**Microservice**

Each functionality is developed as a component and exposed as the restful webservice.This leads to high scalability and faster to develop

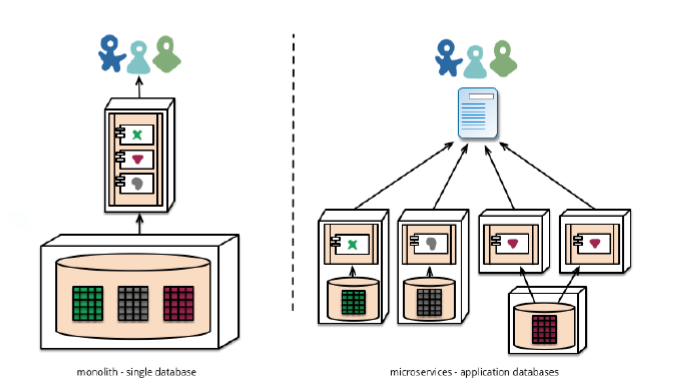
**Disadvantages of Monolothic Application**

* If there is a change in one component , after changing you have to deploy the entire application
* Presentation layer is tightly coupled
* The team have to undergo only one technology. This leads the situation where resources may not be available for that specific technology.
* Since all the components are in single application there will be the network traffic iand this leads to poor scalability
* If there is an issue in the service then the entire application fails

**Advantages of Micro services**

* Each Module can have its own architecture, technology and Db
* If there is a change in one component , then that component is changed and deployed.
* Presentaion layers are loosely coupled
* Each component is developed as a separate service and exposed as a webservice in a server that leads to good scalability
* Identifying resources and skillset of various technologies and makes more flexible to use
* All the components are developed at a time Independently and they are small. This follows the agile process and leads faster development
* One Webservice can be dependent on other webservice in loosely coupled manner
* If there is an issue in the service then oly that service fails but other services still works fine

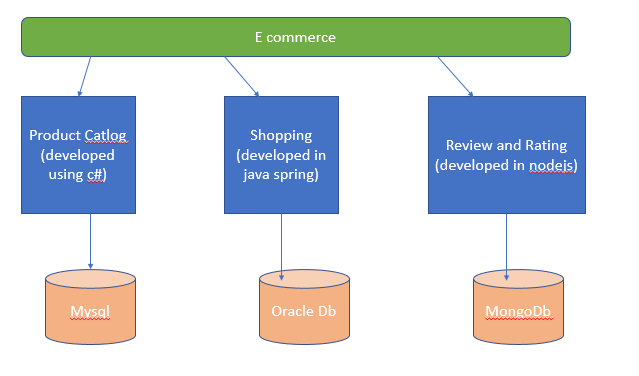
|  |  |
| --- | --- |
| **Monolothiic** | **Microservices** |
| Tightly coupled modules bundled as a single unit. | Loosely coupled independent services |
| The presentation, business layer and integration layer are combined as one unit. | Each functionality is a separate service |
| Deployed as a single war file | Each service is deployed as a set of service instances |



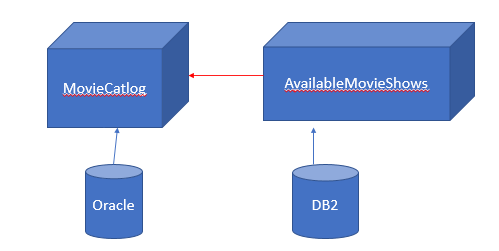
**How Micoservices developed**

* Identify the requirements , resources and skillset
* Break the requirements in to functionalities
* Based on the resources and skillset available say 4 java developers, 5 c# developers and 4 nodejs developers then one module can be developed in java, the other module in c# and other module using nodejs
* Each module have its own architecture , design and DB and deployed in different server
* If there is a change in design/implementation then only that module have to be changed.
* UI layer is developed in a different tier and this leads the layers loosely coupled, Say you can develop Admin functionalities in UI Layer in C#, Customer functionalities UI layer in Angular

**Each Module developed in different technology and DB**

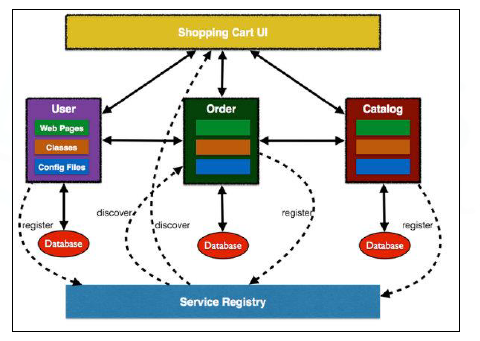


**Services Interact each other**



**Service Registration and Discovery server**

* It is a server which contains the available RESTful webservices UP running.
* Restful webservice is registered in the Service registration server to discover it.
* Eureka is the Service registration server provided by Netfix.
* If any client need to know the services running and its url to consume its services then they can discover using Service Registration.



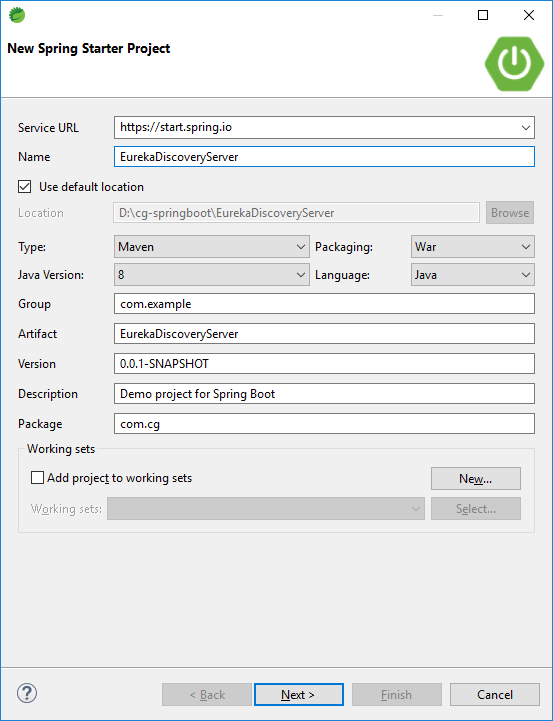
**To create the Eureka ServiceRegistration Server**

* Create the spring boot starter project
* Add the dependency Eureka Server
* Goto SpringBoot Application Main code and annotate @EnableEurekaserver

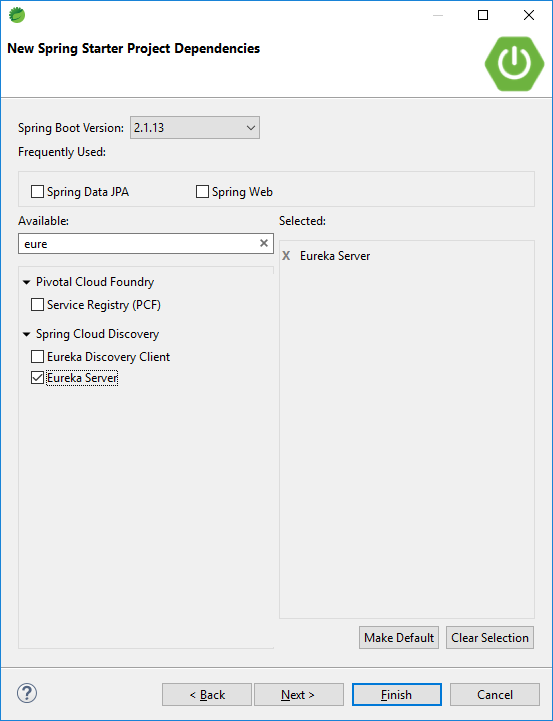
**@EnableEurekaServer**

* It launches the eureka Service registration server and it is ready to register any webservices to discover
* The default port no is 8761

**Create Spring boot starter project**



* Choose the dependency **EurekaServer**



Application.properties

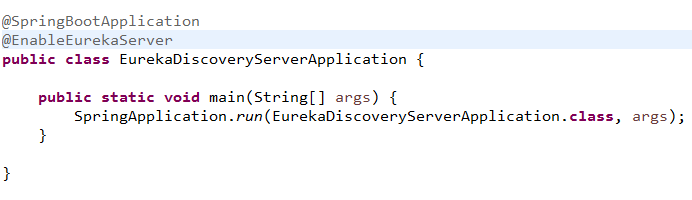
server.port=8761

spring.application.name=eureka-server

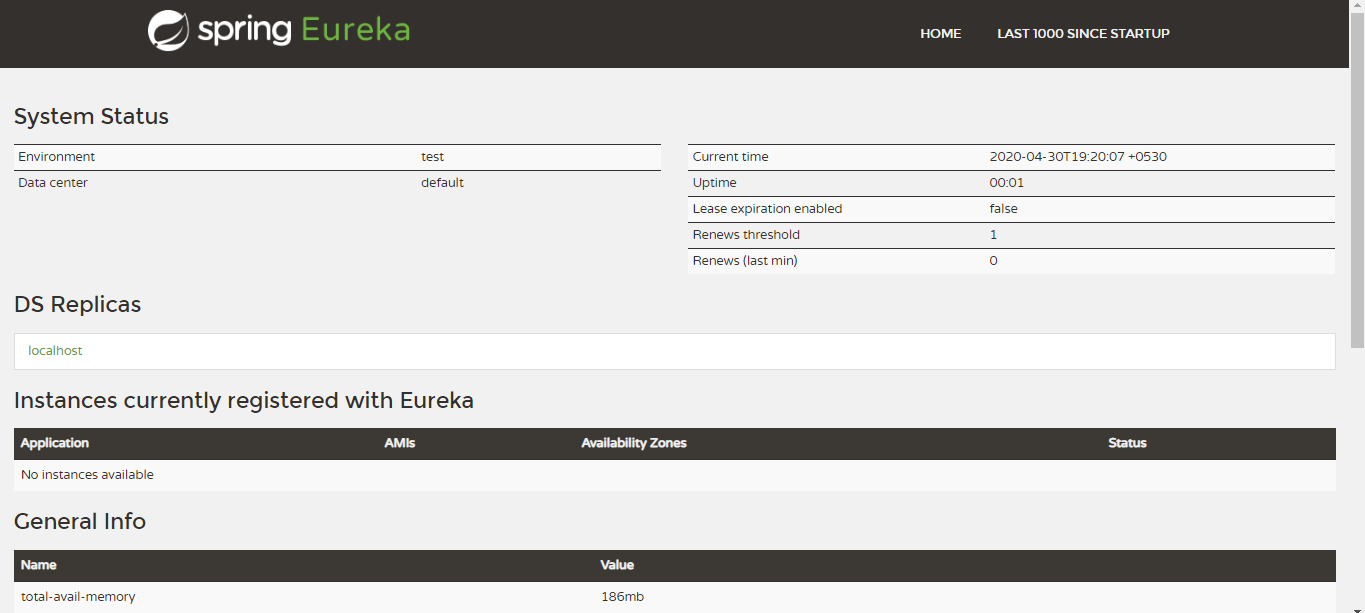
eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

* Annotate **@EnableEurekaServer** to Spring boot main code



* Type the url <http://localhost:8761> in the browser



**Register the webservices in Eureka service registration**

* Create a new Spring boot starter project where micro services available
* Add the following dependency in your Webservice project
* Annoatate **@EnableDiscoveryClient**

**@EnableDiscoveryClient**

* This annotation registers the microservices in Eureka Service Registration.
* To register the micro service in service Registration, provide the eureka server URL in application.properties
  + eureka.client.service-url.default-zone =http://localhost:8761/eureka

**How will you come to know all the microservices of your application are running?**

* Discovery in Eureka Service Registartion server

<properties>

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

<spring-cloud.version>Greenwich.SR5</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

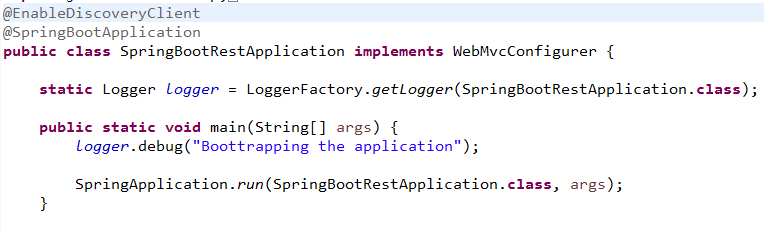
<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

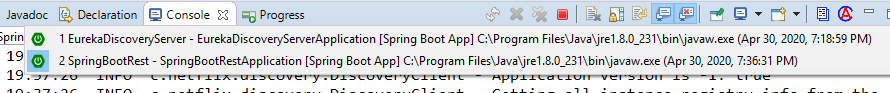
</dependencyManagement>

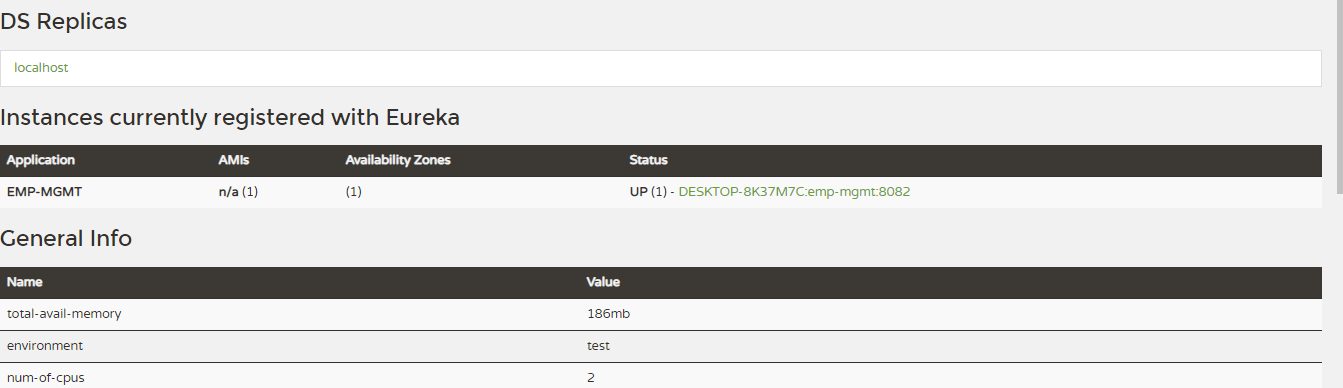


Application.properties

spring.application.name=emp-mgmt

eureka.client.service-url.default-zone=http://localhost:8761/eureka





**Registration of Second Micro service**

Add the following dependency in your Webservice project

<properties>

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

<spring-cloud.version>Greenwich.SR5</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</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>

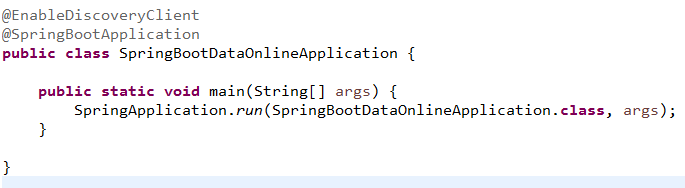
</dependencyManagement>

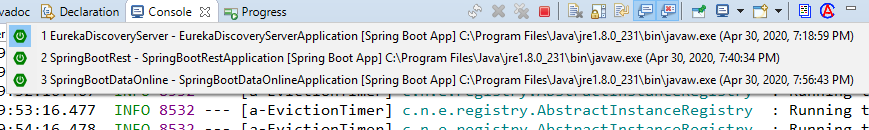
Application.properties

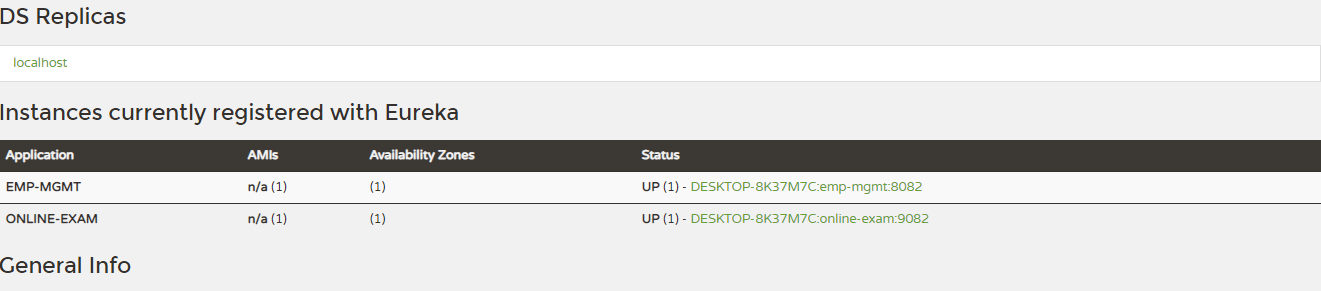
server.port=9082

spring.application.name=online-exam

eureka.client.service-url.default-zone=http://localhost:8761/eureka







**Hysrtix**

* Latency and fault tolerance library
* Also called as Circuit breaker
* Stops cascading failure
* Component of Netfix

**Why Hystrix?**

* If Service A is depenedent on Service B, If service B is down then it uses fallback tolerance logic instead of failing it.

Hystrix DAshBoard

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

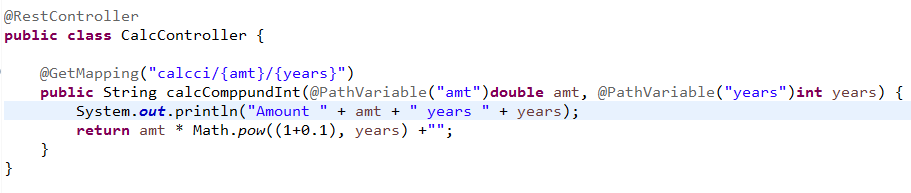
</dependency>

<dependency>

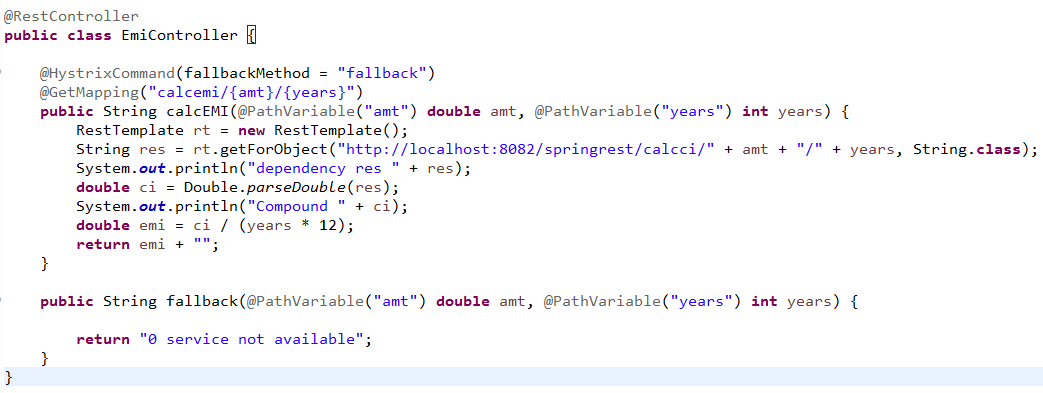
<groupId>org.springframework.cloud</groupId>

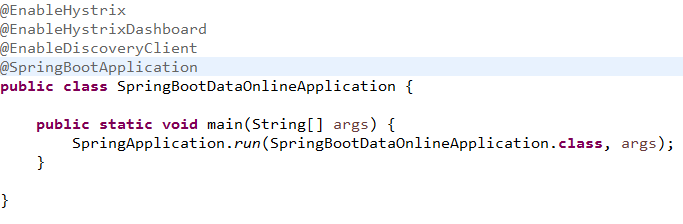
<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

**First Service of Project A running in Port 8082**



**Second Service of Project A running in Port 9082**



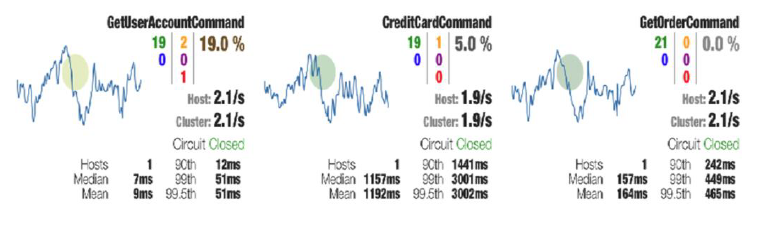


**@EnableHystrix**

* It provides the fault tolerance logic if dependency not available

**@EnableHystrixDashBoard**

* It provides the dashboard to monitor the Hystrix metrics in real time



Fallback method must have same return type and method parameters

**When Fallback() method is run?**

* If the dependent micro service is failed

**How a Micro service interact with another Micro service?**

* Using rest template