Final Project Report

Muchammad Daniyal Kautsar
21/479067/TK/52800
TIF5113 Software Architecture

1 Introduction

We present the architectural design and motivation behind implementing a robust, scalable microservices-based system. The system comprises three primary services: **User Service**, **Catalog Service**, and **Order Service**, each performing distinct roles to manage user accounts, book inventories, and order processes, respectively. We employ modern tools and practices to ensure high availability, scalability, and maintainability.

2 System Architecture

2.1 Microservices Architecture

The system is designed using a **microservices architecture**, which decomposes the application into smaller, independent services. Each service is responsible for a specific business capability and communicates with others through well-defined APIs.

2.2 Key Components

2.2.1 User Service

- **Functionality**: Manages user accounts and authentication.
- **Technology**: Python (Flask), MongoDB.
- **API Endpoints**:
  - `POST /register`: Register a new user.
  - `POST /login`: Authenticate a user.

2.2.2 Catalog Service

- **Functionality**: Handles the inventory of books.
- **Technology**: Python (Flask), MongoDB, Redis (for caching).
- **API Endpoints**:
  - `POST /books`: Add a new book to the catalog.
  - `GET /books`: Retrieve the list of books.
  - `GET /books/{id}`: Retrieve details of a specific book.

2.2.3 Order Service

- **Functionality**: Manages the order process.
- **Technology**: Python (Flask), MongoDB.
- **API Endpoints**:
  - `POST /order`: Place a new order.
  - `GET /orders/{username}`: Retrieve orders for a specific user.

2.3 Supporting Components

2.3.1 Nginx Gateway

- **Role**: Acts as a reverse proxy, routing incoming requests to the appropriate service based on URL paths.
- **Ports**:
  - `5000`: Gateway entry point.
  - `5001-5003`: Service-specific ports.

2.3.2 MongoDB

- **Role**: Stores persistent data for user accounts, books, and orders.
- **Instances**:
  - `userdb`: User data.
  - `catalogdb`: Book data.
  - `orderdb`: Order data.

2.3.3 Redis

- **Role**: Provides caching to enhance the performance of the Catalog Service.

2.3.4 Docker

- **Role**: Containerization and node replication via Docker Swarm for ensuring low downtime, higher availability and scalability.

3 Motivation

- **Modularity and Maintainability:** Independent development, deployment, and scaling of services simplifies management and adaptation to changes.
- **Scalability:** Individual services can be scaled based on their specific load, ensuring efficient resource allocation.
- **Fault Isolation:** Failure of one service does not impact the entire system, maintaining service availability.
- **Technology Agility:** Each service can leverage the most suitable technologies, enabling optimal performance and flexibility.
- **Continuous Deployment:** Regular updates to services are possible without disrupting the overall system, supporting modern DevOps practices.
- **High Availability:** Docker Swarm enables replication and redundancy, ensuring the system can handle failures gracefully.

Appendix

Code Repositories

All of the code repositories can be found at github.

System Architecture Overview

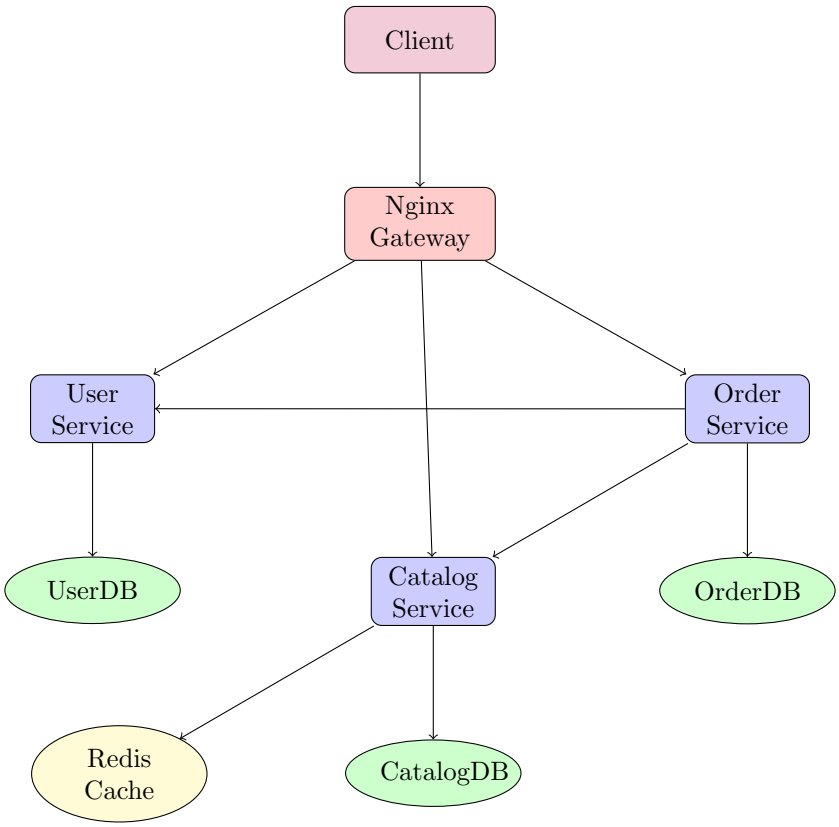

Figure 1: System Architecture Overview