Design Document: Microservices Architecture with Separate MySQL Database

Overview: The proposed design follows a microservices architecture to handle different functionalities of the system. The system consists of multiple microservices, each responsible for specific tasks, and a separate MySQL database to store data.

Microservices:

1. Flask Service:
- Responsible for handling HTTP requests and exposing RESTful APIs.
- Uses the Flask framework to handle API endpoints.
- Receives requests related to generating OSM, adding entitlements, and managing device keys.
- Validates incoming requests, processes the data, and interacts with the Kafka producer and MySQL database.
- Sends messages to the Kafka topic for further processing and stores data in the MySQL database.
- Endpoint examples:
- POST /generate_osm
- POST /add_entitlement
- POST /device_keys
2. Kafka Service:
- Listens to the Kafka topic for incoming messages.
- Receives messages from the Flask service and performs further processing.
- Processes the messages, applies business logic, and performs necessary actions based on the message type.
- Example actions:
- Generate OSM based on the received message.
- Add entitlements to the system.
- Manage device keys.
- Interacts with the MySQL database for data retrieval and storage.
3. UDP Multicast Service:
- Sets up a UDP socket to receive multicast data.
- Decrypts the received data using AES encryption and the provided key.
- Monitors the received data for specific conditions and triggers actions accordingly.
- Measures and calculates the EMM bandwidth.

- Utilizes the Prometheus client library to collect and push metrics to the Pushgateway.

Database:

1. MySQL Database:

- Stores data related to the system, such as generated OSM, entitlements, device keys, and other relevant information.
- Consists of multiple tables:
- generate_osm: Stores generated OSM information.
- entitlements: Stores entitlements information.
- devices: Stores device keys information.
- The microservices interact with the database to store and retrieve data using SQL queries.
- Ensures data integrity, consistency, and scalability.

Integration and Communication:

- The Flask service communicates with the Kafka service through the Kafka producer.
- The Kafka service listens to the Kafka topic and performs necessary actions based on the received messages.
- The UDP Multicast service receives data from the multicast group and decrypts it using AES encryption.
- The UDP Multicast service pushes metrics to the Prometheus Pushgateway for monitoring and analysis.

Deployment and Scalability:

- Each microservice can be deployed as a separate containerized service using containerization technologies like Docker or Kubernetes.
- The Flask service can be scaled horizontally based on the incoming HTTP request load.
- The Kafka service can be scaled based on the message processing load and the number of Kafka topics.
- The MySQL database can be deployed on separate database servers or clusters to handle the data storage and retrieval efficiently.

Conclusion: The proposed microservices architecture with a separate MySQL database provides a scalable and modular design for handling different functionalities of the system. Each microservice is responsible for a specific task, ensuring loose coupling and flexibility. The MySQL database ensures data persistence and reliability. By separating concerns and utilizing appropriate technologies, the system can achieve scalability, fault tolerance, and maintainability.