Group 16 - Distributed Electronic Health Record (EHR) System

Dang Chau Nguyen, Mikhail Bichagov, Piyumi Weebadu Arachchige

***Abstract*—Electronic Health Records (EHR) are essential for contemporary healthcare, but smooth patient care is hampered by data fragmentation throughout hospitals. The goal of this project is to create a distributed, secure EHR system that allows patient data to be shared across several healthcare facilities while maintaining interoperability, privacy, and dependability. The solution preserves data integrity across institutions by utilizing distributed consensus mechanisms (Raft/Paxos), microservice architecture, and OpenEMR [1]. Secure inter-hospital communication, real-time distributed machine learning (ML) to predict patient risk, and effective data replication techniques are some of the major advances.**

The technical design uses Terraform for infrastructure as code (IaC), Python (FastAPI) for operational chores, and LAMP stack (Linux with PHP (Symfony) for business logic. With the help of Docker and Kubernetes for containerization and orchestration, the system is set up on Google Cloud Platform (GCP), Digital Ocean and CSC. Redis manages caching and session management, Cassandra stores system communications, and MariaDB is used for structured patient records. RESTful APIs are used to provide secure communication across hospitals, while Apache Kafka is used to stream messages. Grafana and Prometheus logging, access control, and data encryption all strengthen security and compliance. The distributed machine learning (ML) service allows the system to predict patient risks in real time.

Final Demo:

***Index Terms*—Electronic Health Records, distributed, Synchronized, Backup, Machine Learning**

I. Problem Statement

Data barriers caused by the lack of interoperability of healthcare systems limit hospitals' ability to share patient records effectively. The requirement for consistency in hospital environment, such as medical histories, is important because any inconsistency will cause catastrophic consequences. Scalability and consistency are the opposites of each other: strict consistency enforcement will mean difficulty in scalability. When implemented in several institutions, current centralized EHR systems face issues with scalability. For real-time, safe, and standardized data transmission between healthcare providers, a distributed, privacy-preserving strategy is essential.

By creating a secure, distributed EHR system with cloud-edge integration, distributed machine learning, microservices, and consensus methods, this project seeks to address these issues. Healthcare professionals will have immediate access to correct patient information thanks to its fault-tolerant data replication, secure interhospital communication, and real-time analytics, all of which will preserve data privacy, compliance, and operational effectiveness.

We aim to achieve the following.

- Patients often receive care in several different hospitals, but their medical records are kept separate. Reducing duplicate testing and misdiagnosis, a distributed EHR system would guarantee smooth access to current medical records across institutions.
- All hospitals must maintain uniform and synchronized medical records. This system will ensure dependability by preventing data loss, disputes, and inconsistencies through the use of replication techniques and distributed consensus (Raft/Paxos).
- High workloads cause performance constraints in traditional centralized EHR systems. Hospitals will be able to manage significant increases in patient data while maintaining low-latency access with a cloud-edge hybrid infrastructure.
- Data resilience against malfunctions, cyberattacks, or natural catastrophes is ensured by a distributed design with redundant storage, preventing crucial data loss and guaranteeing continuity of care.

II. System Design

The figure 1 shows the connectivity between different nodes and the technologies that has been used.

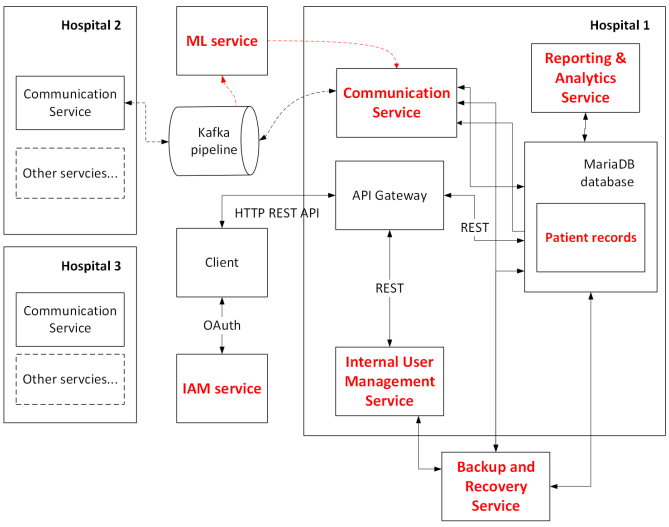

Fig. 1. System Design

A. IAM service

IAM service (Identity and Access Management) is responsible for managing and controlling user access to the resources

by authentication and authorization. IAM is the bottleneck of many systems because it is one of the most frequent service. In our design, we solve this problem by a hybrid approach: a 3rd party provider such as GCP or Azure IAM, and a local Active Directory (AD) to serve as a backup system in case of connection failure. By using protocols that utilize JSON Web Tokens - JWT, such as OAuth2, we effectively decoupled the authentication part between multiple services and IAM service, further reducing the workload and bottleneck of IAM service.

B. Communication service

The communication service is responsible for communication operations, such as communicating the synchronization of database to other hospitals. Communication services are briefly explained further under implementation part. This allows to have a recovery service / system in case original database becomes corrupted.

C. API Gateway

API Gateway is responsible for unifying all API calls and routing them to their respective services.

D. Backup and Recovery Service

To keep the backup operation efficient, the backup processes are designed to run directly on the service that needs backup. Then each service, based on its scaling strategy, will decide the best possible strategy to run the backup. For example, services that use kubernetes and auto scaling groups will just distribute the backup operation to any of its nodes. For SQL services, it will run on the SQL secondary replica so that it will not cause any workload on the primary node. Then the backup are sent to the backup and recovery service for long-term storage.

E. ML service

ML service is responsible for assisting professionals by using patient health record data to predict any risk and other information that professionals need to consult. ML service is designed to be independent from the main health record system. ML service will extract patient data from EHR and then store their data into its own database system. The result will not be posted back directly to EHR, but will be delivered via API calls when required.

Because of that, ML service do not need its database to be maximum consistency - Atomicity, Consistency, Isolation, and Durability - ACID, we can switch to Basically available, Soft state, eventually consistent - BASE database models that support horizontal scaling naturally.

III. Technology and Implementation

A. Infrastructure and Kubernetes Deployment

The Distributed Electronic Health Record (EHR) System services are intended to be scaled horizontally in an efficient manner. Therefore, containerization solution must be utilized. Specifically, we are using Kubernetes in Docker - KIND. Kubernetes - K8s handles deployment, scaling, distribution of node workloads, as well as handling failures such as stale/bad pods, redistribute pods when a node go down. Docker handles containers and emulation. Machine learning services for predictive healthcare analytics, OpenEMR for electronic health record management, and Kafka for real-time streaming all use K8s for deployment.

B. Monitoring and Logging

Visibility into both technical (such as system logs, error rates, and resource utilization) and business intelligence (such as patient outcomes and hospital performance) data is guaranteed by the Monitoring & Reporting Service. Using Grafana dashboards and Prometheus metrics collection, this service compiles and evaluates data from many sources to deliver real-time insights. Technical data also are used for K8s auto scaling rule, so when resource ultilization is high, the auto scaling engine will request the Cloud platform API to request more resources.

C. Database replication

To scale the database horizontally between clinics, we picked Kafka-connect for database scaling and replication. MariaDB by default does not have scaling included, and most of the other available solutions are commercial. We can utilize Kafka-connect, which is inherently distributed and handles scaling automatically without additional user intervention. It is fault tolerance: if one of the nodes shuts down, other nodes take over the responsibility. It has prebuilt connectors for major database systems. MariaDB being one of them. Kafka-connect has a significant amount of documentation available online. We will be able to use many connectors that are available for data synchronization. For example, JDBC and Debezium.

The process of our database replication method follows these steps, which can be visualized by Figure 2:

1) Debezium monitors and captures the changes from MariaDB. The changes are streamed into Kafka topics. It captures only latest changes in database compared to the previous state.
2) Kafka-connect serves as a bridge between source and replica database. This is the place where the topics and change logs are stored.
3) ZooKeeper maintains metadata and coordinates Kafka brokers.
4) JDBC is an API that is used for database connection and query execution. It monitors defined topics according to its configuration and captures only latest changes. Those changes are then streamed into the replicated database.

D. ML service

ML service has multiple strategies to receive data from the EHR system. One strategy is to pull the update from API Gateway. The pros of doing so is the increased flexibility in methods for getting the data, i.e. we will be able to filter out non-consent data transfer patient, and the de-identification process can happen inside EHR system. The cons of that method is the inefficiency of data transferring by REST APIs

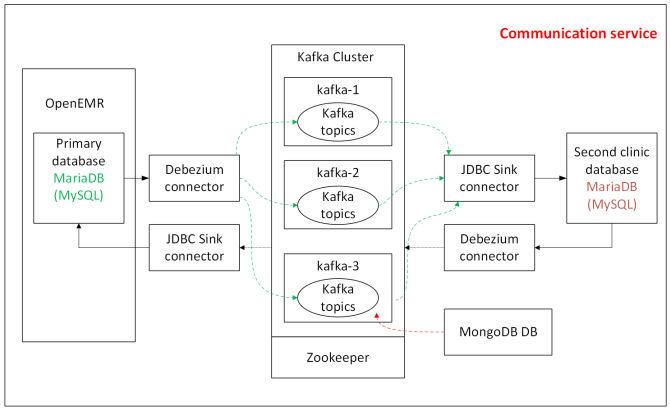

Fig. 2. Kafka Pipeline

call. We used the approach of sending the data directly to ML service by hooking into the Kafka service, which has already been used to synchronize the database, and very efficient as Kafka can synchronize Big Data. In this approach, under practical real-life use, the clinic must ensure the de-identification of user data and the removal of non-consent user happen in the ML service.

To post updates to patient data, a specific Kafka topic called patient_updates is created. As a Kafka consumer, the ML service is always on the lookout for messages and making predictions. Following that, the prediction outcomes are kept in a distinct MongoDB collection called predictions.

This division between the OpenEMR platform and the ML model guarantees the safe processing of private patient information while maintaining system adaptability.

The following format are present in each Kafka message:

Listing 1. JSON format of Kafka message

```
{
  "uuid": ...,
  "chest_pain_type": ...,
  "max_heart_rate_achieved": ...,
  "thal": ...
}
```

ML Service Workflow:

- Updates to patient data are published to the 'patient_updates' topic by the OpenEMR Kafka producer.
- The ML service's Kafka consumer, which is implemented in 'model.py', keeps listening for fresh messages.
- Prior to being input into the model, the received patient attributes undergo extraction, processing, and transformation.
- Heart disease is predicted by the **Logistic Regression** model.
- For later examination, the predicted outcome (heart_disease_present) is saved in MongoDB (predictions collection).

Only these features are being used to train the ML model at this point. To improve prediction accuracy, the team intends to add more clinical parameters to the model during the production phase.

For the testing sake, the default value of prediction has been turned into '1' as in figure I and after executing the ML service, the prediction turns '0' as in figure II. The log indicates details as in figure 3 when the model is successfully executed.

TABLE I
BEFORE ML PREDICTION

thal	chest pain type	max heart rate achieved	heart disease present
reversible defect	4	181	1
normal	3	158	1
normal	2	170	1
reversible defect	4	200	1

TABLE II
AFTER ML PREDICTION

thal	chest pain type	max heart rate achieved	heart disease present
reversible defect	4	181	0
normal	3	158	0
normal	2	170	0
reversible defect	4	200	1

```
ml-service_1  | Successfully inserted prediction for patient 9e1bf73e-4c04-4c64-aa7c-f3b625af9052!
ml-service_1  | Successfully inserted prediction for patient 9e4adb2a-fc4f-45cb-bf29-efa202cf3318!
ml-service_1  | Successfully inserted prediction for patient 9e4bd4fb-0f12-4bb6-99d9-05fc3d1f32b2!
ml-service_1  | Successfully inserted prediction for patient 9e4bd4fb-1781-465d-a70a-1593f1855227!
heart-disease-prediction-openemr_ml-service_1 exited with code 0
```

Fig. 3. ML prediction

IV. RESULTS

Any quantitative results

V. CONCLUSION

The conclusion goes here.

VI. CHALLENGES & FUTURE WORKS

The reason we used KIND at the beginning is because a lot of project starting points was Docker based, most of our team member are more familiar with Docker and K8s appear to be very challenging, so KIND was the smooth transition between Docker and K8s before we can move on to other technologies. We would like to move away from KIND to more robust and lightweight solutions, such as k3s. However, as we soon faces challenges with Kubernetes configuration and Database synchronization, we were unable to move further.

Another aspect is that we would like to work if more time was given is the high speed VPN connection (Cloud VPN) between GCP and on-premise emulation to simulate a redundant system within the clinic that provides degraded access (staff only, read only) in case of catastrophic system failure.

In terms of the Database system and Kafka, we went through a lot of trouble with integrating Kafka with MariaDB. Thanks to the help of the course organizer team, we had been able to go through a lot of trouble, but we still get some technical challenges at the very end of the project. Once we add message from Kafka back to Database, the Database immediately lost track of the origin of the data, and due to the nature of database listener, the committed data get sync back to the Kafka pipelines, causing it to re-emit old event. We were aware of the Schema history topic [2] but we weren't able to config it to work on time. Or the Schema history topic only apply to sync events, while this newly duplicated events will not be distinguished by the system as it is new rather than "relatively old" according to the documentation. Even more, the Schema history topic only available for internal use, so if we want to avoid duplicated events echoing back to the Kafka pipelines, we have to reinvent a similar history topic system, but only for newly committed messages.

For the ML service, we are still limited in theoretical training data and process, with simplified version of data pre-processing. In real life scenarios, we recommend using several processing steps that can process the data in parallel to improve the efficiency and speed of putting data ingression from the main EHR database to refined ML database for ML service.

Role of AI tools in the project

- Consulting for technology and strategy, however, the AI tools have been hallucinating a lot, combined with the constant advertising commercial solutions that are "free" and "widely available", or part of its commercial platform, that caused the AI models to recommend enterprise solutions and misleading our team member, sometimes even interfering (suggest enterprise solution) when the implementation is not working.
- Implementation assistance (installation, configuration, and coding). However, it was not perfect ,and many of the implementation steps have to still be done in traditional methods such as Google search, reading Stackoverflow answers, and reading documentation.
- Assisting in the report writing.

Division of Labour among group members

Chau Nguyen:

- Project management (Team & Tech lead)
- Architecture
- Implement Kubernetes and infrastructure systems
- Code review
- Report & presentation refinement

Mikhail Bichagov:

- Research and implement the synchronization process of EHR database system
- Report writing

Piyumi Weebadu Arachchige:

- Working on ML model
- Research and working on using BASE-databases for ML service
- Working on transferring data from Kafka to ML service
- Working on presentations
- Report writing

Learning Dairy

Chau Nguyen:

- The challenges of scaling and synchronizing ACID databases horizontally without sharding and partitioning. After facing these challenges, I realized the reason why there are so few open-source options are available, and those available community options have a lot of restrictions in place, due to the sheer amount of work in creating them, and maintenance.
- The specific details of distributing data in data streaming solutions. I have used commercial Pub/Sub and other event-message distributing system solutions without understanding the underlying algorithm inside them.
- I understand the importance of synchronizing data strategies and algorithms that need to be applied in order to keep data integrity and resolve once a conflict occurs in the distribution system due to delay between clients.

Mikhail Bichagov:
During the project I have learned:

- Using a remote host / virtual machine for hosting applications.
- How to host application using docker and operate it.
- Kafka-connect, Zookeper, JDBC, and Debezium for database replication

For some it might seem like a long list but coming from a Data Science background, that is a significant amount of new technology.

Piyumi Weebadu Arachchige: I was able to get hands-on experience in fields that I only had basic knowledge.

- ML model creation with Logistic Regression.
- Integrating services into Docker
- Log analysis of Docker containers
- Integrating services into Kafka
- Using MongoDB
- Integrating Grafana and Prometheus
- Drawing diagrams in MS Visio

Time Spent

Chau Nguyen	Approximate 220 hours
Mikhail Bichagov	Approximate 220 hours.
Piyumi Weebadu Arachchige	Approximate 210 hours

References

[1] "The world's leading open-source¡br¿medical record software. — open-emr.org," https://www.open-emr.org/, [Accessed 10-03-2025].

[2] "Debezium connector for MariaDB :: Debezium Documentation — debezium.io," https://debezium.io/documentation/reference/stable/connectors/mariadb.htmlmariadb-schema-history-topic, [Accessed 10-03-2025].