Application of Microservice Architecture to B2B Processes

(IBM Watson Customer Engagement)

by

Arpit Jain

(2015047)

Supervisor(s):

External
Mr. Atul A. Gohad
(IBM ISL, Bangalore)

Internal
Dr. Aparajita Ojha
(PDPM IIITDM Jabalpur)

Computer Science and Engineering

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING JABALPUR

(27th July 2018 – 8th August 2018)

Introduction

The International Business Machines Corporation (IBM) is an American multinational technology company headquartered in Armonk, New York, United States, with operations in over 170 countries. IBM manufactures and markets computer hardware, middleware and software, and provides hosting and consulting services in areas ranging from mainframe computers to nanotechnology.

IBM aims to bring Businesses closer and smarter than ever with the help of their state of the art enterprise software product called B2B Sterling Integrator. IBM B2B Integrator helps companies integrate complex B2B (Business to Business) / EDI (Electronic Data Exchange) processes with their partner communities. IBM aims to transform the B2B Sterling product into Microservice architecture.

Brief Overview

During the duration of my last report, I deployed B2Bi-API WAR (Web Application Archive) file on the SI platform. Building services and adapters requires a solid understanding of the sterling integrator system. I got myself acquainted with the following list of Resources which includes the prerequisite knowledge and experience necessary for successfully creating custom services and adapters (In my case, Rest API client service):

- Java (J2SE) programming knowledge
- General operational and architectural knowledge of the SI system.
- Eclipse IDE programming experience.
- XML Understanding specifically for writing custom Business process definitions.
- API endpoint usage and general understanding of JSON/XML data resources.
- Understanding of RESTful services.
- Multi-threaded programming experience in Java
- Exception Handling in Java
- Ability to write custom APIs

In this Report, I implemented the GET Rest operation for the Rest Client Service.

Report on the Present Investigation

(Progress during this 15-days period)

Creating a REST API Client Service for the B2Bi APIs

Abstract

On a very basic level, to write a custom service for SI platform, there are two ways to deploy a service.

1. Create a pre-packaged Jar Service package consisting of all the service resources and install it later using InstallService shell script file.
2. Manually, copy the required service definition java files and Business Processes to their respective product asset folders. Now, building the Assets and finally install the patch.

The service structure for our REST API client is described as shown –

The **genericrest.jar (The service package file)** file contains:

```
META-INF
```

```
    MANIFEST.MF
testservice
    jars
      testservice
        1_0
          genericrest.jar
    bpml
      genericrest.bpml.in
    servicedefs
      genericrest.xml
genericrest.java
RestRequestHandler.java
RestAPIClientGET.java
genericrestserver.java
genericrestserverImpl.java
serviceinstances.xml
```

These files are explained in the next section.

Service definition (Deployment Descriptor)

The deployment descriptor is in the servicedefs subdirectory. It defines the name of the service, it has a description and a label. The description and label are just references to the language property file that is in the files/properties/lang/en/ directory. This allows you to have language property files for different languages. The deployment descriptor also has the name of the starting Java class in your custom code.

Service definition basically defines the service/adapter. In our case, the genericrest.xml file contains the definition for the service. It has important information like the implementation class of the service and the type of service. In this example, type =" Basic" defines it as a service, type =" Adapter" defines it as an adapter.

```
<?xml version="1.0" encoding="UTF-8"?>
<SERVICES>
<SERVICE name="genericrest"
 description="genericrest.description"
 label="genericrest.label"
 implementationType="CLASS"
 JNDIName="com.sterlingcommerce.woodstock.services.genericrest.genericrest"
 type="Adapter"
 adapterType="STATELESS"
 adapterClass="com.sterlingcommerce.woodstock.services.genericrest.genericrestserverImpl"
 version="1.0"
 SystemService="NO">

 <VARS type="global">
 </VARS>

 <VARS type="wfd">
 <GROUP title="Service Parameters" instructions="Enter the URL of the API endpoint. And select the request type." >
                <VARDEF varname="restoperation" type="String" htmlType="select" label="Action" options="requesttype" />
                <VARDEF varname="url" type="String" htmlType="text" label="URL" maxsize="2048" />
        </GROUP>
 </VARS>

<VARS type="instance">
</VARS>
</SERVICE>
<OPTION name="requesttype">
   <ELE value="GET" displayname="GET"/>
   <ELE value="POST" displayname="POST"/>
   <ELE value="PUT" displayname="PUT"/>
   <ELE value="DELETE" displayname="DELETE"/>
</OPTION>
</SERVICES>
```

The Service Definition File contains 3 Var Sections: global, instance and wfd. This reflects on which layer the parameters are defined.

- Global is a global setting for all instances of this type of service. You configure them in the dashboard under ->Services->Installation/Setup.
- Instance is a setting specific to one instance of this service. You configure this in the dashboard under Deployment->Services->Configuration.
- wfd describes a set of parameters that you can only set during runtime of the process

Service Instance

The service instance defines an instance of the service. The serviceinstances.xml file contains the service instance. It has information like the display name and the target of the service. Because you can manually define a service through the user interface, this file is optional.

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
<service parentdefname="genericrest" name="genericrest"
        displayname="genericrest"
        description="Acts as a generic rest API client to the REST APIs"
        targetenv="all"
        activestatus="1"
        systemservice="0"
        parentdefid="-1">
</service>
</services>
```

Implementation of the service

Class definition

The file genericrest.jar provides the class definition. This JAR file contains following three files (In the proper package hierarchy)–

```
com
   sterlingcommerce
       woodstock
               services
                    genericrest
                        genericrest.java
                        RestRequestHandler.java
                        RestAPIClientGET.java
                        genericrestserver.java
                        genericrestserverImpl.java
```

When installed, these files will be appended to the **<Install_Directory>/install/properties/dynamicclasspath.cfg** file.

genericrest.java – The service which actually acts as a gateway to handle the upcoming requests depending upon the input parameters from the process data.
RestRequestHandler.java – The java file which distributes the current request depending upon the request type.
RestAPIClientGET.java – The java file which executes the GET request.
genericrestserver.java – The java file that is responsible for restarting, shutdown; etc of the genericrest service.
genericrestserverImpl.java – The java file that implements the genericrestserver class.

Updating Classpath for external Jars

The genericrest is implemented in Java. It requires some JAR files.

- hamcrest-core-1.1.jar
- json-simple-1.1.1.jar

- junit-4.10.jar

These files are basically required for JSON parsing of the response obtained from the APIs. To avoid build failures, these files must be added to the Java Class path.

Workflow (Business Process) to test the genericrest service

The file genericrest.bpml.in is a test workflow for testing the service. When it is executed, it sets the **requesttype** and **url** from the user interface into the process data, and then genericrest service is executed. Following is the Code for the Business process-

```
<process name="genericrest">
 <sequence>
    <operation name="Request">
      <participant name='genericrest'/>
      <output message='xout'>
        <assign to='url'> http://dublr024vm.devlab.ibm.com:60633/B2BAPIs/svc/cadigitalcertificates/</assign>
        <assign to='restoperation'>GET</assign>
        <assign to='.' from='*' />
      </output>
     <input message="xin">
       <assign to='.' from='*' />
     </input>
    </operation>
 </sequence>
</process>
```

Installing and using the genericrest service

Depending upon the way you chose to build the service in the Abstract section, you have two different ways of installing the service.

1. Pre-package the application (jar) In the directory format specified above. Use the InstallService.sh or Install Service.cmd file to install the genericrest.jar file. For information about the InstallService file, see bin Directory Files.

2. Or, you can put the resource and the source files of your service in their respective folders inside the assets/product/src and assets/product/resources directory.

   In our case, following are the source (src) and resources files –

```
Source (src)
assets
   product
      src
         com
           sterlingcommerce
               woodstock
                   services
                      genericrest
                         genericrest.java
                         RestRequestHandler.java
                         RestAPIClientGET.java
                         genericrestserver.java
                         genericrestserverImpl.java
```

```
Resources
assets
   product
      resources
         genericrest
             bpml
               genericrest.bpml
             files
               properties
                  lang
                    en
                      genericrest.properties
```

```
                servicedefs
                        genericrest.xml.in
                serviceinstances.xml.in
```

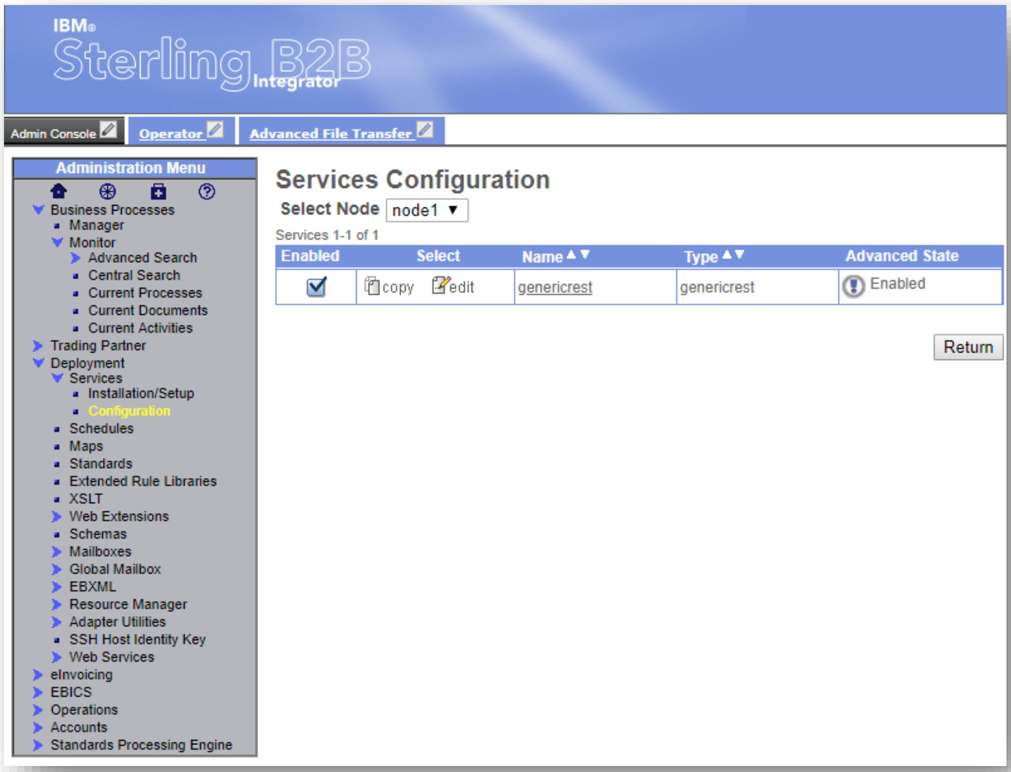
Now, after placing these files as shown above, follow these set of steps

1. Go to assets directory and then run **./build.sh**
   This will build and package all the newly copied files (new service files) into jar files. These jar files will be created in the assets/dist/ directory.

2. Now, go to …/<Install_Directory>/install/bin/ and run the following command –
   ./InstallService.sh <path of the newly created asset jar>
   In our case, this command is –
   ./InstallService.sh /asset/dist/patch/patch_asset_4060603.jar

3. In the same directory, run **./hardstop.sh** followed by **./run.sh**
   This will restart the ISBI server.

Following either of the two ways mentioned above, A service called **genericrest Service** will be shown created in the ISBI dashboard. This service can be seen by going to the **Deployment>>Services>>Configuration** from the left panel menu.



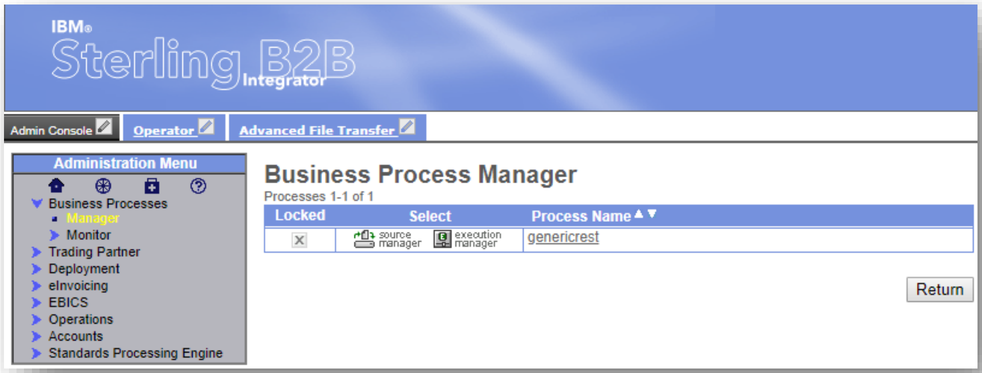
Services Configuration

Select Node node1

Services 1-1 of 1

Enabled	Select	Name	Type	Advanced State
✓	copy edit	genericrest	genericrest	Enabled

A workflow (business process) called **genericrest** will also be created. You can test the service by executing this business process. The response from the Service will be specified in the Process Data itself.

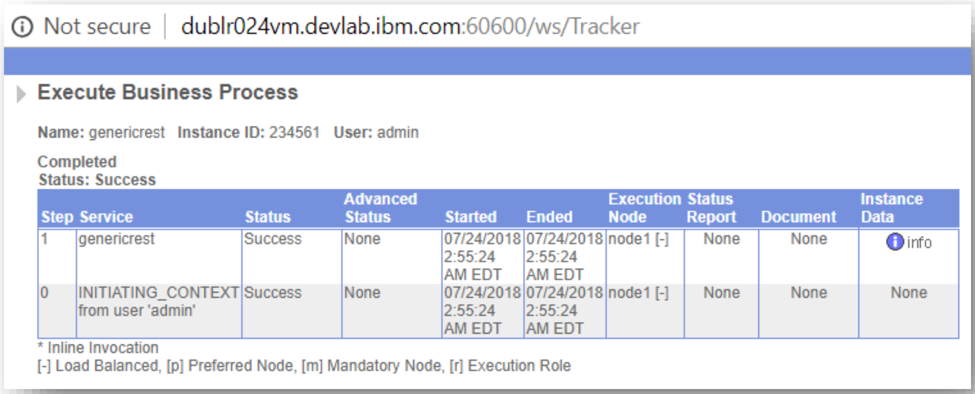


Business Process Manager

Processes 1-1 of 1

Locked	Select	Process Name
☒	source manager, execution manager	genericrest

Business process has been created.



Not secure | dublr024vm.devlab.ibm.com:60600/ws/Tracker

Execute Business Process

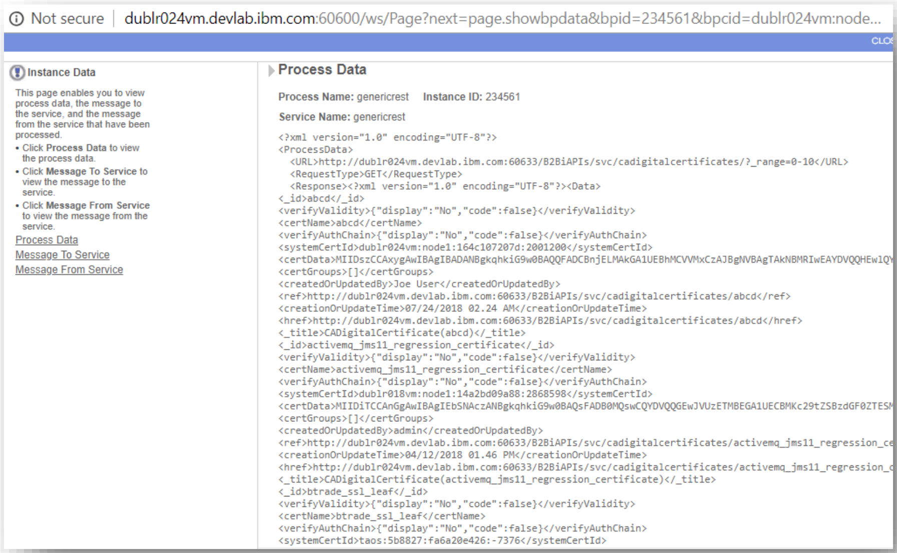
Name: genericrest **Instance ID:** 234561 **User:** admin

Completed
Status: Success

Step	Service	Status	Advanced Status	Started	Ended	Execution Node	Status Report	Document	Instance Data
1	genericrest	Success	None	07/24/2018 2:55:24 AM EDT	07/24/2018 2:55:24 AM EDT	node1 [-]	None	None	info
0	INITIATING_CONTEXT from user 'admin'	Success	None	07/24/2018 2:55:24 AM EDT	07/24/2018 2:55:24 AM EDT	node1 [-]	None	None	None

* Inline Invocation
[-] Load Balanced, [p] Preferred Node, [m] Mandatory Node, [r] Execution Role

Successful execution of the Business process.



Not secure | dublr024vm.devlab.ibm.com:60600/ws/Page?next=page.showbpdata&bpid=234561&bpcid=dublr024vm:node...

Instance Data

This page enables you to view process data, the message to the service, and the message from the service that have been processed.

- Click **Process Data** to view the process data.
- Click **Message To Service** to view the message to the service.
- Click **Message From Service** to view the message from the service.

Process Data
Message To Service
Message From Service

Process Data

Process Name: genericrest **Instance ID:** 234561

Service Name: genericrest

```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessData>
  <URL>http://dublr024vm.devlab.ibm.com:60633/B2BiAPIs/svc/cadigitalcertificates/?_range=0-10</URL>
  <RequestType>GET</RequestType>
  <Response><?xml version="1.0" encoding="UTF-8"?><Data>
<_id>abcd</_id>
<verifyValidity>{"display":"No","code":false}</verifyValidity>
<certName>abcd</certName>
<verifyAuthChain>{"display":"No","code":false}</verifyAuthChain>
<systemCertId>dublr024vm:node1:164c107207d:2001200</systemCertId>
<certData>MIIDszCCAxygAwIBAgIBADANBgkqhkiG9w0BAQQFADCBnjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNBMRIwEAYDVQQHEw1QY
<certGroups>[]</certGroups>
<createdOrUpdatedBy>Joe User</createdOrUpdatedBy>
<ref>http://dublr024vm.devlab.ibm.com:60633/B2BiAPIs/svc/cadigitalcertificates/abcd</ref>
<creationOrUpdateTime>07/24/2018 02.24 AM</creationOrUpdateTime>
<href>http://dublr024vm.devlab.ibm.com:60633/B2BiAPIs/svc/cadigitalcertificates/abcd</href>
<_title>CADigitalCertificate(abcd)</_title>
<_id>activemq_jms11_regression_certificate</_id>
<verifyValidity>{"display":"No","code":false}</verifyValidity>
<certName>activemq_jms11_regression_certificate</certName>
<verifyAuthChain>{"display":"No","code":false}</verifyAuthChain>
<systemCertId>dublr018vm:node1:14a2bd09a88:2868598</systemCertId>
<certData>MIIDiTCCAnGgAwIBAgIEbSNAczANBgkqhkiG9w0BAQsFADB0MQswCQYDVQQGEwJVUzETMBEGA1UECBMKc29tZSBzdGF0ZTESM
<certGroups>[]</certGroups>
<createdOrUpdatedBy>admin</createdOrUpdatedBy>
<ref>http://dublr024vm.devlab.ibm.com:60633/B2BiAPIs/svc/cadigitalcertificates/activemq_jms11_regression_ce
<creationOrUpdateTime>04/12/2018 01.46 PM</creationOrUpdateTime>
<href>http://dublr024vm.devlab.ibm.com:60633/B2BiAPIs/svc/cadigitalcertificates/activemq_jms11_regression_c
<_title>CADigitalCertificate(activemq_jms11_regression_certificate)</_title>
<_id>btrade_ssl_leaf</_id>
<verifyValidity>{"display":"No","code":false}</verifyValidity>
<certName>btrade_ssl_leaf</certName>
<verifyAuthChain>{"display":"No","code":false}</verifyAuthChain>
<systemCertId>taos:5b8827:fa6a20e426:-7376</systemCertId>
```

Process data returned after the successful completion of the Business process.

Results and Discussions

B2Bi REST implementation allows user to programmatically Create, Read, Update and Delete resources in B2Bi. JSON and XML supported as input and output formats. The Rest Client service that I have created will allow the user to place the REST requests on the B2B resources. Also, this service can be further upgraded to have multiple simultaneous calls on several URLs. Further, Input feeding from the user can be made more dynamic in practice. Currently, a Workflow (Business Process) is hardcoded to handle a single request only. A dynamic input will allow the user to fetch input dynamically from the process data of some other process.

Conclusions

During past few weeks, I Created a REST Client Service for the Sterling Integrator. This will allow the End Users to place the REST API calls to the B2Bi APIs using the SI Dashboard. Up to this point, I have completed the implementation of the GET request.

Next Target

My target for the next 15 days is to complete this REST Client service by adding the POST, PUT and DELETE request functionalities too.