Task 4.3HD: Project Check-In – Healthcare Appointment Booking System

I. Overview

The Healthcare Appointment Booking System is designed to streamline the process of scheduling medical appointments, providing an efficient and scalable solution for healthcare providers and patients.

- The primary objective is to develop a web-based application that allows users to book and view appointments, manage patient data securely, and deploy the system on a Kubernetes cluster for scalability and reliability.
- Key functionalities include a user-friendly booking interface, secure data storage for appointment records, and automated scaling to handle peak loads, ensuring a seamless experience during high-demand periods.

Significant progress has been made in this check-in phase:

- Kubernetes has been set up on Docker Desktop, with a test Nginx pod and a Secret (db-secret) deployed to validate the environment.
- A basic Flask application has been developed with two routes (/ and /book) to simulate the booking interface, running locally on localhost:5000.
- Additionally, PostgreSQL has been configured in a Docker container (postgres), with a database (healthcare) and a table (appointments) created to store appointment data, including a sample patient record.

These foundational steps pave the way for further development, including API integration, Kubernetes deployment, and final demonstration on Google Kubernetes Engine (GKE).

II. Work Outline

A/ Architecture of the Project

The Healthcare Appointment Booking System consists of the following major components, designed to work together within a Kubernetes environment:

- **Frontend and Backend (Flask Application)**: A Flask app handles both user-facing interfaces (e.g., booking page) and backend logic (e.g., API endpoints). Currently, it has two routes: / for the homepage ("Healthcare Appointment Booking System") and /book for the booking page ("Book an Appointment").

- **Database (PostgreSQL)**: PostgreSQL stores appointment data in a database named healthcare, with a table appointments (columns: id, patient_name, doctor_id, appointment_time). It will be deployed using a StatefulSet in Kubernetes to ensure data persistence.

- **Kubernetes Infrastructure**:
  - **Pods**: Flask app will run in pods for stateless components.
  - **StatefulSet**: PostgreSQL will use a StatefulSet for reliable storage and replication.
  - **Secrets**: A Secret (db-secret) has been created to store sensitive data (e.g., database password).
  - **Horizontal Pod Autoscaler (HPA)**: Will be implemented to scale Flask pods during peak booking times.
  - **Services**: A Service (e.g., NodePort) exposes the Flask app, as demonstrated with a test Nginx pod.

These components interact as follows: Users access the Flask app via a Service, the app processes requests (e.g., booking an appointment) and interacts with PostgreSQL to store or retrieve data, while Kubernetes manages scaling and reliability.

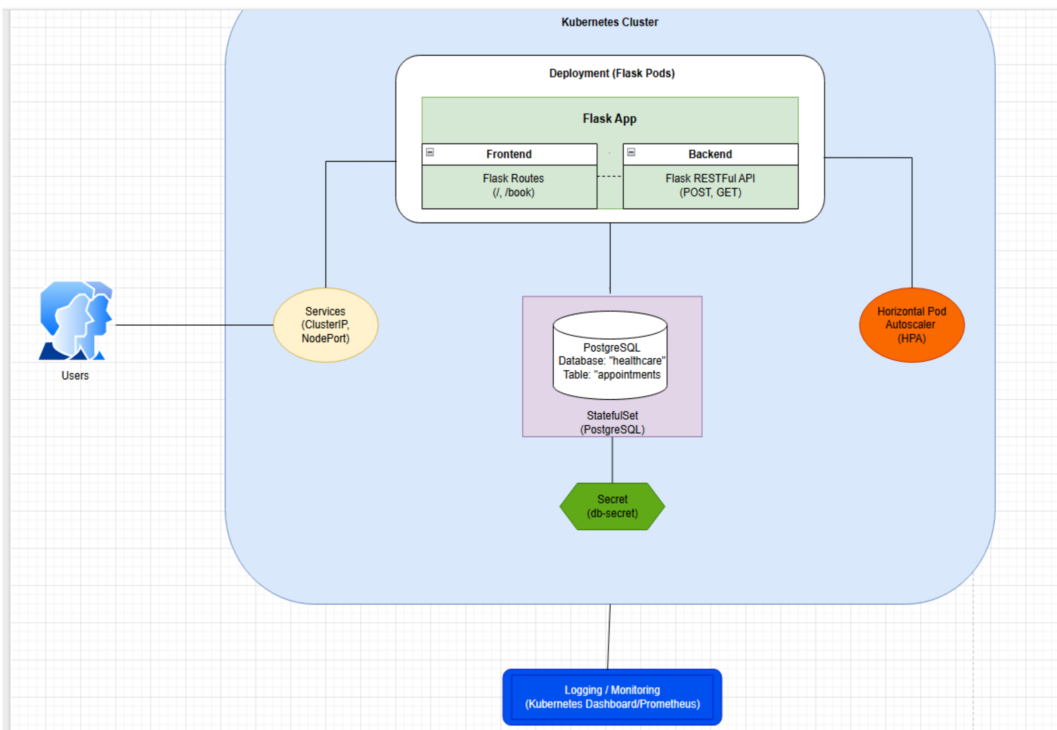

Figure 1: Architecture of Healthcare Appointment Booking System.

B/ Work Completed

Significant progress has been made toward setting up the foundation of the project:

- Configured Kubernetes on Docker Desktop and verified it by deploying a sample Nginx pod with a NodePort Service, accessible via the browser.

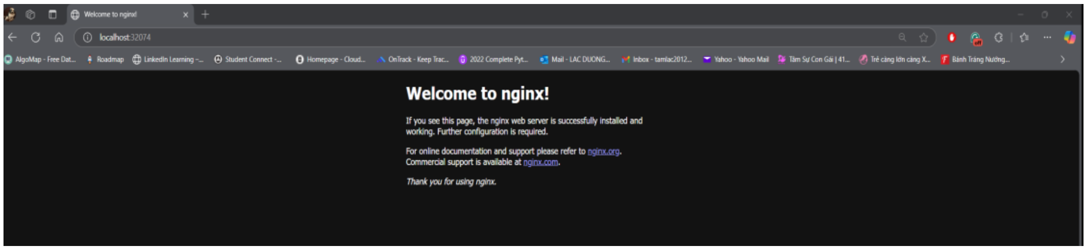

Figure 2: Nginx Pod Deployment on Kubernetes.

- Created a Kubernetes Secret (db-secret) to store a sample database password (qelol669), laying the groundwork for secure data management.

```
C:\Users\Lac T. Duong>kubectl create secret generic db-secret --from-literal=password=qelol669
secret/db-secret created

C:\Users\Lac T. Duong>kubectl get secrets
NAME        TYPE     DATA   AGE
db-secret   Opaque   1      7s
```

Figure 3: Applied Secret service through K8s.

- Developed a basic Flask application with two routes: / (displays "Healthcare Appointment Booking System") and /book (displays "Book an Appointment"). The app runs locally on localhost:5000.

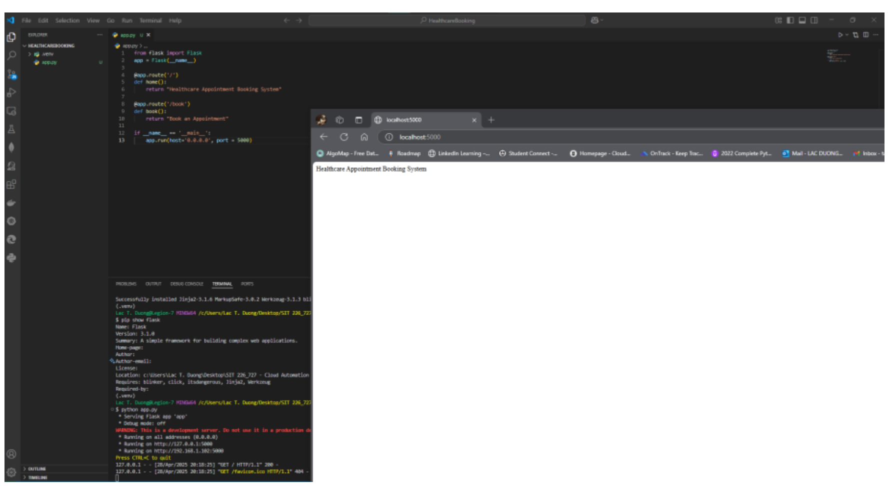

Figure 4: Flask Homepage (/).

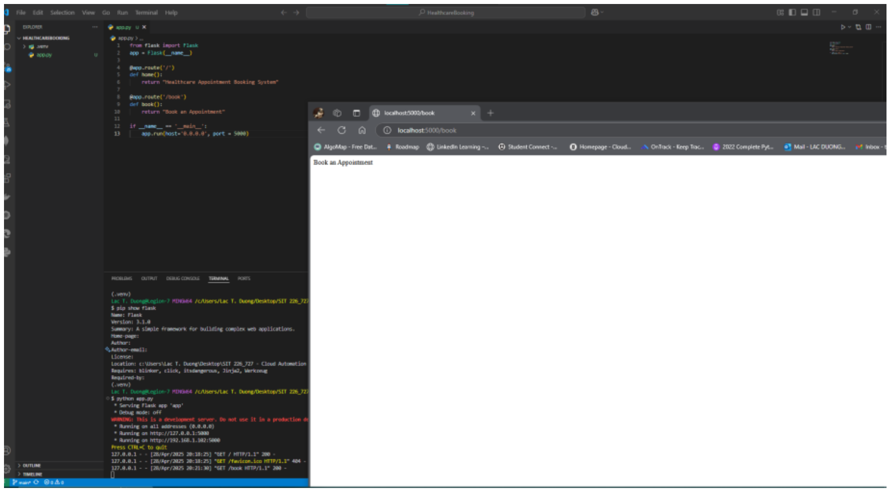

Figure 5: Flask Booking Page (/book).

- Set up PostgreSQL using Docker, created a container named postgres, and confirmed it runs on port 5432.
- Created a database healthcare with a table appointments (columns: id, patient_name, doctor_id, appointment_time). Successfully inserted a sample patient record (e.g., "John Doe") to verify functionality.

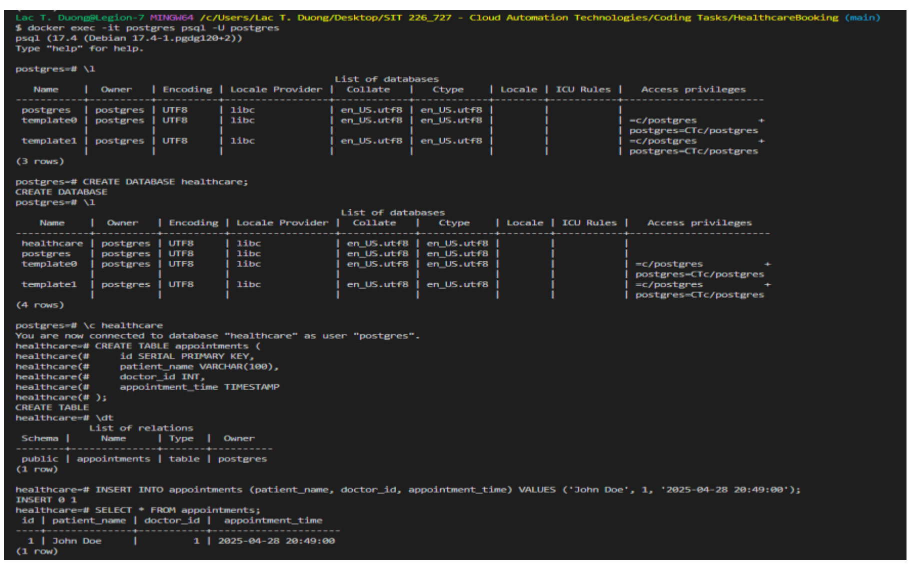

Figure 6: PostgreSQL "appointments" Table Structure.

C/ Work Remaining

Several tasks remain to complete the project and prepare for the final demonstration:

- **Frontend Development**:
    - Enhance the Flask app by adding HTML/CSS templates for a user-friendly booking interface (e.g., a form to select a doctor and time).
- **Backend API Development**:
    - Implement API endpoints in Flask (e.g., POST /book-appointment to save appointments, GET /appointments to retrieve them) and integrate with PostgreSQL for data storage and retrieval.
- **Containerization**:
    - Create a Dockerfile for the Flask app and build a container image to deploy it on Kubernetes.
- **Kubernetes Deployment**:
    - Deploy the Flask app as a pod in Kubernetes, using a Deployment and Service (e.g., ClusterIP or NodePort) for access.
    - Configure a StatefulSet for PostgreSQL to ensure data persistence and integrate it with the Flask app using Secrets for credentials.
    - Set up Horizontal Pod Autoscaling (HPA) to scale Flask pods based on demand (e.g., CPU usage or booking traffic).
    - Implement logging and monitoring (e.g., using Kubernetes Dashboard or Prometheus) to track system performance.
- **Testing and Optimization**:
    - Test the system end-to-end (booking an appointment via the UI, verifying data in PostgreSQL) and optimize performance (e.g., database indexing, pod resource limits).
- **Final Deployment**:
    - Deploy the system on Google Kubernetes Engine (GKE) for the final demonstration, showcasing advanced Kubernetes features.

D/ Dependencies

The remaining tasks have the following dependencies:

- The API endpoints require the PostgreSQL database to be fully set up and accessible (e.g., via StatefulSet).

- Autoscaling (HPA) depends on the Flask app and API being deployed and functional, as scaling will be based on traffic or resource usage.
- The final GKE deployment requires all components (Flask, PostgreSQL, Kubernetes configurations) to be tested locally on Docker Desktop first.

E/ Risks and Mitigation

Potential challenges and their mitigations include:

- **StatefulSet Configuration for PostgreSQL**:
  - Configuring a StatefulSet may be complex due to persistent storage requirements.
  - Mitigation: Refer to Kubernetes documentation (https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/) and test incrementally on Docker Desktop before deploying to GKE.
- **GKE Costs**:
  - Deploying on GKE may incur unexpected costs.
  - Mitigation: Continue using Docker Desktop for development and testing if GKE expenses exceed budget; GKE will only be used for the final demonstration.
- **Performance Issues**:
  - The system may face bottlenecks during peak booking times.
  - Mitigation: Implement HPA and optimize database queries (e.g., add indexes on appointments table) to handle load efficiently.

III. Platform

The project is developed and tested on the following platforms, balancing cost, accessibility, and scalability:

- **Current Platform (Docker Desktop with Kubernetes)**:
  - Development and testing are conducted on Docker Desktop with integrated Kubernetes. This platform was chosen for its zero cost, ease of setup, and ability to simulate a Kubernetes cluster locally.
  - Progress includes deploying a test Nginx pod, creating a Secret (db-secret), and running PostgreSQL in a container (postgres), confirming a stable environment for local development.

- **Planned Platform (Google Kubernetes Engine - GKE)**:
    - The final demonstration will be deployed on GKE, leveraging its robust support for autoscaling, monitoring, and production-grade Kubernetes features.
    - GKE is ideal for showcasing the system's scalability (e.g., HPA for Flask pods) and reliability (e.g., managed StatefulSets for PostgreSQL), aligning with HD requirements.
- **Experience and Learning**:
    - I have gained hands-on experience with Docker Desktop Kubernetes, successfully deploying pods and Secrets.
    - I am currently learning GKE through official documentation (https://cloud.google.com/kubernetes-engine/docs) to prepare for the final deployment.
- **Risks and Backup**:
    - GKE deployment may incur costs beyond the budget.
    - As a backup, I will continue using Docker Desktop for the final demo if needed, ensuring the project remains feasible while still demonstrating Kubernetes capabilities.

IV. Kubernetes Features

The Healthcare Appointment Booking System leverages several Kubernetes features to ensure scalability, reliability, and security:

- **StatefulSets**:
    - PostgreSQL will be deployed using a StatefulSet to manage persistent storage and ensure data consistency for the healthcare database.
    - This is critical for maintaining appointment records, as StatefulSets provide stable network identifiers and persistent volumes, addressing the need for reliable database operations.
- **Secrets**:
    - A Secret (db-secret) has been created to securely store sensitive data, such as the database password (qelol669).
    - This will be extended to manage other credentials (e.g., API keys) in production, ensuring secure access to the database and protecting patient data.

- **Horizontal Pod Autoscaling (HPA)**:
  - HPA will be implemented to automatically scale Flask pods based on demand (e.g., CPU usage or booking traffic).
  - For example, during peak hours (e.g., morning appointment rushes), additional pods will be spun up to handle increased user load, ensuring system responsiveness.
- **Services**:
  - A NodePort Service was used to expose a test Nginx pod, confirming Kubernetes networking capabilities.
  - This will be applied to the Flask app (e.g., using ClusterIP or NodePort) to allow users to access the booking interface via a browser.
- **Logging and Monitoring**:
  - Plans include integrating tools like the Kubernetes Dashboard or Prometheus to monitor pod performance and track system metrics (e.g., response time, pod CPU usage).
  - This will help identify bottlenecks and ensure system reliability during operation.

V. Unit Learning Outcomes

- **ULO1 - Understand and apply cloud resource management concepts**:
  - **Progress**:
    - Configured Kubernetes on Docker Desktop and deployed a test Nginx pod with a NodePort Service, demonstrating resource allocation (e.g., ports, pods).
    - Created a Secret (db-secret) to manage sensitive data, showing secure resource management.
  - **Future**:
    - Will compare resource usage between Docker Desktop (local) and GKE (cloud) during final deployment, analyzing trade-offs (e.g., cost vs. scalability) to deepen understanding of cloud resource management in a healthcare context.
- **ULO2 - Install and configure cloud infrastructure**:
  - **Progress**:

    - Successfully set up Kubernetes on Docker Desktop, deployed a pod (Nginx), and created a Secret (db-secret).
    - Installed PostgreSQL in a Docker container (postgres), created a database (healthcare), and a table (appointments), confirming infrastructure readiness.
  - **Future**:
    - Will configure a StatefulSet for PostgreSQL and HPA for Flask pods, demonstrating advanced configuration skills.
    - Plans include integrating logging/monitoring (e.g., Prometheus) to ensure infrastructure reliability.
- **ULO3 - Analyze and evaluate cloud platforms for business impact**:
  - **Progress**:
    - Evaluated Docker Desktop as a cost-free, local platform for development, suitable for testing Kubernetes features.
    - Identified GKE as the production platform for its scalability and monitoring capabilities, relevant to healthcare system needs (e.g., handling peak booking loads).
  - **Future**:
    - Will assess GKE's business impact (e.g., improved patient experience through scalability) versus costs, using Docker Desktop as a fallback if GKE expenses are prohibitive, ensuring cost-effective deployment while meeting user needs.
- **ULO4 - Work collaboratively in designing and managing cloud applications**:
  - **Progress**:
    - Designed a modular system (Flask app, PostgreSQL, Kubernetes) simulating a team environment.
    - Created an architecture diagram showing component interactions, supporting collaborative design.
  - **Future**:
    - Will manage the application lifecycle by deploying to Kubernetes (e.g., Flask pods, PostgreSQL StatefulSet), testing end-to-end functionality (e.g., booking an appointment).
- **ULO5 - Optimize cloud application performance using metrics**:

- **Progress**:
  - Used Kubernetes Secrets to secure data and tested basic Flask functionality (two routes).
  - Inserted sample data into PostgreSQL (appointments table) to verify database performance.
- **Future**:
  - Will implement HPA to optimize Flask pod scaling based on metrics (e.g., CPU usage).
  - Plans include using Kubernetes Dashboard or Prometheus to collect metrics (e.g., response time, pod usage) and optimize database queries (e.g., indexing appointments table) for better performance.

VI. References

Kubernetes (2025) *Kubernetes Documentation*. Available at: https://kubernetes.io/docs/ (Accessed: 20 April 2025).

Flask *Flask's Documentation*. Available at: https://flask.palletsprojects.com/en/stable/ (Accessed: 23 April 2025).

PostgreSQL (2025) *PostgreSQL Documentation*. Available at: https://www.postgresql.org/docs/ (Accessed: 23 April 2025)

Dockerdocs *Dockerdocs*. Available at: https://docs.docker.com/ (Accessed: 21 April 2025)

Google *Google Kubernetes Engine (GKE)*. Available at: https://cloud.google.com/kubernetes-engine/docs (Accessed: 26 April 2025)