Implementation of the backend for a hospital system with GCP

Pedro Martins[a,1], **Mariia Zhokhova**[a,2] **and Kumar Barua**[a,3]

[a]*Faculdade de Ciências da Universidade do Porto*

Abstract—This project aims to investigate how cloud computing services can be leveraged to develop applications that are scalable, high-performance, and highly available. Specifically, the study focuses on designing and implementing the backend architecture for a hospital management system, incorporating functionalities similar to electronic health record (EHR) databases, such as those found in the MIMIC-III dataset with the help of Google Cloud services. The system will facilitate hospital data management, enable patient registration, and support a secure doctor-patient communication mechanism. Patients will have the ability to submit medical inquiries, which can only be addressed by the physician responsible for their treatment, with each inquiry limited to a single response. Additionally, the project includes the development of companion testing scripts to ensure the reliability and robustness of the system.

Keywords—*GCP, hospital, management, databases*

Application link: Google Cloud
or go to the URL: https://bigquery-bdcc-main-453816.ew.r.appspot.com/login

1. Introduction

Cloud computing has garnered widespread adoption in recent years, primarily due to the increasing volume of data generated across various sectors, including science, government, and modern Internet systems. This data proliferation—often referred to as the "data deluge"—has created significant challenges in managing and processing vast amounts of information. Cloud computing offers an efficient and scalable solution for addressing these challenges by enabling the storage, processing, and management of large datasets in a cost-effective manner [4].

At its core, cloud computing enables users to access a range of computing resources, including servers, storage, databases, networking, software, and analytics, via the Internet. This model obviates the need for reliance on local hardware infrastructure, providing businesses and individuals with access to powerful computing resources without the financial burden of maintaining on-premises equipment.

1.1. Cloud Computing Service Models

Cloud computing is typically classified into three primary service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These models are distinguished by the level of responsibility assumed by the cloud service provider versus the client.

In the IaaS model, the cloud service provider is responsible for managing the underlying infrastructure, which includes computing resources, storage, and networking [1]. The client, however, retains control over the operating systems, applications, and data. The PaaS model abstracts further infrastructure management by offering a pre-configured development environment, allowing clients to focus on application development without concern for the underlying hardware or middleware [4]. In this model, responsibilities are shared between the provider and the client. The SaaS model, which is the most user-centric, delivers fully managed software applications over the Internet. Here, the provider manages all aspects of infrastructure, platform, and application maintenance, while clients interact with the software through web interfaces or application programming interfaces (APIs) [4]. In this model, the provider exercises a high degree of control over the system, while clients are able to access and utilize cloud-based applications remotely. Here are some examples of the main components for each of the models [2]:

1. **Infrastructure as a Service (IaaS)**:
   - **Core Components**: Virtual Machines (VMs), storage disks, private virtual networks
   - **Public Providers**:
     - Amazon Elastic Compute Cloud (EC2)
     - Google Compute Engine
     - Microsoft Azure
   - **Private IaaS Solutions**:
     - OpenStack (open-source)
     - VMware vCloud
2. **Platform as a Service (PaaS)**:
   - **Core Components**: Development frameworks, runtime environments, databases, middleware
   - **Public Providers**:
     - Google App Engine
     - Microsoft Azure App Service
     - Heroku
     - Red Hat OpenShift
3. **Software as a Service (SaaS)**:
   - **Core Components**: Software applications accessible via web browsers, multi-tenancy support
   - **Public Providers**:
     - Google Workspace (formerly G Suite)
     - Microsoft 365
     - Salesforce

– Dropbox

1.2. Hospital management system

A Hospital Management System (HMS) is a software solution—whether web-based, desktop-based, or mobile—that facilitates various hospital workflows. A well-designed HMS ensures the seamless operation of healthcare facilities by integrating administrative, medical, financial, legal, and patient management functions [3]. It plays a crucial role in enhancing efficiency and effectiveness within a hospital. An optimized hospital management workflow enables swift and informed decision-making that benefits both staff and patients. However, implementing such a system has become increasingly challenging due to the rising number of patients requiring critical care and the limited availability of medical facilities. As a result, a robust HMS is essential for maintaining high standards of patient care and operational efficiency. Leveraging cloud computing can address these challenges by ensuring system stability, scalability, and performance while streamlining hospital management operations. In this sense, it is required a well developed design and an implementation that tackles all this problems.

2. Design of the hospital management system

2.1. System Architecture

The design of the hospital system backend involves a multi-tier architecture deployed on the Google Cloud Platform (GCP) under the student program. The system's architecture comprises several key components interacting through REST APIs and cloud services.

2.1.1. Architecture Type

Multi-tier architecture.

2.1.2. Deployment Model

Public Cloud using GCP.

2.1.3. Components

- **Google App Engine:** Hosts the backend service implemented using Flask. It provides REST endpoints for managing various entities such as patients, admissions, progress, questions, and media uploads.
- **Google BigQuery (Database):** Stores structured data, including patient demographics, admissions, progress, and questions. Complex queries are executed using SQL, with parameterized queries used for security and efficiency.
- **Google Cloud Storage (Blob Storage):** Stores media files such as patient images and videos. Media files are uploaded, stored, and made publicly accessible via URLs when needed.
- **Google Cloud Functions (FaaS):** Planned for handling periodic tasks like garbage collection and updating waiting times to maintain efficiency and avoid data overload.

2.1.4. Component Interactions

- Users interact with the system through REST API requests handled by the backend service deployed on Google App Engine.
- Media files are uploaded to and retrieved from Google Cloud Storage, with URLs generated for public access when necessary.
- Google BigQuery handles structured data storage and retrieval, supporting complex querying for tasks such as fetching patient details, updating information, and managing questions related to patients.
- Asynchronous operations, such as updating waiting times and garbage collection, are intended to be managed by Google Cloud Functions.

2.1.5. Endpoints

- **Users:** Create, delete, update patients (`/rest/user`)
- **Media:** Upload, download media files (`/rest/media`)
- **Admissions:** Create and update medical events (`/rest/admissions`)
- **Progress:** Handle medical interventions or test results (`/rest/progress`)
- **Questions:** Create, reply, and list questions related to patients (`/rest/patients/id/question`)

This architecture ensures scalability, availability, and ease of access, rendering it suitable for the efficient management of hospital-related data in a cloud environment.

3. Implementation

The implementation of the hospital system backend focuses on utilizing Google Cloud Platform (GCP) services for creating a scalable and efficient solution. The implementation process includes setting up the backend service, integrating storage systems, handling structured data, and enabling asynchronous tasks.

3.1. Technology Stack

- **Backend Framework:** Flask (Python) deployed on Google App Engine.
- **Database:** Google BigQuery for structured data storage (Patients, Admissions, Progress, Questions).
- **Blob Storage:** Google Cloud Storage for storing patient-related images and videos.
- **Cloud Functions (Planned):** For periodic tasks such as updating waiting times and garbage collection.

3.2. Deployment Process

1. **Setting up App Engine:**
   - The backend service is built using Flask, allowing easy deployment to Google App Engine.
   - App Engine hosts the REST APIs responsible for handling patients, media uploads, admissions, progress, and questions.
   - The deployment process involves setting up an `app.yaml` configuration file specifying runtime, handlers, and application settings.
2. **Database Configuration (BigQuery):**
   - Google BigQuery is used for structured data storage, including Patients, Admissions, Progress, and Questions.
   - SQL queries are executed using the BigQuery Client Library for efficient data retrieval and updates.
   - Data is organized into tables and accessed through SQL queries using the `query()` method.
3. **Blob Storage (Google Cloud Storage):**
   - Patient-related media files are stored in Google Cloud Storage buckets.
   - Uploaded files are processed using the `storage.Client()` library and made publicly accessible via URLs if needed.
   - The storage process ensures security and availability of media files.
4. **Planned Use of Cloud Functions:**
   - Asynchronous tasks such as updating waiting times and garbage collection are intended to be handled using Google Cloud Functions.
   - Cloud Functions will be triggered on a scheduled basis or by specific events.
5. **Authentication:**
   - The project requires the integration of an authentication mechanism for secure access, ensuring that patients and doctors have different access privileges and functionalities.

The implementation ensures efficient interaction between the backend, storage, and database, providing a reliable and scalable solution for managing hospital-related data. Further enhancements, such as authentication and Cloud Functions, will contribute to improving system security and functionality.

4. Database Design

The database design for the hospital system backend is structured to efficiently store and retrieve data related to patients, admissions, progress, and questions. Google BigQuery is used as the primary database system, providing high-performance querying and scalability.

4.1. Database Structure

The system uses a relational-style structure implemented through multiple tables, each serving a specific purpose:

1. **Patients Table**
   - Stores demographic details of patients.
   - Fields:
     - subject_id: Unique identifier for each patient (Primary Key).
     - gender: Gender of the patient.
     - dob: Date of birth of the patient.
     - identity_doc: URL to the patient's identity document or image (stored in Google Cloud Storage).
2. **Admissions Table**
   - Represents hospital visits or admissions.
   - Fields:
     - subject_id: Reference to the patient who owns the admission (Foreign Key).
     - hadm_id: Unique identifier for each admission (Primary Key).
     - admittime: Timestamp of the admission.
     - deathtime: Timestamp of death, if applicable.
     - admission_type, admission_location, discharge_location: Related metadata.
     - insurance, language, religion, marital_status, ethnicity: Demographic data related to the admission.
     - diagnosis: Reason for admission.
     - hospital_expire_flag: Indicator of whether the patient expired during the hospital stay.
3. **Progress Table**
   - Tracks patient progress, interventions, or test results.
   - Fields:
     - subject_id: Reference to the related patient (Foreign Key).
     - progress_id: Unique identifier for each progress entry (Primary Key).
     - description: Description of the intervention or result.
     - timestamp: Time when the progress entry was created.
4. **Questions Table**
   - Stores questions posed by patients and their replies.

- Fields:
  - id: Unique identifier for each question (Primary Key).
  - subject_id: Reference to the related patient (Foreign Key).
  - hadm_id: Reference to the related admission (Foreign Key).
  - diagnosis: The diagnosis associated with the question.
  - question: The question text.
  - answer: The reply to the question, if available.
  - created_at: Timestamp of when the question was created.

4.2. Relationships

- The Patients table is linked to the Admissions, Progress, and Questions tables through the subject_id field.
- The Admissions table is related to the Questions table through the hadm_id field, ensuring questions are associated with specific admissions.
- Data is retrieved through parameterized SQL queries to enhance security and efficiency.

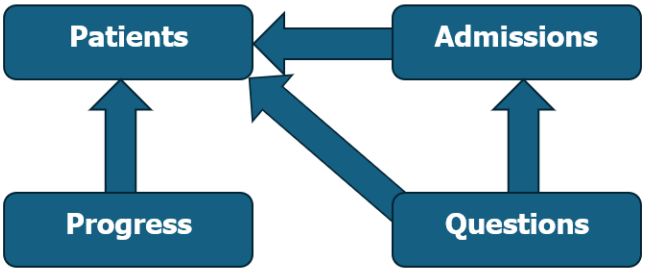

Figure 1. ER Diagram for Hospital Management System

This structure ensures efficient querying and management of data while maintaining data integrity and coherence. The database design provides a scalable approach capable of handling large amounts of structured data while integrating well with other GCP services.

5. Conclusion

The development and implementation of a hospital management system utilizing Google Cloud Platform (GCP) exemplifies the transformative capacity of cloud computing in contemporary healthcare infrastructure. The integration of essential GCP services, such as Google App Engine, BigQuery, Cloud Storage, and Cloud Functions, ensures scalability, high availability, and operational efficiency in the management of hospital data. The architectural design fosters secure and structured interactions between healthcare professionals and patients, thereby safeguarding the integrity and confidentiality of medical records. The incorporation of structured databases and cloud-based media storage enables seamless patient data management, thereby enhancing decision-making processes and improving the overall quality of patient care. The adoption of REST APIs further strengthens accessibility and interoperability, allowing for future system enhancements and integration with additional healthcare services. While the current implementation successfully addresses fundamental requirements, future refinements—such as the incorporation of advanced authentication mechanisms, the enhancement of security protocols, and the automation of asynchronous tasks through cloud functions—will further optimize system performance. This study underscores the pivotal function of cloud-based solutions in the advancement of hospital management systems, proffering a scalable, secure, and efficient framework that can meaningfully contribute to the evolution of digital healthcare.

References

[1] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing: Principles and Paradigms*. Wiley, 2011, ISBN: 978-0-470-88799-8.

[2] P. Mell and T. Grance, "The nist definition of cloud computing", National Institute of Standards and Technology, Tech. Rep. NIST Special Publication 800-145, 2011. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[3] E. J. Okoromi, *Hospital management system*, Lagos, Nigeria, 2021. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1852404/FULLTEXT01.pdf.

[4] Google Cloud. "What is cloud computing?" Accessed: 2025-03-15. (2024), [Online]. Available: https://cloud.google.com/learn/what-is-cloud-computing.

Application link: Google Cloud
or go to the URL: https://bigquery-bdcc-main-453816.ew.r.appspot.com/login