Technical Architecture Document - MysticMail

Alexandra Ștefan, Keren Boingiu

1 General Overview

MysticMail is a cloud-native digital messaging application where users can send messages with an element of uncertainty, using a probability-based mechanism and randomized delay. The system is built on a microservices architecture, utilizing Docker containers and orchestration with Docker Swarm/Kubernetes.

2 Components and Microservices

2.1 Core Microservices

2.1.1 Message Management Microservice

- Responsible for creating, storing, and sending messages based on the selected probability.
- Implements delay and randomization mechanisms for message delivery.
- Exposes a REST API for frontend interaction.

2.1.2 User Authentication and Management Microservice

- Handles user accounts (registration, login, secure password storage using hashing).
- Allows users to view their message history.
- Uses basic authentication logic without advanced security layers like JWT, since the project's focus is not on complex authentication mechanisms.

2.1.3 Notification Microservice

- Sends notifications to users when a message has been delivered.
- Supports email notifications.

2.2 Supporting Components

2.2.1 Database

- PostgreSQL: Relational database for storing messages and users.
- Adminer: User-friendly interface to manage the database

2.2.2 API Gateway (Kong)

- Manages routing and security for API calls between microservices.
- Provides authentication and rate limiting to prevent abuse.

2.2.3 Logging and Monitoring

- Prometheus and Grafana for tracking service performance and real-time analytics.
- Elastic Stack (ELK) for centralized logging.

2.2.4 CI/CD and Deployment

- GitHub CI/CD for automated deployment.
- Docker Swarm/Kubernetes for orchestration and scalability.

3 Communication Between Components

3.1 Microservices communicate through:

- REST API (HTTP) for main interactions.
- Notifications triggered via webhooks and external services (Postman).

3.2 Application Flow Diagram:

1. The user composes a message and selects a delivery probability.
2. The message is stored in the database and managed by the messaging microservice.
3. The system applies the delay/randomization logic and decides on message delivery.
4. If the message is sent, the recipient receives a notification.
5. The sender can view in their history whether the message was sent.

4 Responsibilities Allocation

- **Alexandra** – Responsible for backend development, implementation of some microservices, and API setup.
- **Keren** – Responsible for database setup, API Gateway configuration, and authentication services.
- **Joint Responsibilities:**
  - Collaboratively designing and refining the application architecture.
  - Working together on logging, monitoring, and CI/CD pipelines.
  - Jointly handling testing, debugging, and performance optimization.

5 Technologies Used

Component	**Technology**
Containers and Orchestration	Docker, Docker Swarm/Kubernetes
Cluster Management Ui	Portainer
Database	PostgreSQL
DB Administation service	Adminer
API Gateway	Kong
Logging and Monitoring	Prometheus, Grafana, ELK Stack
CI/CD	GitHub Actions
Message Management Microservice	Python (Flask)
User Authentication Microservice	Python (Flask), Hashed Password
Notification Microservice	Python (Flask), Postman (mock email)

Table 1: Technologies used in MysticMail

6 Diagram

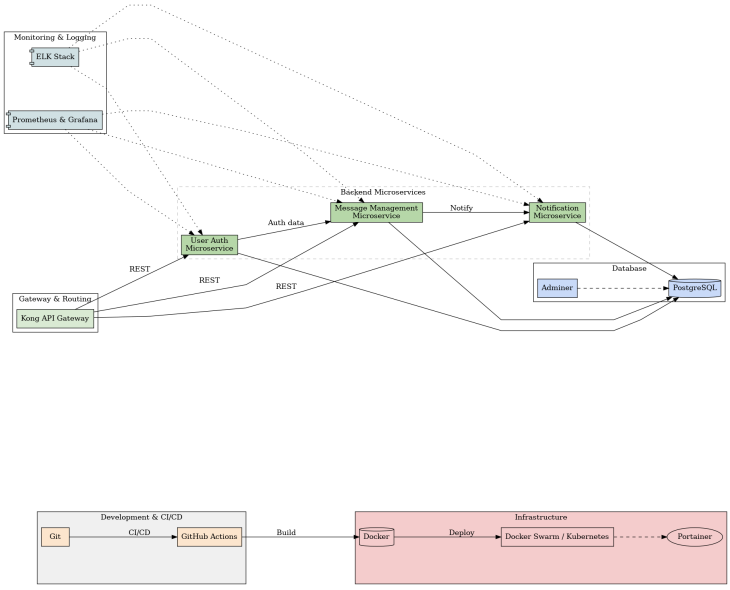

Figure 1: Architecture Diagram

7 Conclusion

MysticMail is built using a modern microservices architecture with containerization to keep it scalable and flexible. It is a structured application with clear, separate components, making it easier to develop, manage, and expand as needed.

8 Github Repository