



# MQ Service Provider for z/OS Connect Enterprise Edition

Mitch Johnson

[mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)

Washington System Center

© 2017 IBM Corporation

## **`/first_what_is_REST?`**

What makes an API “RESTful”?

© 2017 IBM Corporation

2

# REST is an Architectural Style

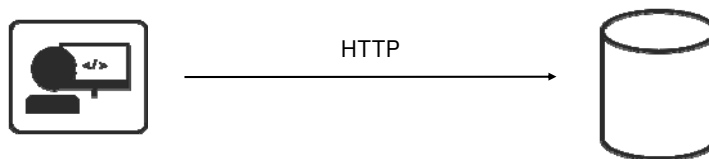


**REST** stands for **R**epresentational **S**tate **T**ransfer.

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

Typically using HTTP... but not all HTTP interfaces are "RESTful".

Simple and intuitive for the end consumer (**the developer**).



© 2017 IBM Corporation

3

# Key Principles of REST



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

GET  
POST  
PUT  
DELETE

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

Query Parameters are used for refinement of the request

URIs represent things (or lists of things)

Request/Response Body is used to represent the data object

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


© 2017 IBM Corporation

4


## RESTful Support



### z/OS Connect Enterprise Edition:

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

**GET** /account?number=1234

**PUT** /account?number=1234 +  (JSON with dollar amount of deposit)

↑  
HTTP Verb conveys the method against the resources; i.e., POST is for create, GET is for balance, etc.

↑  
URI conveys the resource to be acted upon; i.e., Fred's account with number 1234

↑  
The JSON body carries the specific data for the action (verb) against the resource (URI)

REST APIs are increasingly popular as an integration pattern because it is stateless, relatively lightweight, is relatively easy to program

<https://martinfowler.com/articles/richardsonMaturityModel.html>

5

## Some red flags...



(How to know if you are doing it wrong)

### 1. Unique URIs for different operations on the same object

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

### 2. Different representations of the same objects

<p>POST http://www.acme.com/customers          BODY { "firstName": "Joe",                "lastName": "Bloggs",                "addr": "10 Old Street",                "phoneNo": "01234 0123456" }</p>	→	<p>RESPONSE HTTP 201 CREATED          BODY { "id": "12345",                "name": "Joe Bloggs",                "address": "10 New Street",                "tel": "01234 0123456" }</p>
--	---	---

### 3. Operational data in the request body

<p>POST http://www.acme.com/customers/12345          BODY { "updateField": "address",                "newValue": "10 New Street" }</p>	→	<p>RESPONSE HTTP 200 OK          BODY { "id": "12345",                "name": "Joe Bloggs",                "address": "10 New Street",                "tel": "01234 123456" }</p>
--	---	---

© 2017 IBM Corporation

7

## Why is REST popular?



MQ Service Provider for  
z/OS Connect EE

<b>Ubiquitous Foundation</b>	It's based on HTTP, which operates on TCP/IP, which is a ubiquitous networking topology.
<b>Relatively Lightweight</b>	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.
<b>Relatively Easy Development</b>	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.
<b>Increasingly Common</b>	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.
<b>Stateless</b>	REST is by definition a stateless protocol, which implies greater simplicity in topology design. There's no need to maintain, replicate or route based on state.

## How do we describe a REST API?



## /swagger/open\_api

The industry standard framework for describing RESTful APIs.

## Why use Swagger?

It is more than just an API framework



There are a number of tools available to aid consumption:

### Write Swagger

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



### Read Swagger

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



### Consume Swagger

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



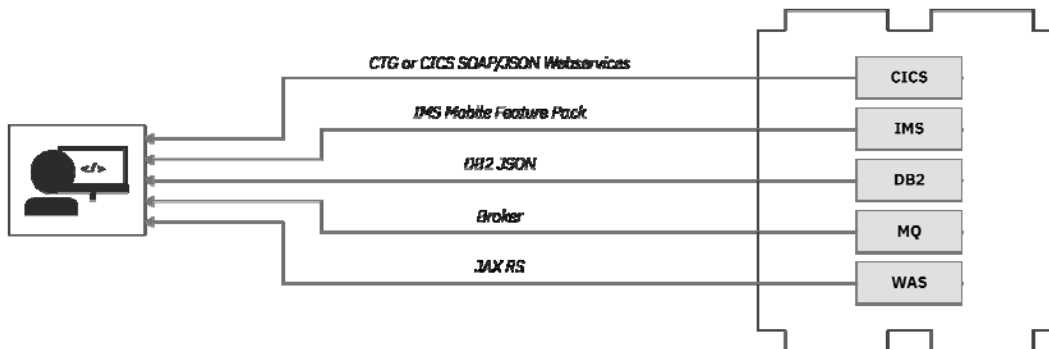


# /zos\_connect\_ee

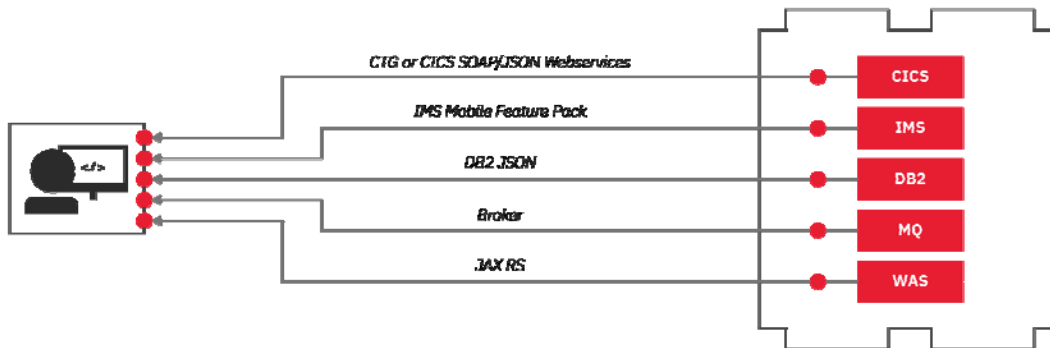
Truly RESTful APIs to and from your mainframe.

## Can we do this today?

MQ Service Provider for z/OS Connect EE



# Can we do this today?



- Completely different configuration and management.
- Multiple endpoints for developers to call/maintain access to.
- These are typically not RESTful!

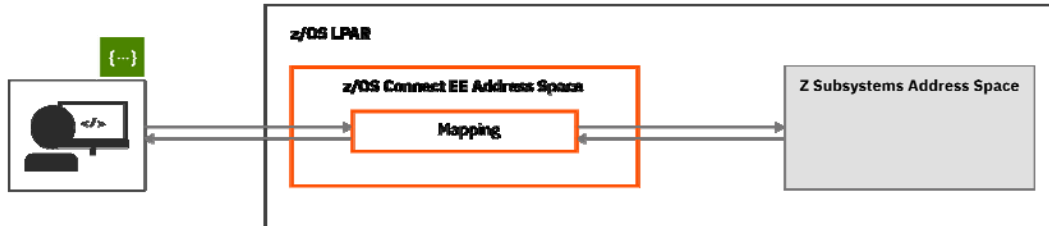
# You need a single entry point!



With sophisticated mapping of truly RESTful APIs to existing mainframe and services data without writing any code.

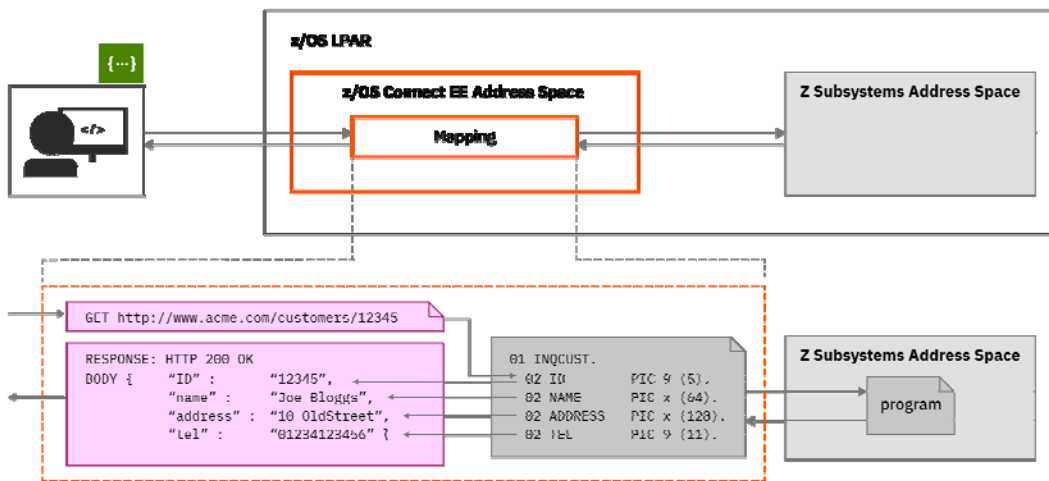
# Data mapping

## Overview



# Data mapping

## A closer look





## COBOL source versus JSON schema



```

01 MINILOAN-COMMAREA.
   10 name pic X(20).
   10 creditScore pic 9(16)V99.
   10 yearlyIncome pic 9(16)V99.
   10 age pic 9(10).
   10 amount pic 9999999V99.
   10 approved pic X.
       88 BoolValue value 'T'.
   10 effectDate pic X(8).
   10 yearlyInterestRate pic S9(5).
   10 yearlyRepayment pic 9(18).
   10 messages-Num pic 9(9).
   10 messages pic X(60) occurs 1 to 99 times
       depending on messages-Num.
  
```

COBOL  
Source

```

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

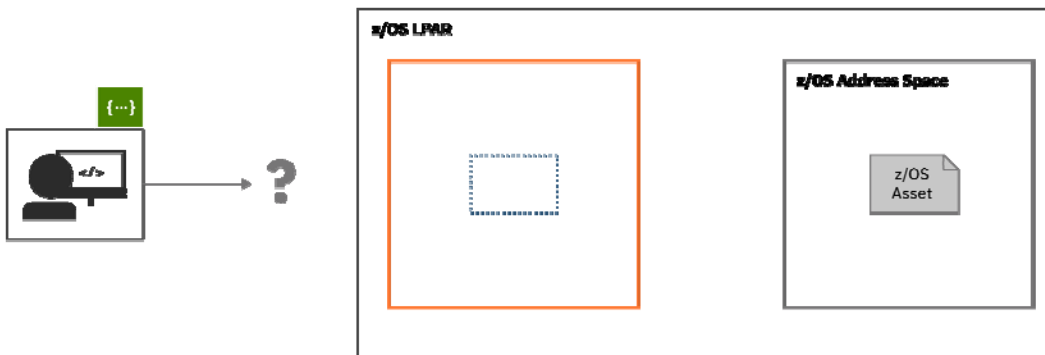
JSON Schema  
equivalent

All information is sent as  
character string and  
numeric precision is  
reduced as an issue

## Six Steps to expose a z/OS Asset



### 1. Install z/OS Connect EE



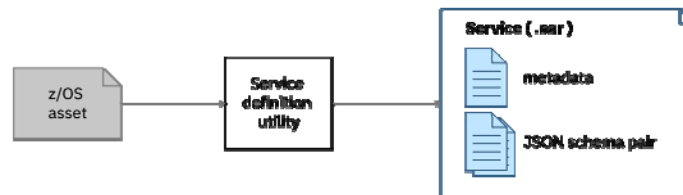
Install, set up, and start a **z/OS Connect EE Server**.

## Six Steps to expose a z/OS application



### 2. Create your service definition

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



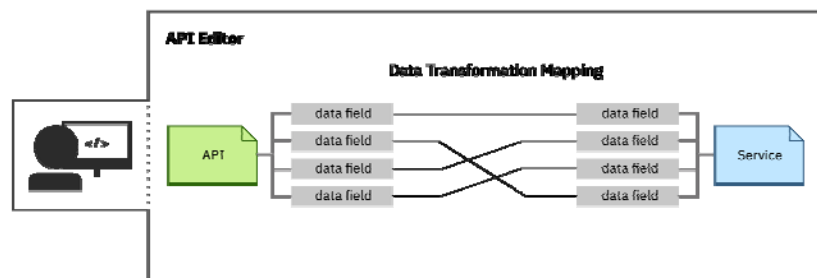
Use a system-appropriate utility to generate a .sar file for the z/OS application

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

## Six Steps to expose a z/OS application



### 3. Create your API



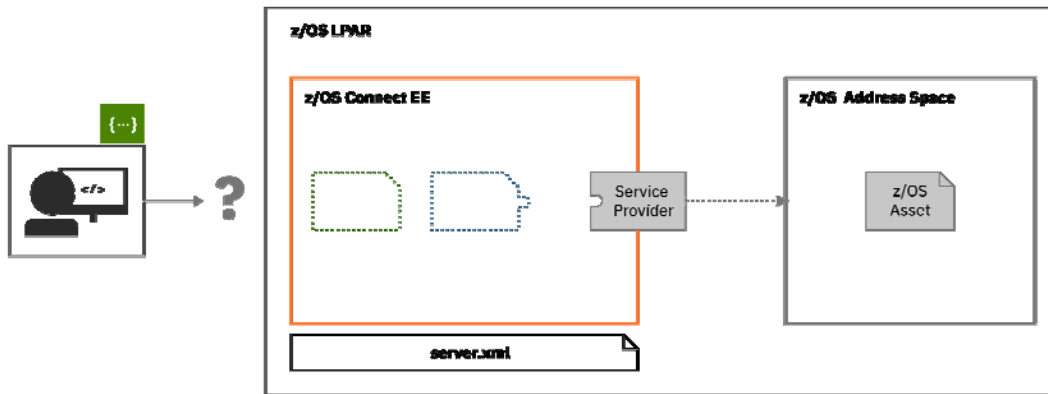
Import your .sar file into the **API toolkit**, and start designing your API.

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

## Six Steps to expose a z/OS application



### 4. Configure your service provider



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

© 2017 IBM Corporation

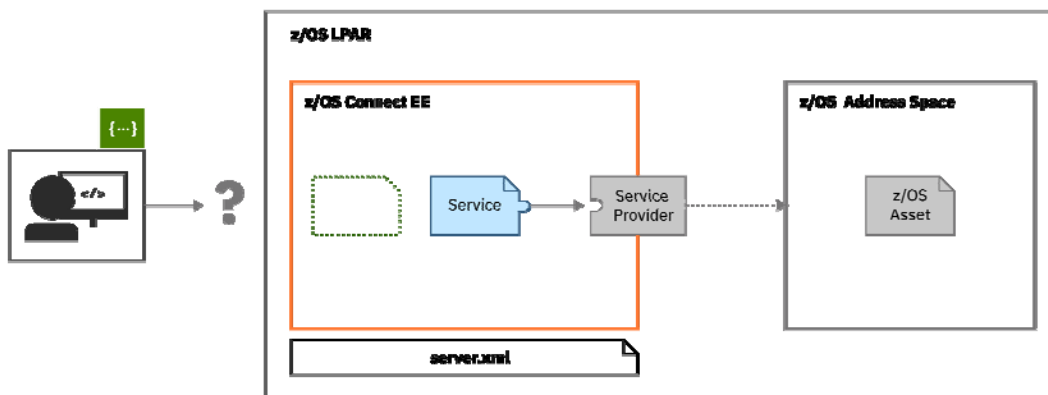
[ibm.biz/zosconnect-configuring](http://ibm.biz/zosconnect-configuring)

22

## Six Steps to expose a z/OS application



### 5. Deploy your service



Deploy the `.sar` file generated by the *service definition utility* by copying the `.sar` file to the *services* directory. (This step uses the `.sar` file generated in **Step 2**.)

© 2017 IBM Corporation

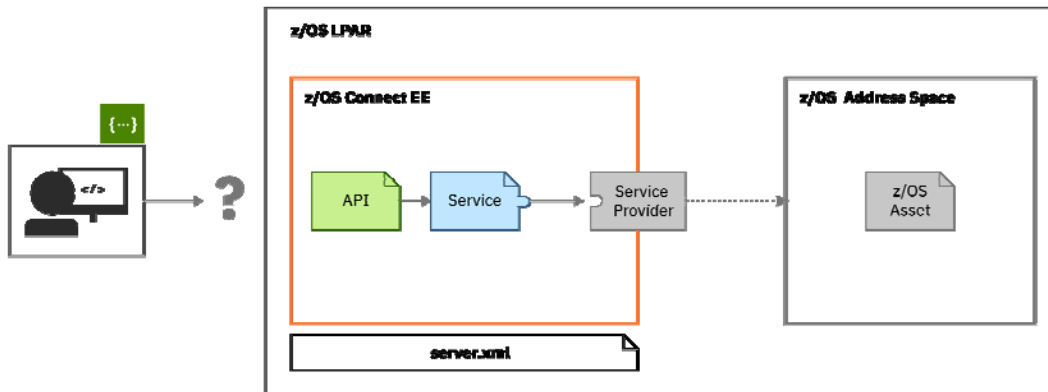
[ibm.biz/zosconnect-define-services](http://ibm.biz/zosconnect-define-services)

23

## Six Steps to expose a z/OS application

MQ Service Provider for  
z/OS Connect EE

### 6. Deploy your API



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

© 2017 IBM Corporation

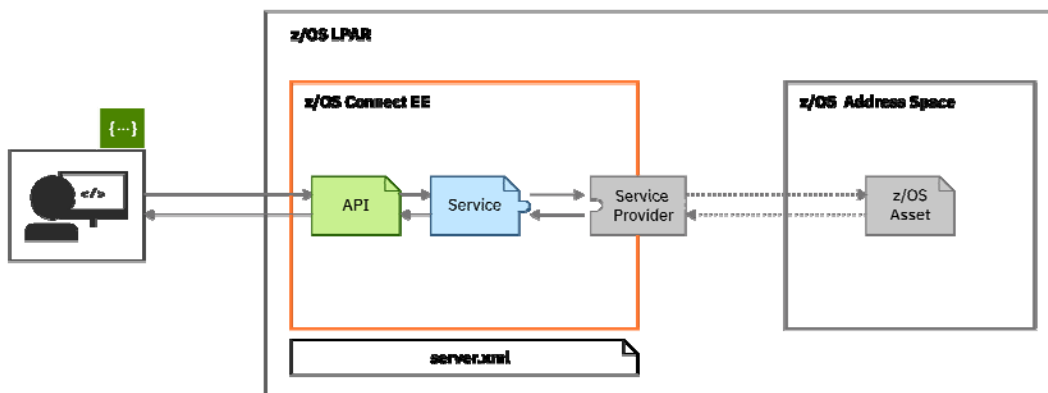
[ibm.biz/zosconnect-deploy-api](http://ibm.biz/zosconnect-deploy-api)

24

## Six Steps to expose a z/OS application

MQ Service Provider for  
z/OS Connect EE

Done



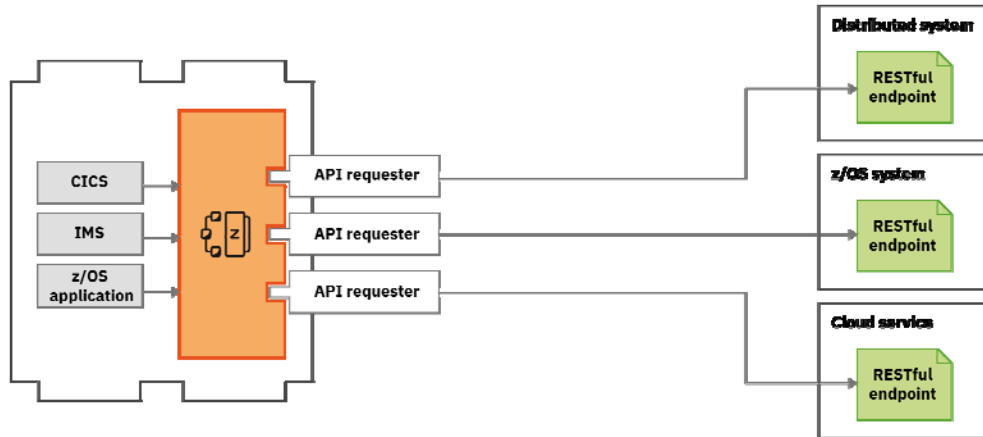
Your API is ready to be consumed: go tell your developers!

© 2017 IBM Corporation

25

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

 MQ Service Provider for z/OS Connect EE



*Including MQ*

© 2017 IBM Corporation

26



## /api\_toolkit/part1

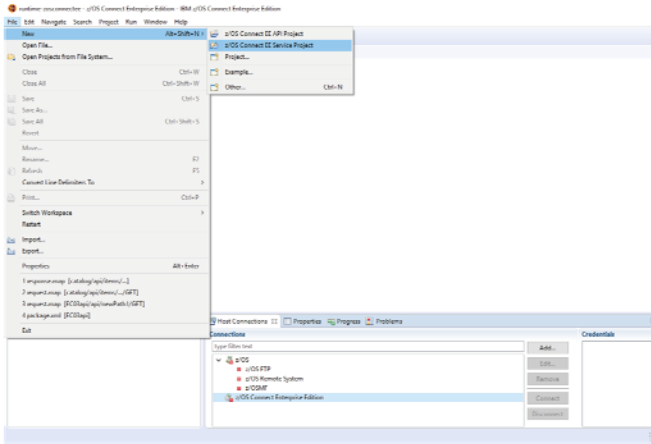
Simple **service creation** for MQ.

© 2017 IBM Corporation

27

# API toolkit

## Creating a service

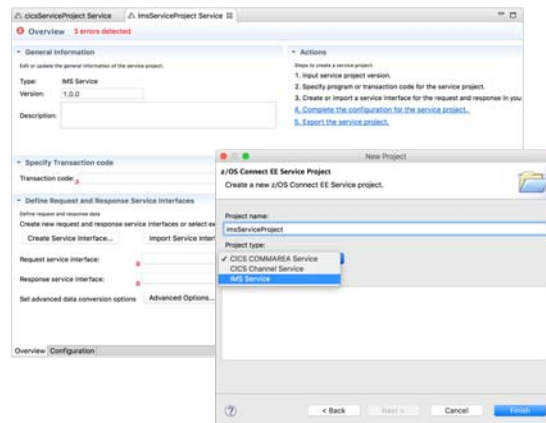
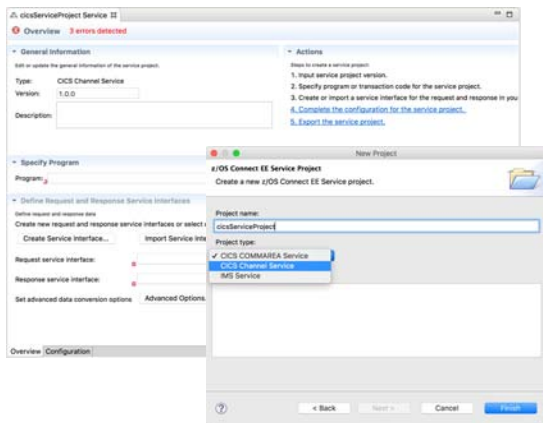


Use the **API toolkit** to create services through Eclipse-based tooling.

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

# API toolkit

## Service creation – a common interface

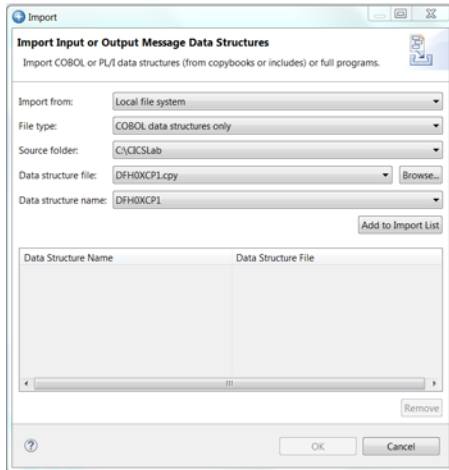


A common interface for service creation, agnostic of back end subsystem.

## API toolkit



### Creating a service



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

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

© 2017 IBM Corporation

30

## API toolkit



### Creating a service – request message

Fields	Include	Interface rename	Default Field Value	Data Type	Field Length	Start Byte
COMMMAREA						
DFHOXCP1						
CA_REQUEST_ID	<input type="checkbox"/>	CA_REQUEST_ID	01INVQ5	CHAR	6	1
CA_RETURN_CODE	<input type="checkbox"/>	CA_RETURN_CODE		DECIMAL	2	7
CA_RESPONSE_MESSAGE	<input type="checkbox"/>	CA_RESPONSE_MESSAGE		CHAR	79	9
% CA_REQUEST_SPECIFIC (Redefine)	<input type="checkbox"/>	CA_REQUEST_SPECIFIC		CHAR	911	88
% CA_INQUIRE_REQUEST redefines	<input type="checkbox"/>	CA_INQUIRE_REQUEST		STRUCT	911	88
% CA_INQUIRE_SINGLE redefines C	<input type="checkbox"/>	inquireSingle		STRUCT	911	88
CA_ITEM_REF_REQ	<input checked="" type="checkbox"/>	itemID		DECIMAL	4	88
FILL_0	<input type="checkbox"/>	FILL_0		DECIMAL	4	92
FILL_1	<input type="checkbox"/>	FILL_1		DECIMAL	3	96
CA_SINGLE_ITEM	<input type="checkbox"/>	CA_SINGLE_ITEM		STRUCT	60	99
FILL_2	<input type="checkbox"/>	FILL_2		CHAR	840	159
% CA_ORDER_REQUEST redefines C	<input type="checkbox"/>	CA_ORDER_REQUEST		STRUCT	911	88

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

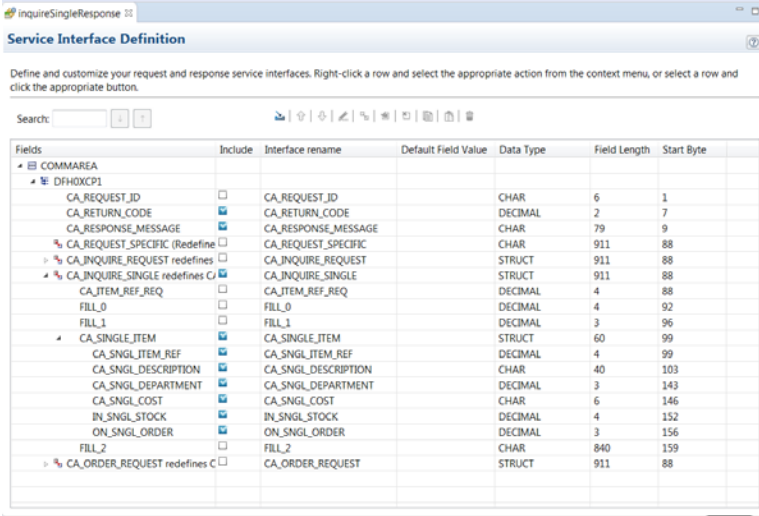
© 2017 IBM Corporation

31

# API toolkit



## Creating a service – response message

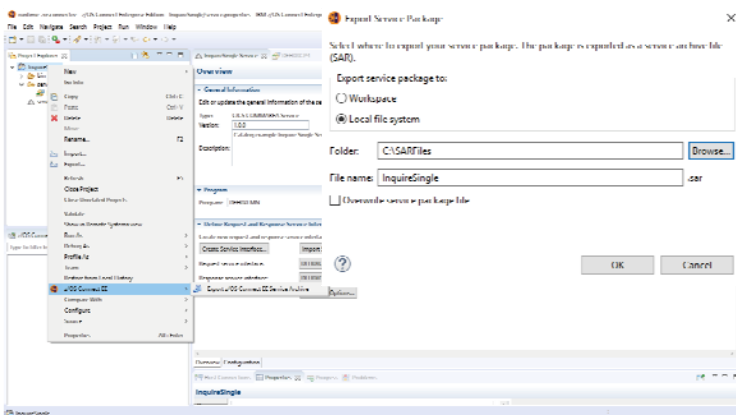


You can then see the imported data structure and can **redact fields** and **rename fields**

# API toolkit



## Creating a service for CICS and IMS



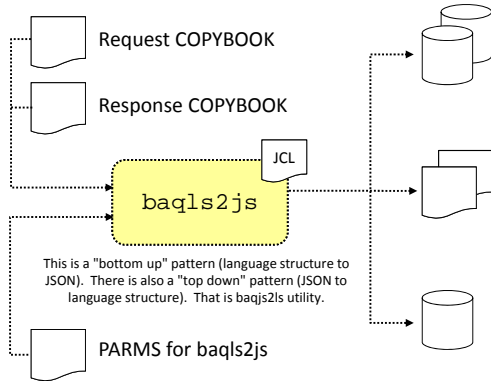
Finally, you can export the service project as a **Service Archive file (.sar)**.



# Creating Services without the Toolkit



For MQ and WOLA use the supplied conversion utility BAQLS2JS



This is a "bottom up" pattern (language structure to JSON). There is also a "top down" pattern (JSON to language structure). That is baqls2js utility.

### BIND Files

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

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

### JSON Schema Files

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

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

### Service Archive (SAR) File

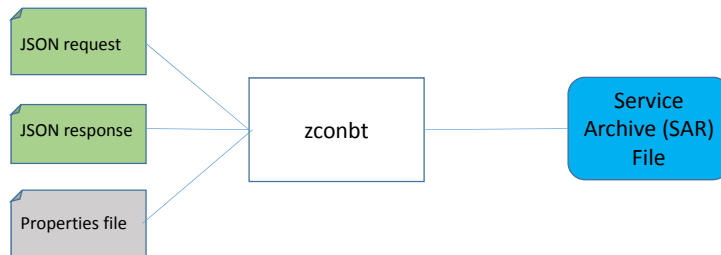
This is a ZIP-format file that contains the JSON schema and some meta-data. This is input to the API Editor (next unit) to create the APIs.

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

# Creating Services without the Toolkit



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



Generate the service archive file



## /api\_toolkit/part2

Quick and easy **API mapping**.

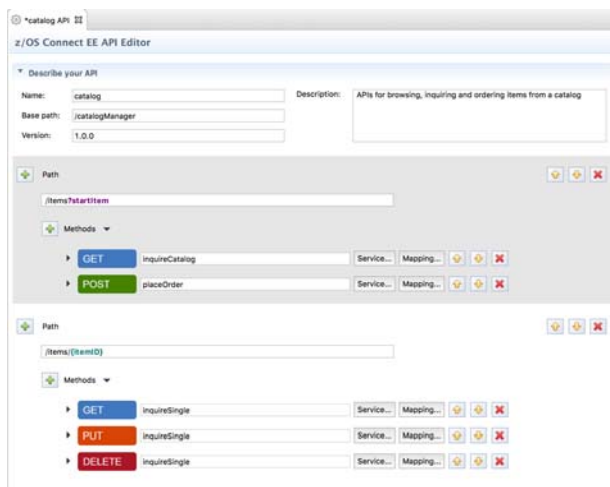
© 2017 IBM Corporation

36

## API toolkit

### API definition

 MQ Service Provider for  
z/OS Connect EE



The screenshot shows the 'z/OS Connect EE API Editor' window. It has a tab for '\*catalog API II'. The 'Describe your API' section includes fields for Name (catalog), Base path (/catalogManager), and Version (1.0.0). The Description is 'APIs for browsing, inquiring and ordering items from a catalog'. Below this, there are two API paths defined:

- Path 1:** /Items?startItem
  - GET:** InquireCatalog (Service... Mapping...)
  - POST:** placeOrder (Service... Mapping...)
- Path 2:** /Items/{itemId}
  - GET:** InquireSingle (Service... Mapping...)
  - PUT:** InquireSingle (Service... Mapping...)
  - DELETE:** InquireSingle (Service... Mapping...)

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

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

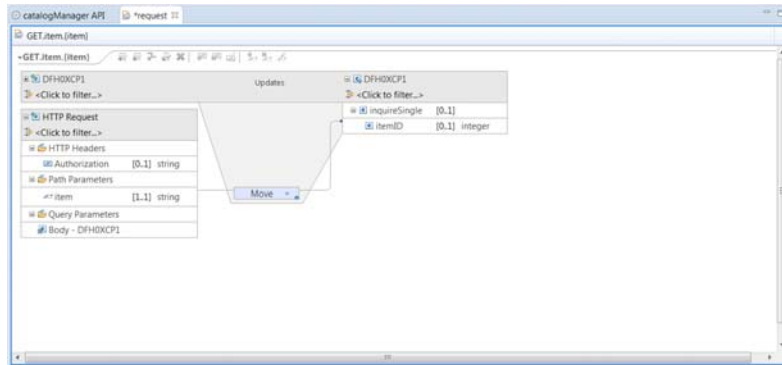
Your services are represented by `.sar` files, which you import into the **API toolkit**, regardless of how the `.sar` was generated.

© 2017 IBM Corporation

37

## API toolkit

### API mapping: Point-and-click interface



Map both the request and response for each API.

Map path and query parameters to native data structures.

Assign static values to fields, useful for Op codes.

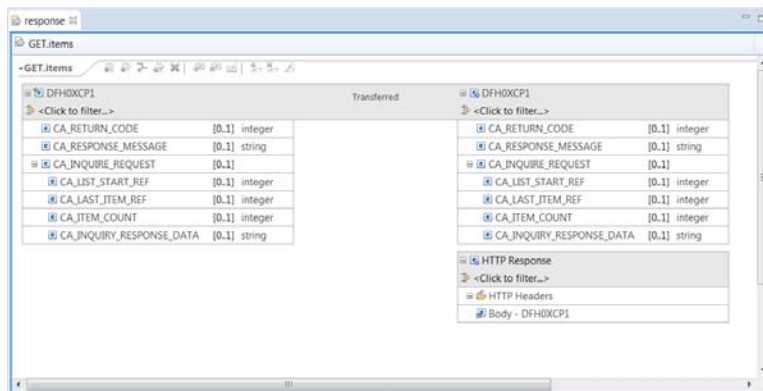
Remove unwanted fields to simplify the API (remember request was set to 01INQC in the SAR).

© 2017 IBM Corporation

38

## API toolkit

### API mapping: Point-and-click interface



Allows the API Developer to remove fields to simplify the API

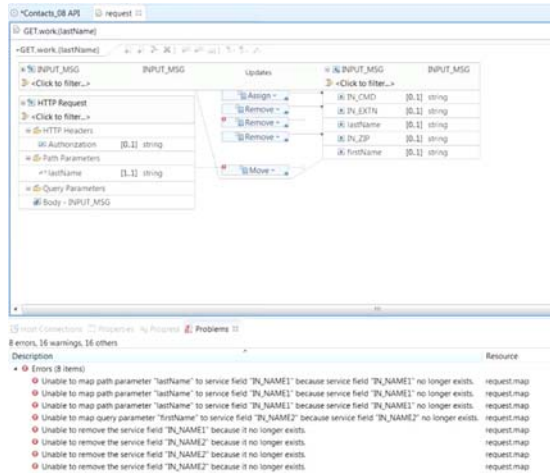
© 2017 IBM Corporation

39

## API toolkit



### Change management



Mappings are preserved when `.sar` files change (for example, when a copybook is updated).

Impact analysis shows you what will break before you import the new `.sar` file.

Broken mappings are highlighted in the **Eclipse Problems** view and in the mapping editor.

© 2017 IBM Corporation

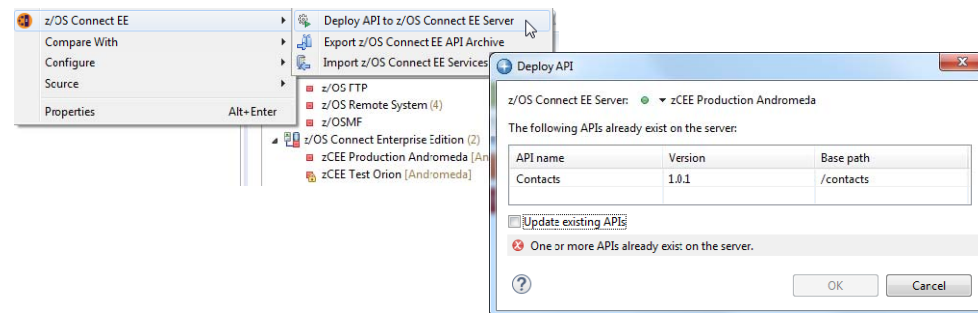
40

## API toolkit



### Server connection and API deployment

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



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

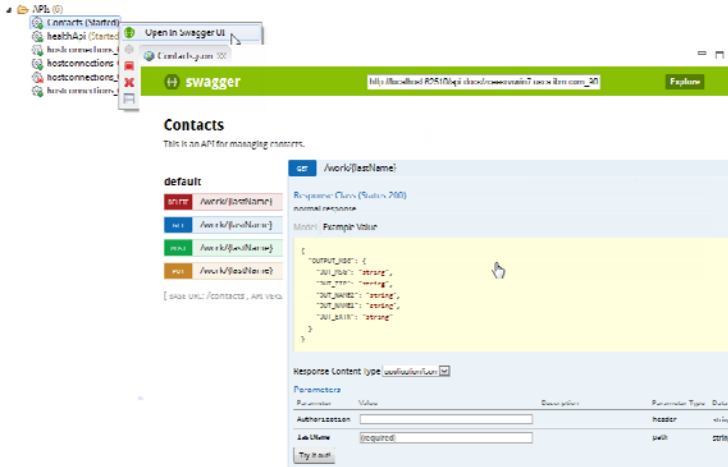
**z/OS Connect EE servers view** allows you to start, stop, and remove APIs from a running server.

© 2017 IBM Corporation

41

# 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.



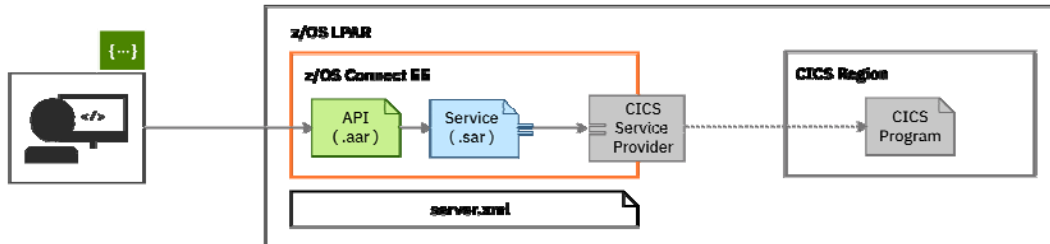
## /common\_scenarios

Typical connection patterns to different subsystems.

# Connect to CICS



## Topology



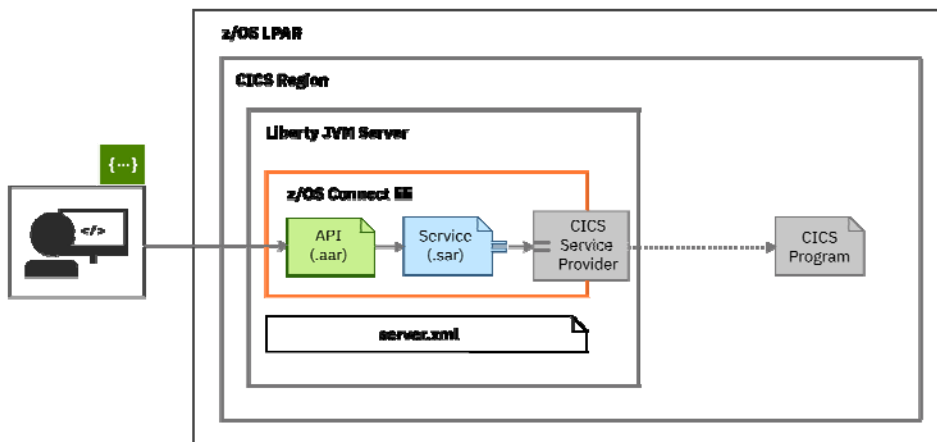
Connection to CICS is configured in `server.xml`.

An IPIC connection must be configured in CICS.

# Connect to CICS (CICS-embedded-scenario)



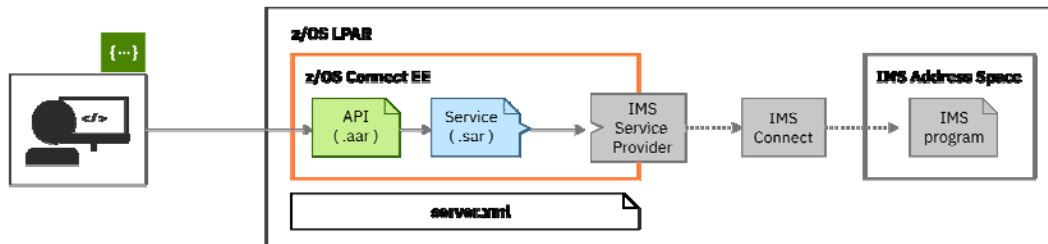
## Topology



## Connect to IMS



### Topology



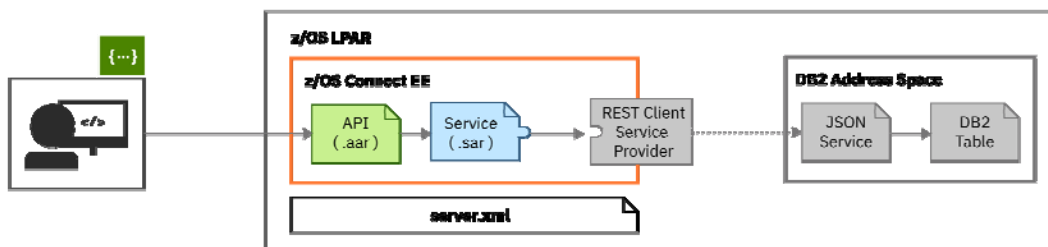
Configure the connection to IMS through `ims-connections.xml` and `ims-interactions.xml` in the IMS service registry.

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

## Connect to DB2



### Topology



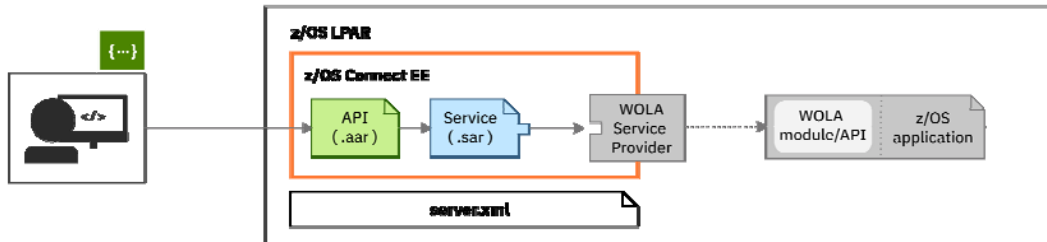
Connection to the JSON Service is configured in `server.xml`.

A JSON Service must be configured in DB2.

## Connect to a WOLA-enabled z/OS application



### Topology



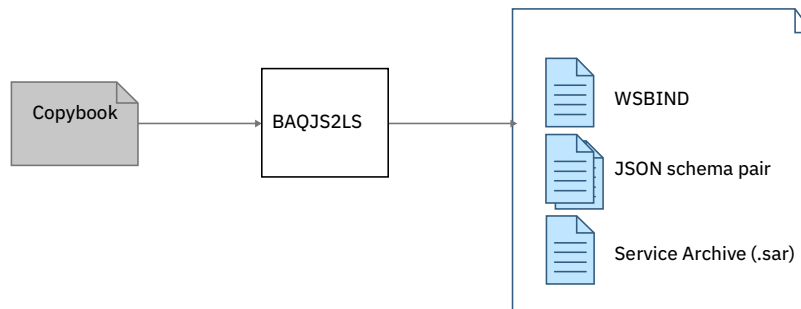
Connection to WOLA is configured in `server.xml`.

The z/OS application must be WOLA-enabled.

## Connect to MQ



### Create the service definition



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



## Generating z/OS Connect EE artifacts



```
//ASSIST EXEC PGM=BPXBATCH
//STDPARM DD DSN=USER1.ZCEE.INPUT(MINILOAN),DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *
JAVA_HOME=/usr/lpp/java/J8.0_64
//
```

```
PGM /shared/IBM/zosconnect/v2r0/bin/baqls2js
PDSLIB=USER1.ZCEE.CNTL
REQMEM=MINILOAN
RESPMEM=MINILOAN
DATA-TRUNCATION=ENABLED
MAPPING-LEVEL=4.0
CHAR-VARYING=COLLAPSE
JSON-SCHEMA-REQUEST=/var/ats/zosconnect/servers/server1/dataXform/json/Miniloan_request.json
JSON-SCHEMA-RESPONSE=/var/ats/zosconnect/servers/server1/dataXform/json/Miniloan_request.json
LANG=COBOL
LOGFILE=/var/ats/zosconnect/servers/server1/dataXform/Miniloan.log
PGMINT=COMMAREA
PGMNAME=ATSCMINX
WSBIND=/var/ats/zosconnect/servers/server1/dataXform/bind/Miniloan.wsbind
SERVICE-ARCHIVE=/var/ats/zosconnect/servers/server1/dataXform/sars/Miniloan.sar
SERVICE-NAME=Miniloan
URI=http://server1.example.org:8080/Miniloan
```

73

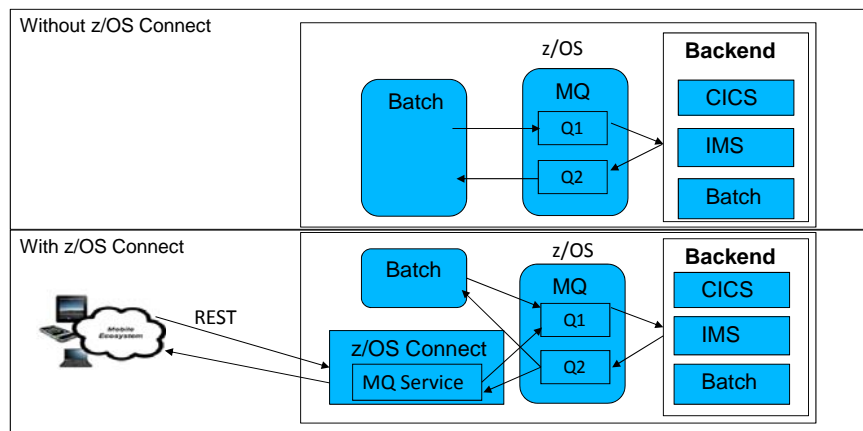
## The MQ Service Provider



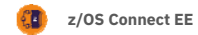
Free of charge z/OS Connect service provider that allows existing services that are fronted by MQ to be accessed via a RESTful front end

- Both z/OS Connect V1 and z/OS Connect EE V2 and later are supported
- Same capabilities in both versions

Clients need to have no knowledge of MQ



## Service types

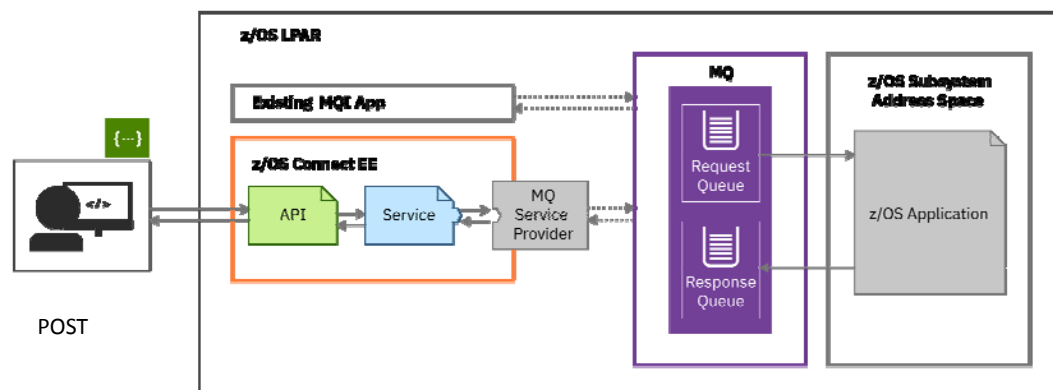


- Each URL in z/OS Connect maps to a service
- With the MQ Service Provider there are two different types of service
  - Two way services
  - One way services
- A two way service provides request/reply messaging:
  1. Client issues HTTP POST with some payload (JSON)
  2. MQ Service Provider sends payload (optional transformation) to one MQ queue
  3. Back end application processes payload and puts response on reply MQ queue
  4. MQ Service Provider gets response (optional transformation) and sends it to client as the body of the HTTP POST response
- A one way service exposes standard MQ verbs against a single destination
  - HTTP POST == MQPUT (queue and topic)
  - HTTP DELETE == MQGET (queue)
  - HTTP GET == MQGET (browse) (queue)

## Connect to MQ



Topology (Two-way service example)

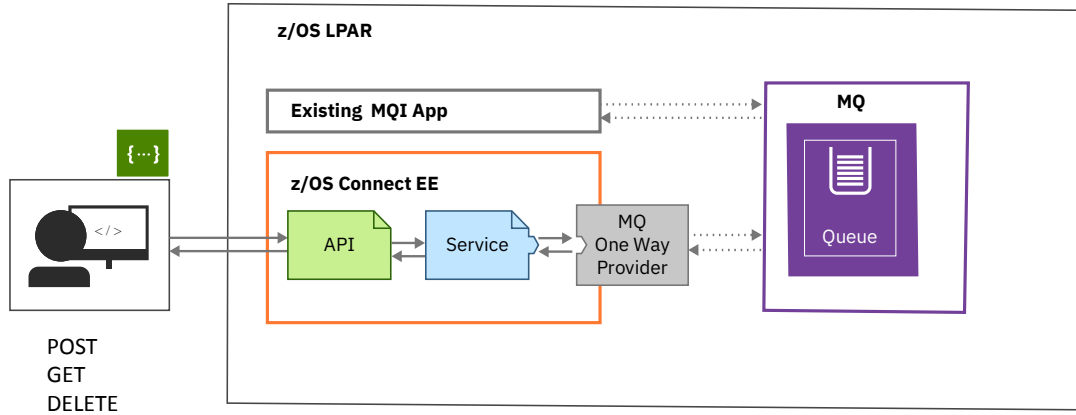


You can also configure one-way services.

# Connect to MQ



Topology (One-way service example)



© 2017 IBM Corporation

[ibm.biz/zosconnect-mq-service-provider](http://ibm.biz/zosconnect-mq-service-provider)

77

# The server.xml File (MQ)



```
<featureManager>
  <feature>jms-2.0</feature>
  <feature>mqzosconnect:zosConnectMQ-2.0</feature>
  <feature>wmqJmsClient-2.0</feature>
  <feature>zosTransaction-1.0</feature>
</featureManager>

<variable name="wmqJmsClient.rar.location"
  value="/usr/lpp/mqm/V9R0M2/java/lib/jca/wmq.jmsra.rar"/>
<wmqJmsClient nativeLibraryPath="/usr/lpp/mqm/V9R0M2/java/lib"/>

<zosconnect_zosConnectService id="miniloan"
  dataXformRef="xformJSON2Byte"
  serviceName="Miniloan"
  serviceRef="Miniloan" />

<mqzosconnect_mqzOSConnectService id="Miniloan"
  connectionFactory="jms/qmgrCF"
  waitInterval="30000"
  destination="jms/request" replyDestination="jms/response" />

<jmsConnectionFactory id="qmgrCF" jndiName="jms/qmgrCF"
  connectionManagerRef="ConMgr1">
  <properties.wmqJMS transportType="BINDINGS" queueManager="MQS1" />
</jmsConnectionFactory>

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

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

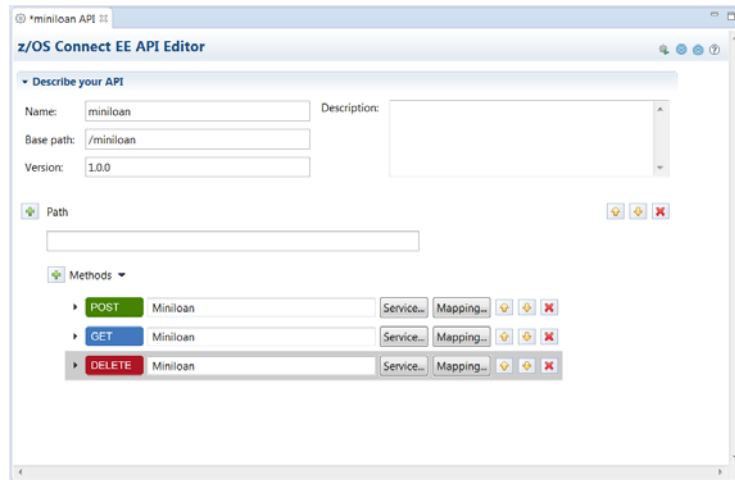
Features related to JMS Support

JMS Connection Factories,

78

## API Editor Eclipse Tooling

MQ Service Provider for  
z/OS Connect EE

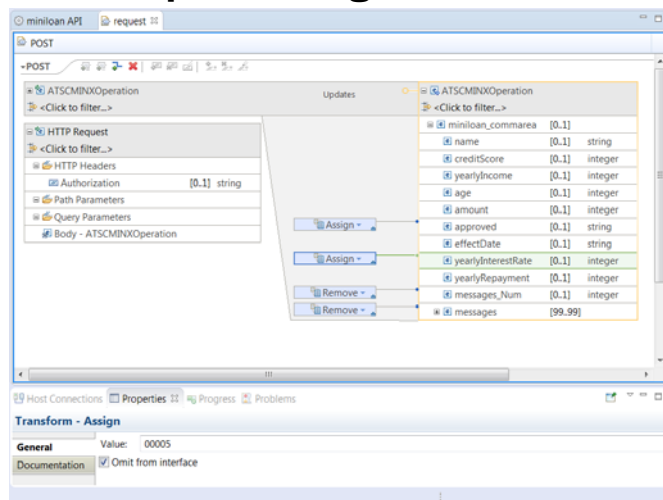


Service Archive (SAR) file imported into API Editor

80

## API Editor Eclipse Tooling

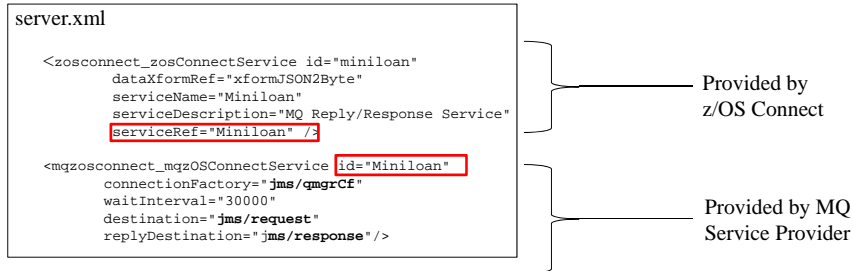
MQ Service Provider for  
z/OS Connect EE



Use the mapping tool to transform the request and response messages

81

## Two way Service Elements



- HTTP POST to **https://<hostname>:<port>/miniloan**
- All MQ related information is held in *mqzOSConnectService element*
  - Overridable via HTTP headers, e.g. *ibm-mq-gmo-waitInterval*
  - Builds on the MQ messaging provider in Liberty. Uses JMS

82

## Queue Manager and Queue Elements (JNDI)

```

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

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

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

83

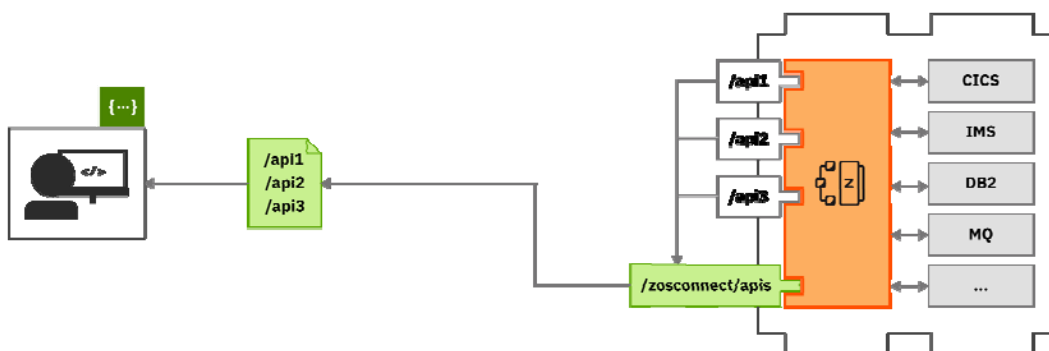


## **/zosconnect/apidocs**

Get the Swagger definitions for your APIs

## **API Documentation**

Get your Swagger on



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



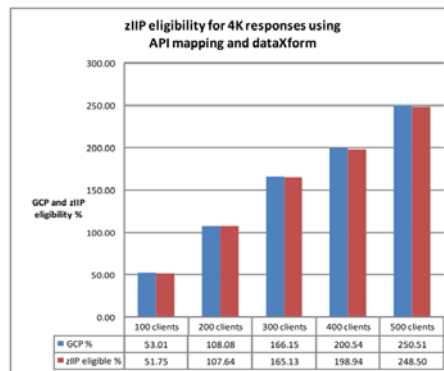
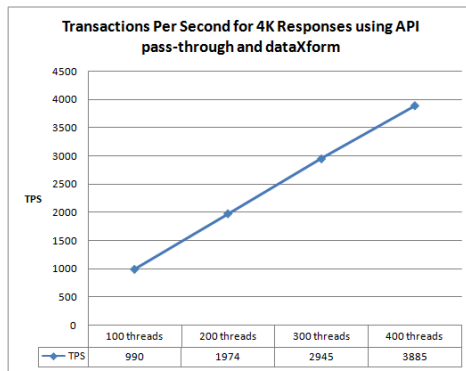
## /miscellaneousTopics

performance, high availability, Liberty

## Performance



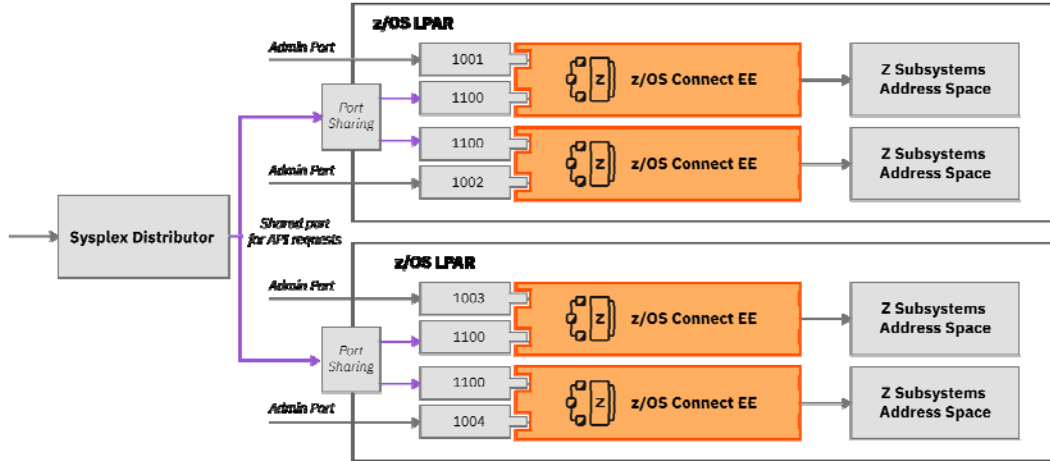
High Speed, High Throughput, Low Cost



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

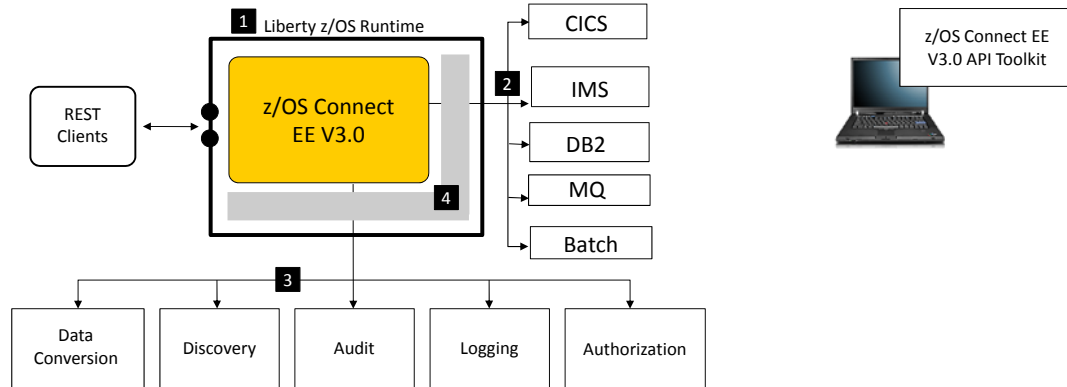
# High Availability

## Topology



# What Does z/OS Connect Provide?

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



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





**/questions?thanks=true**

Thank you for listening.