Sure, here is a detailed system design document for a microservice architecture with MySQL database:

System Overview

The system consists of five microservices:

- SMS-CAS Interface: This microservice accepts requests from SMS, which sends REST API requests on different endpoints such as `/generateosm`, `/adddevice`, `/addentitlement`, `/removedevice`, `/removeentitlement`, and `/removedevice`. The SMS-CAS Interface takes these requests, saves them in the database, and logs them into a log file. It then responds back to SMS with an appropriate response.
- EMMG: This microservice is responsible for generating EMM based on data received in the SMS-CAS Interface. The EMM data is then sent to the Encryptor component.
- Encryptor: This microservice encrypts data in envelope encryption and responds back to EMMG. The Encryptor must support 100 encryptions per second.
- Cycler: This microservice takes encrypted EMM from the database and carousels it based on some fixed configuration. The Cycler logs all the data being carouselled.
- Scheduler: This microservice runs fixed tasks, such as changing keys used for encryption and removing rows from the database for which expiry has passed.

System Architecture

The system architecture is shown in the following diagram:

The SMS-CAS Interface, EMMG, Encryptor, and Cycler microservices are all deployed on the same server. The Scheduler microservice is deployed on a separate server.

Data Model

The data model for the system is shown in the following diagram:

The database consists of the following tables:

- Requests: This table stores all the requests that are received from SMS.
- EMMs: This table stores all the encrypted EMMs.
- Keys: This table stores all the encryption keys.

Interface/API Definitions

The SMS-CAS Interface exposes the following REST API endpoints:

- `/generateosm`: This endpoint generates an EMM for a given device.
- `/adddevice`: This endpoint adds a device to the system.
- `/addentitlement`: This endpoint adds an entitlement to a device.
- `/removedevice`: This endpoint removes a device from the system.
- `/removeentitlement`: This endpoint removes an entitlement from a device.
- `/removedevice`: This endpoint removes a device from the system.

The Encryptor and Cycler microservices do not expose any public APIs.

CI/CD

The system is built and deployed using a CI/CD pipeline. The pipeline is triggered by changes to the code in the GitHub repository. The pipeline consists of the following steps:

- The code is checked out from GitHub.
- The code is built and tested.
- The code is deployed to the production environment.

Fault Tolerance

The system is designed to be fault tolerant. The following measures are taken to ensure fault tolerance:

- The microservices are deployed in a cluster.
- The database is replicated.
- The system is monitored using Prometheus and AlertManager.

Security

The system is secured using the following measures:

- The microservices are deployed behind a load balancer.
- The database is encrypted.
- The requests to the microservices are authenticated using OAuth 2.0.

Testing

The system is tested using the following methods:

- Unit testing
- Integration testing
- System testing

Monitoring

The system is monitored using Prometheus and AlertManager. Prometheus collects metrics from the system and AlertManager sends alerts when the metrics exceed certain thresholds.

Deployment

The system is deployed using Kubernetes. Kubernetes is a container orchestration platform that makes it easy to deploy and manage containerized applications.

Conclusion

This document has described the system design for a microservice architecture with MySQL database. The system is designed to be scalable, fault tolerant, and secure.