**Data Warehouse Design for the IDL Project**

Enmanuel Mateo Perez and Hedy Li

Department of Information Technology

CORE Team

April 4, 2025

Table of Contents

[**Goals** 4](#_Toc194680540)

[**Executive Summary** 4](#_Toc194680541)

[**Key Insights** 4](#_Toc194680542)

[**Introduction** 5](#_Toc194680543)

[**Pipeline Overview** 6](#_Toc194680544)

[**Objectives** 6](#_Toc194680545)

[**Data Sources and Ingestion** 7](#_Toc194680546)

[**Data Sources** 7](#_Toc194680547)

[**Data Transmission** 7](#_Toc194680548)

[**Databases, logs and API** 8](#_Toc194680549)

[**Ingestion Process: Connecting to Apache Kafka for the IDL Project** 8](#_Toc194680550)

[***Containerized Kafka and Kafka Connect Deployment:*** 10](#_Toc194680551)

[***Kafka Topic Management and Verification:*** 10](#_Toc194680552)

[***Kafka Connect and JDBC Connector Integration:*** 11](#_Toc194680553)

[***Development Tools and Dependencies:*** 11](#_Toc194680554)

[***Docker Compose Management:*** 12](#_Toc194680555)

[**Data Processing and Storage Layer: Transforming and Organizing IoT Data** 12](#_Toc194680556)

[***Data Lake Management with Apache Iceberg*** 12](#_Toc194680557)

[***Relational Data Storage with PostgreSQL*** 13](#_Toc194680558)

[***Time-Series Data Management with InfluxDB*** 13](#_Toc194680559)

[***Analytical Database with ClickHouse*** 13](#_Toc194680560)

[***Log Storage for Monitoring and Troubleshooting*** 13](#_Toc194680561)

[**Analysis and Findings** 14](#_Toc194680562)

[**Monitoring and System Health (Pipeline Performance - Thorough)** 15](#_Toc194680563)

[**Technology Evaluation** 17](#_Toc194680564)

[**Data Warehousing Solutions** 17](#_Toc194680565)

[**Data Analytics Tools** 18](#_Toc194680566)

[**Useful Tools for Data Pipeline** 18](#_Toc194680567)

[**Alternative Stack: Batch-Hybrid Approach** 19](#_Toc194680568)

[**Evaluation & Trade-Offs** 19](#_Toc194680569)

[**Recommendation** 25](#_Toc194680570)

[**Appendices** 25](#_Toc194680571)

[**References:** 27](#_Toc194680572)

# **Goals**

The Information and Communications Technologies Research Group (ICTRG) aimed to build a robust data warehouse solution for their Integrated Data Layer (IDL) project, which aggregates and provides access to diverse IoT sensor data. The project's objectives were:

* Establish a Data Warehouse Foundation: Create a data warehouse capable of handling the diverse IoT data streams, providing both on-demand and historical data access to authorized users.
* Conduct a Comprehensive Literature Review: Thoroughly research current data warehousing and analytics best practices, with a focus on open-source technologies, to identify the most suitable tools for the IDL project.
* Evaluate and Propose Solutions: Analyze and compare various open-source data warehousing and analytics stacks, outlining their pros, cons, and potential alternatives.
* Develop Practical Demonstrations: Experiment with the chosen solution using sample sensor data to validate its effectiveness and provide a working demonstration.
* Deliver a Detailed Report and Presentation: Compile a comprehensive report documenting the research findings and proposed solutions, and present the recommended architecture for the IDL data warehouse.

In essence, the project focused on researching, designing, and demonstrating a data warehousing solution to effectively manage and utilize the IoT data within the IDL project, with an emphasis on leveraging open-source technologies.

# **Executive Summary**

### **Key Insights**

The Integrated Data Layer (IDL) project was strategically designed to address the challenge of managing and extracting valuable insights from the diverse and voluminous IoT sensor data collected by the College Office of Research Enterprise (CORE) and the ICTRG. The core objective was to establish a robust, scalable, and efficient data warehouse, built upon open-source technologies, to centralize data ingestion, processing, storage, and visualization. This system aimed to provide researchers with both real-time and historical data access, facilitating data-driven decision-making and fostering innovation through automated workflows and continuous system monitoring.

The research strongly indicates that a combination of open-source tools provides a robust, scalable, and cost-effective solution for the IDL data warehouse. Tools like Apache Kafka for streaming, Apache Iceberg for data lake management, PostgreSQL and ClickHouse for analytical databases, and Grafana for visualization offer a comprehensive stack.

The real-time nature of IoT data necessitates a strong streaming platform. Apache Kafka, coupled with the MQTT Source Connector, provides the necessary infrastructure for reliable and high-throughput data ingestion.

Different database technologies are required to handle the diverse data types and analysis requirements. InfluxDB is ideal for time-series data, while PostgreSQL excel at complex analytical queries on structured data.

Apache Iceberg enables the creation of a data lake, offering flexibility for storing raw data and supporting incremental data processing and updates, essential for evolving IoT datasets.

Grafana provides a powerful and flexible platform for visualizing the data, enabling users to gain actionable insights and monitor system performance effectively.

Apache Airflow is critical for automating data pipelines, ensuring efficient ETL processes and reducing manual intervention.

Prometheus plays a vital role in monitoring the data pipeline's health and performance, enabling proactive identification and resolution of issues.

# **Introduction**

The proliferation of Internet of Things (IoT) devices has generated an unprecedented volume of data, offering immense potential for research and operational advancements. For the College Office of Research Enterprise (CORE) and the Information and Communications Technologies Research Group (ICTRG), this data stream represents a valuable asset, but also a significant challenge in terms of management and utilization. To address this, the Integrated Data Layer (IDL) project was initiated, aiming to construct a comprehensive data warehouse capable of centralizing, processing, and providing access to this diverse IoT data. This report details the research and development efforts undertaken to design and implement a robust data pipeline for the IDL project, focusing on the evaluation and selection of open-source technologies for data warehousing and analytics. The goal is to provide a scalable, efficient, and reliable solution that empowers researchers with actionable insights derived from the wealth of IoT data, ultimately fostering innovation and informed decision-making.

## **Pipeline Overview**

The data pipeline designed for the Integrated Data Layer (IDL) project is a multi-layered architecture, specifically engineered to ingest, process, store, and visualize the diverse and high-volume IoT sensor data. Central to its design is the utilization of open-source technologies, ensuring scalability, reliability, and cost-effectiveness. The Ingestion Layer efficiently collects data from various IoT sensors and other sources, leveraging gateways and message brokers like Mosquitto (MQTT). Apache Kafka serves as the robust and fault-tolerant streaming platform, enabling real-time data ingestion. The Processing and Storage Layer employs Apache Iceberg as the foundational table format for the data lake, facilitating efficient data management, schema evolution, and optimized query performance.

ClickHouse is implemented for high-performance analytical queries, particularly suited for the time-series and aggregation-heavy workloads inherent in IoT data analysis. InfluxDB is utilized for specialized time-series storage, while PostgreSQL handles structured data requiring complex relational queries. Apache Airflow orchestrates the Extract, Transform, Load (ETL) processes, ensuring automated and efficient data transformation. Finally, the Access Layer empowers users to visualize and analyze the data through Grafana dashboards, complemented by Apache Superset for ad-hoc SQL queries and broader analytical exploration. Prometheus continuously monitors system health and performance, ensuring pipeline reliability and proactive issue resolution. This architecture provides a comprehensive and optimized solution for managing the IDL project's IoT data, enabling efficient data warehousing and analytics.

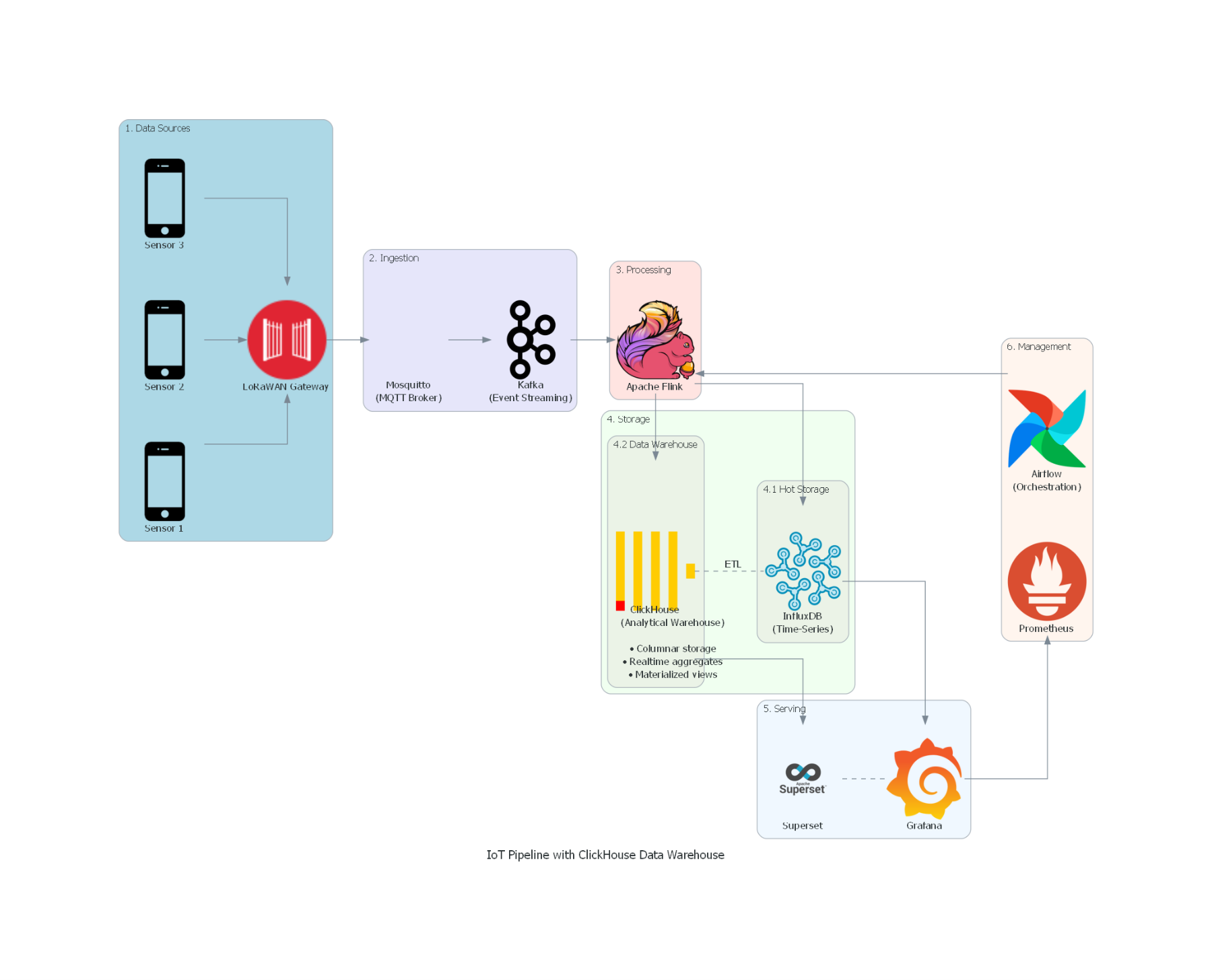
## **Objectives**

This research aimed to design and evaluate an optimal open-source architecture for the IDL project's data warehouse, focusing on efficient data ingestion (Kafka/MQTT), robust data lake management (Apache Iceberg), high-performance analytical querying (ClickHouse), specialized time-series storage (InfluxDB), structured data handling (PostgreSQL), automated workflow orchestration (Airflow), powerful data visualization (Grafana/Superset), and comprehensive system monitoring (Prometheus). The objective was to identify and demonstrate the feasibility of a production-ready data platform that effectively addresses the unique challenges of handling diverse and high-volume IoT data.

**Key Concepts:**

* **ETL (Extract, Transform, Load)**: Data is extracted, transformed into the required format/schema, and then loaded into a storage/processing system.Tools like Apache NiFi vs. streaming frameworks (Apache Flink).
* **OLAP Engines**: OLAP (Online Analytical Processing) engines are designed for interactive querying and exploration of large-scale analytical datasets.
* **CDC**: Change Data Capture. A technique used to identify and capture changes made to data in a database, allowing for real-time updates and synchronization with other systems.
* **Data Lake**: A centralized repository that allows you to store all your structured and unstructured data at any scale. You can run different types of analytics from dashboards and visualizations to big data processing, real-time analytics, and machine learning to help you make better decisions.
* **OLTP**: Online Transaction Processing.
* **HDFS**: Hadoop Distributed File System.
* **MPP**: Massively Parallel Processing.

# **Data Pipeline**



# **Data Sources and Ingestion**

## **Data Sources**

Currently, the primary data source for the IDL project is temperature readings from IoT sensors deployed at NBCC College, providing time-series data transmitted via MQTT. However, the architecture is designed with future scalability in mind, anticipating the integration of diverse data sources. These may include additional IoT sensors capturing environmental and operational data, existing college databases, system logs, external APIs for contextual information, and a device registry. The pipeline's flexibility is crucial to handle the varied data types, formats, and transmission protocols expected from these future integrations, while ensuring data quality, security, and efficient processing for comprehensive analysis.

## **Data Transmission**

* LoRaWAN: They transmit data over long distances with low power consumption to a LoRaWAN gateway.
* MQTT: The LoRaWAN gateway (or another intermediary) then relays the sensor data to a Mosquitto MQTT broker.
* MQTT Publish/Subscribe: The sensors (or gateway) publish temperature readings to specific MQTT topics.
* Kafka MQTT Source Connector: The Kafka MQTT Source Connector subscribes to these MQTT topics and pulls the data into the Kafka stream.
* Data Format: The data arrives in Kafka as messages, in a structured format (JSON) containing temperature values and timestamps.

## **Databases, logs and API**

Data collection for the IDL project currently centers on temperature readings from NBCC College IoT sensors, which transmit data via LoRaWAN and MQTT, ingested into Kafka through the MQTT Source Connector. However, the system is designed to accommodate future expansion, encompassing diverse sources. Additional IoT sensors will employ varied measurement methods and communication protocols (Zigbee, Bluetooth, cellular), requiring data aggregation and normalization. Integration of existing college databases will utilize SQL queries and ETL processes, while system logs will be collected and parsed for relevant information. External APIs will provide supplementary data through HTTP requests, and the device registry will be accessed via database queries or API calls. To ensure data integrity, standardization, validation, security, and scalable error handling will be critical components of the collection methodologies across all sources.

## **Ingestion Process: Connecting to Apache Kafka for the IDL Project**

**Architectural Context**

Apache Kafka serves as the central nervous system of our IDL data pipeline, responsible for handling the high-volume, real-time streaming of IoT sensor data. It operates as a distributed system, comprising brokers that manage data storage in topics. Each topic is further divided into partitions, enabling parallel processing and fault tolerance. In our project, understanding this architecture is crucial for establishing reliable and efficient data flow.

**Client Library Selection**

To interact with Kafka, we leverage client libraries that align with our development environment. For example, Python developers utilize the kafka-python library, while Java developers might employ the official Java client. These libraries provide the necessary tools to create producers (for sending data) and consumers (for receiving data).

**Producer Configuration and Operation:**

* **Bootstrap Servers:** The initial step involves configuring the producer with the addresses of the Kafka brokers (bootstrap servers), allowing it to establish a connection to the cluster.
* **Data Serialization:** We define how message keys and values are serialized (converted into a byte stream) for transmission. For instance, sensor readings might be serialized as JSON or UTF-8 encoded strings.
* **Producer Instantiation:** A producer object is created using the chosen client library, incorporating the configured settings.
* **Message Sending:** The producer's send() method is employed to transmit messages to specific Kafka topics. Each message includes a key and a value, representing the data and its associated identifier.
* **Error Handling and Acknowledgments:** Robust error handling mechanisms are implemented to address potential issues during message sending, such as network disruptions or broker failures. We also configure acknowledgments to ensure reliable message delivery.

**Consumer Configuration and Operation:**

* **Bootstrap Servers and Consumer Group:** Similar to producers, consumers require broker addresses for connection. They are also assigned to a consumer group, which determines how partitions are distributed among consumers for parallel processing.
* **Data Deserialization:** The consumer is configured with deserializers that convert the received byte stream back into the original data format.
* **Topic Subscription:** Consumers subscribe to one or more Kafka topics to receive data.
* **Message Polling and Processing:** A continuous loop is used to poll Kafka for new messages. Upon receiving a message, the consumer processes the data according to the application's logic.
* **Offset Management:** Consumers maintain offsets, which indicate the last consumed message in each partition. Committing offsets periodically prevents message reprocessing in case of consumer failures.

**Security Implementation:**

* **Authentication and Authorization:** To protect sensitive IoT data, we implement authentication and authorization mechanisms. Kafka supports various security protocols, such as SASL and SSL, to control access and ensure data integrity.
* **Data Encryption:** SSL encryption is employed to secure data in transit between producers, consumers, and brokers, preventing unauthorized interception.

Alright, let's break down the Kafka and Kafka Connect setup and verification process into detailed sections for your report:

### ***Containerized Kafka and Kafka Connect Deployment:***

* **Dockerization:**
  + To streamline deployment and ensure consistency, Apache Kafka and Kafka Connect were deployed using Docker containers. This approach offers advantages in terms of portability, isolation, and simplified management.
  + Docker Compose was employed to orchestrate the deployment of multiple containers, including Kafka and Kafka Connect, simplifying the process of starting, stopping, and managing the application stack.
* **Confluent Kafka Connect Image:**
  + The cp-kafka-connect-base image, maintained by Confluent, was selected as the base image for Kafka Connect. This image provides a foundational Kafka Connect worker, allowing for the addition of custom connectors.
* **Port Accessibility:**
  + Verification confirmed that the Kafka Connect container was running and accessible on port 8083, enabling communication with the Connect worker.

### ***Kafka Topic Management and Verification:***

* **Topic Creation:**
  + A Kafka topic named "Sensor1" was created for initial testing and data streaming from the temperature sensors.
  + The kafka-topics.sh command-line tool, executed within the Kafka container, was used to create the topic, specifying partitions and replication factors.
* **Topic Listing:**
  + The kafka-topics.sh command was also used to list existing topics within the Kafka cluster, ensuring that the "Sensor1" topic was successfully created.
* **Consumer Verification:**
  + A Kafka console consumer was launched using the kafka-console-consumer.sh command to receive and display messages from the "Sensor1" topic.
  + This step verified that data was being successfully published to and consumed from the Kafka topic.

### ***Kafka Connect and JDBC Connector Integration:***

* **JDBC Connectors Installation:**
  + The JDBC Source Connector (io.confluent.connect.jdbc.JdbcSourceConnector) and JDBC Sink Connector (io.confluent.connect.jdbc.JdbcSinkConnector) were installed in the Kafka Connect instance.
  + These connectors facilitate the integration of PostgreSQL databases with Kafka, enabling data streaming between the two systems.
* **Connector Configuration:**
  + The configuration of the JDBC connectors was done.
* **Connector Resources:**
  + External resources, including GitHub repositories and YouTube tutorials, were referenced to guide the installation and configuration of the JDBC connectors.
  + Repository for different connectors: <https://github.com/lensesio/stream-reactor>

### ***Development Tools and Dependencies:***

* **Tooling:**
  + scoop and jq.exe were utilized for command-line operations and JSON processing, respectively.
  + SBT (Simple Build Tool) and JDK 8+ were used to build the connection between Kafka and MQTT.
  + Git was used for version control and repository management.
* **Python Libraries:**
  + Pykafka and Paho-mqtt were installed as Python libraries to facilitate communication with Kafka and MQTT brokers.

### ***Docker Compose Management:***

* **Startup and Shutdown:**
  + docker-compose up -d was used to start the Kafka and Kafka Connect containers in detached mode.
  + docker-compose down was used to stop and remove the containers.
* **Container Restart:**
  + docker restart <container\_name\_or\_id> was used to restart specific containers.
  + docker restart $(docker ps -q) was used to restart all running containers.
* **Container Access:**
  + docker exec -it kafka /bin/bash was used to access the Kafka container's shell for command-line operations.

# **Data Processing and Storage Layer: Transforming and Organizing IoT Data**

The Data Processing and Storage layer forms a critical component of the IDL project's data pipeline, serving to transform and organize the raw IoT sensor data ingested from Apache Kafka into a structured and readily analyzable format. This layer ensures data quality, optimizes performance, and facilitates efficient data retrieval for downstream analysis and visualization.

### ***Data Lake Management with Apache Iceberg***

Instead of Apache Hudi, we have chosen Apache Iceberg as the table format for our data lake. Iceberg provides robust capabilities for managing large analytical datasets, particularly crucial for the high-volume streaming data from our IoT sensors. It offers features like schema evolution, enabling us to adapt to changes in data structure without disrupting ongoing queries. Iceberg's time-travel functionality allows for querying historical data versions, valuable for auditing and analyzing data changes over time. Furthermore, it supports flexible partitioning and optimized data layout, enhancing query performance. Atomic updates and deletes ensure data consistency and prevent corruption. In essence, Iceberg delivers reliability, performance, and flexibility, essential for managing the IDL project's evolving data landscape.

### ***Relational Data Storage with PostgreSQL***

PostgreSQL is utilized to store structured data that requires complex querying and analytical capabilities. Its robust support for SQL, ACID compliance, and extensibility makes it ideal for handling data that necessitates intricate relationships and advanced analytical functions. This database complements the data lake by providing a structured environment for data that benefits from relational modeling.

### ***Time-Series Data Management with InfluxDB***

Given the time-stamped nature of IoT sensor data, InfluxDB is employed for its optimized time-series storage and retrieval capabilities. Its high write throughput and time-based querying features are essential for efficiently managing and analyzing sensor readings over time. Data retention policies within InfluxDB allow us to manage storage space effectively by automatically deleting older data.

### ***Analytical Database with ClickHouse***

To facilitate fast and interactive data exploration, ClickHouse is incorporated as a high-performance analytical database. Its columnar storage, MPP architecture, and SQL compatibility enable rapid query execution on large datasets. This empowers users to perform real-time analytics and derive insights from the data efficiently.

### ***Log Storage for Monitoring and Troubleshooting***

Processing logs generated during data transformation and storage are collected and stored for monitoring and troubleshooting purposes. These logs provide valuable insights into system performance, aid in identifying and resolving issues, and support auditing and compliance requirements.

Data Flow within the Layer:

1. Raw data is ingested from Apache Kafka.
2. Data is managed as tables within the data lake using Apache Iceberg.
3. Data is transformed and enriched using tools like Apache Spark or Flink.
4. Transformed data is stored in PostgreSQL, InfluxDB, or ClickHouse based on data type and analysis requirements.
5. Processing logs are generated and stored.

# **Analysis and Findings**

During our evaluation of database solutions for the IDL project's analytical workloads, we observed that the volume and velocity of IoT sensor data necessitate a system capable of handling high-throughput data ingestion and delivering rapid query responses. Our initial analysis focused on PostgreSQL and general-purpose analytical databases. However, we found that these solutions, while robust, exhibited limitations in terms of real-time analytical performance, particularly when dealing with large volumes of time-series data and complex aggregation queries.

**ClickHouse: A High-Performance Analytical Solution:**

To address these limitations, we investigated ClickHouse, an open-source, column-oriented database management system designed for Online Analytical Processing (OLAP). Our findings indicate that ClickHouse offers significant advantages for the IDL project's analytical requirements:

* **Columnar Storage:** ClickHouse's columnar storage format significantly reduces I/O operations and improves query performance, especially for analytical queries that involve aggregations and filtering.
* **Vectorized Query Execution:** ClickHouse utilizes vectorized query execution, processing data in batches, which leads to substantial performance gains compared to row-based processing.
* **High Ingestion Rate:** ClickHouse is optimized for high ingestion rates, allowing us to efficiently handle the continuous stream of IoT sensor data from Kafka.
* **Real-Time Analytics:** ClickHouse's ability to perform real-time analytics enables us to provide immediate insights from the incoming sensor data.
* **SQL Compatibility:** ClickHouse supports standard SQL, making it easy to integrate with existing analytical tools and workflows.

**Recommendation for ClickHouse Implementation:**

Based on our analysis, we strongly recommend implementing ClickHouse as the primary analytical database for the IDL project. ClickHouse's high performance, scalability, and real-time capabilities align perfectly with the project's requirements for rapid data analysis and visualization. Specifically:

* **Data Ingestion from Kafka:** ClickHouse can be configured to directly ingest data from Kafka topics, enabling real-time data streaming and analysis.
* **Time-Series Analysis:** Its optimized time-series capabilities make it ideal for analyzing sensor data trends and patterns.
* **Ad-Hoc Queries and Dashboards:** ClickHouse's SQL compatibility allows for seamless integration with Apache Superset or other visualization tools, enabling users to create interactive dashboards and perform ad-hoc queries.
* **Integration with Iceberg:** Clickhouse's ability to work with data from Iceberg tables would allow for efficient querying of the data lake.

**Expected Benefits:**

By implementing ClickHouse, we anticipate the following benefits:

* **Improved Query Performance:** Significantly faster query response times for analytical workloads.
* **Enhanced Real-Time Analytics:** Ability to provide immediate insights from incoming sensor data.
* **Increased Scalability:** Ability to handle growing data volumes and complex queries.
* **Reduced Infrastructure Costs:** ClickHouse's efficient data processing can reduce the need for expensive hardware.

# **Monitoring and System Health (Pipeline Performance - Thorough)**

Maintaining the health and performance of the IDL project's data pipeline is paramount to ensuring data integrity, availability, and the timely delivery of insights. A comprehensive monitoring strategy, leveraging tools like Prometheus, is essential for proactive issue identification, performance optimization, and overall system reliability.

**Prometheus Metrics Collection and Analysis:**

Metric Exposure: Each component of the IDL data pipeline (Kafka, Iceberg, PostgreSQL, InfluxDB, ClickHouse, Airflow, and custom applications) is configured to expose relevant performance metrics in Prometheus-compatible format. These metrics encompass resource utilization (CPU, memory, disk I/O), latency, throughput, error rates, and application-specific data. This ensures comprehensive monitoring of the entire system.

**Pull-Based Scraping:** Prometheus uses a pull-based approach to gather metrics, periodically scraping data from each component's exposed endpoints. This method allows Prometheus to actively manage the monitoring process, reducing the potential performance overhead on the monitored systems.

**Metric Storage and Querying:** Prometheus stores the collected metrics in a time-series database, enabling efficient querying and analysis. PromQL (Prometheus Query Language) is used to formulate complex queries for exploring metric trends, identifying anomalies, and generating alerts. This allows for detailed investigations into pipeline behavior and performance.

**Key Metrics Monitored:** We monitor critical metrics across all components, including:

* **Kafka:** Message throughput, latency, consumer lag, broker resource utilization.
* **Iceberg:** Query execution time, data ingestion latency, storage utilization.
* **Databases:** Query execution time, connection pool utilization, write/read throughput.
* **Airflow:** DAG run status, task execution time, worker resource utilization.
* **Custom Applications:** Application-specific metrics related to data processing and transformation.

**Alerting:** Prometheus alerting rules are configured to trigger notifications when specific metric thresholds are breached. Alerts are routed to relevant personnel via email, Slack, or other notification channels. This allows for proactive issue resolution and minimizes downtime.

**Dashboarding with Grafana:** Grafana is used to create comprehensive dashboards that visualize the collected Prometheus metrics. These dashboards provide a holistic view of the pipeline's health and performance, enabling real-time monitoring and historical analysis.

**Log Aggregation and Analysis:** Centralized log aggregation is implemented to collect logs from all pipeline components. Log analysis tools are used to identify patterns, errors, and performance bottlenecks, complementing the metric data.

**Recurring Issues and Reliability:** System health assessments involve analyzing metrics and logs to identify recurring issues or patterns. Based on this analysis, recommendations are provided to improve system reliability, such as resource optimization, redundancy implementation, and configuration tuning.

**Bottleneck Identification:** Performance profiling tools and resource utilization analysis are used to identify bottlenecks in data processing and storage. This helps pinpoint areas where optimization is needed.

**Bottleneck Mitigation:** Strategies are developed to mitigate identified bottlenecks, including scaling resources, optimizing queries, improving data partitioning, and implementing caching.

**Impact Analysis:** The impact of bottlenecks on pipeline performance and data delivery is analyzed to prioritize mitigation efforts and ensure critical issues are addressed promptly.

# **Technology Evaluation**

### **Data Warehousing Solutions**

| **Tool** | **Pros** | **Cons** | **IoT Suitability** |
| --- | --- | --- | --- |
| **ClickHouse** | High-speed queries, columnar storage | Schema rigidity, limited UPDATE support | Excellent (time-series focus) |
| **TimescaleDB** | PostgreSQL compatibility, SQL-friendly | Scalability limits for massive datasets | Good (hybrid transactional/analytical) |
| **Apache Doris** | MySQL protocol support, real-time updates, MPP architecture | Smaller community than ClickHouse | Excellent (balanced real-time + batch, SQL-friendly) |

### **Data Analytics Tools**

| **Tool** | **Pros** | **Cons** |
| --- | --- | --- |
| **Apache Superset** | Rich visualizations, SQL editor, open-source | Limited alerting, moderate learning curve |
| **Metabase** | User-friendly, lightweight | Fewer advanced features (e.g., geospatial) |
| **Grafana** | Ideal for time-series dashboards | Tight coupling with monitoring systems |

### **Useful Tools for Data Pipeline**

A robust data pipeline for IoT requires tools for **ingestion**, **processing**, **orchestration**, and **monitoring**. Below are key open-source tools:

* **0. Data Messaging & Ingestion**

| **Tool** | **Description** | **IoT Use Case** |
| --- | --- | --- |
| **Apache Kafka** | Distributed event streaming platform for high-throughput, fault-tolerant messaging. | Real-time sensor data ingestion, decoupling producers/consumers. |
| **Redpanda** | Kafka-compatible, ZooKeeper-free streaming. Simpler deployment. | Edge IoT scenarios with limited resources. |

* **1. Stream Processing & Real-Time ETL**

| **Tool** | **Description** | **IoT Use Case** |
| --- | --- | --- |
| **Apache Flink** | Stateful stream processing with low latency. Supports event-time semantics. | Real-time anomaly detection, windowed aggregations. |
| **Apache Spark** | Batch & streaming processing. Ideal for ML integration (Spark MLlib). | Large-scale batch ETL (e.g., historical data backfills). |
| **Apache Beam** | Unified API for batch/streaming. Runs on Flink, Spark, etc. | Portable pipelines across cloud/on-prem. |

* **2. Workflow Orchestration**

| **Tool** | **Description** | **IoT Use Case** |
| --- | --- | --- |
| **Apache Airflow** | Python-based DAG scheduler for batch workflows. | Scheduling ETL jobs, sensor data backups. |

* **3. Lightweight Data Collection**

| **Tool** | **Description** | **IoT Use Case** |
| --- | --- | --- |
| **Telegraf** | Agent-based metrics collector (part of TICK stack). Supports 200+ plugins. | Pulling sensor data from MQTT |
| **Fluent Bit** | Lightweight log processor/forwarder. Low resource footprint. | Edge device log aggregation. |

* **4. Monitoring & Observability**

| **Tool** | **Description** | **IoT Use Case** |
| --- | --- | --- |
| **Prometheus** | Time-series monitoring with alerting. Pull-based scraping. | Tracking sensor health metrics. |
| **Grafana** | Visualization for Prometheus, InfluxDB, etc. | Dashboards for temperature/humidity trends. |

* **5. Distributed Query Engines**

| **Tool** | **Description** | **IoT Use Case** |
| --- | --- | --- |
| **Trino** | Federated SQL engine (query across databases). Formerly PrestoSQL. | Joining IoT data with business data (e.g., PostgreSQL). |

### **Alternative Stack: Batch-Hybrid Approach**

**Tools**: Apache Spark (batch processing) + TimescaleDB + Metabase.

## **Evaluation & Trade-Offs**

| **Criteria** | **Kafka + Flink + ClickHouse** | **Spark + TimescaleDB** |
| --- | --- | --- |
| **Latency** | Real-time (ms) | Near-real-time (minutes) |
| **Scalability** | Petabyte-scale | TB-scale |
| **Ease of Use** | Moderate (requires DevOps) | High (SQL-centric) |
| **Community Support** | Strong (Apache projects) | Growing (PostgreSQL ecosystem) |

#### **Core Positioning & Use Cases**

| **System** | **Core Focus** | **Use Cases** |
| --- | --- | --- |
| **Hive** | Hadoop-based offline data warehouse | Large-scale batch ETL, offline analytics (e.g., log processing, data lake storage) |
| **Druid** | Real-time time-series analytics | Real-time monitoring, ad analytics, user behavior analysis (low latency + high concurrency) |
| **ClickHouse** | High-performance OLAP columnar database | Wide-table aggregations, log/behavioral analytics (extremely fast single-table queries) |
| **Kudu** | Real-time updatable storage engine | Real-time data updates + analysis (e.g., IoT sensor data, financial transactions) |
| **Greenplum** | Enterprise MPP data warehouse | Complex ETL, business intelligence (BI), hybrid workloads (PostgreSQL-compatible) |
| **Pinot** | Low-latency real-time OLAP engine | Interactive real-time analytics (e.g., clickstream analysis, like LinkedIn’s use case) |
| **PostgreSQL** | General-purpose RDBMS (supports OLTP/OLAP) | Transaction processing, medium-scale analytics (with extensions like TimescaleDB) |
| **StarRocks** | High-performance MPP OLAP engine | High-concurrency real-time analytics, ad-hoc queries (e.g., e-commerce dashboards) |
| **Doris** | Lightweight MPP OLAP engine | Real-time reporting, mid-scale analytics (simplified fork of StarRocks) |

#### **Architecture & Performance**

| **System** | **Storage Model** | **Query Latency** | **Throughput** | **Scalability** | **Real-time Ingestion** |
| --- | --- | --- | --- | --- | --- |
| **Hive** | Row/Column (ORC) | Minutes | High | High (via HDFS) | No (batch-only) |
| **Druid** | Columnar + inverted index | Sub-second | Medium | High (distributed) | Yes (streaming) |
| **ClickHouse** | Columnar | Seconds | Very High | Moderate (sharding via ZooKeeper) | Yes (batch/stream) |
| **Kudu** | Columnar | Seconds | High | High (distributed) | Yes (real-time updates) |
| **Greenplum** | Row/Columnar | Minutes | High | Moderate (MPP scaling) | Yes (batch) |
| **Pinot** | Columnar + inverted index | Sub-second | High | High (auto-sharding) | Yes (streaming) |
| **PostgreSQL** | Row-based (column extensions) | Seconds-minutes | Medium | Limited (replication) | Yes (transactional) |
| **StarRocks** | Columnar | Sub-second | Very High | High (elastic) | Yes (streaming) |
| **Doris** | Columnar | Seconds | High | High (distributed) | Yes (batch/stream) |

#### **Key Features**

| **System** | **Strengths** | **Weaknesses** | **Data Updates** | **Transactions** |
| --- | --- | --- | --- | --- |
| **Hive** | SQL compatibility, mature ecosystem | High latency, not for interactive | Overwrite-only | No |
| **Druid** | Low latency, high concurrency | Complex pre-aggregation, weak joins | No | No |
| **ClickHouse** | Blazing-fast scans, high compression | Poor multi-table joins, no transactions | Limited (via ALTER) | No |
| **Kudu** | Real-time updates, Hadoop integration | Limited storage scale, needs compute engines (e.g., Impala) | Yes | Partial (row-level) |
| **Greenplum** | Complex query optimization, ACID | Costly scaling, weak real-time | Yes | Yes (full ACID) |
| **Pinot** | Fast multi-dimensional aggregations | Complex setup, smaller community | No | No |
| **PostgreSQL** | Versatile, extensible | Performance degrades at scale | Yes | Yes (OLTP) |
| **StarRocks** | High concurrency, vectorized engine | Young ecosystem, fewer optimizations | Yes | Partial (batch) |
| **Doris** | Simple setup, MySQL compatibility | Fewer features vs. StarRocks | Yes | Partial (batch) |

#### **Open-source version vs Paid versions**

| **Tool/Project** | **Open-Source Status** | **Paid Version/Commercial Support** | **Key Differences Between Free and Paid** |
| --- | --- | --- | --- |
| **Apache Hive** | Fully open-source (Apache 2.0 License) | No official paid version, but third-party commercial support (e.g., Cloudera, Hortonworks) | Free version is fully functional; paid support offers enterprise services (e.g., technical support, optimization) |
| **Apache Druid** | Fully open-source (Apache 2.0 License) | Commercial versions available (e.g., Imply, CelerData) | Free version is fully functional; commercial versions offer enhanced features (e.g., management tools, support) |
| **ClickHouse** | Fully open-source (Apache 2.0 License) | Commercial version (ClickHouse Cloud) | Free version is fully functional; commercial version offers managed services, management, and support |
| **Apache Kudu** | Fully open-source (Apache 2.0 License) | No official paid version, but third-party commercial support (e.g., Cloudera) | Free version is fully functional; paid support offers enterprise services |
| **Greenplum** | Open-source (Apache 2.0 License) | Commercial version (VMware Tanzu Greenplum) | Free version is fully functional; commercial version offers advanced features and support |
| **Apache Pinot** | Fully open-source (Apache 2.0 License) | Commercial versions available (e.g., StarTree) | Free version is fully functional; commercial versions offer extended features and managed services |
| **PostgreSQL** | Fully open-source (PostgreSQL License) | Commercial support available (e.g., EnterpriseDB) | Free version is fully functional; commercial versions offer advanced features, support, and management tools |
| **StarRocks** | Partially open-source (Community Edition under Apache 2.0 License) | Commercial version (StarRocks Enterprise) | Community Edition has limited features; Enterprise Edition offers more features (e.g., data security, cluster management) |
| **Apache Doris** | Fully open-source (Apache 2.0 License) | Commercial support available (e.g., CelerData) | Free version is fully functional; commercial versions offer extended features and support |

# **Recommendation**

For the IDL project, we recommend constructing a data platform leveraging a robust open-source stack to ensure scalability, performance, and long-term sustainability. Apache Kafka should be the core for real-time data ingestion, capable of handling high-throughput streams from IoT sensors and diverse sources. Apache Iceberg, replacing Hudi, will serve as the foundational data lake table format, enabling efficient data management, schema evolution, and optimized query performance. ClickHouse is crucial for delivering high-speed analytical queries, especially for the time-series data and complex aggregations required for IoT analysis. Grafana, paired with Apache Superset, will provide powerful visualization and ad-hoc query capabilities, empowering users to explore and derive insights from the data.

To automate workflows and ensure pipeline reliability, Apache Airflow is recommended for orchestrating ETL processes, and Prometheus for comprehensive system monitoring and alerting. Implementing this architecture via Docker containerization will streamline deployments and management, while adopting Infrastructure as Code (IaC) and CI/CD practices will ensure efficient scaling and continuous improvement. We also emphasize the importance of robust data governance, security protocols, and comprehensive user training. Continuous performance optimization and active engagement with the open-source communities surrounding these tools will be essential for the long-term success of the IDL project. This stack provides a powerful, adaptable foundation for the IDL project's data-driven initiatives.

# **Appendices**

**MPP**: Massively Parallel Processing MPP is a computing architecture where multiple processors work on different parts of a task simultaneously. MPP databases (e.g., Greenplum, ClickHouse) distribute data and processing across multiple nodes to achieve high performance for large-scale data analytics.

**OLAP**: OLAP(Online Analytical Processing) refers to systems designed for complex queries and data analysis. OLAP databases (e.g., Apache Druid, StarRocks)

**OLTP**: OLTP refers to systems designed for transactional workloads, such as inserting, updating, and deleting small amounts of data. OLTP databases (e.g., PostgreSQL)

**CDC**: (Change Data Capture)CDC is a technique for tracking and capturing changes in a database (e.g., inserts, updates, deletes) in real time. It is often used for data replication, synchronization, and ETL processes to ensure data consistency across systems.

**ETL**: (Extract, Transform, Load)ETL is a data integration process where data is extracted from source systems, transformed into a usable format, and loaded into a target database or data warehouse. It is a key component of data pipelines and analytics workflows.

# **References:**

**Github, Kafka Documentation, Codes:** <https://github.com/EnmanuelMateo/IDL-NBCC.git>

**Streaming package:** <https://hub.docker.com/r/bitnami/kafka>

**Benchmarking Tools:**[ClickHouse Benchmark](https://benchmark.clickhouse.com/)

**Ecosystem Rankings:** [Database Ranking](https://benchant.com/ranking/database-ranking)

**Docker:** <https://www.docker.com/>

**Confluent Connector-plugins:** <https://www.confluent.io/hub/>