School of Computer Science

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

DEHRADUN, UTTARAKHAND

Container Orchestration and Infrastructure Management

ContactHub: Contacts Management System

Project Report

(2023-2024)

for
5th Semester

SUBMITTED TO:
Ms.Avita Katal
Assistant Professor(SG)
School of Computer Science

SUBMITTED BY:
Sanaya Bhardwaj
SAP ID: 500091835
Roll no.: R2142210690

Part 1: Design and Implementation

INTRODUCTION:

In an increasingly interconnected world, managing personal and professional contacts efficiently is vital. ContactHub-The Contact Manager Microservices project leverages the power of Spring Boot, a robust Java-based framework, to create a scalable and highly maintainable solution for contact management.

MICROSERVICES DESIGN

The purpose and functionality of each microservices:

1. User Service:
Purpose: The User Service is responsible for managing user-related data and authentication. It handles user registration, login, and user profile management.
Functionalities:
The User Service is responsible for managing user-related data and operations, such as user registration, authentication, and profile management. When a client sends a request to the API Gateway that requires user-related information or actions, the API Gateway forwards the request to the User Service. The User Service processes the request and responds to the API Gateway.

2. Contact Service:
Purpose: The Contact Service is focused on managing contact-related data. It handles the creation, retrieval, modification, and deletion of contact records.
Functionalities:
The Contact Service manages contact-related data and operations, including creating, retrieving, updating, and deleting contact records. When a client needs to perform operations on contacts, the API Gateway routes the request to the Contact Service. The Contact Service processes the request and returns the relevant data or updates.

3. Eureka Server:
Purpose: The Eureka Server serves as the service registry and discovery server. It allows microservices to register themselves and discover other services.
Functionalities:
Eureka is used for service discovery and registration. Each microservice, including the User Service and Contact Service, registers itself with the Eureka server upon startup. Eureka maintains a registry of all available microservices and their network locations (IP addresses and ports).

4. API Gateway Service:

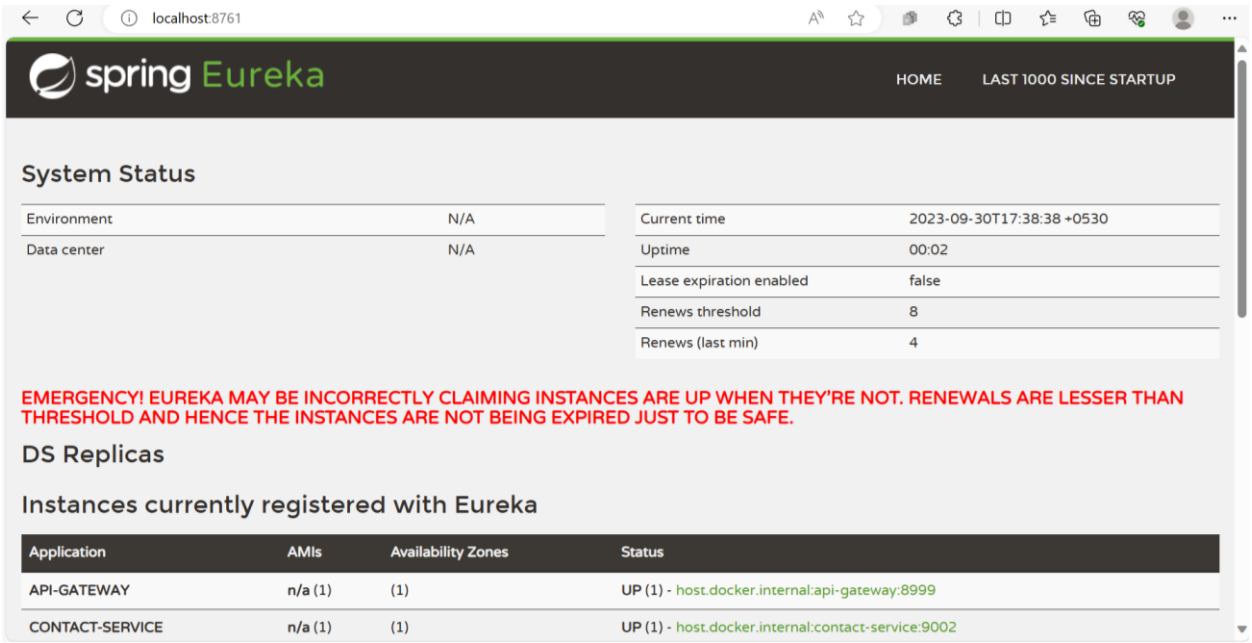
Purpose: The API Gateway Service acts as the entry point for client applications and centralizes requests to various microservices. It can handle authentication, load balancing, and routing.

Functionalities:

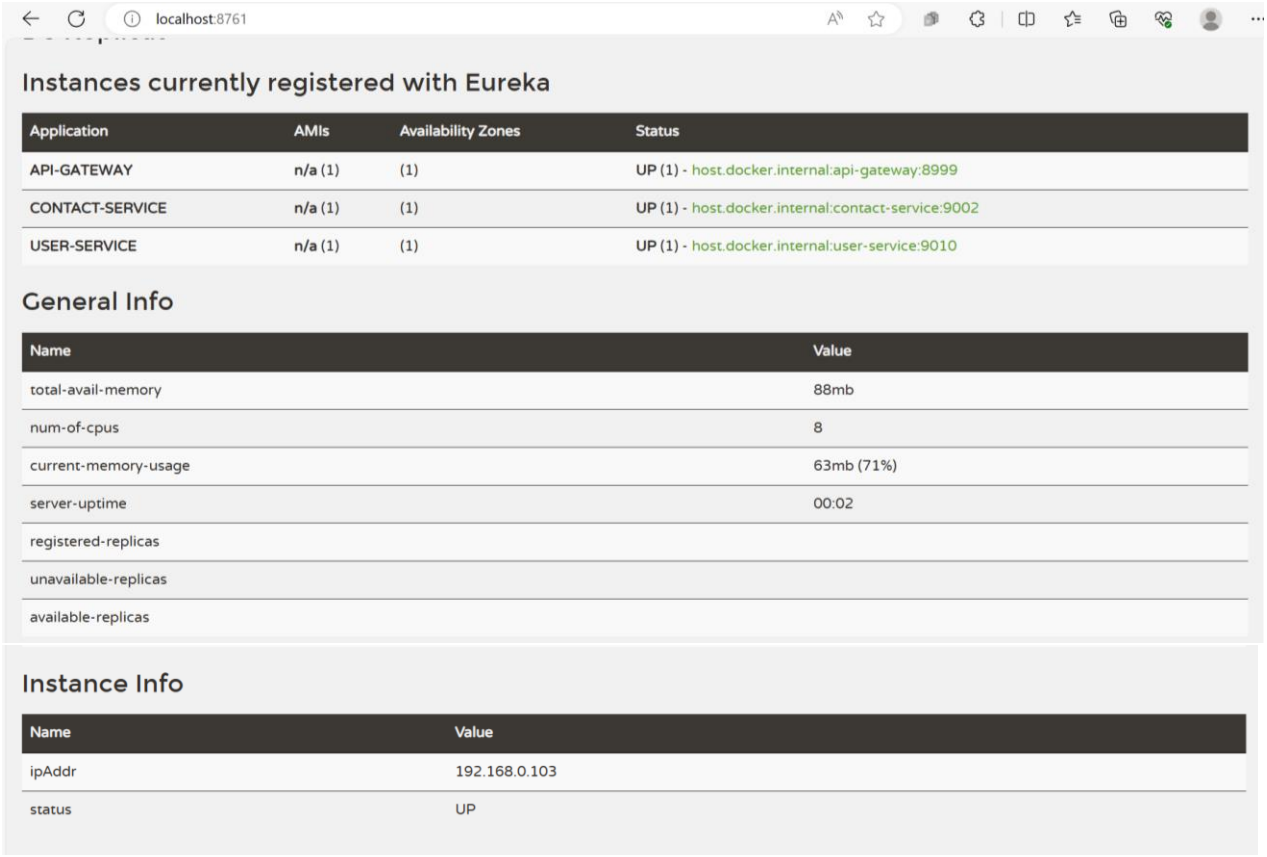
The API Gateway serves as the entry point for external clients (e.g., web or mobile applications) to interact with the microservices. It receives incoming requests from clients and routes them to the appropriate microservice based on the request path or URL. The API Gateway may also handle cross-cutting concerns such as authentication, logging, and rate limiting.

These microservices work together to create a modular and scalable architecture. Each microservice has its own specific area of responsibility, making it easier to develop, test, and maintain. The API Gateway Service provides a single-entry point for clients, offering a unified API while handling cross-cutting concerns such as authentication and load balancing.

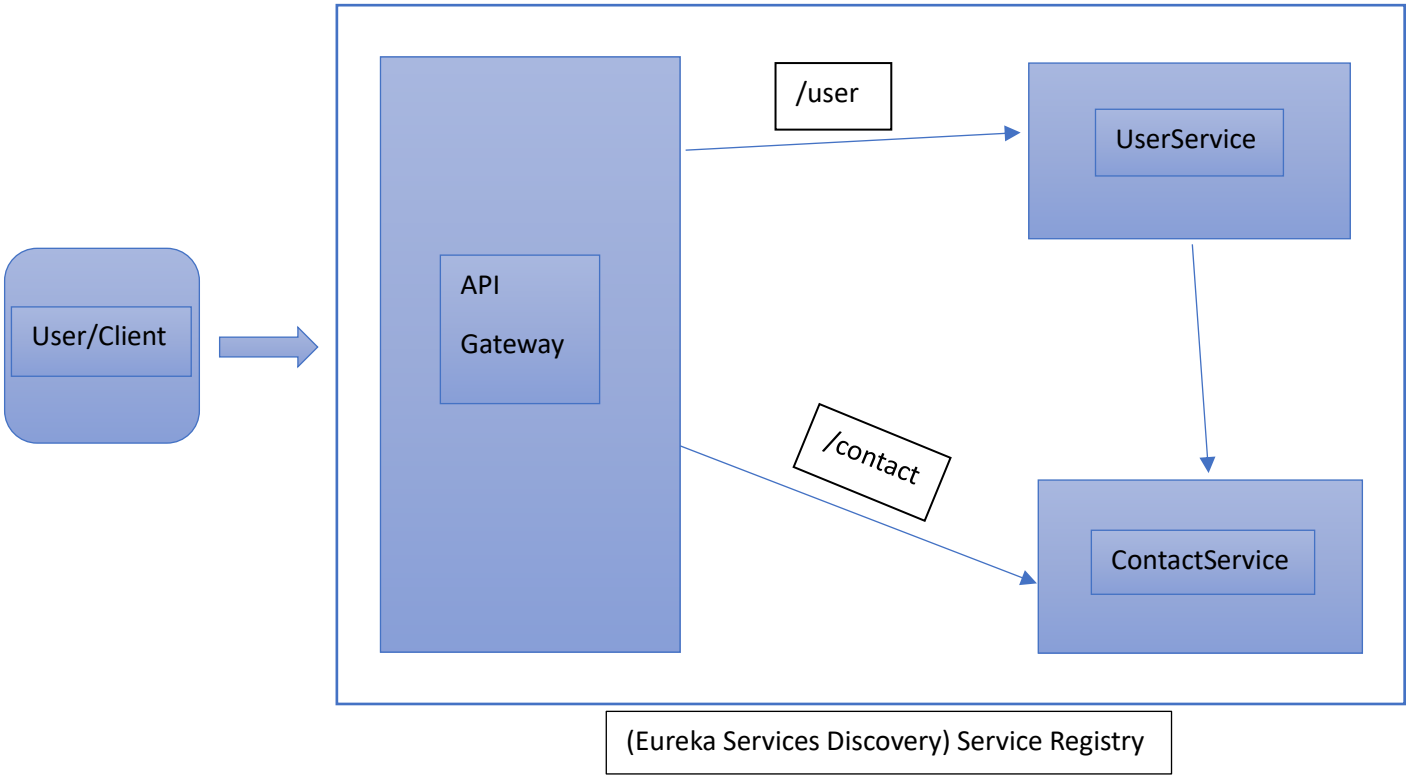
Additionally, Eureka Server ensures that microservices can dynamically discover and communicate with each other, allowing for flexibility in scaling and maintaining the system. This microservices architecture promotes agility, scalability, and the ability to deploy and update individual components independently, ultimately leading to a more resilient and maintainable system.



Spring Eureka shows a real-time catalog of available microservices and their network locations for service discovery in a microservices architecture.



How the microservices will communicate with each other?



- API Gateway is the entry point for client requests.
- Eureka (Service Registry) registers and tracks microservices.
- User Service manages user-related data.
- Contact Service handles contact-related data.
- API Gateway routes client requests to the appropriate service.
- Eureka enables services to discover each other.
- User Service handles user data and operations.
- Contact Service manages contact data.
- Services communicate via HTTP-based RESTful APIs.

Why breaking down the application into specific microservices?

Breaking down the Contact Manager application into specific microservices is justified for the following reasons:

1. Scalability: Microservices allow individual components to scale independently, optimizing resource utilization.

2. Modularity: Microservices promote modular development, making it easier to maintain and update specific functionalities.

3. Flexibility: Each microservice can use the most suitable technology stack for its specific task.

4. Resilience: Isolating services enhances fault tolerance, as a failure in one service does not affect others.

5. Specialization: Microservices are specialized in handling specific tasks, leading to better performance and efficiency.

6. Team Autonomy: Development teams can work independently on microservices, boosting productivity.

7. Improved Deployment: Smaller, focused services are easier to deploy, manage, and scale.

8. Easier Testing: Smaller services are easier to test and debug.

Containerization

Suitable technology for containerization

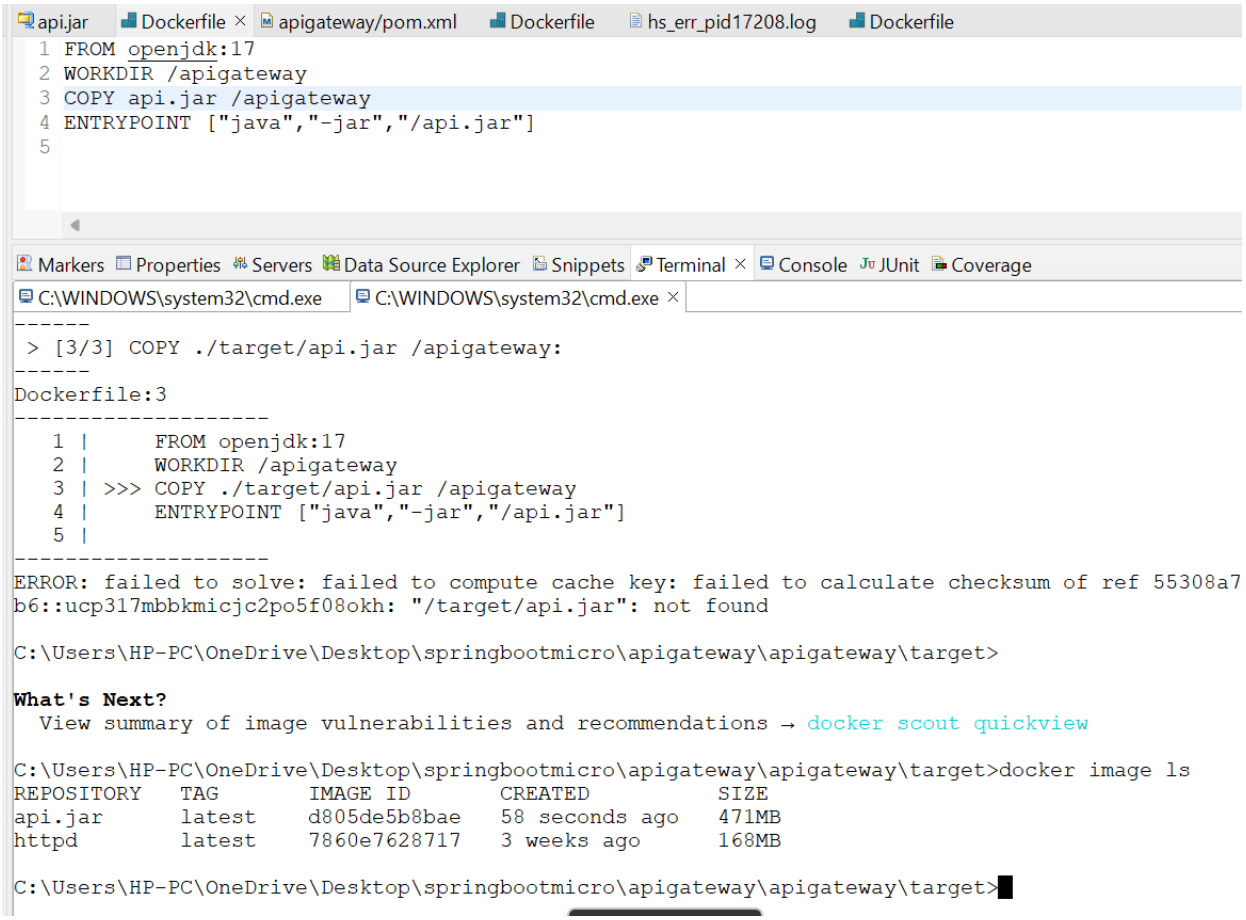
I have chosen Docker for Containerization as it provides a lightweight and consistent way to package, distribute, and run applications and their dependencies, ensuring portability, scalability, and efficient resource utilization.

Its advantages in the context of this project:

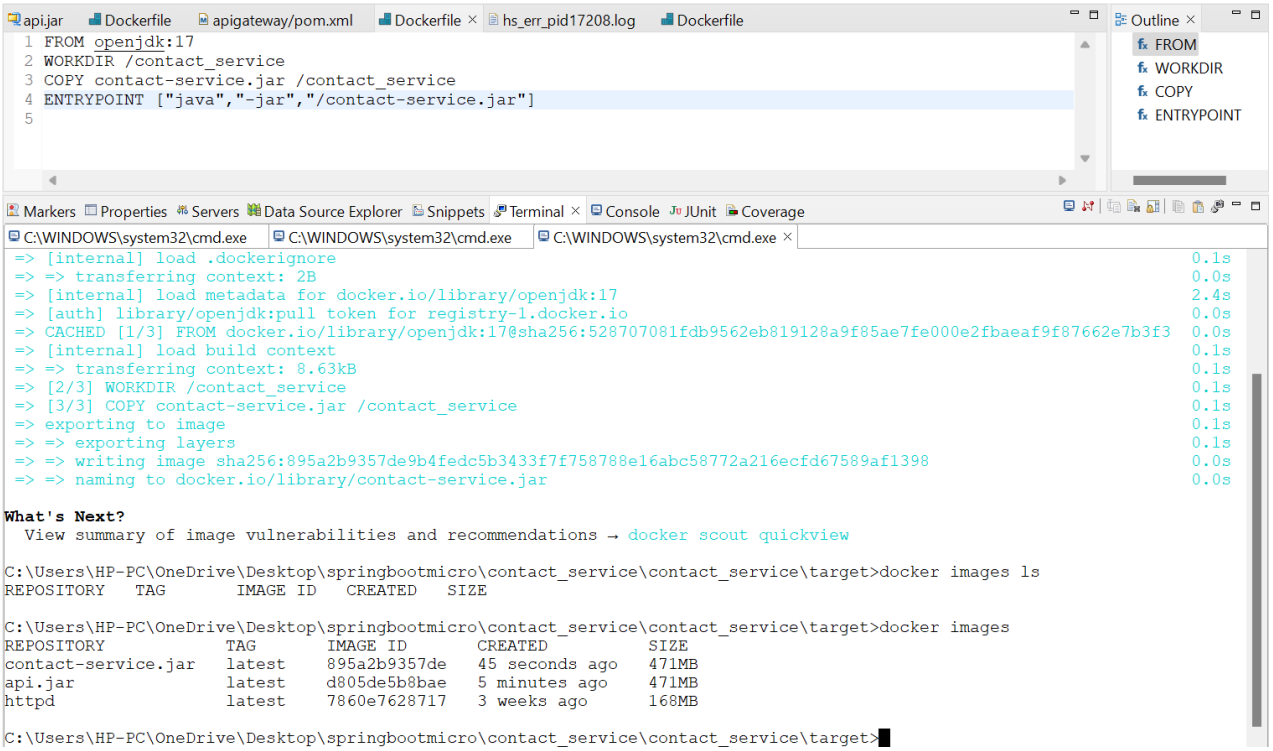
- **Isolation:** Docker containers isolate services, preventing conflicts and ensuring stability.
- **Portability:** Containers are portable across environments, simplifying deployment.
- **Efficiency:** Docker optimizes resource utilization and minimizes overhead.
- **Scalability:** Containers can be easily scaled up or down to accommodate varying loads.
- **Consistency:** Docker ensures consistent environments for development, testing, and production.
- **Dependency Management:** Simplifies managing and versioning application dependencies.
- **Security:** Provides a secure runtime environment, reducing security risks.
- **Fast Deployment:** Rapid provisioning and deployment of microservices.

Provide Dockerfiles for each microservice that include necessary dependencies and configurations.

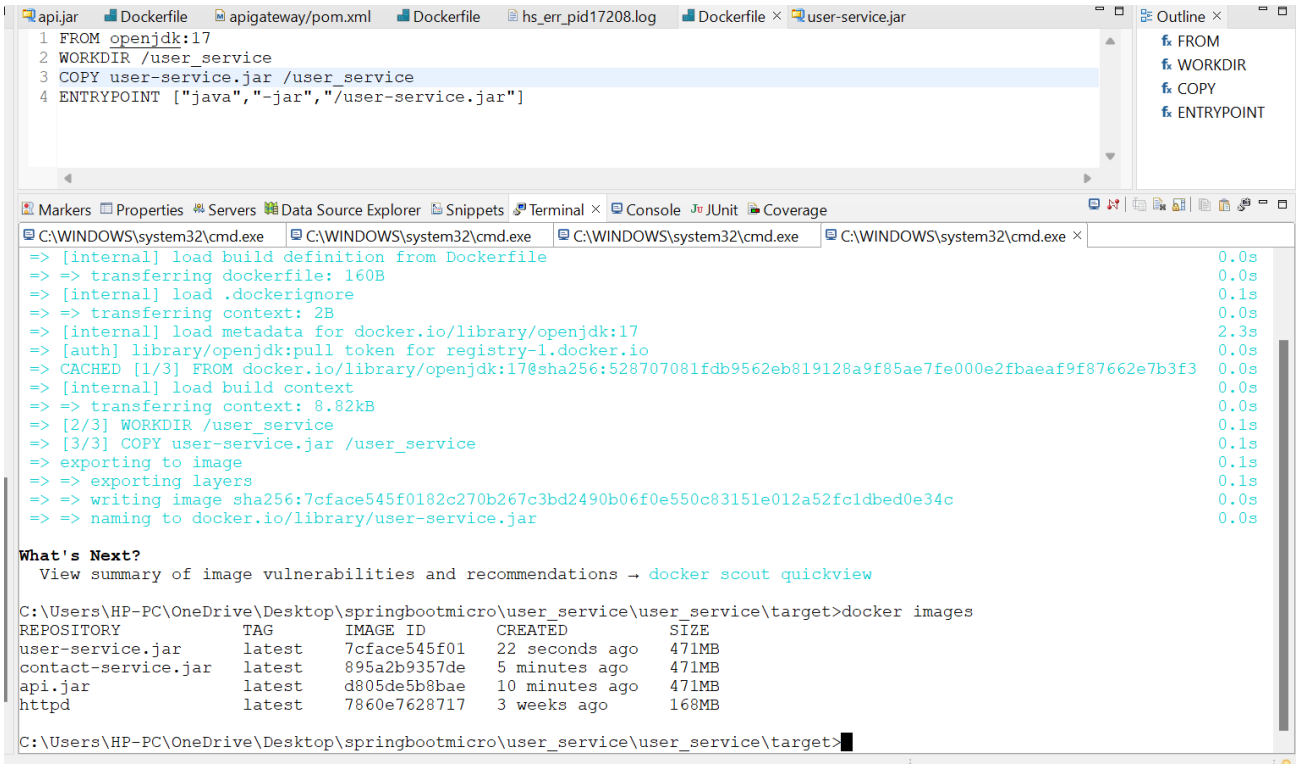
Dockerfile with the Docker image build for API gateway Service:



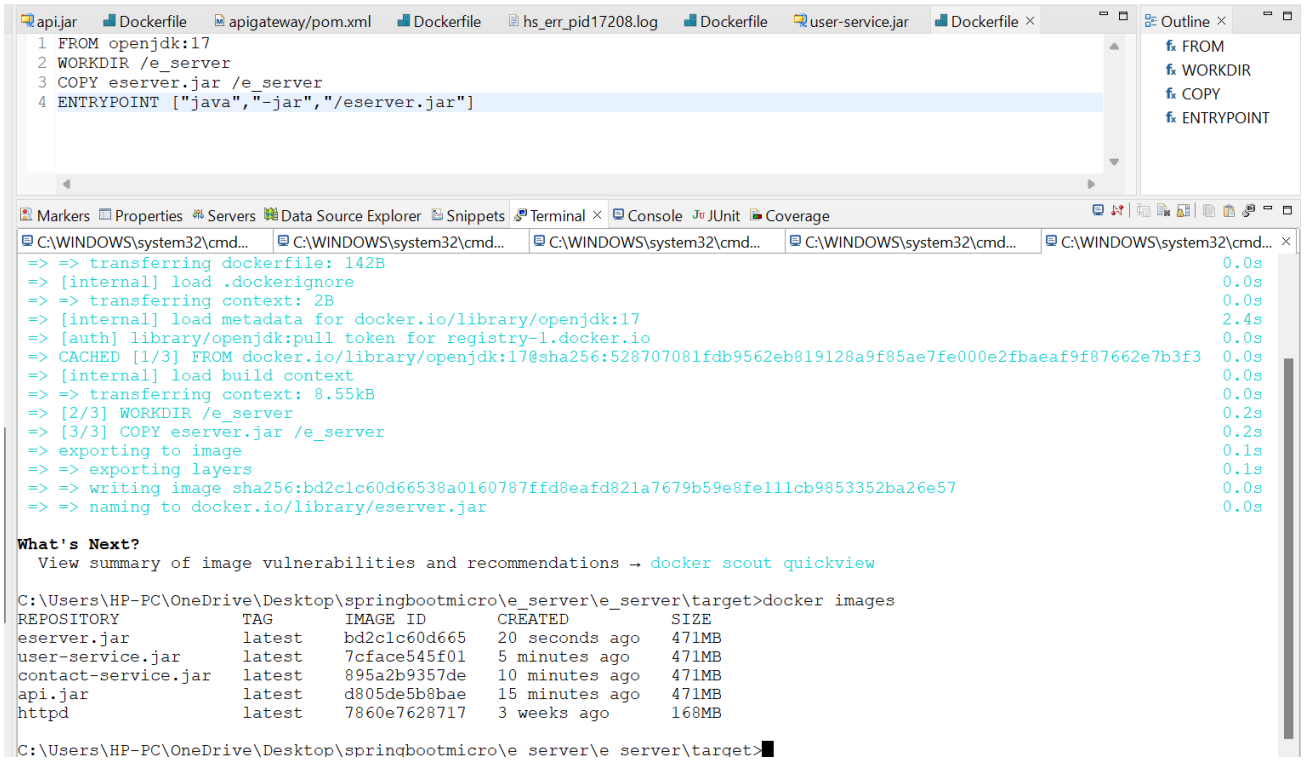
Dockerfile with Docker image build for Contact Service:



Dockerfile with Docker image build for User Service:



Dockerfile with Docker image build for Eureka Server:



Explain how containers facilitate scalability and isolation of microservices.

1. Scalability: Containers allow individual microservices to be scaled independently. Each microservice runs in its own container, making it straightforward to replicate and deploy more instances of a particular service when needed. This ensures that resources are allocated efficiently based on the specific demands of each microservice, enhancing the project's overall scalability.

2. Isolation: Containers provide a high degree of isolation between microservices. Each microservice runs in its own container with its own runtime environment, including its dependencies and libraries. This isolation prevents conflicts and interference between microservices, ensuring that failures or issues in one container do not impact others. It also simplifies updates and maintenance since changes to one microservice's container won't affect the others.

Overall, containers streamline the management of microservices in the Contact Manager project by offering both scalability and isolation, allowing for efficient resource utilization and reliable service operation.

Integration and Interactions

Explain how microservices will interact with each other through APIs or other means.

In the Contact Manager project with microservices (User, Contact, Eureka, API Gateway):

API Gateway: Serves as the entry point for client requests. It routes requests to the appropriate microservices based on the request path or URL.

Eureka (Service Registry): Microservices register themselves with Eureka upon startup. It maintains a registry of all available services and their network locations.

User Service: Manages user-related data. The API Gateway forwards user-related requests to this service.

Contact Service: Handles contact-related data. The API Gateway routes contact-related requests to this service.

Interactions:

- Microservices communicate via HTTP-based RESTful APIs.
- API Gateway routes client requests to the correct microservice.
- Eureka helps services discover each other dynamically.
- User and Contact Services may communicate directly if required (e.g., fetching user details for contact operations).

This architecture ensures efficient, organized, and modular communication between microservices in the Contact Manager project.

GITHUB LINK: https://github.com/sanaya-bhardwaj/ContactHub-Contact_management_system.git