HostelOps

Production Deployment of a Containerized Complaint Management System

1.Running Deployed Application (Docker-Based Execution)

The application is deployed using Docker and Docker Compose.

Technologies Used:

- React.js (Frontend)
- Node.js + Express.js (Backend)
- MongoDB (Database)
- Docker
- Docker Compose
- Nginx (Reverse Proxy)

Execution Command:

```
docker compose up
```

Container Status:

- nginx → Port 80 exposed publicly
- backend → Internal container (port 5000)
- mongo → Internal container (port 27017)
- frontend → Internal container (served via Nginx)

Only Port 80 is publicly accessible.

2.Architecture Diagram (Container + Reverse Proxy + Port Flow)

Architecture Flow:

Client Browser
↓
Port 80 (Public)
↓
Nginx Reverse Proxy (Container)
↓
Backend Container (Node + Express)
↓
MongoDB Container

Architecture Explanation:

- All traffic enters via Port 80
- Nginx acts as reverse proxy
- /api/* routes to backend container
- Backend communicates with MongoDB
- MongoDB is not publicly exposed

This demonstrates production-style container architecture

3.Nginx Configuration Explanation

Nginx is used as a reverse proxy layer.

Why Nginx?

- Centralized request handling
- API routing
- Backend port protection
- Production-like request management

Routing Strategy:

- / → Frontend container
- /api/* → Backend container
-

Request Flow:

Client → Nginx → Backend → Database → Response returned

4.Dockerfile & Container Explanation

Backend Dockerfile:

- Uses Node base image
- Copies application files
- Installs dependencies
- Runs Express server
- Exposes port 5000 internally

Frontend Dockerfile:

- Multi-stage build
- Stage 1: Build React production bundle
- Stage 2: Serve build via Nginx

docker-compose.yml:

- Defines services (frontend, backend, mongo, nginx)
- Creates internal Docker bridge network
- Controls port exposure
- Ensures container restart safety

5.Networking & Firewall Strategy

Port Binding Strategy:

0.0.0.0:80 → nginx:80

Only Port 80 is exposed externally.

Internal Services:

- Backend (5000) → Internal only
- MongoDB (27017) → Internal only
- Frontend container → Internal only

Security Strategy:

- No direct database exposure
- No direct backend exposure
- JWT authentication used
- Role-based access control

6.Request Lifecycle Explanation

Example: Student submits complaint

1. Browser sends HTTP request
2. Request reaches Nginx via Port 80
3. Nginx routes /api/complaints to backend container
4. Backend validates JWT
5. Backend processes request
6. Backend interacts with MongoDB
7. MongoDB returns response
8. Backend sends JSON response
9. Nginx forwards response to client

7.Serverful vs Serverless Comparison

This Project Uses: Serverful Architecture

Because:

- Infrastructure is manually managed
- Docker containers are configured
- Nginx reverse proxy is configured
- Networking rules are defined
- Ports are controlled manually

Serverless Architecture Would:

- Use managed cloud functions
- No manual server configuration

- Auto-scaling managed by cloud provider
- Example: AWS Lambda, Firebase Functions

Feature	**Serverful Architecture (Used in HostelOps)**	**Serverless Architecture**
Infrastructure Control	Fully managed by developer (Docker, Nginx, networking)	Managed by cloud provider
Server Management	Developer configures and maintains containers	No server management required
Deployment Style	Container-based deployment using Docker	Function-based deployment (FaaS)
Scalability	Manual scaling required	Auto-scaling handled by provider
Networking Control	Manual port binding and firewall control	Abstracted networking
Reverse Proxy	Nginx configured manually	Usually handled by platform
Cost Model	Fixed server/container cost	Pay-per-request model
Environment Configuration	Managed via Docker environment variables	Managed via cloud environment configs
Example Technologies	Docker, Nginx, Node.js, MongoDB	AWS Lambda, Firebase Functions
Control Level	High control over infrastructure	Limited infrastructure control
Learning Outcome	Deep understanding of DevOps & deployment	Focus mainly on application logic