Project Name: FarmSense | **Document Type:** Technical Architecture Specification | **Version:** 1.0 | **Date:** February 2026

Project FarmSense Architecture Specification

Table of Contents

1. Executive Summary & System Overview

Project FarmSense is a full-stack precision agriculture application designed to bridge the gap between raw field telemetry and actionable regulatory compliance. The system ingests data from heterogeneous sources—soil sensors, pump telemetry, weather stations, and satellite imagery—to compute high-resolution virtual sensor networks. By leveraging a dual-layer compute architecture (Edge 20m grid vs. Cloud 1m grid), FarmSense delivers real-time operational insights for farmers while ensuring rigorous audit-ready logging for regulatory bodies.

Key Objectives:

- **Precision:** Downscaling satellite and sparse sensor data to a 1m resolution grid.
- **Responsiveness:** Utilizing an adaptive recalculation engine that shifts between 12-hour trends and critical 1-minute updates.
- **Compliance:** Automating SLV 2026 regulatory reporting with non-repudiable audit logs.

2. System Architecture Overview

The FarmSense platform utilizes a tiered Lambda architecture extended to the edge. This hybrid approach ensures low latency for field operations while maintaining the immense computational capacity required for high-resolution spatial interpolation in the cloud.

```
                     High-Level System Topology

                          [ FIELD LAYER ]
                  Sensors | Pumps | Weather Stations
                        ⬇ (LoRaWAN / MQTT)
                         [ EDGE GATEWAY ]
       Data Buffering | 20m Grid Compute | Critical Alerting
                 ⬇ (HTTPS / Secure WebSocket)
                    [ CLOUD INGESTION LAYER ]
           Load Balancers | API Gateway | Kafka Event Bus
                                ⬇
                        [ PROCESSING CORE ]
     Stream Processing (Flink) | Batch Processing (Airflow)
                                ⬇
                       [ DATA PERSISTENCE ]
               Time-Series DB | Spatial DB | Data Lake
                                ⬇
                      [ APPLICATION LAYER ]
          Dashboard API | Compliance Engine | ML Inference
```

3. Data Layer Architecture

The data layer handles the ingestion, validation, and storage of high-velocity time-series data and high-volume geospatial rasters.

Data Sources & Ingestion

Source	Protocol	Frequency	Data Type
Soil Moisture Probes	MQTT / LoRaWAN	15 min	Volumetric Water Content (2 depths)
Vertical Profiling Sensors	MQTT / LoRaWAN	30 min	Multi-depth moisture profile (4-8 levels)
Pump Telemetry	Modbus TCP / MQTT	Real-time (1-5 sec)	Flow rate, Power status, Pressure, Voltage
Weather Stations	REST API / MQTT	10 min	Temperature, Humidity, Rainfall, Wind, Solar Radiation
Sentinel-1 (SAR)	Copernicus Open Access Hub API	6 days (with both satellites)	Backscatter coefficient (soil moisture proxy)
Sentinel-2 (Optical)	Copernicus Open Access Hub API	5 days (with both satellites)	Multispectral (NDVI, NDRE, NDWI, LAI)
Landsat 8/9	USGS Earth Explorer API	16 days	Thermal/Optical (Historical baseline, 2013-present)

Data Preprocessing Pipeline

Raw data undergoes a multi-stage preprocessing pipeline before storage:

1. **Quality Control:** Flagging out-of-range values, stuck sensors, and communication errors.
2. **Temporal Alignment:** Synchronizing data from different sources to common timestamps.
3. **Atmospheric Correction:** Sentinel-2 imagery processed with Sen2Cor for surface reflectance.
4. **Cloud Masking:** Automated cloud detection using QA bands and ML-based classifiers.

5. **Landsat Index Computation:** Pre-computing NDVI, NDMI, LST (Land Surface Temperature) for calibration.

> **Storage Strategy:**
>
> - **Hot Tier (Redis/InfluxDB):** Last 24 hours of sensor data for real-time dashboards.
> - **Warm Tier (PostgreSQL/TimescaleDB):** Current season data for trend analysis and compliance generation.
> - **Cold Tier (S3/Glacier):** Raw satellite imagery and historical logs > 3 years.

4. Edge Computing Layer (20m Grid)

The edge layer is designed to operate with intermittent connectivity. It processes raw telemetry locally to generate a coarse 20m virtual grid, ensuring farmers have visibility even if the cloud connection is severed.

4.1 Edge Hardware Specifications

Component	Specification	Purpose
Compute Module	Raspberry Pi 4 (4GB) or NVIDIA Jetson Nano	Main processing unit for local grid computation
Storage	64GB Industrial microSD + 128GB SSD	OS, application, 30-day data buffer
Connectivity	4G LTE modem + Ethernet backup	Cloud synchronization and remote monitoring
Sensor Interface	LoRaWAN Gateway (8-channel) + MQTT broker	Collect data from field sensors
Power	12V DC with UPS backup (6hr capacity)	Continuous operation during power outages

4.2 Edge Processing Architecture

- **Local Compute Engine:** Lightweight IDW (Inverse Distance Weighting) interpolation for 20m grid.

- **Data Buffer:** SQLite database storing 30 days of telemetry with automatic compression.
- **Alert Engine:** Rule-based system for critical condition detection (moisture < wilting point).
- **Synchronization:** "Store and Forward" mechanism with exponential backoff retry logic.
- **Conflict Resolution:** Timestamp-based priority; cloud-generated data takes precedence.

4.3 Edge Software Stack

- **Language:** Go (compiled binary ~15MB) for minimal footprint and fast startup.
- **Local Web Server:** Embedded HTTP server for local dashboard access via farm WiFi.
- **Sensor Drivers:** Modbus RTU/TCP, MQTT client libraries for diverse sensor integration.
- **Containerization:** Docker Compose for easy deployment and updates.

5. Cloud Computing Layer (1m Grid)

The cloud layer performs heavy-lifting spatial statistics to generate the 1m precision grid required for SLV 2026 compliance.

5.1 Interpolation Methodology

We employ a hybrid **Regression Kriging** approach:

1. **Trend Component:** Derived from high-resolution satellite imagery (Sentinel-2 NDVI at 10m resolution) which provides the spatial structure.
2. **Residual Component:** Derived from ground sensors (soil moisture) using Ordinary Kriging to correct the satellite bias.
3. **Uncertainty Quantification:** Kriging variance provides confidence intervals for each 1m cell.
4. **Result:** A 1m resolution soil moisture map that respects both local measurements and field heterogeneity.

5.2 Cloud Processing Pipeline

```
                Cloud Processing Workflow

         [ Sensor Data + Satellite Imagery ]
                         ⬇
                 [ Data Fusion Layer ]
         Co-registration, Temporal Matching
                         ⬇
             [ Spatial Statistics Engine ]
     Regression Kriging, Bayesian Interpolation
                         ⬇
                  [ Grid Generator ]
      1m × 1m cell generation with uncertainty
                         ⬇
                 [ Quality Assurance ]
        Cross-validation, Outlier detection
                         ⬇
                [ Grid Storage + API ]
     PostGIS database + Vector Tile generation
```

5.3 Batch vs Real-time Processing

Processing Type	Use Case	Technology	Latency
Real-time Stream	Sensor data ingestion, critical alerts	Apache Flink / Kafka Streams	< 5 seconds
Micro-batch	Active mode recalculation (15-min window)	Spark Structured Streaming	5-15 minutes
Batch	Satellite imagery processing, historical analysis	Apache Airflow + Dask	Hours to days

5.4 Scalability Architecture

- **Horizontal Scaling:** Kubernetes auto-scaling based on queue depth and CPU metrics.
- **Geographic Partitioning:** Data partitioned by field ID for parallel processing.
- **Caching Layer:** Redis cluster for frequently accessed grid tiles (dashboard).
- **CDN Integration:** Pre-rendered Mapbox Vector Tiles cached at CloudFront edge locations.

6. Adaptive Recalculation Engine

To optimize compute costs and responsiveness, the system does not recalculate grids on a fixed schedule. Instead, it uses a state-machine based approach with event-driven triggers.

6.1 Recalculation Modes

Mode	Trigger Condition	Recalculation Window	Action
Stable	Variance < 5% over 6 hours	12 Hours	Background batch update
Active	Pump Status = ON or Rain > 2mm	15 Minutes	Standard grid update
Critical	Moisture < Wilting Point or Pump Failure	1 Minute	Priority queue processing + SMS Alerts
Out-of-Turn	Unexpected sensor swing (>20% change in 30 min)	Immediate	Recalculate affected field only

6.2 Judgment-Based Logic Implementation

The adaptive engine uses a combination of rules and trend analysis:

```
# Pseudocode for Mode Transition Logic
def calculate_next_interval(field_id):
    recent_variance = calculate_variance(field_id, window='6h')
    pump_status = get_pump_status(field_id)
    rainfall = get_rainfall(field_id, window='1h')
    moisture_current = get_moisture(field_id)

    if moisture_current < WILTING_POINT or pump_status == 'FAILED':
        return MODE_CRITICAL, interval=60  # 1 minute

    if pump_status == 'ON' or rainfall > 2.0:  # mm
        return MODE_ACTIVE, interval=900  # 15 minutes

    if recent_variance < 0.05:  # Stable field
        return MODE_STABLE, interval=43200  # 12 hours

    # Default: moderate monitoring
```

```
        return MODE_ACTIVE, interval=1800  # 30 minutes
```

6.3 Event-Driven Triggers

- **Sensor Anomaly:** Multiple consecutive out-of-range readings trigger immediate recalculation.
- **Pump State Change:** Transition from OFF to ON initiates Active mode for next 2 hours.
- **Extreme Weather:** Heavy rainfall (>10mm/hr) or high winds trigger Critical mode.
- **Satellite Update:** New Sentinel-2 imagery arrival triggers batch recalibration.

6.4 Configuration Management

All thresholds are configurable per field or crop type via admin dashboard:

- Wilting point moisture level (crop-specific)
- Variance threshold for Stable mode (soil texture-dependent)
- Rainfall trigger level (climate zone-adjusted)
- Alert notification channels (SMS, email, push)

7. Database Schema Design

7.1 Time-Series Schema (TimescaleDB)

```
CREATE TABLE sensor_readings (
    time        TIMESTAMPTZ NOT NULL,
    device_id   UUID NOT NULL,
    field_id    UUID NOT NULL,
    sensor_type VARCHAR(50), -- 'moisture_10cm', 'moisture_30cm', 'flow_rate
    value       DOUBLE PRECISION,
    quality_score FLOAT,     -- 0.0 to 1.0 confidence
    metadata    JSONB,       -- Additional sensor-specific data
    PRIMARY KEY (time, device_id, sensor_type)
);

-- Hypertable conversion for automatic time-based partitioning
SELECT create_hypertable('sensor_readings', 'time');
```

```
-- Create indexes for common queries
CREATE INDEX idx_sensor_field_time ON sensor_readings (field_id, time DESC);
CREATE INDEX idx_sensor_type_time ON sensor_readings (sensor_type, time DESC

-- Retention policy: keep raw data for 3 years, then aggregate
SELECT add_retention_policy('sensor_readings', INTERVAL '3 years');
```

7.2 Spatial Grid Schema (PostGIS)

```
CREATE EXTENSION IF NOT EXISTS postgis;

CREATE TABLE virtual_grid_1m (
    id           BIGSERIAL PRIMARY KEY,
    field_id     UUID REFERENCES fields(id),
    calc_time    TIMESTAMPTZ NOT NULL,
    geom         GEOMETRY(POLYGON, 4326),
    moisture_val DOUBLE PRECISION,
    uncertainty  DOUBLE PRECISION,  -- Kriging variance
    ndvi_val     DOUBLE PRECISION,  -- From satellite
    temperature  DOUBLE PRECISION,  -- Interpolated air temp
    compliance_status VARCHAR(20),  -- 'SAFE', 'WARNING', 'VIOLATION'
    metadata     JSONB
);

-- Spatial index for fast geographic queries
CREATE INDEX idx_grid_geom ON virtual_grid_1m USING GIST(geom);

-- Composite index for field + time queries
CREATE INDEX idx_grid_field_time ON virtual_grid_1m (field_id, calc_time DES
```

7.3 Compliance & Audit Schema (PostgreSQL)

```
CREATE TABLE compliance_logs (
    id           UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    field_id     UUID REFERENCES fields(id),
    log_time     TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    event_type   VARCHAR(50),  -- 'IRRIGATION_EVENT', 'VIOLATION', 'CALIBRATI
    details      JSONB NOT NULL,
    hash         VARCHAR(64),  -- SHA-256 hash for tamper detection
    previous_hash VARCHAR(64), -- Blockchain-style chaining
    user_id      UUID REFERENCES users(id),
    immutable   BOOLEAN DEFAULT true
);

-- Prevent updates/deletes on immutable logs
CREATE RULE no_update_compliance AS ON UPDATE TO compliance_logs
    WHERE OLD.immutable = true DO INSTEAD NOTHING;
CREATE RULE no_delete_compliance AS ON DELETE TO compliance_logs
    WHERE OLD.immutable = true DO INSTEAD NOTHING;
```

7.4 Field & Configuration Schema

```
CREATE TABLE fields (
    id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    farm_id     UUID REFERENCES farms(id),
    field_name  VARCHAR(100),
    geometry    GEOMETRY(MULTIPOLYGON, 4326),
    crop_type   VARCHAR(50),
    soil_type   VARCHAR(50),
    area_ha     DOUBLE PRECISION,
    wilting_point DOUBLE PRECISION,  -- Crop-specific threshold
    created_at  TIMESTAMPTZ DEFAULT NOW(),
    updated_at  TIMESTAMPTZ DEFAULT NOW()
);

CREATE TABLE sensor_devices (
    id          UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    field_id    UUID REFERENCES fields(id),
    device_type VARCHAR(50),  -- 'SOIL_MOISTURE', 'PUMP', 'WEATHER'
    location    GEOMETRY(POINT, 4326),
    install_date DATE,
    calibration_offset DOUBLE PRECISION,
    status      VARCHAR(20) DEFAULT 'ACTIVE'
);
```

7.5 Data Partitioning Strategy

- **Time-series partitioning:** Automatic monthly chunks via TimescaleDB hypertables.
- **Spatial partitioning:** Grid cells partitioned by field_id for parallel processing.
- **Hot/Warm/Cold tiers:**
    - Hot: Last 7 days in high-performance SSD storage (TimescaleDB)

- Warm: 7 days to 1 year in standard storage (PostgreSQL + TimescaleDB continuous aggregates)
- Cold: >1 year compressed in S3 with Parquet format for analytics

8. API Specifications

All APIs follow RESTful standards with OpenAPI 3.0 documentation. Authentication is handled via OAuth2/JWT.

8.1 Authentication & Authorization

```
# JWT Token Structure
{
  "sub": "user_uuid",
  "role": "farmer" | "consultant" | "regulator" | "admin",
  "farm_ids": ["uuid1", "uuid2"],
  "exp": 1735689600,
  "iat": 1735603200
}

# Authorization Levels
- Farmer: Read/Write access to own fields
- Consultant: Read access to multiple farms (multi-tenant)
- Regulator: Read-only access to compliance logs (all farms)
- Admin: Full system access
```

8.2 Core API Endpoints

Data Ingestion API

```
POST /api/v1/ingest/telemetry
Content-Type: application/json
Authorization: Bearer {edge_device_token}

Request Body:
{
  "device_id": "550e8400-e29b-41d4-a716-446655440000",
  "timestamp": "2023-10-15T14:30:00Z",
  "readings": [
    {
      "sensor_type": "moisture_10cm",
      "value": 0.28,
      "quality_score": 0.95
    },
    {
      "sensor_type": "moisture_30cm",
      "value": 0.32,
      "quality_score": 0.98
    }
  ]
}

Response: 202 Accepted
{
  "status": "queued",
  "message_id": "msg_abc123"
}
```

Virtual Grid Query API

```
GET /api/v1/fields/{field_id}/grid/latest
Query Parameters:
  - resolution: "1m" | "20m"
  - layer: "moisture" | "ndvi" | "temperature"
  - format: "geojson" | "mvt" (Mapbox Vector Tiles)
  - bbox: "xmin,ymin,xmax,ymax" (optional, for spatial filtering)

Response: 200 OK (GeoJSON example)
{
  "type": "FeatureCollection",
  "timestamp": "2023-10-15T14:30:00Z",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[lon, lat], [lon, lat], [lon, lat], [lon, lat], [lo
      },
      "properties": {
        "moisture": 0.28,
        "uncertainty": 0.02,
        "status": "SAFE"
      }
    }
    // ... more grid cells
  ]
}
```

Analytics & Forecasting API

```
GET /api/v1/fields/{field_id}/forecast/irrigation
Query Parameters:
  - horizon: 24 | 48 | 72 (hours)
  - confidence: 0.95 (default)

Response: 200 OK
{
  "field_id": "uuid",
  "forecast_time": "2023-10-15T14:30:00Z",
  "predictions": [
    {
      "timestamp": "2023-10-16T08:00:00Z",
      "irrigation_need": true,
      "recommended_mm": 12.5,
      "confidence": 0.92,
      "reasoning": "Predicted soil moisture depletion below 40% FC by mornin
    }
  ]
}
```

Compliance Reporting API

```
GET /api/v1/compliance/report/{field_id}
Query Parameters:
  - start_date: "2023-01-01"
  - end_date: "2023-10-15"
  - format: "pdf" | "json" | "csv"

Response: 200 OK
Content-Type: application/pdf (or JSON/CSV)

# JSON Response Structure:
{
  "field_id": "uuid",
  "report_period": "2023-01-01 to 2023-10-15",
  "generated_at": "2023-10-15T14:35:00Z",
  "total_irrigation_events": 45,
  "total_water_applied_m3": 1250.5,
  "compliance_violations": 0,
  "events": [
    {
      "timestamp": "2023-03-15T06:00:00Z",
      "event_type": "IRRIGATION_START",
      "water_applied_mm": 25.0,
      "duration_minutes": 120,
      "compliance_status": "COMPLIANT",
      "audit_hash": "a7b8c9d0..."
    }
  ]
}
```

8.3 Rate Limiting & Throttling

Endpoint Category	Rate Limit	Burst Allowance
Data Ingestion (Edge Devices)	1000 req/min per device	1500 req/min (burst)
Dashboard Queries (Authenticated)	100 req/min per user	200 req/min (burst)
Compliance Reports (Regulator)	10 req/min per user	20 req/min (burst)

8.4 API Versioning Strategy

- **URL Versioning:** /api/v1/, /api/v2/ for major breaking changes.
- **Backward Compatibility:** v1 maintained for minimum 12 months after v2 release.
- **Deprecation Headers:** Sunset header indicates API version retirement date.

9. Analytics & ML Layer

The analytics engine runs in a containerized environment (Kubernetes CronJobs) accessing the Data Lake.

9.1 Predictive Irrigation Scheduling

- **Model Architecture:** LSTM (Long Short-Term Memory) networks with attention mechanism
- **Input Features:**
  - Historical soil moisture (7-day window)
  - Weather forecast (temperature, rainfall, evapotranspiration)

    - Crop stage and water demand coefficients
    - Field characteristics (soil type, slope, drainage)
- **Output:** 48-hour ahead irrigation recommendations with confidence intervals
- **Training Data:** Multi-year historical records from Landsat-derived indices and weather logs
- **Model Update Frequency:** Weekly retraining during growing season

9.2 Crop Stress Detection

- **Algorithm:** Ensemble of Random Forest and Gradient Boosting classifiers
- **Input Features:**
    - NDVI, NDRE, NDWI from Sentinel-2
    - Soil moisture deficit from ground sensors
    - Temperature stress indices
    - Historical stress patterns
- **Stress Categories:** None, Mild, Moderate, Severe (4-class classification)
- **Early Detection Target:** Identify stress 3-5 days before visible symptoms

9.3 Yield Forecasting

- **Model Type:** Multi-modal deep learning (CNN for imagery + LSTM for time-series)
- **Data Sources:**
    - Satellite time-series (vegetation indices progression)
    - Weather data (growing degree days, water stress integral)
    - Irrigation history and soil moisture patterns
    - Historical yield data for calibration
- **Forecast Horizon:** 30 days before harvest with weekly updates
- **Accuracy Target:** ±10% of actual yield at 30 days pre-harvest

9.4 Anomaly Detection for Pump Telemetry

- **Algorithm:** Isolation Forest for unsupervised anomaly detection

- **Monitored Parameters:** Flow rate, pressure, power consumption, vibration (if available)
- **Alert Triggers:** Cavitation patterns, bearing failure signatures, efficiency degradation
- **Predictive Maintenance:** Estimate time-to-failure based on degradation trends

9.5 Model Training & Deployment Pipeline

```
             ML Pipeline Architecture

                [ Data Lake (S3) ]
          Historical sensor + satellite data
                        ⬇
             [ Feature Engineering ]
                Apache Spark / Dask
                        ⬇
                [ Model Training ]
             Kubeflow / MLflow on K8s
                        ⬇
               [ Model Validation ]
           Cross-validation, A/B testing
                        ⬇
                [ Model Registry ]
               MLflow Model Registry
                        ⬇
              [ Inference Service ]
         TensorFlow Serving / Seldon Core
                        ⬇
                 [ API Gateway ]
         REST/gRPC endpoints for predictions
```

9.6 Feature Engineering Pipeline

- **Temporal Features:** Moving averages, rate of change, seasonality decomposition
- **Spatial Features:** Distance to nearest sensor, terrain slope/aspect, soil variability indices
- **Derived Indices:** Moisture deficit integral, cumulative growing degree days, water stress days

- **Satellite Indices:** NDVI, NDRE, NDWI, LAI (Leaf Area Index), fAPAR

10. Interface Layer Architecture

10.1 Farmer Dashboard

- **Technology Stack:** React 18 + TypeScript, Mapbox GL JS for mapping, Recharts for analytics
- **Key Features:**
  - Real-time field heatmap visualization (1m grid overlay)
  - "Traffic Light" status indicators (Green=Safe, Yellow=Warning, Red=Critical)
  - Irrigation recommendations with explanations
  - Pump status monitoring and control
  - Historical trend charts and comparison views
  - Mobile-responsive design for field access
- **Performance Optimization:**
  - Mapbox Vector Tiles for fast rendering of large grid datasets
  - Progressive Web App (PWA) with offline capability
  - WebSocket connection for real-time updates
  - Lazy loading and code splitting for fast initial load

10.2 Regulatory Compliance Portal

- **Technology Stack:** React + TypeScript, Tailwind CSS, PDF generation via jsPDF
- **Key Features:**
  - Read-only view with tamper-proof audit logs
  - Historical compliance report generation (PDF, CSV export)
  - Field-level and regional aggregation views
  - Violation tracking and trend analysis
  - Immutable log verification (hash chain validation)
  - Role-based access control for different regulator levels

- **Compliance Indicators:** SLV 2026 alignment metrics, water usage vs allocation, violation alerts

10.3 Consultant Multi-Field View

- **Technology Stack:** React + TypeScript, D3.js for advanced visualizations
- **Key Features:**
  - Multi-farm dashboard with side-by-side field comparison
  - Regional stress maps aggregating multiple fields
  - Benchmarking tools (compare field performance)
  - Recommendation engine for agronomic interventions
  - Export capabilities for client reports
- **Multi-Tenancy:** Secure isolation of data across different client farms

10.4 Mobile Application Architecture

- **Technology:** React Native for iOS and Android
- **Core Capabilities:**
  - Simplified field view optimized for mobile screens
  - Push notifications for critical alerts
  - Offline mode with local data caching
  - GPS-based field navigation
  - Manual sensor reading entry (backup for connectivity issues)

10.5 Real-Time Alert System

- **Notification Channels:** SMS (Twilio), Email (SendGrid), Push (Firebase Cloud Messaging), In-app
- **Alert Types:**
  - Critical: Moisture below wilting point, pump failure (immediate)
  - Warning: Approaching stress threshold, unusual readings (within 30 min)
  - Info: Irrigation recommendations, forecast updates (scheduled)
- **Smart Throttling:** Prevent alert fatigue with intelligent grouping and escalation rules

10.6 UI/UX Design Principles

- **Accessibility:** WCAG 2.1 AA compliance, screen reader support, high contrast modes
- **Internationalization:** Multi-language support (English, Spanish, Chinese initially)
- **Color Scheme:** Agriculture-themed green/brown palette with colorblind-friendly indicators
- **Performance Target:** < 2 second load time, 60fps map interactions

11. Technology Stack Recommendations

Component	Technology	Justification
Backend Language	Python 3.11+ (FastAPI)	Native support for geospatial libraries (GDAL, Rasterio, GeoPandas) and ML frameworks (PyTorch, TensorFlow). FastAPI provides high-performance async support, automatic OpenAPI documentation, and excellent type validation.
Edge Module	Go (Golang) 1.21+	Compiles to single binary (~10MB), low memory footprint (<50MB runtime), excellent concurrency with goroutines for handling sensor streams on limited hardware (Raspberry Pi). Fast startup time for edge resilience.
Time-Series DB	TimescaleDB 2.11+	Built on PostgreSQL 15+, allowing seamless joining of time-series data with relational/spatial (PostGIS) business data. Automatic time-based partitioning, compression (90% storage savings), and continuous aggregates for historical analysis. Open-source with commercial support option.
Spatial Database	PostGIS 3.3+	Industry-standard for geospatial data. Supports complex spatial queries, indexing (R-tree, GIST), and seamless integration with TimescaleDB. Proven scalability for millions of polygons.
Message Queue	Apache Kafka 3.5+ / RabbitMQ 3.12	**Kafka:** High-throughput event streaming (1M+ msg/sec), durable log storage, ideal for sensor ingestion pipeline and audit trail. **RabbitMQ:** Complex routing for alert triggers, guaranteed delivery for critical notifications. Recommendation: Use both - Kafka for data pipeline, RabbitMQ for alerts.

Component	Technology	Justification
Stream Processing	Apache Flink 1.17+	True streaming (not micro-batch), exactly-once processing semantics, low latency (<100ms), stateful computations for real-time grid updates. Better for Critical mode processing than Spark Streaming.
Batch Processing	Apache Airflow 2.7+ + Dask	**Airflow:** Workflow orchestration for satellite data ingestion, ML training, report generation. Rich monitoring and retry logic. **Dask:** Scalable Python-native parallel computing for geospatial raster processing. Better integration with GDAL/Rasterio than Spark.
Satellite Data Processing	Google Earth Engine (API) + Rasterio + GDAL	**GEE:** Massive satellite preprocessing in cloud (Petabyte-scale archive), free tier available, pre-computed indices. **Rasterio/GDAL:** Local/cloud custom interpolation logic, full control over processing pipeline. SNAP toolbox for advanced SAR processing.
ML Frameworks	PyTorch 2.0+ / TensorFlow 2.13+ / scikit-learn	**PyTorch:** LSTM models for time-series forecasting, flexible research-to-production workflow. **TensorFlow:** Production-ready serving infrastructure (TF Serving), mobile deployment (TFLite). **scikit-learn:** Classical ML algorithms (Random Forest, Isolation Forest), spatial statistics (Kriging via scikit-gstat).
Model Serving	Seldon Core / TensorFlow Serving	Kubernetes-native model deployment, A/B testing, canary rollouts, metrics collection. Seldon supports multi-framework (PyTorch, TF, XGBoost) with unified API.
Frontend Framework	React 18+ with TypeScript	Large ecosystem, excellent performance, strong typing with TS reduces bugs, extensive mapping library support (Mapbox, Leaflet). Mature server-side rendering (Next.js) for SEO and performance.
Mapping Library	Mapbox GL JS 3.0+	Best-in-class performance for rendering large vector tile sets (the 1m grid). WebGL-accelerated, smooth interactions with millions of features, excellent mobile support. Style customization via Mapbox Studio.
API Gateway	Kong Gateway / AWS API Gateway	Authentication, rate limiting, request transformation, analytics. Kong: open-source, self-hosted option.

Component	Technology	Justification
		AWS API Gateway: fully managed, tight AWS integration.
Container Orchestration	Kubernetes 1.28+ (Amazon EKS / Azure AKS / Google GKE)	Industry standard, cloud-agnostic (portability across AWS/Azure/GCP), auto-scaling, self-healing, rich ecosystem (Helm charts, operators). Managed services reduce operational burden.
Infrastructure as Code	Terraform 1.5+ / Pulumi	Multi-cloud support, declarative configuration, version control for infrastructure. Terraform: mature, large community. Pulumi: allows real programming languages (Python/TypeScript) instead of HCL.
Monitoring & Observability	Prometheus + Grafana + ELK Stack	**Prometheus:** Metrics collection, time-series storage, alerting (Alertmanager). **Grafana:** Visualization dashboards, multi-datasource support. **ELK (Elasticsearch, Logstash, Kibana):** Centralized logging, full-text search, log analytics.
Distributed Tracing	Jaeger / OpenTelemetry	Track requests across microservices, identify performance bottlenecks, debug distributed systems. OpenTelemetry provides vendor-neutral instrumentation.
Cache Layer	Redis 7.0+ (Cluster mode)	In-memory caching for hot data (dashboard queries, frequently accessed grids), pub/sub for real-time updates, geospatial queries support. Cluster mode for high availability.
Object Storage	AWS S3 / Azure Blob / Google Cloud Storage	Durable storage for satellite imagery, historical data archives, compliance logs. Tiered storage (Standard, Infrequent Access, Glacier) for cost optimization. WORM (Write-Once-Read-Many) support for audit logs.
CDN	CloudFront / Cloudflare	Cache static assets (map tiles, dashboard resources) at edge locations, reduce latency, lower origin load. DDoS protection and WAF included.
CI/CD	GitHub Actions / GitLab CI	Integrated with version control, matrix testing (multiple Python versions), artifact management, secrets management. GitHub Actions: free for public repos. GitLab: self-hosted option.
Secret Management	HashiCorp Vault / AWS Secrets	Centralized secret storage (API keys, DB credentials), dynamic secrets generation, audit logging.

Component	Technology	Justification
	Manager	Kubernetes integration for automatic secret injection.

Technology Decision Trade-offs

> **Open Source vs Managed Services:**
>
> - **Self-hosted (Open Source):** Lower long-term costs, full control, customization. Requires DevOps expertise.
> - **Managed Services (AWS/Azure/GCP):** Faster deployment, reduced operational burden, auto-scaling. Higher costs at scale.
> - **Recommendation:** Hybrid approach - managed services for foundational infrastructure (K8s, databases), self-hosted for specialized components (ML training, geospatial processing).

12. Deployment Architecture

The infrastructure is defined as code (Terraform) to ensure national replicability and consistent deployments.

12.1 Cloud Provider Selection

Provider	Strengths	Use Case Fit
AWS	Largest service catalog, mature geospatial services (Ground Station, Location Service), strong IoT suite	Best for comprehensive full-stack deployment
Azure	Excellent enterprise integration, strong AI/ML services, FarmBeats platform	Good for enterprise customers with existing Microsoft stack
GCP	Best data analytics (BigQuery), tight Earth Engine integration, cost-effective compute	Ideal for satellite data processing workloads

Recommendation: AWS as primary provider with multi-cloud capability via Terraform abstraction.

12.2 AWS Reference Architecture

```
AWS Cloud Infrastructure

[ Route 53 ] DNS + Health Checks
⬇
[ CloudFront CDN ] Static assets, map tiles
⬇
[ Application Load Balancer ]
⬇
[ Amazon EKS Cluster ]
- FastAPI Services (Fargate/EC2 nodes)
- Flink Stream Processing
- ML Inference (Seldon Core)
⬇
[ Data Layer ]
- RDS PostgreSQL + TimescaleDB (Multi-AZ)
- Amazon MSK (Kafka) for event streaming
- Amazon ElastiCache (Redis) for caching
- Amazon S3 (raw data, satellite imagery)
⬇
[ Analytics Layer ]
- Amazon SageMaker (ML training)
- AWS Glue (ETL for data lake)
- Amazon Athena (ad-hoc queries on S3)
⬇
[ Edge Connectivity ]
- AWS IoT Core (MQTT broker)
- AWS Greengrass (edge software management)
```

12.3 Kubernetes Cluster Configuration

- **Node Groups:**
  - **General Purpose (t3.large):** API services, lightweight processing
  - **Compute Optimized (c6i.xlarge):** Stream processing, grid calculations
  - **Memory Optimized (r6i.xlarge):** In-memory caching, large dataset processing
  - **GPU Nodes (g4dn.xlarge):** ML inference, satellite image processing
- **Auto-scaling:** KEDA (Kubernetes Event-driven Autoscaling) based on:
  - Kafka consumer lag for ingestion services

    - Queue depth for grid recalculation workers
    - HTTP request rate for API services
- **High Availability:** Multi-AZ deployment, minimum 3 replicas for critical services

12.4 Database Deployment Strategy

- **Primary Database:** Amazon RDS PostgreSQL 15 with TimescaleDB extension
    - Multi-AZ deployment for 99.95% availability
    - Automated backups (point-in-time recovery, 35-day retention)
    - Read replicas for analytics queries (separate from operational traffic)
    - Instance size: db.r6g.xlarge initially (4 vCPU, 32GB RAM)
- **Cache Layer:** Amazon ElastiCache Redis Cluster
    - Multi-AZ with automatic failover
    - 6 shards × 2 replicas for 99.99% availability
    - Node type: cache.r6g.large (2 vCPU, 13GB RAM per node)

12.5 Edge Deployment Architecture

- **Device Management:** AWS IoT Greengrass for over-the-air updates
- **Deployment Package:**
    - Docker Compose stack (Go edge module + SQLite + local MQTT broker)
    - Automatic updates via Greengrass deployment groups
    - Rollback capability in case of failed updates
- **Network Configuration:**
    - Primary: 4G LTE with static IP (via cellular router)
    - Fallback: Ethernet connection (farm network)
    - VPN tunnel (WireGuard) for secure cloud communication

12.6 Scalability & Performance Targets

Metric	Target	Scaling Strategy
Sensor Ingestion	100,000 sensors × 4 readings/hour = 400K msgs/hr	Kafka partitioning by field_id, Flink parallel processing

Metric	Target	Scaling Strategy
Grid Recalculation	1000 fields × 1m grid (avg 10 hectares) in 15 min	Horizontal pod autoscaling, GPU acceleration for large fields
Dashboard Queries	10,000 concurrent users, < 2s response time	Redis caching, CDN for map tiles, load balancing
API Throughput	50,000 req/min across all endpoints	Auto-scaling API pods, rate limiting per user tier

12.7 Disaster Recovery & Backup

- **Recovery Time Objective (RTO):** < 1 hour for full system recovery
- **Recovery Point Objective (RPO):** < 5 minutes (minimal data loss)
- **Backup Strategy:**
  - Database: Automated daily snapshots + continuous transaction log backup
  - Object Storage: S3 versioning + cross-region replication
  - Configuration: Infrastructure code in Git, immutable deployments
- **Disaster Recovery Test:** Quarterly full-stack recovery drill

12.8 Cost Optimization Strategies

- **Compute:** Reserved Instances (1-year) for baseline load, Spot Instances for batch processing (50-70% savings)
- **Storage:** S3 Intelligent-Tiering for automatic lifecycle management, Glacier for >3-year archives
- **Data Transfer:** VPC endpoints to avoid NAT gateway charges, CloudFront caching to reduce origin traffic
- **Monitoring:** Right-sizing recommendations via AWS Compute Optimizer, unused resource alerts

12.9 Monitoring & Observability Stack

- **Metrics:** Prometheus (in-cluster) + Amazon Managed Prometheus (long-term storage)
- **Logging:** FluentBit (log collection) → Amazon OpenSearch Service (ELK alternative)

- **Tracing:** Jaeger (in-cluster) + AWS X-Ray (managed service)
- **Dashboards:** Grafana for technical metrics, custom admin dashboard for business metrics
- **Alerting:** Prometheus Alertmanager → PagerDuty (on-call) / Slack (team notifications)

13. Security & Compliance Framework

> **SLV 2026 Regulatory Alignment:** All compliance logs are hashed and stored in a "WORM" (Write Once, Read Many) compatible S3 bucket configuration with Object Lock to prevent tampering. This ensures non-repudiable audit trails.

13.1 Data Encryption

Data State	Encryption Method	Key Management
In-Transit	TLS 1.3 (all API communications), mTLS (service-to-service in K8s)	Let's Encrypt (public endpoints), AWS Certificate Manager (internal)
At-Rest (Databases)	AES-256 encryption for RDS volumes	AWS KMS (Customer Managed Keys with automatic rotation)
At-Rest (Object Storage)	S3 SSE-KMS (Server-Side Encryption)	AWS KMS with separate keys per data classification
Backups	AES-256 encrypted snapshots	Separate KMS key for backup data

13.2 Identity & Access Management

- **Authentication:**
  - OAuth 2.0 / OpenID Connect for user authentication
  - Multi-Factor Authentication (MFA) required for admin and regulator roles
  - API key authentication for edge devices with automatic rotation
  - SSO integration (SAML 2.0) for enterprise customers
- **Authorization (RBAC - Role-Based Access Control):**
  - **Farmer:** Read/Write on own fields, read-only on shared consultant reports

    - **Consultant:** Read access to assigned client farms, write access for recommendations
    - **Regulator:** Read-only access to compliance logs and reports across all farms
    - **Admin:** Full system access, user management, configuration
    - **Edge Device:** Write-only to ingestion endpoints, read configuration
- **Principle of Least Privilege:** Fine-grained permissions, time-limited access tokens (1-hour JWT expiry)

13.3 Audit Trail & Compliance Logging

- **Immutable Audit Log:**
    - Every grid recalculation logged with input parameters, algorithm version, output hash
    - All configuration changes logged with user ID, timestamp, old/new values
    - Irrigation events logged with precise timestamps, volumes, field conditions
    - Blockchain-style hash chaining: each log entry references previous entry's hash
- **Compliance Report Generation:**
    - Automated daily/weekly/monthly compliance summary reports
    - Violation detection with automatic regulator notification
    - Historical trend analysis for water usage patterns
    - PDF/CSV export with digital signatures for legal validity
- **Retention Policy:**
    - Compliance logs: 10 years (regulatory requirement)
    - Operational logs: 1 year (system troubleshooting)
    - Raw sensor data: 3 years (trend analysis)

13.4 SLV 2026 Specific Requirements

```
# SLV 2026 Compliance Checklist
✓ Real-time water usage monitoring (15-min resolution)
✓ Field-level irrigation event logging with precise timestamps
```

```
  ✓ Soil moisture tracking at regulatory depths (10cm, 30cm minimum)
  ✓ Annual water allocation vs actual usage reporting
  ✓ Violation alerting for over-irrigation or off-schedule watering
  ✓ Immutable audit trail with tamper-proof logging
  ✓ Regulator portal access for inspection and compliance verification
  ✓ Historical data retention for 10+ years
  ✓ Data export in standardized formats (CSV, PDF, GeoJSON)
```

13.5 Data Privacy & GDPR Compliance

- **Personal Data Handling:**
  - Minimal PII collection (only necessary for authentication and contact)
  - Encrypted storage of user credentials (bcrypt with salt)
  - Right to erasure: automated anonymization of user data upon request
  - Data portability: export user data in machine-readable format
- **Consent Management:**
  - Explicit opt-in for data sharing with consultants/third parties
  - Granular consent controls (separate for analytics, marketing, research)
  - Audit log of all consent changes
- **Geographic Data Restrictions:**
  - EU data residency requirements (deploy in eu-west region for EU customers)
  - Data transfer agreements for cross-border compliance data sharing

13.6 Network Security

- **Perimeter Security:**
  - Web Application Firewall (WAF) protecting API endpoints
  - DDoS protection via AWS Shield Standard (free) / Advanced (for critical deployments)
  - Rate limiting and IP allowlisting for edge device ingestion
- **Internal Network Segmentation:**
  - Kubernetes Network Policies isolating namespaces (development, staging, production)

    - Private subnets for databases and internal services (no direct internet access)
    - Bastion hosts (jump servers) for administrative access with MFA
- **Edge Security:**
    - VPN tunnels (WireGuard) for edge-to-cloud communication
    - Device authentication via mutual TLS certificates
    - Firmware signature verification before updates

13.7 Vulnerability Management

- **Dependency Scanning:** Automated scanning of container images (Trivy, Snyk) in CI/CD pipeline
- **Penetration Testing:** Annual third-party security audit and penetration test
- **Patch Management:**
    - Critical security patches applied within 24 hours
    - Regular patches applied within 7 days
    - Automated OS patching for edge devices via Greengrass
- **Incident Response Plan:**
    - Security incident detection via SIEM (Security Information and Event Management)
    - Defined escalation procedures and communication templates
    - Post-incident review and remediation tracking

13.8 Compliance Certifications (Roadmap)

- **ISO 27001:** Information Security Management System (Target: Year 2 post-launch)
- **SOC 2 Type II:** Service Organization Control audit (Target: Year 1 post-launch)
- **GDPR Compliance:** Mandatory for EU operations (Day 1 requirement)
- **Regional Agriculture Standards:** Country-specific compliance (ongoing)

14. Development Roadmap

Phase 1: Foundation & Infrastructure (Weeks 1-4)

Objective: Establish core infrastructure and basic data ingestion pipeline.

- **Infrastructure Setup:**
  - Terraform scripts for AWS EKS cluster, RDS (PostgreSQL + TimescaleDB), S3 buckets
  - Kubernetes namespaces and basic networking (load balancers, ingress)
  - CI/CD pipeline setup (GitHub Actions with EKS deployment)
- **Data Ingestion:**
  - MQTT broker setup (AWS IoT Core or self-hosted Mosquitto)
  - Basic FastAPI ingestion endpoint (sensor_readings table)
  - Kafka cluster deployment and basic producer/consumer
- **Database Schema:**
  - Implement time-series schema (sensor_readings hypertable)
  - Field and device management tables
  - Initial PostGIS setup for spatial queries
- **Validation:**
  - Test sensor data ingestion from 5-10 simulated devices
  - Verify data persistence and basic querying
  - Load testing (10K messages/min ingestion)

Deliverables: Functional data ingestion pipeline, deployed infrastructure, basic monitoring.

Phase 2: Core Processing (Weeks 5-8)

Objective: Implement edge and cloud compute layers for virtual grid generation.

- **Edge Module (Go):**
  - 20m grid IDW interpolation algorithm
  - Local SQLite buffer with automatic compression
  - Store-and-forward synchronization logic
  - Docker Compose packaging and Greengrass deployment
- **Cloud Processing:**

    - Sentinel-2 imagery download and preprocessing pipeline (Airflow DAG)
    - Regression Kriging implementation (Python with scikit-gstat)
    - 1m grid generation and PostGIS storage
    - Mapbox Vector Tile generation for frontend
- **Adaptive Recalculation Engine:**
    - State machine implementation (Stable/Active/Critical modes)
    - Event-driven trigger system (Kafka consumers)
    - Configuration management (field-specific thresholds)
- **Validation:**
    - Test 20m → 1m grid synchronization for 5 fields
    - Verify mode transitions based on simulated events
    - Performance testing (1000 fields × 10 hectares each)

Deliverables: Functional edge module, cloud interpolation engine, adaptive recalculation system.

Phase 3: Analytics & ML Integration (Weeks 9-12)

Objective: Deploy predictive models and analytics capabilities.

- **Irrigation Prediction Model:**
    - LSTM model training on historical Landsat data (2013-2023)
    - Weather API integration (NOAA, OpenWeather)
    - Model deployment on Seldon Core with A/B testing
    - 48-hour forecast API endpoint
- **Crop Stress Detection:**
    - Random Forest classifier training (NDVI, moisture, temperature)
    - Real-time inference on new satellite imagery
    - Stress map generation and alert system
- **Yield Forecasting:**
    - Multi-modal CNN+LSTM model training
    - 30-day pre-harvest forecast endpoint

    - Field-level and regional aggregation views
- **Landsat Historical Calibration:**
    - 10-year Landsat data download and preprocessing
    - Baseline soil variability mapping
    - Sensor calibration adjustment recommendations
- **Validation:**
    - Backtesting models on 2022-2023 data (80/20 train/test split)
    - Real-world validation with 10 pilot farms
    - Accuracy metrics: RMSE, MAE, R^2 for continuous predictions; F1-score for classification

Deliverables: Deployed ML models, forecast APIs, historical calibration system.

Phase 4: Interface & Compliance (Weeks 13-16)

Objective: Build user-facing dashboards and compliance reporting system.

- **Farmer Dashboard:**
    - React application with Mapbox GL JS integration
    - Real-time field heatmap (1m grid overlay)
    - Traffic light status indicators and alerts
    - Historical trend charts and irrigation recommendations
    - Mobile-responsive PWA with offline capability
- **Regulatory Compliance Portal:**
    - Read-only audit log viewer with hash verification
    - Compliance report generation (PDF/CSV export)
    - Violation tracking and trend analysis
    - Multi-field and regional aggregation views
- **Consultant Multi-Field View:**
    - Side-by-side field comparison tools
    - Regional stress maps and benchmarking
    - Recommendation engine for agronomic interventions

    - Client report export functionality
- **SLV 2026 Compliance Implementation:**
    - Immutable audit log with blockchain-style hash chaining
    - WORM S3 bucket configuration for compliance logs
    - Automated daily/weekly/monthly compliance summaries
    - Digital signature for legal validity of reports
- **Alert System:**
    - Multi-channel notification (SMS, Email, Push, In-app)
    - Smart throttling to prevent alert fatigue
    - Configurable alert rules per user/field
- **Validation:**
    - User acceptance testing with 20 farmers, 5 consultants, 3 regulators
    - Load testing (10,000 concurrent dashboard users)
    - Accessibility audit (WCAG 2.1 AA compliance)

Deliverables: Production-ready dashboards, compliance reporting system, alert infrastructure.

Phase 5: Optimization & National Rollout (Weeks 17-20)

Objective: Performance optimization, documentation, and prepare for scale.

- **Performance Optimization:**
    - Database query optimization (index tuning, query plan analysis)
    - Caching strategy refinement (Redis cache hit rate > 80%)
    - CDN configuration for global edge distribution
    - Kubernetes resource tuning and cost optimization
- **Documentation:**
    - API documentation (OpenAPI/Swagger)
    - Deployment runbooks and operational procedures
    - User guides for farmers, consultants, regulators
    - Training materials and video tutorials

- **Security Hardening:**
  - Penetration testing and vulnerability remediation
  - Security audit and compliance certification prep
  - Incident response plan and disaster recovery testing
- **Multi-Region Deployment:**
  - Terraform modules for regional deployment (US, EU, Asia-Pacific)
  - Data residency compliance for different jurisdictions
  - Localization (language support, regulatory adaptation)
- **Pilot Program:**
  - 50-field pilot across diverse climates and crop types
  - Gather feedback and iterate on user experience
  - Real-world performance validation and tuning

Deliverables: Production-hardened system, comprehensive documentation, multi-region deployment capability.

Post-Launch Roadmap (Months 6-12)

- **Month 6:** Mobile app launch (iOS/Android), advanced analytics dashboard
- **Month 9:** Integration with precision agriculture equipment (variable rate irrigation, automated valves)
- **Month 12:** Expansion to additional satellite data sources (PlanetScope for daily imagery), drone integration

Resource Requirements

Role	Headcount	Key Responsibilities
Backend Engineers	3	FastAPI services, data ingestion, grid computation
Data Engineer	1	ETL pipelines, satellite data processing, data lake
ML Engineer	1	Model training, deployment, feature engineering
Frontend Engineers	2	React dashboards, mobile app, UI/UX
DevOps Engineer	1	Infrastructure, CI/CD, monitoring, security

Role	Headcount	Key Responsibilities
Edge/IoT Engineer	1	Edge module development, device management
Product Manager	1	Requirements, roadmap, stakeholder coordination
QA Engineer	1	Testing strategy, automation, quality assurance

Total Team Size: 11 full-time employees

15. Implementation Guidelines

15.1 Development Environment Setup

Prerequisites

- Docker 24+ and Docker Compose 2.20+
- Kubernetes (Minikube or Kind for local development)
- Python 3.11+, Go 1.21+, Node.js 20+
- Terraform 1.5+, kubectl, helm 3+
- PostgreSQL client (psql), Redis CLI

Local Development Stack

```
# docker-compose.yml for local development
version: '3.8'
services:
  postgres:
    image: timescale/timescaledb:latest-pg15
    environment:
      POSTGRES_DB: farmsense_dev
      POSTGRES_USER: dev
      POSTGRES_PASSWORD: dev_password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
  kafka:
    image: confluentinc/cp-kafka:7.5.0
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
    ports:
      - "9092:9092"

  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

volumes:
  postgres_data:
```

15.2 Repository Structure (Monorepo)

```
farmsense/
├── infrastructure/             # Terraform and Kubernetes configs
│   ├── terraform/
│   │   ├── modules/           # Reusable TF modules
│   │   ├── environments/      # Dev, staging, prod configs
│   │   └── main.tf
│   └── kubernetes/
│       ├── base/              # Base K8s manifests
│       ├── overlays/          # Environment-specific overlays
│       └── helm-charts/
│
├── services/                   # Backend microservices
│   ├── ingestion/             # Data ingestion API (FastAPI)
│   │   ├── src/
│   │   ├── tests/
│   │   ├── Dockerfile
│   │   └── requirements.txt
│   ├── compute/               # Grid computation service (Python)
│   ├── analytics/             # ML inference service
│   └── api-gateway/           # Main API gateway
│
├── edge/                       # Edge computing module (Go)
│   ├── cmd/
│   ├── internal/
│   ├── pkg/
│   ├── Dockerfile
│   └── go.mod
│
├── frontend/                   # React applications
```

```
│   ├── farmer-dashboard/
│   ├── regulator-portal/
│   ├── consultant-view/
│   └── shared-components/
│
├── ml/                         # ML models and training pipelines
│   ├── irrigation-prediction/
│   ├── stress-detection/
│   ├── yield-forecast/
│   └── notebooks/             # Jupyter notebooks for experimentation
│
├── data-pipelines/             # Airflow DAGs and ETL scripts
│   ├── dags/
│   └── scripts/
│
├── lib/                        # Shared libraries
│   ├── python/
│   │   ├── farmsense_common/   # Common Python utilities
│   │   └── spatial_utils/      # Geospatial helper functions
│   └── go/
│       └── common/            # Shared Go packages
│
├── docs/                       # Documentation
│   ├── architecture/
│   ├── api/
│   └── guides/
│
├── scripts/                    # Utility scripts
│   ├── setup-dev.sh
│   ├── run-tests.sh
│   └── deploy.sh
│
├── .github/
│   └── workflows/             # CI/CD pipelines
│       ├── backend-ci.yml
│       ├── frontend-ci.yml
│       └── deploy-prod.yml
│
└── README.md
```

15.3 Testing Strategy

Unit Testing

- **Python (Backend/ML):** PyTest with 80%+ coverage target

```
# Example: pytest for interpolation algorithm
def test_regression_kriging():
    sensor_data = create_mock_sensor_data()
    satellite_data = create_mock_satellite_raster()

    result = regression_kriging(sensor_data, satellite_data, grid_resolu

    assert result.shape == (100, 100)  # Expected grid size
    assert 0 <= result.all() <= 1      # Moisture values in valid range
    assert result.uncertainty.mean() < 0.1  # Reasonable uncertainty
```

- **Go (Edge Module):** Standard testing package with table-driven tests

```go
// Example: Go test for IDW interpolation
func TestIDWInterpolation(t *testing.T) {
    tests := []struct {
        name     string
        sensors  []SensorReading
        expected float64
    }{
        {"single sensor", []SensorReading{{Lat: 0, Lon: 0, Value: 0.5}},
        {"multiple sensors", []SensorReading{{Lat: 0, Lon: 0, Value: 0.3
    }

    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            result := IDWInterpolate(tt.sensors, Point{Lat: 0.5, Lon: 0.
            if math.Abs(result-tt.expected) > 0.01 {
                t.Errorf("Expected %f, got %f", tt.expected, result)
            }
        })
    }
}
```

- **JavaScript (Frontend):** Jest + React Testing Library

```javascript
// Example: React component test
test('renders field status with correct color', () => {
  const field = { id: '1', status: 'CRITICAL', moisture: 0.15 };
  render();

  const statusElement = screen.getByTestId('field-status');
  expect(statusElement).toHaveClass('status-critical');
  expect(statusElement).toHaveTextContent('CRITICAL');
});
```

Integration Testing

- **API Integration Tests:** Test database with pytest-postgresql

```
@pytest.fixture
def test_db():
    # Setup test database with schema
    db = setup_test_database()
    yield db
    teardown_test_database(db)

def test_ingestion_endpoint(test_db, test_client):
    response = test_client.post('/api/v1/ingest/telemetry', json={
        'device_id': 'test-device-1',
        'timestamp': '2023-10-15T14:30:00Z',
        'readings': [{'sensor_type': 'moisture_10cm', 'value': 0.28}]
    })

    assert response.status_code == 202

    # Verify data persisted in database
    reading = test_db.query(SensorReading).first()
    assert reading.value == 0.28
```

- **Kafka Integration:** Testcontainers for integration tests with message queues
- **Database Migration Tests:** Validate schema migrations don't break existing data

End-to-End Testing

- **Frontend E2E:** Cypress for critical user flows

```
describe('Farmer Dashboard', () => {
  it('should display field heatmap and allow zoom', () => {
    cy.visit('/dashboard');
    cy.login('farmer@test.com', 'password');

    cy.get('[data-testid="field-map"]').should('be.visible');
    cy.get('[data-testid="field-1"]').click();
    cy.get('[data-testid="field-details"]').should('contain', 'Field 1')

    // Verify map interaction
    cy.get('[data-testid="zoom-in"]').click();
    cy.wait(500);
    cy.get('[data-testid="map-zoom-level"]').should('have.text', '14');
  });
});
```

- **API E2E:** Postman/Newman for full API workflow testing

Performance Testing

- **Load Testing:** Apache JMeter or k6 for API endpoints

```
// k6 load test script
import http from 'k6/http';
import { check, sleep } from 'k6';

export let options = {
  stages: [
    { duration: '2m', target: 100 },  // Ramp up to 100 users
    { duration: '5m', target: 100 },  // Stay at 100 users
    { duration: '2m', target: 0 },    // Ramp down
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests under 500ms
  },
};

export default function () {
  let response = http.get('https://api.farmsense.com/api/v1/fields/123/g
  check(response, { 'status is 200': (r) => r.status === 200 });
  sleep(1);
}
```

- **Stress Testing:** Identify breaking points (max throughput, memory limits)
- **Endurance Testing:** 24-hour soak tests to detect memory leaks

15.4 CI/CD Pipeline

```
# .github/workflows/backend-ci.yml
name: Backend CI/CD

on:
```

```yaml
  push:
    branches: [main, develop]
    paths:
      - 'services/**'
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: timescale/timescaledb:latest-pg15
        env:
          POSTGRES_PASSWORD: postgres
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: Install dependencies
        run: |
          cd services/ingestion
          pip install -r requirements.txt -r requirements-dev.txt

      - name: Run tests
        run: |
          cd services/ingestion
          pytest --cov=src --cov-report=xml

      - name: Upload coverage
        uses: codecov/codecov-action@v3

  build-and-push:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - uses: actions/checkout@v3

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-west-2

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1

      - name: Build and push Docker image
        env:
          ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          IMAGE_TAG: ${{ github.sha }}
        run: |
          docker build -t $ECR_REGISTRY/farmsense-ingestion:$IMAGE_TAG servi
          docker push $ECR_REGISTRY/farmsense-ingestion:$IMAGE_TAG

      - name: Deploy to EKS
        run: |
          aws eks update-kubeconfig --name farmsense-prod --region us-west-2
          kubectl set image deployment/ingestion ingestion=$ECR_REGISTRY/far
          kubectl rollout status deployment/ingestion
```

15.5 Deployment Procedures

Production Deployment Checklist

1. ✓ All tests passing (unit, integration, E2E)
2. ✓ Code review approved by 2+ engineers
3. ✓ Database migration scripts tested in staging
4. ✓ Performance testing completed (no degradation)
5. ✓ Security scan passed (no critical vulnerabilities)
6. ✓ Monitoring dashboards updated for new metrics
7. ✓ Rollback plan documented
8. ✓ Stakeholder notification sent

Deployment Strategy

- **Blue-Green Deployment:** Zero-downtime deployment with instant rollback capability
- **Canary Releases:** Roll out to 5% of users, monitor for 1 hour, then full rollout

- **Database Migrations:** Forward-compatible migrations (additive changes first, removals later)

15.6 Monitoring & Maintenance

Key Metrics to Monitor

Category	Metric	Alert Threshold
Infrastructure	CPU utilization	> 80% for 5 minutes
	Memory utilization	> 85%
	Disk space	> 90% full
Application	API response time (p95)	> 2 seconds
	Error rate	> 1% (4xx/5xx)
	Kafka consumer lag	> 1000 messages
Business	Data ingestion rate	< 50% of normal (data loss)
	Grid recalculation time	> 30 minutes (SLA breach)
	ML model accuracy	< 70% (model degradation)

Operational Runbooks

- **Service Outage Response:** Step-by-step incident response procedures
- **Database Recovery:** Point-in-time recovery and failover procedures
- **Scaling Operations:** Manual scaling procedures for traffic spikes
- **Certificate Renewal:** TLS certificate rotation procedures

Development Best Practices:

- **Code Reviews:** All code changes require peer review before merging
- **Documentation:** Update docs alongside code changes (docs-as-code)
- **Feature Flags:** Use feature flags for gradual rollout of new features
- **Logging:** Structured logging (JSON format) with correlation IDs for request tracing

- **Version Control:** Conventional commits for automated changelog generation