**SERVICE DISCOVERY**

## What is Service Discovery in Microservices?

**A**[**microservices-based application**](https://www.baeldung.com/spring-microservices-guide)**typically runs in virtualized or containerized environments. The number of instances of a service and its locations changes dynamically.**We need to know where these instances are and their names to allow requests to arrive at the target microservice. This is where tactics such as Service Discovery come into play.

**The Service Discovery mechanism helps us know where each instance is located. In this way, a Service Discovery component acts as a registry in which the addresses of all instances are tracked.** The instances have dynamically assigned network paths. Consequently, if a client wants to make a request to a service, it must use a Service Discovery mechanism.

[Microservices](https://www-stage.avinetworks.com/what-are-microservices-and-containers/) service discovery is a way for applications and microservices to locate each other on a network. Service discovery implementations within [microservices architecture](https://www-stage.avinetworks.com/glossary/microservice/) discovery includes both:

• a central server (or servers) that maintain a global view of addresses.

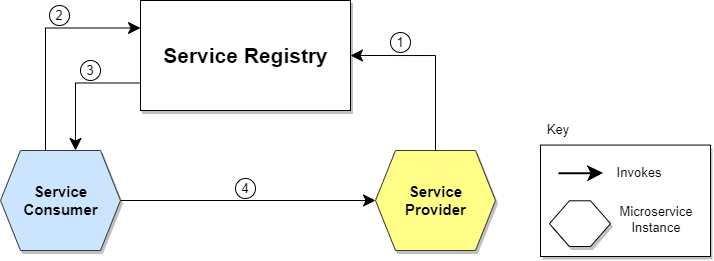
• clients that connect to the central server to update and retrieve addresses.

 How Does Service Discovery Works?

**Service Discovery handles things in two parts. First, it provides a mechanism for an instance to register and say, “I’m here!” Second, it provides a way to find the service once it has registered.**

Let’s clarify the concept we’ve discussed so far with an example: a Service Consumer and a Service Provider (a service exposing [REST API](https://www.baeldung.com/cs/rest-architecture)). The Service Consumer needs the Service Provider to read and write data.

The following diagram shows the communication flow:



Let’s describe the steps illustrated in the diagram:

1. The location of the Service Provider is sent to the Service Registry (a database containing the locations of all available service instances).
2. The Service Consumer asks the Service Discovery Server for the location of the Service Provider.
3. The location of the Service Provider is searched by the Service Registry in its internal database and returned to the Service Consumer.
4. The Service Consumer can now make direct requests to the Service Provider.

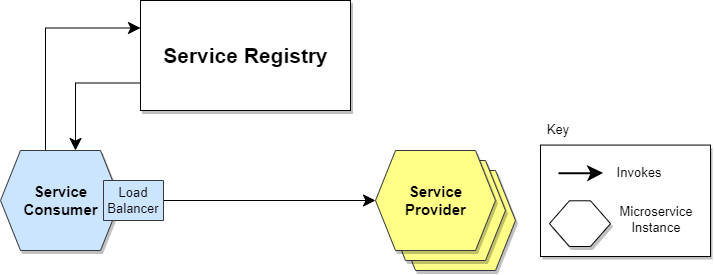
Service Discovery patterns

There are two main Service Discovery patterns: Client‑Side Discovery and Server‑Side Discovery. Let’s clarify them.

* **Client-Side Service Discovery**

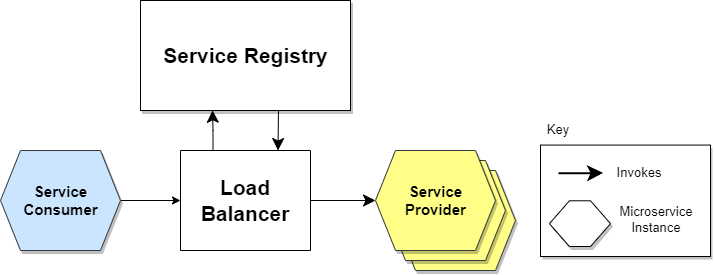
When using Client-Side Discovery, **the Service Consumer is responsible for determining the network locations of available service instances and load balancing requests between them.** The client queries the Service Register. Then the client uses a load-balancing algorithm to choose one of the available service instances and performs a request.

The following diagram shows the pattern just described:

[](https://www.baeldung.com/wp-content/uploads/sites/4/2022/01/Service-Discovery-Client-Side.png)

* **Server side discovery:The alternate approach to Service Discovery is the Server-Side Discovery model, which uses an intermediary that acts as a**[**Load Balancer**](https://www.baeldung.com/zuul-load-balancing)**.** The client makes a request to a service via a load balancer that acts as an orchestrator. The load balancer queries the Service Registry and routes each request to an available service instance.

The following diagram shows how communication takes place:

[](https://www.baeldung.com/wp-content/uploads/sites/4/2022/01/Service-Discovery-Server-Side.png)

## What are the Advantages of Service Discovery (Server-side & Client-side)?

The advantage of Server-side service discovery is that it makes the client application lighter as it does not have to deal with the lookup procedure and makes a request for services to the router.

The advantage of Client-side service discovery is that the client application does not have to traffic through a router or a [load balancer](https://www-stage.avinetworks.com/glossary/load-balancer/) and therefore can avoid that extra hop.

What Is Service Registry?

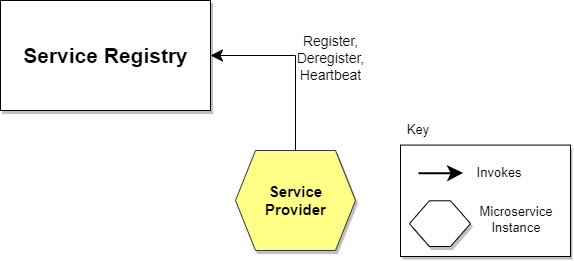
So far, we’ve assumed that the Service Registry already knew the locations of each microservice. But how do this registration and de-registration operation take place?

The Service Register is a crucial part of service identification. **It’s a database containing the network locations of service instances.** A Service Registry must be highly available and up-to-date. Clients can cache the network paths obtained from the Service Registry;

Service Registration Options

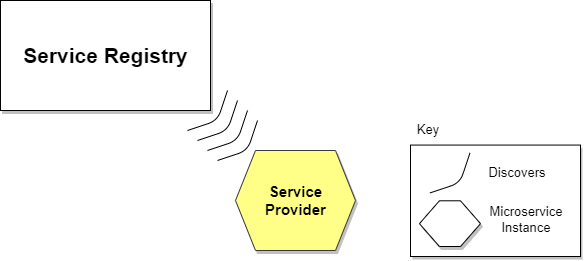
Self-Registration

**When using the self-registration model, a service instance is responsible for registering and de-registering itself in the Service Registry.** In addition, if necessary, a service instance sends [heartbeat](https://martinfowler.com/articles/patterns-of-distributed-systems/heartbeat.html) requests to keep its registration alive. The following diagram shows the structure of this pattern:

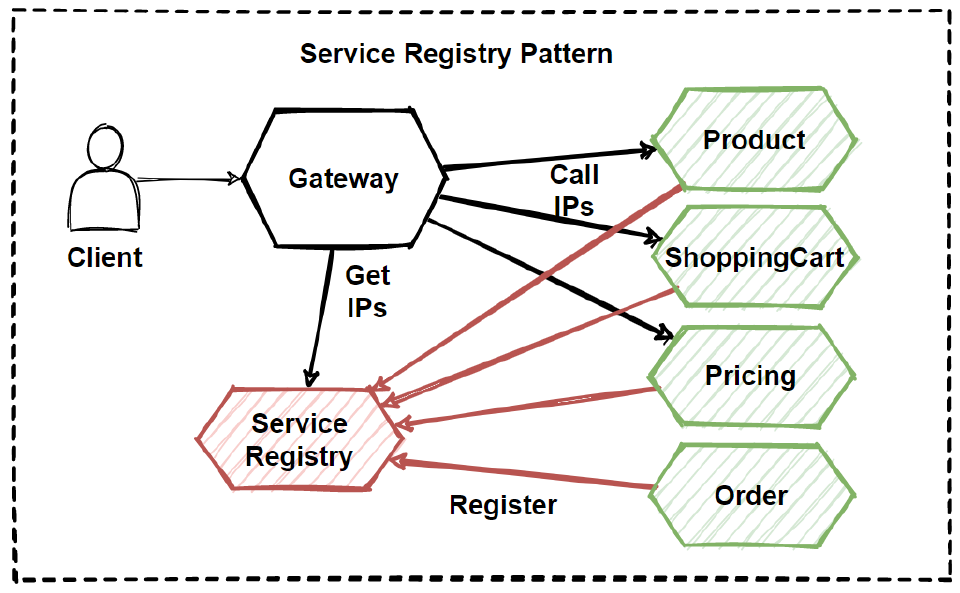
[](https://www.baeldung.com/wp-content/uploads/sites/4/2022/01/Service-Discovery-Self-Registration.png)

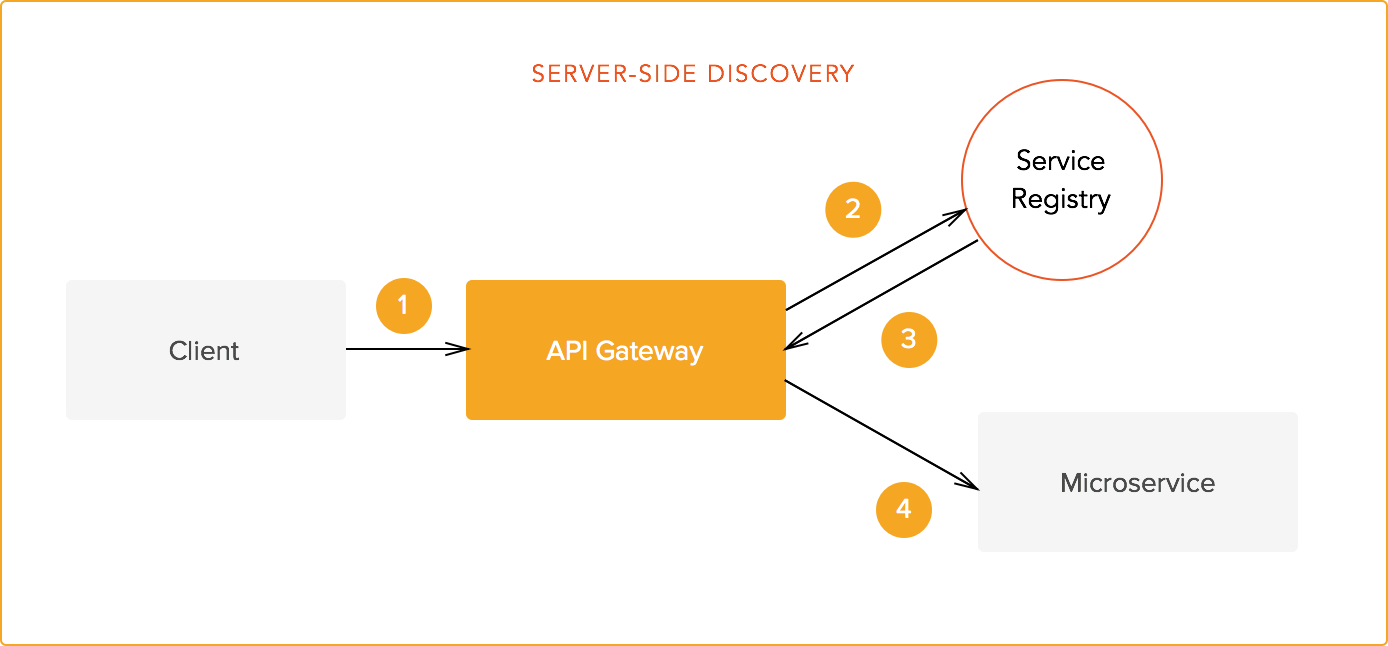
### Third-party Registration

When using the third-party registration model, the service instances aren’t responsible for registration in the Service Registry. Instead, another system component known as the Service Register is responsible for registration. **The Service Register keeps track of changes to running instances by polling the deployment environment or subscribing to events.** When it detects a newly available service instance, it records it in its database. The Service Registry also de-registers terminated service instances. The following diagram illustrates this:

[](https://www.baeldung.com/wp-content/uploads/sites/4/2022/01/Service-Discovery-3rd-Registration.png)

**Diagram elaborating the service discovery server and client using an api gateway**

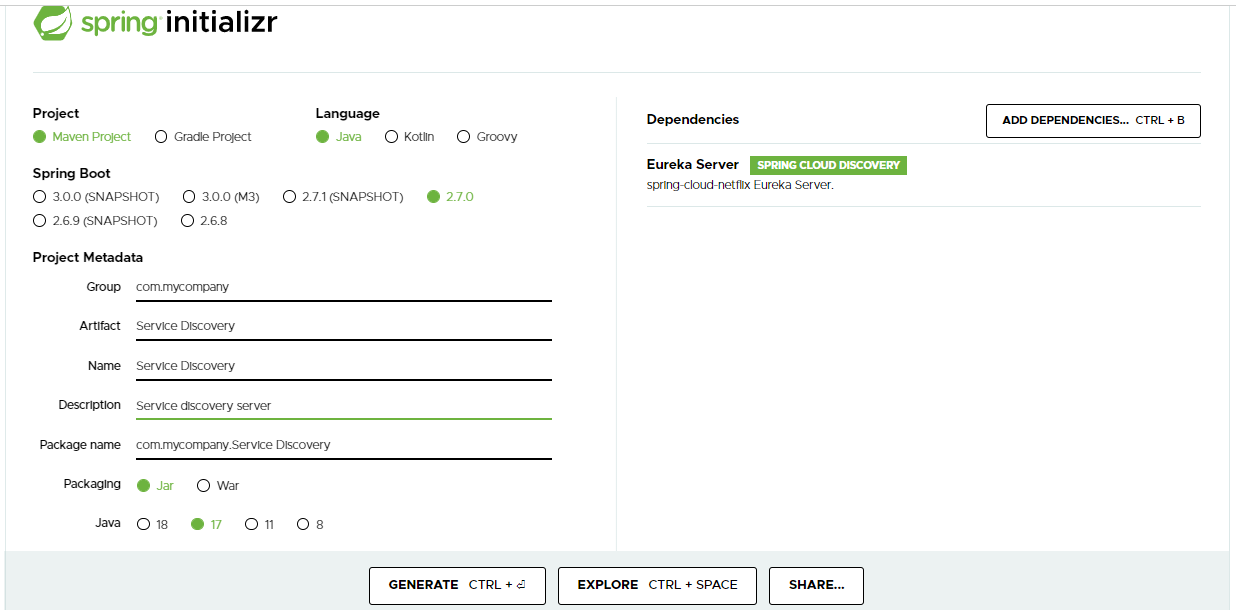




**How to Implement Service Discovery Design pattern?**

**A.SETTING UP THE SERVER**

1.Go to Spring initializer and make a spring boot project and add dependency as below:



Generate this project and import it in your eclipse or intellij.

2.Naw go in application.properties

And paste the following lines

|  |
| --- |
| server.port=8761  eureka.instance.hostname=localhost  eureka.client.register-with-eureka=false  eureka.client.fetch-registry=false |

\*\*The default port for server is 8761

3.Now in your main Application add the following dependency

@EnableEurekaServer



4.Now in your pom.xml add the following

<**spring-cloud.version**>2021.0.3</**spring-cloud.version**>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<**dependencies**>

<**dependency**>

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

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

</**dependency**>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<**dependencyManagement**>

<**dependencies**>

<**dependency**>

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

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

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

<**type**>pom</**type**>

<**scope**>import</**scope**>

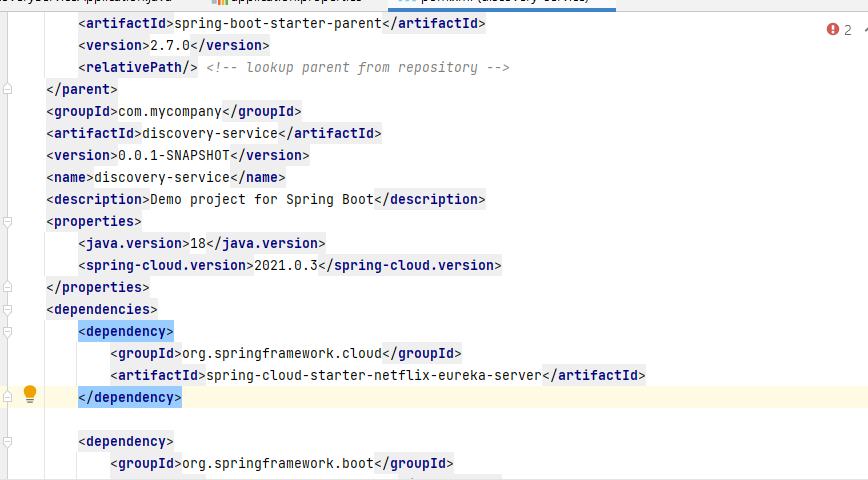
</**dependency**>

</**dependencies**>

</**dependencyManagement**>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

As shown in image below:



5.Now you are ready to go.Run your project.

6.Now go to your browser and go to following url

Localhost:8761

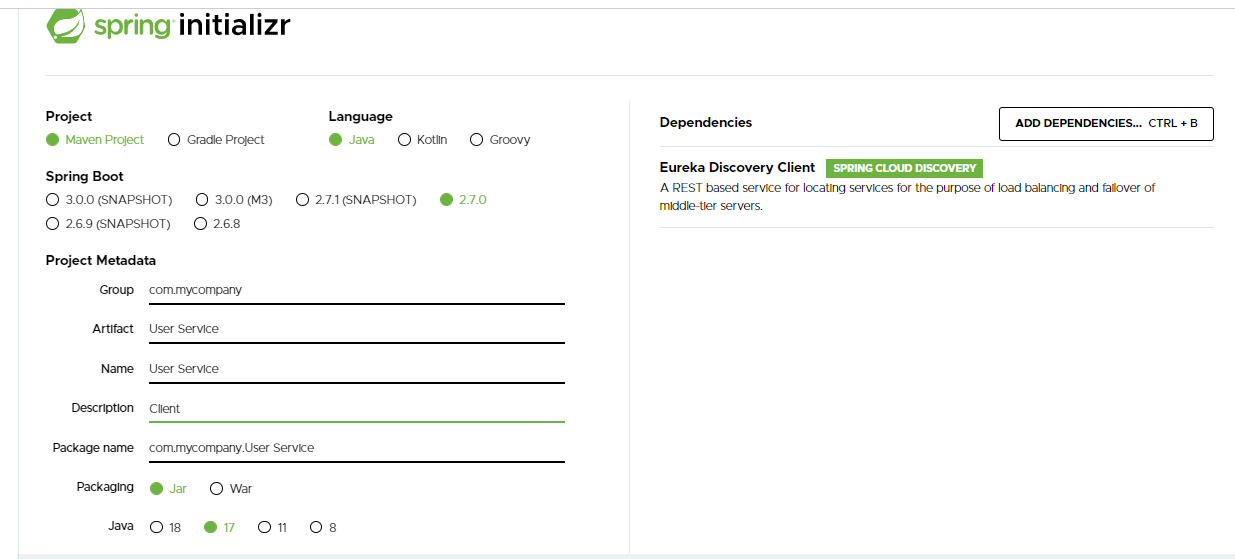
Now your server is ready.The other client services can register against it.

**B.SETTING UP THE CLIENTS**

1.Make your spring boot applications using spring initializer.

Suppose you make userService,orderService…

In the spring initializer make sure to add Eureka client dependency



2.Generate it and import in your eclipse

3.Now in pom.xml ensure to add the following lines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<**dependency**>

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

<**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>

</**dependency**>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<**spring-cloud.version**>2021.0.3</**spring-cloud.version**>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<**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**>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

As shown below:





4.Now go to application.properties and add the following line of code:

**spring.profiles.active**=**local**

**server.port**=**7000**

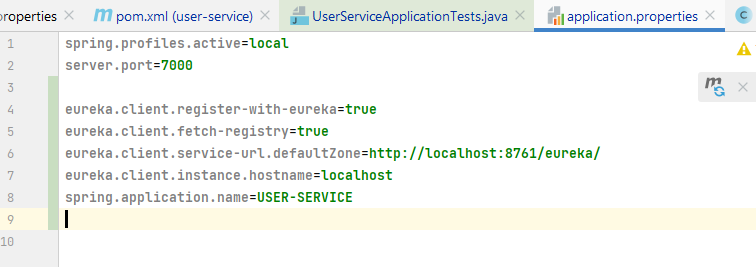
**eureka.client.register-with-eureka**=**true**

**eureka.client.fetch-registry**=**true**

**eureka.client.service-url.defaultZone**=**http://localhost:8761/eureka/**

**eureka.client.instance.hostname**=**localhost**

**spring.application.name**=**USER-SERVICE**



5.Now your client is ready and you can run it.In logs you will see that it says it is registering itself against a service discovery.

