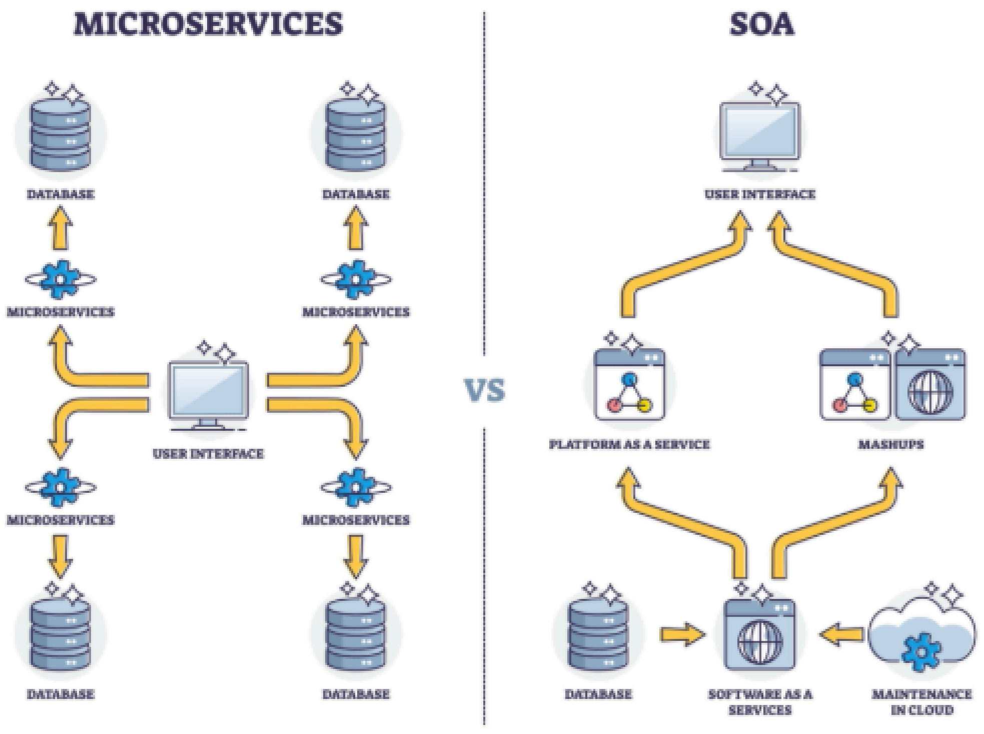
1. Technical Report

Subject: Implementation of Microservices Architecture on AWS
Date: January 3, 2026
Submitted To: Mr. Zunnurain

1.1. Executive Summary

This technical report presents the design, implementation, and cloud deployment of a **Microservices-based application** developed by **Team Shahzaman, Saram, and Rai**. The primary objective of the project was to transition from a traditional monolithic architecture to a distributed microservices architecture using **Spring Boot and Spring Cloud technologies**.

The system was successfully deployed on **Amazon Web Services (AWS)**, demonstrating scalability, modularity, and cloud readiness. By leveraging service discovery and centralized request routing, the application achieves loose coupling between services, enabling independent development, deployment, and scaling of system components.



1.2. System Architecture

The system architecture follows standard microservices design principles and consists of the following core components:

Discovery Service (Eureka Server)

- Acts as a centralized **service registry** for the entire system.
- All microservices automatically register themselves upon startup.
- Enables dynamic service discovery, eliminating the need for hard-coded IP addresses or hostnames.
- Facilitates seamless communication between services.

API Gateway

- Serves as the **single entry point** for all client and frontend requests.
- The frontend interacts exclusively with the API Gateway rather than individual services.
- Responsible for routing incoming requests to the appropriate microservice.
- Provides basic load balancing and request abstraction.

Functional Microservices

- **Service A (e.g., Product / Movie Information Service):**
  Handles static or reference data such as product details or informational content.
- **Service B (e.g., Ratings / Orders Service):**
  Manages transactional or dynamic data such as user ratings or order records.
- **Implementation Details:**
  Each microservice is implemented as an independent Spring Boot application with its own:
  - Controller layer
  - Service layer
  - Repository layer

This separation ensures loose coupling and high cohesion within each service.

Frontend Module

- Implemented to provide a user-friendly interface for interacting with the system.
- Communicates with backend services asynchronously through RESTful APIs exposed via the API Gateway.
- Completes the end-to-end flow of the application by connecting users to backend functionality.

1.3. Deployment Strategy (AWS)

The application was deployed using **Amazon EC2 Infrastructure as a Service (IaaS)**.

- **Instance Configuration:**
  AWS EC2 instances running **Amazon Linux 2 AMI / Ubuntu Server**.
- **Environment Setup:**
  Java Development Kit (JDK) was installed and configured on each instance.
- **Build Process:**
  Executable JAR files were generated using Maven with the command:
  ```
  mvn clean install
  ```
- **Execution:**
  The applications were executed on the server using:
  ```
  java -jar application.jar
  ```

This deployment approach ensures portability and allows services to be independently deployed and managed.

1.4. Challenges & Solutions

Challenge 1: Service-to-Service Communication Failures

- **Issue:**
  Initial communication failures occurred due to hard-coded IP addresses.

- **Solution:**
  Implemented **Spring WebClient / RestTemplate** configured to use **Eureka service names**, enabling dynamic service resolution and improved reliability.

Challenge 2: AWS Port Restrictions

- **Issue:**
  Microservices were inaccessible due to blocked network ports.

- **Solution:**
  AWS **Security Groups** were configured to allow inbound **Custom TCP traffic** on required ports such as:

  - 8761 (Eureka Server)

  - 8081, 8082 (Microservices)

This ensured proper communication between services and external access where required.

1.5. Conclusion

The project successfully demonstrates the implementation of a **decoupled and scalable microservices architecture** using Spring Boot and Spring Cloud technologies. The integration of a frontend module completes the full-stack workflow, while deployment on AWS validates the system’s cloud readiness.

Overall, the project highlights the advantages of microservices in terms of maintainability, scalability, and fault isolation, making it a robust foundation for modern cloud-based applications.