Cloud Research – Implementing Scientific Research Information Systems for Health Data in Open Source Cloud Platforms

Submitted In Partial Fulfillment
Of Requirements For the Degree Of
Third Year
Computer Engineering

Cloud Computing(Elective)

by

Smit Patil
Roll no.-16010122139
Toyash Patil
Roll no.-16010122140
Khushi Poojary
Roll no.-16010122147

Guide
Mr. Zaheed Shaikh

INTRODUCTION

Cloud computing provides on-demand access to a wide array of computing resources—including storage, servers, and applications—over the internet. Its scalable and dynamic nature makes it particularly suited for handling large, complex datasets. However, this flexibility also introduces challenges in areas like resource allocation, performance optimization, and privacy compliance.

In the healthcare sector, the rapid growth of digital health records and real-time monitoring data has created an urgent need for robust, scalable, and privacy-compliant data analytics platforms. Traditional systems often fail to support reproducible research, struggle with scalability, or lack integration for diverse data types such as electronic health records (EHRs) and waveform data.

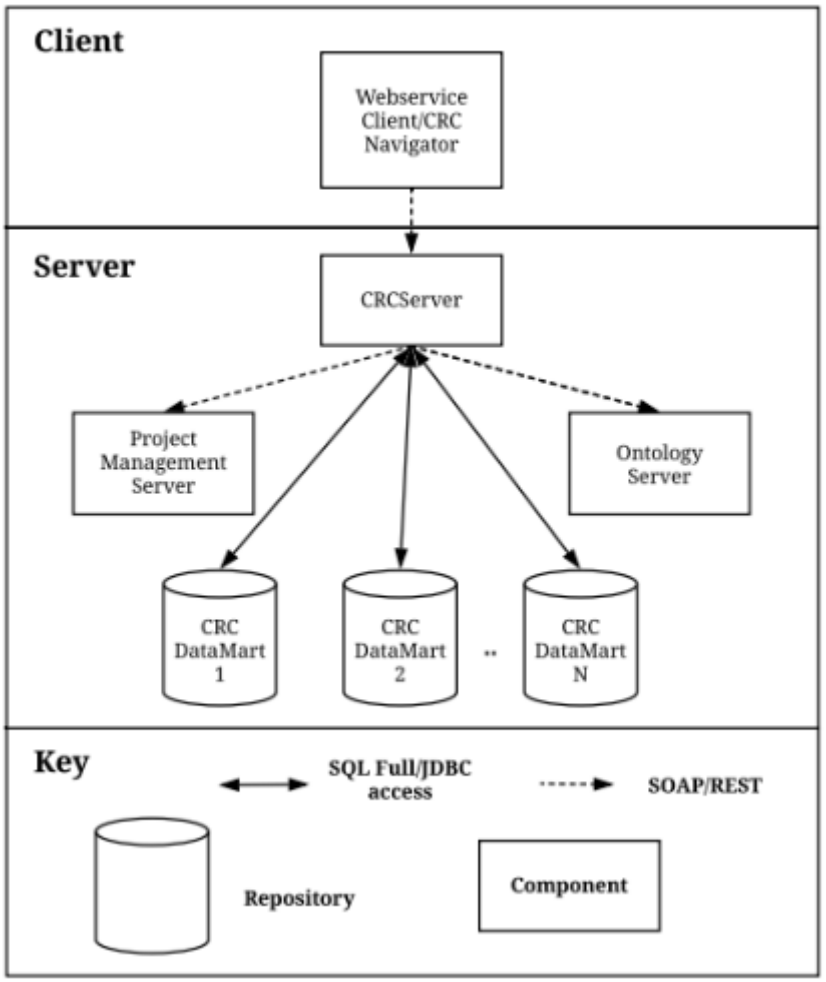
This study presents a cloud-native platform designed specifically to address these gaps. It combines cutting-edge technologies—such as Apache Hadoop, Kubernetes, and JupyterHub—to build a secure, scalable, and user-friendly environment for clinical data analysis. The system is optimized for performance, supports collaboration, and adheres to regulatory standards like HIPAA, making it an ideal solution for modern, data-driven healthcare research.

MOTIVATION

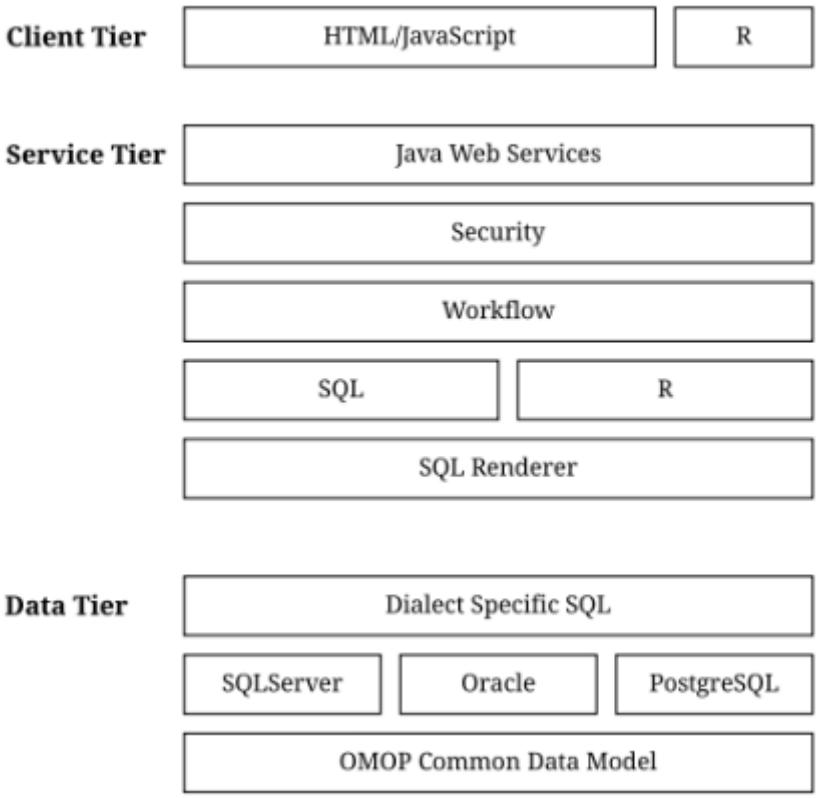
The rapid adoption of cloud computing in healthcare and other data-intensive sectors has highlighted the pressing need for systems that are both highly performant and resource-efficient. As clinical data continues to grow in complexity and volume—ranging from structured electronic health records to unstructured waveform signals—traditional analytics infrastructures struggle to keep up.

Unbalanced resource usage in such cloud environments can lead to system inefficiencies, longer response times, and compromised data accessibility. In mission-critical domains like healthcare, these inefficiencies can delay insights and impact decision-making. Therefore, there is a strong need to design cloud-native platforms that not only scale efficiently but also distribute computational loads intelligently.

This project is driven by the need to evaluate and implement effective load balancing strategies within a privacy-compliant clinical data analytics platform. By exploring the behavior of various load balancing algorithms under realistic workloads, we aim to enhance system responsiveness, ensure high availability, and support seamless, collaborative clinical research across institutions.


Client
Webservice Client/CRC Navigator
Server
CRCServer
Project Management Server
Ontology Server
CRC DataMart 1
CRC DataMart 2
CRC DataMart N
Key
SQL Full/JDBC access
SOAP/REST
Repository
Component

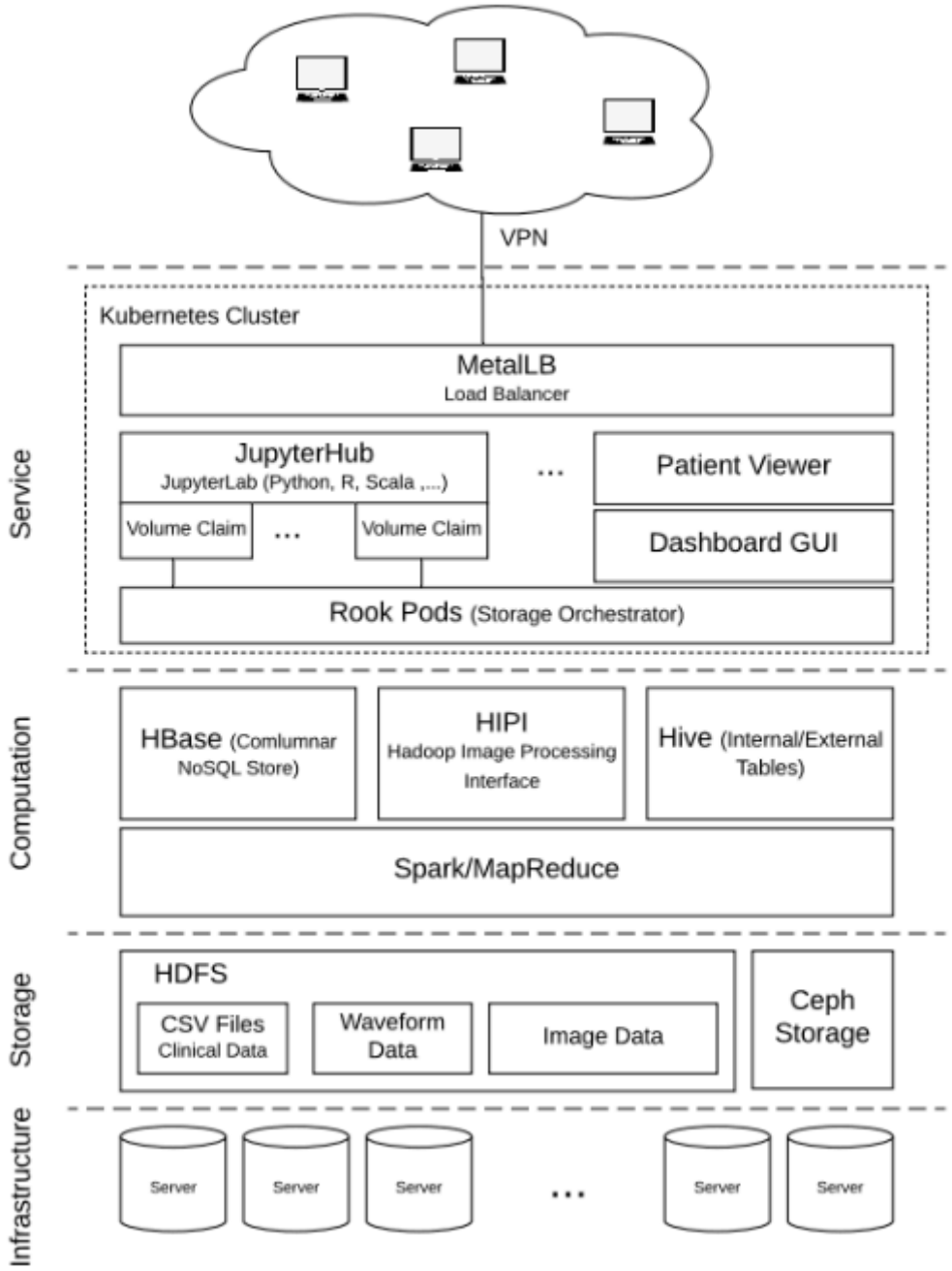
OHDSI Layered Architecture:



OBJECTIVES

The primary objectives of this study are:

- To design and deploy a cloud-native analytics platform tailored for privacy-compliant clinical data processing.
- To integrate scalable open-source technologies like Apache Hadoop, Kubernetes, and JupyterHub for effective data management and computation.
- To evaluate system performance under different configurations, focusing on metrics such as:
    - Query Response Time
    - Scalability and Elasticity
    - Data Throughput
    - Resource Utilization and Cost Efficiency
- To simulate and assess the impact of load distribution strategies in real-world, multi-modal healthcare workloads.
- To identify the most efficient system architecture and configuration suitable for large-scale, geographically distributed cloud infrastructures supporting healthcare research.



METHODOLOGY

The methodology follows a **step-by-step process** to demonstrate the core idea of a collaborative cloud computing framework for health data using **Docker and JupyterLab**, with a **sample patient dataset**.

Steps Followed:

1) Environment Setup using Docker

- Docker is installed on the local system to simulate a containerized cloud environment.
- The official jupyter/base-notebook Docker image is pulled, which includes Python and JupyterLab preconfigured.

```
Unset
docker pull jupyter/base-notebook
```

2) Running a JupyterLab Container

- A Docker container is launched, exposing port 8888 to access JupyterLab.
- The local directory (D:/health_demo) is mounted to /home/jovyan/work inside the container to enable file sharing.

```
Unset
docker run -d -p 8888:8888 -v
"D:/health_demo:/home/jovyan/work" --name health-data-lab
jupyter/base-notebook
```

3) Preparing Health Data

- A small clinical dataset (patient_data.csv) is created with basic patient information: patient_id, age, gender, and diagnosis.
- This file is saved in the shared local folder (D:/health_demo), making it accessible inside the Jupyter notebook.

4) Accessing JupyterLab

- JupyterLab is accessed through a web browser at http://localhost:8888/.
- If required, the access token is retrieved from container logs.

```
Unset
docker logs health-data-lab
```

5) Data Analysis Workflow

- A new Python notebook is created inside JupyterLab.
- Python libraries pandas and matplotlib are installed to perform data operations and visualizations.

```
Unset
!pip install pandas matplotlib
```

- The dataset is read and displayed using pandas.
- A bar chart is generated showing the count of each diagnosis using matplotlib.

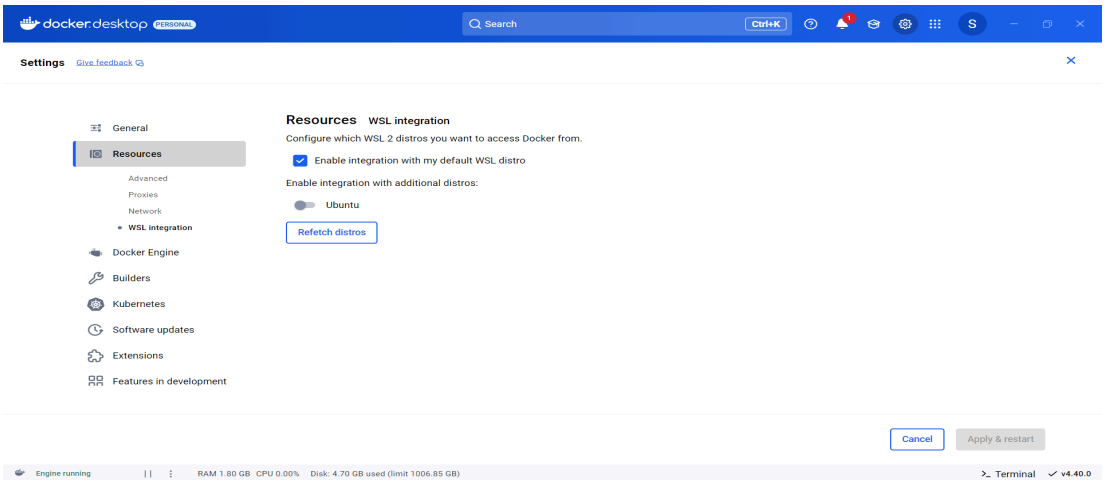
```
Unset
import pandas as pd

import matplotlib.pyplot as plt

df = pd.read_csv('work/patient_data.csv')

print(df)

df['diagnosis'].value_counts().plot(kind='bar')

plt.title('Patient Diagnoses')

plt.show()
```

6) Interpreting Results

- The notebook displays a summary of patient diagnoses.
- Visualization helps in quickly understanding common conditions in the dataset.
- The entire analysis runs inside a **container**, demonstrating the framework's **reproducibility** and **portability**.

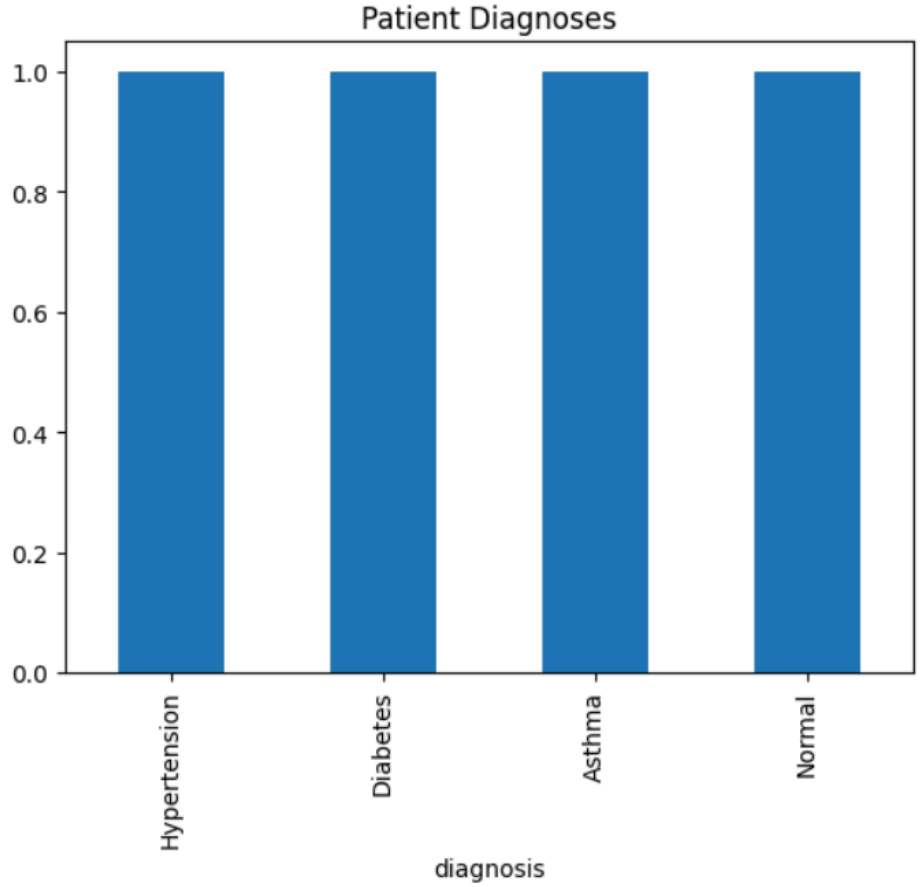
IMPLEMENTATION SCREENSHOTS

```
Setting up containerd (1.7.24-0ubuntu1~24.04.2) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.servi
ce → /usr/lib/systemd/system/containerd.service.
Setting up libnftables1:amd64 (1.0.9-1build1) ...
Setting up nftables (1.0.9-1build1) ...
Setting up libnetfilter-conntrack3:amd64 (1.0.9-6build1) ...
Setting up iptables (1.8.10-3ubuntu2) ...
update-alternatives: using /usr/sbin/iptables-legacy to provide /usr/sbin/ip
tables (iptables) in auto mode
update-alternatives: using /usr/sbin/ip6tables-legacy to provide /usr/sbin/i
p6tables (ip6tables) in auto mode
update-alternatives: using /usr/sbin/iptables-nft to provide /usr/sbin/iptab
les (iptables) in auto mode
update-alternatives: using /usr/sbin/ip6tables-nft to provide /usr/sbin/ip6t
ables (ip6tables) in auto mode
update-alternatives: using /usr/sbin/arptables-nft to provide /usr/sbin/arpt
ables (arptables) in auto mode
update-alternatives: using /usr/sbin/ebtables-nft to provide /usr/sbin/ebtab
les (ebtables) in auto mode
Setting up docker.io (26.1.3-0ubuntu1~24.04.1) ...
info: Selecting GID from range 100 to 999 ...
info: Adding group `docker' (GID 108) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service →
 /usr/lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /us
r/lib/systemd/system/docker.socket.
Setting up dnsmasq-base (2.90-2build2) ...
Setting up ubuntu-fan (0.12.16) ...
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.servi
ce → /usr/lib/systemd/system/ubuntu-fan.service.
Processing triggers for man-db (2.12.0-4build2) ...
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
smitp@SmitSaanj:~$ sudo systemctl start docker
smitp@SmitSaanj:~$ sudo systemctl enable docker
smitp@SmitSaanj:~$ docker --version
Docker version 26.1.3, build 26.1.3-0ubuntu1~24.04.1
smitp@SmitSaanj:~$
```




C:\Windows\System32\cmd.e
[I 2025-04-16 16:42:11.909 LabApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 2025-04-16 16:42:11.910 LabApp] Extension Manager is 'pypi'.
[I 2025-04-16 16:42:11.914 ServerApp] jupyterlab | extension was successfully loaded.
[I 2025-04-16 16:42:11.922 ServerApp] nbclassic | extension was successfully loaded.
[I 2025-04-16 16:42:11.926 ServerApp] notebook | extension was successfully loaded.
[I 2025-04-16 16:42:11.926 ServerApp] Serving notebooks from local directory: /home/jovyan
[I 2025-04-16 16:42:11.926 ServerApp] Jupyter Server 2.8.0 is running at:
[I 2025-04-16 16:42:11.926 ServerApp] http://110eb42e135f:8888/lab?token=d7f340c12fd10e6bc13ef68a01fa10b9631b2550a33ccc06
[I 2025-04-16 16:42:11.926 ServerApp] http://127.0.0.1:8888/lab?token=d7f340c12fd10e6bc13ef68a01fa10b9631b2550a33ccc06
[I 2025-04-16 16:42:11.926 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2025-04-16 16:42:11.931 ServerApp]
To access the server, open this file in a browser:
file:///home/jovyan/.local/share/jupyter/runtime/jpserver-7-open.html
Or copy and paste one of these URLs:
http://110eb42e135f:8888/lab?token=d7f340c12fd10e6bc13ef68a01fa10b9631b2550a33ccc06
http://127.0.0.1:8888/lab?token=d7f340c12fd10e6bc13ef68a01fa10b9631b2550a33ccc06
[I 2025-04-16 16:42:12.880 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-nodejs, javascript-typescript-langserver, jedi-language-server, julia-language-server, pyright, python-language-server, python-lsp-server, r-languageserver, sql-language-server, texlab, typescript-language-server, unified-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
[I 2025-04-16 16:43:02.272 ServerApp] 302 GET / (@172.17.0.1) 0.63ms
[I 2025-04-16 16:43:02.281 LabApp] 302 GET /lab? (@172.17.0.1) 0.87ms
[W 2025-04-16 16:43:29.960 ServerApp] 401 POST /login?next=%2Flab%3F (@172.17.0.1) 2.52ms referer=http://localhost:8888/login?next=%2Flab%3F
D:\health_demo>

```
   patient_id  age gender     diagnosis
0           1   45      M  Hypertension
1           2   60      F      Diabetes
2           3   50      F        Asthma
3           4   30      M        Normal
```



FUTURE SCOPE

The evolution of cloud computing, combined with emerging technologies like 5G, IoT, and smart city infrastructure, points to a growing need for more advanced and intelligent load balancing solutions. Future research in this space could explore the following directions:

- **Integration of AI/ML Models:** Employ machine learning algorithms for predictive and autonomous load balancing that adapts to workload patterns and user behavior in real-time.
- **Dynamic Real-Time Load Management:** Develop systems capable of live decision-making and traffic redirection to optimize performance during peak demand.
- **Energy-Aware Load Balancing:** Optimize data center energy consumption through intelligent scheduling and resource distribution strategies.
- **Scalable Simulations:** Extend testing environments to include larger networks, hybrid cloud architectures, and edge computing ecosystems.
- **Context-Aware Decision Systems:** Incorporate real-time feedback, Service Level Agreements (SLAs), and current network conditions into the load balancing logic.
- **Autonomous and Self-Healing Systems:** Design adaptive, AI-driven load balancers that can self-optimize and recover from failures with minimal human intervention.

As cloud services continue to expand into every aspect of modern life, intelligent, adaptive, and sustainable load balancing mechanisms will be critical to ensuring seamless user experiences and system resilience.

CONCLUSION

This report presented the design and evaluation of a cloud-native analytics platform optimized for privacy-compliant clinical data processing. Through integration of open-source technologies such as Hadoop, Kubernetes, and JupyterHub, the platform demonstrates the potential for scalable, collaborative, and secure data analysis in healthcare.

Simulated performance testing highlighted the platform's ability to handle diverse workloads efficiently while maintaining system responsiveness and reliability. Among various resource management strategies, intelligent workload distribution significantly improved query response times and reduced operational overhead.

The results confirm that with proper load balancing and system configuration, cloud infrastructures can effectively support real-time, multi-user clinical research on a global scale. The platform also establishes a solid foundation for future enhancements involving AI-driven load balancing, edge integration, and energy-efficient optimization.

REFERENCES

1. Apache Hadoop. Available: https://hadoop.apache.org/

2. BD2K PIC-SURE RESTful API. [Online; Accessed: March 3, 2018]. Available: http://bd2k-picsure.hms.harvard.edu/more.html

3. Docker – Empowering App Development for Developers. Available: https://www.docker.com/

4. MetalLB: Bare Metal Load-Balancer for Kubernetes. Available: https://metallb.universe.tf/

5. CILogon: An Integrated Identity and Access Management Platform for Science. [Accessed: 2019]. Available: https://www.cilogon.org/