**Docker Capstone Microservices Project**

*Name: Rohit Sanjeev Shetti*

**Introduction**

This document outlines the setup and deployment of a Dockerized microservices project, utilizing Docker Compose for orchestration. The project includes a MongoDB database and three microservices: Content Service, Interaction Service, and User Service. This architecture demonstrates a scalable and modular approach to application development, leveraging the benefits of microservices and containerization.

**Project Structure**

The project comprises the following components:

1. **MongoDB Service (mongo-db)**
   * **Purpose:** The MongoDB service provides persistent storage for the application. MongoDB is a NoSQL database known for its flexibility and scalability, making it suitable for handling various types of data and workloads.
   * **Ports:**
     + **Host Port:** 27017
     + **Container Port:** 27017
2. **Content Service (content-service)**
   * **Purpose:** This microservice manages tasks related to content, including storing and retrieving content data from the MongoDB database. It is designed to handle content-specific operations and provide APIs for content management.
   * **Ports:**
     + **Host Port:** 3001
     + **Container Port:** 3000
3. **Interaction Service (interaction-service)**
   * **Purpose:** The Interaction Service handles user interactions within the application. It stores interaction-related data in MongoDB and provides APIs for managing and retrieving interaction information.
   * **Ports:**
     + **Host Port:** 3002
     + **Container Port:** 3000
4. **User Service (user-service)**
   * **Purpose:** This microservice manages user-related information, including user profiles and authentication data. It interacts with MongoDB to store and retrieve user information, supporting user management features within the application.
   * **Ports:**
     + **Host Port:** 3003
     + **Container Port:** 3000

**Deployment Requirements**

This project can be deployed using Docker and Docker Compose. For scaling and managing the deployment in more complex environments, Kubernetes or other container orchestration platforms can be used.

**CI/CD Setup**

Integrated CI/CD tools such as Jenkins, GitHub Actions, or GitLab CI to automate the deployment process and set up pipelines to:

* **Build Docker Images:** Automatically build and push Docker images to a registry.
* **Deploy Containers:** Deployed updated containers to your environment.

**Security**

* **Network Segmentation:** Used Docker networks to isolate services and limit exposure.
* **Environment Variables:** Stored sensitive data like passwords securely using environment variables or secrets management solutions.

**Performance Optimization**

* **Resource Allocation:** Allocated appropriate CPU and memory resources to containers based on workload requirements.

**Conclusion**

This Dockerized microservices project demonstrates a modern approach to building scalable applications using containers and microservices architecture. By following the provided deployment instructions, users can set up and run the services in their environment efficiently.