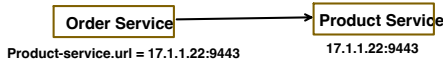


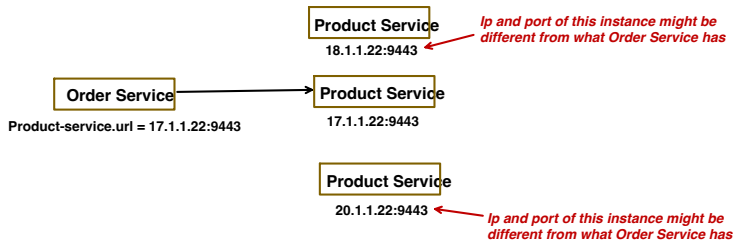
- In Microservices, services or component instances are created and deleted dynamically.
- We can not hardcode the URL of a particular instance, its not scalable and feasible.



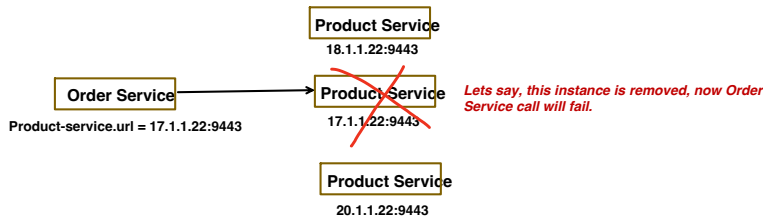
In above example:

- Order service has hardcoded the URL of the Product service and its ok till we have only 1 instance of Product service.

- But in case, when we have multiple Product service instances then....



Also, possible that, particular instance is removed



Now in above example, where multiple instances of Product service is present and Order Service has hardcoded the URL of the Product service, the problem it might face is:

- Single point of failure:

If hardcoded instance of Product Service goes down, Order service will not be able to communicate with any other instance.

- No Load Balancing:

Only one instance of Product Service get overburdened while other instances remain idle.

- Tight Coupling:

Because of hardcoded URL in Order Service, there is tight coupling between Product and Order Service, as without updating the Order Service its not possible to change or move Product Service.

• Difficulty in testing:

Different environment (say production, QA, dev) might uses different URL, which require frequent changes in config and not only cause difficulty in testing but prone to error too.

Solution for above problem is:

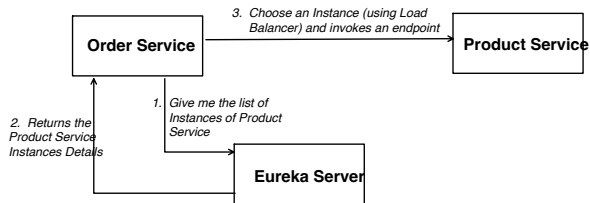
**SERVICE DISCOVERY (like Eureka)**

**Eureka Server**

- Act like a phonebook.
- Has all the instances info of all the registered clients like:
  - Service name
  - Instance id
  - IP
  - Port number
  - Health status
  - etc.

**Eureka Client**

- Register itself with the Server.
- Discovers an instance of other service via Eureka Server



## Lets first set up Eureka Server Application:

Go to Spring Initializer ([start.spring.io](https://start.spring.io))

Project	Language	Dependencies
<input type="radio"/> Gradle - Groovy <input checked="" type="radio"/> Maven	<input checked="" type="radio"/> Java <input type="radio"/> Kotlin <input type="radio"/> Groovy	Spring Web <b>Web</b> <small>Build web, including RESTful applications using Spring MVC. default embedded container.</small>
Spring Boot <input type="radio"/> 4.0.0 (SNAPSHOT) <input type="radio"/> 3.5.1 (SNAPSHOT) <input type="radio"/> 3.5.0 <input type="radio"/> 3.4.7 (SNAPSHOT) <input checked="" type="radio"/> 3.4.5 <input type="radio"/> 3.3.13 (SNAPSHOT) <input type="radio"/> 3.3.12		
<b>Project Metadata</b> Group: <input type="text" value="com.example"/> Artifact: <input type="text" value="EurekaServer"/> Name: <input type="text" value="EurekaServer"/> Description: <input type="text" value="Learning of Service Discovery"/> Package name: <input type="text" value="com.example.EurekaServer"/> Packaging: <input checked="" type="radio"/> Jar <input type="radio"/> War Java: <input type="radio"/> 24 <input checked="" type="radio"/> 21 <input type="radio"/> 17		

### 1. pom.xml

```

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

```

Version for "spring-cloud-starter-netflix-eureka-server" will be automatically resolved by below dependency management.

```

<dependencyManagement>
<dependencies>
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-dependencies</artifactId>
<version>2023.0.1</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>

```

### 2. Enable Eureka Server functionality.

```

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

    public static void main(String[] args) {

        SpringApplication.run(EurekaServerApplication.class, args);
    }
}

```

Tells Spring boot to create necessary beans, which is required for Eureka Server like:

- EurekaController
- Dashboard etc.

### 3. application.properties

```

spring.application.name=eureka-server
server.port=8761

# Since it's a server, we don't want it to register and
# also don't want to fetch the instances details
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

```

Lets start the Server and see the dashboard:

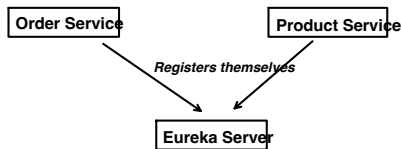
```
2025-06-12T15:22:06.250+05:30 INFO 43767 --- [eureka-server] [    main] o.s.b.e.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actua
2025-06-12T15:22:06.278+05:30 INFO 43767 --- [eureka-server] [    main] o.s.c.n.e.s.EurekaServiceRegistry : Registering application EUREKA-SERVER with eurek
2025-06-12T15:22:06.274+05:30 INFO 43767 --- [eureka-server] [Thread-9] o.s.c.n.e.server.EurekaServerBootstrap : isAs returned false
2025-06-12T15:22:06.274+05:30 INFO 43767 --- [eureka-server] [Thread-9] o.s.c.n.e.server.EurekaServerBootstrap : Initialized server context
2025-06-12T15:22:06.274+05:30 INFO 43767 --- [eureka-server] [Thread-9] c.n.e.r.PeerAwareInstanceRegistryImpl : Got 1 instances from neighboring DS node
2025-06-12T15:22:06.274+05:30 INFO 43767 --- [eureka-server] [Thread-9] c.n.e.r.PeerAwareInstanceRegistryImpl : Renew threshold is: 1
2025-06-12T15:22:06.274+05:30 INFO 43767 --- [eureka-server] [Thread-9] c.n.e.r.PeerAwareInstanceRegistryImpl : Changing status to UP
2025-06-12T15:22:06.274+05:30 INFO 43767 --- [eureka-server] [    main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8761 (http) with context
2025-06-12T15:22:06.274+05:30 INFO 43767 --- [eureka-server] [    main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8761
2025-06-12T15:22:06.278+05:30 INFO 43767 --- [eureka-server] [Thread-9] e.s.EurekaServerInitializerConfiguration : Started Eureka Server
2025-06-12T15:22:06.291+05:30 INFO 43767 --- [eureka-server] [    main] c.e.t.EurekaServerApplication : Started EurekaServerApplication in 1.513 seconds
```

<http://localhost:8761>

localhost:8761

</

Now lets set up Eureka Client Application:



Product Service

1. pom.xml

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

Version for "spring-cloud-starter-netflix-eureka-client" will be automatically resolved by below dependency management.

```
<dependencyManagement>
<dependencies>
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-dependencies</artifactId>
<version>2023.0.1</version>
<type>pom</type>
<scope>import</scope>
</dependency>
</dependencies>
</dependencyManagement>
```

2. application.properties

```
server.port=8082
spring.application.name=product-service

#path of the Eureka Server
eureka.client.service-url.defaultZone=http://localhost:8761/eureka

#by-default values are true only, so we can even skip below configs
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
```

Lets start the Product Service Server and see the dashboard again:

```
2025-06-12T15:42:49.284+05:30 INFO 43993 --- [product-service] [main] com.netflix.discovery.DiscoveryClient : Starting heartbeat executor: renew interval is: 30
2025-06-12T15:42:49.284+05:30 INFO 43993 --- [product-service] [main] e.n.discovery.InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowed rate per min: 1.0
2025-06-12T15:42:49.284+05:30 INFO 43993 --- [product-service] [main] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 1749725169207 with local status change event StatusChangeEvent(timestamp=2025-06-12T15:42:49.284+05:30,ipAddress=192.168.1.101,hostname=localhost,preferredInstance=PRODUCT-SERVICE,leaseDurationMinutes=30,leaseRenewalIntervalSeconds=30,leaseExpirationTimeInSeconds=0,instanceId=192.168.1.101:product-service-8082,instanceGroup=PRODUCT-SERVICE,instanceMetadata={})
2025-06-12T15:42:49.284+05:30 INFO 43993 --- [product-service] [main] o.s.c.n.e.s.EurekaServiceRegistry : Registering application PRODUCT-SERVICE with eureka with status UP
2025-06-12T15:42:49.284+05:30 INFO 43993 --- [product-service] [main] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent(timestamp=2025-06-12T15:42:49.284+05:30,ipAddress=192.168.1.101,hostname=localhost,preferredInstance=PRODUCT-SERVICE,leaseDurationMinutes=30,leaseRenewalIntervalSeconds=30,leaseExpirationTimeInSeconds=0,instanceId=192.168.1.101:product-service-8082,instanceGroup=PRODUCT-SERVICE,instanceMetadata={})
2025-06-12T15:42:49.284+05:30 INFO 43993 --- [product-service] [main] com.netflix.discovery.DiscoveryClient : DiscoveryClient_PRODUCT-SERVICE/192.168.1.101:product-service-8082
2025-06-12T15:42:49.218+05:30 INFO 43993 --- [product-service] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8082 (http) with context path ''
2025-06-12T15:42:49.218+05:30 INFO 43993 --- [product-service] [main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8082
2025-06-12T15:42:49.225+05:30 INFO 43993 --- [product-service] [main] e.c.s.ProductServiceApplication : Started ProductServiceApplication in 1.034 seconds (process identifier pid=43993)
2025-06-12T15:42:49.229+05:30 INFO 43993 --- [product-service] [main] com.netflix.discovery.DiscoveryClient : DiscoveryClient_PRODUCT-SERVICE/192.168.1.101:product-service-8082
```

localhost:8761

**spring Eureka** HOME LAST 1000 SINCE STARTUP

### System Status

Environment	test	Current time	2025-06-12T15:45:35 +0530
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	3
		Renews (last min)	0

### DS Replicas

localhost

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.101:product-service:8082

### General Info

Similarly for Order Service, we can register it with Eureka Server

localhost:8761

**spring Eureka** HOME LAST 1000 SINCE STARTUP

### System Status

Environment	test	Current time	2025-06-12T15:48:29 +0530
Data center	default	Uptime	00:03
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	4

### DS Replicas

localhost

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ORDER-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.101:order-service:8081
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.1.101:product-service:8082

### General Info

Now lets see, how Order Service can invoke Product Service:

## Using RestTemplate

Without Service Discovery

```
public void callProductAPI(String id) {  
    RestTemplate restTemplate = new RestTemplate();  
    String response = restTemplate.getForObject("http://localhost:8082/products/"+id, String.class);  
    System.out.println("Response from Product api call is: " + response);  
}
```

*Specifically mentioning the URL*

## With Service Discovery

```
Import org.springframework.cloud.client.ServiceInstance;
Import org.springframework.cloud.client.discovery.DiscoveryClient;
```

```
@Autowired
DiscoveryClient discoveryClient;

public void callProductAPI(String id) {

    RestTemplate restTemplate = new RestTemplate();
    List<ServiceInstance> instances = discoveryClient.getInstances( serviceId: "product-service");
    URI uri = instances.get(0).getUri();
    String response = restTemplate.getForObject( uri: uri + "/products/" + id, String.class);

    System.out.println("Response from Product api call is: " + response);
}
```

1. Fetching the instances of "product-service"

2. load balancer logic to choose a particular instance.

So with RestTemplate, load balancing (choosing an instance for the product service) logic need to be handled.

## Using FeignClient

So with FeignClient, load balancing is handled automatically and by the framework.

So we need to provide the Load Balancer dependency too, apart from "spring-cloud-starter-netflix-eureka-client"

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

Earlier without Service Discovery:

With Service Discovery:

```
@FeignClient(name = "product-service",
    url = "${feign.client.product-service.url}")
public interface ProductClient {

    @GetMapping(value = "/products/{id}")
    String getProductById(@PathVariable("id") String id);
}
```

No need to provide the URL

```
@FeignClient(name = "product-service")
public interface ProductClient {

    @GetMapping(value = "/products/{id}")
    String getProductById(@PathVariable("id") String id);
}
```

No need of the URL, only the registered name of the product service is required and Load Balancing will also be handled internally by the framework.

## As a curious engineer, few questions comes to our mind:

### Service Registration doubts:

1. How does Eureka Server know whether a client is UP or DOWN?
2. Where and how the data is stored?
3. What if Eureka Server itself goes down? Is it a single point of failure?

### Discovery Doubts:

1. It can cause latency issue, as each call now required 2 hops, first it has to invoke Eureka Server and then the actual call.
2. What if the local cache is stale? Can this lead to calling a dead instance?

Lets try to find an answer one by one:

### 1. How does Eureka Server know whether a client is UP or DOWN?

Through Client de-registration request

Through Client Heart Beat

When client application is gracefully shut down, then eureka client sends the de-registration request to Eureka Server, it mark client status as DOWN.

System Status			
Environment	test	Current time	2025-06-13T11:56:15 +0530
Data center	default	Uptime	00:00
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	0
THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.			
DS Replicas			
localhost			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.37:product-service:8082

Client(product-service) application logs :

robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Getting all Instance registry info from the eureka server
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: The response status is 200
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Starting heartbeat executor: renew interval is: 30
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: InstanceInfoReplicator onDemand update allowed rate per min is: 4
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Discovery Client initialized at timestamp: 1747796222399 with initial instances count: 0
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Registering application PRODUCT-SERVICE with eureka with status UP
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Saw local status change event StatusChangeEvent [timestamp=1747796222395, current=UP, previous=STARTING]
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: DiscoveryClient_PRODUCT-SERVICE/192.168.0.37:product-service:8082: registering service...
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Instance started on port 8082 (http) with context path: /
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Updating port to 8082
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: DiscoveryClient_PRODUCT-SERVICE/192.168.0.37:product-service:8082: registration status: 204
robot-service [	main]	com.netflix.discovery.DiscoveryClient	: Started ProductServiceApplication in 0.979 seconds (Tomcat startup time: 1.133)
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: Re-registering application PRODUCT-SERVICE with eureka with status DOWN
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: Saw local status change event StatusChangeEvent [timestamp=1747796226139, current=DOWN, previous=UP]
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: DiscoveryClient_PRODUCT-SERVICE/192.168.0.37:product-service:8082: registering service...
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: DiscoveryClient_PRODUCT-SERVICE/192.168.0.37:product-service:8082: registration status: 204
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: Shutting down DiscoveryClient ...
robot-service [	infoReplicator-0]	com.netflix.discovery.DiscoveryClient	: Re-registering ...

Eureka Client periodically sends Heart Beat to the Eureka Server.

Lets say, if client shut down without sending any de-registration request (bcoz of Network issue).

then Eureka server wait for the heart beat from Client for a particular interval (decided at the time of registration) and if no Heart Beat received within that time interval, it remove the client instance itself.

Client(product-service) **application.properties**:

server.port=8082
spring.application.name=product-service
#path of the Eureka Server
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
#by-default values are true only, so we can even skip below configs
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
#every 60seconds client will send the heart beat to server
eureka.instance.lease-renewal-interval-in-seconds=60
#telling server to wait this much time, if no heart beat received in this time, then remove me.
#for testing, kept it less than lease-renewal-interval
eureka.instance.lease-expiration-duration-in-seconds=5

After this much seconds, client will send the new heart beat to server. Default time is 30sec

Telling server to wait only this much time for the heart beat. After that, you can remove it from the list. Default time is 90sec

Server (Eureka-server) application logs :

[eureka-server] [s-0-0-0-0-0]	com.netflix.discovery.DiscoveryClient	: Running the evict task with compensationTime 0ms
[eureka-server] [s-0-0-0-0-0]	com.netflix.discovery.DiscoveryClient	: Running the evict task with compensationTime 0ms
[eureka-server] [s-0-0-0-0-0]	com.netflix.discovery.DiscoveryClient	: Registered instance PRODUCT-SERVICE/192.168.0.37:product-service:8082 with status UP (replication=false)
[eureka-server] [s-0-0-0-0-0]	com.netflix.discovery.DiscoveryClient	: Registered instance PRODUCT-SERVICE/192.168.0.37:product-service:8082 with status UP (replication=true)
[eureka-server] [s-0-0-0-0-0]	com.netflix.discovery.DiscoveryClient	: Registered instance PRODUCT-SERVICE/192.168.0.37:product-service:8082 with status DOWN (replication=false)

System Status			
Environment	test	Current time	2025-06-13T11:56:33 +0530
Data center	default	Uptime	00:00
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	0
THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.			
DS Replicas			
localhost			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
PRODUCT-SERVICE	n/a (1)	(1)	DOWN (1) - 192.168.0.37:product-service:8082

Eureka Server **application.properties**:

spring.application.name=eureka-server
server.port=8761
# Since it's a server, we don't want it to register and
# also don't want to fetch the instances details
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
#by default Server do not remove the client, even heart beat is missed.
#so we need to turn this OFF. And allow the server to remove if client heart beat missed.
eureka.server.enable-self-preservation=false
#how frequently eviction task runs
eureka.server.eviction-interval-timer-in-ms=6000

Allowing server to remove the instance, if heart beat is not received.

How often, server check for dead instances

System Status			
Environment	test	Current time	2025-06-13T12:32:29 +0530
Data center	default	Uptime	00:04
		Lease expiration enabled	true
		Renews threshold	3
		Renews (last min)	0
THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.			
DS Replicas			
localhost			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.37:product-service:8082

Because in above config:

- Client will send the heart beat after - 60seconds

• Server will wait for the heart beat only for - 5 seconds

So server, removed the client instance from the list, even Client application is running and up.

System Status			
Environment	test	Current time	2025-06-13T12:32:39 +0530
Data center	default	Uptime	00:04
		Lease expiration enabled	true
		Renews threshold	1
		Renews (last min)	0
THE SELF PRESERVATION MODE IS TURNED OFF. THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.			
DS Replicas			
localhost			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
No instances available			

## 2. How does Eureka Server stores the data

Eureka Server only stores the data in-memory : `Map<String, Lease<InstanceInfo>>`

- Key : `appName/instanceId`  
ex:  
PRODUCT-SERVICE/192.157.2.27:product-service:8082
- value: *InstanceInfo object*
  - Instance ID
  - App name
  - IP address
  - Host name
  - Port
  - Status (UP, DOWN)
  - Last renewed timestamp
  - Lease duration
  - Etc.

## 3. What if Eureka Server itself goes down? Is it a single point of failure?

- Yes, a Single Eureka Server is a Single point of failure, if it goes down.
- Usually, 3 nodes cluster is used.

Say, we have 3 Eureka Server on different machine or container:

- eureka-1 at port 8761
- eureka-2 at port 8762
- eureka-3 at port 8763

Now, each Server is a client too. As they have to register themselves, fetch the registry, replicate changes.

### application.properties for eureka server 1:

```
spring.application.name=eureka-server
server.port=8761
eureka.instance.hostname=localhost
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
eureka.client.service-url.defaultZone= http://localhost:8762/eureka/, http://localhost:8763/eureka/
```

### application.properties for eureka server 2:

```
spring.application.name=eureka-server
server.port=8762
eureka.instance.hostname=localhost
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
eureka.client.service-url.defaultZone= http://localhost:8761/eureka/, http://localhost:8763/eureka/
```

### application.properties for eureka server 3:

```
spring.application.name=eureka-server
server.port=8763
eureka.instance.hostname=localhost
eureka.client.register-with-eureka=true
eureka.client.fetch-registry=true
eureka.client.service-url.defaultZone= http://localhost:8761/eureka/, http://localhost:8762/eureka/
```

### And in Client application.properties:

```
eureka.client.service-url.defaultZone= http://eureka-1:8761/eureka/, http://eureka-2:8762/eureka/, http://eureka-3:8763/eureka/
```

Now even if 1 server is down, it will not impact the client availability.

## 4. It can cause latency issue, as each call now required 2 hops, first it has to invoke Eureka Server and then the actual call.

- Eureka Server does not get called for every request.
- At startup, Client say (order-service) fetch the registry  
`eureka.client.fetch-registry=true`
- And Cache it locally, all future request, uses this local copy to find the instance.

we can control this with config like below, after every 30 seconds client will refresh its cache copy:

```
eureka.client.registry-fetch-interval-seconds=30
```

5. What if the local cache is stale? Can this lead to calling a dead instance?

It's a valid scenario and you can say it's a trade off.

*This call will ultimately fail, till cache is not refreshed with latest data.*

