AegisAlert Deployment Documentation

Overview

The AegisAlert platform is deployed across two AWS EC2 instances, with a focus on security, scalability, and modular architecture. The backend services run in a private EC2 instance within a private subnet, while the public-facing frontend and traffic routing are managed by a public EC2 instance.

---

1. Technology Stack

Frontend:

- Svelte framework for building a responsive UI
- Mapbox for interactive map integration

Backend:

- Django with Django REST Framework for API and application logic
- Django GIS for geospatial data processing
- Celery with Redis for asynchronous task management

Database:

- PostgreSQL with PostGIS extension for geospatial support

Deployment Tools:

- Docker and Docker Compose for container orchestration
- Amazon Linux 2 as the server OS

- Ansible for automation and provisioning
- Terraform for Infrastructure as Code (IaC) to provision AWS resources

---

2. Infrastructure Details

EC2 Instances:

Public EC2 Instance

- Private IP: `10.0.1.105`
- Public IP: `34.230.36.210`
- **Roles:** Frontend hosting, ingress traffic management

Private EC2 Instance

- Private IP: `10.0.2.191`
- **Roles:** Backend services hosting, no direct internet exposure

Both EC2 instances and related networking components were provisioned and managed using **Terraform**, ensuring reproducible and version-controlled infrastructure deployment.

Running Services on Private EC2:

- Django web application container running on port 8000
- PostgreSQL database container with PostGIS enabled
- Redis container
- Celery worker and beat containers for background task processing
- All containers are operational and healthy as per Docker health checks

---

3. Network and Security Configuration

Security Groups:

- Configured to permit port 8000 traffic from the client's IP to the public EC2 instance
- Allowed communication between public and private EC2 instances over necessary ports
- Restricted direct public access to private EC2

Note on Security Configuration (Mock Environment):
For this mock deployment, SSH (port 22) and Django development port (8000) were temporarily opened to `0.0.0.0/0` for testing purposes. This is **not recommended for production** environments. Best practices would involve IP-restricted access for SSH and backend services, or use of AWS SSM and bastion hosts for secure connections.

IP Forwarding and Traffic Routing:

- IP forwarding enabled at the kernel level on the public EC2
- Configured iptables with DNAT rules to forward incoming port 8000 traffic on the public EC2 to the private EC2 backend
- SNAT and FORWARD chains established to ensure correct bidirectional traffic flow
- This setup maintains backend security while allowing seamless access through the public endpoint

---

4. Containerization

Separate Docker Compose configurations were implemented to isolate frontend and backend services:

Backend Compose File:

Includes Django, PostgreSQL, Redis, and Celery services

Frontend Compose File:

Contains the Svelte application container

This separation facilitates independent scaling and deployment of frontend and backend components.

---

5. Deployment Process Summary

- Infrastructure provisioning automated using **Terraform** to create EC2 instances, security groups, and networking components
- Provisioned EC2 instances with **Amazon Linux 2**
- Automated installation of Docker and Docker Compose via **Ansible** playbooks
- Transferred application source code to respective instances securely
- Built and deployed Docker containers using **Docker Compose**
- Verified container health and connectivity
- Established secure networking and routing for public access to backend APIs