Microservices
By
P.Ashok

Index

S NO	Topic	Page No
1	Service Registry & Discovery (Eureka) …….	9
2	Load Balancing Server (Ribbon) ……..	23
3	Declarative Rest Client (Feign) ……	31
4	Circuite Breaker (Hystrix) ……..	75
5	API Gateway (Zuul) ……	81
6	Spring Boot Message Queue (MQ) ……..	95
7	Message Queues (Kakfa) ……..	108
8	Integration Service (Camel) ……..	118
9	Secure Server (OAuth2) ……..	145
10	Cloud Platform (PCF, Docker) ……	157
11	Production Ready Endpoint (Actuator) ……..	169
12	Metrics UI -- Admin (Server/Client) ……	171
13	Mockito ……	232
14	Gradle ……..	250
15	Docker ……	279

MICROSERVICES

1. Monolithic Application:--

A Project which holds all modules together and converted as one Service (one .war file).

=>All most every project in realtime is implemented using this format only.

=>In this case, if no. of users are getting increased, then to handle multiple request (load) use LBS (Load Balancing Server).

=>But few modules needs Extra load, not all. In this case other modules get memory which is waste (no use). Hence reduces performance of server (application).

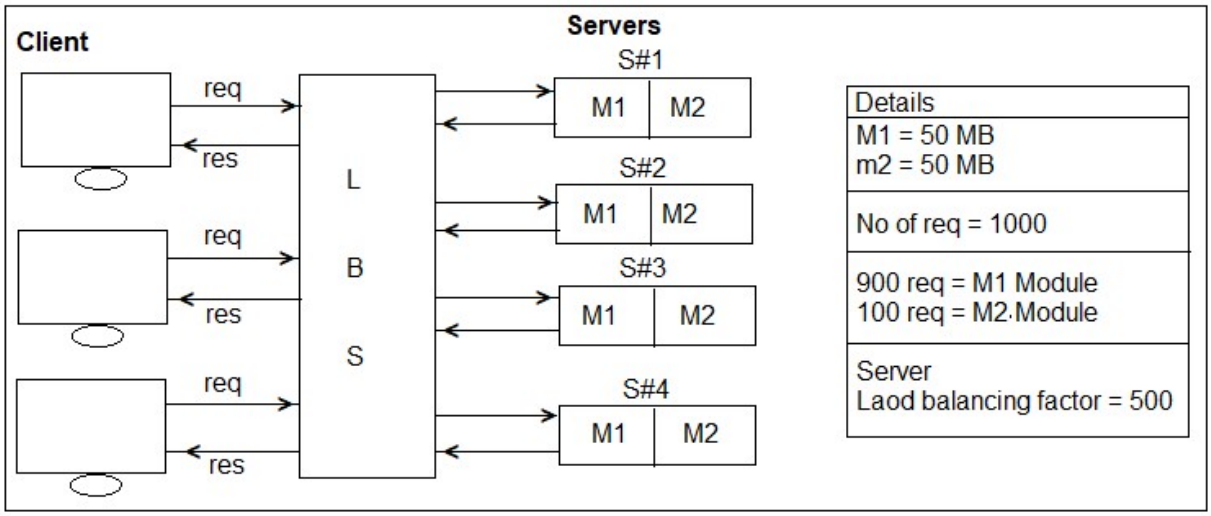
**Consider Project P1 is having 2 modules M1(Register), M2(Login) and their runtime memories are M1=50 MB, M2=50 MB.

=>Here, End user Register only onetime (first time) and login is mostly used module.

->Consider Application needs 100MB size in server as register 50MB and Login 50MB.

->To handle multiple client request multiple instances of sample App must be provided using Load balancing server with Load Register.

Diagram:-- Load Balancing is done using Servers looks like.



=>In above example, M2 Module is getting fewer (100) requests from Client. So, max 2 instances are may be enough. Other 2 instances (memories) are no use. It means near

100MB memory is not used, which impacts server performance.

2. Microservices:-- It is a independent deployment component.

=>It is a combination of one (or more) modules of a Projects runs as one Service.

=>To avaoid monolithic Limitations like memory and time (performance) problems use this design.

Nature of Microservices:--

1>Every Service must be implemented using **Webservices** concept.

2>One or multiples module as one Project.

3>Every service must be **independent** (One service should not effect another one, like for code modifications, version upgrades, downtime for few servers… etc).

4>It should able to communicate with any type of client (Mobile, Web based, 3rd party).

5>Every Services should able to communicate with each other Microservice, It is also called as **"Intra Communication**".

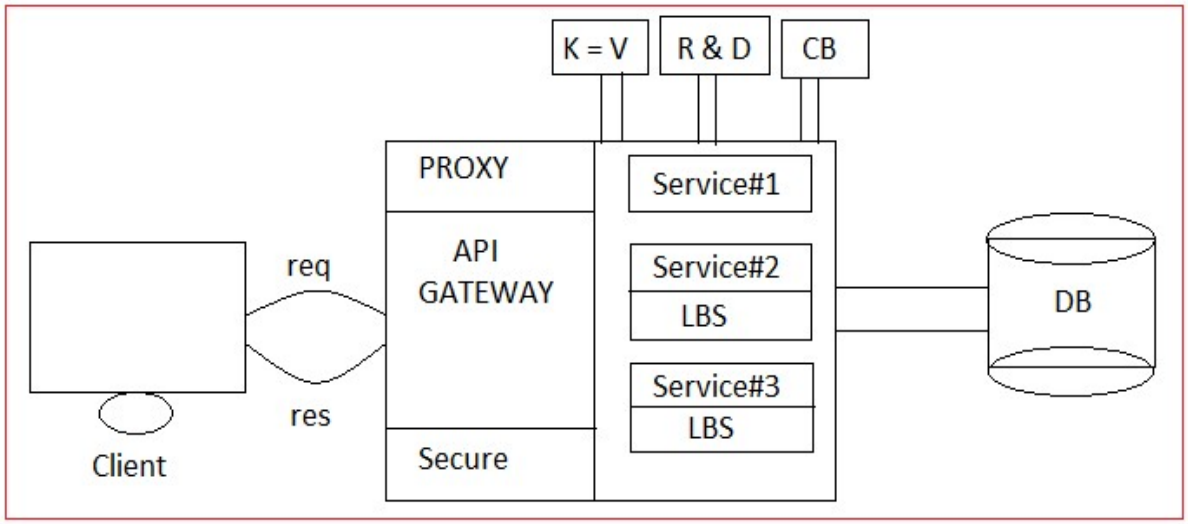
6>Required Services must be supported for **Dynamic Load Balancing** (i.e. one service runs in multiple instances) based on click request.

7>Every microservice should support able to read input data from External **Configuration Server** [Config Server](Dynamic inputs using (_.properties/_.yml), with GIT/Config Server).

8>Service communication (Chain of execution) problems should be able to solve using **Circuite Breaker** [Find other possible…].

9>All Servers must be accessed to Single Entry known as Gateway Service [Proxy Gateway or API Gateway], It supports securing, **metering** and **Routing**.

Microservice Generic Architecture:--



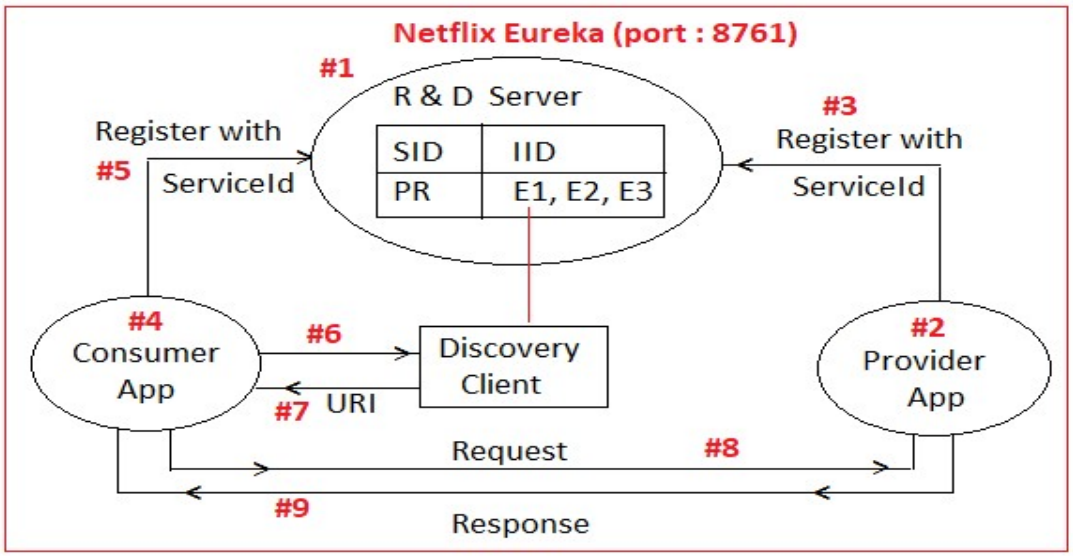
=>This design provides bility to communicate any kind of client application.

Ex:-- Android, Angular, RFID, Web, 3rd party, Swings,… etc.

=>Eureka is a R & D server which supports microservices register, discover with each other for intra communication.

=>Every Microservice should get reistered with Eureka using one ServiceId (one or more instanceId).

=>For consumer application using Discovery client, get URI and make http Request to producer Application.



Component Names:--

1>Service Registry & Discovery	= Eureka
2>Load Balancing Server	= Ribbon
3>Circuite Breaker	= Hystrix
4>API Gateway	= Zuul
5>Config Server	= GitHub
6>Secure Server	= OAuth2
7>Log and Trace	= Zipkin + Sleuth
8>Message Queues	= Kakfa
9>Declarative Rest Client	= Feign
10>Integration Service	= Camel
11>Production Ready Endpoint	= Actuator
12>Metrics UI	= Admin (Server/Client)
13>Cloud Platform With Deploy services	= PCF, Docker

SOA (Service Oriented Architecture):--

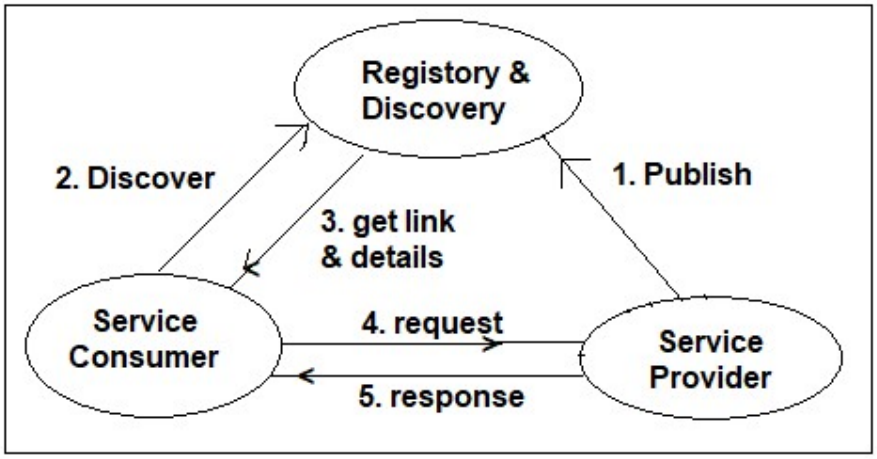
=>It is a Design Pattern used to create communication links between multiple services providers and Consumers (users).

Components of SOA:--

a>Service Registry and Discovery [Eureka]
b>Service Provider [Webservice Provider]
c>Service Consumer [Webservice Client]

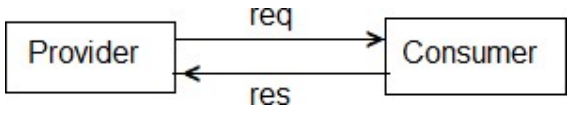
Operations:--

1>Publish
2>Discover
3>Link Details of Provider
4>Query Description (Make Http Request).
5>Access Service (Http Response).



Implementing MicroService Application Using Spring Cloud:--

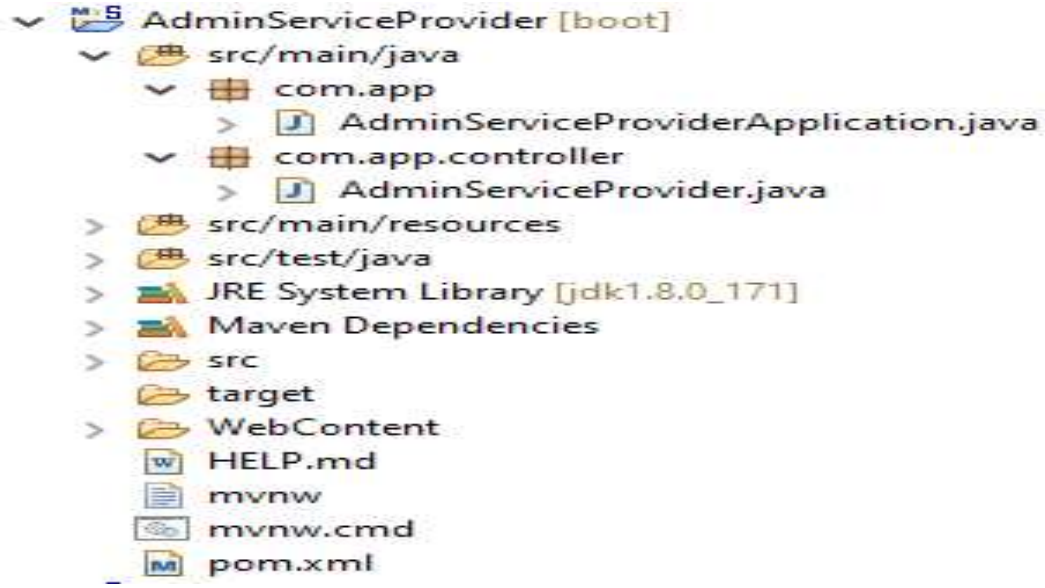
Design #1:-- A Simple Rest WebService using Spring Boot.



=>This application is implemented using Spring Boot Restfull webservices which provides Tightly coupled design, It means any changes in provider application effects consumer application, specially server changes port no changes, context changes etc.
=>This design will not support load balancing.
=>It is implemented using RestController and RestTemplate.

Step#1:- Create Provider Application (Dependencies : web only)
Group Id : org.verizon
ArtifactId : AdminServiceProvider
Version : 1.0

#1. Folder Structure of AdminServiceProvider:--



Step #2:- Define one RestController (AdminServiceProvider.java):--

```
package com.app.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/provider")
public class AdminServiceProvider  {
	@GetMapping("/show")
	public String showMsg() {
```

```
            return "Hello";
        }
}
```

Step #3:- Create Consumer Application (Dependencies : web only)

GroupId : org.verizon
ArtifactId : AdminServiceConsumer
Version : 1.0

#2. Folder Structure of AdminServiceConsumer:--

Step #4:- Define Consumer (call) code (AdminConsumer.java):--

```
package com.app.consumer;
import org.springframework.boot.CommandLineRunner;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class AdminConsumer implements CommandLineRunner {

        public void run (String... args) throws Exception {
                RestTemplate rt = new RestTemplate();
                ResponseEntity<String> resp =
rt.getForEntity("http://localhost:2019/provider/show", String.class);
```

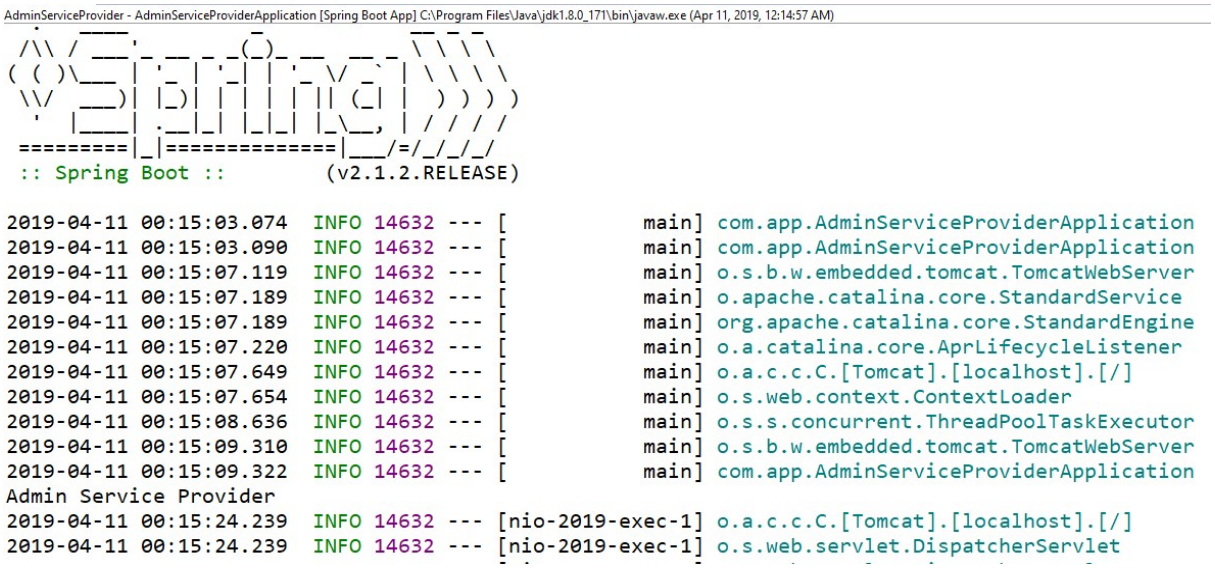
```
            System.out.println(resp.getBody());
            System.exit(0);
        }
}
```

Execution flow SCREEN :--

=>First Run provider (Starter), then consumer and enter below url

1>http://localhost:2019/provider/show

#1 PROVIDER SCREEEN:--



```
AdminServiceProvider - AdminServiceProviderApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_171\bin\javaw.exe (Apr 11, 2019, 12:14:57 AM)
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.2.RELEASE)

2019-04-11 00:15:03.074  INFO 14632 --- [           main] com.app.AdminServiceProviderApplication
2019-04-11 00:15:03.090  INFO 14632 --- [           main] com.app.AdminServiceProviderApplication
2019-04-11 00:15:07.119  INFO 14632 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2019-04-11 00:15:07.189  INFO 14632 --- [           main] o.apache.catalina.core.StandardService
2019-04-11 00:15:07.189  INFO 14632 --- [           main] org.apache.catalina.core.StandardEngine
2019-04-11 00:15:07.220  INFO 14632 --- [           main] o.a.catalina.core.AprLifecycleListener
2019-04-11 00:15:07.649  INFO 14632 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2019-04-11 00:15:07.654  INFO 14632 --- [           main] o.s.web.context.ContextLoader
2019-04-11 00:15:08.636  INFO 14632 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor
2019-04-11 00:15:09.310  INFO 14632 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2019-04-11 00:15:09.322  INFO 14632 --- [           main] com.app.AdminServiceProviderApplication
Admin Service Provider
2019-04-11 00:15:24.239  INFO 14632 --- [nio-2019-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2019-04-11 00:15:24.239  INFO 14632 --- [nio-2019-exec-1] o.s.web.servlet.DispatcherServlet
```

CONSUMER SCREEN:--

```
<terminated> AdminServiceConsumer - AdminServiceConsumerApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_171\bin\javaw.exe (Apr 11, 2019, 12:17:35 AM)
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.2.RELEASE)

2019-04-11 00:17:39.533  INFO 16700 --- [  restartedMain] com.app.AdminServiceConsumerApplication
2019-04-11 00:17:39.549  INFO 16700 --- [  restartedMain] com.app.AdminServiceConsumerApplication
2019-04-11 00:17:39.705  INFO 16700 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2019-04-11 00:17:39.705  INFO 16700 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor
2019-04-11 00:17:43.189  INFO 16700 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer
2019-04-11 00:17:43.253  INFO 16700 --- [  restartedMain] o.apache.catalina.core.StandardService
2019-04-11 00:17:43.253  INFO 16700 --- [  restartedMain] org.apache.catalina.core.StandardEngine
2019-04-11 00:17:43.285  INFO 16700 --- [  restartedMain] o.a.catalina.core.AprLifecycleListener
2019-04-11 00:17:43.550  INFO 16700 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]
2019-04-11 00:17:43.550  INFO 16700 --- [  restartedMain] o.s.web.context.ContextLoader
2019-04-11 00:17:44.081  INFO 16700 --- [  restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor
2019-04-11 00:17:44.503  INFO 16700 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer
2019-04-11 00:17:44.692  INFO 16700 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer
2019-04-11 00:17:44.692  INFO 16700 --- [  restartedMain] com.app.AdminServiceConsumerApplication
Hello
2019-04-11 00:17:44.911  INFO 16700 --- [       Thread-9] o.s.s.concurrent.ThreadPoolTaskExecutor
```

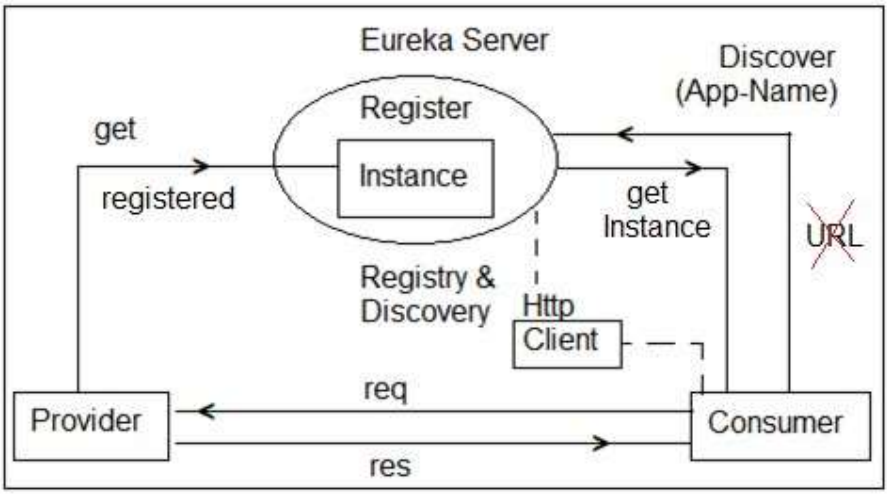
3. MicroService Design and Implementation using Spring Cloud (Netflix Eureka Registry & Discovery):--

=>Registory and Discovery server hold details of all Client (Consumer/Producer) with its serviced and Instance Id.

=>Netflix Eureka is one R & D Server.

=>Use default port no is 8761.

Design#1:- [Basic – No Load Balancing]



Step #1: Create Eureka Server:-- Create one Spring Boot Starter Project with Dependencies : **Eureka Server**

Eureka Server Dependencies:--

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server </artifactId>
</dependency>
```

GroupId : org.verizon
ArtifactId : EurekaServerApp
Version : 1.0

#3. Folder Structure of Eureka Server:--

pom.xml of Eureka Server:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.1.1.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.app</groupId>
	<artifactId>EurekaServerApp</artifactId>
	<version>1.0</version>
	<name>EurekaServerApp</name>
	<description>Demo project for Eureka Server</description>
	<properties>
		<java.version>1.8</java.version>
		<spring-cloud.version>Greenwich.SR1</spring-cloud.version>
	</properties>
	<dependencies>
		<dependency>
```

```
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>
        </dependencies>

        <dependencyManagement>
            <dependencies>
                <dependency>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-dependencies</artifactId>
                    <version>${spring-cloud.version}</version>
                    <type>pom</type>
                    <scope>import</scope>
                </dependency>
            </dependencies>
        </dependencyManagement>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Step#2:- At Starter class level add Annotation **@EnableEurekaServer** Annotation:--

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import  org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApp {
        public static void main(String[] args) {
                SpringApplication.run(EurekaServerApp.class, args);
                System.out.println("Eureka Server Executed:");
        } }
```

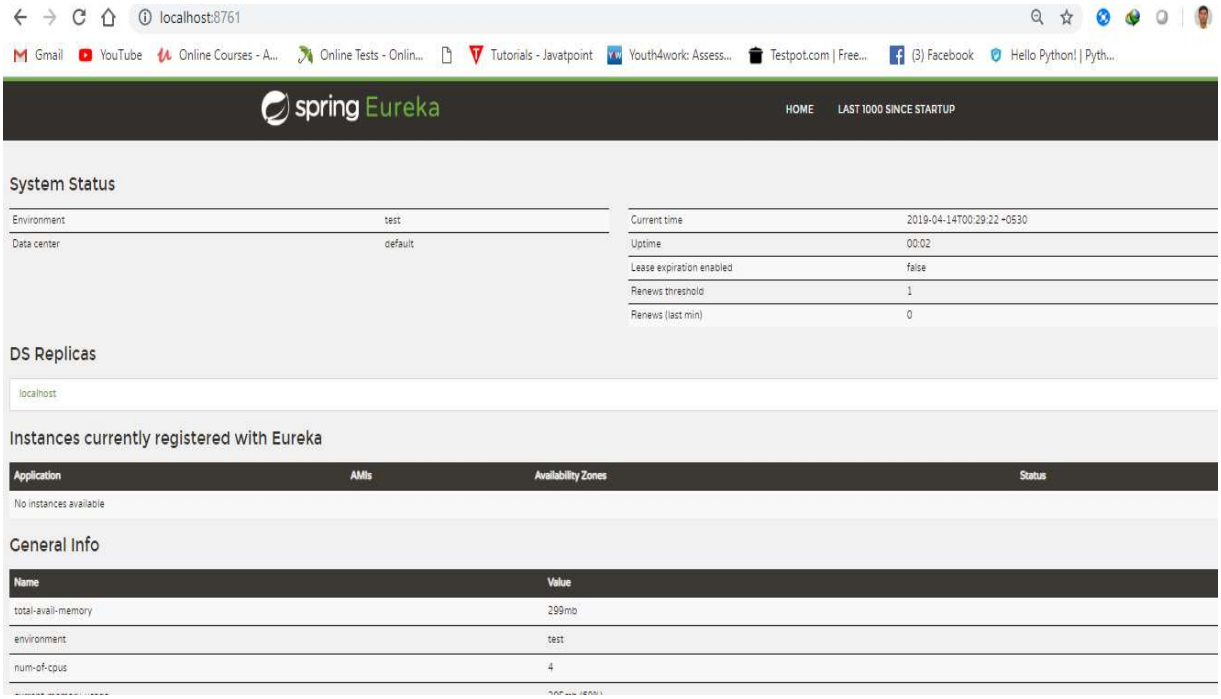
Step #3:- In application.properties add keys

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Step #4:- Run starter class and Enter URL http://localhost:8761 in browser

NOTE:-- Default port no of Eureka Server is 8761.

Screen Short of Eureka Server Dashboard:--



4. Spring Boot Provider (Producer) Application:--

=>Every client service should get registered with Eureka and implement load balancing in case of multiple request processing.

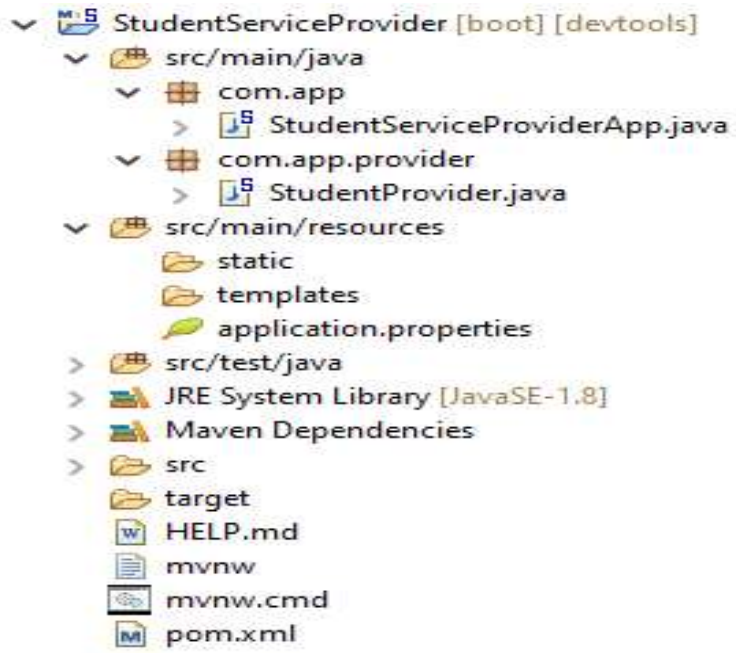
Step #1:- Create one Spring starter App with web and **Eureka Discovery Client** dependencies

Eureka Discovery Client Dependencies:--

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client </artifactId>
</dependency>
```

GroupId : org.app
ArtifactId : StudentServiceProvider
Version : 1.0

#4. Folder Structure of Provider Application:--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.1.1.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.app</groupId>
	<artifactId>StudentServiceProvider</artifactId>
	<version>1.0</version>
	<name>StudentServiceProvider</name>
	<description>Demo project for Microservice Studenet Provider</description>
<properties>
		<java.version>1.8</java.version>
		<spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
```

```
		</dependency>
	</dependencies>

<dependencyManagement>
		<dependencies>
			<dependency>
				<groupId>org.springframework.cloud</groupId>
				<artifactId>spring-cloud-dependencies</artifactId>
				<version>${spring-cloud.version}</version>
				<type>pom</type>
				<scope>import</scope>
			</dependency>
		</dependencies>
</dependencyManagement>
	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>
</project>
```

Step #2:- Add below annotation at Starter class level **@EnableEurekaClient** (given by Netflix) or **@EnableDiscoveryClient** (given by Spring Cloud) both are optional.

Spring Starter class (StudentServiceProviderApp.java):--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class StudentServiceProviderApp {
```

```
	public static void main(String[] args) {
		SpringApplication.run(StudentServiceProviderApp.class, args);
		System.out.println("Student Service provider");
	}
}
```

Step #3:- In application.properties file

```
server.port=9800
spring.application.name=STUDENT-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

Step #4:- Define one Provider Controller (StudentProvider) :--

```
package com.app.provider;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/provider")
public class StudentProvider {

	@GetMapping("show")
	public String showMsg() {
		return "Hello from Provider";
	}
}
```

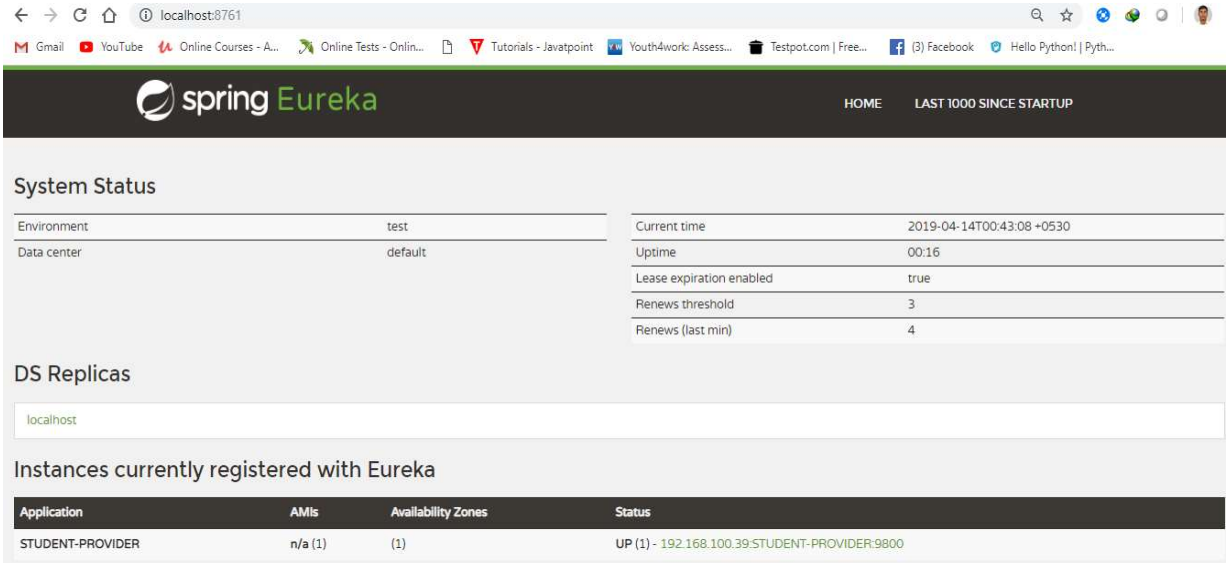
Execution Order:- (RUN Starter Classes)

1. EUREKA SEVER
2. PROVIDER APPLICATION

=>Goto eureka dashboard and check for application

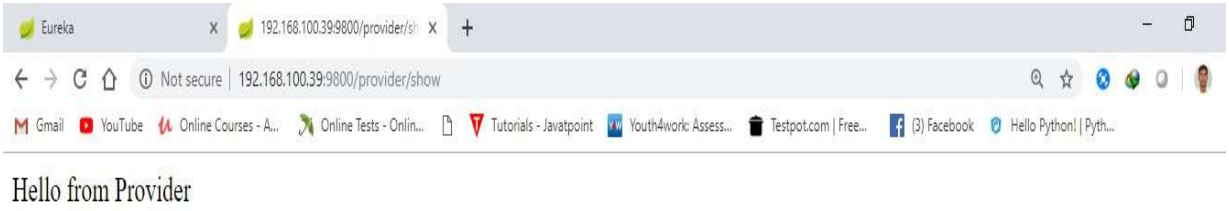
=>Click on url and add /provider/show PATH

#1 Screen Short of Eureka Server Dashboard:--



NOTE:-- Click on URI (Instance) (192.168.100.39:STUDENT-PROVIDER:9800) then add provider path after URI (http://192.168.100.39:9800/provider/show)

#2 Screen Short of Provider Application:--



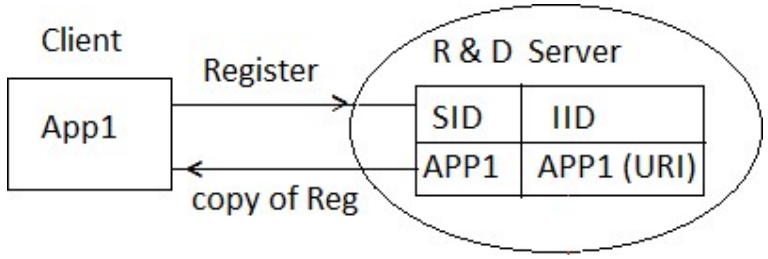
Work flow of Eureka and Producer:--

Step#1:- On startup Eureka server, one Registory is created (it is like a Map).

Step#2:- Every client is identified using serviced and their servers are identified using Instance Id (URI).

Step#3:- Client identifies R & D using Service-url and it is enabled by using @EnableEurekaClient.

Step#4:- Client gets registered first and then fetches the registery (copy).



5. Consumer Application:-- In general Spring Boot application, by using any HTTP Client code, Consumer makes request based on URL (static/hard coded) given in code.
=>This application need to be registered first with R & D server then fetch service registory using any one client component.given as
a>Discovery Client
b>Load Balancing Client (Feign client)

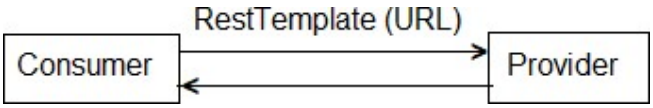
=>Every client component need "Service-Instance" object to communicate with producer application.
=>In general, one instance means One-service where application is running.

=>Every instance must have:

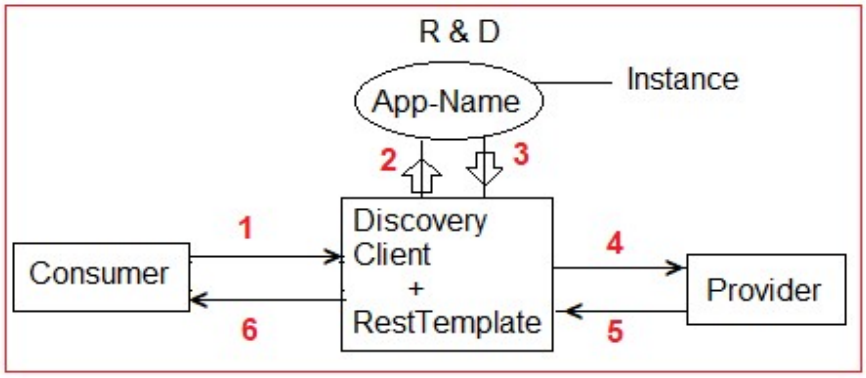
1. Service Id : Project Name
2. Instance Id : Server Instance name)
3. URI : Host, Port, …etc)
4. MetaData : (Headers, Cookies, Sessions…)
5. Current Load Factor… etc.

*** **Hard coding:--** Providing a direct value to a variable in .java file or fixed value.
Ex:- int a = 5;
=>It provides always same output for multiple runs.
=>By using RestTemplate with URL (hard coded) we can make request. But it will not support.
a>If provider IP/PORT number gets changed.
b>Load Balancing Implementation.
=>So, we should use Dynamic Client that gets URL at runtime based on Application name registered in "Registry and Discovery Server (Eureka)".
=>DiscoveryClient is used to fetch Instance based on Application Name and we can read URI of provider at runtime.
=>RestTemplate uses URI (+path)= URL and makes Request to Provider and gets ResponseEntity which is given back to the Consumer.

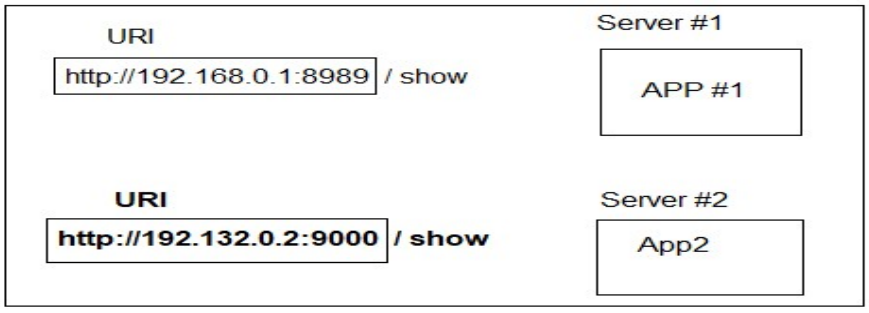
Simple Web Service Example:--



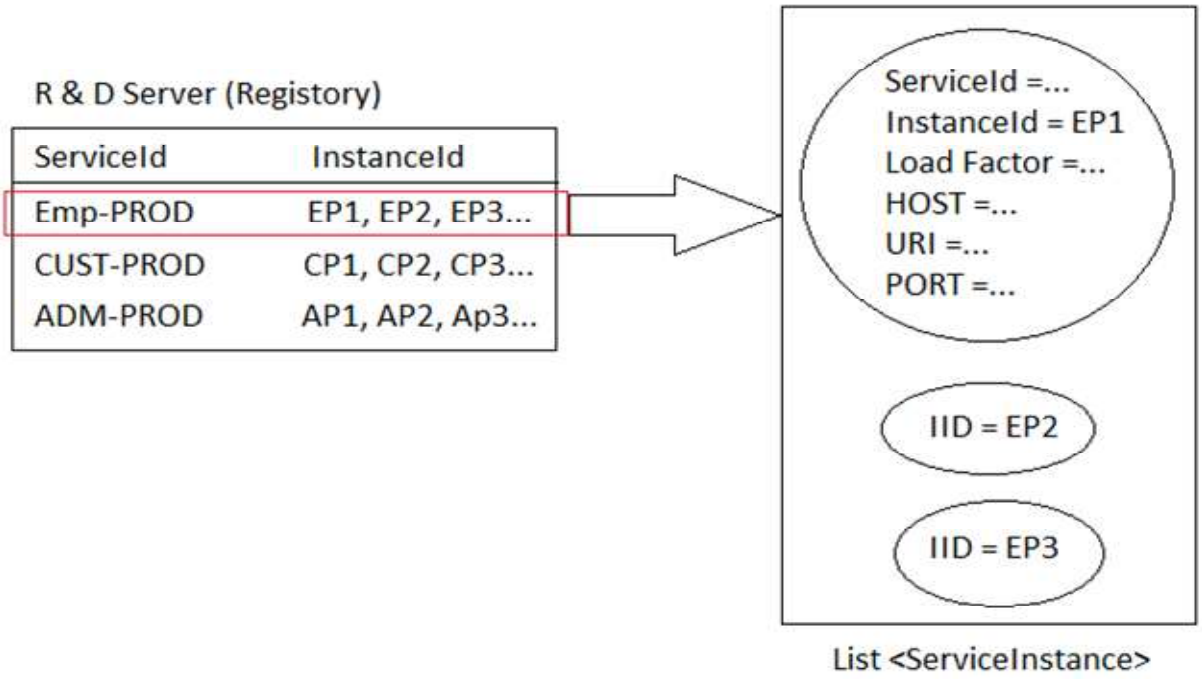
Microservices Example:--



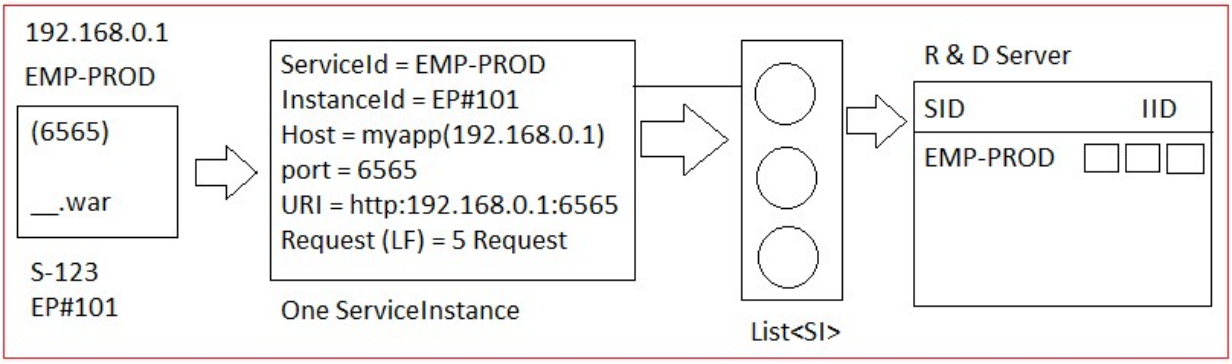
=>If one Application is moved from one Server to another server then URI gets changed (Paths remain same).



=>If one Project is running in 3 Server then current Project is having 3 instances. All these are store in R & D Server and we can fetch details as : List<ServiceInstance>



=>Consider one example producer application running in server 192.168.0.1 (IP).



Consumer code:--

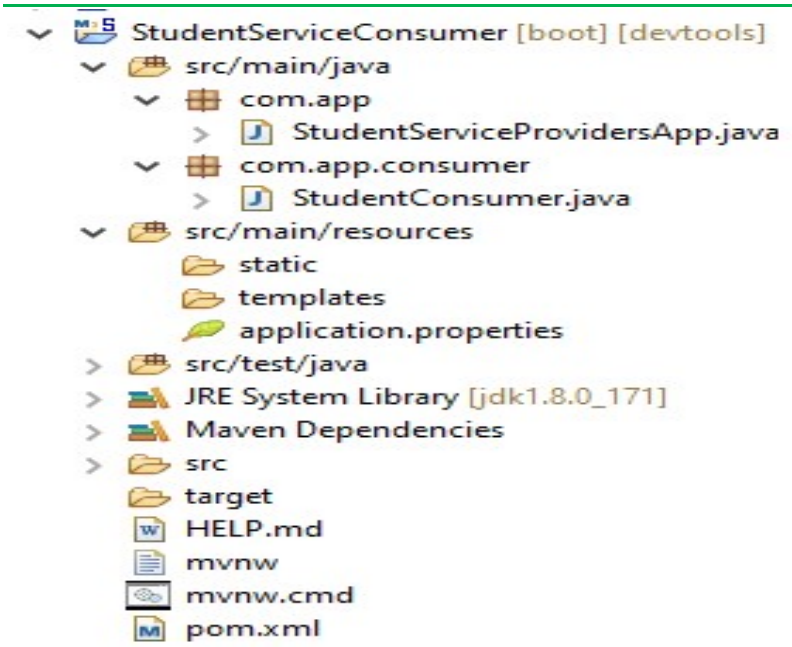
Step #1:- Create one Spring Starter Project using Dependencies web, Eureka Discovery.

GroupId : org.app

ArtifactId : StudentServiceConsumer

Version : 1.0

#5. Folder Structure of Consumer Application:--



Step #2:- At Starter class level add Annotation either **@EnableEurekaClient** or **@EnableDiscoveryClient** (both are optional)

Step #3:- In application.properties file

server.port=9852

spring.application.name=STUDENT-CONSUMER

eureka.client.serviceUrl.defaultZOne=http://localhost:8761/eureka

Step #4:- Define Consumer code with RestTemplate and DiscoveryClient:--

StudentConsumer.java:--

```
package com.app.consumer;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class StudentConsumer
```

```
{
        @Autowired
        private DiscoveryClient client;  //Where DiscoveryClient is a n Instance

        @GetMapping("/consume")
        public String consumeData ()
        {
                RestTemplate rt = new RestTemplate();
List<ServiceInstance> list=client.getInstances("STUDENT-PROVIDER");
ResponseEntity<String> resp =rt.getForEntity(list.get(0).getUri()+"/show", String.class);
                return "FROM CONSUMER=>" +resp.getBody();
        }
}
```

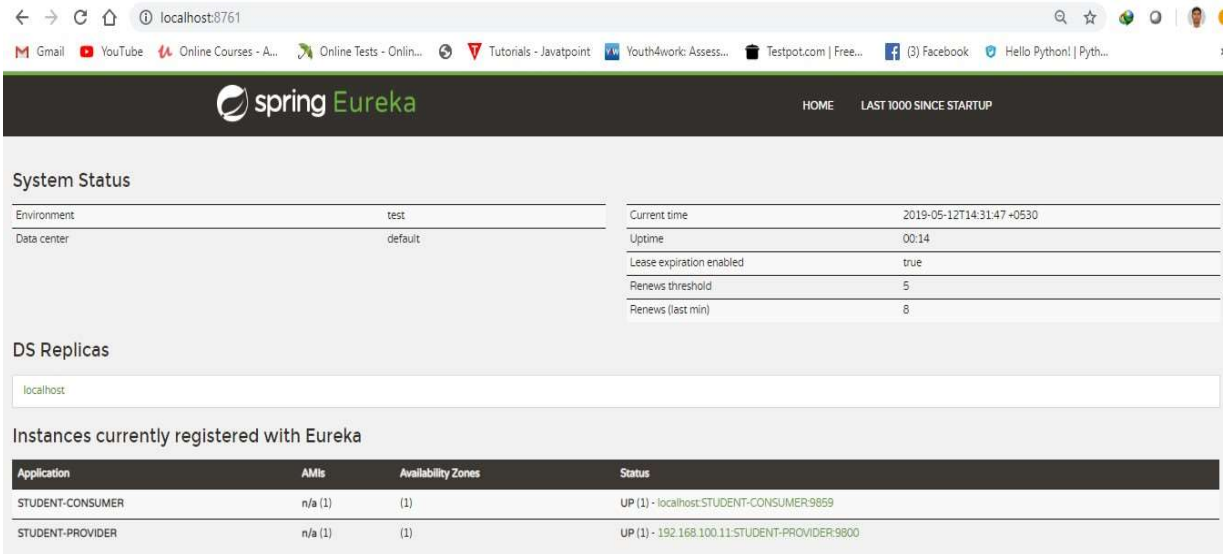
Execution order:--

#1 Run Starter classes in order:--

Eureka server, provider App, Consumer App

#2 Goto Eureka Server Dashboad and click on Client Consumer URI and enter **"/consume"** path after PORT number (http://192.168.100.39:9852/consume).

Screen#1 Eureka Server:--



6. Load Balancing in Spring Cloud:--

=>To handle multiple requests made by any HTTP client (or consumer) in less time, one provider should run as multiple instances and handle request in parallel. Such concept is called as Load Balancing.

=>Spring cloud has provided one Interface “Load Balance Client” which is used to define LBS (Load balancing server) Register.

=>This register holds Load factor over Instances and stores in a map format based on ServiceId. Load factor also called as current load over level over any instance.

=>Load balancing is to make request handling faster (reduce waiting time in queue).

Step to implement Load Balancing:--

a>Create one provider application.

b>Register and provider as multiple instances in Registry & Discovery [Eureka] server every instance with unique ID.

Ex:-- P1-58266, P2-2353424, P3-740986 etc.

c>Define consumer with any one Load Balancing Component (Ex:-- Ribbon, Feign).

d>Ribbon chooses one Provider URL, based on instance Id with the help of LBS register which maintains request count.

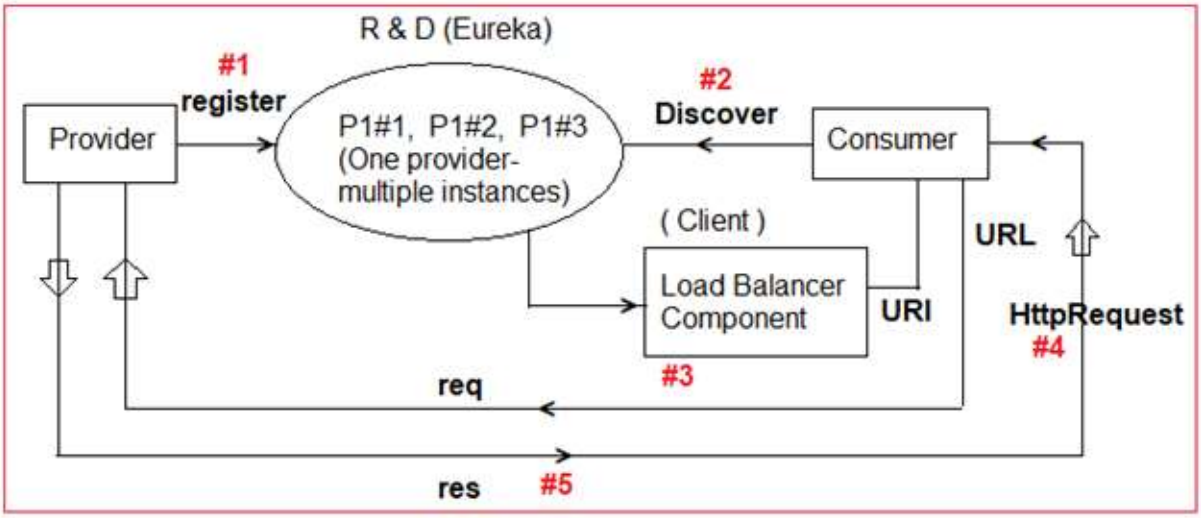
e>Consumer will uses paths to URL and makes Request using “RequestClient”.

[Ex:- LoadBalancerClient (I) or @FeignClient]

f>ResponseEntity is returned by Provider to Consumer.

5.1 Ribbon:-- It is a Netflix component provided for spring boot cloud Load Balancing.

=>It is also called as Client side load Balancing. It means Consumer App, reads URI (which one is free) using LBS register.



Load Balancer Component : Ribbon, Feign

Request Component :

LoadBalancerClient (I)

IS-A

RibbonLoadBalancerClient (C)

Temp memory by Ribbon

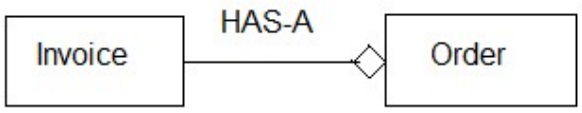
Load Balancer Server

Registry

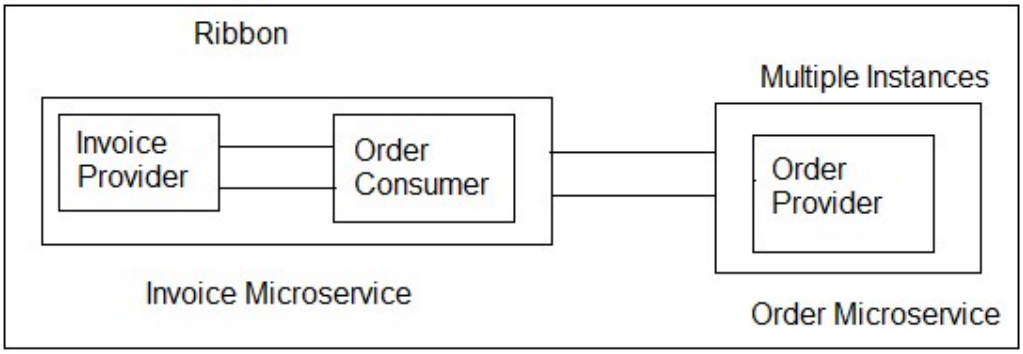
Instance Id	Req Count
P1#1	11
P1#2	10
P1#3	10

=>Ribbon should be enabled by every Consumer.
=>Spring cloud uses LoadBalancerClient (I) for choose an invoice process.
=>Its implementation is provided by Ribbon
=>Netflix has provided one Impl class for above Interface (**LoadBalancerClient**). It supports creating, updating, choosing, … etc of a LBS Register and Instances details.

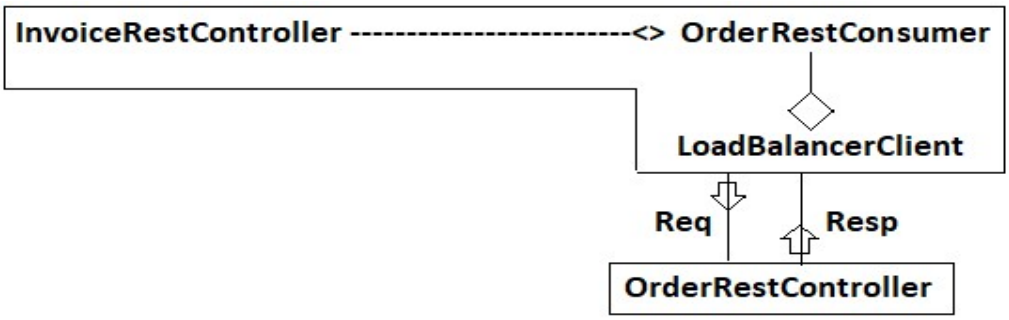
Consider below Example for Ribbon:--



=>Then two projects are created here.Order Application and Invoice Application.
=>Both apps contain their Service Producer codes (RestController).
=>This time child Producer will become ChildConsumer in ParentProducer application.
=>ChildConsumer and childProducer codes communicate using HTTP protocols.



NOTE:-- Ribbon at Consumer side



=>Consumer Application (Invoice Producer) needs LoadBalancerClient which is provided by Netflix-Ribbon. Below dependency must be provided in pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>
```

=>By default application behaves as ServiceId and InstanceId, but this time to provide multiple InstanceIds use below key in properties file.

```
eureka.instance.instance-id=
```

Example: application.properties:--

```
spring.application.name=ORDER-PRO
eureka.instance.instance-id=${spring.application.name}: ${random.value}
```

Coding Steps:--

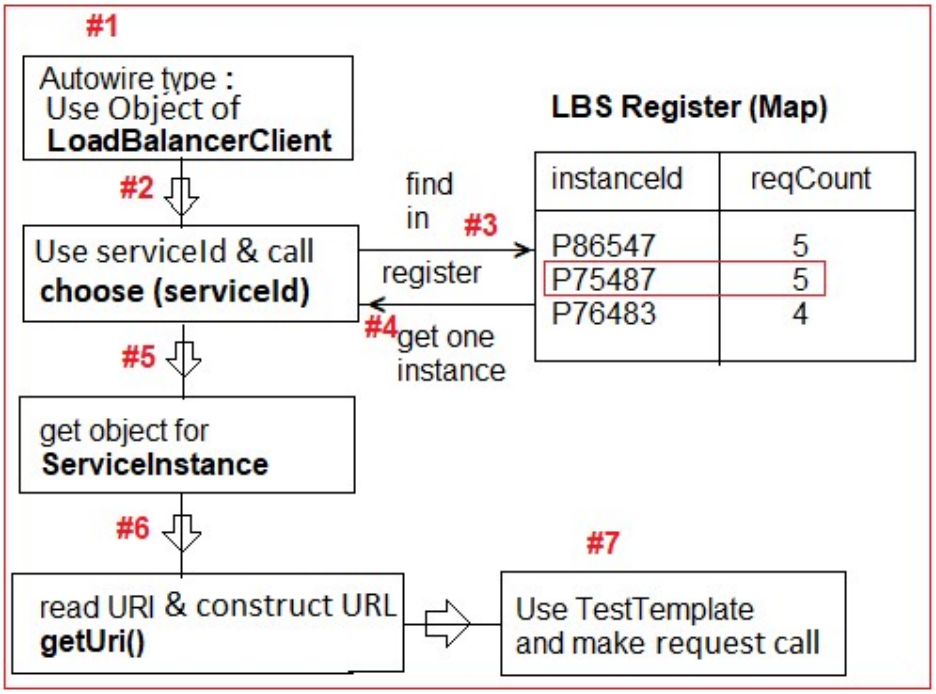
Step#1:- In provider, define Instance Id for Provider Application using key. **"eureka.instance.intance-id"**. If not provided default is taken as Spring App name.

```
eureka.instance.instance-id=${spring.application.name}: ${random.value}
```

[Add in application.properties]

Step#2:- In consumer, add dependency for Ribbon and use LoadBalancerClient (Autowired) and call choose ("App-Name") method to get serviceInstance (for URI)

Consumer Execution flow for Request:--



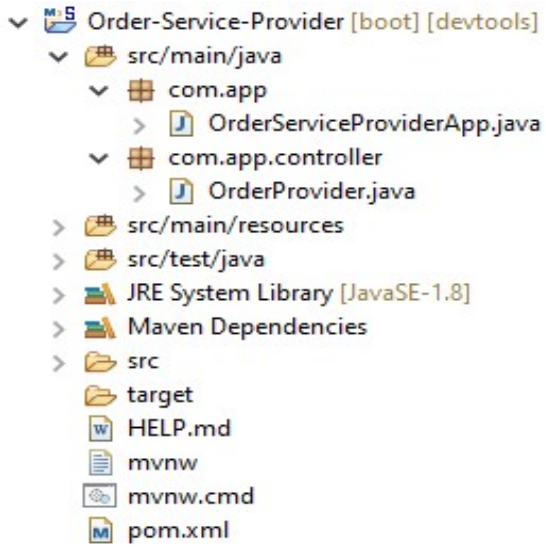
APPLICATION #1:- Configure Eureka Server Dependencies : Eureka Server only

application.properties:--

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

APPLICATION#2:- Create Order Service Provider Application(Child) Dependencies : Eureka Discovery, Web.

#6. Order Service Provider:--



application.properties:--

```
server.port=9800
spring.application.name=ORDER-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

=>If no instance-id is provided then application name (service Id) behaves as instance Id.

#1. Starter class:--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import  org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class OrderServiceProviderApp
 {
        public static void main(String[] args)
        {
                SpringApplication.run(OrderServiceProviderApp.class, args);
        }
}
```

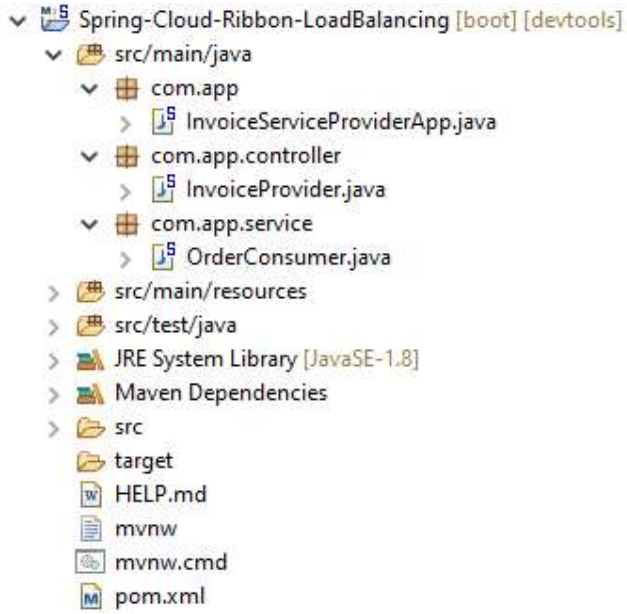
#2. Controller:--

```
package com.app.controller;
import  org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("order")
public class OrderProvider {

	@Value("${server.port}")
	private String port;

	@GetMapping("/status")
	public String getOrderStatus() {
		return "FINISHED:\"Hello from Order Provider\":"+port;
	}
}
```

APPLICATION#3:- Define Invoice Service Consumer(Parent):--

Dependencies : Eureka Discovery, Web, Ribbon

#7. Invoice Service Consumer:--



Ribbon Dependency:--

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
</dependency>
```

application.properties:--

```
server.port=8802
spring.application.name=INVOICE-CONSUMER
eureka.client.serviceUrl.defaultZone     =http://localhost:8761/eureka
```

#1. Consumer starter class:--

#2. Consumer code(OrderConsumer.java):--

```
package com.app.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class OrderConsumer {
        @Autowired
        private LoadBalancerClient client;
        public String getStatus()  {
                String path="/order/status";
                //Choose Service instance based on SID
                ServiceInstance instance=client.choose("ORDER-PROVIDER");
                //Read URI from instance
                String  uri=instance.getUri().toString();
                //Make http Request
                RestTemplate  rt = new RestTemplate();
                ResponseEntity <String>  resp=rt.getForEntity(uri+path, String.class);
                return "CONSUMER=>"+resp.getBody();
        } }
```

#2 Controller code (InvoiceProvider):--

```
package com.app.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.service.OrderConsumer;

@RestController
@RequestMapping("/invoice")
public class InvoiceProvider {

        @Autowired
        private OrderConsumer consumer;

        @GetMapping("/info")
        public String getOrderStatus() {
            return consumer.getStatus();
         }
}
```

Execution:--

a>Start Eureka Server

b>Run OrderProvider starter class 3 times

*** Change every time Port number like : 9800,9801, 9802

c>Run InvoiceProvider Starter class

d>Goto Eureka server Dashboad and Execute Invoice Consumer Instance and type full URL http://localhost:8800/invoice/info

OutPut:--

5.2 Declarative ReSTClient : [Feign Client]:--

Spring cloud supports any HTTP Client to make communication between (Microservices) Provider and Consumer.

=>RestTemplate is a legacy style which is used to make HTTP calls with URL and Extra inputs. Feign Client reduces burden on programmer by avoiding legacy style coding for RestTemplate.

=>RestTemplate with DiscoveryClient makes mask to Provider URL. It means works based on Application Name (Service ID). Even URI gets changed it works without any modification at consumer side.

=>RestTemplate combination always makes Programmer to write manual coding for HTTP calls.

=>Spring Cloud has provided one Action Client [which behaves as Client, but not]. It means, Provide Abstraction at code level by programmer and Implementation is done at runtime by Spring cloud.

=>It is Http client used to create communication links between Consumer and Producer application.

->It is provided by Netflix.
->It works based on Proxy design Pattern.
->At runtime Http calls logic is implemented.
->Supports Load balancing Also.
->Internally uses Ribbon only.
->It is called as Declarative Client.

=>By taking above Inputs Http calls are implemented using one Instance (URI).

=>Feign is **Declarative Client**, which supports generating code at runtime and **proxy Objects** by using **Proxy HTTP Request** calls can be made.

=>It supports even Parameters (Path/Query…) and Global Data Conversion (XML/JSON).

Example Feign client looks like:--

```
@FeignClient("TRACT-PROD")
public interface TrackConsumer  {
    @GetMapping("/tract/info")
    public String getModel();
    @PostMapping("track/insert")
    public String create(@RequestBody Track tr);
  }
```

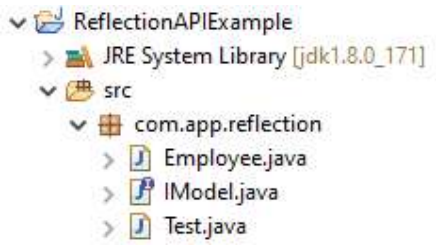
=>It is similar to rest controller but it is **interface** with **abstract** methods.

Proxy:-- It is a code/Object generated at runtime which acts as actual one, but not exist as physical file.

->Proxy Type (Proxy Class)
->Proxy Data (Proxy Objects)

=>Proxies are designed using **CGLibs and Reflection**.
=>Javapoet is a child type of CGLibs(Code Generator Library).

1. Reflection API Example(Normanl Java Application):--



1. IModel interface:--

```
package com.app.reflation;

public interface IModel {
	public String getModelName();
	public IModel getModelObject();
}
```

2. Employee Class:--

```
package com.app.reflation;

public class Employee implements IModel {

	public Employee() {
		System.out.println("Employee Constructor...");
	}
	@Override
	public String getModelName() {
		return "Employee";
	}
	@Override
	public IModel getModelObject() {
		return this;
	}
```

```
	public String toString(){
		return "FROM EMPLOYEE OBJECT";
	}
}
```

3. Test class:--

```
package com.app.reflation;

public class Test {

	public static void main(String[] args)throws Exception {
		//load runtime class
		Class c = Class.forName(args[10]);
		//Create instance
		Object ob = c.newInstance();

		//Downcast to IModel Type
		if(ob instanceof IModel){
			IModel m =(IModel)ob;
			System.out.println(m.getModelName());
			System.out.println(m.getModelObject());
		}
	}
}
```

=>Right click > Run As > Run Configuration

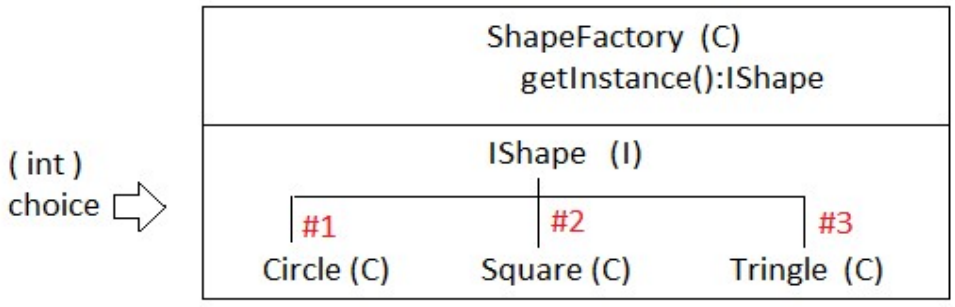
->Click on Arguments > Progarmming args

->Enter Ex: com.app.Employee >Apply & Run

Static Factory Design Pattern using Reflection API and JDK 1.8 Interface features:--

->Input (int)

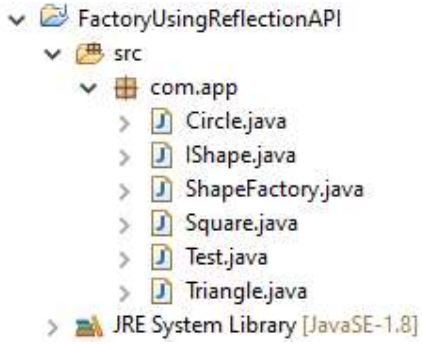
1. Circle
2. Square
3. Triangle



Output:-- Object of class and calling methods.

=>Factory Components
1. IShape (I)
2. ShapeFactory (C)

2. Folder Structure of Factory Dessign Pattern Using Reflection API:--



Code:--

1. IShape Interface:--

```
package com.app;

public interface IShape {

	//must be overridden
	public void showInfo();
	//can be overridden
	public default void shapeMsg() {
		System.out.println("Welcome to ShapeFactory");
	}
	//can not be overridden
	public static void commonMsg() {
		System.out.println("Hello Shape..");
	}
}
```

2. Circle Class:--

```
package com.app;

public class Circle implements IShape {
	public Circle() {
		System.out.println("circle is created..");
	}
```

```
	public void showInfo() {
		System.out.println("This is a circle Object");
	}
	@Override
	public void shapeMsg() {
		System.out.println("Message from circle");
	}
}
```

3. Square Class:--

```
package com.app;

public class Square implements IShape {

	public Square() {
		System.out.println("Square object is created..");
	}
	public void showInfo() {
		System.out.println("This is Square Shape ");
	}
	@Override
	public void shapeMsg() {
		IShape.super.shapeMsg();
		System.out.println("Also Square message Printed..");
	}
}
```

4. Triangle Class:--

```
package com.app;

public class Triangle implements IShape {

	public Triangle() {
		System.out.println("triangle object is created");
	}
	public void showInfo() {
		System.out.println("THis is triangle object");
	}
}
```

5. ShapeFactory Class:--

```
package com.app;

public class ShapeFactory {
```

```
	public static IShape getShape(int ch){
		String cls=choose(ch);
		IShape shpae=null;
		try {
			Object ob=Class.forName(cls).newInstance();
			if(ob instanceof IShape) {
				shpae=(IShape)ob;
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
		return shpae;
	}

	private static String choose(int ch) {

	    String cls=null;
		switch (ch) {
		case 1:
			cls="com.app.Circle";
			break;
		case 2:
			cls="com.app.Square";
			break;
		case 3:
			cls="com.app.Triangle";
			break;
		default:
			break;
		}
		return cls;
	}
}
```

6. Test Class:--

```
package com.app;

import java.util.Scanner;

public class Test {

	public static void main(String[] args) {
```

```
                Scanner sc=new Scanner(System.in);
                System.out.println("Please Choose One option");
                System.out.println("1.Circle");
                System.out.println("2.Square");
                System.out.println("3.Triangle");
                int ch=sc.nextInt();
                if(ch>0 && ch<4) {
                        IShape shape=ShapeFactory.getShape(ch);
                        shape.showInfo();
                        shape.shapeMsg();
                        IShape.commonMsg();
                }else {
                        System.out.println("invalid selection");
                }sc.close();
        }
}
```

Execution process:--

1>Run Test class and Enter number 1-3 and see the Output on Console.

2. Export Project as Executable:--

=>Right Click on Project > Export
=>Search Using “JAR” word
=>Choose “Runnable JAR File”
=>Select Main class to launch
=>Browse for Location and Enter Jar name ex:- d/:myapps/factory.jar
=>Finish

Execute Jar:--

=>Goto Location where JAR is created
=>Shift + Right Click
=>Open cmd Window Here
=>type below command
‘java –jar factory.jar’

Create a windows batch file:--

=>Open notepad =>Type below commands

```
java  -jar factory.jar
```

=>and save with .bat extension

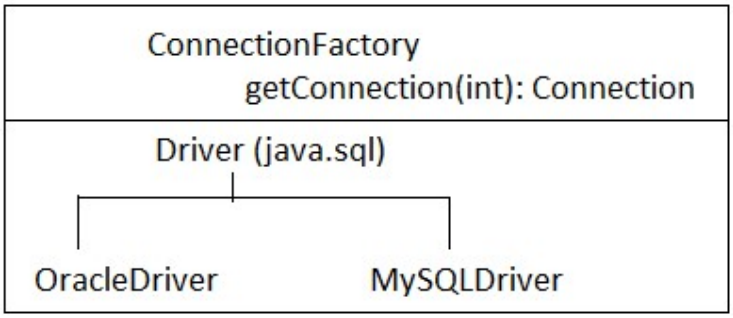
Myapp.bat:--

```
java  -jar factory.jar
```

pause;

=>Double click on batch and enter inputs like (1, 2, 3)

Task:--



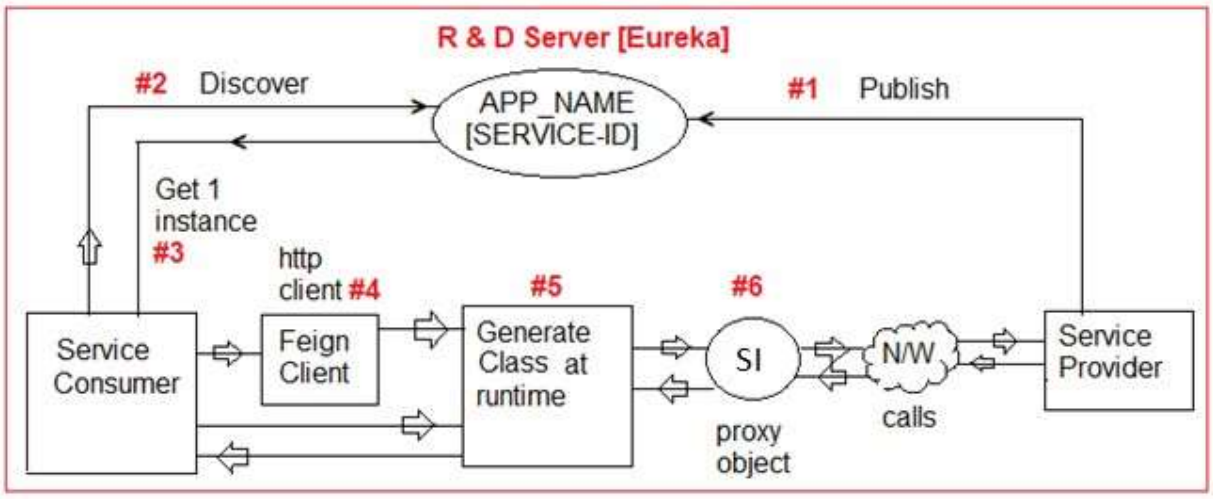
Work Flow Design:--

1. Interface defined with **@FeignClient** validates all details (ServiceId, Paths, MethodTypes…).
2. Uses ServiceId and communicates with R&D it gets LBSR (Load Balancing).
3. A Proxy class is generated which makes all this process. Based on load Factor one Service Instance is choosen.

***Ribbon is used internally.

4. Http Logic is generated using Service Instance (URI).
5. Proxy class supports making Http Request and Response (Using dynamic SI).

Work Flow Design:--



=>Feign Client is an Interface and contains abstraction details, it takes I/P details from programmer like:

a>path (Provider Paths at class and method).

b>Http Method Type (GET, POST…).

c>ServiceId (Application Name).

d>Input Details and Output Type (String, Object, Collection).

e>Parameters (Request, Path,…).

=>We need to apply Annotation at starter class level **@EnableFeignClients.**

=>At interface level apply **@FeignClient(name="serviecId").**

Syntax:-- Feign Client

```
@FeignClient(name="serviceId")
public interface <ClientName> {

        @GetMapping("/path")
        //or @RequestMapping("/path")
        public <Return> <method>(<Params>);
   ...
}
```

Example:--

Provider Code (SID : EMP-PROV):--

```
com.app.rest;

@RestController
@RequestMapping("/emp")
public class EmpProvider {

        @GetMapping("/show")
        public String findMsg() {
                ……
        }
}
```

Consumer Code : Feign Client

```
@FeignClient(name="EMP_PROV")
public interface EmpConsumer {

        @GetMapping ("/emp/show")
        public String getMsg();  //return type and path must be same as Provider
}
```

1. Consider last Ex Eureka Server and Provider Application (ORDER-PROVIDER):--

Step#1:- Create one Spring Boot Starter Project for Consumer (using Feign, web, Eureka Discovery)

Feign Client Dependency:--

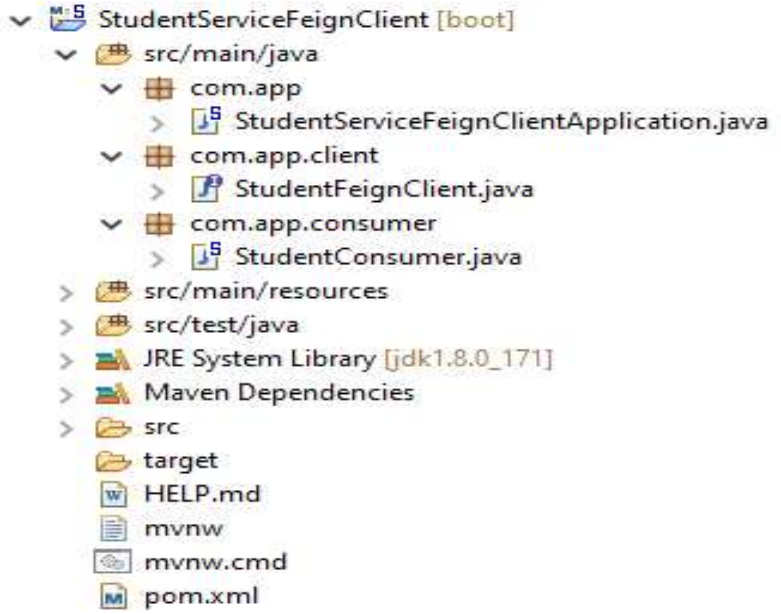
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Eureka Discovery Client Dependency:--

```
<dependency>
     <groupId>org.springframework.cloud</groupId>
     <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

GroupId : org.app
ArtifactId : StudentServiceConsumerFeign
Version : 1.0

#8. Folder Structure of Consumer Application with Declarative RestClient:--



Step#1:- Starter class for Feign client:--

```
package com.app;
import   org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class StudentServiceFeignClientApp {

	public static void main(String[] args) {
		SpringApplication.run(StudentServiceFeignClientApp.class, args);
		System.out.println("Student Consumer Service executed");
	}
	@Bean
	public RestTemplate restTemplate(RestTemplateBuilder builder) {
	   return  builder.build();
	}
}
```

Step#2:- Define one public interface as

```
package com.app.client;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;

//Name must be same as Provide Service-Id
@FeignClient(name="STUDENT-PROVIDER")
public interface StudentFeignClient {

	@GetMapping("order/status")
	public String getMsg();    //Path and Return type same as Provider method
}
```

Step#3:- Use in any consumer class (HAS-A) and make method call (HTTP CALL)

```
package com.app.consumer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.client.StudentFeignClient;
```

```
@RestController
public class StudentConsumer {
	@Autowired
	private StudentFeignClient client;

	@GetMapping("consume")
	public String showData()
	{
		System.out.println(client.getClass().getName());
		return "CONSUMER=>"+client.getMsg();
	}
}
```

NOTE:-- Here client.getMsg() method is nothing but HTTP Request call.

Execution Order:--

1>Start Eureka Server

2>Run Provider Application

3>Run Consumer Application and click on consumer App service id on Eureka Server Dashboad and provider consumer method level path:

http://192.168.100.27:9841/consume

5.2 Load Balancing using Feign Client:--

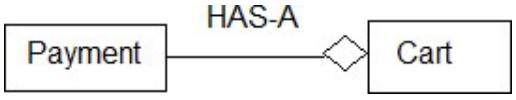
Incase of Manual Coding for load Balancing Ribbon Component is used with Type “LoadBalancingClient” (I).

=>Here, using this programmer has to define logic of consumer method.

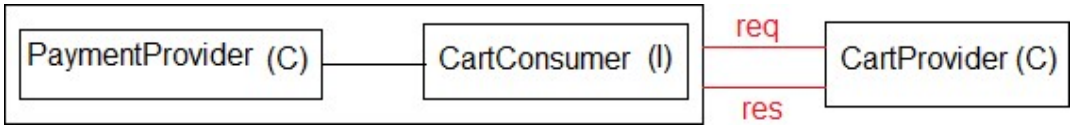
=>Feign Client reduces coding lines by Programmer, by generating logic/code at runtime.

=>Feign Client uses abstraction Process, means Programmer has to provide path with Http Method Type and also input, output details.

=>At Runtime RibbonLoadBalancerClient instance is used to choose serviceInstance and make HTTP call.



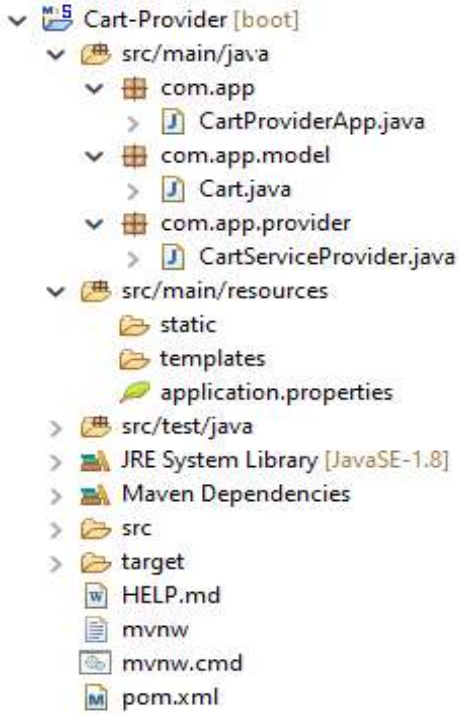
Feign Client:-



Step#1:- Create Spring Starter Project for Eureka Server with port 8761 and dependency **"Eureka Server".**

Step#2:- Create Cart provider Application
Dependencies : web, Eureka Discovery.
GroupId : com.app
ArtifactId : Cart-Provider
Version : 1.0

#9. Folder Structure of LoadBalancing Using Feign Client (Cart-Provider):--



Step #3:- application.properties
server.port=8600
spring.application.name=CART-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
eureka.instance.instance-id=${spring.application.name}:${random.value}

Step #4:- At starter class level : @EnableDiscoveryClient

1>CartProviderApplication.java:--

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import  org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class CartProviderApplication {
        public static void main(String[] args) {
                SpringApplication.run(CartProviderApplication.class, args);
        }
 }
```

2>Model class (Cart.java):--

```
package com.app.model;
//import lombok.Data;

//@Data
public class Cart {

        private Integer cartId;
        private String cartCode;
        private Double cartFinalCost;

        //Default Constructor
        public Cart() {
                super();
        }
        //Parameterized constructor
        public Cart(Integer cartId, String cartCode, Double cartFinalCost) {
                super();
                this.cartId = cartId;
                this.cartCode = cartCode;
                this.cartFinalCost = cartFinalCost;
        }
```

```
	//Set/get Method
	public Integer getCartId() {
		return cartId;
	}
	public void setCartId(Integer cartId) {
		this.cartId = cartId;
	}
	public String getCartCode() {
		return cartCode;
	}
	public void setCartCode(String cartCode) {
		this.cartCode = cartCode;
	}
	public Double getCartFinalCost() {
		return cartFinalCost;
	}
	public void setCartFinalCost(Double cartFinalCost) {
		this.cartFinalCost = cartFinalCost;
	}

	@Override
	public String toString() {
		return "Cart [cartId=" + cartId + ", cartCode=" + cartCode + ",
				cartFinalCost=" + cartFinalCost + "]";
	}
}
```

3>Cart Service Provider code:--

```
package com.app.provider;
import java.util.Arrays;
import java.util.List;
import  org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Cart;
```

```
@RestController
@RequestMapping("/cart")
public class CartServiceProvider
{
	@Value("{server.port}")
	private String port;

	@GetMapping("/info")
	public String getMsg()  {
		return "CONSUMER:"+port;
	}

	@GetMapping("/data")
	public Cart getObj() {
		return new Cart(109, "ABC: "+port, 7868.98);
	}

	@GetMapping("/list")
	public List<Cart> getObjs() {

		return Arrays.asList(
		    new Cart(101, "A :"+port, 876.98),
		    new Cart(102, "B :"+port, 856.98),
		    new Cart(103, "C :"+port, 883.98));
	}
 }
```

Step #3:- Payment Provider App with Cart Consumer code

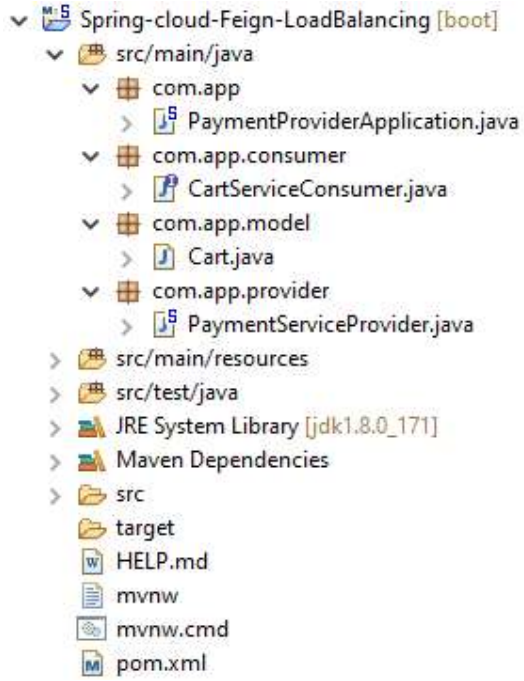
Dependencies : web, Eureka Discovery, Feign

Feign Dependency:--

```
<dependency>
	<groupId>org.springframework.cloud</groupId>
	<artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

GroupId : org.app
ArtifactId : Payment-Provider
Version : 1.0

#10. Folder Structure of Payment-Provider Service:--



1>application.properties:--

```
server.port=9890
spring.application.name=PAYMENT-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

2>At Starter class level add annotation : @EnableFeignClients**

(PaymentProviderApplication.java):--

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
public class PaymentProviderApplication {
```

```
	public static void main(String[] args) {
		SpringApplication.run(PaymentProviderApplication.class, args);
	}
}
```

3>Cart Service Consumer (CartServiceConsumer):--

```
package com.app.consumer;
import java.util.List;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import com.app.model.Cart;

@FeignClient(name="CART-PROVIDER")
public interface CartServiceConsumer
{
	@GetMapping("/cart/info")
	public String getMsg();

	@GetMapping("/cart/data")
	public Cart getObj();

	@GetMapping("/cart/list")
	public List<Cart> getBulk();
}
```

4>Model class (Cart.java):-- Cart Model class same as above Project

```
package com.app.model;

public class Cart {
	private Integer cartId;
	private String cartCode;
	private Double cartFinalCost;

	//Default Constructor
	public Cart() {
		super();
	}
```

```
	//Parameterized constructor
	public Cart(Integer cartId, String cartCode, Double cartFinalCost) {
		super();
		this.cartId = cartId;
		this.cartCode = cartCode;
		this.cartFinalCost = cartFinalCost;
	}
	//Set/get Method
	public Integer getCartId() {
		return cartId;
	}
	public void setCartId(Integer cartId) {
		this.cartId = cartId;
	}
	public String getCartCode() {
		return cartCode;
	}
	public void setCartCode(String cartCode) {
		this.cartCode = cartCode;
	}
	public Double getCartFinalCost() {
		return cartFinalCost;
	}
	public void setCartFinalCost(Double cartFinalCost) {
		this.cartFinalCost = cartFinalCost;
	}
	@Override	//toString method
	public String toString() {
		return "Cart [cartId=" + cartId + ", cartCode=" + cartCode + ",
cartFinalCost=" + cartFinalCost + "]";
	}
}
```

5>PaymentServiceProvider.java:--

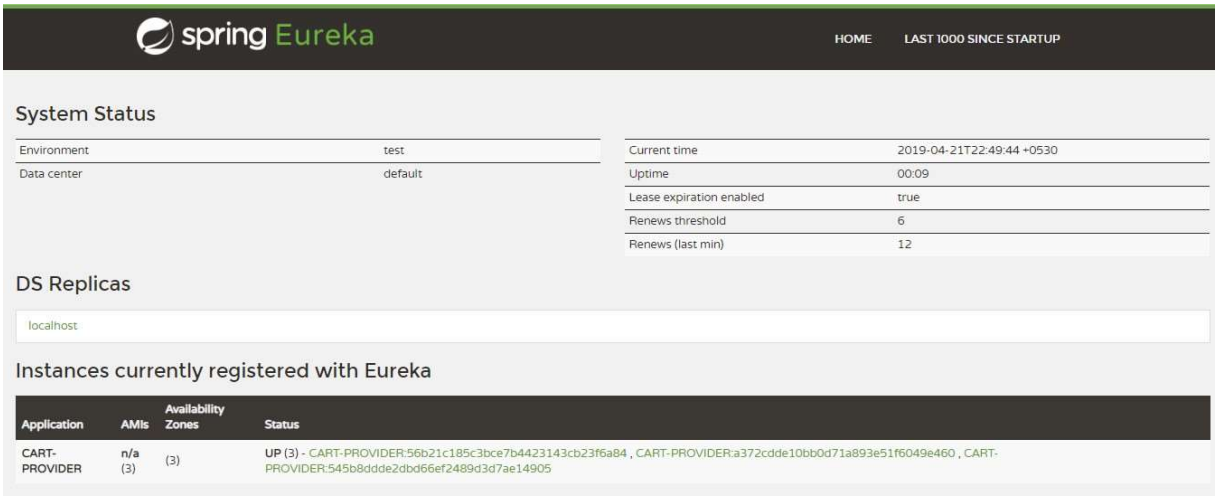
```
package com.app.provider;
import java.util.List;
import	org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.consumer.CartServiceConsumer;
import com.app.model.Cart;
@RestController
@RequestMapping("/payment")
public class PaymentServiceProvider {

	@Autowired
	private CartServiceConsumer consumer;

	@GetMapping("/message")
	public String getMsg() {
		return consumer.getMsg();
	}
	@GetMapping("/one")
	public Cart getOneRow()  {
		return consumer.getObj();
	}
	@GetMapping("/all")
	public List<Cart> getAllRows() {
		return consumer.getBulk();
	}
}
```

Step #4 Execution Order:--

=>Run Eureka Server

=>Run Cart Provider 3 times (with different port)



=>run Payment Provider 1 time

=>Goto Eureka server and Run Payment service and provide bellow URLs one by one.

1>http://192.168.100.17:9890/payment/message

2>http://192.168.100.17:9890/payment/one

3>http://192.168.100.17:9890/payment/all

7. Spring Cloud Config Server:--

It is also called as Configuration Server. In Spring Boot/Cloud Projects, it contains files like : Properties files [application.properties]

=>In every application (Microservice) properties/yml file contain setup related keys like. (Database, ORM, Batch, Email, AOP, Security, Eureka Server, Gateway… etc) in the form of key=value.

=>In some cases Key=Value need to be changed or new key=value need to be added. At this time, we should

->Stop the Server (Application)
->Open/find application.properties file
->add External key=value pairs or do modifications.
->save changes [save file]
->Re-build file [re-create jar/war]
->Re-Deploy [re-start server]

=>In case of multiple microservices these common keys related for every Project. And this is done in All related Project [Multiple Microservices] which is repeated task for multiple applications.

=>**In this case, Config Server concepts is used
**To avoid this repeated (or lengthy) process use application.properties this is placed outside of your Project i.e. known as “config Server”.
=>Writing common properties only one time outside of all project.
=>Easy to modify common properties (one place to modify properties files that effects all projects).
=>Every service instances not required to **stop** restart for effect for properties file modifications (Refersh Scope).

=>Config server Process maintains three (3) properties file. Those are:

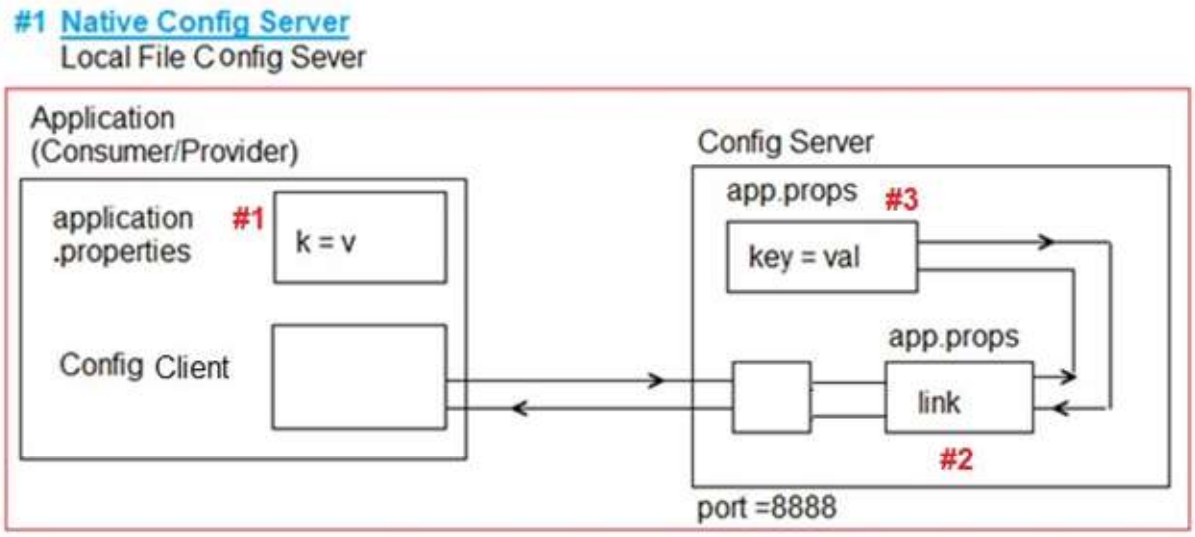
a>One in = Under Project (Microservice)
b>One in = Config Server (link file)
c>One in = Config server (External) also called as Source file.

Types of Config Server:--

Spring Cloud has provided Setup and support for configuration properties. It can be handled in two ways. Those are

a>Native Config Server
b>External Config Server

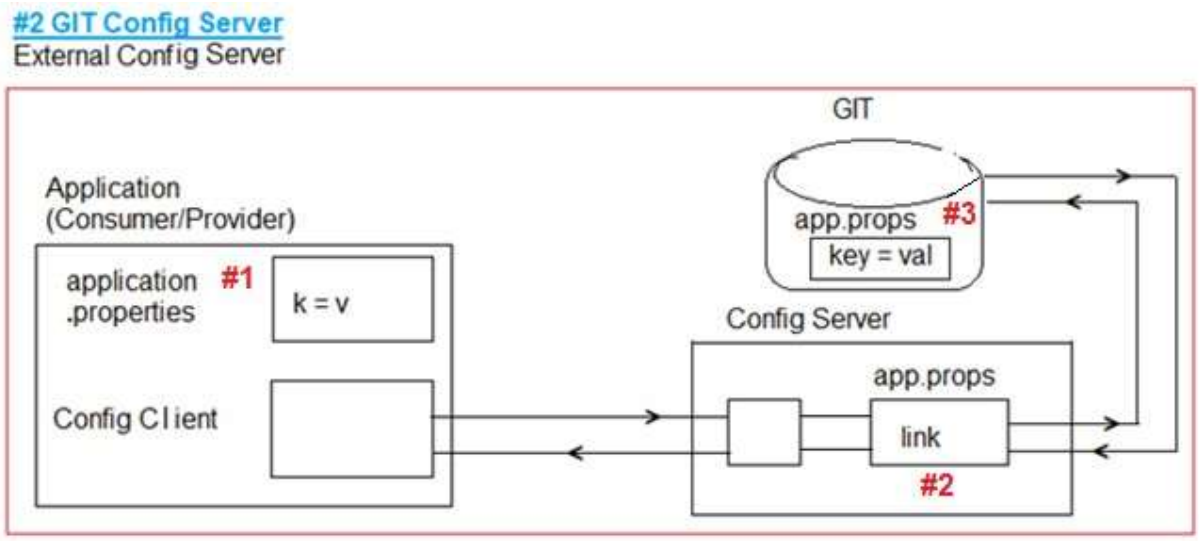
a>Native Config Server:-- It is also called as local file placed in Config server only. In this case one properties file is created inside folder System (or drive).
Ex:-- D:/abc/myapp(Or Under config server location file).



b>External Config Server:-- In this case properties file is placed outside the Config server. Ex: GIT (github).

=> Here, Properties file is placed in External and accesses using its URL. It is also called as Global Location.

**"GIT HUB" is used in this process.



=>In Consumer/Producer Project we should add Config Client dependency which gets config server details at runtime.

=>Config server runs at default port=**8888.**

Step to implements Spring Cloud Config Server-Native & External:--

Step#1:- Create Spring Boot Starter Project for "Config-Server" with dependency : Config Server.

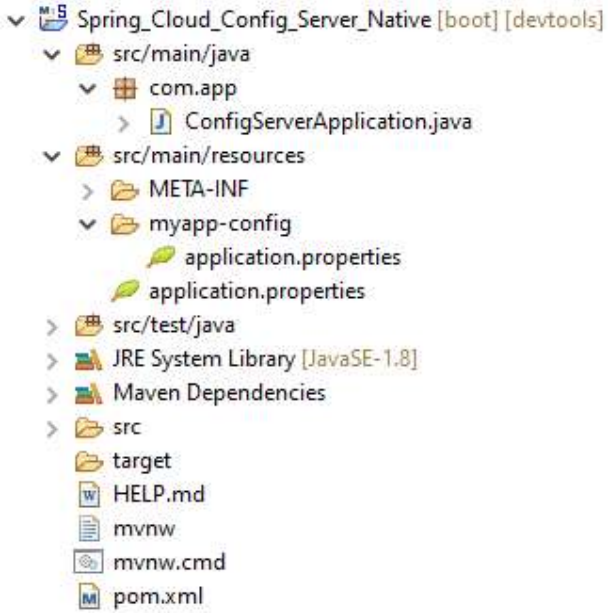
Config Server Dependency:--

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

GroupId : org.app
ArtifactId : Config-Server
Version : 1.0

7.1 Native Config Server:--

#11. Spring cloud config server component using Native:--



Step#2.a:- Native application.properties:-- Create folder "myapp-config" under src/main/resources.

```
server.port=8888
spring.profiles.active=native
spring.cloud.config.server.native.search-locations=classpath:/myapp-config
```

Case#2.b:- External application.properties

server.port=8888

spring.cloud.config.server.git.uri=https://github.com/javabyraghu/configserverex

Step#3:- Create sub folder “myapp-config” in src/main/resources folder.

=>Create file under “myapp-config” **application.properties** inside this folder. It behaves like source

Having ex key: eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

Or

Create Git Project “configserverex” and create application.properties inside this. Place above key and save file.

Step#4:- Provider include resource code in pom.xml to consider properties files to be loaded into memory.

```
<resources>
    <resource>
        <filtering>true</filtering>
        <directory>src/main/resources</directory>
        <includes>
            <include>*.properties</include>
            <include>myapp-config</include>
        </includes>
   </resource>
</resources>
```

pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
            <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.app</groupId>
    <artifactId>Config-Server</artifactId>
    <version>1.0</version>
    <name>Config-Server</name>
    <description>Demo project for Config-Server</description>
    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-config-server</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
```

```
                </dependency>
            </dependencies>
        </dependencyManagement>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
            <resources>
                <resource>
                    <filtering>true</filtering>
                    <directory>src/main/resources</directory>
                    <includes>
                        <include>*.properties</include>
                        <include>myapp-config</include>
                    </includes>
                </resource>
            </resources>
        </build>
</project>
```

Step#5:-- Starter class of ConfigServer App, add annotation: **@EnableConfigServer**

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {
        public static void main(String[] args) {
                SpringApplication.run(ConfigServerApplication.class, args);
                System.out.println("Config Server Executed");
        }
```

}

Step#6:-- In Consumer/Provider Projects pom.xml file add dependency : Config Client (or copy below dependency)

Config Client Dependency:--

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config-client</artifactId>
</dependency>
```

Execution Order:--

=>Eureka Server

=>Config Server

=>Producer (Provider)

=>Consumer

Step#1:- Eureka Server Project Dependency : Eureka Server

=>Write application.properties.

=>Add @EnableEurekaServer annotation

Step#2:- Define Config Server Project Dependency : Config Server

=>Write application.properties

=>Add **@EnableConfigServer** annotation

Step#3:- Create Provider Project (Order)

Dependency : Web, Eureka Discovery, Config Client…

=>Write application.properties

=>Add @EnableDiscoveryClient annotation

=>Write Provider (RestController)

Step #4:- Create Consumer Project (Invoice)

Dependency : Web, Eureka Discovery, Config Client, Feign….

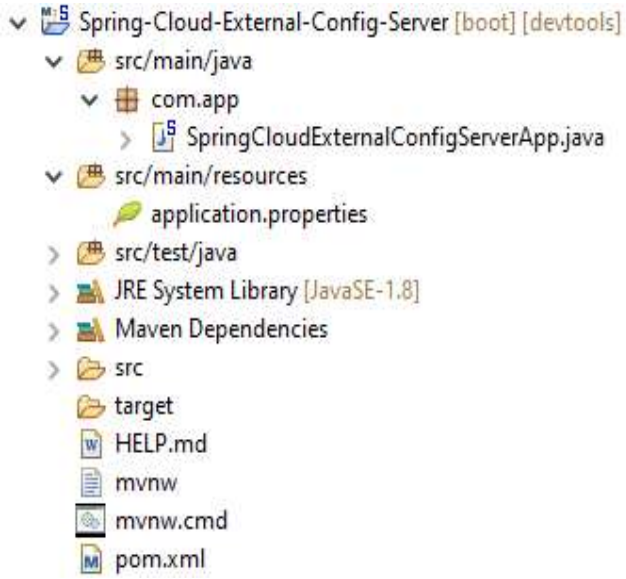
=>Write application.properties

=>Add @EnableFeignClient annotation

=>Write Consumer [Feign Client] and Invoice Provider (RestController)

7.2 External Config-Server:--

#12. Folder Structure of External Config Server:--



Step to configure External Config Server (only modifications):--

Step#1:- In Config Server Project, delete folder myapp-config

Step#2:- In Config Server Project modify application.properties with Git URI

server.port=8888

spring.cloud.config.server.git.uri=https://github.com/javabyraghu/configserverex

or

spring.cloud.config.server.git.uri=https://gitlab.com/udaykumar0023/configserverex

Step#3:- In config Server Project, in pom.xml delete line

<include>myapp-config </include>

pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.app</groupId>
    <artifactId>Spring-Cloud-External-Config-Server</artifactId>
    <version>1.0</version>
    <name>Spring-Cloud-External-Config-Server</name>
  <description>Project for Spring config server</description>

    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-config-server</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
```

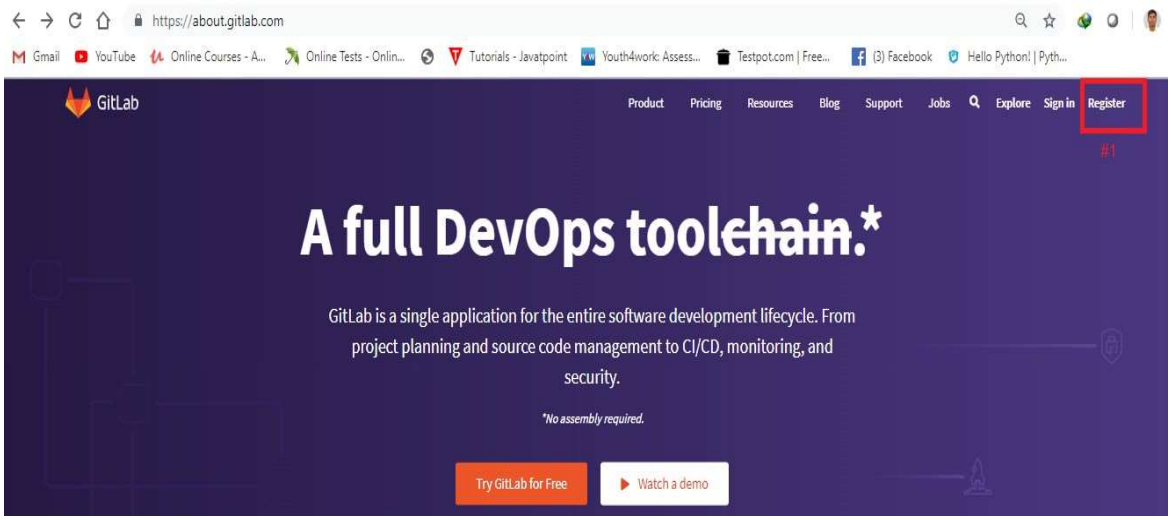
```
            </dependency>
        </dependencies>

        <dependencyManagement>
            <dependencies>
                <dependency>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-dependencies</artifactId>
                    <version>${spring-cloud.version}</version>
                    <type>pom</type>
                    <scope>import</scope>
                </dependency>
            </dependencies>
        </dependencyManagement>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
            <resources>
                <resource>
                    <filtering>true</filtering>
                    <directory>src/main/resources</directory>
                </resource>
            </resources>
        </build>
</project>
```

Step#4:- Create git account and create configserverex Repository. Under this create file application.properties.

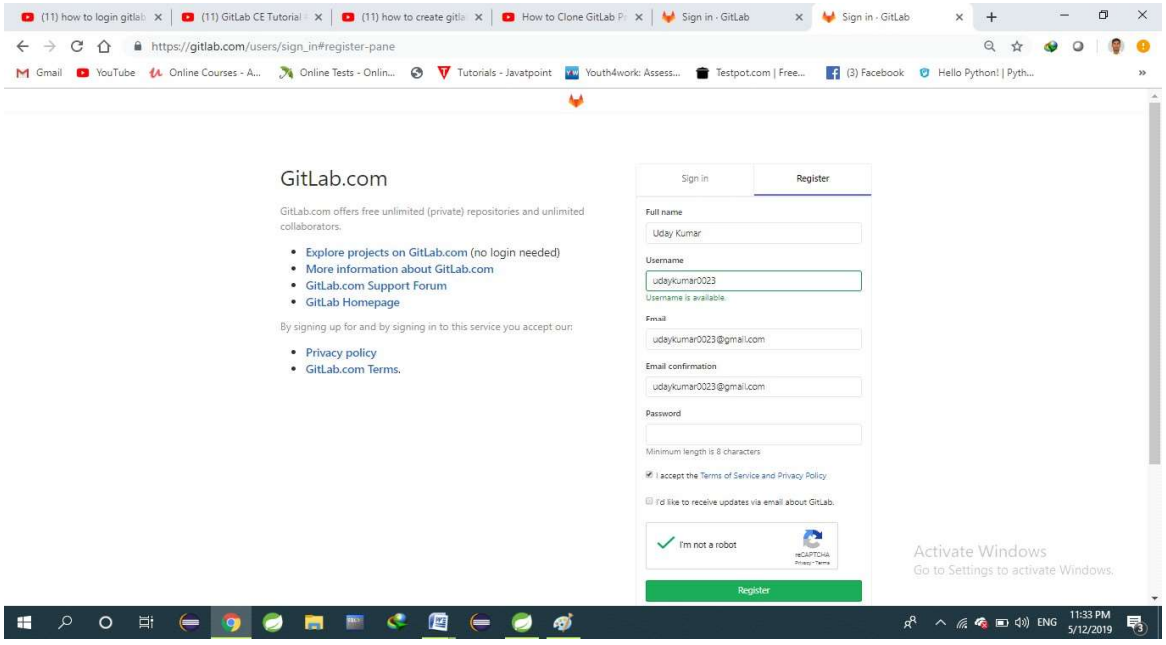
How to create GitLab Account:-- Follow bellow You tube Links
https://www.youtube.com/watch?v=_3l5OelRtk8

Step#1:- Goto https://about.gitlab.com links

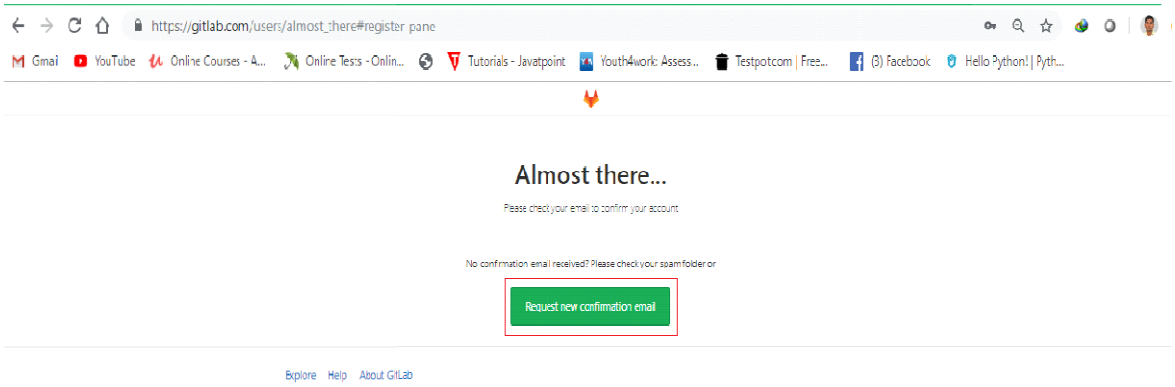
Step#2:- Click on Register Menu



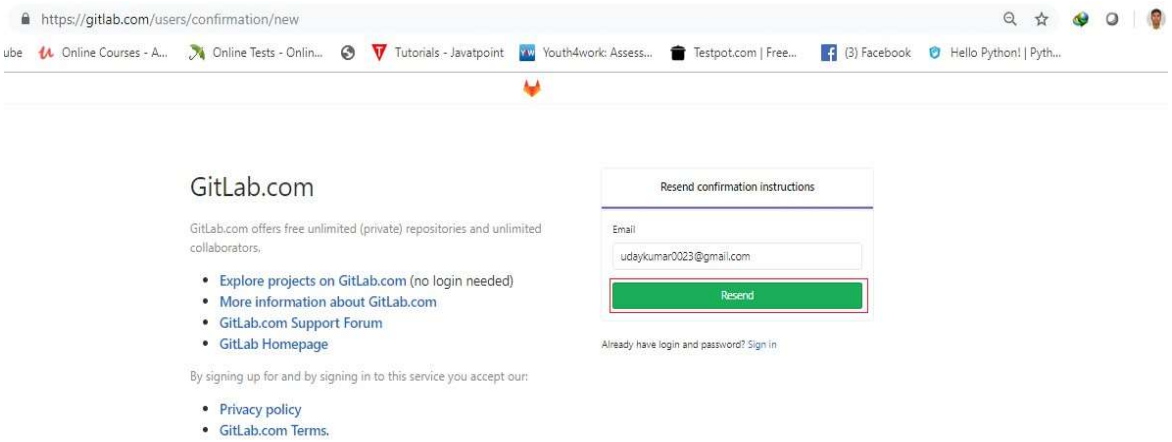
Step#3:- Fill the details and click on Register Button else login with github or Google Account If you have.



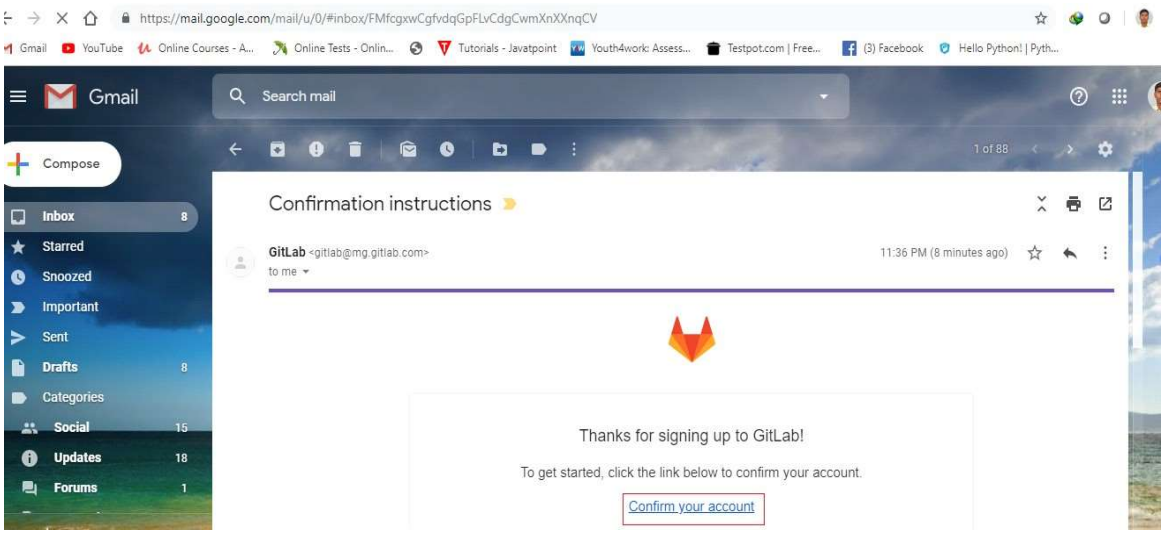
Step#4:- Click on Request new Confirmation Email



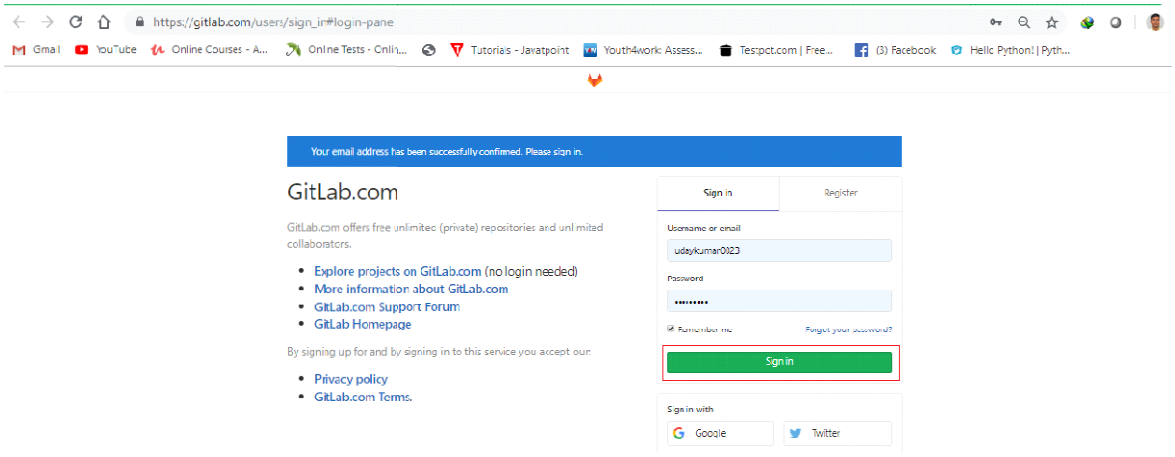
Step#5:- Enter Email address and click on Resend Button.



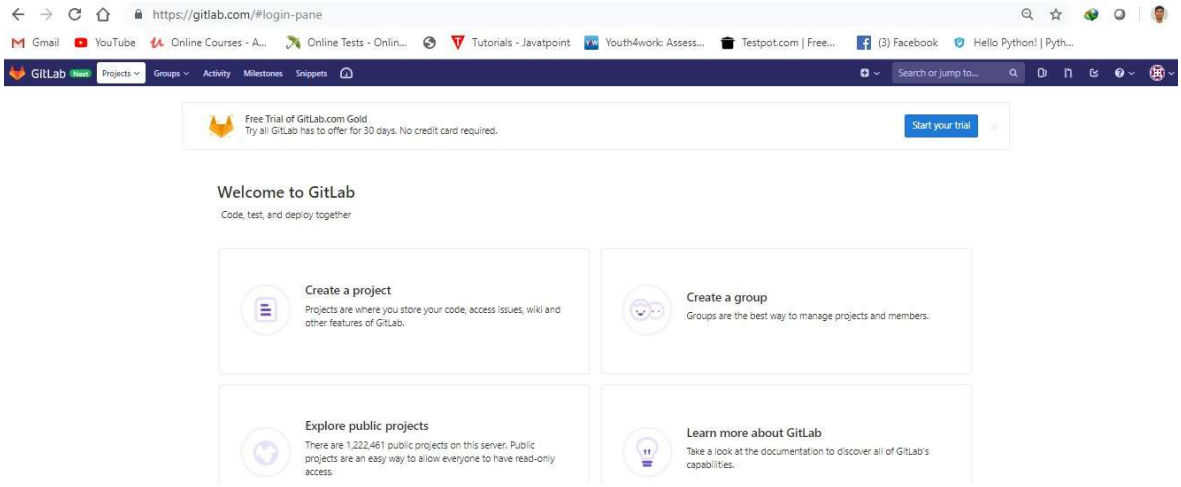
Step#6:- Login to Gmail account and click on Confirm your Account.



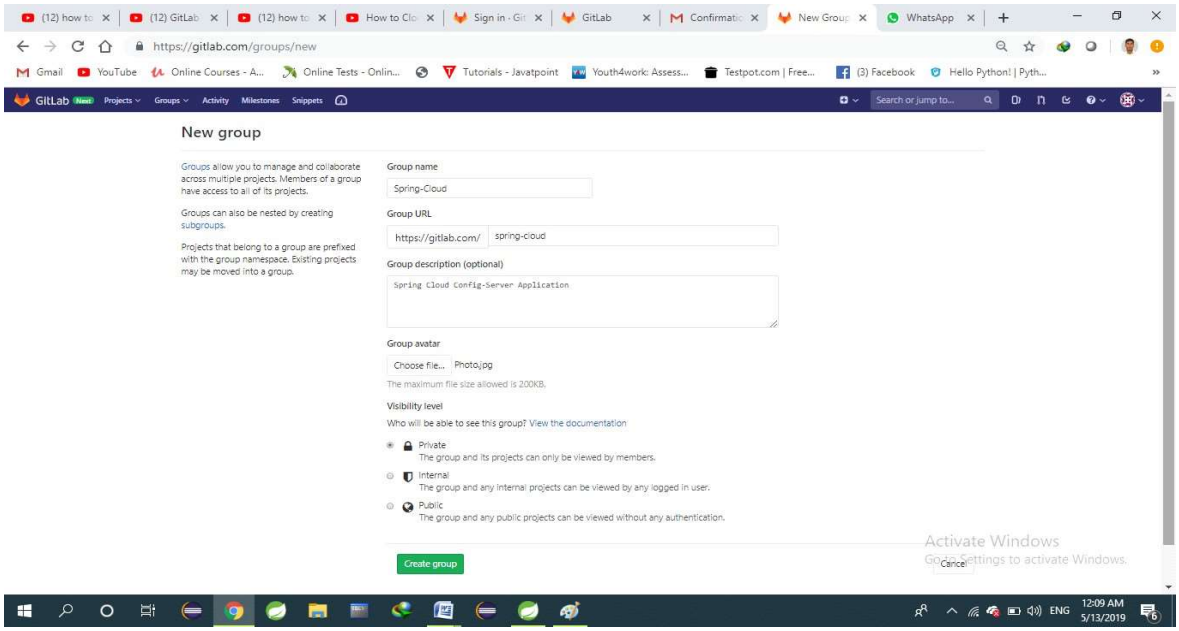
Step#7:- Provide your valid Credentials and click on Sign in Button.



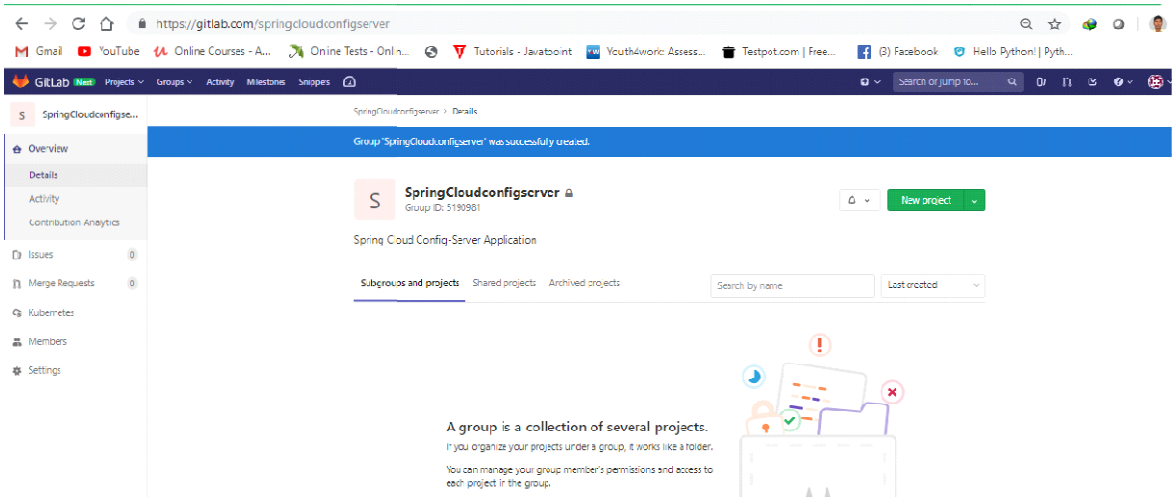
Step#8:-Final screen After Login into Gitlab account, Click on Create Group.



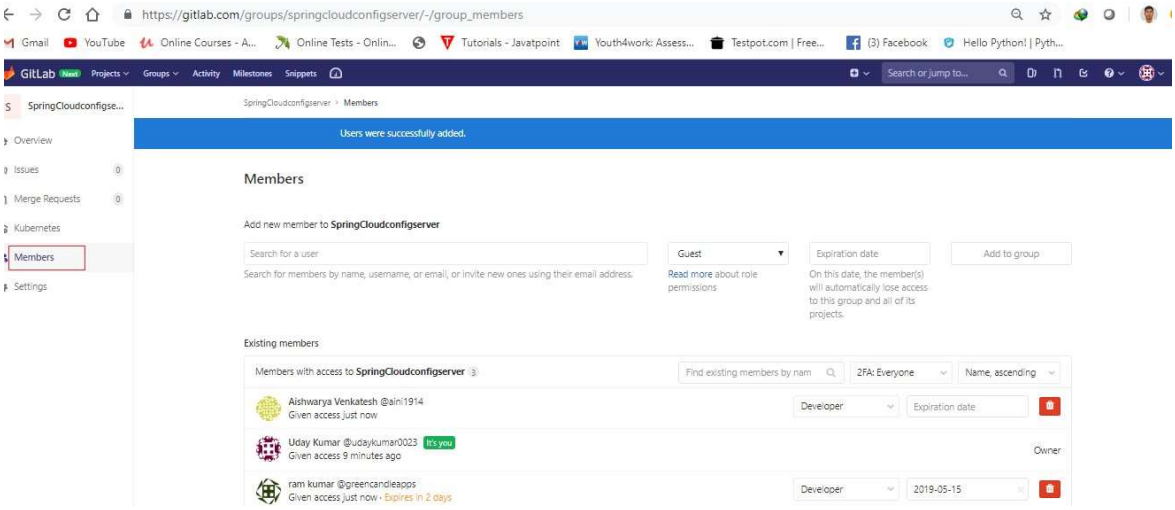
Step#9:- Fill the group name and click on Create Group



Step#10:-After successful Creating a new Group screen looks like.

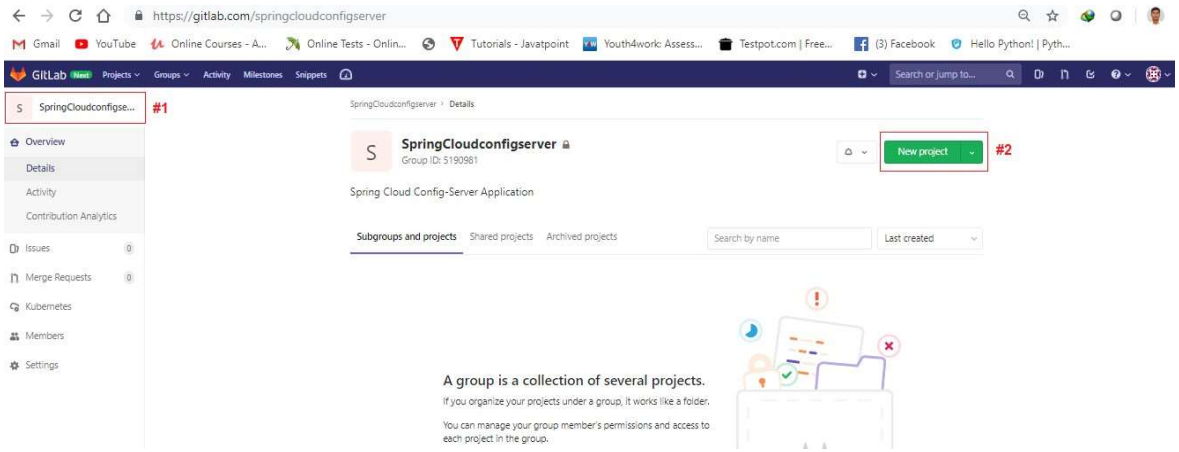


Step#11:- Click on Member Menu to add the new member into your project

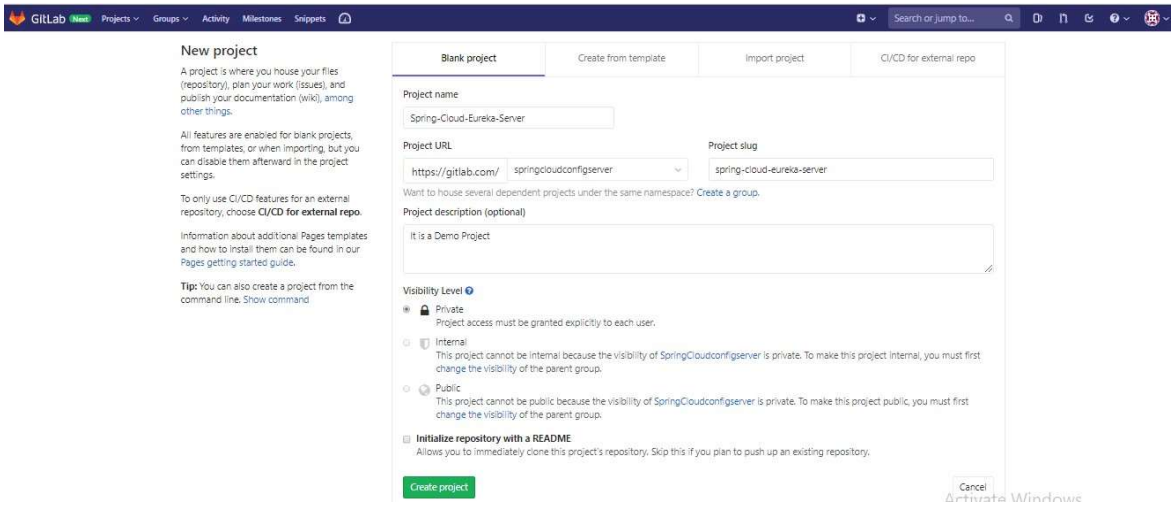


=>Select Member form list, Role Permissions like (Guest, Reporter, Developer, Maintainer, Owner) and Expire date and finally click on Add to Group.

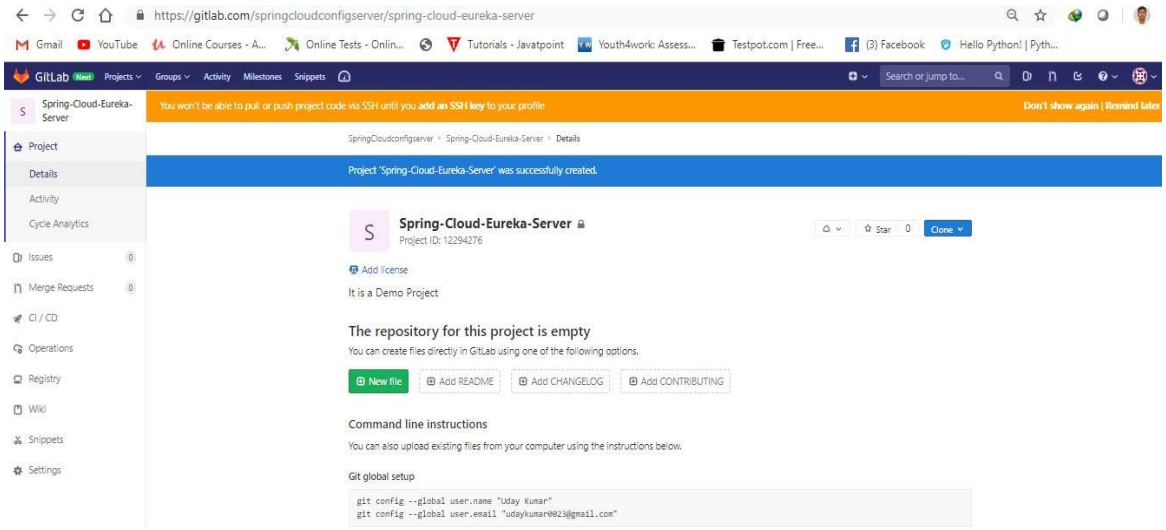
Step#12:- Click on Group name **(SpringCloudconfigserver)** and then after click on **New Project** to create a project.



Step#13:- Give a Project name and Click on Create Project.



Step#14:- It will show the all details.



Create a new repository

```
git clone https://gitlab.com/springcloudconfigserver/spring-cloud-eureka-server.git
cd spring-cloud-eureka-server
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

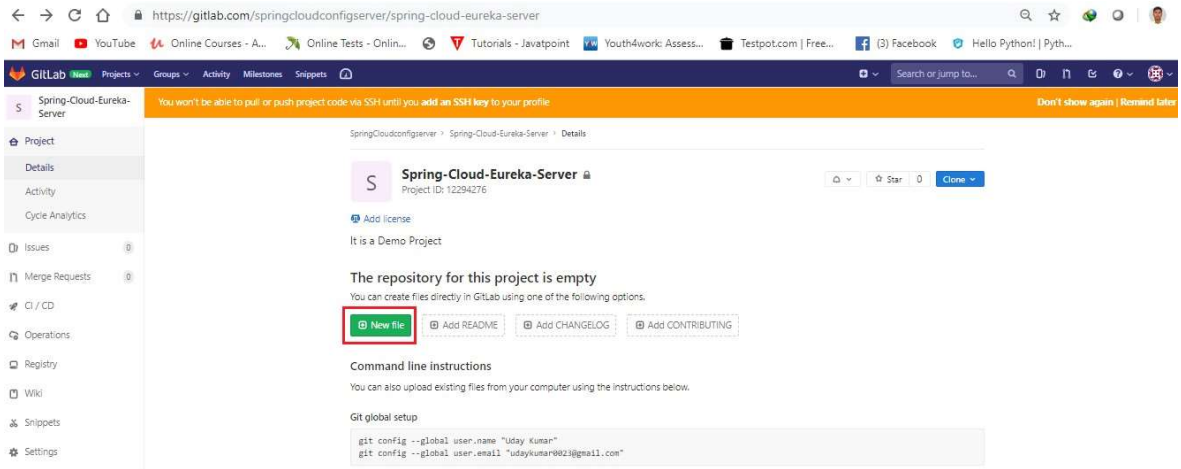
Push an existing folder

```
cd existing_folder
git init
git remote add origin https://gitlab.com/springcloudconfigserver/spring-cloud-eureka-server.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

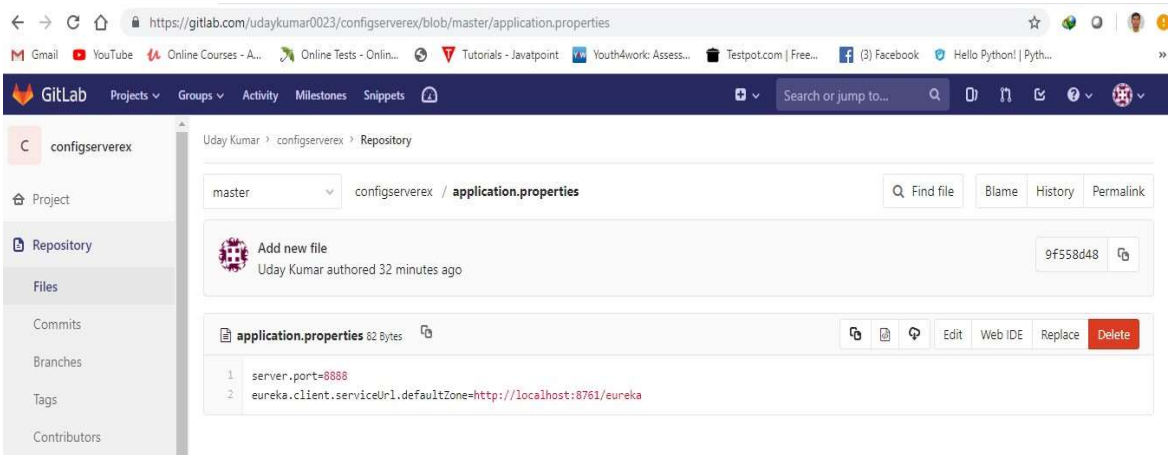
Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin https://gitlab.com/springcloudconfigserver/spring-cloud-eureka-server.git
git push -u origin --all
git push -u origin --tags
```

Step15:- Click on new file to create a file into repository



Step16:- Provide the details in file and then click on commit changes.



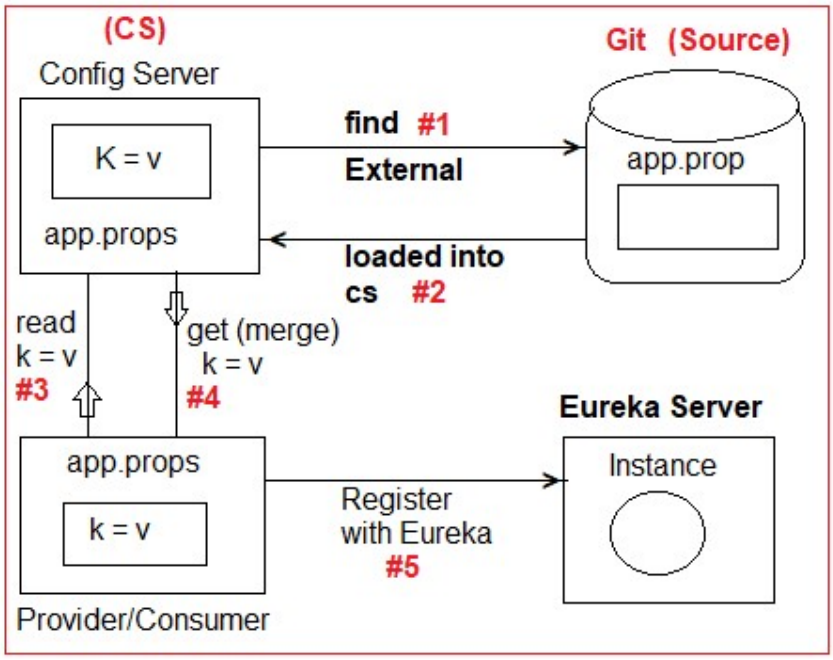
Config Server with Provider/Consumer Execution flow [External Source]:--

=>On Startup ConfigServer (CS) Application it will goto External Source (Git) and read __.properties (or)___.yml file having key=value pairs.

=>Then Provider/Consumer Application (On Startup) will try to read k=v from Config Server and merge with our application.properties.

=>If same key is found in both Config Server and Provider/Consumer App, then 1st priority is : Config Server.

=>After fetching all required props then Provider gets registered with Eureka Server.



=>Every Microservices must have below dependency in pom.xml.

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

=>It behaves as Config Client connected to Config Server.

=>Above Config Client dependency will search forConfig Server in a default location given as : (key = value)

spring.cloud.config.uri=http://localhost:8888

=>To modify above location (IP or PORT) provide this key in Config Client setup using bootstrap.properties.

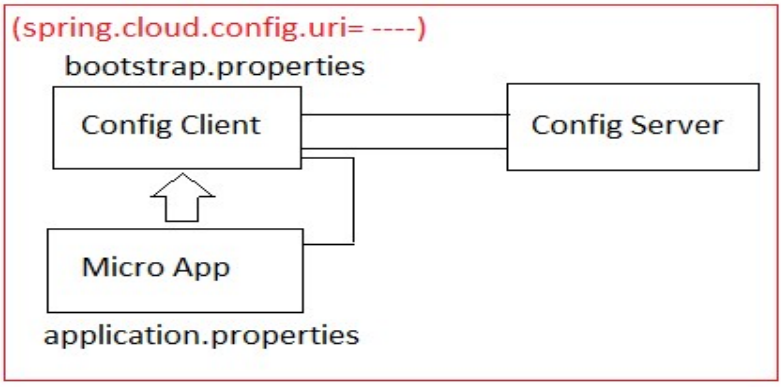
Q>What is bootstrap.properties in Spring Boot (cloud) Programming?

Ans:-- Our Project (child Project) contains input keys in application.properties, in same way Parent Project also maintains one Properties file named as : **bootstrap.properties** which will be loaded before our application.properties.

Q>What is the difference between application.properies file and bootstrap.properties file?

Ans:-- application.properties file is use to provide input to our application, where bootstrap.properties file is used to provide **Input** to parent project (or Configuration Setup) which gets run before our application.

Execution Order:-- 1>bootstrap.properties 2>application.properties



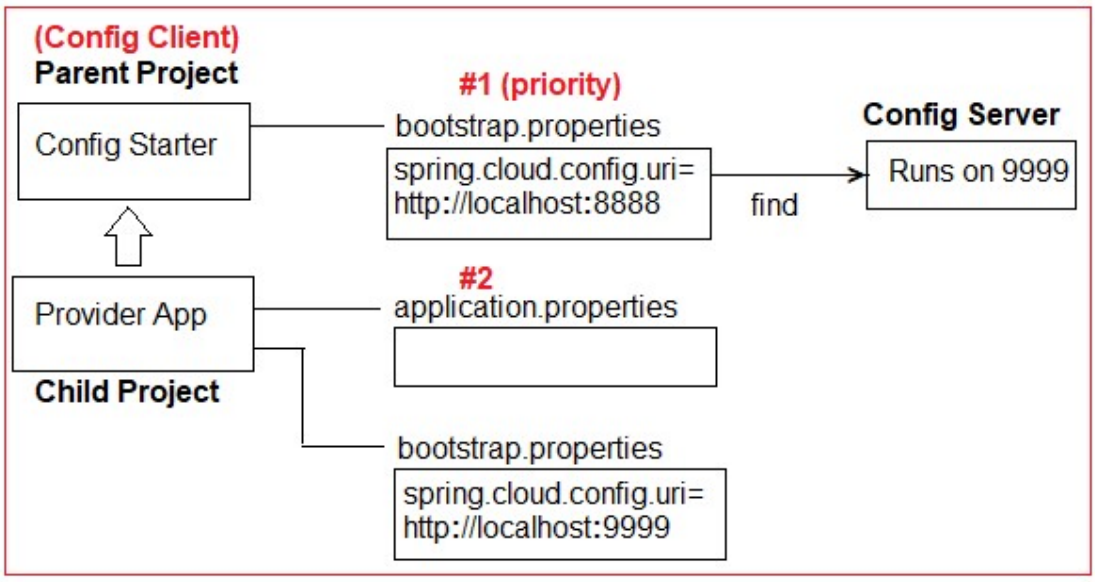
Execution Order:-- Parent Project-loads bootstrap.properties /.yml or Project-loads application.properties /.yml

->We can override this file in our project to provide key-values to be loaded by Parent Project.

**By default Config Server runs on http://localhost:8888 even config client also takes this as default location.

**To modify this IP/PORT use bootstrap.properties file in Our Project.

In this bootstrap.properties file override key : **spring.cloud.config.uri to any other location where Config Server is running.



Coding Changes for Config Server and Provider/Consumer Apps:--

Step#1:- Open application.properties file in Config Server Project and modify port number.
server.port =9999

Step#2:- In Provider/Consumer Project, create file bootstrap.properties under src/main/resources

Step#3:- Add key=value in bootstrap.properties file

spring.cloud.config.uri=http://localhost:8888
=>By default Config client will not get updates or modification from Config Server after starting client + microservice.
=>Only first time config Client fetch the data even to get new Modification after starting Client (without-Restart) use Annotation : @RefreshSocpe.

=>At Config Server application add below dependency in pom.xml.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

=>In application.properties (in Config Server) provide Native or External configuration details. This file is also called as Link file.

```
server.port=8888
spring.cloud.config.uri=http://localhost:8888
spring.cloud.config.server.git.uri=https://gitlab.com/udaykumar0023/configserverex
```

Q>Which class will load bootstrap.properties file for config Server URI fetching?

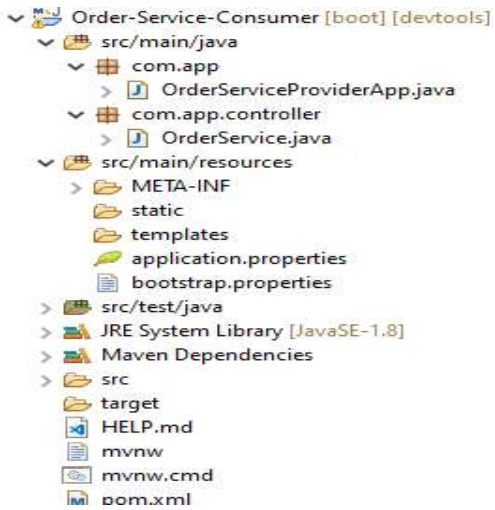
Ans:-- ConfigServicePropertySourceLocator will read config server input by default from http://localhost:8888.

=>It is only **first execution step** in provider Consumer Project.

Provider/Consumer Application:--

Create any Microservice application with Dependencies : Web, Eureka Discovery, Config- Client.

13. Folder Structure of Microservice with bootstrap:--



Step#1:- Starter class

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.context.config.annotation.RefreshScope;

@SpringBootApplication
@EnableDiscoveryClient
@RefreshScope  //Get updates from Config Server Automatically
public class OrderServiceProviderApp {
```

```
	public static void main(String[] args) {
		SpringApplication.run(OrderServiceProviderApp.class, args);
	}
}
```

Step#2:- Controller class.

```
package com.app.controller;
import  org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/order")
public class OrderService {

	@Value("${my.app : Hello Default One}")
	private String port;

	@GetMapping("/status")
	public String getOrderStatus() {
	   return "FINISHED : :"+port;
	}
}
```

Step#3:-1>application.properties:--

```
server.port=7800
spring.application.name=ORDER-PROVIDER
eureka.client.serviceUri.defaultZone=http://localhost:8761/eureka
eureka.instance.instance-id=${spring.application.name}:${random.value}
management.endpoints.web.exposure.include=*
```

2>bootstrap.properties:--

```
spring.cloud.config.uri=http://localhost:8888
```

Execution Order:--

1. Eureka Server
2. Config Server
3. MicroServices (Order)

=>Enter URL : http://192.168.100.27:7800/order/status

=>To enable **referesh** concept for microservices application(Order-APP), follow below steps.

***INSIDE ORDER-APP only.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

=>In application.properties file add

management.endpoints.web.exposure.include=*

=>At Controller class level, add **RefereshScope**

->Run : EurekaConfig Server, CART-APP.

->Enter URL for CartApp.

->Now modify value in Hibhub file.

->Open postman and makes POST request.

POST	http://localhost:7800/actuator/referesh	SEND

=>After showing success msg, Goto OrderApp URL & Refresh.

NOTE:--

***Actuator refresh is ready-made service given by Spring boot that fetch the data from Config Server location to ourApp.

=>It must be made as POST type request onl, using any http client (Ex: POSTMAN).

8. Fault Tolerance API :--

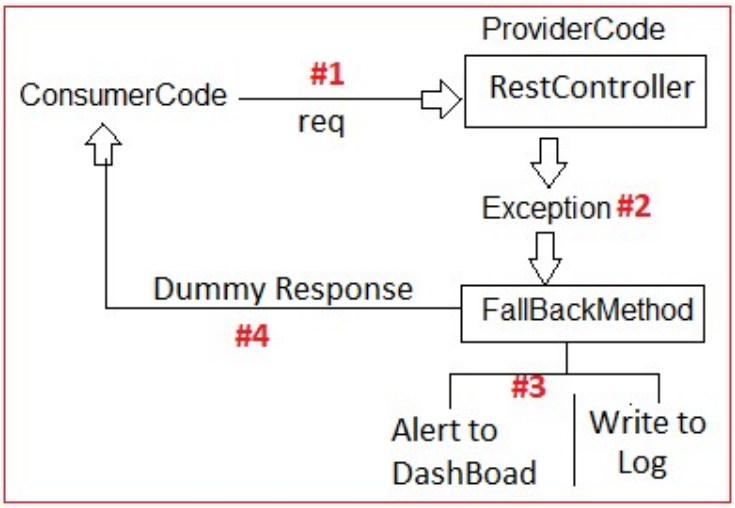
If any Microservice is continuously throwing Exceptions, then logic must not be executed every time also must be finished with smooth termination. Such Process is called as Fault Tolerance.

=>Fault Tolerance is achieved using fallBackMethod and CircuitBreaker Library in distributed system.

=>Fallback or Circuit breakers are used to handle exception which is occurred in Producer Application. It avoid Improper response and Shutdown termination of process.

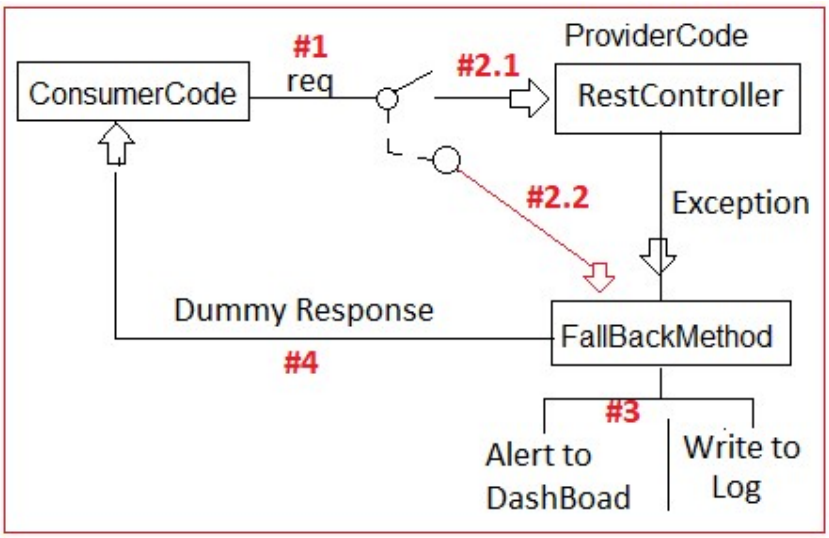
a. **fallBackMethod:-** If Microservice is throwing exception then service method execution is redirected to another supportive method (**fallBackMethod**) which gives dummy output and "alerts to DashBoard" (to Dev, MS, Admin teams). Writes to Log files, Email to admin etc.. is known as Fallback method, It makes termination of process in smooth manner.

FallBackMethod Process:--



b. **CircuitBreaker:-** If Producer App (Service) is throwing exceptions continuously then circuit breaker stops executing Producer code and Exception flow directly linked to fallback method.This is called as Opening a Circuit. It means Avoid invalid process or incomplete process execution and return **"Dummy Message"**. After some time gap (or) no. of request given to fallback, again re-checks once, still same continue to fallBackMethod else execute Microservice (Producer).

CircuitBreaker:--



Q>What is the difference between try-catch and Fallback with CircuitBreaker(CB)?
Ans:--In try-catch always try code is executed even if exception is reguler, where as Fallback with **CircuitBreaker** will stop execution of actual logic if exception is regular.

Hystrix:--

It is an API (set of classes and interfaces) given by Netflix to handle Proper Execution and Avoid repeated exception logic (Fault Tolerance API) in Microservice Programming.

=>It is mainly used in production Environment not in Dev Environment.

=>Hystrix supports FallBack and CircuitBreaker process.

=>It provides Dashboard for UI to view current request flow and Status. (View problems and other details in Services).

Working with Hystrix:--

Step#1:- Create one Microservice Application with dependencies web, eureka Discovery client, Hystrix.

groupId : com.app

artifactId : Spring_Cloud_Hystrix_Server

Netflix Hystrix dependency:-- Add below dependencies in pom.xml

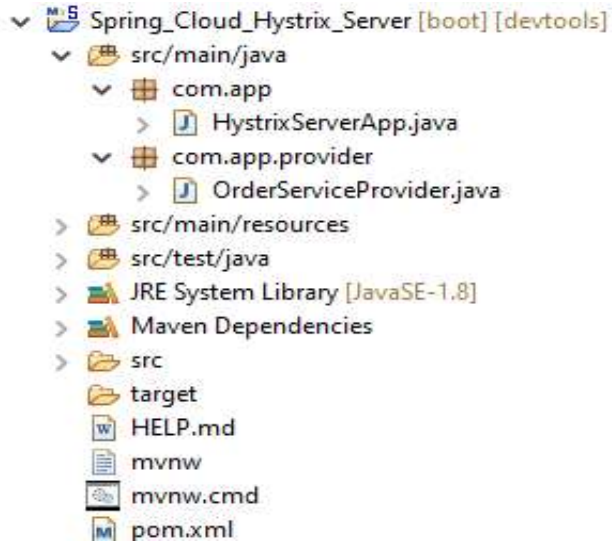
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

Step#2:-- At starter class level apply annotation @EnableCircuitBreaker (or) @EnableHystrix.

=>**@EnableCircuitBreaker** will find concept at runtime using pom.xml dependency. ex: Hystrix, Turbine etc… where as **@EnableHystrix** will execute only Hystrix CircuitBreaker.

Step#3:-- Define one RestController with actual Service and fallback method and apply Annotation : **@HystrixCommand** with Details like fallBackMethod, commandKey…
NOTE:-- Fallback method return type must be same as Actual Service method

#14. Folder Structure of Hystrix (CircuitBreaker) Implementation:--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.1.1.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
```

```
</parent>
<groupId>com.app</groupId>
<artifactId>Spring_Cloud_Hystrix_Server</artifactId>
<version>1.0</version>
<name>Spring_Cloud_Hystrix-Server</name>
<description>Demo project for Hystrix Service</description>

<properties>
	<java.version>1.8</java.version>
	<spring-cloud.version>Greenwich.SR1</spring-cloud.version>
</properties>

<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.cloud</groupId>
		<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.cloud</groupId>
		<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-devtools</artifactId>
		<scope>runtime</scope>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
	</dependency>
</dependencies>
```

```
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Step#1:- application.properties file:--

```
server.port=9800
spring.application.name=ORDER-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

Step#2:- Apply below any one annotation at Starter class.

```
@EnableCIrcuitBreaker
@EnableHystrix
```

HystrixServerApplication.java:--

```
package com.app;
import   org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.hystrix.EnableHystrix;
```

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableHystrix
public class HystrixServerApp {
	public static void main(String[] args) {
		SpringApplication.run(HystrixServerApp.class, args);
	}
}
```

Step#3:- Define RestController with FallbackMethod (**OrderServiceProvider.java**).

```
package com.app.provider;
import java.util.Random;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController
public class OrderServiceProvider {

	@GetMapping("/show")
	@HystrixCommand(fallbackMethod="showFallBack")
	//parameter must be same as fallBackMethod name
	public String showMsg() {
		System.out.println("From service");
	if (new Random().nextInt(10)<=10) {
		throw new RuntimeException("DUMMY");
	}
		return "Hello From Provider";
	}
	//fallBack method
	public String showFallBack() {
		System.out.println("From ballback");
		return "From FallBack method";
	}
}
```

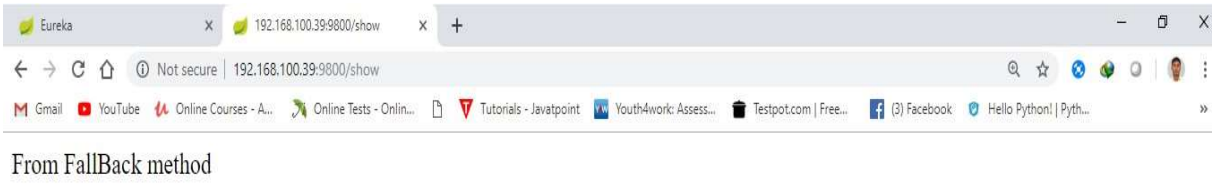
NOTE:-- Fallback method returnType must be same as service method return type.

Execution Process:--

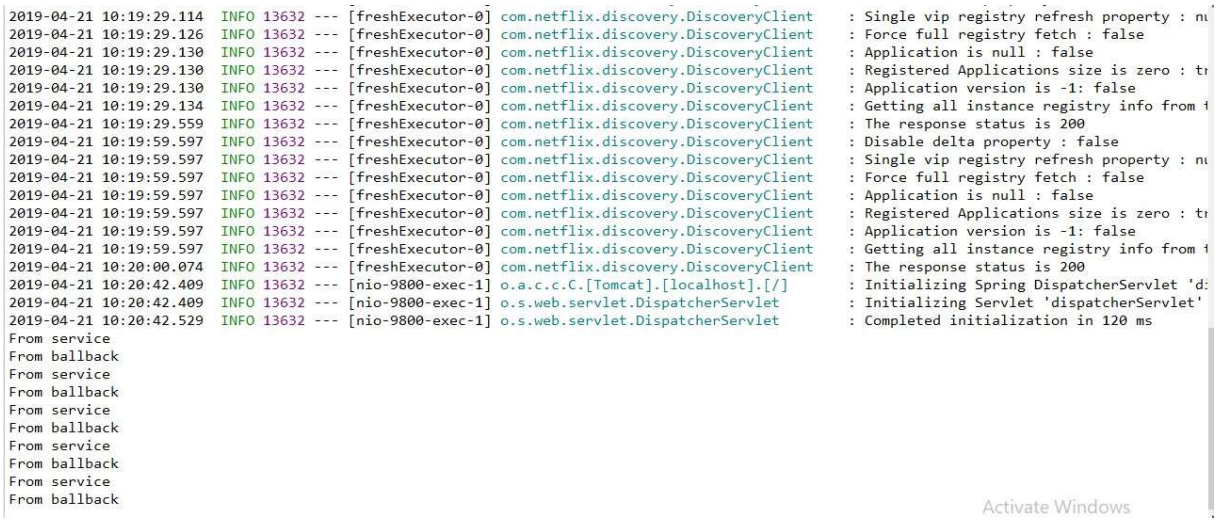
Step#4:- Run Eureka Server and Order Provider.

Step#5:- Goto Eureka and click on **ORDER PROVIDER** Instance and enter URL path **:/show** (http://192.168.100.39:9800/show)

=>Referesh multiple times to, see console.



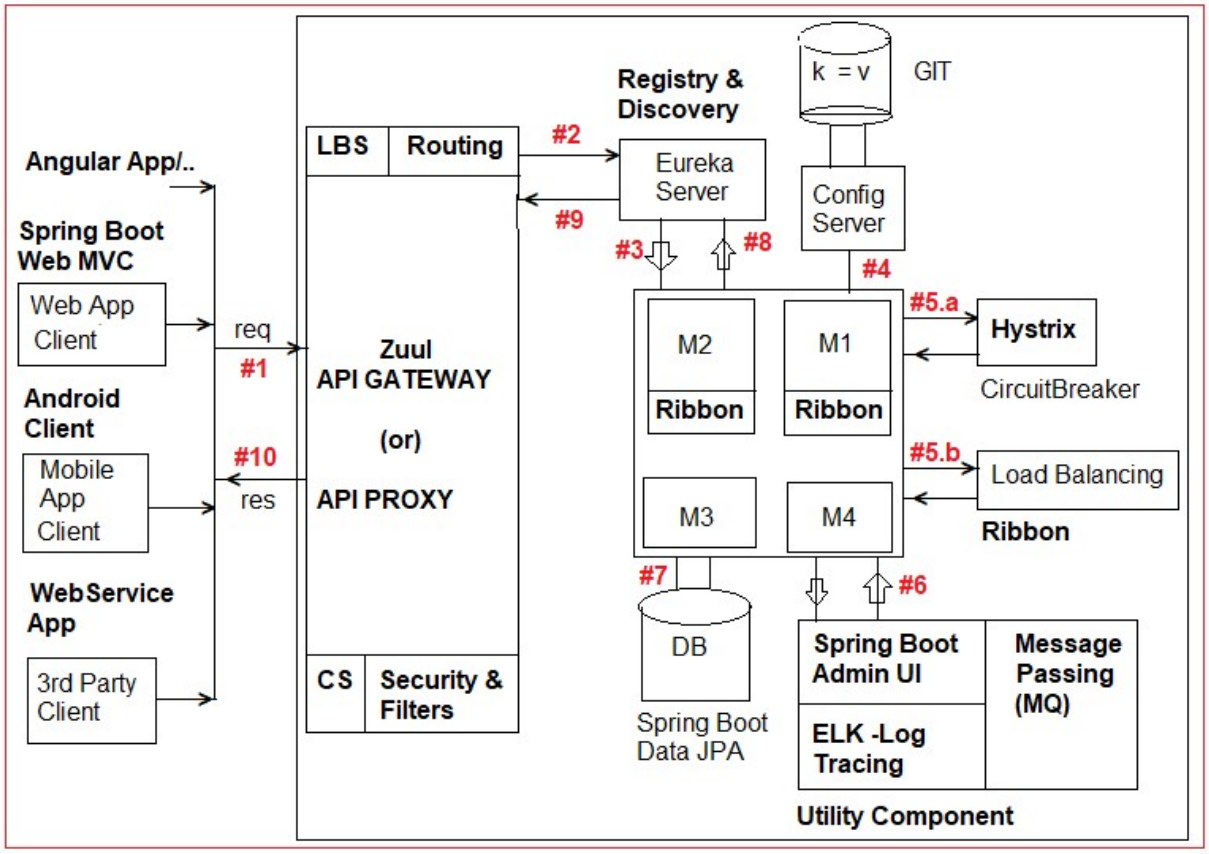
OUTPUT SCREEN OF "HYSTRIX-SERVICE-APP":--



NOTE:-- @HystixCommand(..) used to provide fallback method details. This is used to indicate Hystrix is enable for this method. (Not for all methods in RestController).

=>FallBack Method name can be any one. But Return Type must be same as

Spring Cloud Netflix Microservice Design:--



MQ = Message Queues

CS = Config Server

LBS = Load Balancing Server

ELK = Elastic Search –Logstash – Kibana

---

9. API PROXY /API GATEWAY:--

In one Application there will be multiple Microservices defined.

=>Every Microservice might be running in different servers (IP:PORT) and ports, even connected with load Balancing (Multiple IP:PORTs).

=>Microservices URLs are not directly given to client Applications (Mobile, Web, 3rd party) as there will be multiple addresses exist.

=>So, finally all microservices are grouped (masked) using Gateway which controls every request.

=>By using **unique PATH** (API), Gateway executes Microservice and given response back to client.

***Nature of Gateway:--**

->Single-Entry, one-Exist & one-URL.

->One time Authentication (SSO = Single Sing on) One Security system for all Microservices.

->**Dynamic Routing:-** Supporting with Eureka for finding Microservices using Ribbon. (Find Execution path b/w multiple Microservces).

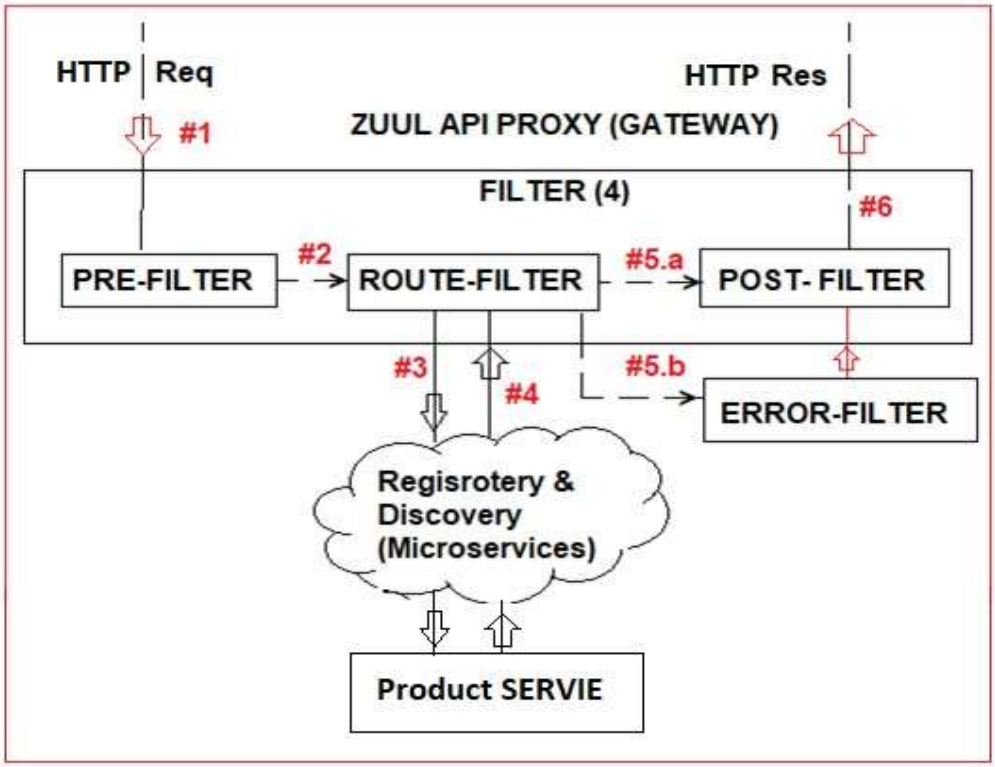
->Avoid Direct URL to client (Avoid CROS origin Request/response Format).

->Global data support (XML/JSON data Exchange).

->Supports Filtering.

=>Spring Cloud Netflix ZUUL behaves as API PROXY for Microservices Application which supports "Integration with any type of client component" (Web, mobile, 3rd party, webservices… etc).

Zuul (API PROXY) working flow:--



Zuul API (PROXY-SERVER) GATEWAY:--

Zuul Server is a Netflix component used to configure "Routing for Microservces".

It is an API gateway provided by Netflix and integrated with Spring Cloud.

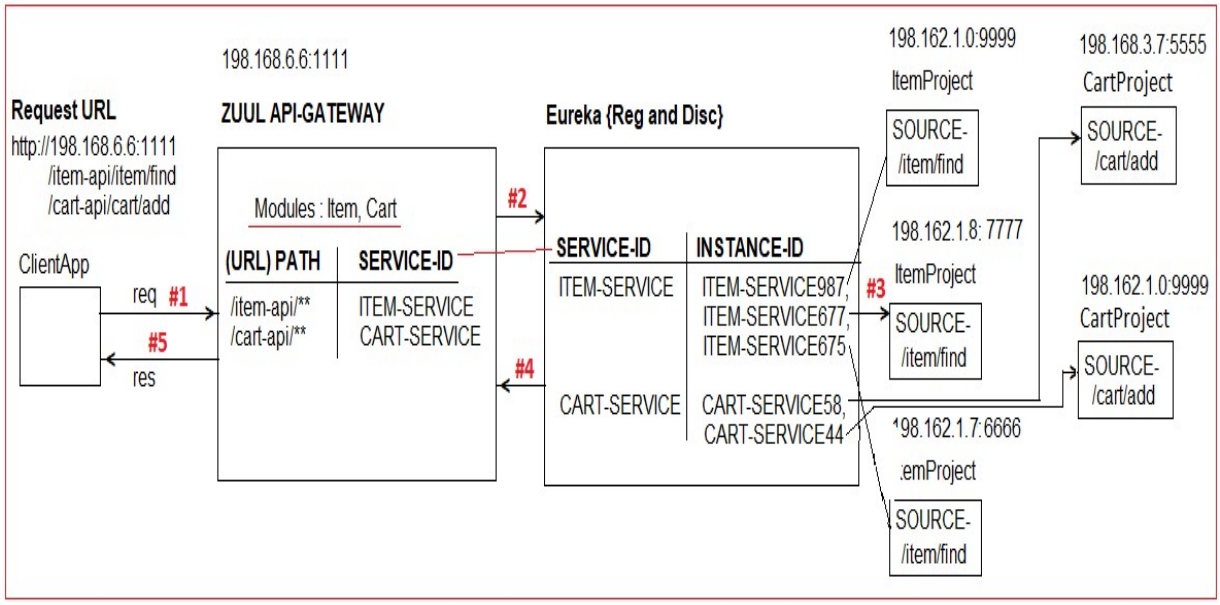
=>Netflix ZUUL supports dynamic routing, monitoring, Security and behaves as Consumer to all Microservices.

=>ZUUL is HTTP Client and server i.e. for UI/3rd party Clients it is a server for Microservices it is a client.
=>Zuul behaves as Consumer Application with R&D (Eureka) server to get all Microservices serviceIds with Instances.

=>Using a keys like :
zuul.routes.<module>.path=
zuul.routes.<module>.serviceId=

=>Before creating Zuul Server, we should have already created:
a. Eureka Server Project
b. Microservices (Ex: PRODUCT-SERVICE, CUSTOMER-SERVICE, STUDENT-SERVICE…) (with load balance implementation)

ZUUL EUREKA-MICROSERVICE WORK FLOW:--



=>Client make HTTP request to ZUUL server (ZUUL IP: PORT)
=>Client URL should have api (route) of Microservice provided by ZUUL. Based on this route (api) ZUUL reads “SERVICE-ID”.
=>Now, zuul behaves as consumer app to R&D fetch register and compare the service-ID, using request made by client.
=>ZUUL uses R&Ds LBSR (Load Balancing Service Register) to choose one instance-Id (URI)of Microservice based on load factor.
=>Now zuul makes Http call to RestController using class level and method level path given by client.
=>Response of RestController (global data) is given back to client (done by zuul).
****

->Here, Zuul Server behaves as Entry and Exit Point.
->It hides all details like Eureka Server and Service Instances from Client,
->Zuul Provides only **ZUUL-URL** and Paths of Services only.
->Zuul takes care of Client (request) Load balancing even.
->Provides Routing based on API (PATH/URL).
->Zuul must be registered with Eureka Server.

=>In Zuul Server Project, we should provide module details like path, service-Id using application.properties (or .yml)
=>If two modules are provided in ZUUL then, only module name gets changed in keys. Consider below example:

MODULE	PATH	SERVICE-ID
Product	/prod-api/**	PROD-SERVICE
Student	/std-api/**	STD-SERVICE

application.properties:--

```
zuul.routes.product.path=/item-api/**
zuul.routes.product.service-id=ITEM-SERVICE
zuul.routes.student.path=/std-api/**
zuul.routes.student.service-id=STD-SERVICE
```

application.yml:--

```
zuul:
routes:
    product:
        path: /item-api/**
        service-id: ITEM-SERICE
    student:
        path: /std-api/**
        service-id: STD-SERVICE
```

ZUUL Proxy:-- Netflix Zuul Server behaves as,
a. >Server:-- When end client componenet made HTTP request to Zuul.

b.>Consumer:-- To identify one service Instance based on Service-Id and communicate using feign + Ribbon (Code generated at runtime based on ServiceId choosen).

=>In simple Zuul is "Server+Consumer".
***To behaves as consumer, Zuul must get registered with Eureka.

Working with ZUUL Filters:--

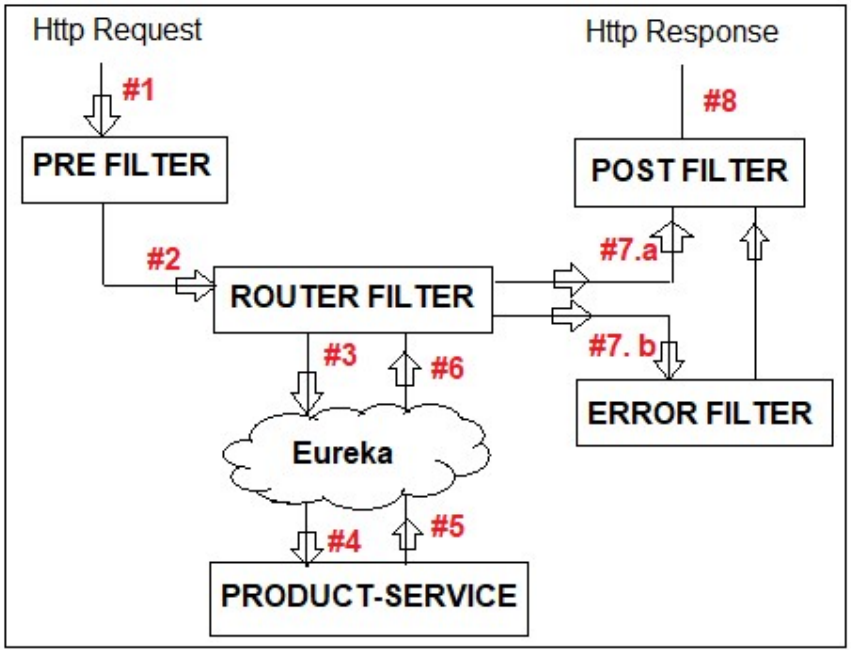
=>Filters are used to validate request and construct valid response. In simple we can also call as **"PRE-POST"** Processing Logic.
=>ZUUL Filter provides even extra types like **ROUTE FILTERS** and **ERROR FILTERS**.
=>ZUUL supports 4 types of filters implementation for Gateway service. Those are

1. **>PRE-FILTER:-** Work on request head body.
2. **>POST-FILTER:-** Work on response head/body.
3. **>ROUTE-FILTER:-** Work on Service Instance object (What is URI, host/port, SSL valid or not)
4. **>ERROR FILTER**:- Executed only if exception is raised (Write exception to log, email…).

=>In general our code (logic) is called as process. To execute extra code before our logic and filter our logic, before choosing logic instance after throwing exception logic filters are used

->When client made request to ZUUL then Pre Filter gets called automatically.
->After validating, request is dispatched to Route Filter.
->Route Filter is like 2nd level validation at Required SERVICE level.
->Route Filter will request to one Microservice based on Service-Id.
->If microservice is not executed properly (i.e throwing exception) then Error Filter is called.
->Finally Post Filter works on Http Response (adds Headers, Encode data, Provide info to client… etc) in case of either success or failure.

Diagram:--



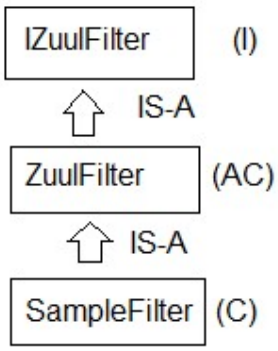
=>Here, Filter is a class must extends one abstract class **"ZuulFilter"** provided by Netflix API.

=>We can define multiple filters in one Application. Writing Filters are **optional**.

=>While creating filter class we must provide Filter **Order** (0, 1, 2, 3…) and Filter Type ("pre", "route", "error", "post")

=>Two filters of same type can have same Order which indicates any Execution order is valid.

=>We can enable and disable Filters using its flags (true/false).



=>To indicate Filter Types, we use its equal constants (public static final String variables), provided as.

TYPE	CONSTANT
Pre	PRE_TYPE
Route	ROUTE_TYPE
Error	ERROR_TYPE
Post	POST_TYPE

=>All above Constants are defined in FilterConstants (C) as global variables (public static final String).
=>Write one class in ZUUL Server and extends ZuulFilter Abstract class, override below method (4) in your filter class.

a>shouldFilter():-- Must be set to 'true' If value is set to false then filter will not be executed.
b>run():-- Contains Filter logic Executed once when filter is called.
c>filterType():-- Provide Filter Constant Must be one Type (pre, post, route, error).
d>filterOrder():-- Provide order for Filter Any int type number like 0, 56, 78.

Ex:- ***Create below class under ZuulServer Application.

```
package com.app.filter;

@Component
public class FilterType extends ZuulFilter {
	/**To enable or Disable current filter Enable(true) or Disable Filter(false)**/
	public boolean shouldFilter() {
		return true;
	}
	/**Define Filter Logic Here**/
	public Object run() throws ZuulException {
		System.out.println("FROM POST FILTER");
		return null;
	}
	/**Specify Filter Type (Pre. Post...)**/
	public String filterType() {
		return FilterConstants.POST_TYPE;
	}
	/**Provider Filter Order for Execution if same type of filters are used**/
	public int filterOrder() {
		return 0;
	}
```

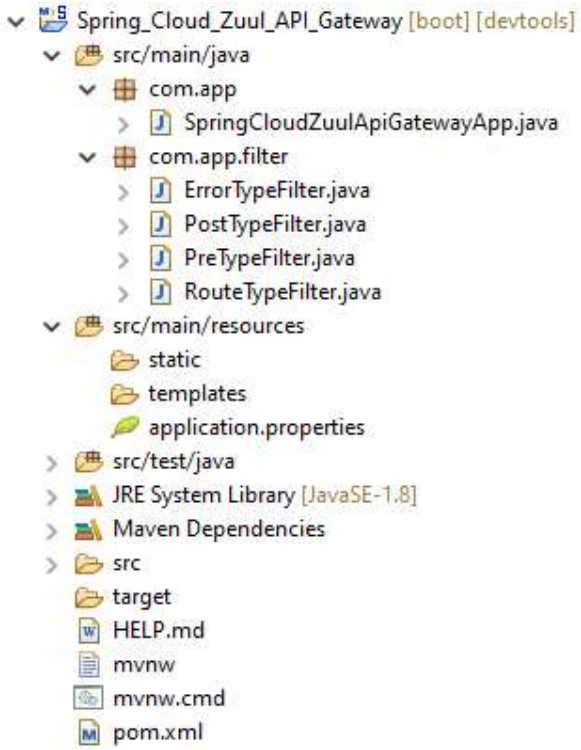
=>FilterConstants is a class having static final variables like PRE_TYPE, POST_TYPE, ROUTE_TYPE, and ERROR_TYPE **(public static final String PRE_TYPE=”pre”)**.
=>FilterConstants even provide global standard constant details HTTP PORT =80, HTTPS PORT:443.

Step#1:- Create Spring starter Project as : **ZUUL SERVER** with dependencies : Web, Zuul, Eureka Discovery.

Zuul Dependency:--

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
```

#15. Folder Structure of Zuul Server (Gateway Server):--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.1.1.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.app</groupId>
        <artifactId>Spring_Cloud_Zuul_API_Gateway</artifactId>
        <version>1.0</version>
        <name>Spring_Cloud_Zuul_API_Gateway</name>
        <description>Demo project for Zuul API Gateway</description>

        <properties>
                <java.version>1.8</java.version>
                <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
```

```
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>
        </dependencies>
        <dependencyManagement>
            <dependencies>
                <dependency>
                    <groupId>org.springframework.cloud</groupId>
                    <artifactId>spring-cloud-dependencies</artifactId>
                    <version>${spring-cloud.version}</version>
                    <type>pom</type>
                    <scope>import</scope>
                </dependency>
            </dependencies>
        </dependencyManagement>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Step#2:- application.properties should have below details like:

```
#zuul.routes.<module>.path=/<module>-api/**
#Service Id name must be Eureka Server Service Id name
#zuul.routes.<module>.service-id=[SERVICE-ID]

server.port=8558
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

```
spring.application.name=ZUUL-PROXY
#Each Microservice url and Service Id configuration
zuul.routes.item.path=/item-api/**
zuul.routes.item.service-id=ITEM-SERVICE
zuul.routes.cart.path=/cart-api/**
zuul.routes.cart.service-id=CART-PROVIDER
```

Step#3:- In Zuul Server Project, at starter class level add annotation :
@EnableZuulProxy.

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableDiscoveryClient
@EnableZuulProxy
public class SpringCloudZuulApiGatewayApp {
        public static void main(String[] args) {
                SpringApplication.run(SpringCloudZuulApiGatewayApp.class,  args);
                System.out.println("Zuul Api Gateway Server is executed");
        }
 }
```

Step#4:- Implement Filters like :
PRE, ROUTE, POST, ERROR
using one abstract class : ZuulFilter (AC) and use FilterConstants (C).

#1. PreTypeFIlter.java:--

```
package com.app.filter;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;
```

```
@Component
public class PreTypeFilter extends ZuulFilter{
	/**Enable(true) or Disable Filter(false)**/
	public boolean shouldFilter() {
		return true;
	}
	/**Define Filter Logic Here**/
	public Object run() throws ZuulException {
		System.out.println("FROM PRE FILTER");
		return null;
	}
	/**Specify Filter Type**/
	public String filterType() {
		return FilterConstants.PRE_TYPE;
	}
	/**Provider Filter Order for Execution**/
	public int filterOrder() {
		return 0;
	}
}
```

#2. RouteTypeFIlter.java:--

```
package com.app.filter;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;

@Component
public class RouteTypeFilter extends ZuulFilter {

	/**Enable(true) or Disable Filter(false)**/
	public boolean shouldFilter() {
		return true;
	}
	/**Define Filter Logic Here**/
	public Object run() throws ZuulException {
```

```
		System.out.println("FROM ROUTE FILTER");
		return null;
	}
	/**Specify Filter Type**/
	public String filterType() {
		return FilterConstants.ROUTE_TYPE;
	}
	/**Provider Filter Order for Execution**/
	public int filterOrder() {
		return 0;
	}
}
```

#3. ErrorTypeFIlter.java:--

```
package com.app.filter;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;

@Component
public class ErrorTypeFilter extends ZuulFilter{

	/**Enable(true) or Disable Filter(false)**/
	public boolean shouldFilter() {
		return true;
	}
	/**Define Filter Logic Here**/
	public Object run() throws ZuulException {
		System.out.println("FROM ERROR FILTER");
		return null;
	}
	/**Specify Filter Type**/
	public String filterType() {
		return FilterConstants.ERROR_TYPE;
	}
	/**Provider Filter Order for Execution**/
```

```
	public int filterOrder() {
		return 0;
	}
}
```

#4. PostTypeFIlter.java:--

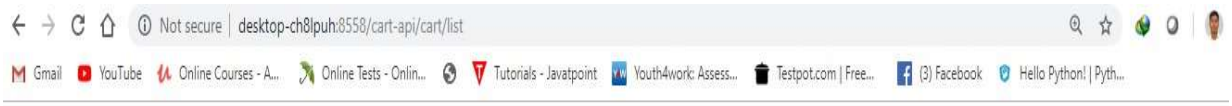
```
package com.app.filter;
import org.springframework.cloud.netflix.zuul.filters.support.FilterConstants;
import org.springframework.stereotype.Component;
import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;

@Component
public class PostTypeFilter extends ZuulFilter{

	/**Enable(true) or Disable Filter(false)**/
	public boolean shouldFilter() {
		return true;
	}
	/**Define Filter Logic Here**/
	public Object run() throws ZuulException {
		System.out.println("FROM POST FILTER");
		return null;
	}
	/**Specify Filter Type**/
	public String filterType() {
		return FilterConstants.POST_TYPE;
	}
	/**Provider Filter Order for Execution**/
	public int filterOrder() {
		return 0;
	}
}
```

Execution Process:--

->http://desktop-ch8lpuh:8558/cart-api/cart/info

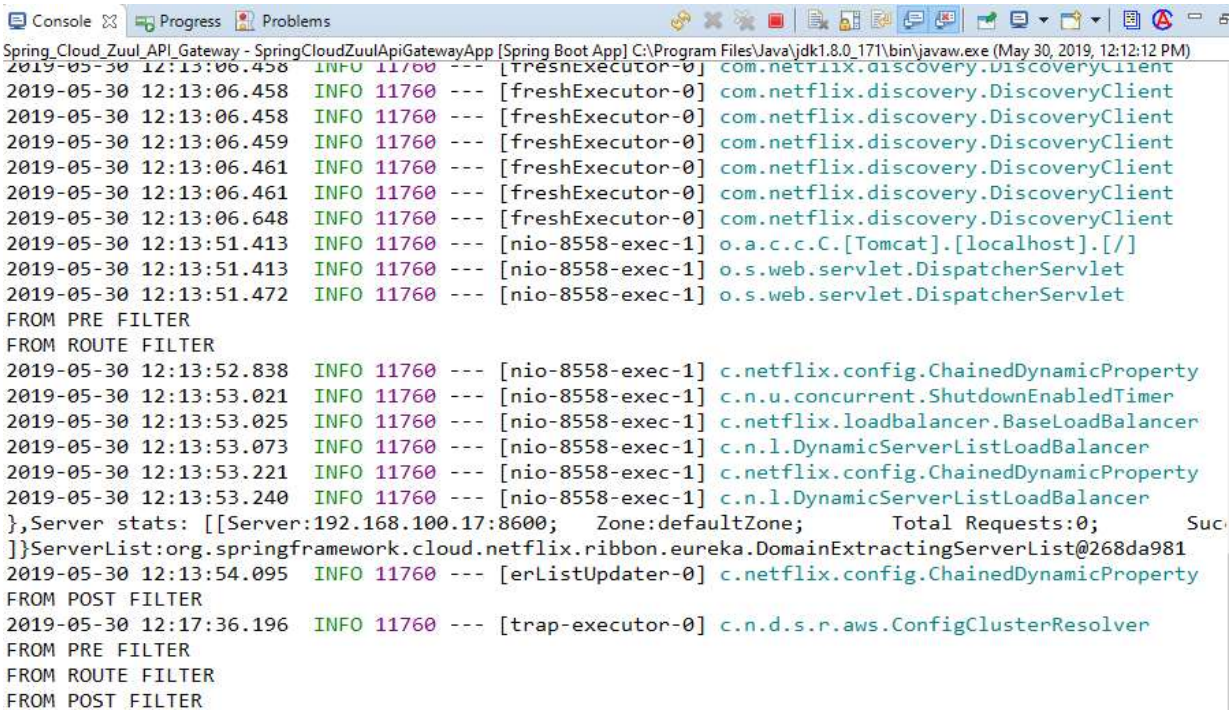
->http://desktop-ch8lpuh:8558/cart-api/cart/list

->http://desktop-ch8lpuh:8558/item-api/item/info



[{"cartId":101,"cartCode":"A :8600","cartFinalCost":876.98},{"cartId":102,"cartCode":"B :8600","cartFinalCost":856.98},
{"cartId":103,"cartCode":"C :8600","cartFinalCost":883.98}]

Console SCREEN of Wateway:--



10. Spring Boot Message Queue (MQ):--

In case of real-time application large data needs to be transferred and processes.
Like Google Server to Amazon, Facebook, etc…
=>Data (Message) Transfer can be done using Message Queues.

Message Queues are used for:--

1.>Data Streaming
2.>Web Activities
3.>Log Agrigartion
4.>Command Oriented Components.

a.>Data Streaming:-- Sending continous data between two application is called data Streaming. In Real-time large data from Files, Networks, and Databases etc…

Ex:-GoogleMap updates for swiggy IRCTC Train updates where is my Train, Flights enquiry and booking status.

b.>Web Activities:-- Sharing search information to partner applications (Know what users are searching and provide related links and ADS in client websites).

Ex:- Google search data given to Cricbuzz, Amazon.

c.>Log Aggrigation:-- Sending log files data from current server to other servers to find Exception and worning details.

d.>Command Oriented Coomponents:-- It is a trigger based service connected to multiple components works based on command.

Ex:- If User books and Order from Amazon then send a message to his mobile (Mobile service component), send a email to user (email service component), send invoice to printer (Remote printer service component).

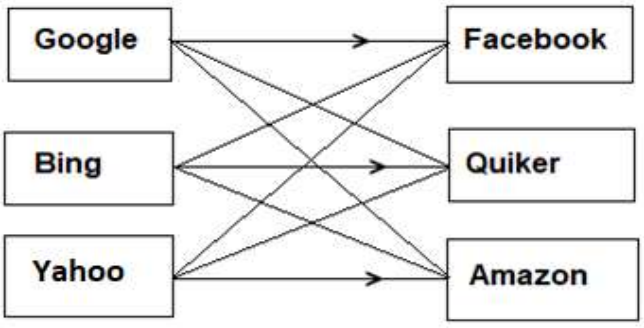
=>Above component are executed on command save.

Distributed Messaging (MQ):--

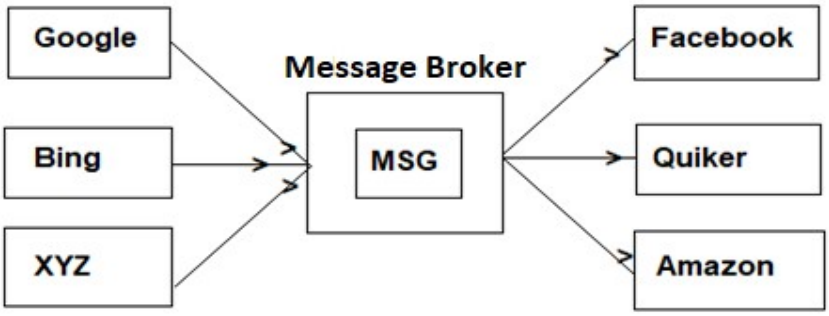
=>It is a process of Sending message from Source Application to one or more Destination application.

=>It is not a concept of request and response. It is implemented using MQ.

=>In case of multiple clients share data without MQ, files look like this:



A design with Message Queue (MQ):--This is simplified by MQ which is given below with one mediator also called as Message Broker.



=>MQ can be implemented using Language based Technologies (or API).
Ex:-- JMS (Java Message Service)
=>Global MQ (between any type of client) Advanced Message Queuing protocol) (AMQP)
=>Apache ActiveMQ, Rabbit MQ, Apache Atrims are different Service providers (Broker software's) for JMS.
=>Apache KAFKA is a service Provider (Broker software) for AMQP).

This are Two Types of MQ:

a.>Basic Message Queuing Protocal (BMQP).
b.>Advanced Message Queuing Protocal (AMQP).

a>Basic Message Queuing Protocal(BMQP):-- It is also called as language specific MQ. In case of Java it is called as JMS (Java message Service). It is language dependent.

Java Message Server:--

It is used to implement Basic Messaging process between application to send receive data. It uses one Broker Software i.e.e called as MOM (message Oriented Middleware).
=MOM contains special type memory (Container) called as Destination that holds Messages.
=>Every application connected to MOM is called as **Client**.
=>There are two types of client Consumer and Producer.

Spring Boot with Apache ActiveMQ:--
Java JMS is simplified using Spring Boot which reduces writing basic configuration for ConnectionFactory, Connection, Session, Destination Creation, Send/Receive Message etc…

JMS supports 2 types of client. Those are.

a.>Producer (Client) : Sends Message to MOM

b.>Consumer (Client) : Read Message from MOM

=>Messages are exchanged using **Message Broker** called as **MOM** (Message Oriented Middleware). Apache ActiveMQ is MOM Software.

Types of Communications in JMS:--

a.>**P2P (Peer-To-Peer) :** -- Sending 1 message to one consumer. In this case destination type is called as "QUEUE".

b.>Pub/Sub (Publish and Subscribe):-- Sending 1 message (same copy) to Multiple consumers. Here, Distination type is "TOPIC".

*** JMS supports two types of Communications.

NOTE:--

a.>Destination is a Special memory created in MOM which holds messages.

b.>Here **Queue** Destination is used for P2P. **Topic** Destination is used for Pub/Sub.

Limitation of JMS:--

a.>Used between Java Applications.

b.>Message (Data) size should be smaller.

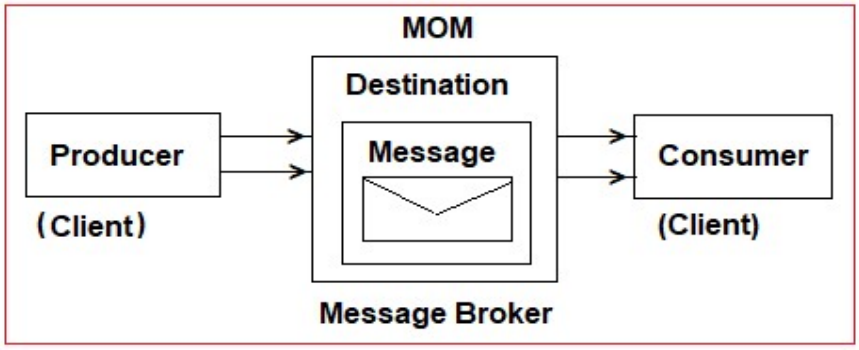
c.>Only one MOM (one instance) runs at a time.

d.>Large data takes lot of time to process.

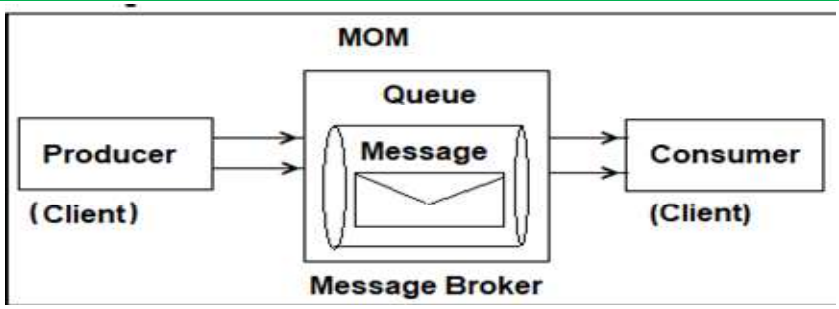
e.>If Pub/Sub model is implemented with more consumers then process will be very slow.

f.>Data may not be delivered (data lose) in case of MOM Stops or Restart.

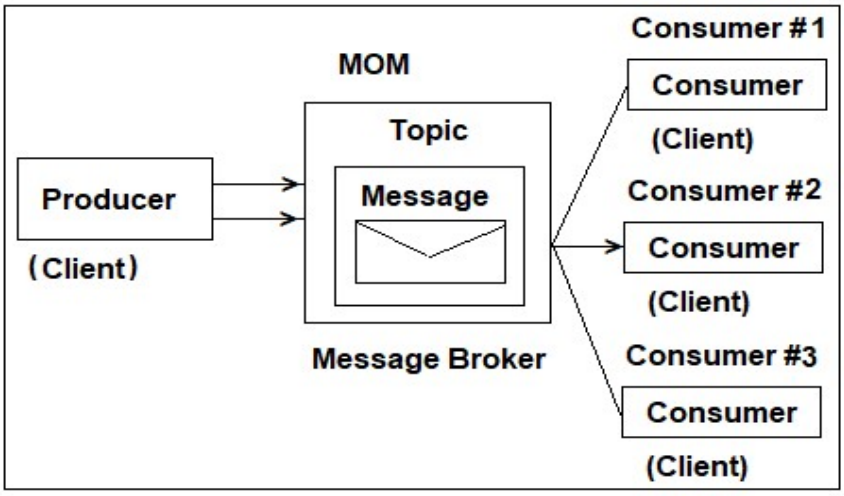
Generic Design of JMS:--



1. Peer to Peer : P2P (Queue):--



2. Pub/Sub : Publish and Subscribe (Topic):--



Steps to Implement ActiveMQ:--

Step#1:- Create one Spring boot starter application (Client Application) using dependencies: ActiveMQ (JMS) (or add below dependency in pom.xml)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
```

pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>
        <parent>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.1.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.app</groupId>
    <artifactId>Spring-Cloud-JMS-ActiveMQ</artifactId>
    <version>1.0</version>
    <name>Spring-Cloud-JMS-ActiveMQ</name>
    <description>Demo project for Spring JMS Implementation</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-activemq</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
```

```
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Step#2:- In application.properties file provide common keys like MQ brokerUrl, User, Password in all Producer and Consumer Application.

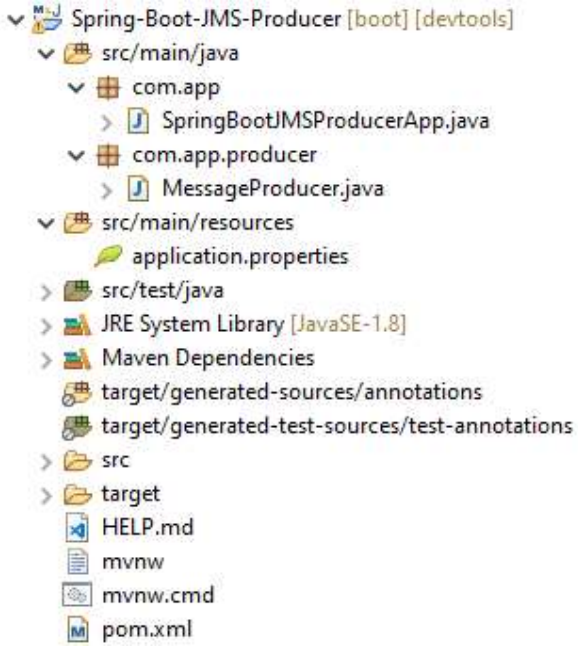
->If we do not specify any type then it behaves as **P2P (Queue).** To make it as Pub/Sub (Topic).

application.properties:-- Add same properties in Producer & Consumer application.

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
spring.jms.pub-sub-domain=false
```

Step#3:- If application is Producer type then use JmsTemplate object and call send() which will send message to MoM.

#16. Folder Structure of JMS Producer:--



#1. Starter class for Producer (SpringBootActiveMqProducerApp.java):--

```
package com.app;
import   org.springframework.boot.SpringApplication;
import    org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJMSProducerApp {
        public static void main(String[] args)
        {
                SpringApplication.run(SpringBootJMSProducerApp.class, args);
                System.out.println("JMS Producer Executed :");
        }
}
```

#2 MessageProducer.java:--

```
package com.app.producer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Component;

@Component
public class MessageProducer implements CommandLineRunner {

    @Autowired
    private JmsTemplate template;

    @Override
    public void run(String... args) throws Exception {
        template.send("my-tpca", (ses)->ses.createTextMessage("AAAAAAAAA"));
        System.out.println("sent from Producer");
    }
}
```

Step#4:- If Application is Consumer type then define one Listener class using destination.

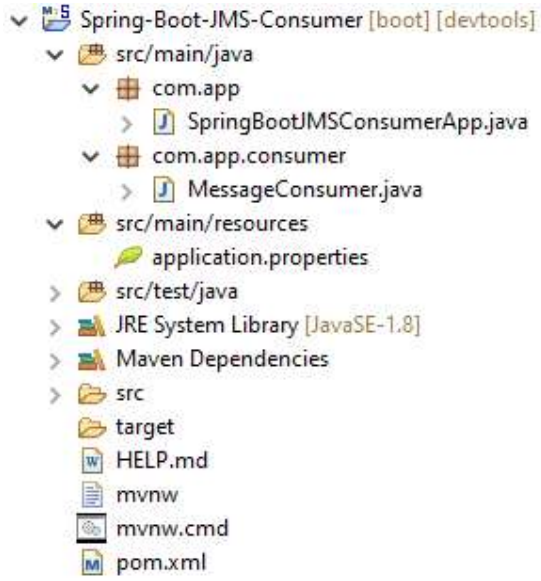
Use code : @JmsListener(destination=" ---- ")

=>It must be Enabled using code: **@EnableJms**

=>In case of JmsTemplate (C) **@EnableJms** is not required.

NOTE:-- Create multiple consumer app in same or different workspaces and set.
spring.jms.pub-sub-domain=true
In application.properties file (or .yml).

#16.a Folder Structure of JMS Consumer1 Application:--



#1. Starter class (SpringBootJMSConsumer.java):--

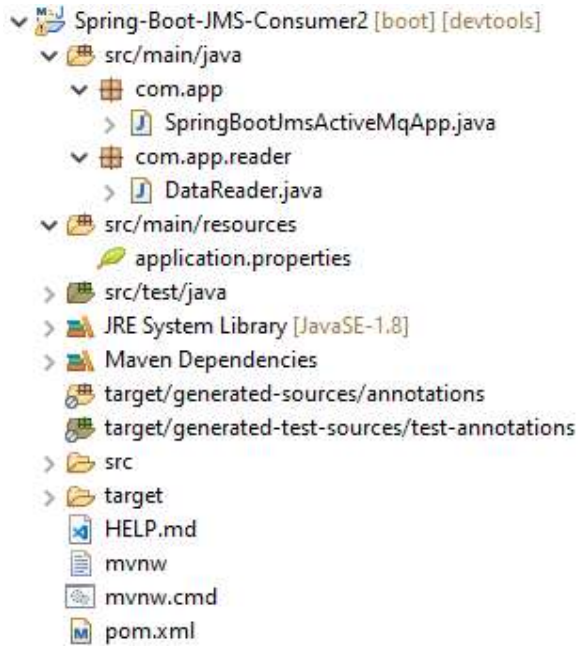
```
package com.app;
import    org.springframework.boot.SpringApplication;
import     org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJMSConsumerApp {

        public static void main(String[] args) {
                SpringApplication.run(SpringBootJMSConsumerApp.class, args);
                System.out.println("JMS Consumer Executed :");
        }
}
```

#2. Consumer class(MessageConsumer.java):--

```
package com.app.consumer;
import org.springframework.jms.annotation.EnableJms;
```

```
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@EnableJms  //optional in case of @JmsListener
@Component
public class MessageConsumer {
	@JmsListener(destination = "my-tpca")
	public void readMessage(String msg)
	{
		System.out.println("from consumer");
		System.out.println("msg is:"+msg);
	}
}
```

#16.b Folder structure of JMS Consumer2:--



#2:- Starter class (SpringBootJmsActiveMQApp.java):--

```
package com.app;
import    org.springframework.boot.SpringApplication;
import     org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJmsActiveMqApp {
```

```
	public static void main(String[] args) {
		SpringApplication.run(SpringBootJmsActiveMqApp.class, args);
		System.out.println("JMS Executed :");
	}
}
```

#2:- Reader class (DataReader.java):--

```
package com.app.reader;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@EnableJms  //optional in case of @JmsListener
@Component
public class DataReader {

	@JmsListener(destination="my-tpca")
	public void getMsg(String msg) {
		System.out.println("consumer#222");
		System.out.println("Message is:=>"+msg);
	 }
}
```

NOTE:-- send (String destinationName, Message (Creator MSG);
=>This method is used to send message from producer Application to MOM.
=>if destination not exist in MOM, Creates new one else uses same.

Execution Order:--

1. >Active MQ Server.
2. >Producer Applicaion Starter class (One Provider App).
3. >Consumer Application Starter class (One or Multiple Consumer Application).

Download and Setup for ActiveMQ:--

=>Get into dependency and click version
Example: <activemq.version>5.15</activemq> download Activemq same version.

Download Link:-- Go to below location

https://activemq.apache.org/components/classic/download/

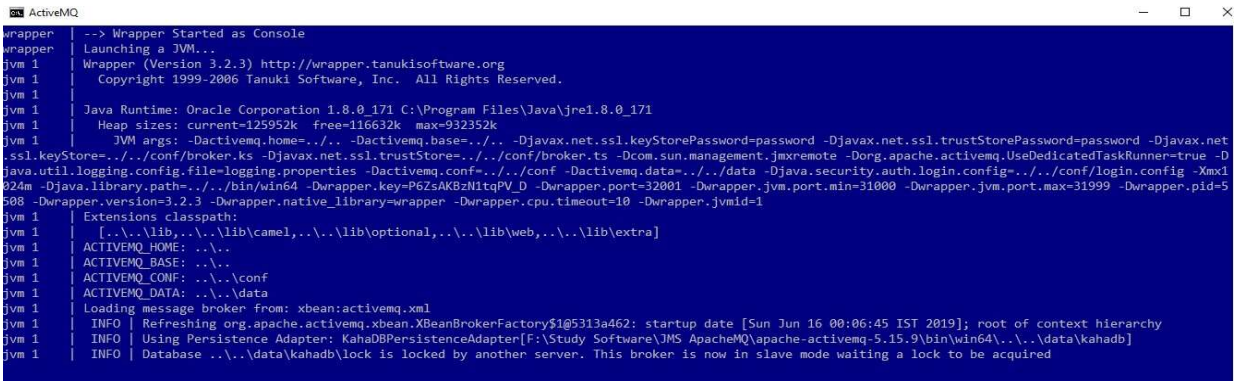
->Click on : apache-activemq-5.15.9-bin.zip

->Extract to one folder

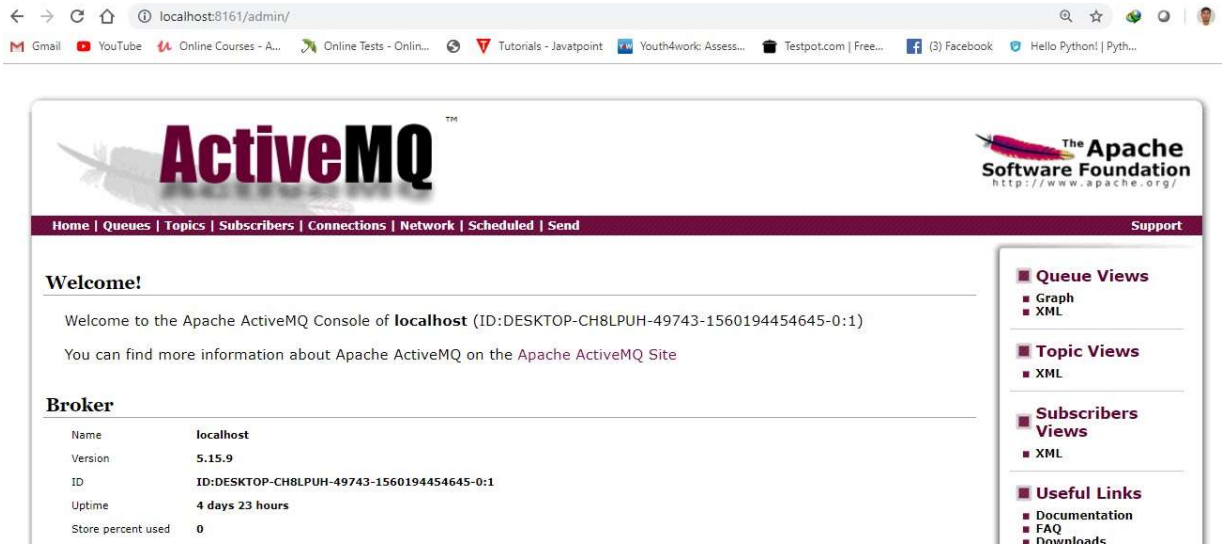
->Goto .. F:\Study Software\JMS ApacheMQ\apache-activemq-5.15.9\bin\win64 (\apache-activemq-5.15.9\bin\win64)

Execution process:--

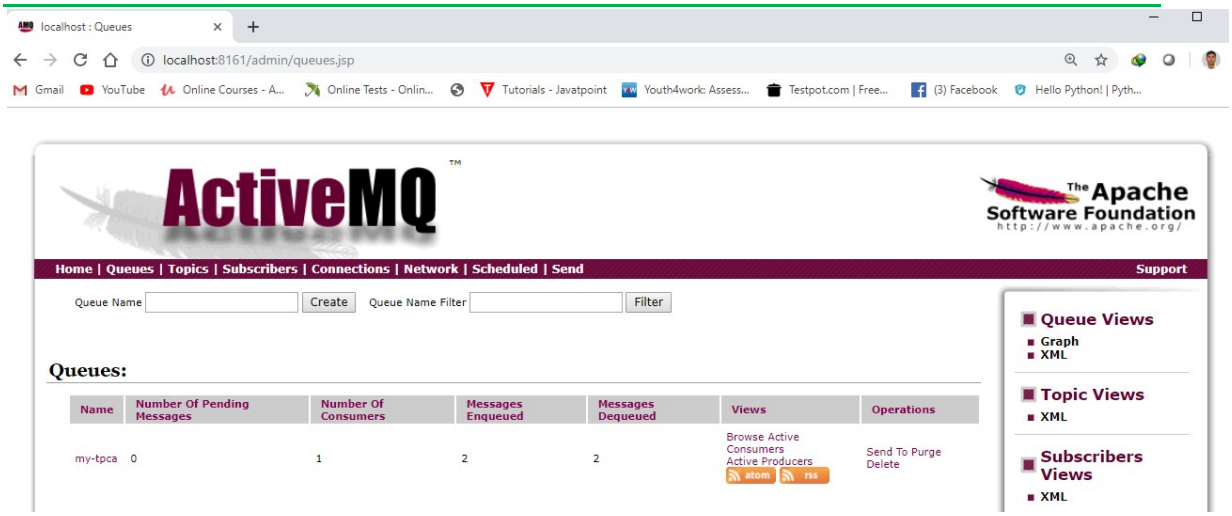
Step#1:- Double click on activemq.bat file



Step#2:- Goto browser and type URL: http://localhost:8161/admin



Step#3:- Click on Queues (P2P)/TOPIC (Pub-Sub) Menu option to see message list.



=>In case of Interface is provided with abstract method then we need to provide implementation using.

1>Child class
2>Anonymous inner class
3>Lambda Expression

=>Example#1:--

#1. Interface:--

```
public interface MessageCreator {
        Message creatMessage (Session s) throws JMSException;
}
```

#2.a. Implement class and Create Object:--

```
public Test implements MessageCreator {
        public Message createMessage(Session s) throws JMS Exception {
                return s.createMessage("Hello...")
        }
}
```

#2.b Anonymous inner class:--

Syntax:--

```
new interfaceName ( ) {
      //Override all methods
}
```

Code:--

```
new MessageCreator () {
public Message createMessage(Session s) throws JmsException {
        return s.createMessage("hello…");
        }
}
```

#2.c Lambda Expression:--

```
messageCreator mc = (s) -> {
  return s.createTestMessage("Hello….");
}
                    Or
Message.create mc = s ->s.createMessage("Hello…");
```

11. Spring Boot Apache Kafka Integration:-- Apache kafka is used for

=>Apache kafka supports AMQP for large data transfer for MQ applications over network.
=>Supports Real-time Data Streaming. It means read continuous and large data from external source like Float Files, Database, networks, etc…

=>It supports data reading/writing from
   a. Databases
   b. Flat File System
   c. Applications (Data Streams).

=>Kafka supports integration with any type of application (language Independent + Plugin required, default JAVA).
=>Kafka supports basic concepts of MQ like
   1. Data Streaming
   2. Web Activities
   3. Log-Aggregations
   4. Command-Components

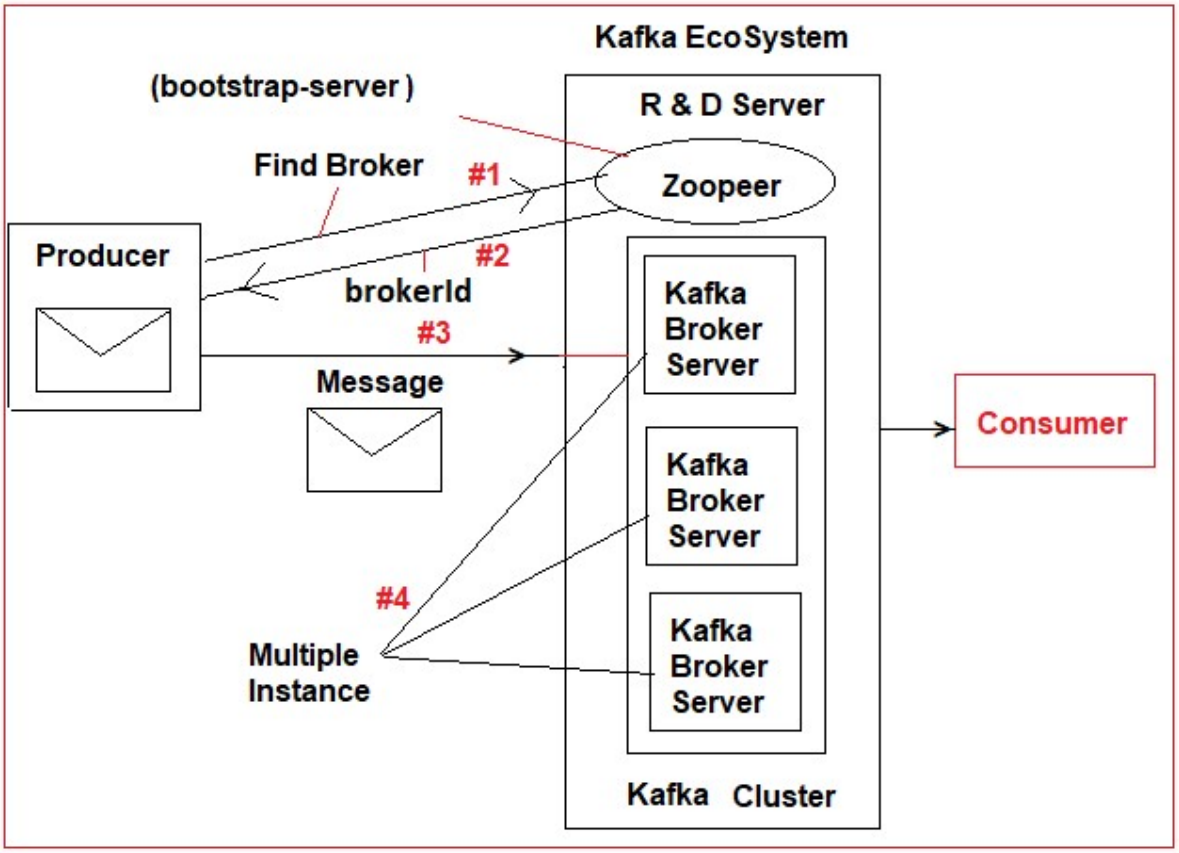
=>Kafka follows below concept even,

a.>Kafka uses Message Broker also called as **Broker Server** which supports MQ operations.
b.>Kafka supports **Load Balancing** for Broker Software to avoid more traffic, i.e. called as **Kafka cluster.** In general cluster is a group of Broker servers (1 to n).
c.>Kafka supports only **Topics** (Pub/Sub Model) and it is only default also.
d.>Kafka Cluster Auto-Scaling is done by **Bootstrap server** (AKA Zookeeper). It behaves as R&D Server.

e.>Data is sent to Topic in <K, V> format. K=TopicName (Destination), V=Data.
f>All these Broker instances must be Registered with Registory and Discovery (R&D) Server. Kafka comes with default **"Zookeeper R&D Server".**
g.> This complete Kafka Software is called as **Kafka EcoSystem** (Kafka Eco-System = Kafka Cluster + Bootstrap Server).

h>. Data Partitions concept is used by kafka to send large data.
i.>Message Brokers will persist the message (save into their memory) to avoid data lose in case of consumer non-available or broker is down.
j.>Kafka works as Protocol independent i.e. works for TCP, FTP, SMTP, HTTP… etc)

Kafka EcoSystem Diagram:--

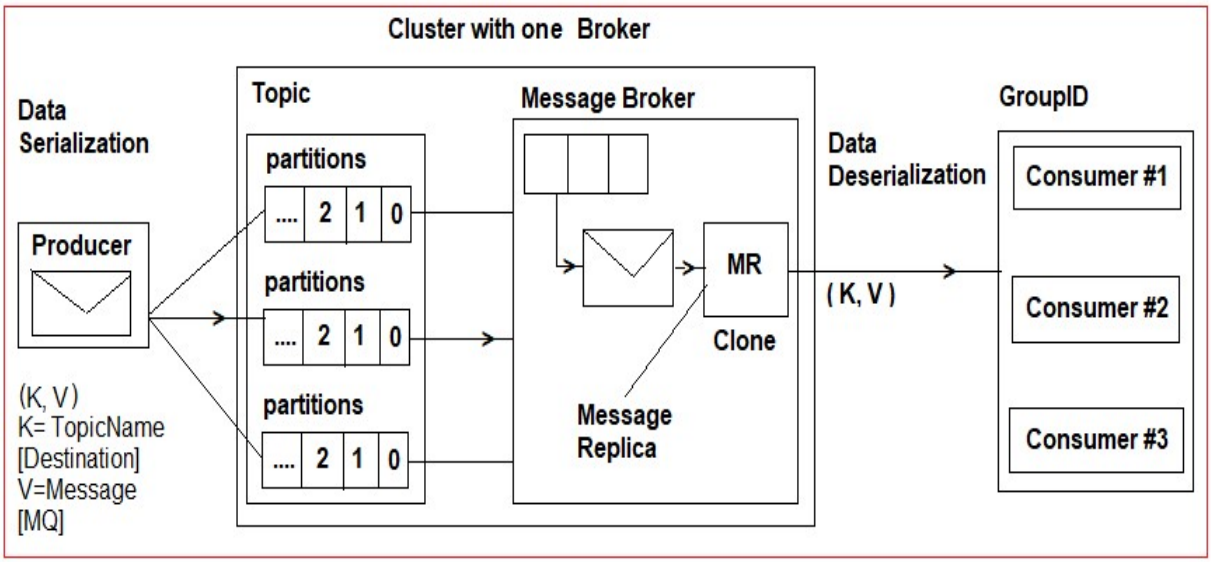


Execution Flow:--

=>Producer Application should get Message Broker details from R & D Server (zookeeper) known as bootstrap-server).
=>Producer gets unique-id (InstanceId) of Message Broker server and sends message to Broker.Producer use **KafkaTemplate<K, V>** to send data to one Broker server.
=>Message Broker will send this message to one or multiple consumers.

=>Producer sends data <k, V> format in Serialized (Converting to binary/Characters formats). Here K=Destination (Topic name) and V= Message.

=>Every Message will be partitioned into Multiple parts in Topic (Destination) to avoid large data sending, by making into small and equal parts (some time size may vary).
=>Broker reads all partitions data and creates its replica (Clone/Mirror obj) to send message to multiple consumers based on **Topic** and **Group-Id**.
=>At Consumer side Deserialization must be applied on K, V to read data. Consumer should also be linked with bootstrap-server to know its broker.



=>Partitions are used to breakdown large message into multiple parts and send same to multiple brokers to make data destination in parallel.

Message Replica:-- It creates multiple copies to one message to publish one message to multiple Consumers.

Kafka Producer and Consumer Setup Details:--

=>For Producer Application we should provide details in application.properties (or .yml).
=>Those are
bootstrap-servers=localhost:9092
key-serializer=StringSerializer

value-serializer=StringSerializer

=>By using this Spring Boot creates instance of “KafkaTemplate<K, V>” then we can call send(k, v) method which will send data to Consumer.

->Here : K=Topic Name, V= Data/Message

=>For Consumer Application we should provide details in application.properties (or .yml)
=>Those are
bootstrap-servers=localhost:9092
key-deserializer=StringDeserializer
value-deserializer=StringDeserializer
group-id=MyGroupId

=>By using this Spring Boot configures the Consumer application, which must be implemented using : @KafkaListener(topics=”—“, groupId=”—“)

Kafka Download and Setup:--
Apache Kafka comes with Unix based software called as Scala 2.x

=>Download link (https://kafka.apache.org/downloads)
=>Choose one mirror=>Extract once (.tgz -> .tar)
=>Extract one more time (.tar -> folder)
=>Open folder and create --.bat file for bootstrap Server (Zookeeper).

***** bat files in kafka to be created*****
1>Server.bat
=>Starts Kafka Server (Message Broker)
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

2>cluster.bat
=>Starts Zoopeer with Kafka Cluster design
.\bin\windows\kafka-server-start.bat .\config\server.properties

Coding Steps:--

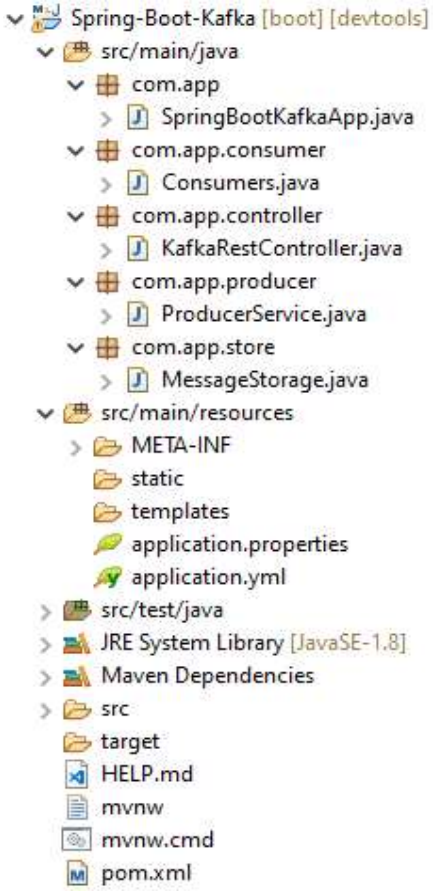
Step#1:- Choose spring apache kafka Dependency while Creating project (Which gives Integration of Spring with Kafka).

groupId : com.app
artifactId : SpringBootKafkaApp
version : 1.0

Kafka Dependency:--

```
<dependency>
   <groupId>org.springframework.kafka</groupId>
   <artifactId>spring-kafka</artifactId>
</dependency>
```

#17. Folder Structure of Kafka:--



Step#2:- add key= value pairs in application (.properties/ .yml) file.

application.properties:--

```
server.port: 9988
#Producer properties
```

```
my-app-topicname: sampletopic
spring.kafka.producer.bootstrap-servers: localhost:9092
spring.kafka.producer.key-serializer:
org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer:
org.apache.kafka.common.serialization.StringSerializer

#Consumer properties
spring.kafka.consumer.bootstrap-servers: localhost:9092
spring.kafka.consumer.key-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer:
org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.group-id: group-id
```

properties.yml:--

```
server:
  port: 9988
my:
  app:
    topicname: sampletopic
spring:
  kafka:
    producer:
      bootstrap-servers: localhost:9092
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer:  org.apache.kafka.common.serialization.StringSerializer

    consumer:
      bootstrap-servers: localhost:9092
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      group-id: groupId
```

Step#3:- Define MessageStorage class

```
package com.app.store;
import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Component;
```

```
@Component
public class MessageStorage {
        private List<String> list= new ArrayList<String>();

        public void put(String message) {
            list.add(message);  //Write the logic to store data in database
        }
        public String getAll() {
           return list.toString();
        }
}
```

Step#4:- Define Consumer class

```
package com.app.consumer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;
import com.app.store.MessageStorage;

@Component
public class Consumer {
        @Autowired
        private MessageStorage storage;

        @KafkaListener (topics="${my.app.topicname}", groupId="groupId")
        public void consume (String message) {
                storage.put(message);
        } }
```

Step#5:- Define Producer code

```
package com.app.producer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
```

```
public class Producer {

	@Value("${my.app.topicname}")
	private String topic;

	@Autowired
	private KafkaTemplate<String, String> template;

	public void sendMessage(String message) {
		template.send(topic, message);
	}
}
```

Step#6:- Define KafkaRestConstroller class

```
package com.app.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import com.app.producer.Producer;
import com.app.store.MessageStorage;

@RestController
public class KafkaRestController {

	@Autowired
	private Producer producer;

	@Autowired
	private MessageStorage storage;

	@RequestMapping("/send")
	public String readInMessage(@RequestParam String message)
	{
		producer.sendMessage(message);
		return "message sent!!";
	}
```

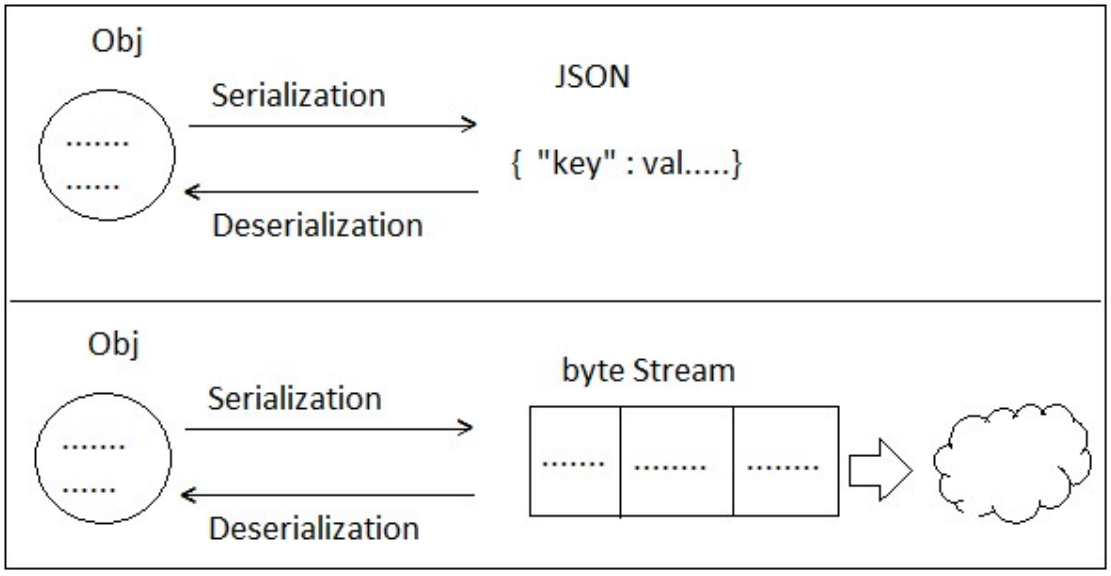
```
        @RequestMapping("/view")
        public String viewMessage() {
                return storage.getAll();
        }
}
```

Coding order:--

1. application.properties:--
2. MessageStorage.java
3. ConsumerService.java
4. ProducerRestConstroller.java
5. KafkaRestController.java

NOTE:-- Use KafkaTemplate <K, V> at producer application to send message(V) to given Topic (K).

NOTE:-- Use @KafkaListener (topics="….", groupId="….") at consumer side to read message (V) using topic (K) with groupId(G).



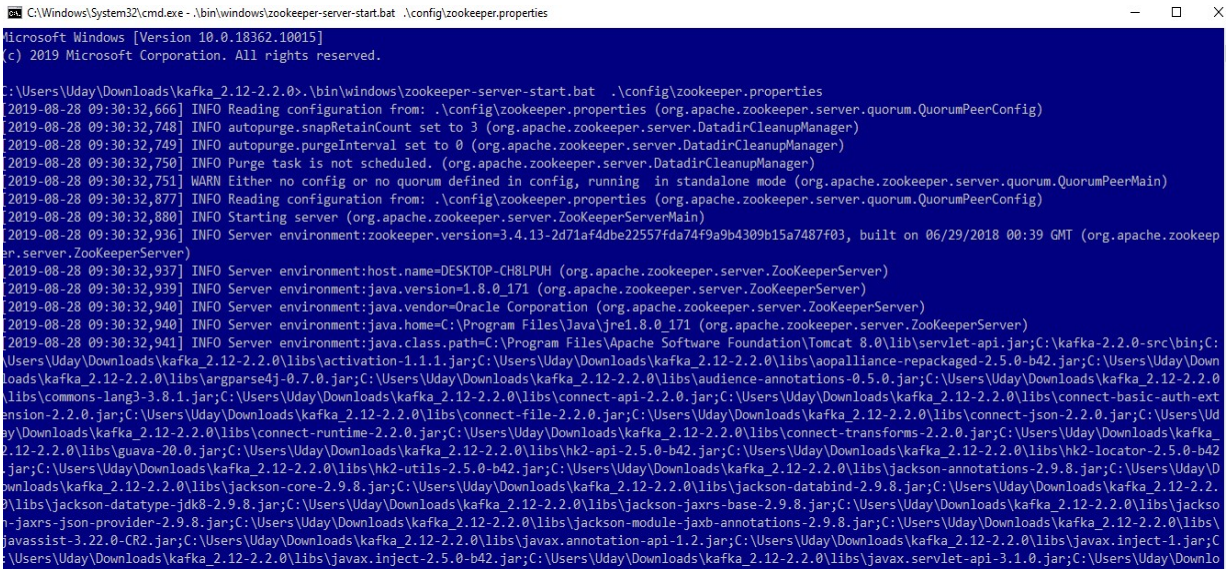
Execution Orde:--

1>Start the Zookeeper Server

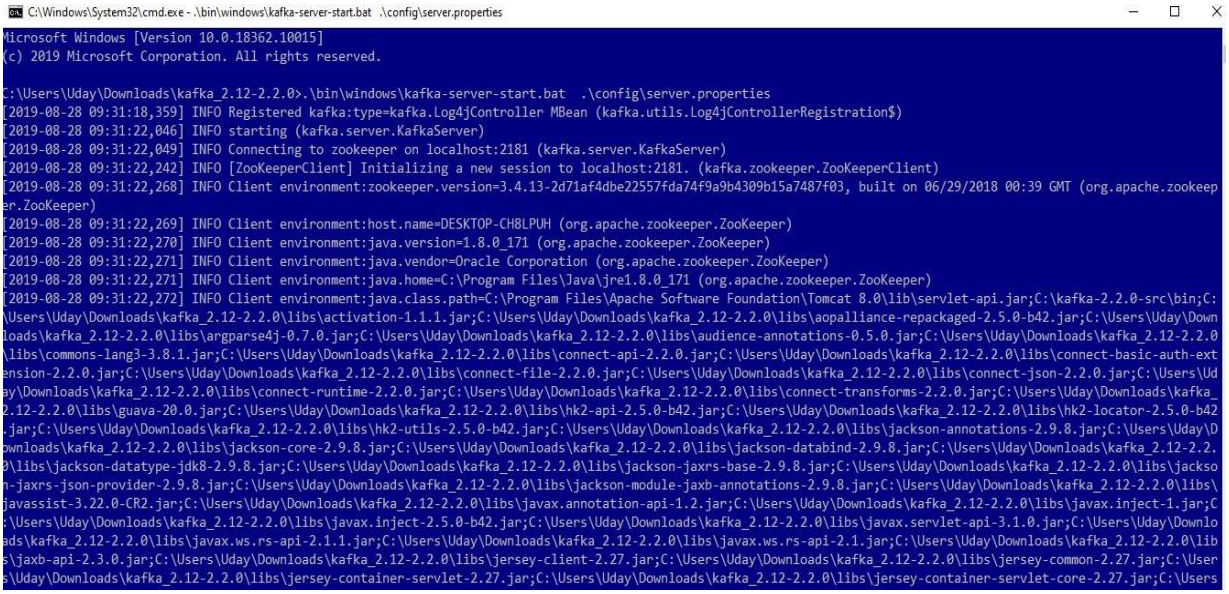
2>Start kafks-server (Cluster)

3>Start Application Starter class (Provider, Consumer)

#1:- Start the Zookeeper Server

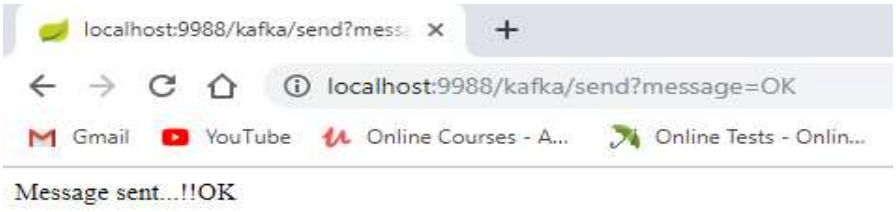


#2:-- Start the kafka-Server (Cluster)

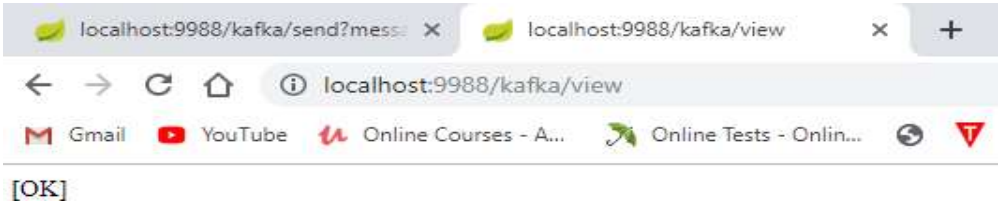


3#:-- ***Run Starter class and enter URLs:--**

1. http://localhost:9988/kafka/send?message=OK



2. http://localhost:9988/kafka/view



12. Spring Boot with Apache Camel:--

Routing:-- It is a process of sending large data from One Application (Source) to another Application (Destination).

=>A source/destination can be File System (.xml, .txt, .csv, .xlsx, .json,...etc), Database (Oracle DB, MySQL DB) or Message Queues using JMS (Active MQ) etc.

=>Apache Camel is open Source and Light weight "**Conditional based Routing Engine**" which supports filtering and processing. It behaves like mediator software between multiple source and destination.

=>Apache Camel is language independent software implemented in Java but also supports integration with different language like Hadoop, Bigdata, PHP, Python, & JavaScript… etc.

=>Compared to Spring Batch Integration tool Apache camel is a light weight tool.

=>Camel supports reading data from different sources even like HTTP, FTP, JMS protocols based.

=>It supports data processing (convertions, calculations, like XML--->JSON, JSON--->Object, Object--->XML, and XML ---> EXCEL etc.s

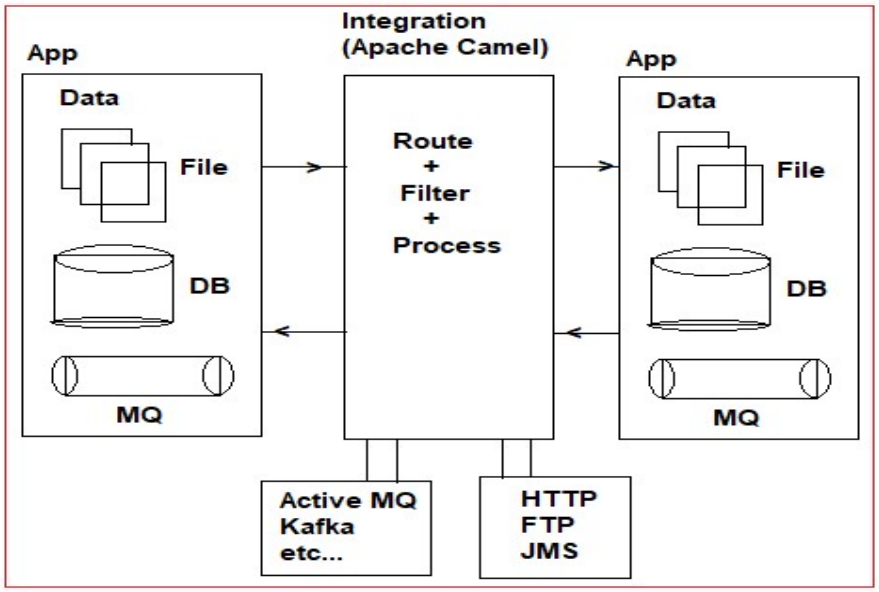
=>It has light weight engine, compared to spring Batch it is faster.

=>Camel mainly supports 3 operations:

a.>Routing :-- Data Transfer

b.>Processing :-- Data Convertion

c.>Filtering:-- Conditional checking



=>Camel supports easy coding using “EIP” (Enterprise Integration Pattern).
=>Format looks like :

[source-details]
.[filter-modes]
.[processing]
.[destination]

Implementing Camel Routing in Spring boot:--

Step#1:- Create one Starter project with Apache Camel dependency. Or else add below mention dependency in pom.xml.

#1:- In pom.xml, we must add below dependency which supports Spring boot integration.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-spring-boot-starter</artifactId>
  <version>2.23.2</version>
</dependency>
```

pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.1.1.RELEASE</version>
                <relativePath /> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.app</groupId>
        <artifactId>Spring-Cloud-Camel-EIP</artifactId>
        <version>1.0</version>
        <name>Spring-Cloud-Camel-EIP</name>
        <description>Demo project for Camel Integration</description>

        <properties>
                <java.version>1.8</java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.apache.camel</groupId>
                        <artifactId>camel-spring-boot-starter</artifactId>
                        <version>2.23.2</version>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
```

```
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Step#2:- Camel avoid main method (thread) execution control and run as independent while working with Spring boot, our starter class is main method. So we should add key=value in properties file as

application.properties:--

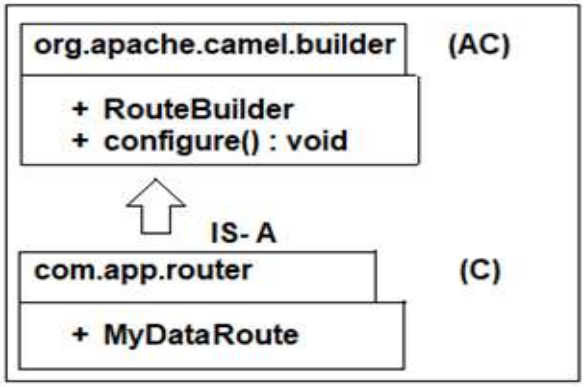
```
camel.springboot.main-run-controller=true
```

Step#3:- Define one RouteBuilder class and configure details of routing, filtering and processing.

=>To implement this we need to write one Router class using **"RoutingBuilder (AC)"** provided by Apache camel having one abstract method: **configure()** which contains coding format like:

```
from (SourceLocation)
   .[filter] . [process].
   .to (DestinationLocation)
```

=>Here Location can be URL/Local File System DB, JMS (Message Queues)… etc.

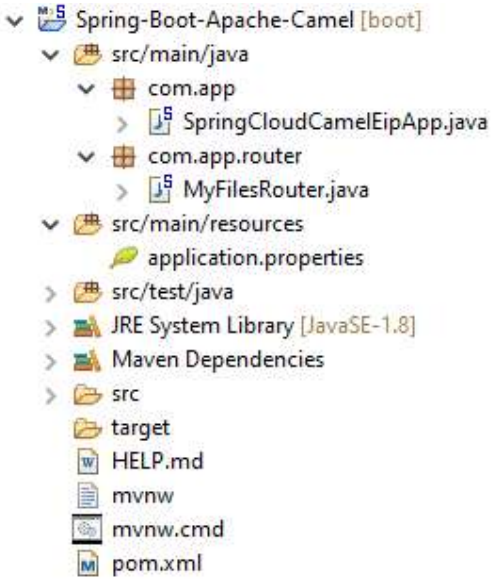


Coding Steps:--

Step#1:- Create Spring Boot starter application with dependencies : Apache Camel

GroupId : com.app
ArtifactId : Spring-Boot-Apache-Camel
Version : 1.0

#18. Folder Structure of Spring Boot Integration with Apache Camel:--



Step#2:- open application.properties (.yml) and add main method-control key as true.

application.properties:--

camel.springboot.main-run-controller=true

application.yml:--

```
camel:
  springboot:
     main-run-controller:  true
```

Step#3:- Define Router class with EIP pattern with file transfer logic.

```
package com.app.router;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesRouter extends RouteBuilder  {
        public void configure() throws Exception  {
                //with static location
                from ("file:D:\\source").to("file:D:\\destination");
        }
}
```

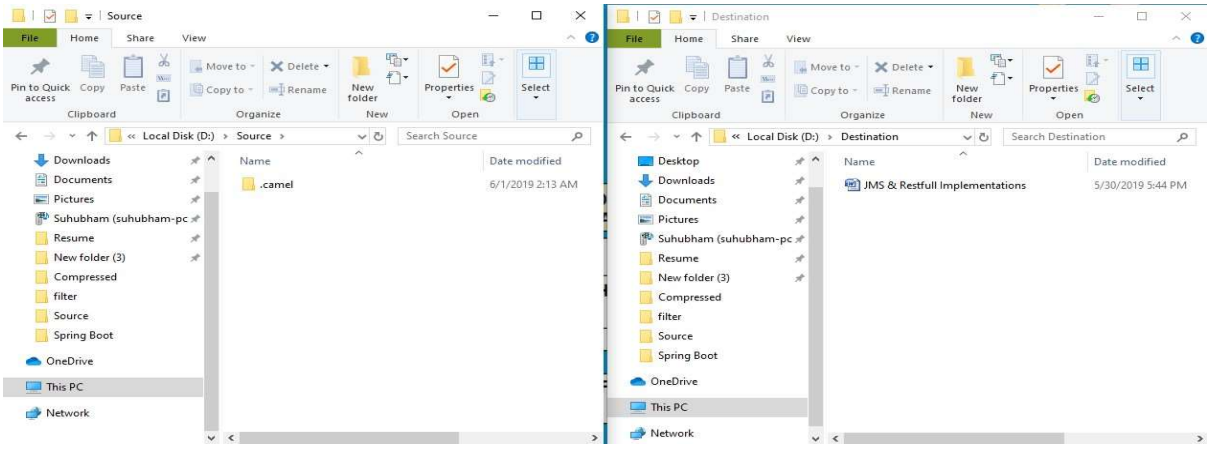
Step#4:- Create two folder : Source and Destination in any Drive (“D: drive”).

Step#5:- Start Application and place files in “D:/source” which will be copied automatically into “D:/destination”.

NOTE:--In source file **.camel** folder is created automatically which contains Copied file and in Destination folder only File will coped but no any folder will be created.

Output Screen :--

Folder#1: Source **Folder#2: Destination**



1. EIP Patterns by Apache Camel:-- EIP stand for **"Enterprise Integration Patterns"** used to define short form code for.

->Data Routing
->Data Filtering
->Data Processing

#1:- from ("file:source") .to("file:destination");
=>It will copy files from source to destination by taking files in backup folder in source with name **.camel**.
=>It supports even same file sending with new data (Operation to Override Program).

#2:- from ("file:source?noop=true") .to("file:destination");
=>To avoid sendingsame files with different (Modified) data again "No operation to override program" should set as true.

#3:- from ("{{source}}").to("{{destination}}");
=>Here **{{location}}** indicates dynamic location that is given as input passing using Properties/Yml files, System args, command line inputs etc.
=>It means locations provided at runtime. For above example add below key in properties file.

application.properties:--
camel.springboot.main-run-controller=true
my.source=file:D://Source?noop=true
my.destination=file:D://Destination

Example EIPs:--
#1>Dynamic Location:--
Location (source/destination) can be passed at runtime using properties/yml files, config server, system arguments… etc.
=>To indicate Location (URL file System, DB, MQ…) comes at runtime use format: {{location}}

Code changes:--
a. applictaion.properties:--
server.port=2344

```
camel.springboot.main-run-controller=true
my.source=file:D://Source?noop=true
my.destination=file:D://Destination
```

b. Router class code

from("{{my.source}}").to("{{my.destination}}");

Ex:--

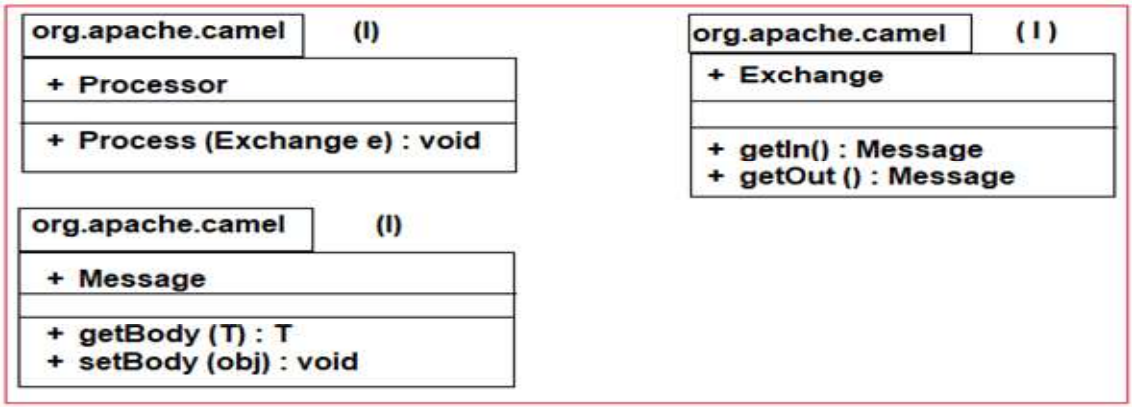
```
package com.app.router;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesRouter extends RouteBuilder {
	public void configure() throws Exception {
		//With Dynamic Location
		from("{{my.source}}").to("{{my.destination}}");
	}
}
```

#2 Data Processing Using Apache Camel:--

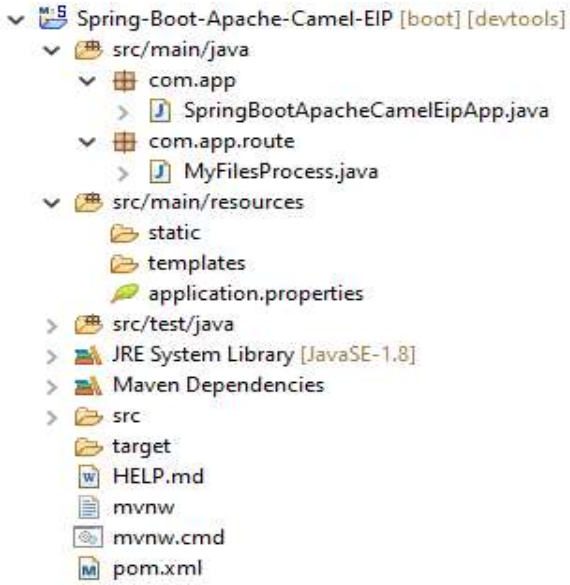
In realtime data will be converted or modified based on requirements. Like XML--->JSON, JSON--->Simple Text, XML--->CSV, CSV--->SQL Format etc..
i.e data converted from one format to another format which can be done using 3 supporting interfaces. Those are:

1>Processor (I) 2>Exchange (I) 3>Message (I)

=>Apache Camel has provided "Processor (I)" defined in package org.apache.camel which has only one abstract method (Functional Interface) i.e process (…).
=>Processor (I) takes help of Exchange (I) to read In message and to set out message.
=>Exchange depends on Message (I) which has message body (get/set method).
=>Our files data is stored in Message (I) format. To read this data set, get methods are provided.



#19. Folder Structure of Spring Boot Apache Camel-EIP:--



Process#1 Write inside Router (Impl) class:--

```
package com.app.route;
import  org.apache.camel.Exchange;
import   org.apache.camel.Message;
import org.apache.camel.Processor;
import          org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProcess extends RouteBuilder {

  public void configure() throws Exception {
        from ("file:D:/Source").process(new Processor()
        {
```

```
        public void process (Exchange ex) throws Exception
        {
              //#1 Read Input Message
              Message m = ex.getIn();
              //#2 Read Body from message
              String body = m.getBody(String.class);
              //#3 do processing
              body ="modified ::"+body;
              //#4 Writer data to out message
              Message m2 = ex.getOut();
              //#5 Set body (Data) to out message
              m2.setBody(body);
             }
       })
       .to("file:D:/Destination?fileName=myFile.txt");
      }
}
```

***Note:-** Here fileName indicates new name for modified message. It can also be passed at run time.

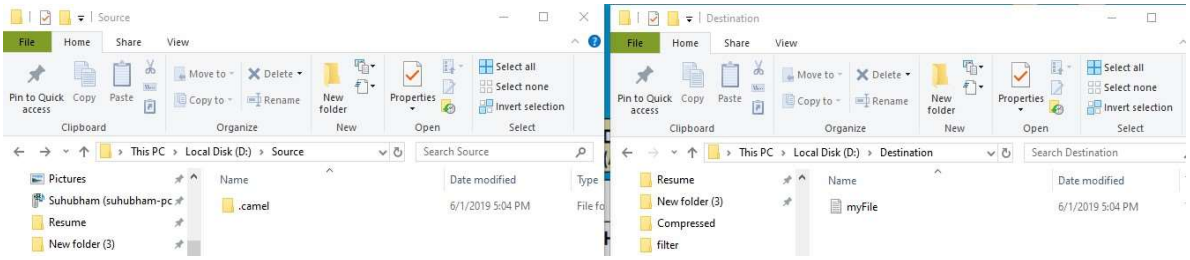
=>fileName must be provided in-case of processor is used, else file generated with ID which has no extension details.

Process#2 Using Lambda Expression## code:--

```
package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProcess extends RouteBuilder {
      public void configure() throws Exception
      {
         from("file:D:/source")
            .process(ex->{
                  String body=ex.getIn().getBody(String.class);
                  body="New AA modified ::"+body;
                  ex.getOut().setBody(body);
```

```
        })
            .to("file:D:/Destination?fileName=myFile.txt");
        }
}
```

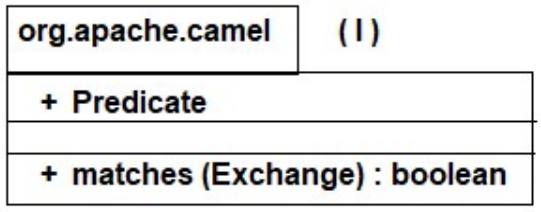
Output Screen:--



#3 Filters using Predicates:--

Predicate is a type expression which returns either true or false based on condition checking. Predicate is a Boolean expression after executing a task.

=>If true next level steps are executed else false execution stopped here only.



=>ValueBuilder (C) is used to execute required Predicates, using method: contains(), startWith(), isNull(), …etc.

=>We can get ValueBuilder (C) object using methods body(), header().

=>header() indicates checking on file name, extension, location … etc.

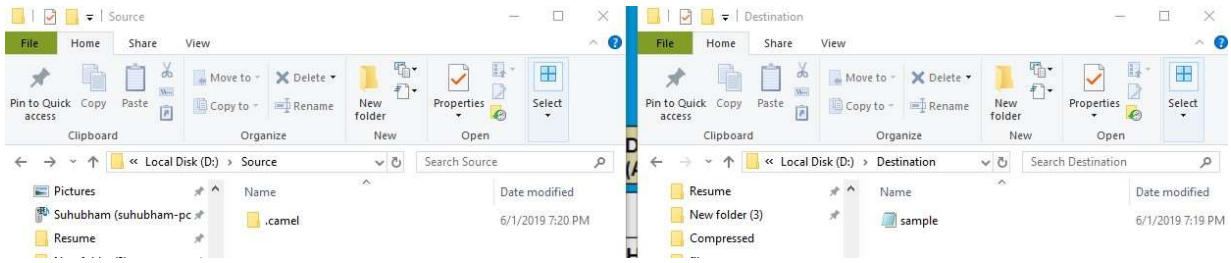
=>body() indicates file data/ content check

Ex#1:- file name having word sample

```
package com.app.route;
import org.apache.camel.Exchange;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProcess extends RouteBuilder {
```

```
    public void configure() throws Exception {
        from("file:D:/Source").filter(header(Exchange.FILE_NAME)
        .contains("sample"))
        .to("file:D:/Destination");
        }
}
```

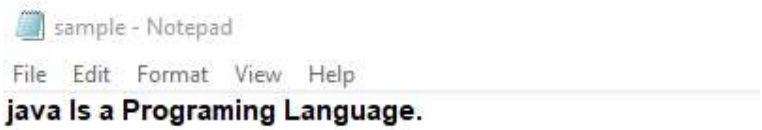
Output:--



Ex#2:- File body starts with word java

```
package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProcess extends RouteBuilder
{
     public void configure() throws Exception {
        from("file:D:/source")
        .filter(body().startsWith("java"))
        .to("file:D:/destination");
    }
}
```

Output Screen file contents:--



Conditional based Routing [Choose –when-otherwise]:--

=>Apache camel supports data routing based on predicates (Type expression) which support verify conditions.

=>A predicate is a Boolean expression after executing a task.

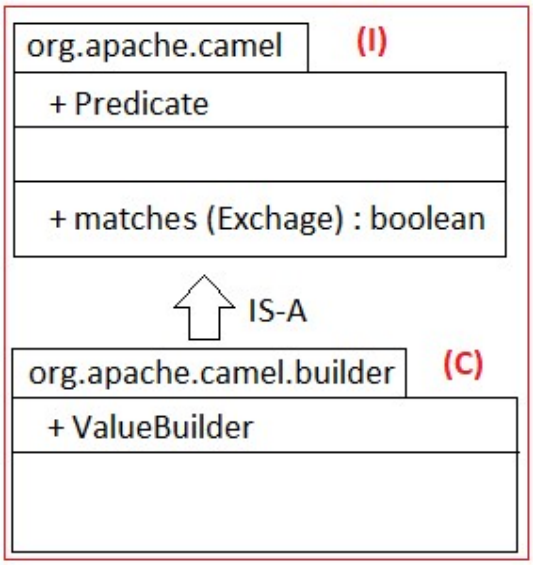
=>To verify inputs files use methods header() and body() given by RouteBuilder.

=>Here header () is usd to verify file details like “filename”, extension, filesize, … etc.

=>Here body () is used to verify file data like “fileContains”, startWith, endWith … etc.

=>To read message (head + body) camel provides Exchange object.

=>To Handle switch-case concept for data routing use choose-when-other.



=>By using ValueBuilder (C) we can do filter() and choice() based predicates execution.

a. filter:-- This is used to check one given boolean expression if true execute next step, else does nothing.

RouteBuilder Impl Class for Filter:--

Example#1:--

```
package com.app.route;
import org.apache.camel.Exchange;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProces extends RouteBuilder {
	public void configure() throws Exception {

		from("file:D:/Source")
		.filter(header(Exchange.FILE_NAME)
```

```
                //.startsWith("Employee")
                //.endsWith("-data.xml")
                .contains("employee"))
                .to("file:D:/Destination");
        }
}
```

Example#2:--

```
package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesProces extends RouteBuilder {
        public void configure() throws Exception {
                from("file:D:/Source")
                .filter(body().startsWith("Hello"))
                .to("file:D:/Destination");
        }
}
```

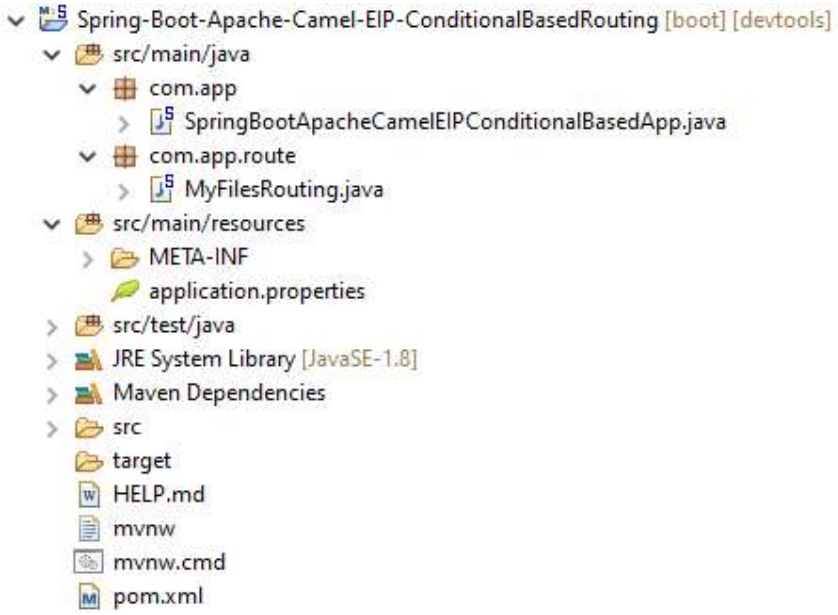
b. choice():-- This is used to execute multiple condition checks in a order, It behaves like switch-case concept.

=>We can provide multiple when() with predicates even with process(..). Finally otherwise() is executed if all conditions are fail.

Format looks like:--

```
from("source")
    .choice()
    .when(condition#1).to("destinatation#1")
    .when(“condition#2).to("destinatation#2")
    ….
    otherwise().to("destinatation#n")
```

#20. Folder Structure of ConditionalBased Routing:--



#1 Starter class (SpringBootApacheCamelEIPConditionalBasedApp):--

```
package com.app;
import    org.springframework.boot.SpringApplication;
import     org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootApacheCamelEIPConditionalBasedApp {

public static void main(String[] args)
 {
SpringApplication.run(SpringBootApacheCamelEIPConditionalBasedApp.class, args);
          System.out.println("Spring Boot Camel Executed :");
   }
}
```

#2 MyFilesRouting.java:--

```
package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyFilesRouting extends RouteBuilder {
```

```
    @Override
    public void configure() throws Exception {
        from("file:D:/Source")
        .choice()
        .when(body().startsWith("java")).to("file:D:/Destination?fileName=a.txt")
        .when(body().startsWith("xml")).to("file:D:/Destination?fileName=b.txt")
        .when(body().startsWith("json")).to("file:D:/Destination?fileName=c.txt")
        .otherwise().to("file:D:/Destination?fileName=d.txt");
    }
}
```

Input file contains:--

#1 If file body contains start with java then create a new file a.txt in destination. Where as body contain is case sensitive.

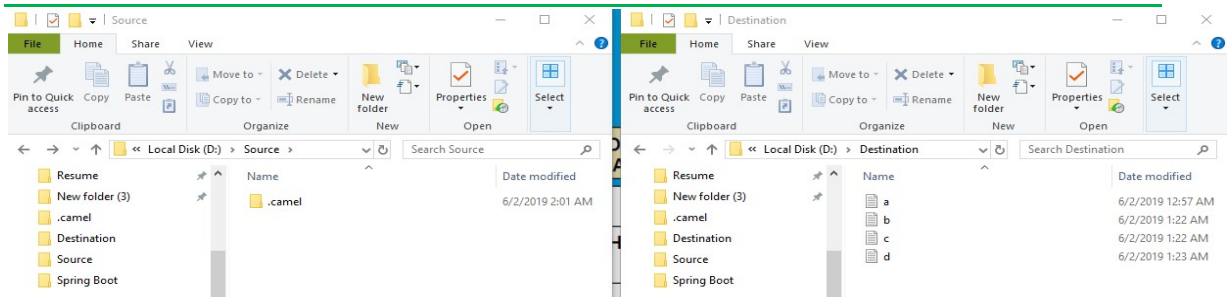
#2 If file body contains start with xml then create a new file b.txt in destination. Where as body contain is case sensitive.

#3 If file body contains start with json then create a new file c.txt in destination. Where as body contain is case sensitive.

#4 If file body contains is not start with **java, xml, json** then create a new file d.txt in destination.

Output Screen:--

Source File:-- **Destination File:--**



2. Apache Camel Intergradations with Active MQ :--

=>Camel supports read/write data from MQ source using EIP patterns and MQ Broker.

=>Patterns used to communicate with MQ using camel is : jms<type>:<destinatation>

=>Here type can be queue or topic. To use this we need to define protocol with prefix "jms", given as

Example : 1.> jms:queue:info (queue Name)

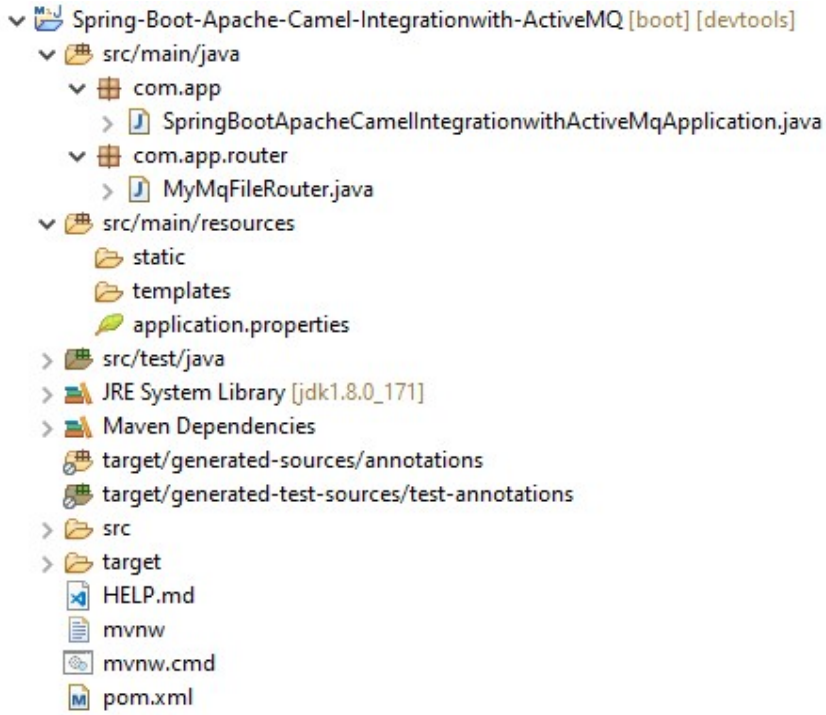
2.> jms:topic:news (topic Name)

=>For this coding along with ActiveMQ and Camel Dependencies we should add ActiveMQ-pool and camel-jms integration dependencies.

Step#2:- Create project and add below dependencies in pom.xml file

```
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
 </dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-jms</artifactId>
    <version>2.24.0</version>
</dependency>
```

#21. Folder Structure of Apache Camel Intergradations with ActiveMQ:--



Step#3:- Provide camel and ActiveMQ properties

application.properties

```
camel.springboot.main-run-controller=true
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

Step#4:- Define Routers

```
package com.app.router;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyMqFileRouter extends RouteBuilder {

	@Override
	public void configure() throws Exception {
		//1. Sending file Data from Source to Message Queue
/*  from("file:D:/Source?fileName=mydata.txt") .to("jms:queue:outdata"); */
```

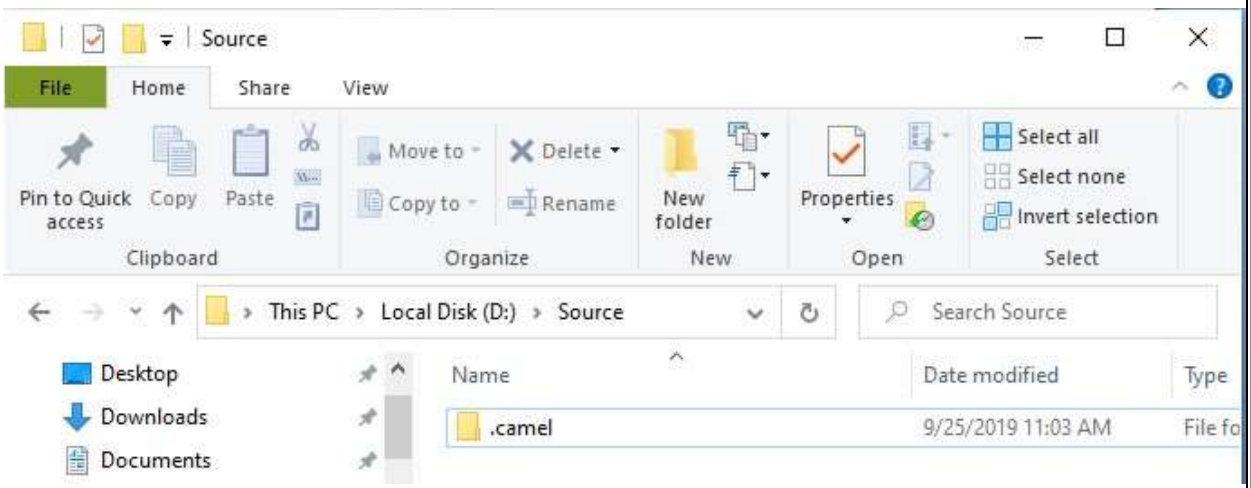
```
                //2. Sending Data from MessageQueue to Destination folder
                from("jms:queue:indata")
                .to("file:D:/Destination?fileName=mydata.txt");
            }
}
```

Execution order:--

a. Run ActiveMQ
b. Run Starter class
c. Create queue “abc”
d. Click on send to link
e. Enter message and send button
f. Open file and view data

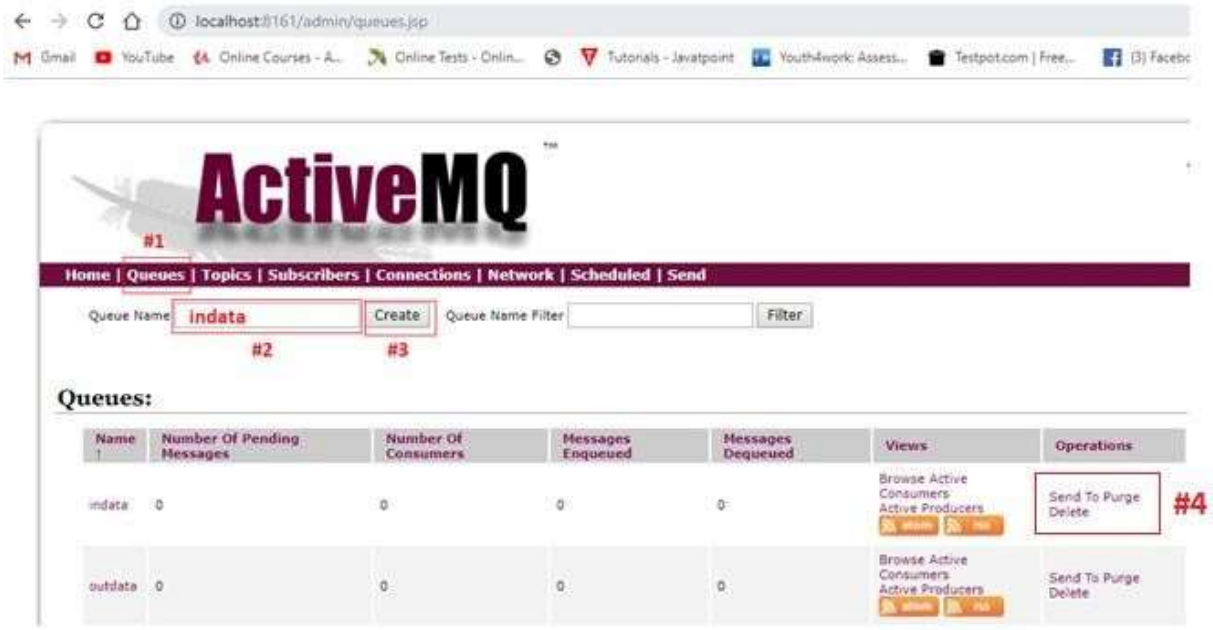
Step#5:- Run starter class and start ActiveMQ using bat
Step#6:- Login to MQ (http://localhost:8161/admin)
=>Click on menu Queue.
=>Enter Queue name and click create [2 queues : outdata, indata]

#1:- Sending data from file to MessageQueue:--
=>Goto Specified file location like(D:/Source) and copy any file or keep any file befor starting Spring starter class.



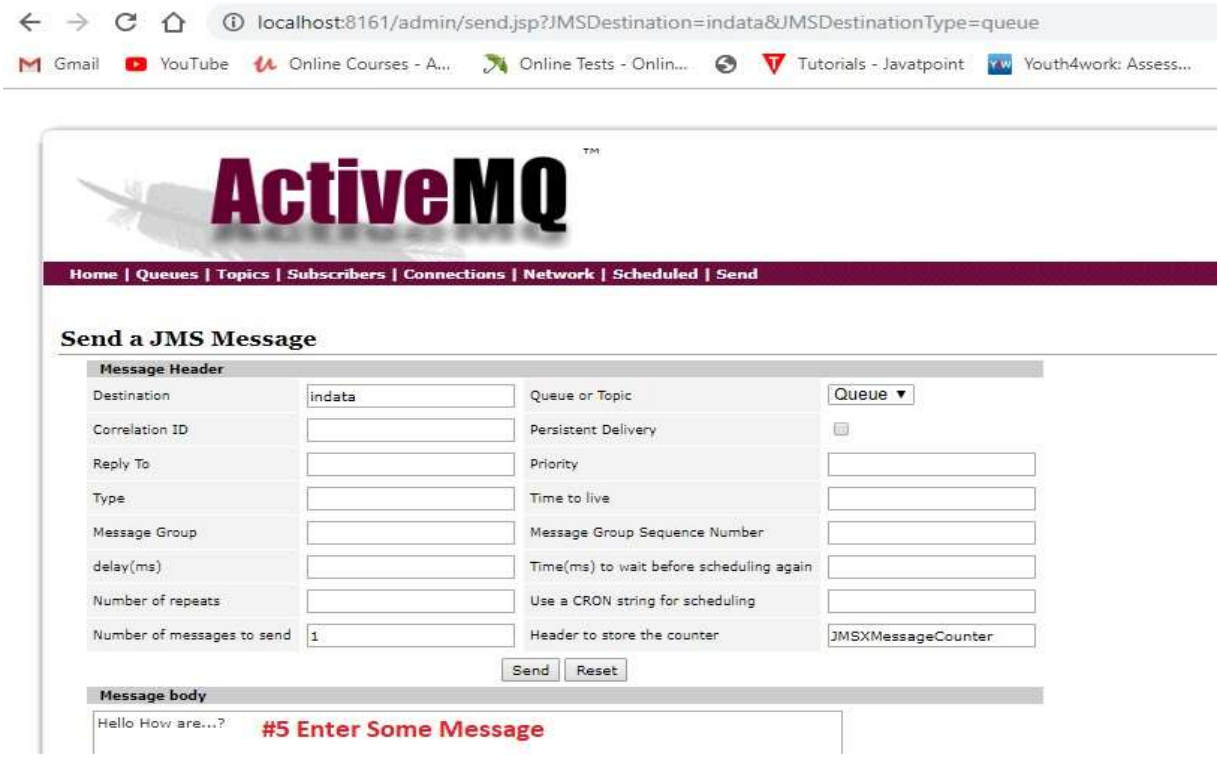
NOTE:-- If you keep any file before, after sending it will be automatically converted to **“.camel”** folder.

#2:- Screen for send data from MessageQueue to Destination:--



=>Click on “send To” option on “indata” queue, Enter message and press send.

=>File will be copied to destination folder.



=>Goto inside D:/Destinatation folder and see the output

3. ApacheCamel Integration with JDBC:--

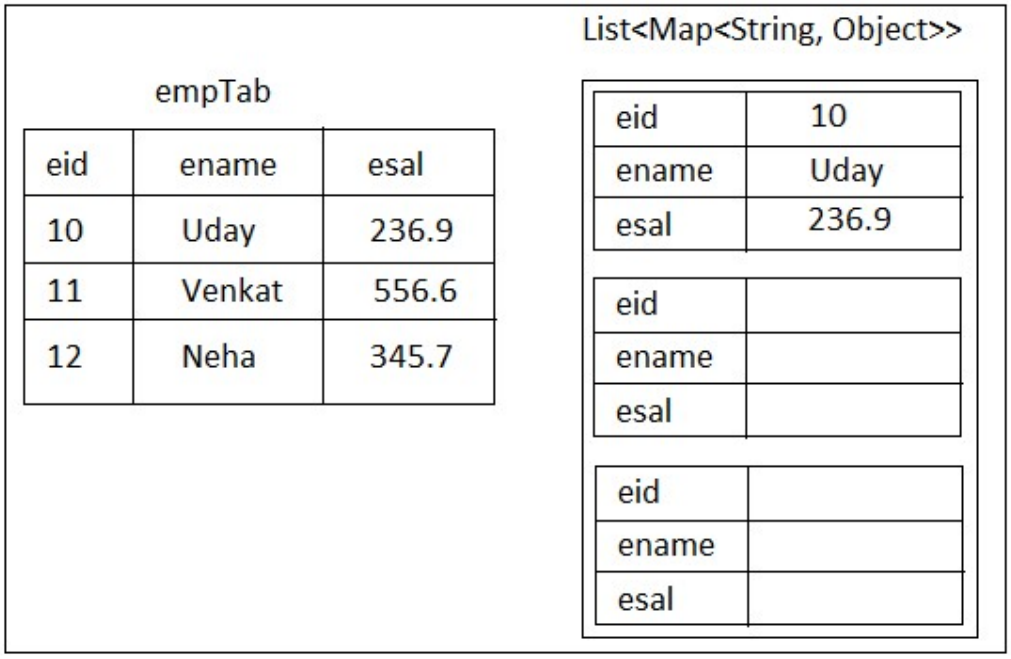
Camel supports reading data or writing data using JDBC took any database like MySQL, POSTGRES, H2, ORACLE etc… for this we need to provide dataSource Object as input to Camel.

=>Here DataSource is an Interface defined in Javax.sql package we need to configure its implementation class object.

=>By using one dataSource and SQL query we can fetch data from Database table.

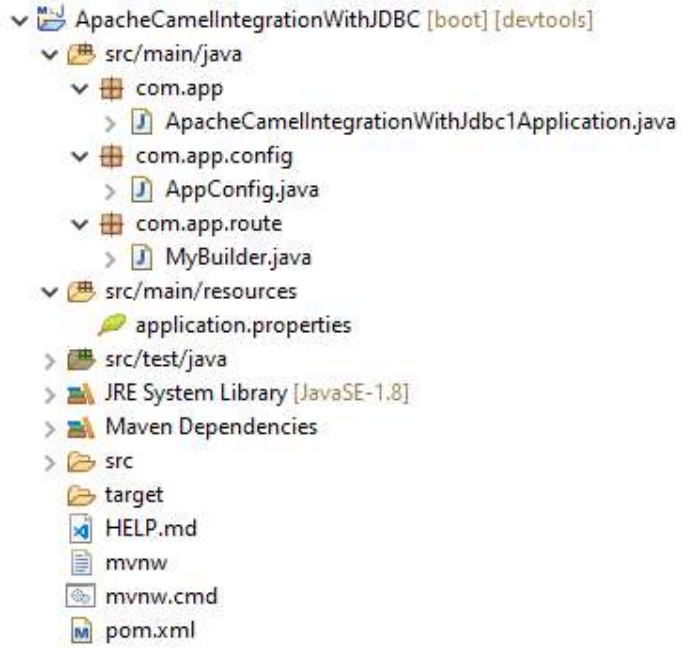
->It will be converted to List<Map<String, Object>> [K=Key=String type, V=Value=Object Type].

=>ResultSet data is converted to List<Map> format by Camel internally, example.



Ex#1: Camel SQL-MQ Integration

22. Folder Structure of Camel JDBC (SQL)-MQ :--



Step#1:- Create project using Apache Camel, Apache ActiveMQ, MySQL connector.

Step#2:- Add Extra MQ Dependencies and camel-jdbc, spring-jdbc, mysqlversion/Oracle based.

pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.1.7.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.app</groupId>
	<artifactId>Apache-Camel-Integration-JDBC</artifactId>
```

```
<version>1.0</version>
<name>ApacheCamelIntegrationWithJDBC1</name>
<description>Camel Integration with JDBC App</description>

<properties>
	<java.version>1.8</java.version>
</properties>

<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-activemq</artifactId>
	</dependency>
	<dependency>
		<groupId>org.apache.camel</groupId>
		<artifactId>camel-spring-boot-starter</artifactId>
		<version>2.24.0</version>
	</dependency>
	<dependency>
		<groupId>org.apache.camel</groupId>
		<artifactId>camel-jdbc</artifactId>
		<version>2.24.0</version>
	</dependency>
	<dependency>
		<groupId>org.apache.camel</groupId>
		<artifactId>camel-jms</artifactId>
		<version>2.24.0</version>
	</dependency>
	<dependency>
		<groupId>org.apache.activemq</groupId>
		<artifactId>activemq-pool</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-jdbc</artifactId>
	</dependency>
```

```
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
                <optional>true</optional>
            </dependency>

            <dependency>
                <groupId>com.oracle</groupId>
                <artifactId>ojdbc6</artifactId>
                <version>11.2.0</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Step#3:- Starter class

```
package com.app;
import org.springframework.boot.SpringApplication;
import     org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ApacheCamelIntegrationWithJdbc1Application {
```

```
  public static void main(String[] args) {
        SpringApplication.run(ApacheCamelIntegrationWithJdbc1Application.class, args);
        System.out.println("Starter class Executed..:");
  }
}
```

Step#4:- Define one DataSource Object to connect with Database.

```
package com.app.config;
import    org.springframework.context.annotation.Bean;
import     org.springframework.context.annotation.Configuration;
import      org.springframework.jdbc.datasource.DriverManagerDataSource;

@Configuration
public class AppConfig {

        @Bean
        public DriverManagerDataSource dsObj() {
                DriverManagerDataSource d = new DriverManagerDataSource();
                d.setDriverClassName("oracle.jdbc.driver.OracleDriver");
                d.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
                d.setUsername("system");
                d.setPassword("system");
                return d;
        }
}
```

Step#5:- Define One RouteBuilder class with logic

```
package com.app.route;
import org.apache.camel.builder.RouteBuilder;
import org.springframework.stereotype.Component;

@Component
public class MyBuilder extends RouteBuilder {

        public void configure() throws Exception {
                from("timer:timer1?repeatCount=1&period=10s")
                .setBody(constant("select * from emptab"))
                .to("jdbc:dataSource")
```

```
                .process(ex-> {
                        Object ob=ex.getIn().getBody();
                        System.out.println(ob.toString());
                        ex.getOut().setBody(ob.toString());
                })
                .to("jms:queue:two");
        }
}
```

application.properties:--

```
camel.springboot.main-run-controller=true
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

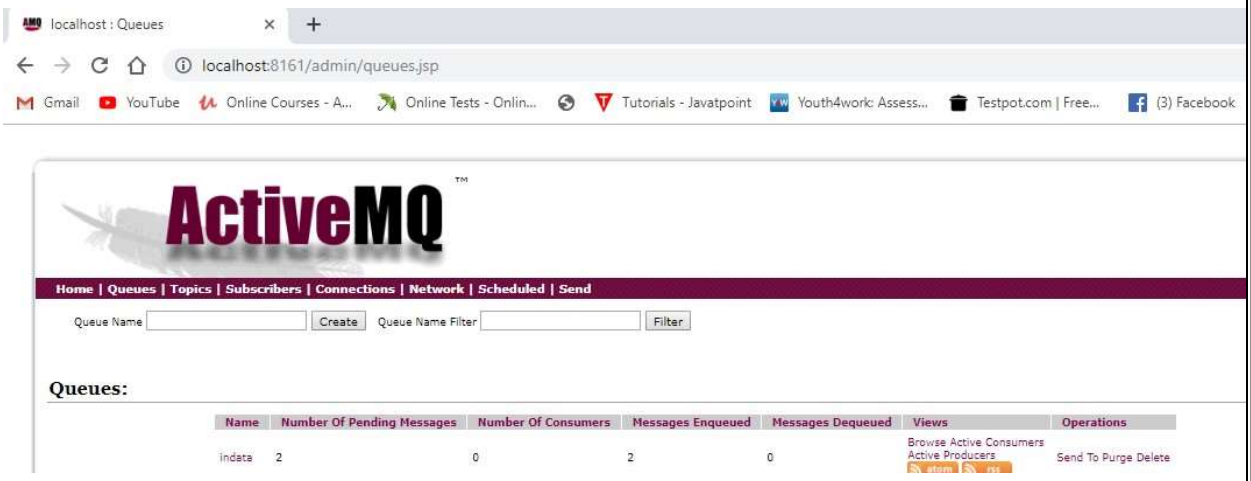
Execution Order:--

#1:-- Create table and insert data.

#2:-- Run ActiveMQ

#3:-- Run Starter class of Project.

#4:-- Goto ActiveMQ (http://localhost:8161/admin), Click on Queue to see the details.



NOTE:--

=>use protocol “timer://…” which takes input like repeatCount (no. of iterations) and period (time gap) in mill sec.

Example:--

```
from("timer:timer1?repeatCount=1&period=10s")
        .setBody(constant("select * from emptab"))
        .to("jdbc:dataSource")
```

Q>SQL V/s JDBC in Camel?

=>Sql Uses Iterator to fetch data from List<Map> and returns Object by Object where as JDBC gets data in List<Map> returns same to next step.

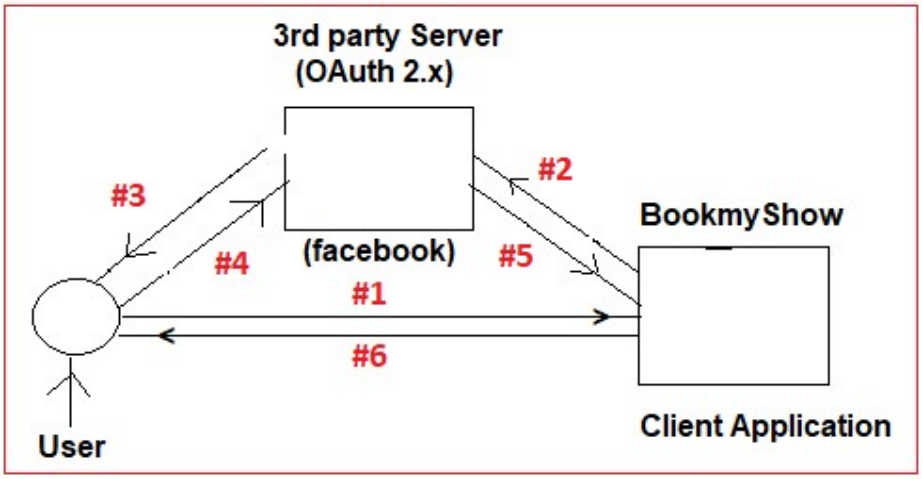
#1:-timer://anyName?period=1000
->It is repeated for every 1Sec
#2:-timer://anyName?period=10s
->It is repeated for every 10Sec
#3:-timer://anyName?repeatCount=4&period=10m
->It is executed for every 10 minutes and 4 times only

Task#1:-- Use sql: insert fetch data from MQ
Task#2:-- use jdbc: select read List<Map> convert to csv file format.

13. Open Authorization (OAuth 2.x):--

A & A = Authentication and Authorization
=>It is a process of secure one application user data. Here,
->Authentication means "IDENTITY OF USER", like username, password, otp …etc.
=>Authorization means "ROLE OF USER" what you can do? Like ADMIN ROLE, END USER, EMPLOYEE ROLE… (It is like permissions/grant).

OAuth 2.x:-- It is standard and framework which provides 3rd party security services to client application which are registered, on behalf of end user.



1.>Browser making request to client Application.
2.>Client asking permission to third party.
3.>Third party Application asking confirmation (Grant) to end user.
4.>User confirmation.
5.>Data shared from 3rd party Application to Client App.
6.>Client gives response to end user.

=>These 3rd party Applications are also called as **"Authorization and Resource Servers".**
->Authorization and Resource Server examples are:- Google, Facebook, Githb, Twitter, Linkedin … etc.
->Example client Applications that are using OAuth2 ares BookMyshow, redbus, yatra, makemytrip, avast, zomato.. etc.
=>OAuth 2.x standard is widely used in small/medium scale/daily used, business application.
=>OAuth2 Provide SSO (Single Sign on) for multiple applications acting as a one Service.

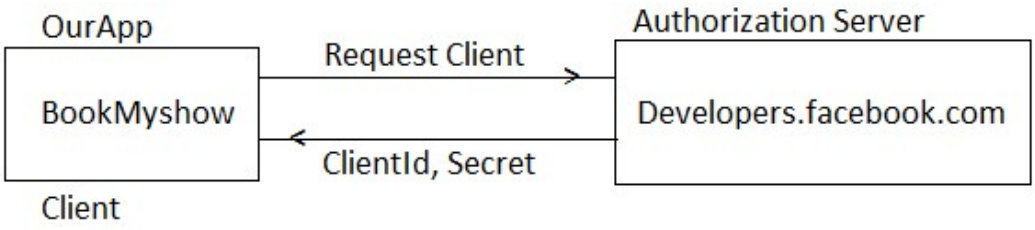
=>***OAuth2 may not be used for high level and Large scale applications (Ex:- Banking Creditcard, Stock Market, finance… etc). These Spring Security ORM is used mostly.

#1:-- Client Register with Authorization Server

Ex: BookMyShow---Register-with--->Facebook Server

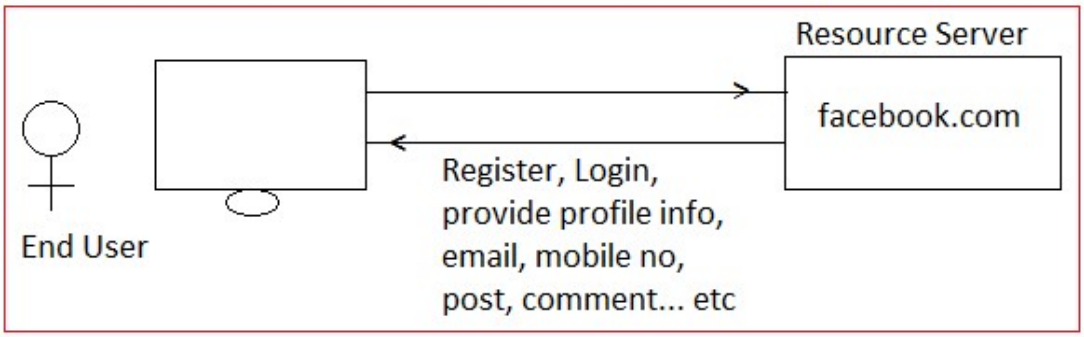
=>Here, every Client Application needs to be register with **AuthorizationServers** First.
=>Client Application gets clientId and clientSecret (like Password) on Successful register at AuthorizationServer.
=>This is like one time setup between Client Application and Authorization Server.

#1. Register Client Application with Authorization Server.

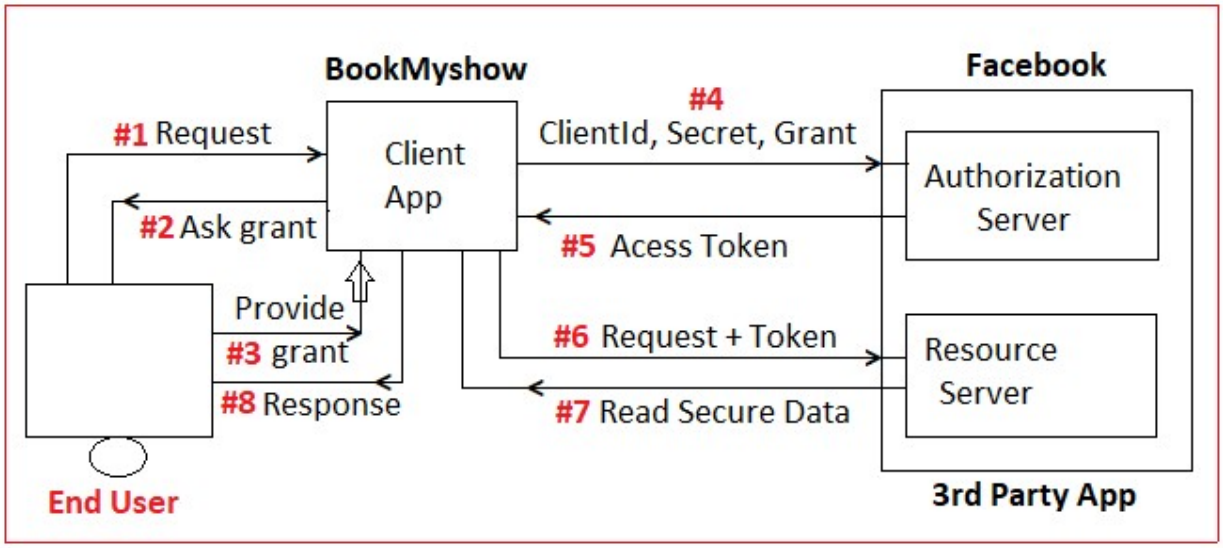


#2:- End User must be Register with Resource Server:--

=>User must be register and Login with 3rd party **Resource Servers** (Like facebook.com).
=>User need to provide profile information, basic data, comments, posts, photos… etc.
=>User Id and password will never be shared with Client Application. Only Users Public and General information is shared with Client Application.



Request Work flow of OAuth2:--



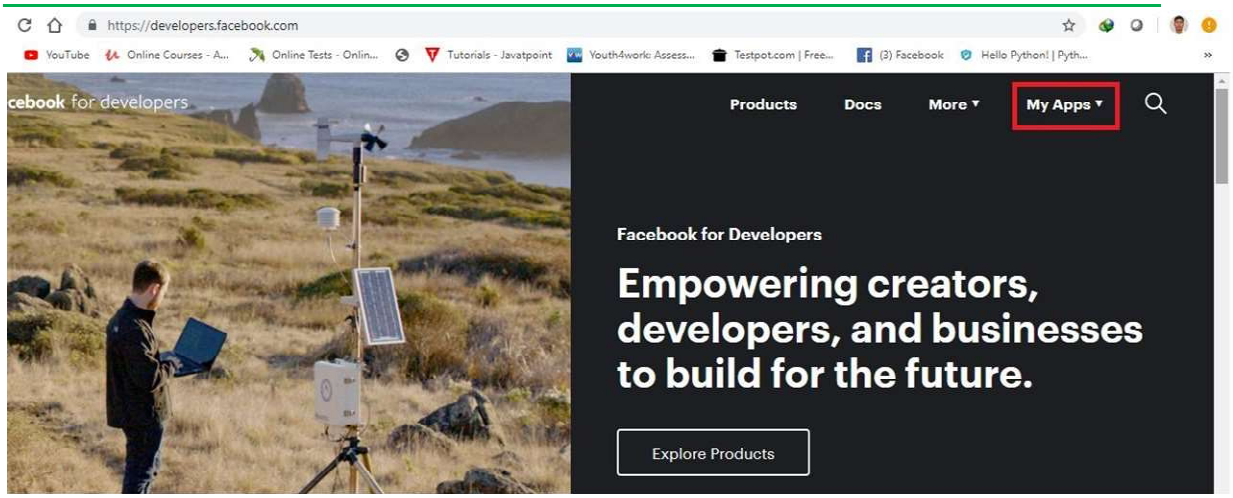
1. End user makes request using browser to client application (BookMyshow) request for "verify using 3rd party service" ex: Facebook, Google… etc.
2. Client Application will ask for Grant from end user, which confirms that access user data.
3. End user has to provide Grant (Permission) to access data.
4. Client makes request to Authorization server using ClientId, secret, user grant.
5. Auth server verifies details and goes to token Management process.
   =A unique number is generated, called as Token which works for User+Client combination.
6. Now, client application makes request to resource Server using Access Token.
7. Resource server returns end user secure data to client.
8. Finally, Client App process the end user request and gives response.

OneTime setup for OAuth2:--

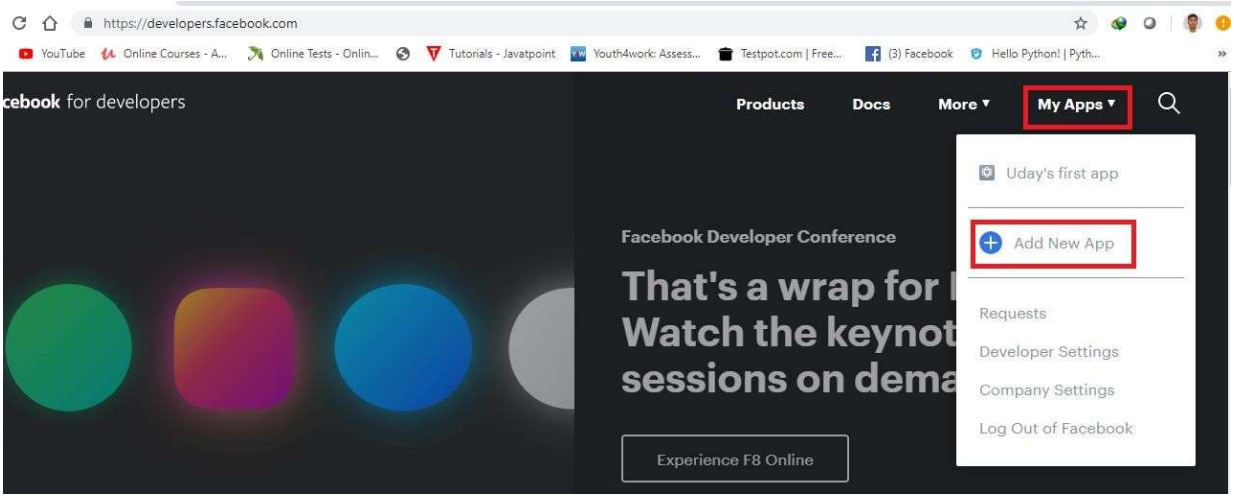
Step#1:- Choose any one (or more) 3rd party **"Authorization & Resource Server"**.
Ex:-- facebook, Google cloud platform (GCP) Github, Linkedin, Twitter… etc.

Step#2:- Here choosing Facebook as 3rd party server for open Authorization link is:
https://developers.facebook.com/

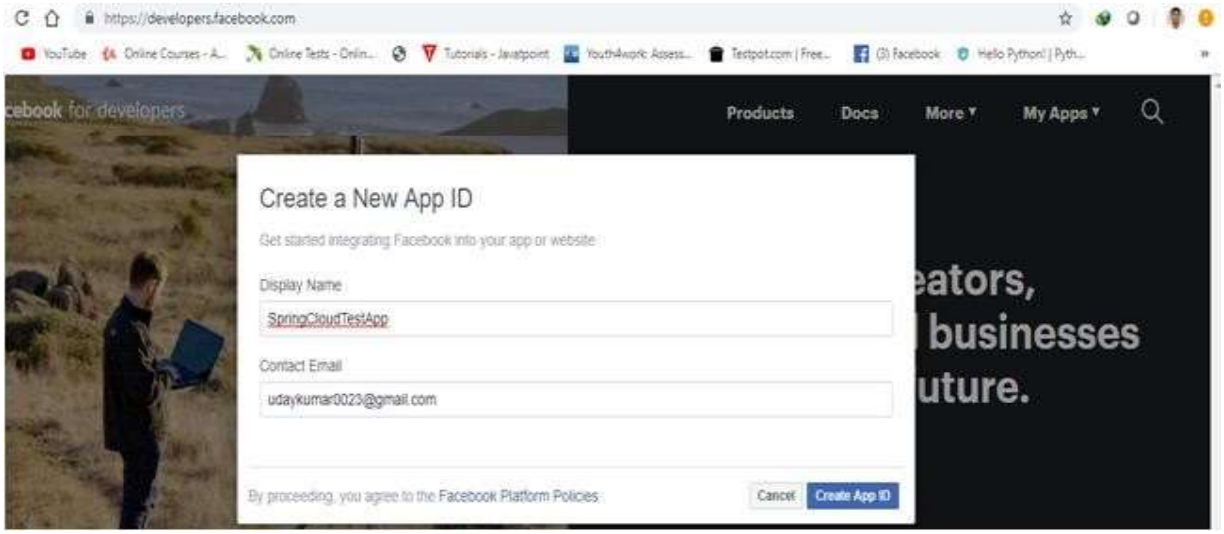
Step#3:- Define one new (client) Application in FB Authorization server which generates ClientId (AppId) and secrete (App Secret)
=>Goto facebook developer page
=>Click on top right corner "My Apps"



=>Choose “Add New App”.



=>Provide Display name (ex: SpringCloudTestApp) and email id : udaykumar0023@gmail.com.



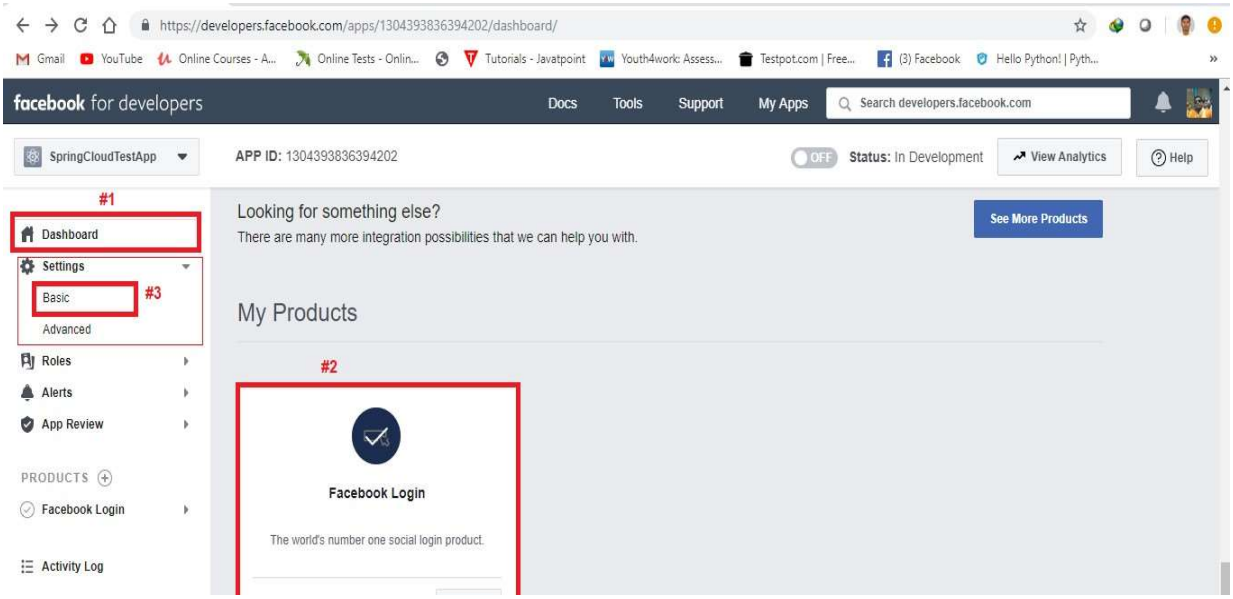
=>Click on “Create App ID”

=>Complete Security verification

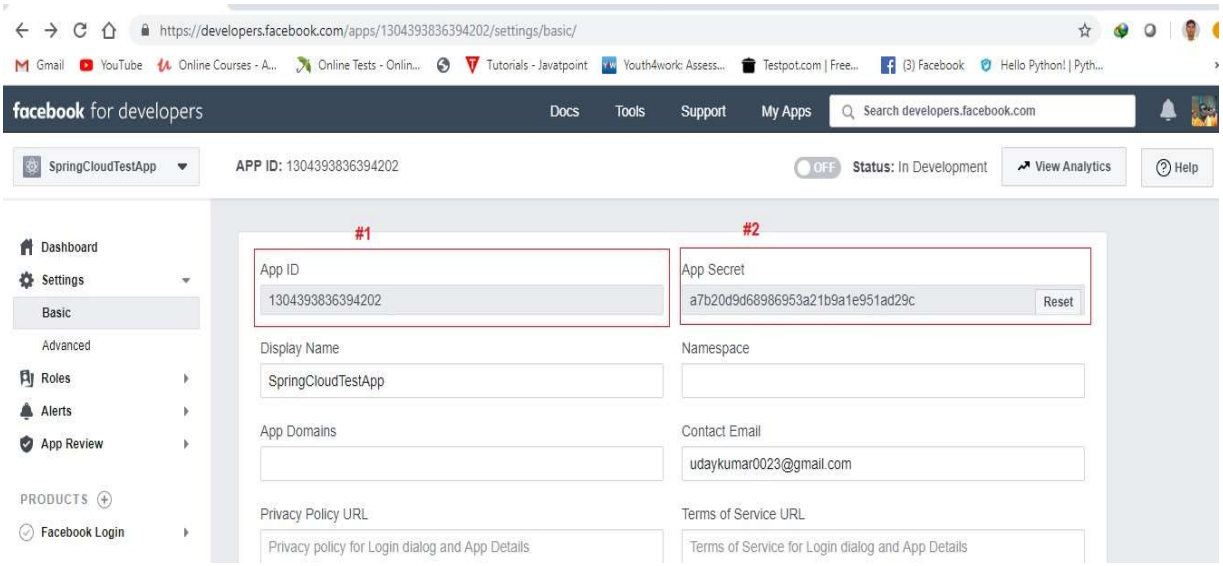
=>Click on “Dashboard”

=>Click on “facebook Login” setup button

=>Click on “Settings” >>Basic



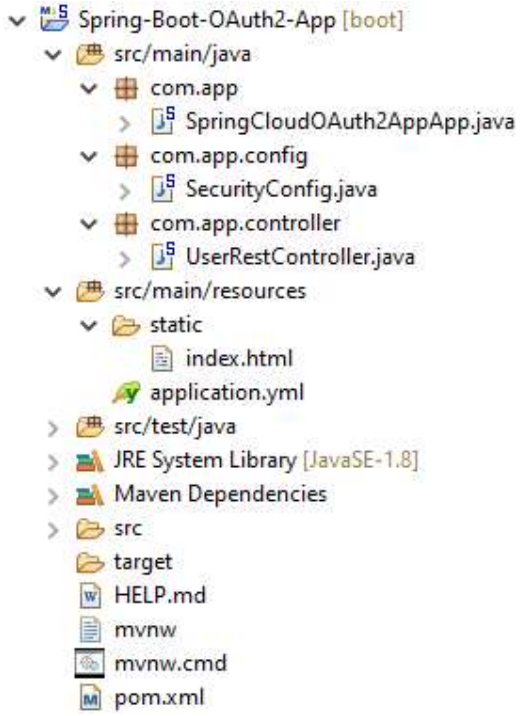
=>Basic AppId (ClientId) and App Secret (ClientSecret).



Step#4:- Create one SpringBoot app with dependencies “Security” & “Cloud OAuth2”.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
```

#23. Folder Structure of OAuth2 Application:--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
```

```
        <version>2.1.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.app</groupId>
    <artifactId>Spring-Cloud-OAuth2-App</artifactId>
    <version>1.0</version>
    <name>Spring-Cloud-OAuth2-App</name>
    <description>Spring Cloud OAuth2 Implementation</description>
    <properties>
        <java.version>1.8</java.version>
        <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-oauth2</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

```
        <dependencyManagement>
                <dependencies>
                        <dependency>
                                <groupId>org.springframework.cloud</groupId>
                                <artifactId>spring-cloud-dependencies</artifactId>
                                <version>${spring-cloud.version}</version>
                                <type>pom</type>
                                <scope>import</scope>
                        </dependency>
                </dependencies>
        </dependencyManagement>
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
</project>
```

Step#5:- Create application.properties / application.yml file using below key value pairs.

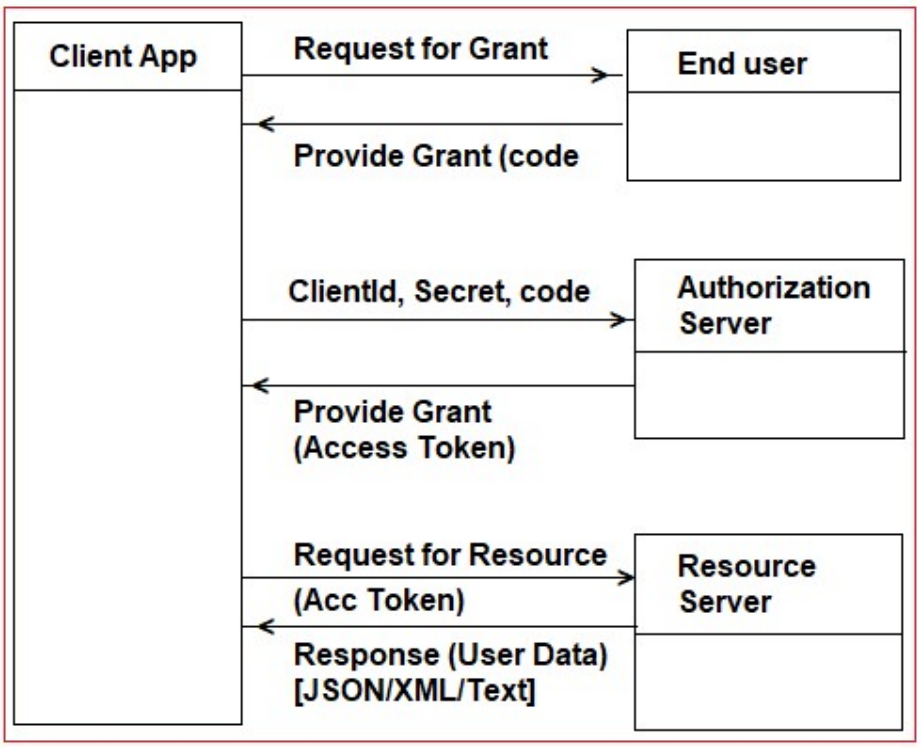
application.properties:--

```
server.port: 9898
#Auth Server details
security.oauth2.client.clientId:  1304393836394202
security.oauth2.client.clientSecret: a7b20d9d68986953a21b9a1e951ad29c
security.oauth2.client.accessTokenUri:
https://graph.facebook.com/oauth/access_token
security.oauth2.client.userAuthorizationUri: https://www.facebook.com/dialog/oauth
security.oauth2.client.tokenName: oauth_token
security.oauth2.client.authenticationScheme: query
security.oauth2.client.clientAuthenticationScheme: form
#Resource Server details
security.oauth2.resource.userInfoUri: https://graph.facebook.com/me
```

application.yml:--

```
server:
   port: 9898
security:
   oauth2:
#Auth Server details
      client:
         clientId : 1304393836394202
         clientSecret : a7b20d9d68986953a21b9a1e951ad29c
         accessTokenUri : https://graph.facebook.com/oauth/access_token
         userAuthorizationUri : https://www.facebook.com/dialog/oauth
         tokenName: oauth_token
         authenticationScheme: query
         clientAuthenticationScheme : form

#Resource Server details
      resource:
         userInfoUri: https://graph.facebook.com/me
```

Request Execution flow :--



NOTE:-- AppInit is provided by Boot, no need to write.

Components Involved : ClientApp, User(Browser), Auth Server (with Token Generator and Resource Server (User data in XML/JSON).

Step#6:- Define SecurityConfig class (SecurityInit is not required, Handled by spring Boot only).
=>Here Authentication Details not required configuring as we are using 3rd party security services.
=>In Authorization config specify which URLs needs type **"Every One Access"** [PermitAll].

SecurityConfig.java:--

```
package com.app.config;
import org.springframework.boot.autoconfigure.security.oauth2.client.
EnableOAuth2Sso;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
WebSecurityConfigurerAdapter;

@Configuration
@EnableOAuth2Sso
public class SecurityConfig extends WebSecurityConfigurerAdapter
{
	protected void configure(HttpSecurity http) throws Exception {
		http.authorizeRequests()
		.antMatchers("/","/login").permitAll()
		.anyRequest().authenticated();
	}
}
```

Step#7:- Define RestController for User (or Any)

```
package com.app.controller;
import java.security.Principal;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class UserRestController {

        @RequestMapping("/user")
        public Principal showUser(Principal p)
        {
            return p;
        }
        @RequestMapping("/home")
          public String showData()
          {
               return "Hello";
        }
}
```

Step#8:- Define one UI Page
ex:-- index.html under src/main/resource, in static folder

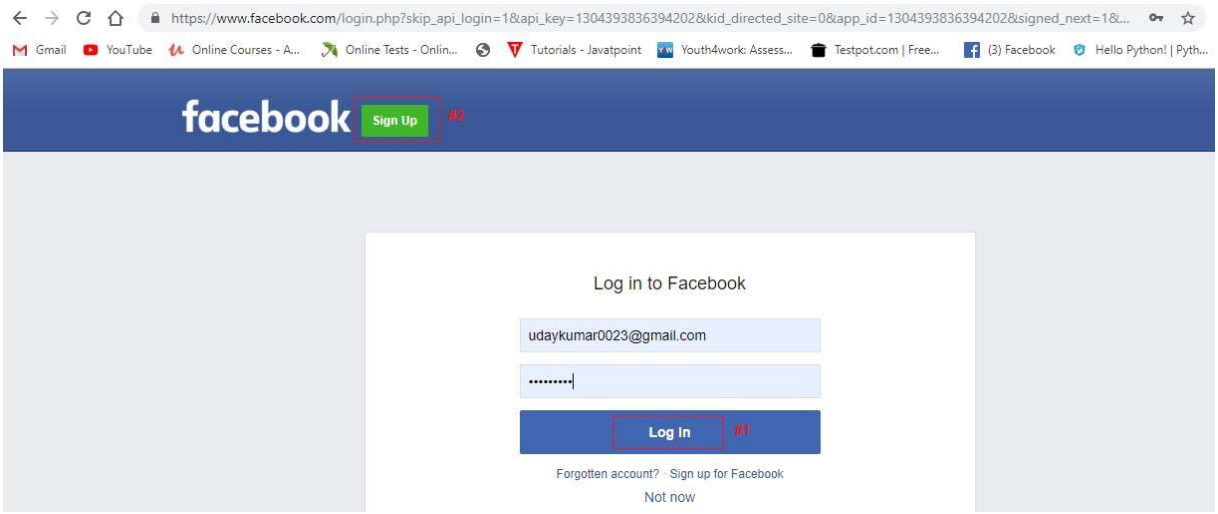
Index.html:--

```
<html>
<body>
<h1>Welcome to Login Page</h1>
<a href="user">Facebook</a>
<hr/>
<form action="#" method="post">
User :  <input type="text">
Pwd : <input type="password">
<input type="submit">
</form>
</body>
</html>
```

Step#9:- Run application and Enter URL http://localhost:9898

Step#10:- Click on facebook (#1 Process) Link and accept name (or) enter details or Else follow Second Process.

=>Click on **Login** if you have Facebook account else Click on **Sign Up** Button.



Task :-- Google Cloud Console

=>Create app and get ClientId, Secret.

14. Pivotal Cloud Foundry (PCF) :--

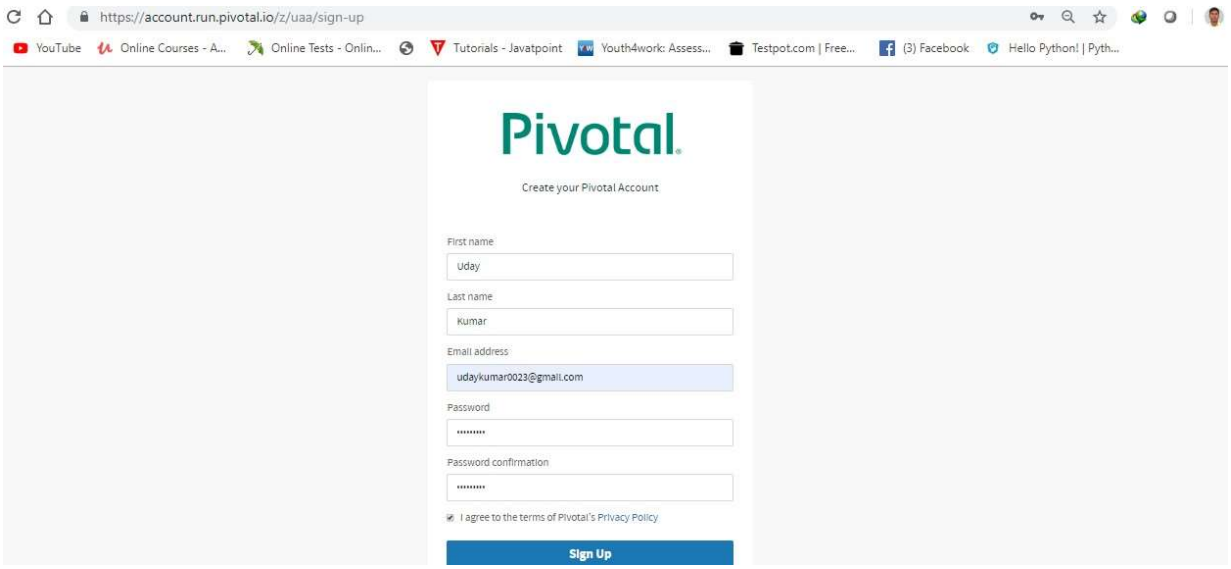
Spring Boot PCF deployment:--

Pivotal Cloud Foundry is a Cloud Deployment Server provides service to Spring Boot and Microservices Applications mainly with all kinds of Environment setup like Databases, server, log tracers etc.

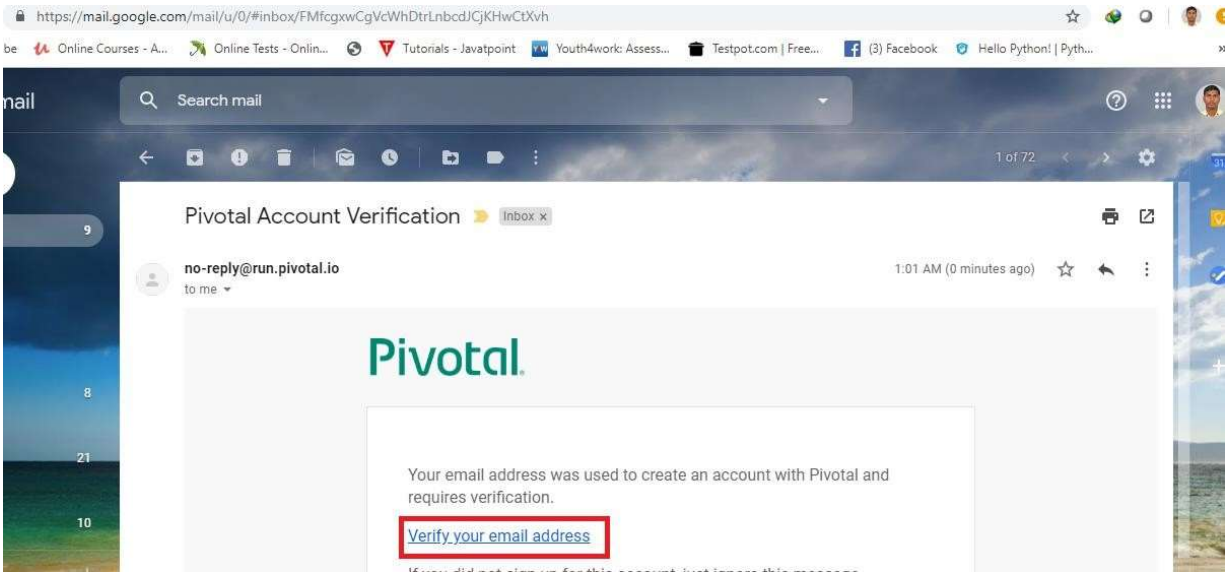
#PCF Account Setup#

Step#1:- Goto https://login.run.pivotal.io/login And click on **"Create Account".**

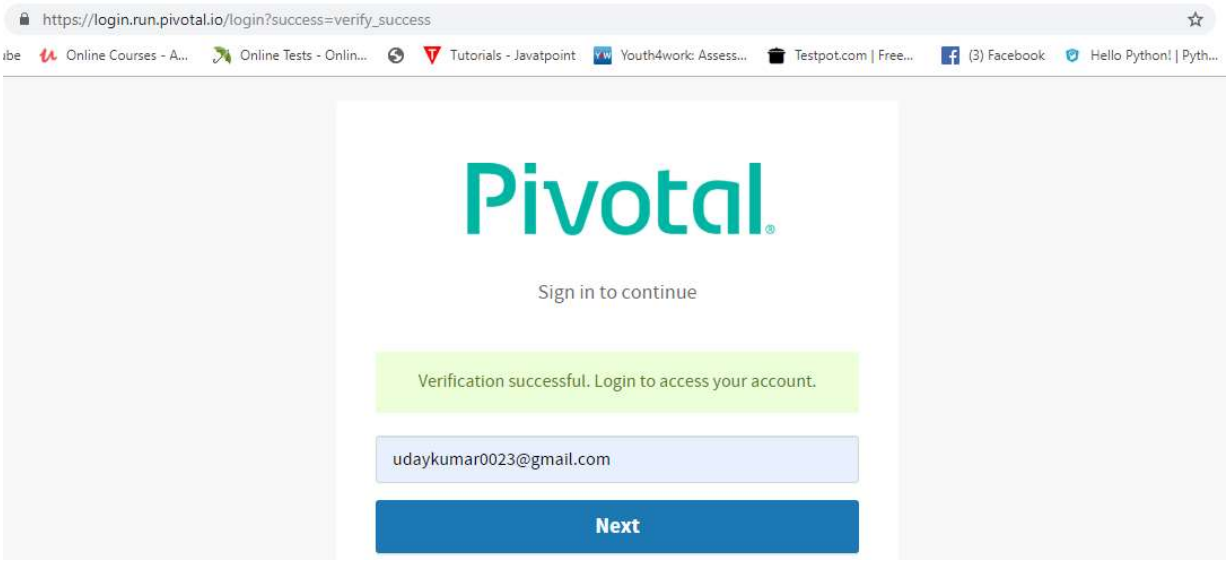
Step#2:- Enter details like name, email, password and conform password & Click on **Sign Up.**



Step#3:- Goto Email Account and verify PCF link

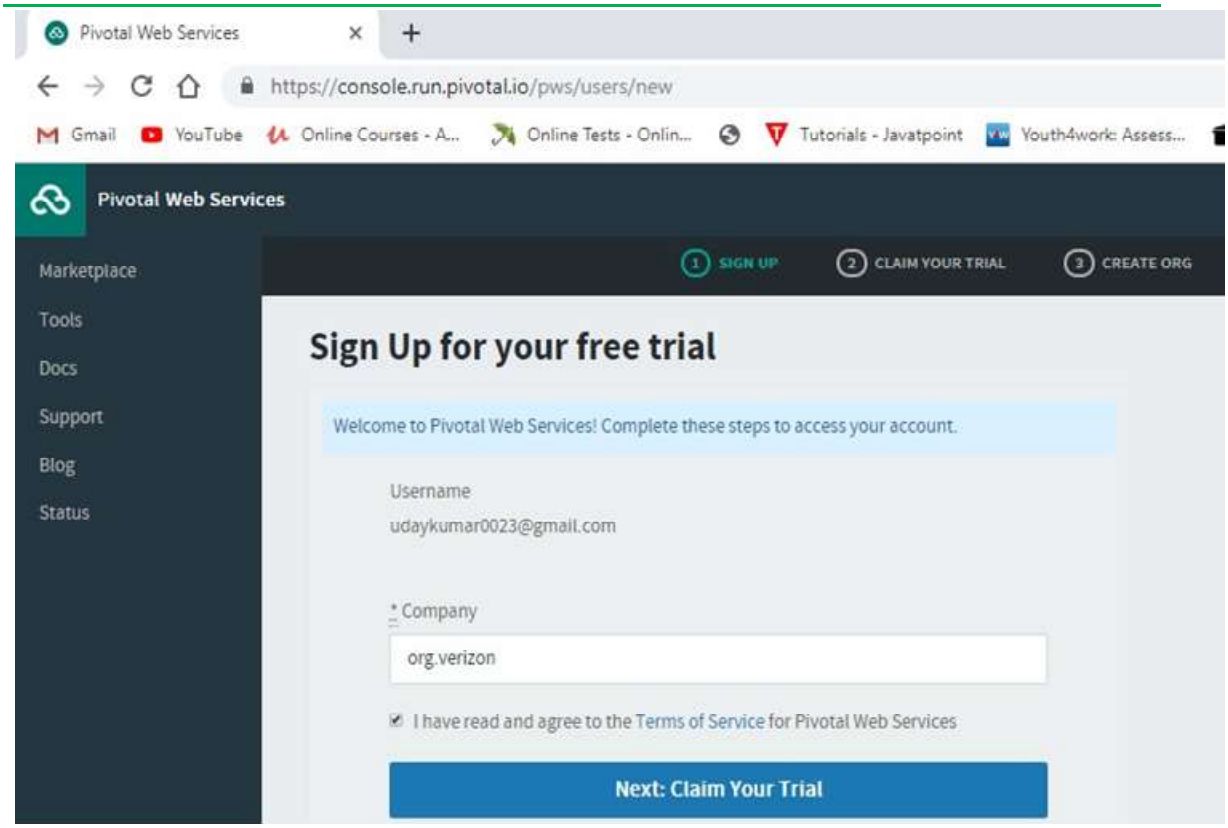


Step#4:- Login to PCF account (Above URL)

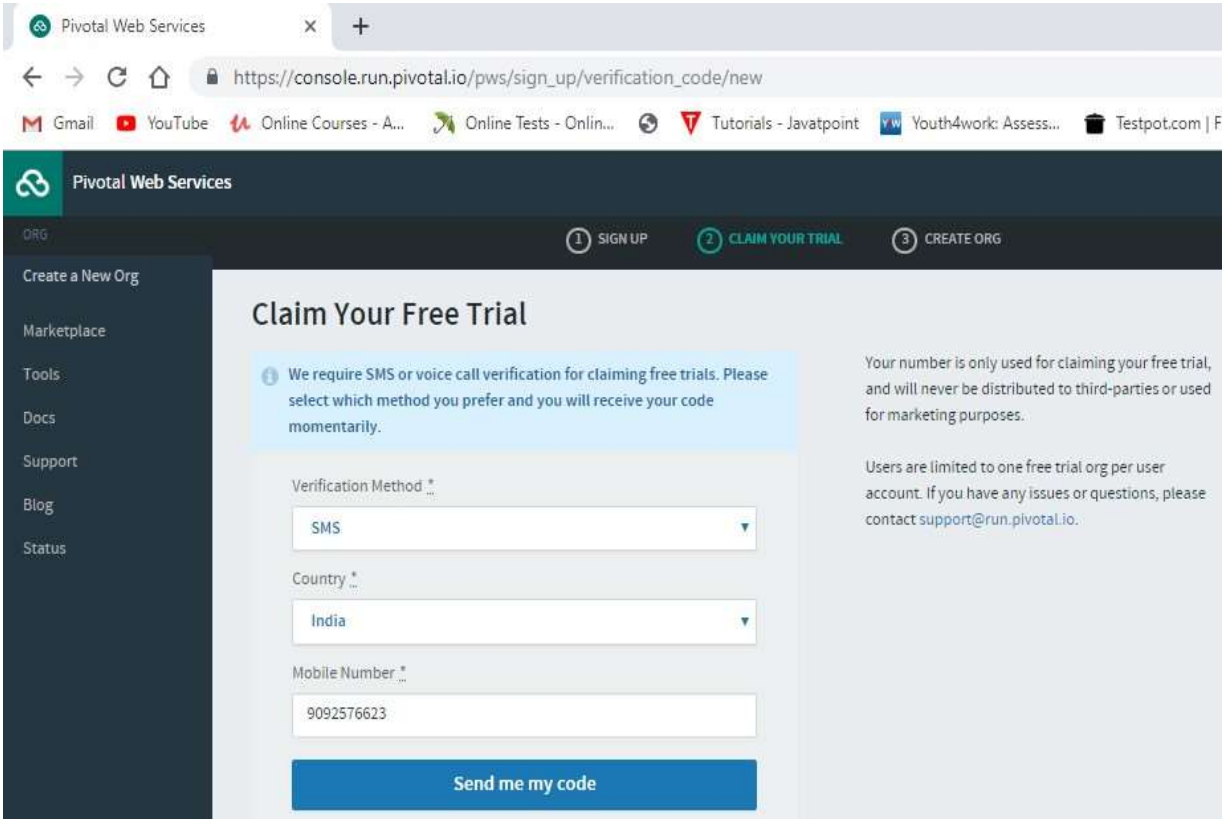


=>After login Click on **"pivotal web service"**

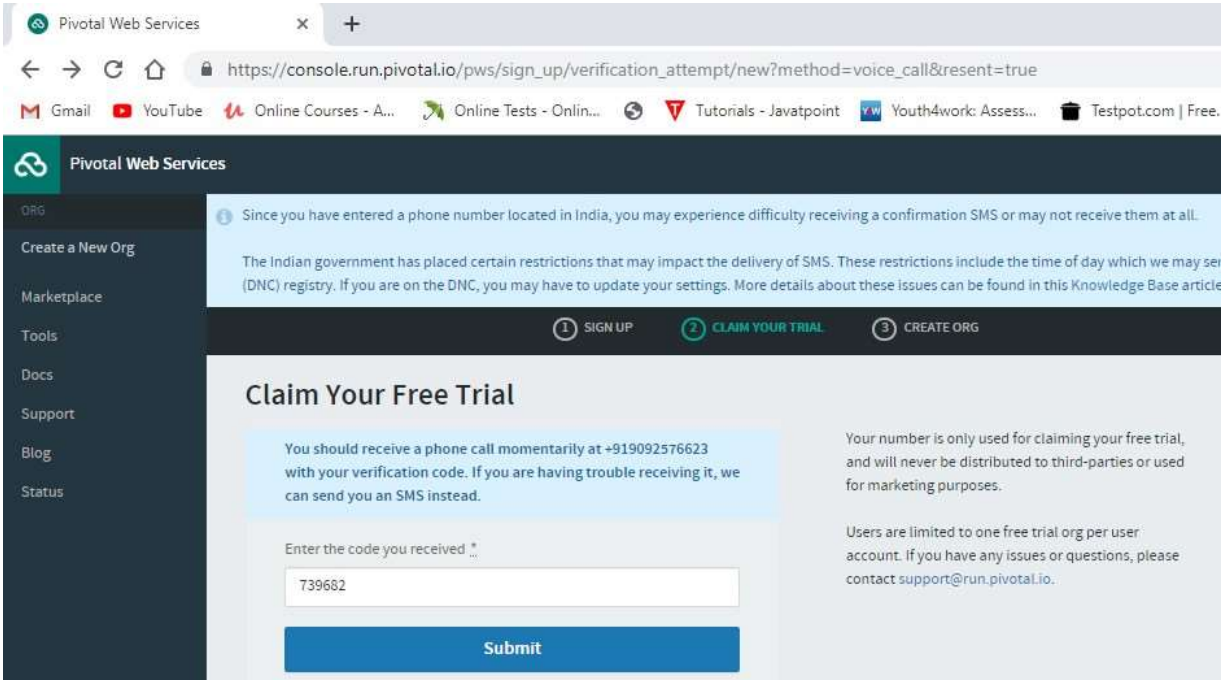
Step#5:- Provide initial details like
=>Company name



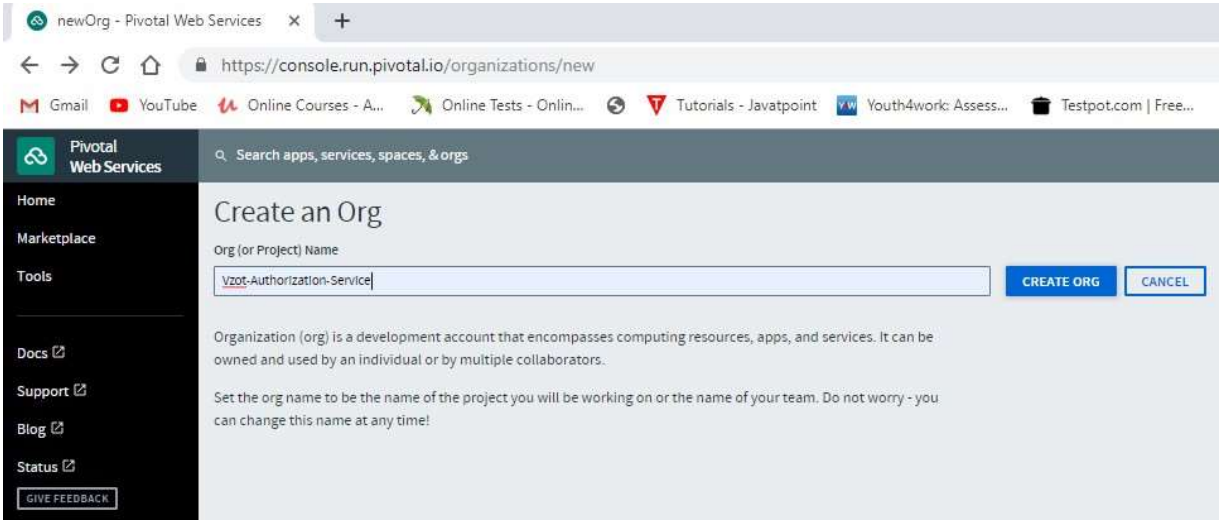
=>Enter Mobile number and verify



=>Enter OTP which is send on give Mobile



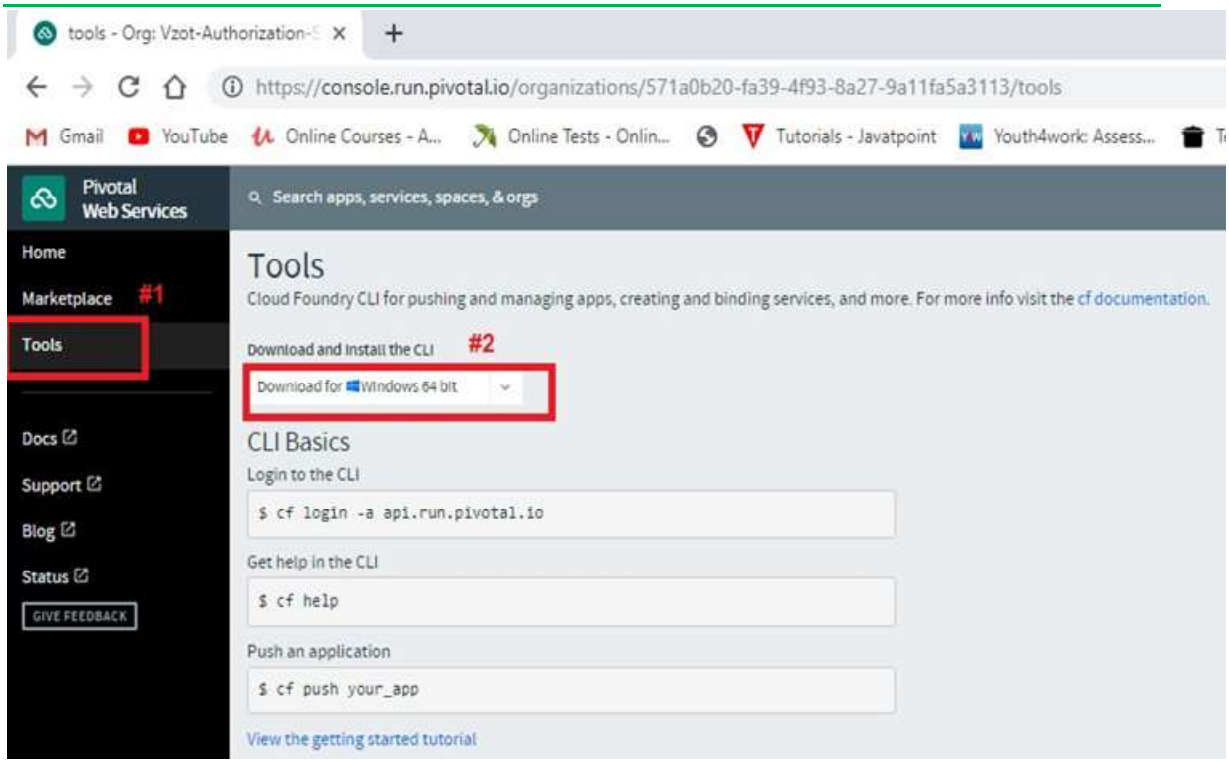
=>Organization (org) name > finish



Step#6:- Download and setup PCF CLI

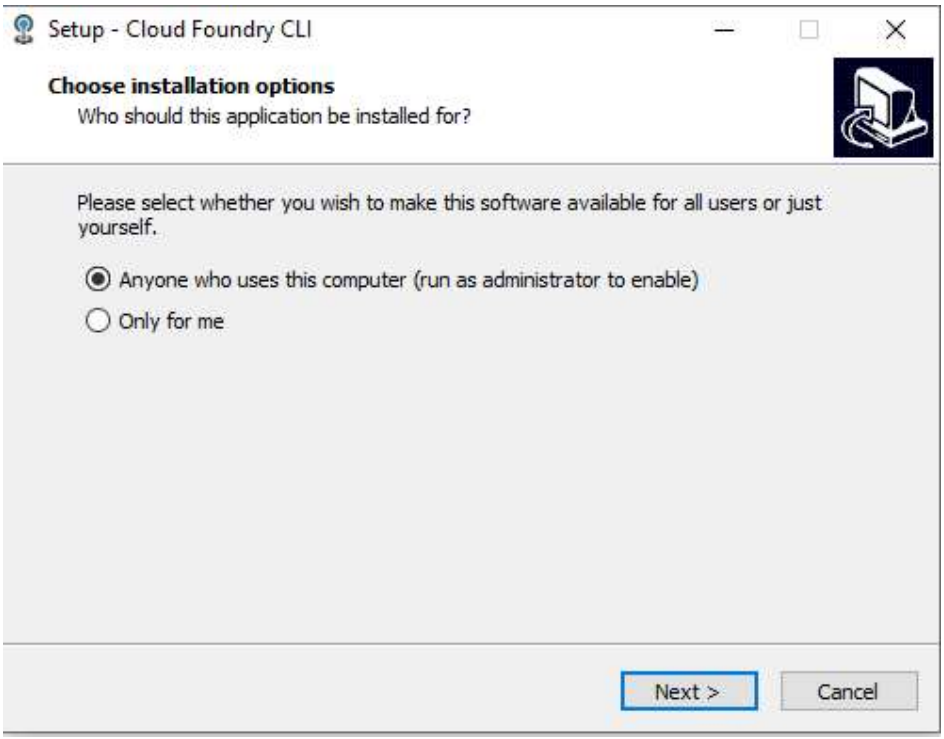
->Click on “Tools” option

->Choose OS and Bit and Download Software

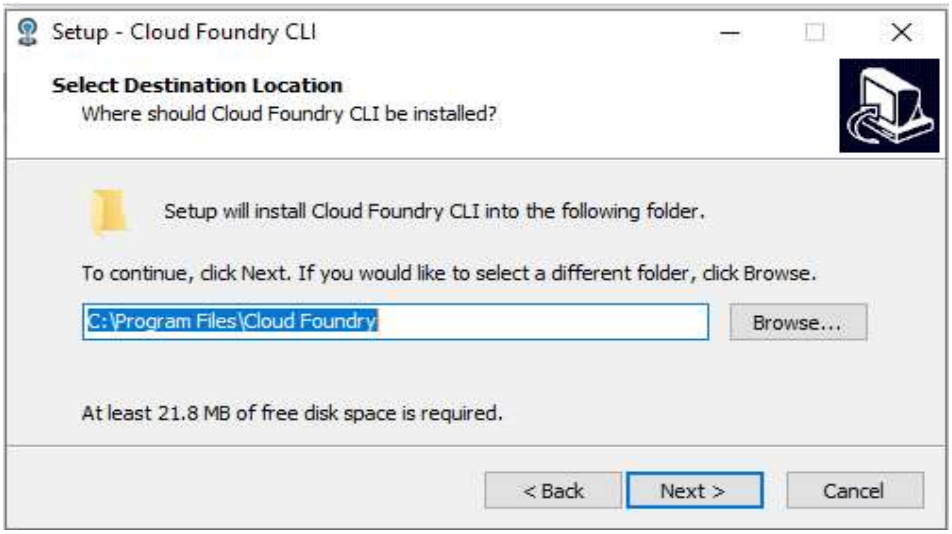


->Extract ZIP into Folder

->Click on “cf_installer.exe”.



=>Click on Next



=>>next > next > Finish

Step#7:- Login and Logout Commands

=>Goto cmd prompt

1>Login command is

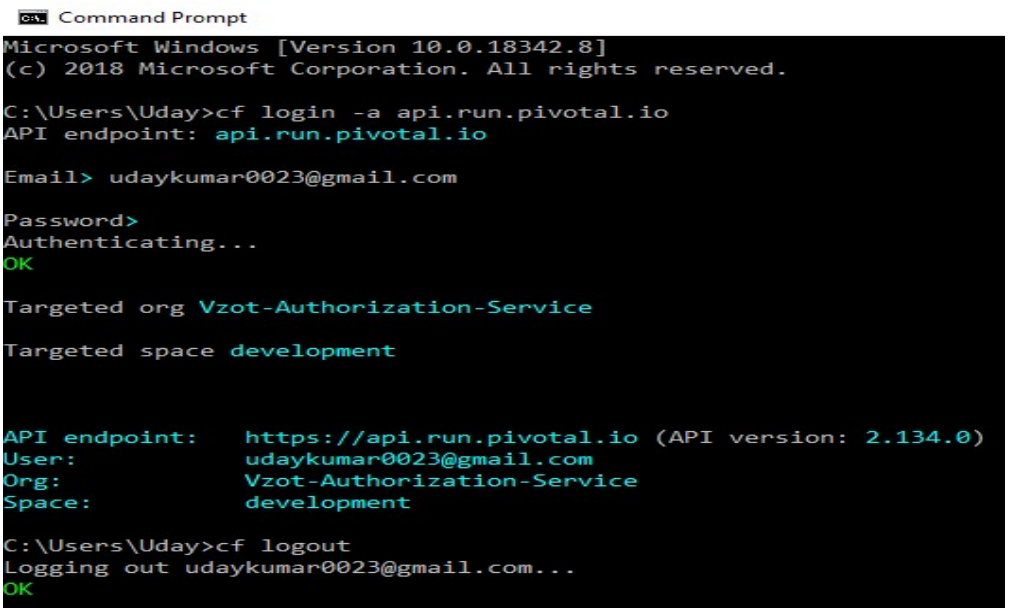
cf login –a api.run.pivotal.io

Email >

Password >

2>Logout command is

cf logout

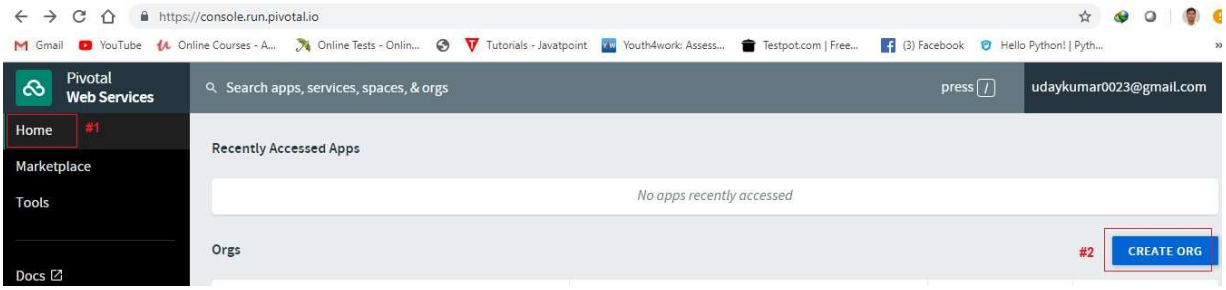


=>Spring Boot WEB MVC + Thymeleaf + Data JPA + H2 (Curd Operation) + Bootstrap

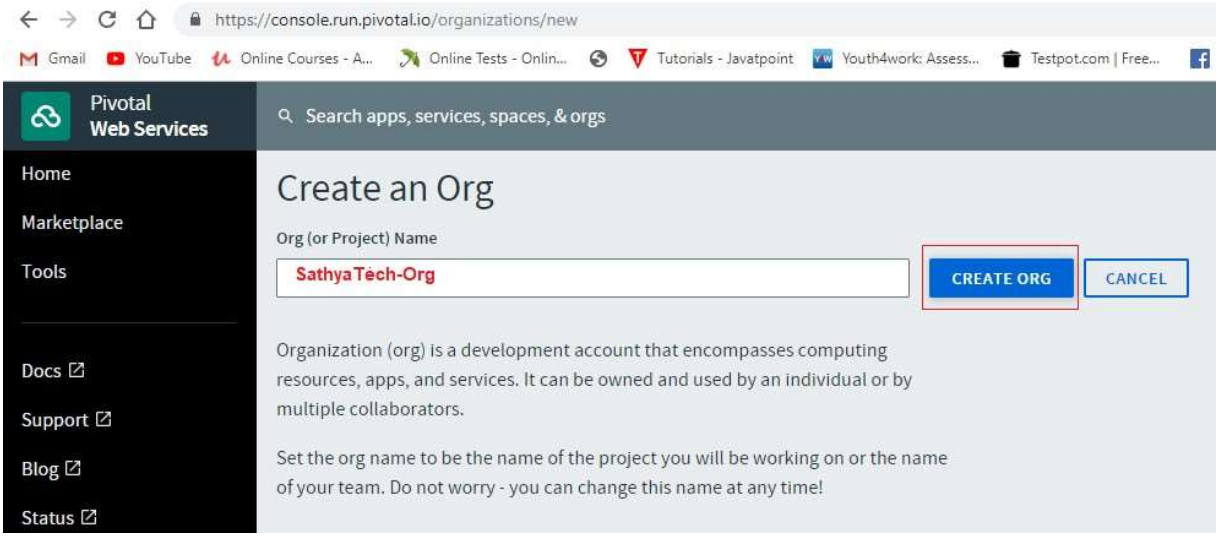
Step#8:- Create Org and space in PCF

=>Login to PCF using browser

=>Click on home > Click on **"CREATE ORG"** button



=>Enter org name ex : App-org



=>Click on **"CREATE ORG"**.

** An org "sample-org" is created with default space (workspace) "development" is created.

->A space (workspace) is a folder or location where project is stored and running.

Step#9:- Create one Spring Boot Starter Project in STS.

=>File > new > Spring Starter Project

=>Enter Details > choose Dependencies (Ex: web)

=>Finish.

Step#10:- Write Code for RestControllers, Services, Dao, and properties/yml files etc.

Step#11:- Clean Application
=>Right click > Run As > Maven Clean
…Wait for status BUILD SUCCESS…
=>This step will clear all old folders and files which are in “project-name/target” folder.

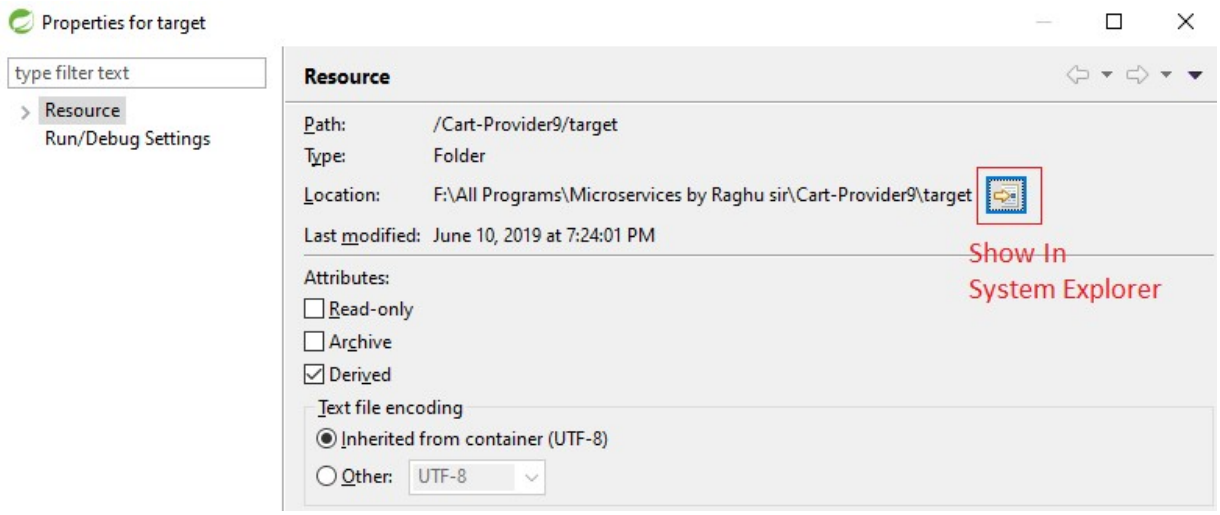
Step#12:- Do setup of JDK in workspace
>Windows > Preferences
>search with “Installed JRE”
>Click on “Add” > browse for location
Ex: “C:\Program Files\Java\jdk1.8.0_201”
>Choose new and delete old JRE
>Apply and close

Step#13:- Update JRE to Project
=>Right click on Project > build path
=>Configure build path > Choose JRE System Lib.
=>Edit > select workspace JRE/JDK
=>Apply and close.

Step#14:- Install Application (build project)
=>Right click > Run As > Maven install
..wait for few minutes, still BUILD SUCCESS..

**(If failed, then Update Maven Project, select force Update and finish.
Repeat step#11, 13 and then 14)

Step#15:- Refresh Project to see updates in “target” folder.
=>Right click on Project > refresh (F5)
.. then..
=>Right click on target folder
=>Properties (or Alt + Enter)
=>click on “Show In System Explorer”



=>Open target folder > find --.jar file
Ex:-- (SpringBootSampleApp-1.0.jar)

Step#16:- Open commandpromt and login to PCF from here
=>shift + mouse right click
=>choose “open cmd window here”
=>Login to PCF

1>cmd/>cf login –a api.run.pivotal.io (F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf login -a api.run.pivotal.io)
=>Enter Email Id and password (which is used for login pcf account)

2>Select ORG in pivotal cloud (Where Vzot-Authoriztion-Service is a Org in PCF account)
Ex:-- cf target -o Vzot-Authorization-Service (F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf target -o Vzot-Authorization-Service)

3>Push Application to created org & space

Cmd/>cf push [Project-name] –p [JARNAME].jar
Syntax:-- cf push app –p SpringBootSampleApp-1.0.jar
Ex:-- cmd>F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf push Cart-Provider9 -p Cart-Provider9-1.0.jar

.. wait for 5 minutes to upload project..

=>Screen for Process 1-3 command (Login-Push Application)

```
F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf login -a api.run.pivotal.io
API endpoint: api.run.pivotal.io

Email> udaykumar0023@gmail.com

Password>
Authenticating...
OK

Select an org (or press enter to skip):
1. SathyaTech-org
2. Vzot-Authorization-Service
```

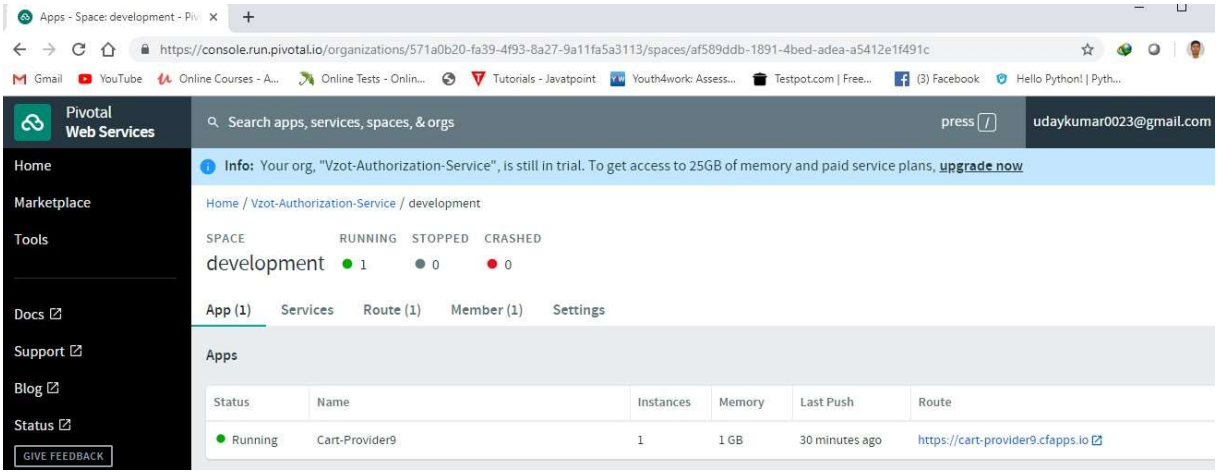
```
F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf target -o Vzot-Authorization-Service
api endpoint:   https://api.run.pivotal.io
api version:    2.136.0
user:           udaykumar0023@gmail.com
org:            Vzot-Authorization-Service
space:          development

F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target>cf push Cart-Provider9 -p Cart-Provider9-1.0.jar
Pushing app Cart-Provider9 to org Vzot-Authorization-Service / space development as udaykumar0023@gmail.com...
Getting app info...
Updating app with these attributes...
  name:               Cart-Provider9
  path:               F:\All Programs\Microservices by Raghu sir\Cart-Provider9\target\Cart-Provider9-1.0.jar
  command:            JAVA_OPTS="-agentpath:$PWD/.java-buildpack/open_jdk_jre/bin/jvmkill-1.16.0_RELEASE=printHeapHist
essorCount=$(nproc) -Djava.ext.dirs=$PWD/.java-buildpack/container_security_provider:$PWD/.java-buildpack/open_jdk_jre/
uildpack/java_security/java.security $JAVA_OPTS" && CALCULATED_MEMORY=$($PWD/.java-buildpack/open_jdk_jre/bin/java-buil
ry=$MEMORY_LIMIT -loadedClasses=18393 -poolType=metaspace -stackThreads=250 -vmOptions="$JAVA_OPTS") && echo JVM Memory
="$JAVA_OPTS $CALCULATED_MEMORY" && MALLOC_ARENA_MAX=2 SERVER_PORT=$PORT eval exec $PWD/.java-buildpack/open_jdk_jre/bi
k.boot.loader.JarLauncher
  disk quota:         1G
  health check type:  port
  instances:          1
  memory:             1G
  stack:              cflinuxfs3
  routes:
    cart-provider9.cfapps.io

Updating app Cart-Provider9...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...
```

```
Waiting for app to start...

name:              Cart-Provider9
requested state:   started
routes:            cart-provider9.cfapps.io
last uploaded:     Mon 10 Jun 23:32:19 IST 2019
stack:             cflinuxfs3
buildpacks:        client-certificate-mapper=1.8.0_RELEASE container-security-provider=1.16.0_RELEASE
                   java-buildpack=     -offline-https://github.com/cloudfoundry/java-buildpack.git#3f4eee2
                   jvmkill-agent=1.16.0_RELEASE open-jdk-...

type:              web
instances:         1/1
memory usage:      1024M
start command:     JAVA_OPTS="-agentpath:$PWD/.java-buildpack/open_jdk_jre/bin/jvmkill-1.16.0_RELEASE=printHe
                   -XX:ActiveProcessorCount=$(nproc) -Djava.ext.dirs=$PWD/.java-buildpack/container_security_
                   -Djava.security.properties=$PWD/.java-buildpack/java_security/java.security $JAVA_OPTS" &&
                   CALCULATED_MEMORY=$($PWD/.java-buildpack/open_jdk_jre/bin/java-buildpack-memory-calculator
                   -loadedClasses=18393 -poolType=metaspace -stackThreads=250 -vmOptions="$JAVA_OPTS") && ech
                   JAVA_OPTS="$JAVA_OPTS $CALCULATED_MEMORY" && MALLOC_ARENA_MAX=2 SERVER_PORT=$PORT eval exe

                   $PWD/. org.springframework.boot.loader.JarLauncher
     state     since                  cpu      memory          disk           details
#0   running   2019-06-10T18:02:42Z   167.5%   189.9M of 1G    147.5M of 1G
```

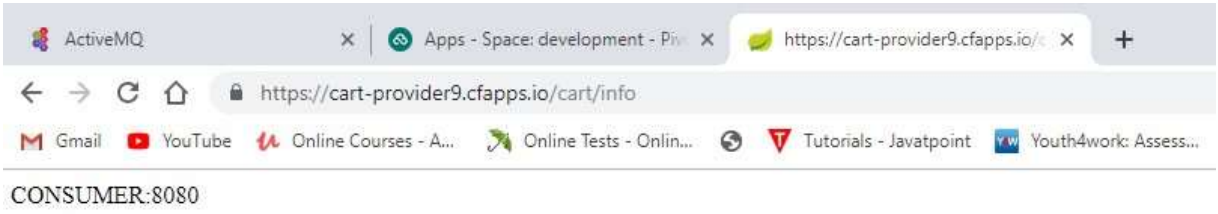
Step#17:- Once success then goto PCF WebConsole (browser) and click on SPACE "development"

=>To see total apps and running app.

=>Click on Route (URL) to execute app



=>Enter Paths to URL ex: /cart/info …



Step#18:- Logout PCF from cmd prompt

Cmd/> cf logout

*** BUILD FAUL at CLEAN

Then Force Update maven Project

=>Project > alt + F5 > choose Force update

=>Apply and close [finish]

*** BUILD FAIL at INSTALL

Then update JDK new version, in place of JRE

*** showing Cached Errors

=>Goto .m2 folder and delete that folder

=>come back to project > force update project

=>Create Project with MySQL Db with CURD operations and upload to PCF

=>Goto Market place and choose mysql provider enter details

Instance name : myappsql

Choose or/space:

>choose plan > finish

=>Come to Home >Click on org > space > service

=>click on MySQl services > Bind App >

=>select Our Project > bind > re-start app

=>Enter URL in browser and execute.

Note:-- Every log line contains four fields:

- Timestamp
- Log type
- Channel
- Message

15. Spring Boot Actuator:--

It provides **"Production Read Endpoints"** which are used to give extra information of Production Server.

=>In simple readymade service used for production server to **"Track and Trace Extra information"** from server.

=>It must be enabled by Programmer and used by Admin/Production Team.

Production Server:-- A server having application that provide service to end user is known as **Production server**.

=>In case of production, we need some times information of it like,

a>DB Details

b>Beans (Objects) Created

c>Memory (Head, ThreadDump, …etc) details.

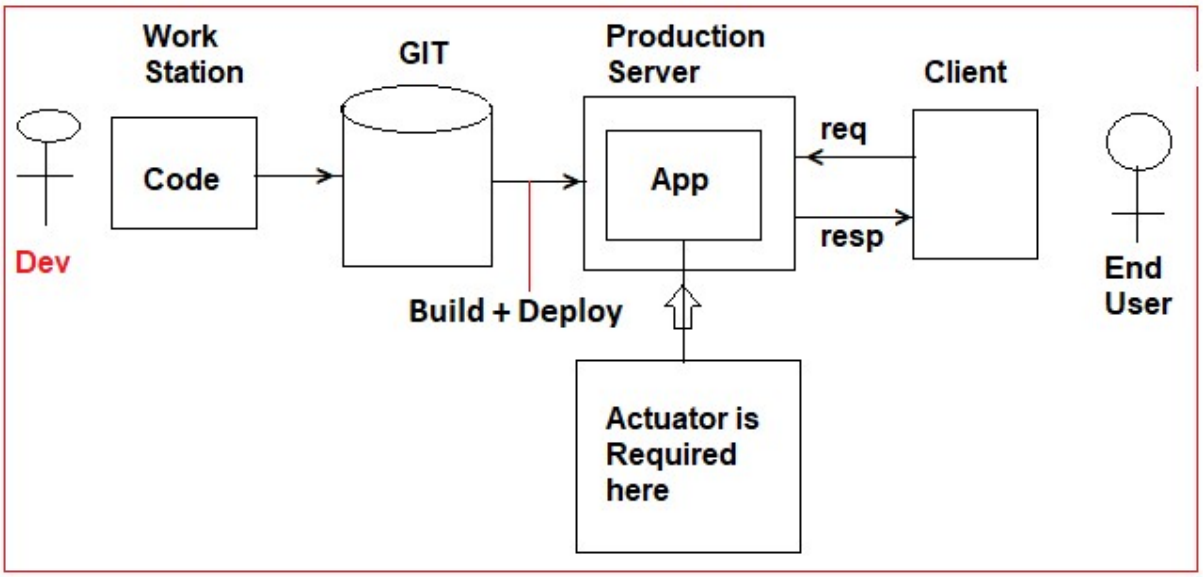
d>Cache Managements…

e>Is app Connected to any Remote networks or tools (Printer, Scanner…)

f>Logfiles and Log Level updates messages etc…

Actuator Dependencies:--

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```



=>By default Actuator enabled 2 endpoints those are :

1>/actuator/health

2>/actuator/info

=>To enable all endpoints use code **management.endpoints.web.exposure.include=*** **[**In application.properties.]

Coding Step:--

Step#1:- Create one starter project with one web and Actuator dependencies.

Step#2:- pom.xml:--

spring-boot-starter-web

spring-boot-starter-actuator

Step#3:- Define one Rest controller

Step#4:- application.properties

server.port=9988

management.endpoints.web.exposure.include=*

Step#5:- Run application and enter URL

1>http:localhost:9988/actuator/info

2>http:localhost:9988/actuator/health

3>http:localhost:9988/actuator/beans

4>http:localhost:9988/actuator/env

Etc. (few more: /httptrace, /threaddump, /caches)

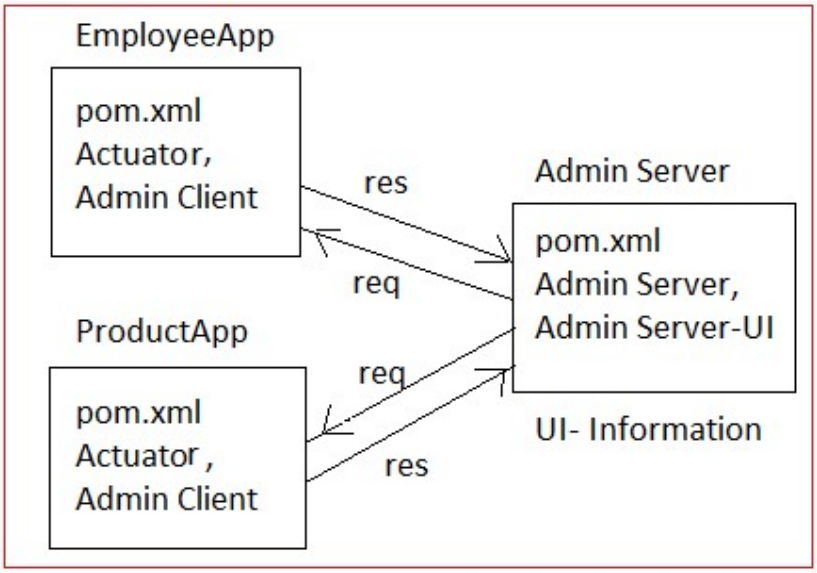
16. Spring Admin UI (Codecentric):--

Only Actuator if we implement in MicroServices then all details we should check manually. These all are made **"ADMIN UI- Track and Trace"** using Spring Boot Admin UI application.

=>Actuator provides all endpoints we need to enter URL using Http client (browser/POSTMAN) manually, which is time consuming process and big task to admin team.

=>To avoid this manual approach use admin setup for actuators, which is provided by 3rd party service **"de.codecentric"**?

=>For this every Microservices should have **actuator** and **admin client** dependencies and one Application with **admin server** and **admin-server-ui** dependencies.

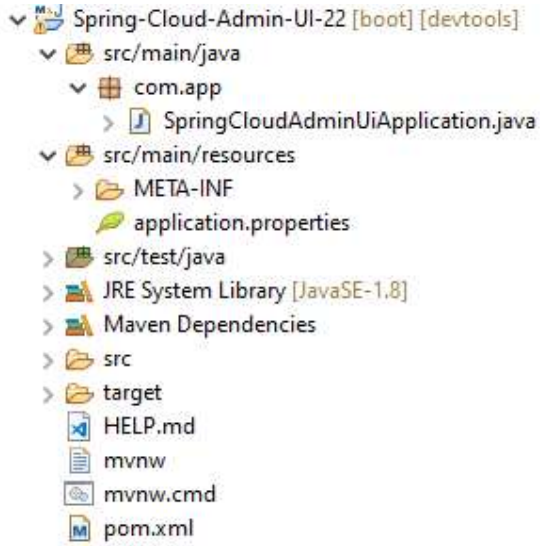


Spring boot Admin Server setup:--

Step#1:- Create Spring Boot Starter app with dependencies : admin server and ui.

```
<dependency>
        <groupId>de.codecentric</groupId>
        <artifactId>spring-boot-admin-starter-server</artifactId>
</dependency>
<dependency>
        <groupId>de.codecentric</groupId>
        <artifactId>spring-boot-admin-server-ui</artifactId>
</dependency>
```

#24. Folder Structure of Admin UI Dashboad:--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.1.1.RELEASE</version>
                <relativePath /> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.app</groupId>
        <artifactId>Spring-Cloud-Admin-UI</artifactId>
        <version>1.0</version>
        <name>Spring-Cloud-Admin-UI</name>
        <description>Spring Cloud Admin Dashboad Implementation</description>
       <properties>
                <java.version>1.8</java.version>
                <spring-boot-admin.version>2.1.4</spring-boot-admin.version>
        </properties>
        <dependencies>
```

```
        <dependency>
            <groupId>de.codecentric</groupId>
            <artifactId>spring-boot-admin-starter-server</artifactId>
        </dependency>
        <dependency>
            <groupId>de.codecentric</groupId>
            <artifactId>spring-boot-admin-server-ui</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>de.codecentric</groupId>
                <artifactId>spring-boot-admin-dependencies</artifactId>
                <version>${spring-boot-admin.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
```

```
            </dependencies>
        </dependencyManagement>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Step#2:- At starter class level provide annotation **@EnableAdminServer**

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import de.codecentric.boot.admin.server.config.EnableAdminServer;

@SpringBootApplication
@EnableAdminServer
public class SpringCloudAdminUiDashboardsApp {
    public static void main(String[] args) {
        SpringApplication.run(SpringCloudAdminUiDashboardsApp.class, args);
    }
}
```

Step#3:- Provide details in application.properties

```
server.port=9999
```

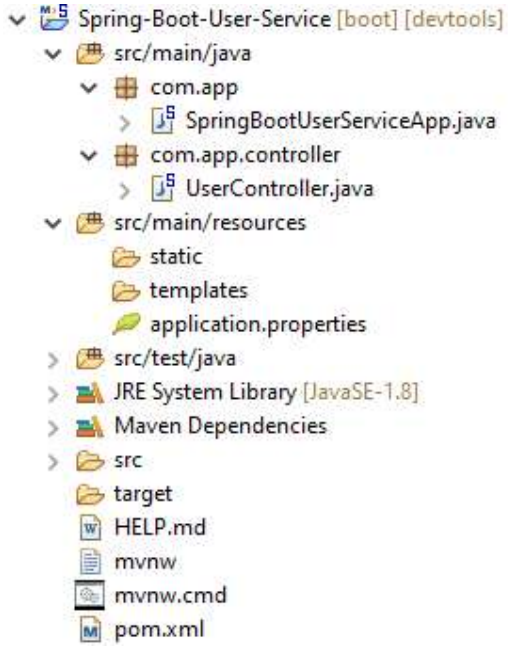
Spring Boot (Any Microservice) Client App:--

Step#1:- Create Spring Boot starter app with dependencies : web, admin client, Actuator.

Admin client dependency:--

```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
</dependency>
```

#25. Folder structure of User-Service:--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.1.1.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.app</groupId>
        <artifactId>Spring-Boot-User-Service</artifactId>
        <version>1.0</version>
        <name>Spring-Boot-User-Service</name>
        <description>Demo project for Spring JMS Implementation</description>

        <properties>
                <java.version>1.8</java.version>
```

```
        <spring-boot-admin.version>2.1.5</spring-boot-admin.version>
        <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>de.codecentric</groupId>
            <artifactId>spring-boot-admin-starter-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
```

```
                <dependency>
                    <groupId>de.codecentric</groupId>
                    <artifactId>spring-boot-admin-dependencies</artifactId>
                    <version>${spring-boot-admin.version}</version>
                    <type>pom</type>
                    <scope>import</scope>
                </dependency>
            </dependencies>
        </dependencyManagement>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Step#2:- provide Server details actuator endpoints in application.properties file.

```
server.port=6666
spring.boot.admin.client.url=http://localhost:9999
management.endpoints.web.exposure.include=*
spring.application.name=USER-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

Step#3:- Starter class of User Microservice

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import  org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
```

```
public class SpringBootUserServiceApp {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootUserServiceApp.class, args);
		System.out.println("User Microservice Executed…");
	}
}
```

Step#4:- Define one RestController (even service, DAL…)

```
package com.app.controller;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestControllers
@RequestMapping("user")
public class UserController {
	@RequestMapping("show")
	public String showUser() {
		System.out.println("Hello User");
	return null;
	}
}
```

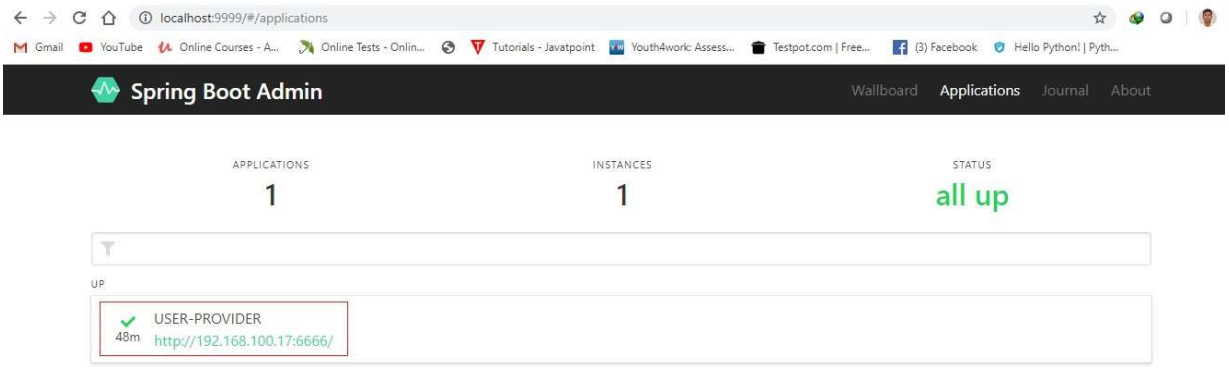
Execution Order:--

1>Run Admin Server

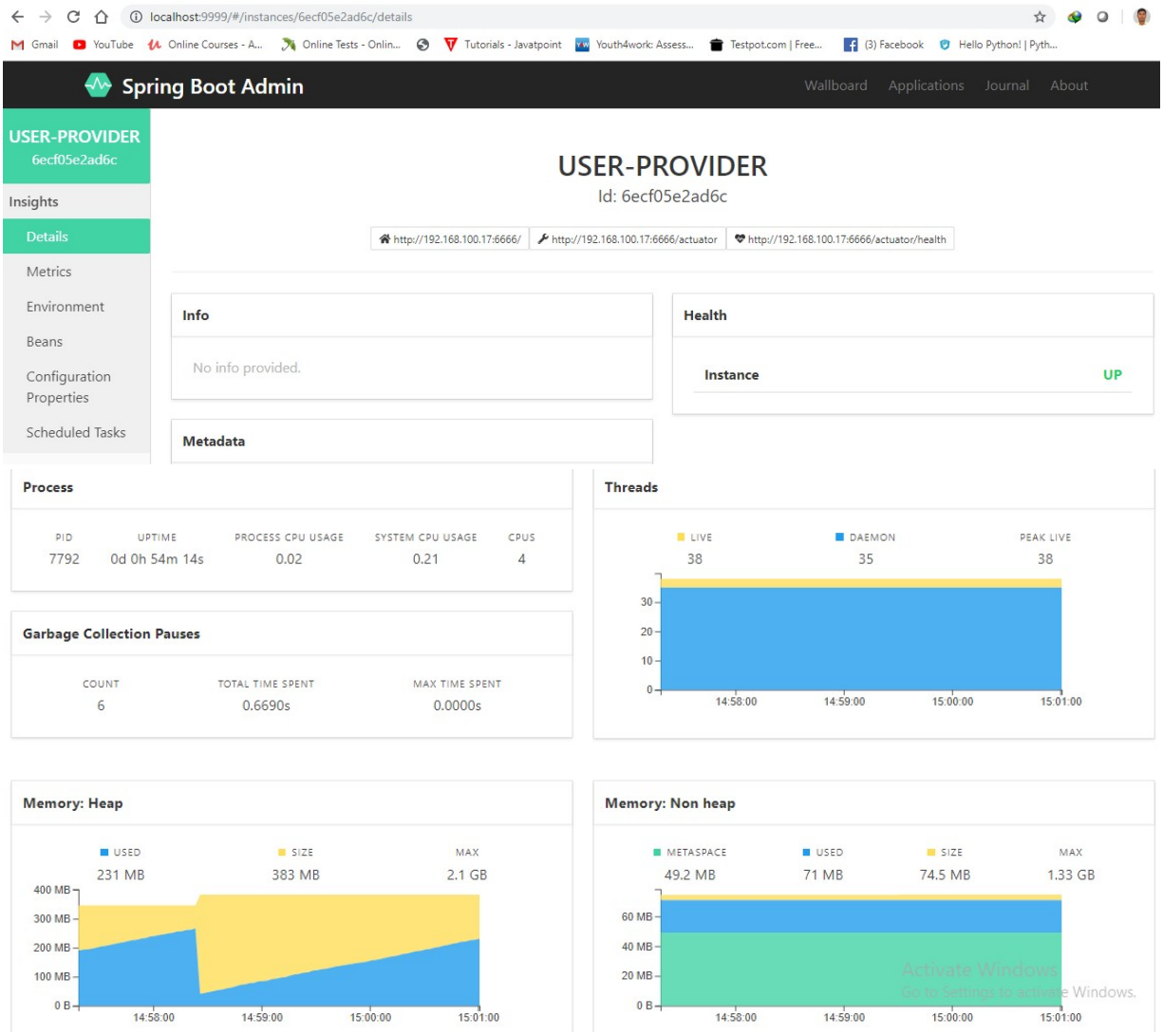
2>Then run ClientApps (SpringBootUserServiceApp)

=>Enter URL : http://localhost:9999

Output Screen:--



=>Click on **"User Provider"** Service to see full details.



NOTE:-- Add bellow all dependencies into all microservices.

1>Add Discovery client dependency in Microservice (Consumer, Producer, Config Server, Zuul Server) to register with Eureka Server.

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

2>Add Config-Client dependency in (Consumer, Provider, Zuul Server) to read properties from Config server.

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId> spring-cloud-config-client </artifactId>
</dependency>
```

3>To Register with Admin Server add below dependency in every application (Consumer, Provider even if Eureka Server, Config Server, ZUUL Server).

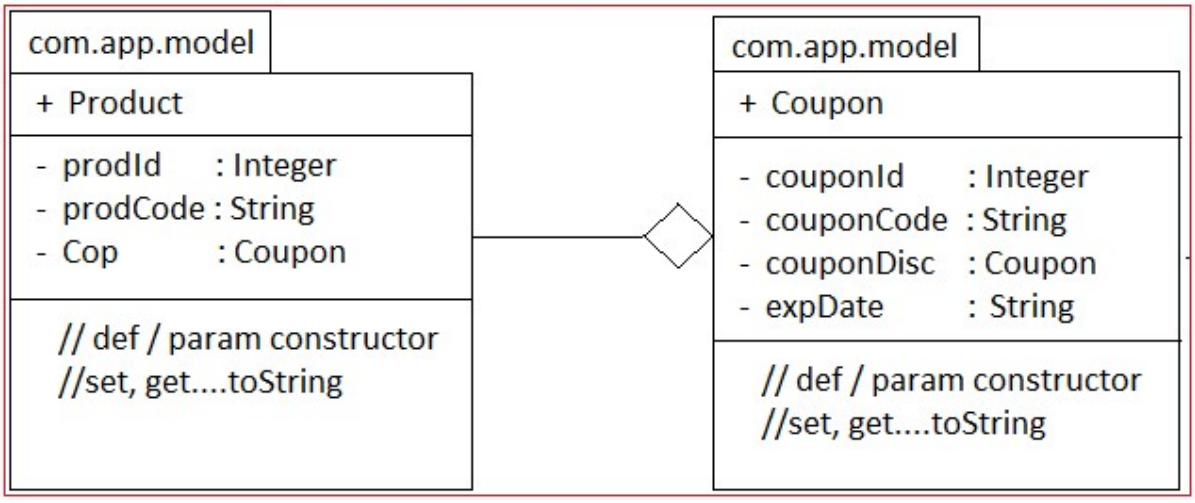
```
<dependency>
        <groupId>de.codecentric</groupId>
        <artifactId>spring-boot-admin-starter-client</artifactId>
</dependency>
```

4>To Provide Production ready Endpoint add actuator in Consumer Application.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Microservice Implementation Task:---

Product.....<>Coupon



NOTE:--1.>Implement two Micro-Service

1> CouponServiceProvider

2> ProductServiceProvider

2.>CouponService ProductShould take JSON from RestClient (POSTMAN) and create sample in Database.

```
{
   "couponId" : 8765,
  "couponCode" : "INDAUG15",
  "couponDisc" : 15,
  “expDate”    : 17/08/2019
}
```

=>Operation (Methods) : [Use Data JPA]

a>Create Coupon and Success Message

b>Get Coupon (full Object) byCoponCode, byCouponId

3.>Create Product using Coupon code only

```
{
   "prodId" : 999,
   "prodCode" : "PENDRIVE",
   "prodCost" : 876.56,
   "coupon" : {"code", : "INDAUG15"}
}
```

Operations:--

a.>Verify coupon code is valid or not also get Coupon Object.

b.>Calculate finalAmt = prodCost – discAmount

=>Store all details in database.

c.>Fetch all Product, fetch all Coupons

=>ProductRestController ---------->CouponRestConsumer(Feign Client)

4.>Coupon App, Product App should be registered with eureka Server.

5.>Common properties keys must be located from config Server.

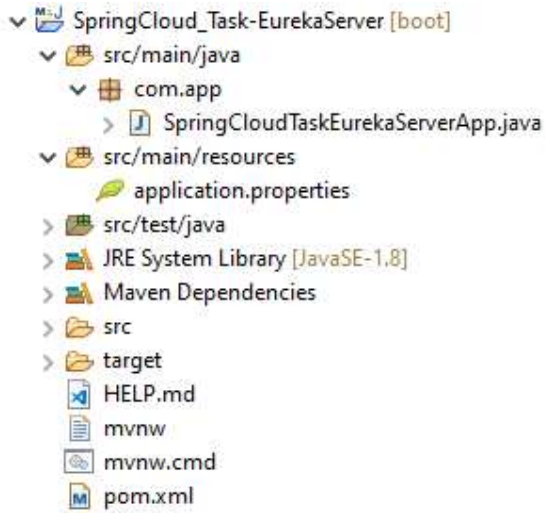
6.>Implement Hystrix for Product save (…).

7.>Enable Load Balancing for Coupon App (Multiple Instances).

8.>Enable Zuul API Gateway for both Apps.

16. MicroService Task:-- Task with Eureka, Config server, Zuul server, Swagger, Caching, Feign Client, Validation Of Data, custom error messages.

1. EurekaServer Folder Structure:--



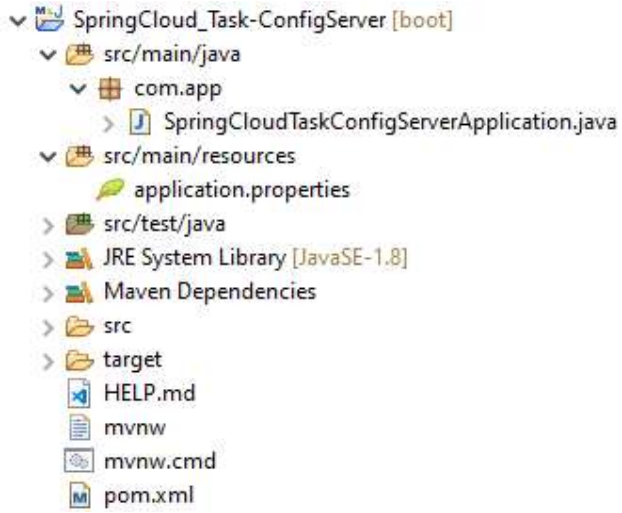
Step#1:- application.properties

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Step#2:-Eureka Server Starter class

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import  org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class SpringCloudTaskEurekaServerApp {

        public static void main(String[] args) {
                SpringApplication.run(SpringCloudTaskEurekaServerApp.class, args);
                System.out.println("Eureka Server Starter...!!");
        }
}
```

2. ConfigServer Folder Structure:--



1>application.properties

```
# Recommonded port
server.port = 8888
# External config file link spring.cloud.config.server.git.uri=https://github.com/saurabh-vaish/MicroserviceTask
#spring.cloud.config.server.git.uri=https://gitlab.com/udaykumar0023/configserverex
```

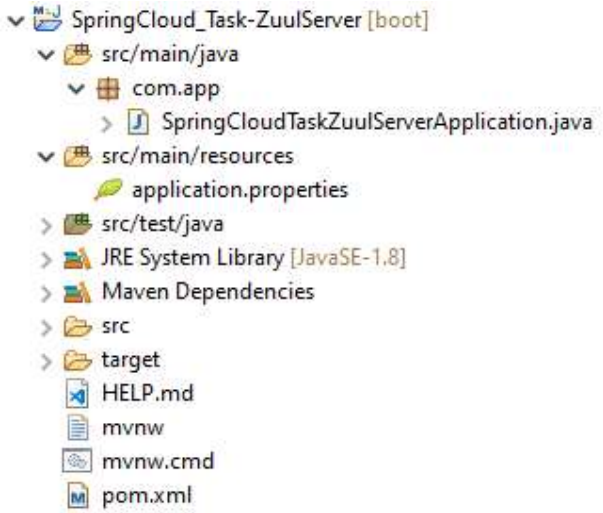
2>Starter class of Config Server

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class SpringCloudTaskConfigServerApplication {

    public static void main(String[] args)  {
        SpringApplication.run(SpringCloudTaskConfigServerApplication.class, args);
        System.out.println("Config Server Executed…!!!");
    }
}
```

External Config Server (Git) application.properties file:--

```
server.port=8888
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

## database connection (Oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system

## connection pooling
### max connections
spring.datasource.hikari.maximum-pool-size=12
### if no connection allocated
spring.datasource.hikari.connection-timeout=20000
### life time of sql
spring.datasource.hikari.max-lifetime=1200000
### idle connections
spring.datasource.hikari.minimum-idle=5
### if no activity then idle
spring.datasource.hikari.idle-timeout=30000
### if no activity then idle
spring.datasource.hikari.auto-commit=true

## jpa properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

3. ZuulServer Folder Structure:--



1>application.properties:--

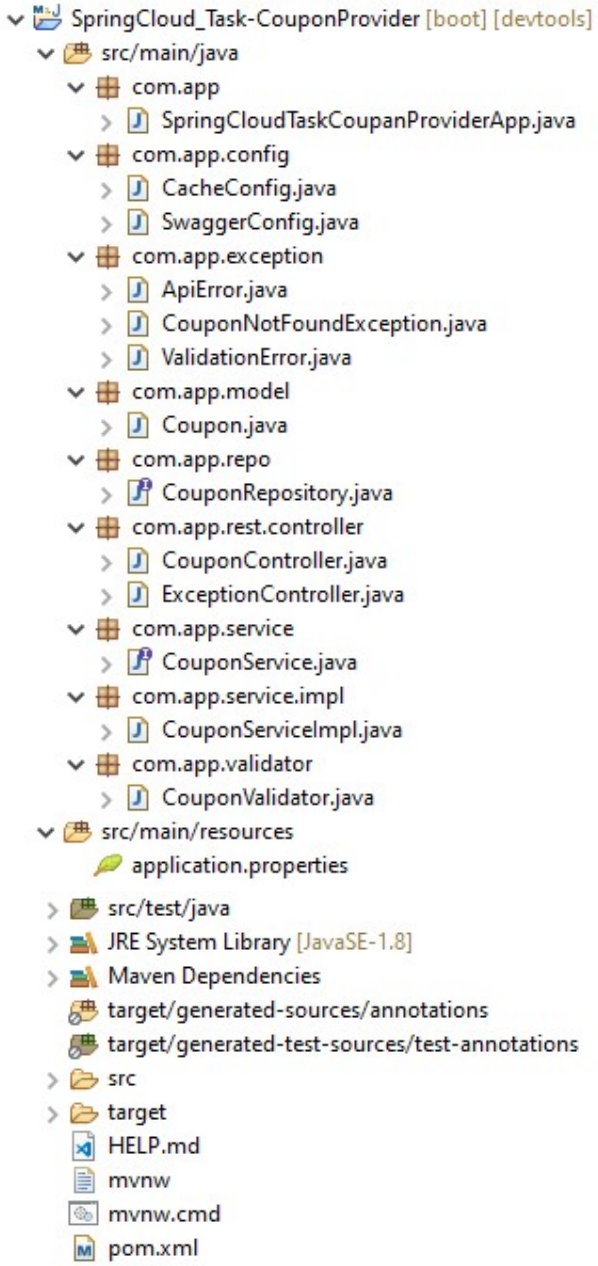
```
# zuul server port
server.port=6565
# service id for eureka server
spring.application.name=ZUUL_PROXY
# service url
eureka.client.service-url.default-zone=http://localhost:8761/eureka
# routing for coupan
zuul.routes.coupanserv.service-id=COUPAN-APP
zuul.routes.coupanserv.path=/coupan-api/**
# routing for product
zuul.routes.productserv.service-id=PRODUCT-APP
zuul.routes.productserv.path=/prod-api/**
```

2>Starter class of Zuul Server

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableZuulProxy
public class SpringCloudTaskZuulServerApplication {
```

```
        public static void main(String[] args) {
            SpringApplication.run(SpringCloudTaskZuulServerApplication.class, args);
            System.out.println("Zuul Server Executed...!!");
        }
}
```

4. CouponService Folder Structure:--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.1.7.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.app</groupId>
	<artifactId>SpringCloud_Task-CouponProvider</artifactId>
	<version>1.0</version>
	<name>SpringCloud_Task-CouponProvider</name>
	<description>Microservices project for Spring Boot</description>

	<properties>
		<java.version>1.8</java.version>
		<spring-cloud.version>Greenwich.SR2</spring-cloud.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-actuator</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
```

```
</dependency>
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.46</version>
        <!-- $NO-MVN-MAN-VER$ -->
        <scope>runtime</scope>
</dependency>

<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
</dependency>
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
</dependency>
```

```
<!--    Swagger  -->
            <dependency>
                  <groupId>io.springfox</groupId>
                  <artifactId>springfox-swagger2</artifactId>
                  <version>2.7.0</version>
            </dependency>
            <dependency>
                  <groupId>io.springfox</groupId>
                  <artifactId>springfox-swagger-ui</artifactId>
                  <version>2.7.0</version>
            </dependency>
<!-- hazel cast caching -->
            <dependency>
                  <groupId>com.hazelcast</groupId>
                  <artifactId>hazelcast</artifactId>
            </dependency>
            <dependency>
                  <groupId>com.hazelcast</groupId>
                  <artifactId>hazelcast-spring</artifactId>
            </dependency>
           <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-test</artifactId>
                  <scope>test</scope>
            </dependency>
            <dependency>
                  <groupId>com.oracle</groupId>
                  <artifactId>ojdbc6</artifactId>
                  <version>11.2.0</version>
            </dependency>
      </dependencies>

      <dependencyManagement>
            <dependencies>
                  <dependency>
```

```
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-dependencies</artifactId>
                        <version>${spring-cloud.version}</version>
                        <type>pom</type>
                        <scope>import</scope>
                    </dependency>
                </dependencies>
            </dependencyManagement>
            <build>
                <plugins>
                    <plugin>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-maven-plugin</artifactId>
                    </plugin>
                </plugins>
            </build>
</project>
```

1>application.properties:--

```
server.port=7777
# service Id
spring.application.name=COUPON-APP
# service-url
eureka.client.service-url.default-zone=http://localhost:8761/eureka
#load balancing
eureka.instance.instance-id=${spring.application.name}:${random.value}
# no need to write for config server port 8888
#spring.cloud.config.uri=http://localhost:8888
#spring.profiles.include=dev
```

2>Starter class:--

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;
import     org.springframework.cloud.netflix.eureka.EnableEurekaClient;
```

```
import org.springframework.cloud.netflix.hystrix.EnableHystrix;
@SpringBootApplication
@EnableEurekaClient
@EnableCaching
@EnableHystrix
public class SpringCloudTaskCoupanProviderApp {

	public static void main(String[] args) {
		SpringApplication.run(SpringCloudTaskCoupanProviderApp.class, args);
		System.out.println("Coupen Provider Executed…!!");
	}
}
```

3>Model class:--

```
package com.app.model;
import java.io.Serializable;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name="coupon_micro")
public class Coupon implements Serializable
 {
	private static final long serialVersionUID = 1L;
```

```
	@Id
	@Column(name="coupon_id")
	private Integer couponId;

	@Column(name="coupon_code", length=25)
	private String couponCode;

	@Column(name="coupon_discount")
	private Double couponDisc;

	@Column(name="coupon_expiry")
	@Temporal(TemporalType.DATE)
	private Date expDate;

	@JsonIgnore
	@Column(name="coupon_valid")
	private Boolean isValid = true;
}
```

4>Repository Interface:--

```
package com.app.repo;
import java.util.Date;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import com.app.model.Coupon;

public interface CouponRepository extends JpaRepository<Coupon, Integer>
{
	public Coupon findByCouponCode(String code);

	@Query("from com.app.model.Coupon as c where c.couponCode =:code
	  and c.expDate >=:date ")
	public Coupon findByCouponCodeAndExpDate(String code, Date date);
}
```

5>Service Interface:--

```
package com.app.service;
import java.util.List;
import com.app.model.Coupon;
public interface CouponService {

	public Coupon saveCoupon(Coupon coupon);
	public Coupon updateCoupon(Coupon coupon);
	public Coupon getCouponById(Integer id);

	public Coupon getCouponByCode(String code);
	public void deleteCouponById(Integer id);
	public List<Coupon> getAllCoupons();

	public boolean isExist(Integer id);
	public boolean isExpired(String code);
}
```

6>ServiceImpl class:--

```
package com.app.service.impl;
import java.util.Date;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Coupon;
import com.app.repo.CouponRepository;
import com.app.service.CouponService;

@Service
public class CouponServiceImpl implements CouponService
```

```
{
	@Autowired
	private CouponRepository repo;

	@Override
	@Transactional
	public Coupon saveCoupon(Coupon coupon) {
		return repo.save(coupon);
	}

	@Override
	@Transactional
	@CachePut(value = "coupon-cache",key="#coupon.couponId")
	public Coupon updateCoupon(Coupon coupon) {
		return repo.save(coupon);
	}

	@Override
	@Transactional(readOnly = true)
	@Cacheable(value = "coupon-cache",key="#couponId")
	public Coupon getCouponById(Integer couponId)
	{
		Optional<Coupon> c= repo.findById(couponId);
		return c.isPresent()?c.get():null;
	}

	@Override
	@Transactional(readOnly = true)
	public Coupon getCouponByCode(String code) {
		return repo.findByCouponCode(code);
	}
	@Override
	@Transactional
	@CacheEvict(value = "coupon-cache",key="#couponId")
	public void deleteCouponById(Integer couponId)  {
		repo.deleteById(couponId);
```

```
	}
	@Override
	@Transactional(readOnly = true)
	public List<Coupon> getAllCoupons() {
		return repo.findAll();
	}

	@Override
	public boolean isExpired(String code) {
		Coupon c= repo.findByCouponCodeAndExpDate(code, new Date());
		return (c!=null)?true:false;
	}

	@Override
	public boolean isExist(Integer id) {
		System.out.println(id);
		return repo.existsById(id);
	}
}
```

7>Validator class:--

```
package com.app.validator;
import java.util.Date;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.app.exception.ApiError;
import com.app.model.Coupon;

@Component
public class CouponValidator implements Validator{

	@Autowired
	private ApiError api=new ApiError();
```

```
@Override
public boolean supports(Class<?> clazz) {
        return clazz.equals(Coupon.class);
}

@Override
public void validate(Object target, Errors errors) {
   Coupon c = (Coupon) target;

   if(StringUtils.isEmpty(c.getCouponId().toString()))
   {
        errors.reject("couponId");
        api.getValidationErrors().add("Please Provide Coupon Id");
    }
    if(StringUtils.isEmpty(c.getCouponCode()))
    {
        errors.reject("couponCode");
        api.getValidationErrors().add("Please Provide Coupon Code ");
    }

    if(StringUtils.isEmpty(c.getCouponDisc().toString()))
    {
        errors.reject("couponDisc");
        api.getValidationErrors().add("Please Provide Coupon Discount ");
    }
    if(c.getCouponDisc()<0)
    {
        errors.reject("couponDisc");
        api.getValidationErrors().add("Please Provide Valid Discount ");
    }

    if(StringUtils.isEmpty(c.getExpDate()))
     {
        errors.reject("expDate");
        api.getValidationErrors().add("Please Provide Coupon Expiray Date");
```

```
            }
            if(c.getExpDate().before(new Date()))
            {
                errors.reject("expDate");
                api.getValidationErrors().add("Please Provide Valid Expiray Date");
            }
        }
}
```

8>Exception Class:--

1. API Class:--

```
package com.app.exception;
import java.util.ArrayList;
import  java.util.Date;
import java.util.List;
import org.springframework.stereotype.Component;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Component
public class ApiError {

        private String errorCode;
        private  String errorDesc;
        private Date errorDate;
        private List<String> validationErrors =new ArrayList<>();
}
```

2. Exception Validation class:--

```
package com.app.exception;

public class ValidationError extends RuntimeException {
```

```
        private static final long serialVersionUID = -5925942273970967113L;
        public ValidationError(String s) {
                super(s);
        }
}
```

3. CouponNotFoundException class:--

```
package com.app.exception;

public class CouponNotFoundException extends RuntimeException {
        private static final long serialVersionUID = 2627003438670611967L;

        public CouponNotFoundException(String s) {
                super(s);
        }
}
```

9>Controller class:--

1. CouponController:--

```
package com.app.rest.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.exception.CouponNotFoundException;
import com.app.exception.ValidationError;
import com.app.model.Coupon;
import com.app.service.CouponService;
import  com.app.validator.CouponValidator;
import     com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController
```

```
@RequestMapping("/rest/coupon")
public class CouponController {

	@Autowired
	private CouponService service;

	@Autowired
	private CouponValidator validator;

	@PostMapping("/save")
	public ResponseEntity<String> saveCoupon(@RequestBody Coupon
                                                    coupon, Errors errors)
	{
		validator.validate(coupon, errors);

		if(!errors.hasErrors()) {
			service.saveCoupon(coupon);
			 return new ResponseEntity<String>("Coupon has been added
                                                    successfully",HttpStatus.OK);
		} else {
			throw new ValidationError("Please Provide Valid Details");
		}
	}

	@GetMapping("/check/{code}")
	public String checkExpiredOrNot(@PathVariable String code) {
		return service.isExpired(code)?"Not Expired":"Expired";
	}

	@GetMapping("/getOne/{id}")
	@HystrixCommand(fallbackMethod = "getCouponException")
	public Coupon getOneCoupon(@PathVariable Integer id)
	{
		Coupon c = service.getCouponById(id);

		if(c!=null)
		{
			return c;
		} else {
			throw new CouponNotFoundException("No Coupon Found");
		}
```

```
	}

	@GetMapping("/getByCode/{code}")
	//@HystrixCommand(fallbackMethod = "getCouponCodeException")
	public Coupon getCouponByCode(@PathVariable String code)
	{
		System.out.println(code);
		Coupon c = service.getCouponByCode(code);

		if(c!=null)
		{
			return c;
		} else {
			throw new CouponNotFoundException("No Coupon Found");
		}
	}

	// fallback method (method name must be same as fallbackMethod name)
	public Coupon getCouponException(Integer id)
	{
		throw new CouponNotFoundException("No Coupon Found");
	}

	public Coupon getCouponCodeException(String code)
	{
		throw new CouponNotFoundException("No Coupon Found");
	}

	@GetMapping("/all")
	public List<Coupon> getAllCoupons()
	{
		List<Coupon> list =service.getAllCoupons();
		System.out.println(list);
		return list;
	}

	@PostMapping("/update")
	public ResponseEntity<String> updateCoupon(@RequestBody Coupon
                                                    coupon,Errors errors)
	{
		if(service.isExist(coupon.getCouponId()))
```

```
		{
			validator.validate(coupon, errors);

			if(!errors.hasErrors())
			{
				service.saveCoupon(coupon);
				return new ResponseEntity<String>("Coupon has been
							updated successfully", HttpStatus.OK);
			} else {
				throw new ValidationError("Please Provide Valid Details");
			}
		} else {
			throw new CouponNotFoundException("No Coupon Found");
	   }
	}

	@DeleteMapping("/delete/{id}")
	public ResponseEntity<String> deleteCoupon(@PathVariable Integer id)
	{
		if(service.isExist(id))
		{
			service.deleteCouponById(id);
			return new ResponseEntity<String>("Coupon Deleted
						Successfully", HttpStatus.OK);
		}  else {
			throw new CouponNotFoundException("No Coupon Found");
		}
	}
}
```

2. ExceptionController class:--

```
package com.app.rest.controller;
import java.util.Date;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import com.app.exception.ApiError;
import com.app.exception.CouponNotFoundException;
```

```
import com.app.exception.ValidationError;

@RestControllerAdvice
public class ExceptionController {

	@Autowired
	private ApiError api;

	@ExceptionHandler(value=CouponNotFoundException.class)
	public ResponseEntity<ApiError> noCouponFound()
	{
		ApiError error = new ApiError();

		error.setErrorCode("400");
		error.setErrorDesc("No Coupon Found ");
		error.setErrorDate(new Date());
		return new ResponseEntity<ApiError>(error,HttpStatus.BAD_REQUEST);
	}

	@ExceptionHandler(value=ValidationError.class)
	public ResponseEntity<ApiError> validationError()
	{
		api.setErrorCode("400");
		api.setErrorDesc("Please Provide Valid Details");
		api.setErrorDate(new Date());
		return new ResponseEntity<ApiError>(api,HttpStatus.BAD_REQUEST);
	}
}
```

10. Config Class:--

1. CacheConfig class:--

```
package com.app.config;
import    org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.hazelcast.config.Config;
import com.hazelcast.config.EvictionPolicy;
import com.hazelcast.config.MapConfig;
```

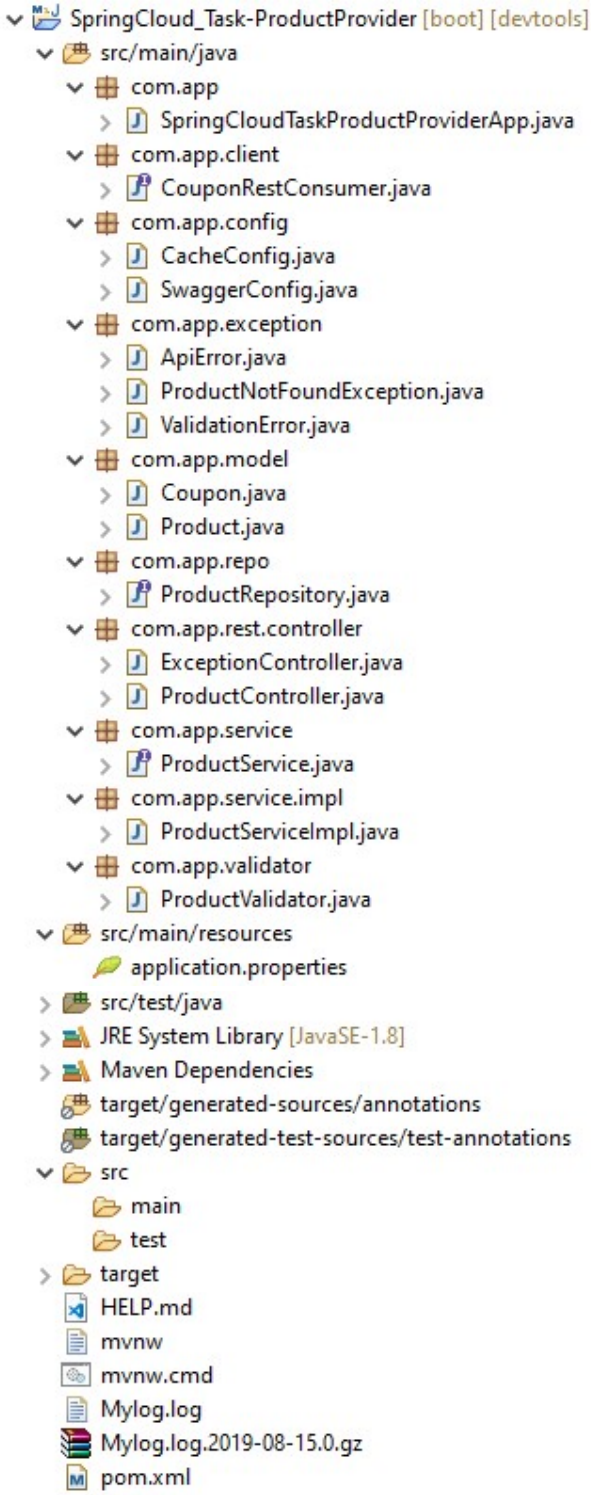
```
import com.hazelcast.config.MaxSizeConfig;
import com.hazelcast.config.MaxSizeConfig.MaxSizePolicy;
@Configuration
public class CacheConfig {

    @Bean
    public Config chacheConfig()
    {
        return new Config()                          // for creating cache
          .setInstanceName("hazelcast-cahe")         // setting cache name
          .addMapConfig(                             // Setting MapConfig  module
              new MapConfig()
                 .setName("coupon-cache")            // MapConfig Name
                 .setTimeToLiveSeconds(2000)         //  max time for object to present in
memory if no activity done
                  .setMaxSizeConfig(new
MaxSizeConfig(200,MaxSizePolicy.FREE_HEAP_SIZE))  // size of objects in MapConfig
                  .setEvictionPolicy(EvictionPolicy.LRU)
                  );
        }
}
```

2. Swagger Implementation:--

```
package com.app.config;
import static springfox.documentation.builders.PathSelectors.regex;
import static pringfox.documentation.builders.RequestHandlerSelectors.basePackage;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import     springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
```

```
@EnableSwagger2
public class SwaggerConfig  {
   @Bean
   public Docket configSwagger()    {

         return new Docket(DocumentationType.SWAGGER_2)
               .select()
               .apis(basePackage("com.app.rest.controller"))
               .paths(regex("/rest.*"))
               .build()
               // optional
               .apiInfo(apiInfo());
   }
         // it is optional only to provide additional details
    private ApiInfo apiInfo()
    {
               /* using builder factory ApiInfoBuilder that will make easy to
                  construct object and return ApiInfo*/
          return new ApiInfoBuilder()
                  .title("My Project Swagger")
                  .description("This is swagger implementation for rest")
                  .contact(new Contact("Saurabh vaish","www.github.com/saurabh-vaish",
                                       "saurabh.vaish1993@gmail.com"))
                  .license("Apache 2.0")
                  .licenseUrl("http://www.apache.org/licenses/LICENSE-2.0.html")
                  .version("1.0")
                  .build();
     }
}
```

5>ProductService Folder Structure Application:--



1. pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.1.7.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.app</groupId>
	<artifactId>SpringCloud_Task-ProductProvider</artifactId>
	<version>1.0</version>
	<name>SpringCloud_Task-ProductProvider</name>
	<description>Microservices project for Spring Boot</description>

	<properties>
		<java.version>1.8</java.version>
		<spring-cloud.version>Greenwich.SR2</spring-cloud.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-actuator</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-config</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
<!-- hazel cast caching -->
        <dependency>
            <groupId>com.hazelcast</groupId>
            <artifactId>hazelcast</artifactId>
        </dependency>
        <dependency>
```

```
                <groupId>com.hazelcast</groupId>
                <artifactId>hazelcast-spring</artifactId>
            </dependency>
 <!-- Swagger -->
            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger2</artifactId>
                <version>2.7.0</version>
            </dependency>
            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger-ui</artifactId>
                <version>2.7.0</version>
            </dependency>
<!—Data Base Use any one-->
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.1.46</version><!--$NO-MVN-MAN-VER$ -->
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>com.oracle</groupId>
                <artifactId>ojdbc6</artifactId>
                <version>11.2.0</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>
        </dependencies>

        <dependencyManagement>
            <dependencies>
                <dependency>
```

```
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-dependencies</artifactId>
                        <version>${spring-cloud.version}</version>
                        <type>pom</type>
                        <scope>import</scope>
                    </dependency>
                </dependencies>
            </dependencyManagement>

            <build>
                <plugins>
                    <plugin>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-maven-plugin</artifactId>
                    </plugin>
                </plugins>
            </build>
</project>
```

2>application.properties:--

```
server.port=9999
# service id
spring.application.name=PRODUCT-APP
# service-url
eureka.client.service-url.default-zone=http://localhost:8761/eureka
#load balancing
eureka.instance.instance-id=${spring.application.name}:${random.value}
# no need to write for config server port 8888
#spring.cloud.config.uri=http://localhost:8888
# loggin keys
logging.level.root=info
logging.level.com.app=DEBUG
logging.file=Mylog.log
logging.file.max-size=5MB
logging.file.max-history=5
```

```
#Database details
spring.datasource.driverClassName=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system

#JPA details
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
```

3>Starter class:--

```
package com.app;
import    org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.hystrix.EnableHystrix;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableEurekaClient
@EnableCaching
@EnableHystrix
@EnableFeignClients
public class SpringCloudTaskProductProviderApp {
        public static void main(String[] args) {
                SpringApplication.run(SpringCloudTaskProductProviderApp.class, args);
                System.out.println("Product Provider Executed…!!");
        }
}
```

4>Model class:--

1. Coupon class:--

```
package com.app.model;
import com.fasterxml.jackson.annotation.JsonIgnore;
```

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Coupon {

	private Integer couponId;
	private String couponCode;

	private Double couponDisc;
	private String expDate;

	@JsonIgnore
	private Boolean applied=false;
}
```

2. Product class:--

```
package com.app.model;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import   javax.persistence.Transient;
import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name="product_micro")
public class Product implements Serializable  {
```

```
	private static final long serialVersionUID = 1L;

	@Id
	private Integer prodId;
	private String prodCode;
	private Double prodCost;

	 @JsonIgnore
	private Double finalCost;

	@Transient
	private Coupon coupon;
}
```

5>Repository Interface:--

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Product;

public interface ProductRepository extends JpaRepository<Product, Integer>   {  }
```

6>Service Interface:--

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface ProductService {

	public Product saveProduct(Product product);
	public Product updateProduct(Product product);
	public Product getProductById(Integer id);
	public void deleteProductById(Integer id);
	public List<Product> getAllProducts();

	public boolean isExist(Integer id);
```

```
	public Double calculateDiscount(Double cost,Double disc);
}
```

7>ServiceImpl Class:--

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cache.annotation.CacheEvict;
import org.springframework.cache.annotation.CachePut;
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.ProductService;

@Service
public class ProductServiceImpl implements ProductService {

	@Autowired
	private ProductRepository repo;

	@Override
	@Transactional
	public Product saveProduct(Product product) {
		return repo.save(product);
	}

	@Override
	@Transactional
	@CachePut(value = "product-cache",key="#product.prodId")
	public Product updateProduct(Product product) {
		return repo.save(product);
	}
```

```
	@Override
	@Transactional(readOnly = true)
	@Cacheable(value = "product-cache",key="#prodId")
	public Product getProductById(Integer prodId) {
		Optional<Product> c= repo.findById(prodId);
		return c.isPresent()?c.get():null;
	}

	@Override
	@Transactional
	@CacheEvict(value = "product-cache",key="#prodId")
	public void deleteProductById(Integer prodId) {
		repo.deleteById(prodId);
	}

	@Override
	@Transactional(readOnly = true)
	public List<Product> getAllProducts() {
		return repo.findAll();
	}

	@Override
	public boolean isExist(Integer id) {
		return repo.existsById(id);
	}

	@Override
	public Double calculateDiscount(Double cost, Double disc) {
		Double dCost = cost*disc/100.0;
		Long value = Math.round(cost-dCost);
		return value.doubleValue();
	}
}
```

8>Validator Class:--

```
package com.app.validator;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.app.exception.ApiError;
import com.app.model.Product;

@Component
public class ProductValidator implements Validator{

	@Autowired
	private ApiError api;

	@Override
	public boolean supports(Class<?> clazz) {
		return clazz.equals(Product.class);
	}

	@Override
	public void validate(Object target, Errors errors) {

		Product p = (Product) target;

		if(StringUtils.isEmpty(p.getProdId().toString()))  {
			errors.reject("prodId");
			api.getValidationErrors().add("Please Provide Product Id");
		}

		if(StringUtils.isEmpty(p.getProdCode())) {
			errors.reject("prodCode");
			api.getValidationErrors().add("Please Provide Product Code ");
		}
		if(StringUtils.isEmpty(p.getProdCost().toString()))   {
			errors.reject("prodCost");
			api.getValidationErrors().add("Please Provide Product Cost ");
```

```
            }
        }
}
```

9>Exception Class:--

1. API Erro class:--

```
package com.app.exception;
import java.util.ArrayList;
import  java.util.Date;
import java.util.List;
import org.springframework.stereotype.Component;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Component
public class ApiError {

        private String errorCode;
        private  String errorDesc;
        private Date errorDate;

        private List<String> validationErrors =new ArrayList<>();
}
```

2. Validation class:--

```
package com.app.exception;

public class ValidationError extends RuntimeException {

        private static final long serialVersionUID = -5925942273970967113L;

        public ValidationError(String s) {
                super(s);
```

```
	}
}
```

3. ProductNotFoundException:--

```
package com.app.exception;

public class ProductNotFoundException extends RuntimeException {

	private static final long serialVersionUID = 1L;

	public ProductNotFoundException(String s) {
		super(s);
	}
}
```

10>ClientRestController class:--

```
package com.app.client;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import com.app.model.Coupon;

@FeignClient("COUPON-APP")
public interface CouponRestConsumer {

	@GetMapping("/rest/coupon/getByCode/{code}")
	public Coupon getCouponByCode(@PathVariable String code);

	@GetMapping("/rest/coupon/check/{code}")
	public String checkExpiredOrNot(@PathVariable String code);
}
```

11>Controller class:--

1. ProductController:--

```
package com.app.rest.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.validation.Errors;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.client.CouponRestConsumer;
import com.app.exception.ProductNotFoundException;
import com.app.exception.ValidationError;
import com.app.model.Coupon;
import com.app.model.Product;
import com.app.service.ProductService;
import  com.app.validator.ProductValidator;

@RestController
@RequestMapping("/rest/product")
public class ProductController {

	@Autowired
	private ProductService service;

	@Autowired
	private CouponRestConsumer consumer;

	@Autowired
	private ProductValidator validator;


	@PostMapping("/save")
	public ResponseEntity<String> saveProduct(@RequestBody Product
                                                  product,Errors errors)
	{
		validator.validate(product, errors);

		if(!errors.hasErrors())
		{
String res= consumer.checkExpiredOrNot (product.getCoupon().getCouponCode());
System.out.println(res);
```

```
Coupon c = consumer.getCouponByCode(product.getCoupon().getCouponCode());
System.out.println(c);
                        if(c==null)
                        {
 return new ResponseEntity<String>("Coupon Not Found",HttpStatus.OK);
                        }
                        else if(res.equals("Expired"))
                        {
return new ResponseEntity<String>("Coupon has been expired !",HttpStatus.OK);

                        }  else {
                                Double dis = c.getCouponDisc();
                                if(product.getCoupon().getApplied())  // if coupon applied
                                {
                                        dis=0.0;
                                }

Double cost = product.getProdCost(); System.out.println(cost+","+dis);
                                product.setFinalCost(service.calculateDiscount(cost, dis));
                                service.saveProduct(product);
return new ResponseEntity<String>("Product has been added successfully",
HttpStatus.OK);
                        }
                } else  {
                        throw new ValidationError("Please Provide Valid Details");
                }
        }

        @GetMapping("/apply")
        public ResponseEntity<Double> applyCoupon(Double cost,Double disc)
        {
                return new ResponseEntity<Double>(service.calculateDiscount(cost,
                                                        disc),HttpStatus.OK);
        }

        @GetMapping("/getOne/{id}")
        //@HystrixCommand(fallbackMethod = "getProductException")
        public ResponseEntity<Product> getOneProduct(@PathVariable Integer id)
        {
                Product p = service.getProductById(id);System.out.println(p);
```

```
		if(p!=null)
		{
			return new ResponseEntity<Product>(p,HttpStatus.OK);
		} else {
			throw new ProductNotFoundException("No Product Found");
		}
	}

	// fallback method
	public ResponseEntity<Product> getProductException(Integer id)
	{
		throw new ProductNotFoundException("No Product Found");
	}

	@GetMapping("/all")
	public List<Product> getAllProducts()
	{
		return service.getAllProducts();
	}

	@PostMapping("/update")
	public ResponseEntity<String> updateProduct(@RequestBody Product
                                                    product, Errors errors)
	{
		if(service.isExist(product.getProdId()))
		{
			validator.validate(product, errors);

			if(!errors.hasErrors())
			{
				service.saveProduct(product);
				return new ResponseEntity<String>("Product has been
updated successfully",HttpStatus.OK);
			} else  {
				throw new ValidationError("Please Provide Valid Details");
			}
		} else {
			throw new ProductNotFoundException("No Product Found");
		}
	}
```

```
	@DeleteMapping("/delete/{id}")
	public ResponseEntity<String> deleteProduct(@PathVariable Integer id)
	{
		if(service.isExist(id)) {
			service.deleteProductById(id);

return new ResponseEntity<String>("Product Deleted Successfully",HttpStatus.OK);
		} else {
			throw new ProductNotFoundException("No Product Found");
		}
	}
}
```

12>Config class:--

1. Config class:--

```
package com.app.config;
import    org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.hazelcast.config.Config;
import com.hazelcast.config.EvictionPolicy;
import com.hazelcast.config.MapConfig;
import com.hazelcast.config.MaxSizeConfig;
import    com.hazelcast.config.MaxSizeConfig.MaxSizePolicy;

@Configuration
public class CacheConfig {

    @Bean
    public Config chacheConfig()
    {
        return new Config()       // for creating cache
            .setInstanceName("hazelcast-cahe")  // setting cache name
            .addMapConfig(                          // Setting MapConfig  module
                new MapConfig()
                .setName("product-cache")       // MapConfig Name
                .setTimeToLiveSeconds(2000) // max time for object to present in
memory if no activity done
```

```
        .setMaxSizeConfig(new      MaxSizeConfig(200,MaxSizePolicy.FREE_HEAP_SIZE))
// size of objects in MapConfig
                        .setEvictionPolicy(EvictionPolicy.LRU)
                        );
    }
 }
```

2. Swagger Confg:--

```
package com.app.config;
import static springfox.documentation.builders.PathSelectors.regex;
import static springfox.documentation.builders.RequestHandlerSelectors.basePackage;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import     springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SwaggerConfig {

        @Bean
        public Docket configSwagger()
        {
                return new Docket(DocumentationType.SWAGGER_2)

                        .select()
                        .apis(basePackage("com.app.rest.controller"))
                        .paths(regex("/rest.*"))
                        .build()

                        // optional
                       .apiInfo(apiInfo());
        }
```

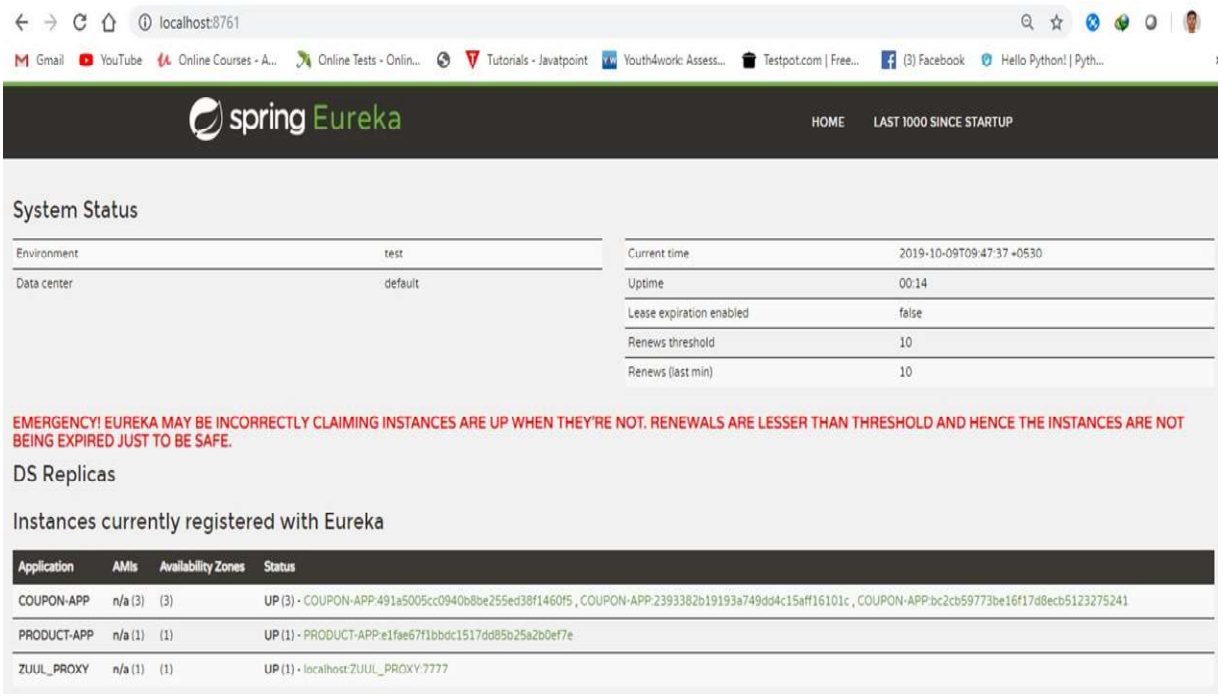
```
        // it is optional only to provide additional details
        private ApiInfo apiInfo()
        {
                return new ApiInfoBuilder()      // using builder factory ApiInfoBuilder that
will make easy to construct object and return ApiInfo
        .title("My Project Swagger")
                .description("this is swagger implementation for rest")
   .contact(new Contact("saurabh vaish","http://www.github.com/saurabh-vaish",
                                "saurabh.vaish1993@gmail.com"))
                .license("Apache 2.0")
                .licenseUrl("http://www.apache.org/licenses/LICENSE-2.0.html")
                .version("1.0")
                .build();
        }
}
```

Execution Order:--

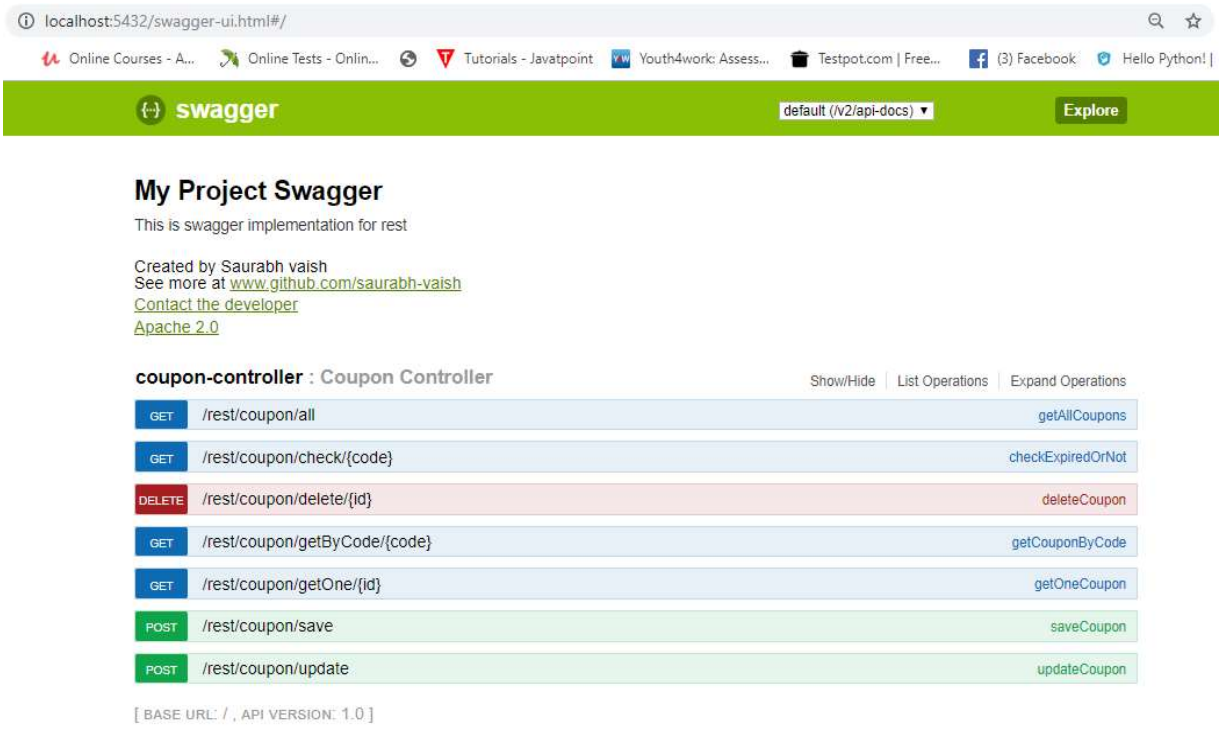
1.>Eureka Server
2.>Config Server
3.>Zuul Server
4.>CouponProvider Application Starter class
5.>ProductProvider Application Starter class

OUTPUT SCREEN STEP BY STEP:--

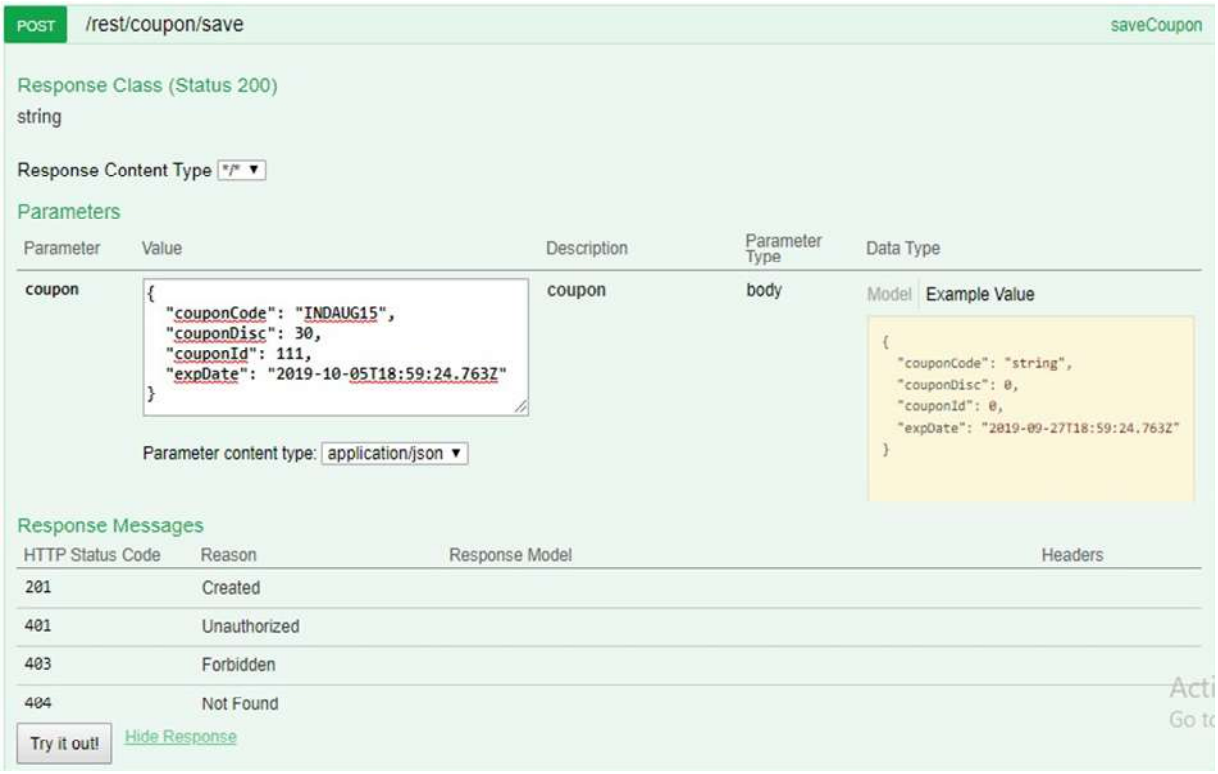
1>Eureka Server (type http://localhost:8761 in browser to see Eureka Dashboard):--



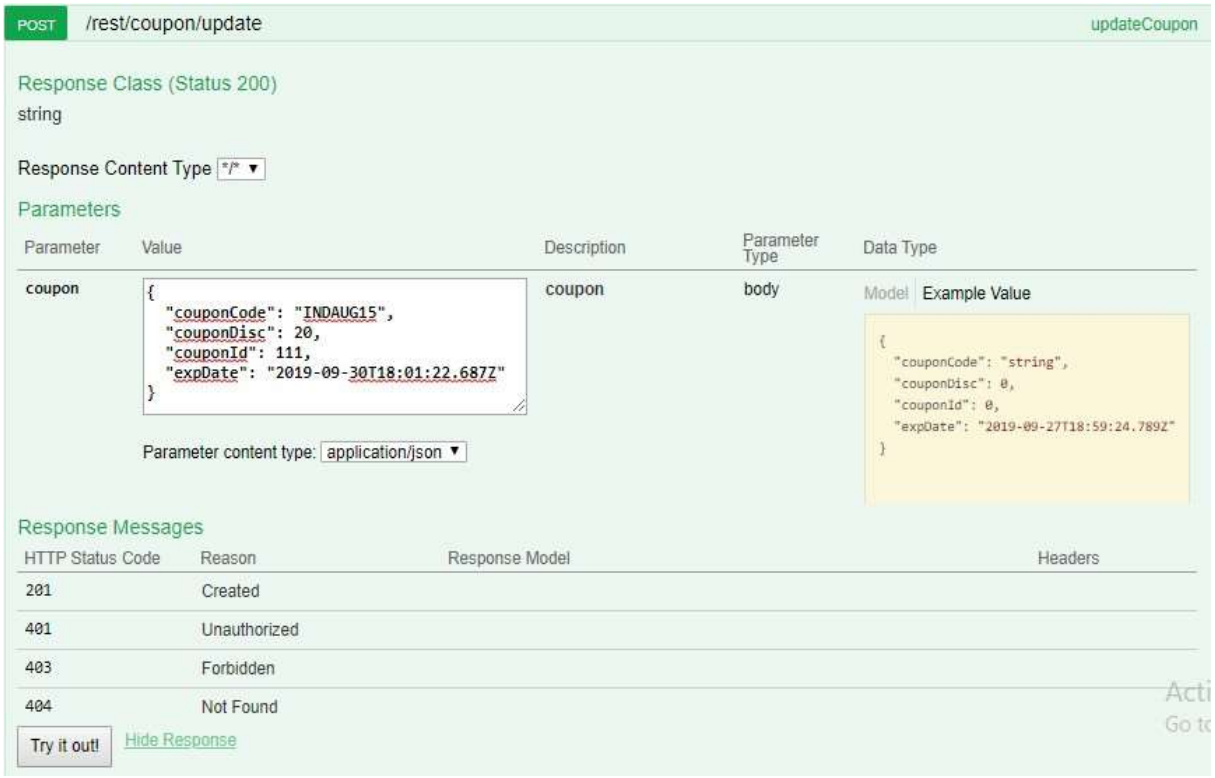
2>CouponProvider Application Swagger SCREEN:--



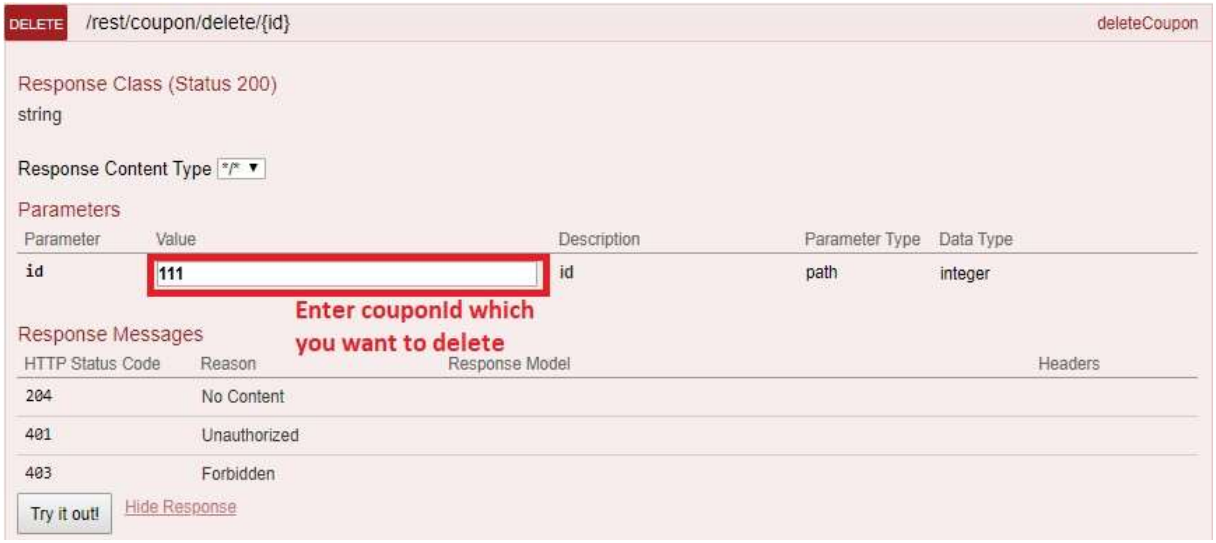
1. Swagger screen for **Save** Operation for CouponProvider Application:--



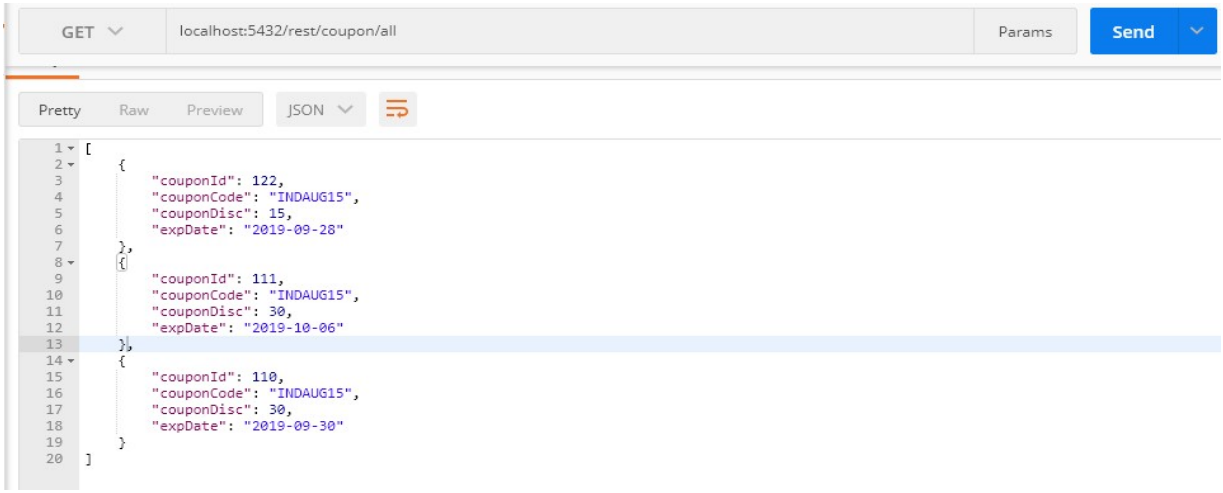
2. Swagger screen for **update** Operation for CouponProvider Application:--



3. Swagger screen for **delete** Operation for CouponProvider Application:--

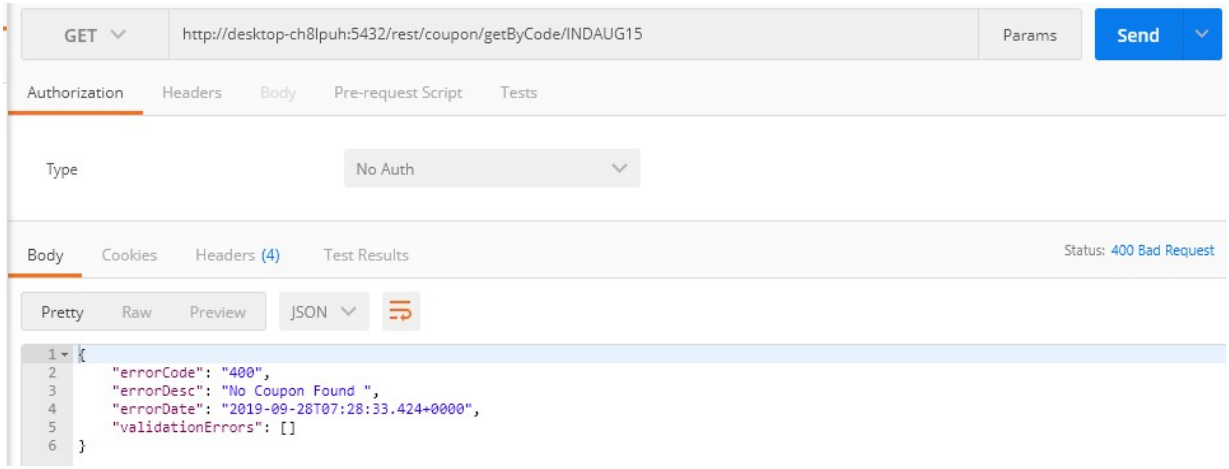


4. POSTMAN screen for **GETALL** Operation for CouponProvider Application:--

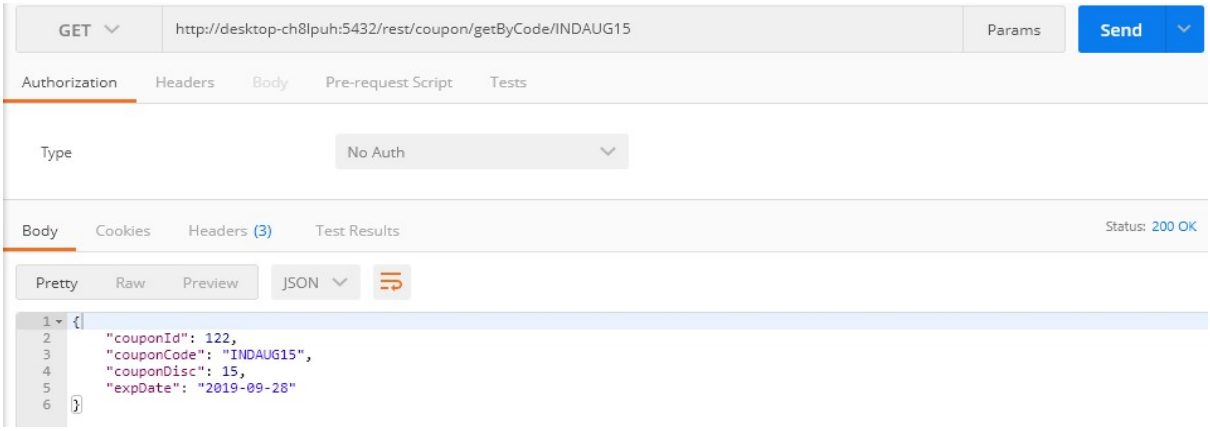


5. POSTMAN screen for **GetByCode** Operation for CouponProvider Application:--

5.1. If code Not available

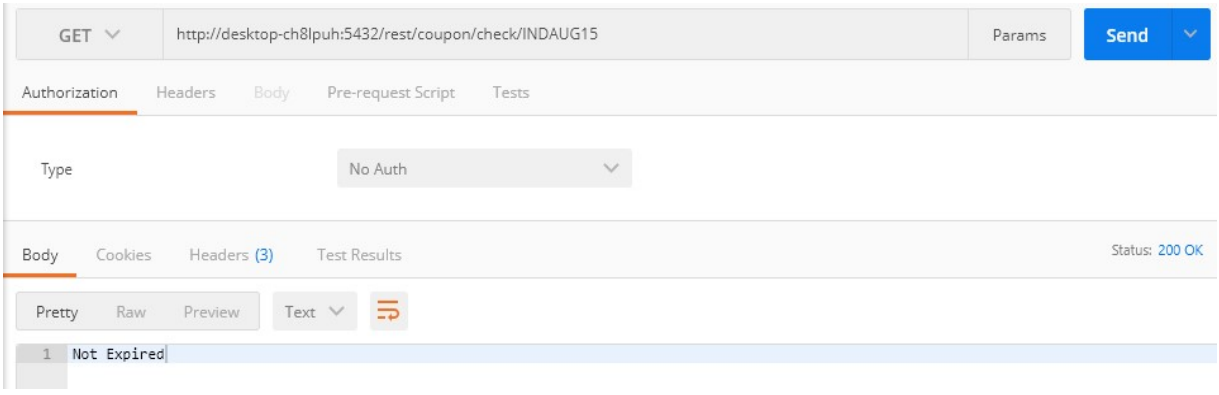


5.2. If Code available

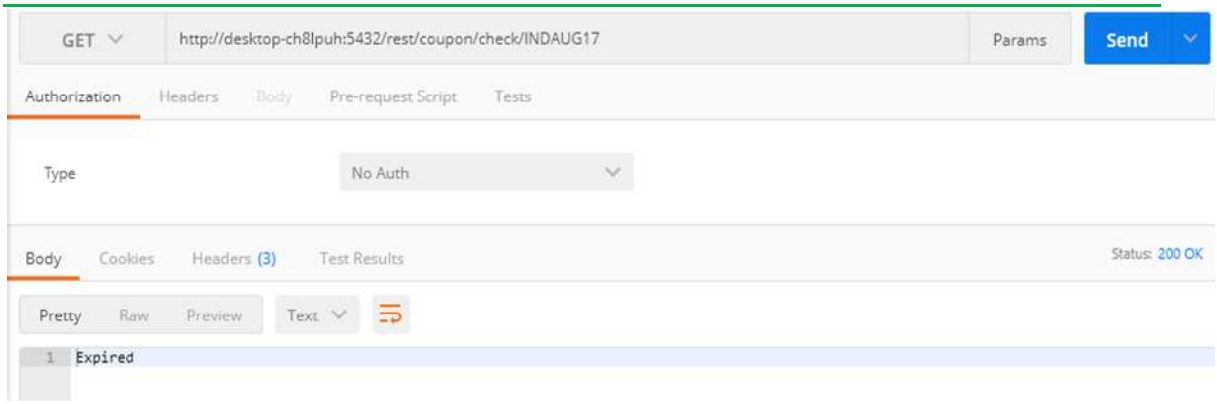


6. POSTMAN screen for **CHECK CODE VALIDITY** Operation for CouponProvider Appl:--

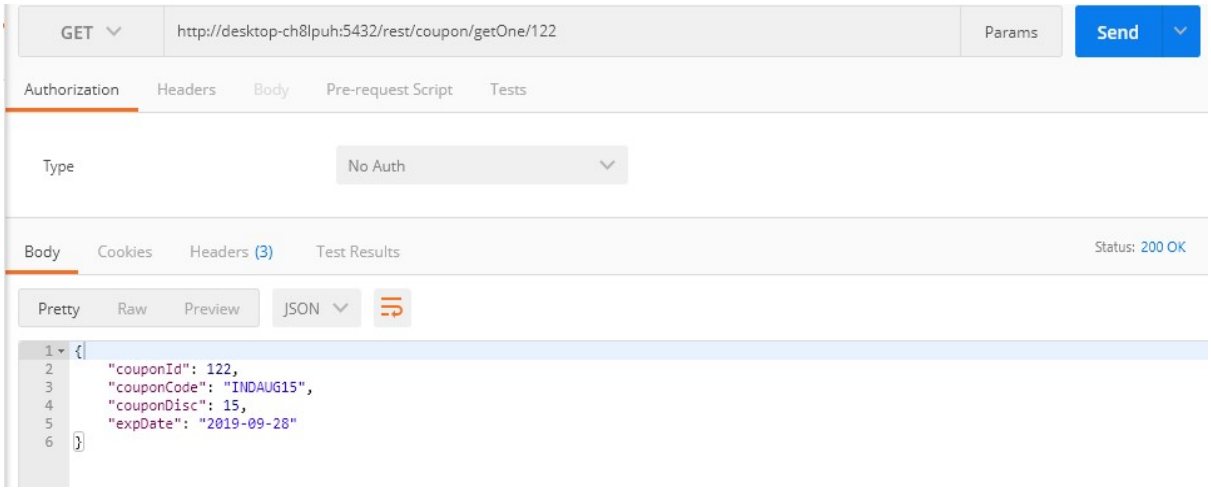
6.1. If Code is not expire.



6.2. If code is expire

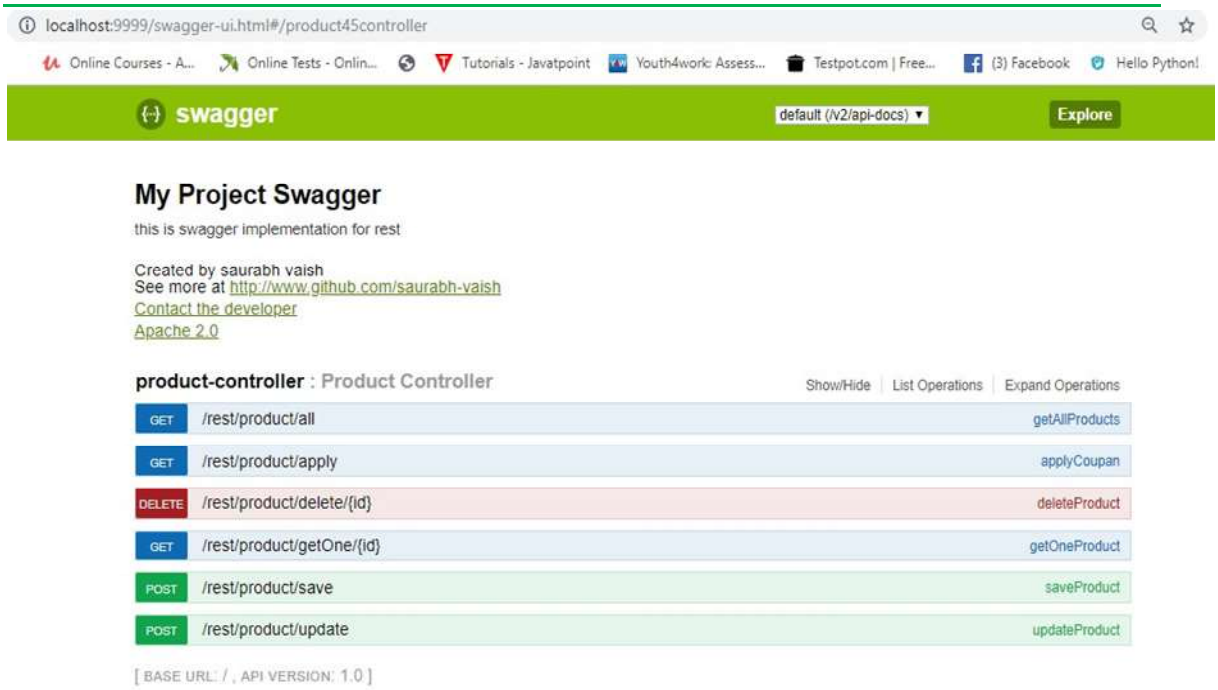


7. Postman screen for **getOne** by couponId Operation for CouponProvider Appl:--



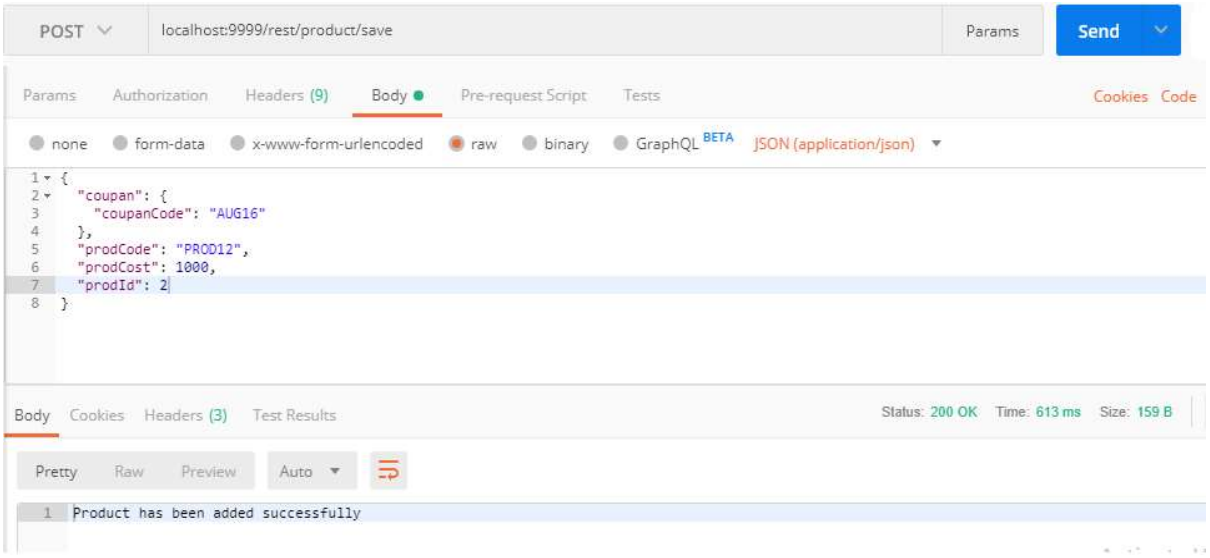
3. ProductService Swagger Screen Shot:--

localhost:9999/swagger-ui.html#/

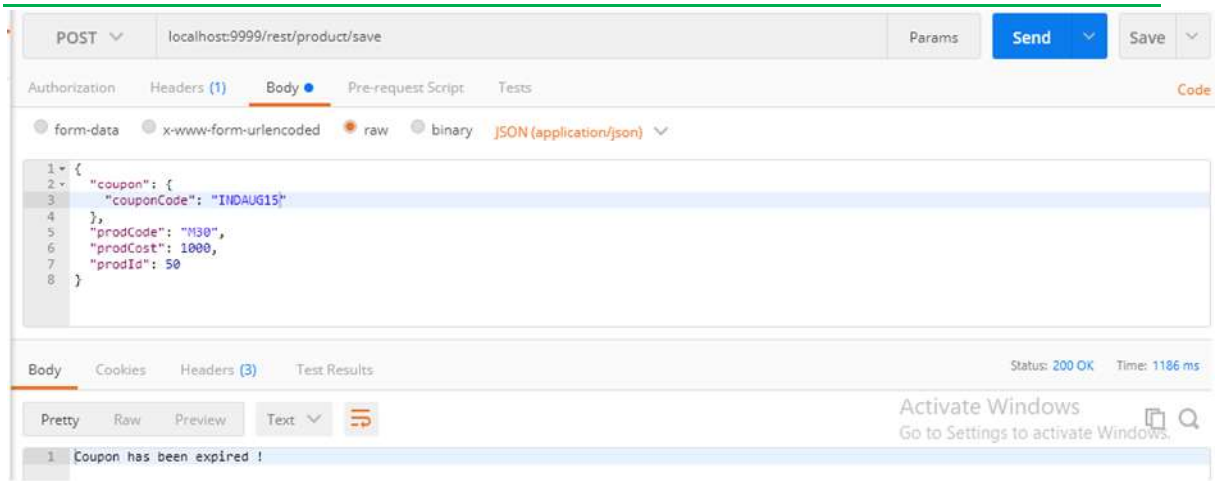


1. Postman screen for **Save** Operation for ProductProvider Application:--

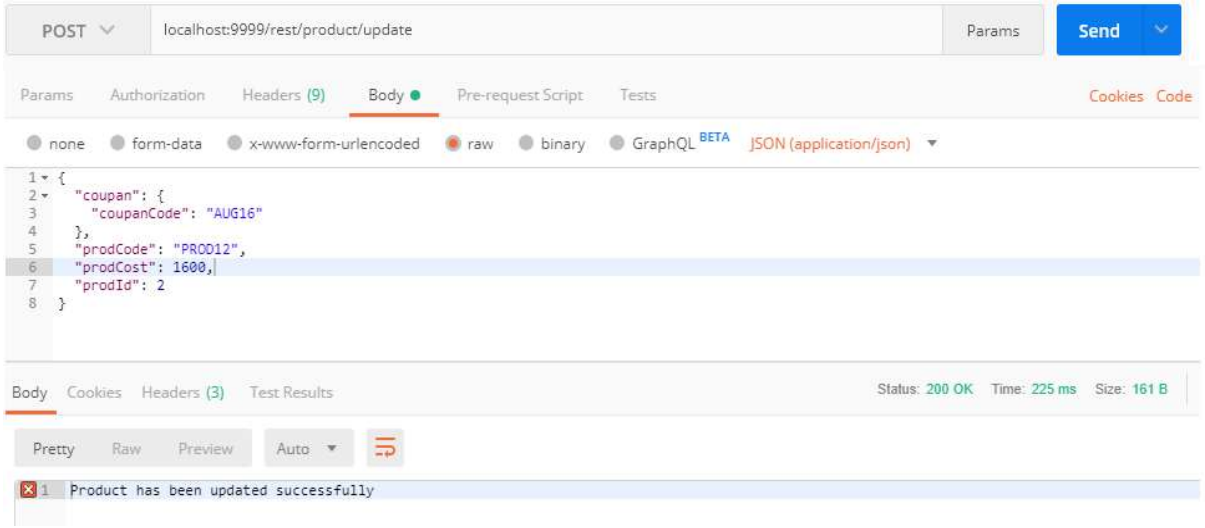
i. When Coupon is valid



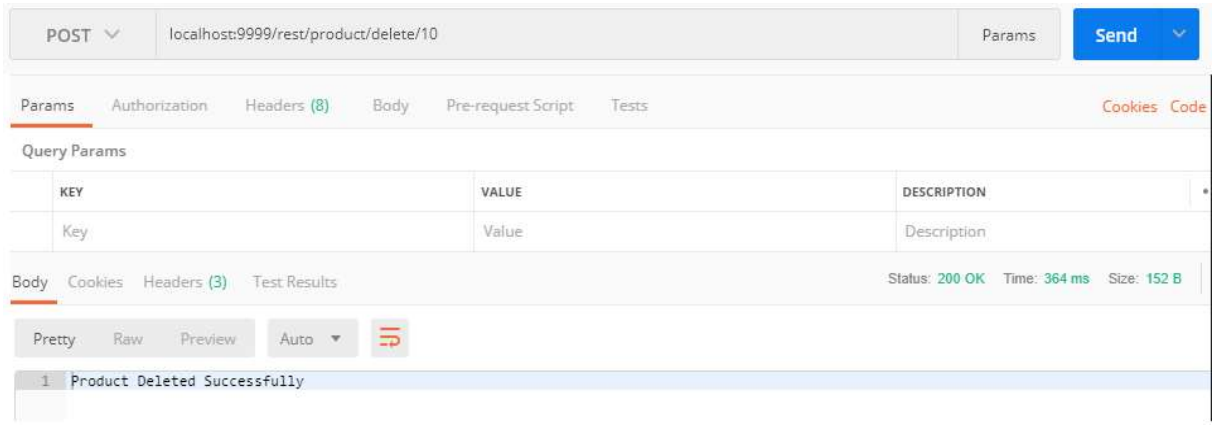
ii. If Coupon is expired



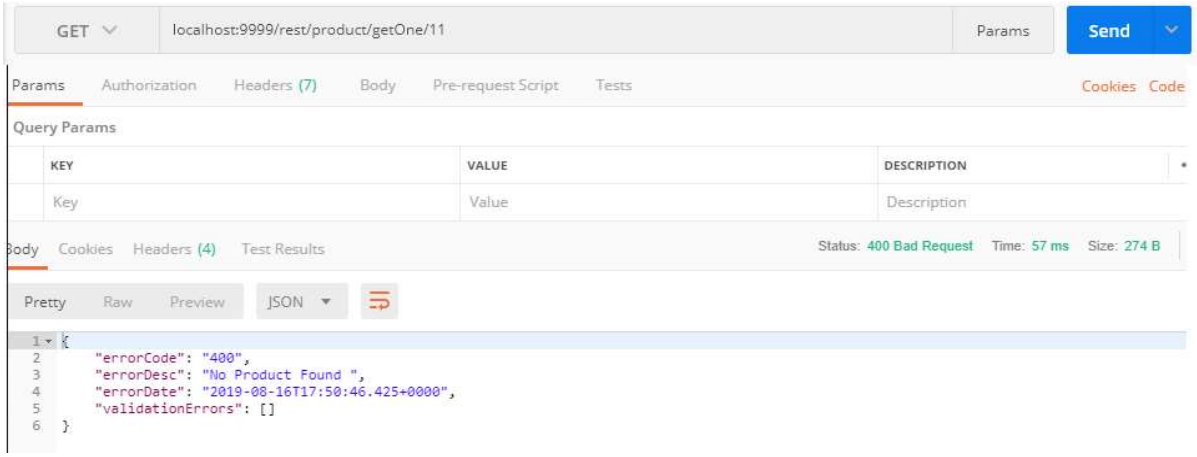
2. Postman screen for **Update** by ProductId Operation for ProductProvider Appl:--



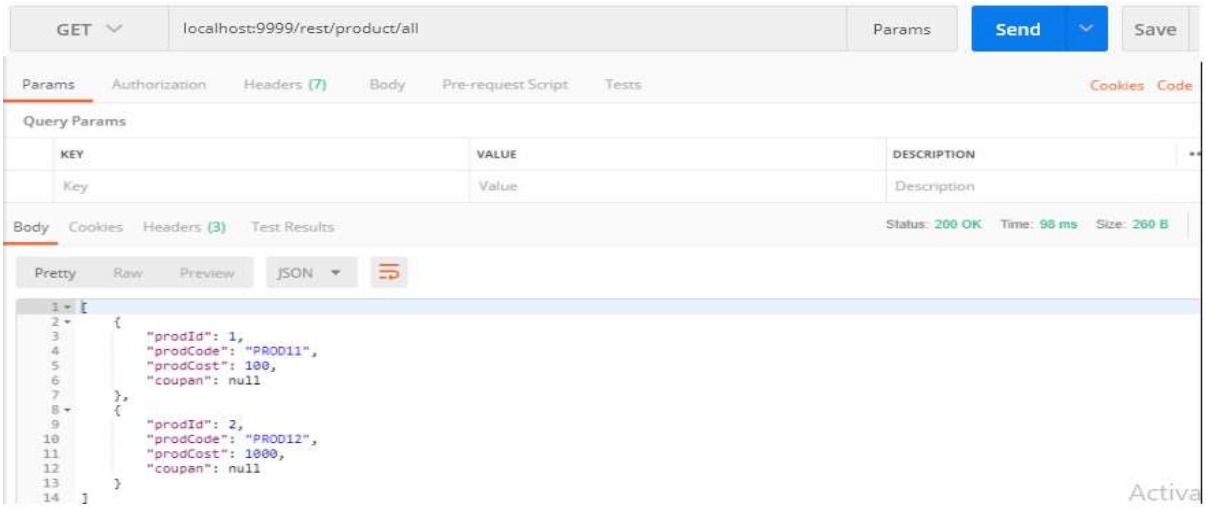
3. Postman screen for **Delete** by productId Operation for ProductProvider Appl:--



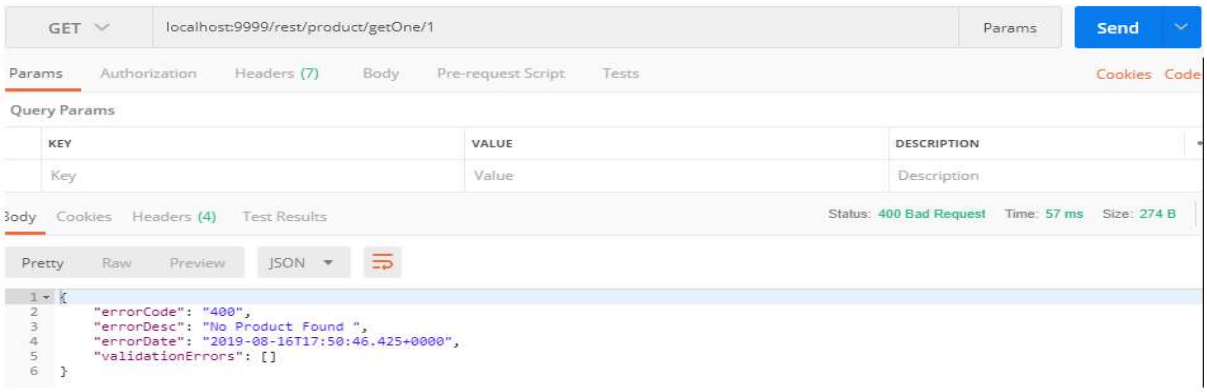
4. Postman screen for **getOne** by productId Operation for ProductProvider Appl:--



5. Postman screen for **geAll record** Operation for ProductProvider Appl:--



6. Postman screen for **getOne** ValidationError ProductProvider Appl:--



MOCKITO

Spring Boot UnitTesting using JUnit+Mocking:--

Mock:-- It is a component which acts like Client to make request.
=>Behaves as Container.
=>Supports object creation and Destroy.
=>Uses Proxy design Pattern.
=>Supports HTTP calls, No need of using any client (**RestTemplate, HttpClient**… etc).

Mocking is implemented using

1. EasyMock
2. Mockito

=>Spring boot uses JUnit 4 + Mockito 2 Integration for UnitTesting of Applications.
=>Here Mockito supports,

a. HTTP Request Creation.
b. Making HTTP Client (GET/POST…).
c. Execute code using Proxies.
d. GetResult and store in HTTP Response objects.

=>Here JUnit is used for

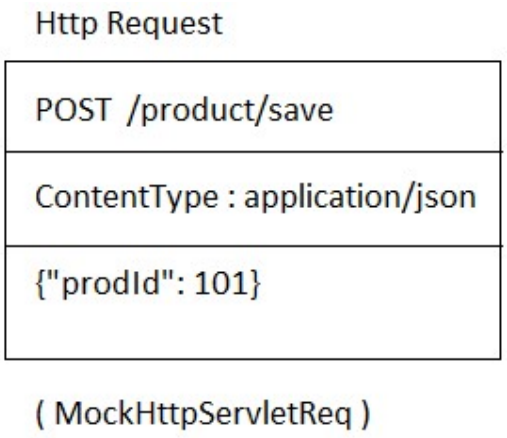
a. Writing Test cases with annotations.
b. Verify Result using assert methods (Return PASS/FALL).

Step#1:- Create one Spring Starter Project and define RestController with methods and URLs added.

Step#2:- Define UnitTesting class (Test case) under src/test/java.
a. Autowire MockMvc (C) Object[acts as Http Client and supports container communication].
b. Call perform method to make Http Request but construct Request object at same time that returns as **MockHttpServletRequest** (use **RequestBuilder** static methods and call).

Ex:- mockMvc.perform(post("/product/save").header("Content-Type", "application/json").content("{\"prodId\":101,….}")).andReturn();

c. Here andReturn() submits HttpRequest above one looks like.

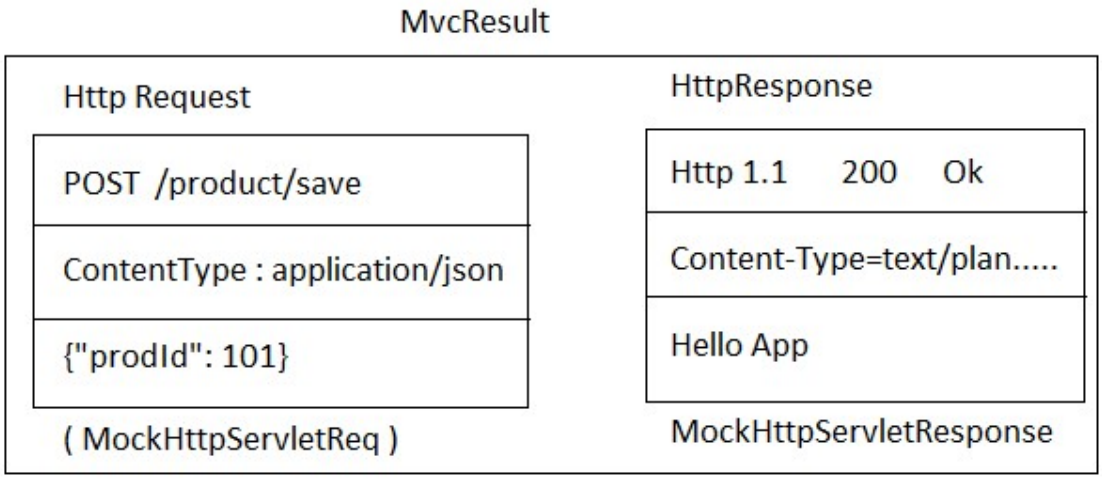


d. Both Request and Response objects are stored in "MvcResult" as,

a. MockHttpServletRequest

b. MockHttpServletResponse.

It looks like



e. Enable this Mock and Test process using Annotations :

@RunWith(SpringRunner.class)
@WebMvcTest

Code:-- RestController:--

```
package com.app.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(name="/emp")
public class EmployeeRestController {

	@GetMapping("/data")
	public String getData() {
		return "Hello";
	}
}
```

Code:- Test class:--

```
package com.app;
import static org.junit.Assert.assertEquals;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.get;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

@RunWith(SpringRunner.class)
@WebMvcTest
@SpringBootTest
public class JUnitMockitoApplicationTests {
```

```
	@Autowired
	private MockMvc mockMvc;

	@Test
	public void testEmpData() throws Exception {
		MvcResult result=mockMvc.perform(get("/emp/data")).andReturn();
		//MockHttpServletRequest req =result.getRequest();
		MockHttpServletResponse  resp =result.getResponse();
		assertEquals("Hi", resp.getContentAsString());
	}
}
```

=>Right click => Run as => JUnit Test

Spring Boot with JUnit and Mockito:--

##Testing : ReST CURD Application ##

=>To implement Unit Testing, we need to follow 4 steps. Given below,

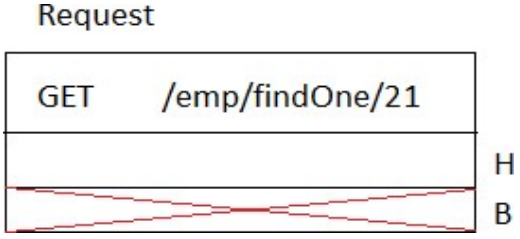
1. Construct Http Request using **RequestBuilders**.
2. Execute Http Request using **MockMvc**
3. Store Response along with Request in **MvcResult**.
4. Use assert API (JUnit) to verify actual details with expected values.

Step#1:- To construct Request Object use RequestBuilder and call static method (Http Method Type).

MockMvcRequestBuilders.post(…)….;

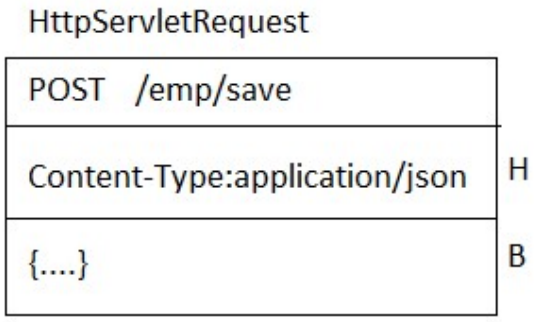
=>It returns HttpServletRequest object.
MockMvcServletRequestBuilder

Ex#1 (GET):-



```
MockHttpServletRequestBuilder request =
MockMvcRequestBuilders.get ("/emp/findOne/21");
```

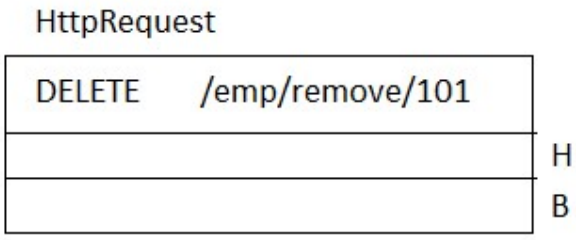
Ex#2 (POST):--



=>Equal code of above Diagram is

```
MockHttpServletRequestBuilder  request = MockMvcRequestBuilders
        .post("/emp/save")
        .header("Content-Type", "application/json")
        //.contentType("application/json")
        //contentType(MediaType.APPLICATION_JSON)
        .content("{...}");
```

Ex#3 (DELETE):-



=>Equal code

```
MockHttpServletRequestBuilder  request =
MockMvcRequestBuilders.delete("/emp/remove/101");
```

EX#4 (PUT):-

HttpRequest

PUT /emp/update	
Content-Type: application/xml	H
<emp> </emp>	B

=>Equal code

```
MockHttpServletRequestBuilder  request = MockMvcRequestBuilders
        .post("/emp/save")
        .contentType("application/xml")
        .content("<emp>...</emp");
```

Step#2:- Execute Request call using MockMvc.

```
MvcResult result = mockMvc.perform(request).andReturn();
```

Step#3:- Get Request/Response Object from Mvc.

```
MockHttpServletRequest req = result.getRequest();
MockHttpServletResponse  resp = result.getResponse();
```

Project Creation step by Step:--

Step#1:- Define one CURD application using one database and RestController.

Step#2:- Check all operations using POSTMAN Screen.

Step#3:- Define one test profile for properties or yml.
=>application-test.properties

Step#4:- Define one class under src/test/java folder and apply annotations.

Step#5:- Apply below annotations over test class.

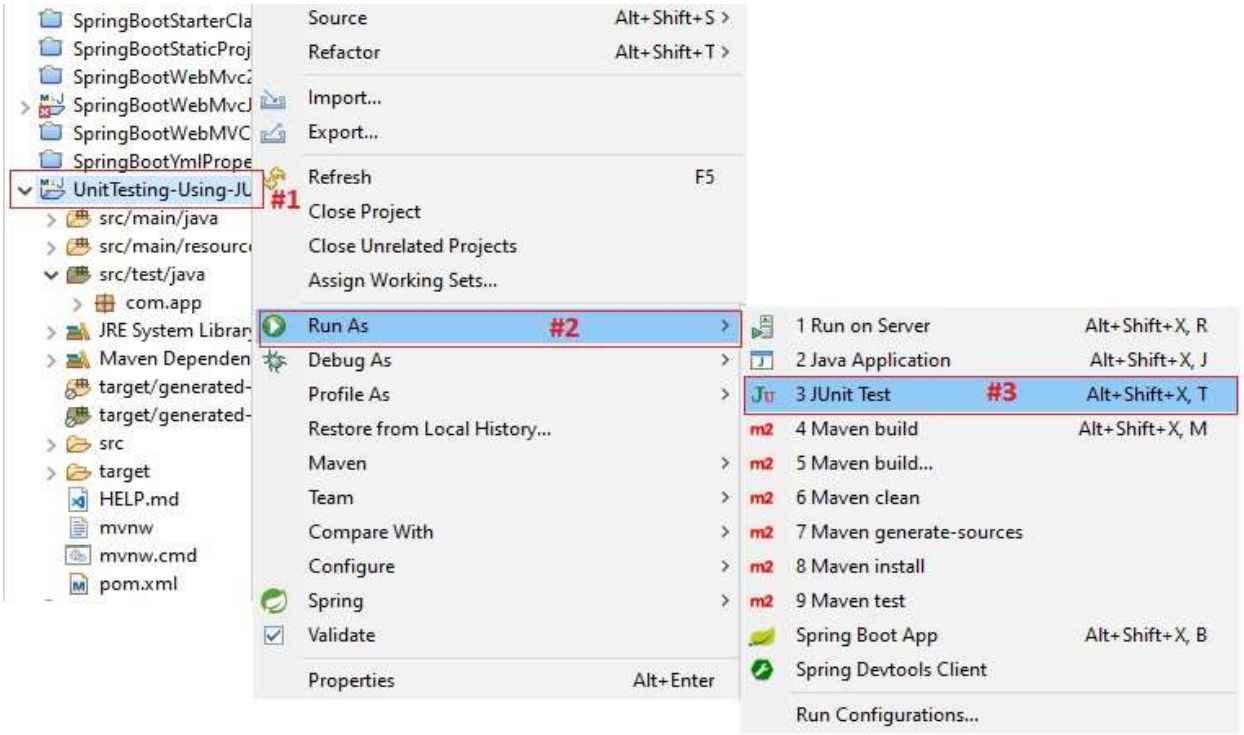
```
@SpringBootTest(webEnvironment=WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestPropertySource("classpath:application-test.properties")
```

Step#6:- Use MockMvc dependency (autowire) in Test class.

Step#7:- Define @Test methods under Test class.

Step#8:- Run Test class using JUnit Test.

=>Right click on Project > Run As > JUnit Test.



NOTE:--

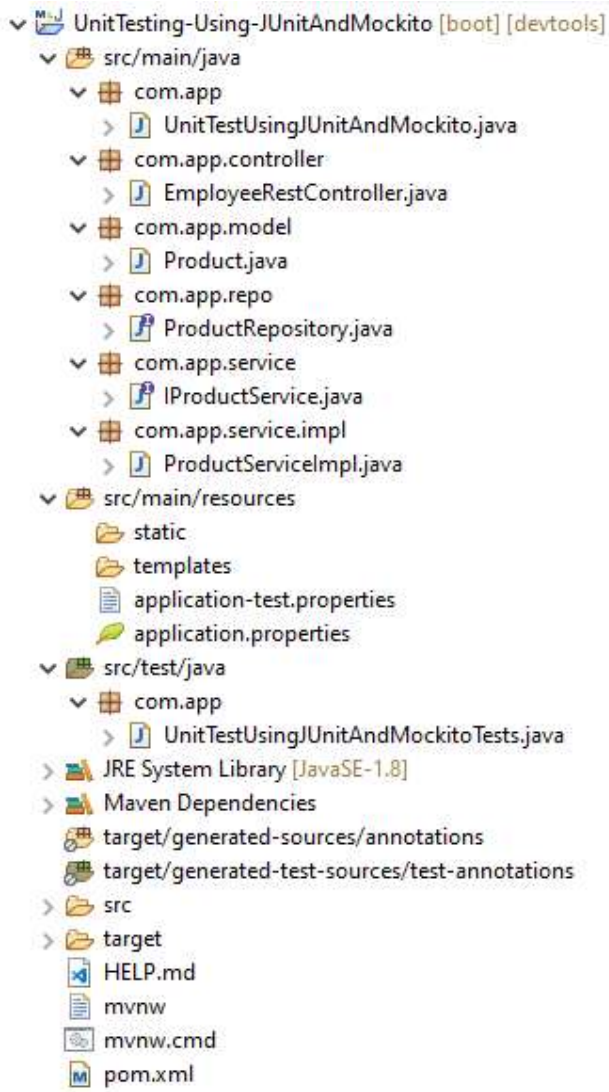
1>@TestPropertySource:- It is used to load properties/yml file into UnitTest

2>@AutoConfigureMockMvc:- It Define all Mock Beans related to environment (Ex: Datasource, ConnectionPool, Cache…).

3>@SpringBootTest:- It define beans and injects them based on relations (Objects for: RestControllers, Services, Repos…etc).

4>@WebMvcTest:- Works only for @RestControllers without service and other dependencies.

1. Folder Structure of JUnit + Mockito Testing:--



pom.xml:--

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.1.8.RELEASE</version>
                <relativePath /> <!-- lookup parent from repository -->
```

```
	</parent>
	<groupId>com.app</groupId>
	<artifactId>UnitTesting-Using-JUnitAndMockito</artifactId>
	<version>1.0</version>
	<name>UnitTesting-Using-JUnitAndMockito</name>
	<description>Spring Boot Test Application</description>

	<properties>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<!-- <optional>true</optional> -->
			<scope>provided</scope>
		</dependency>
		<dependency>
			<groupId>com.oracle</groupId>
			<artifactId>ojdbc6</artifactId>
			<version>11.2.0</version>
		</dependency>
		<!-- <dependency>
			<groupId>org.mockito</groupId>
			<artifactId>mockito-all</artifactId>
			<scope>test</scope>
		</dependency> -->
```

```
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>
        </dependencies>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
</project>
```

Coding Step by Step:--

Step#1: In "application.properties" & "application-test.properties" add bellow code:-

```
server.port=2019
##DataSource##
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=system
#Spring Data JPA
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.open-in-view=true
```

Step#2: Model class (Product.class):--

```
package com.app.model;
import  javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;

@Entity
@Data
@Table
public class Product {

	@Id
	@GeneratedValue
	private Integer prodId;
	private String prodCode;
	private Double prodCost;
	private String vendorCode;
}
```

Step#3: Repository Interface:--

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.app.model.Product;

public interface ProductRepository extends JpaRepository<Product, Integer> { }
```

Step#4: Service Interface:--

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface IProductService  {

	public Integer saveProduct(Product p);
```

```
	public void deleteProduct(Integer prodId);
	public Product getProductById(Integer prodId);
	public List<Product> getAllProducts();
	public boolean isProductExist(Integer id);
}
```

Step#5: ServiceImpl class:--

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.app.model.Product;
import com.app.repo.ProductRepository;
import com.app.service.IProductService;

@Service
public class ProductServiceImpl implements IProductService{

	@Autowired
	private ProductRepository repo;

	public Integer saveProduct(Product p) {
		p=repo.save(p);
		Integer prodId=p.getProdId();
		return prodId;
	}

	public void deleteProduct(Integer prodId) {
		repo.deleteById(prodId);
	}

	public Product getProductById(Integer prodId) {
		Optional<Product> p=repo.findById(prodId);
		if(p.isPresent()) {
			return p.get();
```

```
		}else {
			return new Product();
		}
	}
	public List<Product> getAllProducts() {
		List<Product> prods=repo.findAll();
		return prods;
	}
	@Override
	public boolean isProductExist(Integer id) {
		return repo.existsById(id);
	}
}
```

Step#6: RestController:--

```
package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Product;
import com.app.service.IProductService;

@RestController
@RequestMapping("/product")
public class EmployeeRestController {

	@Autowired
	private IProductService service;
```

```
//1. Post Method
    @PostMapping("/register")
    public ResponseEntity<?> saveProduct(@RequestBody Product product) {
        ResponseEntity<?> response=null;
        try {
            Integer stdId=service.saveProduct(product);
      response = new ResponseEntity<String>(stdId+"-Inserted", HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
      response = new responseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return response;
    }
//2. Get Method
    @GetMapping("/get/{id}")
    public ResponseEntity<?> showOneProducts(@PathVariable Integer id) {
        ResponseEntity<?> response=null;
        boolean exist=service.isProductExist(id);
        if(exist) {
            Product s=service.getProductById(id);
            response=new ResponseEntity<Product>(s, HttpStatus.OK);
        }  else {
            response=new ResponseEntity<>(HttpStatus.NO_CONTENT);
        }
        return response;
    }

//3. Get Method for all data
    @GetMapping("/all")
    public ResponseEntity<?> showAllProducts() {
        ResponseEntity<?> response=null;
        List<Product> Products=service.getAllProducts();
        if(Products!=null && !Products.isEmpty()) {
     response=new ResponseEntity<List<Product>>(Products, HttpStatus.OK);
        } else {
            response=new ResponseEntity<>(HttpStatus.NO_CONTENT);
```

```
		}
		return response;
	}

//4. Delete Method
	@DeleteMapping("/delete/{id}")
	public ResponseEntity<?> deleteProduct(@PathVariable Integer id) {
		ResponseEntity<?> response=null;
		boolean exist=service.isProductExist(id);
		if(exist) {
			service.deleteProduct(id);
	response=new ResponseEntity<String>(id+"-Removed", HttpStatus.OK);
		} else {
		   response=new ResponseEntity<String>("Product NOT FOUND",
				HttpStatus.BAD_REQUEST);
		}
		return response;
	}

//5. Edit method
	@PutMapping("/edit")
	public ResponseEntity<?> editProduct(@RequestBody Product product) {
		ResponseEntity<?> response=null;
		Integer id=product.getProdId();
		boolean exist=service.isProductExist(id);
		if(exist) {
			service.saveProduct(product);
	response = new ResponseEntity<String>(id+"-Updated", HttpStatus.OK);
		}else {
		  response = new ResponseEntity<String>("Product NOT FOUND",
					HttpStatus.BAD_REQUEST);
		}
		return response;
	}
}
```

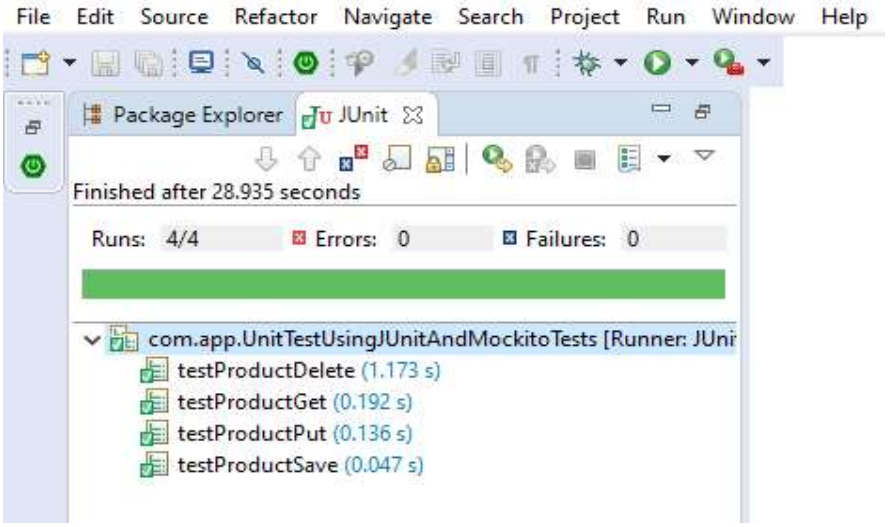
Step#7:JunitAndMockitoTests class:--

```
package com.app;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.delete;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.request.
MockMvcRequestBuilders.put;

import org.junit.Test;
import org.junit.runner.RunWith;
import      org.springframework.beans.factory.annotation.Autowired;
import  org.springframework.boot.test.autoconfigure.web.servlet.
AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockHttpServletResponse;
import org.springframework.test.context.TestPropertySource;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;

@RunWith(SpringRunner.class)
//@WebMvcTest
@SpringBootTest(webEnvironment= SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestPropertySource(locations = "classpath:application-test.properties")
public class UnitTestUsingJUnitAndMockitoTests {

        @Autowired
        private MockMvc mockMvc;
```

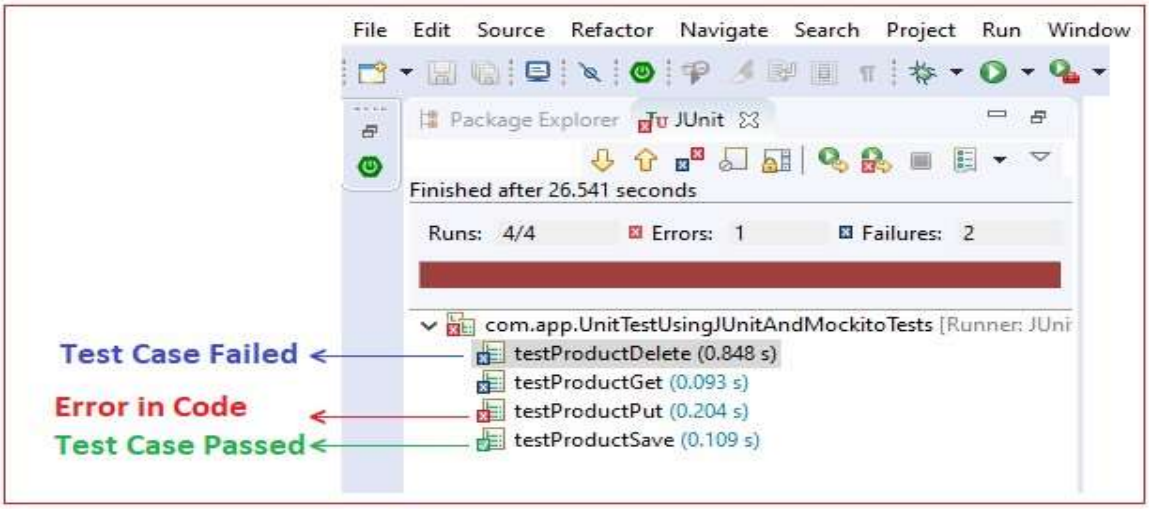
```
//1. Post Method case
        @Test
        public void testProductSave() throws Exception {
            MvcResult result = mockMvc.perform(post("/product/register")
                .contentType(MediaType.APPLICATION_JSON)
.content("{\"prodCode\":\"ABCD\",\"prodCost\":88.55,\"vendorCode\": \"V11\"}"))
                .andReturn();
            MockHttpServletResponse resp = result.getResponse();
            assertEquals(HttpStatus.OK.value(), resp.getStatus());
            System.out.println(resp.getContentAsString());
            assertNotNull(resp.getContentAsString());
        }

//3. Put Method Test Case
                @Test
                public void testProductPut()throws Exception {
                    MvcResult result=mockMvc.perform(put("/product/edit")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content("{\"prodId\":1, \"prodCode\":\"ABCDE\",\"
                         prodCost\":38.55,\"vendorCode\": \"V14\"}")).andReturn();
                MockHttpServletResponse resp=result.getResponse();
                assertEquals(HttpStatus.OK.value(), resp.getStatus());
                System.out.println(resp.getContentAsString());
                assertNotNull(resp.getContentAsString());
                }

//2. Get Method Test Case
                @Test
                public void testProductGet() throws Exception {
        MvcResult result = mockMvc.perform(get("/product/get/4")).andReturn();
                        MockHttpServletResponse resp=result.getResponse();
                        assertEquals(HttpStatus.OK.value(), resp.getStatus());
                        assertNotNull(resp.getContentAsString());
                }
```

```
//4. Delete Method Test Case
        @Test
        public void testProductDelete()throws Exception {
MvcResult result=mockMvc.perform(delete("/product/delete/5")).andReturn();
            MockHttpServletResponse resp=result.getResponse();
            assertEquals(HttpStatus.OK.value(), resp.getStatus());
            System.out.println(resp.getContentAsString());
            assertNotNull(resp.getContentAsString());
        }
}
```

1>Output SCRENN Short of JUnit:--



2>Output Screen for all test cases.



NOTE:-- 1>Green Color(testProductSave) indicate Test cases passed.

2>Blue color(testProductDetele,testProductGet) indicates test cases failed.

3>Red color(testProductPut) indicate error in test class specific method code.

Gradle

Gradle:-- Gradle is a build tool. It works based on Groovey Language.
=>It gets Jars related to Project.
=>It will be connected to parent projects as plugin.
=>Provides Java Application support.
=>Connected to multiple server for jars fetching Ex:- Jcenter, mavenCentral etc.
=>Supports compile our file.
=>Creates .jar/.war file.

=>**Maven is XML based Build tool where Gradle is Language Script.
=>Both are used for same purpose, Gradle is faster a bit compared to Maven.

=>**Gradle is a task based Scripting. It means, for every work/setup we need to write one task. These are pre defined in Gradle.

Format looks like:

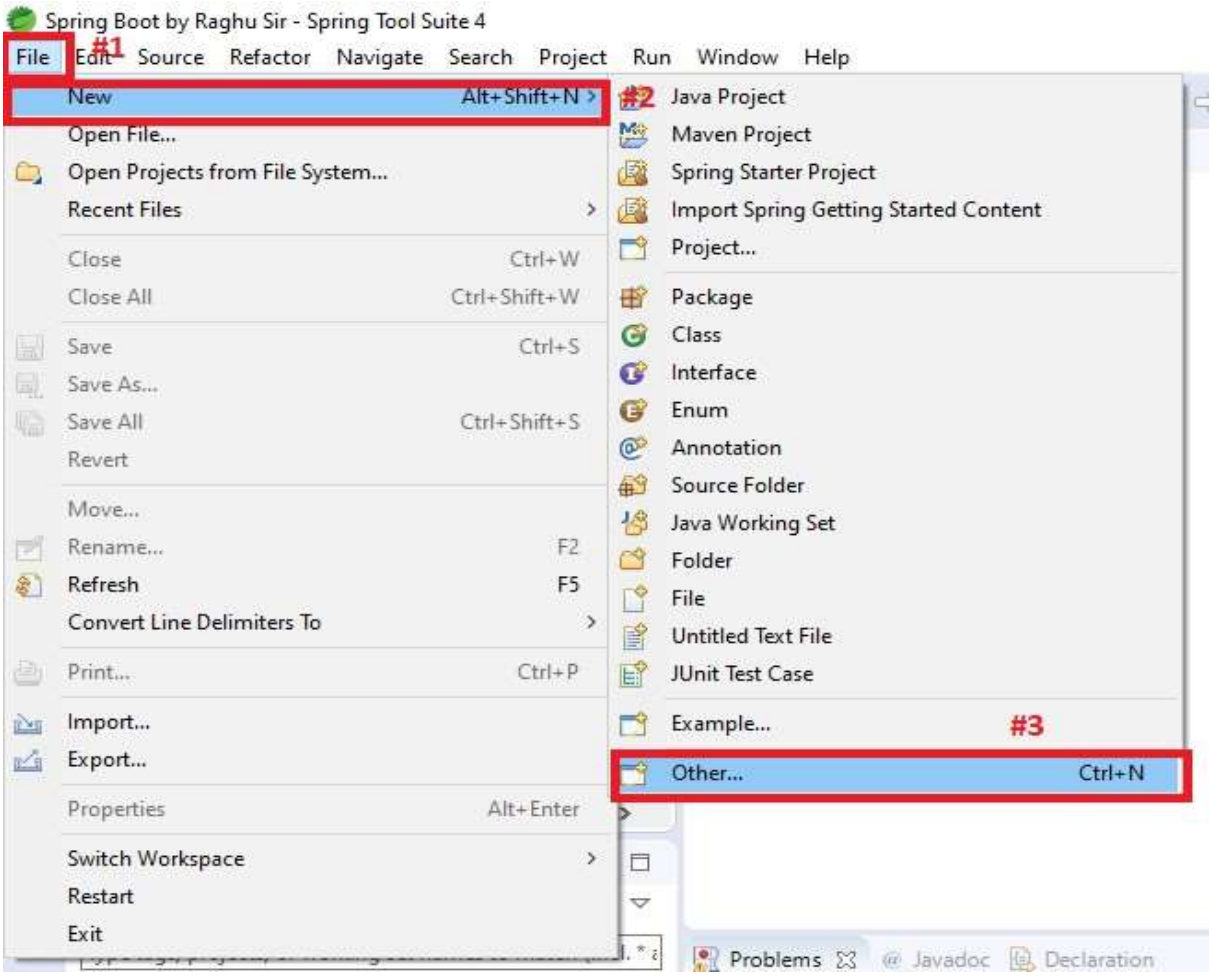
```
task {
    //details
}
```

-------Example Task-------
plugins{ ..... }
bootJar {.....}
repositories {.....}
dependencies { .....}

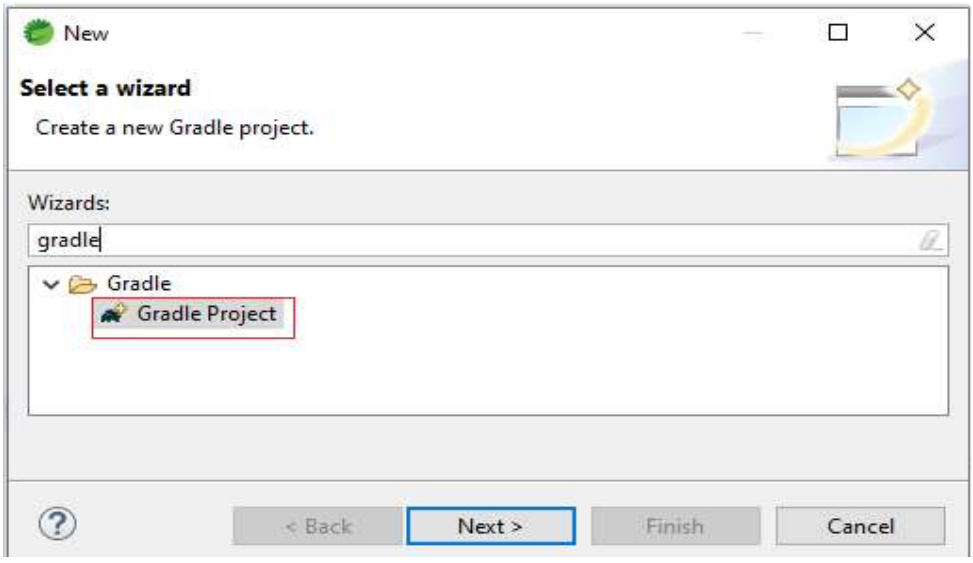
Gradle setup in STS 4:--
=> File -> New -> Other ->
=>Search using Gradle ->
=>Enter Project Name ->
=>Next/Finish.

1. Screen shot step by step to create Gradle Project:-

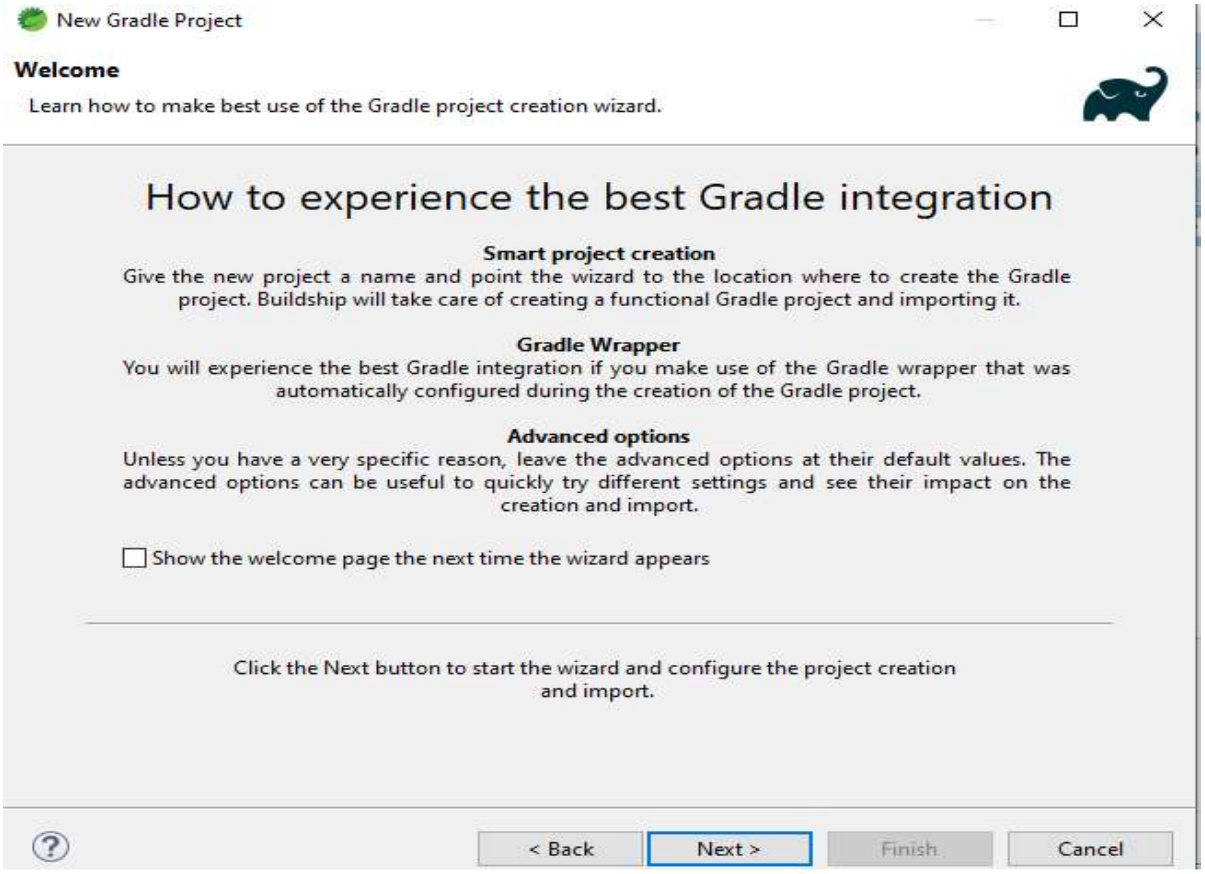
Step#1:- Click on File > New > Other



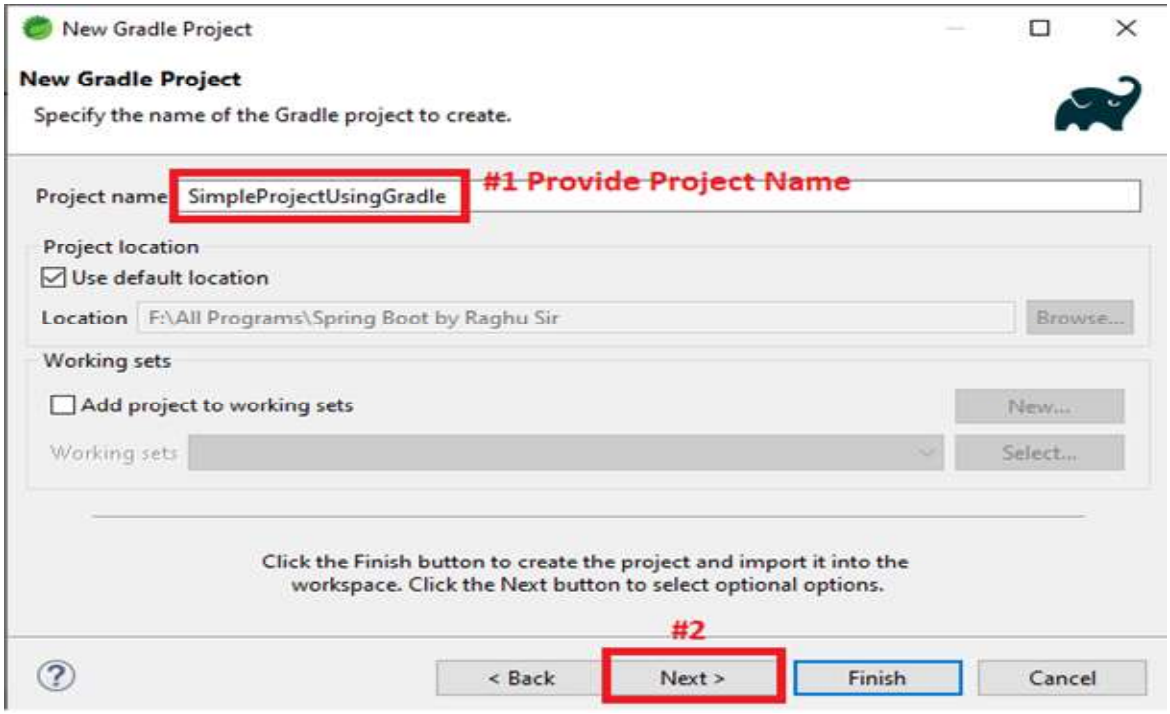
Step#2:- Search with Gradle > Select "Gradle Project" and click on Next.



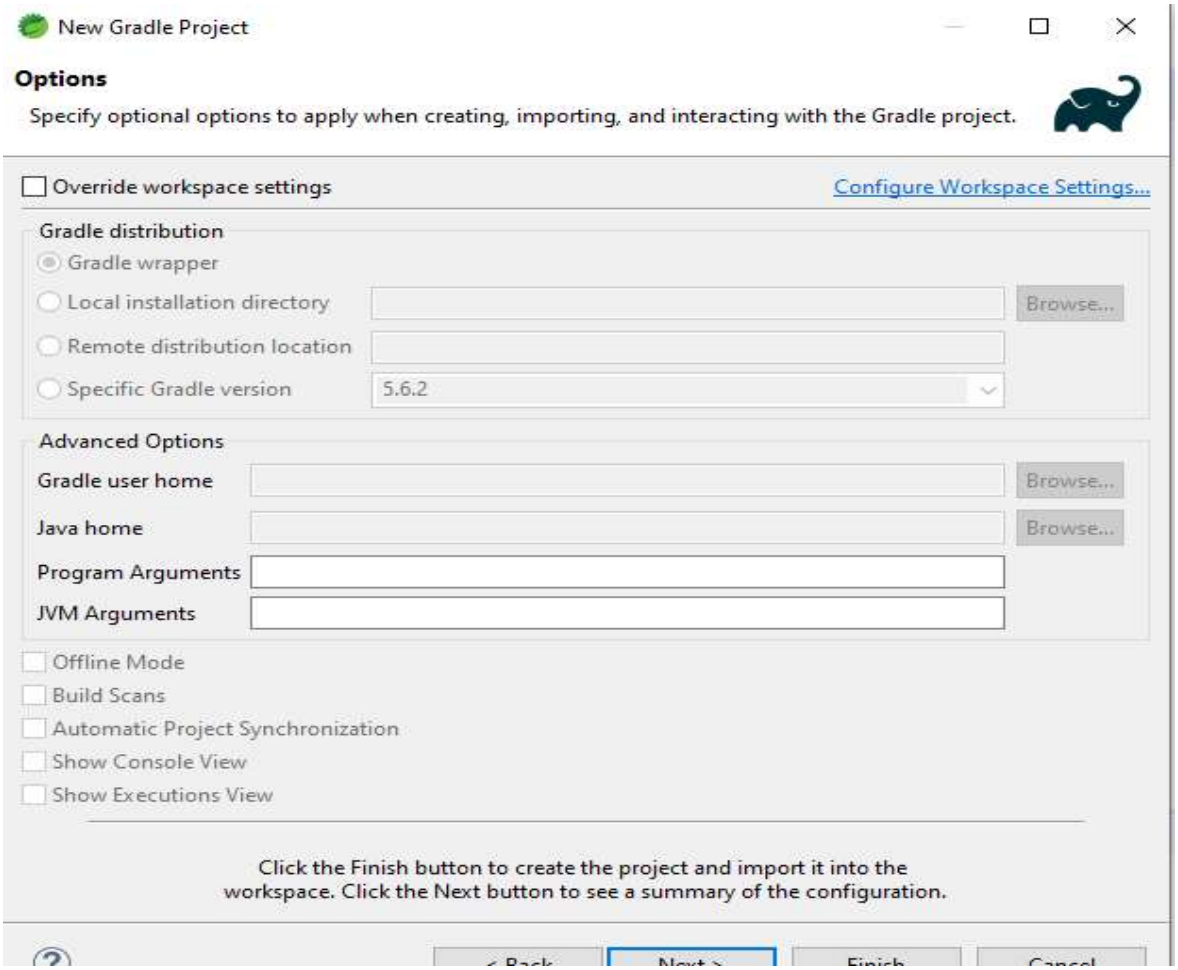
Step#3:- Click on Next.



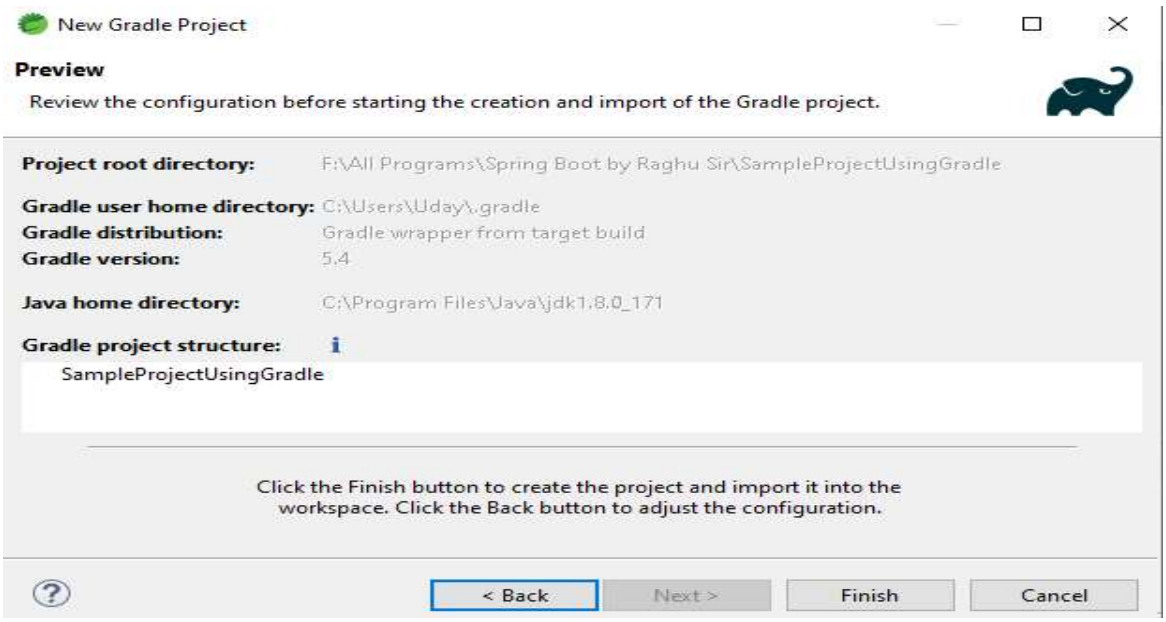
Step#4:- Provide Project Name and Click on Next/Finish.



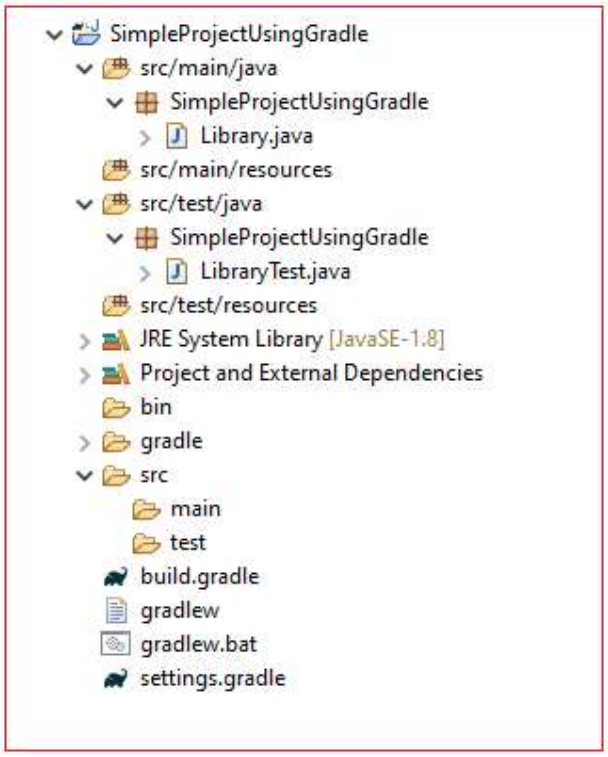
Step#5:- Click on Next.



Step#6:- Click on Next.

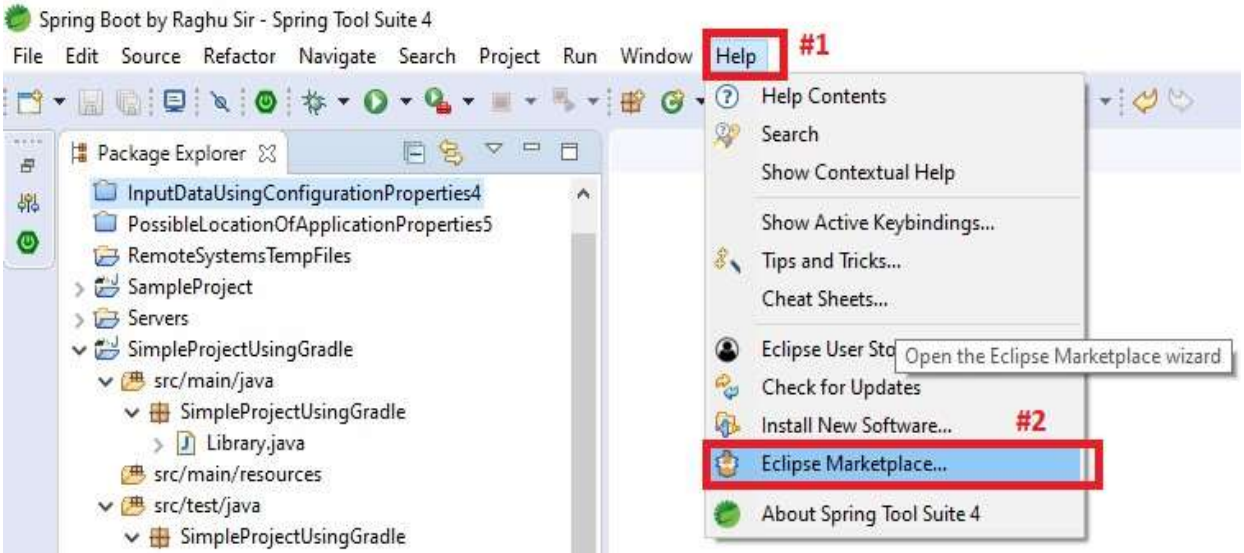


Step#7:- Click on Finish.
=>After click on finish Menu final project is created like below.

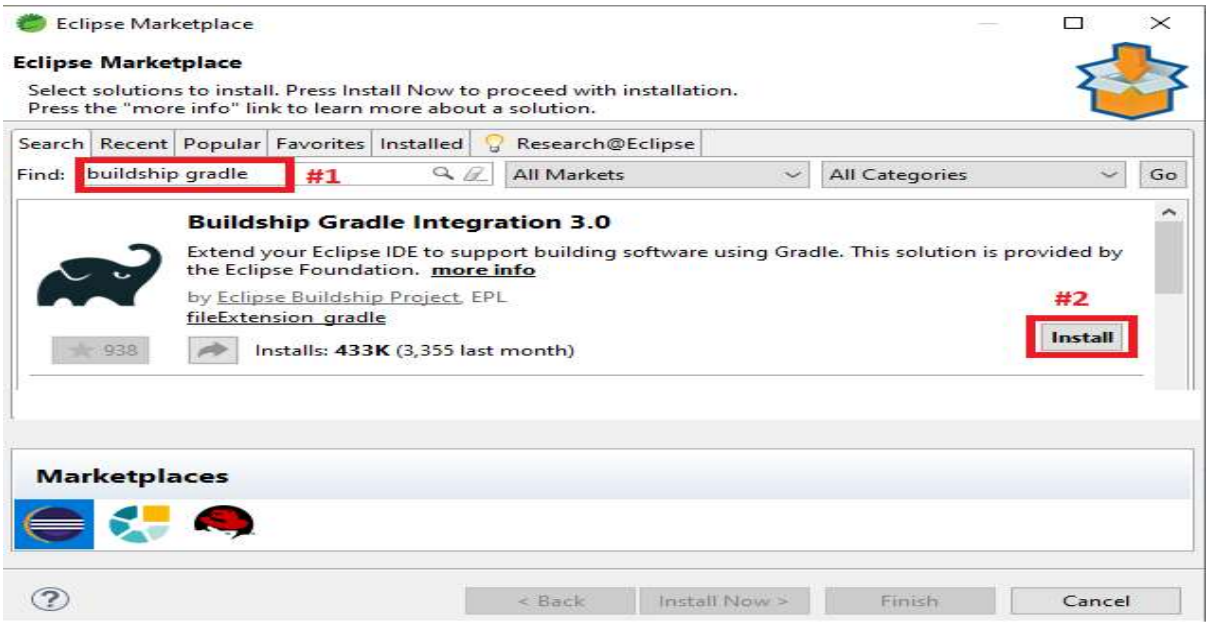


NOTE:--***If Gradle option is not available then goto Eclipse Market place and download “Buildship Gradle Integration”.

Step#1:- Goto Eclipse/STS help Menu and click on “Eclipse Marketplace”.



Step#2:- Search with **"Buildship Gradle"** and click on Install, After install restart the System.



=>Project is created using folder src/main/java, src/test/java … etc.
=>build.gradle is a file which contains all task details. Those are executed by gradlew (or gradlew.bat).

task: dependencies:-- This task is used to provide Jars related to project using one scope.

Dependency contains:--
Scope
Group
Name
Version

Ex:--
compile
group : 'org.springframework',
name : 'spring-context',
version : '5.1.8.RELEASE'

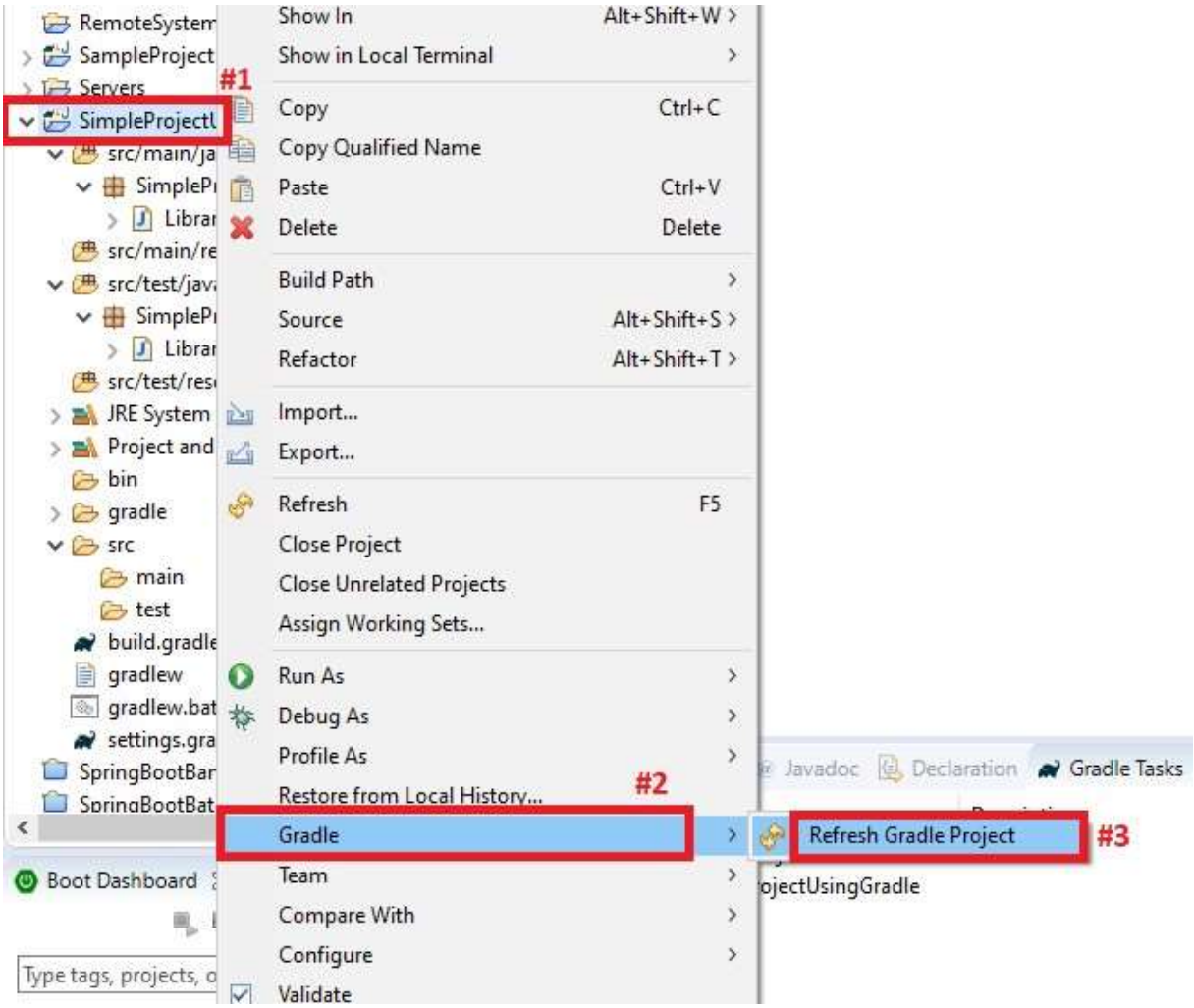
=>We can also write in shot format as
scope 'group:name:version'

Ex:-- testCompile 'junit:junit:4.10'

****Refresh Project (To get updated External dependencies contain(jar))**

=>Right click on Project > Gradle > Refresh Gradle Project

Screen Short:--



Scopes in Gradle:--

a. compile
b. testCompile
c. providedCompile
d. runtime
e. api //same like system in maven

a. compile:-- Jar will be available from Application compile time onwards.
b. testCompile:- Jar used at UnitTesting
c. providedCompile:- Jar given by 3rd party (server, f/w containers…).
d. runtime:- Jar used/loaded at runtime
e. api:- Jar used by Application API & documentation (loaded from system with high priority)

2. Configuring Gradle:-- You need to add the configuration for your application to become "WEB Application". And can be run directly on Eclipse + Tomcat Plugin.

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'eclipse-wtp'

repositories {
        jcenter()
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

dependencies {
        compile 'com.google.guava:guava:23.0'
        testCompile 'junit:junit:4.12'
        providedCompile "javax.servlet:javax.servlet-api:3.1.0"
}

dependencies {
        def tomcatVersion = '8.0.53'
        tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
        "org.apache.tomcat.embed:tomcat-embed-loggingjuli:${tomcatVersion}",
        "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
}

buildscript {
       repositories {
       jcenter()
 }

dependencies {
       classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'
    }
}
```

NOTE:-- Each time there is a change in build.gradle you need to update the project, using the tool of Gradle.
=>Right click on Project > Gradle > Refresh Gradle Project

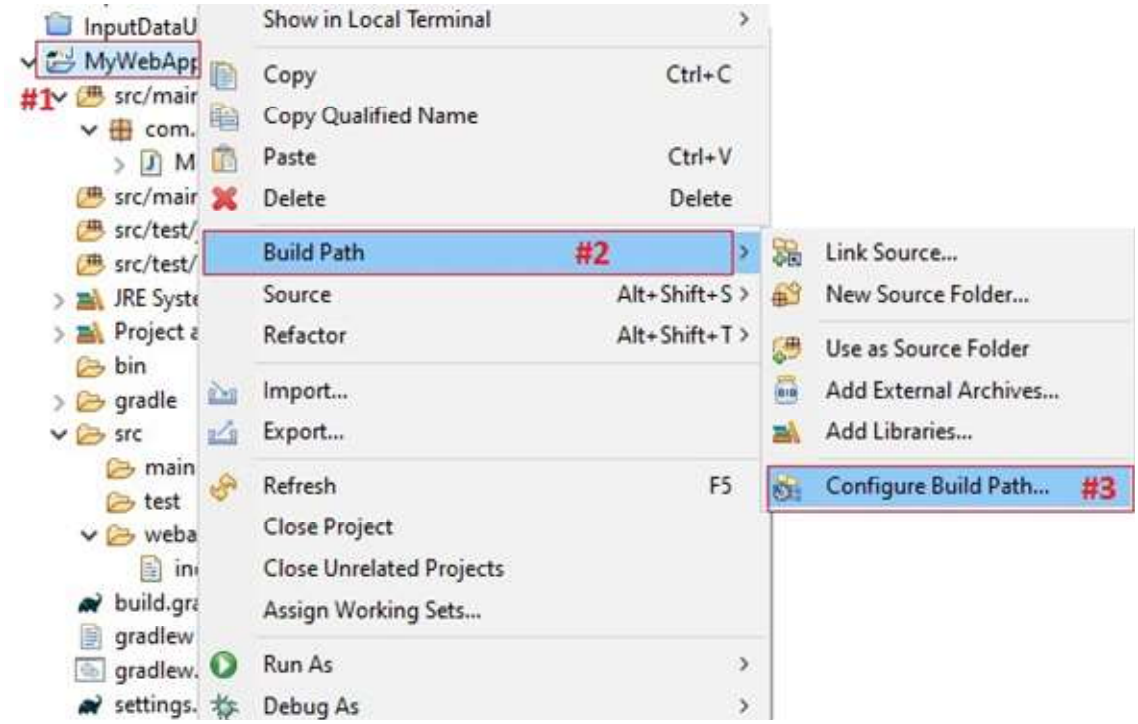
3. Add Folder To Application:--

->In "src/main" folder, you need to create 2 sub folders
->Those are "resources" and "webapp".
->src/main/java: This folder has java sources.
->src/main/resources: This folder can hold property files and other resources
->src/main/webapp: This folder holds jsp and other web application content.

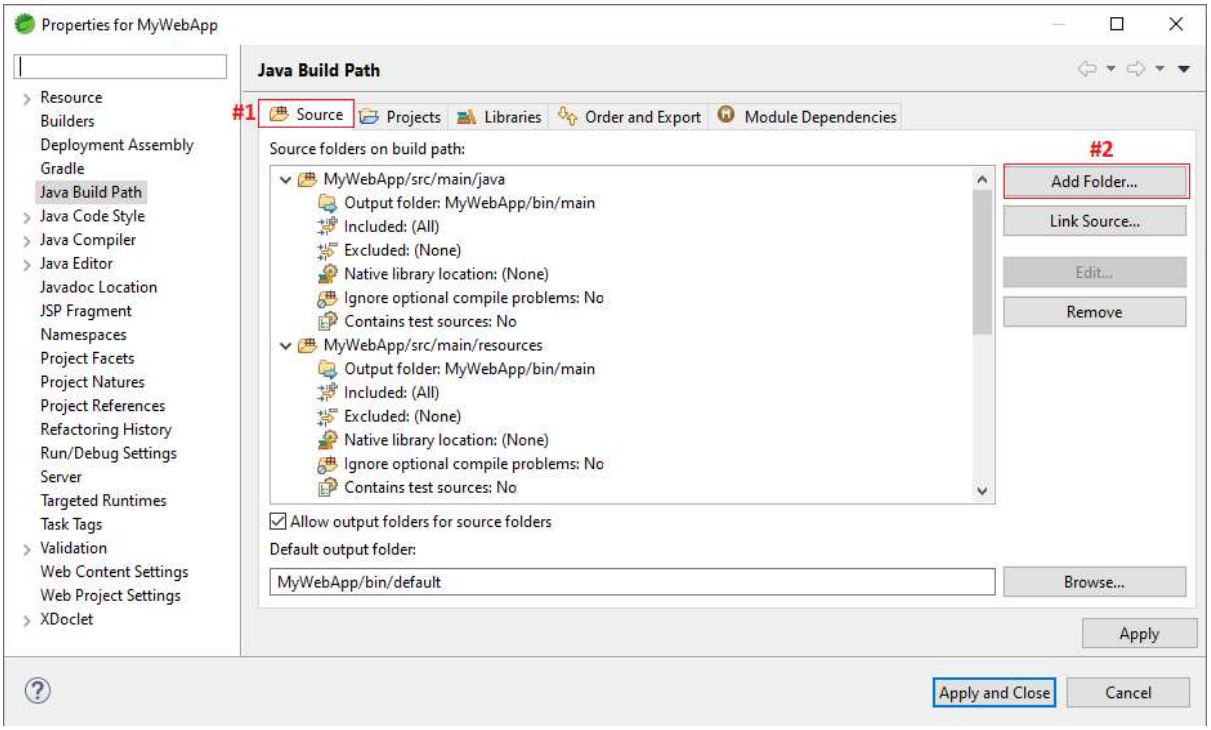
Create folder:--

=>Right click on Project > build path > Configure Build path >
->Source > add folder > new folder >
Enter name ex: src/main/resources -> finish.

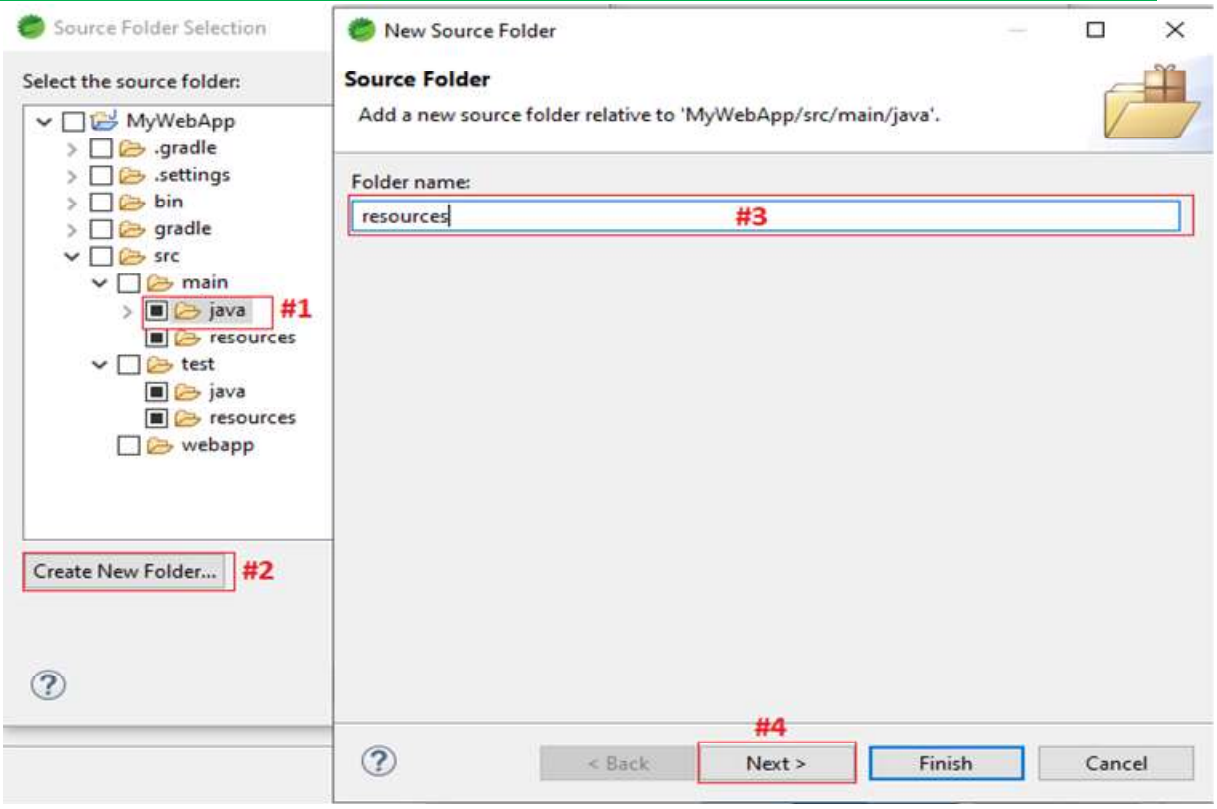
Screen Shot:--
Step#1:- Right click on Project > Build Path > Configure Build Path



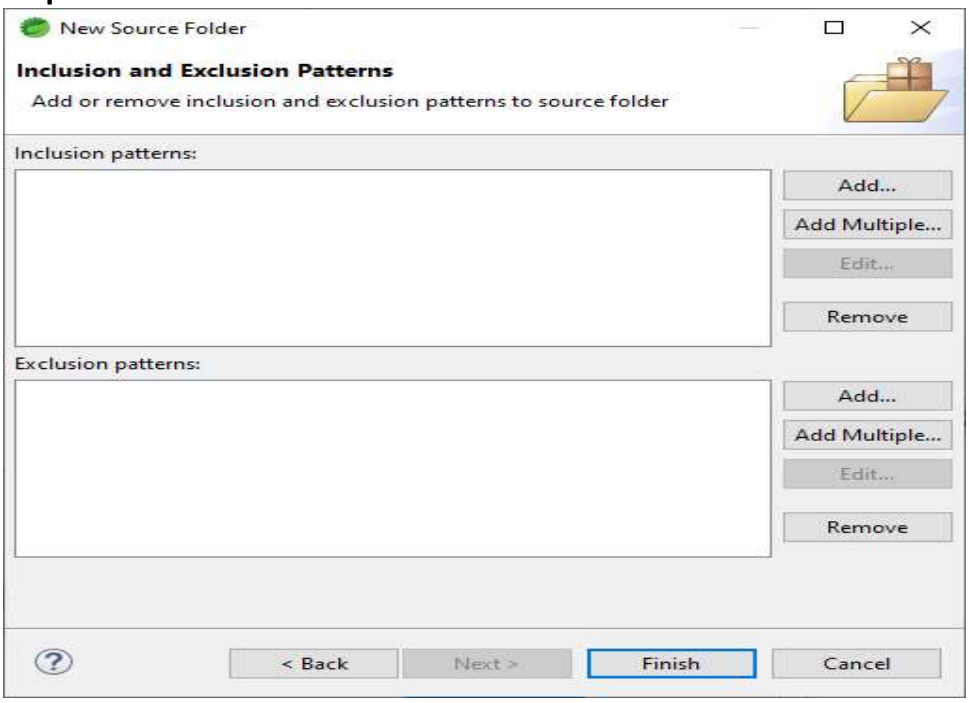
Step#2:- Click on Source Option > Click on "Add Folder".



Step#3:- Select folder in which you want to create folder > Click on Create New Folder > Enter Folder Name like (resources) > click on Next.



Step#4:- Click on Finish



4. Write Code:--

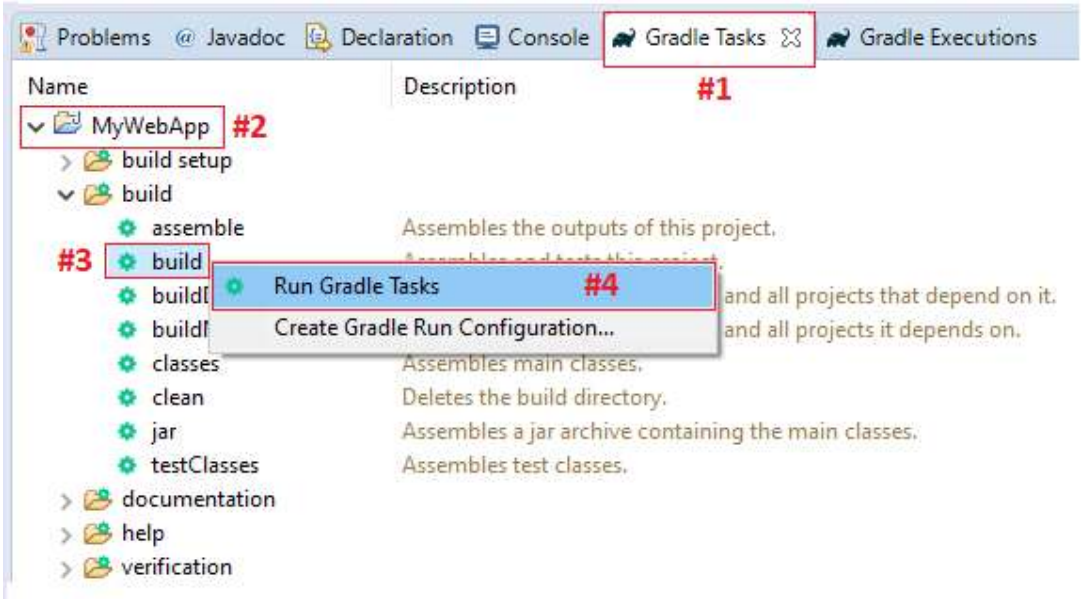
1>Service class, jsp, html etc..

5. Execute Gradle Build Task:--

=>Open “Gradle Task” view >

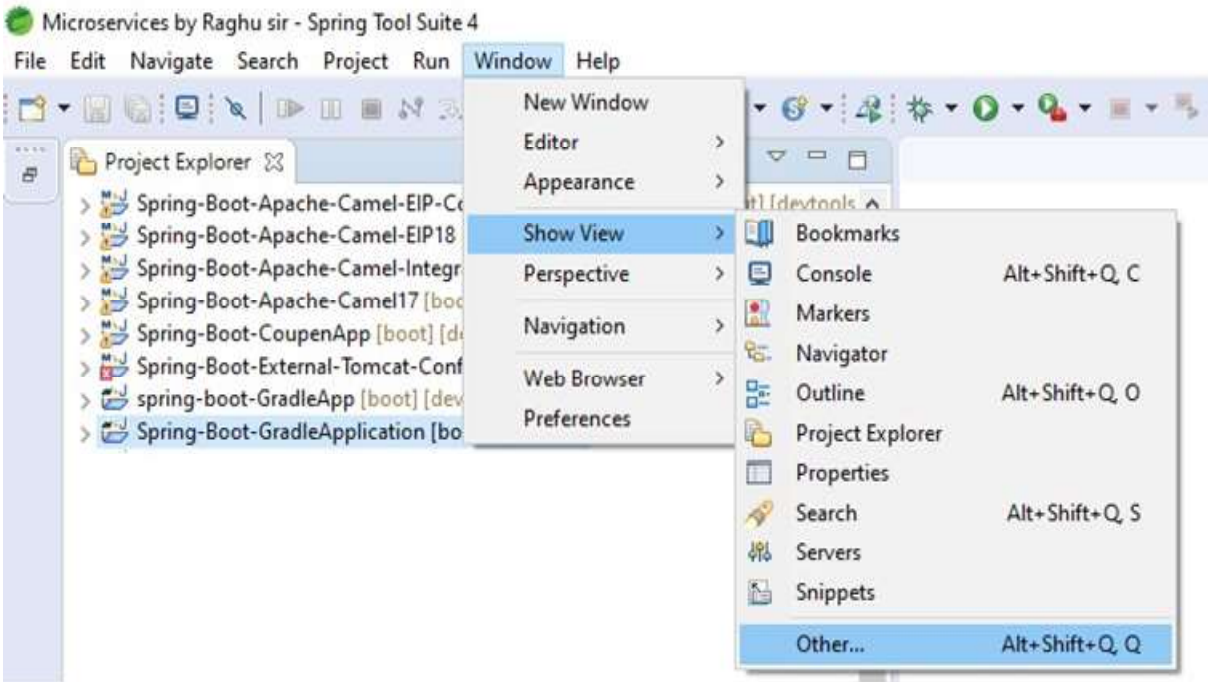
=>Expend Project > Choose build >
=>Right click on build > Run Gradle Tasks

Screen Shot:--

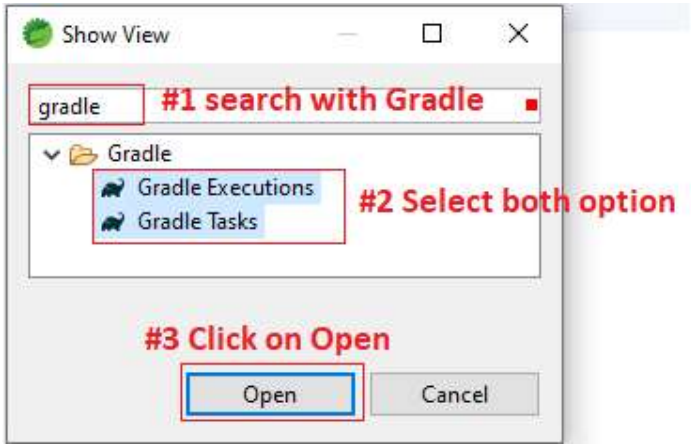


NOTE:-- If "Gradle Tasks" and "Gradle Execution" is not showing bellow then
=>Goto "Window" Menu > Show View > Other...>
=>Search with Gradle > Select both option >
=>Click on Open Button.

Screnn#1:-

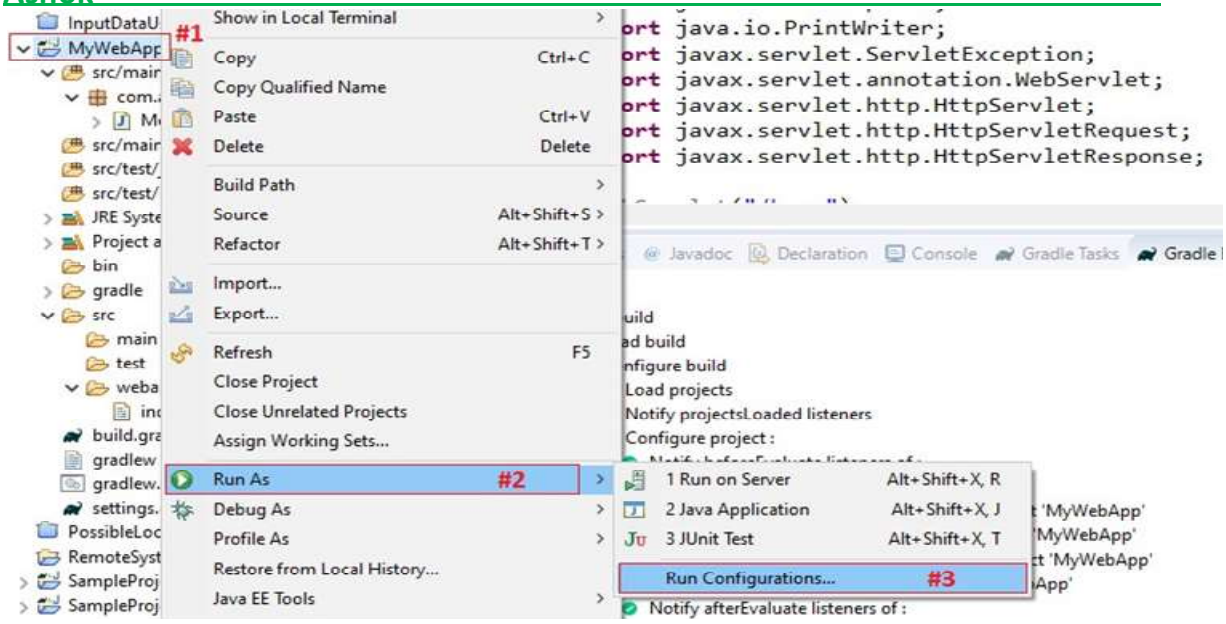


Screen#2:-

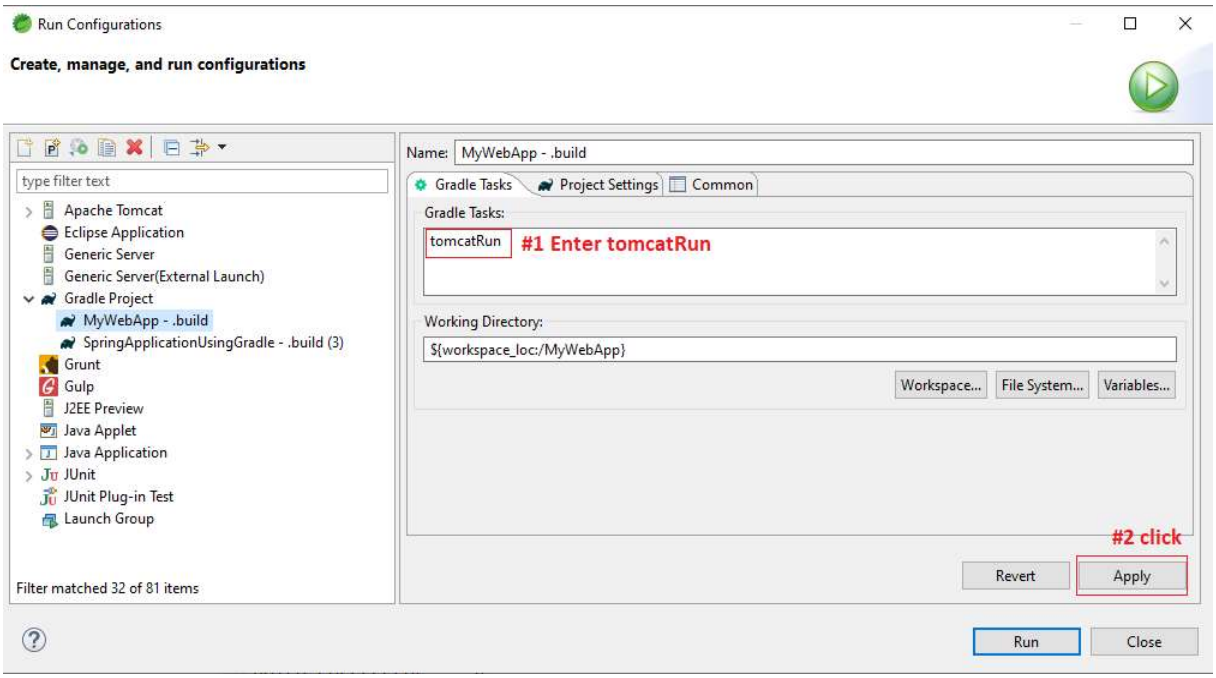


6. Configure To Run Application:--

Step#1:- Right click on Project > Run As > Run Configuration

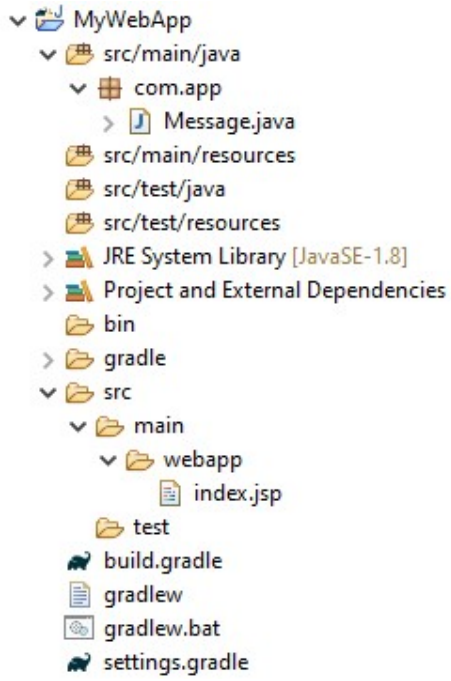


Step#2:- Enter > Apply > Run



1. SERVLETS/JSP WEB APPLICATION USING GRADLE:--

Folder Structure of Servlet Web App:--



Code:--

1. build.gradle:--

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'eclipse-wtp'

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
	//mavenCentral()
	jcenter()
}

dependencies {
providedCompile 'javax.servlet:javax.servlet-api:3.1.0'
compile 'org.springframework:spring-webmvc:5.1.8.RELEASE'
compile 'com.fasterxml.jackson.core:jackson-databind:2.9.5'
runtime 'javax.servlet:jstl:1.2'

def tomcatVersion = '8.0.53'

tomcat  "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
```

```
            "org.apache.tomcat.embed:tomcat-embed-logging-juli:${tomcatVersion}",
            "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
}

buildscript {
repositories {
      //mavenCentral()
      jcenter()
 }

dependencies {
     classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'
    }
 }
```

2. Servlet class (Message.java):--

```
package com.app;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/home")
public class Message extends HttpServlet {

        private static final long serialVersionUID = -4633811587840173475L;

        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                        throws ServletException, IOException {

                PrintWriter out=resp.getWriter();
                out.println("Hello App");
        }
}
```

3. index.jsp:--

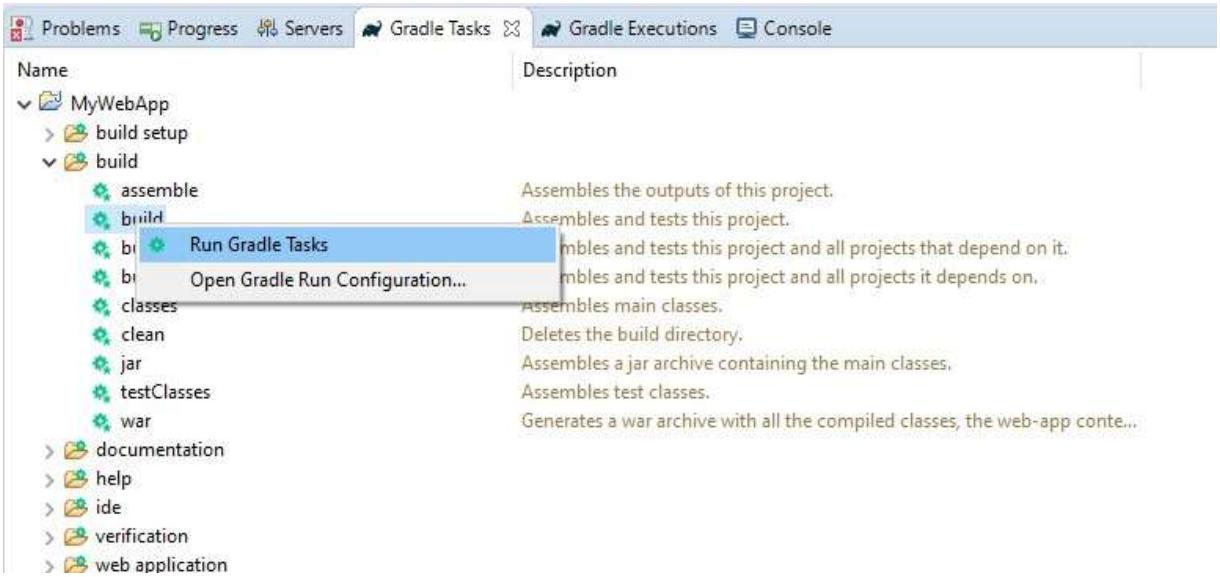
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to All</h1>
</body>
</html>
```

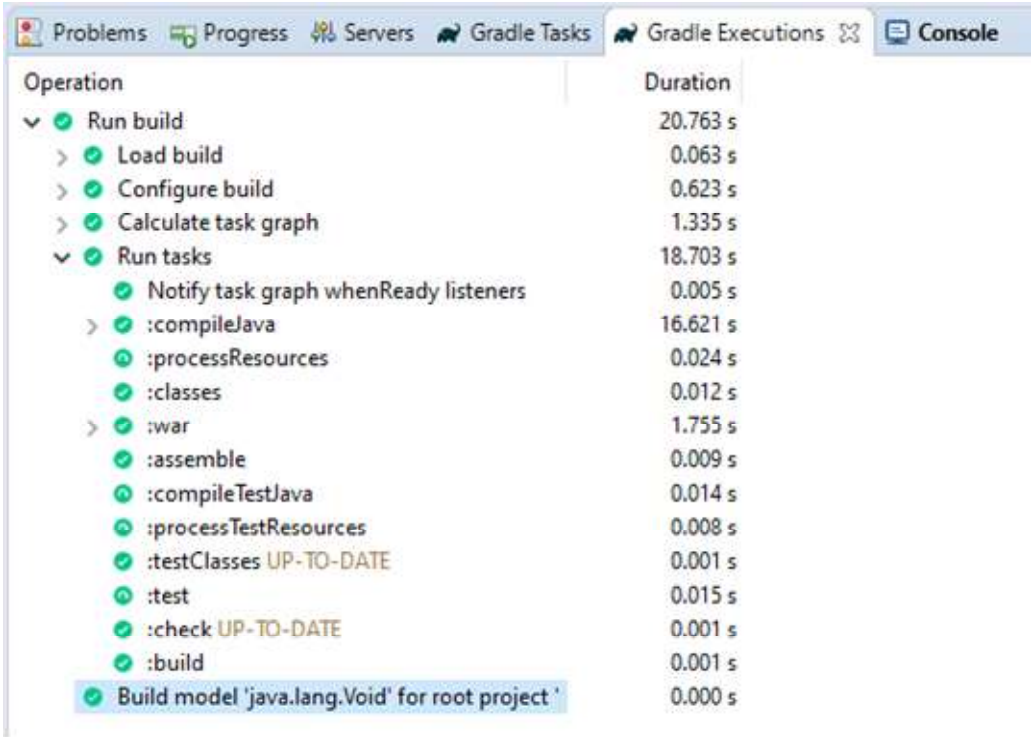
Execution flow:--

Step#1:- Execute Gradle build task

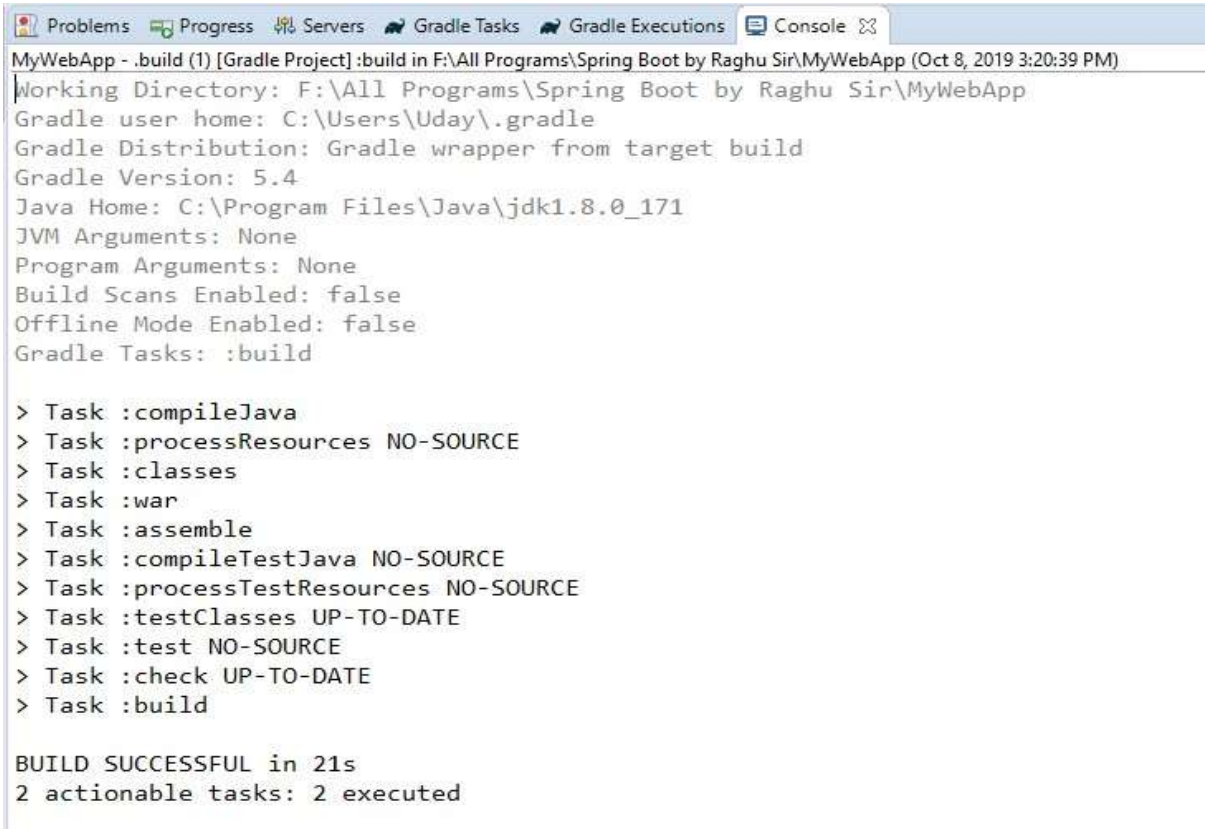
=>Open "Gradle Task" view > Expand Project > choose build > Run Gradle Tasks



=>See Success message here

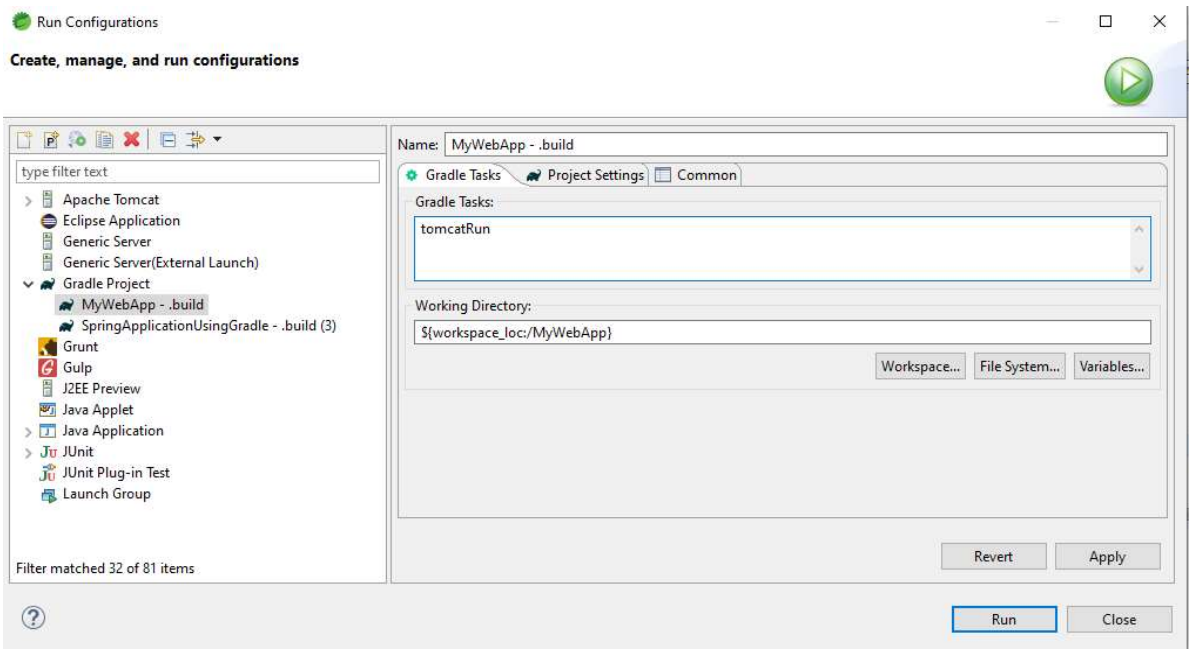


=>See Build success message on console.

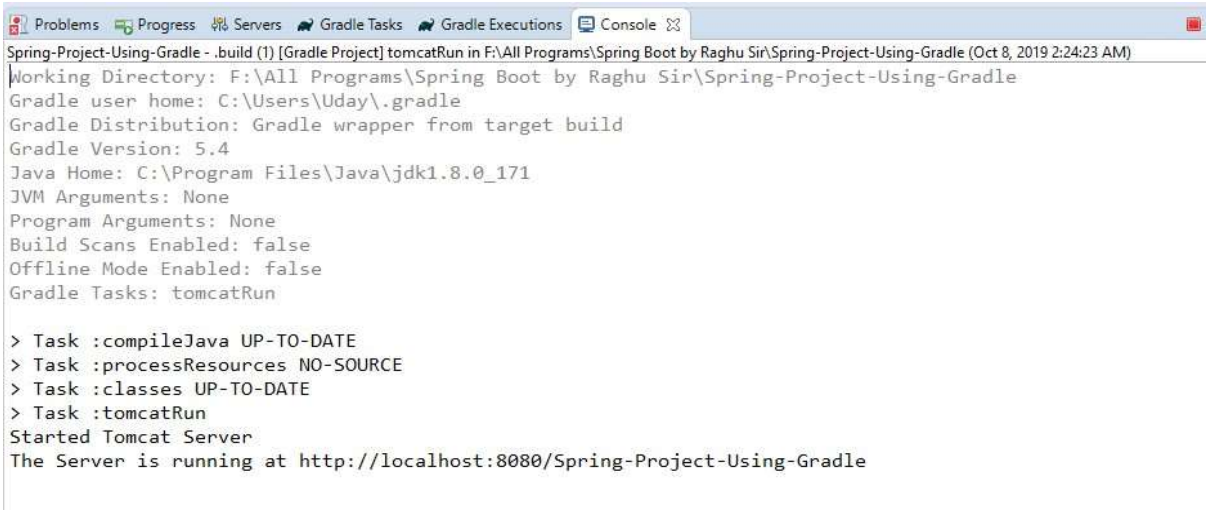


STEP#2:- RUN APPLICATION IN TOMCAT

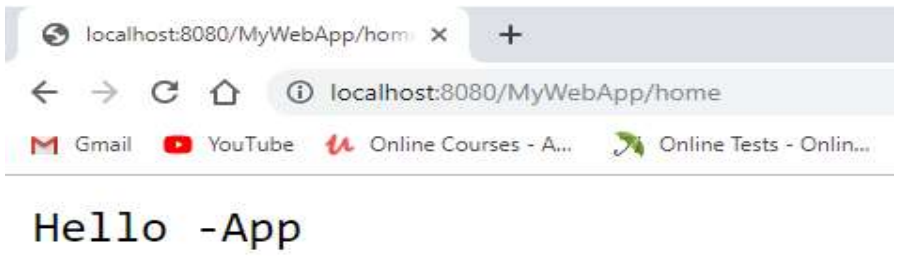
=> Right click on Project > Run As > enter tomcatRun > Apply and Run



=>Goto Console to See the confirmation on which port no Application is running



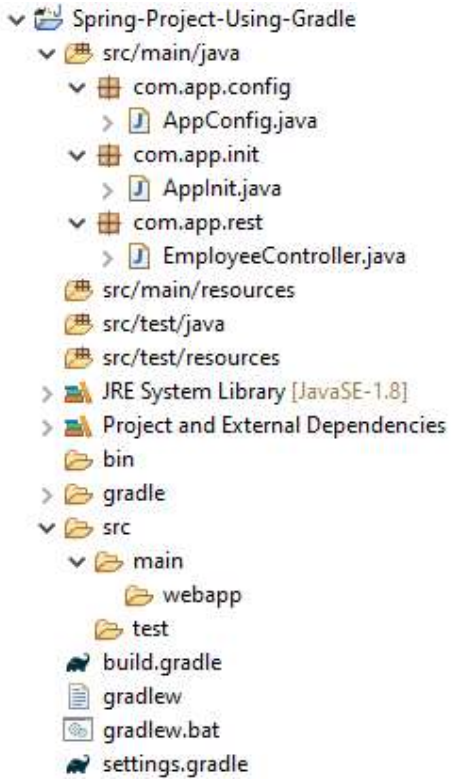
STEP#3:- ENTER URL IN BROWSER (http://localhost:8080/MyWebApp/home)



2. Spring Application Using Gradle:--

Step#1:- Create one Gradle Project
Ex:- Spring-Project-Using-Gradle

Folder Structure of Spring Application using Gradle:--



Step#2:- open build.gradle file and type below content

```
apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'
apply plugin: 'eclipse-wtp'

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
      mavenCentral()
      //jcenter()
}
 dependencies  {
```

```
  providedCompile 'javax.servlet:javax.servlet-api:3.0.1'
  compile 'org.springframework:spring-webmvc:5.1.8.RELEASE'
  compile 'com.fasterxml.jackson.core:jackson-databind:2.9.5'
  runtime 'javax.servlet:jstl:1.2'

  def tomcatVersion = '8.0.53'

  tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
       "org.apache.tomcat.embed:tomcat-embed-logging-juli:${tomcatVersion}",
       "org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}"
}

 buildscript {
     repositories {
         mavenCentral()
         //jcenter()
     }

 dependencies {
      classpath 'com.bmuschko:gradle-tomcat-plugin:2.5'
   }
 }
```

Step#3:- AppConfig class

```
package com.app.config;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import     org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration
@EnableWebMvc
@ComponentScan("com.app")
public class AppConfig {

}
```

Step#4:- AppInit class

```
package com.app.init;
import org.springframework.web.servlet.support.
AbstractAnnotationConfigDispatcherServletInitializer;
import com.app.config.AppConfig;
```

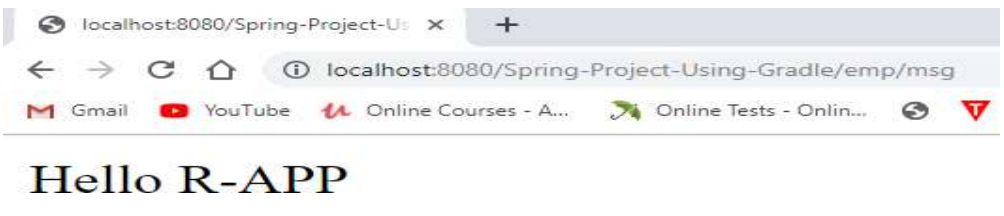
```
public class AppInit extends AbstractAnnotationConfigDispatcherServletInitializer {

	@Override
	protected Class<?>[] getRootConfigClasses() {
		return null;
	}
	@Override
	protected Class<?>[] getServletConfigClasses() {
		return new Class[] {AppConfig.class};
	}
	@Override
	protected String[] getServletMappings() {
		return new String[] {"/*"};
		}
}
```

Step#5:- Controller class

```
package com.app.rest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/emp")
public class EmployeeController  {

	@GetMapping("/msg")
	public String show()  {
		return "Hello R-APP";
	}
}
```

Execution flow:--

Step#1:- Execute Gradle build task
Step#2:- Run application in tomcat
Step#3:- Enter URL in Browser
http://localhost:8080/Spring-Project-Using-Gradle/emp/msg



3. Spring Boot Application Using Gradle:--

=>In spring boot Gradle Application two plugins are provided those are

a. Parent Project plugin

b. BOM (Bill of Managements)

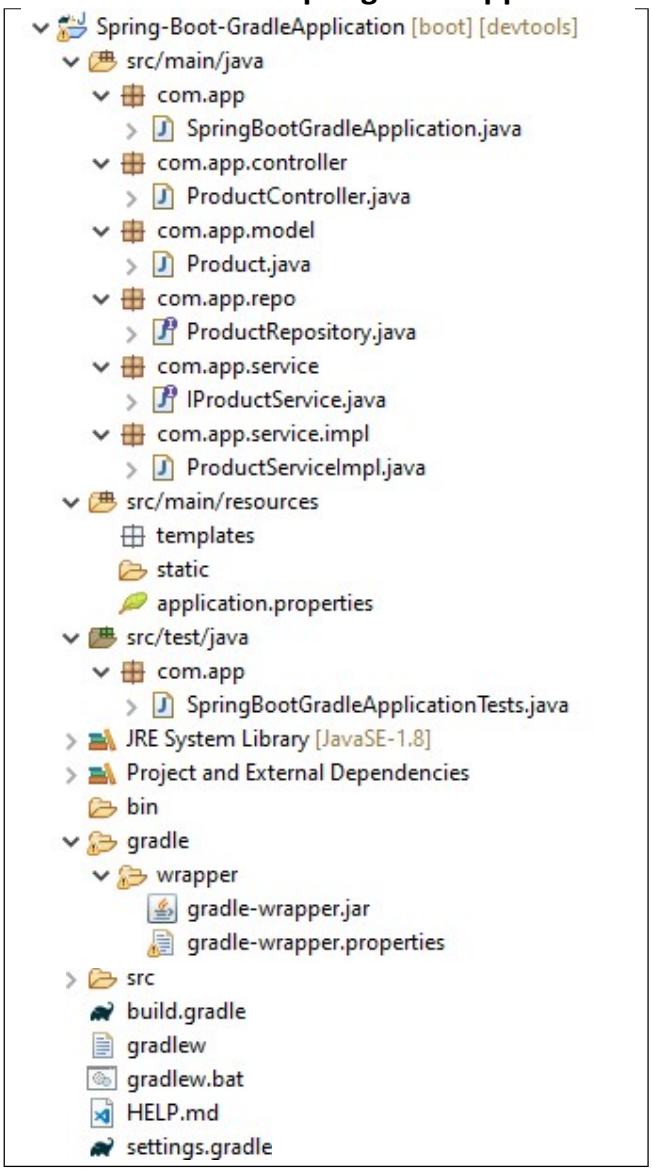
Parent : org.springframework.boot

BOM : io.spring.dependency-management

=>BOM means all child details and their versions as one project (.pom/.gradle file)

Spring Boot Application using Gradle:--

Folder Structure of Spring Boot Application using Gradle:--



Coding Steps of CRUD operation using Gradle:--

1. build.gradle:--

```
plugins {
	id 'org.springframework.boot' version '2.1.8.RELEASE'
	id 'io.spring.dependency-management' version
	'1.0.8.RELEASE'id 'java'
}

group =
'com.app'
version = '1.0'
sourceCompatibility = '1.8'
targetCompatibility = '1.8'

configurations {
	developmentOn
	ly
	runtimeClasspat
	h {
		extendsFrom developmentOnly
	}
}

repositories {
	mavenCentr
	 al()
	 mavenLocal
	 ()
}

dependencies {
	implementation ('org.springframework.boot:spring-boot-starter-web',
				'org.springframework.boot:spring-boot-starter-data-jpa',
				'org.projectlombok:lombok')
```

2. application.properties:--

```
server.port=2019
##DataSource##
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
```

```
spring.datasource.username=system
spring.datasource.password=system
##JPA##
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.open-in-view=true
```

3. Starter class:--

```
package com.app;
import org.springframework.boot.SpringApplication;
import     org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootGradleApplication {

        public static void main(String[] args) {
                SpringApplication.run(SpringBootGradleApplication.class, args);
                System.out.println("Spring Boot Gradle Application...!!");
        }
}
```

4. Model class:--

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;

@Entity
@Table(name="PRODUCTTAB")
@Data
public class Product {

        @Id
        @Column(name="id")
        @GeneratedValue
        private Integer id;
```

```
	@Column(name="code")
	private String prodcutCode;

	@Column(name="name")
	private String productName;
	@Column(name="cost")
	private Double productCost;
}
```

5. Repository Interface:--

```
package com.app.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import com.app.model.Product;

@Repository
public interface ProductRepository extends JpaRepository <Product, Integer>{

	Product getProductById(Integer id);
}
```

6. Service Interface:--

```
package com.app.service;
import java.util.List;
import com.app.model.Product;

public interface IProductService {

	public Integer saveProduct(Product p);
	public List<Product> getAllProducts();
	public void deleteProduct(Integer id);
	public Product getProductById(Integer id);
	public boolean isProductExist(Integer id);
}
```

7. ServiceImpl class:--

```
package com.app.service.impl;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import      org.springframework.transaction.annotation.Transactional;
import com.app.model.Product;
import    com.app.repo.ProductRepository;
import  com.app.service.IProductService;
@Service
public class ProductServiceImpl implements IProductService {

        @Autowired
        private ProductRepository repo;

//1. Save method
        @Transactional
        public Integer saveProduct(Product p) {
                p=repo.save(p);
                Integer prodId=p.getId();
                return prodId;
        }

//2. Get all(List) Product details from Database
        @Transactional(readOnly= true)
        public List<Product> getAllProducts() {
                return repo.findAll();
        }

//3. Delete Record based on ID
        //@Transactional
        public void deleteProduct(Integer id) {
                repo.deleteById(id);
        }

//4. Get Record based on ID
        @Transactional
        public Product getProductById(Integer prodId) {
                Optional<Product> p=repo.findById(prodId);
                if(p.isPresent()) {
                        return p.get();
                }else {
                        return new Product();
                }
        }
 //5. Check product available or not
```

```
	@Override
	public boolean isProductExist(Integer id) {
		return repo.existsById(id);
	}
}
```

8. Controller class:--

```
package com.app.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.app.model.Product;
import com.app.service.IProductService;

@RestController
@RequestMapping("/product")
public class ProductController {

	@Autowired
	private IProductService service;

//1. save student data
	@PostMapping("/save")
	public ResponseEntity<String> save(@RequestBody Product product){
		ResponseEntity<String> resp=null;

		try {
			Integer id=service.saveProduct(product);
   resp = new ResponseEntity<String>("Product '"+id+"' Created", HttpStatus.OK);
		} catch (Exception e) {
  resp = new ResponseEntity<String>(e.getMessage(),
			HttpStatus.INTERNAL_SERVER_ERROR);
			e.printStackTrace();
		}
```

```
		return resp;
	}

//2. get All Records
	@GetMapping("/all")
	public ResponseEntity<?> getAll(){
		ResponseEntity<?> resp=null;

		List<Product> list=service.getAllProducts();

		if(list==null || list.isEmpty()) {
			String message="No Data Found";
			resp=new ResponseEntity<String>(message,HttpStatus.OK);
		} else {
			resp=new ResponseEntity<List<Product>>(list,HttpStatus.OK);
		}
		return resp;
	}

//3. delete based on id , if exist
	@DeleteMapping("/delete/{id}")
	public ResponseEntity<String> deleteById(@PathVariable Integer id)
	{
		ResponseEntity<String> resp=null;
		//check for exist
		boolean present=service.isProductExist(id);
		if(present) {
			//if exist
			service.deleteProduct(id);
  resp=new ResponseEntity<String>("Deleted '"+id+"' Successfully",HttpStatus.OK);

		} else { //not exist
resp=new ResponseEntity<String>("'"+id+"' Not Exist",HttpStatus.BAD_REQUEST);
		}
		return resp;
	}
}
```

Execution:--

=>Right click on Project > Run As > Spring Boot App (Alt+Shift+X, B)

=>Rest the application using postman

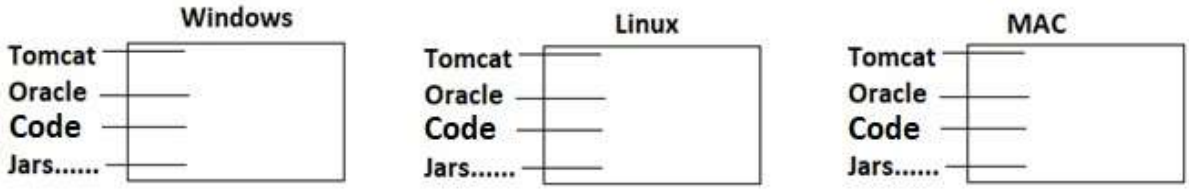
1.> http://localhost:2019/product/save
2.> http://localhost:2019/product/delete/100
3.> http://localhost:2019/product/all

DOCKER

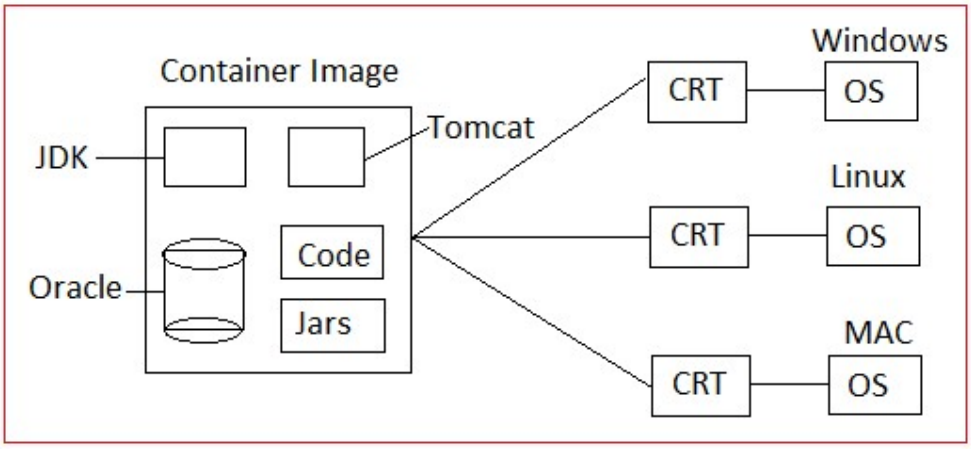
Docker:--

Docker is “CONTAINER SYSTEM” which includes all software’s a unit to run application, On any Platform (Windows, Linux, Mac…).
=>Docker supports running application on cloud servers also.
=>Docker supports follow of “CROSS-OS”. It means docker behaves a middleware between our runtime software and actual operating System.
=>Docker tool is used for Application Deployment (Running Application).

Before Using Container System:--

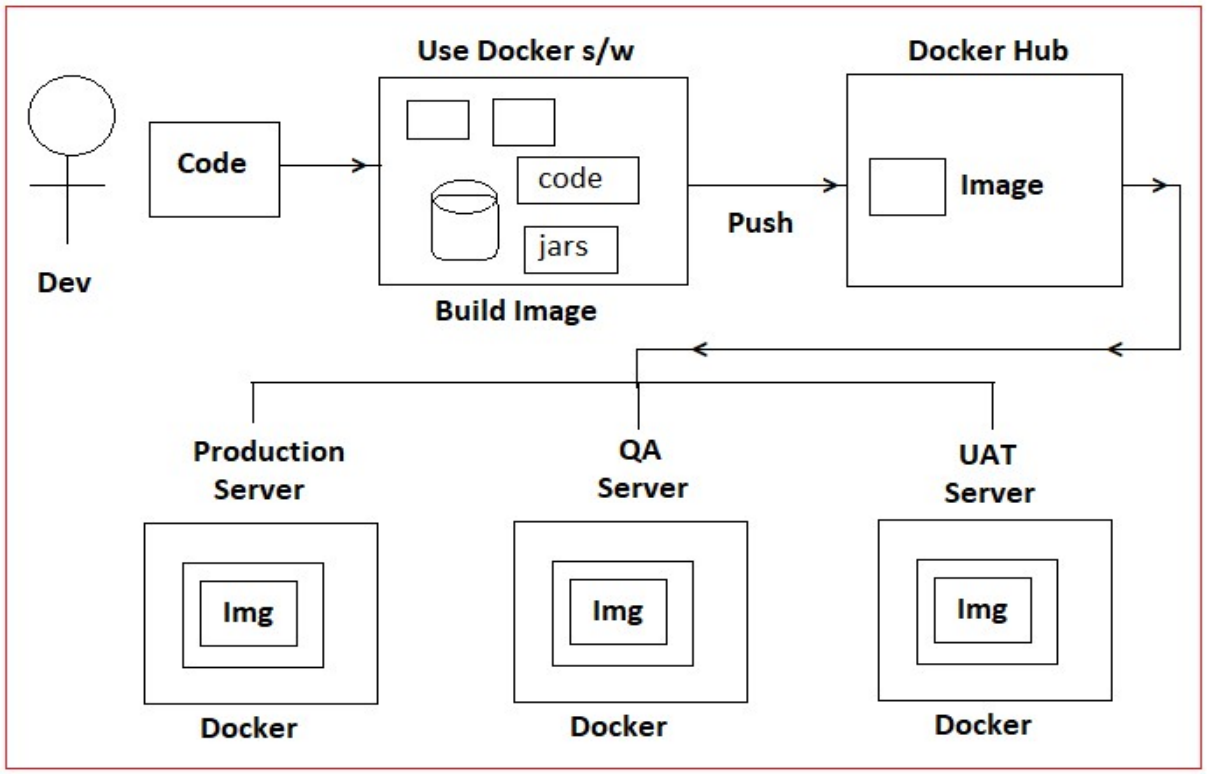


After Container System (Docker):--



=>CRT = Container Run time.

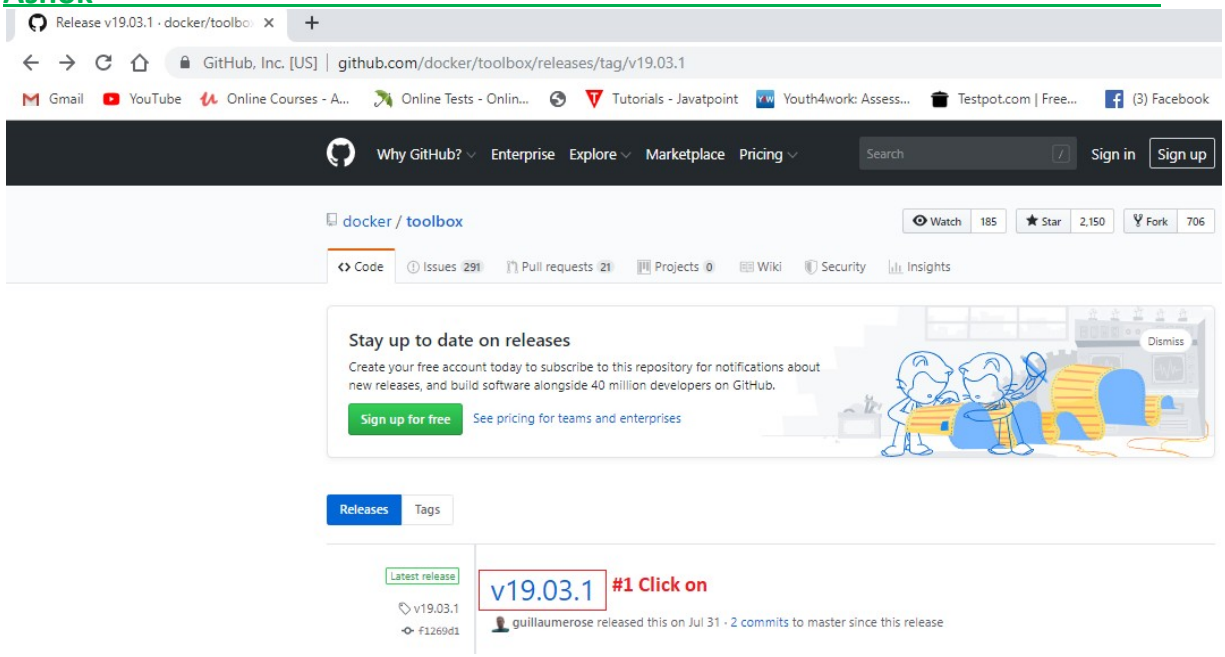
Docker Workflow:--



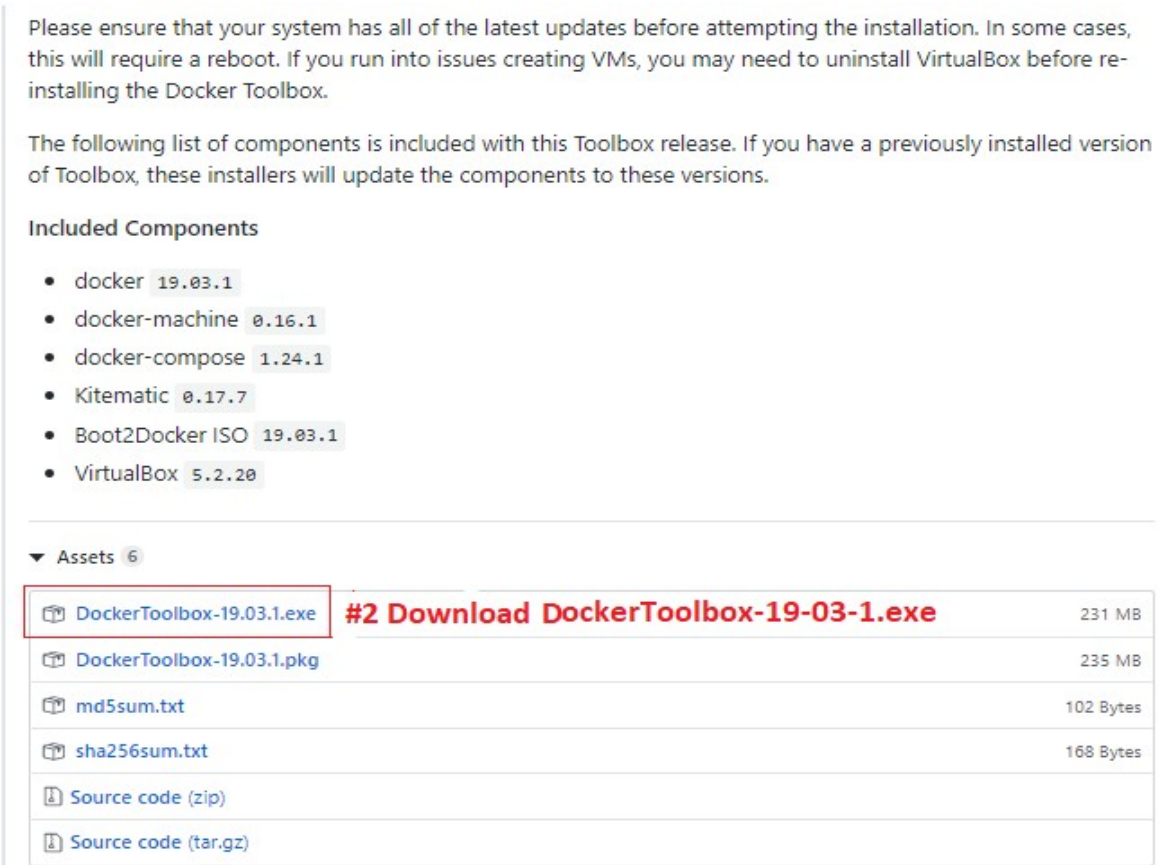
Working with Docker:--

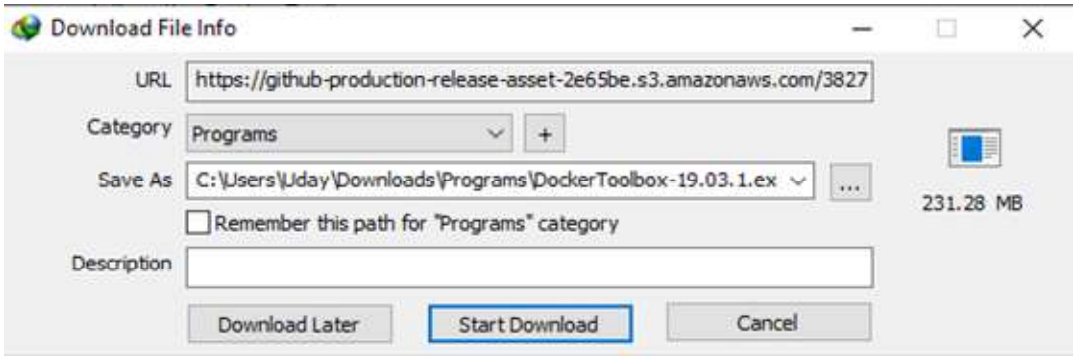
#1:- DOWNLOAD DOCKER TOOLBOX.

a. Download DockerToolbox:-- https://github.com/docker/toolbox/releases



=>Goto down and Click on DockerToolbox-19.03.1.exe



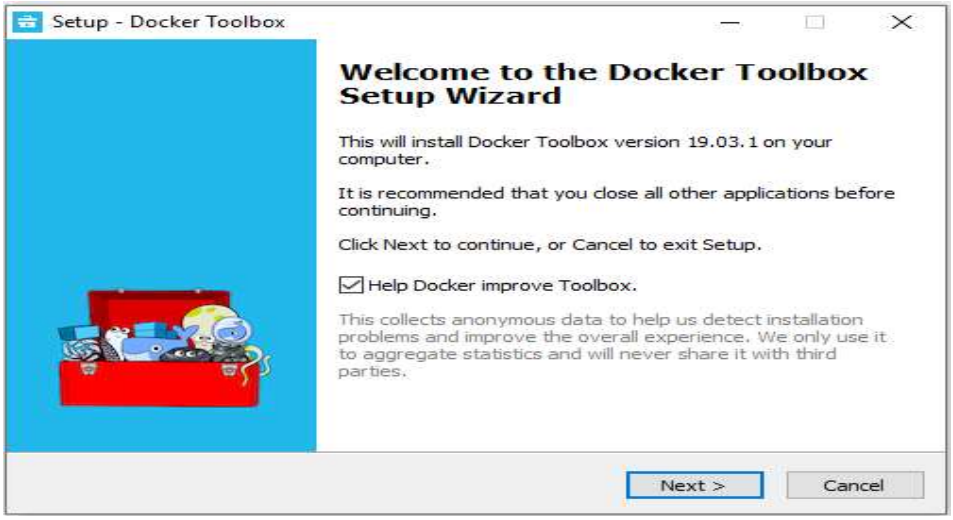


#2. INSTALL DOCKERTOOLBOX:--

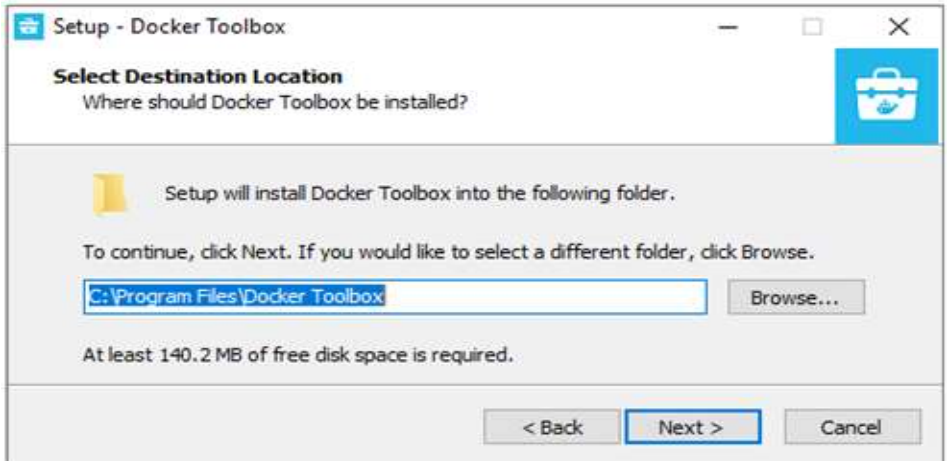
=>Double click on software

=>Click on Yes

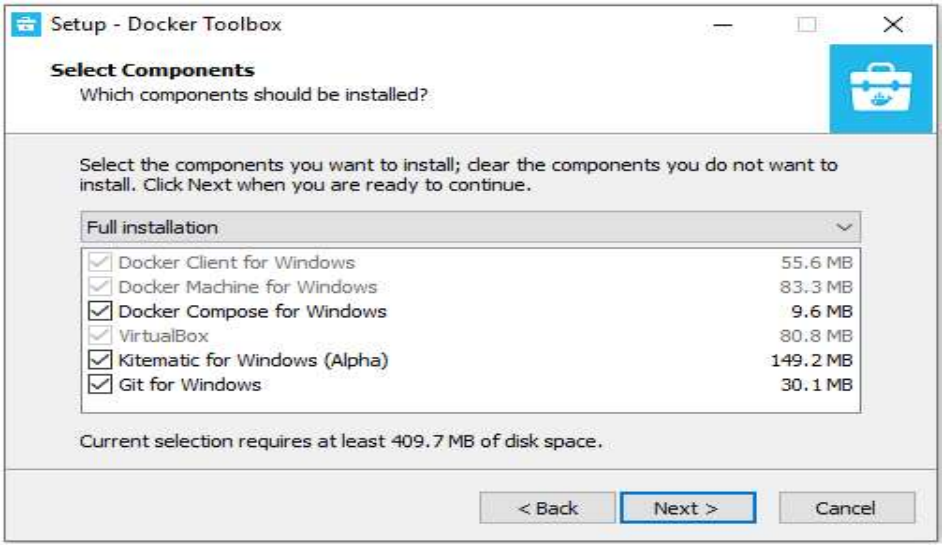
=>Click on Next



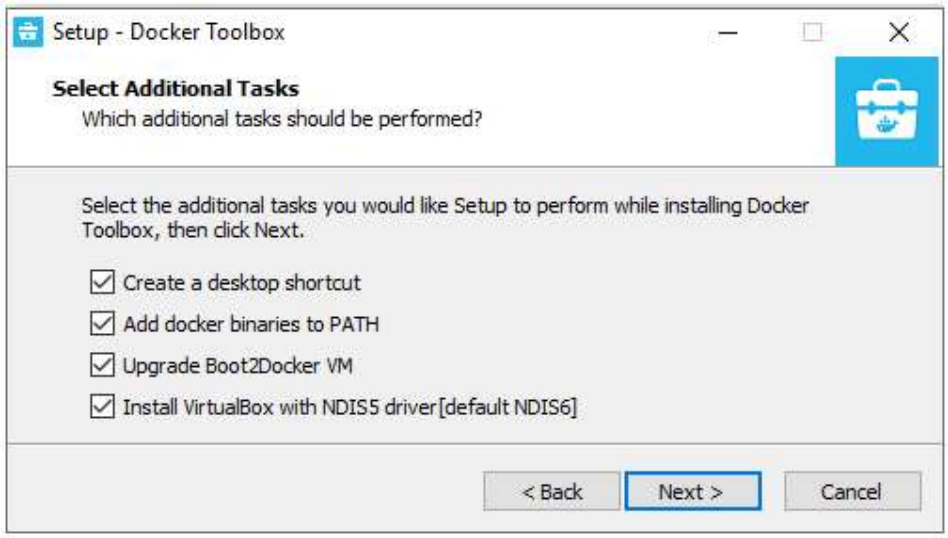
=>Click on Next



=>Click on Next



=>Click on Next

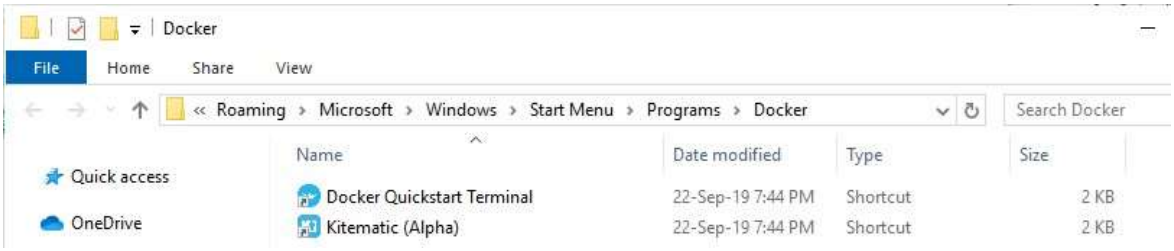


=>Click on Install

=>Click on Install

=>Click on Finish

=>After installation two Icons are created



#3:- CREATE ACCOUNT IN DOCKER HUB

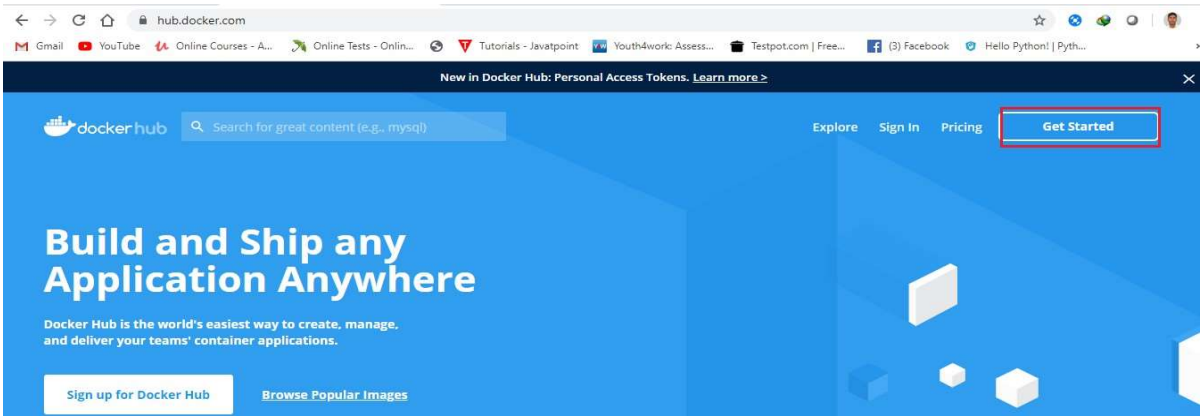
>Goto https://hub.docker.com

>click on signup up for Docker Hub / Get Started option and register once

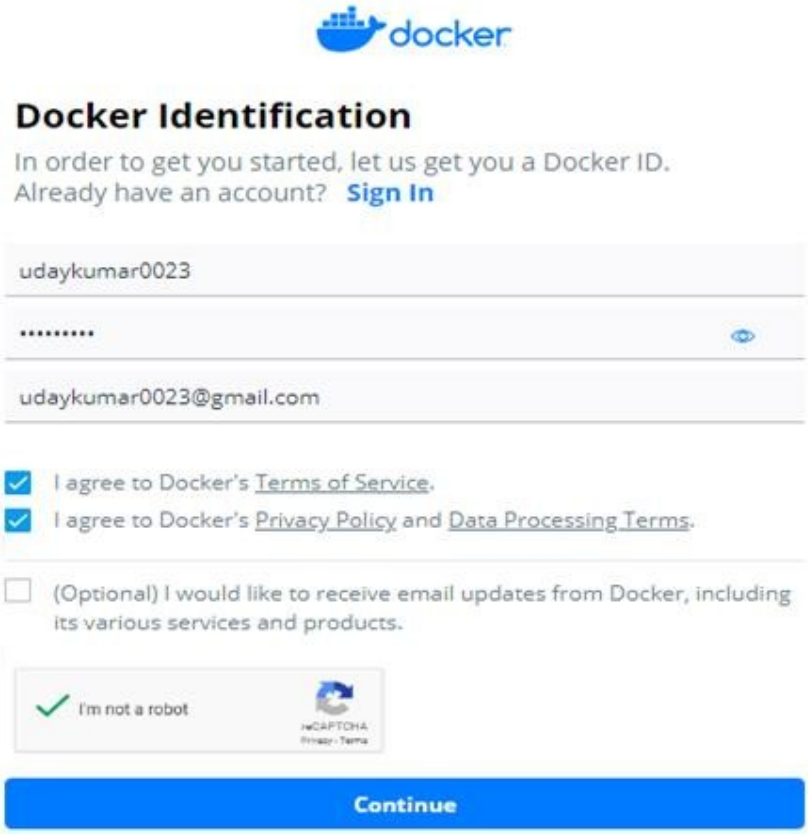
>enter details

>verify email

>Login here



=>After click on Get Started/Sign up for Docker Hub.



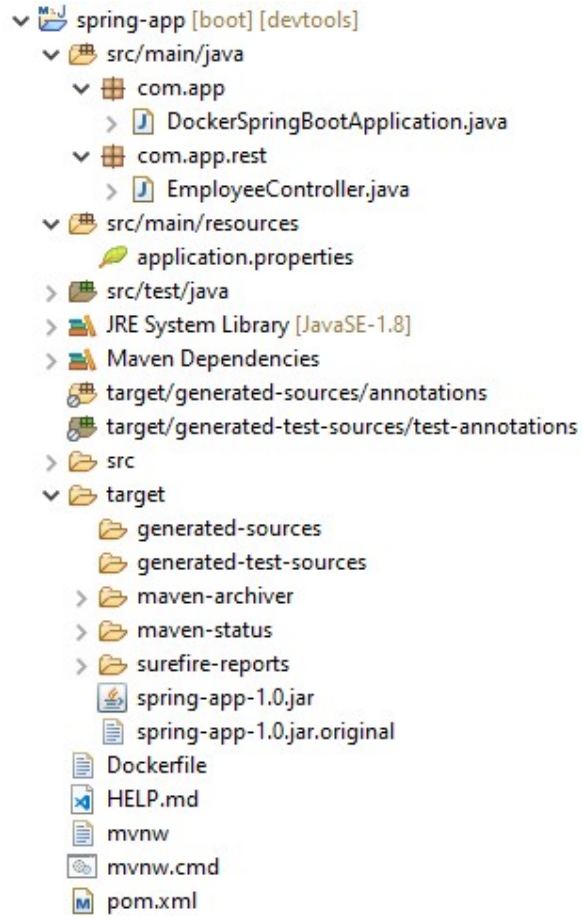
=>Complete the Registration process after Gmail verification.

=>Login to Docker hub.

#4:- CREATE ONE SPRING BOOT APPLICATION:--

=>Spring Boot Application with RestController

FOLDER STRUCTURE OF DOCKER-SPRING-BOOT APPLICATION:--



1. Starter class:--

```
package com.app;
import    org.springframework.boot.SpringApplication;
import     org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DockerSpringBootApplication {
        public static void main(String[] args) {
                SpringApplication.run(DockerSpringBootApplication.class, args);
                System.out.println("Heelo Docker Application");
        }
}
```

2. Controller class:--

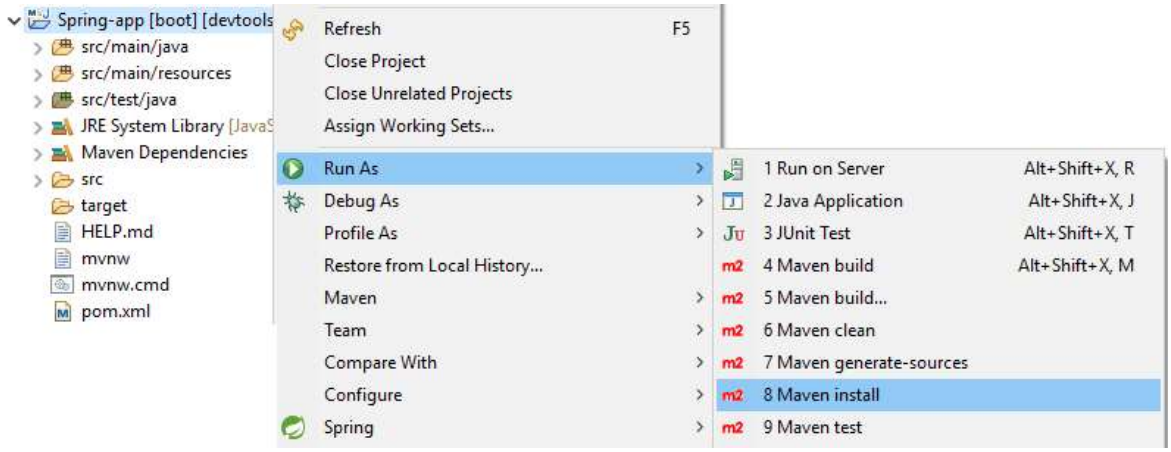
```
package com.app.rest;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
```

```
@RequestMapping("/employee")
public class EmployeeController {

	@GetMapping("/show")
	public String show()  {
		return "Hello Docker";
	}
}
```

3:- Create Jar file

=>Right click on Project

=>Run As > Maven Install (It generates jar file under target folder)



Q>How to rename a jar file?

=>Add a tag <finalName>...</finalName> in side build properties in pom.xml

```
<build>
  <plugins>
        <plugin>
              <groupId>org.springframework.boot</groupId>
              <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
  </plugins>
        <finalName>docker-spring-boot</finalName>
</build>
```

=>Whatever name you are providing inside <finalName> tag by that file name a new jar file is created inside target folder. If we are not provided this tag by default jar name will be artifactId name which is mentioning pom.xml.

5. DOCKERFILE:-- Create/Add one Dockerfile under project

=>Right click on Project => New => file =>Enter file name as “Dockerfile” => finish.

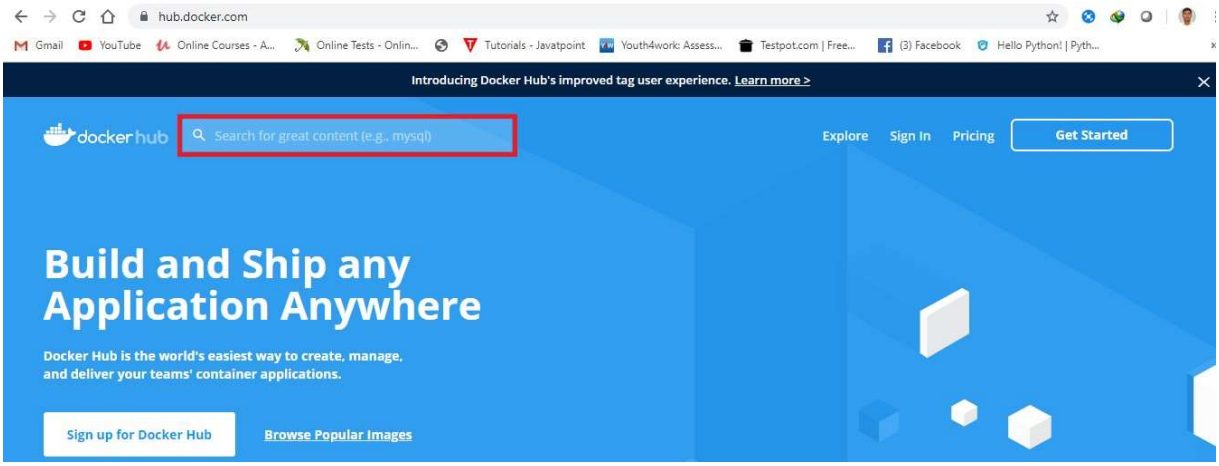
->Naming convention wise Dockerfile name starts with D capital.

=> Write bellow details inside Dockerfile.

```
FROM openjdk:8 //Java software image
ADD  target/spring-app.jar  spring-app.jar
EXPOSE 8080
ENTRYPOINT  ["java", "-jar", "spring-app"]
```

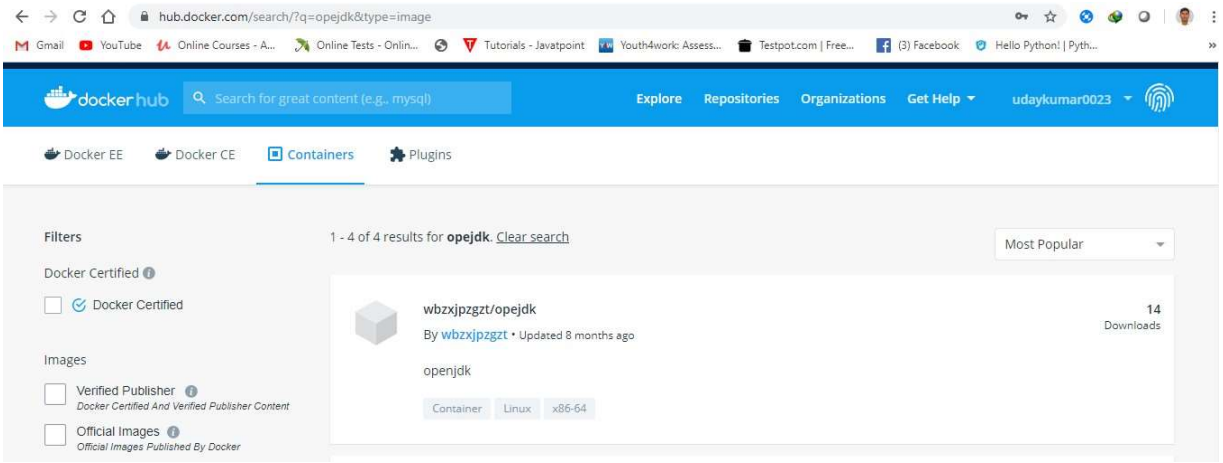
Q>Where to get Software Images.

=>Go to hub.docker.com website and Sign In



=>Search with openjdk, Tomcat, Oracle, MySQL …etc according to your requirement.

->and click on searched result to see the details of software images.



#6:- OPEN DOCKER TERMINAL AND EXECUTE COMMANDS:--

=> Double click on “DockerQuickStart” Installed Icon.

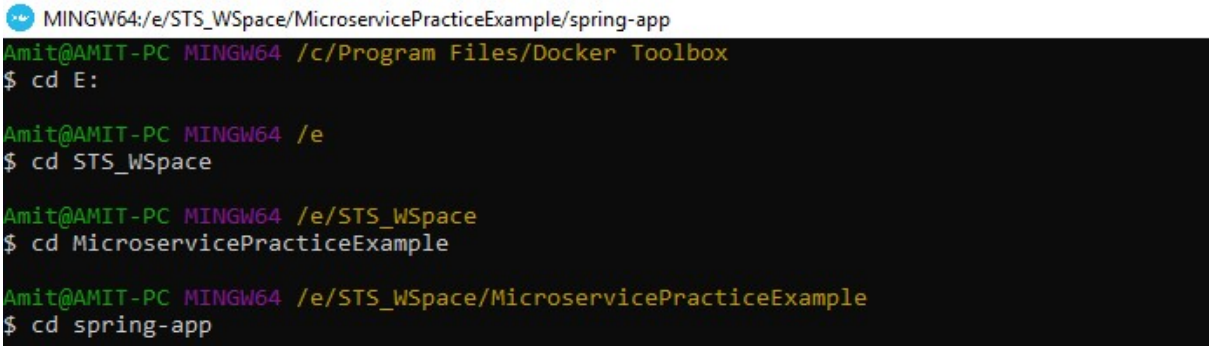
=>Wait for few minutes (First time takes time).

a. Navigate to project folder:--

Ex:- If project is created in STS under

E:\STS_WSpace\MicroservicesPracticeExample\spring-app

=>Then goto this specified location by using cd command step by step.

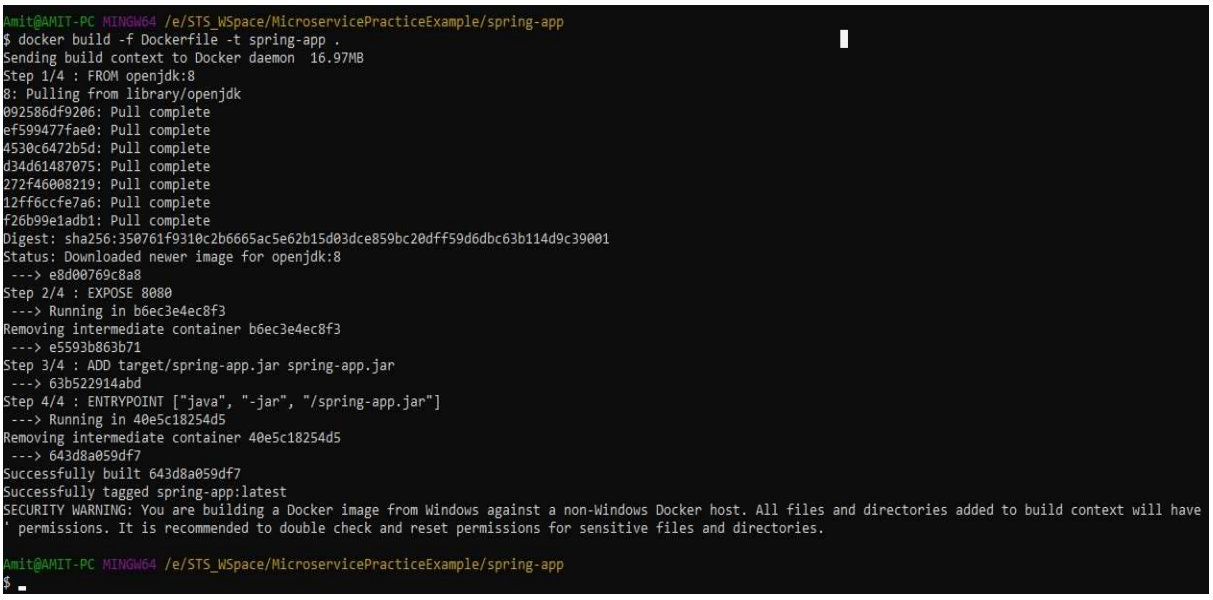


b. build image:--

docker build -f Dockerfile -t <AnyImageName> .

=> $ docker build -f Dockerfile -t spring-app **.**

=>Here dot(.) indicate current directory, must give one space.



c>check all images:--

$ docker images

$ docker image ls

d. run image:--

docker run -p 9090:8080 spring-app

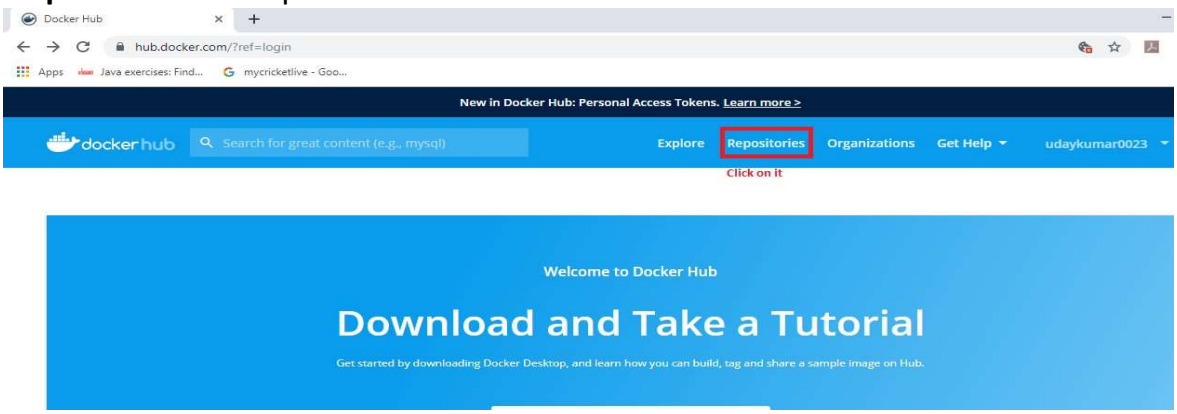
```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker run -p 9090:8080 spring-app

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.8.RELEASE)

2019-10-05 08:05:06.141  INFO 1 --- [           main] com.app.SpringAppApplication             : Starting SpringAppApplication v1.0 on 4e92b75710e4 with PID 1 (/spring-
app.jar started by root in /)
2019-10-05 08:05:06.199  INFO 1 --- [           main] com.app.SpringAppApplication             : No active profile set, falling back to default profiles: default
2019-10-05 08:05:22.336  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8675 (http)
2019-10-05 08:05:22.808  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2019-10-05 08:05:22.816  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.24]
2019-10-05 08:05:24.141  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2019-10-05 08:05:24.150  INFO 1 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 16976 ms
2019-10-05 08:05:27.220  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2019-10-05 08:05:29.978  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8675 (http) with context path ''
2019-10-05 08:05:30.016  INFO 1 --- [           main] com.app.SpringAppApplication             : Started SpringAppApplication in 30.411 seconds (JVM running for 35.081)
Spring Boot Application.....!!
```

=>Here 9090 is docker container port no and 8080 is application port no.
=> Goto browser and enter URL:
Example URL: http://192.168.99.100:9090/show

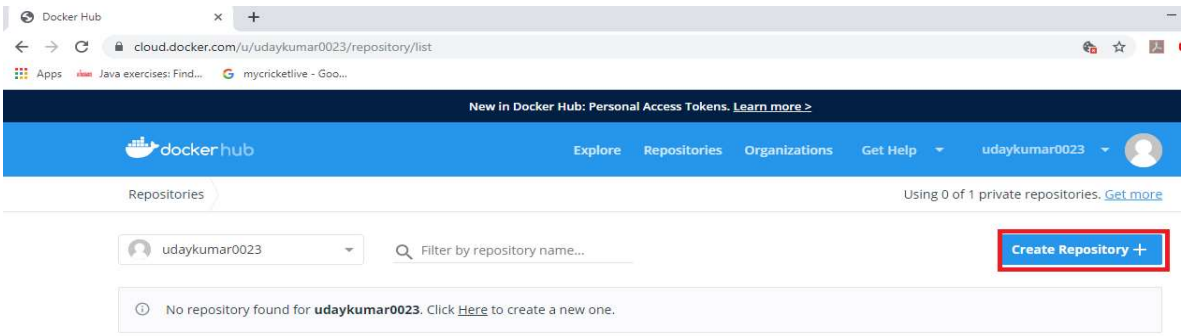
#7. CREATE REPOSITORY IN DOCKER HUB:--

=>Login Docker hub and create one repository [Click on button “Create repository +”]
Ex:--myrepo > Create

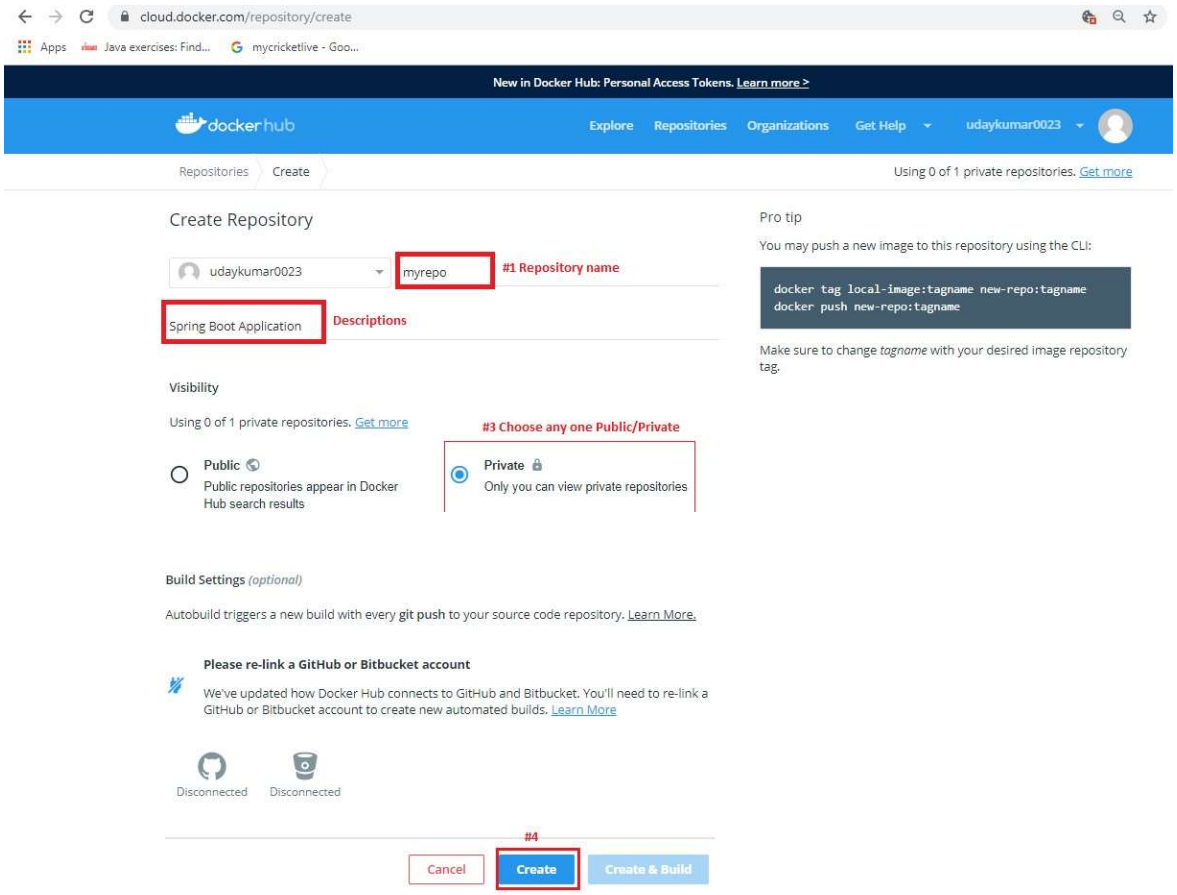
Step#1:- Click on Repositories



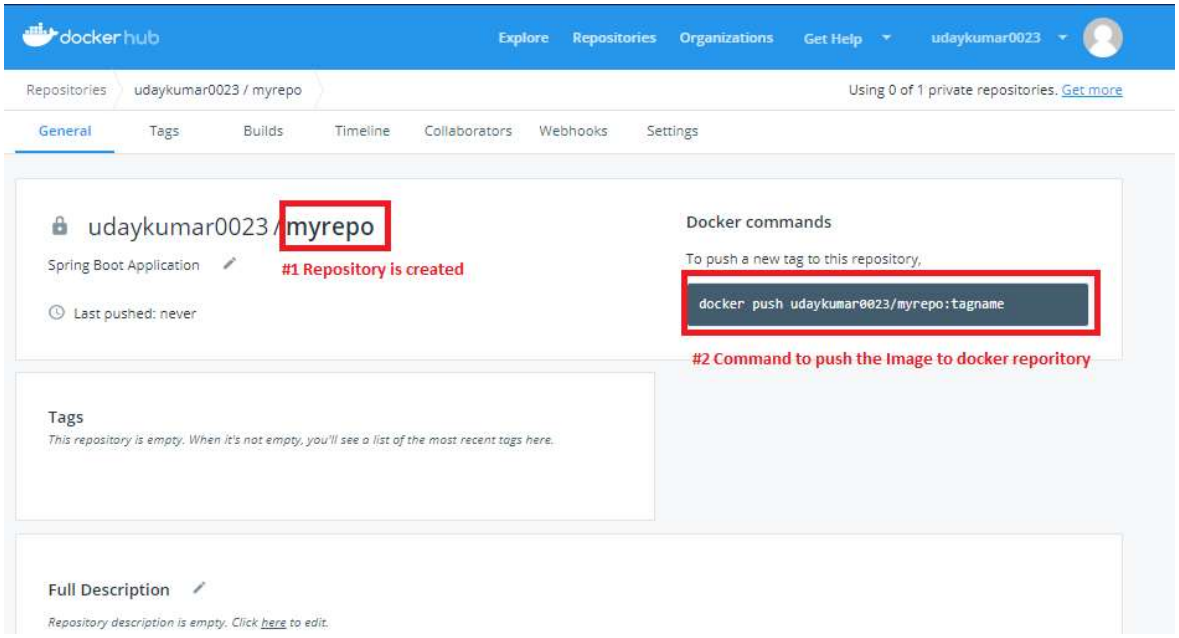
=>Click on “Create Repository +”.



=>Enter the repository name and choose any one visibility like (Public/Private).



=>Final Screen of Repository.



#8. LOGIN TO DOCKER HUB:--

docker login

UserName : docker account username (udaykumar0023)
password: docker password (Uday123)

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: udaykumar0023
Password:
WARNING! Your password will be stored unencrypted in C:\Users\Amit\.docker\config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

#9:- CREATE NEW TAG:--

=>Come to docker terminal and create one tag between docker and hub-repository
=>Format to create tag name is

=>docker tag local-image:tagname reponame:tagname
=>docker tag <imageName> <username>/<repoName>:<tagname>

docker tag spring-app udaykumar0023/myrepo:latest

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker tag spring-app udaykumar0023/myrepo:latest

Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker tag spring-app udaykumar0023/myrepo:latest8

Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
8421087771/myrepo       latest              643d8a059df7        27 hours ago        505MB
myrepo                  latest              643d8a059df7        27 hours ago        505MB
myrepo                  latest5             643d8a059df7        27 hours ago        505MB
spring-app              latest              643d8a059df7        27 hours ago        505MB
udaykumar0023/myrepo    latest              643d8a059df7        27 hours ago        505MB
udaykumar0023/myrepo    latest8             643d8a059df7        27 hours ago        505MB
openjdk                 8                   e8d00769c8a8        3 weeks ago         488MB
```

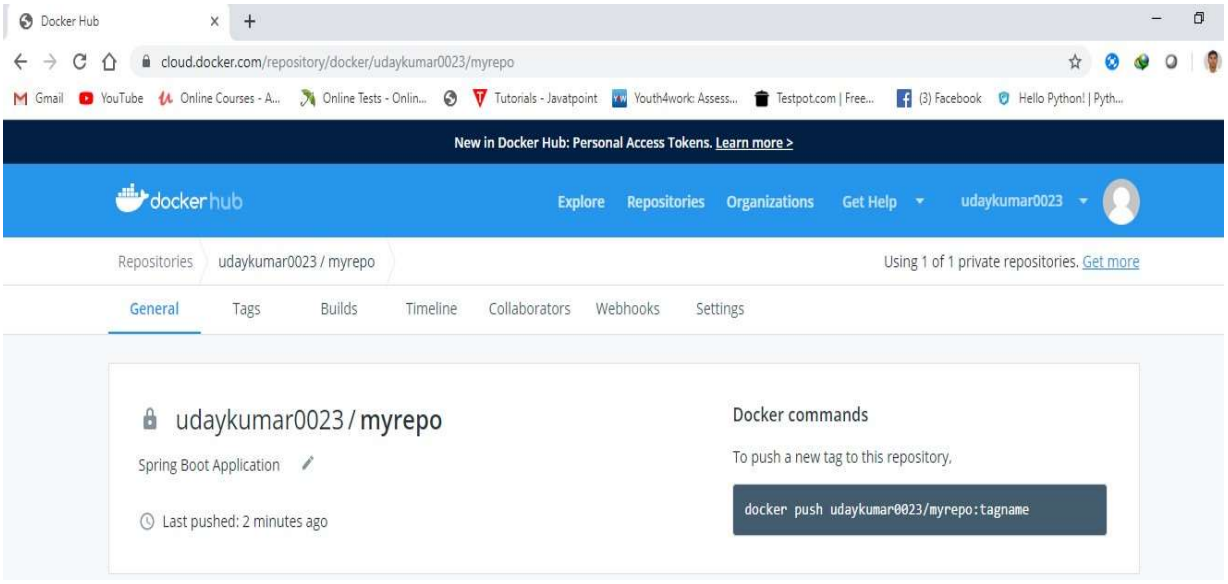
#10:- PUSH IMAGE INTO DOCKER HUB:--

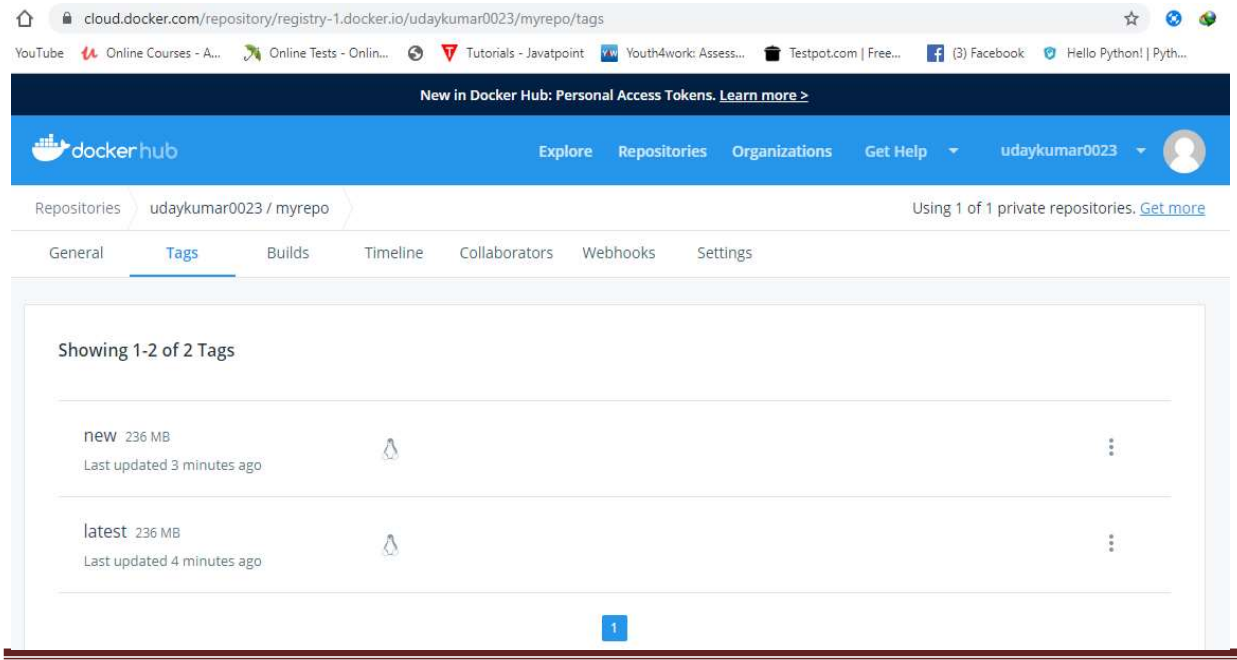
Syntax:- docker push <username></repoName>:tagname

=>docker push udaykumar0023/myrepo:latest

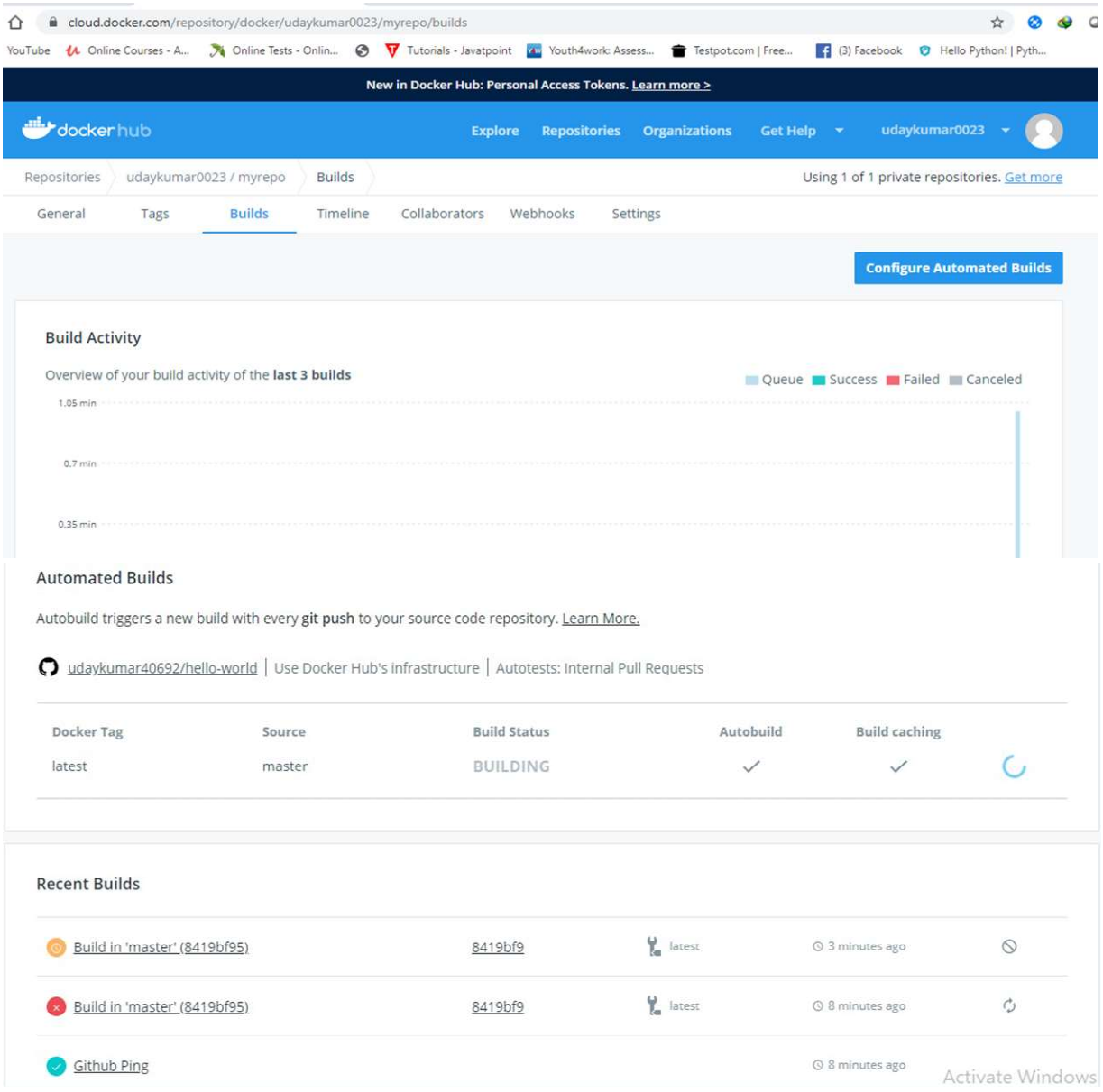
```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker push udaykumar0023/myrepo:latest
The push refers to repository [docker.io/udaykumar0023/myrepo]
eccdd29c4296: Pushed
57e6a3d20ce9: Mounted from library/openjdk
1690af51cb08: Mounted from library/openjdk
5a30999619d7: Mounted from library/openjdk
2e669e0134f5: Mounted from library/openjdk
8bacec4e3446: Mounted from library/openjdk
26b1991f37bd: Mounted from library/openjdk
55e6b89812f3: Mounted from library/openjdk
latest: digest: sha256:3a9c8311bb4da94f0cf8c099d3c353c9bfc6a37ebb16a398375ae5402c9c47d9 size: 2007
```

=>Goto docker hub and refresh to see the latest update:--





#13:- LINK DOCKER HUB WITH GITHUB/BIGBUKET ETC…



#14:- PULL THE IMAGE FROM DOCKER HUB:--

Syntax:-- docker pull <username>/myrepo

docker pull udaykumar0023/myrepo

```
Amit@AMIT-PC MINGW64 /e/STS_WSpace/MicroservicePracticeExample/spring-app
$ docker pull udaykumar0023/myrepo
Using default tag: latest
latest: Pulling from udaykumar0023/myrepo
Digest: sha256:3a9c8311bb4da94f0cf8c099d3c353c9bfc6a37ebb16a398375ae5402c9c47d9
Status: Image is up to date for udaykumar0023/myrepo:latest
docker.io/udaykumar0023/myrepo:latest
```

NOTE:- Press ctrl+c to shutdown the docker container.

MICROSERVICES END