AWS Scalable Web App Infrastructure

Technical Report: FarmStop

A Geospatial Agriculture Marketplace

Prepared by:

Mmela Gabriel Dyantyi
Infrastructure Engineer: Cloud Architecture Track

Date:

Jan 23, 2026

Repository: `https://github.com/MmelIGaba/FarmStop`

Contents

1 Executive Summary

FarmStop is a scalable agricultural marketplace that connects local farms with consumers through geospatial search functionality and dynamic data ingestion. The system leverages Amazon Web Services (AWS) to build a secure, highly available, and production-ready hybrid 3-tier cloud architecture. This report documents the design, implementation, and optimization of the platform's infrastructure, which integrates frontend delivery via S3 and CloudFront, compute using EC2 Auto Scaling and Lambda, persistent storage via RDS PostgreSQL (with PostGIS), and identity management with AWS Cognito.

The infrastructure was defined using **Infrastructure-as-Code (IaC)** with **Terraform**, deployed across a multi-AZ VPC with public/private subnet isolation, and automated via **GitHub Actions CI/CD pipelines**. Monitoring and alerting are implemented through CloudWatch and SNS, ensuring operational visibility.

2 Architecture Design

2.1 System Overview: Hybrid 3-Tier Cloud Architecture

FarmStop employs a **hybrid cloud-native architecture** combining traditional EC2-based workloads with modern serverless components to balance control, cost, and flexibility:

- **Frontend Tier**: Hosted on **Amazon S3** with **CloudFront CDN** for global low-latency delivery. Static React SPA (`Plaasstop`) delivered over edge locations with cache optimization. Designed for public read access with OAI (Origin Access Identity) to restrict direct S3 access.
- **Application Tier**: Backend API built in **Node.js**, hosted on **EC2 instances** managed by an **Auto Scaling Group (ASG)** across two Availability Zones. Traffic routed through an **Application Load Balancer (ALB)** in public subnets, with health checks ensuring instance reliability. Authentication handled via **Amazon Cognito**, with user identities synchronized into PostgreSQL for profile persistence. Geospatial queries executed against **RDS PostgreSQL** using the **PostGIS extension** for distance-based farm lookups (e.g., "find farms within 10km radius").
- **Data Tier**: **Amazon RDS (PostgreSQL)** runs in a private subnet, configured with automated backups, Multi-AZ deployment, and encrypted storage (AES-256). PostGIS enables spatial functions (`ST_DWithin`, `ST_Point`) critical to the core

"Find Farms" feature. Data ingestion pipeline powered by a **Dockerized Python scraper**, deployed as a **Lambda function** (container image up to 10GB), avoiding native dependency conflicts (e.g., `psycopg2`) associated with Lambda's execution environment.

- **Supporting Services**: **S3** also used for staging scraped CSV/JSON farm data. **ECR** stores the scraper Docker image. **CloudWatch Logs** and **Metrics** monitor EC2, Lambda, and ALB performance. **IAM Roles and Policies** enforce least-privilege access across components.

2.2 Network Topology

The architecture spans a custom **VPC** with the following design:

- **Public Subnets (2 AZs)**: ALB, NAT Gateway, Bastion Host (optional SSH access).
- **Private Subnets (2 AZs)**: EC2 instances (backend API), RDS instance.
- **Security Controls**: **Security Groups** restrict traffic: ALB allows HTTP/HTTPS from CloudFront; EC2 only accepts from ALB; RDS only accepts from EC2. **NACLs** enforce subnet-level allow/deny rules (e.g., block non-ephemeral inbound ports). **Route Tables** direct internet traffic via IGW (public) and NAT (private outbound only).

This design ensures high availability during AZ failure and minimizes attack surface by placing databases and application logic in isolated networks.

2.3 Architectural Rationale

The hybrid model was selected over fully serverless to meet these requirements:

By combining **serverful stability** for the API layer with **serverless agility** for intermittent tasks (scraper), FarmStop achieves optimal balance between operational control and scalability.

3 Implementation Details: IaC & CI/CD

3.1 Infrastructure-as-Code (Terraform)

All infrastructure was provisioned using **Terraform 1.8+** located under `/Infrastructure`. The codebase is modularized, enhancing reusability and maintainability:

Requirement	Why Hybrid vs. Fully Serverless
Complex Geospatial Queries	Lambda's cold starts and timeout limits (15 min) are suboptimal for large GIS operations. EC2+RDS offers consistent performance.
Long-Running Scraper Jobs	Lambda timeout constraint mitigated via **containerized Lambda**, allowing longer execution up to 15 mins, packaging complex Python modules (e.g., Selenium, Pandas).
Stateful User Profiles	Direct integration of Cognito with application database required full backend control.
Predictable Performance	ASG provides warm VMs, reducing latency compared to Lambda cold start.

Table 1: Architectural Decision Justification

- **Modules Used**:
  - `networking/`: VPC, subnets, IGW, NAT, route tables.
  - `compute/`: EC2 launch template, ASG, IAM roles.
  - `database/`: RDS PostgreSQL instance with PostGIS enabled via parameter group.
  - `load_balancing/`: ALB, target groups, listeners.
  - `storage/`: S3 buckets, ECR repository, lifecycle policies.

Example RDS PostGIS configuration snippet:

```
resource "aws_db_parameter_group" "farmstop-postgis" {
  family = "postgres15"
  name   = "farmstop-postgis-pg"
  parameter {
    name  = "rds.extensions"
    value = "postgis"
  }
}
```

Listing 1: RDS PostGIS Terraform Configuration

- **State Management**: Local state used for development (no remote backend like S3/Terraform Cloud yet).
- **Variables & Environments**: Parameterized using `.tfvars` files for future multi-environment expansion (dev/staging/prod).

The entire stack deploys from scratch in 12 minutes, verified under AWS Free Tier constraints.

3.2 Authentication & Security Integration

- **AWS Cognito User Pools** handle user registration/login (`FarmStop-Users` pool).
- Backend (Node.js) validates JWT tokens and synchronizes new users into `users` table in RDS.
- IAM roles grant:
  - EC2 instances permissions to access RDS (via security group), S3 (for logs), and CloudWatch.
  - Lambda function access to S3 (ingestion), ECR (pull), and RDS (write).

No hardcoded credentials; all secrets passed via environment variables or Parameter Store.

3.3 CI/CD Pipeline: GitHub Actions

Three primary GitHub Actions workflows are defined in `.github/workflows/`:

1. **Frontend Pipeline** (`deploy-frontend.yml`) On push to `main`:
   (a) Build React app (`npm run build`)
   (b) Sync build folder to `farmstop-frontend` S3 bucket
   (c) Invalidate CloudFront cache to purge stale assets
2. **Scraper Pipeline** (`deploy-scraper.yml`) On push to `scraper` branch:
   (a) Build Docker image using `Dockerfile.scraper`
   (b) Tag and push to ECR
   (c) Update Lambda (`FarmStop-Scraper`) with new image URI
3. **Infrastructure Pipeline** (`terraform.yml`) On merge to `main`:
   (a) `terraform init`
   (b) `terraform plan` (review required)
   (c) Manual approval via GitHub Environments
   (d) `terraform apply` — executes only after approval

Pipeline enforces code reviews, promotes auditability, and prevents accidental drift.

4 Monitoring, Security, and Optimization

4.1 Observability: CloudWatch & SNS

- **Logs**:
  - EC2 logs streamed via CloudWatch Agent (`/var/log/farmstop/*.log`).
  - Lambda logs automatically sent to `/aws/lambda/FarmStop-Scraper`.
- **Metrics**:
  - CPUUtilization (EC2, ASG)
  - NetworkIn/NetworkOut
  - DatabaseConnections (RDS)
  - HTTPCode_ELB_5XX_Count (ALB)
- **Alarm**:
  - High CPU Alarm (`FarmStop-HighCPU`) triggers when ASG average CPU ¿ 80% for 5 minutes → sends SNS notification to `farmstop-alerts@domain.com`.

A basic CloudWatch dashboard visualizes frontend requests, backend latency, database load, and scraper execution frequency.

4.2 Security Controls Implemented

Layer	Controls
Network	Private subnets, security groups, NACLs, no public RDS access
Compute	Minimal IAM roles, no SSH from public internet
Data	RDS encryption at rest (KMS), S3 object encryption, no plaintext secrets
Identity	Cognito with MFA recommended
Threat	None yet; future: GuardDuty, Macie, AWS WAF

Table 2: Security Controls by Layer

4.3 Performance & Cost Optimization

- **Caching Strategy**: Not yet implemented. Currently under evaluation for **Redis on ElastiCache** to cache frequent spatial queries (e.g., popular region lookups). Currently relying on RDS instance performance (db.t3.medium).
- **Auto Healing & Scaling**:

  - ASG scales between 2–6 instances based on CPU threshold (target tracking = 60%).
  - ELB performs `/health` checks every 30 seconds with 5s timeout.

- **Cost Efficiency**:
  - S3 storage lifecycle rules delete older ingestion files after 30 days.
  - EC2 uses spot instances where feasible (not yet enabled).
  - RDS scheduled backups retained for 7 days.

5 Challenges, Limitations, and Future Improvements

5.1 Known Challenges Encountered

1. **Lambda Container Dependency Hell**:
   - Installing `psycopg2`, `geopandas`, and `shapely` in Lambda was impossible via standard layers due to compiled binaries.
   - **Solution**: Migrated scraper to container-based Lambda with custom Docker image, building on Amazon Linux 2 image with precompiled wheels.
2. **PostGIS in RDS Not Enabled by Default**:
   - Required manual creation of parameter group and binding to instance.
   - PostGIS extension had to be manually enabled per database: `CREATE EXTENSION postgis;`.
3. **Terraform State Conflicts in Team Context**:
   - Local state caused issues in collaborative testing; resolved temporarily with `.tfstate` locking convention.

5.2 Current Limitations

5.3 Future Improvements

1. **Secure Communications**:
   - Register domain via Route 53.
   - Request public certificate via **ACM**.
   - Enforce HTTPS on ALB and redirect HTTP → HTTPS.

Limitation	Impact	Status
No ACM Certificate on ALB	Currently using HTTP/self-signed SSL	High Priority
No WAF Protection	Vulnerable to SQLi/XSS attacks	Planned
No Response Caching	Repeated GIS queries affect DB load	Design Phase
ElastiCache Not Provisioned	Latency spikes under load	Future Roadmap
Single-Region Deployment	No disaster recovery	Phase 2 Goal
Manual Terraform Apply	Risk of human error	Move to Terraform Cloud

Table 3: System Limitations and Status

2. **Enhanced Security**:
   - Deploy **AWS WAF** in front of ALB to filter malicious traffic using managed rule sets (e.g., SQLi, XSS).
   - Integrate **AWS Shield Advanced** and **GuardDuty** for threat detection.
3. **Performance Scaling**:
   - Add **ElastiCache (Redis)** for caching geospatial query results.
   - Enable **RDS Read Replicas** for reporting/analytics workloads.
4. **Observability Upgrade**:
   - Instrument with **AWS X-Ray** to trace request path from ALB $\rightarrow$ EC2 $\rightarrow$ RDS $\rightarrow$ Lambda.
   - Forward logs to **OpenSearch or third-party SIEM**.
5. **CI/CD Maturity**:
   - Add automated unit/integration testing in GitHub Actions.
   - Implement canary deployments using **CodeDeploy** for backend updates.
6. **AI-Driven Data Enrichment (Future Vision)** Inspired by research in AI-powered security and data platforms (e.g., Wiley's *Security and Privacy: An Artificial Intelligence Approach*), future versions could:
   - Use **ML models** in SageMaker to predict farm availability or pricing trends.
   - Apply **anomaly detection** to log streams for early breach detection.

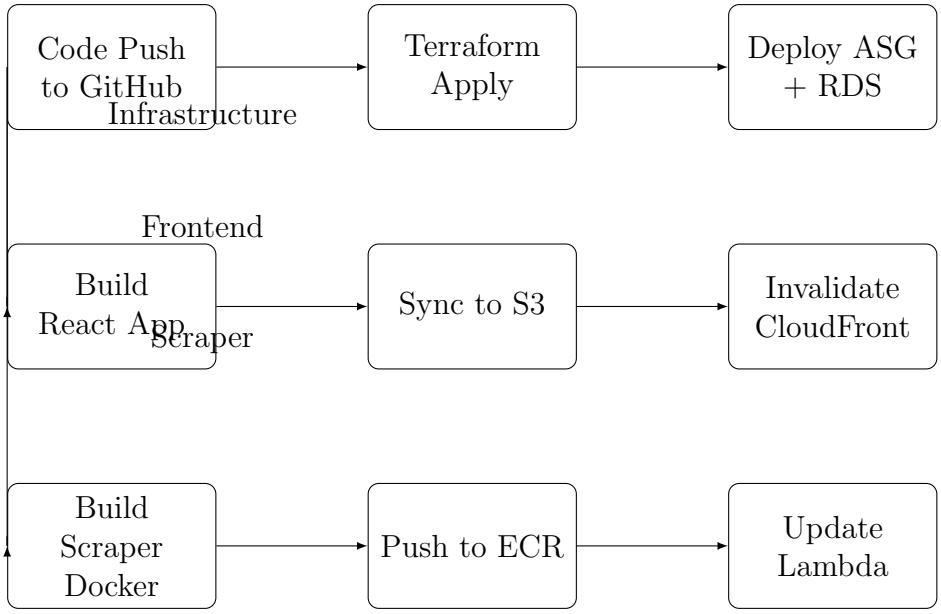

Figure 1: CI/CD Pipeline Workflow

6 Conclusion

FarmStop demonstrates a robust, production-oriented cloud infrastructure built using AWS best practices. Through a **hybrid 3-tier architecture**, it achieves a balance between scalability, performance, and maintainability. The use of **Terraform**, **GitHub Actions**, and **modular design** ensures repeatability, automation, and operational clarity.

Key achievements include successful geospatial query execution over PostGIS, secure identity synchronization between Cognito and PostgreSQL, and reliable containerized scraping via Lambda. Challenges such as dependency management and state isolation were overcome with thoughtful engineering decisions.

While HTTPS enforcement and WAF integration remain outstanding, the system is highly functional and well-positioned for enhancements in security, performance, and observability. With planned upgrades in caching, cross-region deployment, and intelligent data analysis, FarmStop is primed to evolve into a resilient and scalable agri-tech marketplace on AWS.

Appendices (Available in GitHub Repository):

- `https://github.com/MmelIGaba/FarmStop`
- `/Infrastructure/` – Terraform modules and deployment scripts
- `/Plaasstop/`, `/backend/`, `/scraper/` – Application source
- `/docs/diagrams/farmstop-architecture.png` – VPC and flow diagram

- `.github/workflows/` – CI/CD pipelines