Crop Disease Prediction and Alert System

Big Data Technologies – Year 2025
Professor Daniele Miorandi
and Stefano Tavonatti

- Paolo Fabbri – paolo.fabbri-2@studenti.unitn.it
- Filippo Stanghellini – filippo.stanghellini@studenti.unitn.it
- Federico Molteni – federico.molteni@studenti.unitn.it

Submission Date: 28-08-2025

Abstract

- **Brief Description:** The project implements a scalable big data platform for real-time agricultural monitoring and crop disease prediction. By integrating distributed data management with IoT sensors, satellite imagery, and weather sources, the system leverages unsupervised machine learning algorithms and real-time threshold monitoring to detect environmental anomalies and provide early warning capabilities. This enables agricultural stakeholders to proactively manage crop health and minimize disease-related losses through data-driven decision making.

Key Objectives:

1. **Real-time Data Processing Pipeline**: Design and implement a fault-tolerant, distributed streaming architecture using Apache Spark Structured Streaming with micro-batch processing for continuous ingestion and processing of multi-source agricultural data.
2. **ML-Powered Anomaly Detection**: Develop and deploy unsupervised machine learning models (K-means clustering) with automatic retraining capabilities, feature engineering for 10 environmental parameters, and real-time prediction generation for detecting unusual agricultural conditions.
3. **Dual-Mode Intelligent Alerting System**: Create a sophisticated alerting mechanism combining rule-based threshold monitoring (temperature, humidity, soil pH) with specific weather condition rules (wind speed, UV index, precipitation) and ML-based anomaly detection, providing comprehensive coverage for different types of agricultural risks.
4. **Scalable Data Lake Architecture**: Implement a three-zone medallion architecture (Bronze/Silver/Gold) with MinIO object storage, enabling data quality validation, feature engineering, and ML-ready datasets while maintaining temporal partitioning and processing metadata for data traceability.
5. **Real-time Interactive Dashboard**: Develop a comprehensive Streamlit-based monitoring interface with specialized pages for real-time sensor and weather data visualization, ML anomaly detection results with severity classification, threshold-based alert monitoring and management, satellite imagery display with field mapping, and historical analysis of ML predictions and alerts with statistical reporting.

Main results achieved:

1. **End-to-End Distributed Data Pipeline**: Successfully built a production-ready data processing infrastructure that handles real-time streaming from IoT sensors (3 agricultural fields), weather APIs (WeatherAPI.com), and satellite imagery (Copernicus Sentinel-2) through Apache Kafka, processes data using Spark clusters (Master + 2 Workers), and stores results in a structured data lake with automatic partitioning and optimization.
2. **Real-time Streaming Alert System**: Implemented a fault-tolerant alerting service using Spark Structured Streaming that processes validated data streams, applies configurable threshold rules, and generates immediate alerts with severity classification. The system maintains checkpointing for fault recovery and provides REST APIs for alert management and monitoring.
3. **Operational ML Pipeline with Advanced Features**: Deployed a fully operational K-means clustering system that processes 10 engineered features including environmental differentials, temporal patterns, and quality metrics. The model achieves automatic retraining every 7 days, maintains version control with metadata tracking, and provides real-time anomaly scoring with 4-level severity classification (LOW, MEDIUM, HIGH, CRITICAL).
4. **Comprehensive Monitoring Dashboard**: Created a sophisticated 5-page Streamlit interface featuring real-time data visualization with 10-second auto-refresh for sensor and ML data, interactive field filtering, ML prediction analytics with severity classification, threshold alert monitoring and management, satellite imagery display with field location mapping, and historical analysis of ML predictions and alerts with temporal charts and statistical summaries.

What problem are you solving?

Specifically, we address two interconnected challenges:

- **Identification of Anomalous Conditions**: The system monitors and detects in real-time environmental conditions (such as temperature, humidity, and soil pH) that can favor the development of pathogens. This allows us to identify anomalies before the disease symptoms become visible on the plant, overcoming the limitations of systems based solely on leaf image analysis.
- **Lack of Preventive Alerts**: Many farmers react to diseases only after crop production and quality have already been compromised. Our system enhances preventive capability by combining real-time data from field IoT sensors and weather sources. It integrates both threshold-based logic and unsupervised machine learning, ensuring reliable early detection through a balance of simplicity and technological advancement.

Why is this important?

- **Addressing the Core Vulnerability of Agriculture:** Plant diseases are a major threat, causing billions of dollars in economic losses annually. This not only reduces farmers' profits but also endangers the food security of a growing global population. Our system positions itself as a proactive response to this vulnerability, providing a critical tool for risk management.
- **Shifting from Reaction to Prevention:** Many existing solutions focus on visual diagnosis—detecting a problem only after symptoms are visible on the leaves. Our system distinguishes itself by acting **preventively**. By continuously monitoring environmental data, we can identify conditions that favor pathogen development before any damage occurs. This allows for timely intervention, turning a reactive problem into a manageable risk.
- **Fostering Environmental Sustainability:** The proactive nature of our system has significant environmental benefits. By providing early alerts, we enable farmers to apply targeted treatments, reducing the need for widespread and often excessive use of pesticides. This leads to less pollution, lower operational costs, and a more sustainable use of valuable resources like water and fertilizers, which are not wasted on a compromised crop.
- **Building Resilience to Climate Change:** In an era of increasingly unpredictable weather, the ability to adapt and act on informed decisions is key. Our project provides a practical tool for farmers to manage their operations with greater resilience, helping them navigate new and evolving threats posed by a changing climate. It’s not just about managing disease; it’s about empowering farmers to secure their livelihoods in a volatile world.

Data Exploration Insights (1/2)

IoT Sensor Data Source

Data Generation:
Synthetic, IoT sensor data generated using statistical distributions (Gaussian for temperature/pH, uniform for humidity) to simulate real agricultural field sensors. Based on Kaggle agricultural datasets for realistic parameter ranges.

Data Structure:

```
{
  "timestamp": "ISO-8601",
  "field_id": "string",
  "location": "string",
  "temperature": "double",
  "latitude": "double",
  "longitude": "double",
  "humidity": "double",
  "soil_ph": "double"
}
```

Processing Flow:

- ➔ Generated by Python producer
- ➔ Python producer
- ➔ Kafka `sensor_data` topic
- ➔ Bronze zone (JSON with temporal partitioning)
- ➔ Silver zone (validated Parquet + Kafka `iot_valid_data`)
- ➔ Gold zone (10-min aggregated features for ML)

Weather Data Source

Data Generation:
Real-time external API data retrieved from WeatherAPI.com every 60 seconds using REST API calls with API key authentication. Provides current meteorological conditions for Verona region.

Data Structure :

```
{
  "message_id": "uuid",
  "timestamp": "ISO-8601",
  "location": "string",
  "region": "string",
  "country": "string",
  "lat": "double",
  "lon": "double",
  "temp_c": "double",
  "humidity": "integer",
  "wind_kph": "double",
  "condition": "string",
  "uv": "double"
}
```

Processing Flow:

- ➔ WeatherAPI.com
- ➔ Python producer
- ➔ Kafka `weather_data` topic
- ➔ Bronze zone (JSON with temporal partitioning)
- ➔ Silver zone (validated Parquet + Kafka `weather_valid_data`)
- ➔ Gold zone (10-min aggregated features for ML)

Satellite Imagery Data Source

Data Generation:
Real satellite imagery from Copernicus Sentinel-2 L2A via OAuth2 authentication. RGB composite images (B04, B03, B02 bands) with <10% cloud coverage filter, covering Verona agricultural area.

Data Structure :

```
{
  "timestamp": "ISO-8601",
  "image_base64": "base64-
encoded-png-image",
  "location": {
    "bbox": [min_lon, min_lat,
max_lon, max_lat]
  }
}
```

Processing Flow:

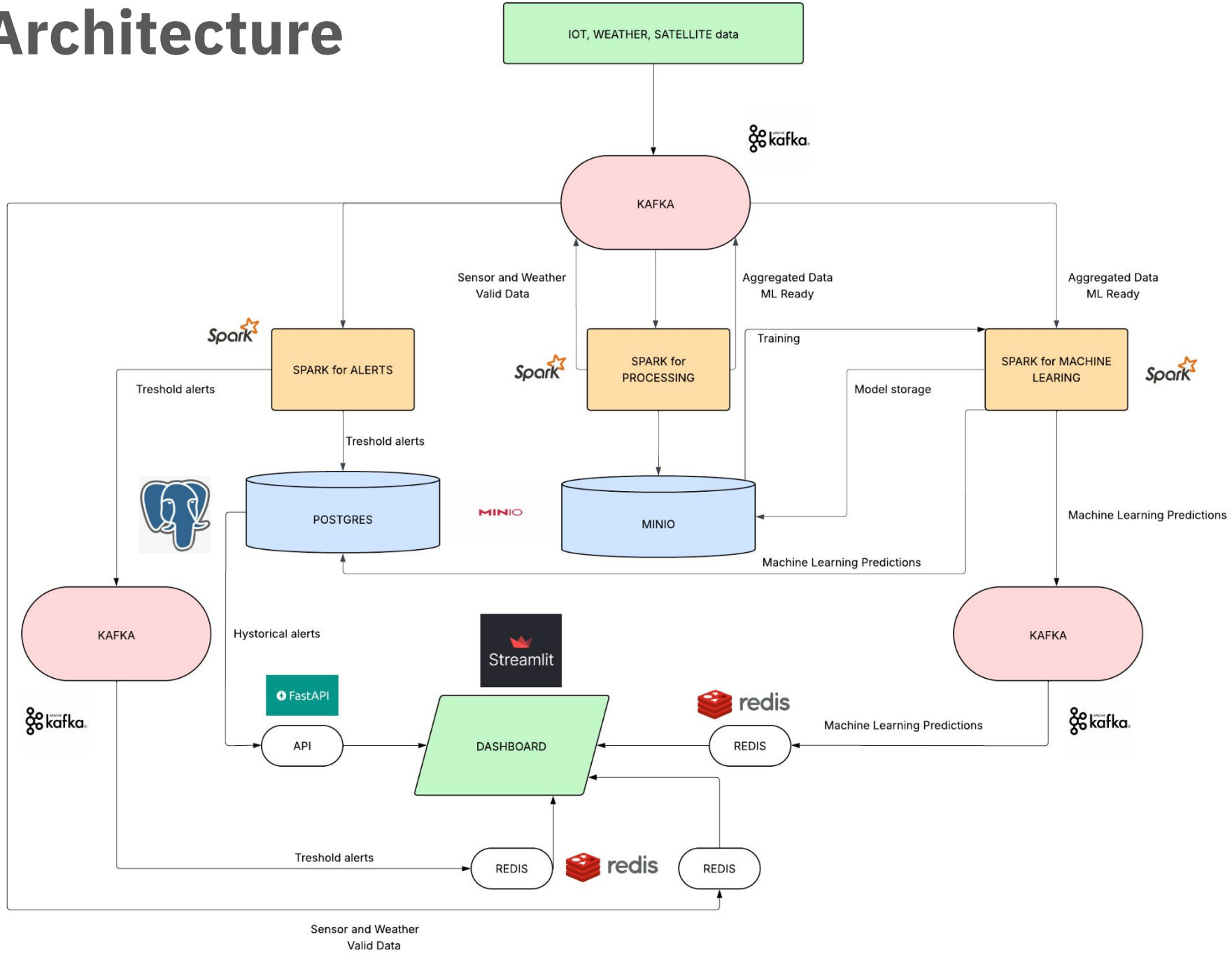
- ➔ Copernicus API
- ➔ Python producer
- ➔ Kafka `satellite_data` topic
- ➔ Bronze zone (metadata JSON + images extracted to MinIO)
- ➔ Silver zone (validated metadata only, images remain in MinIO `satellite-images` bucket)

Data Exploration Insights (2/2)

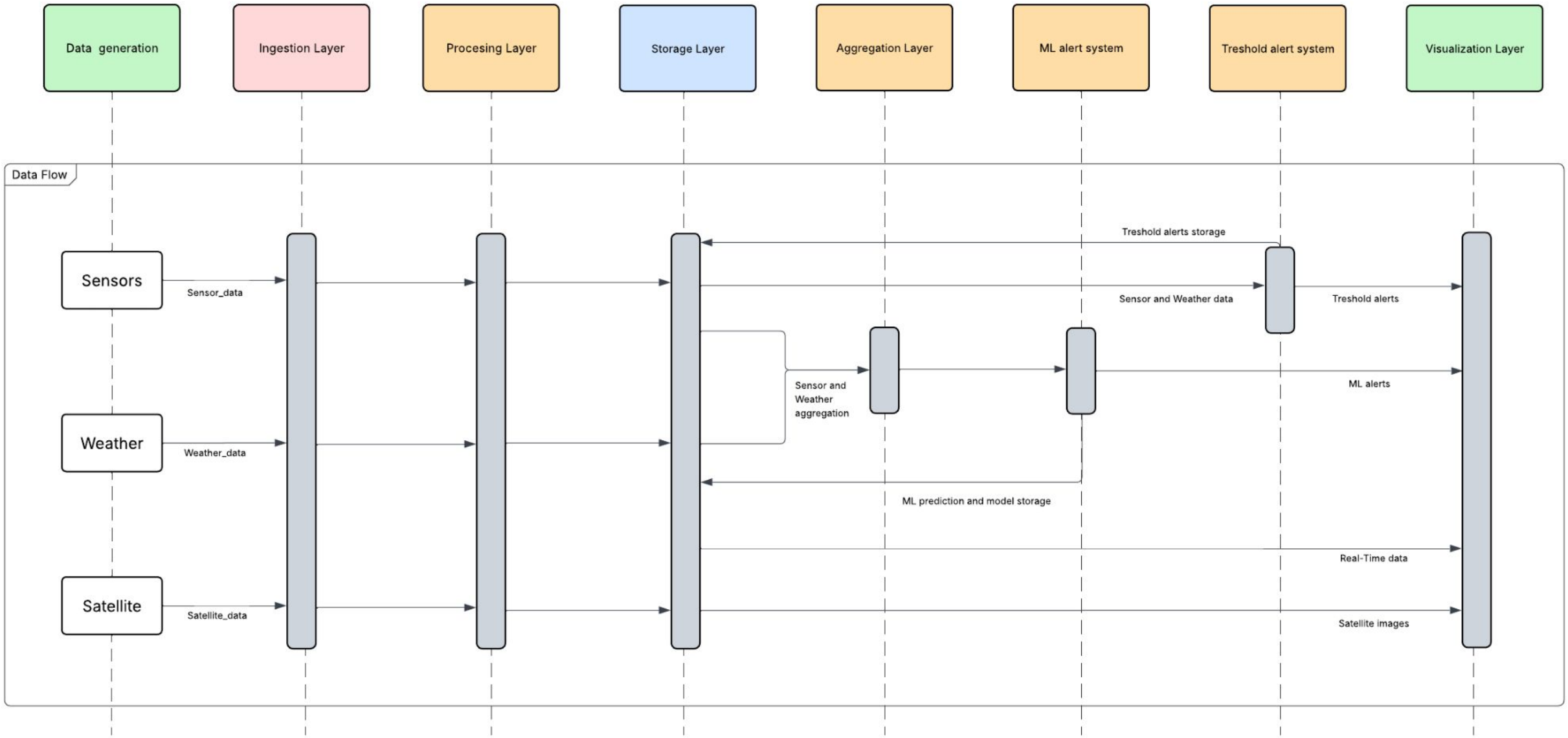
Data Source	Origin	Format	Frequency	Size	Volume	Flow
Sensor (Raw)	Synthetic	JSON	60s per field (3 fields)	~200 bytes/msg	3 msg/min	Kafka sensor_data
Weather (Raw)	WeatherAPI.com	JSON	60s	~400 bytes/msg	1 msg/min	Kafka weather_data
Satellite (Raw)	Copernicus Sentinel-2	JSON + Base64 PNG	60s	~350 KB/msg	1 msg/min	Kafka satellite_data
Sensor (Bronze)	Bronze processor	JSON	30s micro-batches	~250 bytes/record	3 records/min	MinIO s3a://bronze/iot/
Weather (Bronze)	Bronze processor	JSON	30s micro-batches	~450 bytes/record	1 record/min	MinIO s3a://bronze/weather/
Satellite (Bronze)	Bronze processor	JSON metadata	60s batches	~300 bytes/record	1 record/min	MinIO s3a://bronze/satellite/
Satellite Images	Bronze processor	PNG binary	60s	~350 KB/image	1 image/min	MinIO satellite-images/
Sensor (Silver)	Silver processor	Parquet + Kafka	1min micro-batches	~280 bytes/record	~3 records/min	MinIO s3a://silver/iot/ + Kafka iot_valid_data
Weather (Silver)	Silver processor	Parquet + Kafka	1min micro-batches	~500 bytes/record	~1 record/min	MinIO s3a://silver/weather/ + Kafka weather_valid_data
Satellite (Silver)	Silver processor	Parquet	1min micro-batches	~350 bytes/record	~1 record/min	MinIO s3a://silver/satellite/
ML Features (Gold)	Gold processor	Parquet + Kafka	10min windows	~800 bytes/record	0.3 records/min	MinIO s3a://gold/ml_feature/ + Kafka gold-ml-features

Total Daily Volumes:
Raw data ~505 MB/day (99% satellite images), processed data ~1.5 MB/day structured data across all zones.

System Architecture



System Architecture (Data flow)



Technologies and Justifications

Category	Technology	Justification
CORE LANGUAGE	Python	Primary language for all microservices.
API FRAMEWORK	FastAPI	A modern, high-performance web framework used to build RESTful APIs in the crop-disease-service and ml-anomaly-service for serving data and receiving commands.
DATA STREAMING	Apache Kafka	Acts as the central nervous system of our architecture, providing a distributed, fault-tolerant message bus for asynchronous communication between microservices.
STREAM PROCESSING	Apache Spark	The core processing engine for our data lake. Its Structured Streaming capabilities are used for real-time data validation, transformation, and feature engineering.
ML FRAMEWORK	PySpark ML	Used for implementing our K-means clustering model. PySpark ML provides distributed machine learning capabilities that integrate seamlessly with our Spark-based data processing pipeline.
CONTAINERIZATION	Docker	All microservices are containerized with Docker, ensuring consistent, isolated, and reproducible deployments across different environments.
DATABASE	PostgreSQL	Serves as the primary relational database for storing structured data like alerts and ML predictions.
IN-MEMORY CACHE	Redis	Provides a high-speed, in-memory caching layer. The redis-cache-service uses it to store the latest data points for quick retrieval by the dashboard.
OBJECT STORAGE	MinIO	An S3-compatible object storage solution used to implement our medallion data lake architecture (Bronze, Silver, Gold zones) for storing large volumes of unstructured data.
DASHBOARD	Streamlit	A Python framework for building interactive web applications. It allows us to rapidly develop and deploy a user-friendly dashboard for data visualization and monitoring.

Implementation & Code Repository

GitHub Link and Docker Images

GitHub Repository: https://github.com/stanghee/Crop-Disease-Prediction-and-Alert-System
Docker Images: The project uses standard Docker images (Apache Spark 3.5.0, confluentinc/cp-kafka 7.4.0, confluentinc/cp-zookeeper:7.4.0, Redis 7.0, PostgreSQL 15, MinIO) with custom-built services for the microservices components. All services are containerized and orchestrated via Docker Compose.

High-Level Code Structure

The project implements a microservices architecture with 8 specialized services orchestrated by Docker Compose. The core structure follows a medallion data lake pattern (Bronze/Silver/Gold zones) using Apache Spark for distributed stream processing and Kafka for real-time messaging. Services are organized into data producers (IoT sensors, weather API, satellite imagery), data processing engines (storage service with 3-zone architecture, ML anomaly detection, crop disease alerting), real-time caching (Redis with TTL management), and interactive visualization (5-page Streamlit dashboard with real-time updates).

Key Modules/Pipelines

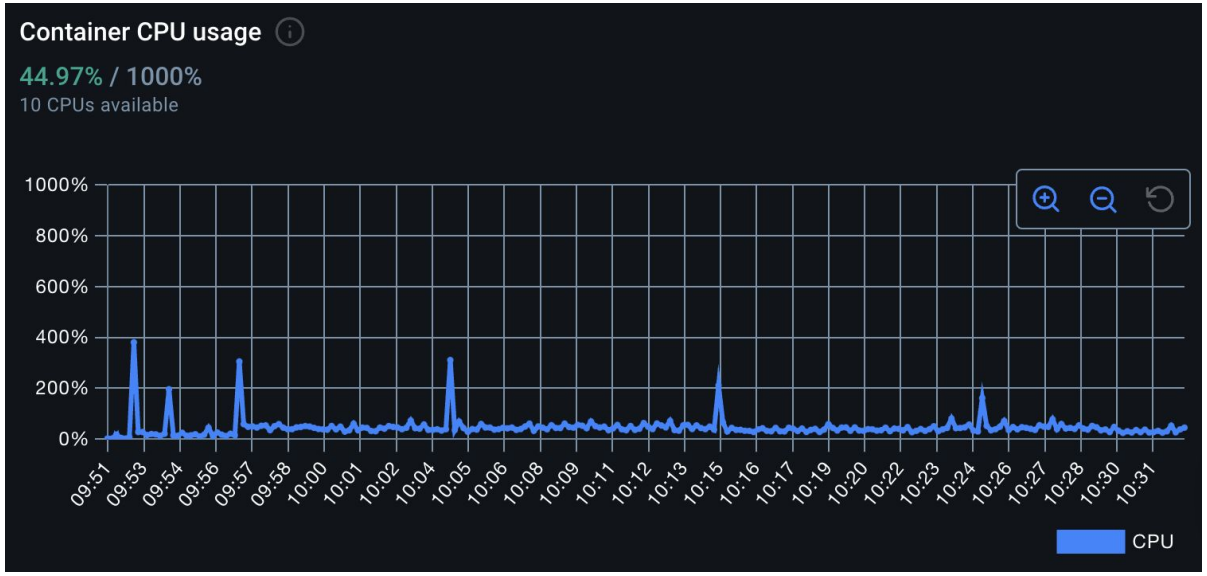
Data Producers: `sensor-service`, `weather-service`, `satellite-service`
Stream Processing: `storage-service` (3-zone medallion architecture), `crop-disease-service` (threshold alerting), `ml-anomaly-service` (K-means clustering)
Caching & Visualization: `redis-cache-service`, `dashboard` (Streamlit with 5 specialized pages)
Infrastructure: Apache Spark cluster (Master + 2 Workers), Kafka, MinIO, PostgreSQL, Redis

How to Run the Project

Clone repository: `git clone https://github.com/stanghee/Crop-Disease-Prediction-and-Alert-System.git`
Configure APIs: Create `.env` file with `WEATHER_API_KEY`, `COPERNICUS_CLIENT_ID`, `COPERNICUS_CLIENT_SECRET`
Start system: `docker-compose up -d`
Access Interfaces:Dashboard(main interface):http://localhost:8501, MinIO console:http://localhost:9001, Spark Cluster:http://localhost:8080/
Stop system: `docker-compose down -v` (includes volume cleanup)

Results & Performance (1/3)

- **System Memory Durability**: The memory usage graph shows a rapid initialization phase, with memory consumption rising quickly to approximately 5.5GB, followed by a gradual and stable increase, plateauing around 6.67GB out of 7.47GB available. This indicates stable operation with no memory leaks or unexpected drops.
- **CPU Durability:** The system utilizes an average of 45% CPU (out of 1000% total, corresponding to 10 available cores), with periodic spikes reaching up to 400% during intensive processing tasks. These spikes are short-lived and return quickly to baseline, demonstrating efficient CPU resource allocation and the ability to handle bursty workloads.
- **Memory Analysis**: Memory usage remains consistently high but stable after the initial ramp-up, suggesting effective memory management and absence of excessive garbage collection or memory fragmentation. The system makes full use of allocated resources without exceeding limits.
- **CPU Analysis**: CPU utilization displays regular, predictable spikes, which are primarily associated with the training and inference phases of the machine learning model (K-means). During these periods, Spark performs intensive computations for anomaly detection, while in the remaining time, CPU usage reflects standard streaming, validation, and aggregation tasks. The overall pattern confirms that the system is capable of handling real-time workloads without overloading the available hardware resources.

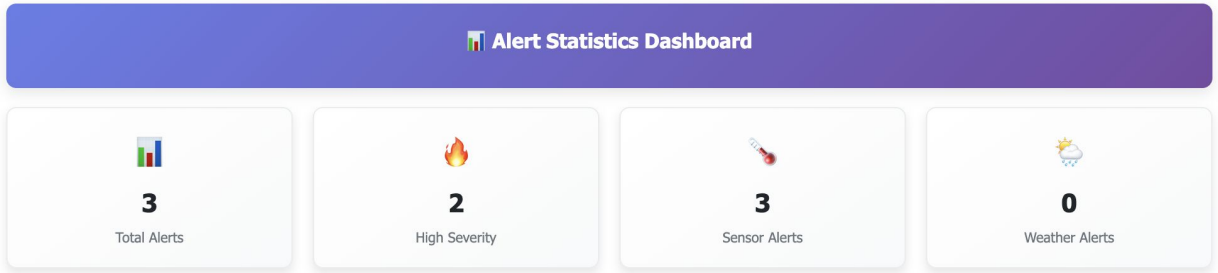


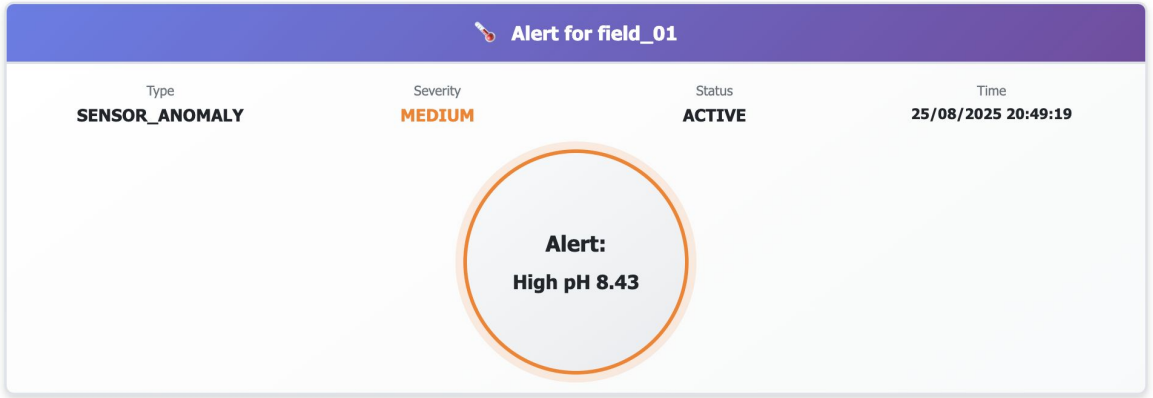
Results & Performance (2/3)

Threshold Alerts

This page is used to display alerts generated according to predefined thresholds.

- The purpose of this system is to enable farmers to quickly identify any anomalies caused by the exceeding of specific thresholds.
- In the first image, we can see an overview of the alerts, providing a complete picture.
- Meanwhile, as shown in the second and third images, we proposed alert notifications to give precise information about the severity and the current value compared to the exceeded threshold.





Results & Performance (3/3)

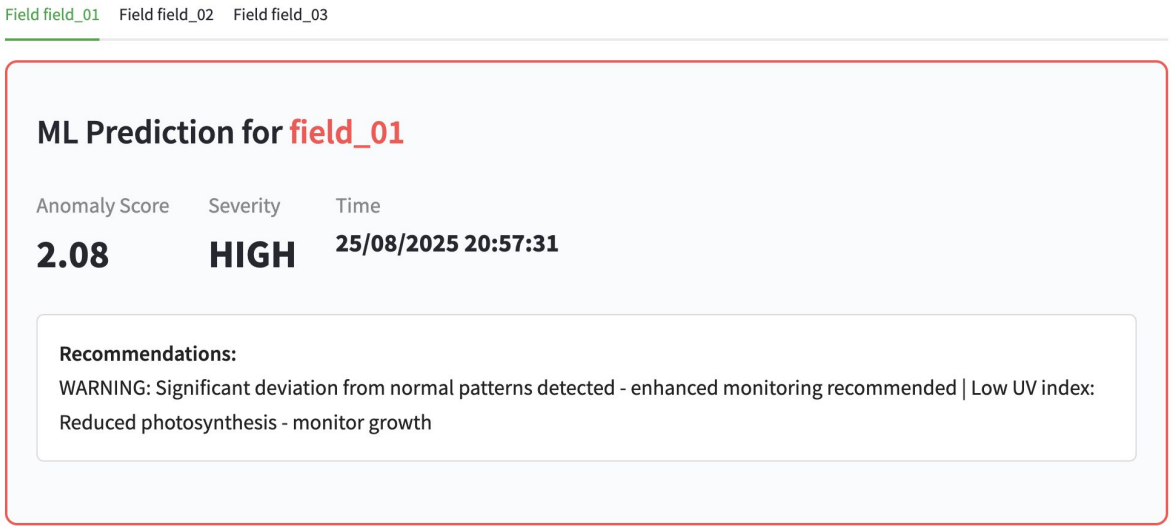
ML Insight

This page is used to display potential anomalies and recommendations generated by a machine learning model.

- Unlike the threshold alerts, which are designed to provide simple yet functional notifications, the machine learning system is intended to help farmers detect anomalies in the collected data and receive corresponding recommendations.

- In the first image, we can see notifications divided by each agricultural field. These notifications include information about the severity of the anomaly, a specific value associated with that severity, and finally, the recommendation related to the detected anomaly.

- In the second image, we can also observe the inclusion of the most significant features for the user regarding that specific alert. This allows the farmer to gain a more complete understanding of the problem.



Features

temperature	27.60
humidity	58.77
soil_ph	6.03
temp_differential	3.27
humidity_differential	1.77

Lessons Learned (1/4)

1. Choosing the Right Tools: A Strategic Approach

The selection of technologies and their integration into the project's pipeline was one of the most significant lessons we learned. A thorough understanding of each tool's features, performance, and capabilities was fundamental to keeping the project on a coherent and manageable track.

We learned that the choice is not just a matter of preference, but a strategic analysis that balances several factors:

- **Team Knowledge**: The learning curve and existing expertise within the team.
- **Project Timeline**: The speed of integration and implementation each tool offers.
- **Complexity**: The inherent difficulty in managing and maintaining the technology.

For example, complex technologies like the ones we used require a deep understanding not only of their individual functionalities but also of how they integrate with each other to ensure coherence and robustness. This experience taught us to make conscious choices that respected the team's time constraints and capabilities, demonstrating that tool selection is a crucial phase that determines the overall direction and success of the project.

2. Work Management: Collaboration and Division of Tasks

The most important lesson we learned was the importance of smart work management and effective collaboration. This was crucial, especially in balancing our different personal commitments. We understood the importance of judiciously dividing tasks, assigning each of us duties that reflected our individual qualities and abilities. This approach not only ensured a high-quality contribution from everyone, but also allowed us to optimize delivery times. Flexibility was another key factor: we worked in shifts, sometimes in pairs, and organized regular briefings to align and define the next steps. Another challenge, overcome through careful management, was the physical distance. The use of communication tools like Discord was essential to overcome geographical barriers, maintaining a constant flow of information and a sense of team.

This experience projected us into a context that, more than any other academic project, reflected key aspects of a real work environment—such as communication, teamwork, and the management of both human and technological resources. While differences remained—particularly the absence of a formal supervisor—facing real-world challenges relying on the guidance provided by the professor’s lectures made the experience valuable and formative.

Lessons Learned (2/4)

3. Time Management

A complex project always requires a balance between striving for perfection and meeting deadlines. We realized that time is the most limited resource, and not every path leads to meaningful improvements. As we moved forward, our awareness of what we had accomplished grew, along with our knowledge of the tools we used and the possible changes we could implement. This made us more conscious of how, in a real-world context, we could manage and develop more precise and effective solutions. At the same time, the experience pushed us to generate new ideas for handling real-world scenarios, but deadlines forced us to make choices: focusing on what was truly essential. For instance, in our project we would have liked to integrate a system allowing the user to set threshold values based on soil characteristics, plant type, plant age, and other relevant factors. Just like in a work environment, timelines are an integral part of the process and require managing time in a focused and effective way.

4. Access to Resources and Data Quality

Our project confirmed a fundamental principle of data science: "garbage in, garbage out." The difficulty in obtaining high-quality and open-source data presented a significant challenge, forcing us to recalibrate certain parts of our project approach. Despite having clear insights and ideas, we were unable to fully implement them due to the limited accessibility of comprehensive datasets.

- **Weather Data**: Although many open-source weather data sources exist, accessing more in-depth information for detailed analysis often requires a subscription to paid services. This financial constraint limited the granularity of our predictive analyses.
- **Satellite and Drone Imagery**: For crop health analysis, satellite imagery was a necessary substitute. Access to drone imagery is extremely complex and costly, and is typically a resource for large agricultural companies. While a viable solution, using satellite imagery prevented us from performing a detailed and specific analysis on the health of individual plants, a level of detail that drone imagery could have offered.
- **IoT Sensor Data**: The acquisition of data from IoT sensors is also a burdensome process, as it requires the physical installation of devices in the field. Consequently, high-quality data from real sensors is difficult to obtain, making the integration of such data a complex challenge in a project with limited resources.

Lessons Learned (3/4)

What worked well:

Medallion Architecture: A Solid Foundation

The implementation of the Medallion architecture proved to be solid within our project, providing a reliable foundation for data management. This structure, which divides data into zones (Bronze, Silver, Gold), greatly facilitated our work. We were able to:

- **Ensure consistency**: The data was logically cataloged and stored from the very beginning.
- **Simplify retrieval**: Information became easier to find and use for the various project purposes.
- **Accelerate development**: The clear subdivision allowed us to have the data already prepared and organized for analysis.

In short, the Medallion architecture gave us a strong and easily scalable storage base, a crucial starting point that made the entire project simpler and more manageable.

Apache Kafka

The integration of **Apache Kafka** in the project was one of the key strengths. Its message delivery mechanism between the different stages and tools of the pipeline proved extremely useful, especially in scenarios where managing data transfer directly would have been more complex. Kafka allowed us to maintain **consistency and parallelism** in the data flow while optimizing **performance and reliability**. A concrete example was the real-time management of **threshold** data: transferring information from **Spark** to **Redis** for dashboard visualization initially presented some challenges. The messages from our Spark Streaming service were unable to connect effectively to Redis, causing misalignment between the data displayed on the dashboard and the actual state of the system. Thanks to Kafka, we were able to ensure **consistency** across the project and simplify the communication flow between the threshold generation service and the dashboard, avoiding bottlenecks and streamlining the architecture.

Threshold Management

The **threshold management service** was undoubtedly one of the aspects that worked best in the project. We focused on **simplicity and immediacy of alerts**, providing clear and easily interpretable messages to give farmers an initial indication of what is happening. We drew inspiration from other industry solutions, which helped us better understand which features to integrate and how to implement them, allowing us to define **consistent and reliable values and thresholds** for crop disease monitoring. The service was designed to **coexist seamlessly with the machine learning system**, avoiding conflicts within the pipeline and ensuring no inconsistencies arose at the project level. In addition, it helped provide **clear and concise labels and guidance** for the end user. Finally, its implementation contributed to the creation of **robust historical data**.

Lessons Learned (4/4)

What didn’t work:

Machine Learning

Implementing the **Machine Learning** model proved to be a particularly complex challenge in the project. The main difficulty was the lack of **ground truth** in our dataset, which made both model training and the creation of tailored recommendations to detect issues based on the data much harder.
We experimented with integrating tools other than Spark; initially, we considered using an Isolation Forest model with scikit-learn, which was well suited for our unlabeled anomaly detection setting. However, after several implementation challenges, we opted for a **fully integrated Spark solution** using a **K-Means** model—a reasonable compromise between task relevance and pipeline consistency.
However, the challenges didn’t stop there: we had to define **appropriate metrics for anomaly detection** and account for **seasonality in the data**, which we addressed by introducing transformations based on **sine and cosine functions**.

Resource Configuration in Spark Clusters

Configuring the resources used by the services through the **Spark cluster** was quite challenging, especially at the beginning, when we didn’t yet have a clear idea of how to manage them effectively and whether it was necessary to integrate external solutions (such as **Kubernetes**) or rely on Spark’s built-in cluster functionalities.
Given the scope of the project and the goal of building a demo, we decided to go with a **Spark cluster**, creating one **master** and two **workers** to handle the memory requirements of the services.
Deciding how to **allocate resources within the cluster** was not simple: we had to determine how much memory to assign to each Spark service and how to distribute **cores** among the workers. This required careful analysis and multiple tests to identify the most suitable configuration and build a **stable and high-performing system**.

Limitations & Future Work (1/2)

Limitations of the Current System

The current implementation, while functional for a demo, comes with several important limitations. First, the system relies on **synthetic IoT sensor data** instead of real-world readings. For production, actual sensor data collected from multiple agricultural fields would be necessary to ensure accurate and context-aware insights. Another critical limitation is the **simplicity of the machine learning approach**. For demonstration purposes, we used forced alerts with low thresholds rather than deploying a fully validated model. This makes the system unreliable for real-world disease prediction and risk assessment. Similarly, the **thresholds are hardcoded and static**, meaning they are the same for all users and don't adapt to seasonality or crop-specific conditions. **Data quality and coverage are also major issues.** We only simulated **three fields**, which is clearly insufficient for real-world scenarios. The dataset also lacks critical agricultural details like crop types, soil conditions, historical yields, and management practices. This makes **personalized recommendations practically impossible** and limits scalability to large-scale agricultural operations.

We also **don't integrate drone imagery**. Currently, the system uses free satellite images, which are lower quality and not always up to date. High-resolution drone imagery—or premium satellite imagery—would significantly improve monitoring accuracy and allow for **early detection of localized issues** like pest outbreaks or irrigation problems. On the infrastructure side, the current architecture is built around a **single Spark master and two workers**, which works for small demos but isn't viable for production-scale workloads. The same applies to other components: **single-broker Kafka, single-node MinIO, single-instance Redis**, and a **Streamlit dashboard that doesn't support high concurrency**. Additionally, managing Spark's shuffle and checkpoint storage efficiently remains a challenge, as our current setup relies on local storage rather than persistent solutions like S3. Lastly, **observability and resilience are minimal**. We lack centralized metrics, proper logging, and automated data quality checks.

What Would Break First When Scaling, and Why?

When scaling beyond the current 3-field demonstration, the system would likely experience cascading failures beginning with Spark's state management infrastructure. The current architecture stores **Spark checkpoints** locally and maintains sliding window states in memory, which becomes unsustainable when processing data from hundreds or thousands of sensors. As data volumes increase exponentially, the Silver and Gold zone processors would consume excessive memory attempting to maintain 10-minute sliding windows across numerous fields, ultimately leading to out-of-memory errors and checkpoint corruption.
The **single Kafka broker configuration** represents the second critical failure point**.** Currently designed for demonstration with minimal message throughput, the system would quickly overwhelm the single broker when faced with realistic sensor deployment densities. Agricultural IoT networks typically involve dozens of sensors per field reporting multiple parameters every few minutes, creating message volumes orders of magnitude higher than the current 5 messages per minute. Without proper partitioning and replication, message loss and processing delays would cascade through the entire pipeline, making real-time alerting impossible precisely when farmers need it most. A third bottleneck is the database layer. A **single PostgreSQL instance** without replication or sharding can quickly become saturated, both in terms of write throughput (when ingesting continuous sensor data, alerts, and processed metrics) and read throughput (when serving dashboards, APIs, or decision-support systems). This central bottleneck risks stalling both the ingestion and the analytics side of the pipeline.
Finally, the **absence of a centralized monitoring and alerting system** represents a critical operational weakness. In a real-world deployment, without observability tools (e.g., Prometheus, Grafana, ELK stack) and automated alerting, it would be extremely difficult to detect anomalies in performance, resource saturation, or component failures before they escalate into service outages.

Limitations & Future Work (2/2)

Potential Improvements based on time and resources

Data & Coverage [Resources]
We'll replace synthetic inputs with **real IoT sensor streams** and expand beyond the current **three fields** to broader coverage. Alongside this, we'll enrich datasets with **agronomic context** (crop types, soil characteristics, historical yields, management practices) and enforce **data contracts and quality checks** end-to-end so recommendations can be truly field-specific and reliable.

ML & Analytics [Time & Resources]
We'll retire forced/low thresholds and adopt **adaptive, per-user thresholds** that account for **seasonality** and crop requirements. On the modeling side, we'll introduce **more advanced anomaly detection** (e.g., distance-based on robust embeddings, tree-based methods, or neural approaches like autoencoders) and transition to **supervised models** when ground truth becomes available. We'll also add proper **MLOps**: model registry and versioning, drift/health monitoring, scheduled retraining on Gold data, and a lightweight **feature store** to keep features consistent across training and inference.

Imagery & Remote Sensing [Resources]
We'll integrate **drone imagery** and/or **premium satellite feeds** to complement free satellite sources, enabling higher-resolution and more frequent observations. On top of that, we'll compute **vegetation indices** (e.g., NDVI/EVI/NDWI) and add **spectral analytics** tailored to crop disease signals, with an image-focused ML component to flag early stress patterns.

Alerts & User Experience [Time & Resources]
Alerts will move beyond the dashboard with **external notifications** (SMS, messaging apps, voice) including escalation policies, de-duplication, and rate limiting. We'll explore a **mobile app** for in-field use (acknowledge alerts, view maps, attach photos), and improve the dashboard for **actionable insights**: clearer severity, recommended next steps, and per-field history.

Scalability & Platform [Time & Resources]
We'll run Spark on **Kubernetes** for elastic resource management, dynamic allocation, and better fault tolerance. Spark **state/shuffle and checkpoints** will use persistent object storage (MinIO/S3). **Kafka** will scale to multiple brokers with increased partitions and replication. **PostgreSQL** will gain time-series-friendly partitioning (or TimescaleDB if needed). **MinIO** will move to a distributed setup (or managed S3) with lifecycle policies; **Redis** to a clustered, persistent deployment with appropriate TTLs. For the UI we'll introduce a more scalable presentation layer.

Observability, Reliability & DevOps [Time]
We'll add centralized **metrics, logs, and tracing** (e.g. DataDog, Sentry), plus **data quality validation** in the pipeline (expectations on Silver/Gold). Regular **backups**, disaster-recovery.

References & Acknowledgments

References

- Medallion Architecture (Databricks Glossary)
- What is the medallion lakehouse architecture? (Databricks Docs)
- Kmeans Documentation (Apache spark)
- Wolfert, S., Verdouw, C. N., & Bogaardt, M. J. (2017). Big data in smart farming – A review. Agricultural Systems, 153, 69–80.
- Kaloxylos, A., Eigenmann, R., Teye, F., Politopoulou, Z., Wolfert, J., Shrank, C., Dillinger, M., Lampropoulou, I., Antoniou, E., Pesonen, L., Nicole, H., Thomas, F., Alonistioti, N., & Kormentzas, G. (2016). Farm management systems and the future Internet era. Computers and Electronics in Agriculture, 89, 130–144.
- Chergui, N., & Kechadi, M. T. (2022). Data analytics for crop management: a big data view. Journal of Big Data, 9(123).