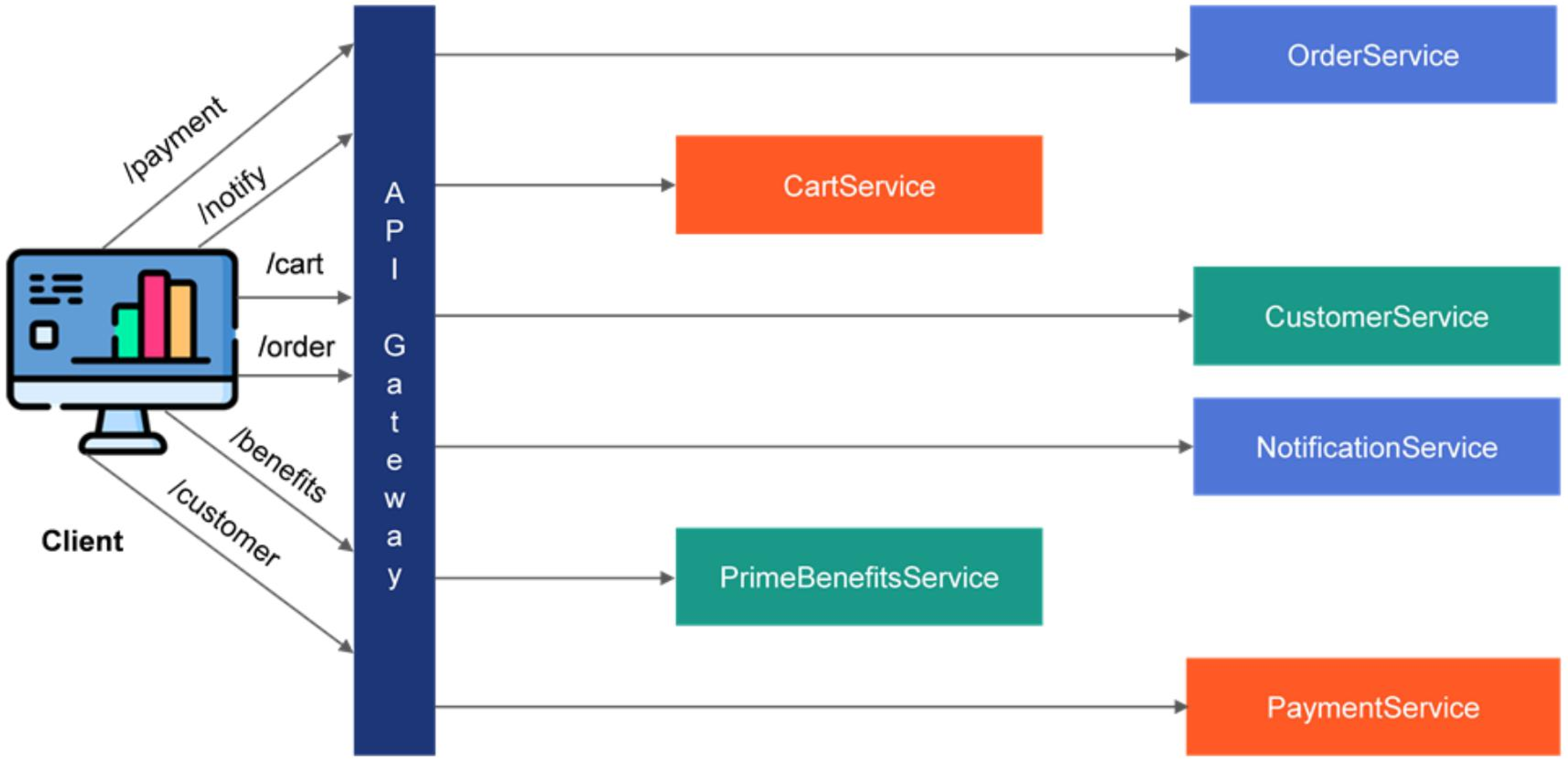
Application Workflow – Multiple Services



Slide Note

While making an online purchase, we need to register in the shopping application, then buy products by adding products to the cart, after which we choose a payment option and are redirected to the payment gateway. After the payment is done the shipping details are captured and order is confirmed.

Think and Tell

- Routing happens through an API Gateway in applications with multiple microservices.
- Will the API Gateway maintain the details of all the services in the application?
- How will the API Gateway know the health of a particular service?
- Will the API Gateway route the request to a service even when the service is down?

Slide Note

Register Microservices on a Netflix Eureka Discovery Server

Menu

00:03:00:00:39 03/ 39

Slide Note

Learning Objectives

- Describe Eureka Server
- Implement the service Discovery Server using Eureka
- Register the services on the Eureka Server
- Describe a load balancer

Slide Note

Eureka Server

Menu

00:05 00:00:39 05/ 39

Slide Note

- Microservices-based architecture is gaining a lot of popularity as it gives many benefits in terms of maintainability, scalability, high availability, etc.
- This task can be hugely complex, depending on number of services and their dynamic nature.
- Think of it as a look-up service where microservices (clients) can register themselves and discover other registered microservices. When a client microservice registers with Eureka it provides metadata such as host, port, and health indicator, thus allowing for other microservices to discover it

Service Registration and Discovery

- **Microservices are highly scalable and available. This can be achieved by executing multiple instances of the same microservice at any given point in time.**
- **While working with multiple instances of microservices there are many challenges.**
- **Maintaining a list of all the addresses of the microservices becomes cumbersome.**
- **In a distributed system, maintaining this service address will be difficult.**
- `Service registration and discovery` **is a design pattern that helps in solving the problem.**
- **In this pattern there is a dedicated server responsible for maintaining the list of addresses of the microservices that are deployed and removed in the distributed environment.**
- **This is also called a registry and will act like a register book of all the microservices.**
- **This is achieved by** `Eureka Server`**.**

Menu

00:06:00:00:39 06/ 39

Slide Note

Eureka Server helps maintain a very stale ecosystem of microservices collaborating among each other.

Eureka Server is an application that holds the information about all client-service applications.

Think of it as a look-up service where microservices (clients) can register themselves and discover other registered microservices. When a client microservice registers with Eureka it provides metadata such as host, port, and health indicator, thus allowing for other microservices to discover it. The Discovery Server expects a regular heartbeat message from each microservice instance. If an instance begins to consistently fail to send a heartbeat, the Discovery Server will remove the instance from his registry. This way, we will have a very stable ecosystem of microservices collaborating among each other, plus we don't have to manually maintain address of other microservice, which is a next to impossible task if the scale up/down is very frequent, on demand and we use virtual host to host the services specially in the cloud environment.

Eureka Server is an application that holds the information about all client-service applications. Every microservice 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.

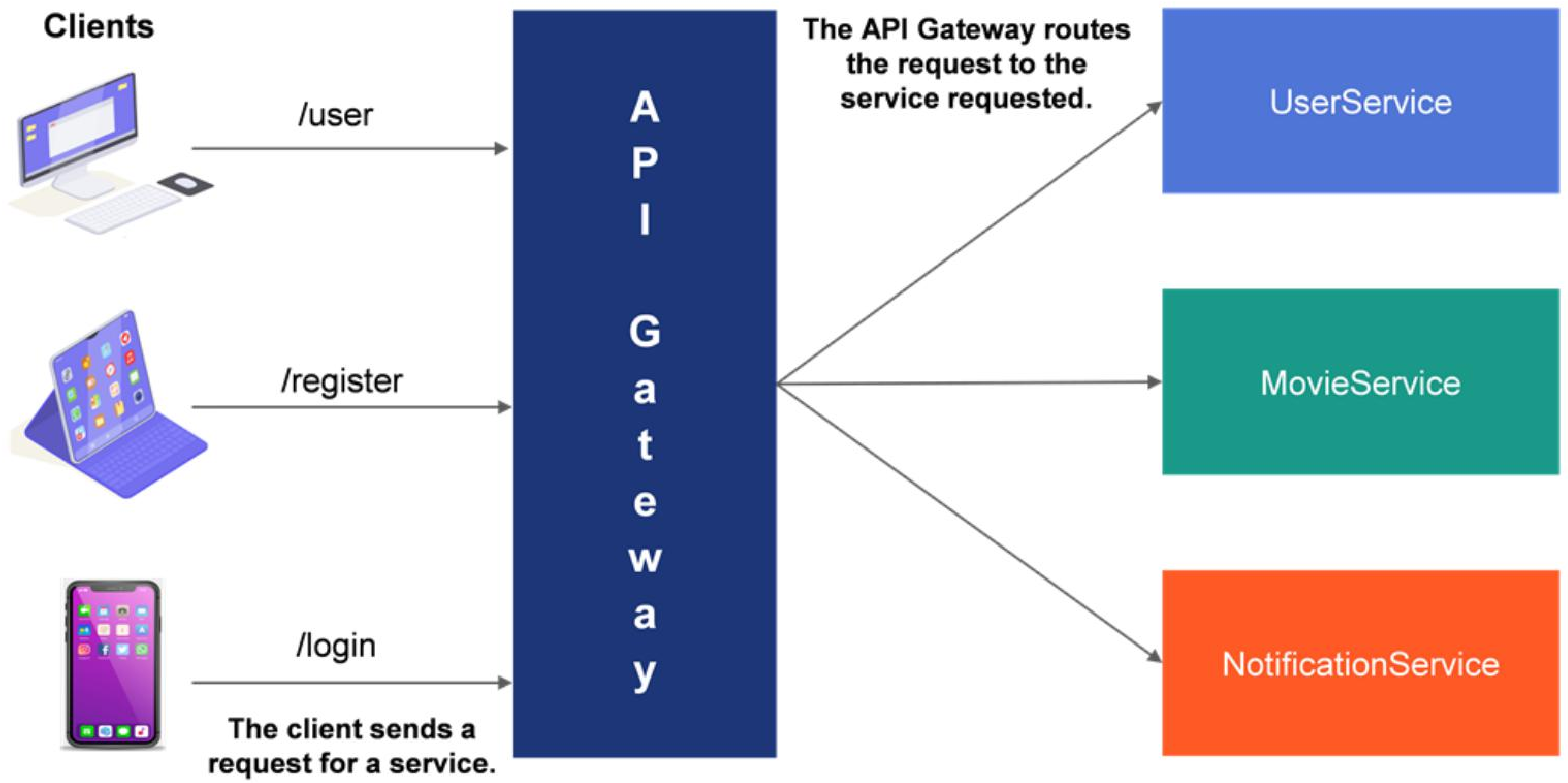
What Is Eureka Server?

- Eureka Server acts as a registry where microservices are registered.
- The registered services can be discovered by other registered microservices for effective communication between them.
- When a microservice registers with Eureka it provides metadata such as host, port, and health indicator, thus allowing for other microservices to discover it.
- Eureka server has information of all the microservices running, like the port number and IP address of every microservice registered with it.
- Eureka Server is also known as Discovery Server.

Menu

Slide Note

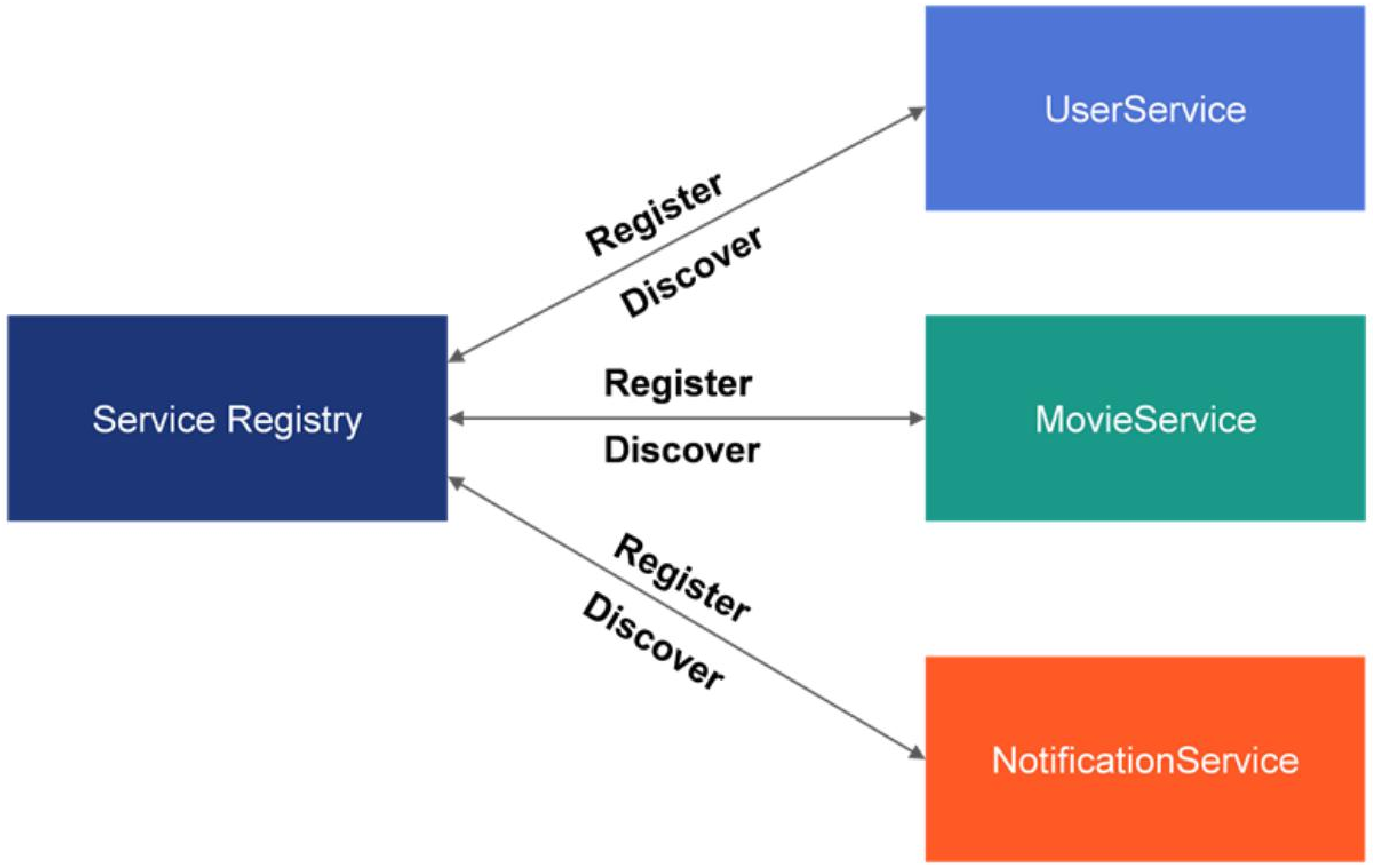
API Gateway Routes Request to Services


Clients
/user
/register
/login
The client sends a request for a service.
API Gateway
The API Gateway routes the request to the service requested.
UserService
MovieService
NotificationService

Menu 00:08:00:00:39 08/ 39

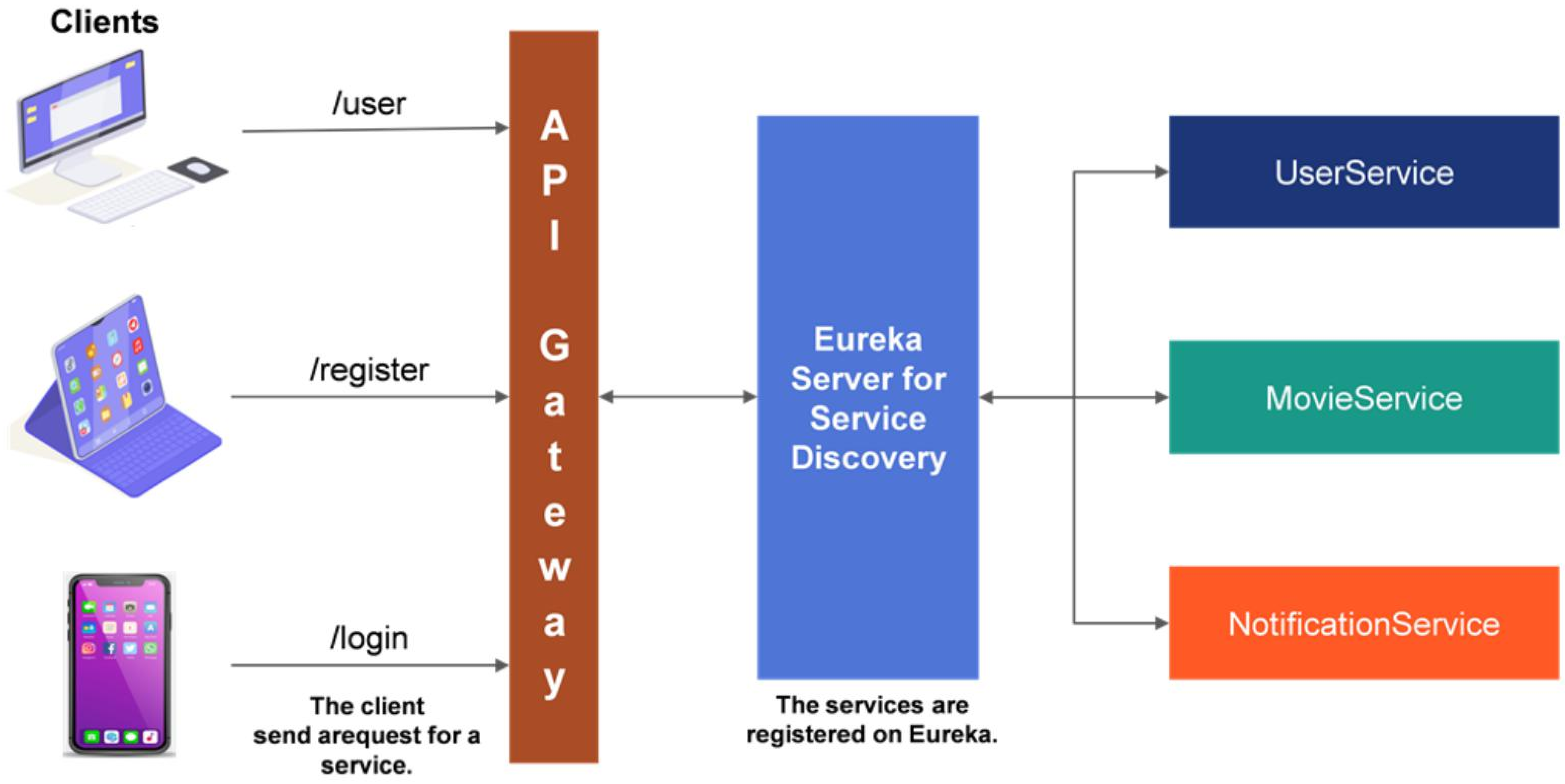
Slide Note

How Service Discovery Works



Menu

Service Discovery and API Gateway


Clients
/user
/register
/login
The client send arequest for a service.
API Gateway
Eureka Server for Service Discovery
The services are registered on Eureka.
UserService
MovieService
NotificationService

Slide Note

Service Discovery Design Pattern - How It Works

- Services are registered to the Discovery Server.
- The service discovery server listens for the registered service when it starts up.
- The service discovery server sends a heartbeat continuously to check for services.
- There is a timeout period after which its assumed that the service is offline.
- Once the time-out threshold is reached, the server assumes that the service is down and does not route the client request to that service until the service starts working again.
- The service discovery pattern is called noninvasive, as it does not alter the code in any of the microservices.

Menu

00:11:00:00:39 11/ 39

Slide Note

Quick Check

How does the service discovery server know that a service is down?

1. Heartbeat
2. Pulse
3. Time-out
4. Discovery

Menu

Slide Note

Quick Check: Solution

How does the service discovery server know that a service is down?

1. Heartbeat
2. Pulse
3. Time-out
4. Discovery

Slide Note

Implementing the Service Discovery Server Using Eureka

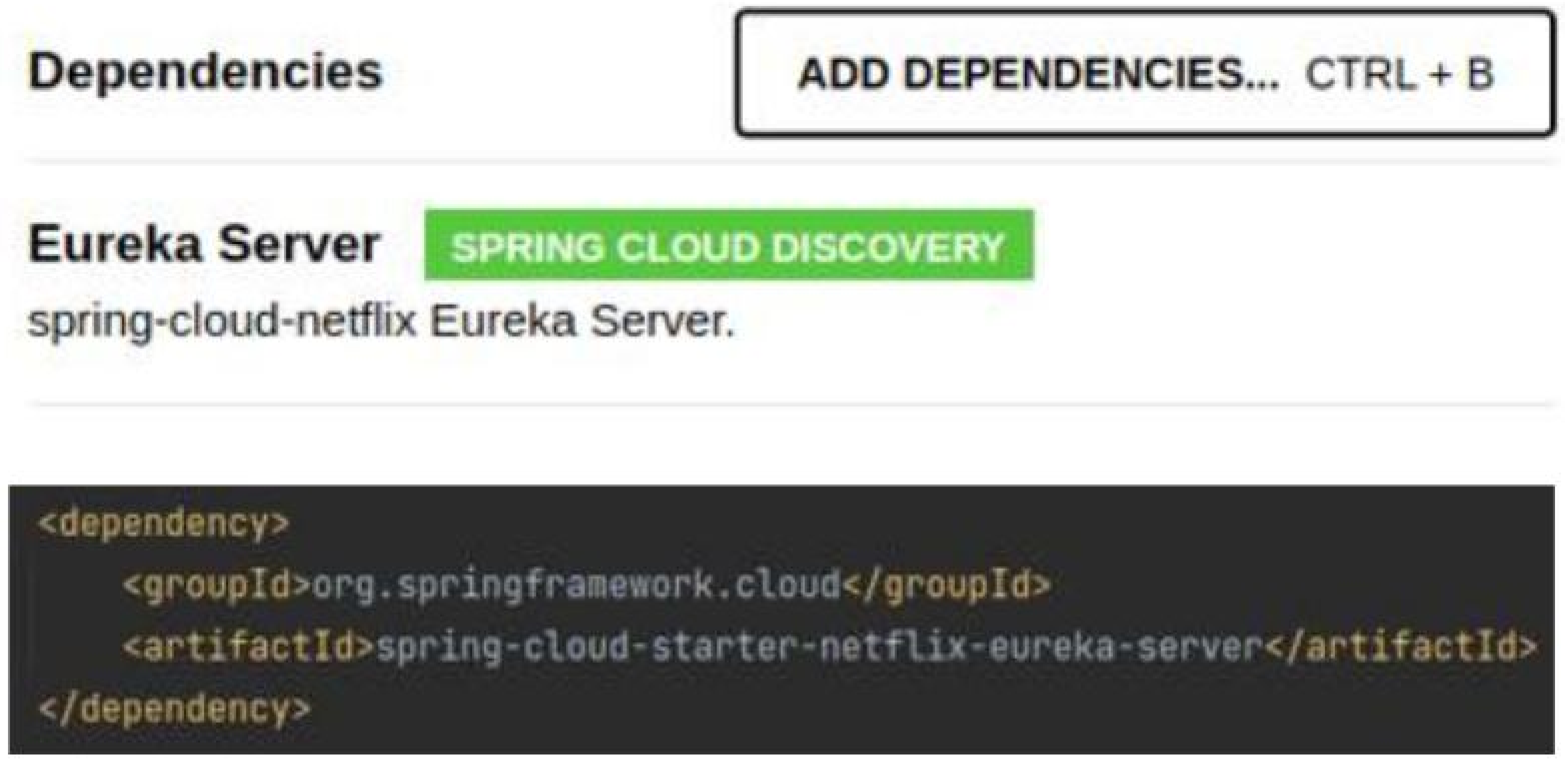
Menu

00:14:00:00:39 14/ 39

Slide Note

Create the Eureka Server

- Create a service, name it 'Eureka Server', and add the dependency below to it.

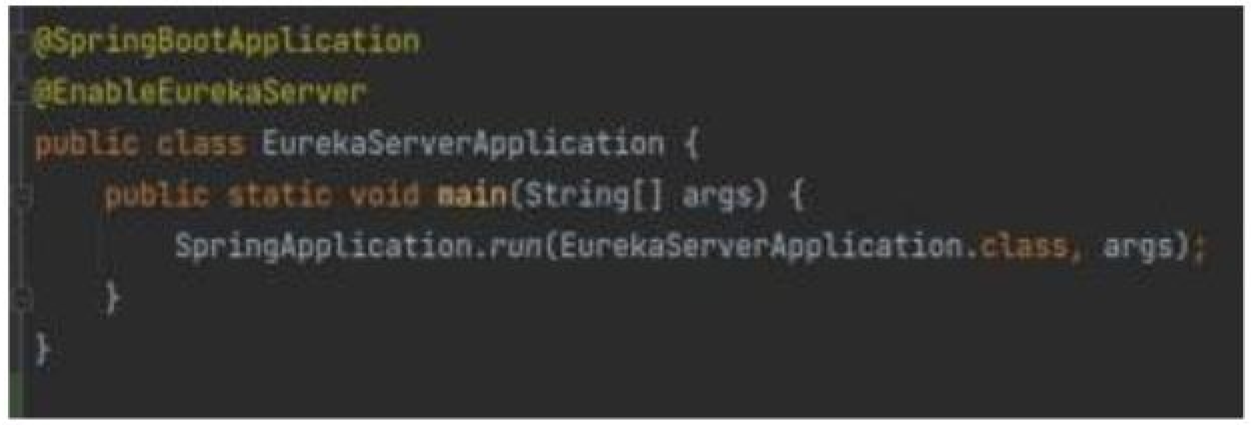


spring-cloud-netflix Eureka Server.

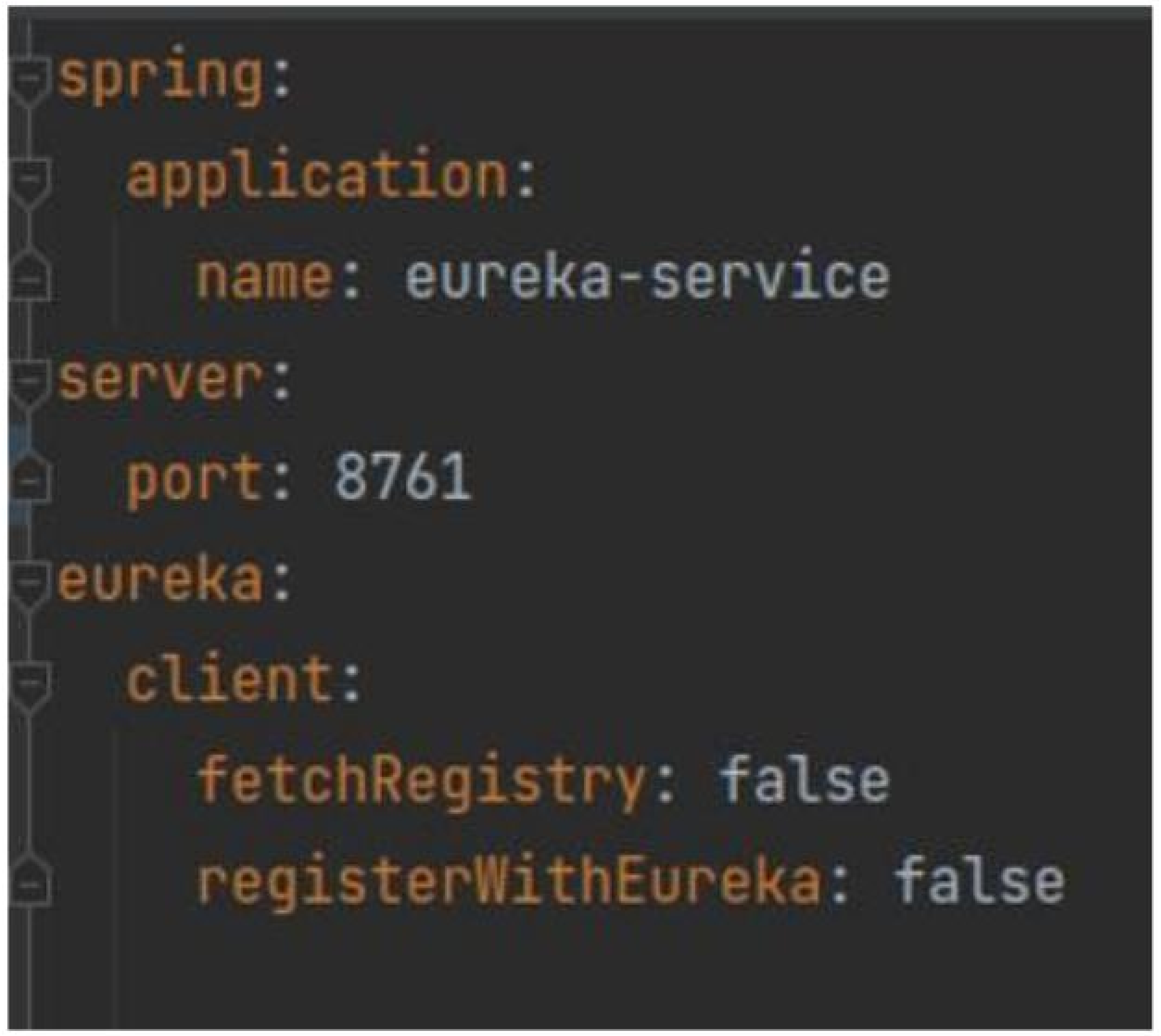
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Slide Note

Enable the Server in the Application



```java
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

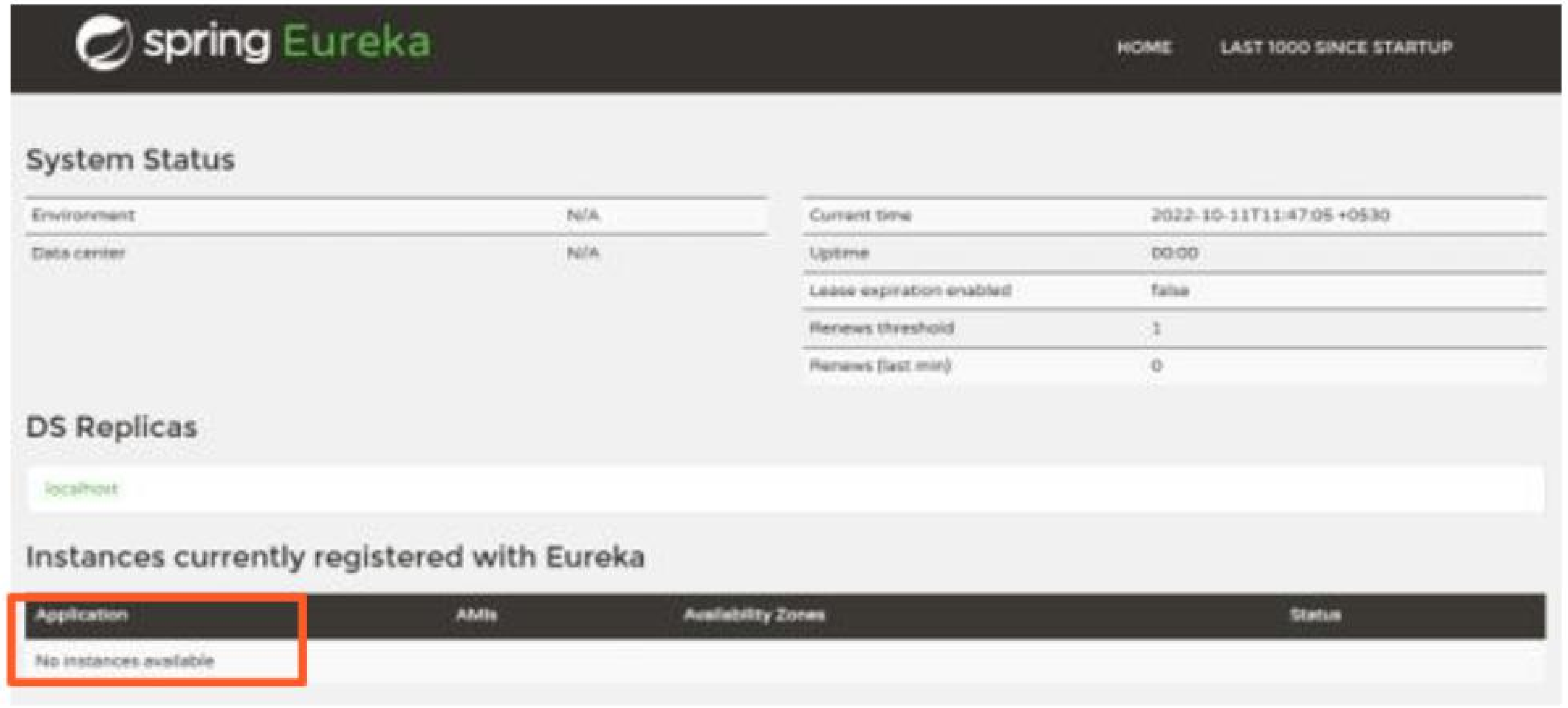


```yaml
spring:
  application:
    name: eureka-service
server:
  port: 8761
eureka:
  client:
    fetchRegistry: false
    registerWithEureka: false
```

- **Annotate the main class with** `@EnableEurekaServer;` **this will act as a Eureka Server.**
- **Mention the service name and the server port where the server will run in the** `application.properties` **or** `application.yml` **file.**
- `registerWithEureka: false` **means informing the server not to register itself in the service registry.**

Slide Note

Start Eureka Server



System Status

Environment	N/A
Data center	N/A

Current time	2022-10-11T11:47:05 +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			

- Start the server and access the service running at http://localhost:8761/.
- At this point, no service is registered here as expected.
- Once started, the client services server will automatically update the details of the client services.

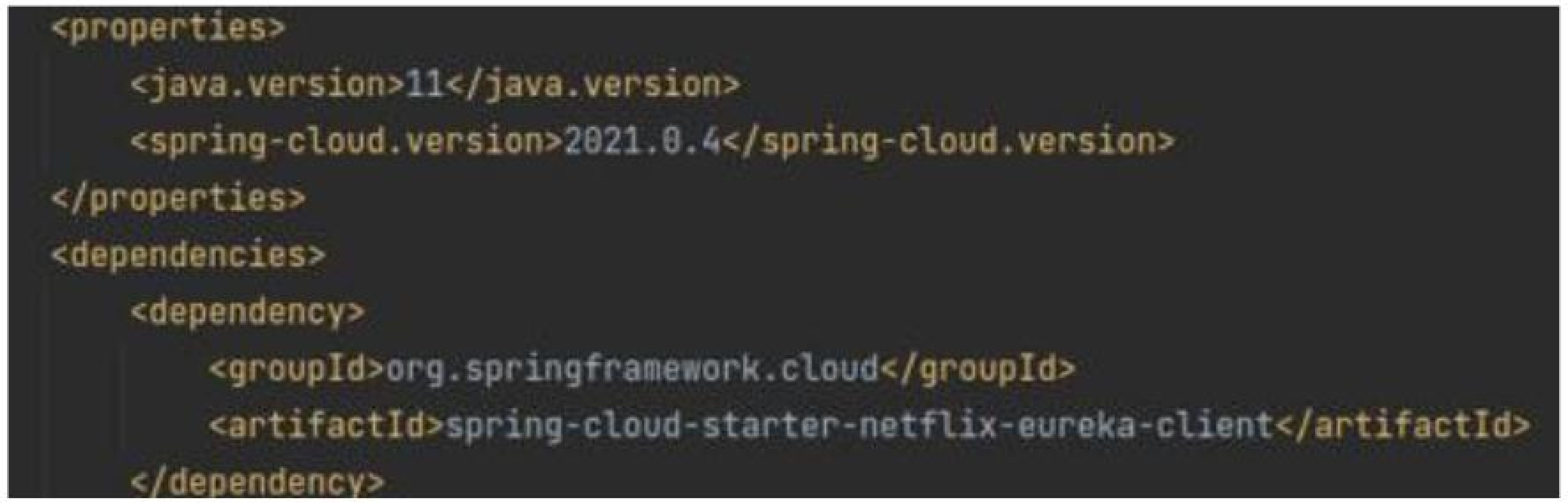
Slide Note

Registering the Services on the Eureka Server

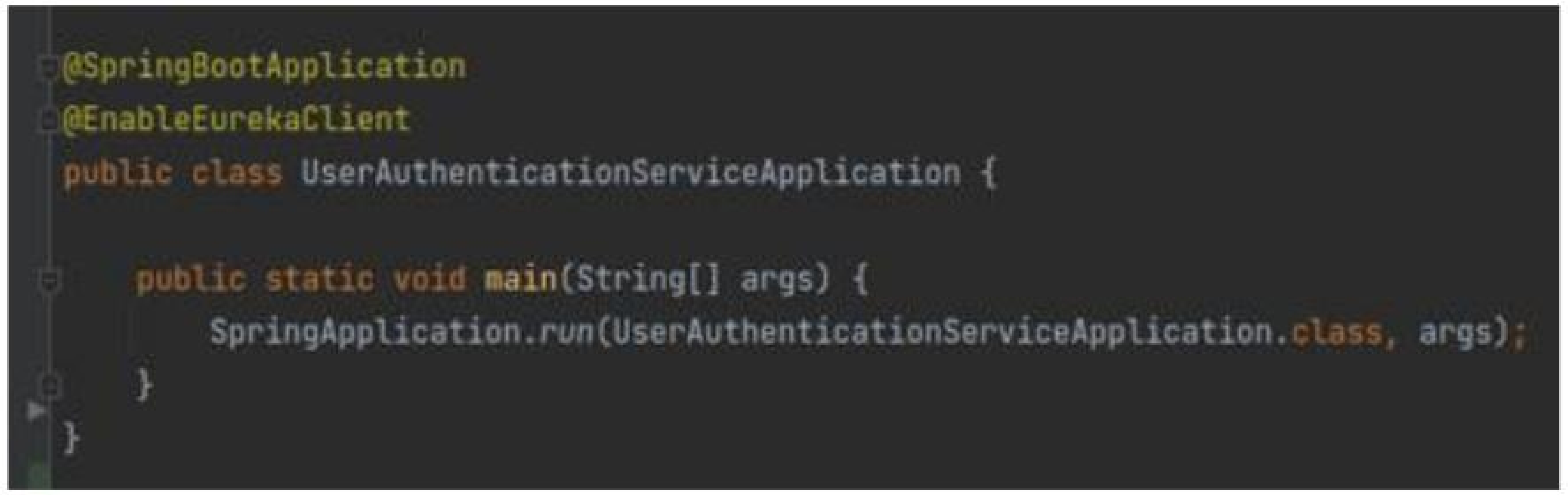
Menu

Slide Note

Create Eureka Client



```
<properties>
    <java.version>11</java.version>
    <spring-cloud.version>2021.0.4</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
```



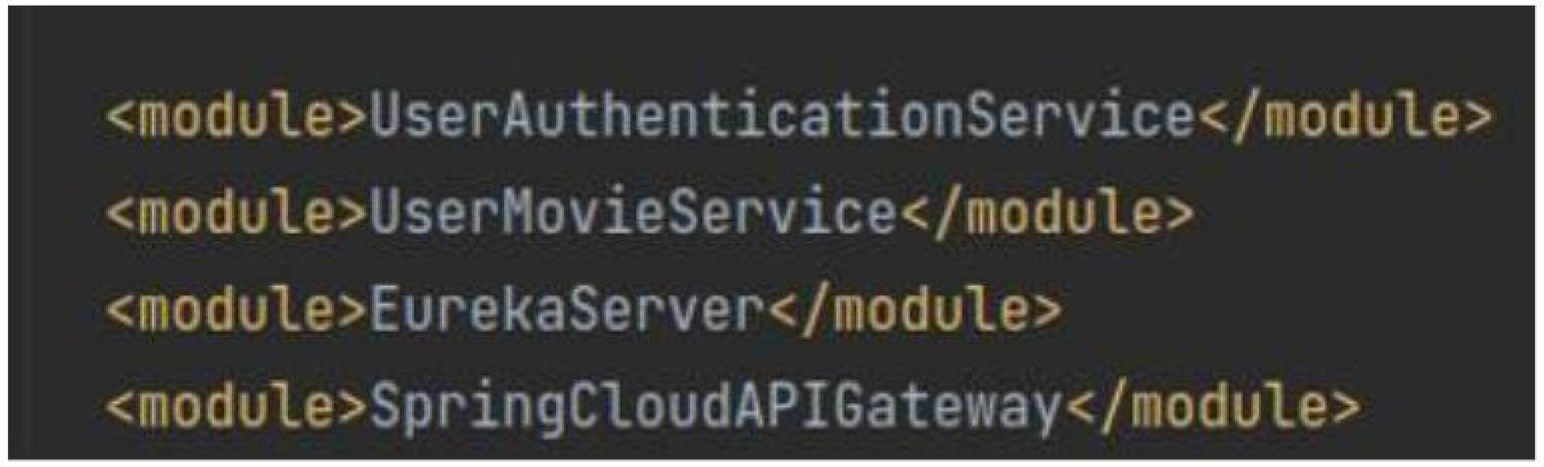
```
@SpringBootApplication
@EnableEurekaClient
public class UserAuthenticationServiceApplication {

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

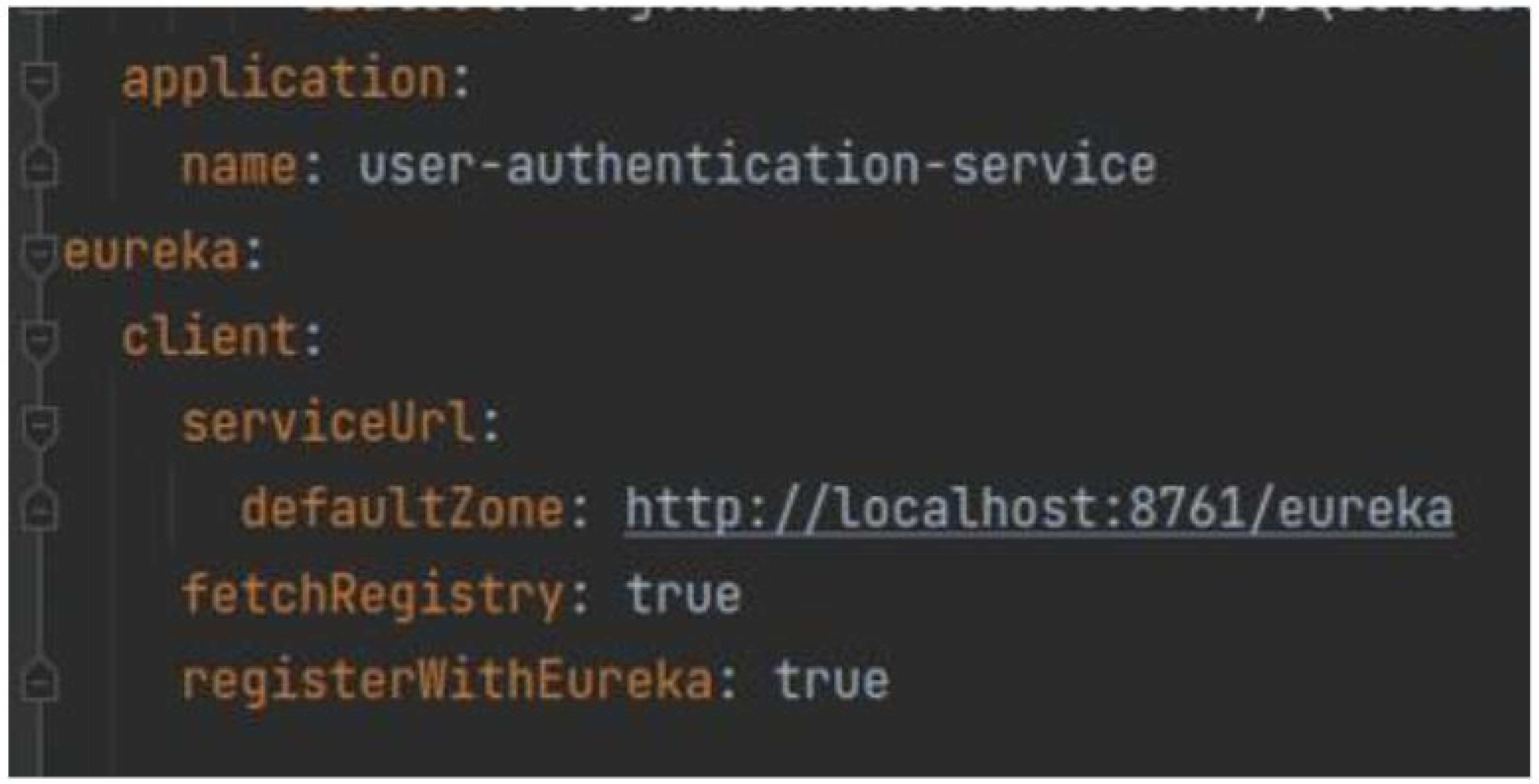
- To register a microservice, also known as a Eureka client, on the Eureka server, follow the steps steps.
- Step 1: Add the dependencies below in the pom.xml of the service.
- Step 2: Enable the microservice with the `@EnableEurekaClient` annotation.

Menu

Create Eureka Client (contd.)



```
<module>UserAuthenticationService</module>
<module>UserMovieService</module>
<module>EurekaServer</module>
<module>SpringCloudAPIGateway</module>
```

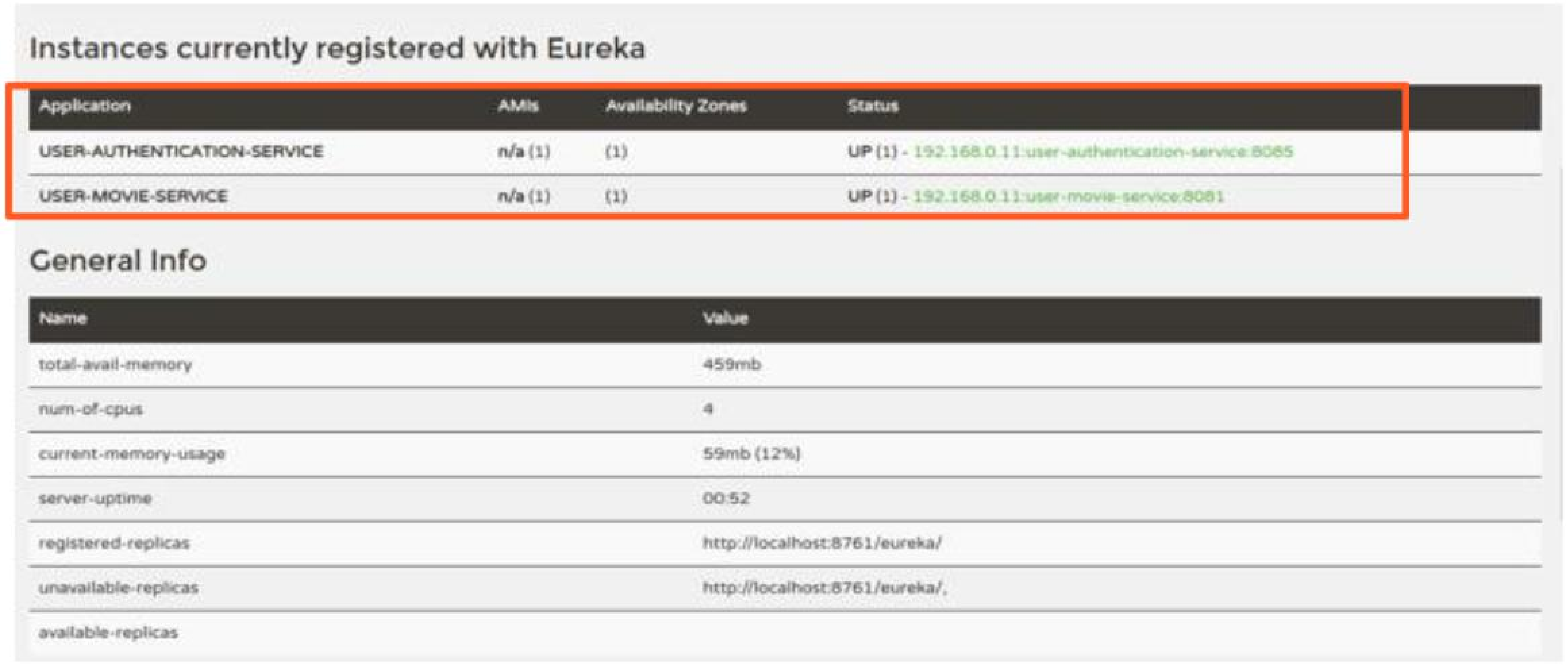


```
  application:
    name: user-authentication-service
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
    fetchRegistry: true
    registerWithEureka: true
```

- **Step 3: To register the Spring Boot application in Eureka Server, you need to add the following configuration in the application.properties file.**
- **DefaultZone sets the Eureka Server URL in the configuration.**
- `registerWithEureka` **is set as true. This means it will register with Eureka Server.**
- **Add the Eureka Server as a module in the parent pom.**
- **Step 4: Run the individual services.**

Eureka Server With Registered Services

- Refresh the browser at http://localhost:8761/.



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.11:user-authentication-service:8085
USER-MOVIE-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.11:user-movie-service:8081

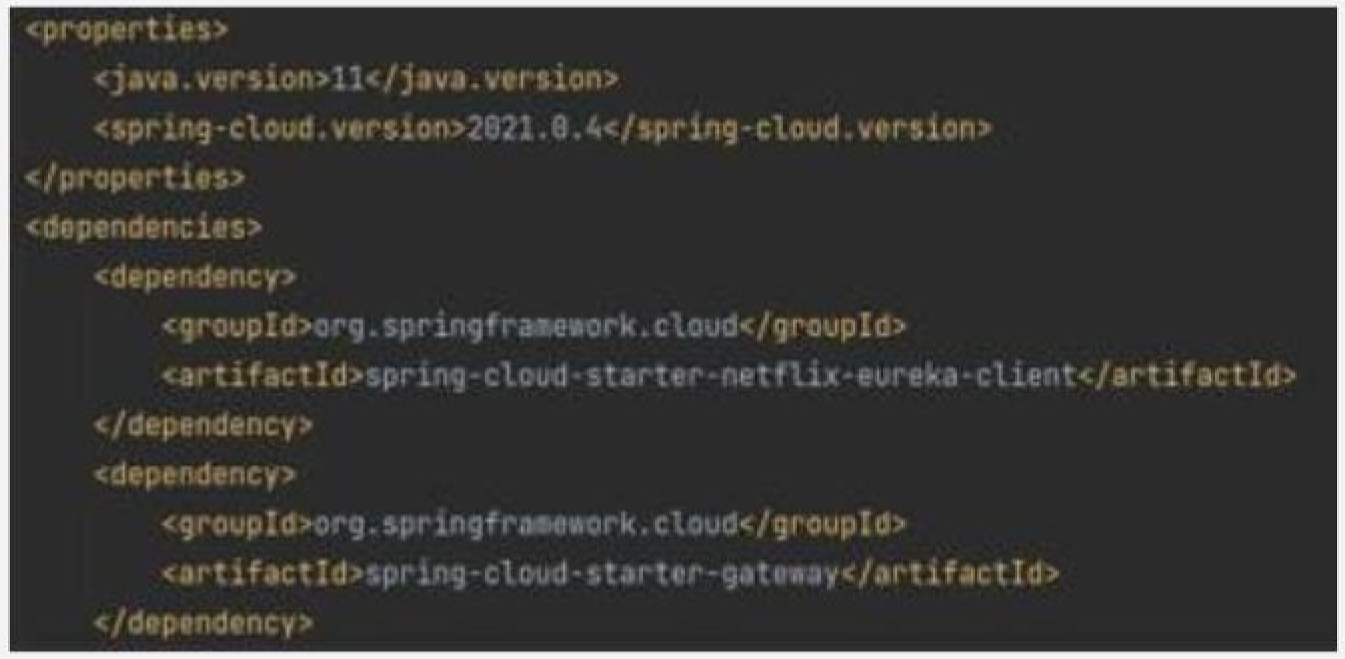
General Info

Name	Value
total-avail-memory	459mb
num-of-cpus	4
current-memory-usage	59mb (12%)
server-uptime	00:52
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

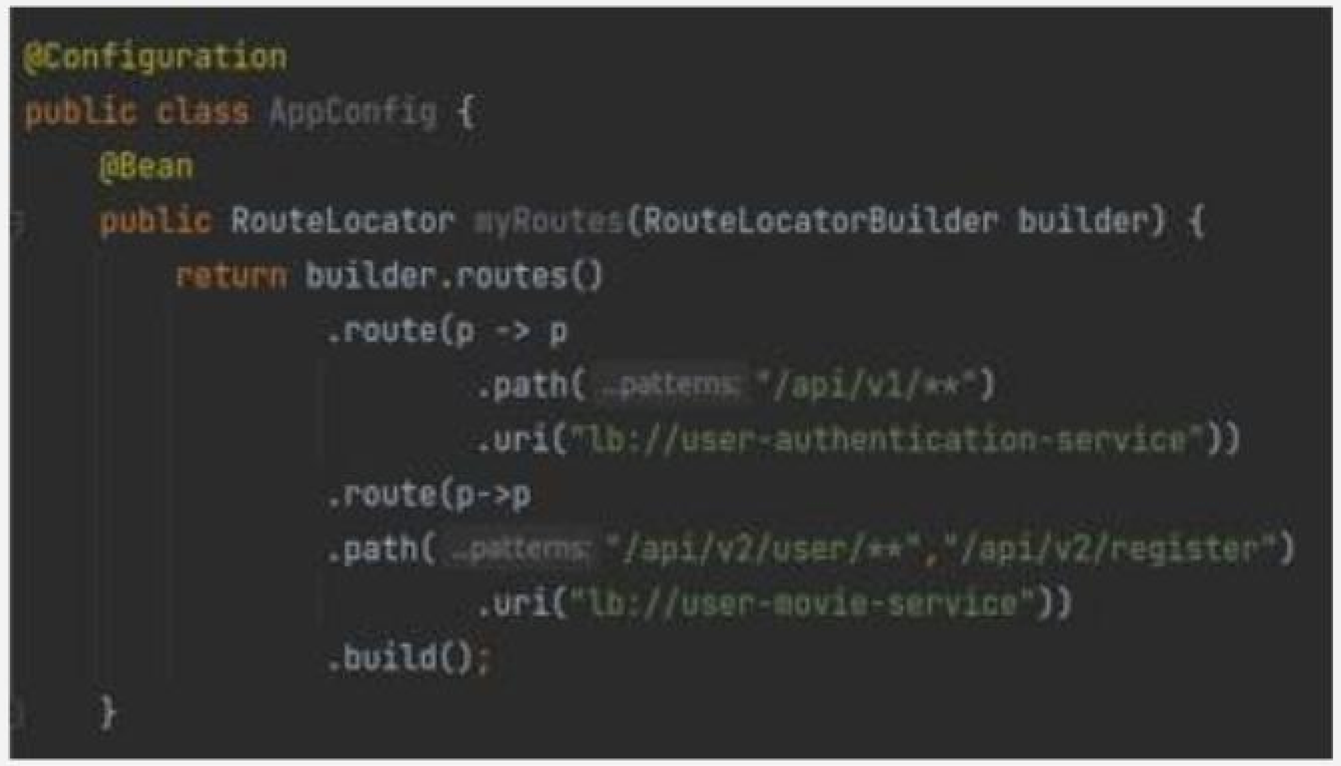
- Here, `UserAuthenticationService` and `UserMovieService` are Eureka clients. They are getting registered with the Eureka Server.

Slide Note

Register the API Gateway to the Eureka Server



```xml
<properties>
    <java.version>11</java.version>
    <spring-cloud.version>2021.0.4</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
```

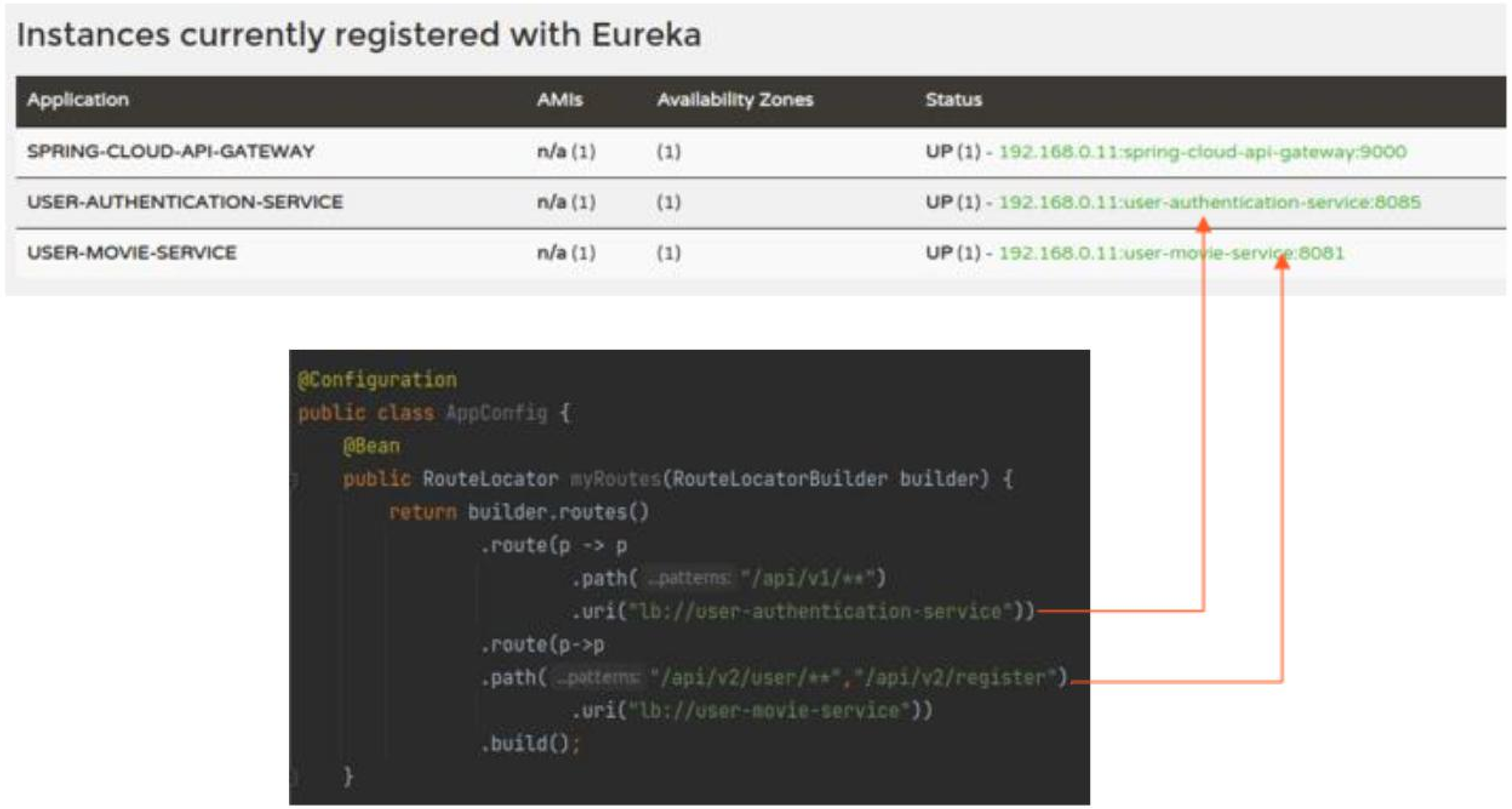


```java
@Configuration
public class AppConfig {
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(p -> p
                        .path( ...patterns: "/api/v1/**")
                        .uri("lb://user-authentication-service"))
                .route(p->p
                .path( ...patterns: "/api/v2/user/**","/api/v2/register")
                        .uri("lb://user-movie-service"))
                .build();
    }
```

- The Spring Cloud API Gateway must be registered on the Eureka Server as a client.
- You need to add these dependencies.
- The route can also be written using the application name in the `application.yml` file, instead of the URI of the application.
- The port number and host on which the microservice or Eureka client runs is registered on the Eureka server, so we need not mention the URI.
- Here, `lb` stands for load balancing which we will discuss in the upcoming slides.

Eureka Server With the Registered Services

- Refresh the browser at http://localhost:8761/.



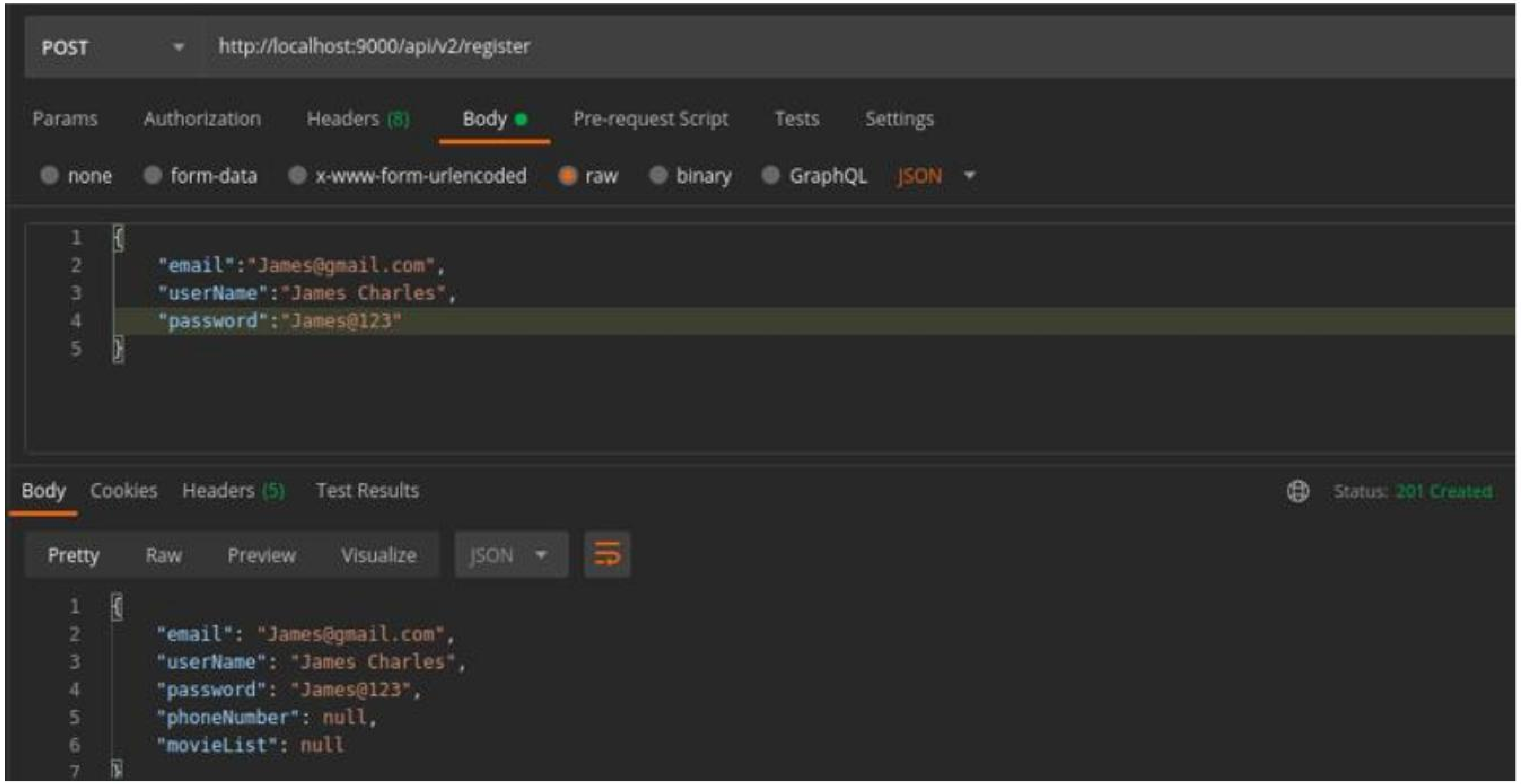
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SPRING-CLOUD-API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.0.11:spring-cloud-api-gateway:9000
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.11:user-authentication-service:8085
USER-MOVIE-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.11:user-movie-service:8081

```
@Configuration
public class AppConfig {
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(p -> p
                        .path( ...patterns: "/api/v1/**")
                        .uri("lb://user-authentication-service"))
                .route(p->p
                .path( ...patterns: "/api/v2/user/**","/api/v2/register")
                        .uri("lb://user-movie-service"))
                .build();
    }
```

Slide Note

Postman Output – Register a New User



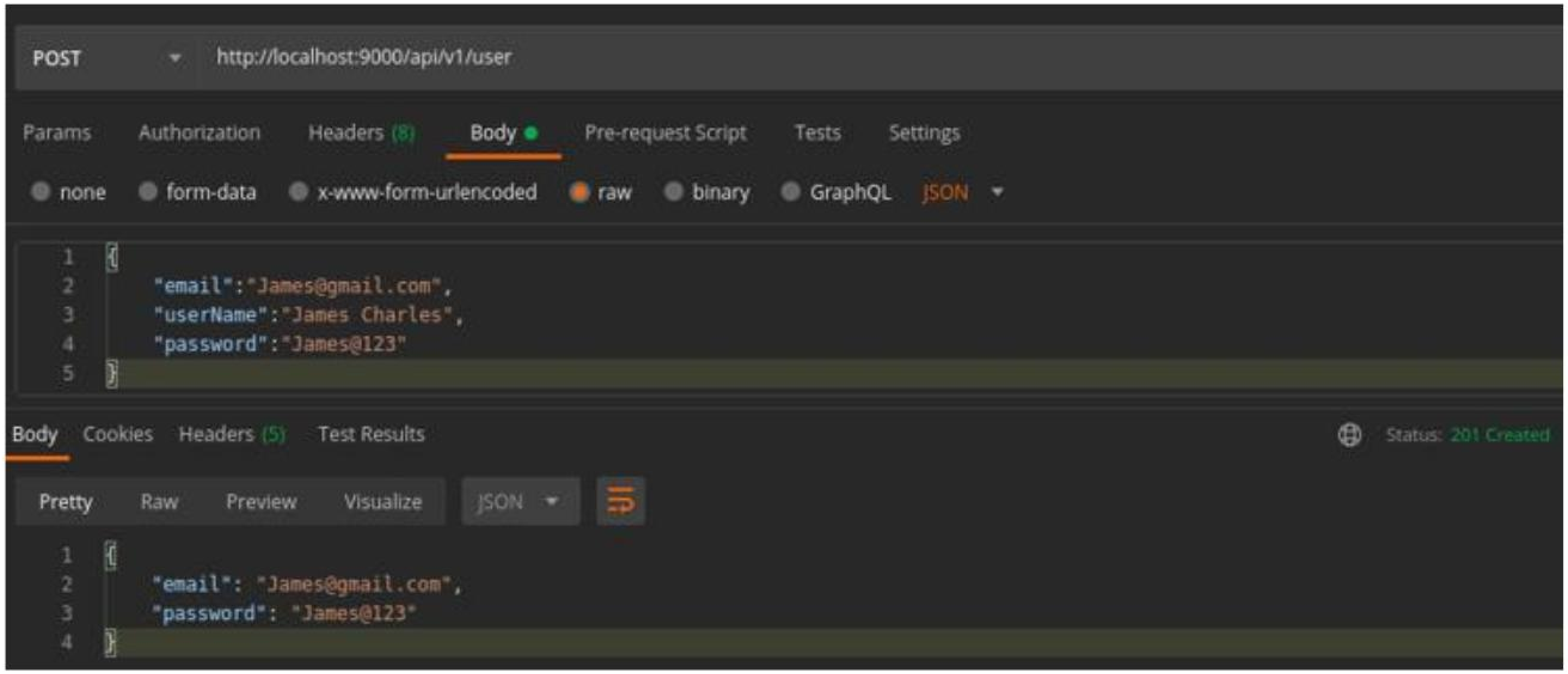
POST http://localhost:9000/api/v2/register

```
{
    "email":"James@gmail.com",
    "userName":"James Charles",
    "password":"James@123"
}
```

Status: 201 Created

```
{
    "email": "James@gmail.com",
    "userName": "James Charles",
    "password": "James@123",
    "phoneNumber": null,
    "movieList": null
}
```

Postman Output – Save User Credentials



POST http://localhost:9000/api/v1/user

Body: ● raw JSON

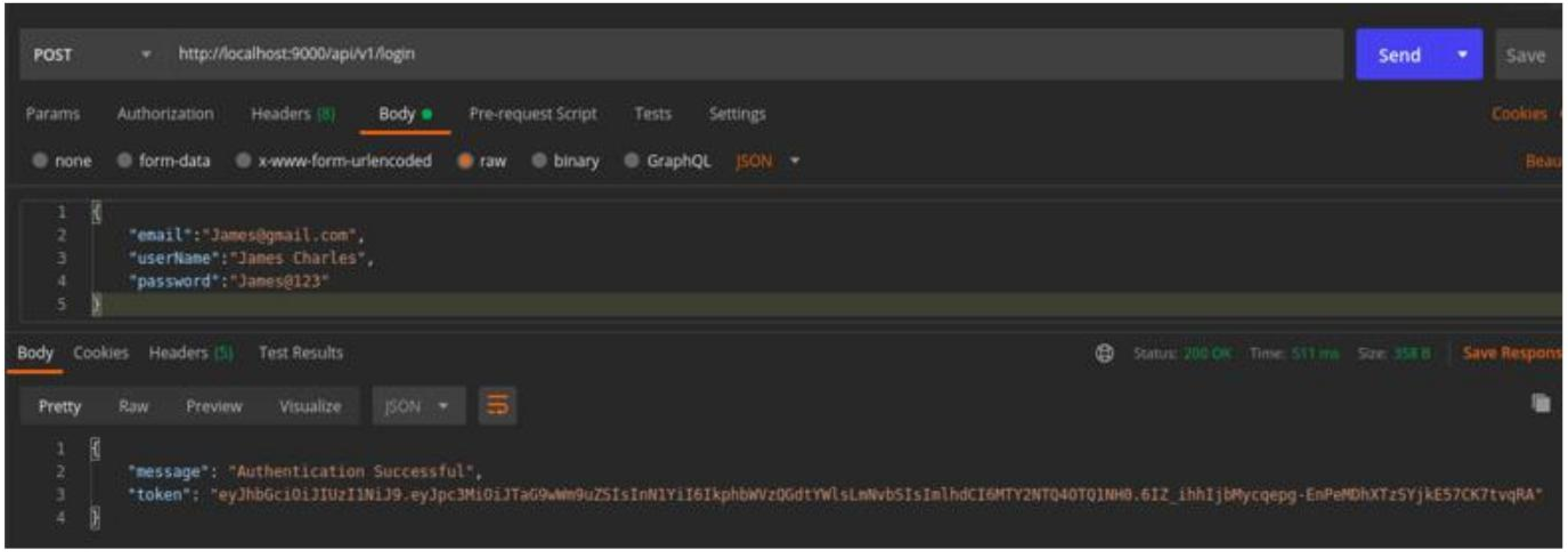
```
{
    "email":"James@gmail.com",
    "userName":"James Charles",
    "password":"James@123"
}
```

Status: 201 Created

```
{
    "email": "James@gmail.com",
    "password": "James@123"
}
```

Slide Note

Postman Output – Log in a New User



POST http://localhost:9000/api/v1/login

```
{
    "email":"James@gmail.com",
    "userName":"James Charles",
    "password":"James@123"
}
```

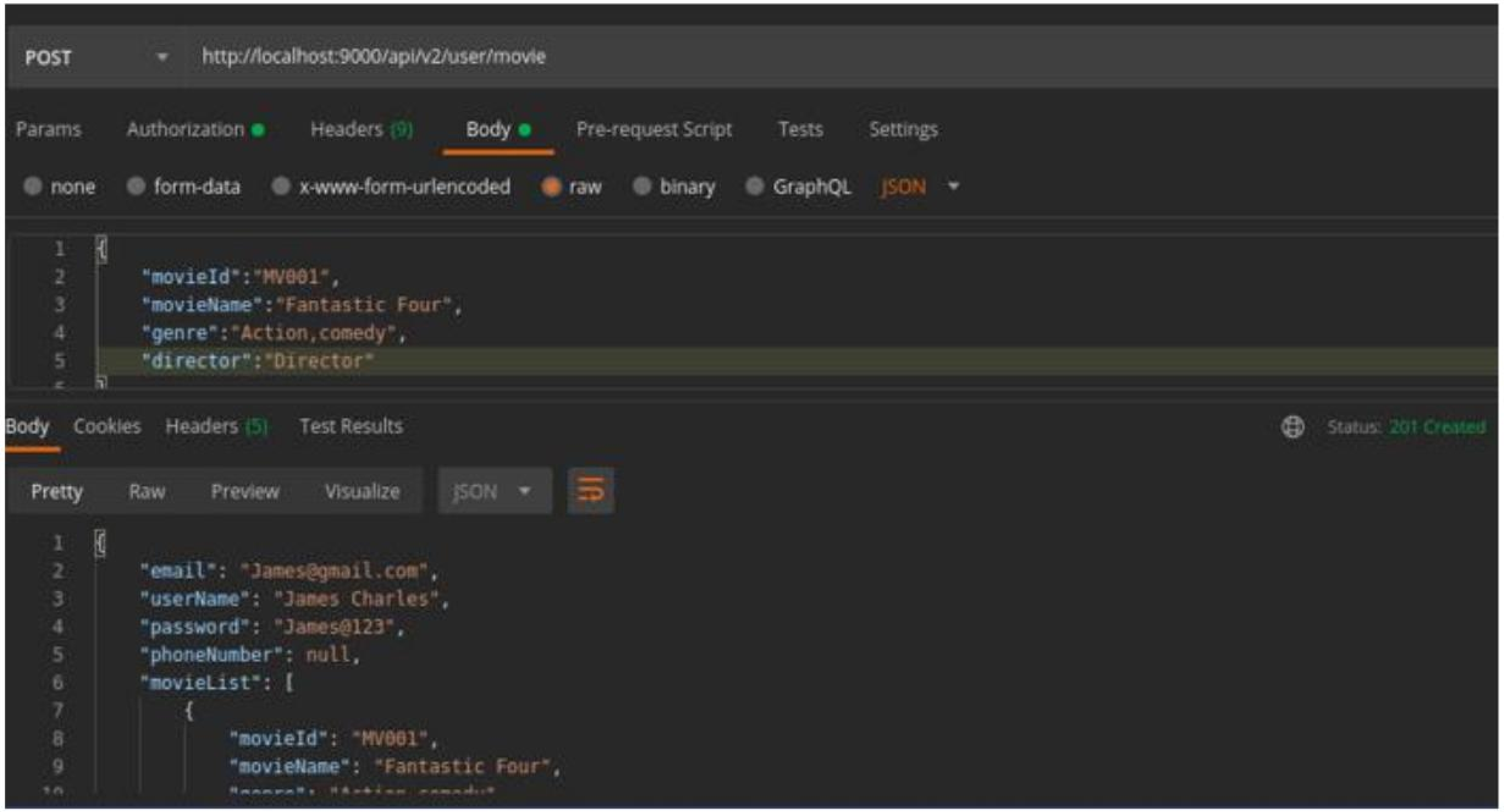
Status: 200 OK Time: 511 ms Size: 358 B

```
{
    "message": "Authentication Successful",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJTaG9wWm9uZSIsInN1YiI6IkphbWVzQGdtYWlsLmNvbSIsImlhdCI6MTY2NTQ4OTQ1NH0.6IZ_ihhIjbMycqepg-EnPeMDhXTzSYjkE57CK7tvqRA"
}
```

Menu

00:26 00:00:39 26/ 39

Postman Output – Add the Favorite Movie for a User



http://localhost:9000/api/v2/user/movie

```
{
    "movieId":"MV001",
    "movieName":"Fantastic Four",
    "genre":"Action,comedy",
    "director":"Director"
}
```

Status: 201 Created

```
{
    "email": "James@gmail.com",
    "userName": "James Charles",
    "password": "James@123",
    "phoneNumber": null,
    "movieList": [
        {
            "movieId": "MV001",
            "movieName": "Fantastic Four",
```

Slide Note

Streaming Application

Consider a streaming application that enables users to watch movies on any smart device. The application provides multiple features, and some features are accessible only to registered users. Let's create multiple microservices.

A user must first register with the application.

1. Use credentials such as id and password to log in.
2. Access the features provided by the streaming application, like adding favorites, compiling a watch later list, etc.

DEMO

Menu

Slide Note

Streaming Application (contd.)

Let's create a parent project called MovieApplication.

- This will contain the UserAuthenticationService and the UserMovieService as microservices.
- Create a Eureka Discovery Server and register the services on a Eureka Server.
- Check the solution here.

DEMO

Menu

00:29 00:00:39 29/ 39

Slide Note

Load Balancers

Menu

00:30 00:00:39 30/ 39

Slide Note

Think and Tell

- Modern high-traffic websites must serve hundreds of thousands, even millions, of concurrent requests from users or clients. The response needs to be served in a fraction of a second.
- The response can consist of images, text, video, or some application data, all in a fast and reliable manner.
- How can the services meet such high-volume requirements?
- What will happen when one of the services goes down?

Menu

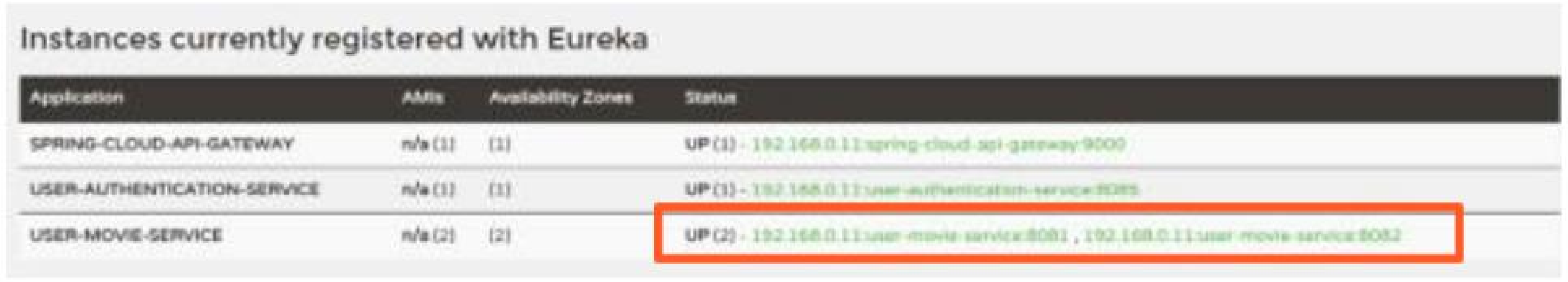
Slide Note

Load Balancer: Features and Functions

- Load balancing refers to efficiently distributing incoming request traffic across a group of backend servers.
- It acts as the traffic cop sitting in front of your servers and routing the client requests across all servers equally. This is to ensure that no single server is overworked, resulting in lowering the performance
- If a server goes down, the load balancer redirects traffic to the remaining servers.
- It ensures high availability and reliability by sending requests only to servers that are online.
- Spring Cloud Gateway and Eureka make an amazing combination to scale Spring applications easily in production environments and load balances them effectively.

Slide Note

Load Balancing Using Eureka and Spring Cloud Gateway



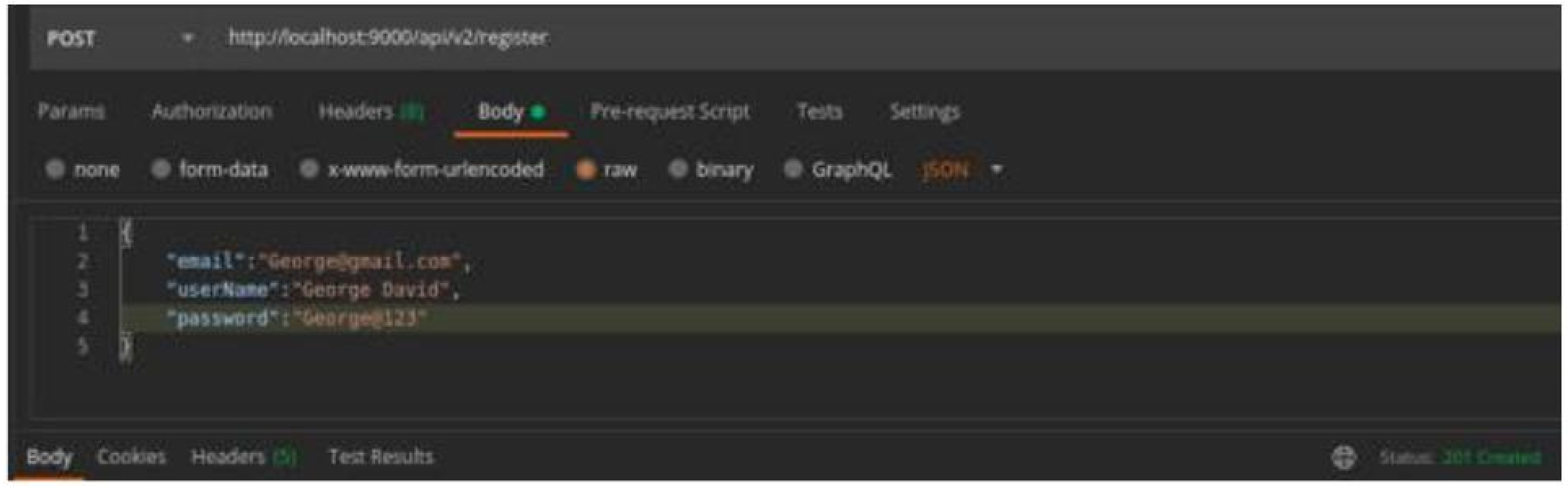
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SPRING-CLOUD-API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.0.11:spring-cloud-api-gateway:9000
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.11:user-authentication-service:8085
USER-MOVIE-SERVICE	n/a (2)	(2)	UP (2) - 192.168.0.11:user-movie-service:8081 , 192.168.0.11:user-movie-service:8082

- The two User-Movie-Service two instances are up.
- The first instance runs on port 8081 and the second instance runs on port 8082.

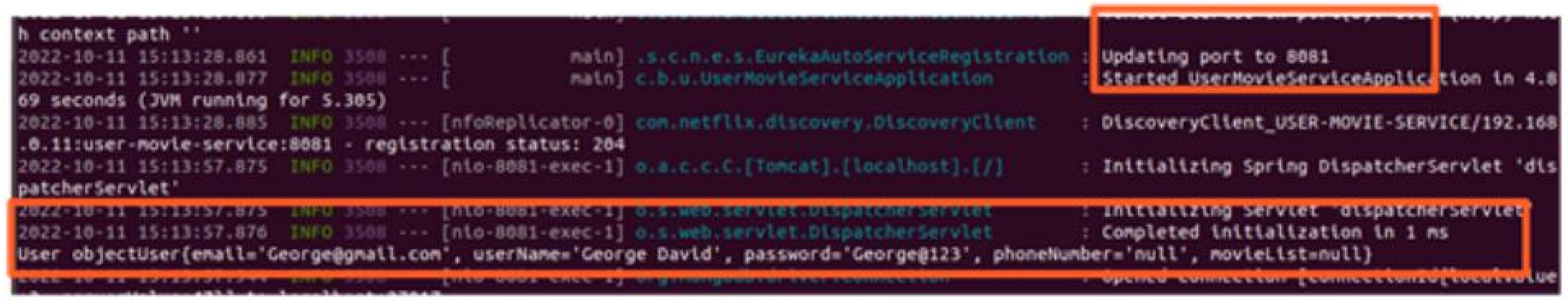
Slide Note

Postman Output – Register a New User



http://localhost:9000/api/v2/register

```
{
    "email":"George@gmail.com",
    "userName":"George David",
    "password":"George@123"
}
```



```
h context path ''
2022-10-11 15:13:28.861  INFO 3508 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8081
2022-10-11 15:13:28.877  INFO 3508 --- [           main] c.b.u.UserMovieServiceApplication        : Started UserMovieServiceApplication in 4.8
69 seconds (JVM running for 5.305)
2022-10-11 15:13:28.885  INFO 3508 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_USER-MOVIE-SERVICE/192.168
.0.11:user-movie-service:8081 - registration status: 204
2022-10-11 15:13:57.875  INFO 3508 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dis
patcherServlet'
2022-10-11 15:13:57.875  INFO 3508 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2022-10-11 15:13:57.876  INFO 3508 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
User objectUser{email='George@gmail.com', userName='George David', password='George@123', phoneNumber='null', movieList=null}
```

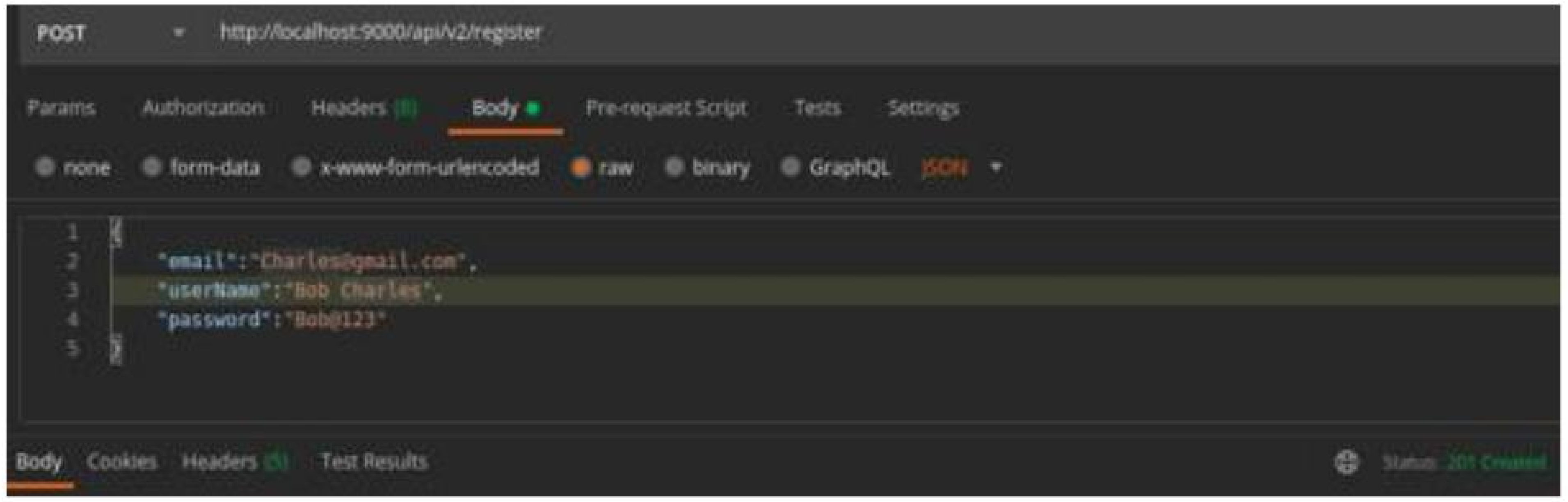
- While registering the user the UserMovieService running on port 8081 is gets called.

Menu

00:34 00:00:39 34/ 39

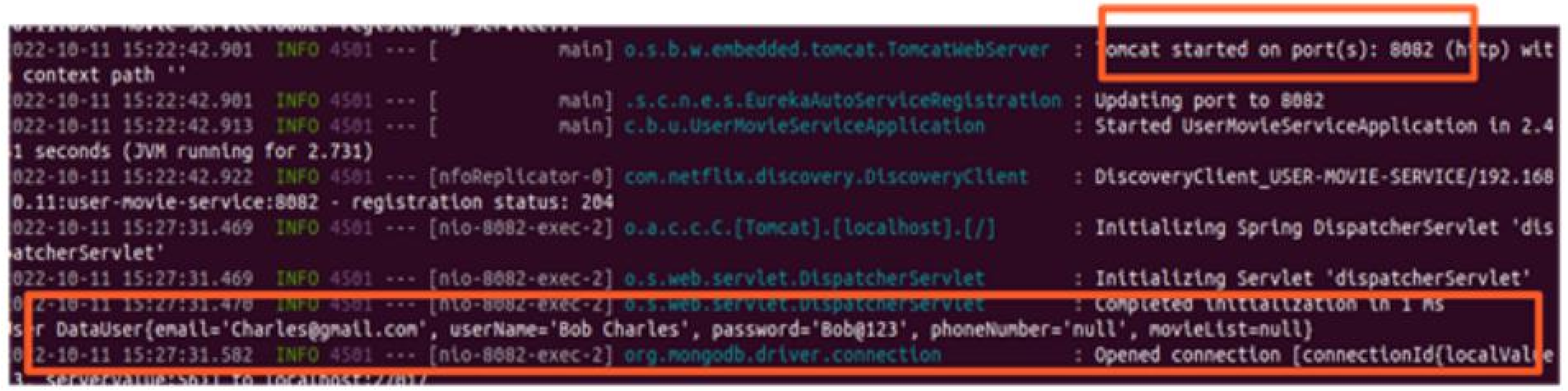
Slide Note

Postman Output – Register a New User (contd.)



```
http://localhost:9000/api/v2/register
```

```
{
    "email":"Charles@gmail.com",
    "userName":"Bob Charles",
    "password":"Bob@123"
}
```



```
022-10-11 15:22:42.901  INFO 4501 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : omcat started on port(s): 8082 (http) wit
 context path ''
022-10-11 15:22:42.901  INFO 4501 --- [           main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8082
022-10-11 15:22:42.913  INFO 4501 --- [           main] c.b.u.UserMovieServiceApplication        : Started UserMovieServiceApplication in 2.4
1 seconds (JVM running for 2.731)
022-10-11 15:22:42.922  INFO 4501 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_USER-MOVIE-SERVICE/192.168
0.11:user-movie-service:8082 - registration status: 204
022-10-11 15:27:31.469  INFO 4501 --- [nio-8082-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dis
atcherServlet'
022-10-11 15:27:31.469  INFO 4501 --- [nio-8082-exec-2] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
022-10-11 15:27:31.470  INFO 4501 --- [nio-8082-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
ser DataUser{email='Charles@gmail.com', userName='Bob Charles', password='Bob@123', phoneNumber='null', movieList=null}
022-10-11 15:27:31.582  INFO 4501 --- [nio-8082-exec-2] org.mongodb.driver.connection            : Opened connection [connectionId{localValue
```

- Now let's assume that for some reason UserMovieService does not run on port 8081.
- If the request comes for registering the user, the load balancer will detour the call to the service running on port 8082.
- This is possible due to the Spring Cloud API because no port number is mentioned in the application file of spring cloud API.
- The API Gateway automatically checks the port that is up for microservices and then redirects the request accordingly.

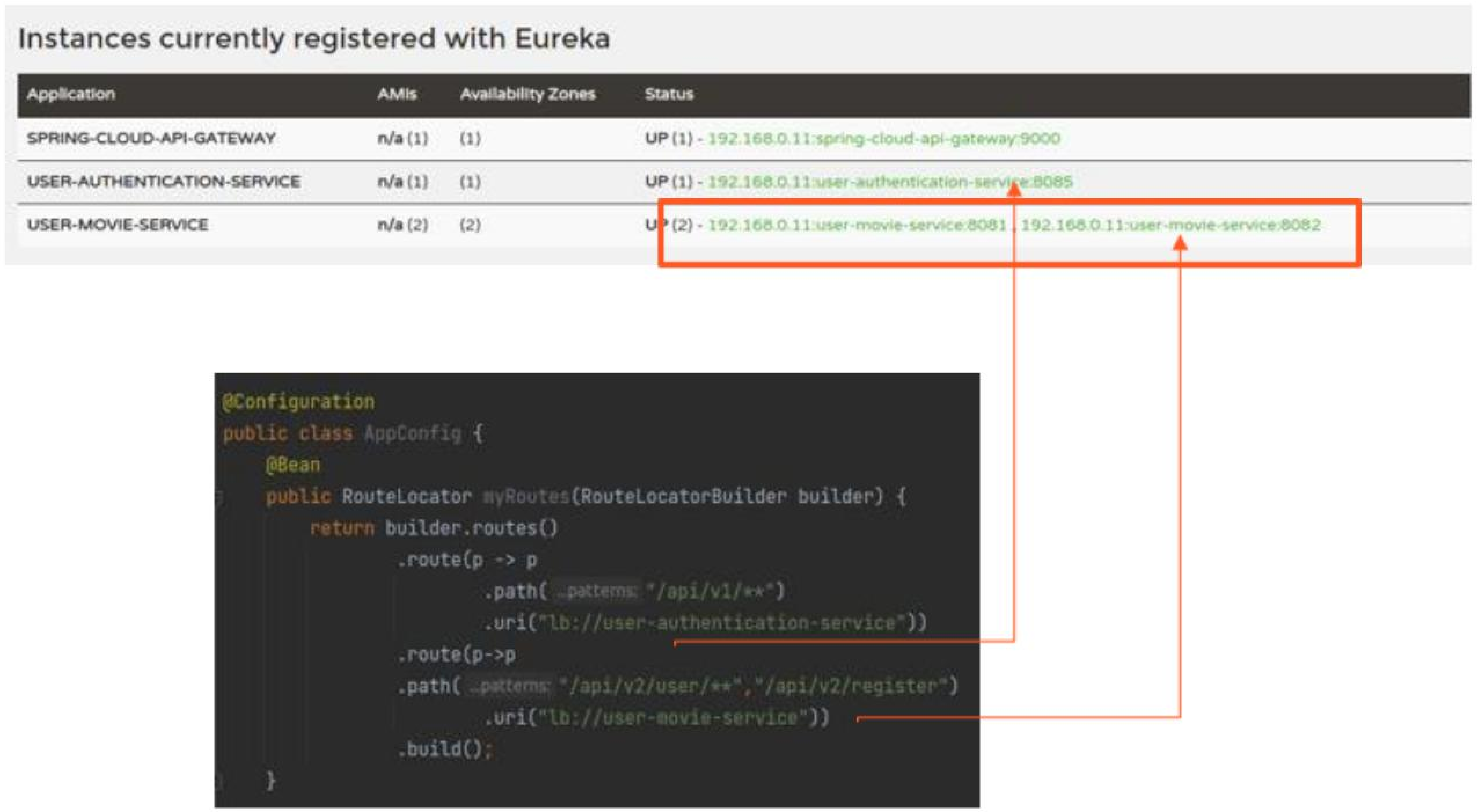
Menu 00:35:00:00:39 35/ 39

Slide Note

Add one point that lb is load balancer here

Eureka Server With the Registered Services

- Refresh the browser at http://localhost:8761/.



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SPRING-CLOUD-API-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.0.11:spring-cloud-api-gateway:9000
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - 192.168.0.11:user-authentication-service:8085
USER-MOVIE-SERVICE	n/a (2)	(2)	UP (2) - 192.168.0.11:user-movie-service:8081 , 192.168.0.11:user-movie-service:8082

```java
@Configuration
public class AppConfig {
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(p -> p
                        .path( ...patterns: "/api/v1/**")
                        .uri("lb://user-authentication-service"))
                .route(p->p
                .path( ...patterns: "/api/v2/user/**","/api/v2/register")
                        .uri("lb://user-movie-service"))
                .build();
    }
```

Menu

00:36 00:00:39 36/ 39

Slide Note

Quick Check

What is the default port of the Eureka Server?

1. 8080
2. 8090
3. 8761
4. 8763

Menu

00:37:00:00:39 37/ 39

Slide Note

Quick Check: Solution

What is the default port of the Eureka Server?

1. 8080
2. 8090
3. 8761
4. 8763

Menu

00:38:00:00:39

38/ 39