

# SPRING MICROSERVICES

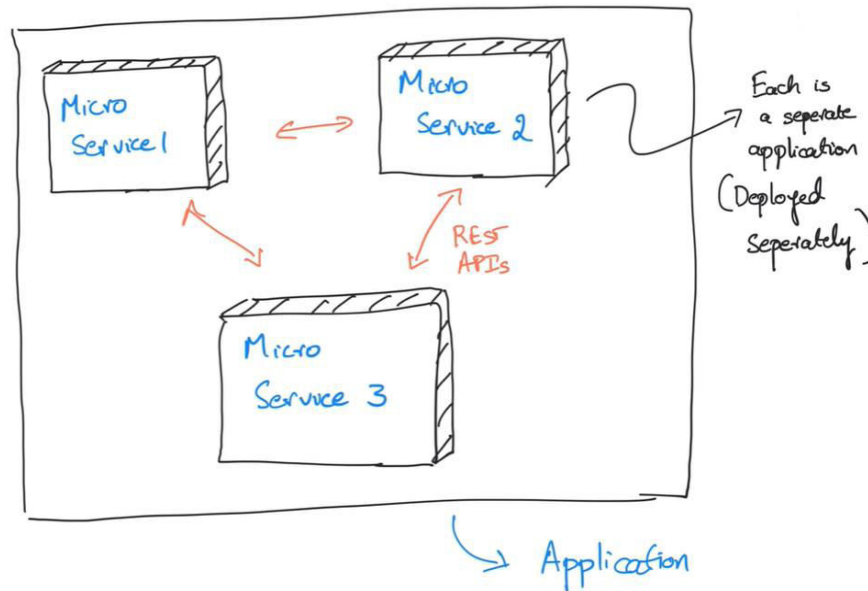
---

# Microservices introduction

- In a monolithic architecture, an application is delivered as a single deployable software artifact
  - All the UI (user interface), business, and database access logic are packaged together into a single application artifact and deployed to an application server.
- Microservices is an architecture wherein all the components of the system are put into individual components, which can be built, deployed, and scaled individually.
  - a particular way of designing software applications as suites of independently deployable services.
- Microservices are loosely coupled services which are combined within a software development architecture to create a structured application.
  - A microservices architecture allows the individual services to be deployed and scaled independently (typically via containers), worked on in parallel by different teams, built in different programming languages, and have their own continuous delivery and deployment stream.

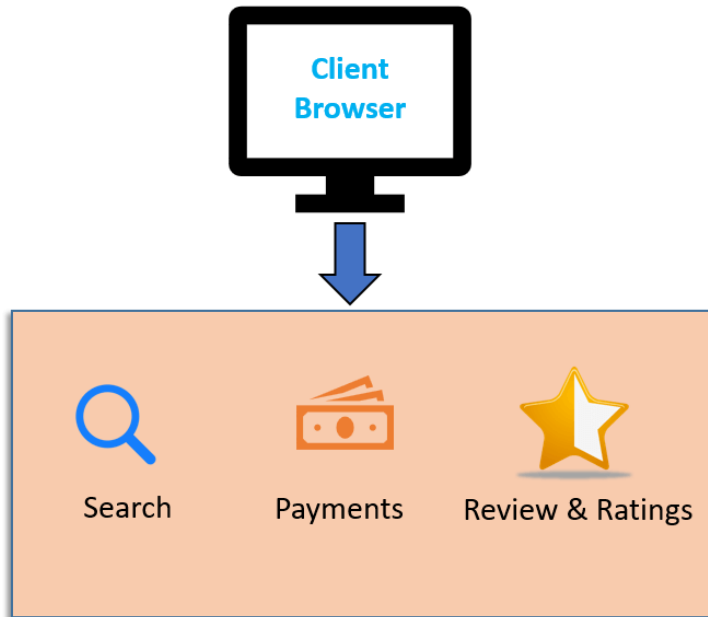
# Microservices introduction

- An architecture style used by many organizations today as a game changer to achieve a high degree of agility, speed of delivery, and scale. They give us a way to develop more physically separated modular applications.



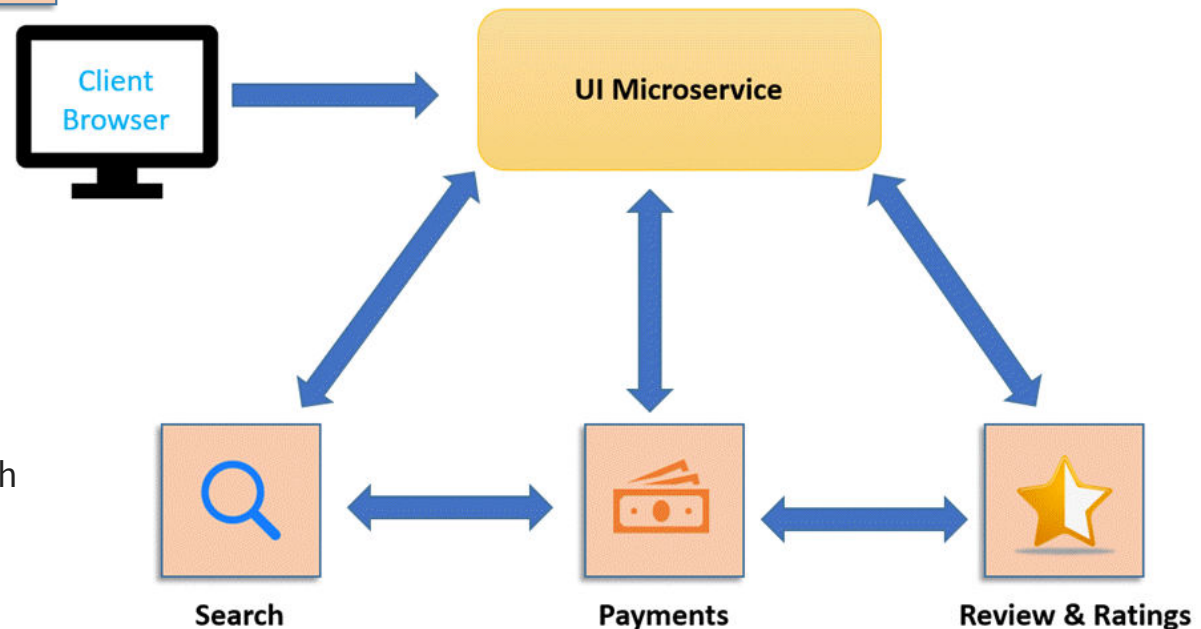
- Microservices are a way to break down your app into stand-alone independent mini applications that can be run on different hardware or server instances.
- They all talk to each other via REST API's.
- And work together to provide the functionality of your product

## Monolithic Architecture E-Commerce Application



- In any e-commerce application, there are some standard features like Search, Review & Ratings, and Payments.
- When the developer of the eCommerce site deploys the application, it is a single Monolithic unit.
- The code for different features like Search, Review & Ratings, and Payments are on the same server.
- To scale the application, you need to run multiple instances(servers) of these applications.

## Microservice Architecture E-Commerce Application



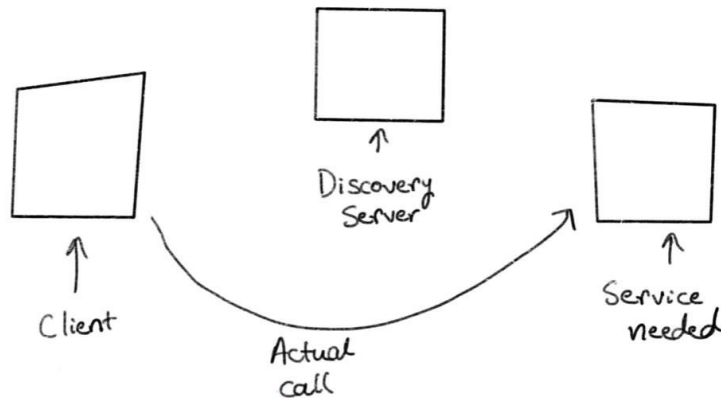
Each microservice is focused on single business capability. Search, Rating & Review and Payment each have their instance (server) and communicate with each other.

# Microservices

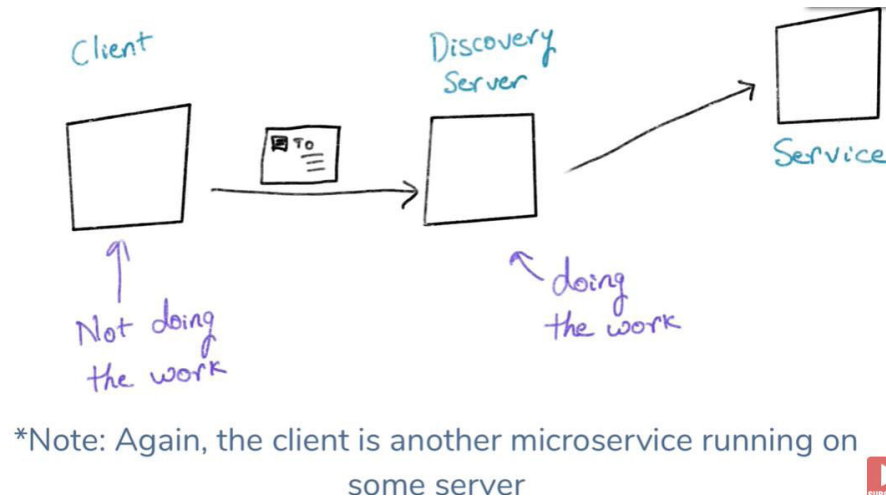
- The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an [HTTP resource](#) API.
  - Each of these services is responsible for discrete task and can communicate with other services through simple APIs to solve a larger complex business problem.
  - These services are built around business capabilities and independently deployable by fully automated deployment machinery.
  - There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.
- Microservices provide an approach for developing quick and agile applications, resulting in less overall cost.

# Service discovery

- There are 2 patterns to use for service discovery:
  - **Client side service discovery** : very popular. Client is also a microservice that uses the discovery MS to discover the MS it wants. Then makes actual call to the MS!
    - So, the client MS has to make 2 calls – one to the service registry and one to the actual MS it needs.
  - **Server side service discovery** :



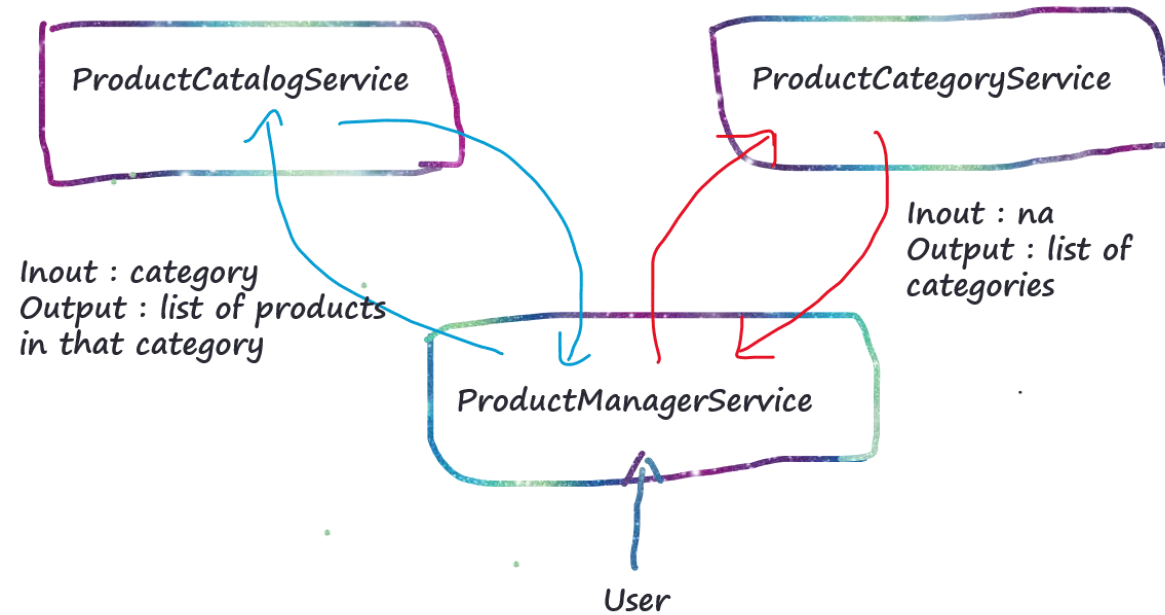
\*Note: client is also a microservice running on some server



\*Note: Again, the client is another microservice running on some server

Regardless of which model you use to locate service, you need to use the right technology. Popular one used for Client side service discovery is Eureka. For server side there are many solutions like NGINX and AWS

# First Demo using Spring Boot



- Use Spring Initializr

**Project**

☒ Maven Project ☐ Gradle Project

**Language**

☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**

☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (M2) ☐ 2.5.5 (SNAPSHOT) ☒ 2.5.4 ☐ 2.4.11 (SNAPSHOT) ☐ 2.4.10

**Project Metadata**

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 16 ☐ 11 ☒ 8

**Dependencies** ADD DEPENDENCIES... CTRL + B

**spring Web** WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

```

@RestController
@RequestMapping("/product")
public class ProductManagerResource {

    @RequestMapping("/{category}")
    public List<Product> getCatalog(@PathVariable("category") String category) {
        System.out.println("in contr : " + category);
        RestTemplate restTemplate = new RestTemplate();
        List<Product> list = Arrays.asList(new Product(101, "Prod-1", "cat1", 3000),
                                           new Product(102, "Prod-2", "cat2", 3500),
                                           new Product(103, "Prod-3", "cat1", 4000),
                                           new Product(104, "Prod-4", "cat2", 2000));

        return list;
    }
}

```

```

public class Product {
    int prodId;
    String pname;
    String cat;
    int price;
}

```

\*application.properties

```

server.port=8082

```

ProductCatalogService

- src/main/java
  - com.trg
    - ProductCatalogServiceApplication.java
    - com.trg.model
      - Product.java
    - com.trg.resource
      - ProductCatalogResource.java
  - src/main/resources

localhost:8082/product/aa

```

[
  - {
      prodId: 101,
      pname: "Prod-1",
      cat: "cat1",
      price: 3000
    },
  - {
      prodId: 102,
      pname: "Prod-2",
      cat: "cat2",
      price: 3500
    },
  - {
      prodId: 103,
      pname: "Prod-3",
      cat: "cat1",
      price: 4000
    },
  - {
      prodId: 104,
      pname: "Prod-4",
      cat: "cat2",
      price: 2000
    }
]

```



```

@Service
public class ProductService {

    List<Product> prodlist = new ArrayList<>();

    public ProductService() {
        prodlist.add(new Product(101, "Prod-1", "cat1", 3000));
        prodlist.add(new Product(102, "Prod-2", "cat2", 4000));
        prodlist.add(new Product(103, "Prod-3", "cat1", 2000));
        prodlist.add(new Product(104, "Prod-4", "cat2", 5000));
        prodlist.add(new Product(105, "Prod-5", "cat1", 3500));
    }

    public List<Product> getProductByCategory(String cat) {
        List<Product> plist = new ArrayList<>();

        for(Product p : prodlist){
            if(p.getCat().equals(cat))
                plist.add(p);
        }
        return plist;
    }
}

```

```

@RestController
@RequestMapping("/productCatalog")
public class ProductCatalogResource {

    @Autowired
    ProductService productService;

    @RequestMapping("/{category}")
    public List<Product> getCatalog(@PathVariable("category") String category) {
        List<Product> list = productService.getProductByCategory(category);
        return list;
    }
}

```

```

@RestController
@RequestMapping("/productManager")
public class ProductManagerResource {

    static final String REST_URI = "http://localhost:8083/productCatalog/";
    RestTemplate restTemplate = new RestTemplate();
    Product p = new Product();

    @RequestMapping("/{category}")
    public List<Product> getCatalog(@PathVariable("category") String category) {
        List<Product> filterList = new ArrayList<Product>();

        List<LinkedHashMap<String, Object>> products =
            restTemplate.getForObject(REST_URI + category, List.class);

        if (products != null) {
            for (LinkedHashMap<String, Object> map : products){
                p.setProdId((Integer)map.get("prodId"));
                p.setName((String)map.get("pname"));
                p.setCat((String)map.get("cat"));
                p.setPrice((Integer)map.get("price"));
                filterList.add(p);
            }
        } else {
            System.out.println("No course exists-----");
            return filterList;
        }
    }
}

```

← → ↻ ⓘ localhost:8082/productManager/cat2

```

[
  - {
    prodId: 104,
    pname: "Prod-4",
    cat: "cat2",
    price: 5000
  },
  - {
    prodId: 104,
    pname: "Prod-4",
    cat: "cat2",
    price: 5000
  }
]

```

```

@SpringBootApplication
public class ProductManagerServiceApplication {

    @Bean
    public RestTemplate getRestTemplate(){
        return new RestTemplate();
    }

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

```

```

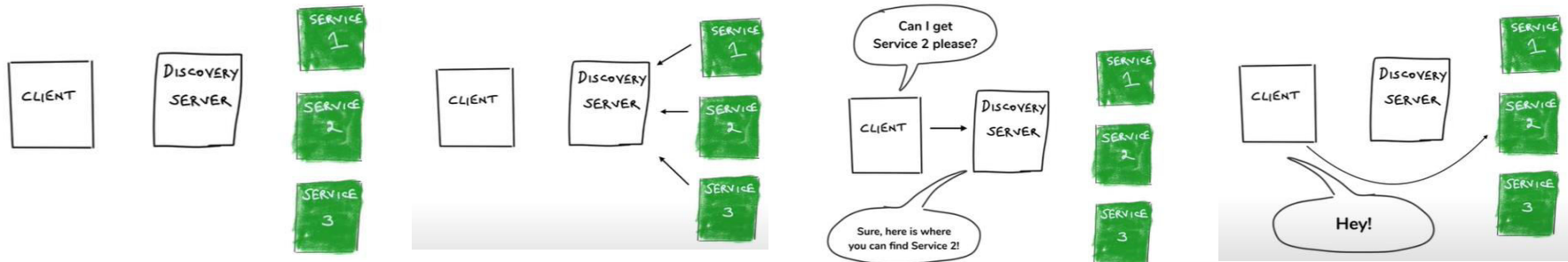
@RestController
@RequestMapping("/productManager")
public class ProductManagerResource {

    @Autowired
    RestTemplate restTemplate;
}

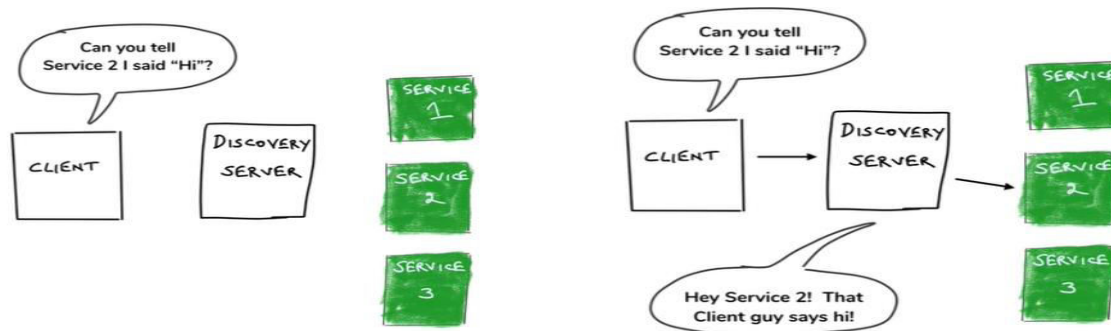
```

# Understanding Service Discovery

- Client side service discovery



- Server side service discovery



# Understanding Service Discovery

- Spring Cloud
  - Spring Cloud is a [new project](#) in the [spring.io family](#) with a set of components
  - Spring Cloud framework provides tools for developers to build a robust cloud application quickly.
  - We can also build the microservice-based applications, for example, configuration management, **service discovery**, circuit breakers, intelligent routing, cluster state, micro-proxy, a control bus, one time tokens, etc.
  - To a large extent Spring Cloud 1.0 is based on components from [Netflix OSS](#). Spring Cloud integrates the Netflix components in the Spring environment in a very nice way using auto configuration and convention over configuration similar to how [Spring Boot](#) works.

Netflix Component Name	Functionality
Eureka	Service Registration and Discovery
Ribbon	Client Side Load Balancing
Hystrix	Circuit Breaker
Zuul	Edge Server, Intelligent Routing

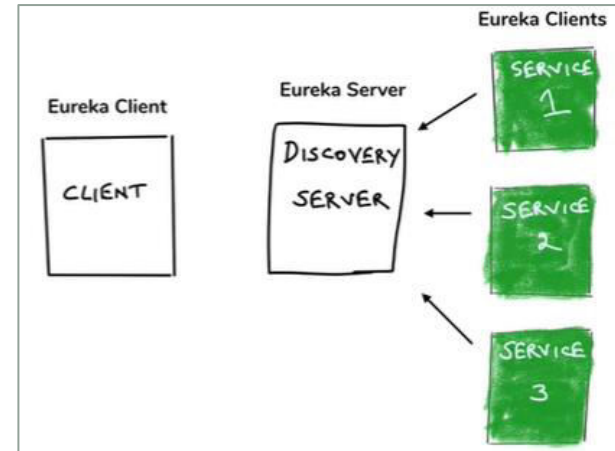
- Eureka is a REST based service that is primarily used in the AWS cloud for locating services for the purpose of load balancing and failover of middle-tier servers. **We call this service, the Eureka Server.**
- Eureka also comes with a Java-based client component, the **Eureka Client**, which makes interactions with the service much easier.
- Eureka Server is an application that holds the information about all client-service applications.
- Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address.
- Eureka Server is also known as **Discovery Server**.
- **Eureka Server comes with the bundle of Spring Cloud**. And is a open source project from Netflix

# Understanding Service Discovery

- Spring Cloud uses Client side service discovery
  - ***Client-side service discovery* allows services to find and communicate with each other without hard-coding hostname and port.**
  - The only 'fixed point' in such an architecture consists of a *service registry* with which each service has to register.
- The Technology that implements Service discovery in Spring Cloud is **Eureka**.
- There are lot of other technologies that Spring integrates with, but Eureka is mostly the technology of choice

# Spring Cloud Netflix

- Eureka Server comes with the bundle of Spring Cloud.
- Steps to make Service discovery work:
  - Start up a Eureka server
  - Have microservices register (publish) using Eureka client
  - Have microservices locate (consume) using Eureka client



**Project**

☒ Maven Project ☐ Gradle Project

**Language**

☒ Java ☐ Kotlin

☐ Groovy

**Spring Boot**

☐ 2.4.0 (SNAPSHOT) ☐ 2.4.0 (M3)

☐ 2.3.5 (SNAPSHOT) ☒ 2.3.4 ☐ 2.2.11 (SNAPSHOT)

☐ 2.2.10 ☐ 2.1.18 (SNAPSHOT) ☐ 2.1.17

**Project Metadata**

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 15 ☐ 11 ☒ 8

## Dependencies

ADD DEPENDENCIES... CTRL + B

### Eureka Server

SPRING CLOUD DISCOVERY

spring-cloud-netflix Eureka Server.

In Spring Initializer download the Spring Boot project with Eureka server dependency

Note: Use a pre 2.4 version for starter-parent

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</art
  <version>2.3.0.RELEASE</version>
  <relativePath/> <!-- lookup parent from rep
</parent>
<groupId>com.trg</groupId>
<artifactId>Product-discovery-server1</artifact
<version>0.0.1-SNAPSHOT</version>
<name>Product-discovery-server1</name>
<description>Demo project for Spring Rest</desc
<properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>Hoxton.SR4</spring-cl
  <maven-jar-plugin.version>3.1.1</maven-jar-
</properties>
```

# Building a Eureka Server

- Add `@EnableEurekaServer` annotation to Spring Boot Application class
  - The `@EnableEurekaServer` annotation is used to make your Spring Boot application acts as a Eureka Server.

```
@SpringBootApplication
@EnableEurekaServer
public class ProductDiscoveryServerApplication {

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

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



# Building a Eureka Server

- By default, the Eureka Server registers itself into the discovery.
- You should add the below given configuration into your application.properties file or application.yml file.

```
#application.properties
server.port=8761
# so that Eureka doesnt register with itself
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

```
#application.yml file
eureka:
  client:
    registerWithEureka: false
    fetchRegistry: false
  server:
    port: 8761
```

- `eureka.client.register-with-eureka = false` - Makes it so the server does not attempt to register itself.
- `eureka.client.fetch-registry = false` - With this, we inform customers that they must not store the data of the available instances in their local cache. This means that customers must consult the Eureka server whenever they need to access a service. In production, this is often set to true to expedite requests. This cache is updated every 30 seconds by default.

**If you try running this on Java 11, you will get errors. Because from Java 9/10 onwards, support for JAXB is taken off. To resolve, I have to bring other following dependencies into pom.xml**

←

→

↺

localhost:8761

🔍

☆


ABP

▶▶

🛑

🌐

⚙️



HOME    LAST 1000 SINCE STARTUP

### System Status

Environment	test
Data center	default

Current time	2021-08-22T09:08:34 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

### DS Replicas

localhost

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

### General Info

# Client app

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.3.0.RELEASE</version>
  <relativePath /> <!-- lookup parent from repository -->
</parent>
<groupId>com.trg</groupId>
<artifactId>ProductCatalogService</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>ProductCatalogService</name>
<description>Demo project for Spring Rest</description>
<properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>Hoxton.SR4</spring-cloud.version>
  <maven-jar-plugin.version>3.1.1</maven-jar-plugin.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>
</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>
```

# Client app

← → ↻ ⓘ localhost:8761 🔍 ☆

## System Status

Environment	test	Current time	2021-0
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	3
		Renews (last min)	0

## DS Replicas

localhost


## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
UNKNOWN	n/a (1)	(1)	UP (1) - LAPTOP-CG2KSI6E:8083

```
application.proper... application.proper... ✕ ProductDiscoverySer...  
1 server.port=8083  
2 spring.application.name=product-catalog-service
```

```
application.proper... ✕ application.proper... ProductDiscoverySer...  
1 server.port=8082  
2 spring.application.name=product-manager-service
```

← → ↺ ⓘ localhost:8761 🔍 ☆ ABP ⏮

 **spring** Eureka

HOME LAST 1000

## System Status

Environment	test	Current time	2021-08-22T11:17:10
Data center	default	Uptime	00:25
		Lease expiration enabled	true
		Renews threshold	5
		Renews (last min)	8

## DS Replicas

localhost

## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PRODUCT-CATALOG-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-CG2KSI6E:product-catalog-service:8083
PRODUCT-MANAGER-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-CG2KSI6E:product-manager-service:8082

```

@RestController
@RequestMapping("/productManager")
public class ProductManagerResource {

    @Autowired
    RestTemplate restTemplate;

    static final String REST_URI = "http://localhost:8083/productCatalog/";

```

### Instances currently registered with Eureka

Application	AMIs	Availability Zon
PRODUCT-CATALOG-SERVICE	n/a (1)	(1)
PRODUCT-MANAGER-SERVICE	n/a (1)	(1)

```

@SpringBootApplication
public class ProductManagerServiceApplication {

    @Bean
    @LoadBalanced
    public RestTemplate getRestTemplate(){
        return new RestTemplate();
    }

```

localhost:8082/productManager/cat2

```

[
  - {
      prodId: 104,
      pname: "Prod-4",
      cat: "cat2",
      price: 5000
    },
  - {
      prodId: 104,
      pname: "Prod-4",
      cat: "cat2",
      price: 5000
    }
]

```

# Invoking REST services programmatically

- RestTemplate : is the central class within the Spring framework for executing synchronous HTTP requests on the client side.
  - RestTemplate provides higher level methods that correspond to each of the six main HTTP methods that make invoking many RESTful services a one-liner and enforce REST best practices.

HTTP Method	REST Template Method	Description
POST	postForObject	Post object and expect its instance as a response.
	postForEntity	Post object and expect a response of ResponseEntity type.
	postForLocation	Post object and read location of it.
GET	getForObject	Read the object based on a given URL.
	getForEntity	Read the ResponseEntity instance based on a given URI.
PUT	put	Perform the PUT of a given object against a given URI.
DELETE	delete	Delete objects based on a given URI.
HEAD	headForHeaders	Read headers based on a given URL and return the HttpHeaders object.
OPTIONS	optionsForAllow	Return the value of the Allow header for a given URI.
Any	Exchange	Perform any HTTP method against a server and exchange HttpEntity and ResponseEntity instances.



```

public class CourseRESTClient {
    static final String REST_URI = "http://localhost:8081/course";
    static RestTemplate restTemplate = new RestTemplate();

    private static void listAllCourses() {
        System.out.println("\n Testing listAllPersons API-----");
        List<LinkedHashMap<String, Object>> coursesMap =
            restTemplate.getForObject(REST_URI + "/courses", List.class);

        if (coursesMap != null) {
            for (LinkedHashMap<String, Object> map : coursesMap)
                System.out.println("Course : id=" + map.get("id") +
                                   ", name=" + map.get("name") +
                                   ", Desc=" + map.get("desc"));
        } else
            System.out.println("No course exists-----");
    }

    private static void getCourse(int id) {
        System.out.println("\n Testing getPerson API-----");
        Course course =
            restTemplate.getForObject(REST_URI + "/courses/" + id, Course.class);
        System.out.println(course);
    }

    private static void createCourse(Course c) {
        System.out.println("\n Testing create Course API-----");
        Course course =
            restTemplate.postForObject(REST_URI + "/courses", c, Course.class);
        System.out.println("Newly created course : " + course);
    }

    public static void main(String args[]) {
        listAllCourses();
        getCourse(101);
        Course c = new Course(105, "ReactJS", "React desc");
        createCourse(c);
    }
}

```

```

Testing listAllPersons API-----
00:33:31.602 [main] DEBUG org.springframework.web.client.R
00:33:31.631 [main] DEBUG org.springframework.web.client.R
00:33:31.650 [main] DEBUG org.springframework.web.client.R
00:33:31.654 [main] DEBUG org.springframework.web.client.R
Course : id=101, name=Spring, Desc=Spring quickstart
Course : id=102, name=Java, Desc=Java fundamentals
Course : id=103, name=NodeJS, Desc=Node essentials
Course : id=105, name=ReactJS, Desc=React desc

Testing getPerson API-----
00:33:31.688 [main] DEBUG org.springframework.web.client.R
00:33:31.725 [main] DEBUG org.springframework.web.client.R
00:33:31.729 [main] DEBUG org.springframework.web.client.R
00:33:31.729 [main] DEBUG org.springframework.web.client.R
Course [id=101, name=Spring, desc=Spring quickstart]

Testing create Course API-----
00:33:31.789 [main] DEBUG org.springframework.web.client.R
00:33:31.789 [main] DEBUG org.springframework.web.client.R
00:33:31.793 [main] DEBUG org.springframework.web.client.R
00:33:31.803 [main] DEBUG org.springframework.web.client.R
00:33:31.803 [main] DEBUG org.springframework.web.client.R
Newly created course : Course [id=105, name=ReactJS, desc=

Testing listAllPersons API-----
00:33:31.805 [main] DEBUG org.springframework.web.client.R
00:33:31.805 [main] DEBUG org.springframework.web.client.R
00:33:31.807 [main] DEBUG org.springframework.web.client.R
00:33:31.807 [main] DEBUG org.springframework.web.client.R
Course : id=101, name=Spring, Desc=Spring quickstart
Course : id=102, name=Java, Desc=Java fundamentals
Course : id=103, name=NodeJS, Desc=Node essentials
Course : id=105, name=ReactJS, Desc=React desc

```