Backend Program: Course 3: Structure

Program — **Courses** — **Learning Sprints**

Build job-ready skills

Enterprise Application Development By Using Spring Framework

Build competencies

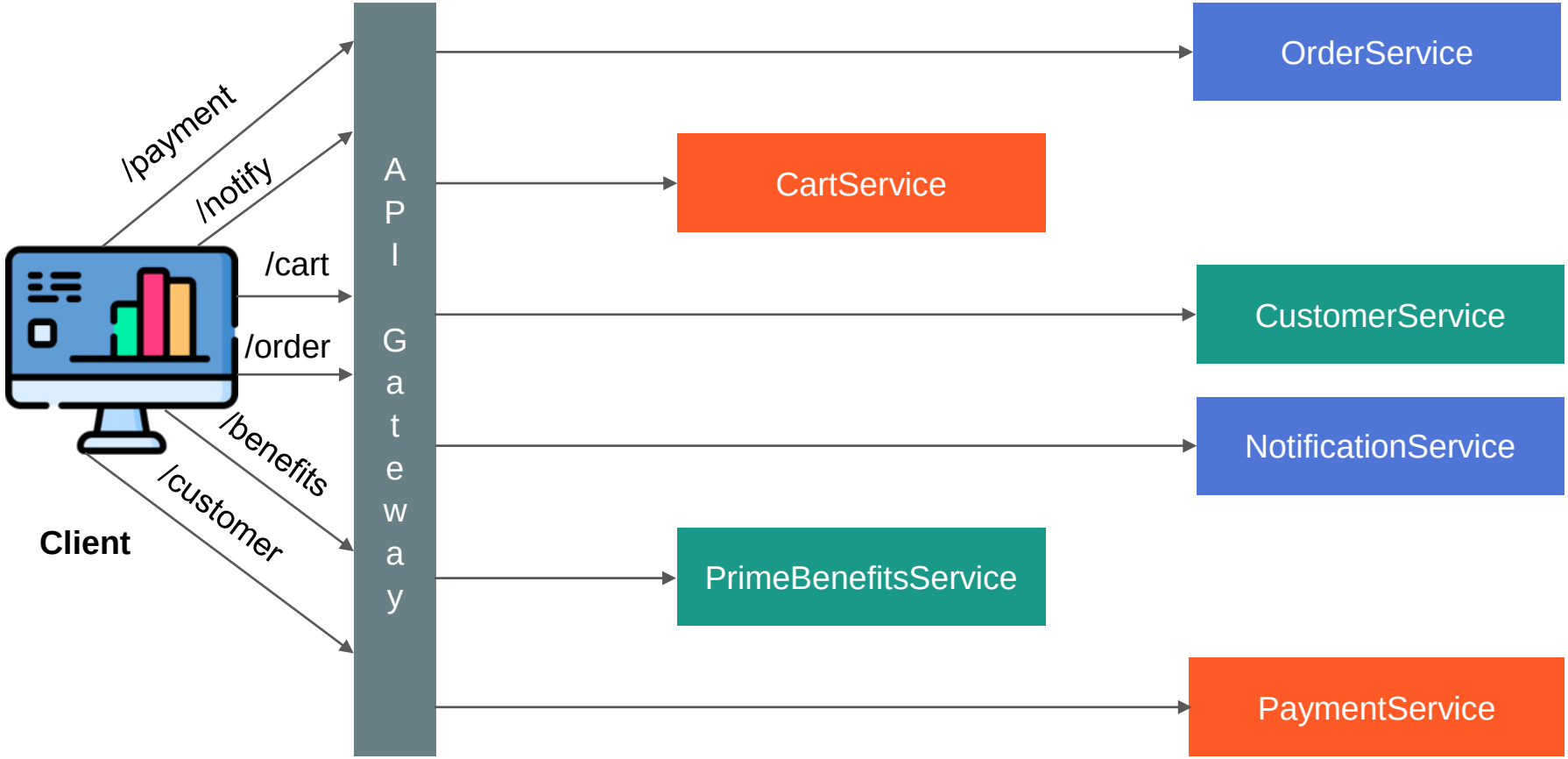
- CRS1 : Spring Framework Foundation
- CRS 2 : Creating Restful Services Using SpringBoot
- CRS 3 : Creating and Managing Microservices
- CRS 4 : Establishing Communication Among Microservices

Perform specific tasks

- Work with Virtual Machine
- Enable Continuous Integration within a Backend Application to Support a Collaborative Development Environment
- Containerize RESTful services and Database by Using Docker
- Containerize RESTful Services and Database by using Docker Compose
- Create Microservices by Using Spring Boot
- Create a Single-Entry Point to Route the Request Coming for Different Microservices Using Spring Cloud
- Register Microservices on a Netflix Eureka Discovery Server
- Handle the Failure of Microservices by Implementing the Circuit Breaker Pattern Using Hystrix

© NIIT · StackRoute

Application Workflow – Multiple Services



© NIIT · StackRoute

Think and Tell

In an application with multiple microservices, routing happens through an API gateway.

- 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 still route the request to a service that is down?

© NIIT • StackRoute

Register Microservices on a Netflix Eureka Discovery Server

© NIIT • StackRoute

Learning Objectives

- Define the service discovery design pattern
- Implement the service discovery server using Eureka
- Register the services on the Eureka server

© NIIT • StackRoute

Microservices Design Patterns - Service Discovery Design Pattern

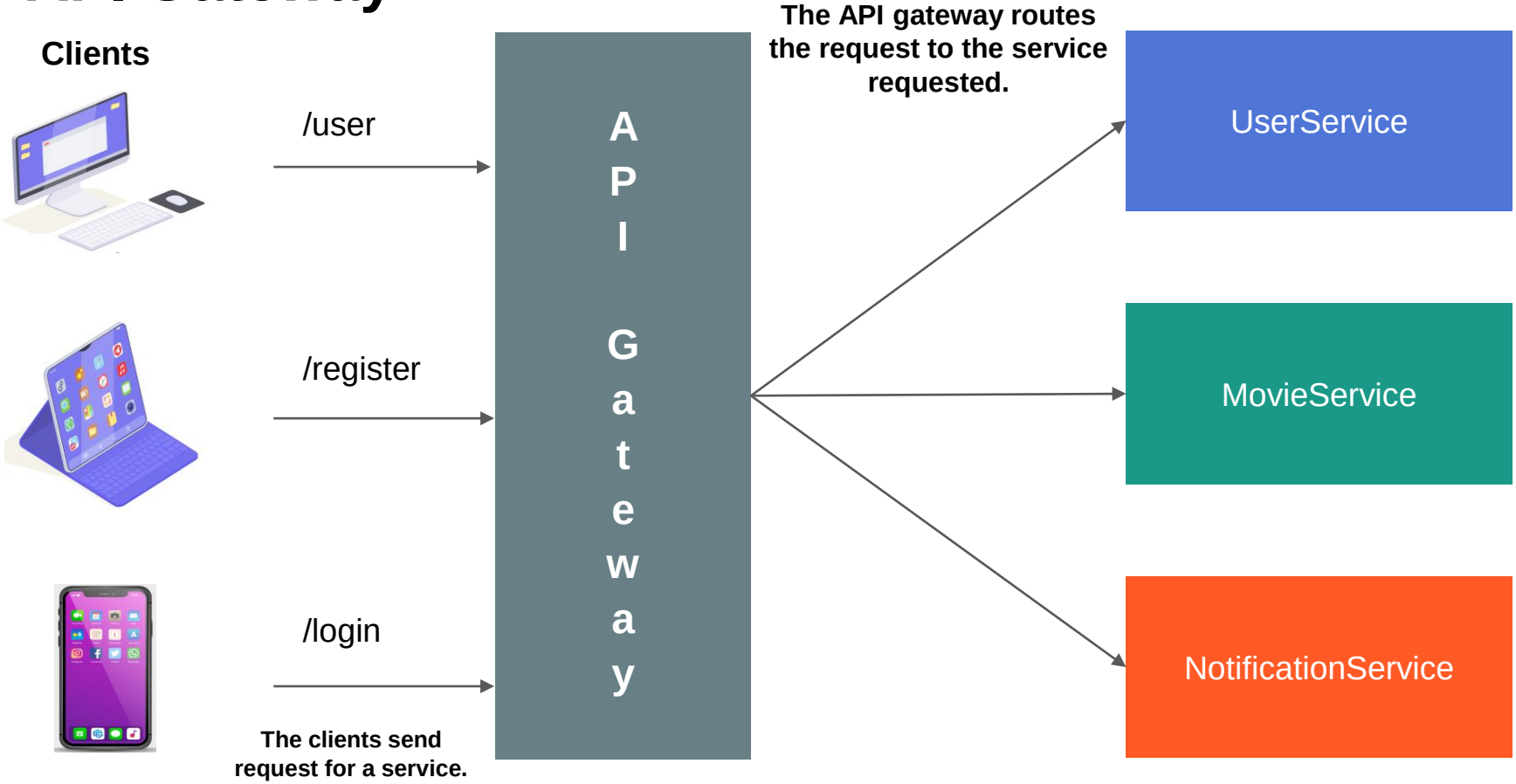
© NIIT • StackRoute

Service Discovery

- Services typically need to call one another for effective working of an application.
- It is how microservices locate each other on a network.
- Multiple instances of the same microservice can be executed at any given point in time.
- Service discovery makes it easy for clients to be serviced depending on the availability of a service.
- Service discovery is the first step towards granular scaling.
- Implementation includes a server that maintains a list of services registered on it.
- Clients connect to the server to update and retrieve the service addressed.

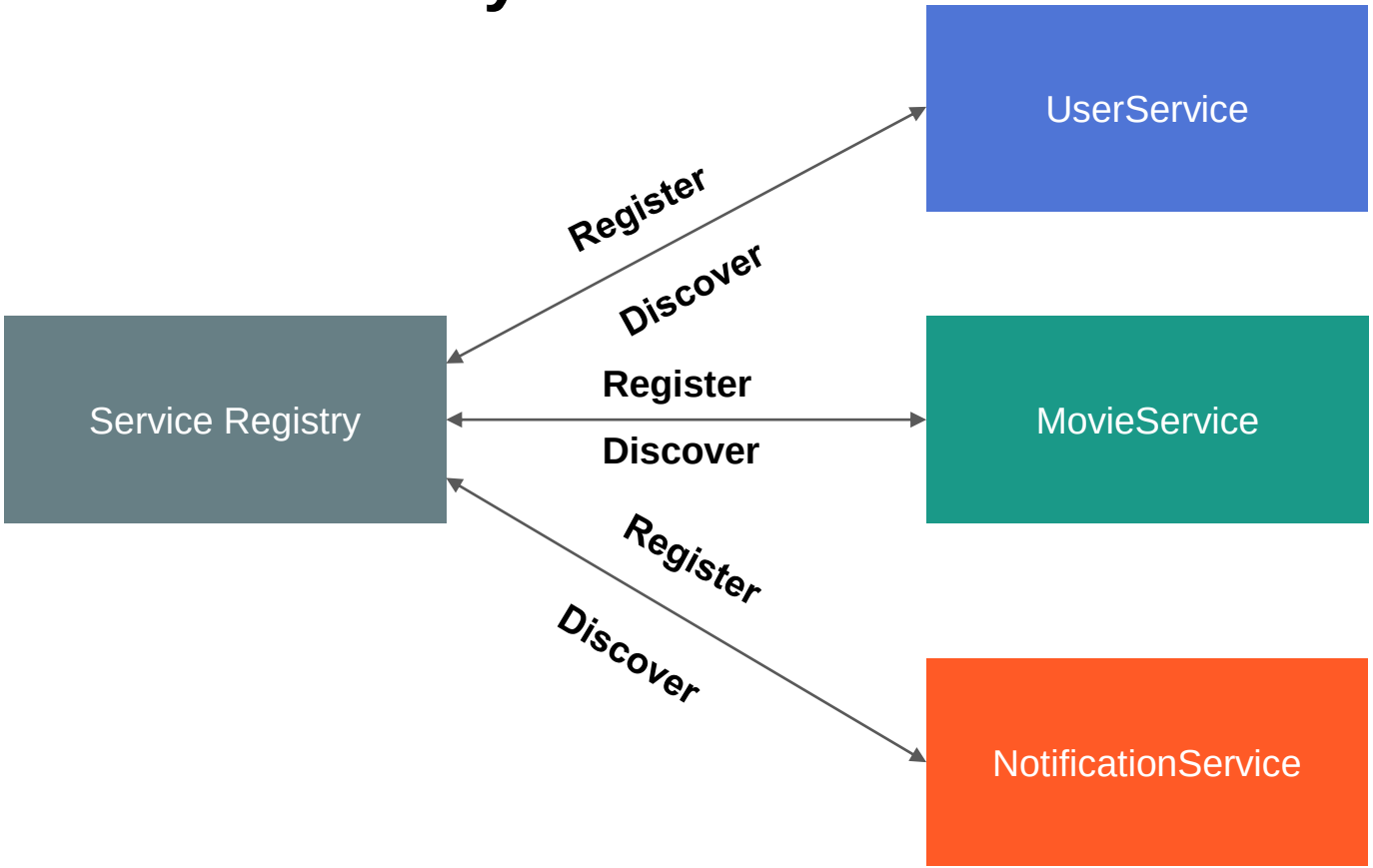
© NIIT · StackRoute

API Gateway



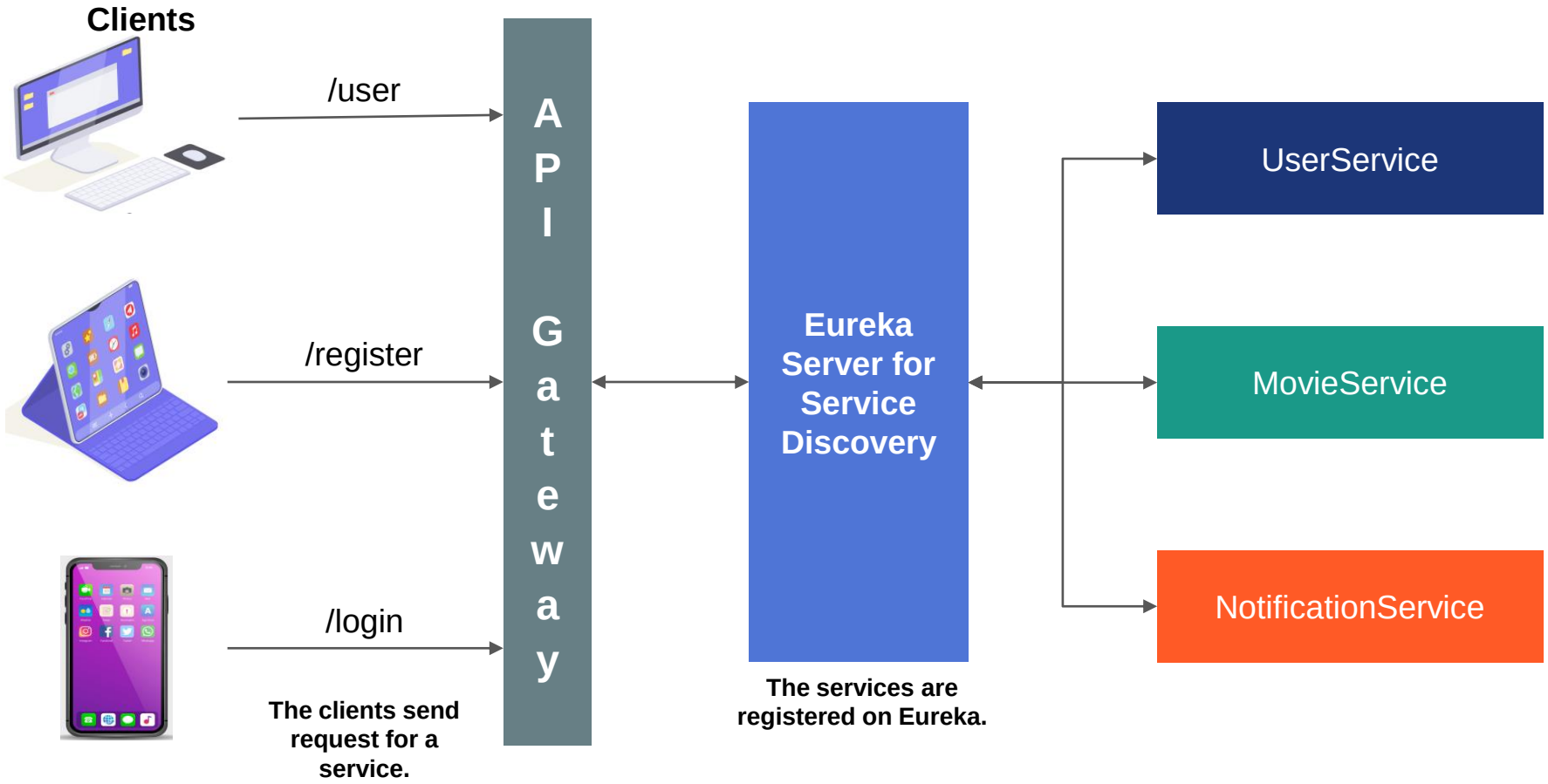
© NIIT · StackRoute

Service Discovery



© NIIT · StackRoute

Service Discovery and API Gateway



© NIIT · StackRoute

Service Discovery Design Pattern – How Does This Work?

- Services are registered to the discovery server.
- Service discovery server listens for the registered service when they start up.
- The service discovery server sends a heartbeat continuously to check for services.
- There is a timeout period to assume 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 it is up.
- The service discovery pattern is called non-invasive as it does not alter the code in any of the microservices.

© NIIT • StackRoute

Quick Check

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

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

Quick Check: Solution

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

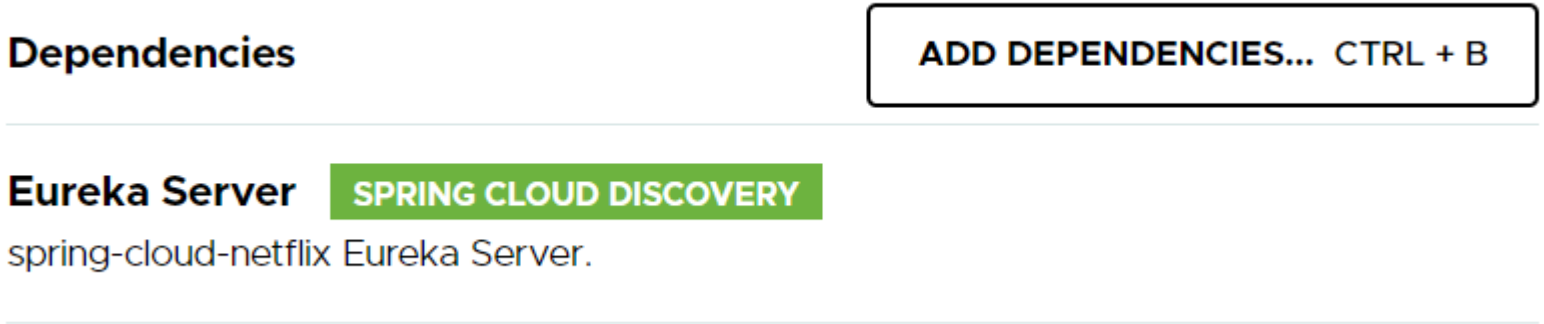
1. Heartbeat
2. Pulse
3. **Time out**
4. Discovery

Implementing Service Discovery

© NIIT • StackRoute

Create the Discovery Server

- Spring Cloud Netflix provides Netflix OSS integrations for Spring Boot apps through autoconfiguration. With a few simple annotations you can quickly enable and configure the common patterns inside the application.
- The patterns provided include Service Discovery (Eureka), Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client-Side Load Balancing (Ribbon).
- As the first step, create a Eureka Discovery Server. Use Spring initializr and bootstrap the dependencies.



Enable the Server in the Application

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

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

application.yml

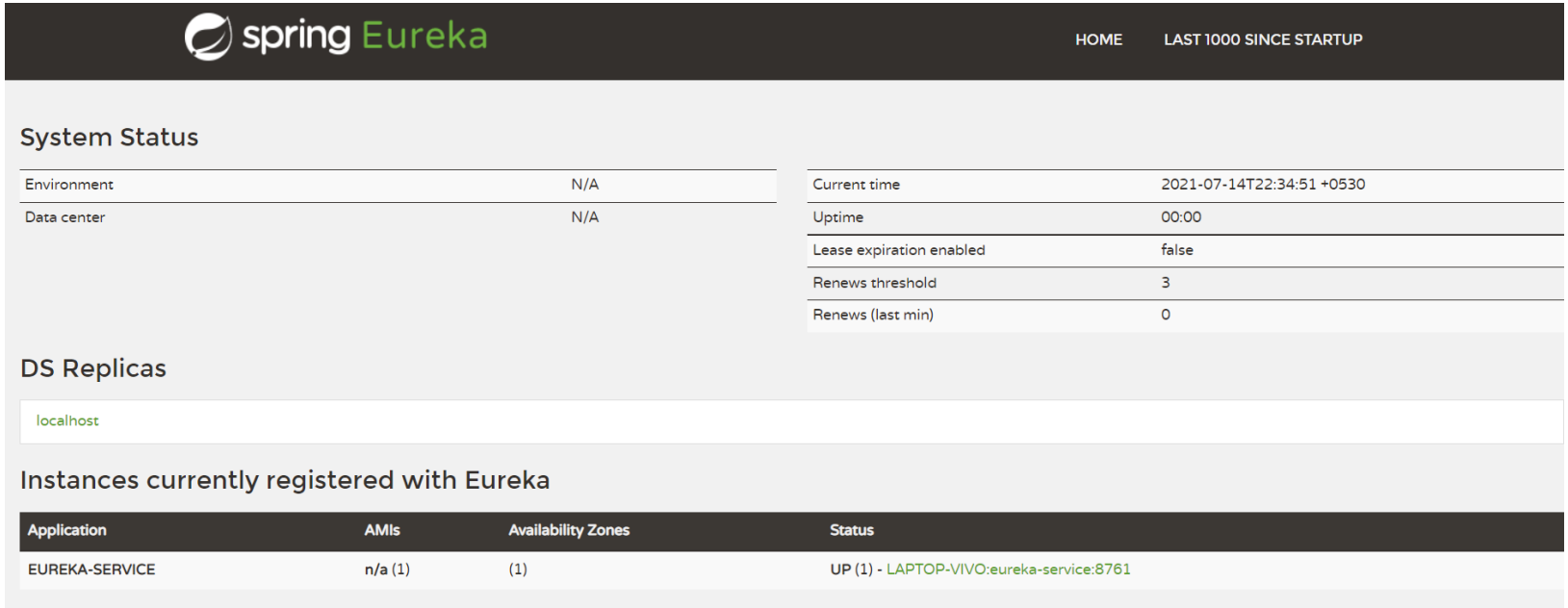
```yaml
spring:
  application:
    name: eureka-service
server:
  port: 8761
```

- Use the `@EnableEurekaServer` annotation in the main class.
- Mention the service name and the server port where the server will run in the `application.properties` or `application.yml` file.

© NIIT • StackRoute

Eureka Server

- Start the server and access the service running at http://localhost:8761/eureka.



© NIIT · StackRoute

Register the Services on the Eureka Server

- To register a microservice that is also called a Eureka client, on the Eureka server, follow the given steps.
- Step 1 : Add the below dependencies in the pom.xml of the service.

```xml
<properties>
    <java.version>11</java.version>
    <spring-cloud.version>2020.0.3</spring-cloud.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
```

```xml
<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>
```

© NIIT · StackRoute

Register the Services on the Eureka Server (contd.)

- Step 2 : Enable the microservice with @EnableEurekaClient annotation.

```java
@SpringBootApplication
@EnableEurekaClient
public class UserAuthenticationServiceApplication {

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

}
```

- Step 3 : Modify the application.yml file.

```yaml
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka
    fetchRegistry: true
    registerWithEureka: true
```

- Step 4 : Run the individual services.

Eureka Server With the Services Registered

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

spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	N/A
Data center	N/A

Current time	2021-07-14T22:41:10 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	6
Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:eureka-service:8761
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:user-authentication-service:8085
USER-MOVIE-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:user-movie-service:8081

© NIIT · StackRoute

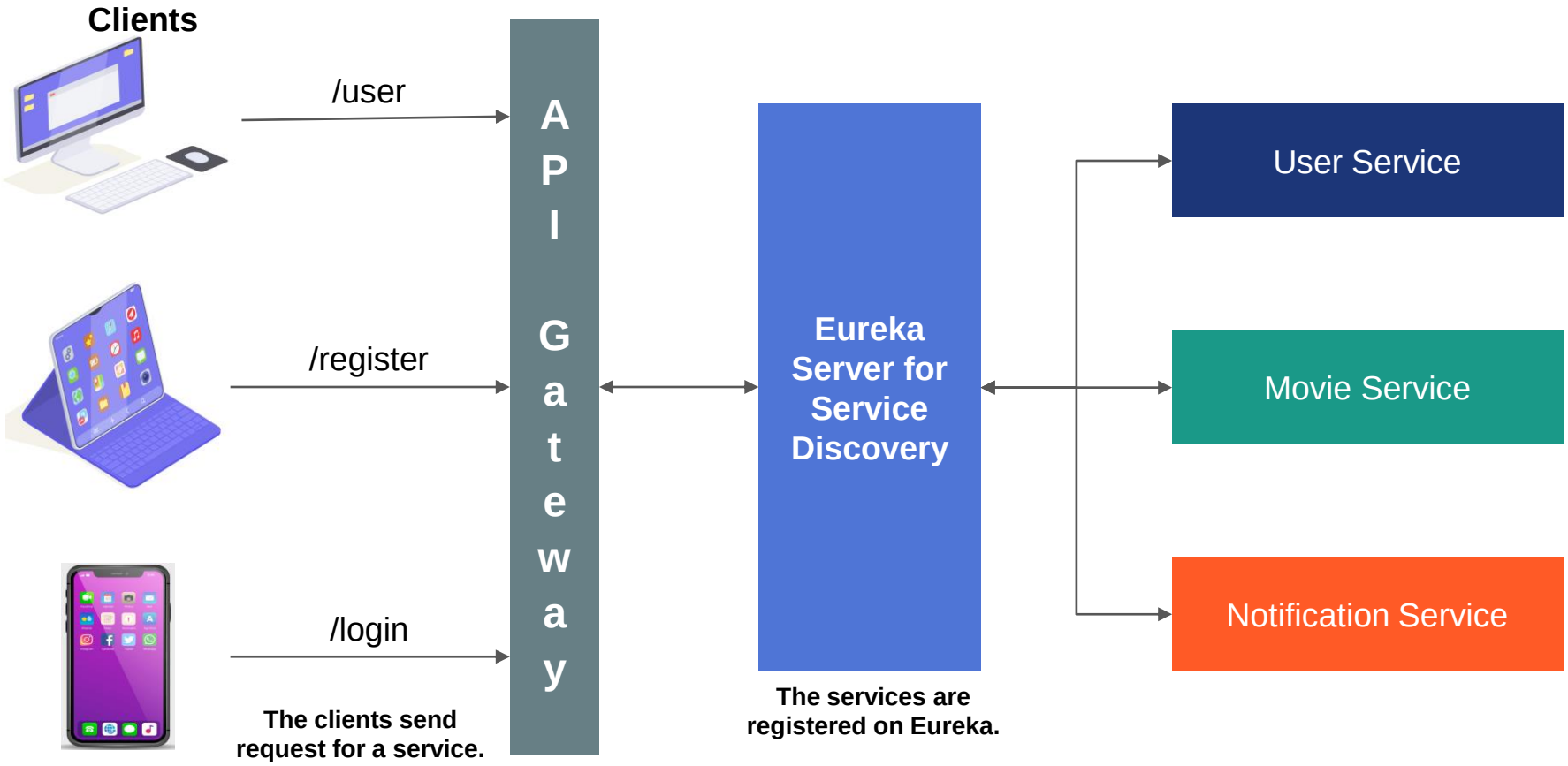
Register the API Gateway Onto the Eureka Server

- The Spring Cloud API Gateway must be registered on the Eureka server. As a client, we need to add these dependencies.
- As shown in the image, the route can also be written using the application name we configured in the application.yml file, instead of the uri of the application.

```
@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();
    }
}
```

© NIIT · StackRoute

Service Discovery and API Gateway



© NIIT · StackRoute

Services Registered on the Eureka Server

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:eureka-service:8761
SPRING-CLOUD-API-GATEWAY	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:spring-cloud-api-gateway:9000
USER-AUTHENTICATION-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:user-authentication-service:8085
USER-MOVIE-SERVICE	n/a (1)	(1)	UP (1) - LAPTOP-VIVO:user-movie-service:8081

Postman Output – Register a New User

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

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

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
1 {
2     "email":"tina.thomas@gmail.com",
3     "userName":"Tina Thomas",
4     "password":"1234"
5
6 }
```

Body | Cookies | Headers (3) | Test Results | Status: 201 Created

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2     "email": "tina.thomas@gmail.com",
3     "userName": "Tina Thomas",
4     "password": "1234",
5     "phoneNumber": null,
6     "movieList": null
7 }
```

© NIIT · StackRoute

Postman Output – Save User Credentials

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

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "email":"tina.thomas@gmail.com",
    "password":"1234"
}
```

Body | Cookies | Headers (3) | Test Results

Status: 201 Created

Pretty | Raw | Preview | Visualize | JSON

```
{
    "email": "tina.thomas@gmail.com",
    "password": "1234"
}
```

© NIIT • StackRoute

Postman Output – Login to the Movie Service

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

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "email":"tina.thomas@gmail.com",
    "password":"1234"
    
}
```

Body | Cookies | Headers (3) | Test Results

Status: 200 OK Time: 266 ms Size: 318 B

Pretty | Raw | Preview | Visualize | JSON

```
{
    "message": "Authentication Successful",
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJTaG9wWm9uZSIsInN1YiI6InRpbmEudGhvbWFzQGdtYWlsLmNvbSIsImlhdCI6MTYyNjU0MTA3Nn0.
        4piWDcFSZF0heyQ1zj6DHuLZ0m_I6inV4MOnp9cuBco"
}
```

© NIIT · StackRoute

Postman Output – Add the Favourite Movie for a User

POST http://localhost:9000/api/v2/user/movie/tina.thomas@gmail.com

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "movieId":"M001",
    "movieName":"The Shawshank Redemption",
    "genre":"Drama",
    "leadActors":["Tim Robbins","Morgan Freeman"],
    "director":"Frank Darabont",
    "yearOfRelease": 1994,
    "rating":8

}
```

Body | Cookies | Headers (4) | Test Results — Status: 201 Created

Pretty | Raw | Preview | Visualize | JSON

```
{
    "email": "tina.thomas@gmail.com",
    "userName": "Tina Thomas",
    "password": "1234",
    "phoneNumber": null,
    "movieList": [
        {
            "movieId": "M001",
            "movieName": "The Shawshank Redemption",
            "genre": "Drama",
```

© NIIT • StackRoute

Quick Check

What is the default port of the Eureka Server ?

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

Quick Check: Solution

What is the default port of the Eureka Server ?

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

Streaming Application

Consider a streaming application that enables users to watch movies on any smart device. The application provides multiple features to all its registered users. A user needs to register with the application in order to access some of its features. Let us create multiple microservices for the streaming application.

1. A user must first register with the application.
2. Use credentials such as id, password to login.
3. Access the features provided by the streaming application, like adding favourites, compiling a watch later list, etc.

Let us 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. **Dockerize the application.**

DEMO

Key Takeaways

- Service discovery pattern
- Implement service discovery
- Spring Cloud Eureka server for Service Discovery
- Register microservices onto a Eureka server

© NIIT • StackRoute

Thank you!

© NIIT • StackRoute