What is Eureka?

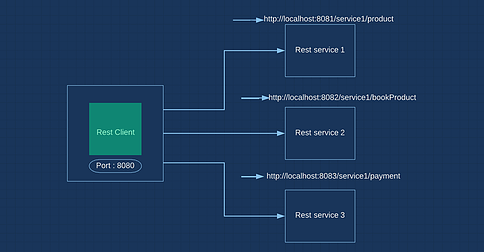
In simple word Eureka is a service Registry or we can say it is an embedded server provided by Netflix third party which integrate with spring framework.

Main purpose to use Eureka : Micro service Registration and Discovery with Spring Cloud and Netflix's Eureka

What is the use of Eureka?

Normally in Micro Service Architecture Design we are developing separate Services and exposing each API as service Endpoint and whenever we required to access other services in simple we are accessing it  as a Rest client using third party API either Rest Template or Client Builder

Architectural diagram:



As per this diagram we can see there we have 4 micro services and each are interlink with each other to perform business operation

Here rest client accessing 3 services right .how by using Rest Template and passing the URL as per image

Think is my service have only one end point?

No there may chance my service can have n number of Endpoint So who will remember all the end point URL to access their features? To overcome this issue Netflix team came up with Eureka concept, let’s see now how they reduces developers burden.

As I already mention Eureka is Service Registry, so he give a features to us i.e. just register your all micro services with Eureka server then no need to call each service by passing exact URL like host name, port just give your service name with which you registered with Eureka server and at end just add endpoint URL

**For example:**

Normally we are using below URL to access Payment endpoint

http://localhost:8083/service3/payment

Instead of pass this URL from client just pass like below you no need to bother about what’s the port and host name Eureka will take care this he will auto route it. If that service registry with Eureka

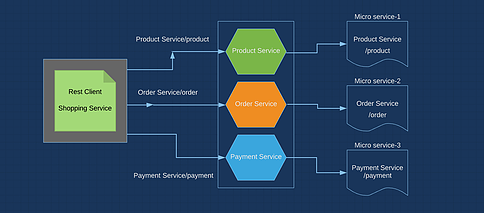
http://serviceName/service3/payment

Assume am giving service name as Payment Service So Eureka will excepting the URL to pass to consume rest from client like below

http://PaymentService/service3/payment

how Eureka will identify which port this payment service deployed and what is the host info?

As we are registering our application in Eureka server so Eureka internally keeping one instance of our service, so based on service name it will go to that particular Service and search the last append end point URL Then it will delegate Request to that service



So see the above diagram as I registered my all services in Eureka

So I can now access them like below

For Product Service: http://PRODUCT-SERVICE/product

For Order Service: http://ORDER-SERVICE/product

For Payment Service: http://PAYMENT-SERVICE/product

Let’s come to development part

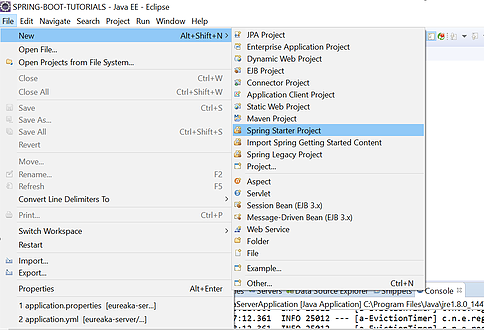
We have 2 develop 3 micro services now to check behavior of Eureka

1. Eureka Server API (It is just simple to bootstrap our Eureka Server)
2. Order Service (Rest resource who will expose Rest end point)
3. My Shopping (Who will going to consume Rest service using Service name without knowing port and host details .That’s what the main moto of Eureka Service Registry)

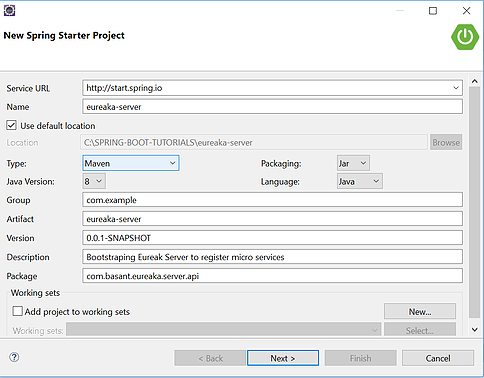
We will develop one by one from Scratch. Let’s build up first API Eureka Server

Steps:( Bootstrap Eureka Server)

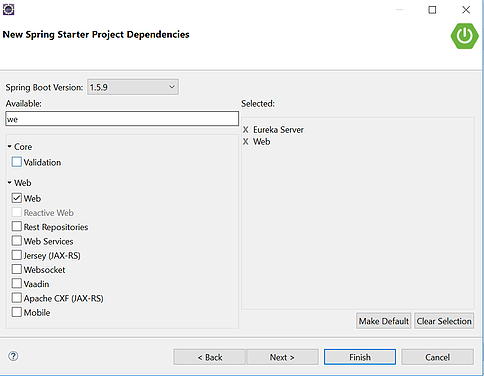
1. Open Eclipse workspace Click on File section then go to spring starter project



2. Then mention your artifact id, package, group id as below



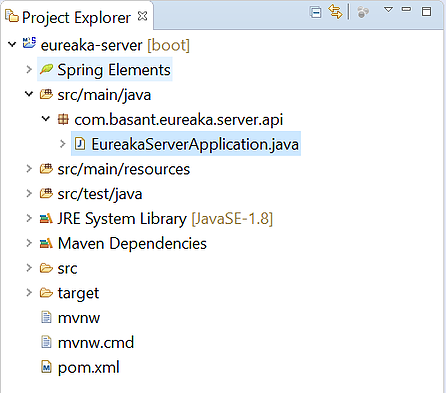
3. Click next then add required dependency from console, to up this Eureka server we need Eureka-server dependency



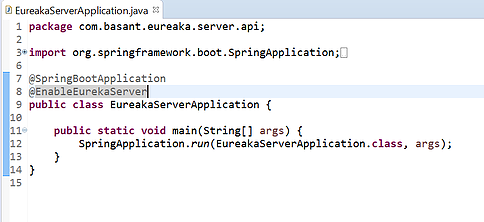
4. Then click finish, your project will be available in Eclipse workspace

https://static.wixstatic.com/media/86938e_2b34356621c741fa867baedb80f42974~mv2.png/v1/fill/w_484,h_53,al_c,usm_0.66_1.00_0.01/86938e_2b34356621c741fa867baedb80f42974~mv2.png

5. Open Directory

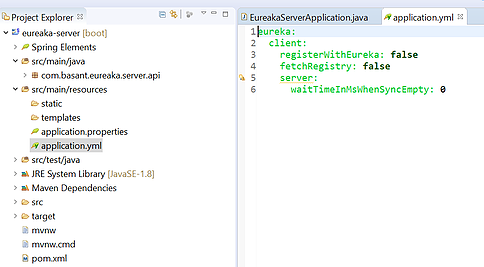


6. Then go to main class who contains main method and annotated like below



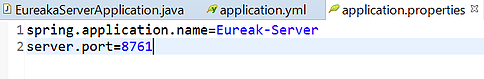
@EnableEureakaServer annotation will take responsible to enable embedded Eureka Server

7. Let’s add few configuration info so first create one yml file inside src/main/resources/ with name application.yml



With this configuration we are informing behave this application as embedded Eureka server

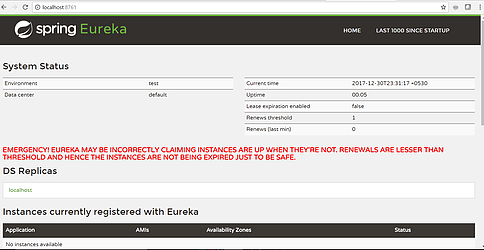
8. Let’s change port number and use port 8761, any other port also you can use but Eureka internally using 8761 so better to use this.



 9. Let’s run our application then check server is up or not

https://static.wixstatic.com/media/86938e_3ceedbfda84e47b4aa97b0730302beef~mv2.png/v1/fill/w_484,h_47,al_c,usm_0.66_1.00_0.01/86938e_3ceedbfda84e47b4aa97b0730302beef~mv2.png

10. Let’s hit the URL from browser localhost: 8761 and see the Eureaka Server Dashboard



See above screen, there is one section instance currently registered with Eureka

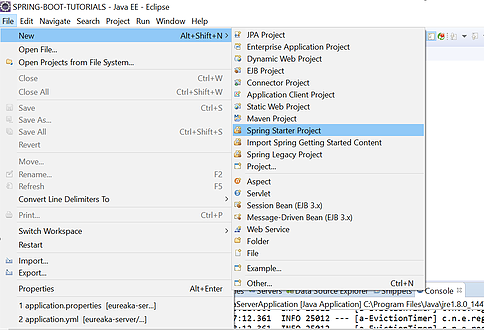
This section is empty as we didn’t register anything, we just start Eureka server

Now let’s register rest 2 service one by one then we will check behavior of Eureka

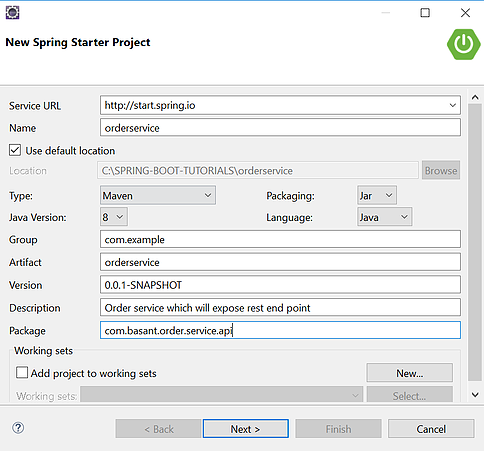
First build server side application i.e. Order Service

Steps.( Server side application )

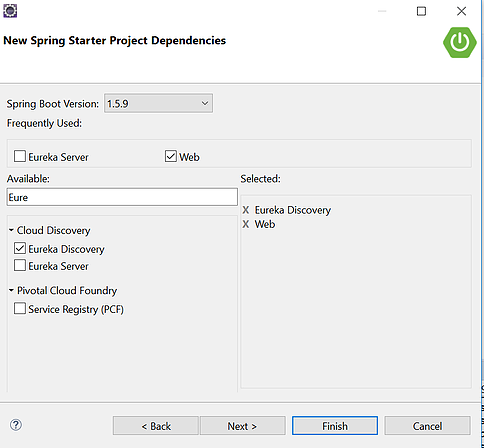
1. Go to eclipse click on file Section then click spring starter project Go to eclipse click on file Section then click spring starter project



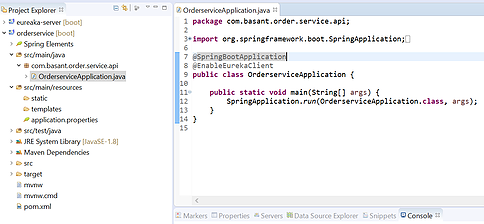
 2. Mention your artifact id, group id and package information like below



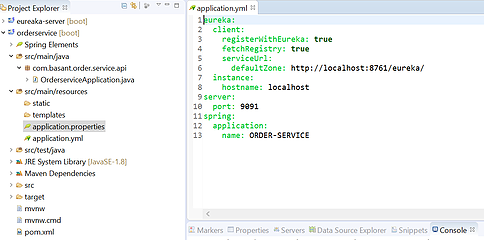
3. Then add required dependency like below



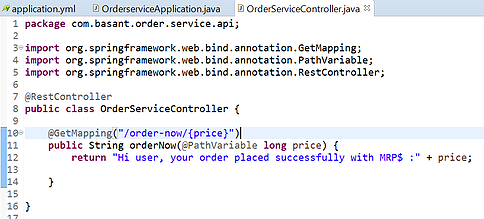
4. Click on finish, then order service API will import to your eclipse then open main class who contains main method , add @EnableEureakaClient at class level , reason we are informing this order service is one micro service which is Eureka client not server like below



5. To deploy this order service in Eureka server we need to specify some configuration where we need to mention the service name(with which name it will display in Eureka dashboard and with same name we can access endpoints), and in which Eureka port we are going to register this order service and what is the host So for that create one application.yml inside src/main/resource like below



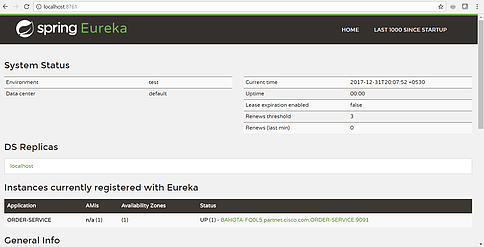
6. Now we are done with all configuration setup, so let’s write one method in controller which we will expose as rest end point so that we can access from my shopping client like below



 7. Now let’s run our application

https://static.wixstatic.com/media/86938e_57e1902786d64ead91235ae51ad7d5ab~mv2.png/v1/fill/w_484,h_54,al_c,usm_0.66_1.00_0.01/86938e_57e1902786d64ead91235ae51ad7d5ab~mv2.png

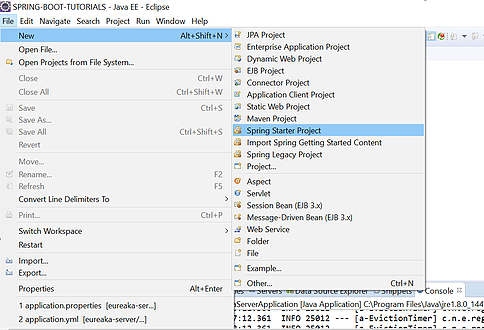
8. Let’s see Eureka server console whether our order service instance is registered or not



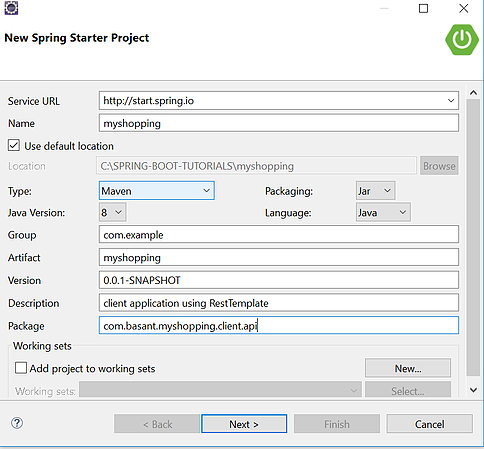
As per image we are good our server is up and server side application is registered in Eureka successfully so now to access it from my shopping client, let’s start develop that and deploy that also in Eureka

**Steps:( Client side application )**

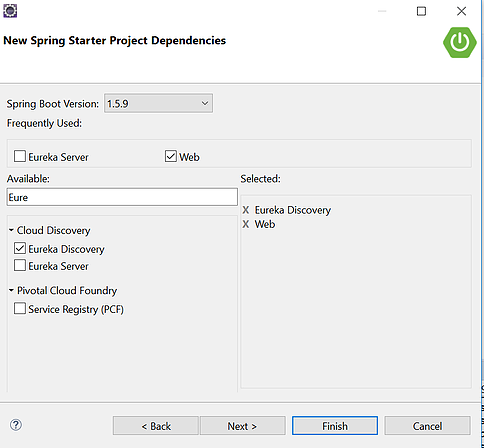
1. Open Eclipse console then go to file section then new create spring starter project as below​



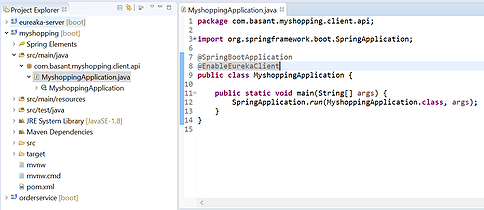
2. Mention your group id, artifact id and package information like below



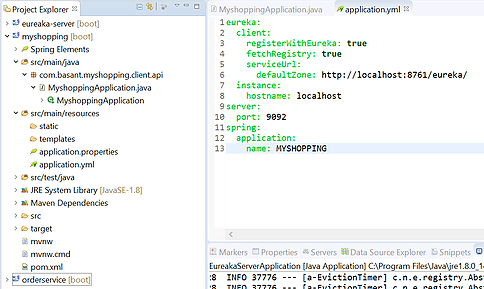
3. Now click next and add required dependency like below



4. Now click finish and open application in eclipse, go to main class and annotated @EnableEureakaClient at class level like below



5. To deploy this order service in Eureka server we need to specify some Configuration where we need to mention the service name (with which name it will display in Eureka dashboard and with same name we can access endpoints), and in which Eureka port we are going to register this order service and what is the host So for that create one application.yml inside src/main/resource like



6. Now we are done with configuration, let’s access Order service from myshopping client using Rest Template , so let’s write controller class



Mark the URL which I pass to Rest client,

String URL ="http://ORDER-SERVICE/order-now" + "/{price}";

Normally when we are accessing endpoint how we need to pass

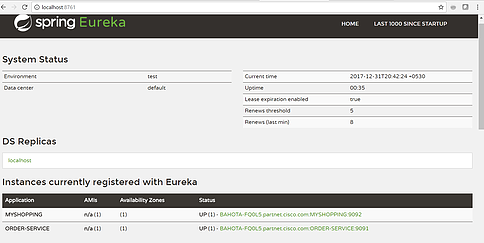
String URL ="http://localhost:9091/order-now" + "/{price}";

So as I deploy my OrderService in Eureka that’s why am able to access it using service name instead of passing entire syntax

7. Let’s start our shopping application and check the console

https://static.wixstatic.com/media/86938e_d0b4c314d1ef436aa2c132f60e5c8843~mv2.png/v1/fill/w_484,h_53,al_c,usm_0.66_1.00_0.01/86938e_d0b4c314d1ef436aa2c132f60e5c8843~mv2.png

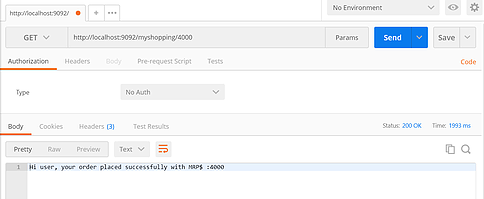
8. Our application is up now let’s check in Eureka server console whether it Register or not



9. As per image both client and server register in Eureka let’s try to access order service from myshopping by passing service name order-service, which I mention in client controller

i.e. http://localhost:9092/myshopping/4000

Response:



See without giving host and port information in URL which I am accessing using Rest Template am able to access service. Here for example I just tried with one endpoint similarly  we can play with bunch of MS using Eureka

**Docker**

**Docker Containers:**

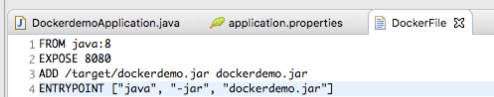
A Docker container is a tool that makes it very easy to deploy and run an application using containers. A container allows a developer to create an all-in-one package of the developed application with all its dependencies. For example, a Java application requires Java libraries, and when we deploy it on any system or VM, we need to install Java first. But, in a container, everything is kept together and shipped as one package, such as in a Docker container

## Spring Boot Applications

Spring Boot is a framework that eases the development of web applications. It has a lot of pre-configured modules that eliminate the manual addition of dependencies for developing an application with Spring. This is the sole reason for this being one of the favorites for creating microservices

### **What Is a Dockerfile?**

Docker gives the user the capability to create their own Docker images and deploy them in Docker. To create your own Docker image, we have to create our own Dockerfile. Basically, a Dockerfile is a simple text file with all the instructions required to build the image.



Docker file that will contain the steps to be executed by Docker to create an image of this application and run that image from Docker

1. FROM java:8 means this is a Java application and will require all the Java librariesk so it will pull all the Java-related libraries and add them to the container.
2. EXPOSE 8080 means that we would like to expose 8080 to the outside world to access our application.
3. ADD /target/dockerdemo.jar dockerdemo.jar  
   ADD <source from where Docker should create the image> <destination>
4. ENTRYPOINT [“java”, “-jar”, “dockerdemo.jar”] will run the command as the entry point as this is a JAR and we need to run this JAR from within Docker.

These are the four steps for that will create an image of our Java application to be able to run Docker.

Okay, we have two pieces ready:

1. Java – Spring Boot application
2. Dockerfile that will create the image to be run in the Docker container.

To load these up in the Docker container, we have to first create the image and then run that image from the Docker container. We need to run certain commands in the folder that contains the Dockerfile.

**Following components are the pillar of the architecture:**

**1.**[**Config servers**](http://javaonfly.blogspot.in/2017/06/microservicespring-cloud-config-server.html)**:**To keep the Properties file centralized and shared by all Microservices, we will create a config server which is itself a Microservice, and manages all microservices properties files and  those files are versioned controlled; any change in the properties will automatically publish to all microservices without restarting the services. One thing to remember is that every microservice communicates with the config server to get properties values, so the config server must be a highly available component; if it fails, then all microservices fail because it can't resolute the properties values! So, we should take care of the scenario - the config server should not be an SPF (single point of failure), so we will spin up more than one container for the config server.

**2.**[**Eureka Discovery server**](http://javaonfly.blogspot.in/2017/07/microservicespring-cloud-eureka-server.html)**:** The main goal of Microservices is decentralizing the different components based on the business features, so that each component - aka microservice - can be scaled as per need, so for a particular microservice, there are multiple instances and we can add and remove instances as per the need, so the way monoliths do load blancing is not going to work in a microservice paradigm. As it spawns containers on the fly, containers have dymanic IP addresses, so to track all instances of a service, a manager service will be needed, so when the containers are spawned, it registers itself to the manager and the manager keeps the track of the instances; if a service is removed, the manager removes it from the manager's service registry. If other services need to communicate with each other, it contacts a discovery service to get the instance of another service. Again, this is a highly available component; if the discovery service is down, microservices can't communicate with each other, so the discovery service must have multiple instances.

3. **Components, aka services**: Components are the key ingredients in Microservice architecture. By component, I mean a utility or business feature which can be managed or updated independently. It has a predefined boundary and it exposes an API by which other components can communicate with this service. The idea of microservices is breaking down a complete business functionality into several independent small features which will communicate with each other to produce the total business functionality. If any portion of the functionality changes in the future, we can update or remove that component and add a new component to the architecture. So, Microservice architecture produces a proper modular architecture with proper encapsulation and properly defined boundaries.

4. [**Gateway Service**](http://javaonfly.blogspot.in/2017/08/microservices-communication-zuul-api.html)**:**A microservice is a collection of independent services which collectively produces a business functionality. Every microservice publishes an API, generally a REST API, so as a client, it is cumbersome to manage so many endpoint URLs to communicate with. Also, think about another perspective: if some application wants to build an authentication framework or security checking, they have to implement across all services, so that would be repeating itself against DRY. If we have a Gateway service, which is internet facing, the client will call only one endpoint and it delegates the call to an actual microservice, and all the authentication or security checking will be done in the gateway service.

Now we have a basic understanding how different parts of a microservice work together. In this tutorial, I will create an employee search service which will return employee information, an EmployeeDashBoard Service which will invoke the search service and show the results, a Eureka server so that these services can register themselves, and a gateway service to reach out to these services from outside. Then we will deploy our services in docker container and use DockerCompose to spawn the Docker containers. I will use Spring Boot for this tutorial.