# Project - High Level Design

# on

# IAC-PROVISIONING

# AGRICULTURE-SYSTEM

Course Name: DevOps Fundamentals

***Institution Name:*** Medicaps University – Datagami Skill Based Course

*Student Name(s) & Enrolment Number(s):*

| Sr no | Student Name | Enrolment Number |
|---|---|---|
| 1 | Vidhi Gupta | EN22CS3011075 |
| 2 | Tejashwi Bharti | EN22CS3011036 |
| 3 | Tanishka Soni | EN22CS3011020 |
| 4 | Tanish Patidar | EN22CS3011016 |
| 5 | Sriyansh Shrivastava | EN22CS301986 |

Group Name: 11D9

Project Number: D0-11

Industry Mentor Name: Mr. Vaibhav

University Mentor Name: Dr. Ritesh Joshi

Academic Year: 2025-2026

# Table of Contents

**Backend Layer (Flask + Gunicorn)** – Processes API requests and business logic.

**Database Layer (MySQL)** – Stores user and crop data.The backend communicates with the database using secure internal Docker networking.

### 2.2. Process Flow:

- Developer pushes code to Git repository.
- Terraform provisions AWS EC2 infrastructure.
- Docker is installed on the server.
- Docker Compose builds and starts backend and database containers.

Users access the application via Public IP and exposed port.

### 2.3. Information Flow:

- User sends request through browser.
- Request reaches backend API.
- Backend processes logic and interacts with MySQL database.
- Database sends response to backend.
- Backend returns response to user.

**Flow:**
User → Frontend → Backend → Database → Backend → User

### 2.4. Components Design:

The system consists of the following components:

- **Terraform Module** – Provisions AWS EC2 and networking.
- **Docker Engine** – Manages containers.
- **Backend Container** – Runs Flask application using Gunicorn.
- **Database Container** – Runs MySQL server.
- **Docker Network** – Enables secure communication between containers.

Each component is isolated and independently manageable.

### 2.5. Key Design Considerations:

- Infrastructure automation using IaC
- Containerization for portability
- Separation of concerns (3-tier architecture)
- Scalability through cloud deployment
- Reduced manual configuration errors
- Security via Docker networking and controlled ports

### 2.6. API Catalogue:

- GET / → Health check / Backend status
- POST /users → Create new user
- GET /users → Fetch user details

- POST /crops → Add crop data
- GET /crops → Retrieve crop records

All APIs follow RESTful principles and communicate using JSON format.

## Flow Diagram

 Developer pushes code to GitHub.

 CI/CD pipeline triggers.

 Terraform provisions infrastructure.

 Ansible installs:

- Docker

Docker Compose

 Docker builds backend and frontend images.

 Containers are started.

 Application becomes accessible via Public IP.

User Browser

↓

Frontend (React/Vite)

↓

AWS EC2 Server

↓

Docker Engine

↓

Flask API (Docker Container)

↓

MySQL Database (Docker Container or RDS)

↓

Response to Frontend

**Infrastructure Components:**

- EC2 Instance
- Security Groups (Ports 22, 5000)
- Docker Engine
- Docker Compose

**Application Components:**

- agri-backend container
- agri-mysql container
- Frontend container

**3 Key Design Considerations**

- Infrastructure reproducibility
- Scalability
- Container isolation
- Automated provisioning
- Secure credential management (.env)
- Minimal manual intervention

**Interfaces**

User Browser

↓

Frontend (React/Vite)

↓

AWS EC2 Server

↓

Docker Engine

↓

Flask API (Docker Container)

↓

MySQL Database (Docker Container or RDS)

↓

Response to Frontend

**Data Design**

**Entities:**

- User
- Crop

**13. Data Model**

**User**

- id
- name
- email
- password

**Crop**

- id
- crop_name
- season
- price

---

**14. Data Access Mechanism**

- MySQL Connector (Python)
- ORM-like structured service layer
- Docker internal networking

---

**15. Data Retention Policies**

- Data stored in MySQL volume
- Backup via manual export or snapshot
- EC2 EBS snapshot supported

---

**16. Data Migration**

- Terraform supports new environment provisioning
- Database schema auto-created at container startup
- Manual SQL migration supported

---

**17. Interfaces**

**External Interfaces:**

- AWS Cloud
- GitHub Repository
- Jenkins / GitHub Actions

**Internal Interfaces:**

- REST APIs
- Docker Network

---

**18. State and Session Management**

- Stateless backend

- JWT-based authentication (if implemented)
- No session persistence on server

---

**19. Caching**

- No caching implemented (optional: Redis)
- Can be extended for performance optimization

---

**20. Non-Functional Requirements**

- Availability: 99%
- Scalability: Horizontal via Docker/K8s
- Maintainability: Modular code
- Portability: Dockerized environment

---

**21. Security Aspects**

- Security Group firewall rules
- SSH Key-based access
- Environment variables for secrets
- No hardcoded credentials
- Docker network isolation

---

**22. Performance Aspects**

- Gunicorn multi-worker support
- Docker containerization
- Lightweight Python slim image
- MySQL optimized connection

---

**23. References**


- Terraform Documentation
- Docker Documentation
- Flask Documentation
- AWS EC2 Documentation
- GitHub Actions Documentation