What is Service Discovery in Microservices?

Service Discovery is the mechanism that allows microservices to locate and communicate with each other dynamically without hardcoding IP addresses or ports.[1] [2]

In a microservices environment (like Kubernetes or AWS), services are constantly scaling up (new instances) or down (instances dying). IP addresses change frequently.

- **Without Service Discovery:** Service A calls Service B at `http://192.168.1.50`. If Service B restarts on a new IP, Service A breaks.
- **With Service Discovery:** Service A asks a "Registry" (phonebook) "Where is Service B?", and gets the current, healthy list of IPs.[3] [1]

14. Types of Service Discovery (Client-side vs Server-side)

The key difference is **who** is responsible for looking up the address: the caller (Client) or a middleman (Router/Load Balancer).[4] [2]

A. Client-Side Discovery

- **How it works:**
  1. The **Client** (e.g., Service A) queries the **Service Registry** (e.g., Eureka) to get the list of available IPs for Service B.
  2. The Client selects one IP (using a load balancing algorithm like Round Robin) and makes the call directly.[4]
- **Pros:** Fewer network hops (Client → Provider); client can make smart load balancing decisions.
- **Cons:** Client must implement discovery logic (tight coupling to the registry). If you use multiple languages (Java, Go, Python), you need discovery libraries for each.[2] [4]
- **Example Tools:** Netflix Eureka, HashiCorp Consul (when used with client libraries).

B. Server-Side Discovery

- **How it works:**
  1. The **Client** makes a request to a generic Load Balancer or Router (e.g., `http://service-b/api`).
  2. The **Router** queries the Service Registry, picks a healthy instance, and forwards the request.[2] [4]

- **Pros:** Client is simple (doesn't know about discovery logic); works easily with any programming language.
- **Cons:** One extra network hop (Client → Router → Provider); the Router must be highly available.[4] [2]
- **Example Tools:** Kubernetes Service (K8s handles this natively), AWS Elastic Load Balancer (ELB), NGINX.

Summary Table

Feature	Client-Side Discovery	Server-Side Discovery
Who looks up IP?	The Client Service	The Router / Load Balancer
Coupling	High (Client knows registry)	Low (Client knows only Router)
Complexity	Client needs smart library	Infrastructure needs smart router
Network Hops	1 (Direct)	2 (Via Router)
Common Example	Spring Boot + **Eureka**	**Kubernetes** Services

⁂

1. https://www.baeldung.com/cs/service-discovery-microservices
2. https://edgedelta.com/company/blog/what-is-service-discovery
3. https://api7.ai/blog/service-discovery
4. https://www.solo.io/topics/microservices/microservices-service-discovery
5. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg
6. https://dev.to/apisix/service-discovery-key-to-unleashing-microservices-architecture-356c
7. https://middleware.io/blog/service-discovery/
8. https://www.f5.com/company/blog/nginx/service-discovery-in-a-microservices-architecture
9. https://konghq.com/blog/learning-center/service-discovery-in-a-microservices-architecture
10. https://www.f5.com/es_es/company/blog/nginx/service-discovery-in-a-microservices-architecture
11. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg
12. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg
13. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg
14. https://www.geeksforgeeks.org/java/service-discovery-and-service-registry-in-microservices/