

LivePlay

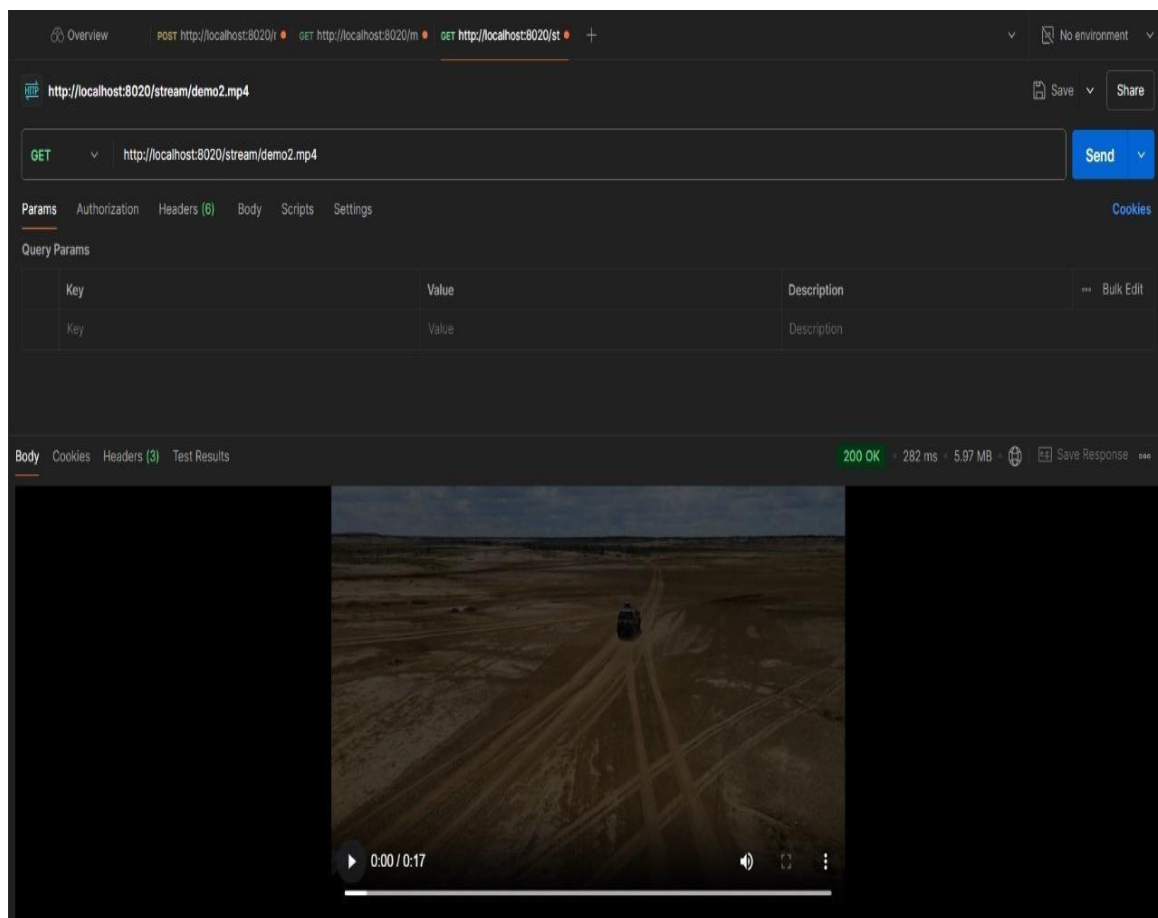
1. Introduction

- **Overview:** This project is a microservice-based video streaming platform developed using Spring Boot and PostgreSQL. It consists of various services that manage video streaming, movie data, and related operations, utilizing components like API Gateway, Eureka, a Configuration Server, and Zipkin.
- **Objective:** To efficiently manage and stream videos while ensuring scalability, fault tolerance, and ease of maintenance in a microservices environment.

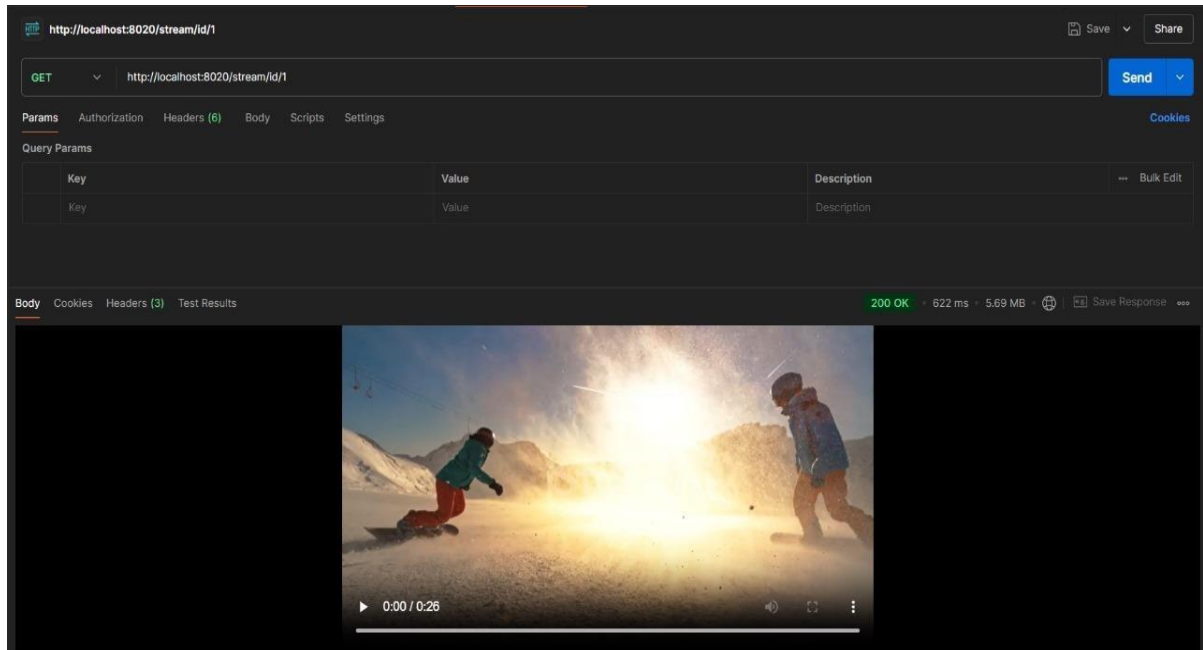
2. Project Architecture

2.1 Microservices

- **Video Service:** Manages video streaming functionalities and metadata.
 - **APIs:**
 - `@GetMapping("/stream/{videoPath}")` : Streams a video by its file path.



- `@GetMapping("/stream/id/{videoID}")`: Streams a video using its unique ID.

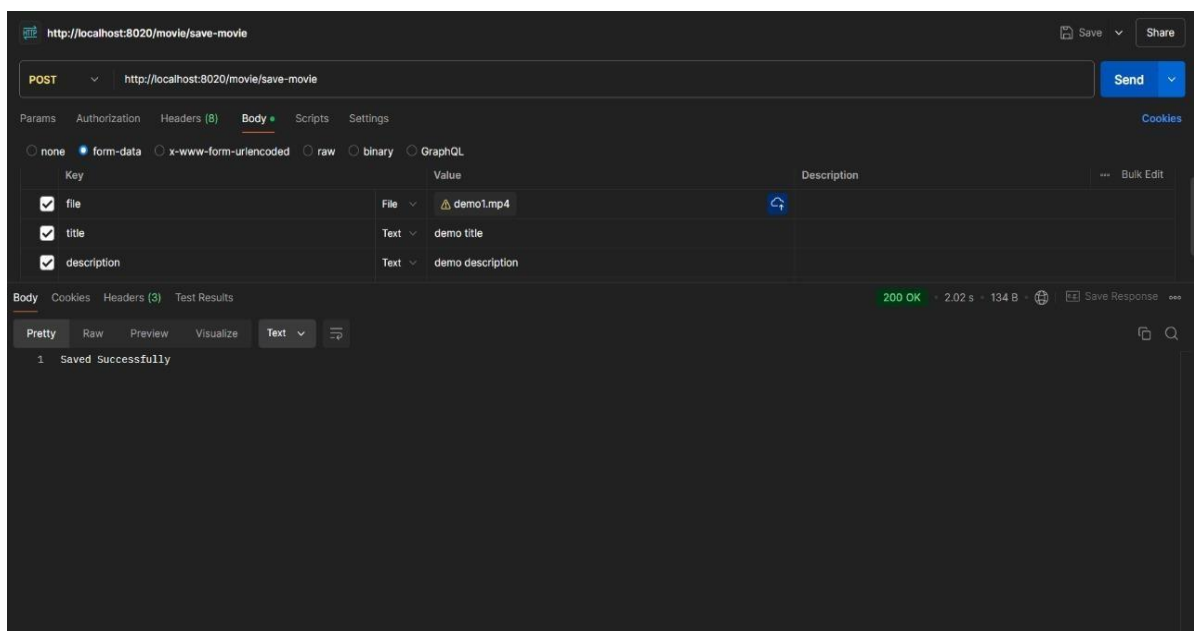


- `@GetMapping("/movie/get-path/{movieID}")`: Retrieves the video path by movie ID for **Interprocess communication**.

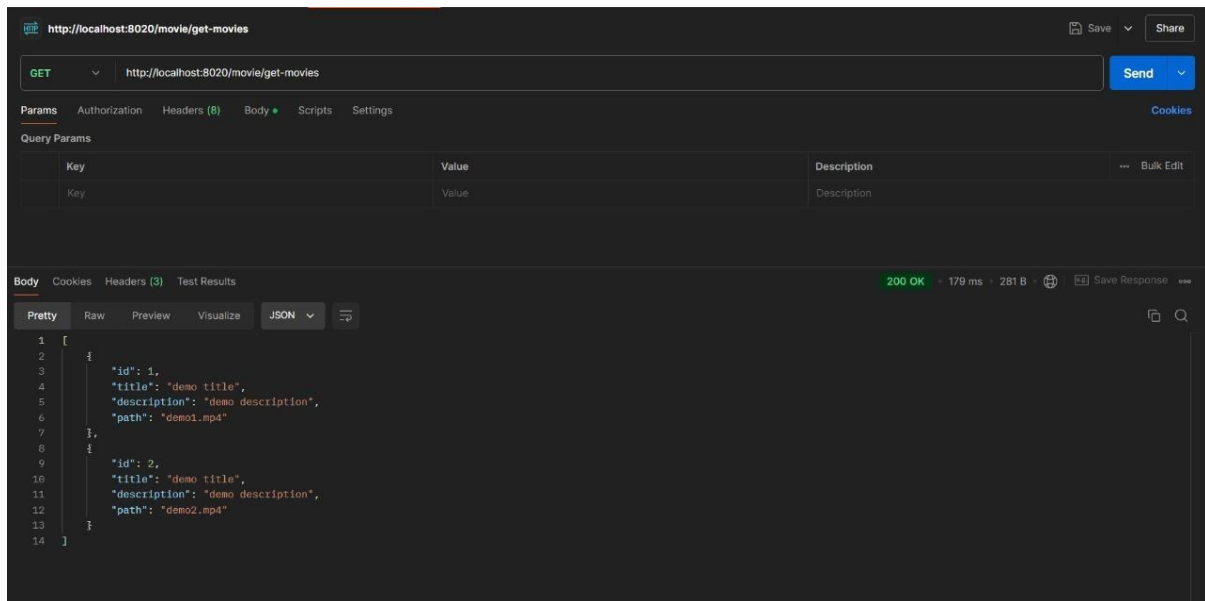
- **Movie Service:** Manages movie-related data.

- **APIs:**

- `@PostMapping("/movie/save-movie")`: Saves movie details.

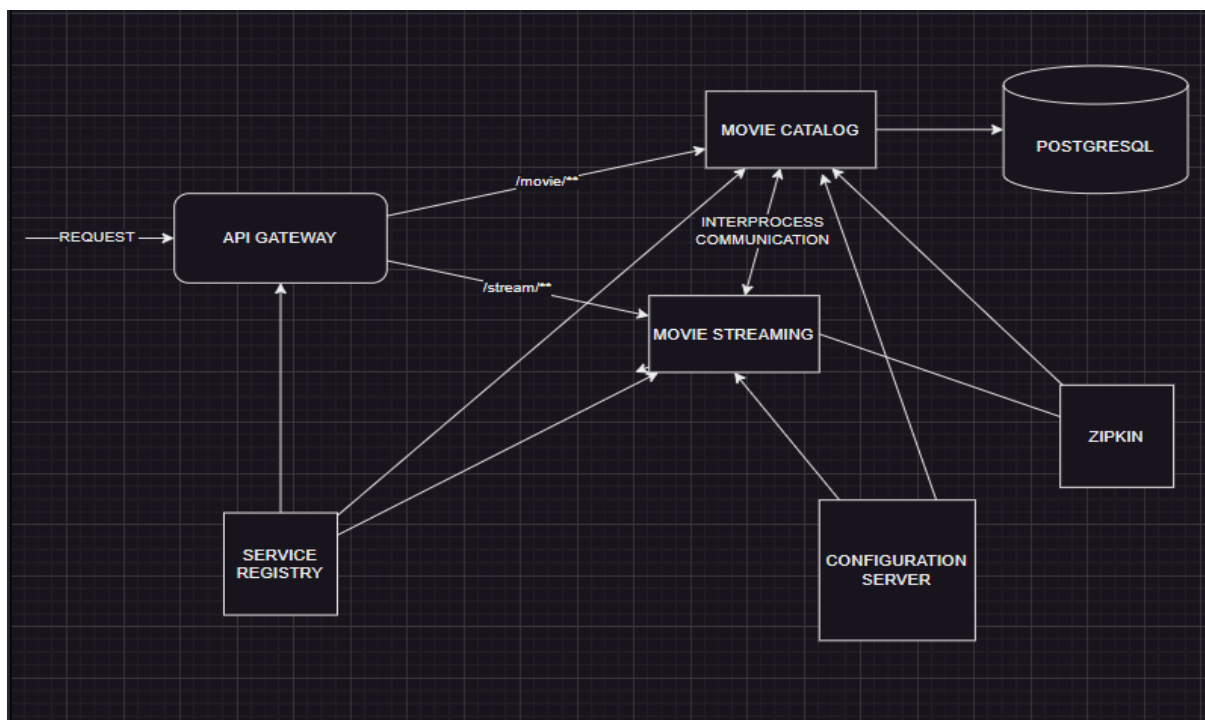


- `@GetMapping("/movie/get-movies")` : Fetches all available movies.



- **API Gateway:** Acts as a central entry point for clients, routing requests to the appropriate services.
- **Eureka Service Discovery:** Facilitates dynamic registration and discovery of services, allowing them to locate one another without hardcoded URLs.
- **Configuration Server:** Centralizes management of configurations across services.

3. Implementation Details



3.1 API Gateway

- **Routing Configuration:** Routes incoming requests to appropriate microservices, ensuring efficient traffic management.

3.2 Eureka Service Discovery

- **Server Configuration:** The Eureka server enables other services to register and discover each other dynamically.
- **Client Configuration:** Each microservice registers with the Eureka server for easy discovery and communication.

The screenshot shows the Eureka web interface in a browser window. The address bar shows 'localhost:8761'. The interface has a dark theme. At the top, it shows 'Data center: default'. On the right, there's a summary of server status: Uptime: 00:06, Lease expiration enabled: false, Renewals threshold: 8, and Renewals (last min): 8. Below this, a red warning message states: '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.' Underneath the warning, it says 'DS Replicas'. Then, it says 'Instances currently registered with Eureka'. Below this is a table with columns: Application, AMIs, Availability Zones, and Status. The table lists four applications: API-GATEWAY, CONFIG-SERVER, MOVIE-CATALOG, and MOVIE-STREAMING-SERVICE. Each application has a status of 'UP (1)' and a link to its details. At the bottom, there's a 'General Info' section with a table showing 'Name' and 'Value' for 'total-avail-memory' (88mb) and 'num-of-cpus' (12).

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - JALALUDEENZUBAIR03.mshome.net:api-gateway:8020
CONFIG-SERVER	n/a (1)	(1)	UP (1) - JALALUDEENZUBAIR03.mshome.net:config-server:8888
MOVIE-CATALOG	n/a (1)	(1)	UP (1) - JALALUDEENZUBAIR03.mshome.net:movie-catalog:8090
MOVIE-STREAMING-SERVICE	n/a (1)	(1)	UP (1) - JALALUDEENZUBAIR03.mshome.net:movie-streaming-service:8989

Name	Value
total-avail-memory	88mb
num-of-cpus	12

3.3 Configuration Server

- **Server Setup:** The configuration server serves centralized configurations for all microservices.
- **Client Setup:** Each microservice fetches its configuration from the configuration server during startup.

3.4 Interprocess Communication

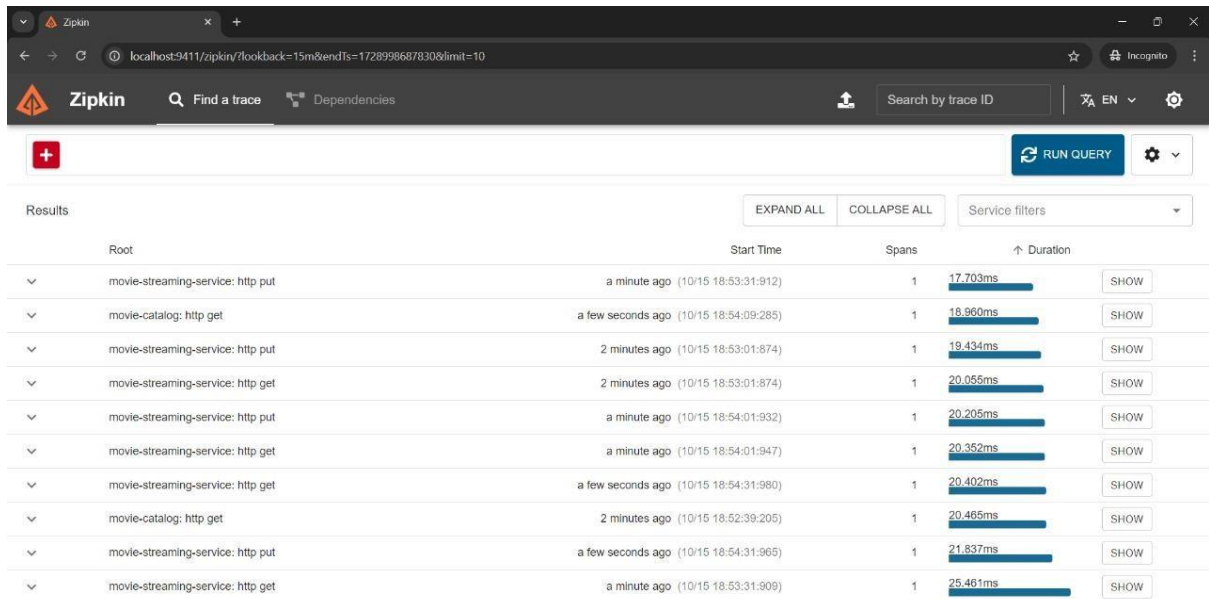
- **Feign Clients:** Utilize Feign Clients for declarative REST communication between services. The API `@GetMapping("/movie/get-path/{movieID}")` is a key example, where the Video Service calls the Movie Service to retrieve the path of a video associated with a specific movie ID.

3.5 Circuit Breaker

- **Resilience Implementation:** Implement Resilience4j or Hystrix to handle service failures gracefully, ensuring the system remains robust during downtime.

3.6 Distributed Tracing with Zipkin

- **Zipkin Integration:** Integrate Zipkin for distributed tracing, allowing you to monitor and visualize request flows across microservices. This helps in identifying latency issues and understanding service interactions.
 - Ensure that each service sends trace data to the Zipkin server to facilitate this monitoring.



4. Key Components

- **API Gateway:** Routes requests to microservices.
- **Eureka:** Enables service discovery.
- **Configuration Server:** Centralizes configuration management.
- **Zipkin:** Provides distributed tracing capabilities for performance monitoring.

5. Conclusion

This documentation outlines the setup of a video streaming platform using Spring Boot with a microservice architecture, focusing on key components such as API Gateway, Eureka, Configuration Server, and Zipkin for distributed tracing. The platform is designed to be scalable and resilient, ensuring a smooth user experience.