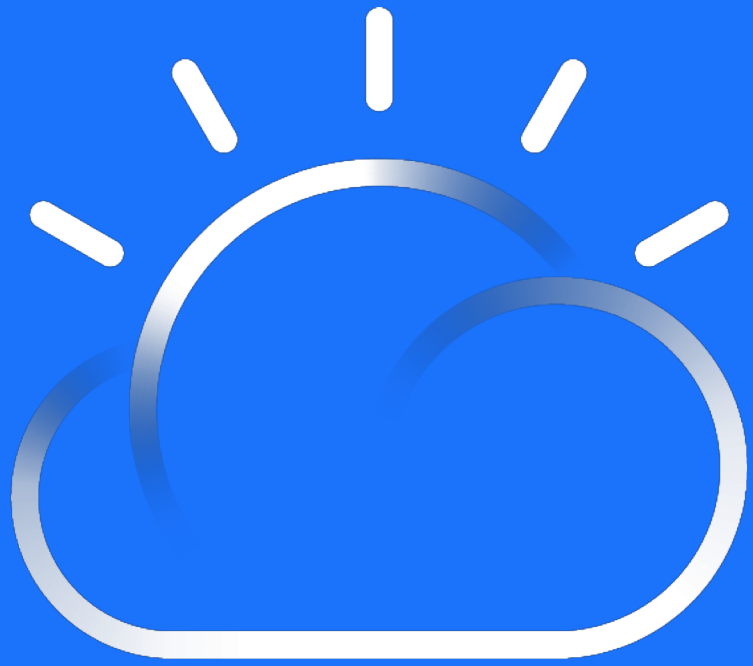
Integration Technical Conference 2019

M05: REST APIs and MQ

Matt Leming
lemingma@uk.ibm.com



IBM **Cloud**

IBM

Agenda

- Why REST?
- Why MQ?
- Putting them together: The MQ messaging REST API
- z/OS Connect EE and MQ

NB: I will not be covering ACE as part of this presentation. However ACE is a very powerful way of exposing REST APIs that interact with MQ and many other systems.

Why REST?

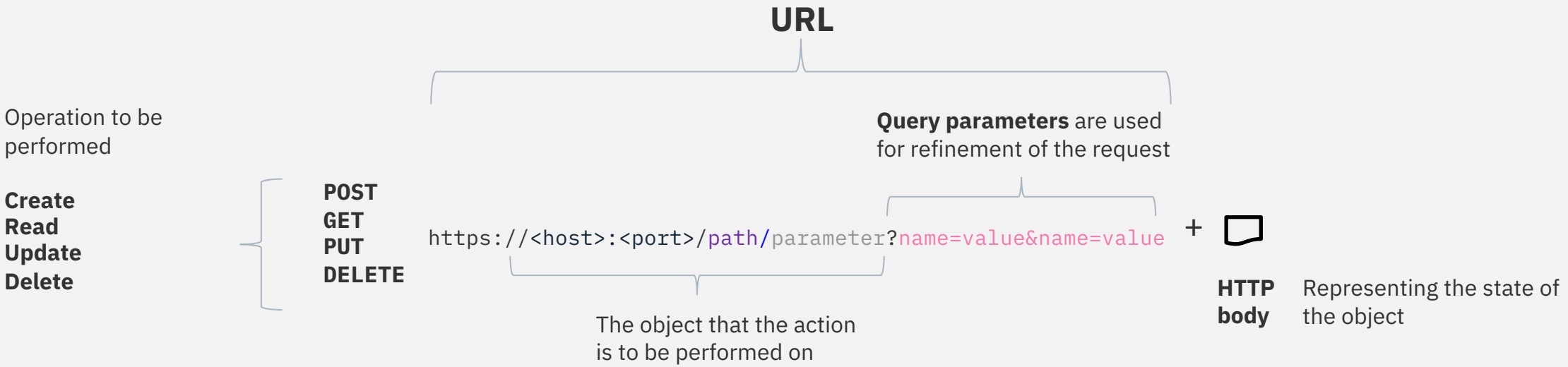
Why REST?

REST was coined in 2000 by Roy Fielding as a result of his work on the various w3 standards that make up the web

REST exposes resources as URLs and operations on those resources using HTTP verbs

REST APIs can be consumed from any programming language and environment for as long as it supports HTTP

REST APIs provide a simple, light-weight, loosely-coupled way to expose synchronous programmatic interfaces to resources



Why MQ?

Why MQ?

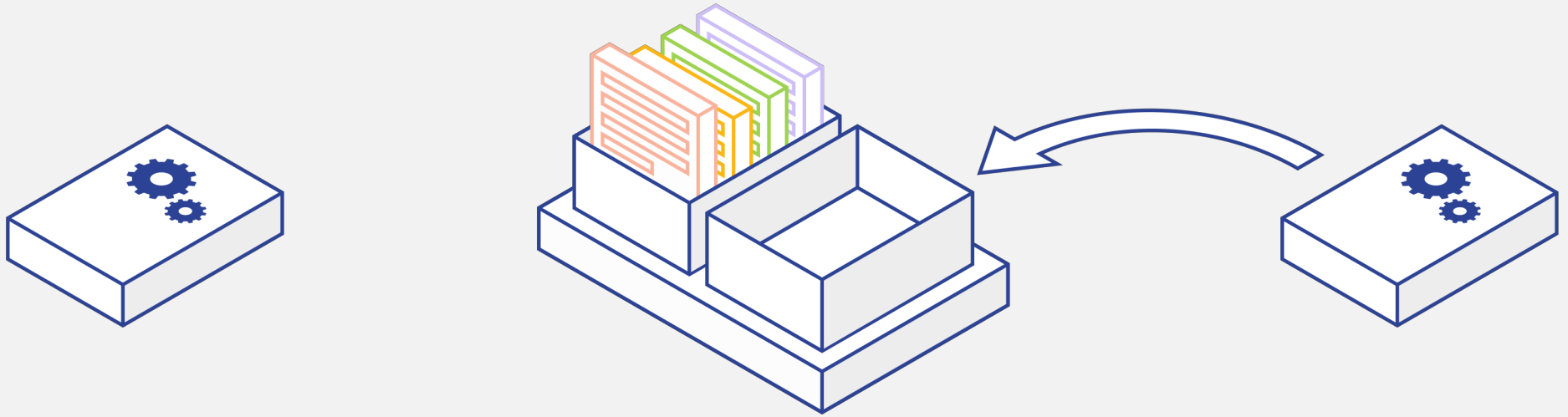
MQ provides a simple set of APIs for allowing applications to communicate, and share data, in an asynchronous, decoupled manner

Applications communicate indirectly via queues in a queue manager which means that they can continue to work together even when they aren't available at the same time

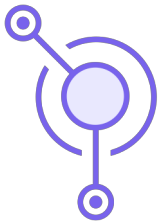
MQ provides the ability to queue messages when they are being produced faster than can be consumed, acting as a "shock absorber" when necessary

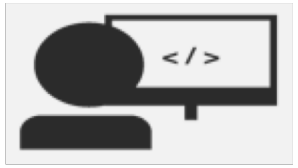
MQ provides simple scaling: just point another instance of the application at the queue

MQ provides advanced capabilities such as transactions, AMS and built in HA/DR



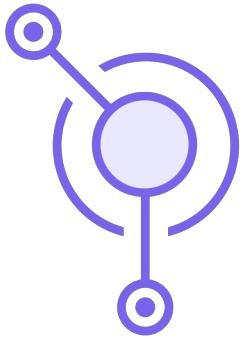
Comparing MQ and REST APIs





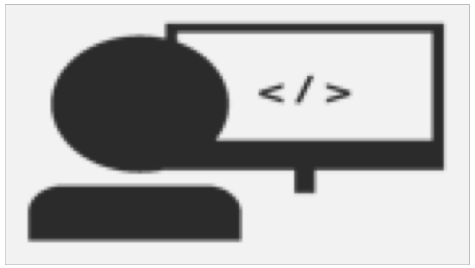
A simple, but powerful, API available in a wide range of programming languages and supported platforms	REST APIs can be called from practically any programming language and platform
Applications communicate **asynchronously** via MQ: they don't need to be available at the same time. Provides request/response, fire-and-forget and pub/sub interactions	Applications communicate **synchronously** via REST: they need to be available at the same time, and expose the actual API! Provides request/response interactions
Scaling with MQ is typically a question of starting more backend applications against a single queue manager. Scaling past that requires technologies such as queue sharing groups or MQ clusters	Scaling of REST typically requires use of an IP sprayer to share work around a group of identical backend applications
MQ provides support for transactions, allowing several messaging operations to be grouped together in a single atomic unit of work, with reliable message delivery	Each and every HTTP operation is unrelated to previous ones. Applications need to code compensating logic, or use techniques such as idempotency, in case of failures
Provides AMS for protecting data end to end without application changes	TLS provided but encryption beyond that point would require changes from both the consumer and provider of the API

Example scenarios



Healthcare provider updates a patient's record after treatment.

This represents a mission critical update from the healthcare provider to the patient's record. The transactional, and reliable delivery capabilities of messaging are desirable here.



Online store exposing read only view of its product catalogue to partners.

This naturally lends itself to being exposed as a RESTful API so it can be accessed from as wide a range of environments as possible. Each interaction is read-only and independent of others.

Putting them together:
The MQ messaging REST API

The MQ messaging REST API

A built in REST API for messaging provided on all platforms

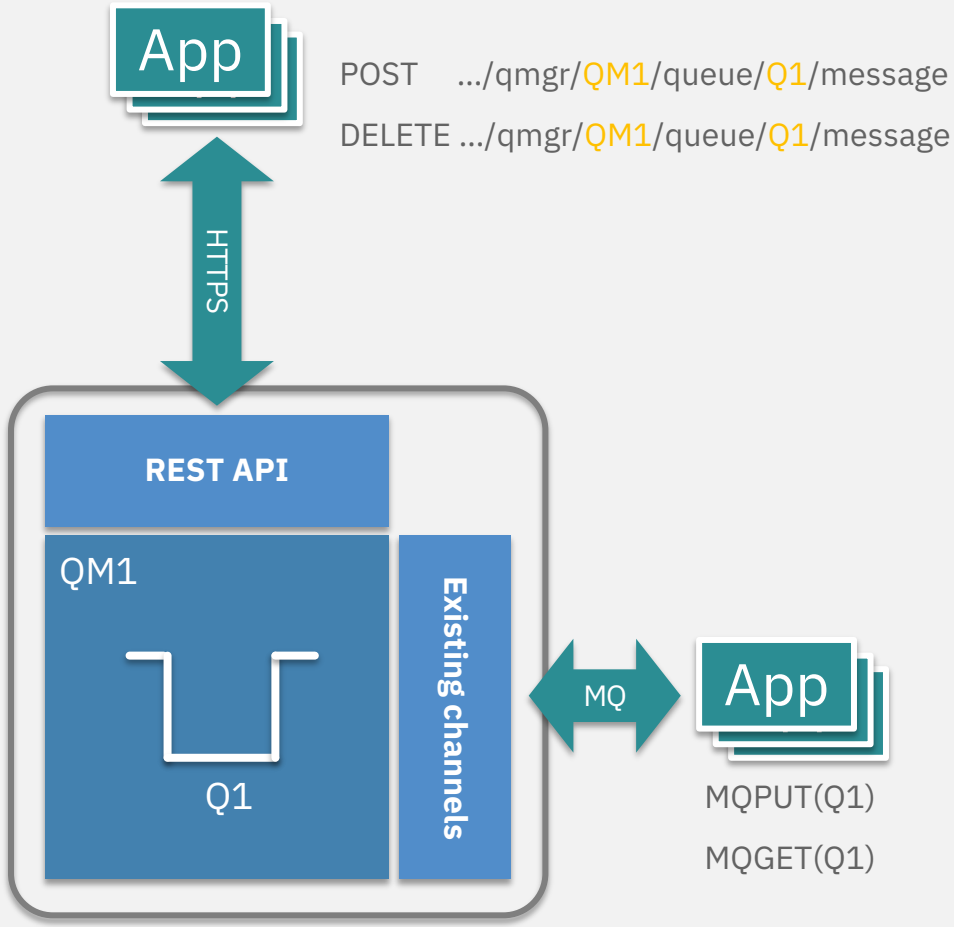
Allows you to interact with existing applications, using existing MQ APIs, from anywhere!

Replaces the older HTTP bridge technology which required a user provided web-server

Originally added in 9.0.4 CD, and now available in 9.1.0 LTS

Currently provides support for sending and receiving text based messages to queues

Will continue to evolve in future CD releases



Sending a message

Send a message to a target queue / queue manager using the HTTP POST verb

Message must have a text based Content-Type HTTP header. Results in a message with a MQMD.Format of MQSTR

Certain MQMD properties exposed as HTTP headers

Authentication performed using caller's user ID. Caller must be a member of the MQWebUser role

```
POST    .../qmgr/QM1/queue/Q1/message
-H "Content-Type: application/xml"
-H "ibm-mq-md-correlationId: 414d5346ef..."
-H "ibm-mq-md-expiry: 60000"
-H "ibm-mq-md-persistence: persistent"
-D "<payload>data</payload>"
```

Receiving a message

Get a message from a target queue / queue manager using the HTTP DELETE verb

Only works if message on queue has a MQMD.Format of MQSTR. Otherwise message will remain on queue and an error is returned

Query parameters can be used to select by correlation and / or message id, as well as specifying a wait interval

Message body is returned as HTTP response body

Certain MQMD fields returned as HTTP headers on response

DELETE .../qmgr/QM1/queue/Q1/message

?correlationId=414d5346ef...

&wait=60000

Response headers:

ibm-mq-md-replyTo:Q2@QMGR1
ibm-mq-md-persistence=persistent
ibm-mq-md-correlationId=414d5346ef...

Response body:

<payload>data</payload>

Performance improved in 9.1.2

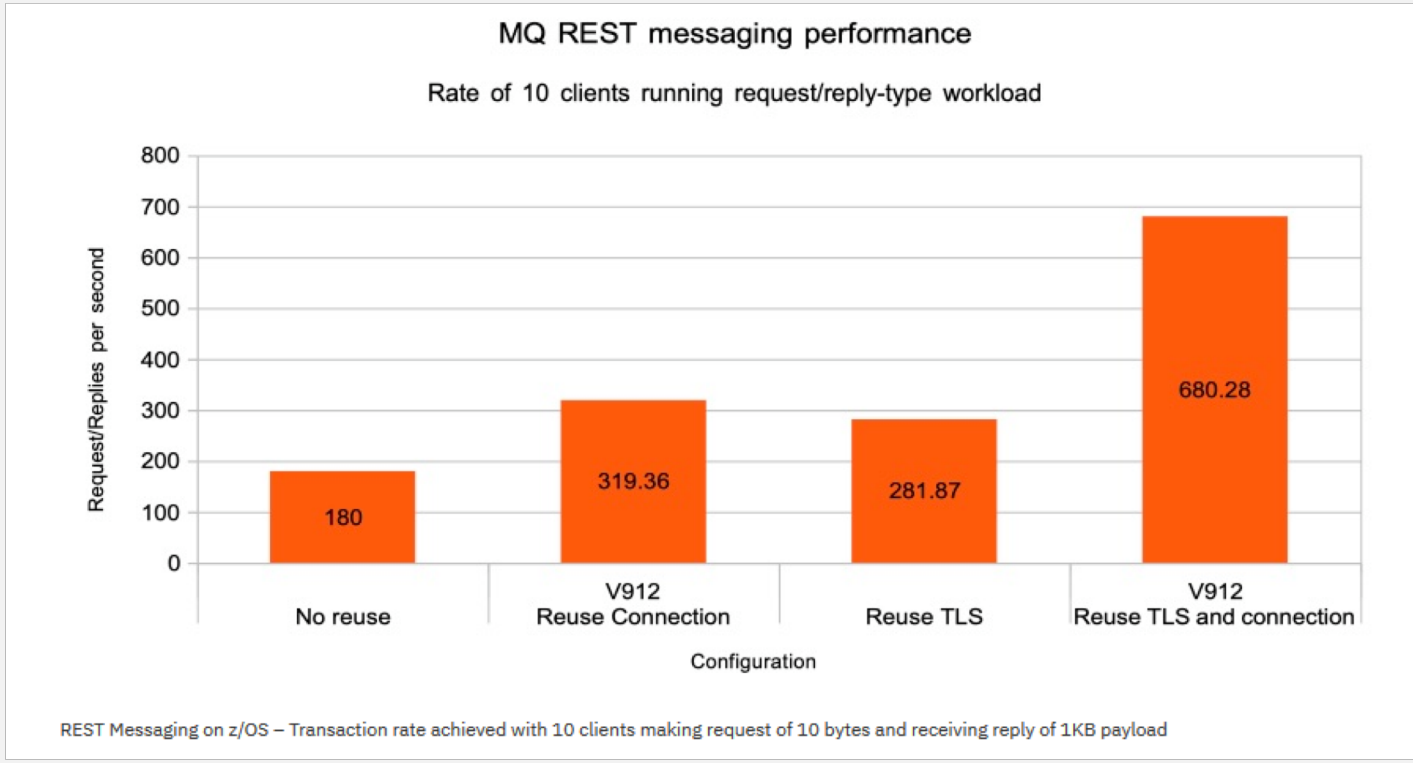
When using the messaging REST API there are two areas which have a big performance impact:

Initial connection to MQ:

In 9.1.2 the messaging REST API now uses a connection pool which will reuse JMS connections where possible

TLS handshake:

Each request to the REST API can end up doing a new TLS handshake which really affects performance. Many clients support the ability to perform an abbreviated handshake which reuse the result of an earlier full handshake



In our z/OS test environment using both of these resulted in 3.75 throughput improvement and 25 times reduction in cost

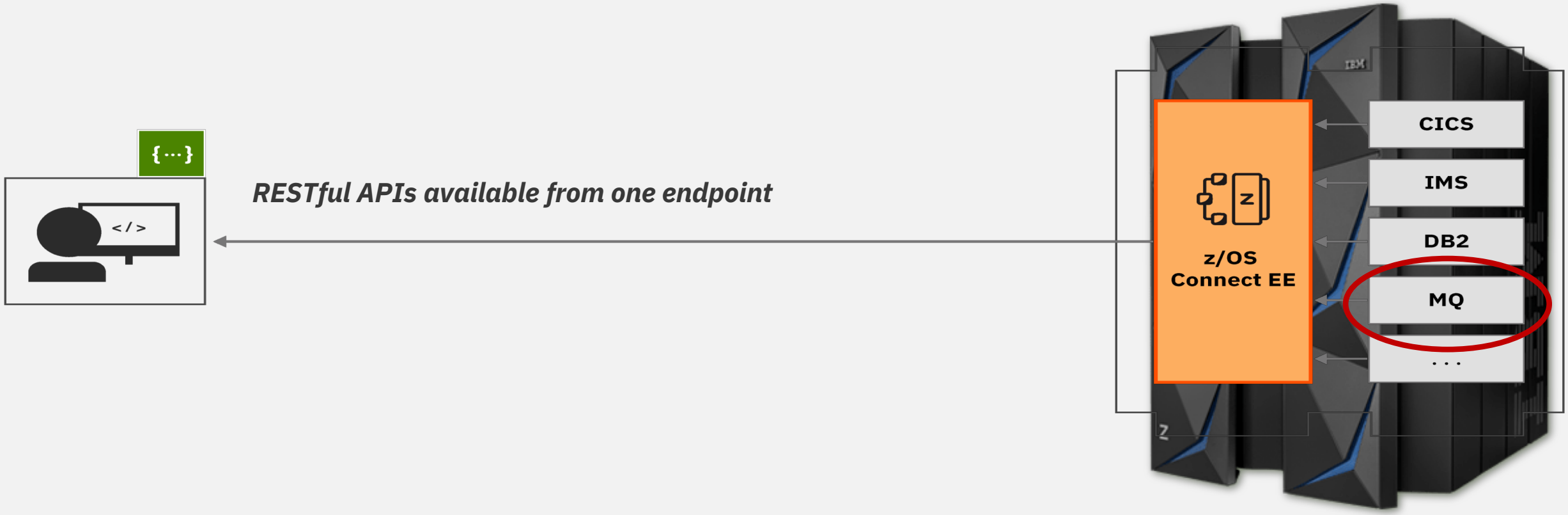
z/OS Connect EE and MQ

Accessing backend systems using HTTP today

- Completely different configuration and management
- Multiple endpoints for developers to call/maintain access to

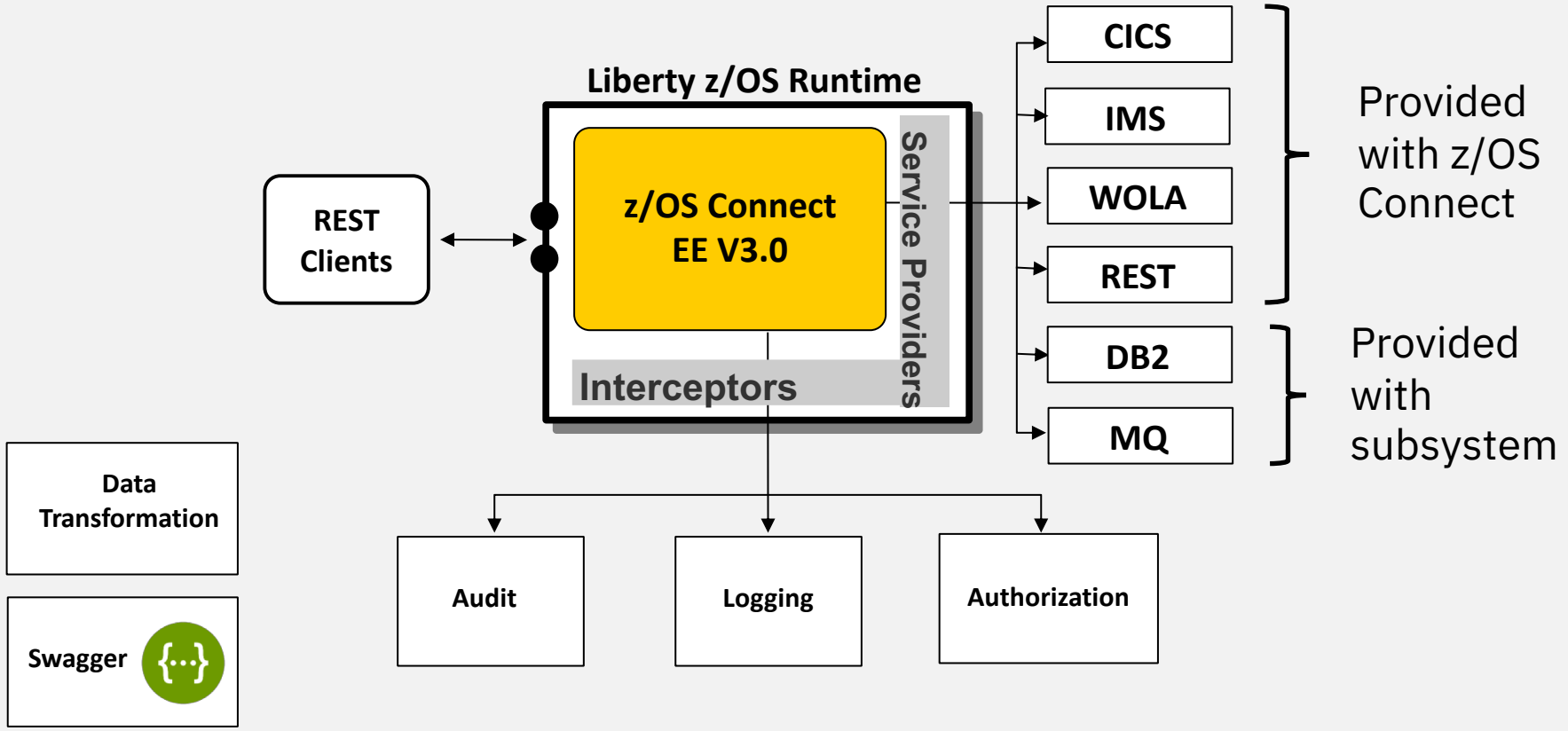
Simplified access to backend systems with z/OS Connect EE

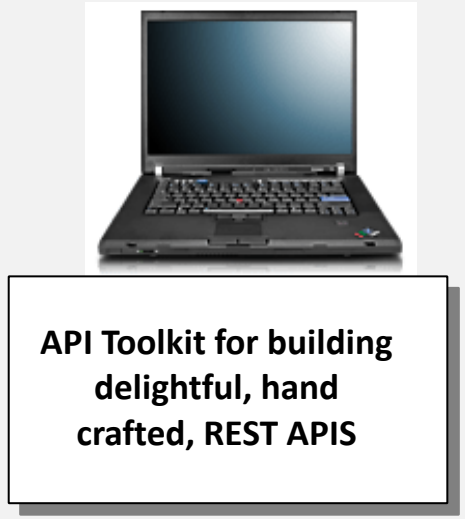
A single entry point is better!



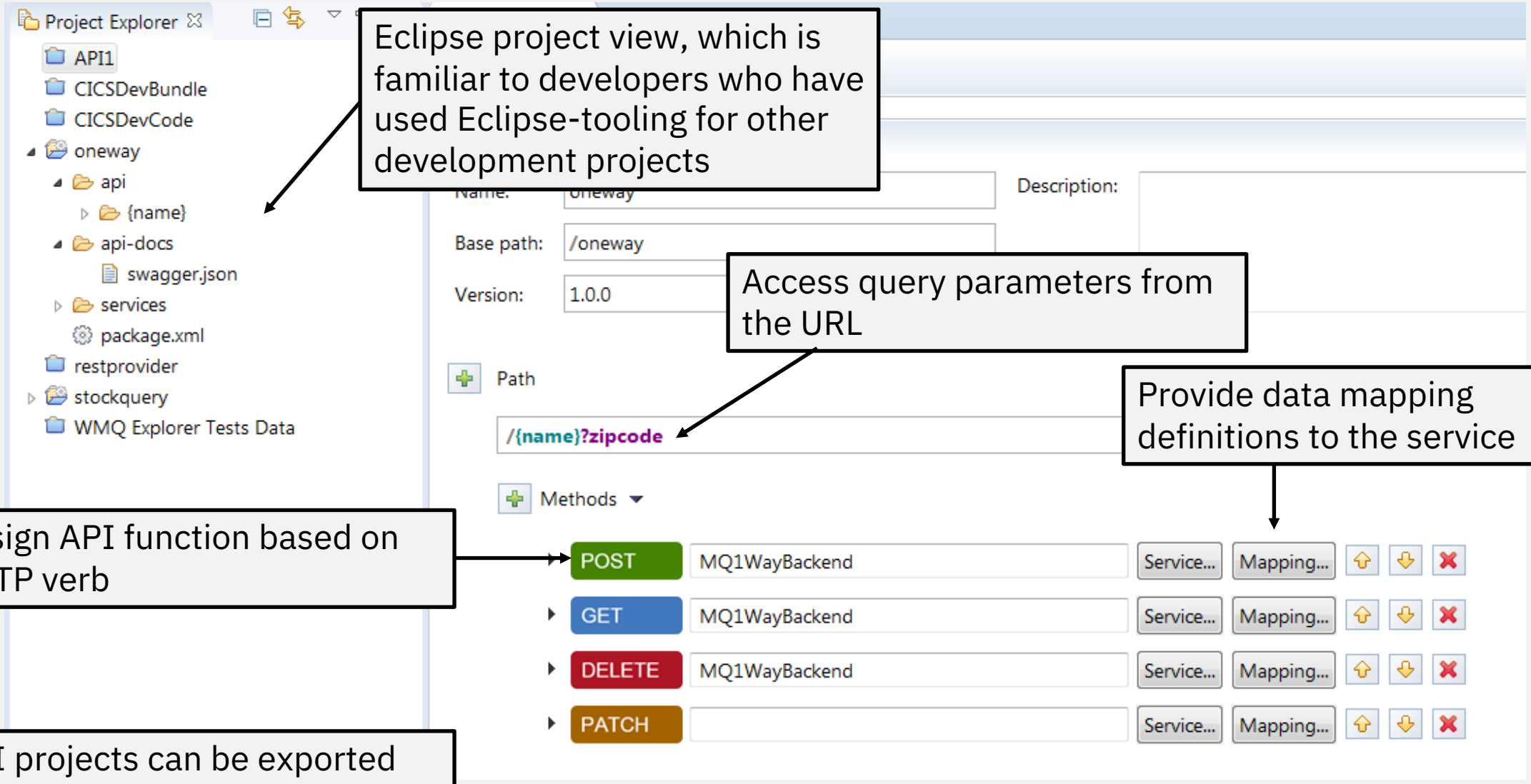
What is z/OS Connect EE?

- A way to expose your existing mainframe assets as REST APIs, without writing any code
- Maximize your return on investment!





API Toolkit



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

Access query parameters from the URL

Provide data mapping definitions to the service

Assign API function based on HTTP verb

API projects can be exported and imported for portability between developers

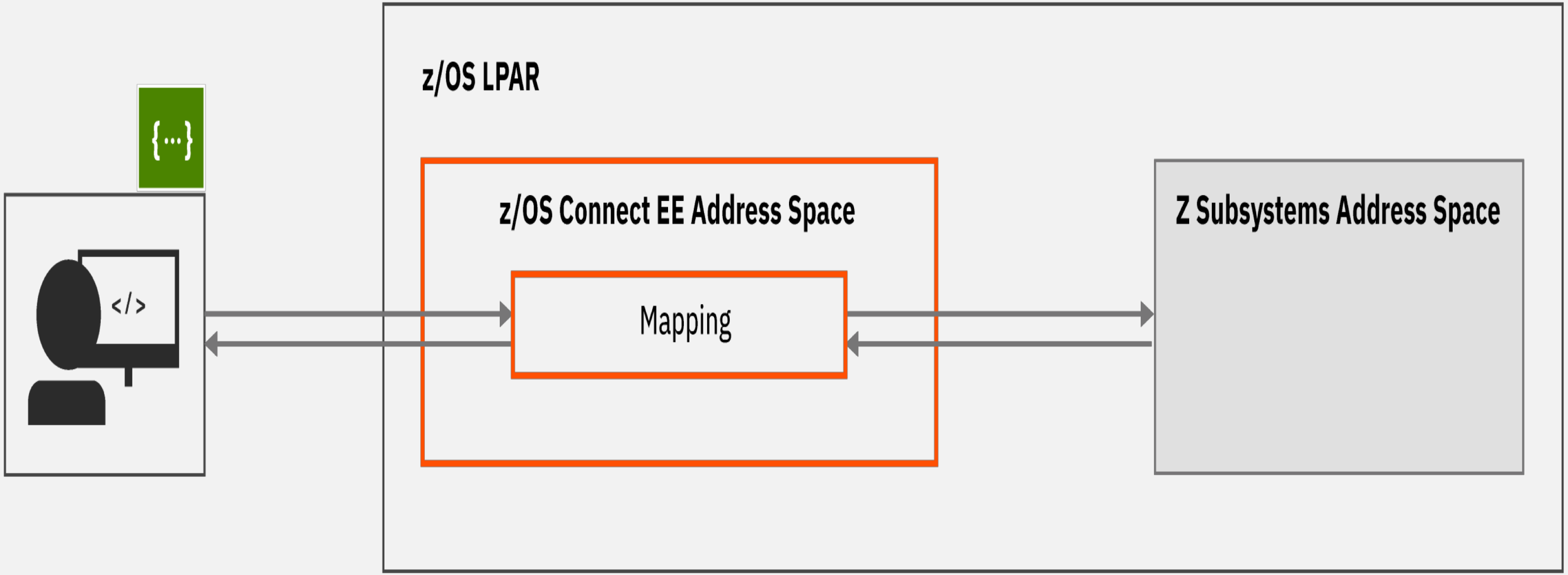
API definitions are created through the tool, which is consistent across backend systems (CICS, IMS, etc.)

A key challenge...

How do I get from the data structures my backend systems understand such as **COBOL**, **PL/I** or **C** to something that make sense for a REST API where data can be encoded in the **URL**, the **JSON** payload and even **HTTP headers**

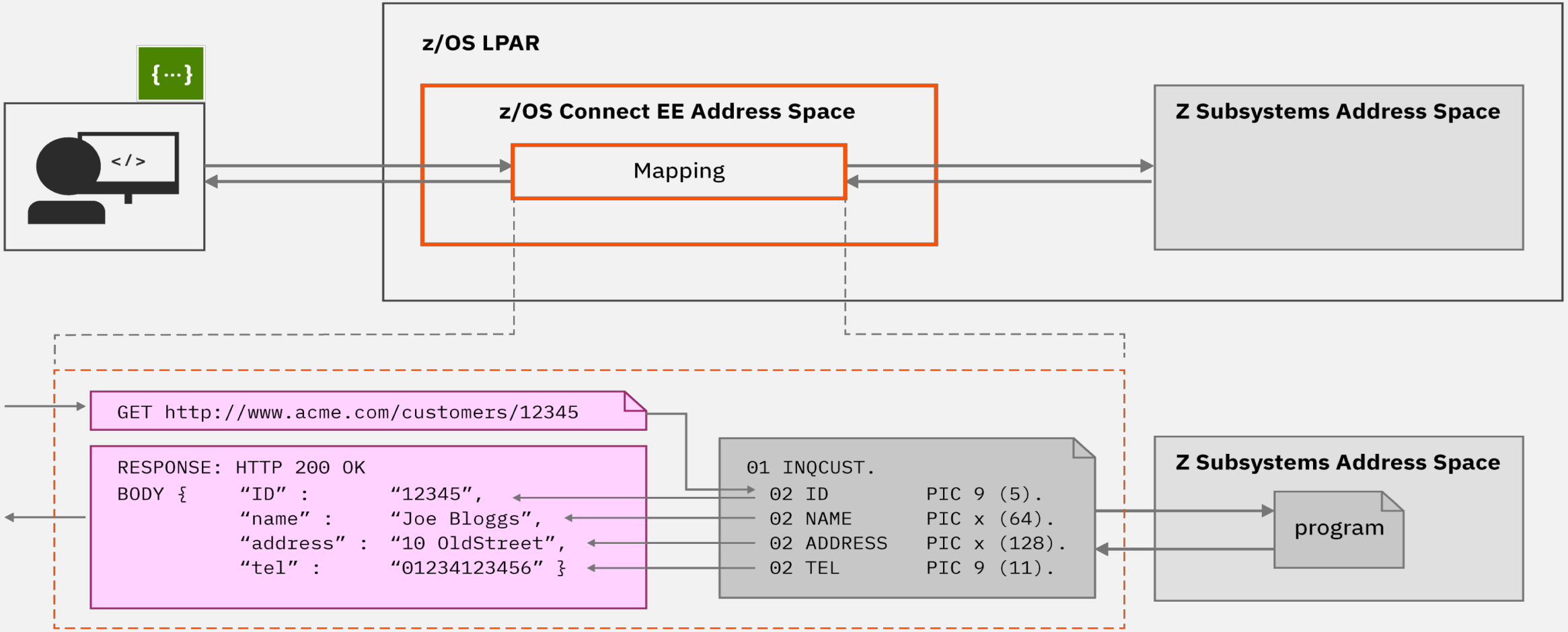
Data mapping

Overview – map REST API payload to backend services



Data mapping

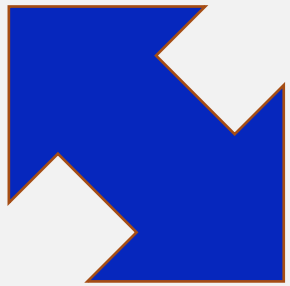
- A closer look
  - Map path and query parameters
  - Granular mapping of data structures



COBOL source vs 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



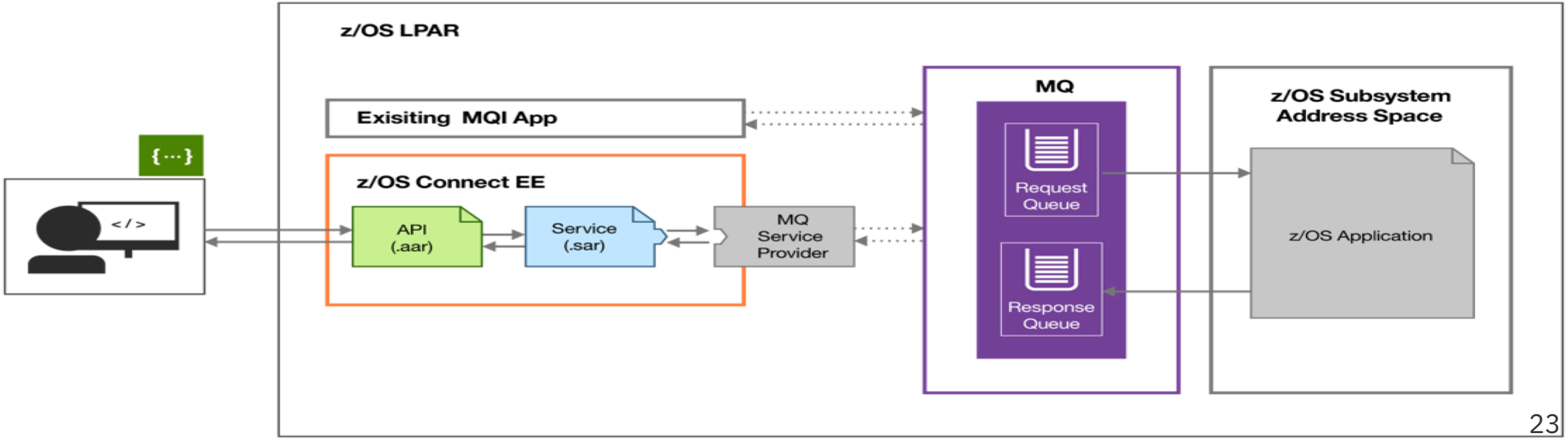
JSON schema equivalent

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

All information is sent as character string

The MQ service provider

- Free of charge z/OS Connect service provider, shipped with MQ
- Allows existing services fronted by MQ to be accessed via a RESTful front end
  - EE V2 (and above) supported
- Developers need no knowledge of MQ
- Can use HTTP headers to override configuration defaults for more control of MQ interaction



Service types

- Each URL in z/OS Connect maps to a service
- With the MQ service provider there are two types of services:
  - One way: uses a single MQ destination

    or

  - Two way: uses a pair of MQ queues, for request reply messaging

One way services

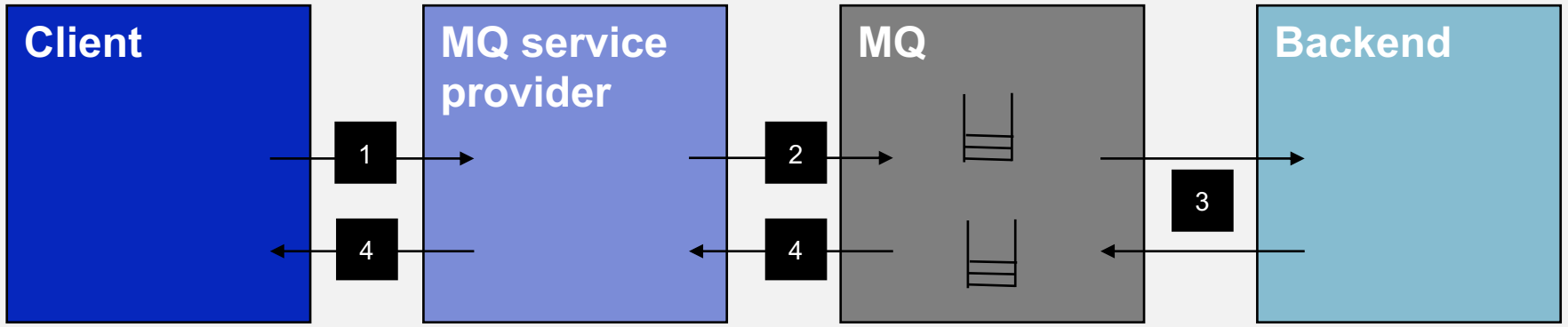
A one way service exposes standard MQ verbs against a single destination

- **HTTP POST** = **MQPUT** (queue and topic)
- **HTTP GET** = **MQGET (browse)** (queue)
- **HTTP DELETE** = **MQGET (destructive)** (queue)

Two way service

Two way service provides request/reply messaging:

1. Client issues HTTP POST with JSON payload
2. MQ service provider in z/OS Connect sends payload (optional transformation) to an MQ queue
3. Back end application processes payload and puts response on reply queue
4. MQ service provider gets response (optional transformation) and sends it to the client as the body of the HTTP POST response



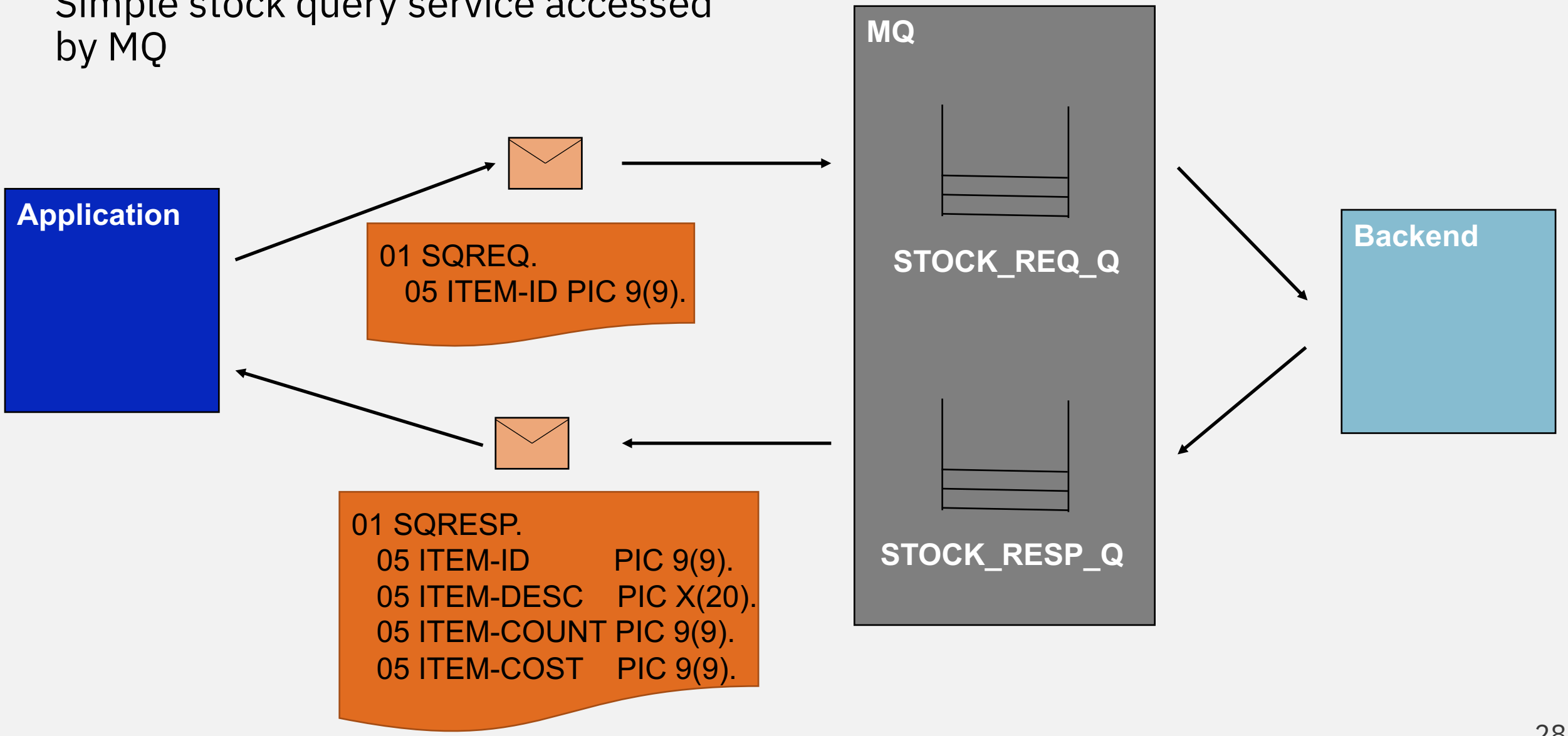
Obtaining the MQ service provider

- MQ service provider for z/OS Connect is:
  - Provided as a USS component in MQ V9.1 LTS, was added originally in 9.0.1 CD
  - E.g. /mqm/V9R1M0/zosconnect/v2.0/lib
- Also available on Fix Central (if you are on V8 or V9)
- Only contains the MQ service provider function:
  - You need to provide z/OS Connect EE
  - You also need to provide the MQ resource adapter: use the version from your MQ installation

- Packaged as a Liberty feature:
  - Properties file
  - JAR file
  - Liberty feature manifest (*.mf) file

An example of using the MQ service provider

Simple stock query service accessed by MQ

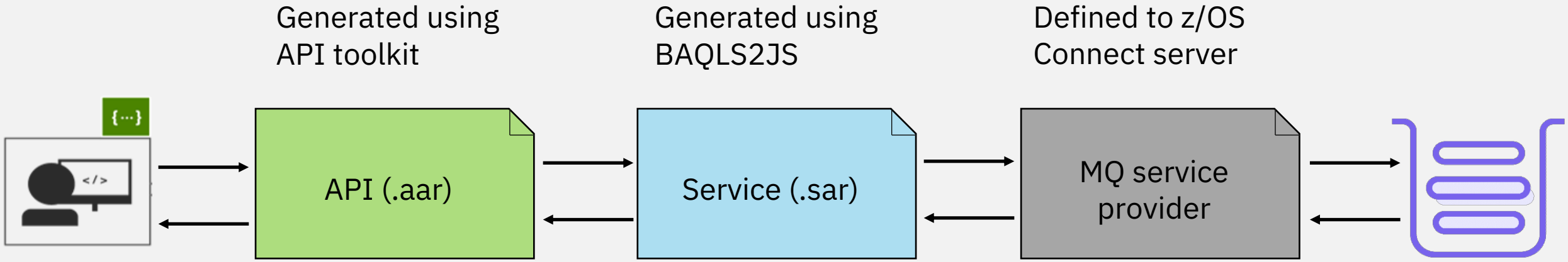


Goal

- Extend stock query service so that it can be accessed using a RESTful API
- Want request URL to contain the item id => common in REST APIs
- Can’t break existing users of the service

Steps

1. Generate the SAR file
2. Generate the RESTful API using the API toolkit
3. Deploy API to z/OS Connect server
4. Install the MQ service provider
5. Define MQ resources
6. Define the service
7. Sanity checks
8. Try it out!



Step 1: generate the SAR file: BAQLS2JS

```
//GNZTUSAR JOB (0),MSGCLASS=X,CLASS=A,NOTIFY=&SYSUID,REGION=500M
//ASSIST   EXEC PGM=BPXBATCH
//STDPARM  DD *
PGM  /ZOS202/usr/lpp/IBM/zosconnect/v3r0/bin/baqls2js
LOGFILE=/u/mleming/zCEE/GNZTUSAR/logs/GNZTUSAR.log
LANG=COBOL
PDSLIB=MLEMING.COBOL
REQMEM=SQREQ
RESPMEM=SQRESP
JSON-SCHEMA-REQUEST=/u/mleming/zCEE/GNZTUSAR/schema/SQ_request.json
JSON-SCHEMA-RESPONSE=/u/mleming/zCEE/GNZTUSAR/schema/SQ_response.json
WSBIND=/u/mleming/zCEE/GNZTUSAR/wsbind/SQ.wsbind
PGMNAME=SQ
PGMINT=COMMAREA
SERVICE-ARCHIVE=/u/mleming/zCEE/GNZTUSAR/StockQuote.sar
SERVICE-NAME=SQ
MAPPING-LEVEL=4.3
/*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD *
JAVA_HOME=/java/java80_bit64_sr5_fp20/J8.0_64
//
```

Request and response copybooks

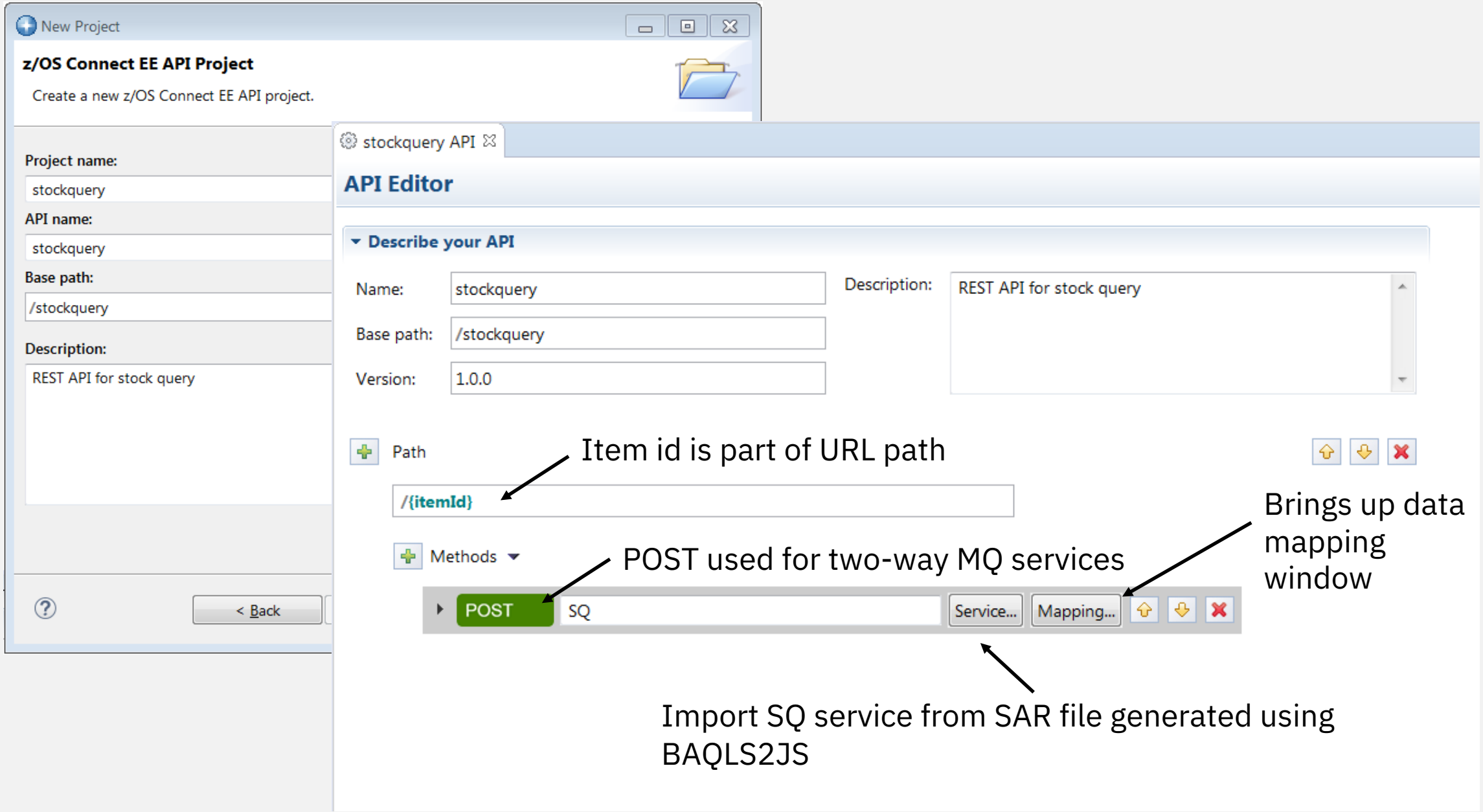
Request and response schemas

Used for data mapping

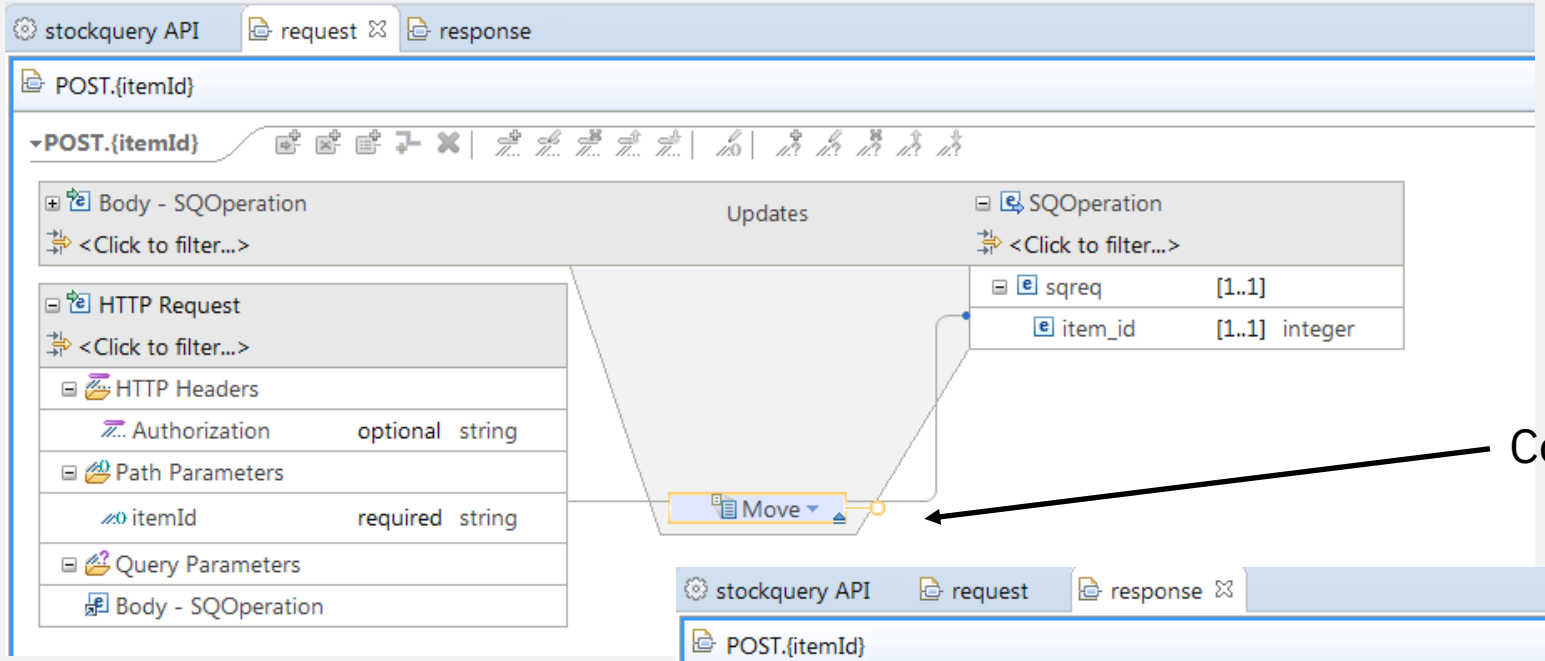
SAR file, which contains a copy of the request and response schemas

This will be important later when linking the API to the actual service instance in z/OS Connect

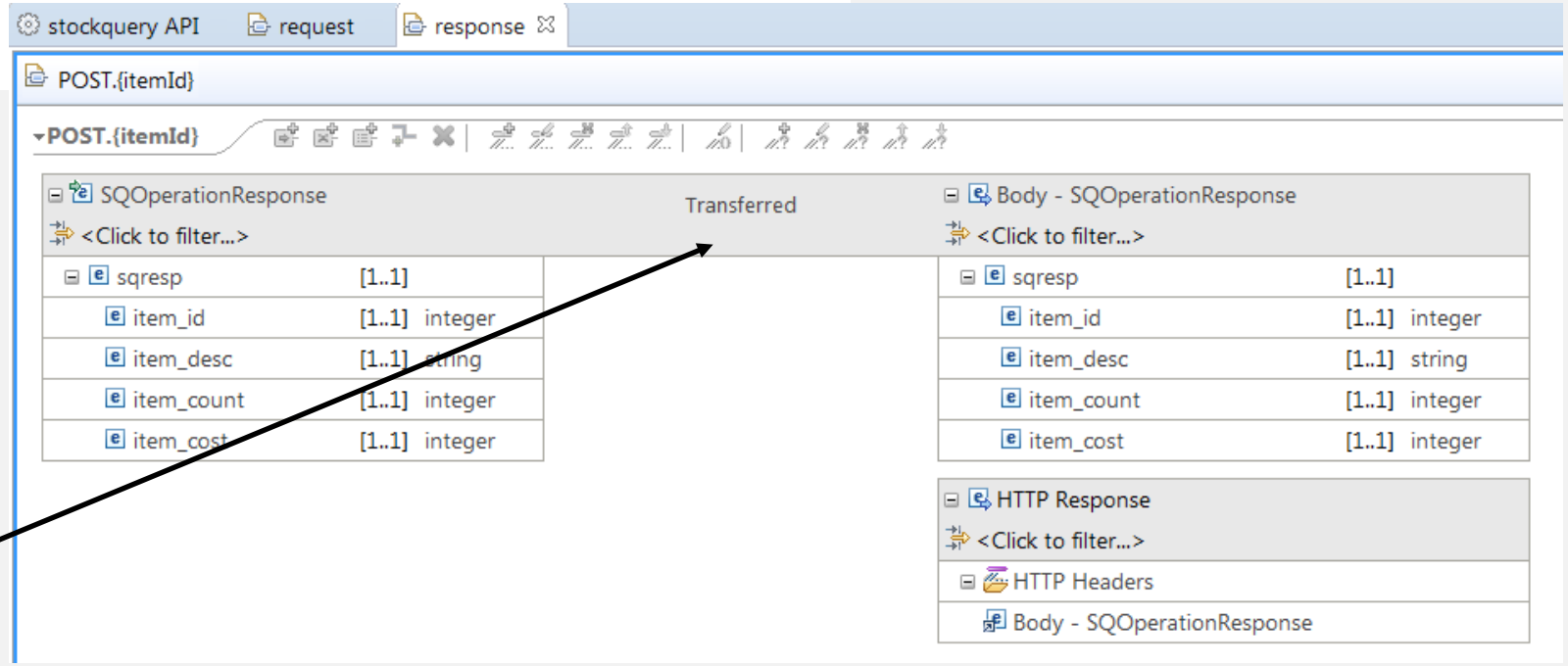
Step 2: generate the RESTful API using the API Toolkit



Step 2: generate the RESTful API using the API Toolkit

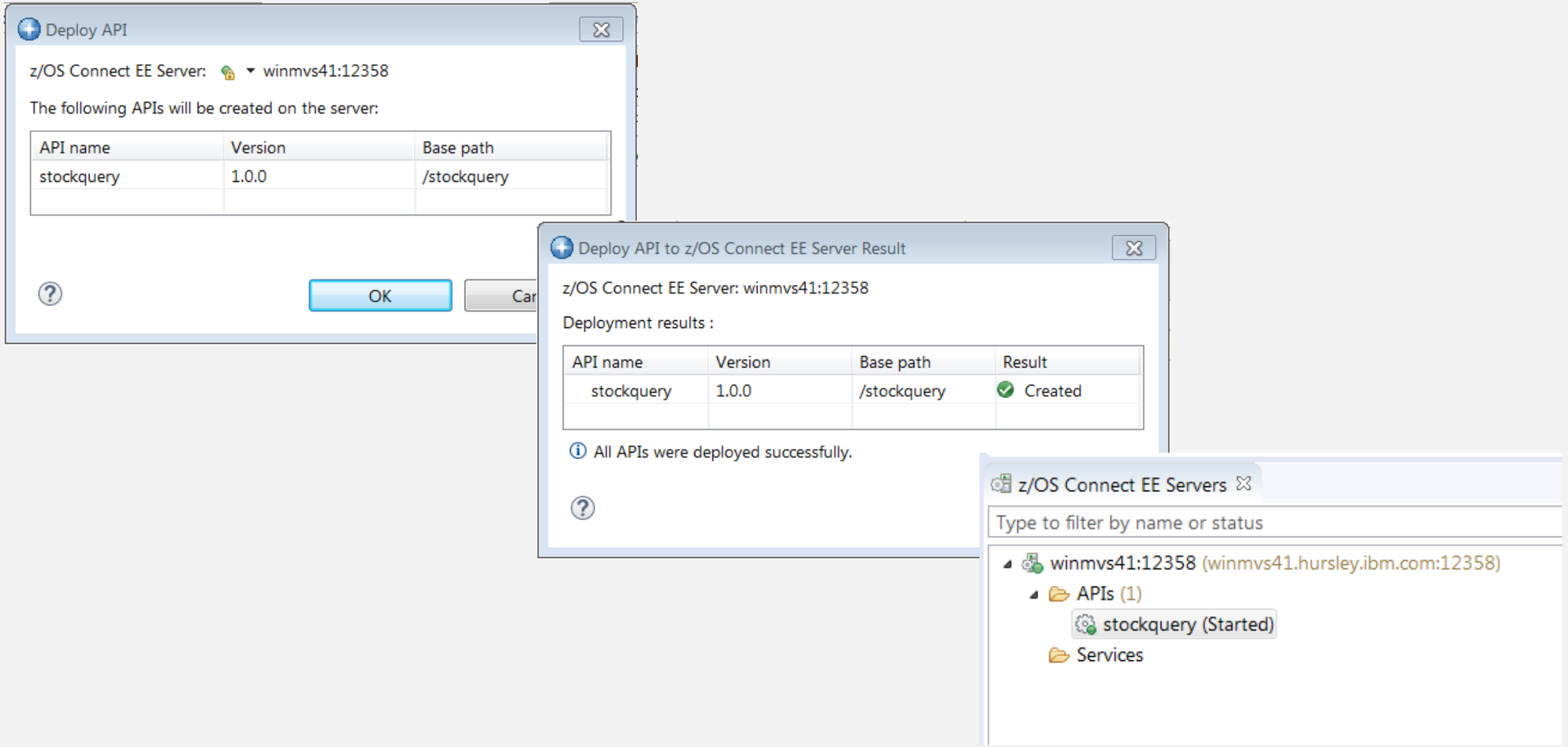


Copy the item id from the URL

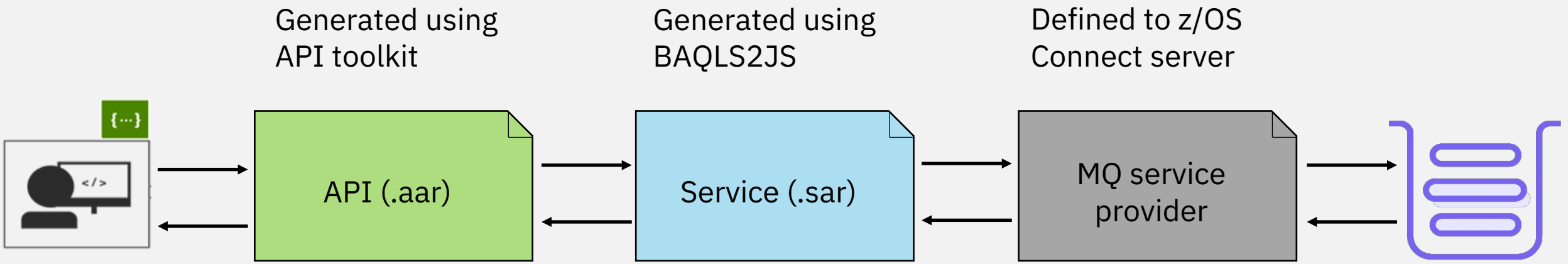


Response is mapped back without change

Step 3: deploy API to z/OS Connect server



Recap



Step 2: generate the RESTful API using the API Toolkit
+
Step 3: deploy API to z/OS Connect server

Step 1: generate the SAR file

Next...

Step 4: install the MQ service provider

```
<server>
 <featureManager>
  <feature>zosconnect:zosconnect-2.0</feature>
  <feature>appSecurity-2.0</feature>
  <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=“ /mqm/V9R1M0/java/lib/jca/wmq.jmsra.rar "/>

 <wmqJmsClient nativeLibraryPath="/mqm/V9R1M0/java/lib/"/>
```

MQ service provider

MQ Client

MQ resource adapter

MQ native libraries. Required for bindings connections

This only needs to be done once regardless of the number of MQ services deployed

Step 5: define MQ resources

```
<jmsConnectionFactory id="mq21CF" jndiName="jms/mq21CF" connectionManagerRef="cm1">
  <properties.wmqJms transportType="BINDINGS" queueManager="MQ21"/>
 </jmsConnectionFactory>

 <connectionManager id="cm1" maxPoolSize="5"/>

 <jmsQueue id="sqReqQ" jndiName="jms/sqReqQ">
  <properties.wmqJms baseQueueName="STOCK_REQ_Q"/>
  <properties.wmqJms targetClient = "MQ" CCSID="37"/>
 </jmsQueue>

 <jmsQueue id="sqRespQ" jndiName="jms/sqRespQ">
  <properties.wmqJms baseQueueName="STOCK_RESP_Q"/>
  <properties.wmqJms targetClient = "MQ" CCSID="37"/>
 </jmsQueue>
```

Defines connection to queue manager

Used for connection pooling

Request queue

Response queue

Each service will need its own queue/topic definitions
Connection factories can be reused between services unless you need to tweak properties

Step 6: define the service

```
<zosconnect_zosConnectService id="stockQuoteService" invokeURI="/stockQuoteServiceNoAPI"
                serviceName="SQ" serviceRef="stockQuoteService_mq"
                dataXformRef="xformJSON2Byte"/>

<mqzosconnect_mqzOSConnectService id="stockQuoteService_mq" connectionFactory="jms/mq21CF"
                destination="jms/sqReqQ" replyDestination="jms/sqRespQ"
                replySelection="msgIDToCorrelID" waitInterval="30000"/>

<zosconnect_zosConnectDataXform id="xformJSON2Byte"
                bindFileLoc="${XFORM_ROOT}/bindfiles" bindFileSuffix=".wsbind"
                requestSchemaLoc="${XFORM_ROOT}/json" requestSchemaSuffix=".json"
                responseSchemaLoc="${XFORM_ROOT}/json" responseSchemaSuffix=".json"/>
```

The API generated in the toolkit locates the service using the serviceName

Defines a service to z/OS Connect

The destination, replyDestination and connectionFactory attributes map to the MQ resources we defined earlier

Defines an MQ service instance

Where to locate the data transformation

Each service using the MQ service provider will need both of the top two elements. The bottom element can be used for multiple services.

Step 7: sanity checks

- First check there are no errors in the z/OS Connect Server logs
- Check that the API exists

```
curl -k https://winmvs41.hursley.ibm.com:12358/zosConnect/apis

{"apis":[{"name":"stockquery","version":"1.0.0",
"description":"REST API for stock query",
"adminUrl":"https://winmvs41.hursley.ibm.com:12358/zosConnect/apis/stockquery"}]}
```

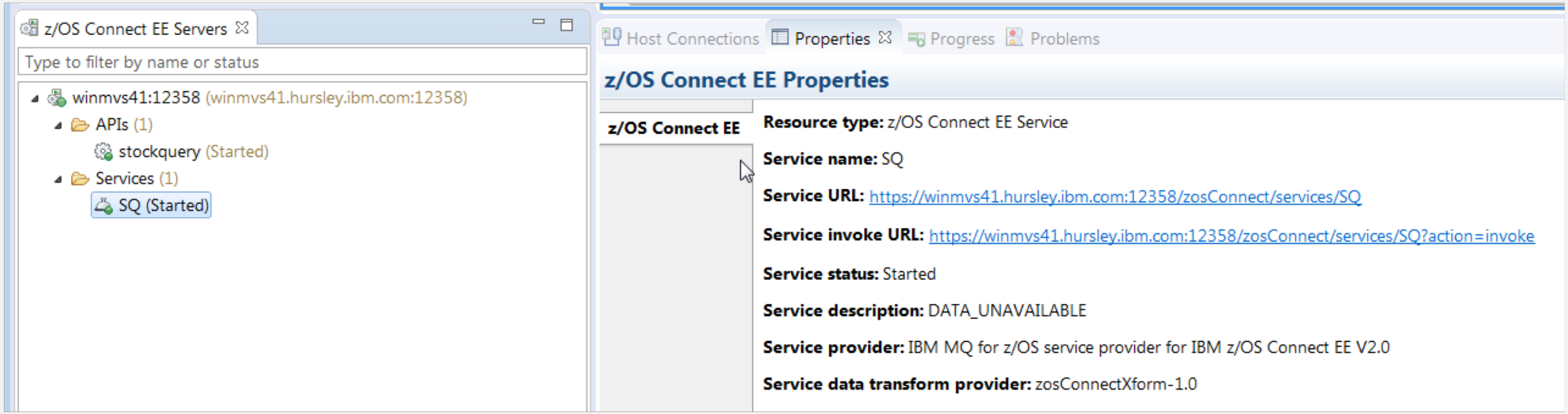
- Check that the service exists

```
curl -k https://winmvs41.hursley.ibm.com:12358/zosConnect/services

zosConnectServices":[{"ServiceName":"SQ",
"ServiceDescription":"DATA_UNAVAILABLE",
"ServiceProvider":"IBM MQ for z/OS service provider for IBM z/OS Connect EE V2.0",
"ServiceURL":"https://winmvs41.hursley.ibm.com:12358/zosConnect/services/SQ"}]}
```

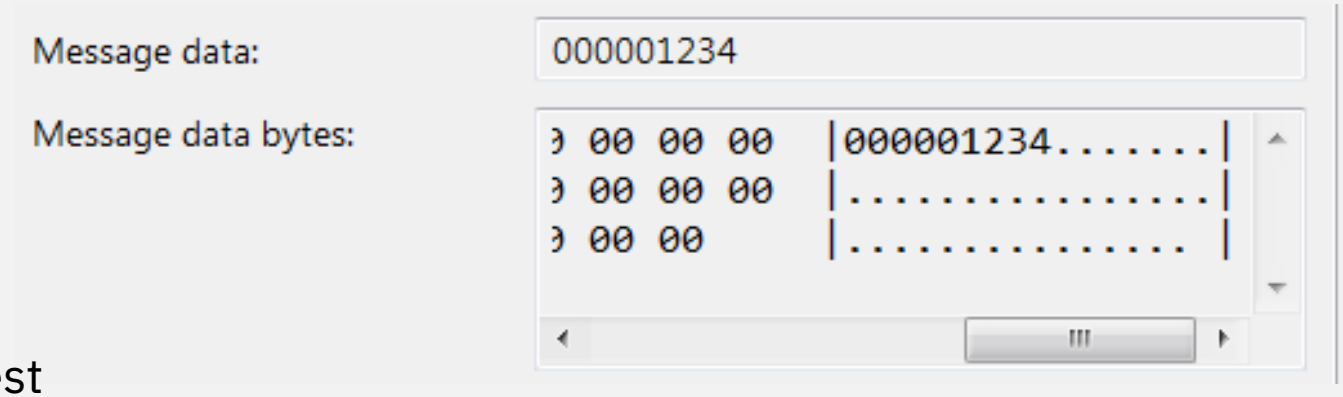
Step 7: sanity checks

Or use the API editor



Step 8: try it out!

curl -i -k -X POST -H "Content-Type: application/json" https://winmvs41:12358/stockquery/1234



Back-end waiting for request

Request message

Queue name	Queue type	QSG disposit...	Open input count	Open output count	Current queue depth
STOCK_REQ_Q	Local	Queue man...	1	0	1
STOCK_RESP_Q	Local	Queue man...	1	0	0

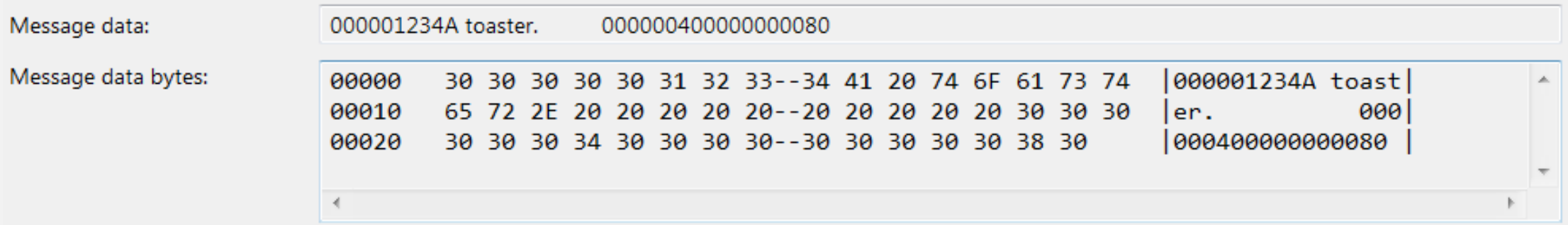
MQ service provider waiting for response

Step 8: try it out!

Back end receives request message and generates response

Queue name	Queue type	QSG disposit...	Open input count	Open output count	Current queue depth
STOCK_REQ_Q	Local	Queue man...	0	0	0
STOCK_RESP_Q	Local	Queue man...	1	0	1

Response message



Step 8: try it out!

```
curl -i -k -X POST -H "Content-Type: application/json"
https://winmvs41.hursley.ibm.com:12358/stockquery/1234

{
  "SQOperationResponse": {
    "sqresp": {
      "item_count":400,
      "item_cost":80,
      "item_desc":"A toaster.",
      "item_id":1234
    }
  }
}
```

Or you could use the built in swagger support in the API toolkit

Coming soon...

IBM MQ for z/OS 9.1.1 announcement

Statement of general direction

IBM intends to deliver the following new capabilities within future Continuous Delivery releases:

• The ability to apply and remove Advanced Message Security policies transparently between Advanced Message Security (AMS) and non-AMS enabled queue managers

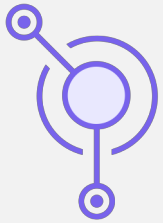
• An enhanced z/OS Connect Enterprise Edition service provider for MQ to add support for the Build and API Toolkits, and also automated service deployment

• zHyperwrite support for MQ log files

Statements by IBM regarding its plans, directions, and intent are subject to change or withdrawal without notice at the sole discretion of IBM. Information regarding potential future products is intended to outline general product direction and should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for IBM products remain at the sole discretion of IBM.

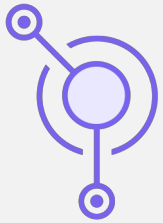
https://www-01.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_ca/0/897/ENUS218-510/index.html&lang=en&request_locale=en#sodx

Comparing IBM MQ for z/OS and z/OS Connect EE



Powerful APIs (MQI, JMS, XMS, MQLight) accessible from a wide variety of supported platforms. Built-in REST APIs for messaging and administration operations	Enable z/OS assets to create or consume REST APIs tailored to application needs. Simple to use Eclipse-based API composition tooling
Asynchronous transactional messaging able to handle millions of messages every second. Assured once and once only delivery	Synchronous requests able to trigger thousands of backend transactions every second. Uses stateless and non-transactional HTTP protocol
Highly available services and data with shared queues and automatic recovery	Highly available service configurations using multiple servers or LPARs
Developers use a range of procedural and object oriented languages and frameworks. MQ service provider available for zCEE	Developers use widely available skillsets (node.js, JSON, REST etc.). Data is then transformed appropriate to the service provider

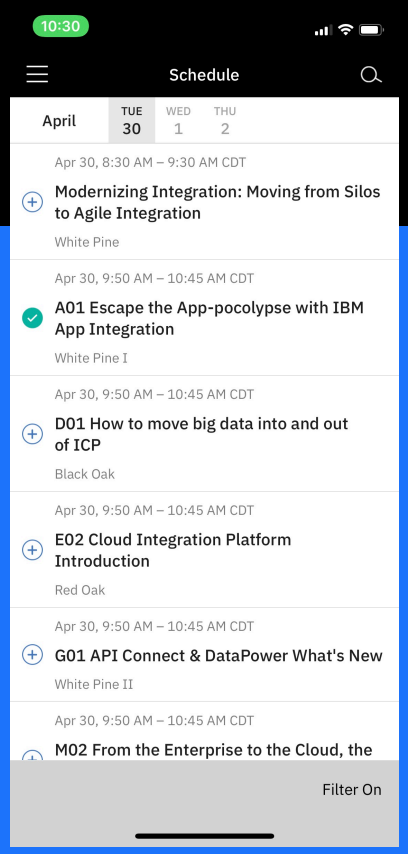
Comparing the MQ messaging REST API and z/OS Connect EE

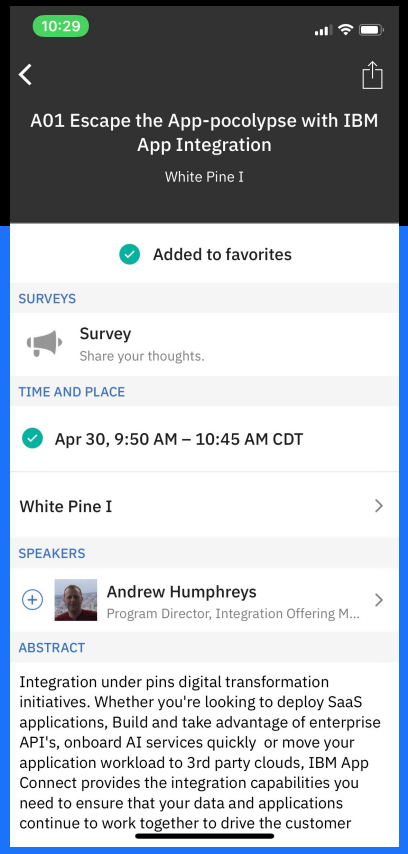


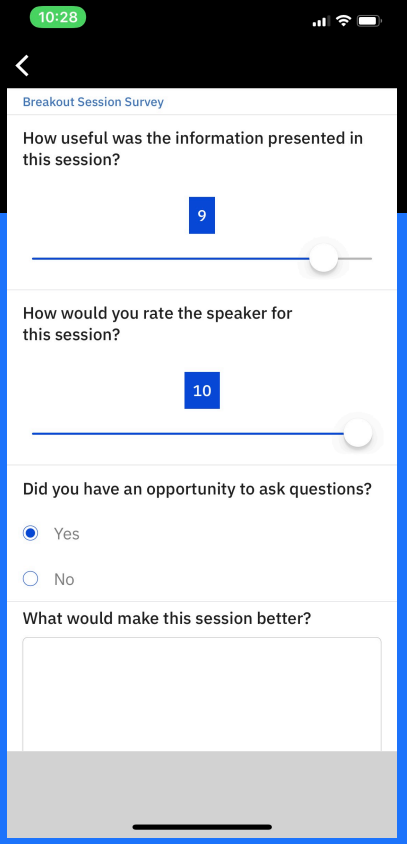
A REST API for messaging, very obvious that MQ is being used	Allows customizable truly RESTful APIS to be built that hide the underlying z/OS subsystem, regardless of what it is
Currently limited function: allows text messages to be sent and received to/from queues. Will be extended in future CD releases	Both two-way and one-way services supported. One-way services support browse, and can publish to topics
No data transformation, but does allow MQ codepage conversion	Built in support for conversion to/from JSON to COBOL, PL/I and C
Tightly integrated with MQ, minimal set up required other than for security	Requires service to be defined using SAR, and API Toolkit for API customization. Needs configuration to enable connectivity to MQ

Summary

- Why REST?
- Why MQ?
- Putting them together: The MQ messaging REST API
- z/OS Connect EE and MQ







Don’t forget to fill out the survey!

Select your session, select survey, rate the session and submit!

Thank You

Matt Leming
lemingma@uk.ibm.com