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

(21st August 2018 – 8th October 2018)

Interim Review

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.

Brief Overview

Until now (Post Midterm review), I have completed several tasks pertaining to the B2Bi Sterling Integrator. Proceeding in stages, Firstly, I researched about the Java Spring-Boot microservice architecture and Web REST-API construction. Secondly, I researched about IBM TenX framework. All the IBM B2B APIs are built using the IBM TenX framework only. Thirdly, I learnt about deploying a sample SpringBoot application on the Sterling Integrator Liberty Server. Using this knowledge of Tenx and SpringBoot, I transformed two B2B APIs (originally Tenx based) namely User Accounts and UserVirtualRoot to SpringBoot framework. Then I deployed these new converted APIs on the same Liberty WebSphere server where the original B2B APIs were installed.

Now, that I had two implementations of the same set of APIs (Tenx and SpringBoot), I had to run the performance benchmarking against each other. Outputs of the performance testing were outstandingly good and are shown below in the next section. To experiment further and to be sure with the results, we flooded the server Database with a large number of objects to perform the extensive testing. For the easy usage documentation of these new APIs, I added the Swagger Hub API Documentation dependency to the Spring Code. Lastly, these new microservice spring-based APIs were lacking the UI front so I got myself acquainted to the Angular JS frontend framework. As a part of IBM internal hackathon, I developed a combined UI and a SpringBoot API Gateway (Secured using IBM w3id) in less than 24 hours.

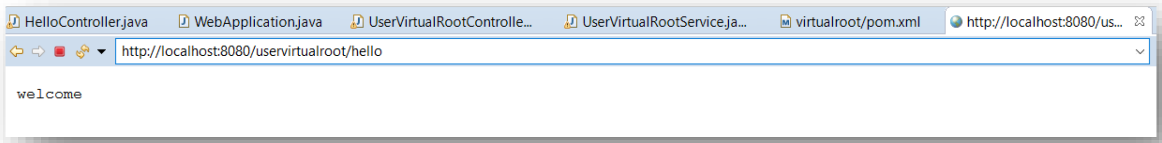
Apart from my main work, as a part of the IBM **XLence Program** for IBM Interns, we just completed an end-to-end workflow of our deployment framework called DFRAME and gave a demo to the stakeholders and managers.

Report on the Present Investigation

(Progress until the Interim review)

Task 1: Researching about SpringBoot Architecture, IBM TenX Framework and deploying a sample SpringBoot application on Liberty server

During this duration, I researched and developed a sample REST controller which returns Welcome message when a GET call to **/hello URL** is made. In this sample application, we have created a class HelloController and annotated it with @RestController which calls REST API and provides json response.


HelloController.java
WebApplication.java
UserVirtualRootControlle...
UserVirtualRootService.ja...
virtualroot/pom.xml
http://localhost:8080/us...

```
http://localhost:8080/uservirtualroot/hello

welcome
```

(Sample Spring Boot API)

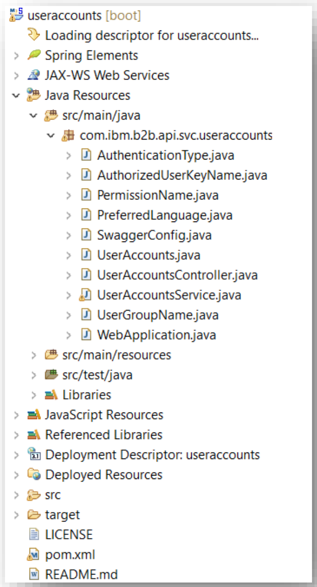
Task 2: Converting UserAccounts and UserVirtualRoots APIs to SpringBoot

Conversion of UserVirtualRoots API to SpringBoot was explained in the last report. Here, I will cover the conversion of UserAccounts API.

This task required the following steps (demonstrated in detail in the last report).

1. Creating the Spring Starter project.
2. Change the POM.XML dependency file for adding the dependencies.
3. Changing the code structure of Java Source files. (Tenx to Springboot format)

   Here, there are following classes –
   (Code Cannot be revealed due to confidentiality clauses)

   ```
   useraccounts [boot]
     Loading descriptor for useraccounts...
   > Spring Elements
   > JAX-WS Web Services
   v Java Resources
     v src/main/java
       v com.ibm.b2b.api.svc.useraccounts
         > AuthenticationType.java
         > AuthorizedUserKeyName.java
         > PermissionName.java
         > PreferredLanguage.java
         > SwaggerConfig.java
         > UserAccounts.java
         > UserAccountsController.java
         > UserAccountsService.java
         > UserGroupName.java
         > WebApplication.java
     > src/main/resources
     > src/test/java
     > Libraries
   > JavaScript Resources
   > Referenced Libraries
   > Deployment Descriptor: useraccounts
   > Deployed Resources
   > src
   > target
     LICENSE
     pom.xml
     README.md
   ```

   - **POJO Classes:** Classes to represent the object entity structure.

     UserAccounts.java
     AuthenticationType.java
     AuthorizedUserKeyName.java
     PermissionName.java
     PreferredLanguage.java
     UserGroupName.java

   - **Controller Class:** This class is used to map the URL and Request Type to the respective Method Calls.

     UserAccountsController.java

   - **Service Class:** This class is used to implement the actual functions for CRUD operations.

     UserAccountService.java

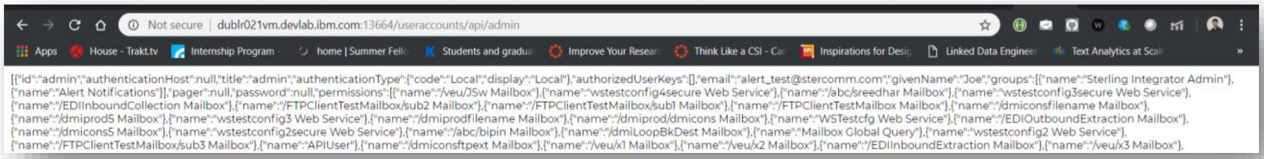
   - **Web Application Class:** This class is used to start the SpringBoot.

     WebApplication.java

4. Package the Application as a WAR file using Maven Install
5. Deploy the File on the Liberty Server
   - Copy the generated WAR file to the Liberty Server's App directory.
   - Edit the server.xml file to add the External entry for Liberty to recognize your app WAR as an endpoint.

6. Restart the Server
   - Run these Commands from the install/bin/ directory.
     ./hardstop.sh -> ./run.sh -> ./startLiberty.sh

7. Access the API on the specified URL



```
[{"id":"admin","authenticationHost":null,"title":"admin","authenticationType":{"code":"Local","display":"Local"},"authorizedUserKeys":[],"email":"alert_test@stercomm.com","givenName":"Joe","groups":[{"name":"Sterling Integrator Admin"},
{"name":"Alert Notifications"}],"pager":null,"password":null,"permissions":[{"name":"/veu/JSw Mailbox"},{"name":"wstestconfig4secure Web Service"},{"name":"/abc/sreedhar Mailbox"},{"name":"wstestconfig3secure Web Service"},
{"name":"/EDIInboundCollection Mailbox"},{"name":"/FTPClientTestMailbox/sub2 Mailbox"},{"name":"/FTPClientTestMailbox/sub1 Mailbox"},{"name":"/FTPClientTestMailbox Mailbox"},{"name":"/dmiconsfilename Mailbox"},
{"name":"/dmiprod5 Mailbox"},{"name":"wstestconfig3 Web Service"},{"name":"/dmiprodfilename Mailbox"},{"name":"/dmiprod/dmicons Mailbox"},{"name":"WSTestcfg Web Service"},{"name":"/EDIOutboundExtraction Mailbox"},
{"name":"/dmicons5 Mailbox"},{"name":"wstestconfig2secure Web Service"},{"name":"/abc/bipin Mailbox"},{"name":"/dmiLoopBkDest Mailbox"},{"name":"Mailbox Global Query"},{"name":"wstestconfig2 Web Service"},
{"name":"/FTPClientTestMailbox/sub3 Mailbox"},{"name":"APIUser"},{"name":"/dmiconsftpext Mailbox"},{"name":"/veu/x1 Mailbox"},{"name":"/veu/x2 Mailbox"},{"name":"/EDIInboundExtraction Mailbox"},{"name":"/veu/x3 Mailbox"},
```

Task 3: Performance benchmarking (TenX vs SpringBoot)

Aim of this benchmark was to compare the performance of using SpringBoot as a base B2B API framework against IBM Tenx API framework. The API which we picked for comparison was UserAccounts.

These tests have been done in two stages –

Case 1 : Sparsely populated DB of UserAccounts

- **Single-Read Request using TenX. (1507ms)**
- **Single-Read Request using SpringBoot. (1210ms)**

Case 2 : Densely populated DB of UserAccounts (10000 UserAccounts)

For performing extensive testing, we wrote a script that floods the UserAccounts Database by adding (sequentially posting) 10000 temporary User Accounts to the API endpoint.

- **Single-Read Request using TenX. (10508ms)**
- **Single-Read Request using SpringBoot. (593ms)**
- **Read-All Request using TenX. (61340ms, This Request fetches nearly 1000 records)**
- **Read-All Request using SpringBoot. (58066ms, This Request fetches 10000 records)**

Some Noteworthy Points

1. These tests have been performed on the same machine and on the same WebSphere Liberty server.

2. For Read-All requests, TenX only fetches maximum 1000 records.

Conclusions

1. In case of Sparsely populated DB, the difference between SpringBoot and TenX is not very drastic. (SpringBoot is nearly 200-250ms faster).

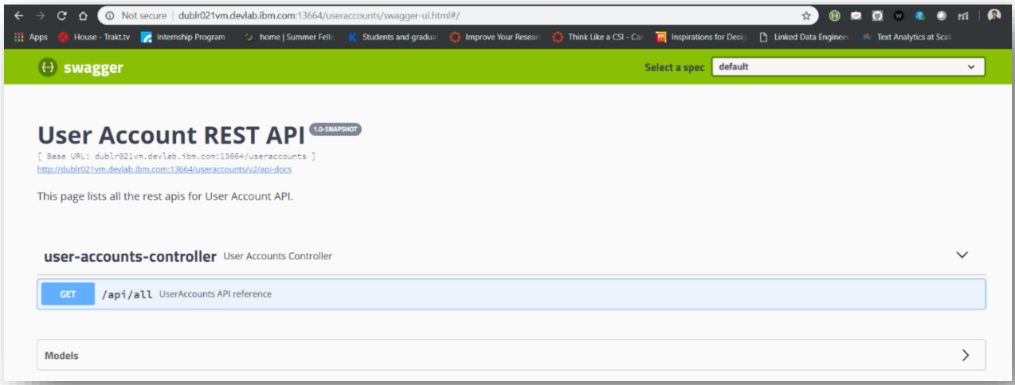
2. In the case of Densely populated DB, the difference is huge.
   a. Single-Read Call
      Effective time for TenX = 10000ms (Approx. Average)
      Effective time for SpringBoot = 600ms (Approx. Average)
      Performance Ratio = 10000/600 = 16.66 times faster execution of SpringBoot

   b. Read-All Call
      Effective time for TenX = 60000ms/1000 records = 60ms (Approx. Average)
      Effective time for SpringBoot = 58000ms/10000 records = 5.8ms (Approx. Average)
      Performance Ratio = 60/5.8 = 10 times faster execution of SpringBoot

3. Speaking holistically, on an average, SpringBoot is nearly ~10-12 times faster than IBM TenX.

	TenX	**SpringBoot**	**Performance Ratio**
Single GET Request	12 seconds	1 second	~12 times faster
Bulk GET Request	68 seconds	6 seconds	

Task 4: Adding Swagger Hub Documentation Dependency


Not secure | dublr021vm.devlab.ibm.com:13664/useraccounts/swagger-ui.html#/
swagger
Select a spec
default
GET
Models

User Account REST API 1.0-SNAPSHOT

[Base URL: dublr021vm.devlab.ibm.com:13664/useraccounts]

http://dublr021vm.devlab.ibm.com:13664/useraccounts/v2/api-docs

This page lists all the rest apis for User Account API.

user-accounts-controller User Accounts Controller

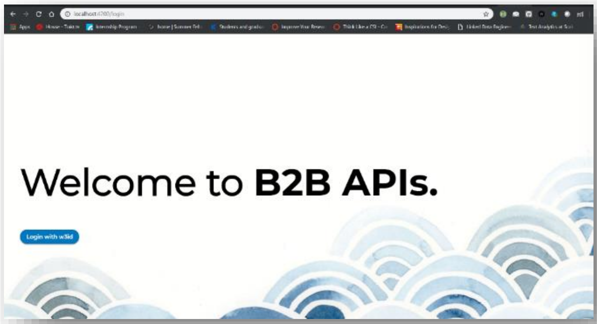
/api/all UserAccounts API reference

Task 5: Adding UI using Angular JS to SpringBoot APIs

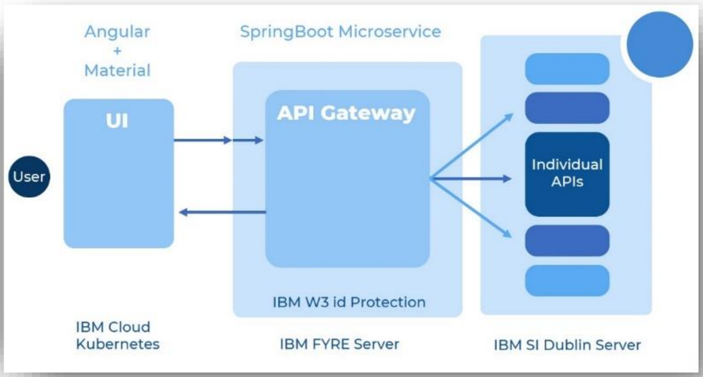
Now, the next task was to add the UI to the newly created APIs.

I chose Angular JS for this because it has large community support and is extremely modular. It uses a Language called Typescript (by Microsoft) for code.

On the Login page, user needs to click on the **Login with w3id** Button to be authenticated with his/her IBM ID.



Welcome to **B2B APIs.**

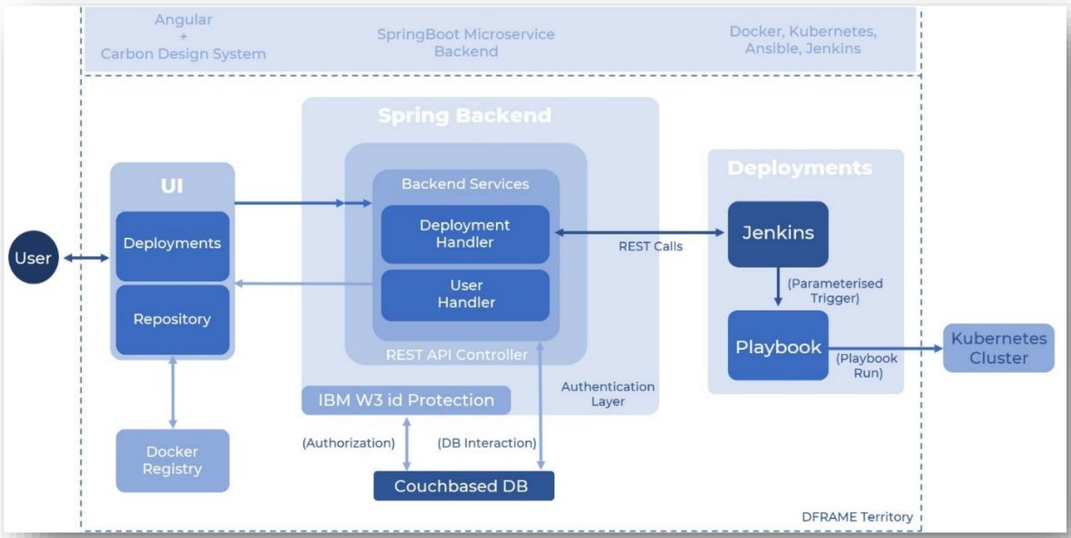


[Link]

The above UI was deployed as a microservice separate than the Backend Spring based API Gateway on IBM Cloud Kubernetes Cluster by building it as a docker image.

Task 6 (Apart from the main work): DFRAME Docker deployment framework

As a part of the XLence Program or the IBM interns, we developed a Docker image deployment framework that builds and deploys an Application Image in just a few clicks from the intuitive UI. View the Workflow for this application below-



Results and Discussions

The heart of B2Bi is composed of several business processes which are used to automate the operation of B2B services (Mailbox, Invoice, Order; etc). By using REST APIs, you can perform certain B2B functions using Sterling B2B Integrator. B2B REST APIs were released starting with Sterling B2B Integrator V5.2.6.1. These Rest APIs were built using core Java and IBM TenX framework. IBM and Sterling team is exploring about some alternative frameworks for converting these APIs which would make the execution of calls faster. Separate microservices can be deployed on the B2B SI platform server to enhance its functionality. During this duration, I transformed few of the B2B APIs to the SpringBoot framework to demonstrate to the team the potential performance improvement. All B2B APIs currently come packaged as a single WAR file. So, I took the goal to convert all these bundled APIs into separate Spring microservices and deploy them distinctively too.

Conclusions

During past few weeks, I was involved in converting a few B2B APIs like UserVirtualRoot and UserAccounts to Spring Boot Rest API Framework and deploy it in the form of a WAR file on the liberty server of the Sterling Integrator. All the IBM B2B APIs are currently built over the IBM TENX framework that is slow in function as it has useless layering of procedure calls. My team wants to test if converting these APIs to Spring Boot framework will make them faster in execution. For this purpose, currently I am assigned for POCs (Proof of Concept). In this report I have also shown the performance improvement by using the SpringBoot framework over the normal TenX. It is nearly 12 times faster for bulk level calls. This is a huge improvement. My team will consider pitching this idea to the stakeholders and then eventually converting all the B2B APIs to SpringBoot. Customers will also be excited about this speed and performance improvement.

Next Target

Since this POC of proving that SpringBoot is faster than TenX is done, my target for the next 15 days is to create a new service for Sterling Integrator – JSON/XML Transformer. This will allow the customers to use inbuilt XML/JSON service for their data conversion needs. Also, I would need make the Rest API client service (Pre-Midterm) more production ready as it is going to be passed to the QA (Quality Assurance) stage for final testing.