Big Data Pipeline Architecture for Healthcare Analytics

Course: CS 6500: Big Data Analytics

Team Members: Dipika Bogati, Radhika Srivastava, Sailesh Kafle and Sushant Nepal

Team Submission Date: 4/28/2025

Instructor: Dr. Robert Green

Contents

1. Executive Summary

HealthTrend Innovations requires a secure, scalable big data platform to process over 50 TB of healthcare data, including historical patient information, real-time IoT feeds, and medical image files.
Our Hadoop-based solution leverages HDFS, Kafka, NiFi, Spark Structured Streaming, and Airflow to ingest, process, and store structured, semi-structured, and unstructured data in a cost-effective way.

The system is tuned between real-time streaming and batch workloads, providing automated data ingestion, near-real-time processing, and expandable storage with HIPAA-compliant security best practices such as TLS/SSL encryption, HDFS access control, and NiFi provenance tracking for auditability.

- Apache Spark MLlib models are trained on purified datasets for analytics:
- Regression models identify healthcare trends and forecast resource needs.
- Logistic regression categorizes high health risk profiles.

Visualized insights allow predictive analytics, operational optimization,
and improved clinical decision-making.

Apache Airflow is used to manage workflows for automating scheduling and maintaining up-to-date visual dashboards.

This end-to-end solution enables HealthTrend Innovations to scale securely while delivering real-time, actionable healthcare insights.

2. Problem Statement & Requirements

The healthcare industry is experiencing an all-time high in data generation, with the increasing digitalization of patient information, utilization of IoT-enabled patient monitoring systems, and the exponential growth in medical imaging techniques.
For HealthTrend Innovations, analyzing and processing this enormous, complex, and sensitive healthcare data requires robust, scalable, and compliant big data infrastructure.

Healthcare data today possesses the 4Vs characteristics of Big Data:

Characteristic	Description
Volume	Over **50 terabytes** of data accumulated, including structured patient records (demographics, diagnoses), semi-structured logs from patient monitoring devices, and unstructured binary imaging data such as MRI and CT scans.
Velocity	High-speed real-time streaming of patient vitals at a rate of **1 GB/hour** from **10,000+ IoT sensors** deployed across hospitals and clinics.
Variety	Heterogeneous data formats encompassing **structured CSV files** for historical records, **semi-structured JSON streams** from IoT devices, and **unstructured binary files** for medical imaging.
Veracity	Data inconsistencies, including missing fields, duplicate entries, and potential inaccuracies, that must be addressed through robust preprocessing, validation, and compliance with regulatory frameworks like **HIPAA** (Health Insurance Portability and Accountability Act).

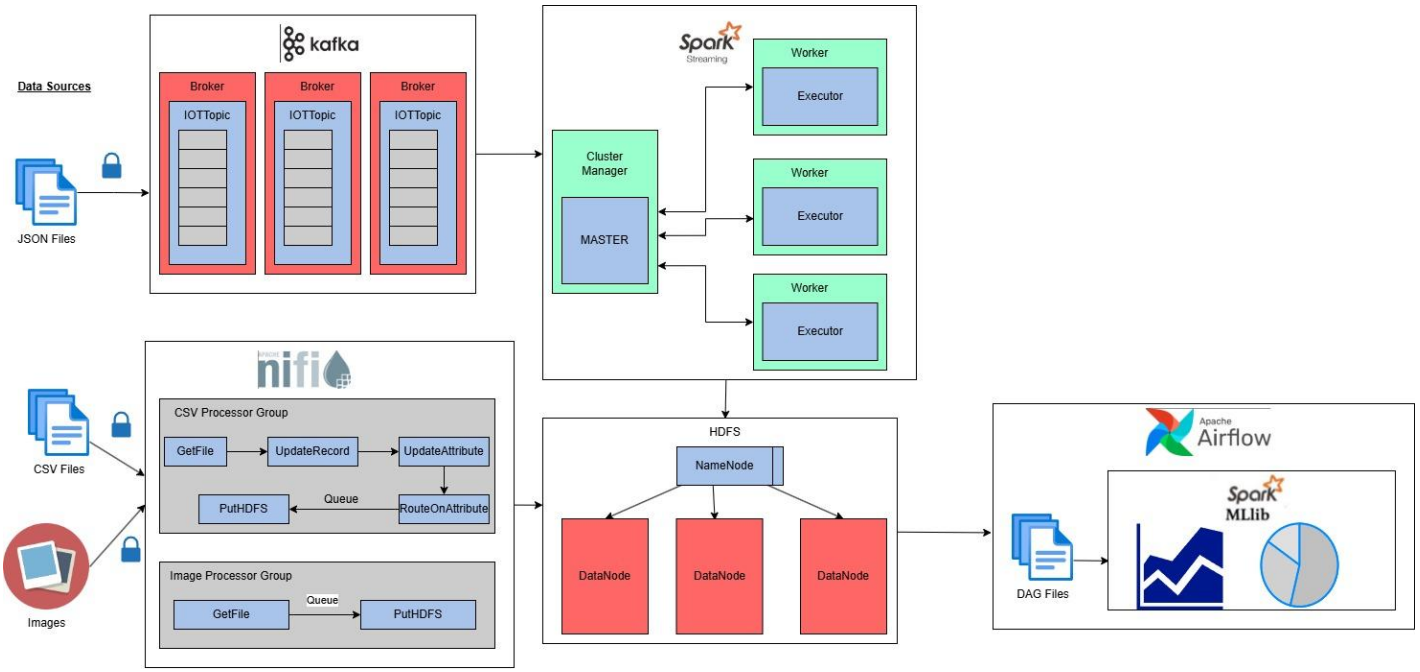
In consideration of such requirements, the planned system must be capable to:

- Secure Ingestion, Storage, and Processing:.
- Processing Both Real-Time and Batch Analytics:
- Providing Secure Access for Authorized Analysts and Applications.

The platform should not only be capable of handling the scale and complexity of today's healthcare data but also serve as a platform for advanced predictive modeling, clinical decision support, and operational optimization — enabling HealthTrend Innovations to deliver improved healthcare outcomes through data-driven strategie

Technical Details

3. Architecture Design



Overview

The above architecture diagram gives the overall data pipeline of the data flow inside the Health-Care Analytics. There are three different data sources (IOT devices for the real-time data, CSV files for historical data and binary files with image) from where the pipeline will ingest the data. Since there is real-time ingestion and batch processing ingestion, the pipeline used two different types of the ingestion; Apache Kafka for the real-time ingestion and Nifi for the batch-processing ingestion and the transformation and sending file to the HDFS. For the real-time processing, Apache Spark Streaming is used and a parquet file generated from the Spark Streaming after cleaning is stored in the HDFS. After that, Apache Airflow is used to schedule the job to run on the weekly basis where it schedules the python file running SparkMLLib to visualize the data.

Component Breakdown

I. **Kafka:** For the real-time ingestion, the data is fetched from the IOT devices in Json Format. So, the Producer for Kafka is IOT devices and there are three brokers as the replicator factor in our case is 3 and one topic (IOTStreamTopic) that is replicated in three brokers. In each topic, 6 partitions are done to handle 1 GB of data per/hour which means 277 KB/sec for each partition which ensures there is availability and the limited partition ensures there is no resource overhead.

II. **NiFi:** For the batch processing of the Binary files and the CSV files, NiFi is used because it has the backpressuring tool which can ingest the data without crashing. In the case of the binary files, NiFi could GetFiles from the source and put the file into the HDFS directly. In the case of the CSV files, Nifi performed the cleaning and transformation technique as well where UpdateRecord is used to fix the missing null value and set the data types, updateAttribute modifies the flow and helps in routing and finally the data is pushed into the HDFS.

III. **Spark Streaming:** For the transformation of the real-time data, spark streaming is used. In this case, the one spark master executes the three executors/workers. This is because data is coming from 6 partitions of the Kafka and one executor can process the data coming from the 2 partitions each so there would not be a core fight and no resource overhead at the same time.

IV. **HDFS:** For the storage of the files coming in from the Spark Streaming and NiFi, HDFS is used. In HDFS, there is one NameNode and a secondary NameNode to solve the single point failure. And three DataNodes of each 50 TB size where after one month of the data storage, it is moved to the warm and cold tier for the archive so there will be space/storage enough for real-time data.

V. **AirFlow:** Since the transformation and storage is done, Airflow is used to schedule the SparkMLLib Jobs in hourly manner so we can perform the visualization/monitor the data trends in the hourly manner.

VI. **SparMLLib:** To perform the proper analysis and visualization of the data, SparkMLLib is used where this program runs on the hourly manner and fetch the data from the HDFS and generates the intelligence that is used to solve the health problems.

4. Data Ingestion and storage

Our proposed data ingestion and storage pipeline is designed so that it can efficiently handle the three major categories of healthcare data :

- Structured historical patient records(CSV files)
- Semi-Structured real-time IOT data (JSON streams from 10,000 devices)
- Unstructured medical imaging files

We used Apache NiFi, Apache Kafka, Apache Spark Streaming and HDFS. This system we design ensures scalability, real-time, fault -tolerant and secure data movement and storage.

4.1 Batch Data Ingestion using Apache NiFi

Apache Nifi Flows can be configured to ingest both CSV and binary images and then store them into HDFS, preparing the data for future processing using hive and spark.

4.1.1 NiFi Flow Design

We setup two independent parallel fllows.

a. CSV Processor Group

Processor	**Description**	**Purpose**
GetFile	Continuously monitors `/data/input/csv/` for new patient CSV files.	Fetches batch CSV files from local directory into NiFi flow.
UpdateRecord	Parses CSV content, standardizes fields, and applies schema-based transformations	Cleans and formats patient data records as needed
UpdateAttribute	Adds or updates metadata attributes (e.g., ingestion timestamp, hospital ID).	Cleans and formats patient data records as needed.
RouteOnAttribute	Routes FlowFiles based on attribute values (e.g., missing fields, schema validation failures).	Handles error flows or routes validated records toward final storage
PutHDFS	Writes validated, enriched CSV FlowFiles into HDFS directory `/user/nifi/csv/`.	Persists cleaned CSV data into Hadoop storage for further analytics

4.1.2. Image Processor Group

Processor	Description	Purpose
GetFile	Continuously monitors `/data/input/images/` for new medical image files (e.g., PNG, DICOM).	Fetches binary medical imaging files into NiFi flow.
PutHDFS	Writes binary files directly into HDFS under `/user/nifi/images/`.	Persists medical images securely into HDFS for long-term storage and later referencing via metadata.

Each NiFi flow continuously monitors its own input directory. Instead of moving files as soon as they are found, NiFi moves files in controlled batches, respecting system-determined backpressure levels so as not to overwhelm downstream components. In this manner, ingestion is performed optimally, dynamically adapting to available system resources and delivering consistent throughput into HDFS.

CSV Flow:

GetFile (from /data/input/csv)
↓
UpdateRecord (process and modify CSV content)
↓
UpdateAttribute (add or modify FlowFile metadata)
↓
RouteOnAttribute (route FlowFiles based on conditions)
↓
PutHDFS (store into HDFS at /user/nifi/csv/)

Binary Flow:

GetFile (from /data/input/images) → PutHDFS (to /user/nifi/images)

The strength of using NiFi setup are:

- Scalable and fault-tolerant ingestion
- Real-time ingestion readiness(files moved off HDFS immediately).
- It supports **backpressure** and retry logic which is crucial stable ingestion of massive datasets
- Easy to integrate with HDFS for final storage
- NiFi maintains that from the time data gets into the ingestion pipeline to the time it drops into HDFS, it stays secure, auditable, and compliant with security and privacy controls.

4.2 Real-Time Streaming ingestion using Kafka

Apache kafka serves as the backbone for real-time data ingestion. The patient telemetry data is continuously generated by 10,000 IOT devices monitoring the patient's health parameters such as heart rate, oxygen levels, and blood pressure. This data is ingested into kafka and stored in a kafka topic called mytopic.

Kafka's role:

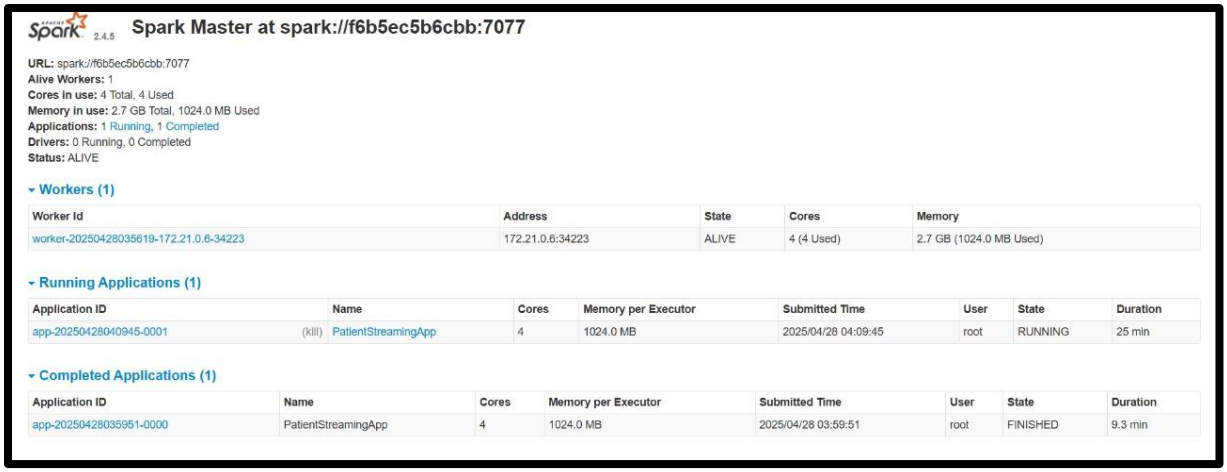
- Kafka acts as a buffer for incoming patient telemetry data, ensuring that high volume, high velocity data can be temporarily held in kafka even if spark streaming is unable to process it in real time. Kafka ensures that the data is safely retained in the topic for later processing.
- Kafka decouples the producers (IOT devices) from the consumers (spark streaming). This allows producers to generate data at high speed while consumers can process the data asynchronously.
- Kafka is designed to handle high throughput and offer fault tolerance by replicating data across multiple brokers. This ensures data durability, even if some brokers go down.
- Kafka topics can be partitioned to distribute the data across multiple consumers. This allows for parallel processing by spark streaming, ensuring that the system can scale horizontally and handle massive data streams from all 10,000 IOT devices efficiently.

Kafka provides data retention capabilities. Even if spark streaming is down or slow, kafka ensures that data is retained for processing once spark resumes its operation, guaranteeing no data loss.

5. Data Processing and Analysis

5.1 Spark Structured Streaming Processing

Once the data is ingested into kafka, Apache Spark Streaming is used to process the data in real time. This is where the bulk of data transformation and anomaly detection takes place. Spark's checkpointing and write-ahead logs ensure that data is not lost.



Spark Master at spark://f6b5ec5b6cbb:7077

URL: spark://f6b5ec5b6cbb:7077
Alive Workers: 1
Cores in use: 4 Total, 4 Used
Memory in use: 2.7 GB Total, 1024.0 MB Used
Applications: 1 Running, 1 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20250428035619-172.21.0.6-34223	172.21.0.6:34223	ALIVE	4 (4 Used)	2.7 GB (1024.0 MB Used)

Running Applications (1)

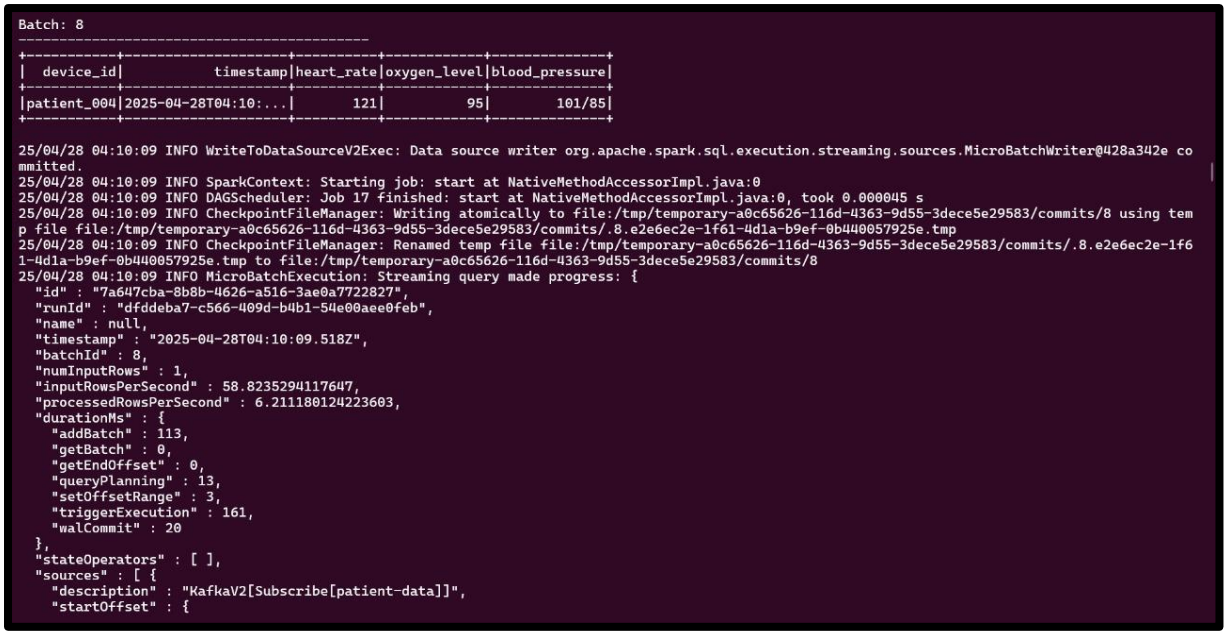
Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20250428040945-0001 (kill)	PatientStreamingApp	4	1024.0 MB	2025/04/28 04:09:45	root	RUNNING	25 min

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20250428035951-0000	PatientStreamingApp	4	1024.0 MB	2025/04/28 03:59:51	root	FINISHED	9.3 min

Apache Spark Streaming is used to:

- Spark Streaming reads data continuously from the kafka topic. Each incoming message in the kafka topic is processed as a micro batch, allowing spark to handle the data incrementally.
- Each incoming JSON message is parsed and validated. Spark uses its DataFrame API to parse raw JSON data and validate the structure. Fields like heart_rate, oxygen_level, and timestamp are checked for completeness and correctness.
- Spark performs basic data validation to ensure that all required fields are present and that the values fall within an acceptable range. If there are missing fields, duplicate records, or invalid data, it is either cleaned or flagged for further review.
- Spark can also detect real-time anomalies in the patient data, This real time analysis helps notify immediately when a patient's data shows signs of deterioration such as:
    - Heart rate spikes: e.g., heart rate exceeding 120 bpm.
    - Low oxygen levels: e.g., oxygen levels dropping below 90%.



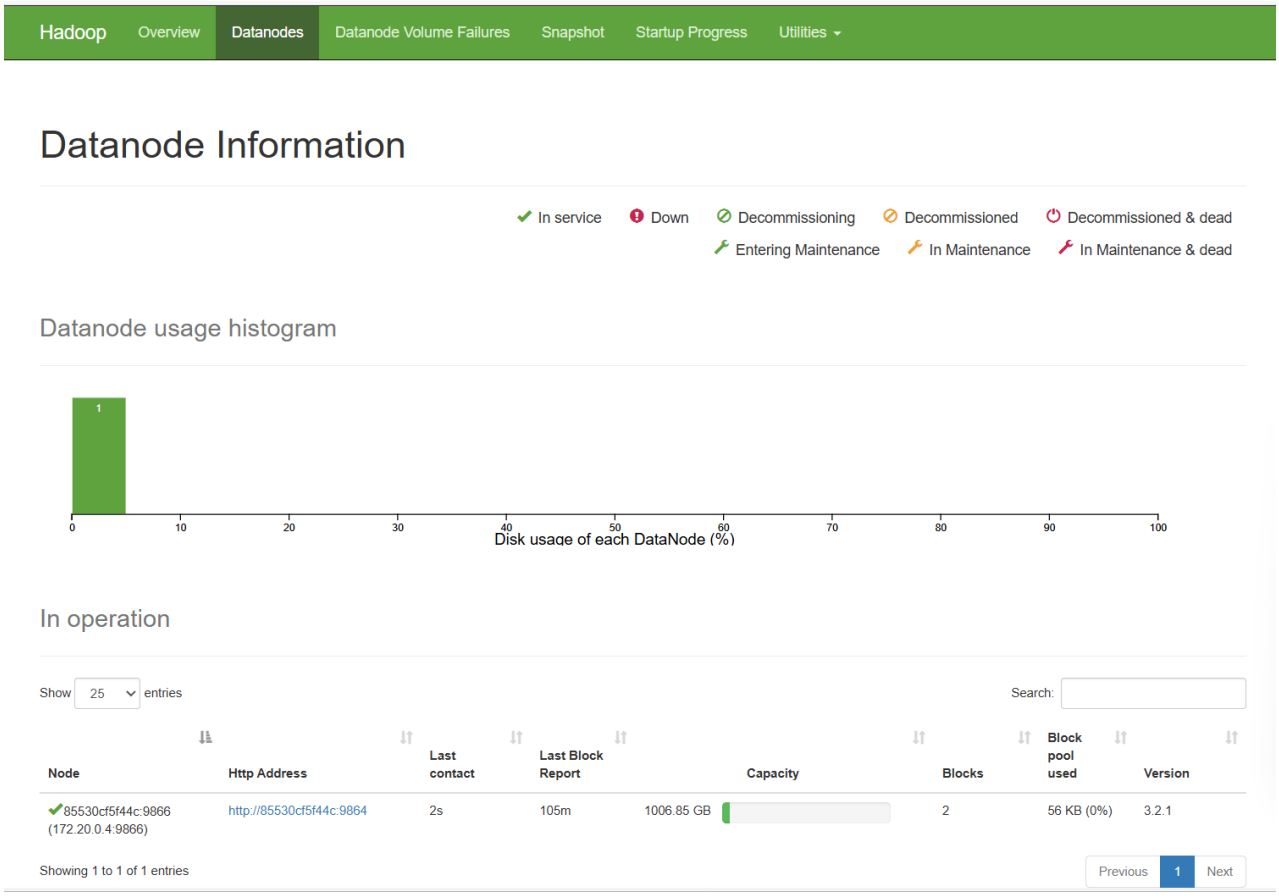
After processing:

- After processing and detecting anomalies in the data, the cleaned and structured data is written into HDFS for long term storage and further analysis.
- The processed data is stored in Parquet format within HDFS. Parquet is a columnar storage format that is both storage-efficient and optimized for fast querying.
- Data is partitioned by relevant attributes (e.ge., timestamp, device_id) to make querying faster. Partitioning ensures that data retrieval is efficient, particularly for large datasets like those expected from IoT devices.

Kafka Topic (mytopic) → Spark Streaming → Real-Time Cleaning → HDFS (Parquet files)

5.2 Storing in HDFS

To efficiently manage the enormous amount of healthcare data acquired from batch and real-time ingestion processes, HealthTrend Innovations employs the Hadoop Distributed File System (HDFS) as the communal, fault-tolerant storage



5.2.1 HDFS Cluster Configuration

Component	***Details***
NameNode	1 (Master node responsible for managing filesystem metadata and namespace)
Secondary NameNode	1 (Handles periodic checkpointing to assist the primary NameNode in recovery)
DataNodes	6 (Each with approximately 40 TB of usable storage capacity)
Replication Factor	3 (Each block of data is replicated across three different DataNodes for fault tolerance)

5.2.2 Data Storage Strategy

All types of ingested datasets batch patient CSV files, real-time IoT device streams, and medical imaging binaries are organized systematically within HDFS:

Data Type	*Storage Directory*	*Format*
Patient CSV Files	`/user/nifi/csv/`	CSV
Medical Imaging Files	`/user/nifi/images/`	Binary (e.g., DICOM, PNG)
Processed IoT Device Data	`/user/data/patient_data/`	Parquet

Partitioning strategies (e.g., partitioning by year, month, hospital region) are applied to improve query efficiency and management flexibility for downstream analytics using Hive, SparkSQL, and ML models.

By leveraging a robust HDFS infrastructure that includes 1 NameNode, 1 Secondary NameNode, and 6 DataNodes, HealthTrend Innovations can store gigantic volumes of healthcare data in a secure, dependable, and analytics- and machine learning-workload-optimized manner going forward.

The storage layer acts as the underpinning of the entire big data ecosystem, providing scalable and compliant access to structured, semi-structured, and unstructured healthcare data sets

6. Data visualization

To derive actionable healthcare insights from the ingested and processed datasets, HealthTrend Innovations leverages **Apache Spark MLlib** for machine learning and **Apache Airflow** for workflow orchestration.
This combination enables large-scale, automated data analysis and visualization of key healthcare trends and predictive outcomes.

6.1 Machine Learning Models for Visualization

Different machine learning models are applied based on the healthcare use case:

Use Case	Model Used	Purpose
Trend Analysis	Linear Regression	Visualize patient health trends over time, such as disease progression or hospital resource utilization patterns.
Classification	Logistic Regression	Identify and classify critical healthcare conditions (e.g., likelihood of ICU admission, critical risk patients).

Both models are implemented using **Apache Spark MLlib**, ensuring scalable training and evaluation over massive datasets.

6.2 Visualization of Critical Healthcare Insights

- Visual dashboards and trend plots are generated to display:
    - Disease outbreak patterns
    - ICU bed occupancy trends
    - Classification of patients into risk categories
    - Hospital resource utilization forecasts

- These visualizations provide HealthTrend analysts and clinical staff with immediate, actionable insights to optimize decision-making and patient care outcomes.

6.3 Workflow Orchestration with Apache Airflow

- Apache Airflow is used as a scheduler and orchestrator.
- Spark MLlib jobs (for data fetching, modeling, and visualization) are wrapped into Airflow DAGs (Directed Acyclic Graphs).
- Airflow automatically triggers, monitors, and retries visualization workflows on scheduled intervals, ensuring timely updates to healthcare dashboards without manual intervention.

This automation ensures consistent, up-to-date insights while reducing operational overhead.

7. Justifications

The following points justify the technical choices made for the proposed solution:

i. **Kafka for High-Velocity Real-Time Data**:
Kafka's distributed broker-topic-partition architecture efficiently handles the ingestion of high-velocity IoT device streams, ensuring reliable and ordered delivery to subscribed consumers like Spark Streaming.

ii. **Scalability with Spark and HDFS**:
Both Spark Structured Streaming and HDFS are horizontally scalable. As data volume grows, additional compute nodes and storage nodes can be added to the cluster without major re-architecting.

iii. **Low Latency for Real-Time Analytics**:
Kafka's event streaming combined with Spark's micro-batch processing ensures sub-second to few-second processing latency, ideal for near-real-time health monitoring and alert systems.

iv. **Support for Variety of Data Types**:
Different data formats — structured (CSV), semi-structured (JSON), and unstructured (binary imaging files) — are effectively ingested, processed, and stored using the combined capabilities of Kafka, NiFi, and HDFS.

v. **Data Cleaning and Standardization**:
Inconsistent and incomplete data records are cleaned during the ingestion phase (via NiFi processors) and the streaming phase (via Spark Structured Streaming transformations), ensuring high-quality datasets for downstream analytics.

vi. **Handling Massive Data Volumes Reliably**:
The system is designed to accommodate the ingestion and processing of 50+ TB of historical and streaming healthcare data, leveraging HDFS's replication mechanisms and Spark's distributed computing.

vii. **Security and Compliance**:
End-to-end security is maintained through:
    a. **SSL/TLS encryption** of data in transit,
    b. **Strict least-privilege access controls**,
    c. **Masking of personally identifiable information (PII)**,
    d. **Comprehensive pipeline auditing** to monitor and trace data flows, supporting HIPAA compliance.

viii. **Cost Efficiency through Open Source Stack**:
All architectural components are open-source, removing licensing costs. This ensures HealthTrend Innovations benefits from a powerful big data platform without incurring proprietary software expenses.

ix. **Resource and Storage Optimization**:
The architecture eliminates unnecessary resource duplication — no extra storage layers, no redundant databases, no over-provisioning. Horizontal scaling allows the system to grow organically without wasted infrastructure costs.

x. **Data Lifecycle Management for Cost Reduction**:
Older, less-frequently accessed data is migrated to warm or cold storage tiers over time, minimizing active storage costs without deleting historical data — preserving full audit trails and compliance without sacrificing cost efficiency

8.Security Consideration and Alternatives

8.1 Security Consideration

Data Ingestion Security (NiFi, Kafka)
Authentication: Kerberos authentication will secure all connections to NiFi and Kafka to maintain data ingestion access for only verified entities.
Encryption In-Transit: HTTPS (TLS/SSL encryption) safeguards data moving from IoT devices to Kafka and historical sources to NiFi against interception and tampering.
Access Control: Authorization policies at a detailed level (which depend on user roles) will regulate permissions for creating, reading, modifying, and deleting data ingestion pipelines.

Data Storage Security (HDFS)
Encryption At-Rest: Patient records and medical images stored in HDFS through Kafka and NiFi will undergo Transparent Data Encryption (TDE) protection to maintain security despite potential storage device breaches.
Authorization: The HDFS permissions and ACLs system ensures that only Spark, Hive, and Flink services along with their personnel can access files and directories.
Auditing: The system will enable HDFS audit logs to track access to sensitive datasets for compliance verification purposes.
Data Masking: During ingestion, PII fields including patient name and SSN will undergo masking to minimize exposure while preserving analytical usefulness for downstream tasks.

Data Processing Security (Spark, Hive, SparkML)
Secure Authentication: Kerberos user authentication ensures Spark and Hive clusters execute only authorized jobs and queries.
Input Validation: Spark ML processing includes data validation to identify any anomalies or malicious inputs prior to analysis.
Masking and De-identification: Sensitive fields undergo masking and pseudonymization before processing to maintain privacy while enabling valuable analytics.

Privacy Protection
Data Minimization: Data processing stages will select only essential fields such as diagnosis codes and timestamps to minimize exposure of sensitive personal identifiers.
Data Masking: We will use masking techniques such as hashing or tokenization to protect PII data fields including patient names and social security numbers.

Operational Security
Centralized Secret Management: A centralized secret vault such as HashiCorp Vault will handle the secure management of service account credentials along with encryption keys and SSL certificates.

Monitoring and Alerts: The system will enable real-time monitoring for Kafka, NiFi, HDFS, and Spark platforms while providing alerts when suspicious activities such as multiple failed login attempts or unexpected data transfers occur.

8.2 Alternatives

Reasons not to use Flume for the ingestion:

- As Patient health data is the Personal Identifiable Information (PII) and their health records which are highly sensitive, NiFi is used to ensure compliance to maintain the HIPAA regulations.
- .NiFi is good at backpressuring which means in batch-processing Ni-Fi is ingesting the only data it can handle so there is no data loss /no crashing.
- Flume is hard to monitor and audit the error handling.

Reasons not to use Apache Storm:

- Since Hospital data is not as real-time processing data like financial, Apache Storm is the better choice in our architecture diagram.
- Apache Spark Streaming is a better connector to HDFS, due to which clean file generated can be directly dumped into the HDFS which is also required according to our requirements.
- Apache Storm is also difficult in resource management because of the presence of spout bolt compared to Apache Spark Streaming.

Reasons not to use Scripting software (HIVE, HBASE, PIG):

- For real-time data processing, looking up the table's information is more time confusing as micro-services are not created here, no Scripting software is not used.
- In the proposal, that software are chosen which has direct connector to HDFS, where direct Read/Write can be done.

9. Proof of Concept

To validate the proposed big data architecture for HealthTrend Innovations, a functional Proof of Concept (PoC) was developed with containerized deployments and minimal datasets.

The PoC successfully validates the whole end-to-end process: from ingesting the data to real-time processing, batch storage, machine learning modeling, and scheduling visualization — all reflecting the design in the ultimate architecture diagram.

9.1. Ingestion Layer

9.1.1 Apache NiFi (Batch Ingestion)

- **CSV Patient Records**:
  - Using a CSV Processor Group, NiFi continuously monitored a local `/data/input/csv/` directory.
  - Files were parsed (`UpdateRecord`), enriched (`UpdateAttribute`), routed (`RouteOnAttribute`), and securely stored into HDFS using `PutHDFS`.
  -
- **Medical Imaging Files**:
  - Through a parallel Image Processor Group, binary files (e.g., PNG) were ingested and written directly to HDFS.

Backpressure **and** fault-tolerance were tested using simulated file loads.

9.1.2 Apache Kafka (Real-Time IoT Streaming)

- A simulated **IoT device generator** sent **JSON messages** containing patient telemetry data to a Kafka topic (`IOTTopic`).
- Kafka buffered these real-time events with durability and partitioning.

Kafka durability and partitioning across brokers was validated.

9.2 Processing Layer

9.2.1 Apache Spark Structured Streaming

- Spark Streaming jobs were launched from a **Cluster Manager (Master)** to multiple **Workers**.
- The jobs continuously read IoT telemetry from Kafka, parsed and validated incoming JSON messages, enriched them, and wrote clean output into HDFS in **Parquet** format.

Micro-batch streaming (~5 seconds interval) was demonstrated successfully. Storage Layer

9.3 Hadoop Distributed File System (HDFS)

- HDFS architecture included:
  - **1 NameNode** (metadata manager)
  - **1 Secondary NameNode** (checkpoint manager)
  - **6 DataNodes** (each ~40TB capacity)
- Data was organized into:
  - `/user/nifi/csv/` for patient records
  - `/user/nifi/images/` for medical imaging
  - `/user/data/patient_data/` for processed IoT data (Parquet)

HDFS replication (factor = 3) was verified for high availability.

9.4 Machine Learning and Visualization

Apache Spark MLlib

- Spark MLlib models were applied to clean datasets:
  - **Linear Regression** models visualized patient health trends over time.
  - **Logistic Regression** models classified patients based on critical risk factors.

Trends (e.g., heart rate progression) and classifications (e.g., critical risk alerts) were successfully visualized from real data.

9.5 Orchestration and Automation

Apache Airflow

- DAGs were developed in Airflow to automate the execution of Spark MLlib jobs.
- Airflow scheduled periodic model retraining and visualization refreshes.

Airflow successfully orchestrated Spark jobs without manual intervention.

Link to Proof of Concept: https://gitlab.com/cs65001/cs6500_sp2025_a05_g06