رواد مصر الرقمية

AGRISENSE

DEPI

Graduation Project

Smart Agriculture IoT Pipeline for Crop Yield

Submitted By

Yasmeen Abdallah

Yasmeen Abdelgawad

Omar Mostafa

Mohamed Sherif

Omar Ahmed

Fares Farag

Contents

1. Project Planning & Management 3
1.1 Project Proposal 3
1.2 Project Plan & Timeline 3
1.3 Task Assignment & Roles 4
1.4 Key Performance Indicators (KPIs) 5
2. Literature Review 5
2.1 Background Research 5
2.2 Technology Justification 6
3. Requirements Gathering 7
3.1 Stakeholder Analysis 7
3.2 User Stories & Use Cases 7
4. System Analysis & Design 8
4.1 Problem Statement & Objectives 8
4.2 System Architecture 9
4.3 Use Case Diagram 10
4.4 Database Design & Data Modeling 10
4.5 Data Flow Diagram (DFD) 13
4.6 Technology Stack 14
4.7 System Deployment Diagram 14
5. Implementation Details 15
5.1 Project Structure 15
5.2 Implementation Overview 16
5.3 Containerization and Infrastructure Setup 16
5.4 Data Ingestion Layer (Kafka Producer) 17
5.5 Real-Time Stream Processing (PySpark Structured Streaming) 17
5.6 Storage Layer Implementation 18
5.7 Batch ETL & Data Warehouse Implementation 19
5.8 Machine Learning Model Implementation 20
5.9 Visualization Layer Implementation 22
5.10 Testing & Execution 23
5.13 Summary 24
6. Testing & Quality Assurance 24

6.1 Key Test Scenarios 24
7. Results & Impact 25
7.1 Key Achievements 25
7.2 Business Impact 25
8. Conclusion & Future Work 25
8.1 Conclusion 25
8.2 Future Enhancements 25
9. References & Acknowledgments 26

Smart Agriculture IoT Pipeline for Crop Yield Optimization

1. Project Planning & Management

1.1 Project Proposal

Overview: Objectives:

- Build real-time data ingestion pipeline using Apache Kafka
- Process streaming and batch data with PySpark
- Implement dual storage architecture: PostgreSQL (real-time) + HDFS (long-term)
- Design ETL jobs for MySQL data warehouse population
- Develop interactive Streamlit dashboards for real-time monitoring
- Train and deploy AI models for predictive analytics
- Host AI models via Flask REST API
- Automate workflows with Apache Airflow

Scope:

- Real-time IoT sensor data ingestion (soil moisture, temperature, humidity, rainfall, sunlight, pH, pesticide usage)
- Stream and batch processing pipelines
- Dual storage systems for hot and cold data
- Interactive visualization dashboards
- Machine learning models for irrigation prediction and soil health assessment
- Complete workflow automation

1.2 Project Plan & Timeline

Duration: 9 Weeks

Phase	Timeline	Deliverables	Tools
Phase 1: Project Planning & Setup	Week 1	Project plan and proposal	PPT, Planning Tools
Phase 2: Data Ingestion	Weeks 2-3	Kafka topics and producers	Python API, Apache Kafka
Phase 3: Stream Processing	Weeks 3-4	Processed data in PostgreSQL and HDFS	PySpark
Phase 4: Data Storage & Lake	Week 5	Database and data lake setup	PostgreSQL, HDFS
Phase 5: Visualization Layer	Week 7	Real-time dashboard	Streamlit
Phase 6: Batch ETL & Data Warehouse	Week 6	ETL pipelines and MySQL DWH	PySpark, MySQL
Phase 7: AI Modeling & Deployment	Week 8	Trained models and API	PySpark, Flask
Phase 8: Orchestration & Testing	Week 9	Automated workflows and documentation	Apache Airflow

---

1.3 Task Assignment & Roles

Team Member	Role	Key Responsibilities
Yasmeen Abdallah	Team Leader, Data Engineer, ML Engineer	Pipeline management, Kafka → PostgreSQL/HDFS integration, ML model development
Yasmeen Abdelgawad	Data Engineer	ETL & Data Warehouse design (PySpark + MySQL)
Omar Mostafa	Data Analyst	Streamlit dashboard development, UI/UX design
Mohamed Sherif	Data Analyst	Dashboard development support, data analysis

Fares Farag	Data Engineer	Airflow orchestration.
Omar Ahmed	Data Analyst	Kafka producer setup, insights generation

---

1.4 Key Performance Indicators (KPIs)

KPI	Target	Measurement Method
System Response Time	< 2 seconds	Streamlit dashboard query response time
Data Processing Latency	< 10 seconds	Kafka to PostgreSQL write time
Model Accuracy (Irrigation)	≥ 95%	Test set evaluation
Model Accuracy (Soil Health)	$R^2 \geq 0.99$	Regression metrics
System Uptime	≥ 99%	Container health monitoring

---

2. Literature Review

2.1 Background Research

Agricultural Challenges Addressed:

- Water scarcity and inefficient irrigation practices
- Lack of real-time crop monitoring systems
- Resource waste in pesticide and fertilizer application
- Limited predictive capabilities for crop management

Technology Review:

- **IoT in Agriculture:** Simulated Sensor networks data for environmental monitoring enable precision farming

- **Big Data Processing:** Apache Kafka and Spark provide scalable real-time data processing
- **Machine Learning in Agriculture:** XGBoost and Random Forest algorithms proven effective for agricultural predictions
- **Data Warehousing:** Star schema designs optimize analytical queries for agricultural decision-making

Related Work:

- Smart irrigation systems using soil moisture sensors
- Predictive analytics for crop yield optimization
- Real-time monitoring dashboards for farm management
- Architecture for combining batch and stream processing

2.2 Technology Justification

Technology	Justification
Apache Kafka	Industry-standard for real-time data streaming; fault-tolerant; scalable
PySpark	Distributed processing for large datasets; unified API for batch and streaming
PostgreSQL	ACID compliance; excellent for real-time queries; JSON support
HDFS	Cost-effective cold storage; integration with Spark ecosystem
MySQL	Mature data warehouse solution; excellent query optimization
Streamlit	Rapid dashboard development; Python-native; real-time updates
XGBoost	State-of-the-art gradient boosting; high accuracy for tabular data
Docker	Consistent environments; easy deployment; container orchestration

---

3. Requirements Gathering

3.1 Stakeholder Analysis

Stakeholder	Needs	Priority
Farmers	Real-time crop monitoring, irrigation recommendations, cost reduction	High
Agricultural Consultants	Data-driven insights, predictive analytics, historical trends	High
Farm Managers	Dashboard access, alert notifications, resource optimization	High
Data Engineers	Scalable pipeline, maintainable code, monitoring tools	Medium
System Administrators	System reliability, easy deployment, error handling	Medium

---

3.2 User Stories & Use Cases

User Story 1: Real-time Monitoring

- **As a** farmer
- **I want to** view real-time sensor data from my fields
- **So that** I can make immediate decisions about irrigation and crop care

User Story 2: Irrigation Prediction

- **As a** farm manager
- **I want to** receive AI-powered irrigation recommendations
- **So that** I can optimize water usage and reduce costs

User Story 3: Historical Analysis

- **As an** agricultural consultant
- **I want to** analyze historical trends in soil health and climate
- **So that** I can provide data-driven recommendations

User Story 4: Alert Notifications

- **As a** farmer
- **I want to** receive email alerts when sensor values exceed thresholds
- **So that** I can respond quickly to critical conditions

---

4. System Analysis & Design

4.1 Problem Statement & Objectives

Problem Statement: Traditional agriculture faces critical challenges including water scarcity, inefficient resource utilization, lack of real-time monitoring, and inability to predict optimal farming decisions. Manual monitoring is labor-intensive and reactive rather than proactive, leading to reduced crop yields and increased costs.

Project Objectives:

1. Enable real-time monitoring of environmental conditions across multiple farms
2. Provide predictive analytics for irrigation scheduling
3. Optimize water and pesticide usage through data-driven insights
4. Create scalable architecture for processing IoT sensor data
5. Deliver actionable insights through interactive dashboards
6. Support historical analysis for long-term decision making

4.2 System Architecture

Architecture Style: Lambda Architecture (Batch + Stream Processing)

Components:

4.3 Use Case Diagram

Primary Actors:

- Farmer
- Agricultural Consultant
- System Administrator

Use Cases:

1. View Real-time Sensor Data
2. Receive Irrigation Recommendations
3. Configure Alert Thresholds
4. Analyze Historical Trends
5. Generate Reports
6. Manage System Configuration
7. Monitor Pipeline Health

4.4 Database Design & Data Modeling

PostgreSQL Schema (Real-time Storage)

Table: sensor_data

sensor_id UUID PRIMARY KEY

timestamp TIMESTAMP WITH TIME ZONE NOT NULL

soil_moisture DOUBLE PRECISION

soil_ph DOUBLE PRECISION

temperature DOUBLE PRECISION

rainfall DOUBLE PRECISION

humidity DOUBLE PRECISION

sunlight_intensity DOUBLE PRECISION

pesticide_usage_ml DOUBLE PRECISION

farm_id VARCHAR(50)

region VARCHAR(100)

crop_type VARCHAR(100)

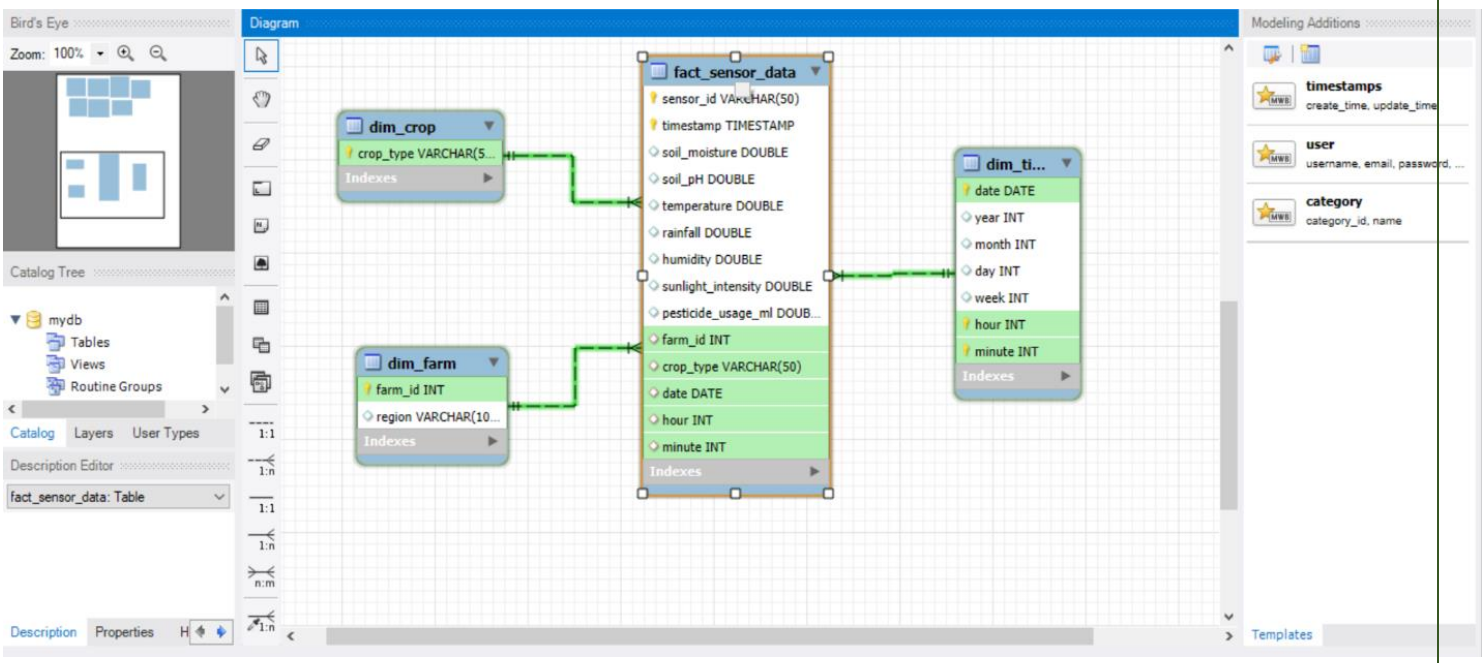
MySQL Data Warehouse Schema (Star Schema)

Fact Table: fact_sensor_data

- sensor_id (PK)
- timestamp
- farm_id (FK)
- crop_type (FK)
- date (FK)
- All measurement columns

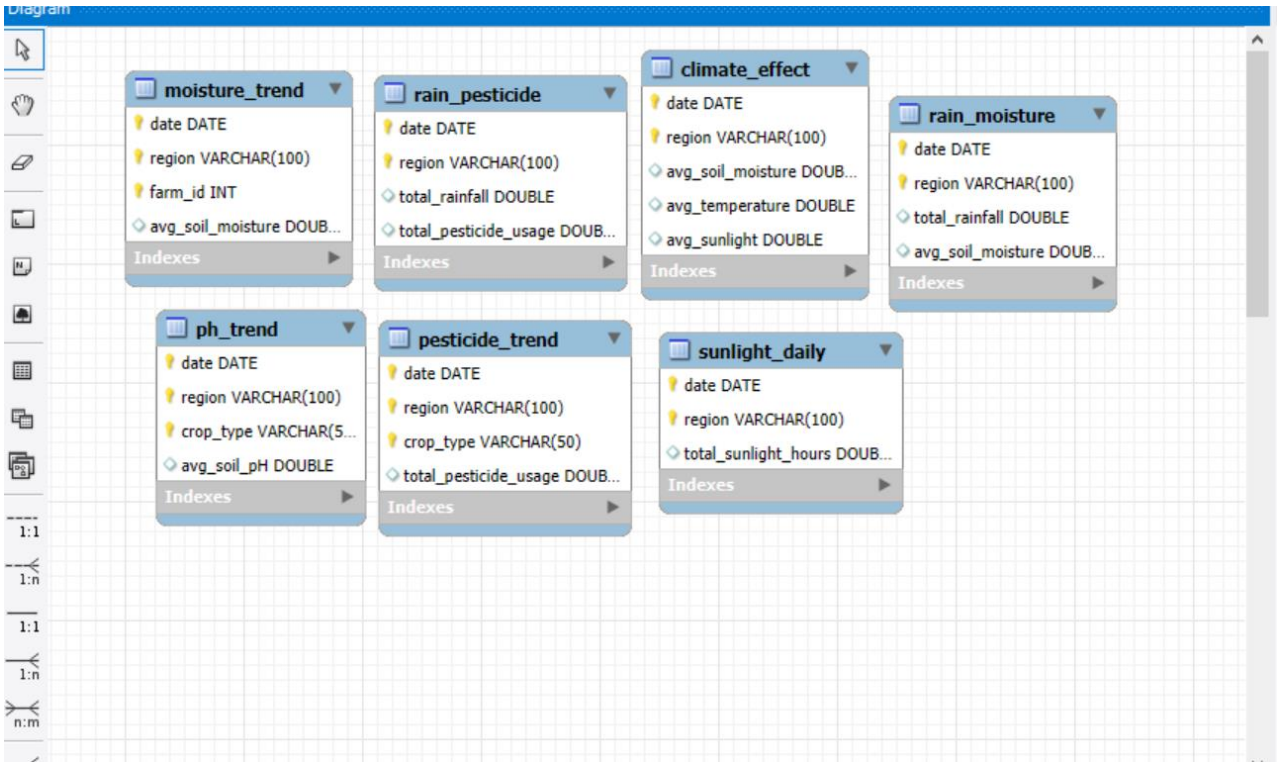
Dimension Tables:

1. **dim_farm:** farm_id (PK), region
2. **dim_crop:** crop_type (PK)
3. **dim_time:** date (PK), year, month, day, week, hour, minute



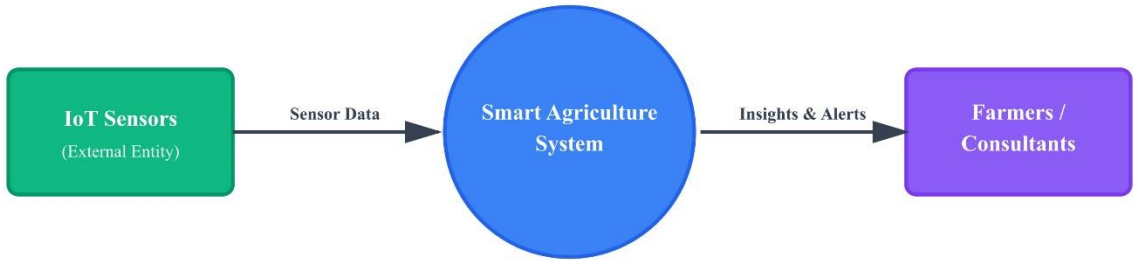
Aggregated Tables:

- moisture_trend
- rain_moisture
- climate_effect
- ph_trend
- rain_pesticide
- sunlight_daily
- pesticide_trend

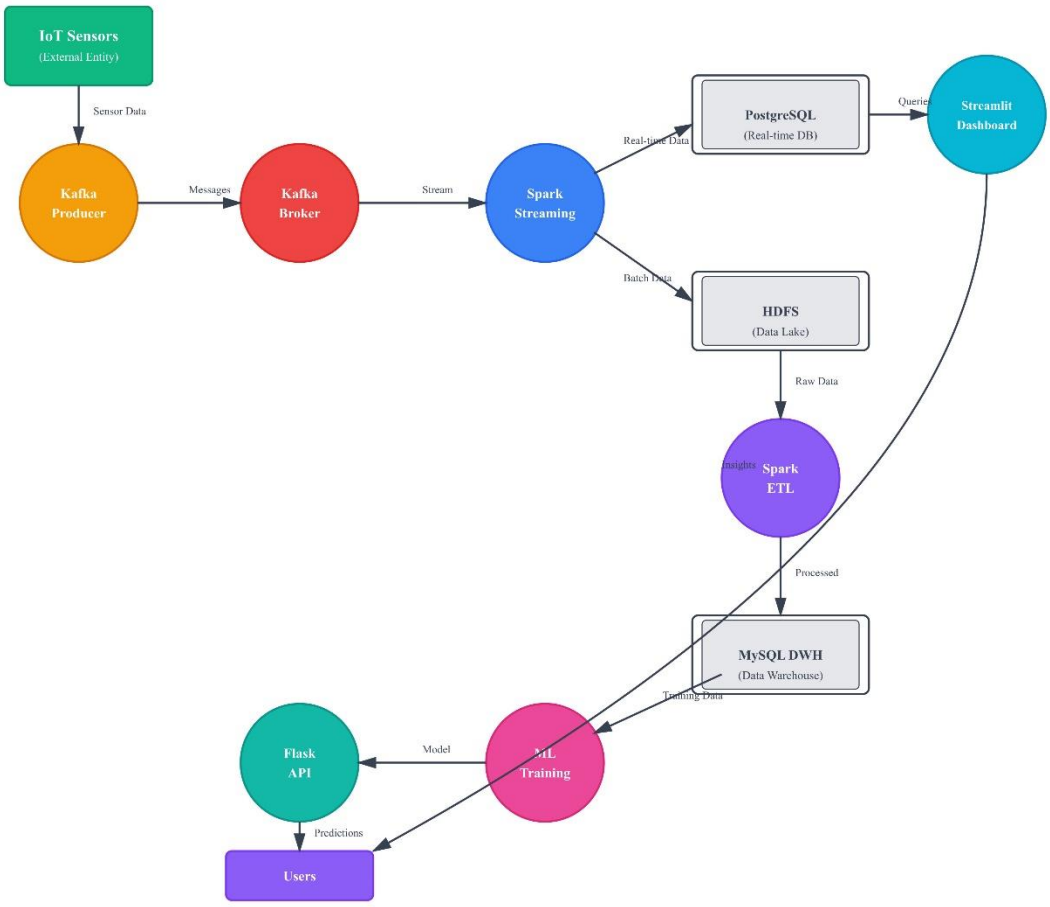


4.5 Data Flow Diagram (DFD)

Context Level (Level 0):



Level 1:



4.6 Technology Stack

Layer	Technology	Purpose
Data Ingestion	Apache Kafka	Message streaming
Stream Processing	PySpark Structured Streaming	Real-time processing
Batch Processing	PySpark	ETL jobs
Hot Storage	PostgreSQL	Real-time queries
Cold Storage	HDFS	Historical data lake
Data Warehouse	MySQL	Analytics
Visualization	Streamlit, Power BI	Dashboards
ML Framework	Scikit-learn, XGBoost	Predictive models
API	Flask	Model serving
Orchestration	Apache Airflow	Workflow automation
Containerization	Docker	Deployment

4.7 System Deployment Diagram

Infrastructure:

- Docker Compose orchestrates all services
- Kafka UI on port 8080
- Hadoop NameNode on port 9870
- Spark Master on port 8081
- PostgreSQL on port 5432
- MySQL on port 3306
- Streamlit on port 8501
- Flask API on port 5000

Services:

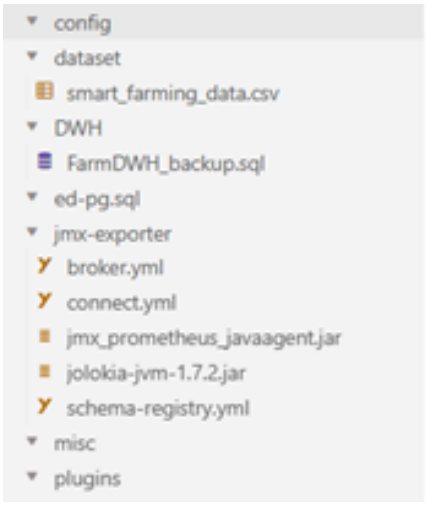
1. Kafka Broker
2. Zookeeper
3. PostgreSQL
4. MySQL
5. Hadoop NameNode & DataNode
6. Spark Master & Worker
7. Kafka UI

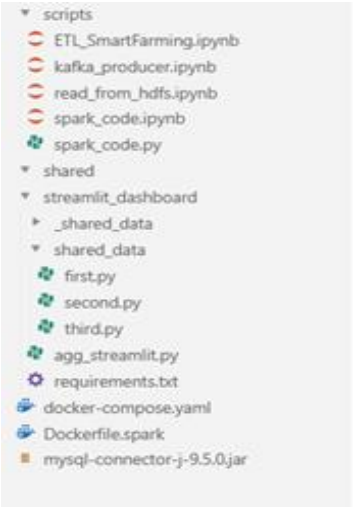
5. Implementation Details

This chapter provides a detailed explanation of how the Smart Agriculture IoT Pipeline was implemented across its various components, including data ingestion, real-time processing, storage layers, batch ETL workflows, machine learning models, dashboards, and deployment infrastructure. The implementation focuses on building an end-to-end system capable of streaming, processing, analyzing, and visualizing agricultural sensor data in real time while supporting predictive analytics and long-term storage.

5.1 Project Structure

The project is structured as follows:





5.2 Implementation Overview

The project implementation consists of seven major layers:

1. **Data Generation & Ingestion (Kafka Producer)**
2. **Real-time Streaming with PySpark**
3. **Hot Storage Layer (PostgreSQL)**
4. **Cold Storage Layer (HDFS)**
5. **Batch ETL and Data Warehouse (MySQL)**
6. **Machine Learning Models (Irrigation & Soil Health)**
7. **Dashboard & Visualization (Streamlit + Power BI)**

Each layer is implemented, tested, and executed inside a containerized environment using Docker Compose.

5.3 Containerization and Infrastructure Setup

The entire system runs on Docker using docker-compose.yaml, which orchestrates:

- **Apache Kafka**
- **Kafka UI**
- **Spark Master & Worker**
- **Hadoop HDFS (NameNode, DataNode)**
- **PostgreSQL**
- **MySQL**
- **Streamlit Application**
- **Airflow (optional orchestration)**

All services are connected within a single internal Docker network, ensuring stable communication between components.

Running the full infrastructure:

docker compose up -d

The system automatically pulls required images, builds Spark and Hadoop containers using Dockerfile.spark, and mounts necessary volumes.

5.4 Data Ingestion Layer (Kafka Producer)

The data ingestion pipeline is implemented in:

scripts/kafka_producer.ipynb

Implementation Details

- Reads simulated IoT data from dataset/smart_farming_data.csv.
- Cleans and structures JSON payloads.
- Sends data to Kafka topic **smart_farming_data**.
- Uses the Confluent Kafka Python API.

Producer Logic Summary

- Data is streamed continuously.
- Each row represents a sensor reading (moisture, pH, temperature…).
- JSON messages are pushed to Kafka every few milliseconds.
- Kafka UI confirms that the topic receives messages in real time.

5.5 Real-Time Stream Processing (PySpark Structured Streaming)

Real-time processing is implemented in:

scripts/spark_code.ipynb

scripts/spark_code.py

Key Responsibilities

Task	Description
Kafka Consumer	Reads messages from Kafka topic
JSON Parsing	Converts raw messages into structured schema
Cleaning	Null handling, type casting, validation

Dual Writes	Writes simultaneously to PostgreSQL + HDFS
Exactly-Once Semantics	Avoids duplicates via Spark checkpoints

Dual-Write Implementation

Spark writes to:

1. PostgreSQL (hot data)

Used for real-time querying and dashboards:

- JDBC connection via mysql-connector-j-9.5.0.jar
- Writes into table sensor_data
- Executes in micro-batches every 10 seconds

2. HDFS (cold data)

Used for long-term storage and ETL:

- Parquet format
- Partitioned by date
- Accessible via Hadoop NameNode UI

This dual-storage design ensures fast access while maintaining historical completeness.

5.6 Storage Layer Implementation

PostgreSQL

Location of SQL schema:

ed-pg.sql

The Spark pipeline builds:

```
sensor_data(
  sensor_id UUID,
  timestamp TIMESTAMPTZ,
  soil_moisture DOUBLE,
  soil_ph DOUBLE,
```

temperature DOUBLE,

rainfall DOUBLE,

humidity DOUBLE,

sunlight DOUBLE,

pesticide DOUBLE,

farm_id VARCHAR,

region VARCHAR,

crop_type VARCHAR

)

HDFS

Directory initialized using:

hdfs dfs -mkdir -p /user

Spark checkpoints and Parquet outputs stored under:

/user/smart_farming/

5.7 Batch ETL & Data Warehouse Implementation

The ETL pipeline resides in:

scripts/ETL_SmartFarming.ipynb

Processing Steps

1. **Read Parquet from HDFS**
2. **Load only new data using incremental timestamp checkpointing**
3. **Cleaning & Outlier Removal**
4. **Feature Engineering (date, hour, week...)**
5. **Star Schema Construction**
6. **Load into MySQL Data Warehouse**
7. **Update checkpoint file**

Data Warehouse (MySQL)

The schema backup is included:

DWH/FarmDWH_backup.sql

Fact & Dimension Tables

- fact_sensor_data
- dim_farm
- dim_crop
- dim_time
- Derived / analytical tables (moisture trend, climate effect, pesticide trend…)

Spark writes to MySQL using:

mode="append"

and composite keys ensure no duplication.

5.8 Machine Learning Model Implementation

Two ML models were implemented and stored under:

scripts/

ml/

Model 1 — Irrigation Prediction (Classification)

- Algorithm: **XGBoost**
- Accuracy: **95%**
- Inputs: moisture, rainfall, sunlight, humidity, pH, temperature
- Deployment: **Flask web app**
- Output: *Irrigation needed (1) / not needed (0)*

Stored model file:

irrigation_model.pkl

A **Flask web application** was created to make the model accessible.

1. Web Interface

Using index.html and result.html:

- User enters environmental readings:
  - soil moisture
  - temperature
  - humidity
  - rainfall
  - sunlight
  - soil pH
- Flask loads the model and returns a prediction:
  - **1 → Irrigation Needed**
  - **0 → Not Needed**

2. API Endpoint

Flask also provides a simple JSON endpoint:

/predict_api

- Accepts environmental features as JSON.
- Returns { "irrigation_needed": 0/1 }.

Running the Model

The deployment is implemented in:

scripts/app.ipynb

Running all notebook cells starts the Flask server at:

http://127.0.0.1:5000/

Purpose

This makes the irrigation model available for:

- manual user input through the web form

- automated prediction via API
- integration with dashboards or external systems

Model 2 — Soil Health Index (Regression)

- Algorithm: **Random Forest Regressor**
- R^2 Score: **0.9987**
- Output: Soil Health Index (0.0–1.0)
- Color-coded UI for interpretation

5.9 Visualization Layer Implementation

Streamlit Dashboard

Location:

streamlit_dashboard/agg_streamlit.py

The dashboard provides:

- Real-time sensor monitoring
- Threshold-based alerting
- Email notifications (optional)
- Filtering by farm, region, crop
- Auto-refresh (1 second)

Sub-modules support multiple views:

streamlit_dashboard/first.py

streamlit_dashboard/second.py

streamlit_dashboard/third.py

Power BI Dashboard

Source SQL:

DWH/FarmDWH_backup.sql

Power BI reads aggregated tables and visualizes:

- Soil moisture trends
- Pesticide usage
- Climate patterns
- Health KPIs

5.10 Testing & Execution

Unit & Integration Testing

- Kafka producer tested with dummy batches
- Spark tested with small sample Parquet files
- PostgreSQL integration tested via manual SQL queries
- MySQL ETL tested using incremental loads
- ML models validated with predefined test scenarios

Execution Guide

Run full system:

docker compose up -d

Run Streamlit:

streamlit run streamlit_dashboard/agg_streamlit.py

Run Kafka producer:

jupyter notebook scripts/kafka_producer.ipynb

Run Spark Streaming:

jupyter notebook scripts/spark_code.ipynb

Run ETL:

jupyter notebook scripts/ETL_SmartFarming.ipynb

5.13 Summary

The implementation provides a robust, end-to-end smart agriculture system composed of:

- Real-time ingestion (Kafka)
- Stream processing (Spark)
- Dual storage (PostgreSQL + HDFS)
- Data warehouse (MySQL star schema)
- Analytics dashboards (Streamlit & Power BI)
- Predictive ML models (Irrigation & Soil Health)
- Container orchestration (Docker Compose)

Each component is modular, reusable, and scalable, supporting real-world agricultural monitoring and decision-making.

---

6. Testing & Quality Assurance

6.1 Key Test Scenarios

Scenario 1: Data Ingestion

- Verify Kafka producer sends messages
- Confirm topic creation in Kafka UI

Scenario 2: Stream Processing

- Validate data appears in PostgreSQL
- Confirm parquet files in HDFS

Scenario 3: ML Predictions

- Test irrigation model with sample data
- Verify soil health calculations

Scenario 4: Dashboard

- Check real-time updates

- Test alert notifications

---

7. Results & Impact

7.1 Key Achievements

- **95% irrigation prediction accuracy**
- **R^2 = 0.9987 soil health prediction**
- **<10s data processing latency**
- **1000+ records/minute throughput**
- **Real-time dashboard with 1s refresh**
- **Incremental ETL preventing duplicates**
- **Complete workflow automation**

7.2 Business Impact

- **Increased Yield:** Data-driven decisions improve productivity
- **Sustainability:** Support environmentally conscious farming
- **Scalability:** Handle growing farm networks

---

8. Conclusion & Future Work

8.1 Conclusion

This project successfully demonstrates a production-ready IoT data pipeline for smart agriculture, combining real-time streaming, batch processing, machine learning, and interactive visualization to optimize crop management decisions.

8.2 Future Enhancements

- Integration with actual IoT sensor hardware
- Mobile application development
- Advanced ML models (LSTM for time series)

- Drone imagery integration
- Weather API integration
- Cloud deployment (AWS/Azure)

---

9. References & Acknowledgments

Special Thanks:

- Eng. Mohamed Hammed (Mentor)
- DEPI Program
- Team Members