AEGIS – Technical Installation Manual

Client: Tyto Insights × DNS.Business
Project: **AEGIS**(Advanced Evidence Gathering and Investigation System) – Digital Forensics Case Management System
Team: Incident Intel
Version: 1.0
Date: August 2025

Table of Contents

1. Introduction

AEGIS is a secure, containerised digital forensics and investigative collaboration platform. It enables case management, evidence handling, chain-of-custody, reporting, and secure real-time collaboration for DFIR teams. This manual describes prerequisites, configuration, installation, deployment and verification steps for developers and operators.

1.1 System Components

- Frontend: React + TypeScript web app (TailwindCSS, component library).
- Backend: Go microservices (API Gateway, Case Service, Evidence Service, Report Service, Audit Logging, Secure Chat).
- Storage: **IPFS** (content-addressed evidence storage via Kubo; RPC API :5001, Gateway :8080)
- Infrastructure: Docker & Docker Compose for containerized deployment.
- Realtime: WebSockets / Socket.IO for secure chat, presence, and annotation threads.
- Security: OAuth 2.0 + JWT, RBAC, encryption in transit (TLS) and at rest.
- DevOps: Docker & Docker Compose for environment parity; GitHub Actions for CI/CD deployments over SSH.

1.2 Prerequisites

Operating System:

- Ubuntu 22.04 LTS (recommended for deployment)
- Windows 10/11 or macOS (for local development)

Core Services

- Git ≥ 2.40
- Docker ≥ 24.0
- Docker Compose ≥ 2.20
- Node.js ≥ 20.0 (with npm or pnpm) for frontend dev
- Go ≥ 1.24
- PostgreSQL ≥ 15
- MongoDB ≥ 7.0
- IPFS **Kubo** (go-ipfs) ≥ **0.26** (or Docker image `ipfs/kubo:latest`)

2. Installation

Clone the Repository:

```
git clone https://github.com/COS301-SE-2025/AEGIS.git
cd AEGIS
```

Setup Environment V ariables:

```
cp .env.example .env
nano .env
```

Create your environment file from the example and edit values:

cp .env.example .env

- *Key variables (typical):*
- *POSTGRES_USER=aeGIS*
- *POSTGRES_PASSWORD=change_me*
- *POSTGRES_DB=aegis*
- *POSTGRES_HOST=postgres*
- *POSTGRES_PORT=5432*
- *MONGO_URI=mongodb://mongo:27017/aegis*
- *MONOG_INITDB_ROOT_USERNAME=userName*
- *MONGO_INITDB_ROOT_PASSWORD=mongo_pass..*
- *MONGO_DB_NAME=mongo_db_name*
- *IPFS_API=http://ipfs:{port}/*
- *JWT_SECRET_KEY=secret_key*

Backend Installation:

```
cd api
go mod tidy
```

Frontend Installation:

```
cd frontend
npm install   # or pnpm install
```

3. Deployment / Running

3.1 Local Development (Docker compose)

Run the container:

- docker-compose up -d --build

This starts:

- Frontend at http://localhost:5173
- PostgreSQL, IPFS & MongoDB
- API Gateway + Microservices

Frontend (Dev Mode):

- cd frontend
- npm run dev

Backend (Dev Mode):

- cd api
- go mod tidy
- go run main.go

3.2 Production Deployment (UBUNTU SERVER)

Install Docker & Compose (Ubuntu 22.04):

- sudo apt-get update && sudo apt-get install -y ca-certificates curl gnupg
- sudo install -m 0755 -d /etc/apt/keyrings
- curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
- echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo $VERSION_CODENAME) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

- sudo apt-get update && sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

Create a deploy directory and set environment:

- sudo mkdir -p /opt/aegis && sudo chown $USER:$USER /opt/aegis
- cd /opt/aegis && git clone https://github.com/COS301-SE-2025/AEGIS.git
- cd AEGIS && cp .env.example .env && nano .env

Start services

- docker compose up -d --build

4 CI/CD via GitHub Actions (SSH deploy)

Secrets required in GitHub → Settings → Secrets and variables → Actions:

- SERVER_HOST = <your.server.host>
- SERVER_USER = <linux_user>
- SERVER_SSH_PORT = <ssh_port>
- SERVER_SSH_KEY = <private_key (OpenSSH format)>

Minimal workflow step:

```
uses: appleboy/ssh-action@v0.1.6
 with:
  host: ${{ secrets.SERVER_HOST }}
  username: ${{ secrets.SERVER_USER }}
  key: ${{ secrets.SERVER_SSH_KEY }}
  port: ${{ secrets.SERVER_SSH_PORT }}
  script: |
        cd /home/${{ secrets.SERVER_USER }}/AEGIS
        git pull origin main
        docker compose down || docker-compose down
        docker compose up -d --build || docker-compose up -d --build
```

Testing the SSH key locally before committing:

```
ssh -i ~/.ssh/gha_ed25519 -p <port> <user>@<host> 'echo ok'
```

5 Post-Installation Verification

- Frontend (dev):open http://localhost:5173 and load AEGIS UI
- API health: http://<server-or-local>:8080/health
- IPFS evidence flow:
    1. Upload a small file through the app
    2. Read via gateway
        - curl -X POST -F file=@test.txt "$IPFS_API_URL/api/v0/add"

6 Security Hardening Checklist

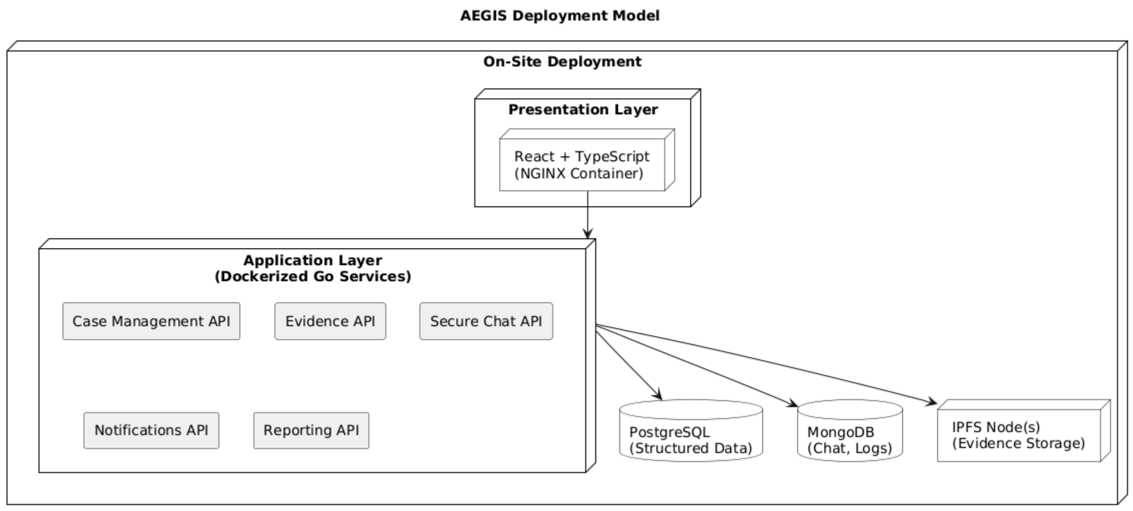
- Enforce HTTPS (TLS) via reverse proxy (Nginx/Traefik).
- Create a non-root deploy user; restrict SSH by key only; disable password auth.
- Rotate JWT secrets and storage credentials; use least-privilege access.
- Enable firewall rules (allow 80/443 and custom SSH port).
- Regularly update images and dependencies; monitor with Dependabot/Trivy.

7. Troubleshooting

- SSH: 'could not parse as int for flag port' → your port secret is empty or not an integer. Ensure SERVER_SSH_PORT is set (e.g., 22).
- SSH: 'ssh.ParsePrivateKey: ssh: no key found' → paste a valid OpenSSH private key into SERVER_SSH_KEY (include BEGIN/END lines).
- SSH: 'handshake failed: unable to authenticate' → check user/host/port and that the public key is in ~/.ssh/authorized_keys on the server.
- Docker: service won't start → run 'docker compose logs -f <service>' and check .env values (DB hosts, credentials).

8 Deployment Model

- For production, deploy on an Ubuntu server with Docker Compose.
- Reverse proxy: Nginx or Traefik for SSL/TLS termination.
- Database volumes mapped to persistent storage.
- CI/CD pipeline auto-deploys from GitHub → server via SSH.



9 Additional Notes

- All dependencies are version-pinned in package.json (frontend) and go.mod (backend).
- Use docker-compose down -v to reset all containers and volumes.
- For testing: run npm run test (frontend) and go test ./... (backend).