# SPRING CLOUD NETFLIX DEVELOPER GUIDELINES

TATA

TATA CONSULTANCY SERVICES

## TABLE OF CONTENTS

# 1 INTRODUCTION

## 1.1 DESCRIPTION

This document helps to understand the basic of Spring Cloud Netflix architecture. Gives overview of Spring Cloud Netflix, tell How to create Spring Boot application and implement in

- ✓ **Service Discovery (Eureka),**
- ✓ **Circuit Breaker (Hystrix),**
- ✓ **Intelligent Routing and**
- ✓ **load Balancing (Zuul)**

It also specifies about spring and REST standards. It will be used by each developer joining to in Netflix architecture.

This document also covers installation of required software's to develop application using Spring Netflix architecture and standard followed by project team in terms of project creation.

This document we are discussing about the use case "Shopping list functionality" and how to implement in Spring Netflix architecture.

**Use case:**

- ✓ **Create shopping list**
- ✓ **Modify shopping list – add product and delete product**
- ✓ **Delete shopping list**

### 1.2 RELATED DOCUMENTS

Please refer the links given below.

http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html

http://projects.spring.io/spring-cloud/spring-cloud.html

https://spring.io/blog/2014/06/03/introducing-spring-cloud

### 1.3 GLOSSARY

(if Any)

## 2 PREREQUISITES & INSTALLATION

### 2.1 NETFLIX SRPRING CLOUD INSTALLATION

This section provides the installation instructions to the developers for setting up their development systems on Windows platform.

### 2.1.1 S/W RECOMMONDATIONS

Netflix spring cloud stack supports Windows 7/CentOS/UBUNTU OS

### 2.1.2 H/W RECOMMONDATIONS

Following are the minimum hardware system requirements:

- Windows 7 or more (32bit or 64 bit)
- I3 processor or more
- 2 GB RAM minimum
- 50 GB Hard Disk space

## 2.2 INSTALLATION

Download and Install spring-tool-suite-3.7.3.RELEASE-e4.5.2. (STS) in the system and launch it. It will create the default workspace. It is good to have following plugins installed in your eclipse.

Good to have the following prerequisite supporting tools

- Notepad ++
- Chrome 49.0

**Maven version to be available**

- Maven 3.3

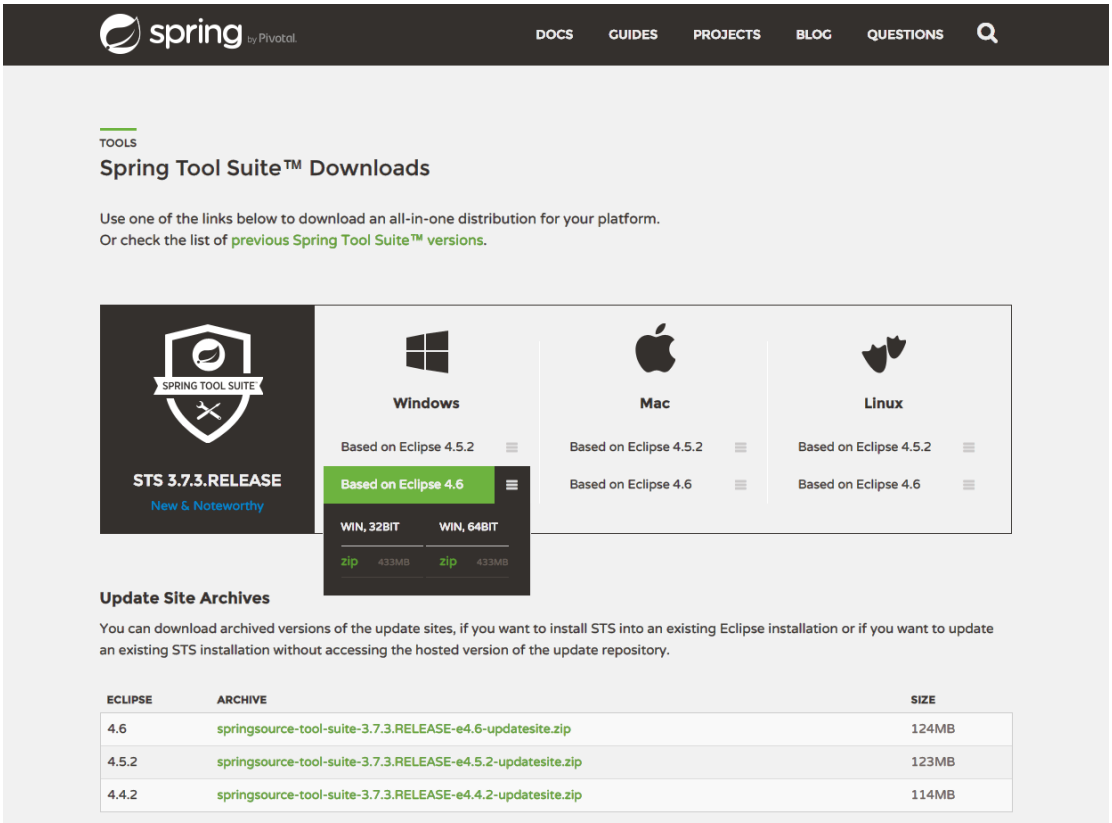
### 2.2.1 JAVA 1.8 INSTALLATION

Netflix global platform requires Java 1.8 to be installed in order to use the unique features of Java 8 effectively.

- Download and install the **jdk1.8.0_66** from the following link for the respective OS
- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Run the executable and install the JDK.

### 2.2.2 STS INSTALLATION

- Download the STS version spring-tool-suite-3.7.3.RELEASE-e4.5.2 from the following link
- https://spring.io/tools/sts/all




- Extract the downloaded zip file.
- Once extracted the STS is ready for use. Click STS.exe and start the application development.

## 2.2.3 COUCHBASE DATABASE INSTALLATION

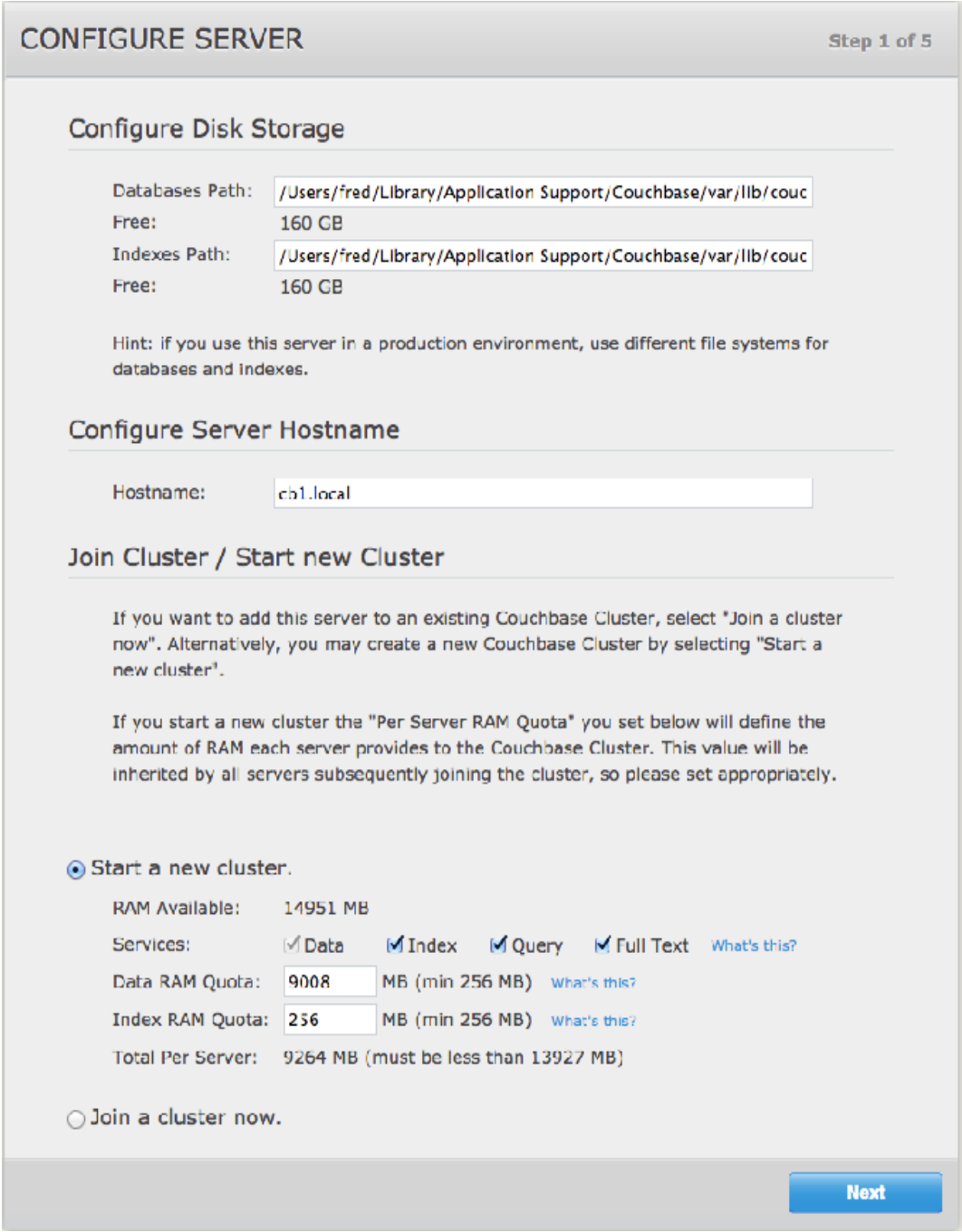
- Download the Couchbase server from the following link for the OS that you will be using http://www.couchbase.com/nosql-databases/downloads#PreRelease To install on Microsoft Windows:
- Double-click the downloaded executable file to start the installer. Follow the installer prompts.
- If desired, you can change the installation location.
- If the default port is unavailable, the installer prompts you for a different port to use for server administration.
- When the installer asks if you want to increase the number of ephemeral ports, click Yes.
- After the installation is complete, restart Couchbase Server to apply the port changes.

**Set up Couchbase Server**

- To set up Couchbase Server in a nonproduction environment, you can accept the default values provided on most of the set-up screens.
- Open a browser and navigate to http://hostname:8091/.
- In the URL, hostname represents the name or IP address of the computer that hosts Couchbase Server. If Couchbase Server is running locally, enter localhost for the host name.
- Click Setup and follow the below steps

**Step 1**

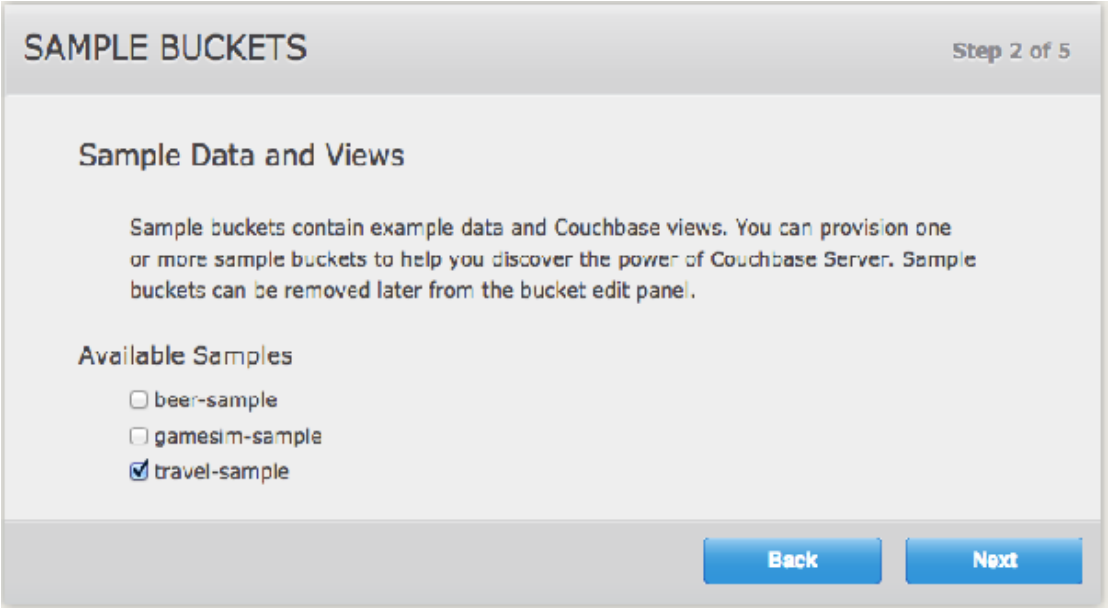
On the Configure Server screen given below, click Next to accept the default values for a new cluster.




**Step 2**

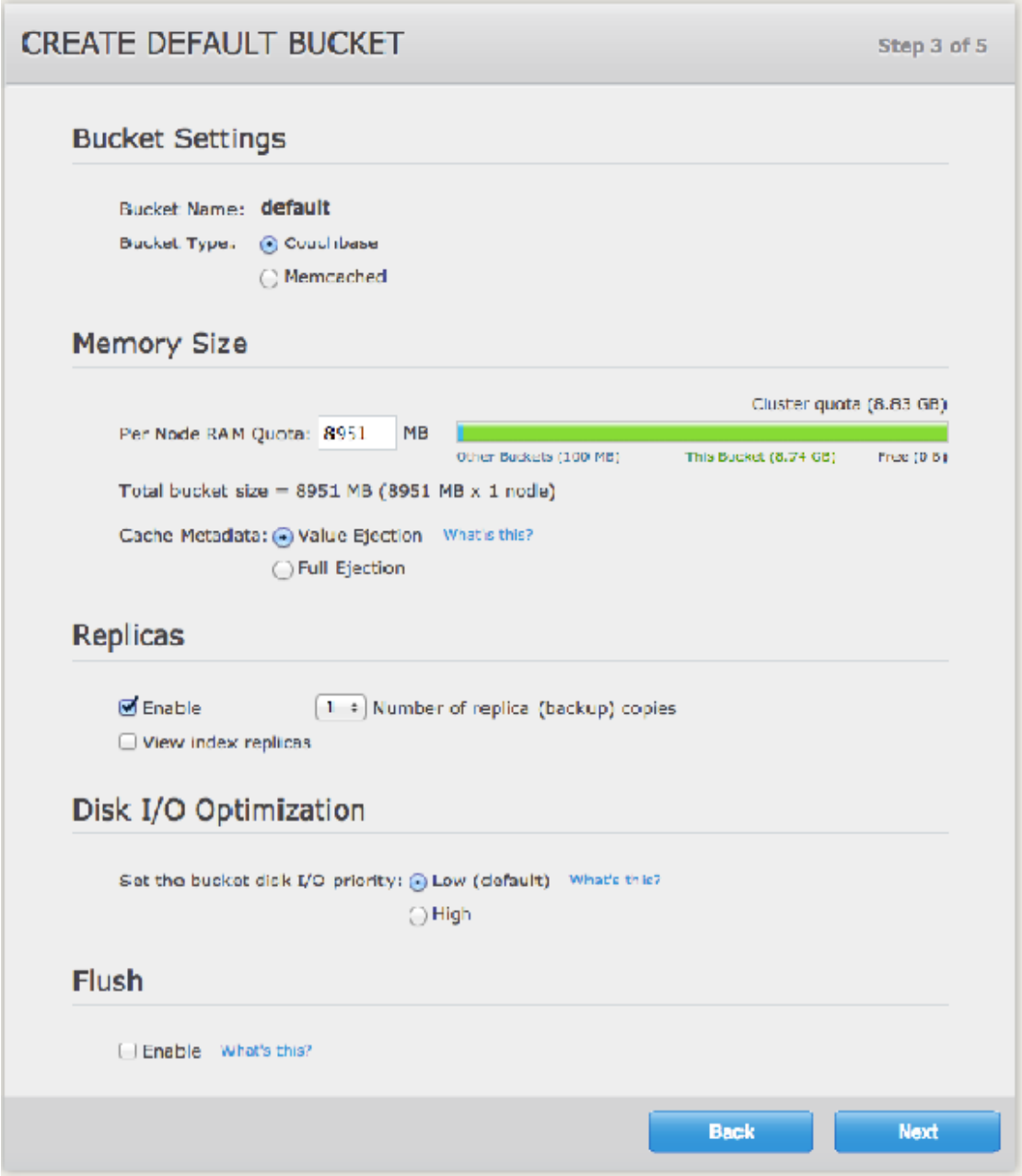
On the Sample Buckets screen given below, under Available Samples select the two samples

We will use later in this tutorial: beer-sample and travel-sample and click Next.



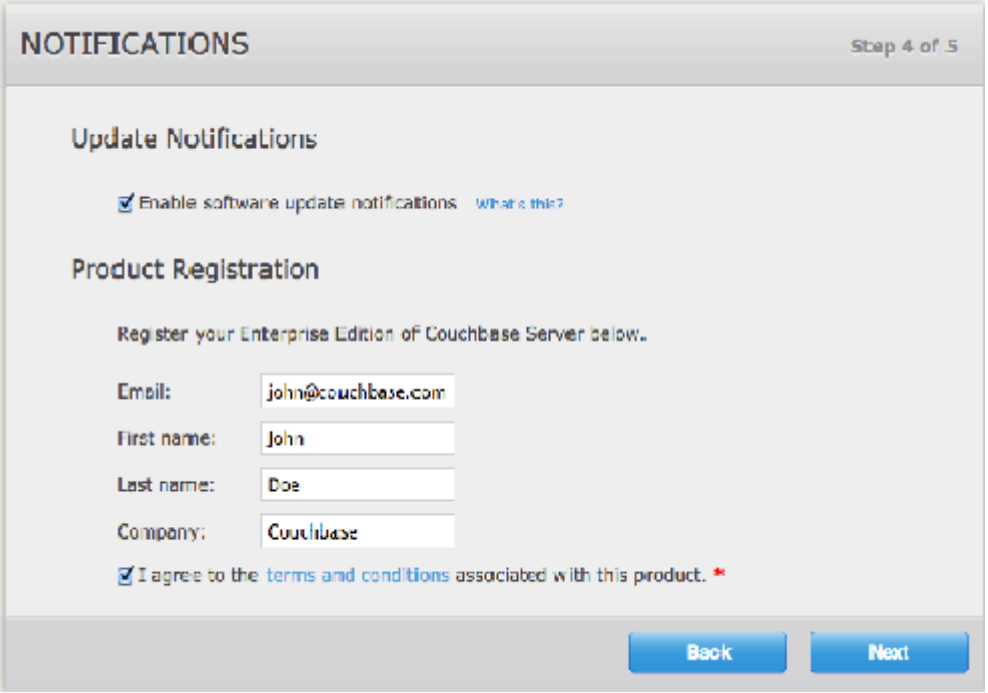

**Step 3**

On the Create Default Bucket screen, under Memory Size set the Per Node RAM Quota to 100 MB and click Next.



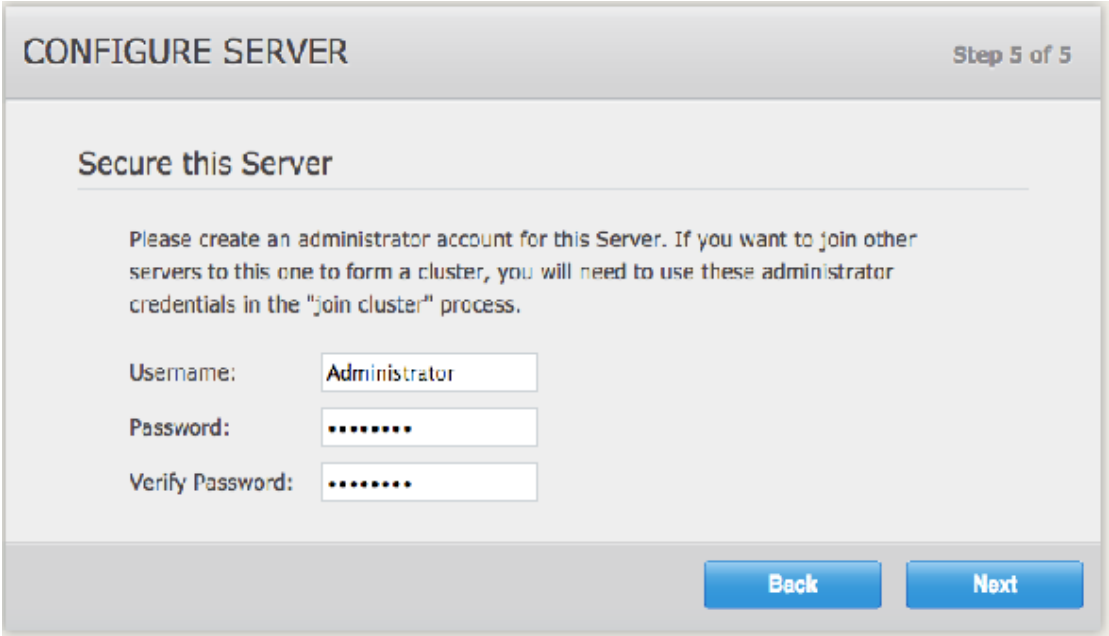

**Step 4**

On the Notifications screen enter your registration information, agree to the terms and conditions, and click Next.

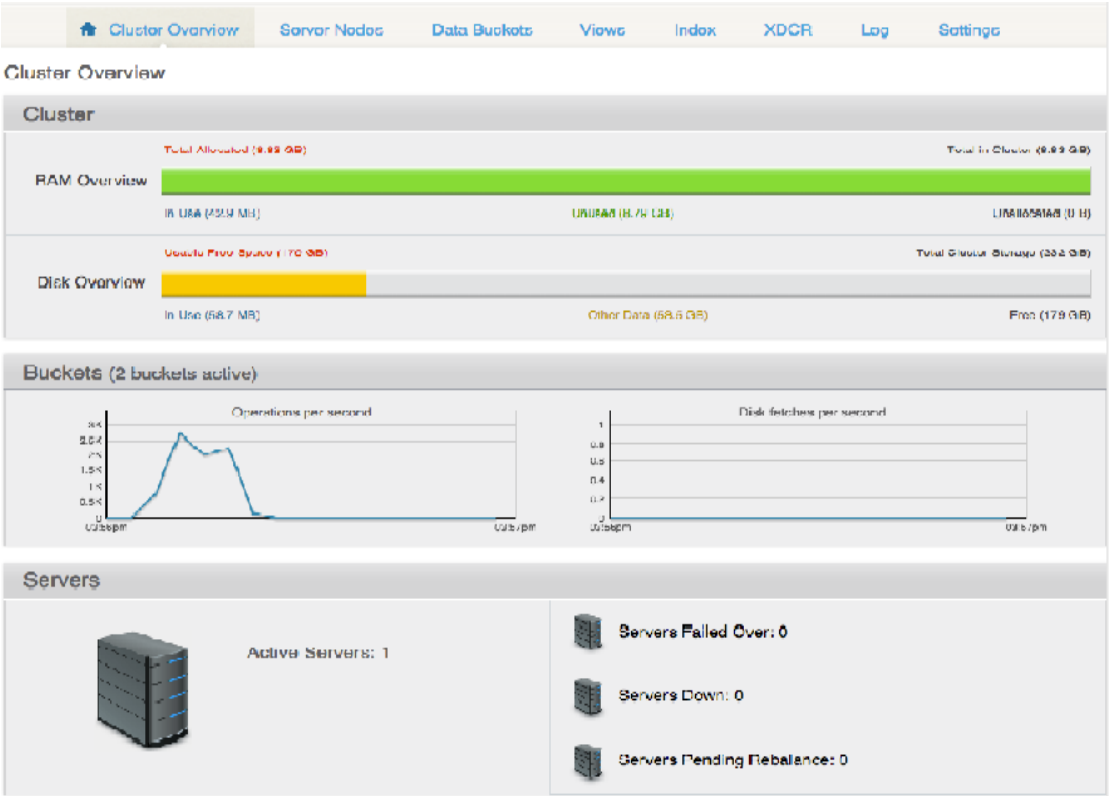



**Step 5**

On the Configure Server screen enter and verify a password for the administrator account, and click Next.




The Couchbase Web Console opens and displays the Cluster Overview.




While the Couchbase Web Console is open, click on the tabs in the top menu and look at the screens to familiarize yourself with the information and available options.

# 3 NETFLIX SPRING CLOUD ARCHITECTURE

The e-commerce Engineering Team carefully selected this microservices tiering design pattern to align it best for the future. We established the following tiers based on the initial recommendation from David Raccah, the consultant from LinkedIn, E-Bay and several other companies that have successfully delivered global microservices

## 3.1 SERVICE ORIENTED ARCHITECTURE

The Blueprint defines a multi-tiered Service-Oriented Architecture (SOA) with loosely coupled, well-defined tiers (A,B,C,D, and Z). Each service is fine-grained, with lower tier (Tier C) services representing objects and middle tier (Tier B) services performing object aggregation and orchestration.

## 3.2 SERVICE APIS

SOA is a mature technology that is well-suited for e-commerce applications and fundamental to architectures in comparable companies, such as Walmart and eBay. All inter-service API's are all REST/JSON-based. Tier B and Tier Care Java-based services. Tier A can be built using variety of languages, such as Node.js, Java, GO or any other REST capable programming language.

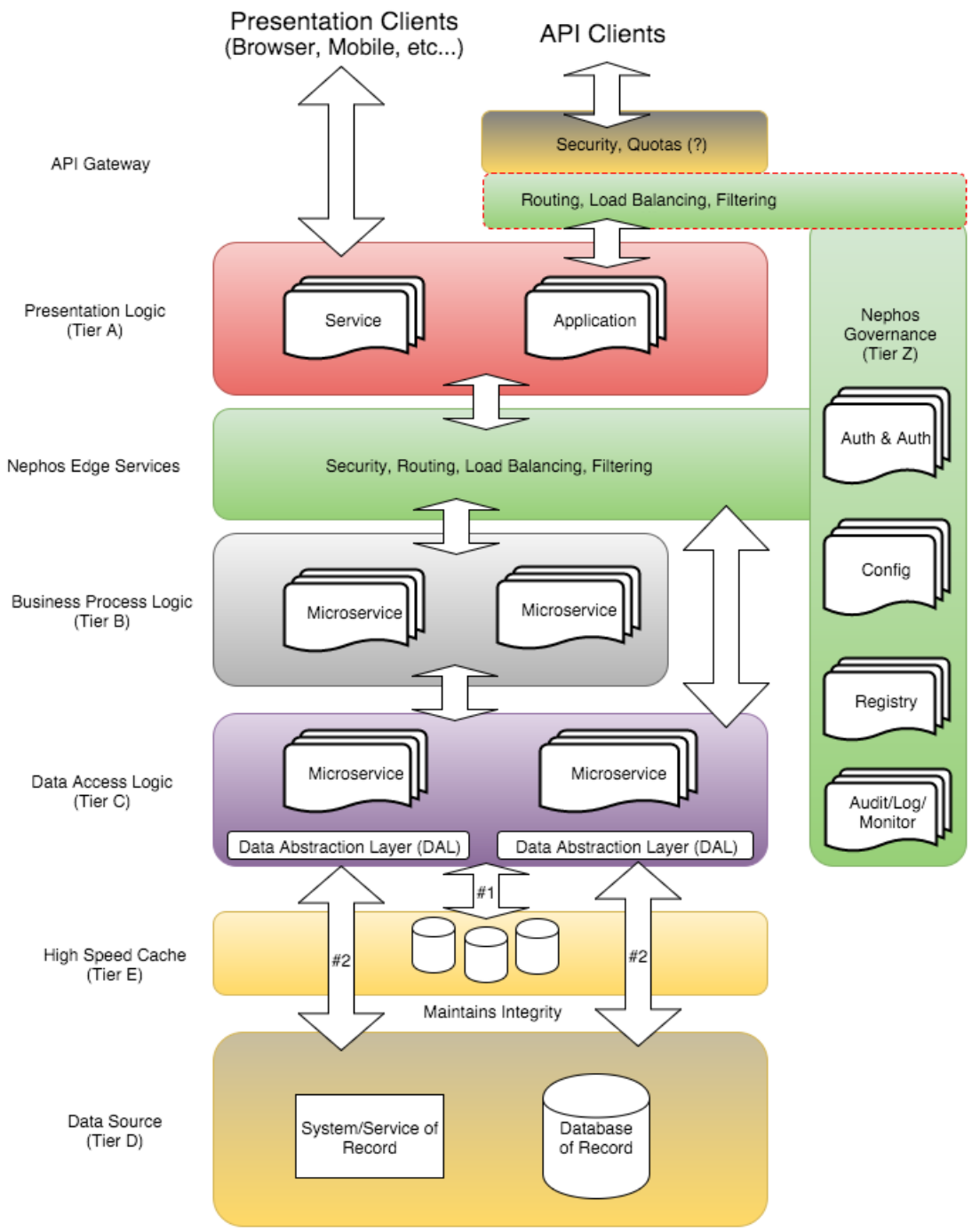
## 3.3 MICRO SERVICES

What are microservices? Martin Fowler discusses them at length in his blog pages but the Wikipedia definition is: In computing, microservices is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs. These services are small, highly decoupled and focus on doing a small task, facilitating a modular approach to system-building

When comparing microservices with traditional SOA, here are some distinctive differences

| Microservices | SOA |
|---|---|
| Many very small components | Fewer more sophisticated components |
| Business logic lives inside a single service domain | Business logic can live across multiple domains |
| Simple wire protocols, e.g. HTTP with XML or JSON | Enterprise Service BUS (ESB) like layers between services |
| API driven with SDKs/Clients | Middleware |

## 3.4 GLOBAL PLATFORM SERVICE TIERS




### 3.4.1 TIERS A – PRESENTATION TIER

Tier A translates B and C Tiers to the end customer, usually in the form of UI or application. Tier A is the Front End layer that makes calls into the Backend (Tier B) via the Service Registry through the Edge Services. Tier A is most likely to change frequently to suit evolving customer interests and business needs.

### 3.4.2 EDGE TIER

Tier A accesses the lower Tiers through an edge service. It has low single-digit latency and handles all the appropriate service authorization. Each call must go through these services. Below them, Tiers B and C may not require additional auth & auth. A simple service registry (Tier Z) provides location. These edge services can be thought of as the lightweight security Tier for access to the data center.

### 3.4.3 TIER B - BUSINESS SERVICES

Orchestration (getItems/getCart/search etc.,) Tier B is the orchestration Tier. It is also the highest level API of the backend. Tier A, the Front End calls into Tier B through the ESB. Here is the majority of business logic as Tier B takes data retrieved from the C Tier and packages it for the presentation Tier. The Tier B also joins data together from Tier C. All Tier B services support Grind Type in their calls. Tier B services are stateless.

### 3.4.4 TIER C – CRUD DATA SERVICES

The C Tier is the data service -- the lowest level of abstraction of the Tiers (getBasicSku, getPrice, getInventory etc.,) . The tier represents state for the system. The C Tier/DAL's job is to provide a layer of 'abstraction' above the physical sources residing in D-Layer and to aggregate/compose them together to provide logical data entities via API calls which in turn provide CRUD operations. Tier D components are legacy data sources and external 3P APIs. All Tier C services support Grind Type in their calls.

### 3.4.5 TIER D - DATA SOURCE OR DATABASE

Tier D is composed mostly of legacy databases and external systems – systems that are not controlled by the Platform. The platform provides the Data Access Layer that helps domain teams implement example, MongoDB (Asgard) and WCBE (DB2) live in Tier D. Magnus and OFL, both external systems. an external system, will be in Tier D.

### 3.4.6 TIER E - CACHING

The caching tier provides services to the C layer so that the 50ms response time can be achieved. The caching tier will have 2-3 caching technologies available to domains to use. Domains are not expected to create their own caching solutions.

### 3.4.7 TIER Z – CROSS CUTTING COMMON SERVICES

Tier Z provides is a vertical Tier that serves all Tiers. It provides light weight authentication and authorization (auth&authz), session, user tracking, service registry, governance, integration library, protocols, monitoring, and quality of service. Tier Z comprises the majority of the Platform

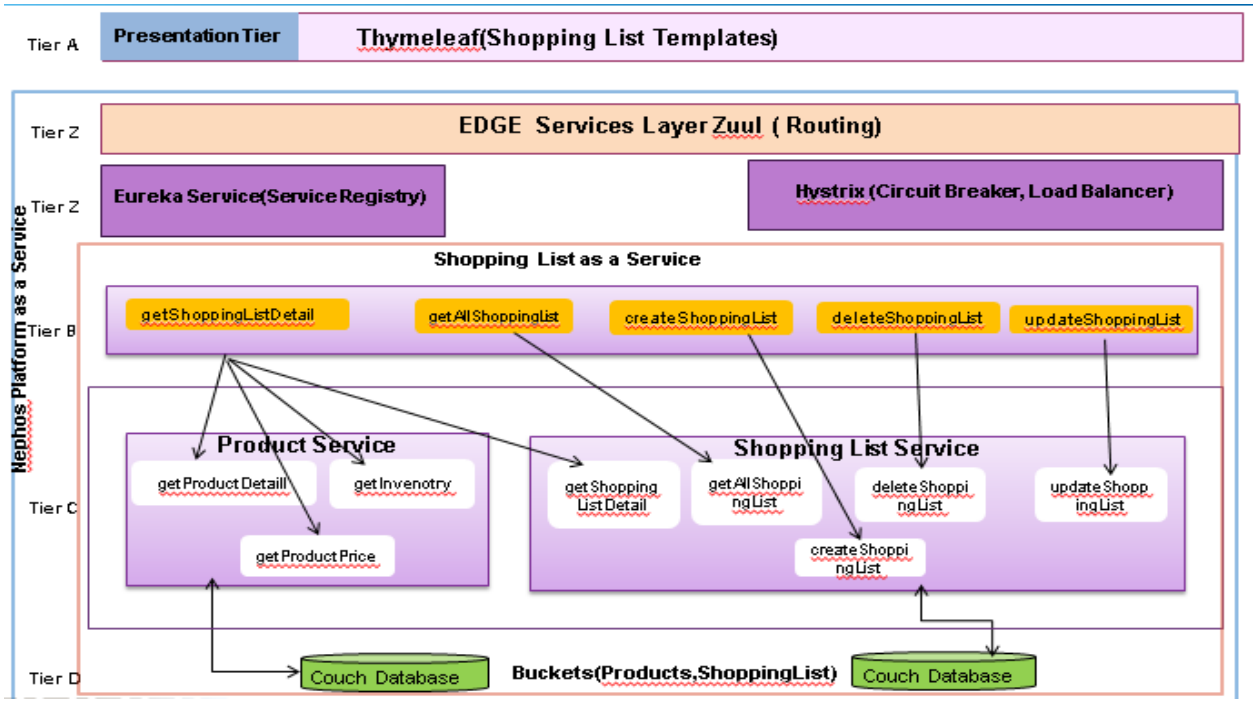
## 3.5 SHOPPING LIST POC ARCHITECTURE

PoC is aimed to touch base with the all technology stack of global platform and to design all the tiers in micro service architecture.

Following are the objectives of this PoC:

1. Realize an existing use case in the proposed Global Nephos Platform technology stack
2. Setup Environment for the POC (STS IDE, Spring Boot, Couchbase, Java 8)
3. Create couchbase buckets to store shopping list related documents.
4. Expose shopping list as service for the Create, Modify, View, Delete functionalities
5. Registering services in Eureka(Service Registry Client)
6. Create Edge service and Implementing routing, load balancer
7. Fault tolerance micro services using circuit breakers.
8. Create Thymeleaf templates for presentation layer which will call shopping list services through edge services.

Please go through the image to get the complete idea of the tiers and the functionalities implemented in PoC.



Each request from presentation layer hits Zuul, where we will do routing of the requests to respective service in Tier B. This service look up will make use of Eureka service. The calls from Tier B to Tier C will happen through Hystrix Command and feign client. Our services in Tier C will do transactions from the Couchbase DB which is Tier D.

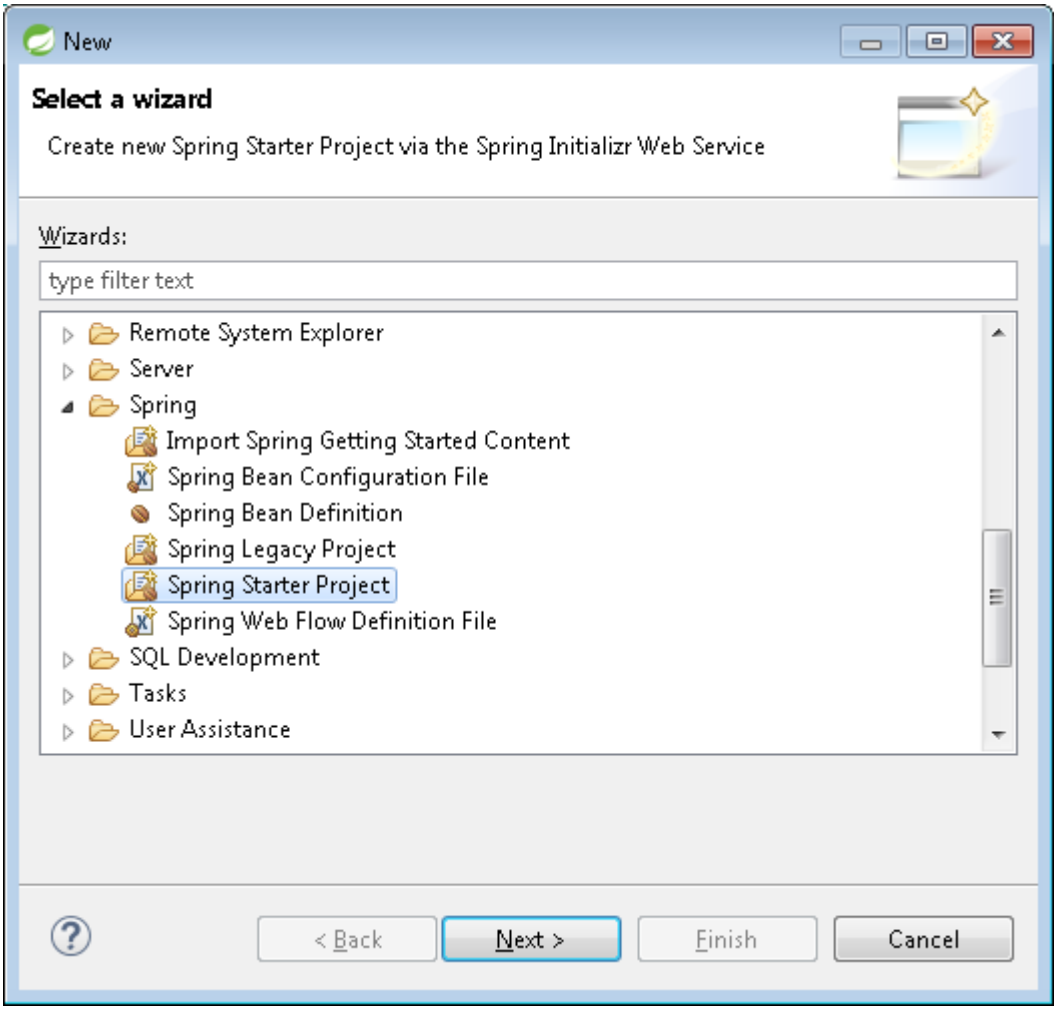
# 4 PROJECTS CONFIGURATUION

## 4.1 GIT REPO CONFIGURATION

Currently this is not in scope.

## 4.2 CREATE SPRING BOOT PROJECT

- Select new project option in STS and select Spring Starter Project.

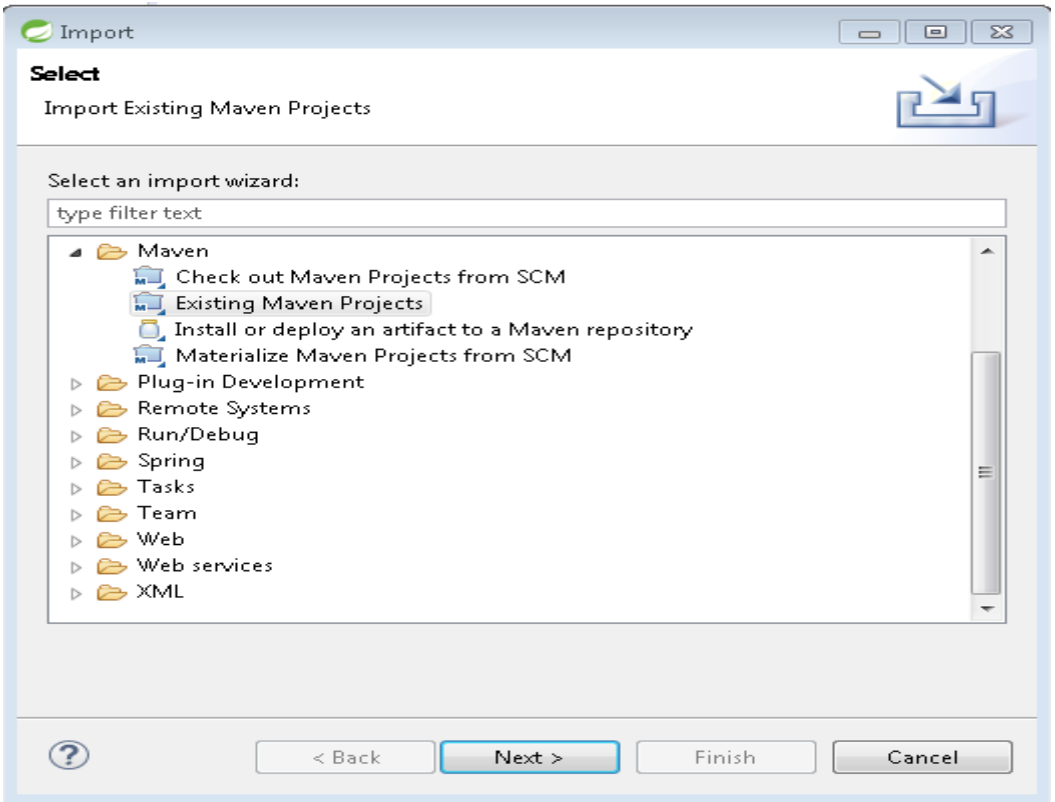



- Select type as maven and Boot version as 1.3.3, this version will be defaulted as when you download the STS.
- The project is created and you can add the source now.
- Also do check the pom.xml contains the dependency for the spring boot

```
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<java.version>1.8</java.version>
</properties>
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>
</dependencies>
```

## 4.3 IMPORT SPRING BOOT MAVEN PROJECT

- Select file and import and select maven option and select existing project



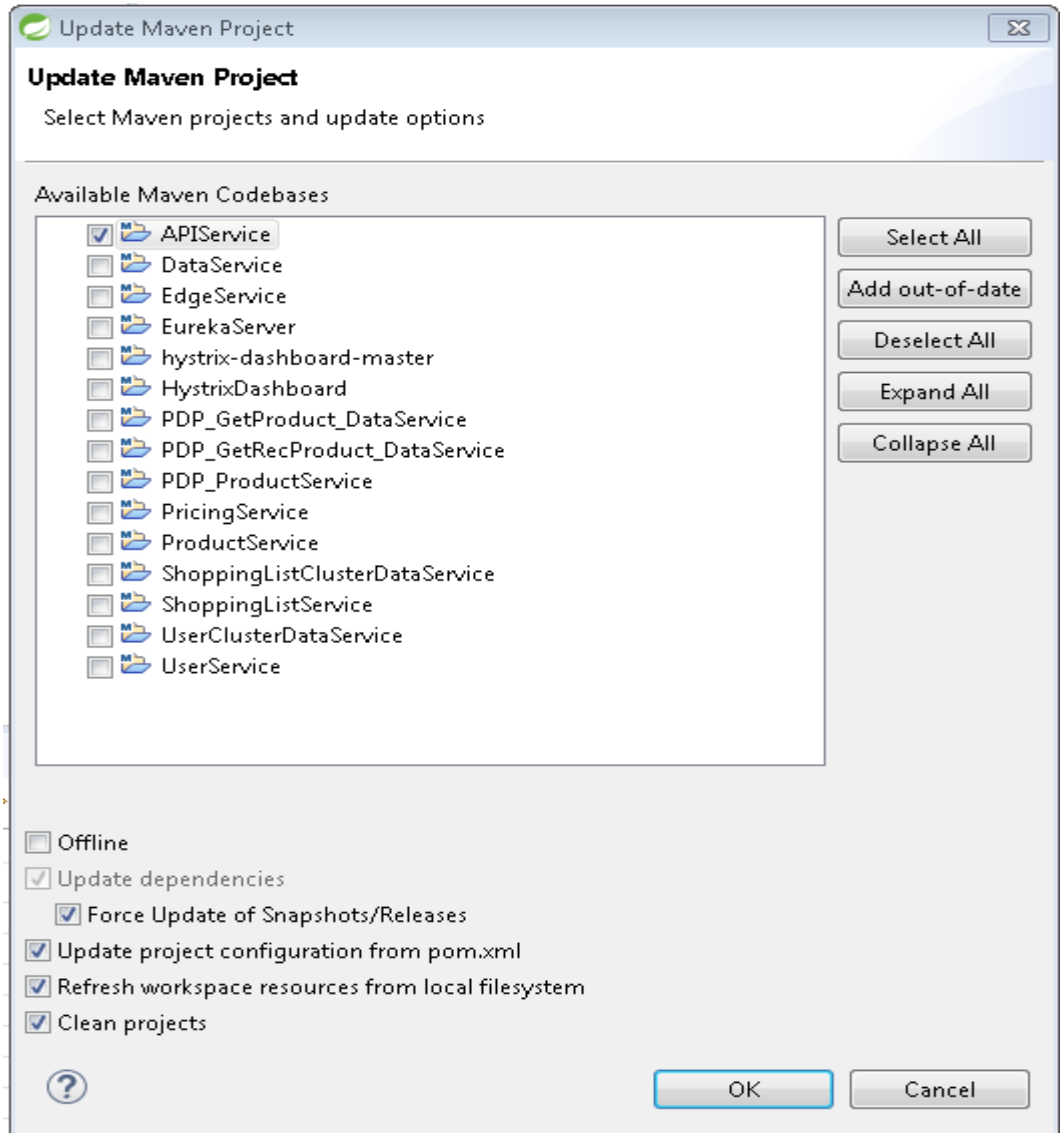

- Select the project location and click ok.
- The project gets loaded in STS and selects the project build to do a clean build.
- Then select the project right click and select maven and update project.
- Then select the project right click and select run as and click maven clean and then maven install.

- When the build is successful, start the server and give then run the application.
- As of now we are using eureka to get an end point for the service, do see the eureka section to get more inputs on that.
- Here when we import the project the eureka and all the server will be in place in the server view in STS IDE.

## 4.4 MAVEN PLUGIN DOWNLOADS

Resolve the dependency using maven update.

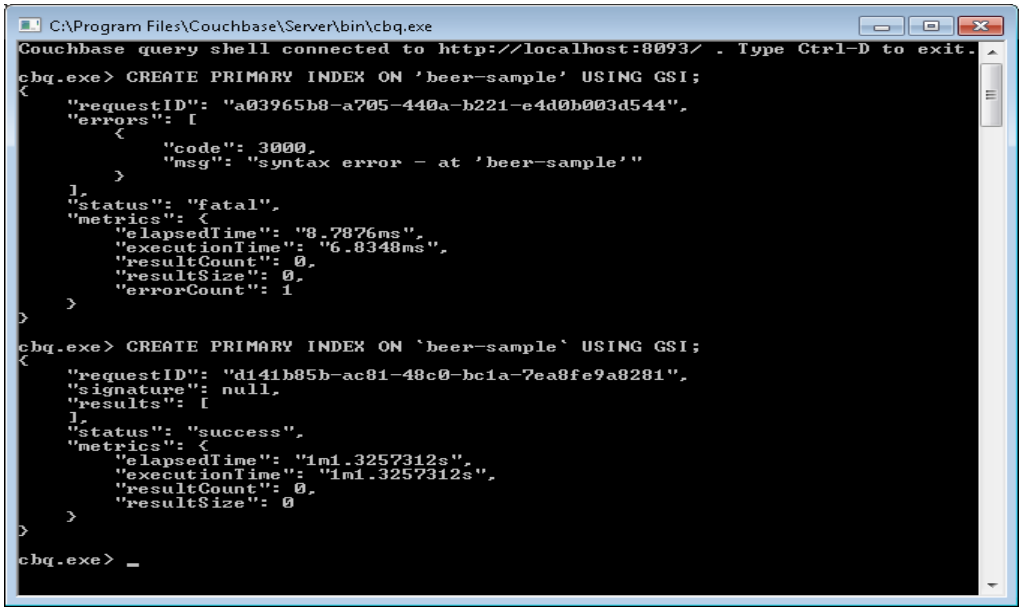
- Select the project right click and select maven and update project the dependency gets updated for the project that are required for use.



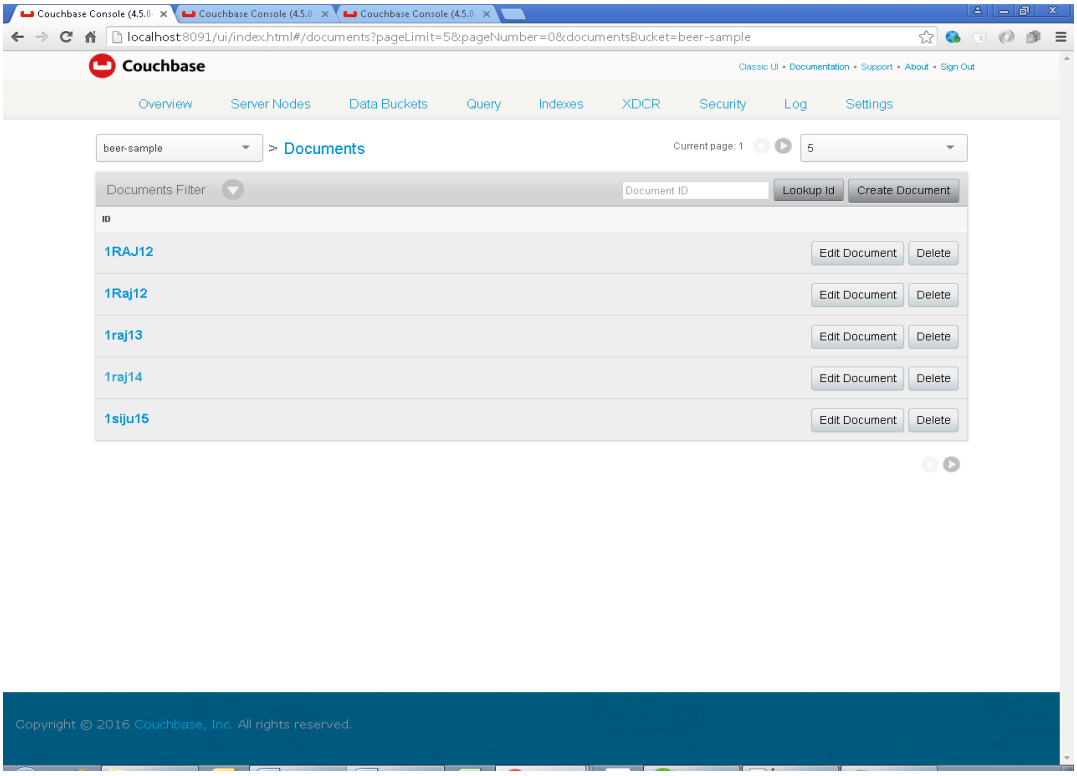
## 4.5 DATABASE SETUP AND MASTER DATA CREATION

**Create bucket and setup master data.**

- Bucket is a database, as the database we are using is a document database.
- Select data Buckets and click the create new data bucket and give the name of the bucket and select the ram space and click next/ok.
- Now the bucket is created now we need to do the data setup.
- After the creation is done we can do the index creation this can be done using GUI window or the couchbase query window as below.




- Then in GUI or console above select query and then run the query , after successful execution we can see the documents that we created as listed here.

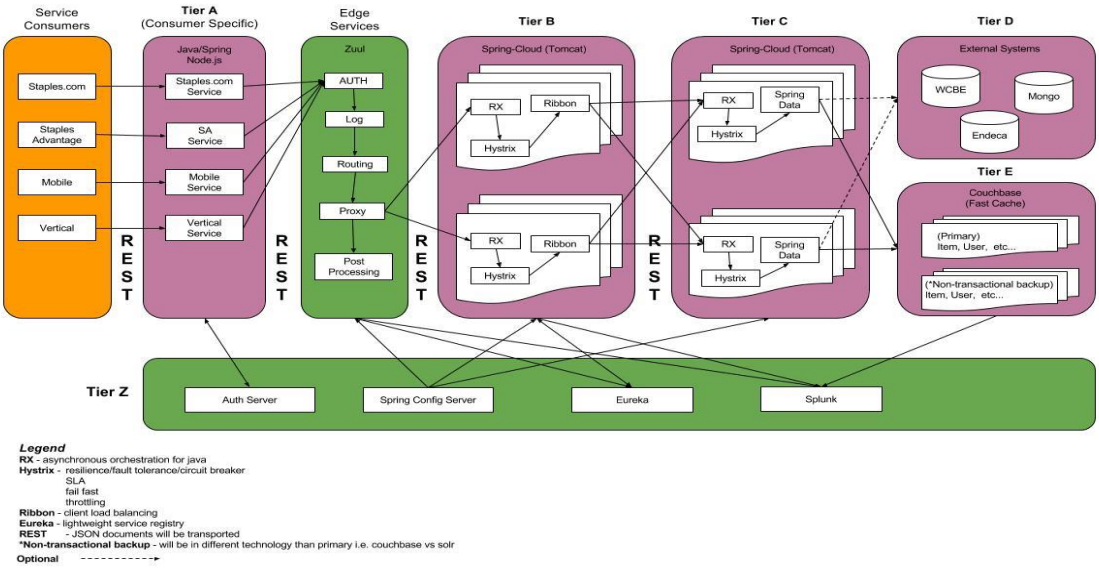



As per the data set up, adding the format of the query as this query is for COUCHBASE DOCUMENT DATABASE.

| **Example Create Bucket for - shoppinglist** | |
|---|---|
| Insert the user information in User document type | INSERT INTO ShoppingList (KEY, VALUE) VALUES ("user01", **{"documentType":"user",** "memberId":"A01", "firstName":"senthil", "lastName":"vel S", "passwd":"abc@123", "eMail":"senthil@gmail.com", "mobileNo":"91-9932322322"}) |
| Insert the product information in Product document type | INSERT INTO ShoppingList (KEY, VALUE) VALUES ("product01", **{"documentType":"product"**, "productId":"1001", "productName":"Pencil", "productShortDesc":"Pencil", "productLongDesc":"Pencil", "status":"A", "price":"9.00", "imagePath":"url", "categoryId":"1", "productCode":"", "brandName":"Test", "modelNumber":"Test", "partNumber":"1001", "rating":"3", "instockStatus":"Active"}) |
| Insert the shopping list information in ShoppingList document type | INSERT INTO ShoppingList (KEY, VALUE) VALUES ("SL001", **{"documentType":"shoppingList"**, "orderId":"001", "orderItemId":"001", "orderStatus":"Active", "product":[{"productId":"1001", "quantity":"1", "price":"10.0", "partName":"ABC"}, {"productId":"1002", "quantity":"1", "price":"10.0", "partName":"ABC"}, {"productId":"1003", "quantity":"1", "price":"10.0", "partName":"ABC"} ], "memberId":"A01", "shoppingListName":"ABC", "createdDate":"23-mar-2016", "shoppingListType":"Shared" }) |

# 5 UNDERSTANDING PROJECT STRUCTURE AND PACKAGE

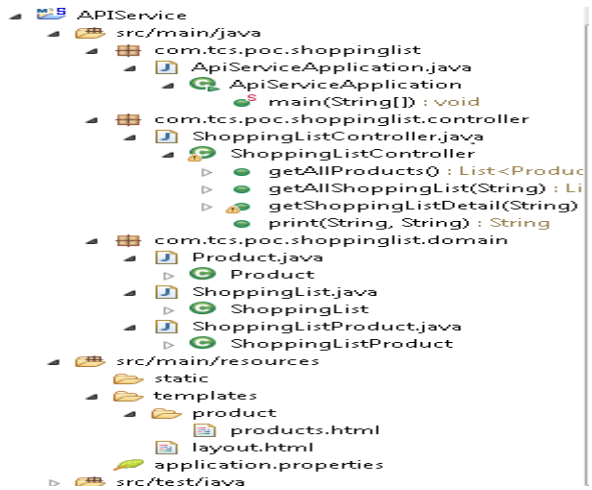
In this Section, we are describing the project folder structure and packages for eureka server (Tier Z), Edge Service (Tier Z), API Layer (Tier A), Business Layer (Tier B) and Data Layer (Tier C).




A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/main/test; and so on

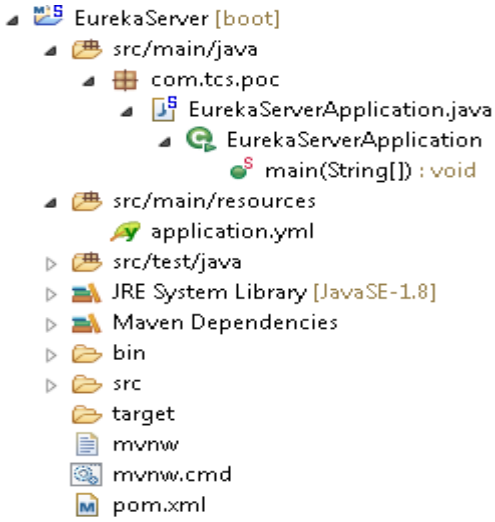
## 5.1 API SERVICE – TIER A

APIService is not boot application. API Service helps to retrieve the data from micro services and give the data to presentation layer. All the Front end code (HTML, angular JS) is integrated with Spring Rest application. Controller package contains all the rest method and domain package contains value object.



## 5.2 EUREKA SERVER – TIER Z

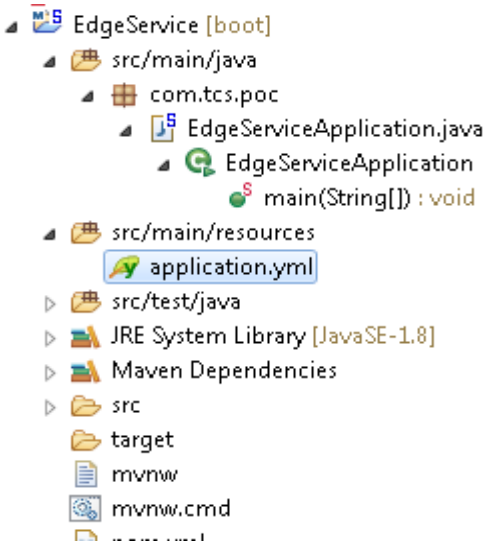
Eureka Server is a spring micro service registry. It contains all the registered services.




## 5.3 EDGE SERVICE – TIER Z

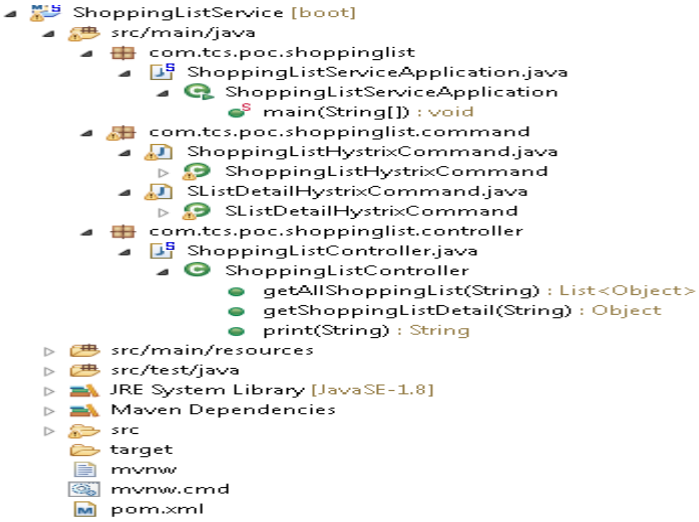
EdgeService basically referred as spring Zuul in spring Netflix.

In our Shopping List use case, we have implemented Zuul configurations for routing mechanism (application.yml).



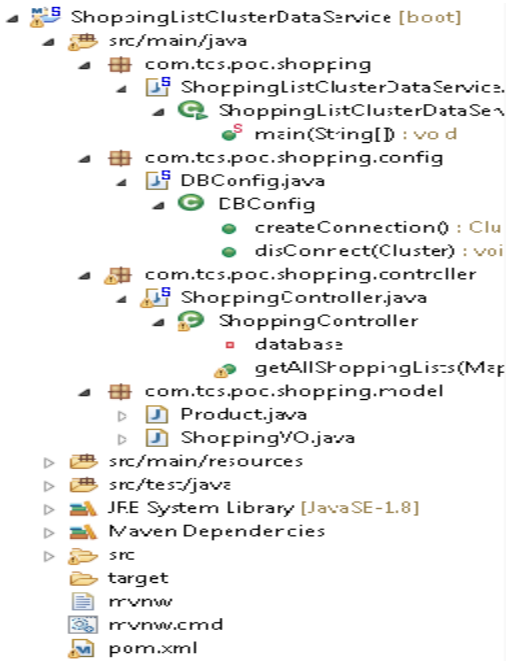
## 5.4 BUSINESS SERVICE – TIER B

ShoppingListService is a micro service. We should enable the fallback mechanism with help of HystrixCommand in each method in the service which was exposed. Controller package contains all the rest method and domain package contains value object. Command package contains Hystrix implementation.




## 5.5 DATA SERVICE – TIER C

ShoppingListDataService is a spring boot application. We are enabling cluster based database connection with couch base database to retrieve the data from them. Controller package contains all the rest method and model package contains value object. Config folder contains all database related configuration.



# 6 SPRING NETFLIX COMPONENT OVERVIEW

Project provides Netflix OSS integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other spring programming model idioms. With a few simple annotations you can quickly enable and configure the common patterns inside your application and build large distributed systems with battle-tested Netflix components. The patterns provided include Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client Side Load Balancing (Ribbon).

## 6.1 EUREKA

Eureka is a REST (Representational State Transfer) based service that is primarily used in the AWS cloud for locating services for the purpose of load balancing and fail over of middle-tier servers.

Eureka helps you find the information about the services you want to communicate with but does not impose any restrictions on the protocol method of communication.

Eureka is the Netflix Service Discovery Server and Client. The server can be configured and deployed to be highly available, with each server replicating state about the registered services to the others

## 6.2 ZUUL

Netflix Zuul - Edge Server Zuul is (of course) our gatekeeper to the outside world, not allowing any unauthorized external requests pass through. Zuul also provides a well-known entry point to the micro services in the system landscape.

Using dynamically allocated ports is convenient to avoid port conflicts and to minimize administration but it makes it of course harder for any given service consumer. Zuul uses Ribbon to look-up available services and routes the external request to an appropriate service instance.

In this blog post we will only use Zuul to provide a well-known entry point, leaving the security aspects for coming blog posts.

## 6.3 HYSTRIX

In a distributed environment, inevitably some of the many service dependencies will fail. Hystrix is a library that helps you control the interactions between these distributed services by adding latency tolerance and fault tolerance logic. Hystrix does this by isolating points of access between the services, stopping cascading failures across them, and providing fallback options, all of which improve your system’s overall resiliency.

## 6.4 FEIGN CLIENT

It makes writing web service clients easier. Feign connect your code to HTTP API'S with minimal overhead. Maps to the REST API methods that are exposed by a service. Customizable decoders and error handling. Feign works by processing annotations into a templatized request.

## 6.5 CONFIG SERVER

Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system. With the Config Server you have a central place to manage external properties for applications across all environments. Spring Cloud Config consists of Client and Server. An application moves through the deployment pipeline from develop to test and into production, we can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate.

## 6.6 ACTUATOR

Spring Boot Actuator is a sub-project of Spring Boot. It adds several production grade services to your application with little effort on your part. The SpringApplication.run() command knows how to launch the web application. All you need to do is run the below code

```
package hello;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class HelloWorldConfiguration {
public static void main(String[] args) {
SpringApplication.run(HelloWorldConfiguration.class, args); } }
```

## 6.7 TURBINE

Turbine is a tool for aggregating streams of Server-Sent Event (SSE) JSON data into a single stream.The targeted use case is metrics streams from instances in an SOA being aggregated for dashboards.

For example, Netflix uses Hystrix which has a real time dashboard that uses Turbine to aggregate data from 100s or 1000s of machines.

# 7 SPRING BOOT APPLICATION – SHOPPING LIST USE CASE

In this section, will discuss more on spring boot application creation, configuration, method level code, run the spring boot application.

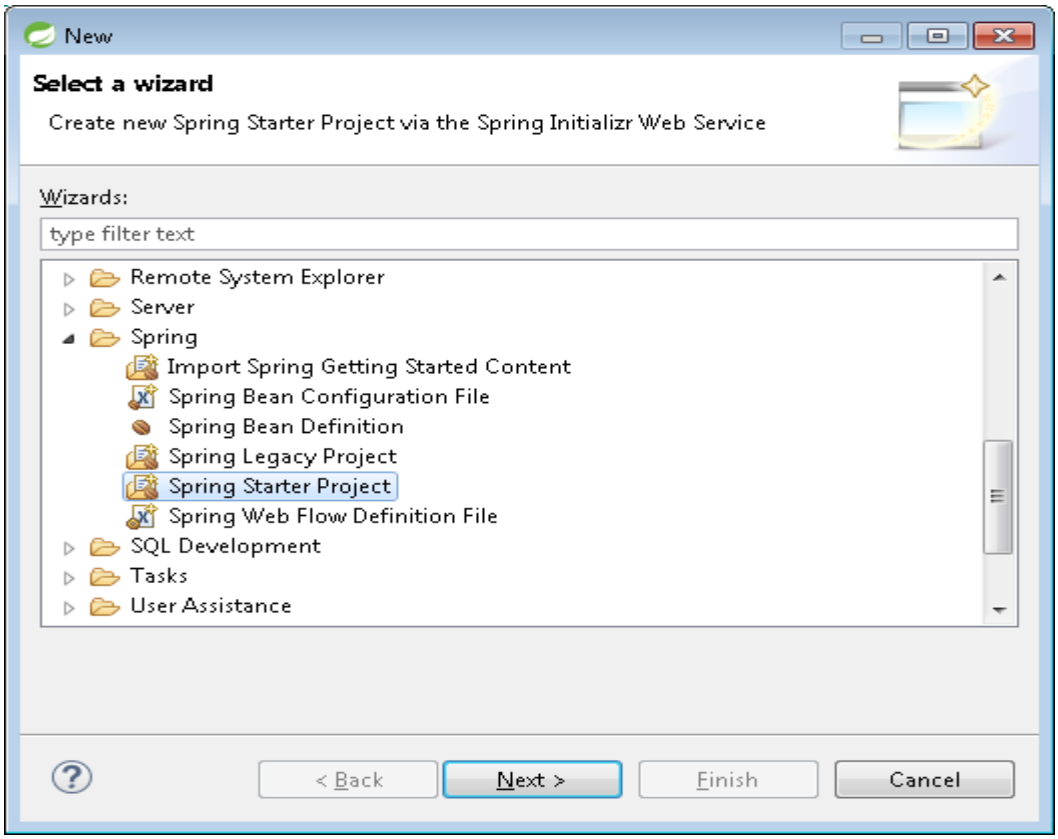
## 7.1 EUREKA SERVER

Eureka Server application is boot application. Eureka is a service discovery server and client. It is REST (Representational State Transfer) based service that is primarily used in the AWS cloud for locating services for the purpose of load balancing and failover of middle-tier servers.

### 7.1.1 APPLICATION CREATION

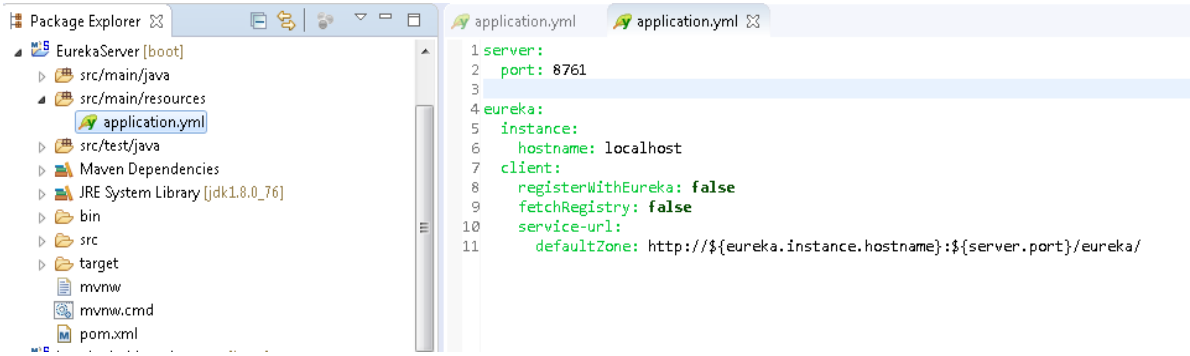
In STS, Create new project for Eureka Server and select Spring Starter Project and specify the package structure as com.tcs.poc.shoppinglist.

In next screen, select the boot version 1.3.3 and Dependencies, we need selet "**Eureka Server**" and click Finish.



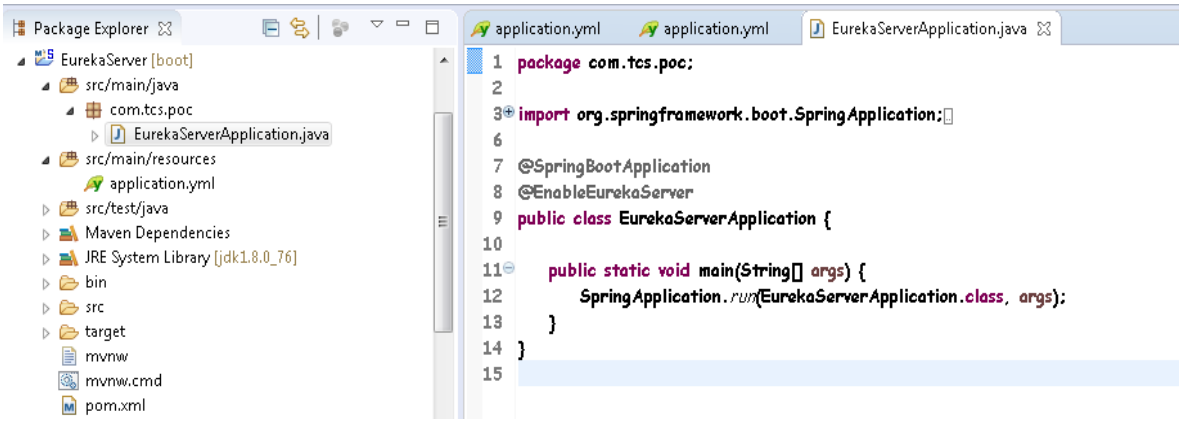

### 7.1.2 EUREKA SERVER APPLICATION CONFIGURATION

In Eureka server project, create application.yml file and add server port and application name.




## 7.1.3 METHOD LEVEL CONFIGURATION

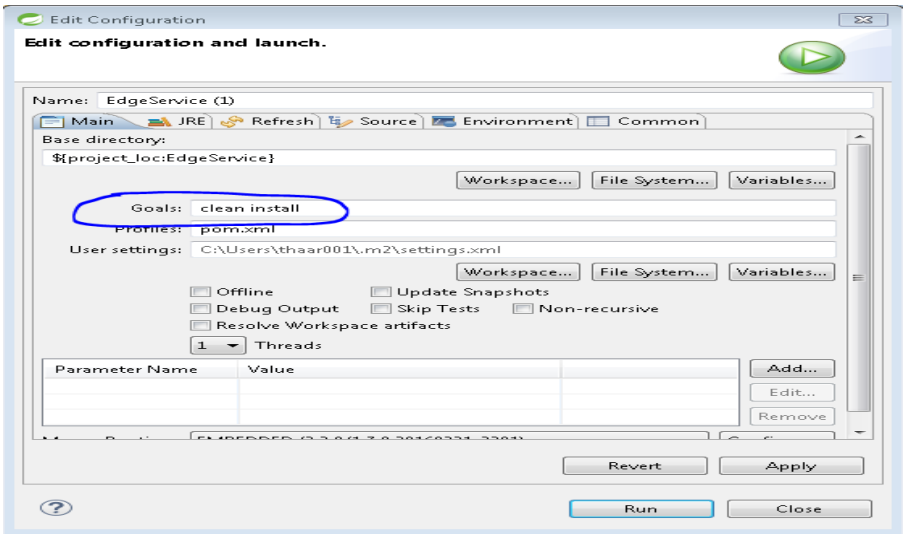
In Eureka Server application, we need to enable the eureka server @EnableEurekaServer.

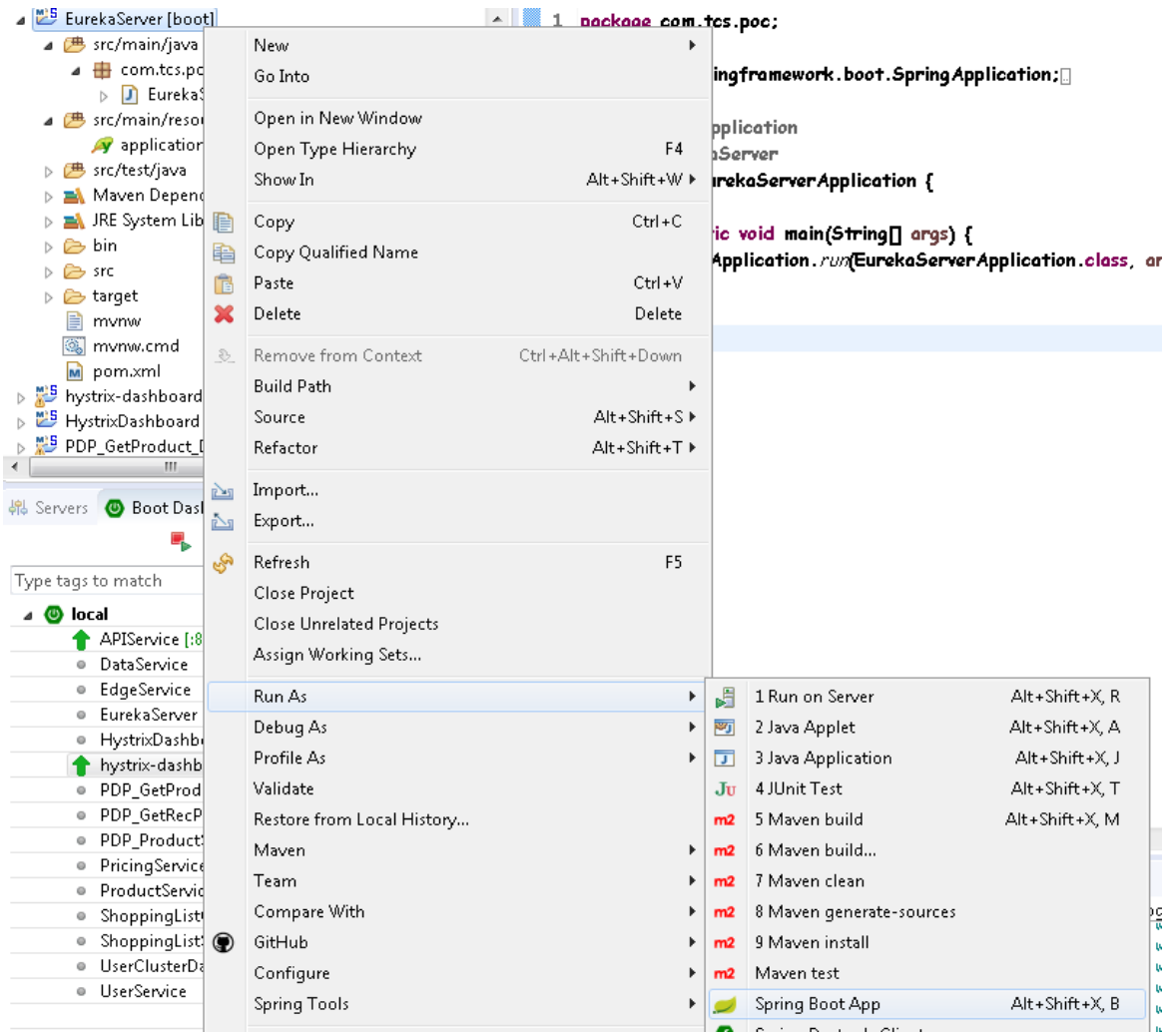



## 7.1.4 RUN APPLICATION

Before runing the application, clean and install the application using Maven build. Then Run the application using Spring boot.

**Maven Build:**






## 7.1.5 EUREKA SERVER DASHBOARD

**To access the Eureka Dashboard**

http://localhost:8761

## 7.2 EDGE SERVER - ZUUL

Edge Server application is boot application. Edge Server Zuul is (of course) our gatekeeper to the outside world, not allowing any unauthorized external requests pass through. Zuul also provides a well-known entry point to the micro services in the system landscape.

**Zuul**

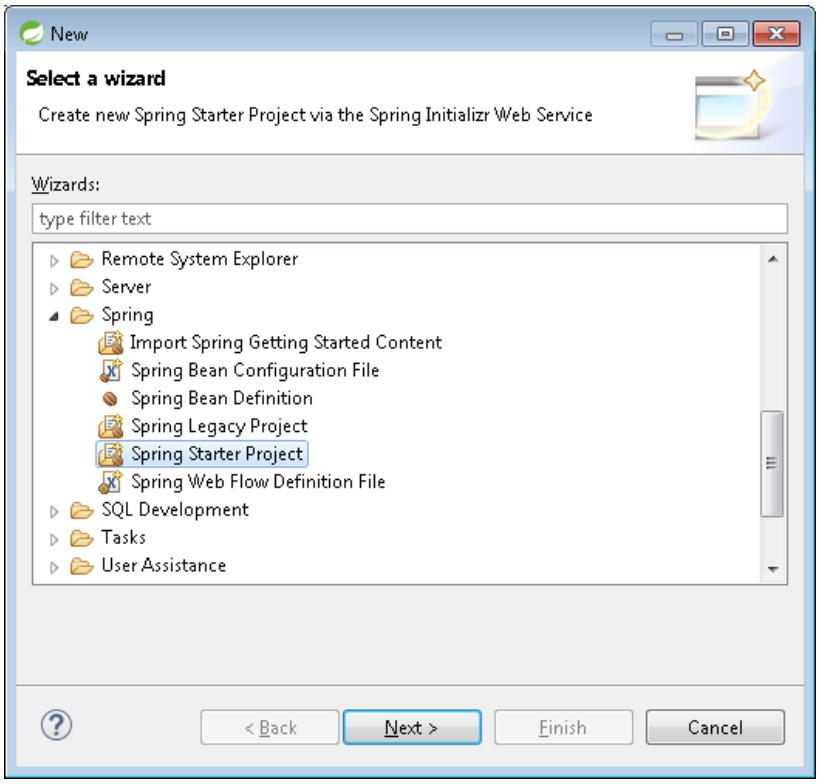
- ✓ Edge server providing routing, filtering and proxying
- ✓ @EnableZuulProxy
- ✓ Integrates with Eureka for service looking
- ✓ Uses Ribbon for client side load balancing

In this POC we have used routing alone,.

### 7.2.1 APPLICATION CREATION

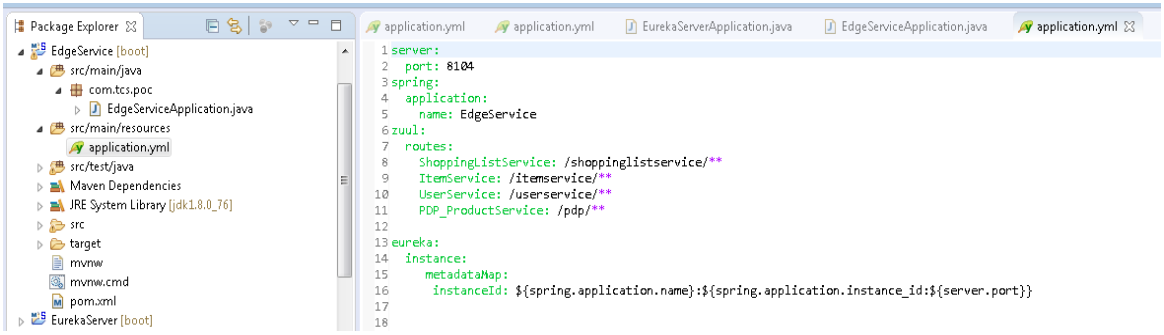
In STS, Create new project for Zuul Server and select Spring Starter Project and specify the package structure as com.tcs.poc.shoppinglist.

In next screen, select the boot version 1.3.3 and Dependencies, we need selet **"Eureka Discovery"** and **"Zuul"** click Finish.



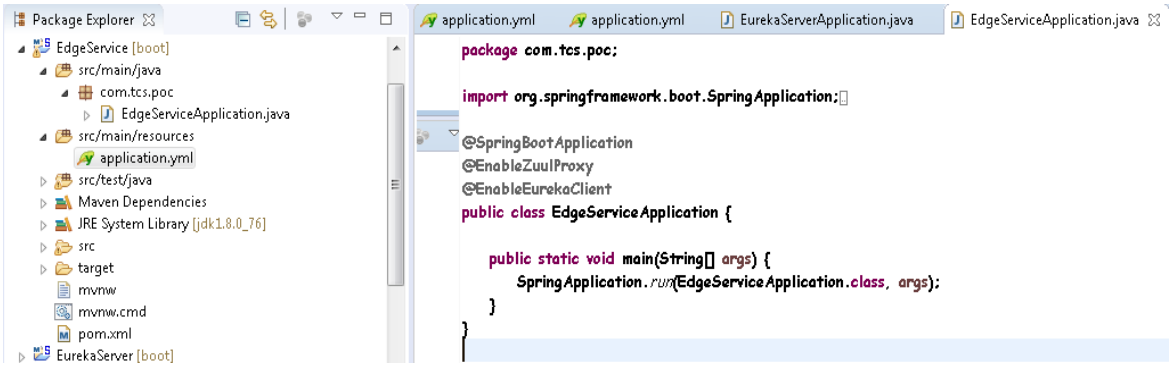
## 7.2.2 EDGE SERVER APPLICATION CONFIGURATION

In Edge server project, create application.yml file and add server port and application name.



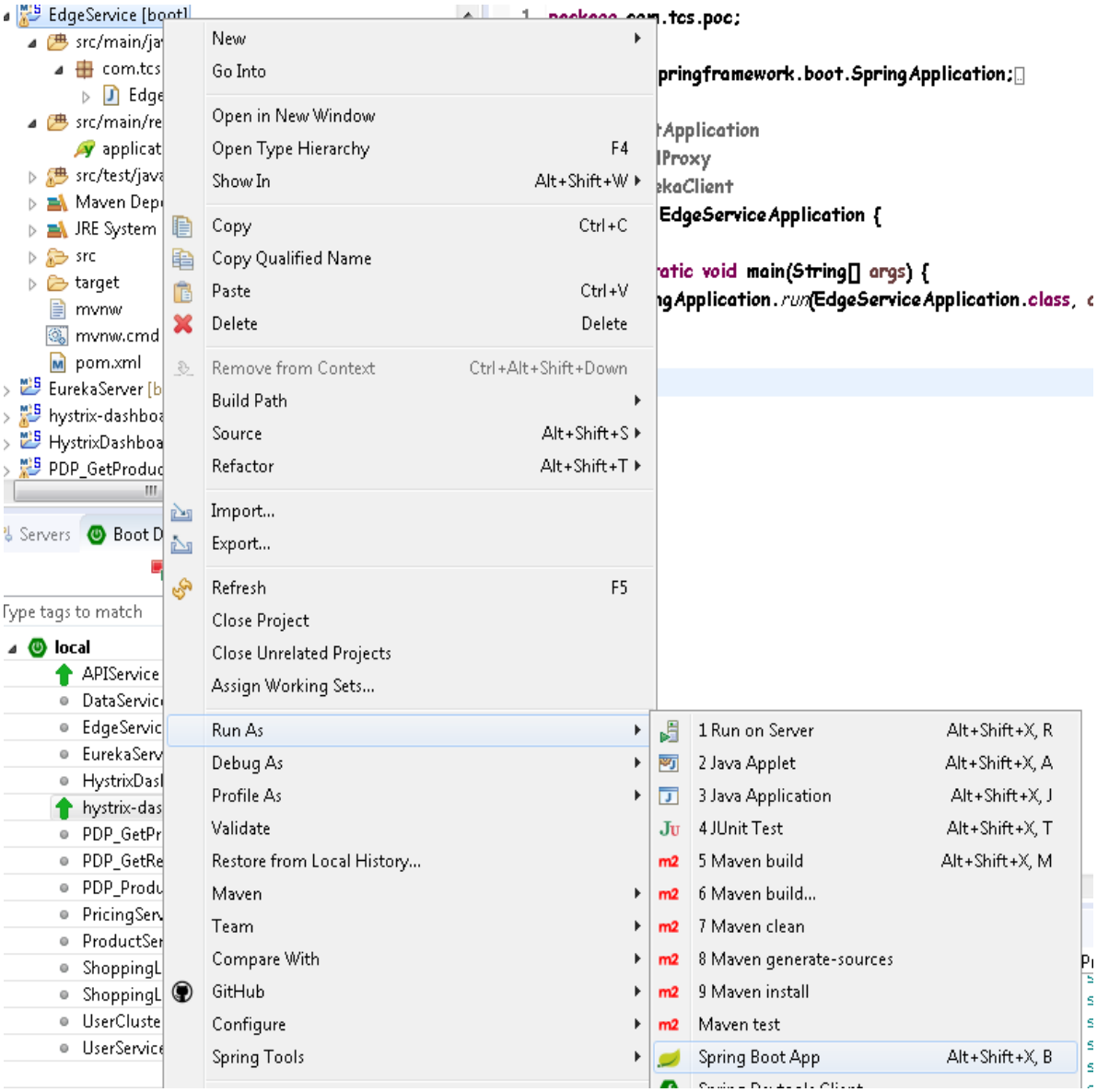

## 7.2.3 METHOD LEVEL CONFIGURATION

In Edge Server application, we need to enable Zuul configuration using annotations @EnableEurekaClient & @EnableZuulProxy.



## 7.2.4 RUN ZUUL APPLICATION

Before runing the application, clean and install the application using Maven build. Then Run the application using Spring boot.

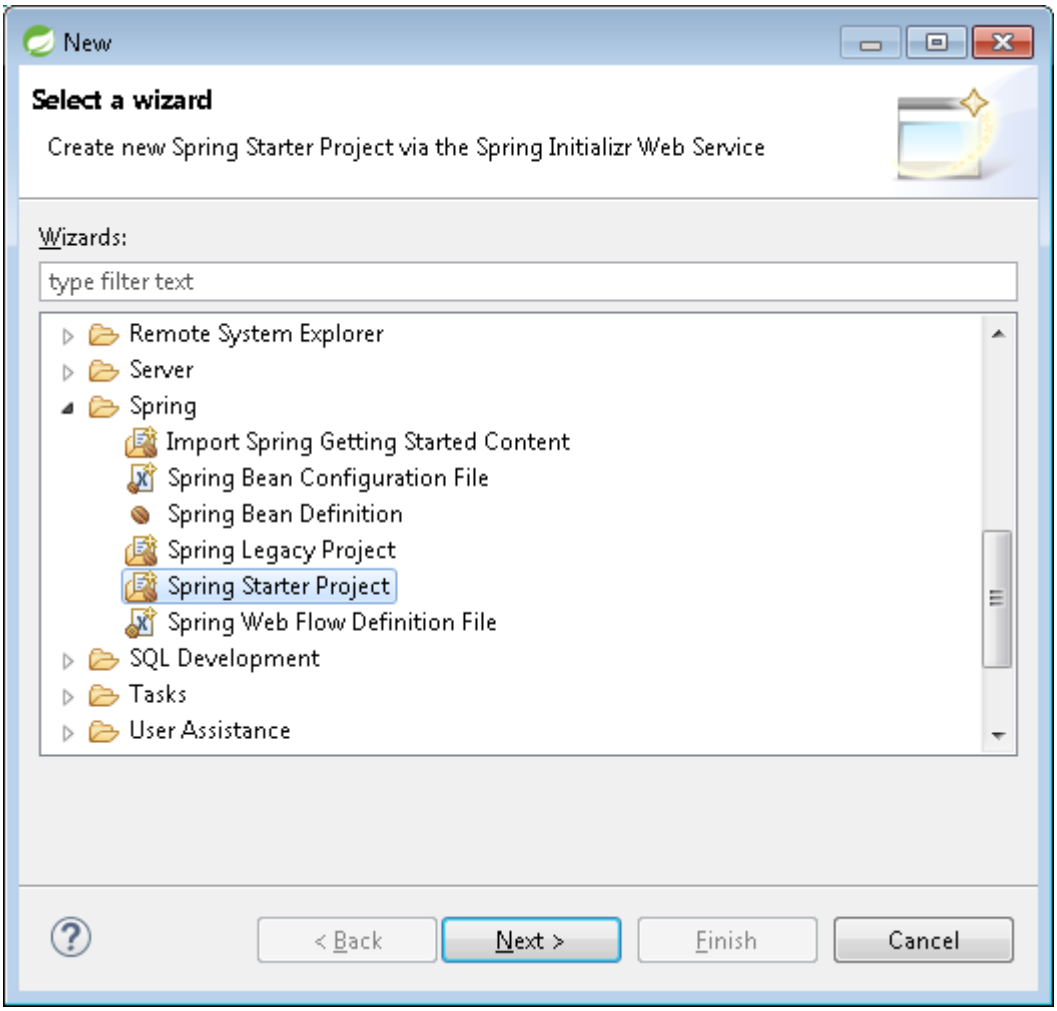


## 7.3 API SERVICE

APIService is not boot application. API Service helps to retrieve the data from micro services and give the data to presentation layer.

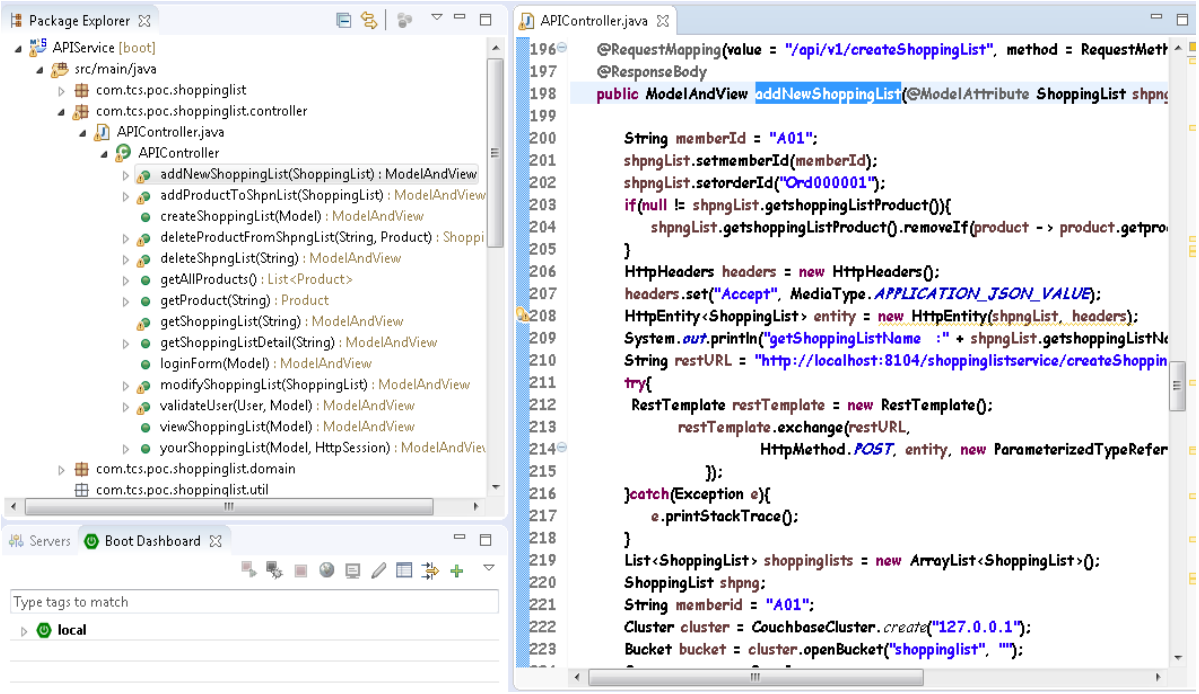
### 7.3.1 APPLICATION CREATION

In STS, Create new project and select Spring Starter Project and specify the package structure as com.tcs.poc.shoppinglist.



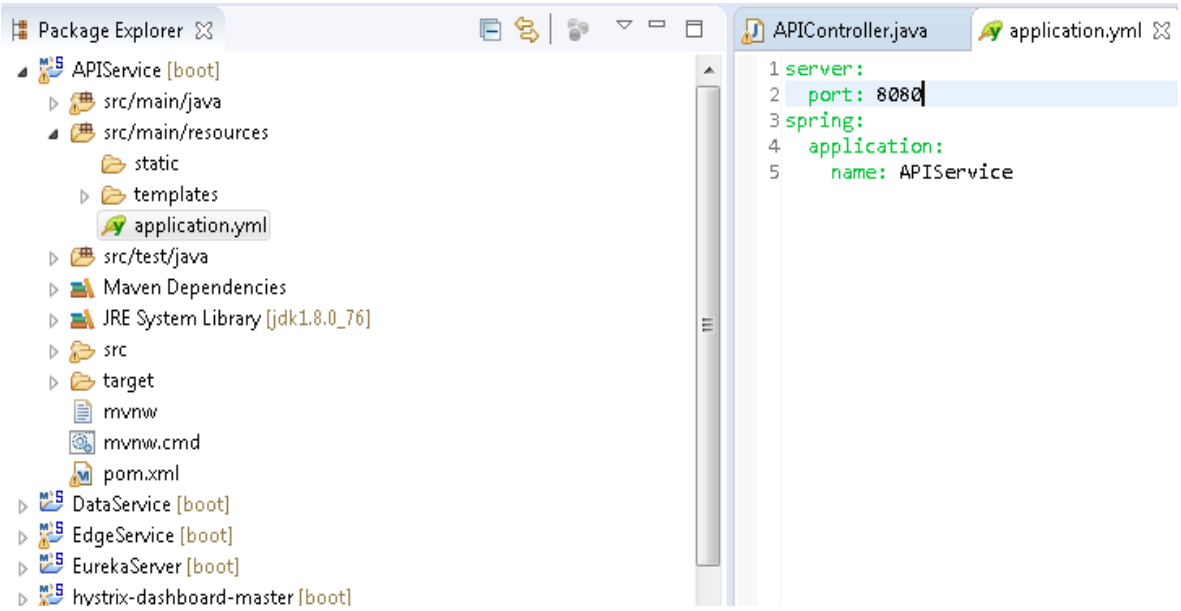
## 7.3.2 CREATE REST CONTROLLER

In API Service Project, create rest controller in com.tcs.poc.shoppinglist.controller package and create all the method with is needed for you shopping list use case.




## 7.3.3 API SERVICE APPLICATION CONFIGURATION

In API Service Project, create application.yml file and add server port and application name.

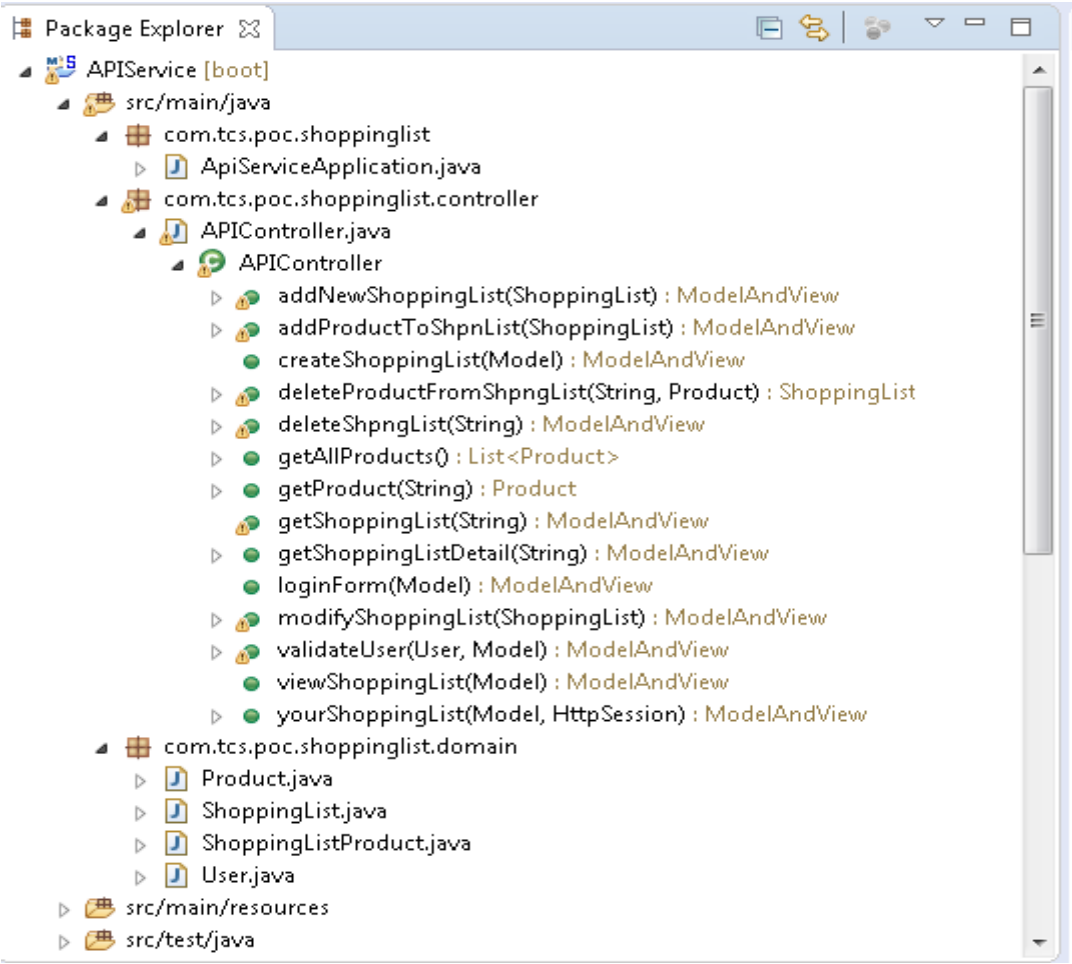


### 7.3.4 METHOD LEVEL DESIGN

In shopping list PoC, created user, product, shoppinglist and shopling list product domain, which wil be used to set and get the values.

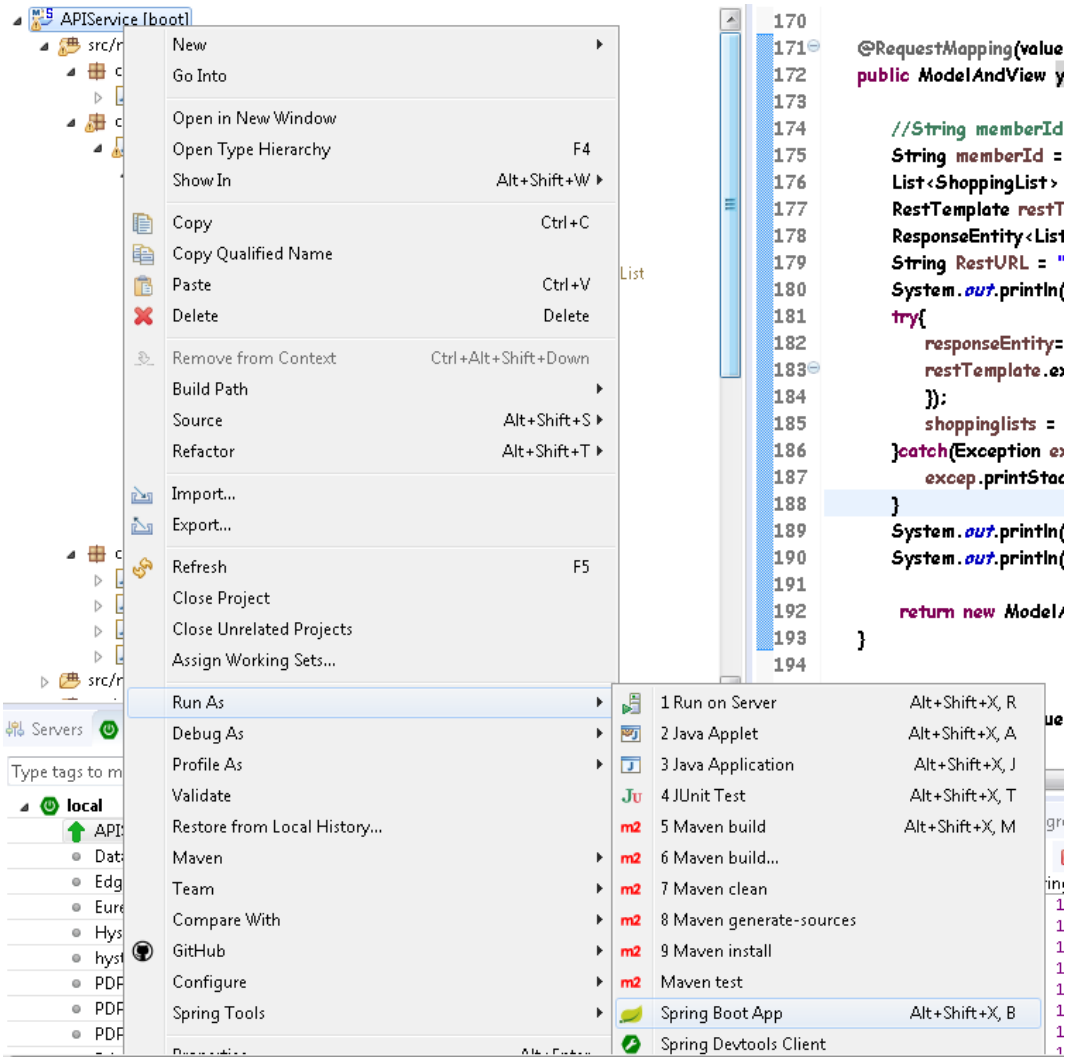
In the controller class had been created which methods were used in shopping list use case. rest of methods were used by front end development team to render the html template.

**addNewShoppingList** – create new shoppinglist
**addProductToShpnList** – adding new product to shopping list
**deleteProductFromShpngList** – delete product from shopping list
**deleteShpngList –** delete the shopping list
**getAllProducts** – get all the products
**getProduct** – get product detail.
**getShoppingList** – get all the shopping list for login user.
**getShoppingListDetail** – get the shopping list detail
**modifyShoppingList** – modify the shopping list
**validateUser** – validate the user



## 7.3.5 RUN APPLICATION

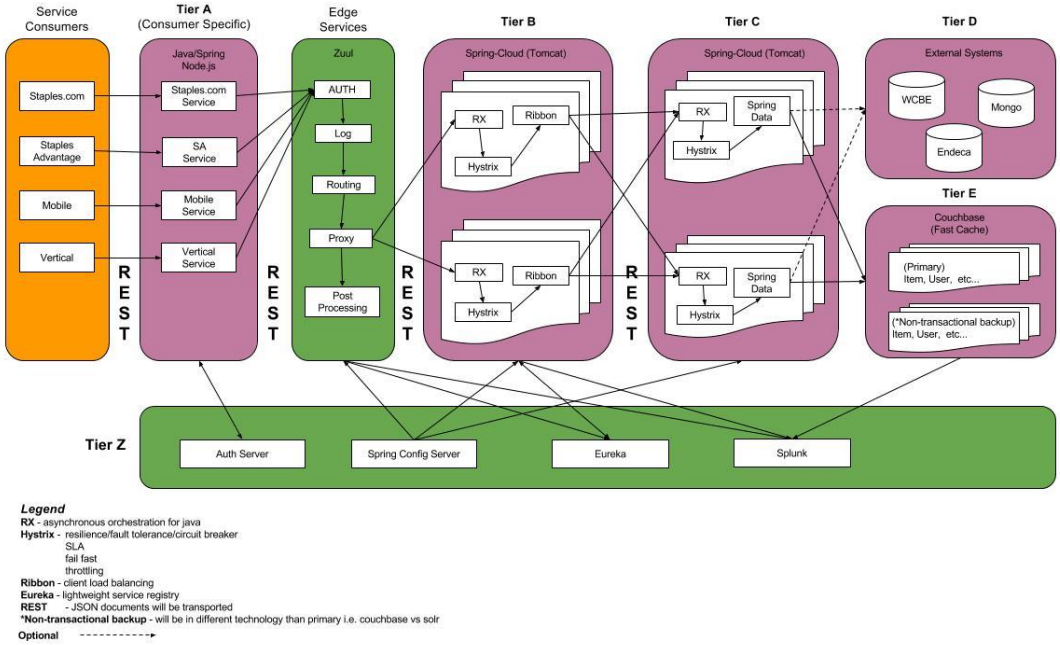
Before runing the application, clean and install the application using Maven build. Then Run the application using Spring boot.



## 7.4 MICRO SERVICE – TIER B

Shopping list micro service application was developed by using spring Netflix.

Shopping list Service performs all business logic, rules, validation and data retrieval.




**Note:** Business service (Tier-B) only interacts with Tier-C.

Implemented **hystrix command** in every business service methods.

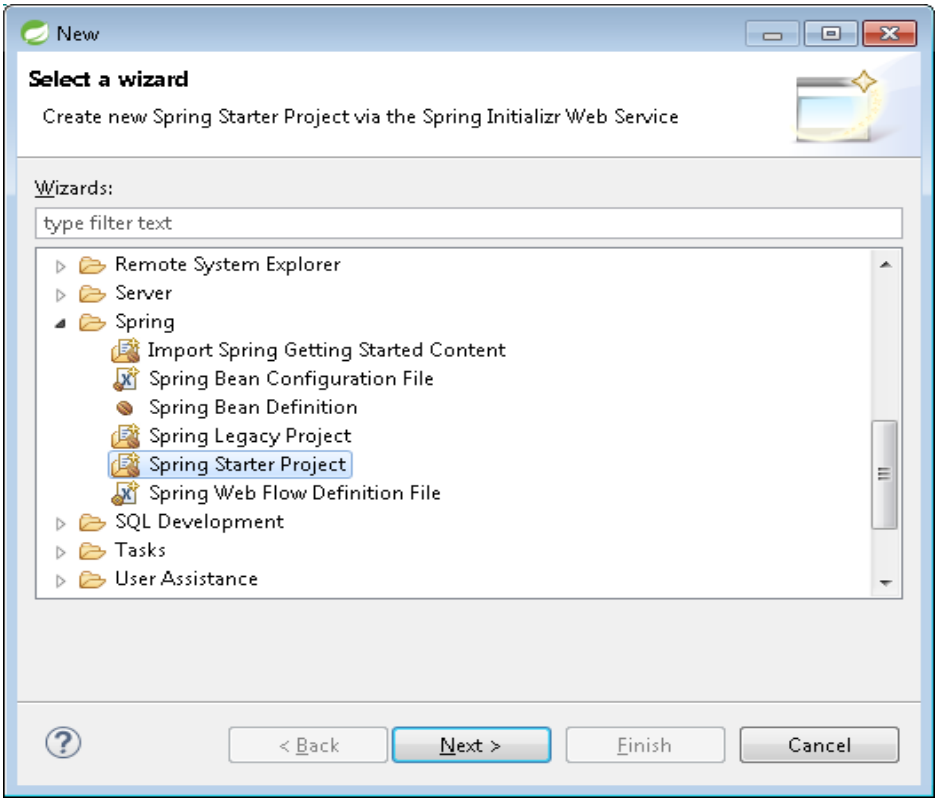
In aggregated service, we have used **feign client** which helps to interact with other micro services, other 3rd party systems etc.

Apart from shopping list service, we have two more services deals product& user services which help to support shopping list functionalities.

### 7.4.1 APPLICATION CREATION

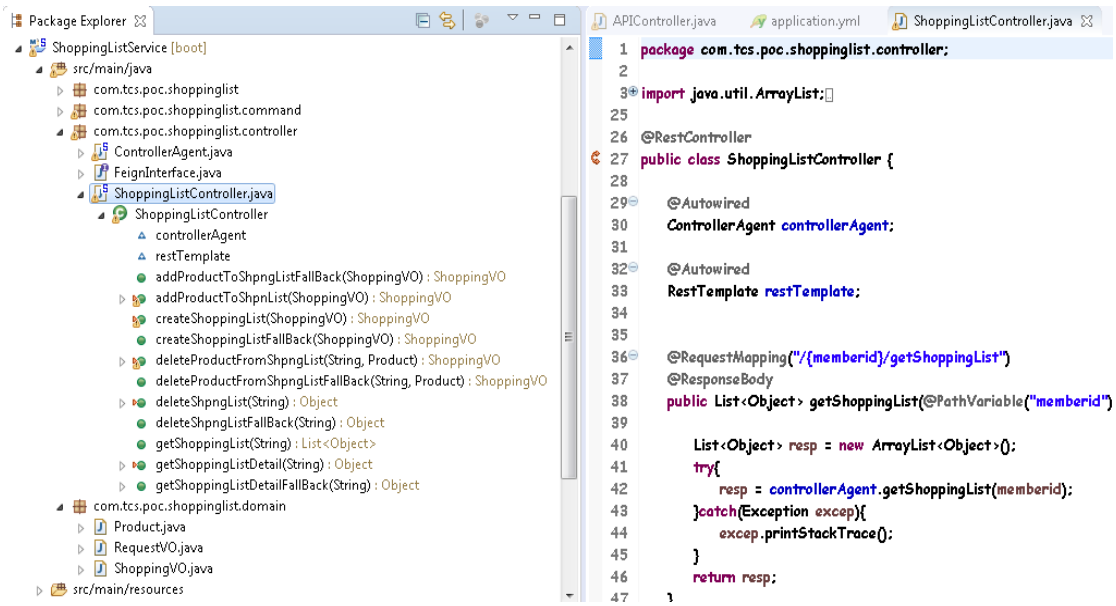
In STS, Create new project for shoppinglistservice and select Spring Starter Project and specify the package structure as com.tcs.poc.shoppinglist.

In next screen, select the boot version 1.3.3 and Dependencies, we need selet **"Eureka Discovery" and "Hystrixl"** click **Finish**.



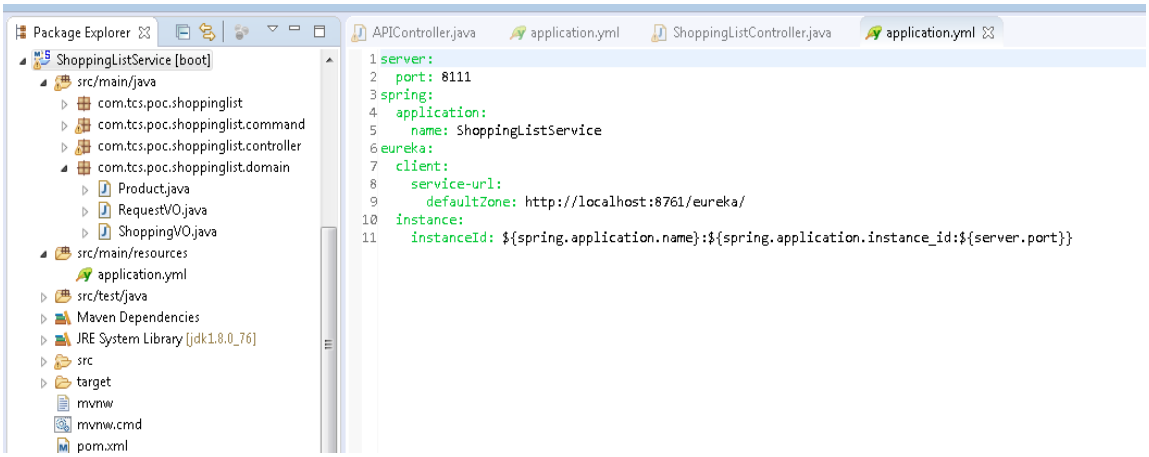

## 7.4.2 CREATE REST CONTROLLER

In Shopping list microservice Service Project, create a rest controller under com.tcs.poc.shoppinglist.controller package and create all methods required forshopping list functionality.



## 7.4.3 SHOPPING LIST SERVICE APPLICATION CONFIGURATION

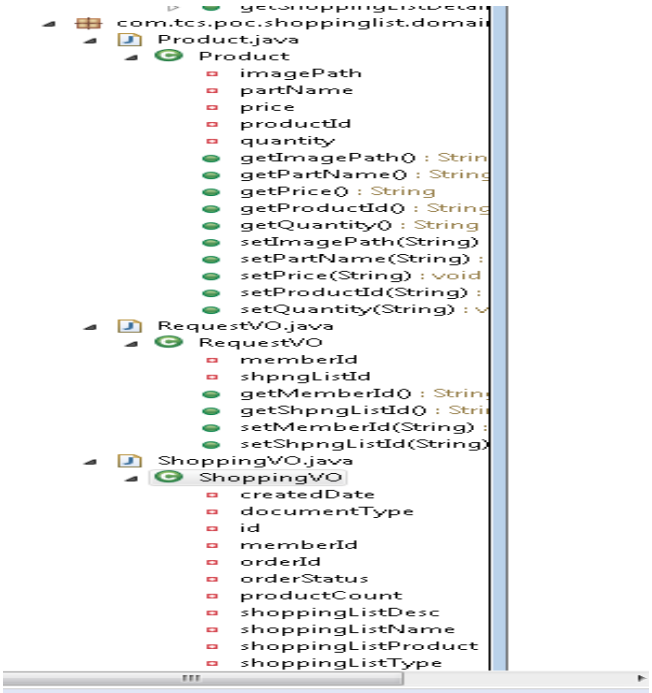
In Shopping list Service Project, create application.yml file with add server port and **application name and eureka** server path and zone details as well.




## 7.4.4 METHOD LEVEL DESIGN

In shopping list micro service, we have used **shoppinglist** and **product** domain, which wil be used to set and get the values shopping list details.

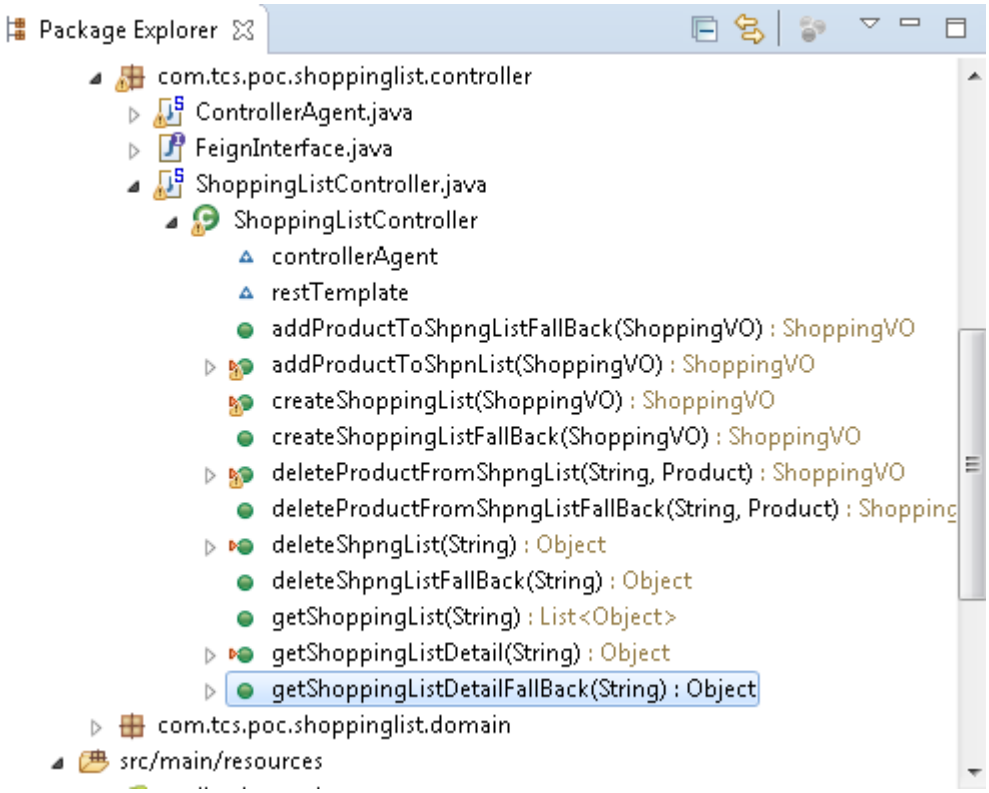
i.e request and response vo



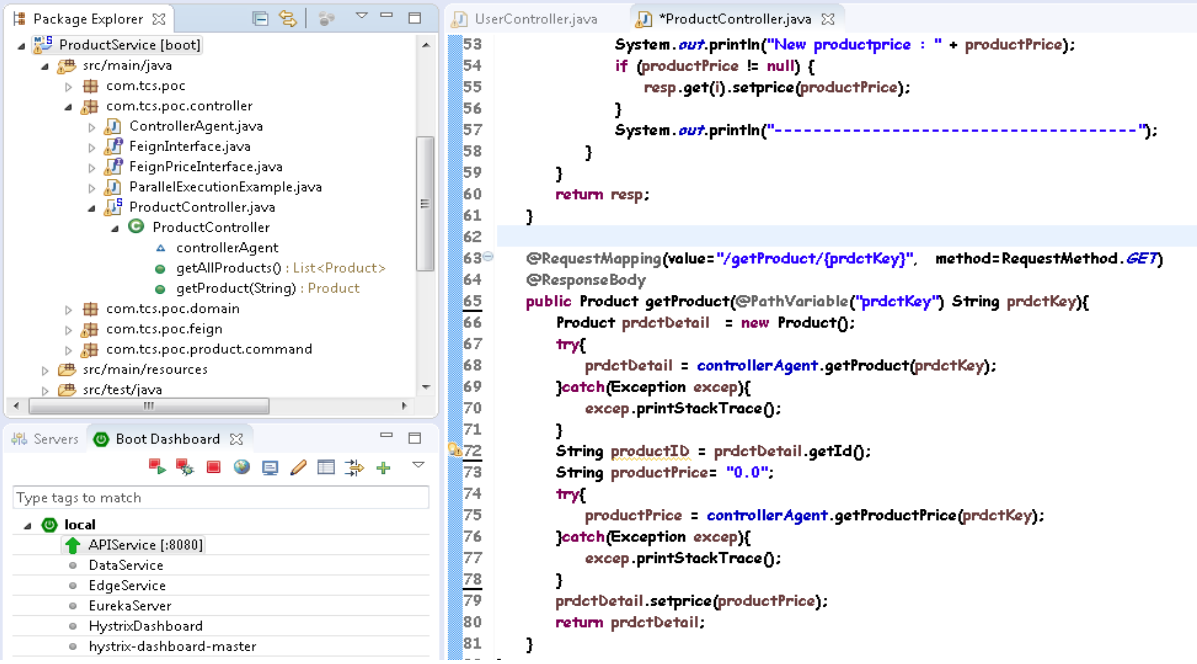

In the controller class deals shopping list functionality related methods as follows,

**addNewShoppingList** – create new shoppinglist
**addProductToShpnList** – adding new product to shopping list
**deleteProductFromShpngList** – delete product from shopping list

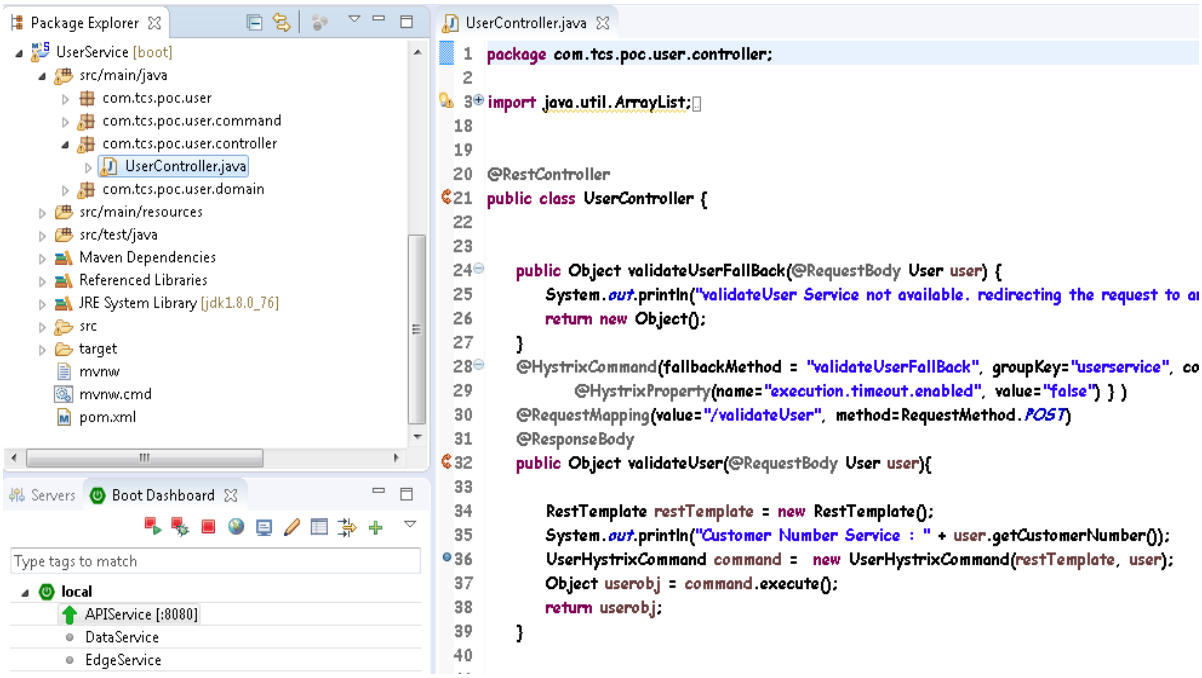
**deleteShpngList** – delete the shopping list
**getShoppingList** – get all the shopping list for login user.
**getShoppingListDetail** – get the shopping list detail
**modifyShoppingList** – modify the shopping list




In the Product service controller class, we have **getAllProduct** and **getProduct** methods which will use the shoppling list use case.



In the user service controller class, we have **validateUser** method which will use the shoppling list use case for login functionality.




## 7.4.5 ENABLE HYSTRIX IN ALL BUSINESS METHOD

In each method in business service, we need to enable to hystrix command. So that whenever method got fail, fallback method is going invoke and your request is processing without any issue.

```
public ShoppingVO addProductToShpngListFallBack(@RequestBody ShoppingVO shoppingList){
    System.out.println("addProductToShpngListFallBack Service not available. redirecting the request to another Server");
    return new ShoppingVO();
}
@HystrixCommand(fallbackMethod = "addProductToShpngListFallBack", groupKey="shoppinglistservice", commandKey="addProductToShpnList",  commandProperties = {
        @HystrixProperty(name="execution.timeout.enabled", value="false") } )
@RequestMapping(value = "/addProductToShpngList", method = RequestMethod.POST)
@ResponseBody
public ShoppingVO addProductToShpnList(@RequestBody ShoppingVO shoppingList) {
    ResponseEntity<ShoppingVO> responseEntity = null;
    HttpHeaders headers = new HttpHeaders();
    headers.set("Accept", MediaType.APPLICATION_JSON_VALUE);
    HttpEntity<Product> entity = new HttpEntity(shoppingList, headers);
    String restURL = "http://ShoppingListDataService/addProductToShpngList/";
    try{
     responseEntity =
            restTemplate.exchange(restURL,
                        HttpMethod.POST, entity, new ParameterizedTypeReference<ShoppingVO>() {
                });
    }catch(Exception e){
        e.printStackTrace();
    }
    return (ShoppingVO) responseEntity.getBody();
}
```

## 7.4.6 ENABLE FEIGN CLIENT

One of the business method (getshoppinglist) , we were implemented feign client to invoke the data service. For feign client implementation, we need to write interface and agent class to invoke.

**In controller class code**

```
@RequestMapping("/{memberid}/getShoppingList")
@ResponseBody
public List<Object> getShoppingList(@PathVariable("memberid") String memberid){

    List<Object> resp = new ArrayList<Object>();
    try{
        resp = controllerAgent.getShoppingList(memberid);
    }catch(Exception excep){
        excep.printStackTrace();
    }
    return resp;
}
```

**In feign interface code**

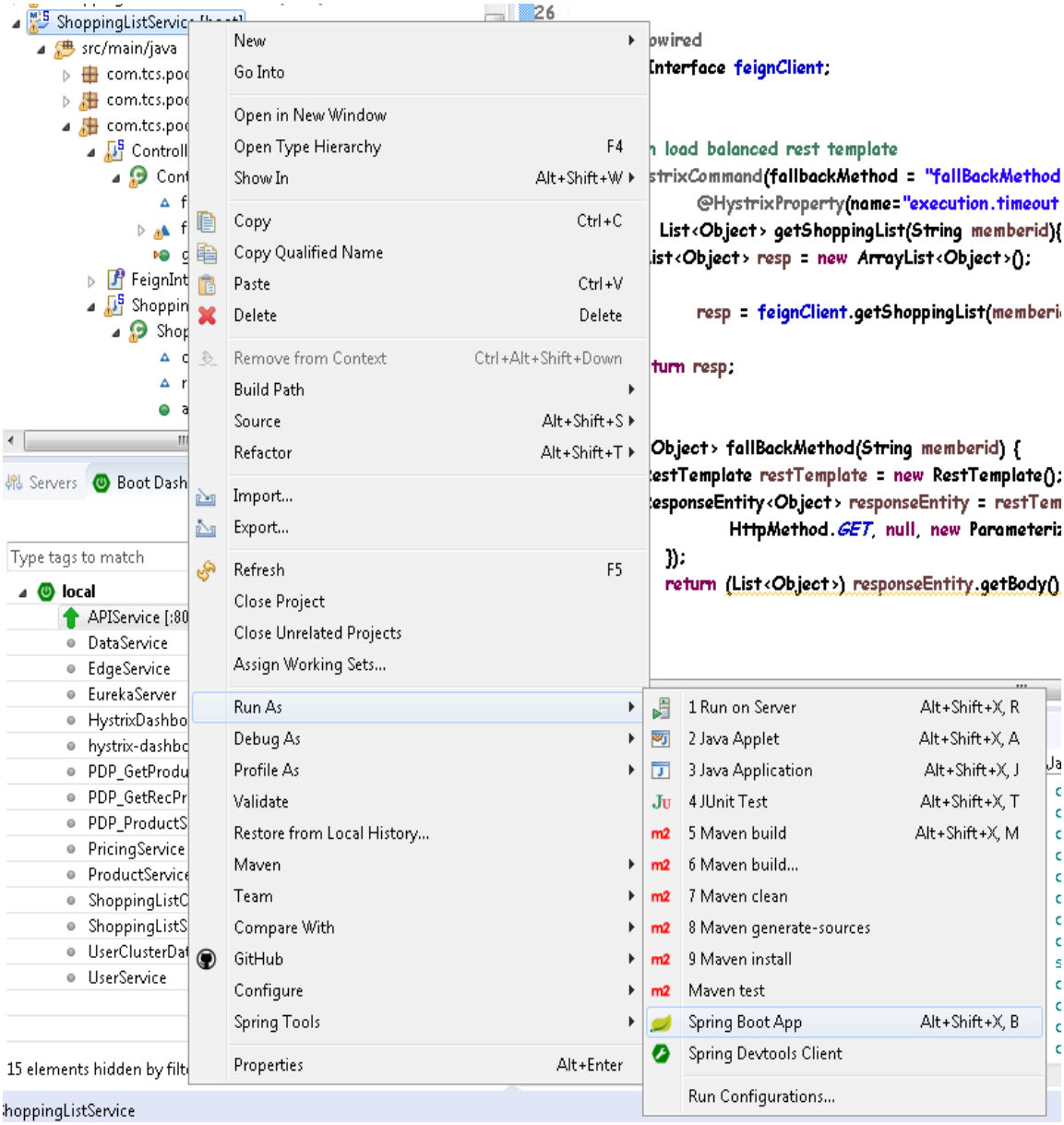
```
@FeignClient("http://ShoppingListDataService")
public interface FeignInterface {

    @RequestMapping(value="/{memberid}/getShoppingList", method = RequestMethod.GET)
    List<Object> getShoppingList(@PathVariable("memberid") String memberid);
```

**In Agent code**

```
@Service
public class ControllerAgent {

    @Autowired
    FeignInterface feignClient;


    //with load balanced rest template
     @HystrixCommand(fallbackMethod = "fallBackMethod", commandProperties = {
            @HystrixProperty(name="execution.timeout.enabled", value="false") })
    public List<Object> getShoppingList(String memberid){
        List<Object> resp = new ArrayList<Object>();

            resp = feignClient.getShoppingList(memberid);

       return resp;
    }

    List<Object> fallBackMethod(String memberid) {
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<Object> responseEntity = restTemplate.exchange("http://localhost:8106/getShoppingList/"+memberi
                HttpMethod.GET, null, new ParameterizedTypeReference<Object>() {
          });
          return (List<Object>) responseEntity.getBody();
       }
}
```

## 7.4.7 RUN APPLICATION

Before runing the application build and install the application using Maven build. Then Run the application using Spring boot.



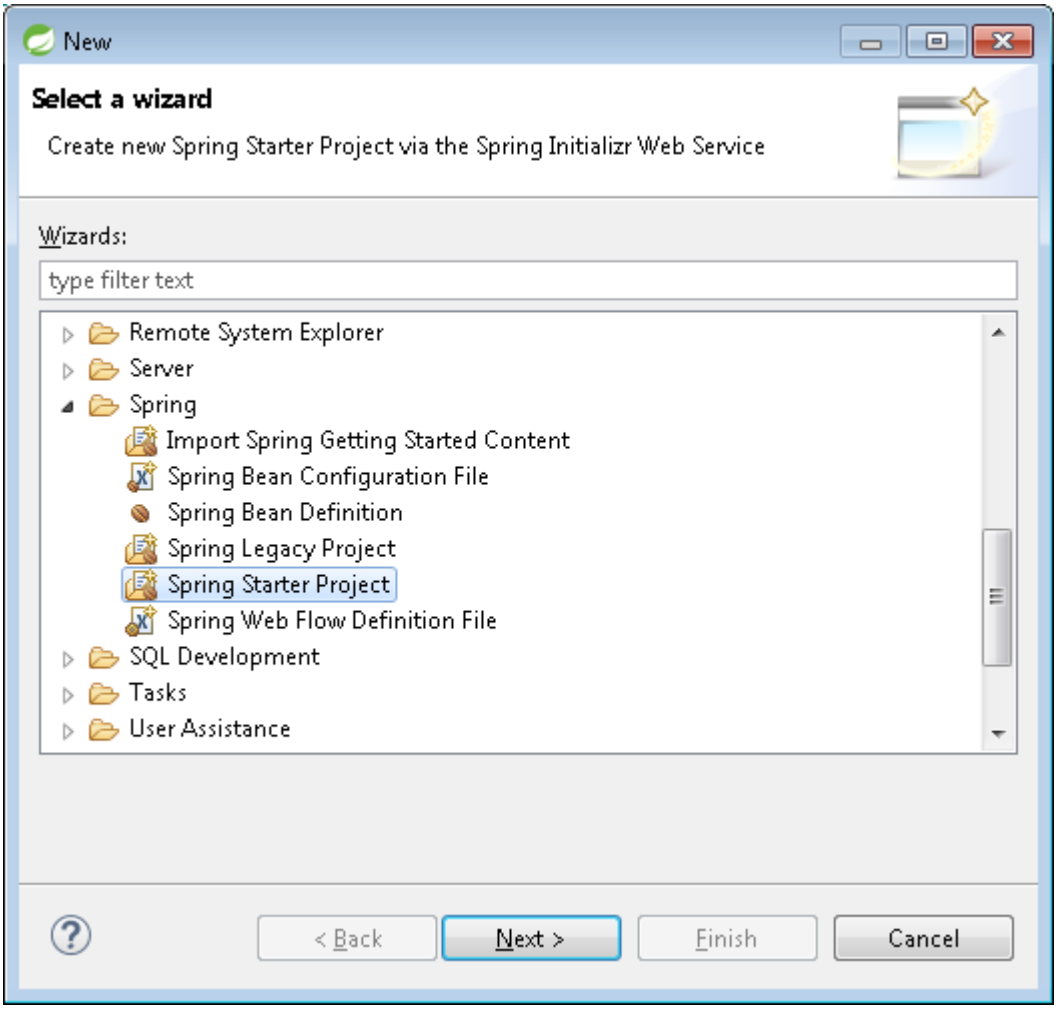
## 7.5 MICRO SERVICE - TIER C

Shopping list data service is micro service application using spring Netflix. Data Service is used to get the record from database and sent back the request data in Service layer. Apart from shopping list data service, we need to create product data service and user data service which will use for shopping list use case.

### 7.5.1 APPLICATION CREATION

In STS, Create new project and select Spring Starter Project and specify the package structure as com.tcs.poc.shoppinglist.

In next screen, select the boot version 1.3.3 and Dependencies, we need selet **"Eureka Discovery" and "Hystrix"** click Finish.



## 7.5.2 CREATE REST CONTROLLER

In Shopping list data service Project, create rest controller in com.tcs.poc.shoppinglist.controller package and create all the method with is needed for you shopping list functionality.

```
@RestController
public class ShoppingController {

    @Autowired
    private Bucket bucket;

    @RequestMapping(value = "/{memberid}/getShoppingList", method = RequestMethod.GET)
    @ResponseBody
    public List<ShoppingVO> getShoppingLists(@PathVariable("memberid") String memberId) {

        List<ShoppingVO> shpngList = new ArrayList<ShoppingVO>();
        ShoppingVO shpng;
        try {
            Gson gson = new Gson();
            N1qlQueryResult queryResult = bucket.query(N1qlQuery
                    .simple("SELECT META(shoppinglist).id,*  FROM shoppinglist where documentType = 'ShoppingList' a
                            + memberId + "'"));
            List<N1qlQueryRow> shpngListRow = queryResult.allRows();

            for (N1qlQueryRow row : shpngListRow) {
                shpng = new ShoppingVO();
                shpng = gson.fromJson(row.value().get("shoppinglist").toString(), ShoppingVO.class);
                shpng.setId(row.value().get("id").toString());
                shpngList.add(shpng);
            }

        } catch (Exception excep) {
            excep.printStackTrace();
        }
        return shpngList;
    }
```

## 7.5.3 SHOPPINGLIST DATA SERVICE APPLICATION CONFIGURATION

In Shopping list data Service Project, create application.yml file with add server port and application name and eureka server path and zone details.

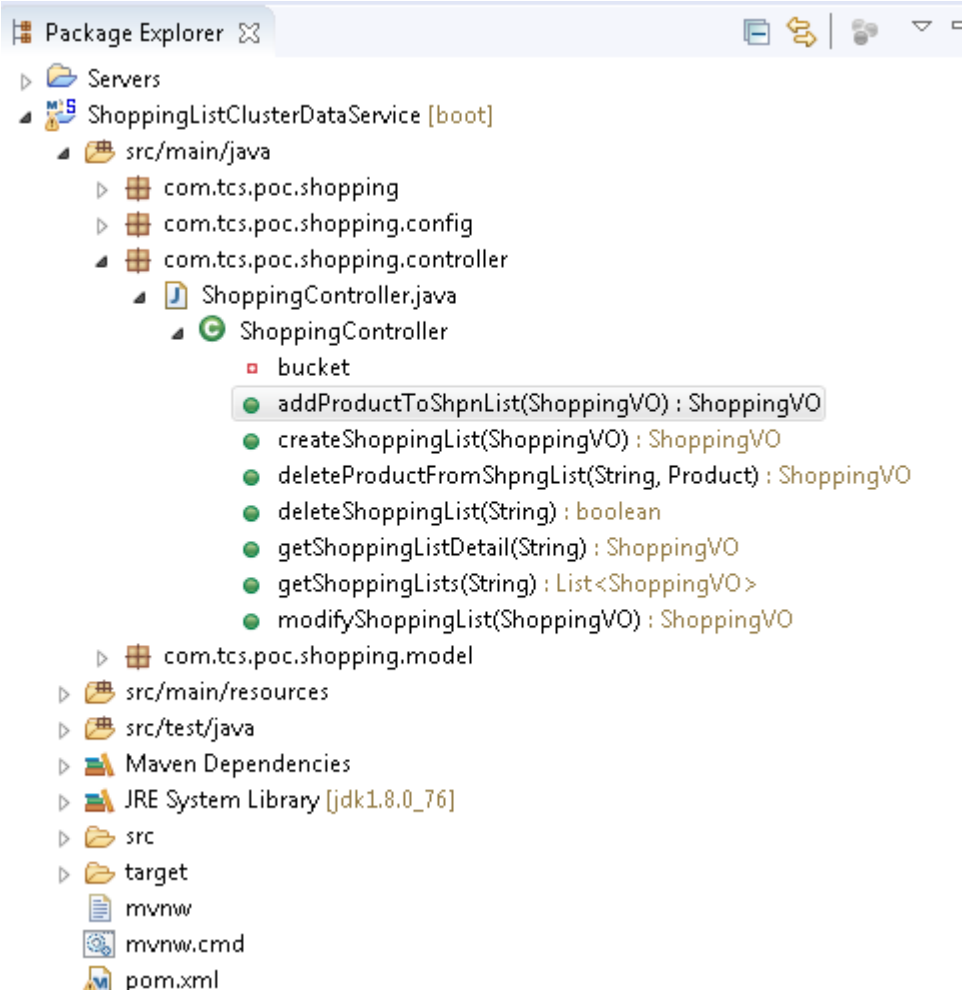
```
server:
  port: 8106
spring:
  application:
    name: ShoppingListDataService
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    instanceId: ${spring.application.name}:${spring.application.instance_id:${server.port}}
```

### 7.5.4 METHOD LEVEL DESIGN

In shopping list PoC, created user, product, shoppinglist and shopling list product domain, which wil be used to set and get the values.

In the controller class we have create below methods were used in shopping list functionality,

**addProductToShpnList** – adding new product to shopping list
**deleteProductFromShpngList** – delete product from shopping list
**deleteShpngList** – delete the shopping list
**getShoppinggList** – get all the shopping list for login user.
**getShoppingListDetail** – get the shopping list detail
**modifyShoppingList** – modify the shopping list
**createShoppingList** – create the shopping list



In the Product data service controller class, we have **getAllProduct and getProduct** methods which will use the shoppling list use case.

```
Package Explorer
DataService [boot]
  src/main/java
    com.tcs.poc.shoppinglist
    com.tcs.poc.shoppinglist.config
    com.tcs.poc.shoppinglist.controller
      ProductDataController.java
        ProductDataController
          productRepository
          getAllProducts() : List<Product>
          getProduct(String) : Product
    com.tcs.poc.shoppinglist.domain
  src/main/resources
    application.yml
  src/test/java
  Maven Dependencies
  JRE System Library [jdk1.8.0_76]
  src
  target
```

```
*ControllerA...   application.yml   ShoppingCon...   UserControll...   *UserCluste...   ProductDataC...

1  package com.tcs.poc.shoppinglist.controller;
2
3  import java.util.ArrayList;
16
17 @RestController
18 public class ProductDataController {
19
20     @Autowired
21     private ProductRepository productRepository;
22
23     @RequestMapping("/getAllProducts")
24     @ResponseBody
25     public List<Product> getAllProducts(){
26         List<Product> prdctList = new ArrayList<Product>();
27         try{
28             Query query = new Query();
29             prdctList = productRepository.getAllProducts(query);
30         }catch(Exception excep){
31             excep.printStackTrace();
32         }
33         return prdctList;
34     }
```

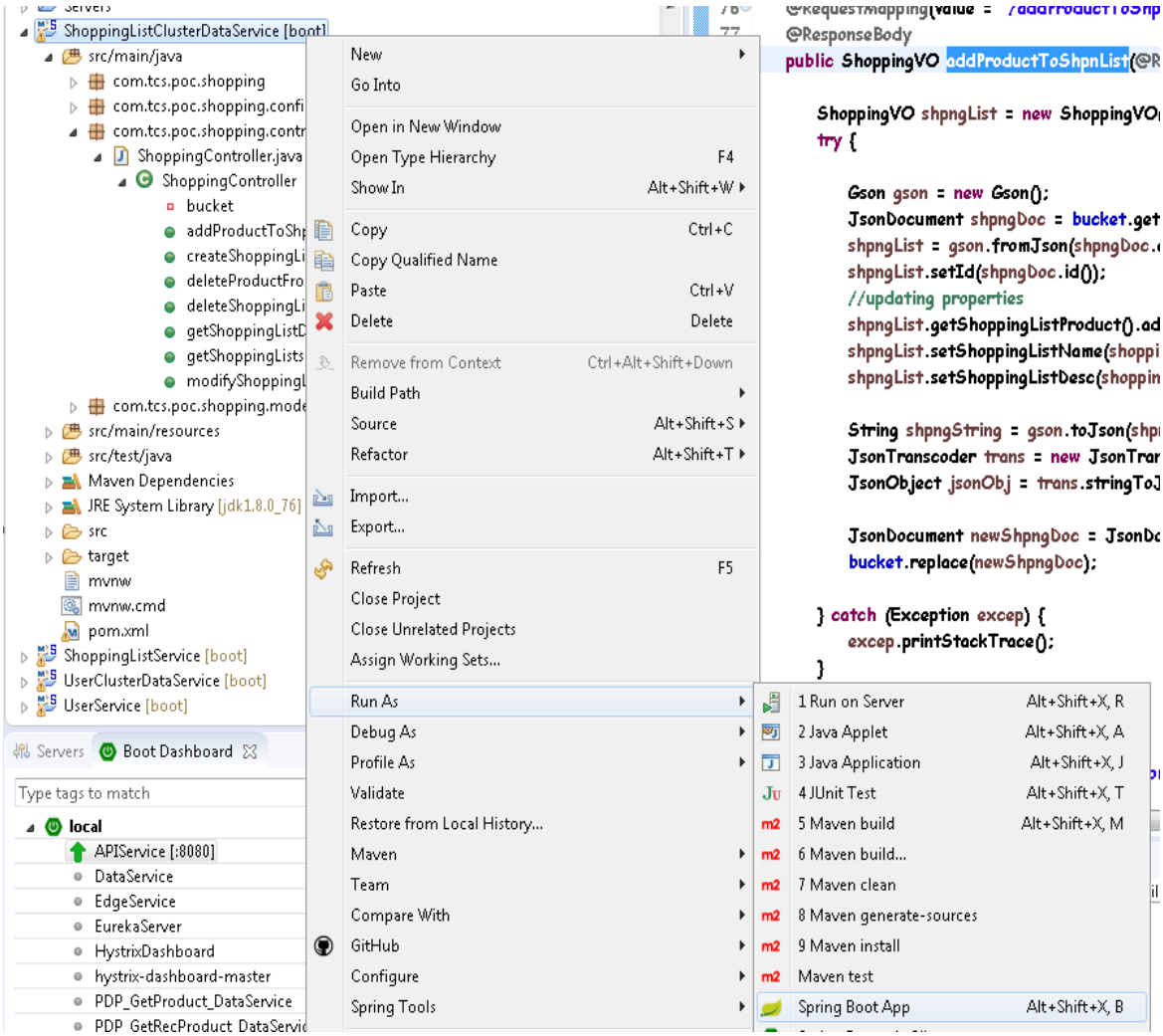
In the user data service controller class, we have **validateUser** method which will use the shoppling list use case for login functionality.

```
Package Explorer
  target
  mvnw
  mvnw.cmd
  pom.xml
ShoppingListService [boot]
UserClusterDataService [boot]
  src/main/java
    com.tcs.poc.user
    com.tcs.poc.user.controller
      UserClusterController.java
        UserClusterController
          validateUser(User) : Object
    com.tcs.poc.user.domain
  src/main/resources
  src/test/java
  Maven Dependencies
  Referenced Libraries
  JRE System Library [jdk1.8.0_76]
```

```
FeignInterf...   *ControllerA...   application.yml   ShoppingCon...   UserControll...   *UserCluste...

19 @RestController
20 public class UserClusterController {
21
22     @RequestMapping(value="/validateUser", method=RequestMethod.POST)
23     @ResponseBody
24     public Object validateUser(@RequestBody User user ){
25         Object userObject=null;
26         User responseUser=null;
27         try {
28             Cluster cluster = CouchbaseCluster.create("127.0.0.1");
29             Bucket bucket = cluster.openBucket("shoppinglist", "");
30             String email = user.geteMail();
31             String passwd = user.getpasswd();
32             String customerNumber = user.getCustomerNumber();
33             System.out.println("Customer Number API : " + user.getCustomerNumber());
34             //  JsonDocument userDoc = bucket.get("user01");
35             N1qlQueryResult queryResult =
36                     bucket.query(N1qlQuery.simple("SELECT * FROM shoppinglist where documentType = 'user' "
37                         + "and eMail = '"+email+"' and passwd ='" +passwd+"' and customerNumber ='"+customerNumber +"'"))
38             for(N1qlQueryRow row: queryResult.allRows()){
39                 userObject = row.value().get("shoppinglist");
40             }
41             Gson gson = new Gson();
42             responseUser = gson.fromJson(userObject.toString(), User.class);
43         } catch(NullPointerException nexce){
44             return "Invalid Login";
45         } catch(Exception exce){
46             exce.printStackTrace();
47         }
48         return responseUser;
```

## 7.5.5 RUN APPLICATION

Before runing the application, build applciation and install the application in Maven. Then Run the application using Spring boot.



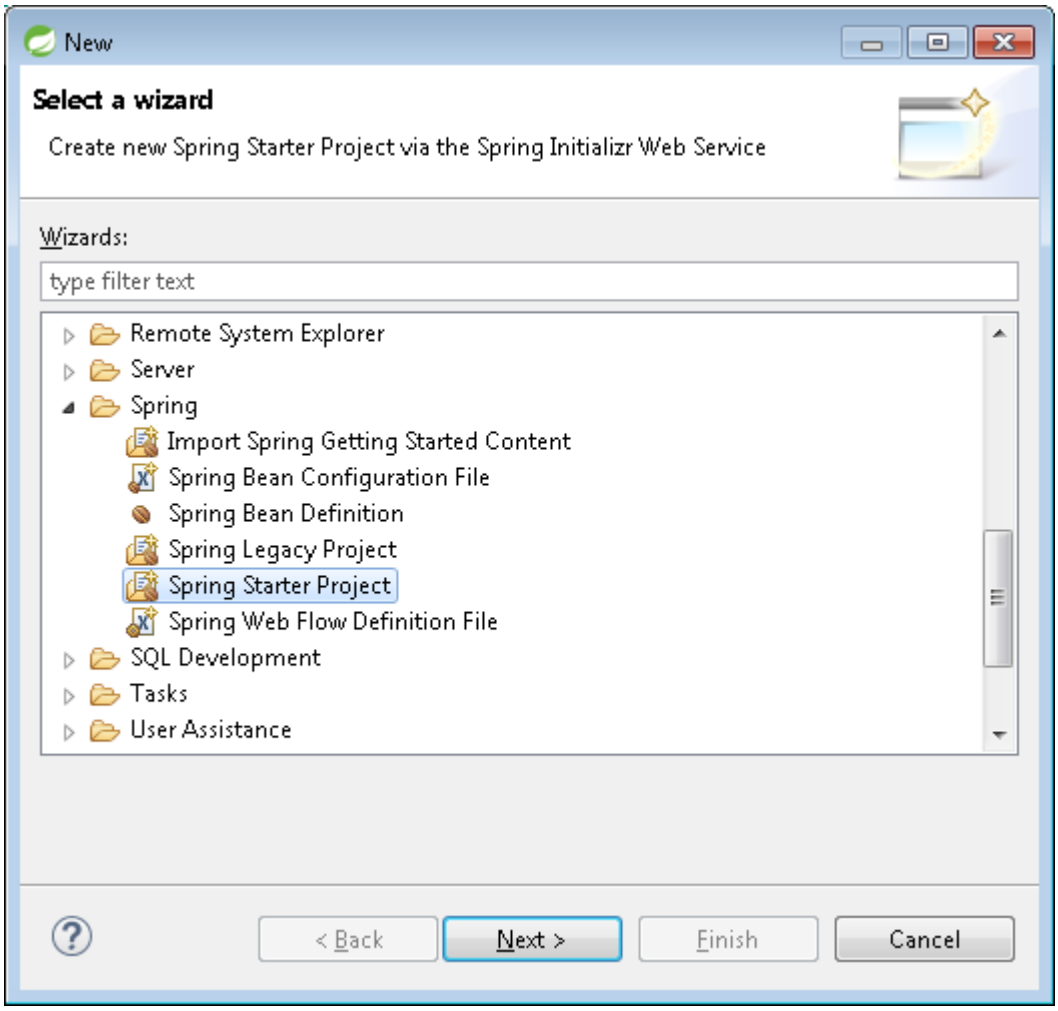
## 7.6 HYSTRIX DASHBOARD

Hystrix dashboard application is boot application. Hystrix dashboard allows you to monitor Hystrix metrics in real time.

### 7.6.1 APPLICATION CREATION

In STS, Create new project for Hystrix Dashboard and select Spring Starter Project and specify the package structure as com.tcs.poc.shoppinglist.

In next screen, select the boot version 1.3.3 and Dependencies, we need selet "**Hystrix Dashboard**" and click Finish.



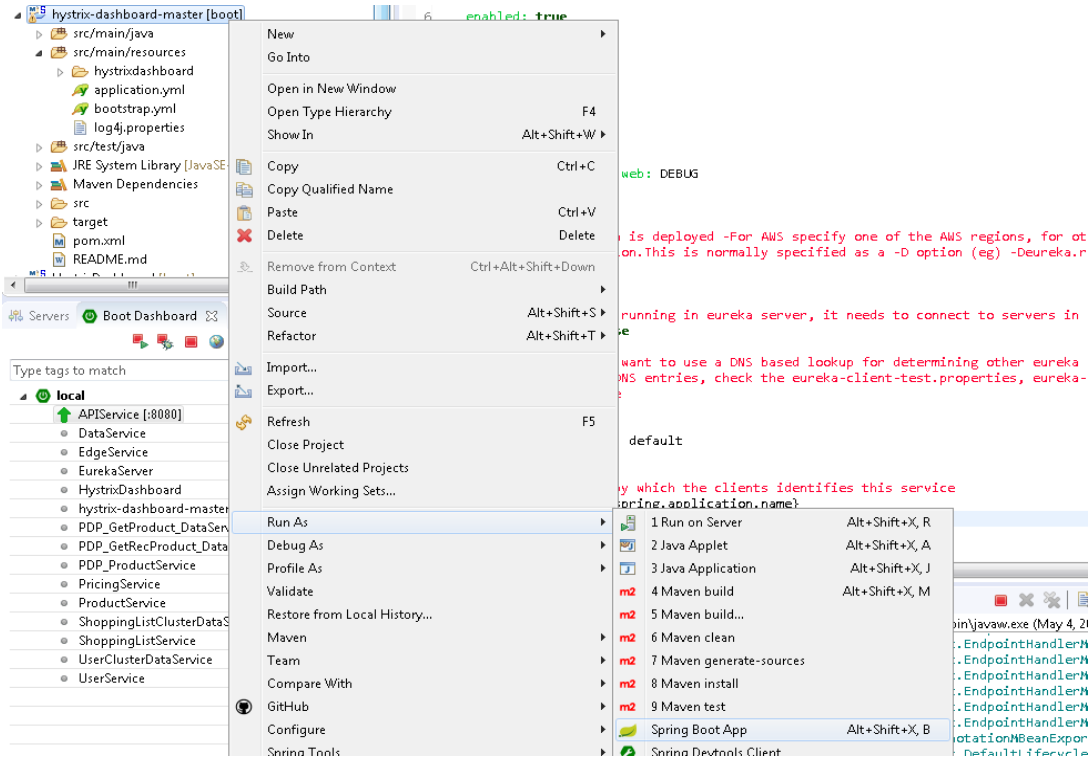
## 7.6.2 HYSTRIX DASHBOARD APPLICATION CONFIGURATION

In Hystrix Dashboard Project, create application.yml file with add server port and application name.

```yaml
info:
  component: Hystrix Dashboard

endpoints:
  restart:
    enabled: true
  shutdown:
    enabled: true

server:
  port: 7979

logging:
  level:
    ROOT: INFO
    org.springframework.web: DEBUG

eureka:
  client:
    #Region where eureka is deployed -For AWS specify one of the AWS regions, for other datacenters specify a
    #indicating the region.This is normally specified as a -D option (eg) -Deureka.region=us-east-1
    region: default


    #For eureka clients running in eureka server, it needs to connect to servers in other zones
    preferSameZone: false

    #Change this if you want to use a DNS based lookup for determining other eureka servers. For example
    #of specifying the DNS entries, check the eureka-client-test.properties, eureka-client-prod.properties
    #shouldUseDns: false

    us-east-1:
      availabilityZones: default

  instance:
    #Virtual host name by which the clients identifies this service
    virtualHostName: ${spring.application.name}
```

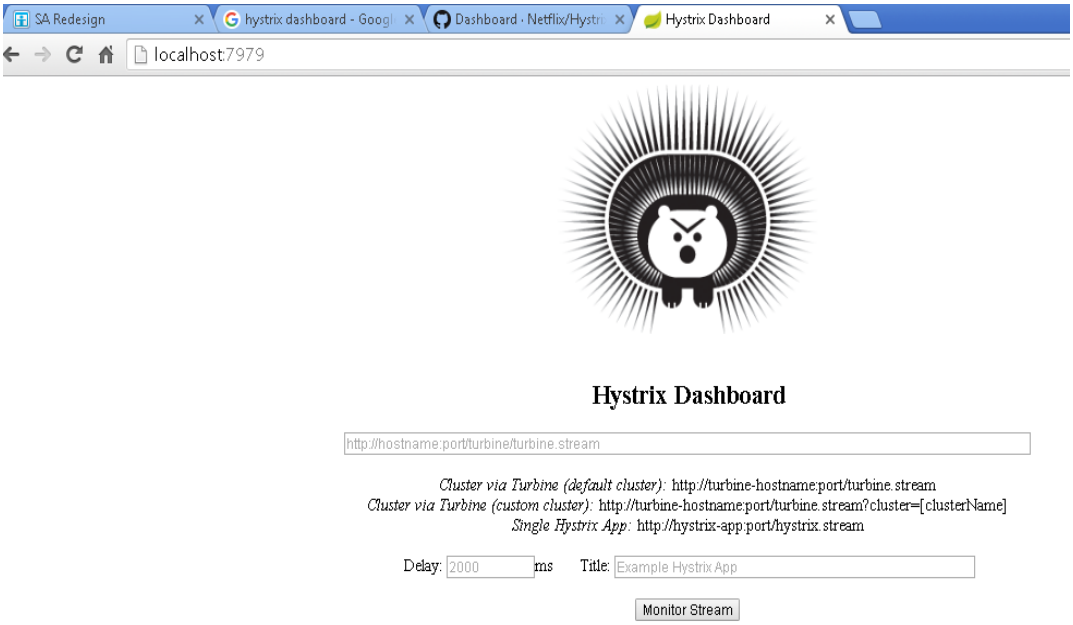
## 7.6.3 RUN HYSTRIX DASHBOARD APPLICATION

Before runing the application, clean and install the application using Maven build. Then Run the application using Spring boot.



### 7.6.4 HYSTRIX DASHBOARD

After runing the application, Access the http:///localhost:7979/




To monitor a single server you would use a URL such as

http://localhost:8110/hystix.stream to monitor Hystrix metrics.

# 8 ADDITIONAL RESOURCES

Please refer the below links to explore more,

| Topic | Content Sources |
|---|---|
| **Java 8 – Lamdbas** | http://techbus.safaribooksonline.com/video/programming/java/9780134383644 |
| | http://techbus.safaribooksonline.com/video/programming/java/9781771374743/lambdas/video229805?query=((java+8+lambda))#snippet |
| | http://techbus.safaribooksonline.com/video/programming/java/9781771373364/introduction-to-lambda-programming-part-1/95-2015-03-03?query=((java+8+lambda))#snippet |
| | http://techbus.safaribooksonline.com/video/programming/java/9780134540603/lesson-6-interfaces-lambda-expressions-and-inner-classes/corj_06_05?query=((java+8+lambda))#snippet |
| | http://www.lynda.com/Java-tutorials/Java-SE-8-New-Features/156621-2.html |
| | http://techbus.safaribooksonline.com/book/programming/java/9781449370831 |
| | https://www.youtube.com/watch?v=8pDm_kH4YKY |
| | |
| **RxJava – Observables, Couchbase Integration** | WIKI of github for RxJava<br>https://github.com/ReactiveX/RxJava/wiki/How-To-Use-RxJava<br>https://github.com/ReactiveX/RxJava/wiki/Observable |
| | http://reactivex.io/intro.html |
| | https://www.youtube.com/watch?v=Dk8cR1Kxj0Y |
| | http://reactivex.io/documentation/observable.html<br>http://reactivex.io/documentation/operators.html<br>http://reactivex.io/documentation/single.html<br>http://reactivex.io/documentation/scheduler.html |
| | http://docs.couchbase.com/developer/java-2.1/observables.html |
| | http://docs.couchbase.com/developer/java-2.0/observables.html |
| | https://www.youtube.com/watch?v=3EOKYPbvwHQ |
| | |
| **Spring Boot** - Key starter POM's (actuator, web, thymeleaf, couchbase, redis) | http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/ |
| | http://techbus.safaribooksonline.com/video/web-development/9780133890204/lesson-1-starting-with-spring/lesson_1_1?query=((spring+cloud))#snippet |
| | http://techbus.safaribooksonline.com/video/programming/java/9780134192468?bookview=overview |
| | http://techbus.safaribooksonline.com/video/programming/java/9781491942680/spring-boot-demo/p38?query=((spring+boot))#snippet |
| | Getting started with Spring boot - https://www.youtube.com/watch?v=sbPSjI4tt10 |
| | https://www.youtube.com/watch?v=X5DRXCKJH_M |

| | |
|---|---|
| Spring Cloud Netflix<br>Zuul - Edge routing and filtering (external end point)<br>Hystrix - Circuit Breaker pattern, enables OOTB monitoring spring actuator<br>Hystrix Dashboard - Allows to monitor Hystrix streams in visual format<br>Turbine - Combine multiple Hystrix streams<br>Eureka - Service Discovery<br>Ribbon - Client Side Load balancing | http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html<br>http://projects.spring.io/spring-cloud/spring-cloud.html |
| | https://www.youtube.com/watch?v=6wvVANQ6EJ8 |
| | https://www.youtube.com/watch?v=ZyK5QrKCbwM |
| | https://www.youtube.com/watch?v=AARMi1mB-HY |
| | https://spring.io/blog/2014/06/03/introducing-spring-cloud |
| | https://www.youtube.com/playlist?list=PL62plycqXx-QKMyHqLem4Nh00Wnd2emwr |
| | http://callistaenterprise.se/blogg/teknik/2015/05/20/blog-series-building-microservices/ |
| | Part 1,2,3,4 from the following link - http://techbus.safaribooksonline.com/video/software-engineering-and-development/9781491927991/architecture/video217987?query=((spring+cloud))#snippet |
| | **Hystrix** -<br>https://www.youtube.com/watch?v=Jtcx7vAo33E<br>https://www.youtube.com/watch?v=RzlluokGi1w<br>https://github.com/Netflix/Hystrix/wiki<br>https://ahus1.github.io/hystrix-examples/manual.html<br>http://www.ebaytechblog.com/2015/09/08/application-resiliency-using-netflix-hystrix/<br>http://techbus.safaribooksonline.com/book/programming/java/9781449374631/spring-cloud/idm140138731732112_html?query=((hystrix+dashboard))#snippet |
| | **Hystrix dashboard** - https://github.com/Netflix/Hystrix/wiki/Dashboard<br>https://www.youtube.com/watch?v=v3pAMHdR4BM |
| | **Turbine** - https://github.com/Netflix/Turbine/wiki<br>Follow wiki documentation for turbine - https://github.com/Netflix/Turbine/wiki/Getting-Started-(1.x) |
| | |
| **Spring Cloud Config** - Centralized Configuration Management exposes GIT | http://cloud.spring.io/spring-cloud-config/<br>http://cloud.spring.io/spring-cloud-config/spring-cloud-config.html |
| | |
| **Spring Cloud Security - OAuth Server** | http://cloud.spring.io/spring-cloud-security/spring-cloud-security.html<br>https://www.youtube.com/watch?v=USMl2GNg2r0 |
| | https://www.youtube.com/watch?v=MLfL1NpwUC4 |