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 – 7th September 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 created a full-fledged REST API client for the B2Bi APIs. This client leverages the use of Business Processes for executing the REST CRUD calls to the APIs. Now my task is to convert the existing B2B Rest APIs to the Spring Framework and then compare the execution call time of the of the modified Spring APIs with the previously created IBM TENX APIs.

In this Report, I created a sample Spring Boot REST application and deployed it on the Sterling Integrator WebSphere Liberty server.

About the Spring Framework

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to, or even replacement for the Enterprise JavaBeans(EJB) model.

Spring Boot follows some of these basic annotations-

@RestController
This is a stereotype annotation that combines @ResponseBody and @Controller. More than that, it gives more meaning to your Controller and also may carry additional semantics in future releases of the framework.Using this annotation we can make our class to implement a REST API, thus serving only JSON,XML or custom MediaType content.

@Service
Using this annotation, we can make our class as service class and can inject dependencies to other classes. We don’t need to instantiate this class again in controller, we just only need to create an object and use its methods.

@Entity
It’s an annotation of JPA (Java Persistence APIs) that can make class as table in the database.

@Id
It’s an annotation of JPA (Java Persistence APIs) that can make variable as primary key in the database.

We can add multiple dependencies using maven and need not install the associated jars. Automatically all the jars get installed.

Databases
Spring boot supports all types of relational and non-relational databases. We just need to add the dependency of it and can use any database in our application

Spring Security

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications. Apart from this, it can support other securities such as Apache Shiro.

Report on the Present Investigation

(Progress during this 15-days period)

Spring Boot is a framework designed to simplify the creation of new services. For the simple use cases, the needed libraries are already bundled in the fitting combinations and versions in so-called spring starters. In any Spring Boot based REST API, following is the Separation of layers where each layer is an individual module/project.

REST API

- Packaged as war
- It has rest controllers that handle request/responses
- depends on Service Module below

Service

- Packaged as jar
- Business logic abstractions, this layer has no idea how to communicate with data source.
- It will be autowired in rest controllers
- Depends on DAO/Repository module below

DAO/Repository

- Packaged as jar
- Talks to data source directly, has operations commonly known as CRUD. It could be simple jdbc, JPA, or even file access.
- Depends on domain module below

Domain

- Packaged as jar
- It has your domain models, normally POJO classes. If you are using ORM, they are ORM entities.
- It could also have DTO (Data Transfer Object), which are still under hot debates. Use it or not is your call.

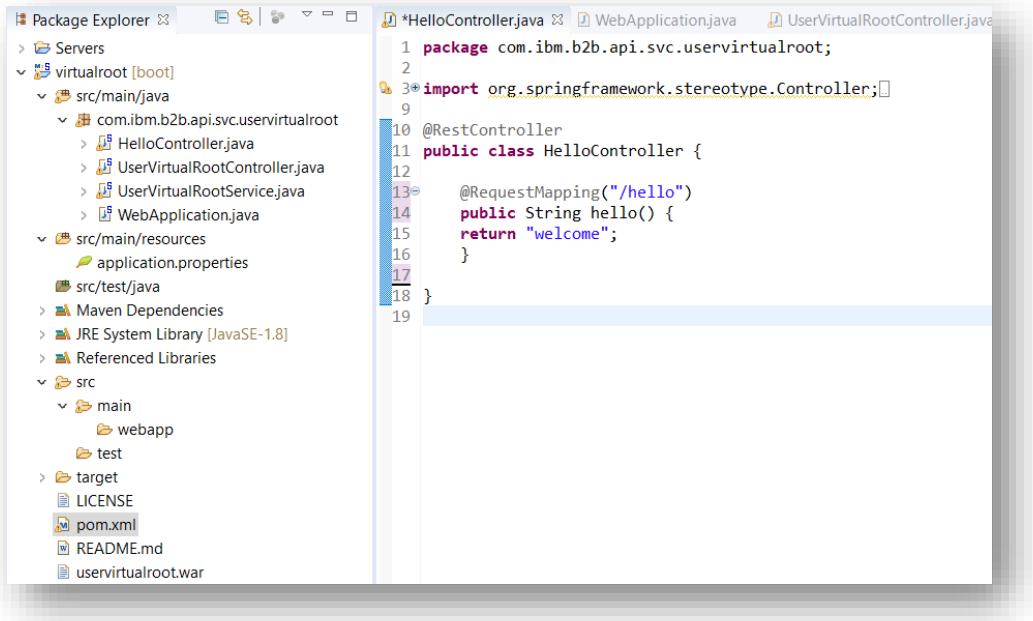
More modules such as utility, third party integration, etc. can be added. but the above are highly recommended to have.

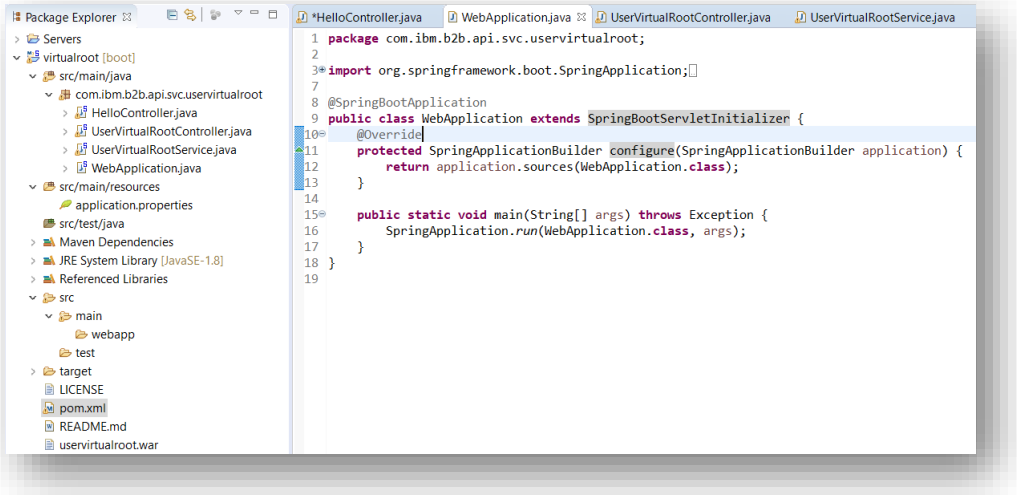
Build/Dependency Management Tools - There are plenty of them like maven and gradle versions. We used **Maven with Spring**. It simply works for the above project structure.

Creating a Sample REST Spring Application in STS (Spring Tool Suite) IDE

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.

We can also provide individual mappings for http requests such as @PostMapping and can also set this in @RequestMapping(method="POST").

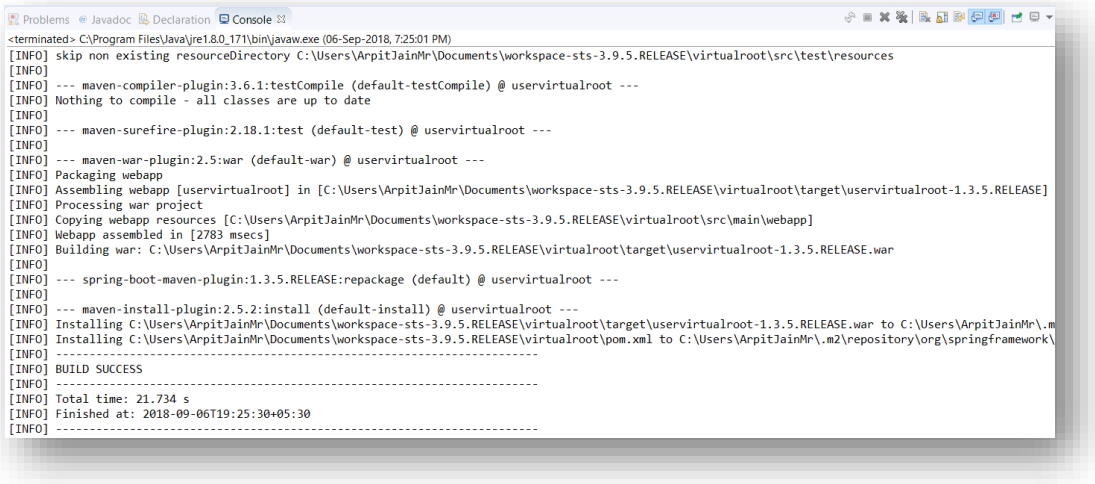




(Application class to start the spring boot)

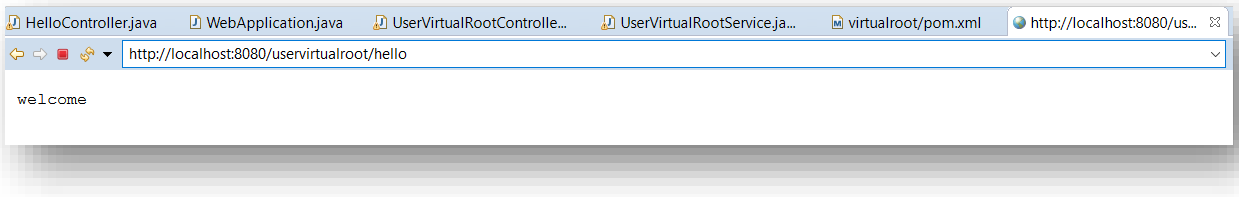
Packaging the Service as a WAR

Exported the application into a WAR file using Maven Install packaging.



Installing/Deploying the WAR on the WebSphere Liberty Server

This WAR file is now copied to the Liberty server and the server.xml configuration file is modified to create an entry for the new app. The server is then restarted, and the liberty server is also run for reconfiguration.



Results and Discussions

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.

Conclusions

During past few weeks, I was involved in making a sample Spring Boot application 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. My team wants to convert these APIs to Spring Boot framework.

Next Target

My target for the next 15 days is to transform a small API like **UserVirtualRoot** to the Spring Boot framework and measure how fast is it when compared to the older framework.