# **Project Document – Gen AI DevOps Assistant**

**1. Project Overview**

**We are building a Gen AI-based DevOps Assistant that provides:**

**Interview Bot – helps with DevOps interview preparation.**

**Code Explainer – explains code snippets with real-time examples.**

**DevOps Assistant – assists with troubleshooting common DevOps issues.**

**The app has a 3-tier architecture:**

**Frontend: Next.js (React framework)**

**Backend: Python (FastAPI)**

**LLM (AI Model): OpenAI GPT-3.5 Turbo (via API key)**

**Deployment will be done via Docker and Kubernetes, with CI/CD pipeline integration.**

**2. Architecture Diagram**

User ---> Frontend (Next.js) ---> Backend (Python FastAPI) ---> LLM Model (OpenAI API)

Frontend: Collects user input, sends it to backend.

Backend: Acts as proxy, applies prompt templates, forwards to LLM.

LLM API: Generates response.

Database (Optional): For logging queries & responses.

**3. Tech Stack**

Languages: Python, JavaScript

Frameworks: FastAPI (backend), Next.js (frontend)

Containerization: Docker, Docker Compose

Orchestration: Kubernetes (K8s)

Cloud: AWS EC2 (for testing), can extend to EKS/GKE/AKS

CI/CD: GitHub Actions / Jenkins

Monitoring: Prometheus + Grafana

**4. Implementation Steps**

**Step 1 – Local Setup**

1. Launch Ubuntu VM on AWS EC2 (t2.medium, ports 3000 & 8000 open).

2. Update packages:

sudo apt update && sudo apt upgrade -y

3. Install dependencies:

Python + venv

Node.js (via NVM)

Git

**Step 2 – Backend Setup (FastAPI)**

1. Clone repo & create virtual environment:

git clone <repo\_url>

cd backend

python3 -m venv venv

source venv/bin/activate

2. Install dependencies:

pip install -r requirements.txt

3. Create .env file with OpenAI API key:

OPENAI\_API\_KEY=your\_api\_key\_here

4. Run backend:

uvicorn main:app --host 0.0.0.0 --port 8000

**Step 3 – Frontend Setup (Next.js)**

1. Go to frontend directory:

cd frontend

2. Install dependencies:

npm install

3. Create .env file:

NEXT\_PUBLIC\_API\_BASE\_URL=http://<EC2-IP>:8000

4. Start frontend:

npm run dev

5. Access app → http://<EC2-IP>:3000

**Step 4 – Dockerization**

Backend Dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt --no-cache-dir

COPY . .

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

Frontend Dockerfile

FROM node:18-alpine AS build

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build

FROM node:18-alpine

WORKDIR /app

COPY --from=build /app/.next .next

COPY --from=build /app/node\_modules node\_modules

COPY --from=build /app/public public

COPY --from=build /app/package.json .

EXPOSE 3000

CMD ["npm", "start"]

Docker Compose

version: "3"

services:

  backend:

    build: ./backend

    ports:

      - "8000:8000"

    env\_file:

      - ./backend/.env

  frontend:

    build: ./frontend

    ports:

      - "3000:3000"

    depends\_on:

      - backend

Run:

docker-compose up --build

**Step 5 – Kubernetes Deployment**

Create K8s manifests:

deployment-backend.yaml

deployment-frontend.yaml

service-backend.yaml

service-frontend.yaml

Apply:

kubectl apply -f k8s/

Expose frontend with LoadBalancer / Ingress.

**Step 6 – CI/CD Pipeline**

**1. GitHub Actions Workflow (.github/workflows/deploy.yml)**

Steps:

Checkout code

Run unit tests

Build Docker images

Push to Docker Hub/ECR

Deploy to Kubernetes (via kubectl)

**2. Optional Jenkins Setup:**

Jenkins pipeline to build/test/deploy on every commit.

**Step 7 – Monitoring & Logging**

Prometheus + Grafana for metrics.

ELK Stack (Elasticsearch, Logstash, Kibana) for logs.

**5. Best Practices**

* Use .env files, never hardcode API keys.
* Multi-stage Docker builds for lightweight images.
* GitHub Actions/Jenkins for CI/CD.
* Kubernetes readiness & liveness probes.
* Auto-scaling using HPA (Horizontal Pod Autoscaler).

**6. Future Improvements**

* Integrate database (MongoDB/Postgres) for storing queries.
* Add authentication (JWT/OAuth2).
* Support multiple AI models (Claude, LLaMA, Gemini).
* Deploy on serverless (AWS Lambda + API Gateway).
* Add observability (OpenTelemetry).