI 2 versus an OpenAPI 3 Response Message



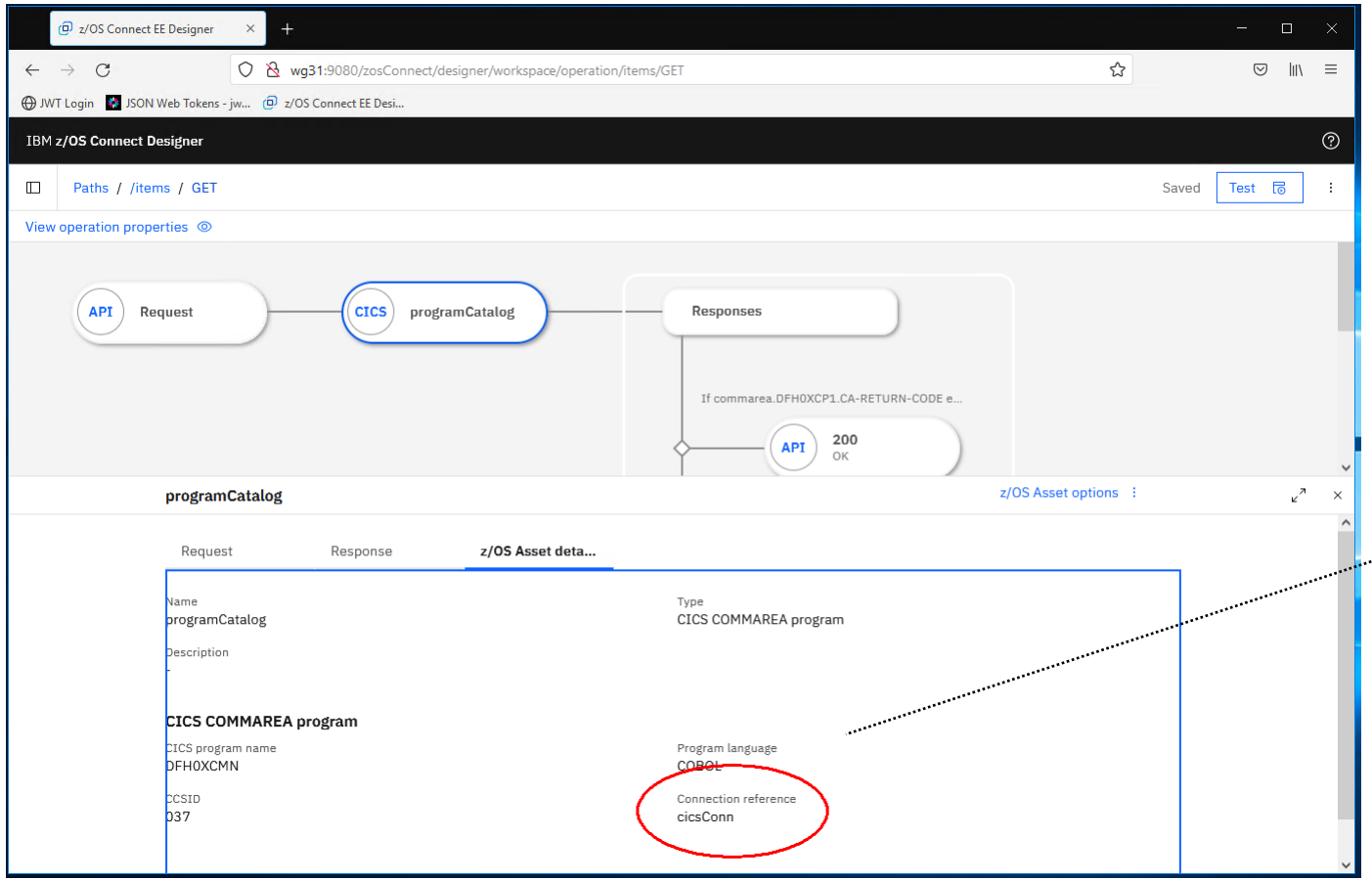
Eclipse based - z/OS Connect API Toolkit -
The contents of the API's response message are directly derived from the z/OS resource's response and usually a subset of the resource's response contents.



Web browser - z/OS Connect Designer – The contents of the API's response message are must be mapped to the z/O resource's response and JSONata coding, see URL https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=connect-what-is-jsonata for more information on JSONata.

mitchj@us.ibm.com

Server XML - Accessing a CICS program using IPIC

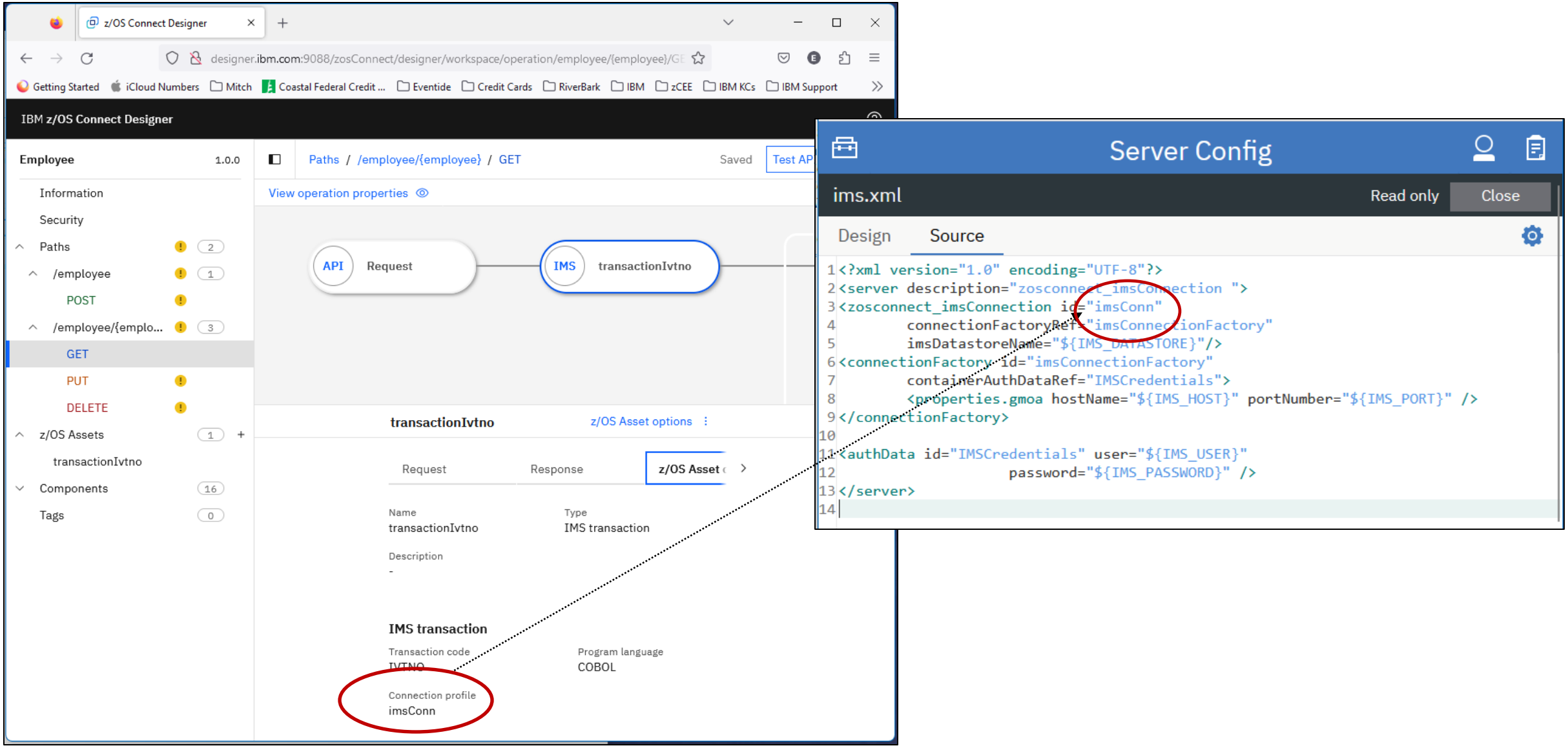


The connection references identifies a zosconnect_cicsIpicConnection configuration element which provides the connection details to a CICS region.

Note that this is a different CICS feature than the CICS feature for OpenAPI 2 server. The features are not interchangeable.

mitchj@us.ibm.com

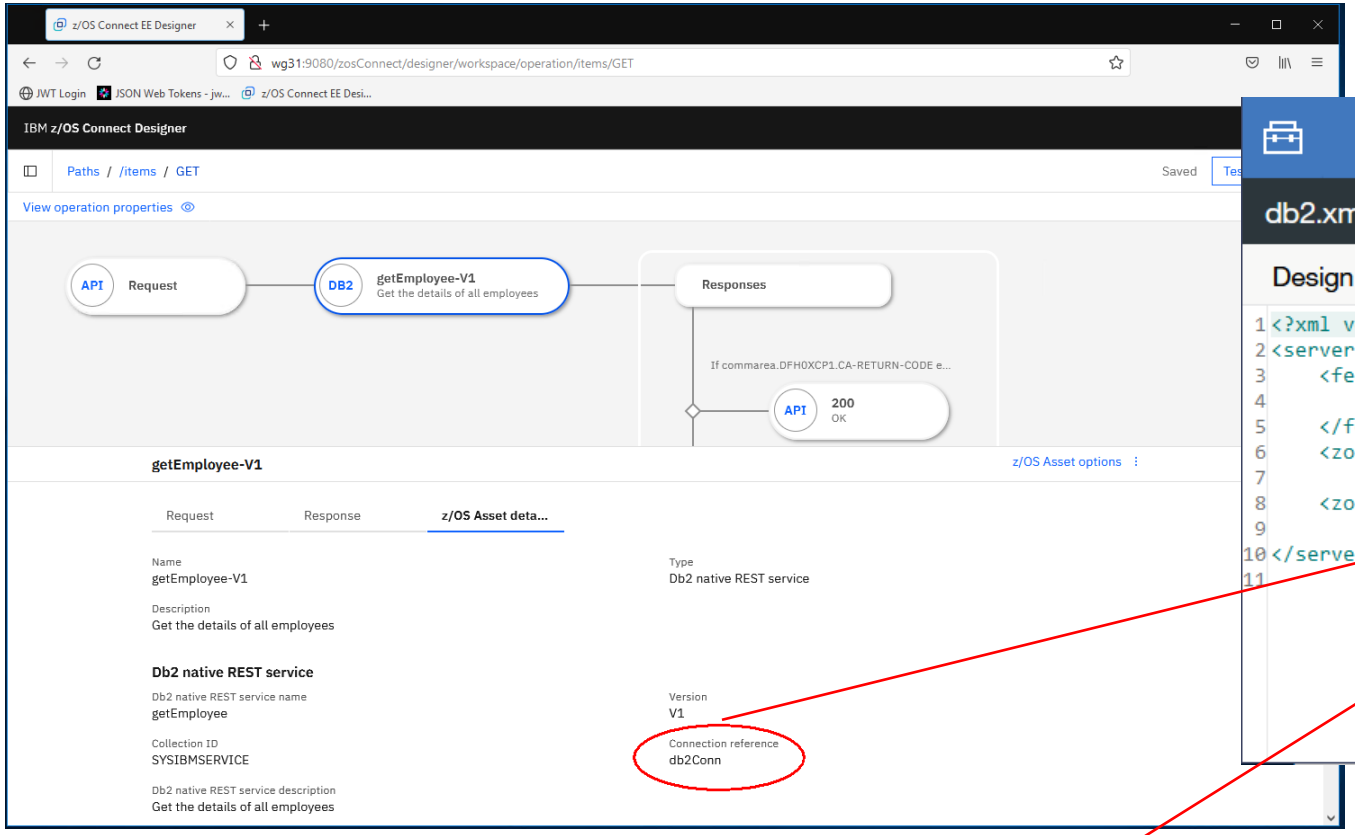
Server XML - Accessing an IMS transaction

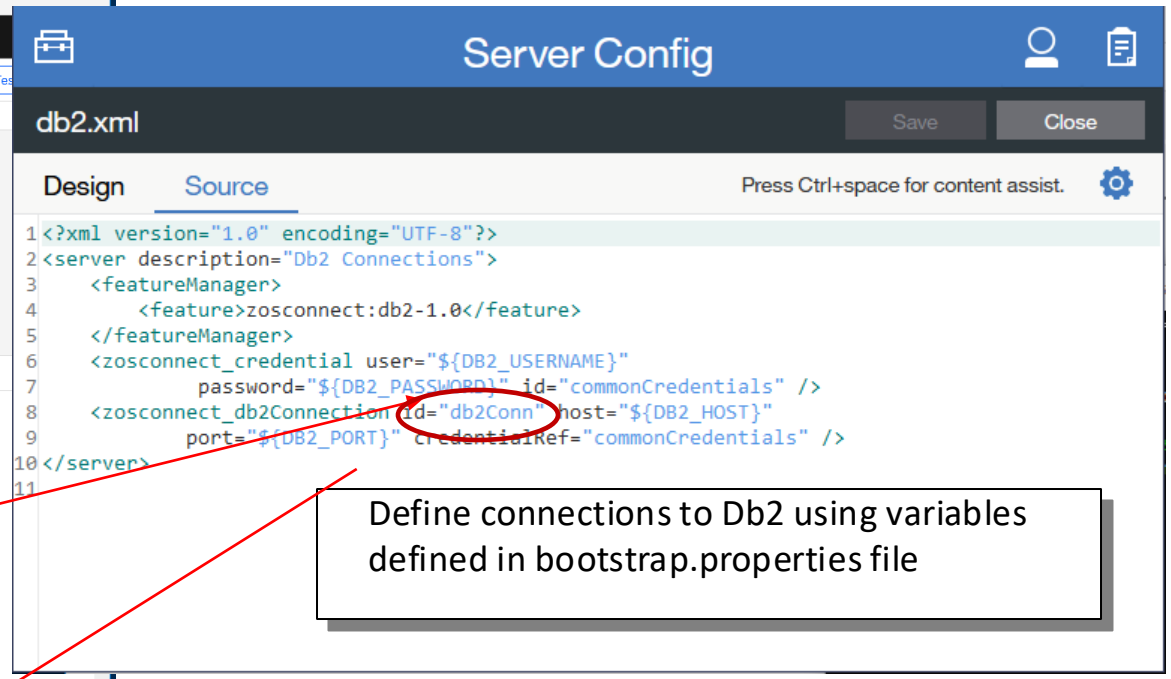


The connection references identifies a zosconnect_imsConnection element. Which provides the connection details to IMS.

mitchj@us.ibm.com

Server XML - Accessing a Db2 REST service





Define connections to Db2 using variables defined in bootstrap.properties file

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

The connection references identifies a zosconnect_db2Connection configuration element which provides the connection details to a DB2 DDF task.

Note that this is a new feature for accessing Db2.

mitchj@us.ibm.com

EJB roles for an OpenAPI 3 z/OS Connect API

```
openapi: 3.0.0
. . .
servers:
- url: /
x-ibm-zcon-roles-allowed:
- Manager
. . .
paths:
  /items:
    get:
     operationId: itemsGet
     . . .
  /items/{id}:
    get:
     . . .
     operationId: itemsIdGet
     x-ibm-zcon-roles-allowed:
        - Staff
 /orders:
     post:
     . . .
     operationId: ordersPost
     x-ibm-zcon-roles-allowed:
       - Staff
```

From the OpenApi document, the value for %role% would be either ***Manager*** *or* ***Staff****.*

So, the required SAF EJB roles to be defined would be:

- *BBGZDFLT.CatalogManager.**Manager***
- *BBGZDFLT.CatalogManager.**Staff***

```
REDFINE EJBROLE BBGZDFLT.CatalogManager.Manager
REDFINE EJBROLE BBGZDFLT.CatalogManager.Staff
```

Access to use the GET method to invoke /items would require read access to EJB role *BBGZDFLT.CatalogManager.Manager.*
Access to use the GET method to invoke /items/{id} and the POST method to invoke /orders would require read access to EJB role *BBGZDFLT.CatalogManager.Staff.*

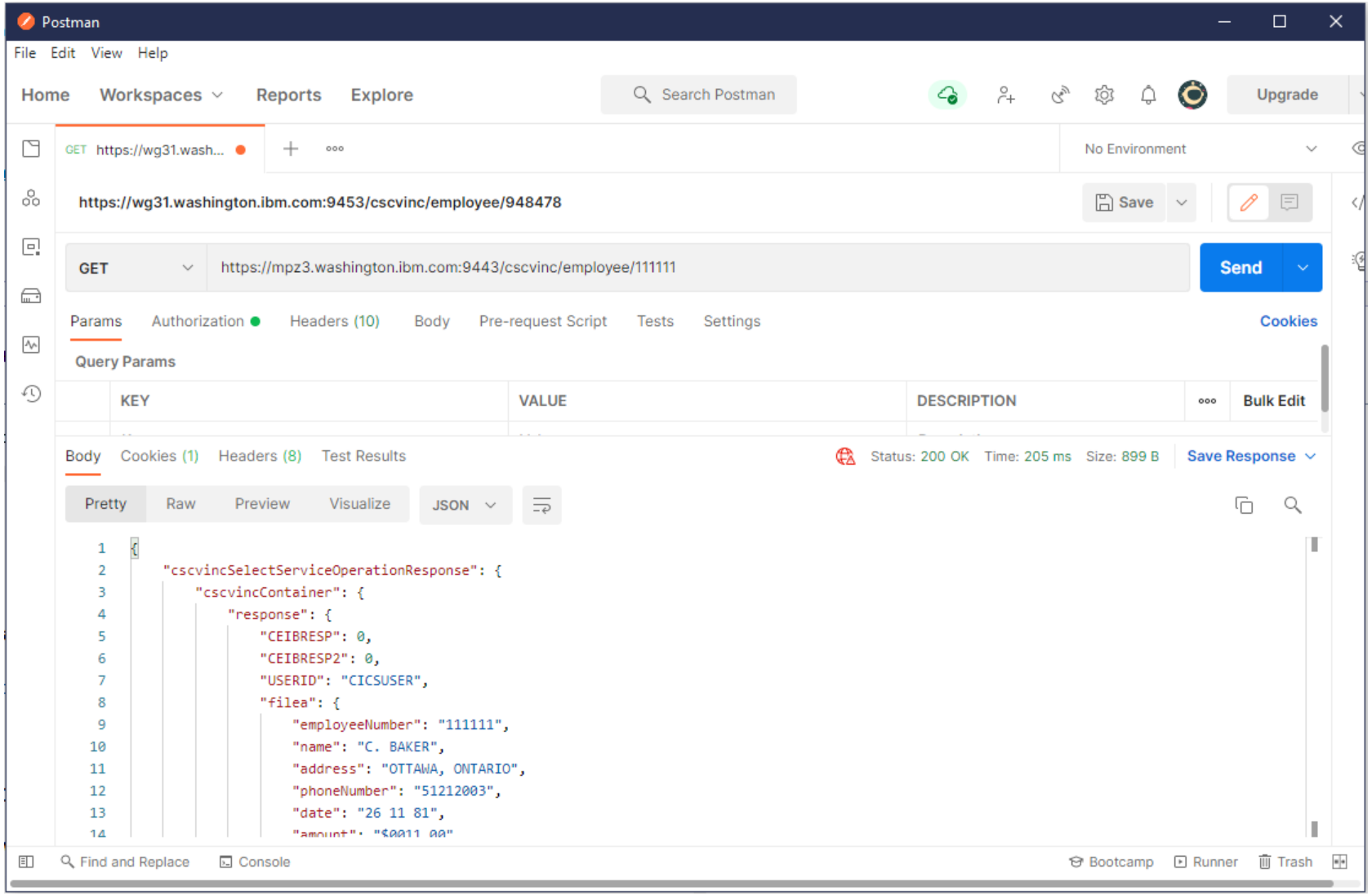
```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />

<webApplication id="CatalogManager" location="${server.config.dir}/apps/api.war" contextRoot="catalog"
name="CatalogManager"/>

<safRoleMapper profilePattern=%profilePrefix%.%resourceName%.%role%
```

mitchj@us.ibm.com

What REST test tooling is available?

mitchj@us.ibm.com

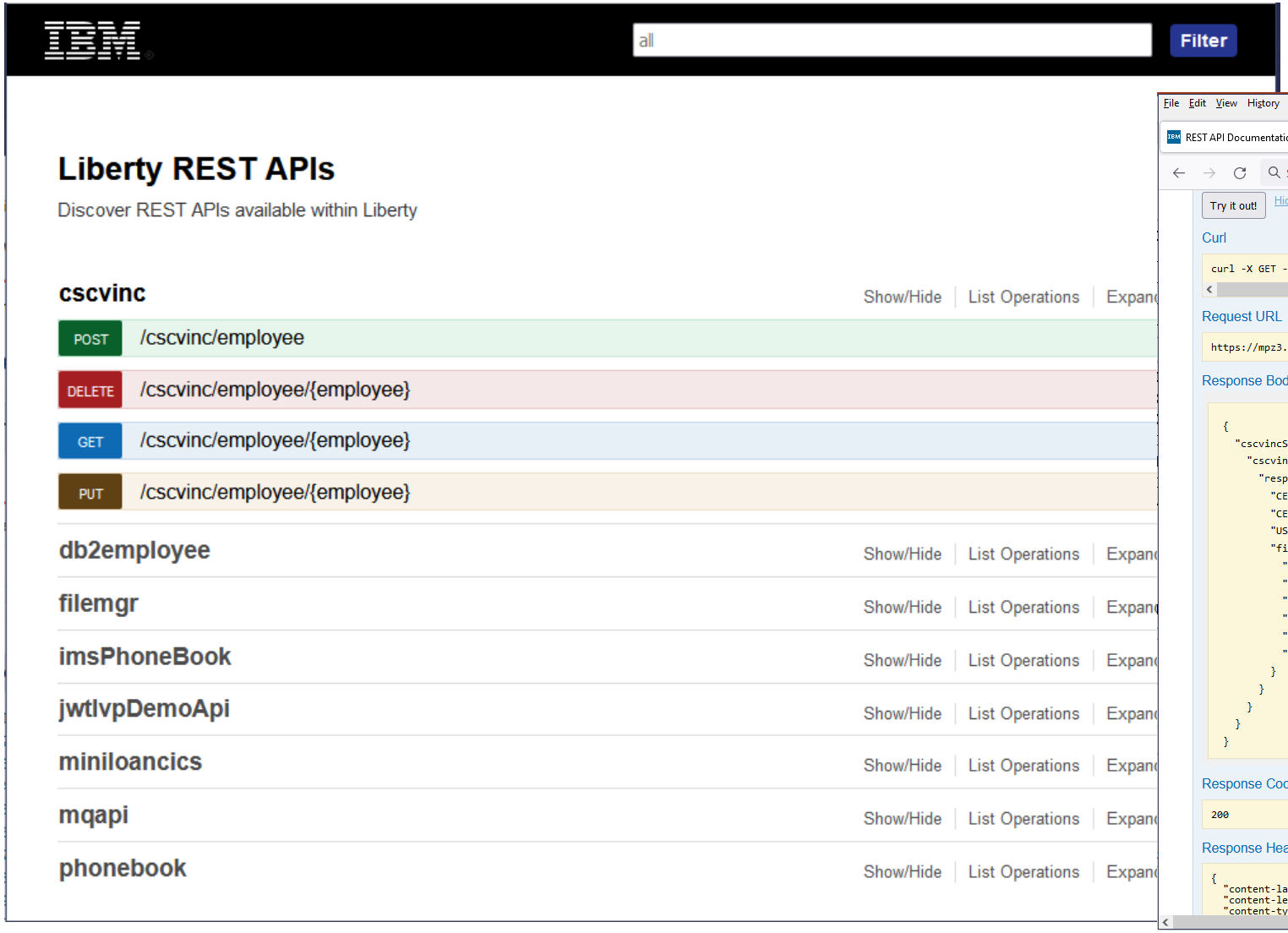
Testing an API with Postman

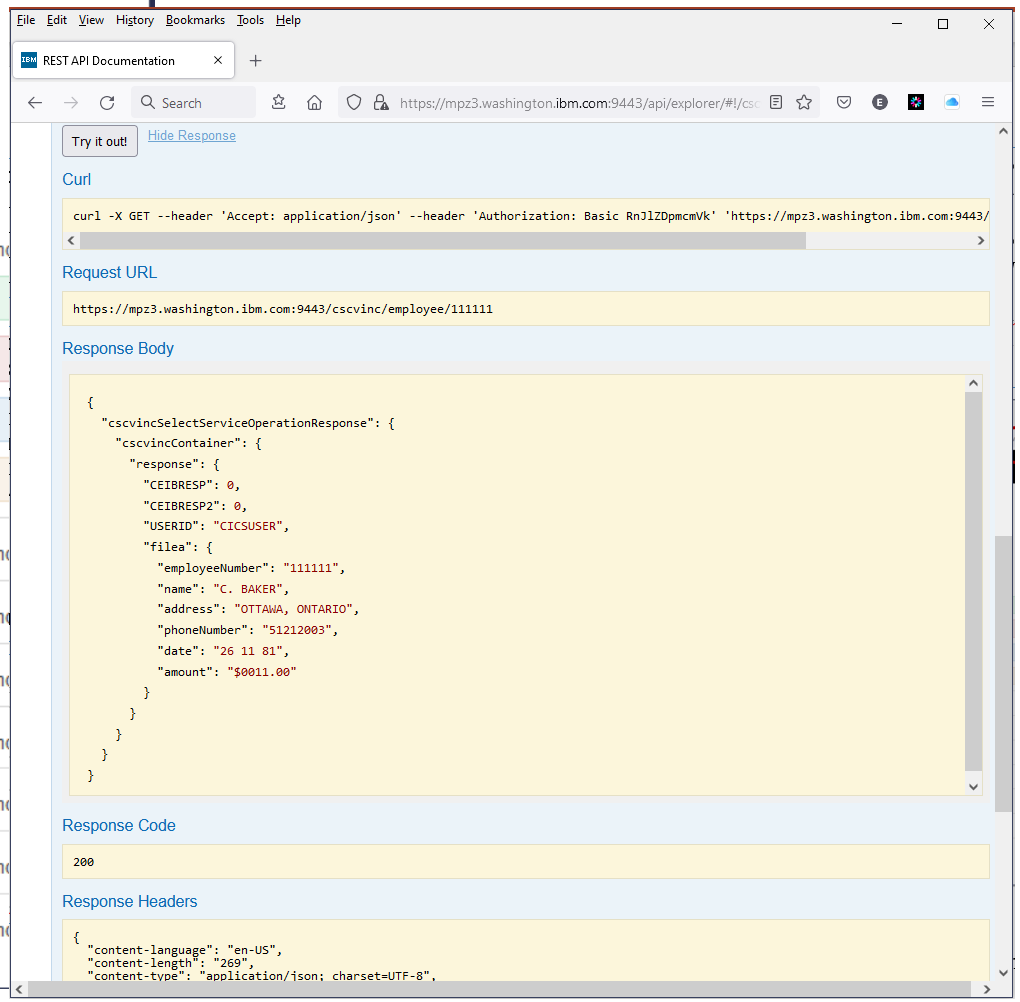


https://www.postman.com/downloads/

mitchj@us.ibm.com

Testing an API with the API Explorer*

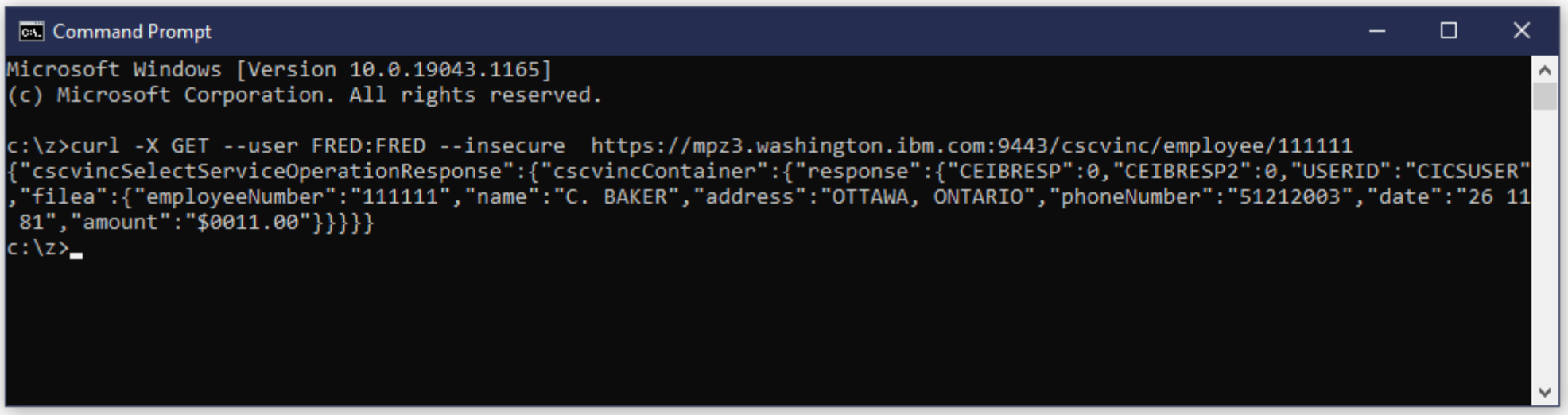




*with zCEE V3.0.48 or later

mitchj@us.ibm.com

Testing an API using the cURL tool



https://curl.se/download.html

Tech-Tip: Examples of curl in JCL and in scripts, bat or commands files

```
//*************************************************************
//*  SET SYMBOLS
//*************************************************************
//EXPORT EXPORT SYMLIST=(*)
// SET CURL= '/usr/lpp/rocket/curl'
//*************************************************************
//*  CURL Procedure
//*************************************************************
//CURL PROC
//CURL EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
// PEND
//*************************************************************
//*  STEP CURL - use cURL to deploy API cscvinc
//*************************************************************
//DEPLOY EXEC CURL
BPXBATCH SH export CURL=&CURL; +
$CURL/bin/curl -X PUT -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc?status=sto+
pped > null; +
$CURL/bin/curl -X DELETE -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc > null; +
$CURL/bin/curl -X POST -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
--data-binary @/u/johnson/cscvinc.aar +
--header "Content-Type: application/zip" +
https://wg31.washington.ibm.com:9445/zosConnect/apis
//*************************************************************
//*  STEP CURL - use cURL to invoke the API cscvinc
//*************************************************************
//INVOKE EXEC CURL
//SYSTSIN  DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X GET -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/cscvinc/employee/000100
```

```
@echo off
set TARGET=wg31.washington.ibm.com
set FILE=cscvinc_1.0.0
curl -X PUT --user user1:user1 --insecure
https://%TARGET%:9483/zosConnect/apiRequesters/%FILE%?status=stopped
curl -X DELETE --user user1:user1 --insecure
https://%TARGET%:9483/zosConnect/apiRequesters/%FILE%
curl -X POST --user user1:user1 --data-binary @%FILE%.ara  --header
"Content-Type: application/zip" --insecure
https://%TARGET%:9483/zosConnect/apiRequesters
curl -X GET -user user1:user1 --insecure
https://%TARGET%:9483/cscvinc/employee/000100
echo ''
```

Change the status of API to stopped

Delete or remove the API from the server

Deploy the API to the server

Execute the API

mitchj@us.ibm.com

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

/security

How is security implement?

mitchj@us.ibm.com

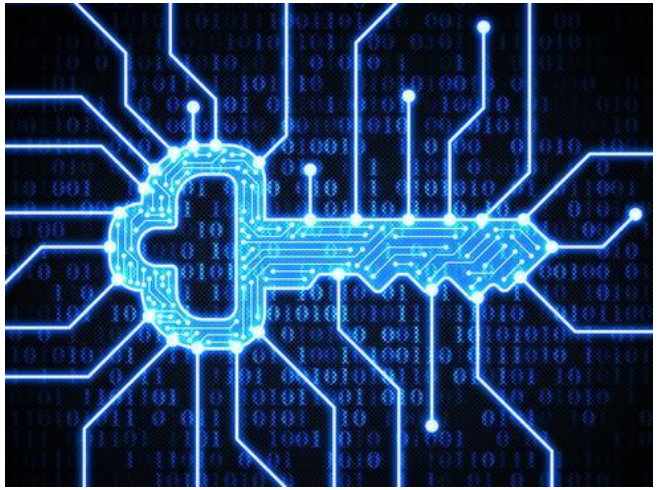
General security terms or considerations

Security involves

- Identifying who or what is requesting access (**Authentication)**
  - Basic Authentication
  - Mutual Authentication using Transport Layer Security (TLS), formerly known as SSL
  - Third Party Tokens

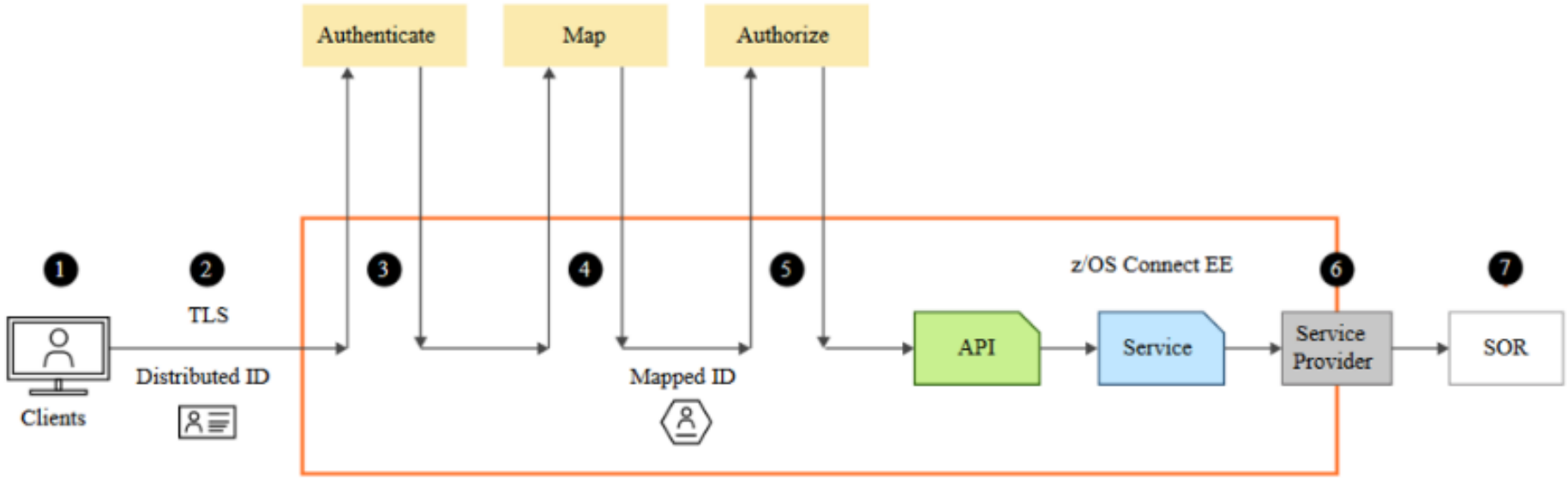
- Ensuring that the message has not been altered in transit (**Data Integrity)** and ensuring the confidentiality of the message in transit (**Encryption)**
  - TLS (encrypting messages and using a digital signature)

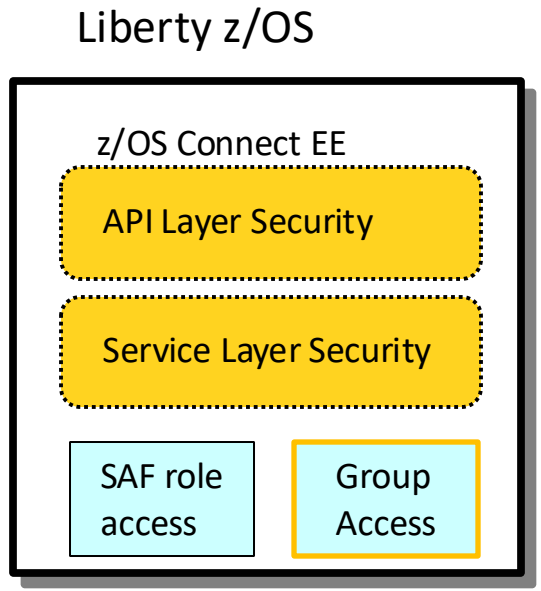
- Controlling access (**Authorization)**
  - Is the authenticated identity authorized to access to z/OS Connect
  - Is the authenticated identity authorized to access a specific API, Services, etc.



mitchj@us.ibm.com

Details of the OpenAPI2 z/OS Connect API Provider security flow

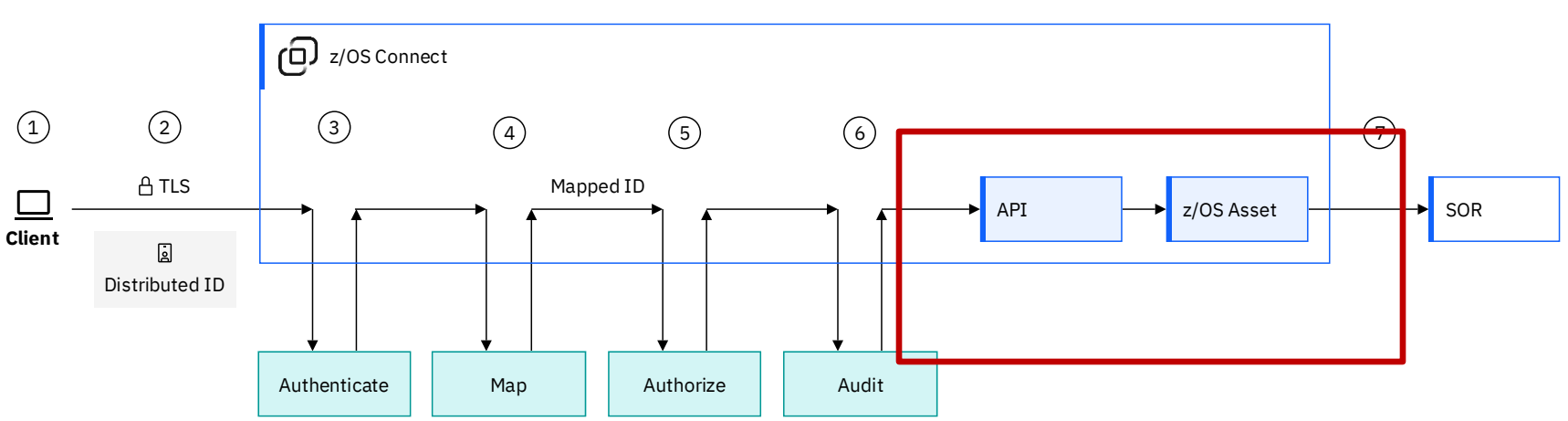


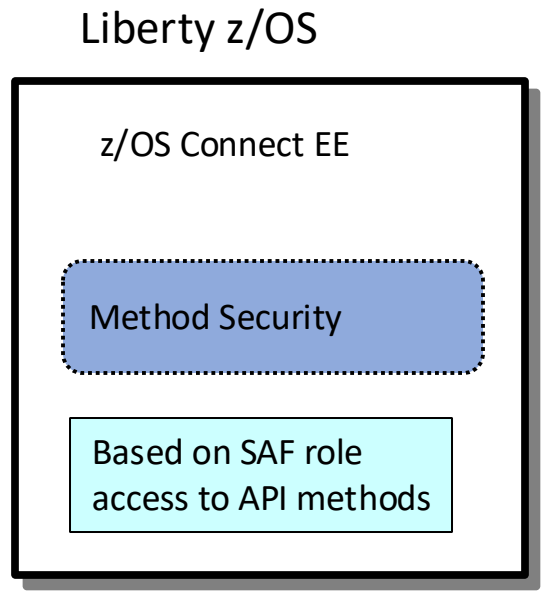


1. The credentials provided by the client
2. Secure the connection to the z/OS Connect server
3. Authenticate the client. This can be within the z/OS Connect server or by requesting verification from a third-party server
4. Map the authenticated identity to a user ID in the user registry
5. Authorize the mapped user ID to connect to z/OS Connect and optionally authorize user to invoke actions on APIs
6. Secure the connection to the System of Record (SoR) and provide security credentials to be used to invoke the program or to access the data resource
7. The program or database request may run in the SoR under the mapped ID

mitchj@us.ibm.com

Details of the OpenAPI 3 z/OS Connect API Provider security flow





The flow includes the following security steps that can be performed by IBM z/OS Connect. The credentials are provided by the client. These can be a user ID and password, a JWT, or a TLS certificate.

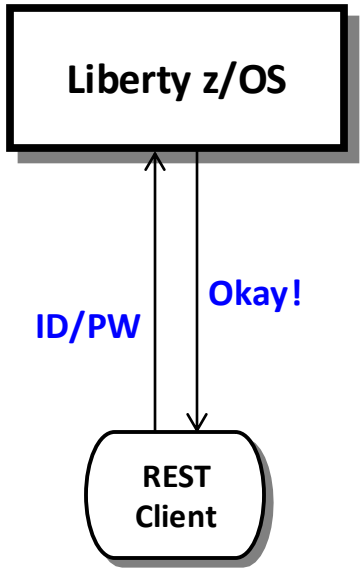
1. The credentials, including the identity, are passed on the connection between the client and IBM z/OS Connect. The identity is typically a distributed ID, such as an X.509 distinguished name and associated LDAP realm that originates from a remote system. Alternatively, the identity might be a SAF user ID. The data that is sent on the connection can be encrypted using TLS.
2. The client is authenticated. This can be within IBM z/OS Connect or by requesting verification from a third-party server.
3. The authenticated identity can be mapped to a user ID in the IBM z/OS Connect user registry.
4. The user is authorized to invoke the API operation if they have access to the role.
5. The API request is audited by using the Liberty Audit feature.
6. The authenticated user identity can be propagated to the System of Record (SoR) when the SoR supports this capability. Alternatively, the SoR connection can be configured to use a functional identity.

mitchj@us.ibm.com

Liberty Authentication

Several different ways this can be accomplished:

Basic Authentication



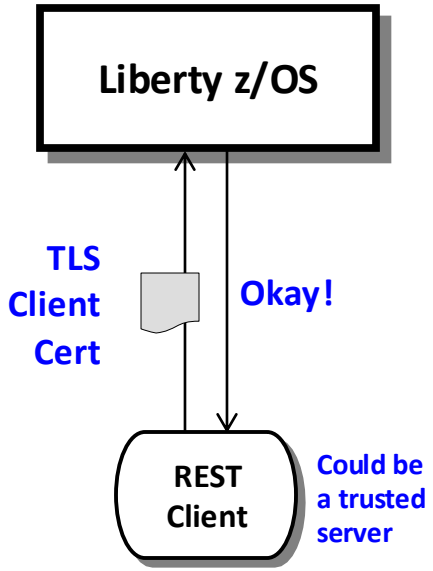
Server prompts for ID/PW

Client supplies ID/PW or ID/Passticket

Server checks registry:

- Basic (server.xml)
- LDAP
- SAF

Client Certificate



Server prompts for cert.

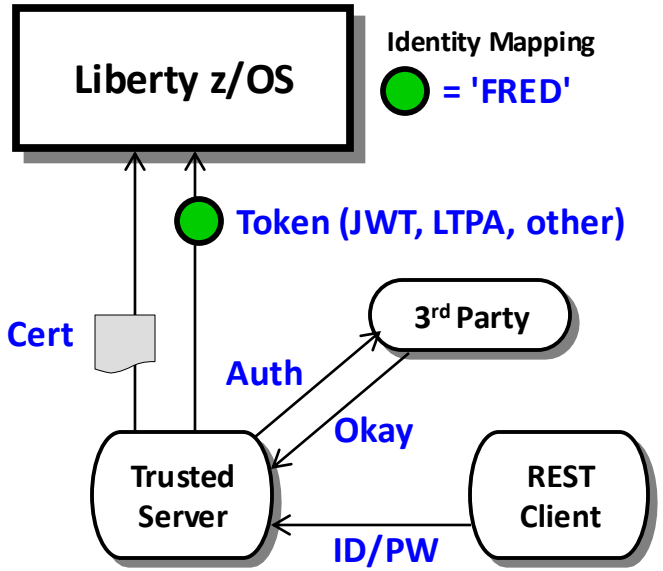
Client supplies certificate

Server validates cert and maps to an identity

Registry options:

- LDAP
- SAF

Third Party Authentication



Client authenticates to 3rd party sever

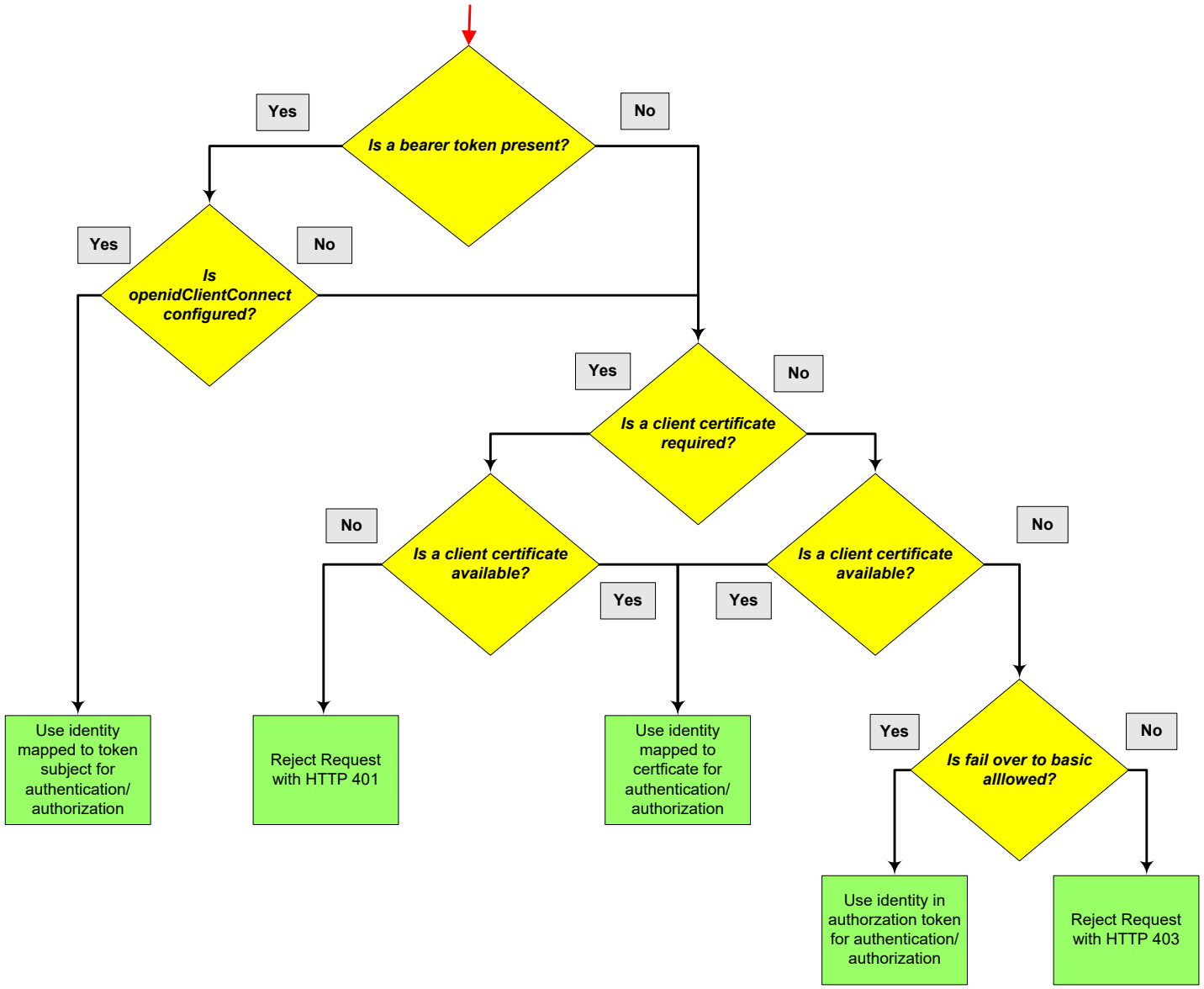
Client receives a trusted 3rd party token

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

Registry options:

- LDAP
- SAF

Authentication credential precedence order for determining authorization identity



Security for OpenAPI 2 z/OS Connect APIs are configured using z/OS Connect elements

The *requireAuth* attribute controls whether an inbound request must provide credentials using one of the three authentication methods, e.g., basic, client certificate, or third-party token.

```
<zosconnect_zosConnectManager
         requireAuth="true|false"
         requireSecure="true"/>
```

Globally, requires that users specify security credentials to be authenticated order to access APIs, services and API requesters, unless overridden on the specific resource definitions.

```
<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="catalog"
          requireAuth="true|false"
          requireSecure="true"/>
 </zosconnect_zosConnectAPIs>

 <zosconnect_services>
   <service id="selectByEmployee"
       name="selectEmployee"
       requireAuth="true|false"
       requireSecure="true"/>
</zosconnect_services>

<zosconnect_apiRequesters>
    requireAuth="true|false"
    <apiRequester name="cscvincapi_1.0.0"
        requireAuth="true|false"
        requireSecure="true"/>
 </zosconnect_apiRequesters>
```

Requires that users specify security credentials to be authenticated in order to access the API.

Requires that users specify security credentials to be authenticated in order to directly access the service. This attribute is ignored when the service is invoked from an API, then only the API requireAuth attribute is relevant.

Requires that users specify security credentials to be authenticated in order to access all API requesters. If the requireAuth attribute is not set, the global setting on the zosconnect_zosConnectManager element is used instead, unless the requireAuth attribute is overridden on the specific API requester.

mitchj@us.ibm.com

Security for OpenAPI 3 z/OS Connect APIs is configured using Liberty elements

```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />

<webApplication id="catalogManager" name="catalogManager"
              location="${server.config.dir}/apps/api.war" contextRoot="/catalogManager" />

<safRoleMapper profilePattern=%profilePrefix%.%resourceName%.%role%
```

The *name* attribute of the webApplication for the deployed WAR file determines the name of the EJBRoles used manage access to the API's methods.

*From the example OpenApi document, the value for %role% would be either **Manager** or **Staff**.*

So, the required SAF EJB roles to be defined would be:

- *BBGZDFLT.catalogManager.**Manager***
- *BBGZDFLT.catalogManager.**Staff***

```
REDFINE EJBROLE BBGZDFLT.catalogManager.Manager
REDFINE EJBROLE BBGZDFLT.catalogManager.Staff
```

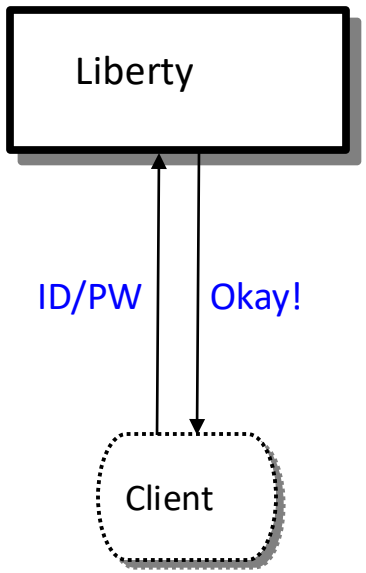
Access to use the GET method to invoke /items would require read access to EJB role *BBGZDFLT.catalogManager.Manager.*
Access to use the GET method to invoke /items/{id} and the POST method to invoke /orders would require read access to EJB role *BBGZDFLT.catalogManager.Staff.*

mitchj@us.ibm.com

Authentication - Basic Authentication

Several different ways this can be accomplished:

Basic Authentication

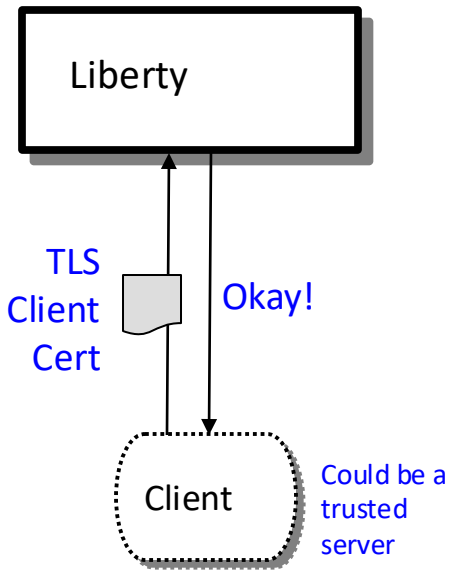


Client supplies an identity/password or an identity/PassTicket

Server checks registry:

- **Basic (server.xml)**
- **SAF**

Client Certificate



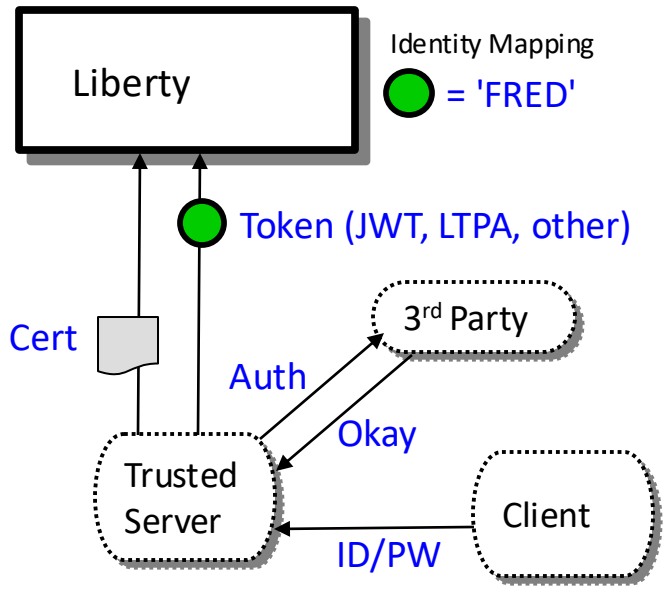
Client supplies client certificate

Server validates client certificate and maps it to an identity

Registry options:

- SAF

Third Party Authentication



Client authenticates to 3rd party sever

Client receives a trusted 3rd party token

Token flows to server and is mapped to an identity

Registry options:

- We do not need to know the details.

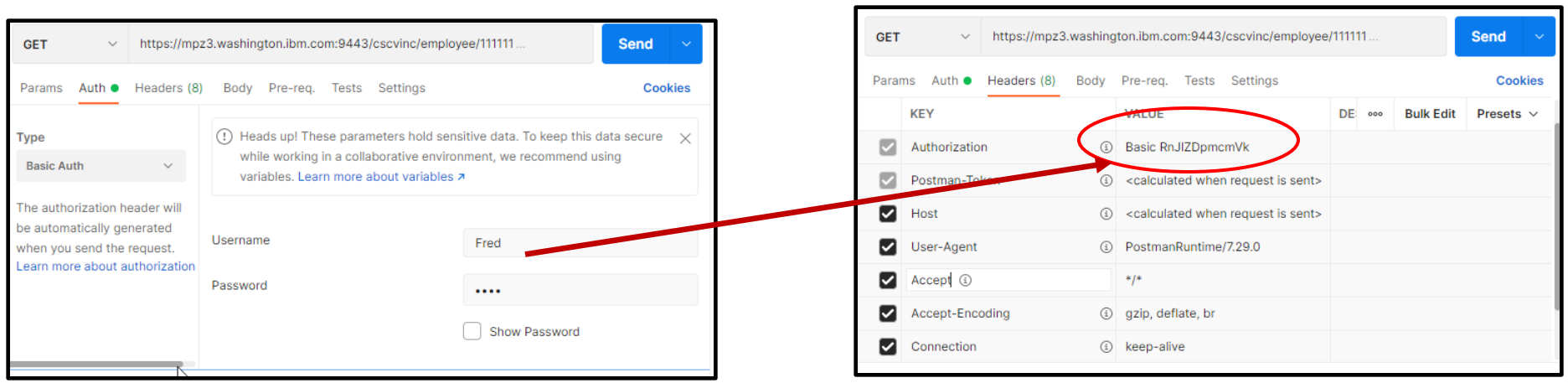
mitchj@us.ibm.com

Basic authentication – The client provides an identity and password

- The required server XML security configuration:

```
<featureManager>
     <feature>appSecurity-2.0</feature>
     <feature>zosSecurity-1.0</feature>
 </featureManager>

 <webAppSecurity allowFailOverToBasicAuth="true" />

 <safRegistry id="saf" />
 <safAuthorization racRouteLog="ASIS" />
 <safCredentials unauthenticatedUser="WSGUEST"
    profilePrefix="BBGZDFLT" />
```

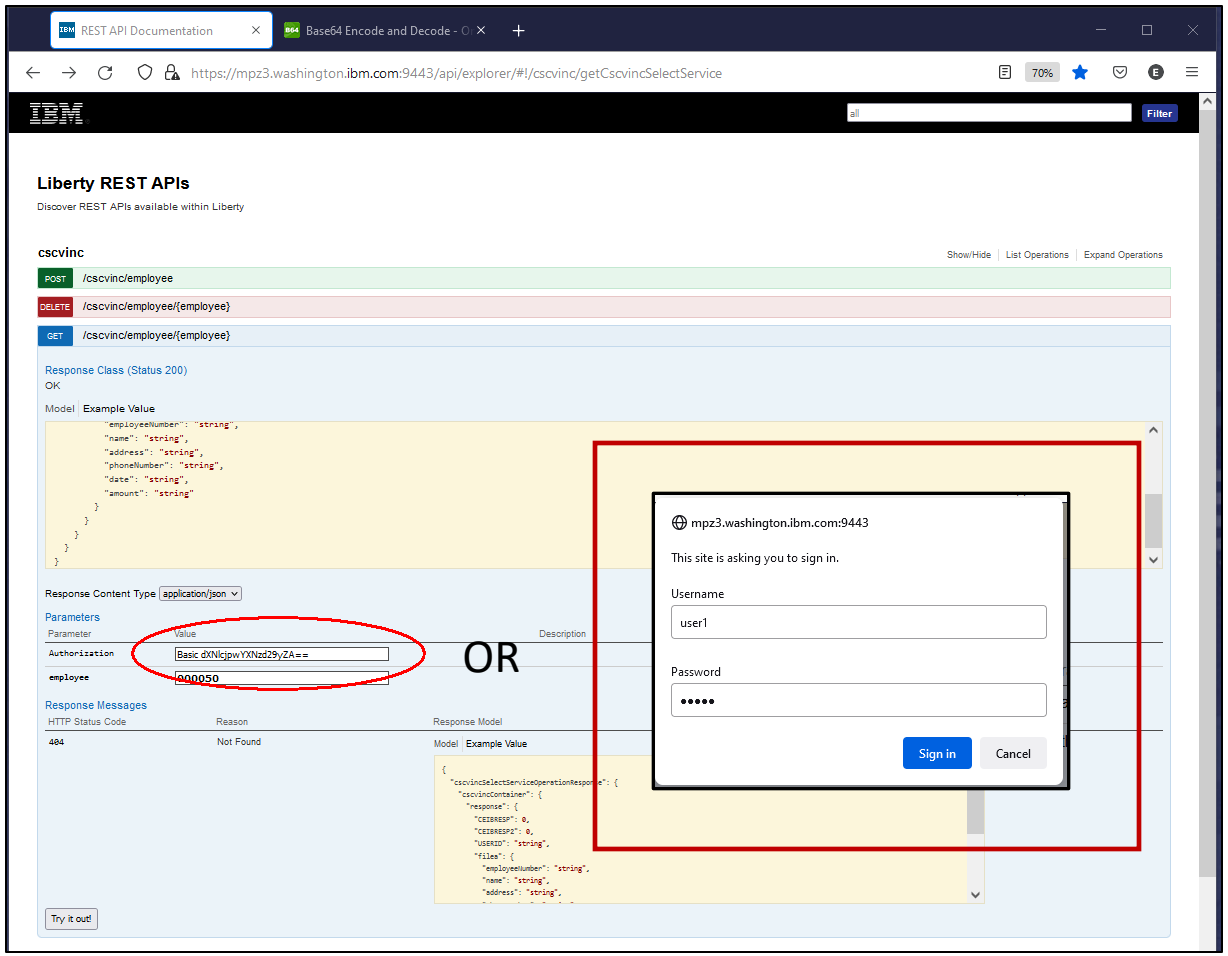
- When sending a request to a Liberty server running z/OS Connect, basic authentication information (identity and password) is provided in the HTTP header in a Basic *Authorization t*oken with the identity and password encoded using Base64.
  - An example with Postman:

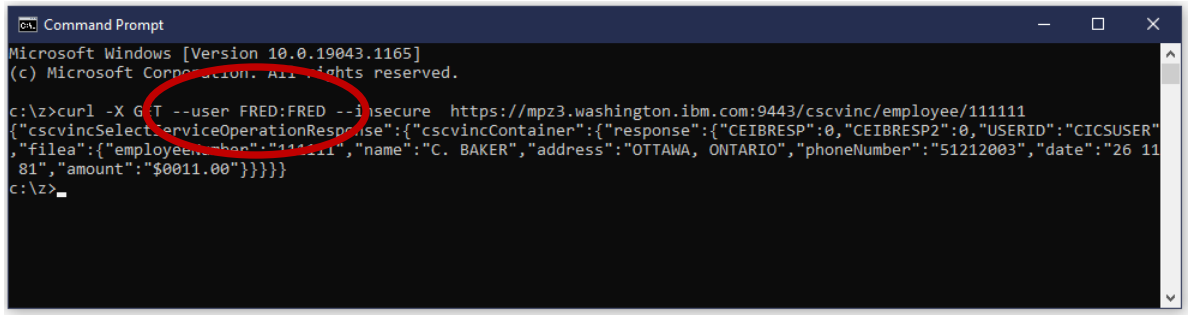


https://www.base64encode.org/

There are multiple ways to provide an identity and password

- When sending a request to a Liberty server running z/OS Connect, basic authentication information (identity and password) is provided in the HTTP header in a Basic *Authorization t*oken with the identity and password encoded or formatted using Base64.
  - Examples using the API Explorer feature , cURL, and a Java client.





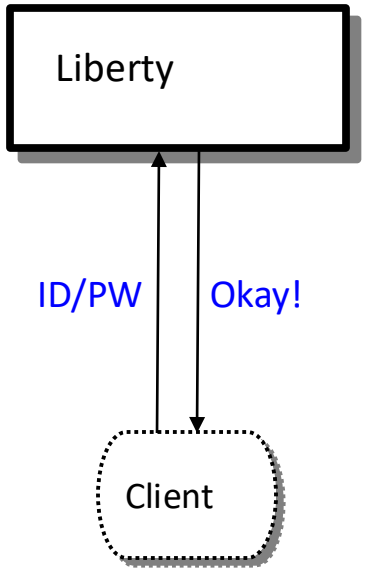
mitchj@us.ibm.com

https://www.base64encode.org/

Authentication - TLS Mutual Authentication

Several different ways this can be accomplished:

Basic Authentication

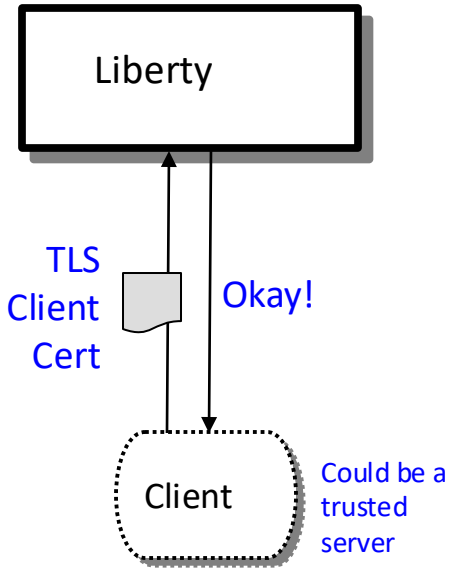


Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

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

Client Certificate



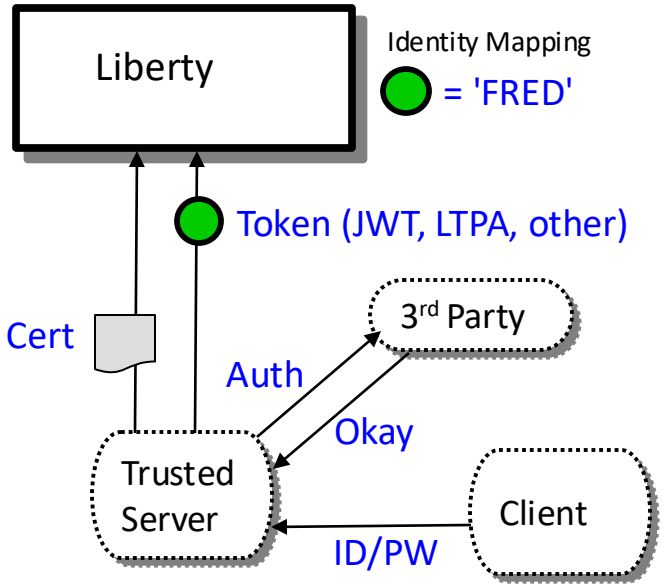
Server prompts for client certificate.

Client supplies personal certificate

Server validates client certificate and maps it to an identity

Registry options:
- **SAF**

Third Party Authentication



Client authenticates to 3rd party sever

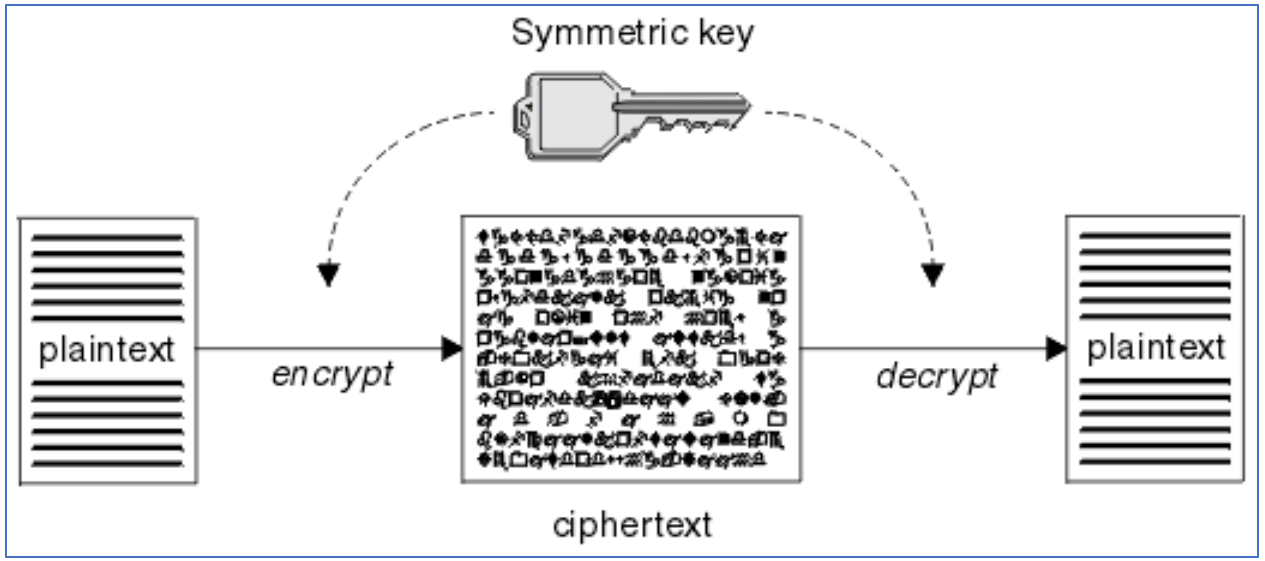
Client receives a trusted 3rd party token

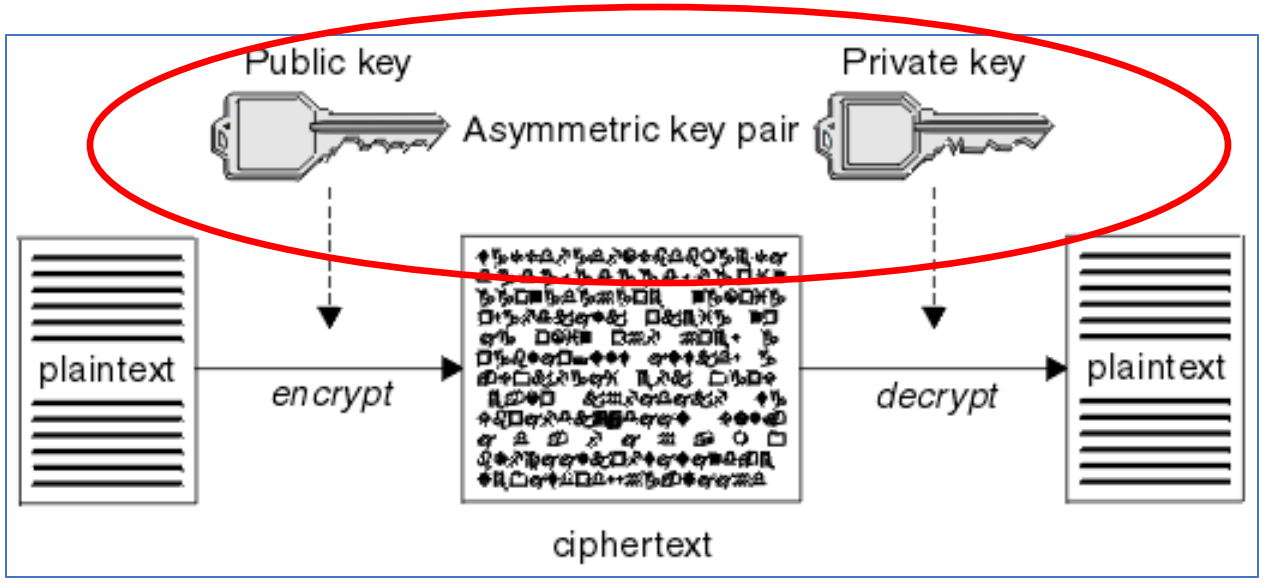
Token flows to Liberty z/OS and is mapped to an identity

Registry options:
- We may not need to know these details.

mitchj@us.ibm.com

Tech-Tip: Symmetric key v. Asymmetric key pairs





A symmetric key is a key shared by the endpoints. Both endpoints use the same key to encrypt and decrypt messages.

An asymmetric key pair is the preferred solution. There is no risk of compromise by sending a symmetric or shared key outside of a protected communication flow.

A message encrypted with a public key can only be decrypted by endpoint that has the private key. The privacy of the messages is ensured.

If an endpoint can successfully decrypt a message message encrypted received with a private key, the endpoint sending the message has successfully asserted its validity by proving it has the private used to encrypt the message.

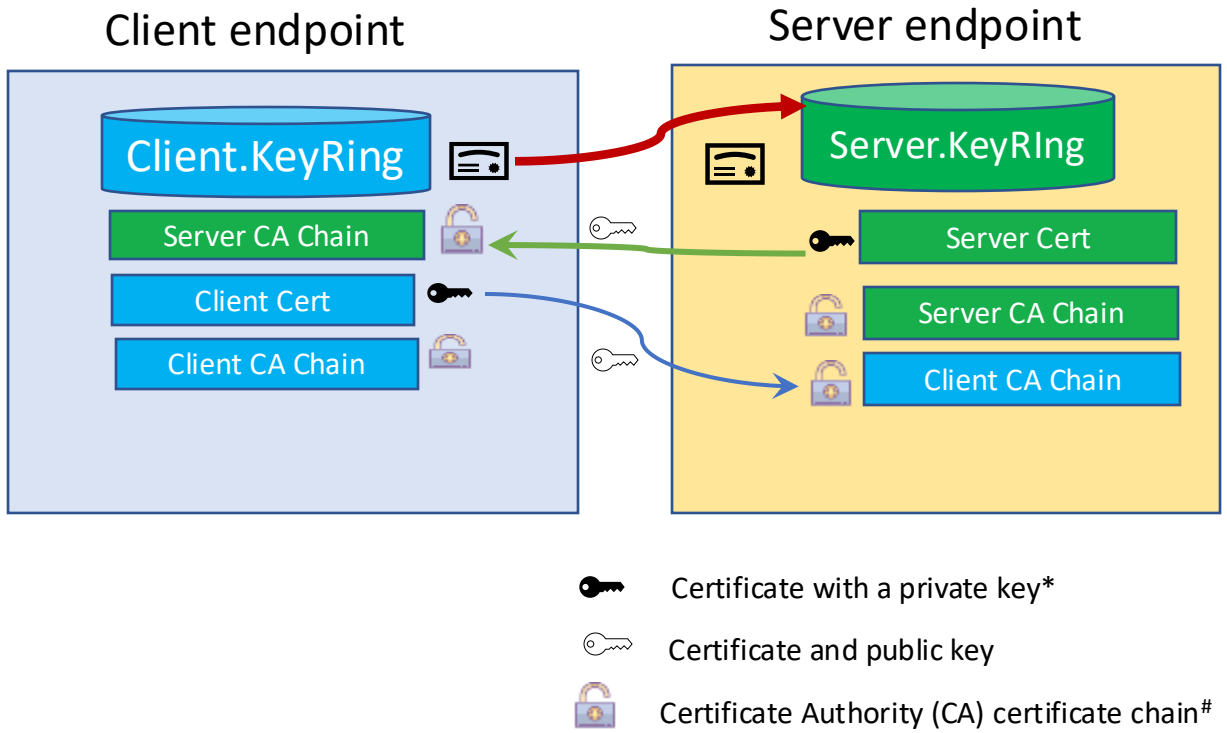
mitchj@us.ibm.com

The basic TLS Handshake Flow (HTTPS)

The HTTPS protocol involves a TLS handshake –

Server Authentication (always occurs when HTTPS is the protocol)

Mutual Authentication (optional, at the request of the server endpoint)



*For server and/or mutual authentication to work, the endpoint sending the client certificate must use a personal certificate with a private key. The private key is required to decrypt (or encrypt) a message digest that is sent from the other endpoint during the handshake flow. Generation of a message digest also requires access to the CA certificate used to sign the certificate.

#Refers to the set or of certificates used to issue the server or client personal certificate including any intermediate certificates up to and including the root CA.

mitchj@us.ibm.com

Liberty JSSE (HTTPS) server XML configuration

```
<!-- Enable features -->
   <featureManager>
        <feature>transportSecurity-1.0</feature>
   </featureManager>

   <sslDefault sslRef="DefaultSSLSettings"
     outboundSSLRef="OutboundSSLSettings" />

   <ssl id="DefaultSSLSettings"
      keyStoreRef="CellDefaultKeyStore"
      trustStoreRef="CellDefaultKeyStore"
      clientAuthenticationSupported="true"
      clientAuthentication="true"/>

   <keyStore id="CellDefaultKeyStore"
      location="safkeyring:///Liberty.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />

   <ssl id="OutboundSSLSettings"
      keyStoreRef="OutboundKeyStore"
      trustStoreRef="OutboundKeyStore"/>

   <keyStore id="OutboundKeyStore"
      location="safkeyring:///zCEE.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />
```

SSL repertoires

Key ring for server certificate send to for clients

Key ring for client connections to server endpoints

OpenAPI 2

```
<zosconnect_zosConnectManager
            requireAuth="true"
            requireSecure="true|false"/>

  <zosconnect_zosConnectAPIs>
     <zosConnectAPI name="catalog"
            requireAuth="true"
            requireSecure="true|false"/>
  </zosconnect_zosConnectAPIs>

  <zosconnect_services>
    <service id="selectByEmployee"
        name="selectEmployee"
        requireAuth="true"
        requireSecure="true|false"/>
 </zosconnect_services>

 <zosconnect_apiRequesters>
     requireAuth="true|false"
     <apiRequester name="cscvincapi_1.0.0"
         requireAuth="true"
         requireSecure="true|false"/>
  </zosconnect_apiRequesters>
```

safkeyring:///**KeyRing** v safkeyring:**//owner/KeyRing**

Tech/Tip: Regarding *clientAuthentication* and *clientAuthenticationSupported*. Understand the implications of the interactions between these attributes. There may instances where you want to use HTTPS, but not always with mutual authentication Consider setting *clientAuthentication* to false when setting *clientAuthenticationSupported* to true.

mitchj@us.ibm.com

The Liberty JSSE server XML configuration for outbound connections

```
<!-- Enable features -->
   <featureManager>
        <feature>transportSecurity-1.0</feature>
   </featureManager>

   <ssl id="cicsTLSSettings"
      keyStoreRef="CICSKeyStore"
      trustStoreRef="CICSKeyStore"
      clientKeyAlias="Liberty Client Cert"/>
   <keyStore id="CICSKeyStore"
      location="safkeyring:///Liberty.CICS.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />
   <ssl id="db2TLSSettings"
      keyStoreRef="Db2KeyStore"
      trustStoreRef="Db2KeyStore"
      clientKeyAlias="Liberty Client Cert"/>
   <keyStore id="Db2KeyStore"
      location="safkeyring:///Liberty.Db2.KeyRing"
      password="password" type="JCERACFKS"
     fileBased="false" readOnly="true" />
  <ssl id="otherTLSSettings"
      keyStoreRef="OtherKeyStore"
      trustStoreRef="OtherKeyStore">
      outboundConnection
        host="wg31.washington.ibm.com"
        port="9555"
        clientCertificate="Client Cert"/>
   </ssl>
   <keyStore id="OtherKeyStore"
      location="safkeyring:///Other.KeyRing"
      password="password" type="JCERACFKS"
      fileBased="false" readOnly="true" />
```

```
<sslDefault sslRef="DefaultSSLSettings"
            outboundSSLRef="OutboundSSLSettings" />

<zosconnect_authorizationServer sslCertsRef="SSL repertoire"/>
<zosconnect_cicsIpicConnection sslCertsRef="cicsTLSSettings"/>
<zosconnect_db2Connection sslCertsRef="db2TLSettings"> *
<zosconnect_endpointConnect sslCertsRef= "SSL repertoire"/>
<zosconnect_zosConnectRestClient sslCertsRef="SSL repertoire"/>
<zosconnect_zosConnectServiceRestClientConnection sslCertsRef="SSL repertoire"/>
```

```
F BAQSTRT,REFRESH,KEYSTORE
F BAQSTRT,REFRESH,KEYSTORE,ID=CICSKeyStore
F BAQSTRT,REFRESH,KEYSTORE,ID=Db2KeyStore
F BAQSTRT,REFRESH,KEYSTORE,ID=OtherKeyStore
```

mitchj@us.ibm.com

* OpenAPI 3 only

Tech/Tip: Combining TLS mutual and basic authentication

```
//*************************************************************
//*  SET SYMBOLS
//*************************************************************
//EXPORT EXPORT SYMLIST=(*)
// SET CURL= '/usr/lpp/rocket/curl'
//*************************************************************
//*  CURL Procedure
//*************************************************************
//CURL PROC
//CURL EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
// PEND
//*************************************************************
//*  STEP CURL - use cURL to deploy API cscvinc
//*************************************************************
//DEPLOY EXEC CURL
BPXBATCH SH export CURL=&CURL; +
$CURL/bin/curl -X PUT -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc?status=sto+
pped > null; +
$CURL/bin/curl -X DELETE -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/zosConnect/apis/cscvinc > null; +
$CURL/bin/curl -X POST -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
--data-binary @/u/johnson/cscvinc.aar +
--header "Content-Type: application/zip" +
https://wg31.washington.ibm.com:9445/zosConnect/apis
//*************************************************************
//*  STEP CURL - use cURL to invoke the API cscvinc
//*************************************************************
//INVOKE EXEC CURL
//SYSTSIN  DD *,SYMBOLS=EXECSYS
BPXBATCH SH export CURL=&CURL; $CURL/bin/curl -X GET -s +
--cacert /u/johnson/CERTAUTH.PEM --user FRED:FRED +
https://wg31.washington.ibm.com:9445/cscvinc/employee/000100
```

```
<httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

<sslDefault sslRef="DefaultSSLSettings"
   outboundSSLRef="DefaultSSLSettings" />

<ssl id="DefaultSSLSettings"
    keyStoreRef="CellDefaultKeyStore"
    trustStoreRef="CellDefaultKeyStore"
    clientAuthenticationSupported="true"
    clientAuthentication="true"/>

<keyStore id="CellDefaultKeyStore"
    location="safkeyring:///Liberty.KeyRing"
    password="password" type="JCERACFKS"
    fileBased="false" readOnly="true" />
```

```
<httpEndpoint id="AdminHttpEndpoint"
    host="*"
    httpPort="-1"
    httpsPort="9445"
    sslOptionsRef="mySSLOptions"/>

<sslOptions id="mySSLOptions"
    sslRef="BatchSSLSettings"/>

<ssl id="BatchSSLSettings"
   keyStoreRef="CellDefaultKeyStore"
   trustStoreRef="CellDefaultKeyStore"
   clientAuthenticationSupported="true"
   clientAuthentication="false"/>
```

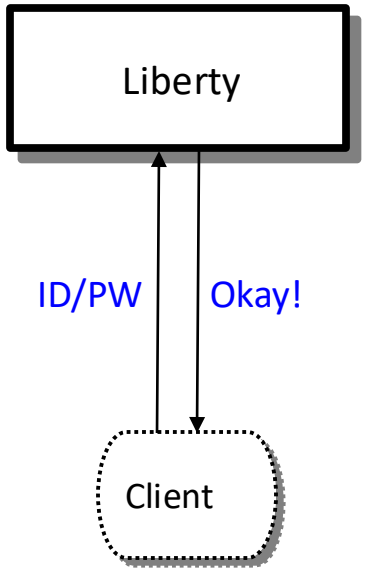
https://www.rocketsoftware.com/platforms/ibm-z/curl-for-zos

mitchj@us.ibm.com

Authentication - Third Party Authentication

Several different ways this can be accomplished:

Basic Authentication

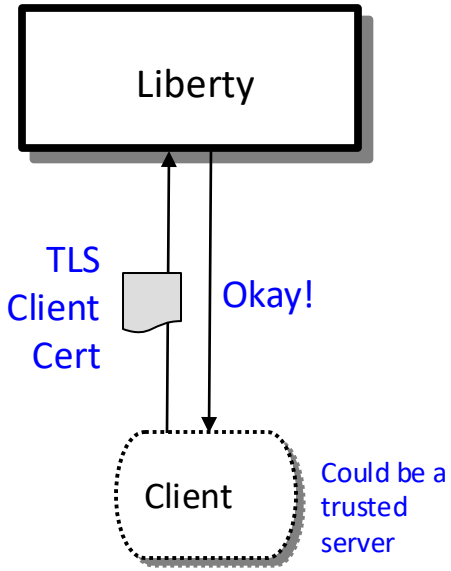


Server prompts for ID/PW

Client supplies ID/PW or ID/PassTicket

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

Client Certificate



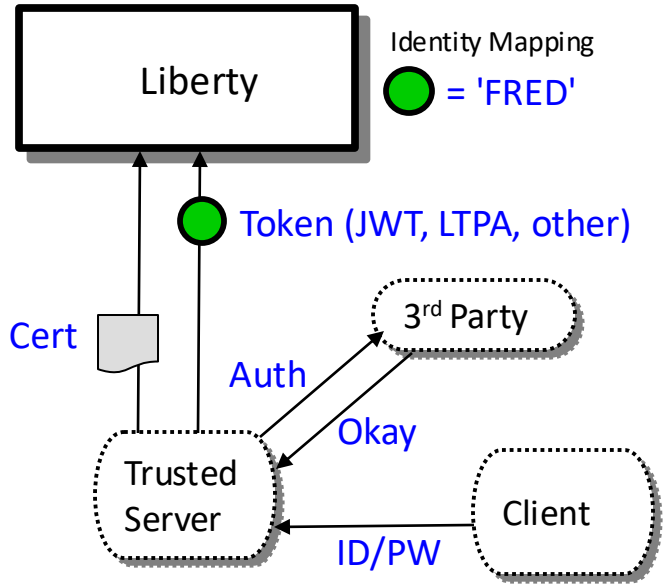
Server prompts for client certificate.

Client supplies certificate

Server validates client certificate and maps to an identity

Registry options:
- SAF

Third Party Authentication



Client authenticates to 3rd party sever

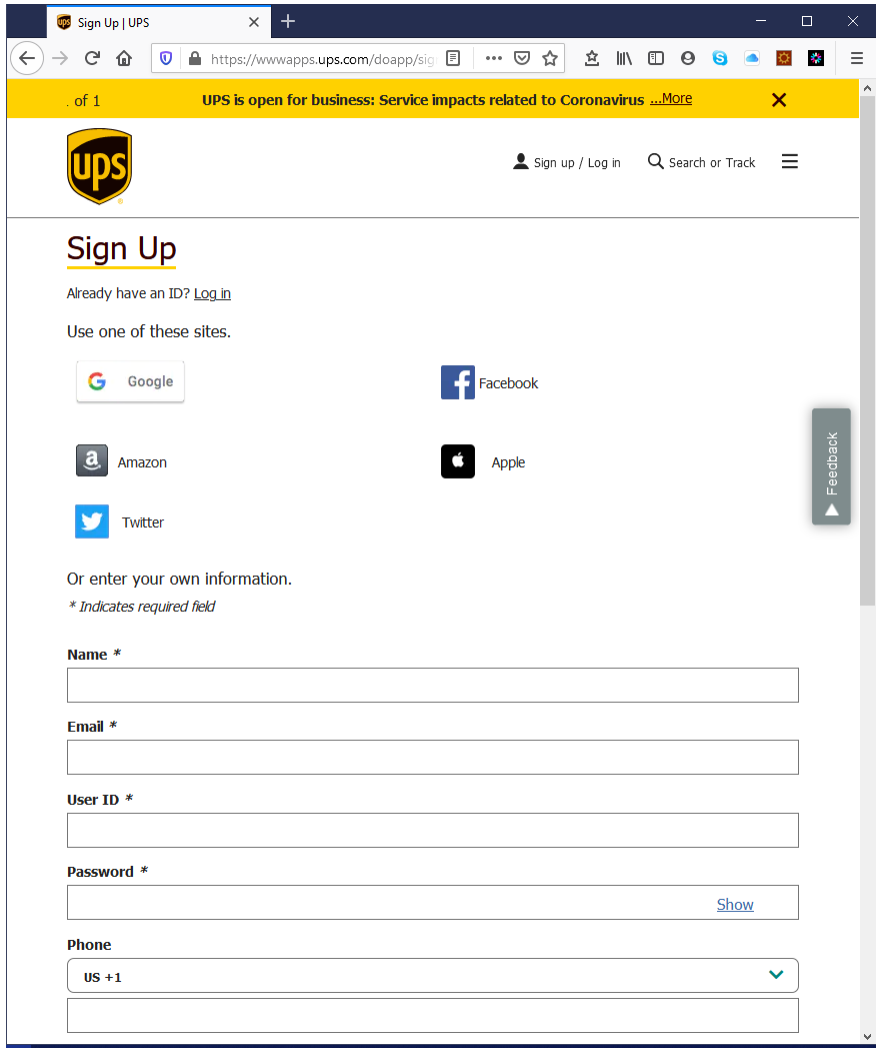
Client receives a trusted 3rd party toker

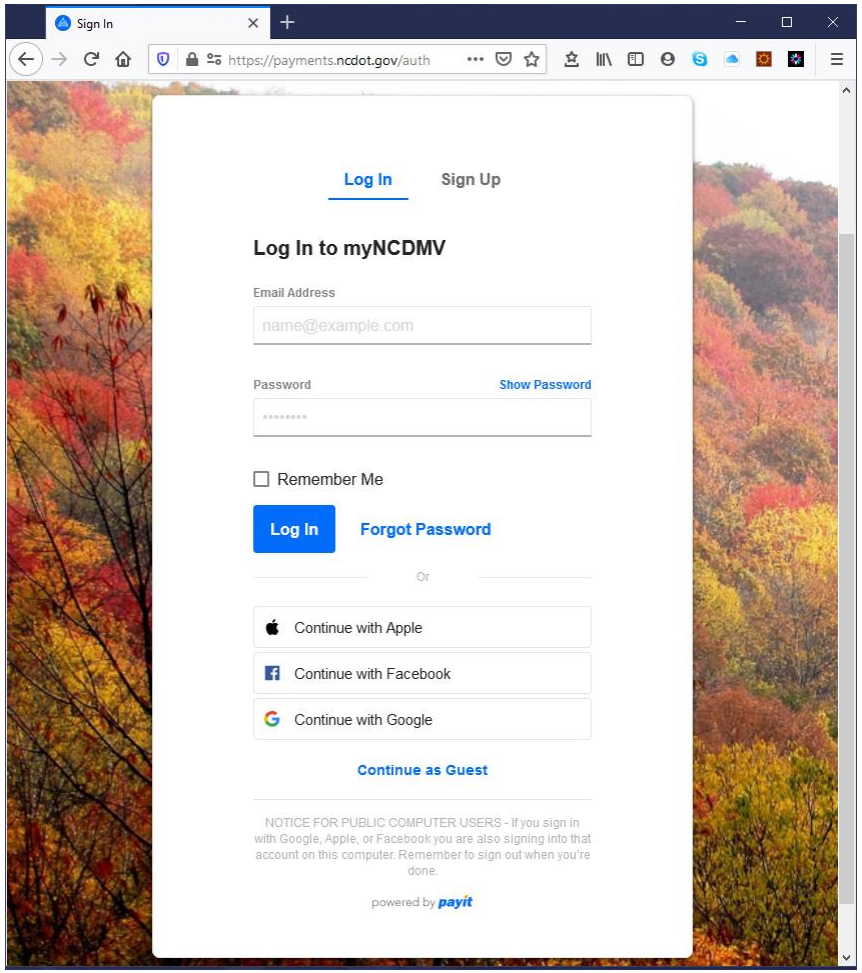
Token flows to Liberty z/OS and is mapped to an identity

Registry options:
- **We may know these detail.**

mitchj@us.ibm.com

Third Party Authentication Examples





mitchj@us.ibm.com

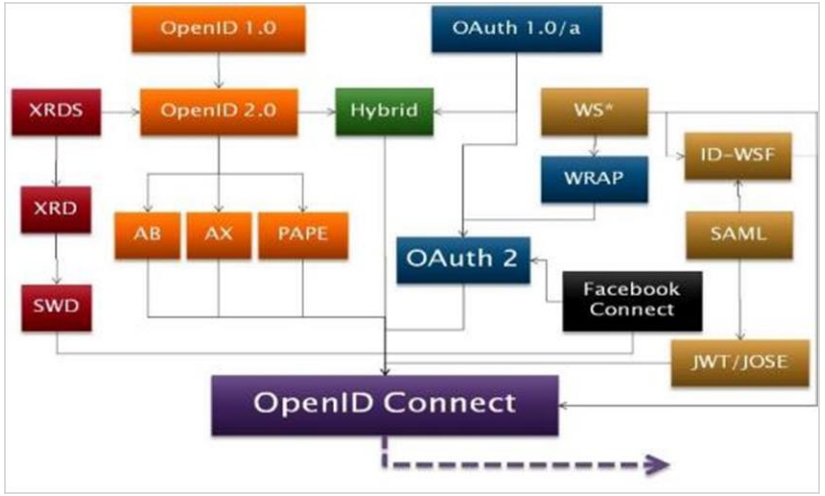
Open security standards

- **OAuth** is an open standard for access delegation, used as a way to grant websites or applications access to their information without requiring a password.
- **OpenID Connect** is an authentication layer on top of Oauth. It allows the verification of the identity of an end-user based on authentication performed by an authorization server.
- **JWT** (JSON Web token) defines a compact and self-contained way for securely transmitting information between parties as a JSON object

See the YouTube videos:

OAuth 2.0 and OpenID Connect (in plain English)
https://www.youtube.com/watch?v=996OiexHze0

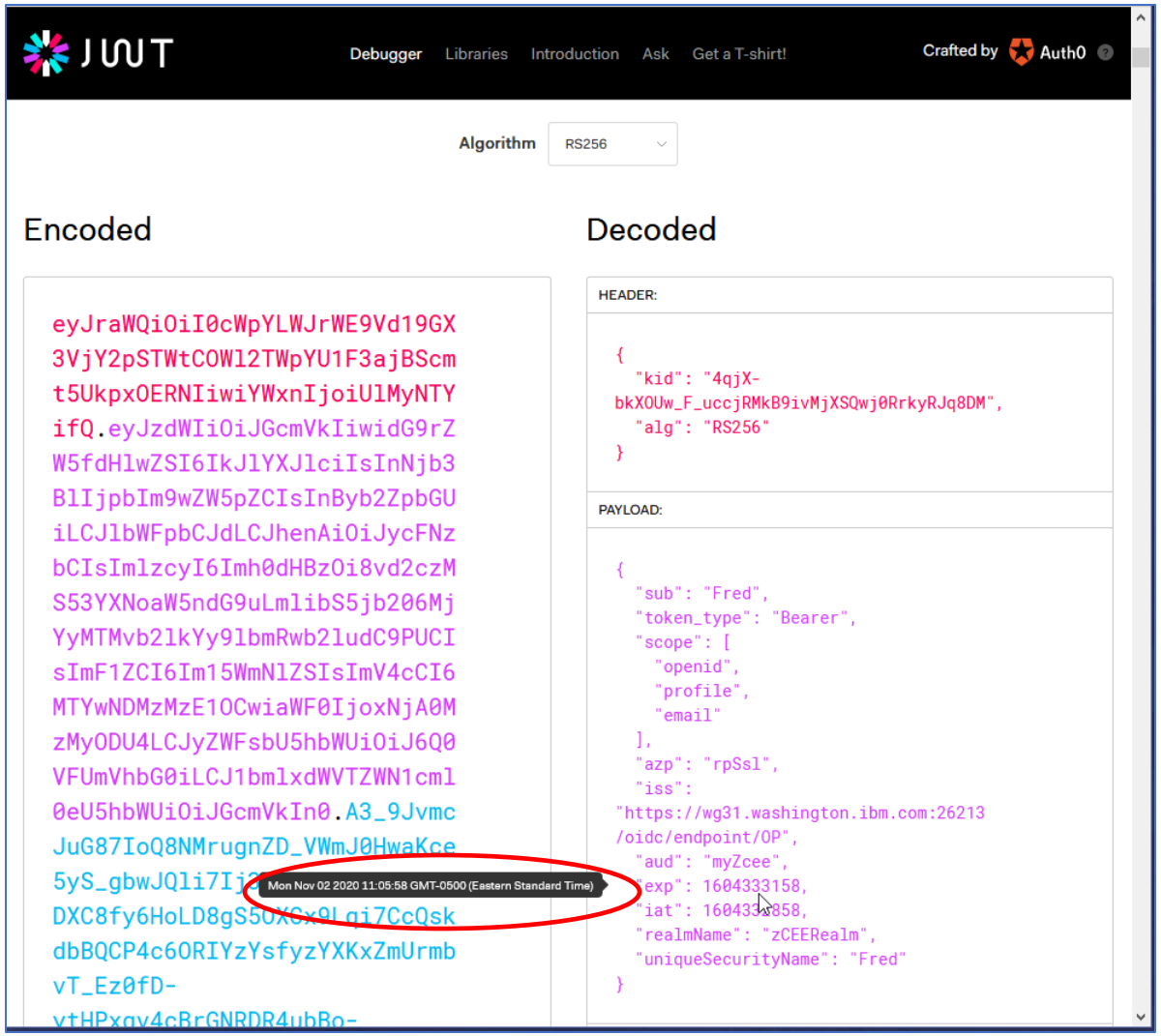
OpenID Connect on Liberty
https://www.youtube.com/watch?v=fuajCS5bG4c



mitchj@us.ibm.com

What is a 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



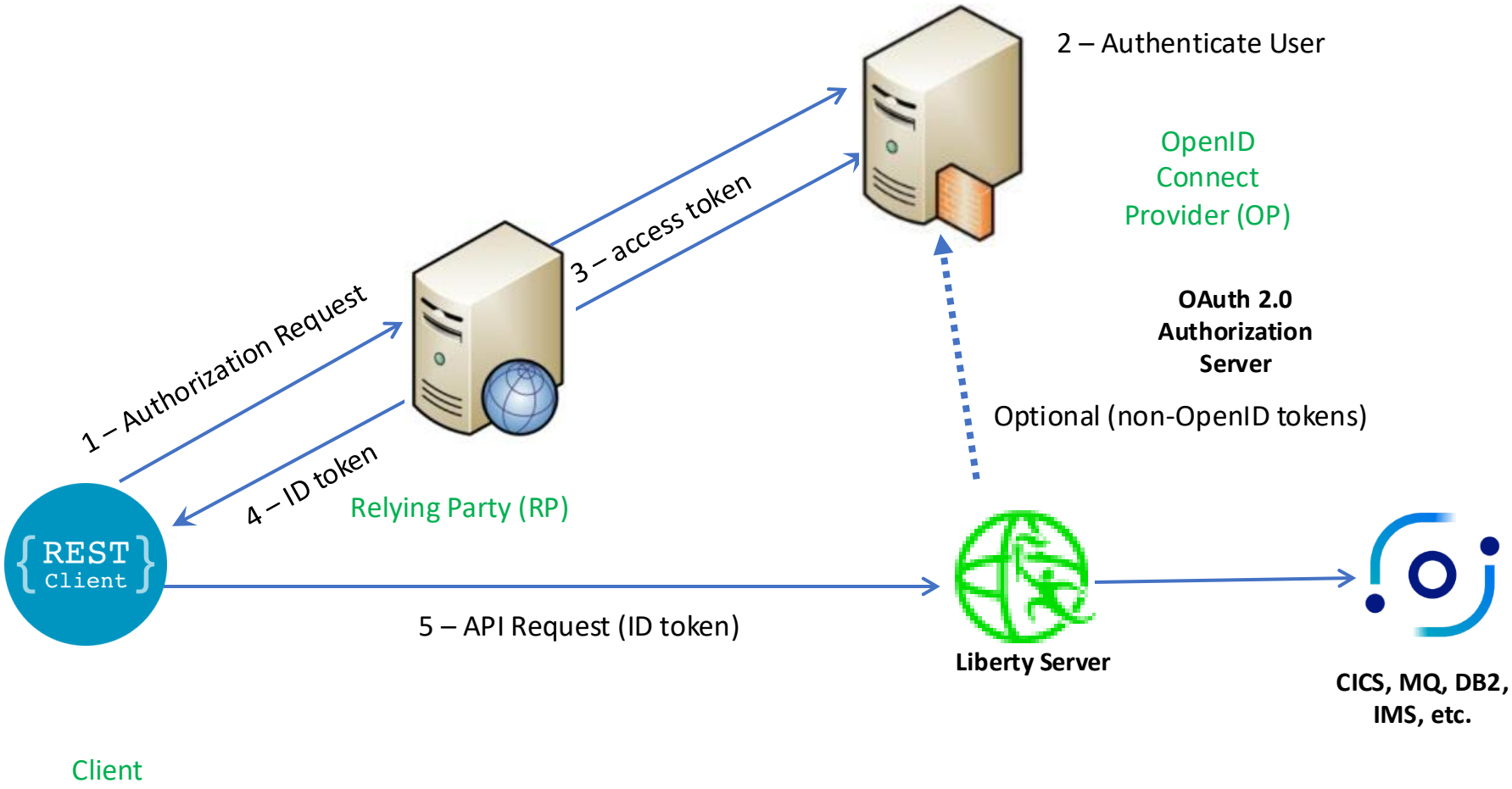
Values derived from the OAUTH configuration:

- signatureAlgorithm="**RS256**"
- accessTokenLifetime="**300**"
- resourceIds="**myZcee**"

https://jwt.io

mitchj@us.ibm.com

Typical Authorization Flow for an OpenID Connect token to a z/OS Connect API Provider



mitchj@us.ibm.com

Liberty OpenID Client identity mapping configuration attributes


```
<safRegistry id="saf" />
<safAuthorization racRouteLog="ASIS" />
<safCredentials unauthenticatedUser="WSGUEST"
       mapDistributedIdentities="true"
       profilePrefix="BBGZDFLT" />
```

Use distributed identity filters to map the distributed identities to SAF user IDs, using IDIDMAP resources and the RACMAP command.

```
<authFilter id="ATSAuthFilter">
   <requestUrl  id="ATSDemoUrl"
    name="ATSRefererUri"
    matchType="contains"
    urlPattern="/cscvinc/employee|/db2/employee|/mqapi/loan"/>
</authFilter>
<openidConnectClient id="ATS"
      httpsRequired="true"
      authFilterRef="ATSAuthFilter"
      inboundPropagation="required"
      scope="openid profile email"
      audiences="myZcee"
      issuerIdentifier=https://wg31.washington.ibm.com:26213/oidc/endpoint/OP
      mapIdentityToRegistryUser="false"
      signatureAlgorithm="RS256"
      userIdentityToCreateSubject="sub"
      trustAliasName="JWT-Signer-Certificate"
      trustStoreRef="jwtTrustStore"
      authnSessionDisabled="true"
      disableLtpaCookie="true">
  </openidConnectClient>
<keyStore fileBased="false" id="jwtTrustStore"
     location="safkeyring:///JWT.KeyRing"
     password="password" readOnly="true" type="JCERACFKS"/>
```

Specifies whether to map the identity to a registry user. If this is set to false, then the user registry (SAF) is not used to create the user subject.

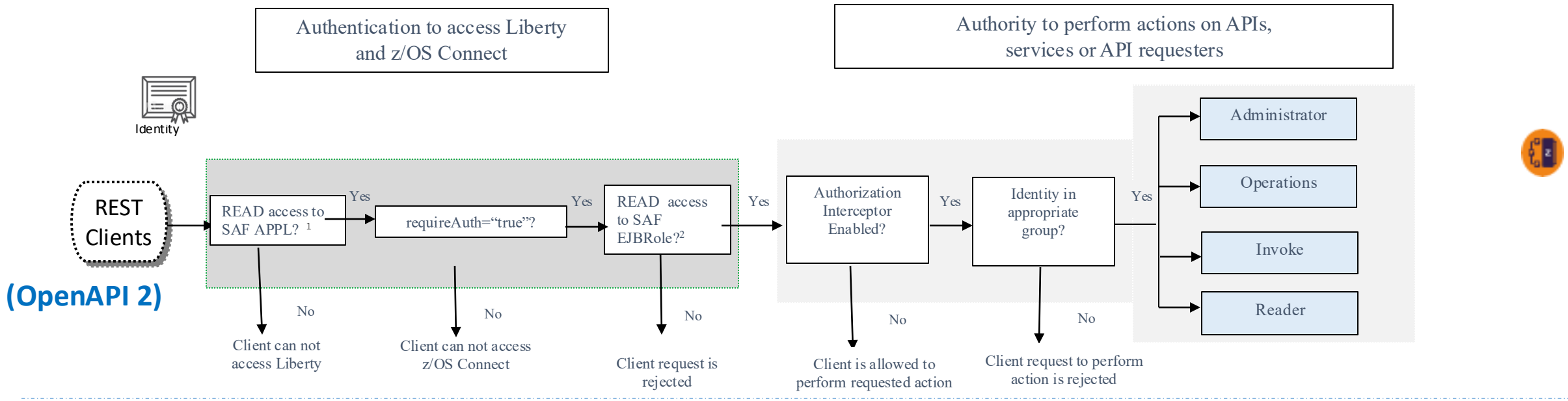
mitchj@us.ibm.com

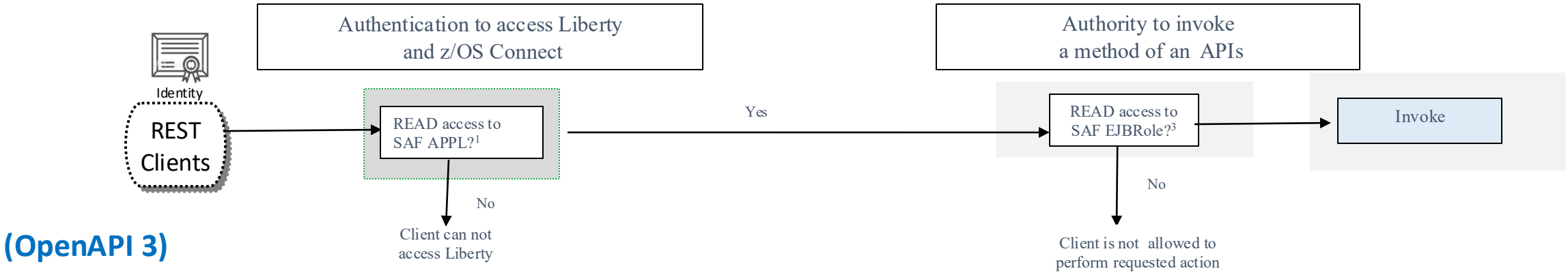
Authorization

Once we have an identity, then what?

mitchj@us.ibm.com

Security flow with z/OS Connect EE





[1]RDEFINE APPL *profilePrefix*
[2]**RDEFINE EJBROLE *profilePrefix*.zos.connect.access.roles.zosConnectAccess**
[3]**REDFINE EJBROLE *profilePrefix.resourceName.role***

mitchj@us.ibm.com

Slide 132

z/OS Connect OpenAPI 2 Administrative RESTful APIs

z/OS Connect administration API

Interface providing meta-data and life-cycle operations for z/OS Connect services, APIs and API requesters.

APIs : Operations for working with APIs

Show/Hide | List Operations | Expand Operations

GET	/apis	Returns a list of all the deployed z/OS Connect APIs
POST	/apis	Deploys a new API into z/OS Connect
DELETE	/apis/{apiName}	Undeploys an API from z/OS Connect
GET	/apis/{apiName}	Returns detailed information about a z/OS Connect API
PUT	/apis/{apiName}	Updates an existing z/OS Connect API

Services : Operations for working with services

Show/Hide | List Operations | Expand Operations

GET	/services	Returns a list of all the deployed z/OS Connect services
POST	/services	Deploys a new service into z/OS Connect
DELETE	/services/{serviceName}	Undeploys a service from z/OS Connect
GET	/services/{serviceName}	Returns detailed information about a z/OS Connect service
PUT	/services/{serviceName}	Updates an existing z/OS Connect service
GET	/services/{serviceName}/schema/{schemaType}	Returns the request or response schema for a z/OS Connect service

API Requesters : Operations that work with API Requesters.

Show/Hide | List Operations | Expand Operations

GET	/apiRequesters	Returns a list of all the deployed z/OS Connect API Requesters
POST	/apiRequesters	Deploys a new API Requester into z/OS Connect and invoke an API Requester call
DELETE	/apiRequesters/{apiRequesterName}	Undeploys an API Requester from z/OS Connect
GET	/apiRequesters/{apiRequesterName}	Returns the detailed information about a z/OS Connect API Requester
PUT	/apiRequesters/{apiRequesterName}	Updates an existing z/OS Connect API Requester

mitchj@us.ibm.com

z/OS Connect OpenAPI 2 Authorization Functions

Operations - Ability to perform all z/OS Connect operations and actions except for function *Invoke*. The following operations/actions are allowed:

APIs:

- *To obtain a list of all APIs (GET).**
- For a specific API, get its details and API Swagger document (GET) and *deploy (POST)**, update (PUT), start(PUT), stop(PUT), and delete(DELETE) it.

Services:

- *To obtain a list of all services or statistics for all services (GET).**
- For a specific service, get its details, request and response schemas, statistics (GET) and *deploy(POST)**, update(PUT), start(PUT), stop(PUT), and delete(DELETE) it.

API Requesters:

- *To obtain a list of all API requesters* (GET).*
- For a specific API requester, get its details (GET) and *deploy (POST)**, update(PUT), start(PUT), stop(PUT), and delete(DELETE) it.

*These APIs use either the POST or GET method to invoke the REST APIs whose URIs have no path parameter. Therefore, the name of the API, or service or API Requester is ignored. For authorization, only the default or global groups list can be used since no specific group list can be determined (for deployment, the name is embedded in the archive file).

mitchj@us.ibm.com

z/OS Connect OpenAPI 2 Authorization Functions

Reader - Ability for:

APIs:

- *To obtain a list of all APIs* (GET) .*
- For a specific API, get its details and API Swagger document (GET).

Services:

- *To obtain a list of all services* (GET). *
- For a specific service, get its details and request and response schemas (GET).

API Requesters:

- *To obtain a list of all API requesters* (GET).*
- For a specific API requester, get its details (GET) .

Invoke - Ability to invoke user APIs, services and/or API requesters (POST,PUT,GET,DELETE,+).

Admin - All z/OS Connect actions are allowed, including all corresponding *Operations*, *Invoke*, and *Reader* actions configured for the same z/OS Connect resource.

*These APIs use either the POST or GET method to invoke the REST APIs whose URIs have no path parameter. Therefore, the name of the API, service or API Requester is not available. For authorization, only the default or global groups list since no specific group list can be determined (for deployment, the name is embedded in the archive file.

mitchj@us.ibm.com

z/OS Connect OpenAPI 2 Administrative Groups

z/OS Connect uses group security for controlling authorization for accessing APIs. There are sets of default global groups for functional roles are configured in a zosConnectManager configuration element as shown below:

```
<zosconnect_zosConnectManager
          globalInterceptorsRef="interceptorList_g"
          globalAdminGroup="SYSPGRP " globalOperationsGroup="GBLOPERS"
          globalInvokeGroup="GBLINVKE" globalReaderGroup="GBLRDR"/>
```

There are four classes of groups available controlling z/OS Connect functions, administration, operations, invoking and reader in our server. An authenticated identity membership in one or more of these groups provides access to the corresponding function to that identity.
There is also a way to provide an alternative set of groups for functional roles for specific APIs, services, and API requesters in subordinate configuration elements in our server.

```
<zosConnectAPI name="cscvinc"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>

<service name="cscvincSelectService"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/>

<apiRequester name="cscvinc_1.0.0"
        adminGroup="CSCADMIN"  operationsGroup="CSCOPERS"
        invokeGroup="CSCINVKE" readerGroup="CSCREADR"/
```

mitchj@us.ibm.com

Interceptor - server XML example (OpenAPI 2)

```
<zosconnect_zosConnectManager
        globalInterceptorsRef="interceptorList_g"
        globalAdminGroup="SYSPGRP"
        globalOperationsGroup="GBLOPERS"
        globalInvokeGroup="GBLINVKE"
        globalReaderGroup="GBLRDR"/>

<zosconnect_authorizationInterceptor id="auth"/>
<zosconnect_auditInterceptor id="audit"/>
<zosconnect_zosConnectInterceptors id="interceptorList_g"
      interceptorRef="auth"/>
<zosconnect_zosConnectInterceptors id="interceptorList_a"
      interceptorRef="auth,audit"/>

<zosconnect_zosConnectAPIs>
   <zosConnectAPI name="catalog"
      runGlobalInterceptorsRef="true"
      adminGroup="aapigrp1,aapigrp2"
      operationsGroup="oapigrp1,oapigrp2"
      invokeGroup="iapigrp1,oapigrp2"
      readerGroup="rapigrp1,rapigrp2"/>
</zosconnect_zosConnectAPIs>

<zosconnect_apiRequesters>
   <apiRequester name="cscvincapi_1.0.0"
      runGlobalInterceptorsRef="false"
      interceptorsRef="interceptorList_a"
      adminGroup="aaprgrp1,aaprgrp2"
      operationsGroup="oaprgrp1,oaprgrp2"
      invokeGroup="iaprgrp1,oaprgrp2"
      readerGroup="raprgrp1,raprgrp2"/>
</zosconnect_apiRequesters>

<zosconnect_services>
  <service id="selectByEmployee" name="selectEmployee"
      runGlobalInterceptorsRef="false"
      interceptorsRef="interceptorList_a"
      adminGroup="asrvgrp1,asrvgrp2"
      operationsGroup="osrvgrp1,osrvgrp2"
      invokeGroup="isrvgrp1,isrvgrp2"
      readerGroup="rsrvrgrp1,rsrvgrp2"/>
</zosconnect_services>
```

Global interceptor list – authorization interceptor only

Alternative interceptor list – authorization and audit interceptors

This avoids duplication of interceptors

Note that these are z/OS Connect configuration elements. Documented in the z/OS Connect KC

mitchj@us.ibm.com

Tech/Tip: Server XML example – combining TLS/AUTH interceptor (OpenAPI 2)

```
 <zosconnect_zosConnectManager
           requireAuth="true"
           requireSecure="true"
           globalInterceptorsRef="interceptorList_g"
           globalAdminGroup="SYSPGRP"
           globalOperationsGroup="GBLOPERS"
           globalInvokeGroup="GBLINVKE"
           globalReaderGroup="GBLRDR"/>

<zosconnect_authorizationInterceptor id="auth"/>
<zosconnect_zosConnectInterceptors id="interceptorList_g"
         interceptorRef="auth"/>

  <zosconnect_apiRequesters>
     <apiRequester name="cscvincapi_1.0.0"
         requireSecure="false"
         invokeGroup="iaprgrp1"/>
  </zosconnect_apiRequesters>
```

Global TLS security and authentication are enabled.

TLS security is disabled for this API requester archive artifact. Avoiding the HTTP 302 REDIRECT error.

This configuration would allow a MVS batch job to authenticate to z/OS Connect and use HTTP for the protocol (when an AT-TLS outbound policy is not available). Only authorization identities which are members of groups identified as administrators or invokers would be authorized to invoke this API requester.

F BAQSTRT,ZCON,CLEARSAFCACHE

mitchj@us.ibm.com

Tech-Tip: Solution for z/OS Connect Authorization Levels (OpenAPI 2)

```
<zosconnect_zosConnectManager
        globalInterceptorsRef="interceptorList_g"
        globalAdminGroup="SYSPGRP" globalOperationsGroup="GBLOPERS,CSCOPERS,DB2OPERS"
        globalInvokeGroup="GBLINVKE" globalReaderGroup="GBLRDR"/>

<zosconnect_zosConnectAPIs>
    <zosConnectAPI name="cscvinc" operationsGroup="CSCOPERS" invokeGroup="CSCINV"/>
    <zosConnectAPI name="db2employee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE"/>
</zosconnect_zosConnectAPIs>

<zosconnect_services>
    <service name="cscvincSelectService" operationsGroup="CSCOPERS" invokeGroup="CSCINV"/>
    <service name="selectEmployee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE"/>
</zosconnect_services>

<zosconnect_apiRequesters>
    <apiRequester name="cscvincSelectService" operationsGroup="CSCOPERS" invokeGroup="CSCINV"/>
    <apiRequester name="selectEmployee" operationsGroup="DB2OPERS" invokeGroup="DB2INVKE"/>
</zosconnect_apiRequesters>
```

- Now members of groups SYSPGRP, GBLOPERS,CSCOPERS, DB2OPERS and GBLRDR can connect to a z/OS server from the API toolkit.
- Members of groups SYSPGRP, GBLOPERS,CSCOPERS, and DB2OPERS can deploy new artifacts.
- Only members of group CSCOPERS and DB2OPERS can manage artifacts after they are deployed.

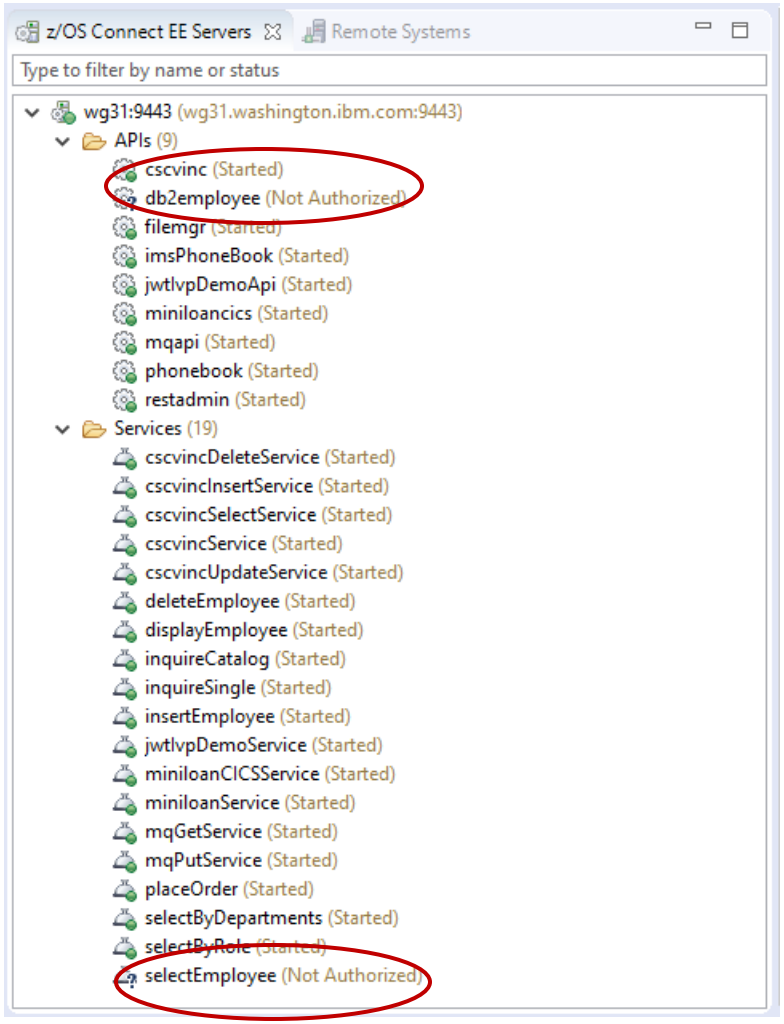
mitchj@us.ibm.com

When a partial list of subordinate groups are provided, the corresponding default global groups for the absence groups are used.

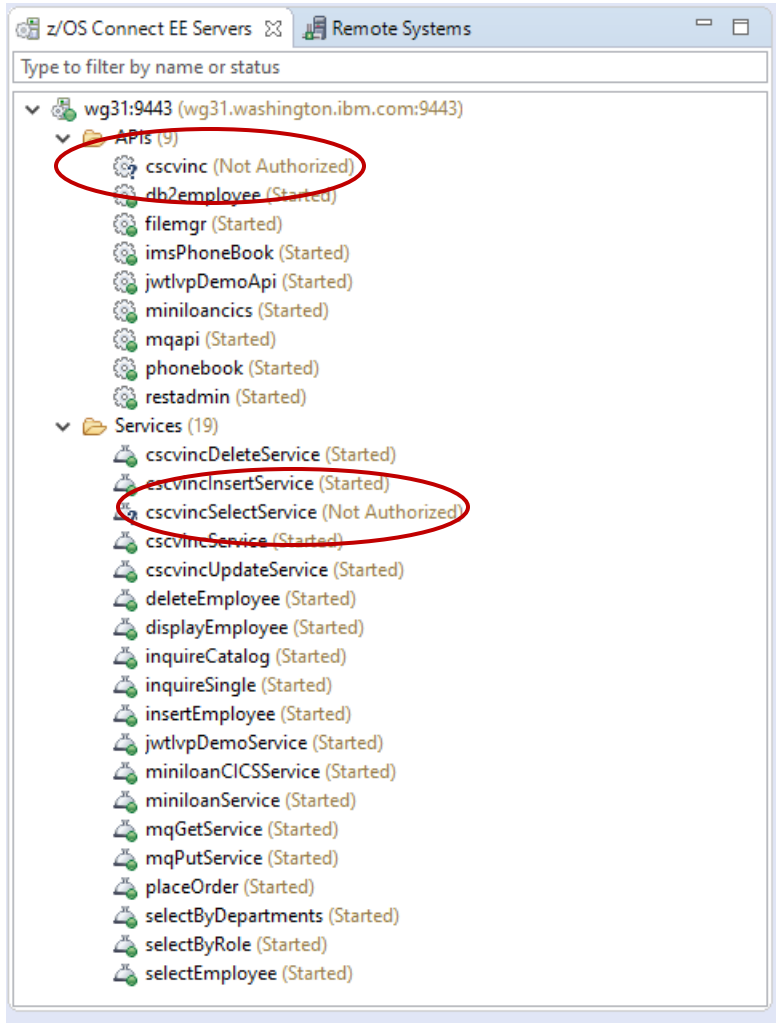
Tech-Tip: z/OS Toolkit and authorization status (OpenAPI 2)

Members of CSCOPERS and DB2OPERS can now connect to a server from the API Toolkit

CSCOPERS



DB2OPERS



mitchj@us.ibm.com

EJB roles for z/OS Connect (OpenAPI 3)

```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />

<webApplication id="CatalogManager" location="${server.config.dir}/apps/api.war" name="CatalogManager"/>

<safRoleMapper profilePattern=%profilePrefix%.%resourceName%.%role%
```

```
openapi: 3.0.0
. . .
servers:
- url: /
x-ibm-zcon-roles-allowed:
- Manager
. . .
paths:
  /items:
    get:
     operationId: itemsGet
     . . .
  /items/{id}:
    get:
     . . .
     operationId: itemsIdGet
     x-ibm-zcon-roles-allowed:
        - Staff
 /orders:
     post:
     . . .
     operationId: ordersPost
     x-ibm-zcon-roles-allowed:
       - Staff
```

*The value for %role% would be either **Manager** or **Staff**.*

So, the required SAF EJB roles to be defined would be:

- *BBGZDFLT.CatalogManager.Manager*
- *BBGZDFLT.CatalogManager.Staff*

Access to use the GET method to invoke /items would require read access to EJB role *BBGZDFLT.CatalogManager.Manager.* Access to use the GET method to invoke /items/{id} and the POST method to invoke /orders would require read access to EJB role *BBGZDFLT.CatalogManager.Staff.*

mitchj@us.ibm.com

Flowing identities to back-end z/OS systems

Basic authentication - Identity and Password

Server XML Configuration elements where basic authentication can be provided.

```
<connectionFactory id="imsTM"> containerAuthDataRef=“IMScredentials">
<authData id=“IMScredentials" user= "identity" password= "password"/>

<connectionFactory id="imsDB">
<properties.imsudbJLocal  databaseName="DFSIVPA"   user="identity" password="password"/>
</connectionFactory>

<zosconnect_cicsIpicConnection id="CICS"  authDataRef=“CICScredentials"/>
<zosconnect_authData id=“CICScredentials" user= "identity" password= "password"/>

<zosconnect_zosConnectServiceRestClientConnection id="Db2" basicAuthRef=“db2Auth"/>
<zosconnect_zosConnectServiceRestClientBasicAuth id="db2Auth"
    userName="identity"  password="password"/>

<jmsQueueConnectionFactory jndiName=“MQ“>
          <properties.wasJms userName=“identity" password=“password"   />
</jmsQueueConnectionFactory>
```

The value of the password can be encoded in the server XML configuration file. Using the ***securityUtility*** shipped with WebSphere Liberty Profile.

mitchj@us.ibm.com

Using securityUtility to encrypt passwords

Best practice : use encryption for passwords instead of base64 encoding

- **securityUtility** – located in <wlp_install_dir>/wlp/bin Usage: securityUtility {encode|createSSLCertificate|help} [options]
  - For encryption, use encode --key=encryption_key
    - Specifies the key to be used when encoding using AES encryption. This string is hashed to produce an encryption key that is used to encrypt and decrypt the password. The key can be provided to the server by defining the variable **wlp.password.encryption.key** whose value is the key. If this option is not provided, a default key is used.

    ./securityUtility encode --encoding=aes --key=myKey myPassWord

    {aes}AHO0aXdiVD96u4oMRhoKeYH3U7aDqtFXTuHFBsO98Wlb
  - Support was added at Liberty 22.0.0.1 for storing an AES password encryption key in a SAF key ring, see URL https://www.ibm.com/docs/en/was-liberty/zos?topic=slia-storing-aes-password-encryption-key-in-saf-key-ring

    ./securityUtility encode --encoding=aes --keyring=safkeyring://JOHNSON/Liberty.KeyRing --keyringType=JCERACFKS

    --keyLabel="Johnson Client Cert" myPassWord
  - Also supports 1-way hash encoding – for passwords in server.xml with basicRegistry
    - For hash, use encode –encoding=hash

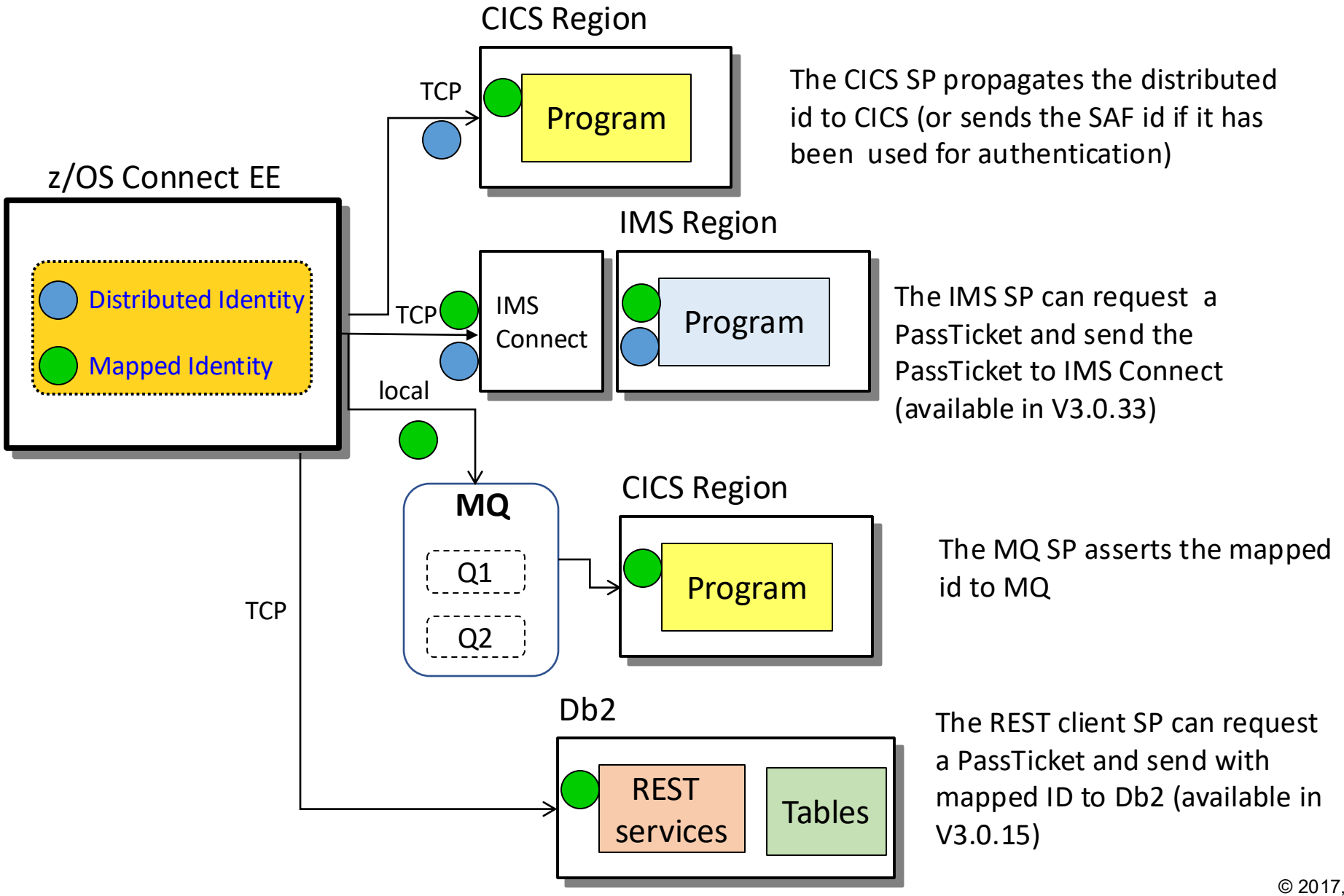
    ./securityUtility encode –-encoding=hash XXXXXXXX

    {hash}ATAAAAAIcqTmHn5qZahAAAAAIMjzy+hP8YFaIO6LiCreVe4etRLUS9a25eVuYtx6WKiv

See the WebSphere Application Server for z/OS Liberty *securityUtility* command at URL:
https://www.ibm.com/docs/en/was-liberty/zos?topic=applications-securityutility-command

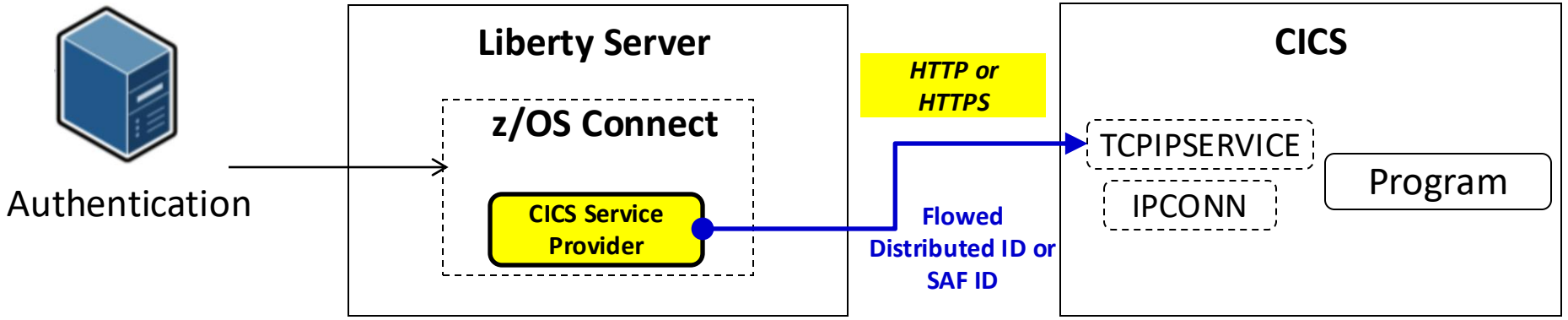
mitchj@us.ibm.com

Flowing an identity to a back-end subsystem



mitchj@us.ibm.com

Flowing a user ID with CICS service provider

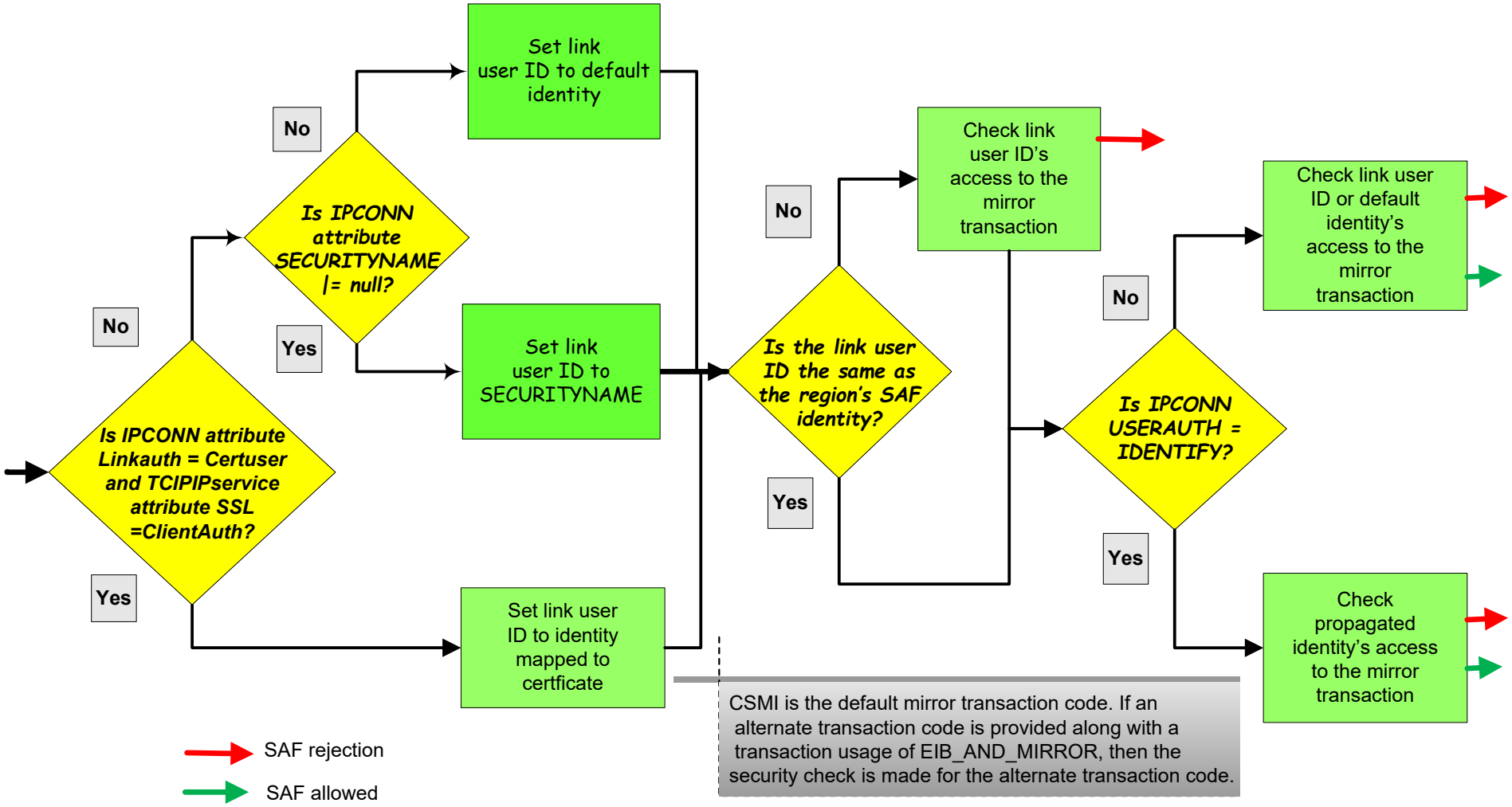


Distributed identities can be propagated to CICS and then mapped to a RACF user ID by CICS. You can then view the distinguished name and realm for a distributed identity in the association data of the CICS task. **Important**: If the z/OS Connect server is not in the same Sysplex as the CICS system, you must use an IPIC TLS (JSSE) connection that is configured with client authentication.

If a SAF ID is used for authentication (e.g., basic authentication with a SAF registry) then the SAF ID is passed to CICS.

mitchj@us.ibm.com

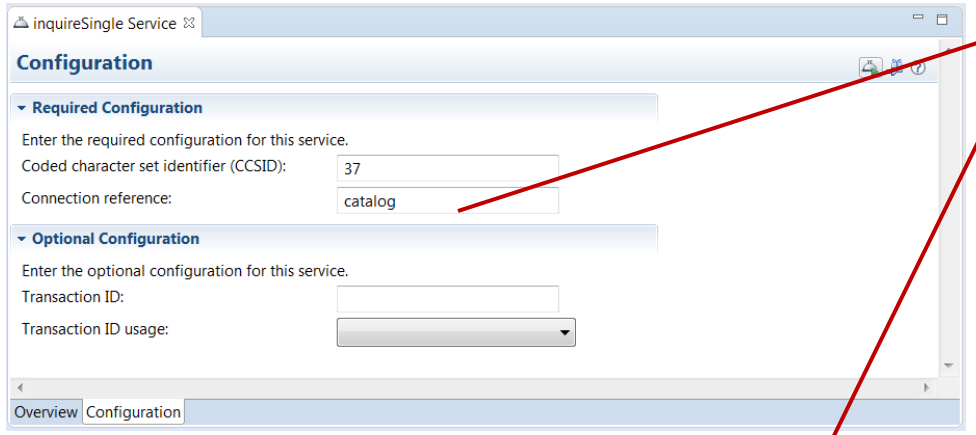
Tech/Tip: CICS IPIC Security with USERAUTH(IDENTIFY)

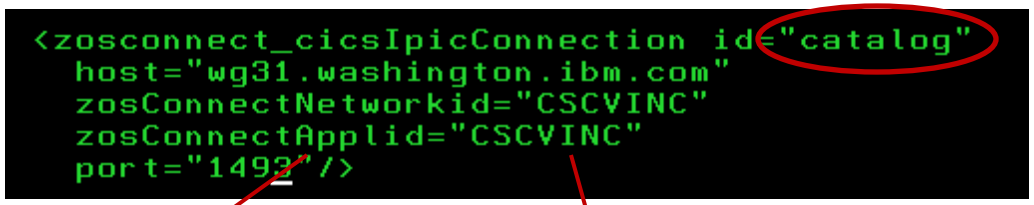


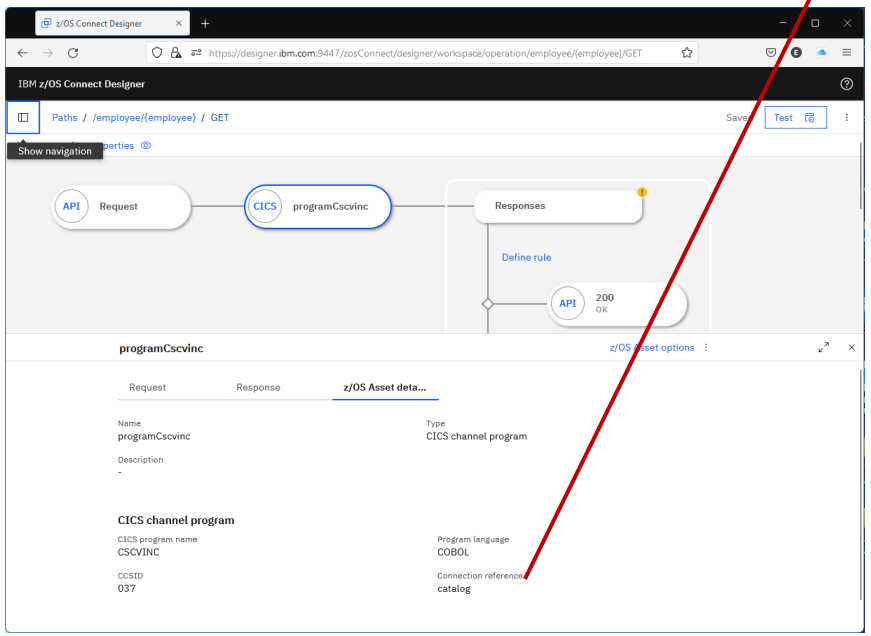
mitchj@us.ibm.com

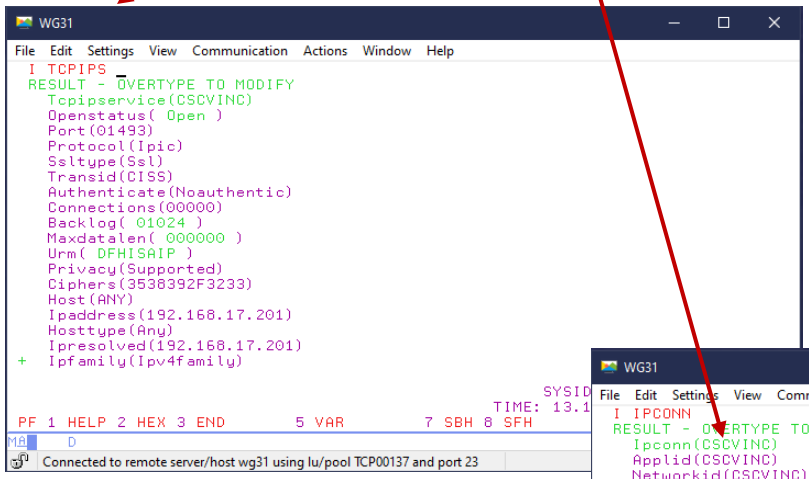
Flowing an identity to CICS

The zosconnect_cicsIpicConnection element is the key :



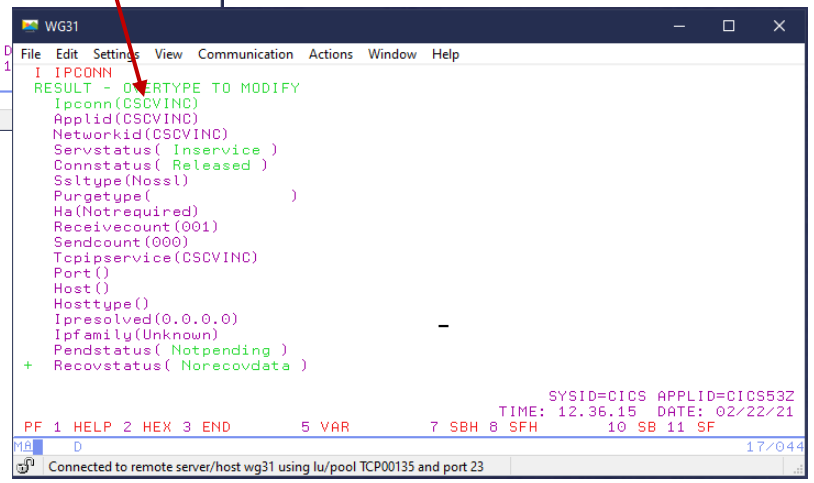






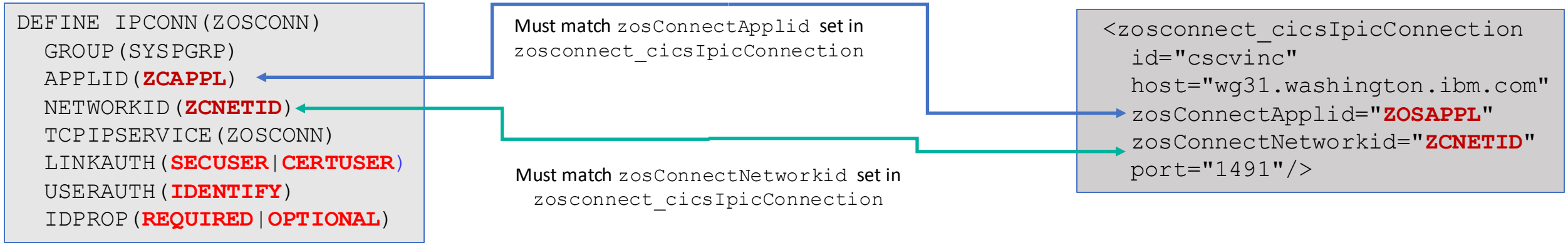
CICS TCPIPService

CICS IPCONN



mitchj@us.ibm.com

CICS IPCONN Resource



LINKAUTH Determines the user identity to be used for link security. The value is either ***CERTUSER*** or ***SECUSER***. A value of ***CERTUSER*** sets the link identity to the identity associated with the client certificate received from the client endpoint (TLS mutual authentication is required). A value of ***SECUSER*** sets the link identity to the value of the *SECURITYNAME* attribute as defined in the IPCONN resource.

USERAUTH Identifies how the identity under which the attached transaction attach security will run. Since a password is not available, a value of ***VERIFY*** is not possible. A value of ***LOCAL*** means the current link identity is used. A value of ***DEFAULTUSER*** means the CICS default identity is used. For identity propagation purposes, the value of **USERAUTH** should be ***IDENTIFY*** (no password will be required) so the identity provided by the client is used for executing the attached transaction. TLS must be used if the client is in a different Sysplex.

IDPROP Determines whether the original distributed identity authenticated by the z/OS Connect server is also propagated to CICS in a ddition to the mapped identity used for z/OS Connect authorization checks. A value of ***NOTALLOWED*** does not propagate the original distributed identity. A value of ***OPTIONAL*** will propagate to CICS the original distributed identity, if available. A value of ***REQUIRED*** requires that the original distributed identity be propagated to CICS. TLS must be used if the client is in a different Sysplex.

CERTIFICATE Provides the label of the certificate connected to the CICS key ring to be used for server endpoint certificate during a TLS handshake.

mitchj@us.ibm.com

Identity Propagation and CICS High Availability

Assume the service installed in a server files use the following *Connection reference* values:

- cscvinc
- catalog
- miniloan

If identity propagation is required for all connection, then the CICS IPCONN resources defined in the CICs that correspond to a zosconnect_cicsIpicConnection configuration elements must be dedicated to that z/OS Connect server and connection reference can not be reused.

Simplify administration by still sharing a common cicsIpicConnection XML configuration element by using variables and a bootstrap properties file or "variables" XML file

Server baqsvr1's bootstrap.properties

```
ipicPort=1491
cicsHost=dvipa.washington.ibm.com
serverPrefix=baqsvr1
```

Server baqsvr2's bootstrap.properties

```
cicsHost=dvipa.washington.ibm.com
ipicPort=1491
serverPrefix=baqsvr2
```

Server baqsvr3's bootstrap.properties

```
cicsHost=dvipa.washington.ibm.com
ipicPort=1491
serverPrefix=baqsvr3
```

ipicIDProp.xml

```
<zosconnect_cicsIpicConnection id="cscvinc"
  host="${cicsHost}"
  zosConnectNetworkid="${wlp.server.name}"        → baqsvr1 or baqsvr2
  zosConnectApplid="${wlp.server.name}"
  sharedPort="true" port="${ipicPort}"/>
<zosconnect_cicsIpicConnection id="catalog"
  host="${cicsHost}"
  zosConnectNetworkid="${serverPrefix}C"          → baqsvr1C or baqsvr2C
  zosConnectApplid="${serverPrefix}C"
  sharedPort="true" port="${ipicPort}"/>
<zosconnect_cicsIpicConnection id="miniloan"
  host="${cicsHost}"
  zosConnectNetworkid="${serverPrefix}M"          → baqsvr1M or baqsvr2M
  zosConnectApplid="${serverPrefix}M"
  sharedPort="true" port="${ipicPort}"/>
```

mitchj@us.ibm.com

CICS IPCONN and TCPIPService resources for HA

CICS Specific TCPIPService - IPIC

```
TCpipservice    : IPIC[1]
GROup           : SYSPGRP
Urm          ==> DFHISAIP
POrtnumber   ==> 01492
STatus       ==> Open
PROtocol     ==> IPic
TRansaction  ==> CISS
Host         ==> ANY
Ipaddress    ==> ANY
SPeciftcps   ==>
```

CICS Generic TCPIPService - IPICG

```
TCpipservice    : IPICG[1]
GROup           : SYSPGRP
Urm          ==> DFHISAIP
POrtnumber   ==> 01491
STatus       ==> Open
PROtocol     ==> IPic
TRansaction  ==> CISS
Host         ==> ANY
Ipaddress    ==> ANY
SPeciftcps   ==> IPIC
```

A client connects first to the CICS region's generic port (1491) and then the CICS region redirects the client to the region's specific port (1492).

```
I IPCONN ACQ
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Ipc(BAQSVR1 ) App(BAQSVR1 ) Net(BAQSVR1 ) Ins Acq Nos
       Rece(001) Sen(000) Tcp(IPIC    )
 Ipc(BAQSVR1C) App(BAQSVR1C) Net(BAQSVR1C) Ins Acq Nos
       Rece(001) Sen(000) Tcp(IPIC    )
 Ipc(BAQSVR1M) App(BAQSVR1M) Net(BAQSVR1M) Ins Acq Nos
       Rece(001) Sen(000) Tcp(IPIC    )
 Ipc(BAQSVR2 ) App(BAQSVR2 ) Net(BAQSVR2 ) Ins Acq Nos
       Rece(001) Sen(000) Tcp(IPIC    )
 Ipc(BAQSVR2C) App(BAQSVR2C) Net(BAQSVR2C) Ins Acq Nos
       Rece(001) Sen(000) Tcp(IPIC    )
 Ipc(BAQSVR2M) App(BAQSVR2M) Net(BAQSVR2M) Ins Acq Nos
       Rece(001) Sen(000) Tcp(IPIC    )
```

Number of IPCONN resources equals the number of zCEE server times the number of unique connection references

[1]CICS requires the specific TCPIPService be installed before the corresponding generic TCPIPService resource. TCPIPServices are installed in alphabetically order, so the name of specific service must be alphabetically prior to the name of the generic TCPIPService.

mitchj@us.ibm.com

CICS IPIC connection processing for high availability load balancing*

If the *reconnectInterval* attribute is set, at the specified time interval, a check is made to see if a new connection attempt should be attempted

A new connection is established if the current connection properties are not the preferred connection properties:

- If *reconnectInterval, preferredSpecificHost* and *preferredSpecificPort* are not set,
  - New connection attempts are disabled (this is the default behavior).
- If *reconnectInterval* is set and *preferredSpecificHost* and *preferredSpecificPort* are not set,
  - A new connection is attempted at the interval specified by the *reconnectInterval* time. Use this to enable regular connection rebalancing.
- If *reconnectInterval* and *preferredSpecificPort* are set and *preferrredSpecificHost* is not set,
  - A new connection is attempted at the expiration time interval and if the current connected port in use does not match the preferred port
  - Relevant when shared port is for a single LPAR
  - Specific CICS region is preferred
- If *reconnectInterval* and *preferredSpecificHost* are set and *preferredSpecificPort* is not set
  - A new connection is attempted at the expiration time interval and if the current host in use does not match the preferred port
  - Relevant when shared port is across Sysplex
  - Any CICS region on a specific LPAR is preferred
- If *reconnectInterval*, *preferredSpecificHost* and *preferredSpecificPort* are all set
  - A new connection is attempted at the expiration time interval time and if both the current host and port in use do not match the preferred host and port
  - Relevant when shared port is on a single LPAR or across a Sysplex
  - Specific CICS region is preferred.

When the reconnection attempt results in a new connection to a CICS region, new requests are sent over the new connection. Previous connections will continue and when all requests have completed processing, the previous or old connection will be closed.

mitchj@us.ibm.com

*Available at V3.0.56 and later

Tech/Tip: A PassTicket provides an alternative to a password

- A PassTicket is generated by or for a client by using a secured sign-on key (whose value is masked or encrypted) to encrypt a valid *RACF identity* combined with the *application name* of the targeted resource. Also embedded in the PassTicket is a time stamp (based on the current Universal Coordinated Time (UCT)) which sets the time when the PassTicket will expire (usually 10 minutes).

- Access to PassTickets is managed using the RACF PTKTDATA class.

- For z/OS Connect, a RACF PassTicket can be used for basic authentication when connecting from any REST client on any platform to a z/OS Liberty server and for requests from a z/OS Connect server accessing IMS and Db2.

- ***PassTickets do not have to be generated on z/OS using RACF services.*** IBM has published the algorithm used to generate a PassTickets, see manual *z/OS Security Server RACF Macros and Interfaces, SA23-2288-40. Github has examples using Java, Python and other example are available on other sites*.

mitchj@us.ibm.com

PassTickets and IMS

- Basic authentication to IMS Connect using a PassTicket depends on the APPL parameters configured in IMS Connect.

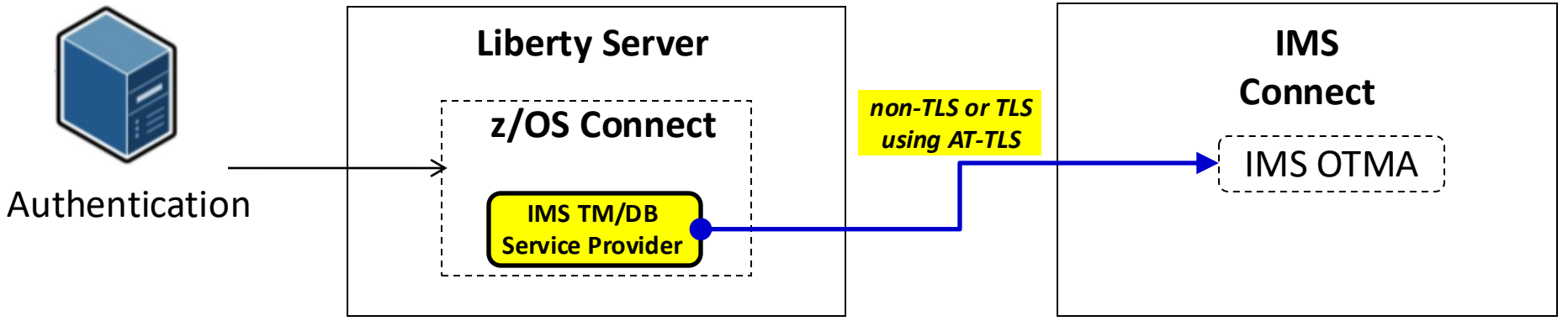
```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,DRU=HWSYDRU0,
TMEMBER=OTMAMEM,APPL=IMSTMAPL)
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1),
     DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSDBAPL)
```

RDEFINE PTKTDATA IMSTMAPL SSIGNON(0123456789ABCDEF)) APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.IMSTMAPL.* UACC(NONE)
PERMIT IRRPTAUTH.IMSTMAPL.* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)

RDEFINE PTKTDATA IMSDBAPL SSIGNON(0123456789ABCDEF)) APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.IMSDBAPL.* UACC(NONE)
PERMIT IRRPTAUTH.IMSDBAPL .* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)

mitchj@us.ibm.com

Flowing an identity to IMS Connect (TM)




```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
DATASTORE=(GROUP=OTMAGRP,ID=IVP1,MEMBER=HWSMEM,DRU=HWSYDRU0,
TMEMBER=OTMAMEM,APPL=IMSTMAPL)
```

Authentication options:

1. **User ID / password**
2. **PassTicket support**

```
<connectionFactory containerAuthDataRef="Connection1_Auth" id="IVP1">
<properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000"/>
</connectionFactory>
<authData id="Connection1_Auth"  user="USER1"password="{xor}GhIPExAGDwg="/>
```

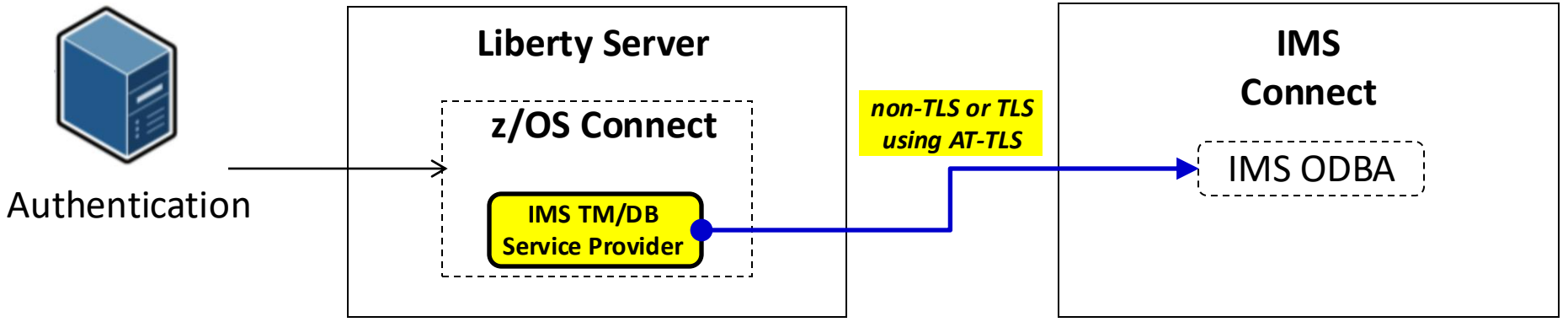
Specify a user identity and password to be used in the request to IMS Connect

```
<connectionFactory containerAuthDataRef="Connection1_Auth" id="IVP1">
<properties.gmoa hostName="wg31.washington.ibm.com" portNumber="4000
applicationName="IMSTMAPL"/>
</connectionFactory>
```

Request a PassTicket
And use it in the request to IMS Connect

mitchj@us.ibm.com

Flowing an identity to IMS Connect (DB)




```
HWS=(ID=IMS15HWS,XIBAREA=100,RACF=Y,RRS=Y)
TCPIP=(HOSTNAME=TCPIP,PORTID=(4000,LOCAL),RACFID=JOHNSON,TIMEOUT=5000)
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMS15HWS,TMEMBER=PLEX1),
DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000,APPL=IMSDBAPL)
```

Authentication options:

1. **User ID / password**
2. **PassTicket support**

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

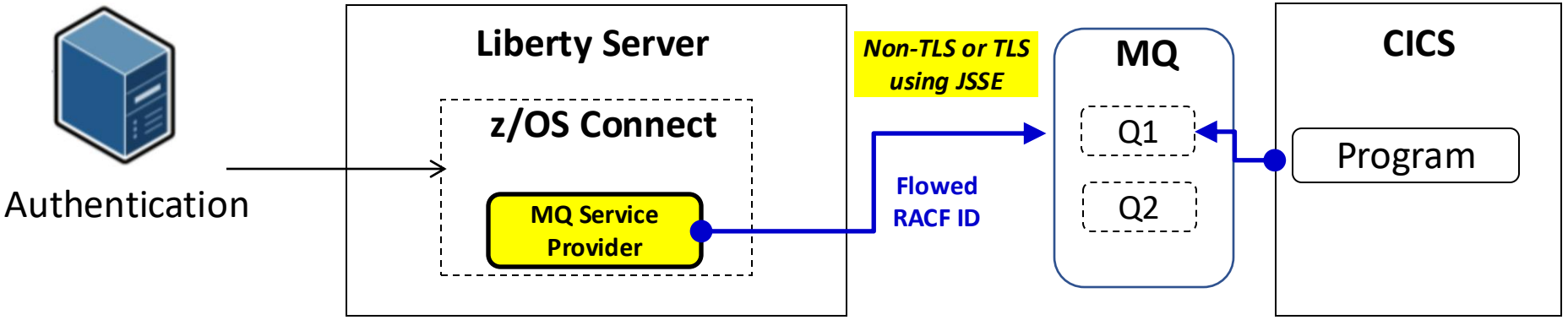
Specify a user identity and password to be used in the request to IMS Connect

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

Request a PassTicket
And use it in the request to IMS Connect

mitchj@us.ibm.com

Flowing a user ID with MQ service provider



Set **`useCallerPrincipal=true`** to flow the authenticated RACF user ID

```
<zosconnect_services>
    <service name="mqPut">
        <property name="destination" value="jms/default"/>
        <property name="useCallerPrincipal" value="true"/>
    </service>
</zosconnect_services>
```

Define identity propagation to MQ

mitchj@us.ibm.com

PassTickets and Db2

- Basic authentication Db2 using a PassTicket depends on the Db2 configuration.

```
DSNL080I  -DSN2 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DSN2LOC            USIBMWZ.DSN2APPL  USIBMWZ.DSN0APPL
DSNL084I TCPPORT=2446  SECPORT=2445  RESPORT=2447  IPNAME=-NONE
DSNL085I IPADDR=::192.168.17.201
DSNL086I SQL    DOMAIN=WG31.WASHINGTON.IBM.COM
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL106I SESSIDLE = 001440
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

```
DSNL080I  -DSNC DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
DSNL081I STATUS=STARTD
DSNL082I LOCATION           LUNAME            GENERICLU
DSNL083I DSN2LOC            -NONE             -NONE
DSNL084I TCPPORT=2446  SECPORT=2445  RESPORT=2447  IPNAME=DB2IPNM
DSNL085I IPADDR=::192.168.17.252
DSNL086I SQL    DOMAIN=WG31.WASHINGTON.IBM.COM
DSNL086I RESYNC DOMAIN=WG31.WASHINGTON.IBM.COM
DSNL089I MEMBER IPADDR=::192.168.17.252
DSNL105I CURRENT DDF OPTIONS ARE:
DSNL106I PKGREL = COMMIT
DSNL106I SESSIDLE = 001440
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

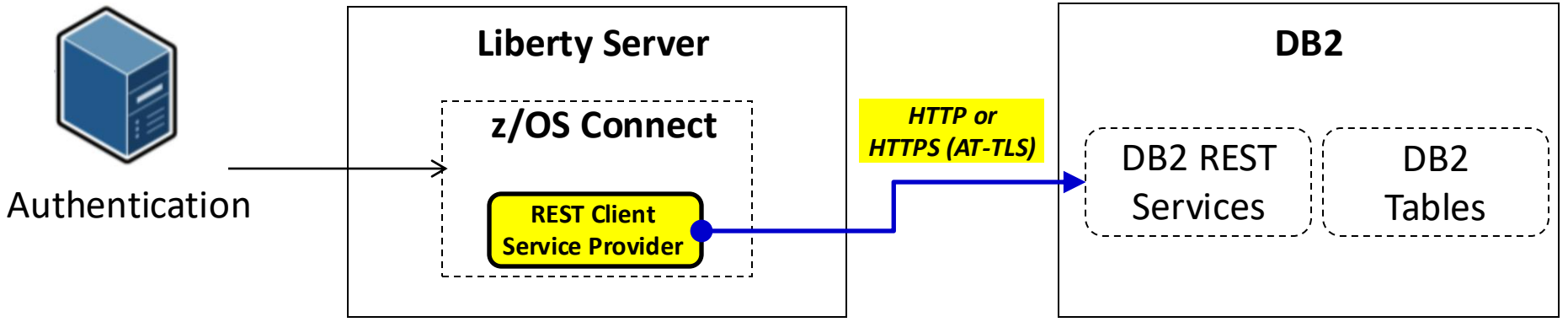
Which value should be used for applName is determine for use in RACF resources is determine as shown below.

- If GENERICLU is defined, use the second part of GENERICLU for *applName*, e.g., ***DSN0APPL***
- If GENERICLU is not defined, use the second part of LUNAME for *applName*, e.g., ***DSN2APPL***
- If neither GENERICLU or LUNAME is defined, use the value of the IPNAME for *applName,* e.g., ***DB2IPNM***

```
RDEFINE PTKTDATA DSN2APPL SSIGNON(0123456789ABCDEF)) APPLDATA('NO REPLAY PROTECTION') UACC(NONE)
RDEFINE PTKTDATA IRRPTAUTH.DSN2APPL.* UACC(NONE)
PERMIT IRRPTAUTH.DSN2APPL.* ID(LIBSERV) CLASS(PTKTDATA) ACCESS(UPDATE)
```

mitchj@us.ibm.com

Flowing the identity for the REST client SP (Db2)




```
<zosconnect_zosConnectServiceRestClientConnection id="Db2Conn"
    host="wg31.washington.ibm.com"
    port="2446"
    basicAuthRef="dsn2Auth" />
<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth"
    userName="USER1"
    password="USER1"/>
```

Authentication options:

1. **User ID / password**
2. **TLS Client Certificate (JSSE)**
3. **PassTicket support**

Specify a user identity and password to be used in the HTTP header with the Db2 REST Service

```
<zosconnect_zosConnectServiceRestClientConnection id="Db2Conn"
    host="wg31.washington.ibm.com"
    port="2446"
    basicAuthRef="dsn2Auth" />

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

z/OS Connect requests a PassTicket from RACF

mitchj@us.ibm.com

Tech/Tip: Db2 REST Security

- Access to Db2 REST services requires READ access to the Db2 subsystem DSNR REST resource. i.e., permit READ access to this resource to the identity in question, for example

  PERMIT DSN2.REST CLASS(DSNR) ID(USER2) ACC(READ) where DSN2 is the Db2 subsystem ID
  SETROPTS RACLIST(DSNR) REFRESH

- Db2 package access is also required. If a user is not able to display a valid Db2 REST services in the z/OS Connect Db2 services development tooling or by using a **POST** to the Db2 provided REST interface URL of http://wg31.washington.ibm.com:2446/services/DB2ServiceDiscover, then they may not have sufficient access to the package containing the service.

  For example, if service *zCEEService.selectEmployee* is defined to Db2 but not visible in the z/OS Connect tooling or if a **GET** request to URL http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee fails with message:

```
{
  "StatusCode": 500,
  "StatusDescription": "Service zCEEService.selectEmployee discovery failed due to
  SQLCODE=-551 SQLSTATE=42501, USER2 DOES NOT HAVE THE PRIVILEGE TO PERFORM OPERATION EXECUTE
  PACKAGE ON OBJECT zCEEService.selectEmployee. Error Location:DSNLJACC:35"
}
```

The user needs to be granted execute authority on package *zCEEService.selectEmployee* with command:

GRANT EXECUTE ON PACKAGE "zCEEService"."selectEmployee" TO USER2 or
GRANT EXECUTE ON PACKAGE "zCEEService"."*" TO USER2

mitchj@us.ibm.com

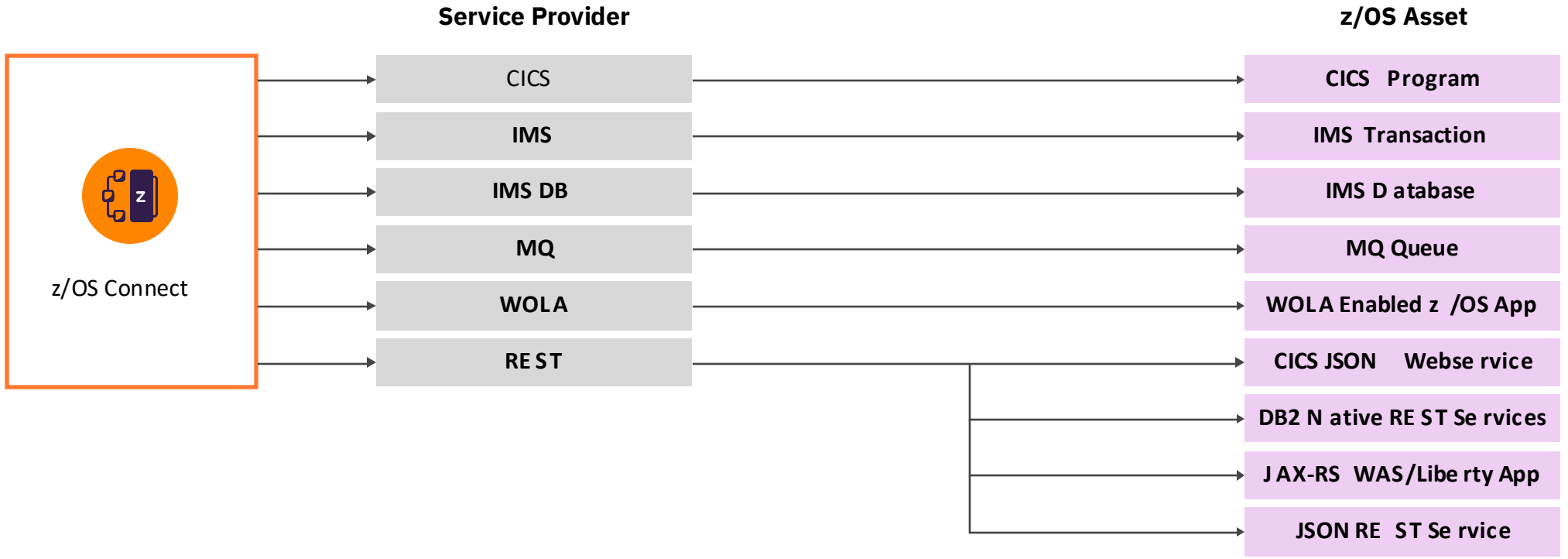
/miscellaneousTopics

performance, high availability, Liberty

mitchj@us.ibm.com

What resources can z/OS Connect map to?

And which service provider could I use?

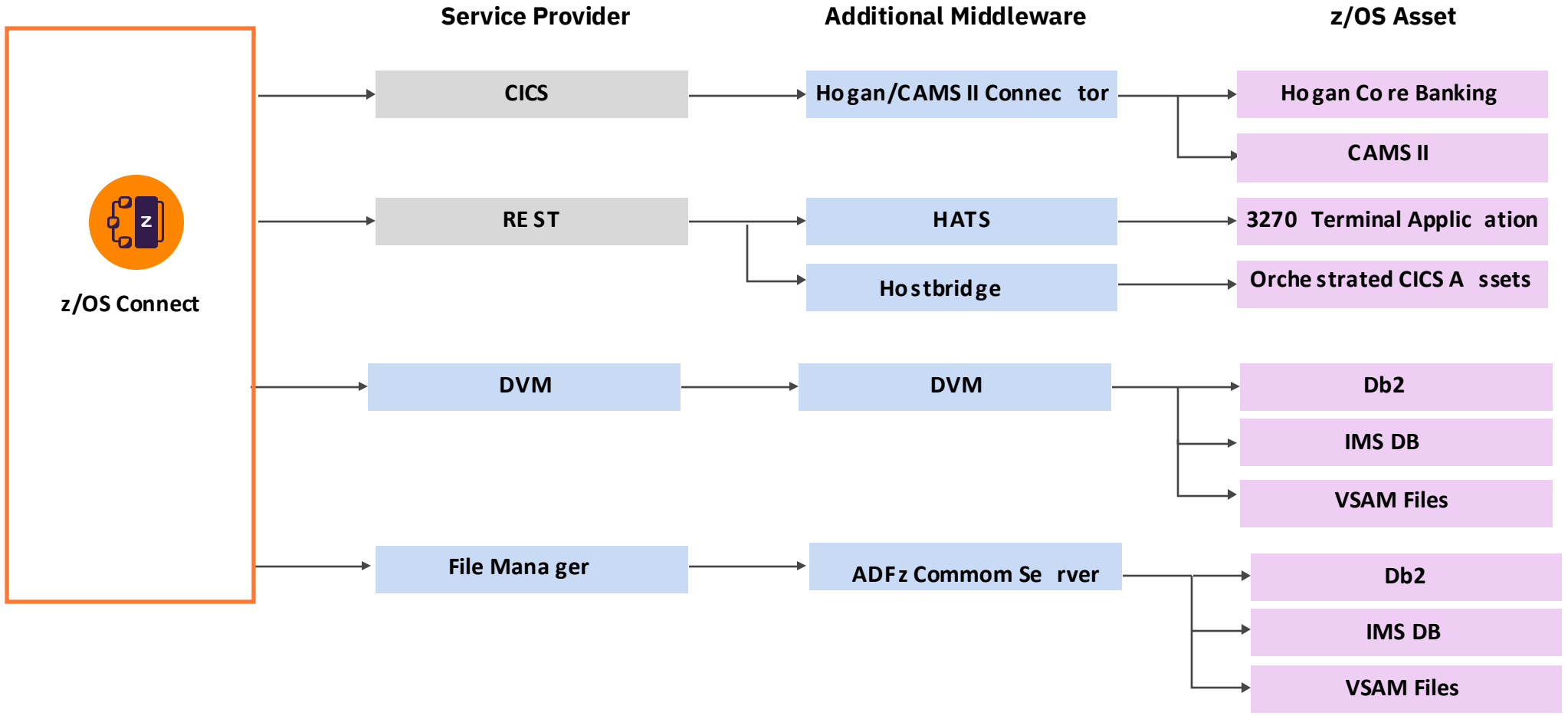


The core **service providers** included with z/OS Connect provide API access to a wide range of z/OS resources.

mitchj@us.ibm.com

Additional Middleware

Additional value from the ecosystem



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

mitchj@us.ibm.com

API Policies

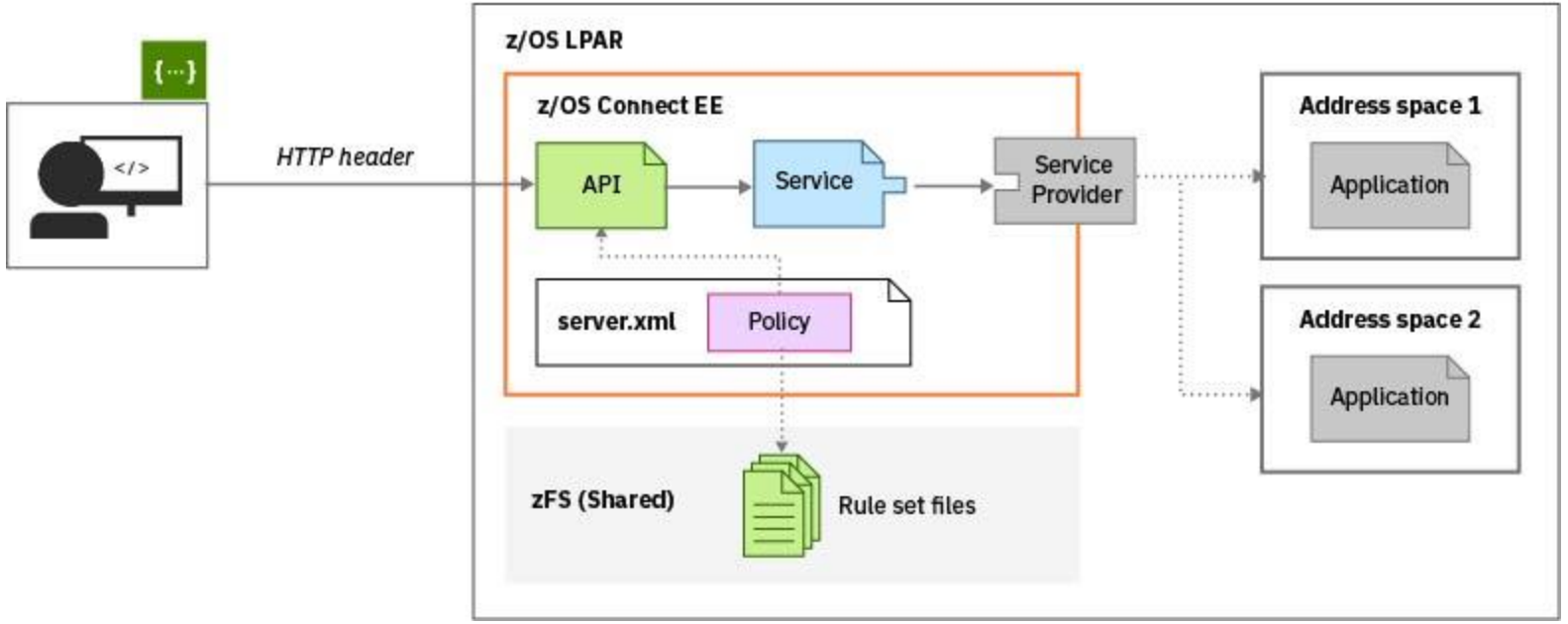
- HTTP header properties can be used to select alternative for IMS (V3.0.4) , CICS (V3.0.10), Db2 (V3.0.36) or MQ (V3.0.39)
- Policies can be configured globally for every API in the server or for individual APIs (V3.0.11)

CICS attributes
- cicsCcsid
- cicsConnectionRef
- cicsTransId

IMS attributes
- imsConnectionRef
- imsInteractionRef
- imsInteractionTimeout
- imsLtermOverrideName
- imsTranCode
- imsTranExpiration

Db2 attributes
- db2ConnectionRef
- db2CollectionID

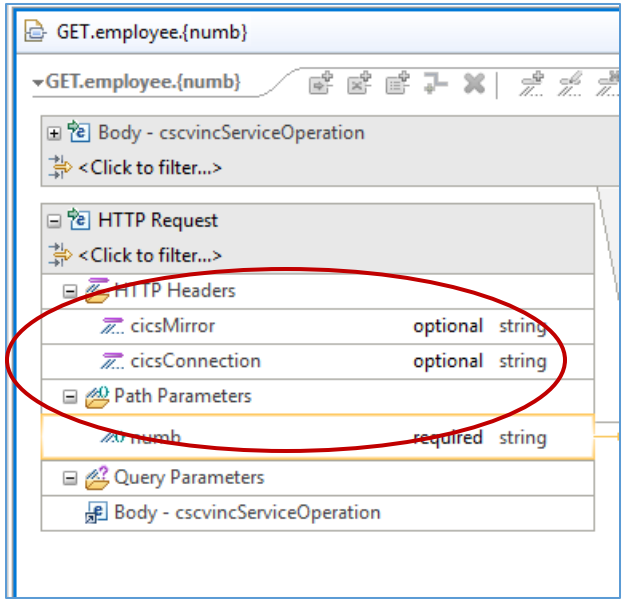
MQ attributes
- mqConnectionFactory
- mqDestination
- mqReplyDestination



mitchj@us.ibm.com

A sample API Policies for CICS

```
<ruleset name="CICS rules">
     <rule name="csmi-rule">
         <conditions>
             <header name="cicsMirror" match="ANY_VALUE"/> *
         </conditions>
         <actions>
             <set property="cicsTransId" value="${cicsMirror}"/>
         </actions>
     </rule>
     <rule name="connection-rule">
         <conditions>
             <header name="cicsConnection" value=" match="ANY_VALUE"/>
         </conditions>
         <actions>
             <set property="cicsConnectionRef" value="${cicsConnection}"
         </actions>
     </rule>
 </ruleset>
```



Curl

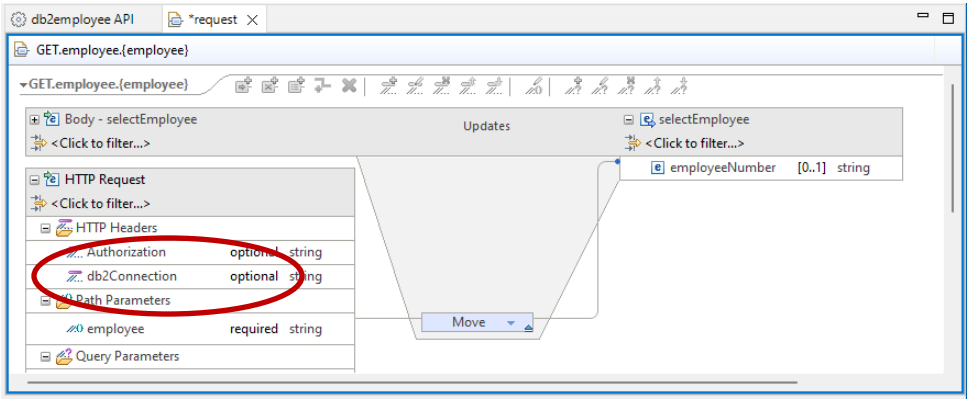
```
curl -X GET --header 'Accept: application/json' --header 'cicsMirror: MIJO' --header 'cicsConnection: cscvinc' 'https://m
```

*Transaction MIJO needs to be a clone of CSMI (e.g., invoke program DFHMIRS)

mitchj@us.ibm.com

A sample API Policies for Db2

```
<ruleset name="Db2 rules">
     <rule name="connection-rule">
       <conditions>
           <header name="db2Connection" value="" match="ANY_VALUE"/>
       </conditions>
       <actions>
           <set property="db2ConnectionRef" value="${db2Connection}"/>
       </actions>
     </rule>
 </ruleset>
```



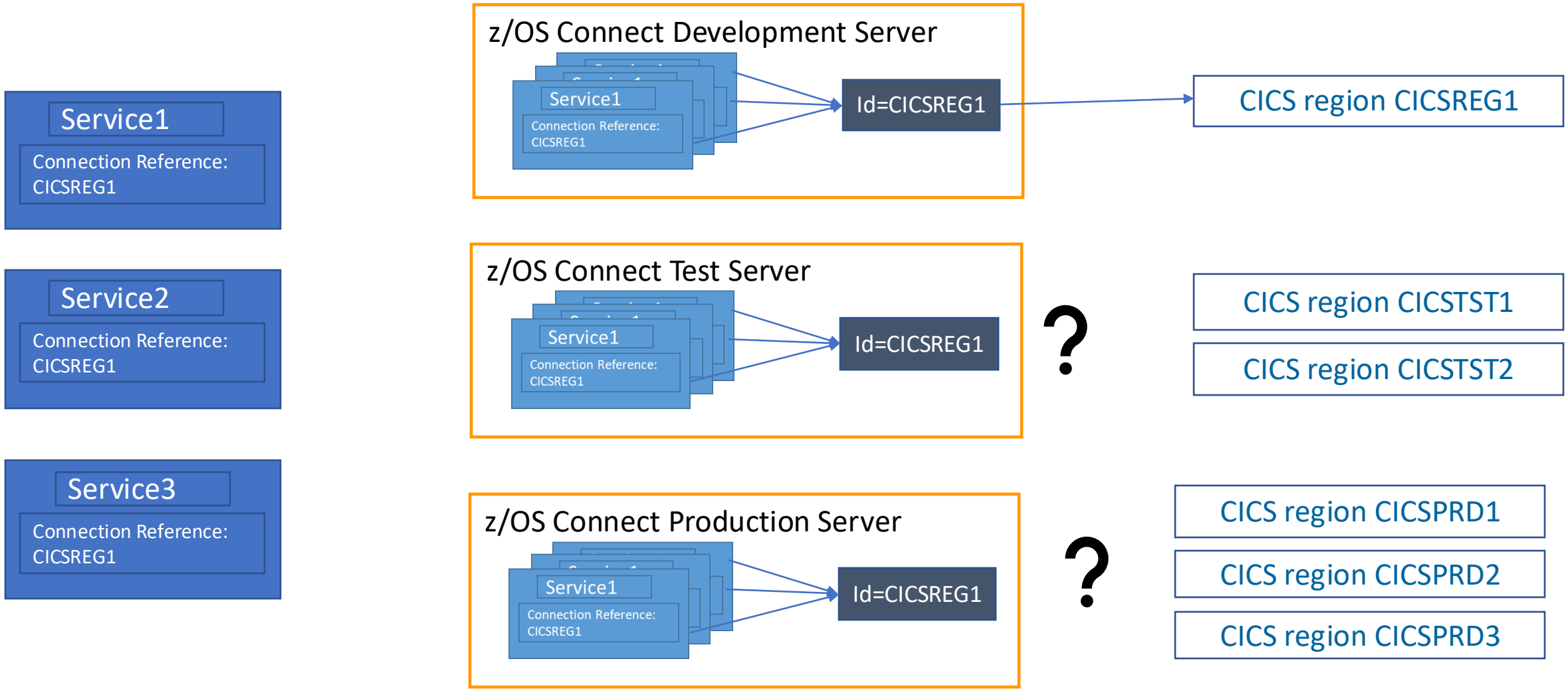

```
<zosconnect_zosConnectServiceRestClientConnection id="Db2Conn"
       sslCertsRef="db2SSLSettings" host="wg31.washington.ibm.com" port="2445" basicAuthRef="dsn2Auth" />
<zosconnect_zosConnectServiceRestClientBasicAuth id="dsn2Auth" applName="DSN2APPL"/>
<ssl id="db2SSLSettings"  keyStoreRef="Db2KeyStore"  trustStoreRef="Db2KeyStore"/>

<zosconnect_zosConnectServiceRestClientConnection id="fred"
       sslCertsRef="fredSSLSettings"  host="wg31.washington.ibm.com" port="2445" />
<ssl id="fredSSLSettings" keyStoreRef="Db2KeyStore"  trustStoreRef="Db2KeyStore" clientKeyAlias="FRED"/>

<zosconnect_zosConnectServiceRestClientConnection id="user1"
       sslCertsRef="user1SSLSettings"  host="wg31.washington.ibm.com"  port="2445" />
<ssl id="user1SSLSettings" keyStoreRef="Db2KeyStore"  trustStoreRef="Db2KeyStore" clientKeyAlias="USER1"/>

<keyStore id="Db2KeyStore" location="safkeyring:///zCEE.Db2.KeyRing"
       password="password" type="JCERACFKS" fileBased="false" readOnly="true" />
```

Requires APAR PH53162

mitchj@us.ibm.com

Use naming conventions for service/endpoint connection references

Don’t couple service and API requester connection names to specific systems or endpoints

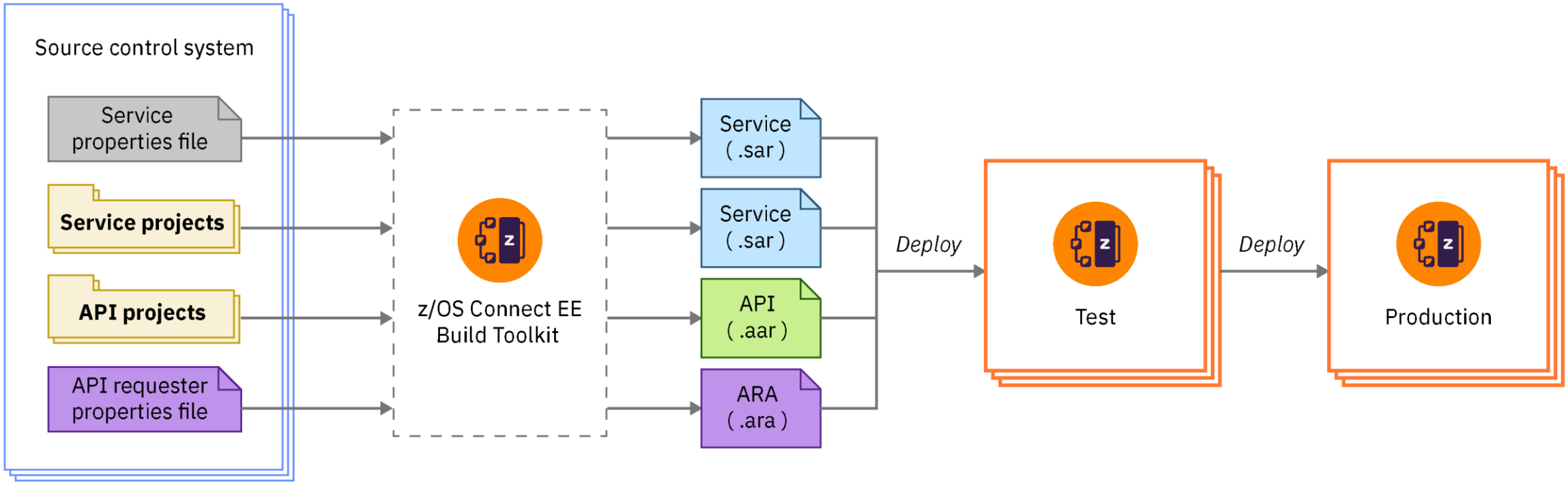


mitchj@us.ibm.com

DevOps using z/OS Connect RESTful Administrative APIs

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 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 deployment directories or by using the REST Admin API



mitchj@us.ibm.com

ibm.biz/zosconnect-devops

Agenda

- An Introduction to REST API

- Enabling RESTful API to z/OS resources, e.g.
  - CICS
  - Db2
  - IMS/TM
  - IMS/DB
  - MQ

- An overview of z/OS Connect API Provider Security

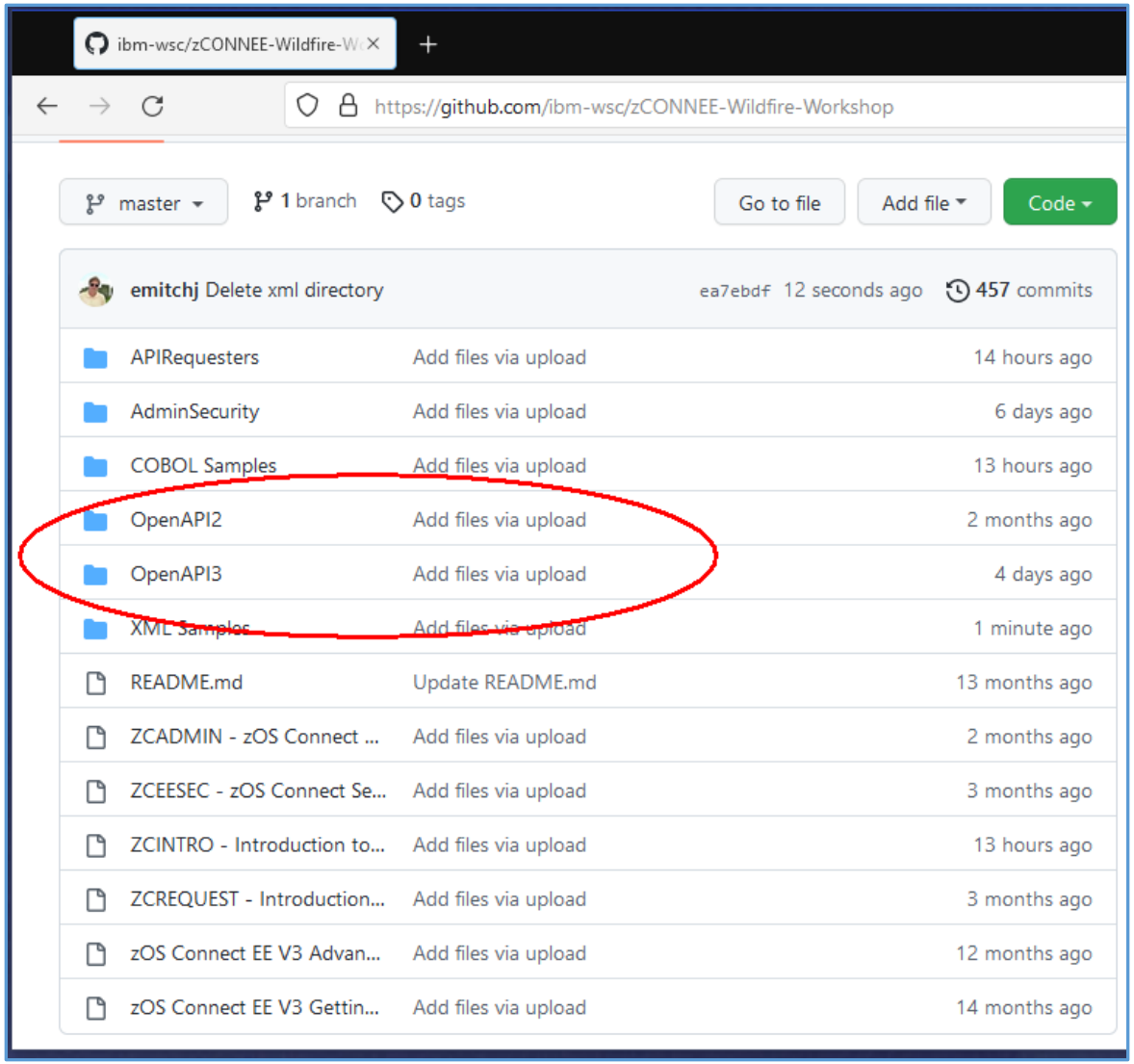
- Miscellaneous Topics

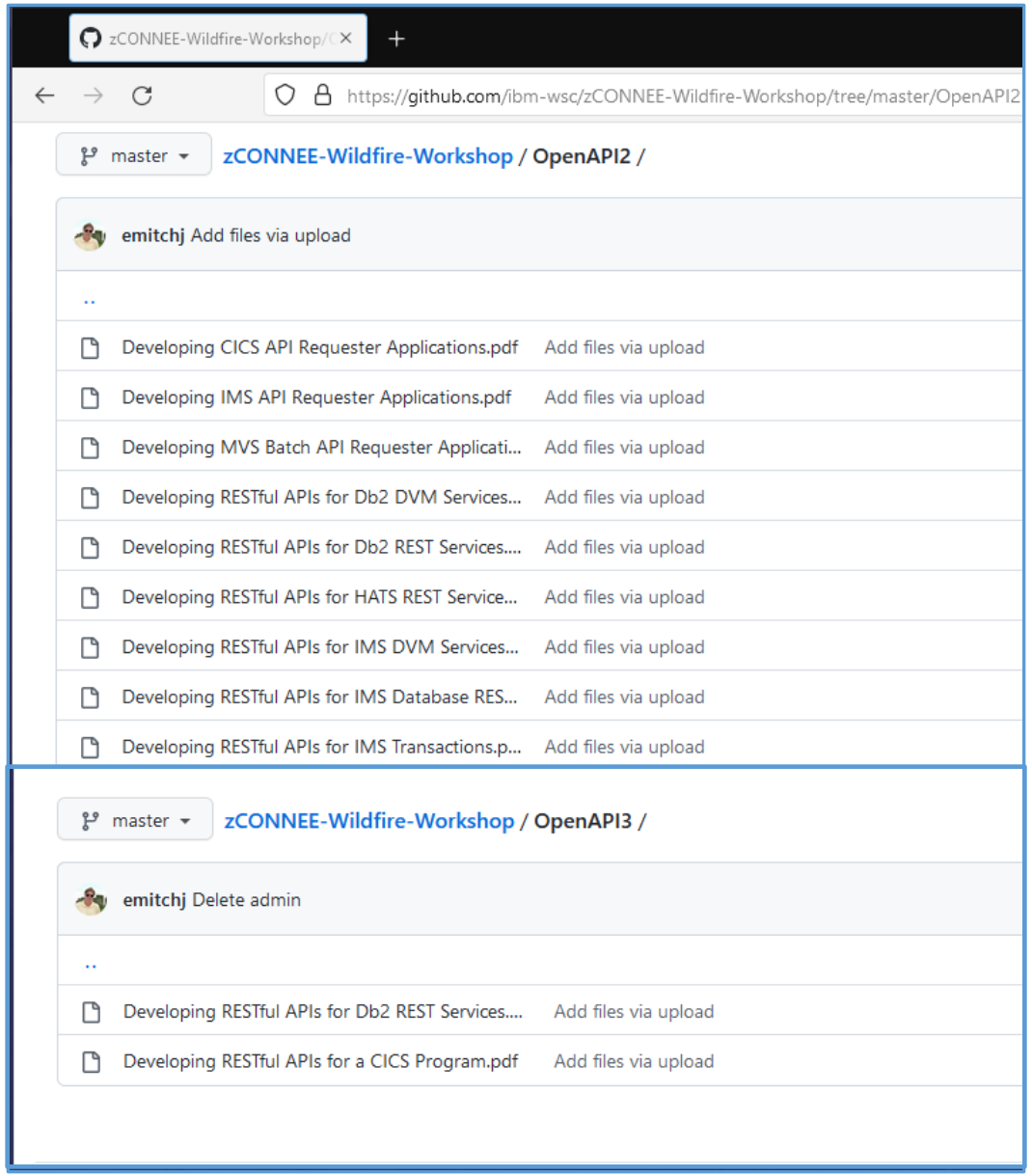
*

mitchj@us.ibm.com

z/OS Connect Wildfire Github Site

https://ibm.biz/zCEEWorkshopMaterial





- **Contact your IBM representative to schedule access to these exercises**

mitchj@us.ibm.com

IBM Washington Systems Center

The WSC Requests your Feedback!

Please scan the QR Code or go to this link:
https://tinyurl.com/bdey6tmv

For Event Code, Enter: **ZOSCON** + the current date **(DDMMYY)**

mitchj@us.ibm.com

Thank you for listening and your questions

mitchj@us.ibm.com