CHAPTER 9

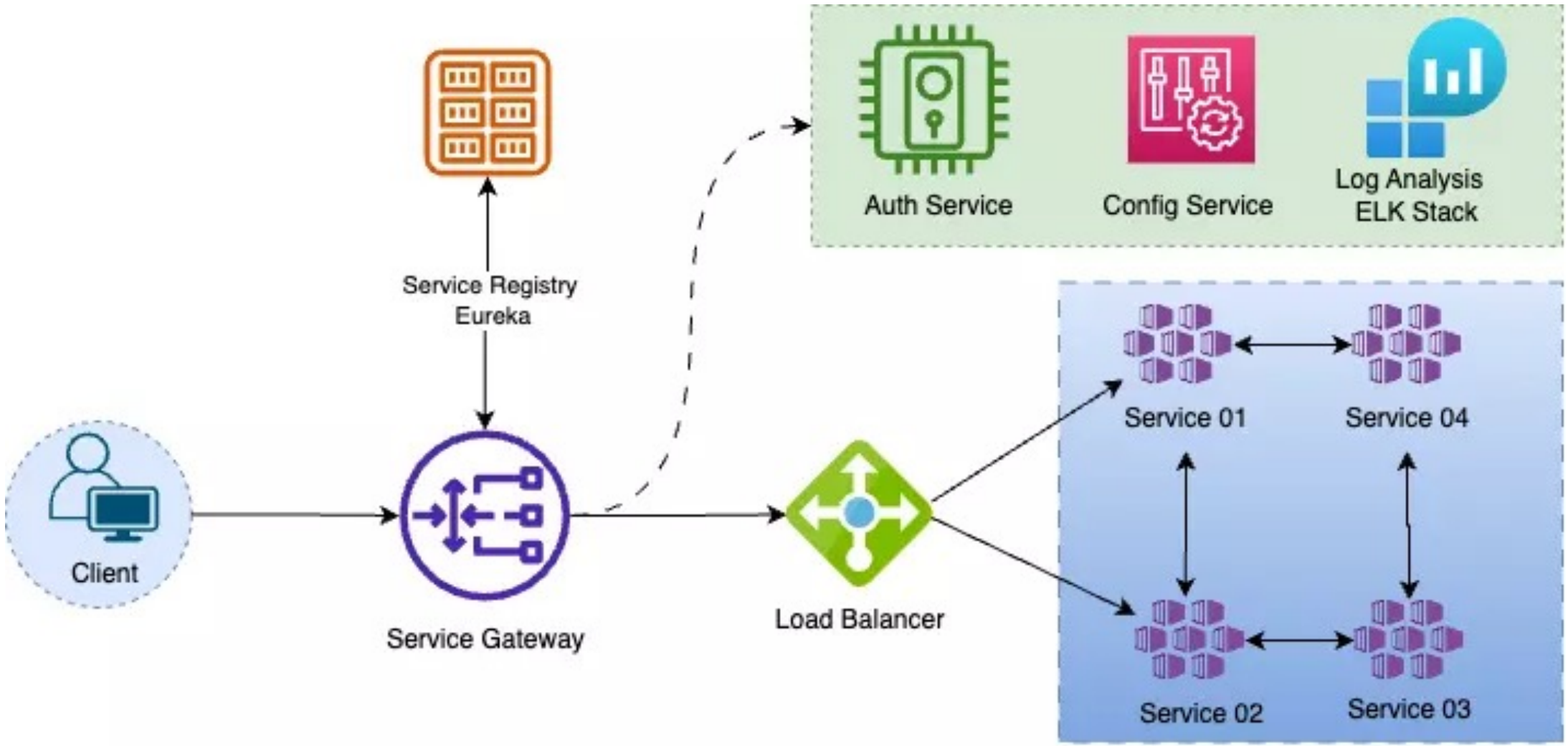
MICROSERVICES (cont.)

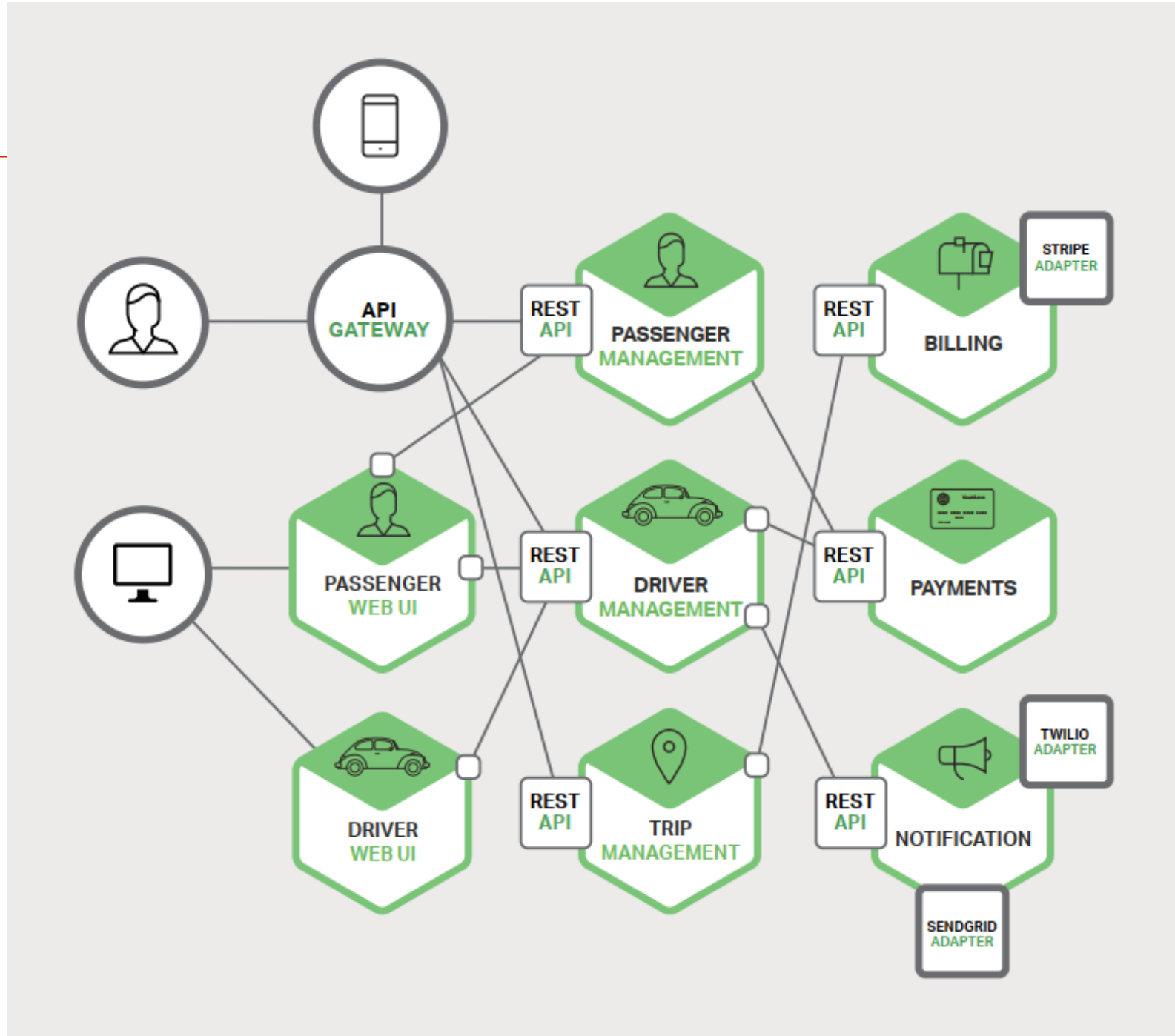
Chapter 9: MICROSERVICES (cont.)

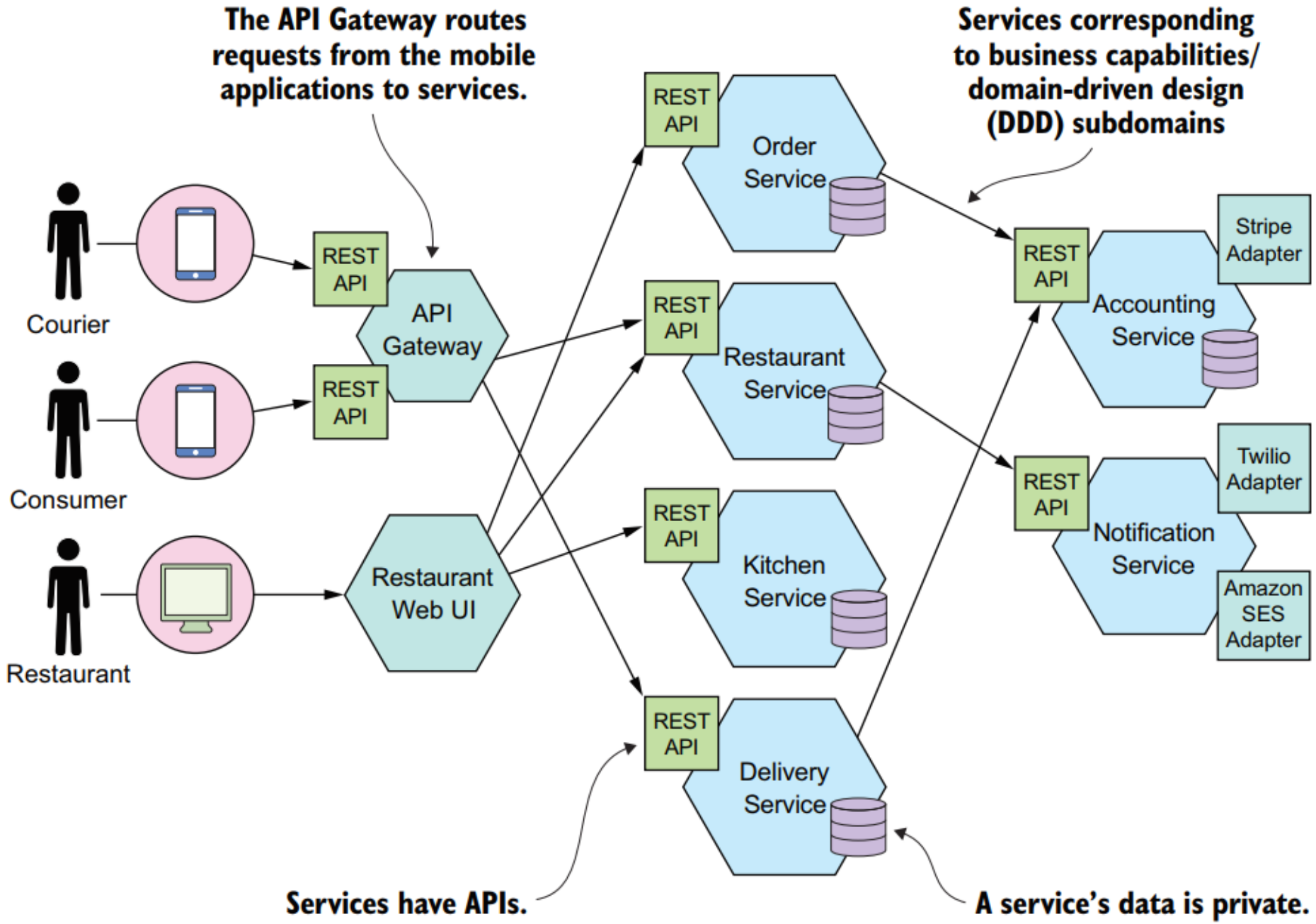
OUTLINE

1. Service Discovery
2. Service Registry
3. Role API gateway

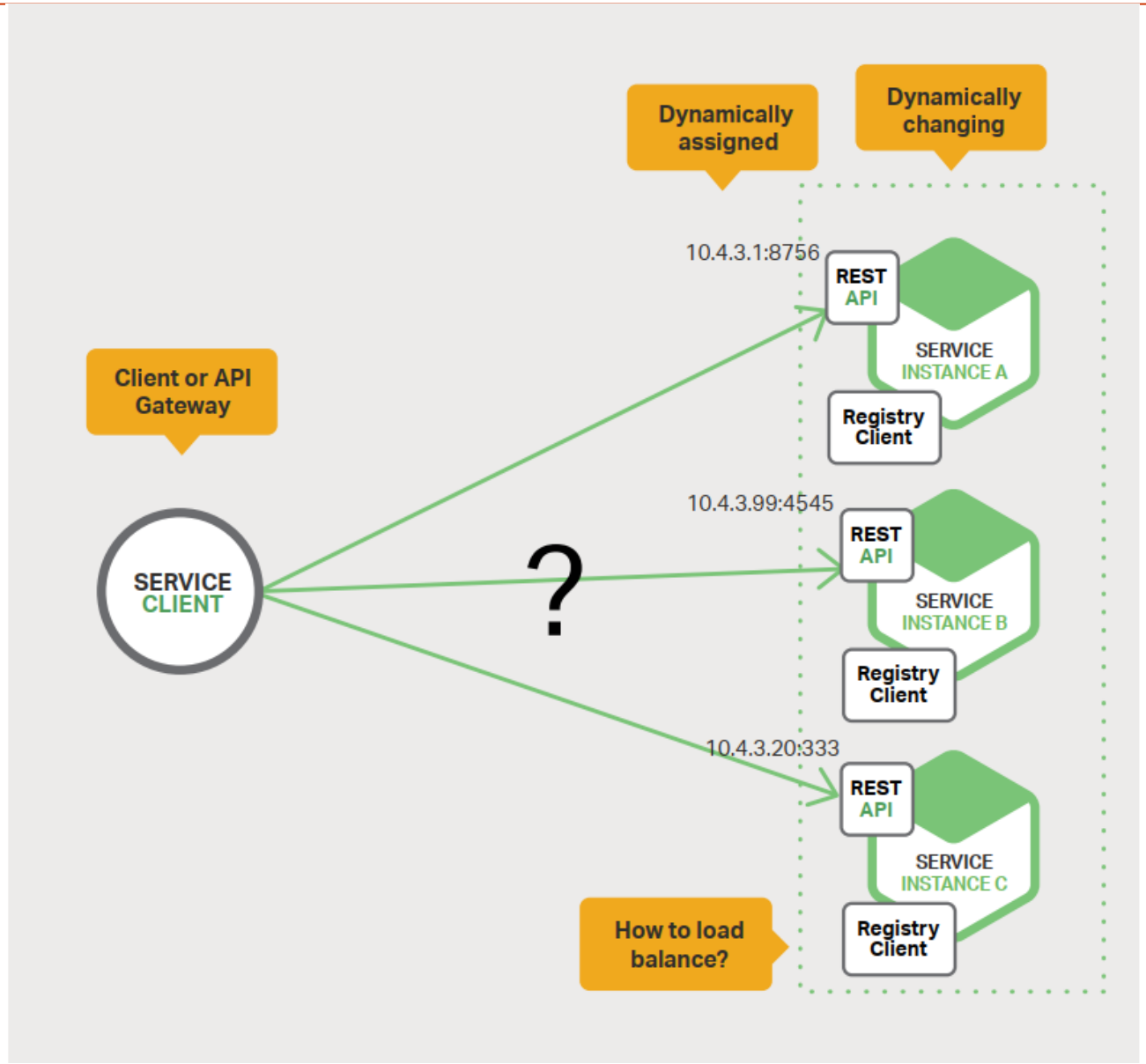
Chapter 9: MICROSERVICES (cont.)




API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
STRIPE
ADAPTER
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
TWILIO
ADAPTER
SENDGRID
ADAPTER


The API Gateway routes requests from the mobile applications to services.
Services corresponding to business capabilities/ domain-driven design (DDD) subdomains
Courier
Consumer
Restaurant
REST API
API Gateway
Restaurant Web UI
Order Service
Restaurant Service
Kitchen Service
Delivery Service
Accounting Service
Notification Service
Stripe Adapter
Twilio Adapter
Amazon SES Adapter
Services have APIs.
A service's data is private.

Service Discovery



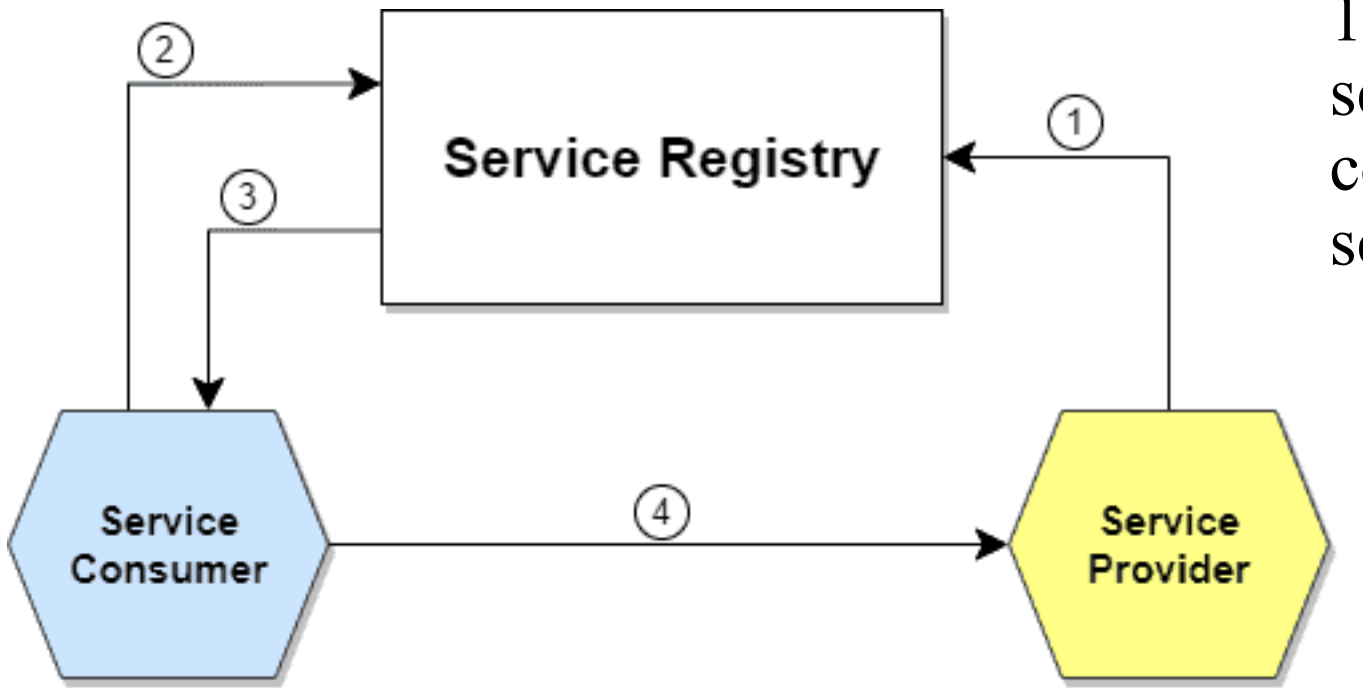
Service Discovery (cont.)

- Let’s imagine several microservices that make up a more or less complex application. These will communicate with each other somehow (API Rest, gRPC).
- A microservices-based application 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.

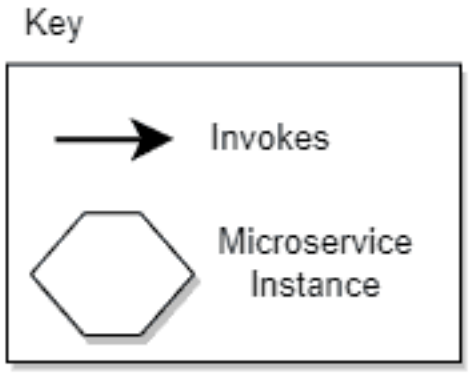
Service Discovery (cont.)

- Service discovery in microservices is the process of dynamically locating and identifying available services within a distributed system. In a microservices architecture, where applications are composed of many small, independent services, service discovery plays a crucial role in enabling communication and collaboration between these services.
  - In a microservices environment, services are often deployed across multiple instances or containers, making it challenging for clients to locate and communicate with the appropriate service instances.
  - Service discovery solves this problem by providing mechanisms for service registration, discovery, and resolution.

How does Service Discovery works?



1. The location of the Service Provider is sent to the Service Registry (a database containing the locations of all available service instances).



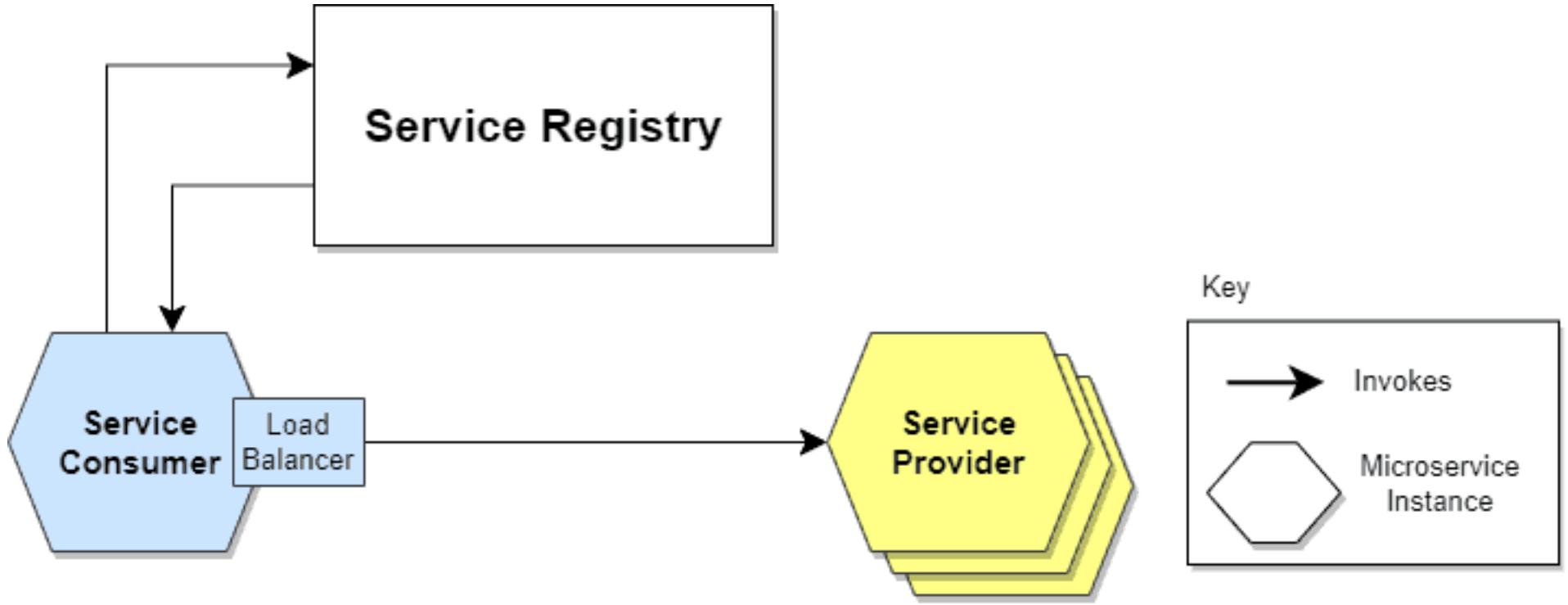
2. The Service Consumer asks the Service Discovery Server for the location of the Service Provider.
3. The location of the Service Provider is searched by the Service Registry in its internal database and returned to the Service Consumer.
4. The Service Consumer can now make direct requests to the Service Provider.

Service Discovery (cont.)

- There are two main Service Discovery patterns:
  - Client-Side Discovery
  - Server-Side Discovery

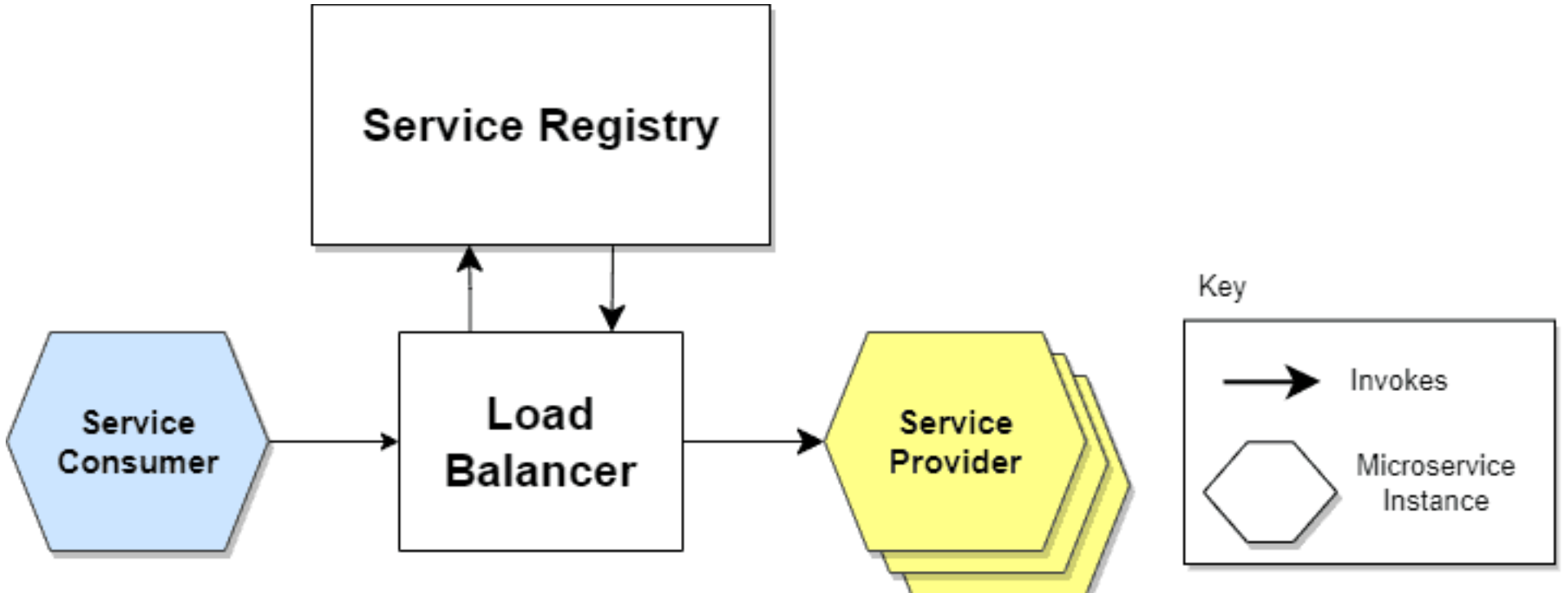
Client-Side Discovery

- **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.



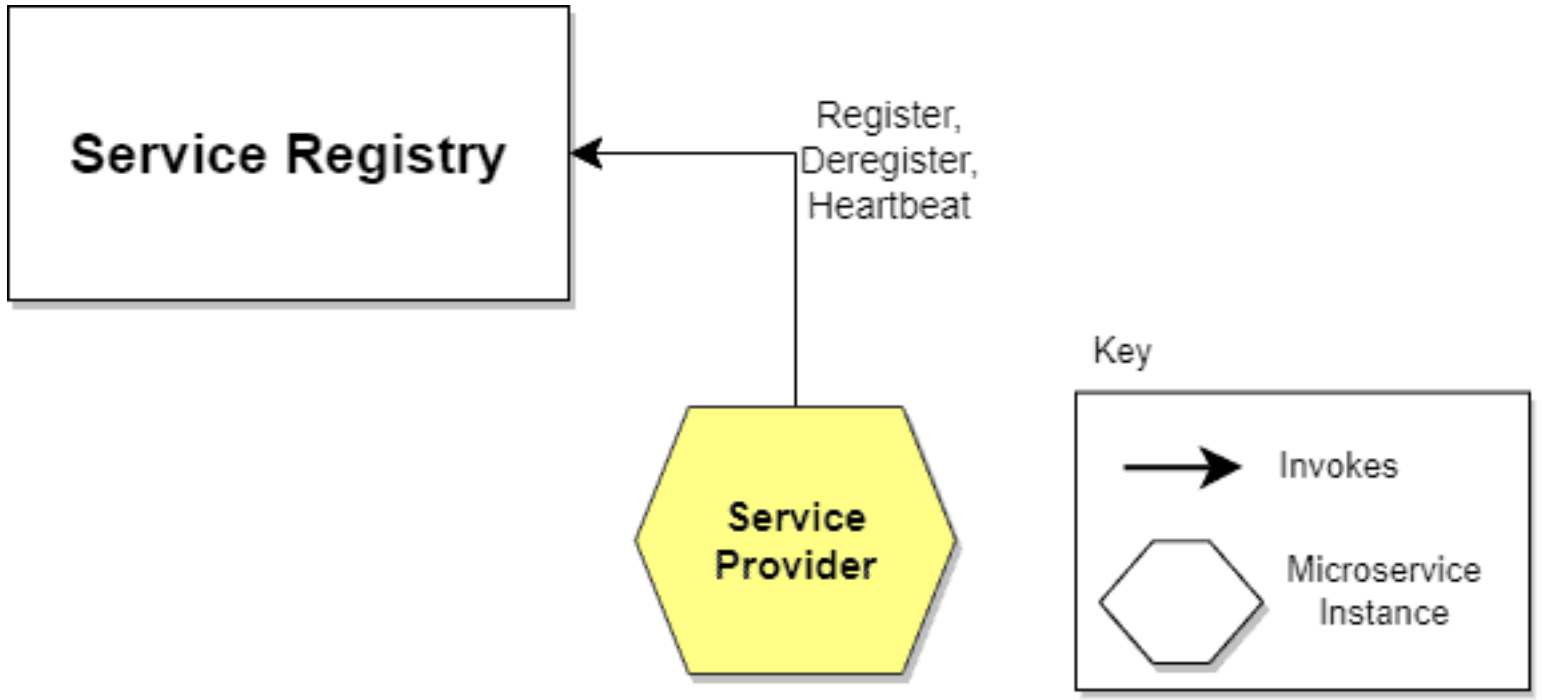
Server-Side Discovery

- **The alternate approach to Service Discovery is the Server-Side Discovery model, which uses an intermediary that acts as a Load Balancer.** 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.



Service Registry

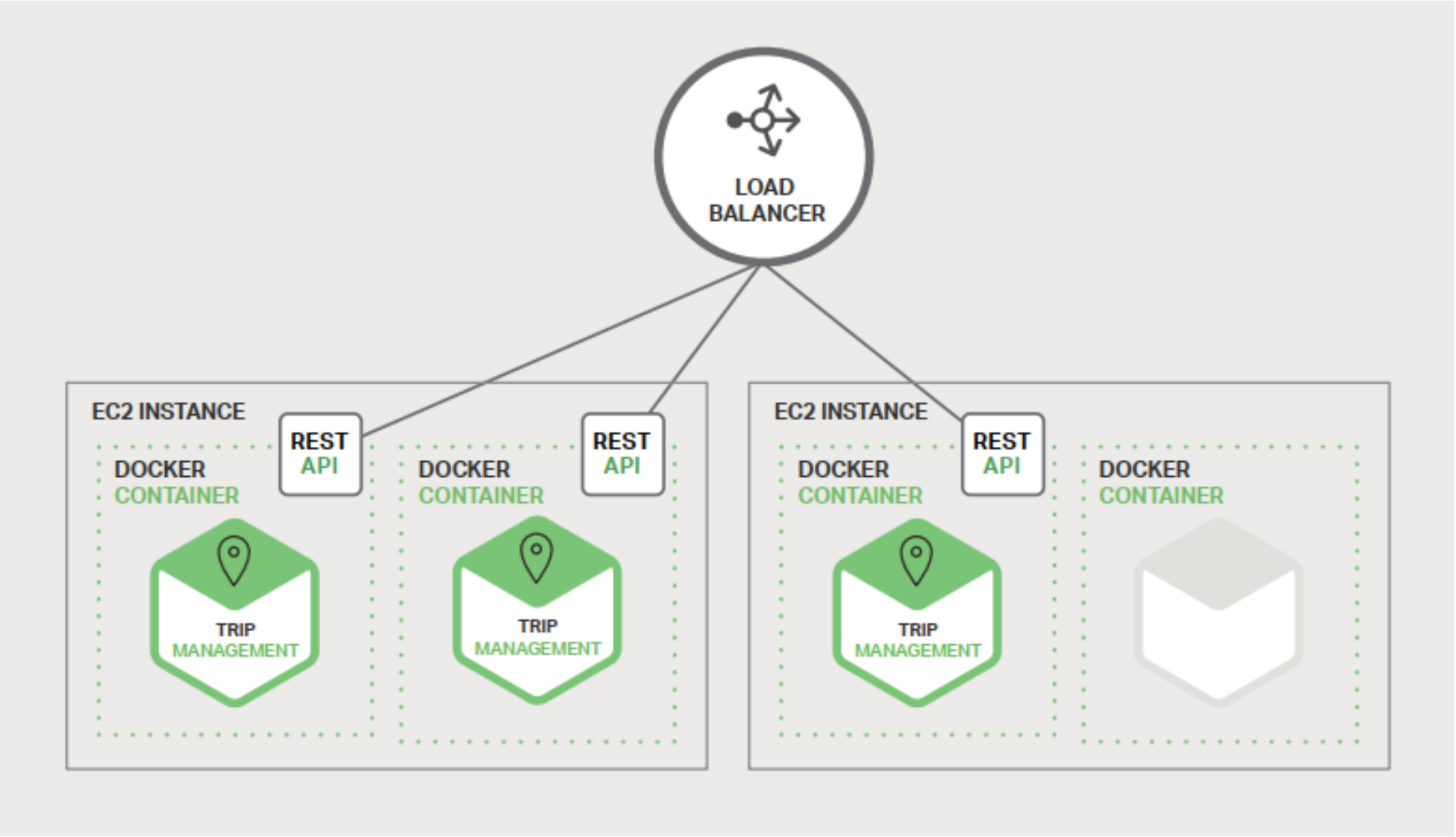
- When a microservice instance starts up or becomes available, it registers itself with the Service Registry. This registration process includes providing metadata such as the service name, network location (IP address and port), health status, and possibly other attributes.



Load balancing

- Load balancing is the method of distributing network traffic equally across a pool of resources that support an application.
- Modern applications must process millions of users simultaneously and return the correct text, videos, images, and other data to each user in a fast and reliable manner.
- To handle such high volumes of traffic, most applications have many resource servers with duplicate data between them.
- A load balancer is a device that sits between the user and the server group and acts as an invisible facilitator, ensuring that all resource servers are used equally.

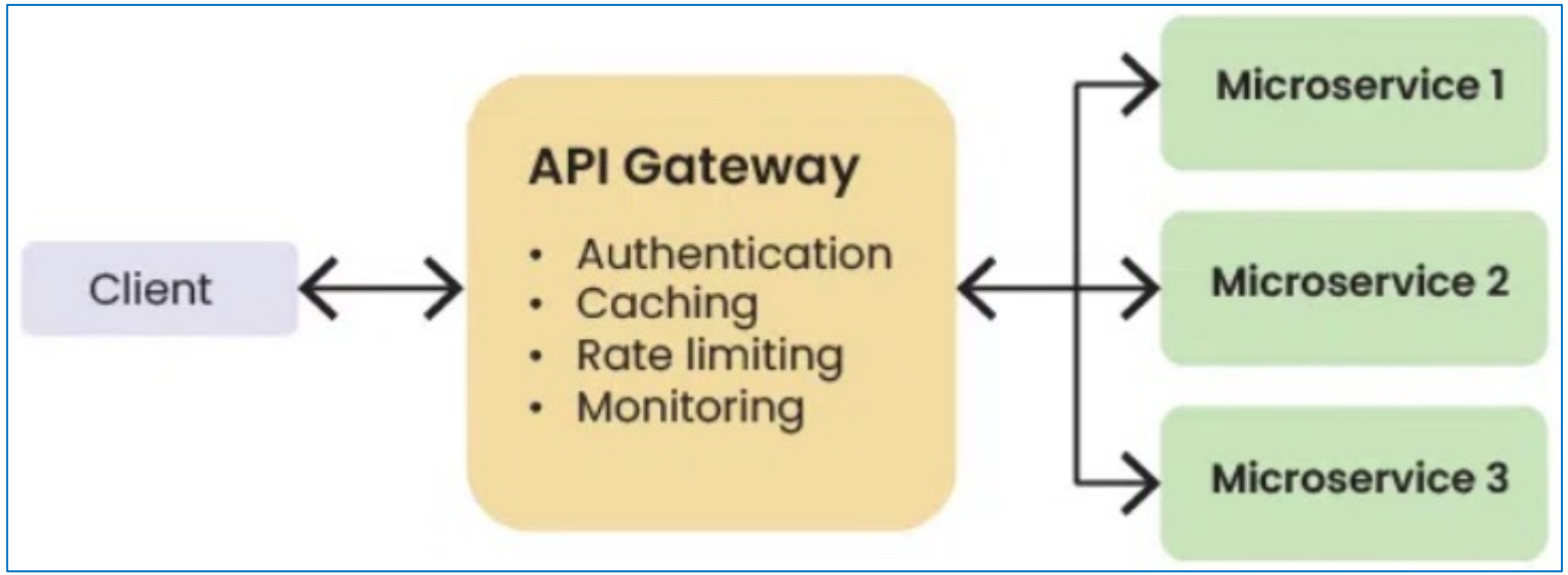
Load balancing (cont.)

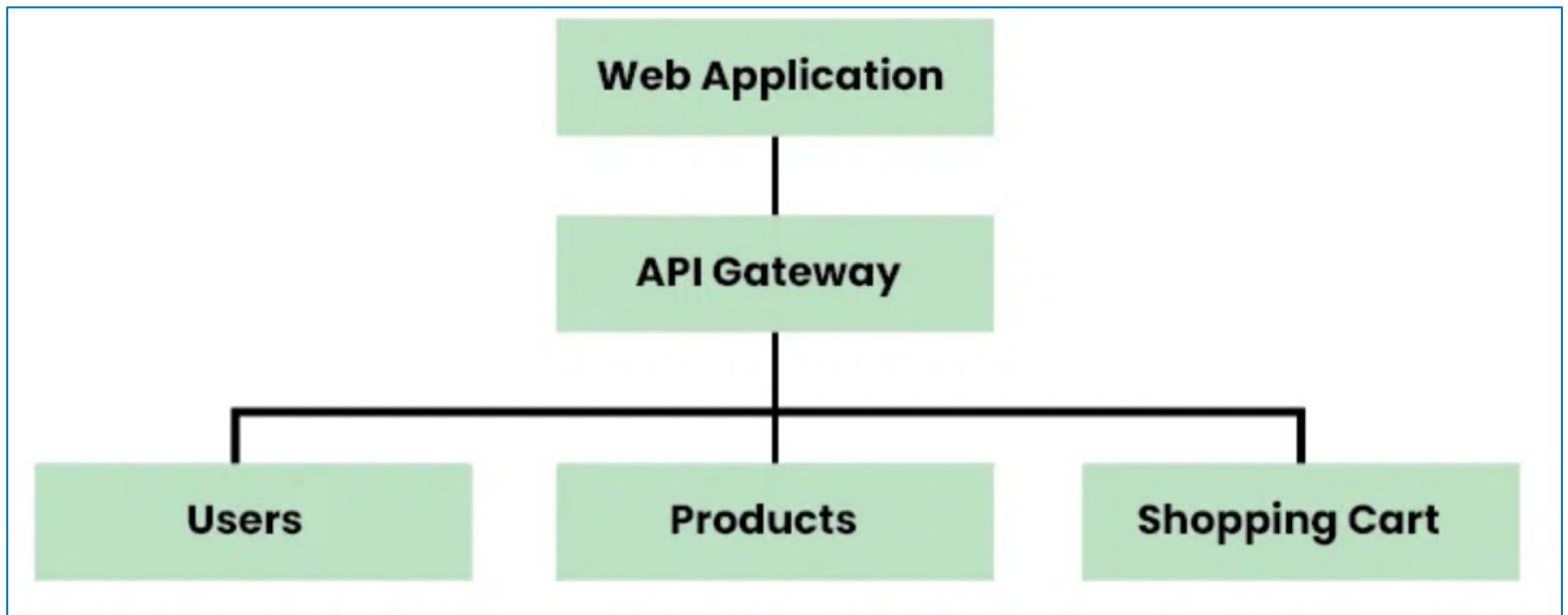


API Gateway

- An API Gateway is a key component in system design, particularly in microservices architectures and modern web applications. It serves as a centralized entry point for managing and routing requests from clients to the appropriate microservices or backend services within a system.
- One service that serves as a reverse proxy between clients and backend services is the API Gateway. After receiving incoming client requests, it manages a number of responsibilities, including rate limitation, routing, and authentication, before forwarding the requests to the appropriate backend services.

API Gateway (cont.)





Spring Cloud - Gateway

- This project provides an API Gateway built on top of the Spring Ecosystem, including: Spring 6, Spring Boot 3 and Project Reactor. Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency.
- Spring Cloud Gateway is built on Spring Boot, Spring WebFlux, and Project Reactor.
- Spring Cloud Gateway requires the Netty runtime provided by Spring Boot and Spring Webflux.
- https://docs.spring.io/spring-cloud/docs/current/reference/htmlsingle/#spring-cloud-gateway

Spring Cloud – Netflix Eureka

- https://docs.spring.io/spring-cloud/docs/current/reference/htmlsingle/#spring-cloud-Netflix

Eureka Flow

Sample

- Create 3 service:
  - identity-service
  - product-servicer
  - api-gateway
- Create eureka-server

Sample (cont.): identity-service

- Dependencies
  - Spring boot
  - Spring Web
  - Spring Data JPA
  - Spring Security
  - **Eureka Discovery Client**
  - …

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

Sample (cont.): identity-service

- **application.properties**

src/.../application.properties

```
spring.application.name=identity-service
server.port=8081
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
```

Sample (cont.): identity-service

```java
@SpringBootApplication
@EnableDiscoveryClient
public class IdentityServiceApplication {

    public static void main(String[] args) {

        SpringApplication.run(IdentityServiceApplication.class, args);
    }
}
```

Sample (cont.): product-service

- Dependencies:
  - Spring boot
  - Spring Web
  - Spring Data JPA
  - Spring Security
  - **Eureka Discovery Client**
  - …

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

Sample (cont.): product-service

- **application.properties**

application.properties

```
spring.application.name=product-service
server.port=8082
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/
```

Sample (cont.): product-service

```
@SpringBootApplication
@EnableDiscoveryClient
public class ProductServiceApplication {


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


}
```

Sample (cont.): eureka-server

- Dependencies
  - **Eureka Server**
  - …

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

Sample (cont.): eureka-server

- application.properties

application.properties

```
spring.application.name=eureka-server


server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetchRegistry=false
eureka.server.enable-self-preservation=false
```

Sample (cont.): eureka-server

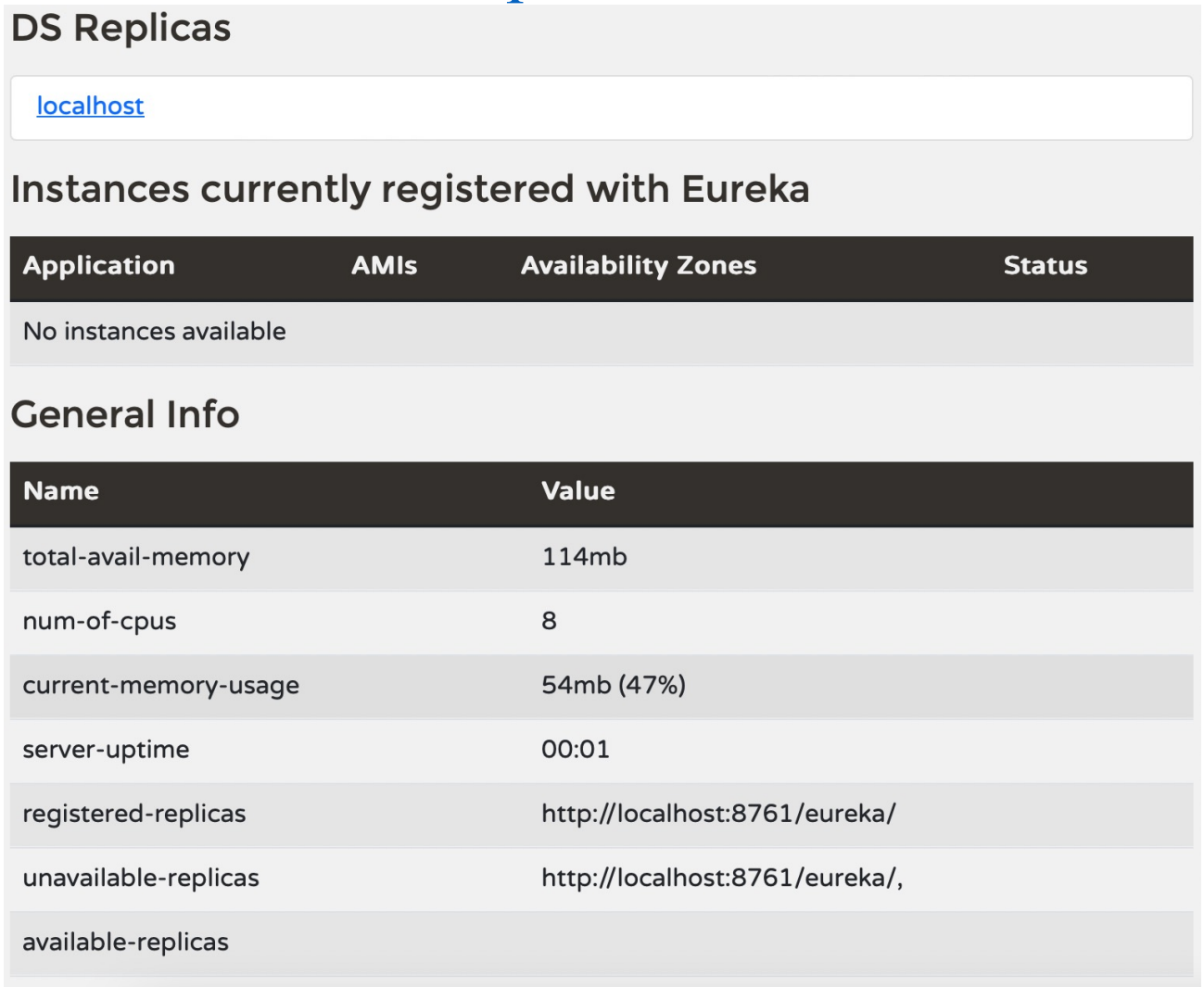
```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {


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


}
```

Sample (cont.): How to run?

- Step 1: run Eureka server http://localhost:8761



DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	114mb
num-of-cpus	8
current-memory-usage	54mb (47%)
server-uptime	00:01
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

Sample (cont.): How to run?

- Step 2: run API Gateway server http://localhost:8080

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.188.193:api-gateway:8080

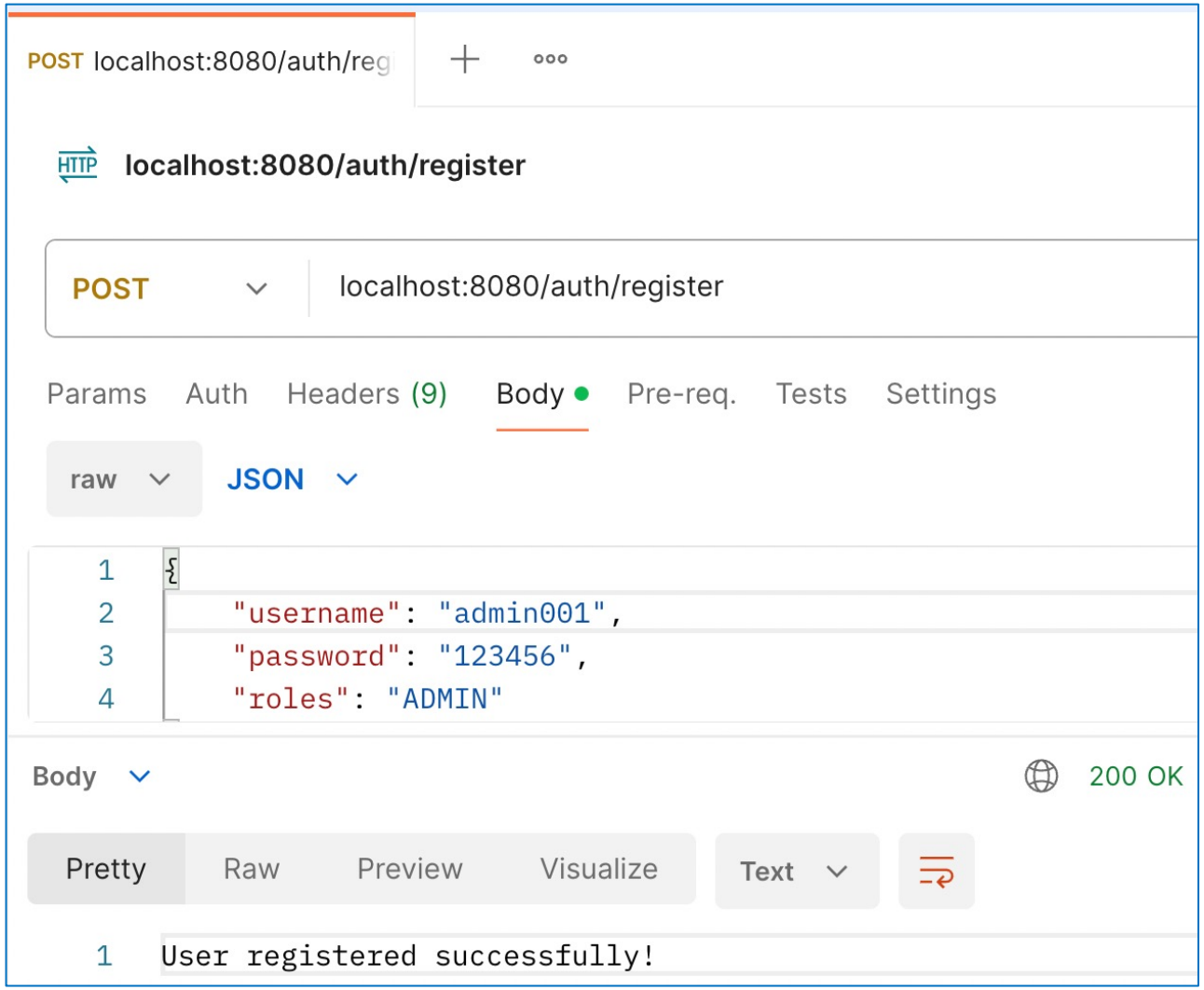
- Step 3: run Identity Service http://localhost:8081

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.188.193:api-gateway:8080
IDENTITY-SERVICE	n/a (1)	(1)	UP (1) - 192.168.188.193:identity-service:8081

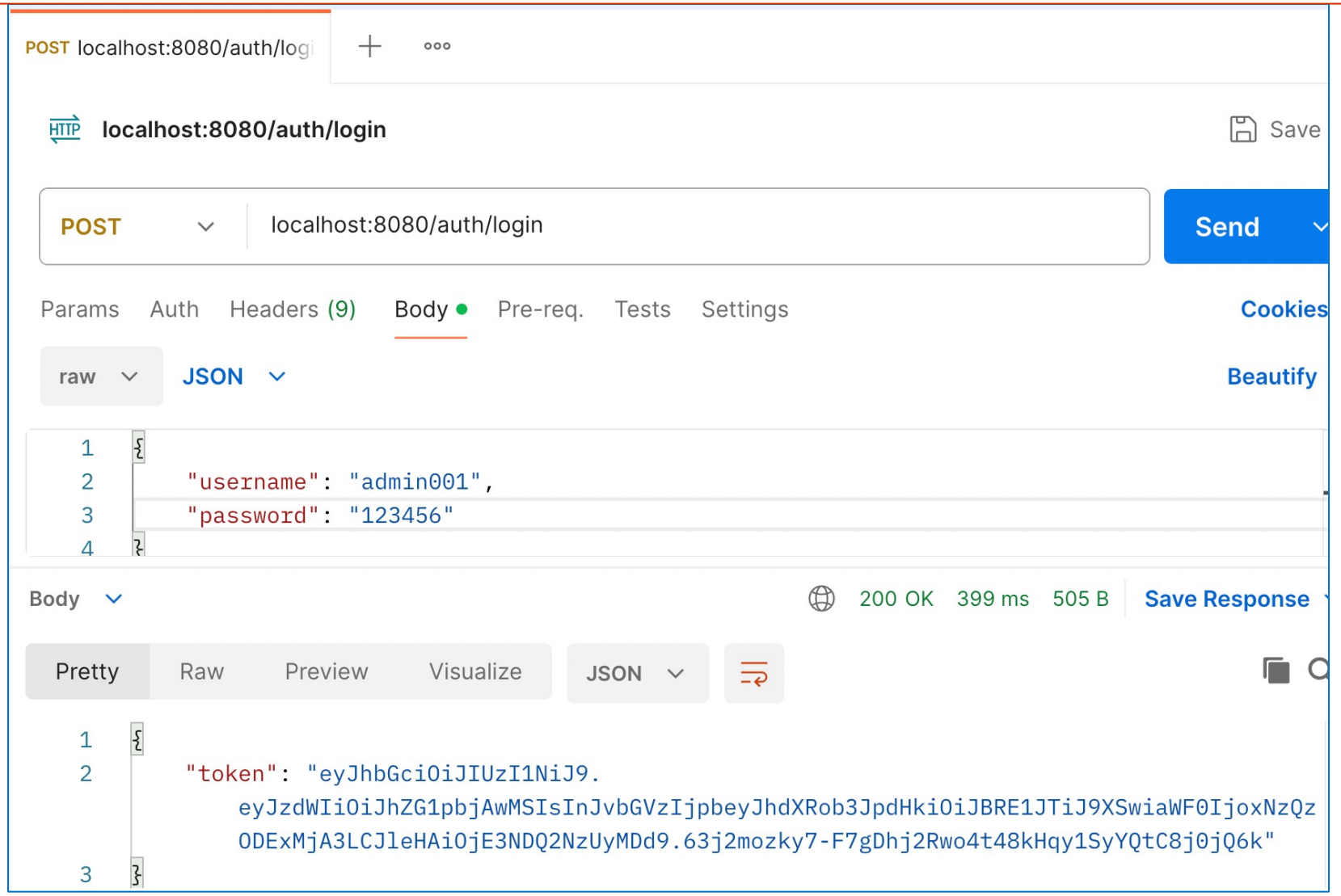
Sample (cont.): How to run?

- Step 4: run API Product Service, Order Service, Customer Service
- Step 5: Test API



Sample (cont.): How to run?

- Step 5: Test API



Ref

1. https://spring.io/guides/gs/service-registration-and-discovery
2. https://docs.spring.io/spring-cloud/docs/current/reference/htmlsingle/#service-discovery-eureka-clients
3. https://www.baeldung.com/cs/service-discovery-microservices

Q&A