Spring Cloud & MicroServices

11 November 2024 11:29 AM

Monolithic vs. Microservices Architecture

1. **Monolithic Architecture**:
   - **Definition**: A single, unified application with tightly integrated services and a single codebase.
   - **Example Structure**: Imagine an ERP project with different roles (Admin, Faculty, Student) all managed within a single application. Each role is a service under the same codebase, but they can't be easily separated.
   - **Disadvantages**:
     - **Complexity in Maintenance**: Implementing changes is challenging as the entire application must be updated or redeployed, even for small modifications.
     - **Technology Limitations**: Updating the tech stack (e.g., changing languages or frameworks) for the whole application is complex and risky.
     - **Tight Coupling**: With everything integrated, changes in one part of the application often impact others, making it difficult to scale or manage.

2. **Microservices Architecture**:
   - **Definition**: An architectural style where each core function is an independent service with its own codebase.
   - **Features**:
     - **Independent Services**: Each function (like Admin, Faculty, Student) is its own service, with individual deployments and separate codebases.
     - **Inter-Service Communication**: Services communicate through APIs, such as REST, enabling data exchange using tools like RestTemplate.
     - **Flexible Technology Stack**: Each service can have its own technology stack, optimized for its needs.
   - **Example Structure**:
     - **Admin Microservice**: Built with Spring Boot MVC (Port: 2001).
     - **Student Microservice**: Developed in Django (Port: 2002).
     - **Faculty Microservice**: Built using Node.js & Express (Port: 2003).
   - **Advantages**:
     - **Independent Deployment**: Each service can be updated or scaled independently.
     - **Scalability**: Each microservice can scale independently based on load.

Spring Cloud for Microservices

Spring Cloud is a suite of tools designed to support cloud-native, microservices-based applications.

- **Purpose**: It simplifies building, deploying, and scaling microservices applications on the JVM. It is especially beneficial in a cloud environment due to its range of supporting libraries.

Advantages:

- Allows seamless inter-service communication.
- Facilitates the development of cloud-ready applications.
- Adopts the Spring Boot model, making it compatible with Spring's tools and libraries.

Key Components in Spring Cloud

1. **Eureka** - Service Discovery:
   - **Overview**: A discovery tool for microservices, developed by Netflix. It maintains a registry of services to help them discover each other dynamically.
   - **Core Components**:
     - **Service Registry**: Services register with the Eureka server, providing metadata (e.g., location, port) to facilitate discovery.
     - **Service Discovery**: Clients (microservices) can query Eureka for instances of other services, identifying them by criteria like port number or hostname.
     - **Load Balancing**: Eureka includes client-side load balancing to distribute requests across service instances.
     - **Fault Tolerance**: Eureka removes failed instances from its registry, directing clients only to available instances.
   - **Configuration**:
     - **Eureka Server**: Runs on port 8761 by default, hosting the registry.

        - **Client Properties**:
            - □ eureka.client.register-with-eureka=false: Ensures the server doesn't register itself as a client.
            - □ eureka.client.fetch-registry=false: Prevents the server from fetching microservices from external sources, managing its own registry.
            - □ eureka.server.wait-time-in-ms-when-sync-empty=0: Specifies the wait time before serving requests, set to 0 for immediate service.
    - **Startup Order**:
        - Start the **Eureka Server** before the Eureka clients to ensure they can register with the server upon startup.

2. **Spring Cloud API Gateway**:
    - **Purpose**: A unified entry point for requests across multiple microservices, regardless of individual port numbers.
    - **Components**:
        - **Route**: Each route directs requests to a specific service or URL, defined by:
            - □ **ID**: The name of the service.
            - □ **Destination URI**: The URL or service location.
            - □ **Predicates**: Functions that check if a request meets specified criteria (e.g., path, header, or parameter).
            - □ **Filters**: Spring Gateway filters that modify requests or responses as needed.
    - **Predicates and Filters**:
        - **Predicate**: A condition checker (similar to Java's Predicate function) that tests if an HTTP request meets specific criteria.
        - **Filter**: Adjusts request or response data based on requirements.
    - **Outcome**:
        - Using the API Gateway, clients access services on a single port (e.g., 2000), simplifying access to microservices without needing their individual port numbers.
3. **Actuator**:
    - **Purpose**: Provides monitoring and management endpoints, offering health and metrics data for microservices, helping administrators keep the services functional and optimized.

REST Template:

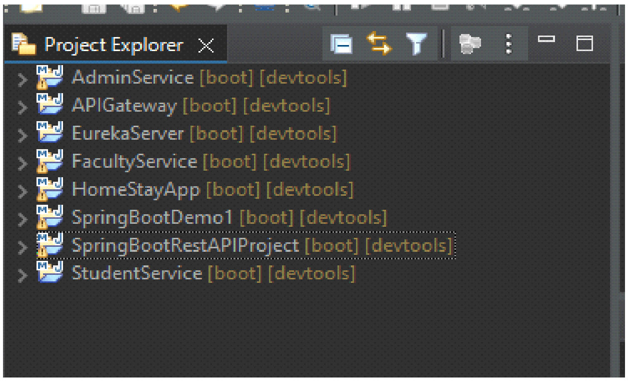
- A Spring utility that facilitates data exchange between applications, allowing a microservice to make HTTP requests to other services.
- **Examples**:
    - Fetching a list of posts from a remote API: https://jsonplaceholder.typicode.com/posts
    - Accessing a specific post: https://jsonplaceholder.typicode.com/posts/1

Dependencies Required for Each Microservice Project

1. **Admin Service**:
    - Spring Web, DevTools, MySQL Driver, Spring Data JPA, Eureka Discovery Client, Actuator.
2. **Employee Service**:
    - Spring Web, DevTools, MySQL Driver, Spring Data JPA, Eureka Discovery Client, Actuator.
3. **API Gateway**:
    - DevTools, Eureka Discovery Client, Cloud Gateway, Actuator.
4. **Eureka Server**:
    - DevTools, Eureka Server.

Sample Project to Execute the Spring Microservice Demonstration :

- **Step-1:** Create individual Spring Boot projects for each module (AdminService, StudentService, FacultyService) using Spring Initializr or STS, with unique configurations and dependencies per module.

- 

AdminService [boot] [devtools]
APIGateway [boot] [devtools]
EurekaServer [boot] [devtools]
FacultyService [boot] [devtools]
HomeStayApp [boot] [devtools]
SpringBootDemo1 [boot] [devtools]
SpringBootRestAPIProject [boot] [devtools]
StudentService [boot] [devtools]

- **Step-2:** Create individual Spring Boot projects for the APIGateway and EurekaServer using Spring Initializr or STS, with appropriate configurations for API Gateway routing and Eureka Server for service discovery.
- **Step-3:**
  Assign unique server ports to each individual project and set a common port for the APIGateway as follows:
  - **AdminService**: server.port=2001
  - **StudentService**: server.port=2003
  - **FacultyService**: server.port=2002
  - **EurekaServer**: server.port=8761
  - **APIGateway**: server.port=2000

  This setup allows the APIGateway on port 8080 to route requests to each module.
- **Step-4:** Now to test the application give the Log statement in the Application.java file in each services
- **Step-5:** Now, give the aapplication.properties in each Service projects

Now **The Application.properties** should be configured as :

Admin Service:	spring.application.name=AdminService
	server.port= 2001
	
	eureka.instance.hostname=localhost
	eureka.instance.appname=admin
	
	management.endpoint.info.enabled=true
	management.endpoints.web.exposure.include=*

APIGateway:	spring.application.name=APIGateway
	
	server.port=2000
	
	eureka.instance.hostname=localhost
	eureka.instance.appname=APIGateway
	
	# Routes configuration for different services using MVC
	spring.cloud.gateway.mvc.routes[0].id=AdminService
	spring.cloud.gateway.mvc.routes[0].uri=http://localhost:2001
	spring.cloud.gateway.mvc.routes[0].predicates[0]=Path=/admin/**
	
	spring.cloud.gateway.mvc.routes[1].id=FacultyService
	spring.cloud.gateway.mvc.routes[1].uri=http://localhost:2002
	spring.cloud.gateway.mvc.routes[1].predicates[0]=Path=/faculty/**
	
	spring.cloud.gateway.mvc.routes[2].id=Student

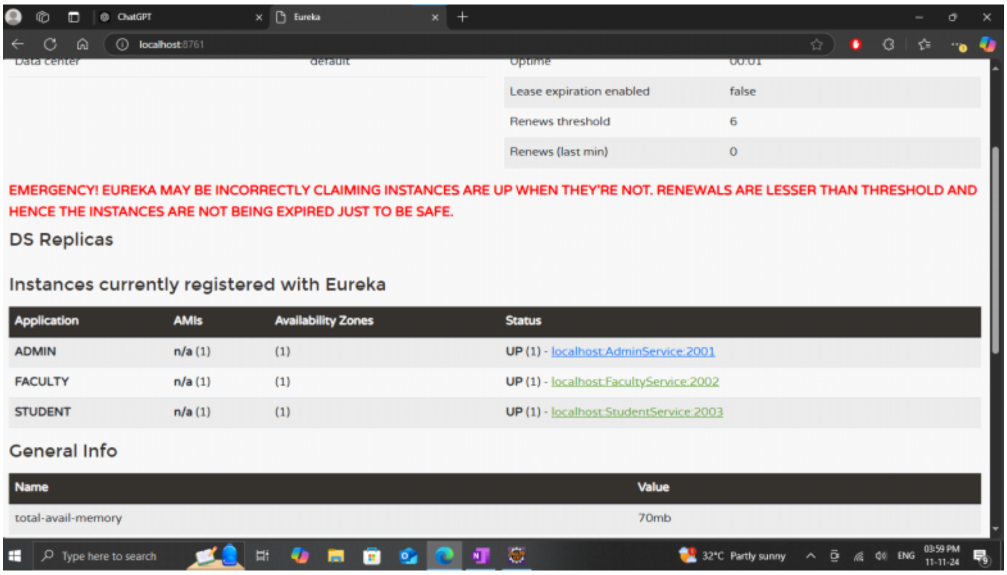
	Service
	spring.cloud.gateway.mvc.routes[2].uri=http://localhost:2003
	spring.cloud.gateway.mvc.routes[2].predicates[0]=Path=/student/**
	
	management.endpoint.info.enabled=true
	management.endpoints.web.exposure.include=*

Eureka Server:	spring.application.name=EurekaServer
	server.port=8761
	
	eureka.instance.hostname=localhost
	
	eureka.client.register-with-eureka=false
	eureka.client.fetch-registry=false
	# we set this to false since, it does not register itself in the list along ther server , other clients. BEcause it acts as server but not as a client
	
	# 2nd as false, since it till not receive the registered microservices (eureka clients) list from anywhere. It will create and maintain the list itself
	
	eureka.server.wait-time-in-ms-when-sync-empty=0
	#it sepcifies the amount of time in millisec, the eureka server should wait when its registary is empty, before it starts serving request
	# 0 means server will not wait at all and will immediately start serving the request, even if there are no registered services itself.

Student Service :	spring.application.name=StudentService
	server.port=2003
	
	eureka.instance.hostname=localhost
	eureka.instance.appname= Student
	
	management.endpoint.info.enabled=true
	management.endpoints.web.exposure.include=*

FacultyService	spring.application.name=FacultyService
	server.port=2002
	
	eureka.instance.hostname=localhost
	eureka.instance.appname=Faculty
	
	management.endpoint.info.enabled=true
	management.endpoints.web.exposure.include=*

- **Step-6:** Now since, we have configured all the application.property,
  Open the Boot Dashboard then make sure we need to run the Eurekaserver first then later we run remaining Services.
- **Step-7:** In http://localhost:8761 we can check all the services that are listed below



Data center | default | Uptime | 00:01

Lease expiration enabled	false
Renews threshold	6
Renews (last min)	0

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

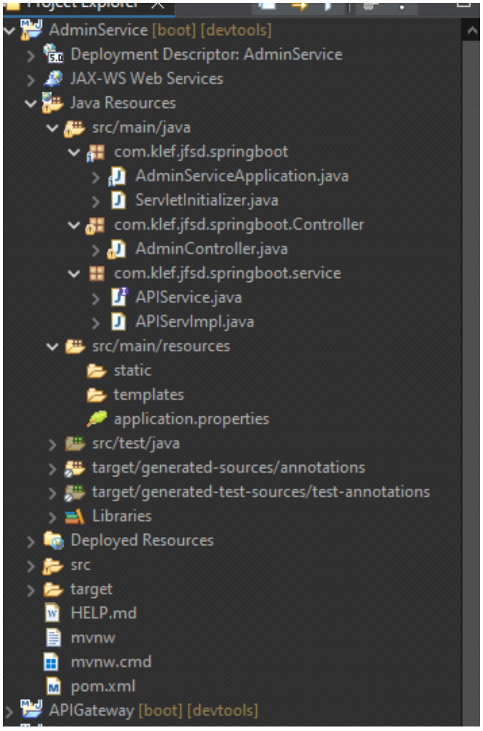
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ADMIN	n/a (1)	(1)	UP (1) - localhost:AdminService:2001
FACULTY	n/a (1)	(1)	UP (1) - localhost:FacultyService:2002
STUDENT	n/a (1)	(1)	UP (1) - localhost:StudentService:2003

General Info

Name	Value
total-avail-memory	70mb

- **Step-8:** Now to randomly test the API Gate way, we do focus on the AdminService, to create the Controller, Service etc..



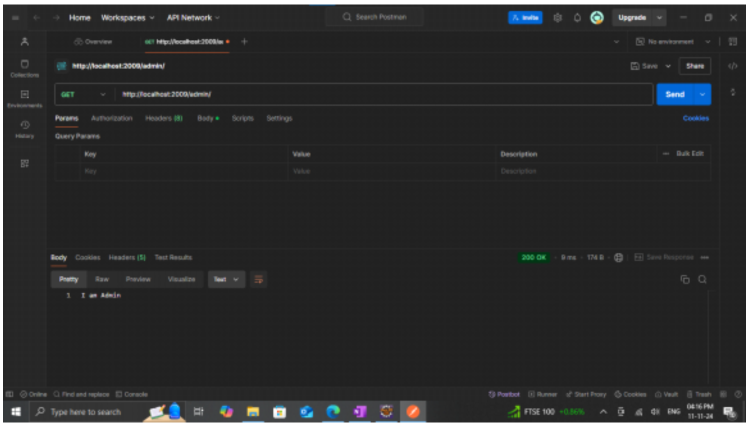
```
Project Explorer
AdminService [boot] [devtools]
  Deployment Descriptor: AdminService
  JAX-WS Web Services
  Java Resources
    src/main/java
      com.klef.jfsd.springboot
        AdminServiceApplication.java
        ServletInitializer.java
      com.klef.jfsd.springboot.Controller
        AdminController.java
      com.klef.jfsd.springboot.service
        APIService.java
        APIServImpl.java
    src/main/resources
      static
      templates
      application.properties
    src/test/java
    target/generated-sources/annotations
    target/generated-test-sources/test-annotations
    Libraries
  Deployed Resources
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
APIGateway [boot] [devtools]
```

- **Step-9:** Now to implement the rest template

REST Template:

- A Spring utility that facilitates data exchange between applications, allowing a microservice to make HTTP requests to other services.
- **Examples**:
  - Fetching a list of posts from a remote API: https://jsonplaceholder.typicode.com/posts
  - Accessing a specific post: https://jsonplaceholder.typicode.com/posts/1

- **Step-10:** Start the APIGateway server and Use the JSON placeholder to fetch the data from the API, and we can check it in the postman

- 

http://localhost:2009/admin/

GET http://localhost:2009/admin/

200 OK · 9 ms · 174 B

```
I am Admin
```

Postman :

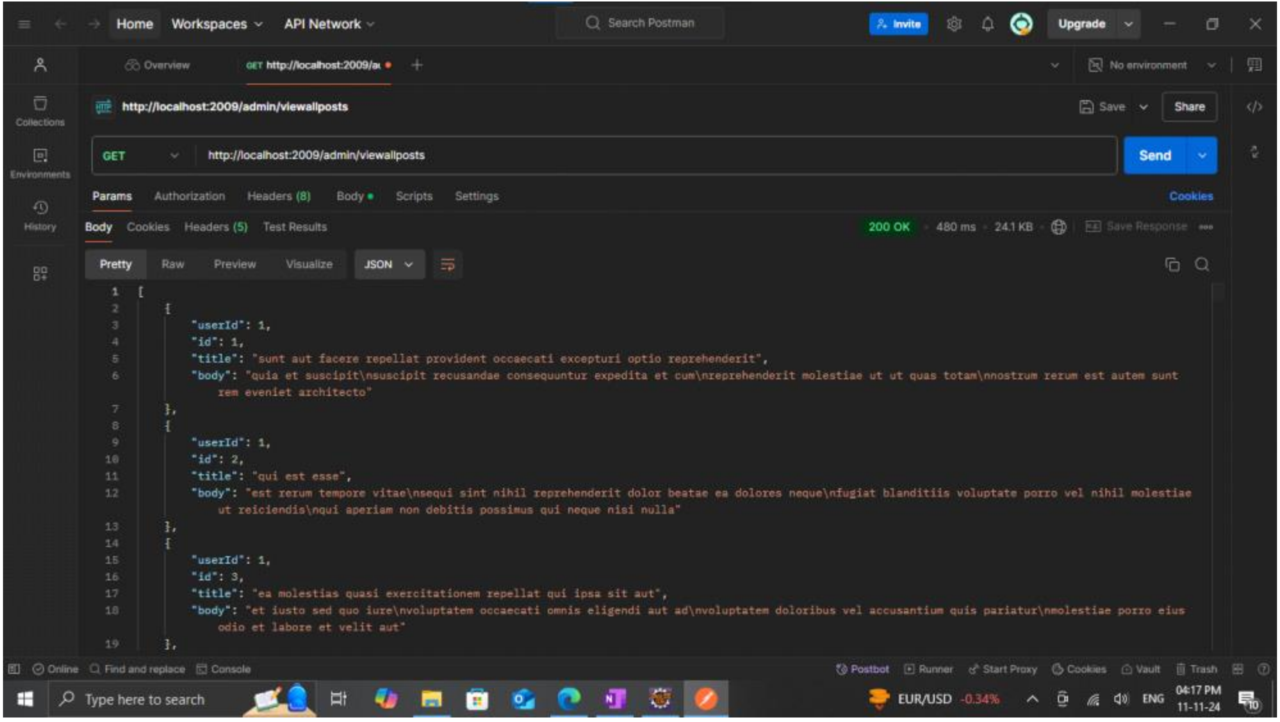
AdminController

```java
package com.klef.jfsd.springboot.Controller;

import org.springframework.web.bind.annotation.RestController;

import com.klef.jfsd.springboot.service.APIService;
import com.netflix.discovery.converters.Auto;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;


@RestController
@RequestMapping("admin")
public class AdminController
{
	@Autowired
	private APIService apiservice;
	@GetMapping("/")
	public String adminHome() {
		return "I am Admin";
	}
	
	@GetMapping("viewallposts")
	public List<Object> viewallposts()
	{
		return apiservice.displayAllPosts();
	}
```

	@GetMapping("displaypost")
	public Object displaypost(int id)
	{
	return
	apiservice.displayPostById(id);
	}
	
	}

AdminServiceApplication.java	package com.klef.jfsd.springboot;
	
	import org.springframework.boot.SpringApplication;
	import
	org.springframework.boot.autoconfigure.SpringBootApplication;
	import org.springframework.context.annotation.Bean;
	import org.springframework.web.client.RestTemplate;
	
	@SpringBootApplication
	public class AdminServiceApplication {
	
	public static void main(String[] args) {
	SpringApplication.run(AdminServiceApplication.class,
	args);
	System.out.println("Admin service is running");
	}
	
	@Bean
	public RestTemplate restTemplate()
	{
	return new RestTemplate();
	}
	
	}

Output: Viewallposts



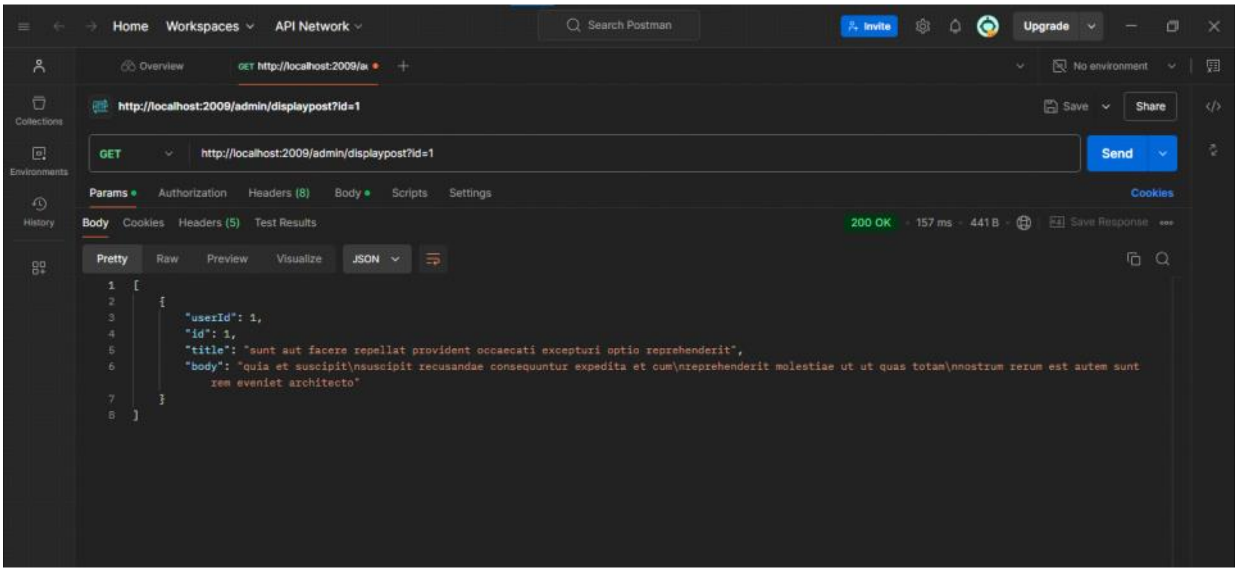
http://localhost:2009/admin/viewallposts

GET http://localhost:2009/admin/viewallposts

200 OK · 480 ms · 24.1 KB

```
[
    {
        "userId": 1,
        "id": 1,
        "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
        "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt
            rem eveniet architecto"
    },
    {
        "userId": 1,
        "id": 2,
        "title": "qui est esse",
        "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae
            ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
    },
    {
        "userId": 1,
        "id": 3,
        "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
        "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius
            odio et labore et velit aut"
    },
```

DisplayPost by id:



http://localhost:2009/admin/displaypost?id=1

GET http://localhost:2009/admin/displaypost?id=1

Send

200 OK · 157 ms · 441 B

```
[
    {
        "userId": 1,
        "id": 1,
        "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
        "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt
            rem eveniet architecto"
    }
]
```

-LikithKandepu