REST API Enablement of z/OS Subsystems with z/OS Connect Enterprise Edition & API Connect

Carsten Mai
IBM Software zHybridCloud

München, 22.9.2016

© Copyright IBM Corporation 2016.
Enterprises must learn to “Innovate like a Startup”

“The ‘Uber syndrome’ – where a competitor with a completely different business model enters your industry and flattens you.” Judy Lemke, CIO, Schneider, United States

“52% of the Fortune 500 firms since 2000 are gone.” – R. Ray Wang
The Strategy is Simple…

**Unleash Enterprise Investments to Disrupt Competitors**

Existing Enterprise IT Investments → Exposed as APIs → Self Service Consumed by Developers → To Develop Innovative Apps → Delivering Differentiated Customer Experiences
Two Types of API Appearing - System & Interaction APIs

**System APIs:**
APIs that pass through data from a system of record “unchanged”

**Interaction APIs:**
Invoke one or more System API’s or data sources, and manipulate the returned data with new logic

Promote reuse across new applications
z/OS Connect Enterprise Edition (EE) V2.0

- Provides “System APIs” creation for z/OS subsystem applications
- Integrates with “IBM API Connect” (more on this later) for enterprise-class API management
  - Create, Run, Secure
- Delivers RESTful APIs as a discoverable, first-class resource with OpenAPI Spec (Swagger 2.0) descriptions
  - Ready for consumption by today's enterprise application developers and integration with API management solutions
- Comprehensive tooling that enables API developers to create RESTful APIs from z/OS-based assets
- Supports standard JSON message format and conversion to z/OS subsystem backend format requirements

Announce Dec 1, 2015
GA Dec 11, 2015
**Simplified Overview of REST/JSON**

**REST** - “Representational State Transfer” ... which uses HTTP and HTTP verbs to allow a client to interact with a server over the TCP/IP network.

**JSON** - “JavaScript Object Notation” ... a name/value pair representation of data that is relatively lightweight and generally simpler to handle and parse than XML.

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

### REST Client

- **POST /account/deposit**
  ```json
  { "account" : "12345", "amount" : "100.00" }
  ```

### REST Listener

- **GET /account/balance**
  ```json
  { "account" : "12345", "amount" : "100.00" }
  ```

CICS, IMS, DB2, or wherever account data is maintained
Mid-Tier Function or Device
REST-handling and data conversion handled in the mid-tier, and other connectivity mechanism (SOAP/HTTP, JCA, JMS, JDBC) used to connect to backend.

Direct to z/OS Backend
REST-handling and data conversion handled by each backend system (such as CICS or IMS).

"Gateway" to the LPAR
REST-handling and data conversion handled by a function on the LPAR, with backend systems unchanged.
1. z/OS Connect EE a Java servlet
2. “Service Provider” = backend connectivity
3. “Interceptors” = configurable function
4. Extensible interface = flexibility

A long-running task using WOLA APIs to host a service via IMS Connect Type 2 or 4.
High-Level Overview of z/OS Connect EE V2.0

1. **Runtime Server**
   - Hosts APIs you define to run
   - Connects with backend system
   - You may have multiple instances

2. **Tooling Platform**
   - Integrates with an Eclipse environment
   - Define APIs
   - Define data mapping
   - Deploy APIs to runtime server
   - Export API archive for other tools to deploy

---

**Eclipse**
- IBM CICS Explorer V5.3
- IBM IMS Explorer for Development V3.2
- IBM Explorer for z/OS Aqua V3.0

**Backend Systems**
- IBM z/OS V1R13, V2.1+
- IBM 64-bit SDK for z/OS, Java Technology Edition V7.1.0 or V8.0.0
RESTful Support

z/OS Connect EE V2.0:

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

**DELETE**  /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)
Eclipse-based Tooling for z/OS Connect EE V2.0

Eclipse project view, which is familiar to developers who have used Eclipse-tooling for other development projects.

Assign API function based on HTTP verb.

Access query parameters from the URI.

Provide data mapping definitions to the service.

API projects can be exported and imported for portability between developers.

API definitions are created through the tool, which is consistent across backend systems (CICS, IMS, etc.)

YouTube Demo: https://youtu.be/HjE8wdvX3I0
The API mapping model adds a powerful abstraction layer between the API consumer and the underlying z/OS assets.

- Mapping of HTTP headers, path parameters (URI templates), and query parameters to the fields in the request message JSON body.
- Pass-through, redaction, or defaulting of fields in the request or response message JSON body.
- Mapping and defaulting of HTTP headers in the HTTP response message.
API Archive (AAR) -- API Packaging

API Archive (AAR) File
- ZIP-format file
- Contains Swagger documentation of service
- Contains JSON schema and API information
- Produced by tooling
- Exportable to server runtime | Consumable by tooling

Provides a standardized method for defining, transporting and deploying services
Because the service definitions have been encapsulated into a deployable unit, it becomes eligible for deployment by automated tools. This further enhances productivity.
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 V1 and EE supported
- Same capabilities in both versions
- Clients need 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 service provides request/reply messaging:
    - Client issues HTTP POST with some payload (JSON)
    - MQ Service Provider sends payload (optional transformation) to one MQ queue
    - Back end application processes payload and puts response on reply queue
    - MQ Service Provider gets response (optional transformation) and sends it to client as the body of the HTTP POST response

  - 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)

- Allows more advanced interactions with MQ
Example two-way service - configuration

```xml
<zosConnectService id="zosconnIQ"
    invokeURI="/iq"
    serviceName="iq"
    serviceRef="iq"
    dataXformRef="xformJSON2Byte"/>

<mqzOSConnectService id="iq"
    connectionFactory="jms/cf1"
    destination="jms/iqRequestQ"
    waitInterval="10000"
    replySelection="msgIDToCorrelID"
    replyDestination="iqResponseQ"
    receiveTextCCSID="37" />

<zosConnectDataXform id="xformJSON2Byte"
    bindFileLoc="$\{XFORM_ROOT\}/bindfiles"
    bindFileSuffix=".bnd"
    requestSchemaLoc="$\{XFORM_ROOT\}/json"
    requestSchemaSuffix=".json"
    responseSchemaLoc="$\{XFORM_ROOT\}/json"
    responseSchemaSuffix=".json" />
```
Delivery

- MQ service provider for z/OS Connect
  - Provided in USS component of MQ 9.0.1
  - Supported with MQ 8 and MQ 9 queue managers

- Will also be made available on Fix Central

- Only contains the MQ service provider feature. You need to provide z/OS Connect (V1 or V2)
The inclusion of OpenAPI Spec 2.0 support in z/OS Connect EE V2.0 makes exchange of API information standardized, which provides compatibility with a wider set of devices and functions.
Two Types of API Appearing - System & Interaction APIs

**System APIs:**
APIs that pass through data from a system of record “unchanged”

**Interaction APIs:**
Invoke one or more System API’s or data sources, and manipulate the returned data with new logic

Promote reuse across new applications
To Build Interaction API's, and Manage, Secure and Run all your APIs:

API Connect V5.0

Create
Run
Secure
Manage
IBM API Connect V5

Simplified & Comprehensive API foundation

What is API Connect?
An integrated creation, runtime, management, and security foundation for enterprise grade API’s and Microservices to power modern digital applications

What does API Connect provide?
• Automated, visual and coding options for creating APIs
• Node.js and Java support for creating Microservices
• Integrated enterprise grade clustering, management and security for Node.js and Java
• Access control over API’s, API Plans and API Products*
• Advanced API usage analytics
• Customizable, self service developer portal for publishing APIs
• Policy enforcement, security and control

© Copyright IBM Corporation 2016.
As the number of APIs grows, the need to systematically manage the APIs becomes apparent. API Connect provides a way to improve business value of APIs created.
Designed to work together...

Unleash Enterprise Investments to Disrupt Competitors

Existing Enterprise IT Investments

Exposed as APIs

Self Service Consumed by Developers

To Develop Innovative Apps

Delivering Differentiated Customer Experiences

z/OS Connect Enterprise Edition V2.0

API Connect
Two way example

HTTP POST to https://winmvs41.hursley.ibm.com:12342/mq40

All MQ related information is held in mqzOSConnectService element
- Sensible defaults
- Overridable via HTTP headers, e.g. ibm-mq-gmo-waitInterval
- Builds on the MQ messaging provider. Uses JMS
Example two-way service

- Insurance quote service which takes a request from an MQ queue and sends a response to another queue
- COBOL request

01 QUOTREQT.
  05 NAME     PIC A(20).
  05 ADDR     PIC X(20).
  05 ZIPCODE  PIC 9(9).
  05 BIRTHDAY PIC 9(8) COMP-3.

- COBOL response

01 QUOTRESP.
  05 ACCEPTED PIC A(1).
  05 REASON   PIC X(20).
  05 COST     PIC 9(8).

- So COBOL -> JSON conversion needed using built in tooling
Example two-way service – request

POST https://winmvs41.hursley.ibm.com:12362/lq

Headers

Data

Select one of the options below to include data with the request.

Custom

Enter the data and its corresponding MIME type below.

application/json

```
{
   "OLACB01Operation":{
      "quotreq":{
         "name": "Matt Leming",
         "addr": "IBM US",
         "zipcode": 123456789,
         "birthday": 10271979
      }
   }
}
```
Example two-way service – request on queue

<table>
<thead>
<tr>
<th>iqRequest</th>
<th>Local</th>
<th>Queue management</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>iqResponse</td>
<td>Local</td>
<td>Queue management</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Coded character set identifier: 37

Encoding: 273

Message data:

<table>
<thead>
<tr>
<th>Matt Leming</th>
<th>IBM US</th>
<th>123456789</th>
<th>Éµ</th>
</tr>
</thead>
</table>

Message data bytes:

| 00000 | D4 81 A3 A3 40 D3 85 94--99 95 87 40 40 40 40 40 40 |
|-------|------------------|---------|
| 00010 | 40 40 40 40 C9 C2 D4 40--E4 E2 40 40 40 40 40 40 |
| 00020 | 40 40 40 40 40 40--F1 F2 F3 F4 F5 F6 F7 F8 |
| 00030 | F9 01 02 71 97 9F -- |
Example two-way service – response on queue

<table>
<thead>
<tr>
<th>Coded character set identifier</th>
<th>37</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding</td>
<td>273</td>
</tr>
<tr>
<td>Message data</td>
<td>YGood credit rating 000012345</td>
</tr>
<tr>
<td>Message data bytes</td>
<td>00000</td>
</tr>
<tr>
<td></td>
<td>00010</td>
</tr>
</tbody>
</table>
Example two-way service – response

```
{"OLACB01OperationResponse":{"quoteresp":{"accepted":"Y","reason":"Good credit rating","cost":1234}}}"
```