12. Vẽ và giải thích góc nhìn triển khai của thiết kế Event Sourcing được đề xuất trong bài lab 03? Liệt kê các công cụ có thể sử dụng và các bước cần thực hiện để triển khai hệ thống theo góc nhìn đề xuất.

**Trả lời**

**Deployment Diagram**

A diagram of a computer

AI-generated content may be incorrect.

1. **Tổng quan hệ thống:**

Hệ thống quản lý đơn hàng này được triển khai với kiến trúc Event Sourcing và CQRS, tập trung vào khả năng mở rộng, giám sát và quản lý môi trường.

* 1. Môi trường triển khai (Deployment Environments)
* Development (Phát triển):
  + Sử dụng Dev Branch và Dev DB (cơ sở dữ liệu phát triển).
  + Môi trường này dành cho các nhà phát triển để làm việc trên các tính năng mới hoặc sửa lỗi.
* Staging (Thử nghiệm):
  + Có Preview Deploy và Staging DB (cơ sở dữ liệu thử nghiệm).
  + Đây là môi trường tiền sản xuất, nơi các tính năng mới được kiểm thử kỹ lưỡng trước khi triển khai ra môi trường Production.
* Production (Sản xuất):
  + Đây là môi trường trực tiếp phục vụ người dùng cuối.
  + Bao gồm SSR Server (Server-Side Rendering cho Frontend), Event Store (lưu trữ sự kiện) và Read Projection DB (cơ sở dữ liệu đọc).
* CI/CD: Quá trình triển khai giữa các môi trường được tự động hóa thông qua các công cụ CI/CD, cho phép đẩy mã từ Staging lên Production (Promote).
  1. Hạ tầng và Thành phần chính trong Production Environment
* Client Devices (Thiết bị người dùng):
  + Người dùng truy cập hệ thống qua Web Browser (Chrome, Firefox, Edge) hoặc Mobile Browser (iOS/Android).
* CDN (Content Delivery Network):
  + Phục vụ các Static Assets (JS, CSS, Images) cho Frontend, giúp tăng tốc độ tải trang và giảm tải cho các máy chủ chính.
* Frontend Server:
  + Chạy ứng dụng Next.js App (Order UI) ở chế độ SSR.
  + Tương tác với API Server thông qua Load Balancer.
* Load Balancer:
  + Gồm API Load Balancer và Frontend Load Balancer.
  + Đảm bảo phân phối lưu lượng truy cập đồng đều đến các máy chủ API và Frontend, cung cấp tính sẵn sàng cao và khả năng mở rộng.
* API Server (Node.js):
  + API Gateway: Cổng vào duy nhất cho các API, xử lý các tác vụ như xác thực, ủy quyền, và giới hạn tốc độ (rate limiting).
  + Controller Layer: Xử lý các yêu cầu từ client, điều phối đến Command Handler hoặc Query Handler.
  + Command Handler: Xử lý các yêu cầu ghi dữ liệu, tạo ra các sự kiện và gửi đến Event Store (append()).
  + Query Handler: Xử lý các yêu cầu đọc dữ liệu, truy vấn từ Read Model DB (readProjection()).
* Database Server:
  + Event Store (PostgreSQL - events): Lưu trữ tất cả các sự kiện thay đổi trạng thái hệ thống. Đây là nguồn sự thật duy nhất và bất biến.
  + Snapshot Store (PostgreSQL or MongoDB): Lưu trữ các bản ghi trạng thái (snapshot) của các đối tượng nghiệp vụ tại các thời điểm nhất định để tối ưu hóa quá trình khôi phục trạng thái, giảm thiểu thời gian cần thiết để replay các sự kiện.
  + Read Model DB (PostgreSQL or MongoDB): Cơ sở dữ liệu riêng biệt được thiết kế và tối ưu hóa cho các truy vấn đọc. Dữ liệu trong Read Model được xây dựng bằng cách chiếu (project) từ các sự kiện trong Event Store.
  1. Các dịch vụ hỗ trợ và Giám sát (Admin's Perspective)
* External Services (Dịch vụ bên ngoài):
  + Auth Service (JWT, OAuth2): Dịch vụ xác thực và cấp quyền cho người dùng, đảm bảo bảo mật hệ thống.
  + Notification Service (Email/SMS): Dịch vụ gửi thông báo tự động cho người dùng (ví dụ: xác nhận đơn hàng, cập nhật trạng thái).
* Monitoring (Giám sát):
  + Log Aggregator: Thu thập và tập trung các log từ tất cả các thành phần hệ thống, hỗ trợ việc theo dõi, gỡ lỗi và phân tích sự cố.
  + Metrics Dashboard (Grafana/Prometheus): Cung cấp các biểu đồ và số liệu về hiệu suất hệ thống (CPU, RAM, latency, request per second), giúp quản trị
  1. Luồng triển khai & tương tác
  + User gửi request từ trình duyệt hoặc thiết bị di động.
  + CDN trả về static assets (JS/CSS/Image).
  + Frontend Load Balancer phân phối request đến Next.js server để render giao diện.
  + Khi người dùng thao tác (ví dụ tạo đơn hàng), request được gửi qua API Load Balancer đến API Server.
  + API Gateway định tuyến request tới Controller Layer.
  + Controller gọi Command Handler để ghi dữ liệu hoặc Query Handler để đọc dữ liệu.
  + Command Handler ghi event vào Event Store và snapshot nếu cần.
  + Query Handler đọc từ Read Model DB để trả kết quả nhanh.
  + Các external services (Auth, Notification) được gọi khi cần xác thực hoặc gửi thông báo.
  + Log và metrics được gửi tới Log Aggregator và Metrics Dashboard để giám sát.
  + viên theo dõi sức khỏe và phát hiện các vấn đề tiềm ẩn.
  1. Ý nghĩa trong triển khai thực tế
* Tách biệt frontend và backend giúp mở rộng độc lập.
* Load balancer bảo đảm khả năng mở rộng và tính sẵn sàng cao.
* Event Store và Read Model DB tách biệt giúp ghi và đọc tối ưu theo CQRS.
* External services giúp giảm tải cho hệ thống chính và dễ tích hợp.
* Multi-environment deployment giúp CI/CD mượt mà từ dev → staging → production.

1. **Các công cụ đề xuất cho từng thành phần**

|  |  |  |
| --- | --- | --- |
| **Thành phần** | **Công cụ đề xuất** | **Lý do lựa chọn** |
| **Cloud Provider** | AWS (Amazon Web Services) | Hệ sinh thái dịch vụ đầy đủ, phổ biến, và tích hợp tốt với nhau. |
| **Containerization** | Docker, Docker Compose | Tiêu chuẩn ngành để đóng gói ứng dụng và môi trường, giúp đồng nhất giữa các môi trường. |
| **Orchestration** | Kubernetes (AWS EKS) | Quản lý, mở rộng và tự động hóa việc triển khai container ở quy mô lớn, phù hợp với kiến trúc microservices. |
| **Infrastructure as Code** | Terraform | Quản lý tài nguyên hạ tầng bằng code, giúp tự động hóa, tái sử dụng và theo dõi thay đổi một cách an toàn. |
| **CI/CD** | GitHub Actions | Tích hợp sẵn với GitHub, dễ cấu hình, hỗ trợ mạnh mẽ cho việc build, test, và deploy Docker/Kubernetes. |
| **Database Server** | AWS RDS for PostgreSQL | Dịch vụ PostgreSQL được quản lý, tự động backup, vá lỗi, và dễ dàng mở rộng. |
| **CDN** | AWS CloudFront | Tích hợp tốt với các dịch vụ AWS khác (S3, Load Balancer), hiệu năng cao trên toàn cầu. |
| **Load Balancer** | AWS Application Load Balancer (ALB) | Cân bằng tải lớp 7 (HTTP/HTTPS), linh hoạt, tích hợp với Kubernetes Ingress. |
| **API Gateway** | AWS API Gateway | Dịch vụ được quản lý, xử lý xác thực, rate limiting, caching, và tích hợp với AWS Lambda hoặc các dịch vụ backend. |
| **Auth Service** | AWS Cognito hoặc Auth0 | Cung cấp giải pháp xác thực và quản lý người dùng hoàn chỉnh (JWT, OAuth2), giảm tải cho đội phát triển. |
| **Notification Service** | AWS SES (Email), AWS SNS (SMS) | Dịch vụ tin cậy, chi phí hiệu quả để gửi thông báo. |
| **Log Aggregator** | AWS CloudWatch Logs hoặc ELK Stack | CloudWatch tích hợp sẵn, dễ sử dụng. ELK mạnh mẽ hơn cho việc tìm kiếm và phân tích log phức tạp. |
| **Metrics & Dashboard** | Prometheus + Grafana | Bộ đôi mã nguồn mở mạnh mẽ, linh hoạt. Có thể dùng dịch vụ được quản lý như **Amazon Managed Service for Prometheus/Grafana**. |

1. **Các bước cần thực hiện để triển khai hệ thống**

Quá trình này được chia thành các giai đoạn, từ thiết lập môi trường local cho đến khi triển khai hoàn chỉnh lên Production.

**Giai đoạn 1: Thiết lập môi trường phát triển Local (Containerization)**

**Mục tiêu**: Tạo một môi trường phát triển mô phỏng gần nhất với Production, cho phép lập trình viên làm việc hiệu quả.

1. **Viết Dockerfile**:
   * Tạo Dockerfile cho **API Server (Node.js)**: Định nghĩa các bước để build ứng dụng Node.js, cài đặt dependencies, và khởi chạy server.
   * Tạo Dockerfile cho **Frontend Server (Next.js)**: Định nghĩa các bước để build ứng dụng Next.js.
2. **Tạo Docker Compose**:
   * Tạo file [docker-compose.yml](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) ở thư mục gốc của dự án.
   * Định nghĩa các services cần thiết:
     + api-server: Build từ Dockerfile của Node.js.
     + frontend-server: Build từ Dockerfile của Next.js.
     + postgres-db: Sử dụng image postgres:latest chính thức. Định nghĩa volumes để lưu trữ dữ liệu DB ngay cả khi container bị xóa.
     + prometheus: Sử dụng image prom/prometheus.
     + grafana: Sử dụng image grafana/grafana.
   * Cấu hình network để các container có thể giao tiếp với nhau.
3. **Chạy môi trường Local**: Lập trình viên chỉ cần chạy lệnh docker-compose up để khởi tạo toàn bộ hệ thống trên máy của mình.

**Giai đoạn 2: Cung cấp hạ tầng trên Cloud (Infrastructure as Code)**

**Mục tiêu**: Sử dụng Terraform để tự động tạo tất cả tài nguyên hạ tầng cần thiết trên AWS.

1. **Thiết lập mạng (VPC)**: Viết code Terraform để tạo một Virtual Private Cloud (VPC) với các public và private subnets, Internet Gateway, và NAT Gateway để cách ly mạng.
2. **Tạo Database Server**: Viết code Terraform để tạo một instance **AWS RDS for PostgreSQL**. Cấu hình các thông số như kích thước, username, password (lưu an toàn trong AWS Secrets Manager).
3. **Tạo Container Registry (ECR)**: Tạo một repository trên **AWS ECR** để lưu trữ các Docker image của ứng dụng.
4. **Tạo Kubernetes Cluster (EKS)**: Viết code Terraform để tạo một cluster **AWS EKS**, bao gồm các worker nodes để chạy container.
5. **Cấu hình Load Balancer & CDN**:
   * Cài đặt **AWS Load Balancer Controller** vào cluster EKS.
   * Viết code Terraform để tạo **AWS CloudFront**, trỏ đến Load Balancer sẽ được tạo sau này.

**Giai đoạn 3: Xây dựng CI/CD Pipeline**

**Mục tiêu**: Tự động hóa quá trình build, kiểm thử, và triển khai ứng dụng mỗi khi có thay đổi trong source code.

1. **Tạo Workflow trên GitHub Actions**: Tạo file .github/workflows/main.yml.
2. **Định nghĩa các bước trong Pipeline**:
   * **Trigger**: Pipeline sẽ được kích hoạt khi có push lên dev branch (triển khai lên Staging) hoặc khi tạo tag mới (triển khai lên Production).
   * **Build & Test**:
     + Checkout code.
     + Chạy npm install và npm test cho cả backend và frontend để đảm bảo chất lượng code.
   * **Build & Push Docker Image**:
     + Nếu test thành công, build Docker image cho API và Frontend.
     + Đăng nhập vào **AWS ECR**.
     + Đẩy (push) các image vừa build lên ECR với một tag duy nhất (ví dụ: Git commit hash).
   * **Deploy to Kubernetes**:
     + Sử dụng kubectl (đã được xác thực với cluster EKS) để áp dụng các file cấu hình Kubernetes (.k8s/deployment.yml, .k8s/service.yml, .k8s/ingress.yml).
     + File deployment.yml sẽ được cập nhật để trỏ đến Docker image tag mới nhất trên ECR.

**Giai đoạn 4: Triển khai lên môi trường Staging**

**Mục tiêu**: Triển khai phiên bản mới nhất của ứng dụng lên một môi trường giống hệt Production để kiểm thử cuối cùng.

1. **Merge code**: Lập trình viên merge feature branch vào dev branch.
2. **Tự động triển khai**: GitHub Actions pipeline được kích hoạt, tự động build và triển khai phiên bản mới lên **namespace staging** trong cluster EKS.
3. **Cấu hình môi trường**: Ứng dụng trên Staging sẽ sử dụng connection string đến **Staging DB** và các API key dành cho môi trường thử nghiệm (quản lý bằng Kubernetes Secrets).
4. **Kiểm thử**: Đội QA và các bên liên quan tiến hành kiểm thử toàn diện trên môi trường Staging.

**Giai đoạn 5: Triển khai lên môi trường Production (Promote)**

**Mục tiêu**: Đưa phiên bản đã được kiểm thử thành công ra phục vụ người dùng cuối.

1. **Tạo Release**: Sau khi Staging được duyệt, một người có thẩm quyền (release manager) sẽ tạo một Git tag mới (ví dụ: v1.0.0) trên main branch.
2. **Kích hoạt Production Pipeline**: Việc tạo tag sẽ kích hoạt workflow triển khai Production trên GitHub Actions.
3. **Triển khai an toàn**: Pipeline sẽ lấy **chính xác Docker image đã được kiểm thử trên Staging** và triển khai nó vào **namespace production** trong cluster EKS.
4. **Tích hợp dịch vụ**:
   * Cấu hình **AWS API Gateway** để trỏ đến service của API Server trong Kubernetes.
   * Cấu hình ứng dụng với các credentials của **AWS Cognito**, **SES**, **SNS** dành cho Production.
5. **Kích hoạt giám sát**:
   * Triển khai **Prometheus** và **Grafana** vào cluster (hoặc kết nối tới dịch vụ được quản lý).
   * Cấu hình ứng dụng để đẩy logs lên **CloudWatch Logs**.
   * Tạo các dashboard trên **Grafana** để theo dõi sức khỏe hệ thống và thiết lập cảnh báo (alerting) khi có sự cố.

**Mã nguồn triển khai**

**1. Containerization (Docker)**

Đây là các file để "đóng gói" ứng dụng của bạn thành các image có thể chạy ở bất cứ đâu.

**order-management/Dockerfile (Cho Backend)**

File này định nghĩa cách build image cho API server Node.js.

# Stage 1: Build the application

FROM node:18-alpine AS builder

WORKDIR /usr/src/app

# Copy package files and install dependencies

COPY package\*.json ./

RUN npm install

# Copy the rest of the source code

COPY . .

# Build TypeScript to JavaScript

RUN npm run build

# Stage 2: Production image

FROM node:18-alpine

WORKDIR /usr/src/app

# Copy only necessary files from the builder stage

COPY --from=builder /usr/src/app/package\*.json ./

COPY --from=builder /usr/src/app/dist ./dist

# Install only production dependencies

RUN npm install --omit=dev

# Expose the port the app runs on

EXPOSE 3001

# Command to run the application

CMD [ "node", "dist/index.js" ]

**frontend/Dockerfile (Cho Frontend)**

File này định nghĩa cách build image cho ứng dụng Next.js.

# Stage 1: Build the application

FROM node:18-alpine AS builder

WORKDIR /app

# Copy package files and install dependencies

COPY package\*.json pnpm-lock.yaml ./

RUN npm install -g pnpm && pnpm install

# Copy the rest of the source code

COPY . .

# Build the Next.js application

RUN pnpm run build

# Stage 2: Production image

FROM node:18-alpine

WORKDIR /app

# Copy necessary files from the builder stage

COPY --from=builder /app/public ./public

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

COPY --from=builder /app/node\_modules ./node\_modules

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

# Expose the port the app runs on

EXPOSE 3000

# Command to run the application

CMD ["npm", "start"]

[docker-compose.yml](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)**(Cho môi trường phát triển Local)**

File này giúp bạn khởi chạy toàn bộ hệ thống trên máy local chỉ bằng một lệnh.

version: '3.8'

services:

  # Backend API Server

  api-server:

    build:

      context: ./order-management

      dockerfile: Dockerfile

    ports:

      - "3001:3001"

    volumes:

      - ./order-management:/usr/src/app

      - /usr/src/app/node\_modules

    environment:

      - DATABASE\_URL=postgres://user:password@postgres-db:5432/orders\_db

    depends\_on:

      - postgres-db

  # Frontend Next.js Server

  frontend-server:

    build:

      context: ./frontend

      dockerfile: Dockerfile

    ports:

      - "3000:3000"

    volumes:

      - ./frontend:/app

      - /app/node\_modules

      - /app/.next

    environment:

      - NEXT\_PUBLIC\_API\_URL=http://api-server:3001

  # PostgreSQL Database

  postgres-db:

    image: postgres:15-alpine

    ports:

      - "5432:5432"

    environment:

      - POSTGRES\_USER=user

      - POSTGRES\_PASSWORD=password

      - POSTGRES\_DB=orders\_db

    volumes:

      - postgres\_data:/var/lib/postgresql/data

volumes:

  postgres\_data:

**2. CI/CD (GitHub Actions)**

File này tự động hóa quá trình build, test và deploy.

**.github/workflows/main.yml**

name: CI/CD Pipeline

on:

  push:

    branches:

      - main # Trigger khi push lên branch main

jobs:

  build-and-deploy:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v3

      - name: Configure AWS Credentials

        uses: aws-actions/configure-aws-credentials@v2

        with:

          aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

          aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

          aws-region: us-east-1 # Thay đổi region của bạn

      - name: Login to Amazon ECR

        id: login-ecr

        uses: aws-actions/amazon-ecr-login@v1

      - name: Build, tag, and push Backend image to Amazon ECR

        env:

          ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

          ECR\_REPOSITORY: order-management-backend

          IMAGE\_TAG: ${{ github.sha }}

        run: |

          docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG -f ./order-management/Dockerfile ./order-management

          docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

      - name: Build, tag, and push Frontend image to Amazon ECR

        env:

          ECR\_REGISTRY: ${{ steps.login-ecr.outputs.registry }}

          ECR\_REPOSITORY: order-management-frontend

          IMAGE\_TAG: ${{ github.sha }}

        run: |

          docker build -t $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG -f ./frontend/Dockerfile ./frontend

          docker push $ECR\_REGISTRY/$ECR\_REPOSITORY:$IMAGE\_TAG

      # Thêm các bước để deploy lên Kubernetes (EKS) hoặc dịch vụ khác ở đây

      # Ví dụ:

      # - name: Deploy to Amazon EKS

      #   run: |

      #     aws eks update-kubeconfig --name my-cluster --region us-east-1

      #     kubectl set image deployment/backend-deployment backend-container=$ECR\_REGISTRY/$ECR\_REPOSITORY\_BACKEND:$IMAGE\_TAG

      #     kubectl set image deployment/frontend-deployment frontend-container=$ECR\_REGISTRY/$ECR\_REPOSITORY\_FRONTEND:$IMAGE\_TAG

**3. Kubernetes Manifests**

Các file này định nghĩa cách ứng dụng của bạn sẽ chạy trên Kubernetes.

**.k8s/deployment.yml**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: backend-deployment

spec:

  replicas: 2 # Chạy 2 bản sao để đảm bảo tính sẵn sàng cao

  selector:

    matchLabels:

      app: backend

  template:

    metadata:

      labels:

        app: backend

    spec:

      containers:

      - name: backend-container

        image: <your-aws-account-id>.dkr.ecr.us-east-1.amazonaws.com/order-management-backend:latest # Sẽ được thay thế bởi CI/CD

        ports:

        - containerPort: 3001

        env:

        - name: DATABASE\_URL

          valueFrom:

            secretKeyRef:

              name: app-secrets

              key: DATABASE\_URL

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: frontend-deployment

spec:

  replicas: 2

  selector:

    matchLabels:

      app: frontend

  template:

    metadata:

      labels:

        app: frontend

    spec:

      containers:

      - name: frontend-container

        image: <your-aws-account-id>.dkr.ecr.us-east-1.amazonaws.com/order-management-frontend:latest # Sẽ được thay thế bởi CI/CD

        ports:

        - containerPort: 3000

**.k8s/service.yml**

apiVersion: v1

kind: Service

metadata:

  name: backend-service

spec:

  selector:

    app: backend

  ports:

    - protocol: TCP

      port: 80

      targetPort: 3001

  type: NodePort # Hoặc ClusterIP nếu chỉ giao tiếp nội bộ

---

apiVersion: v1

kind: Service

metadata:

  name: frontend-service

spec:

  selector:

    app: frontend

  ports:

    - protocol: TCP

      port: 80

      targetPort: 3000

  type: NodePort

**.k8s/ingress.yml**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: main-ingress

  annotations:

    kubernetes.io/ingress.class: "alb" # Sử dụng AWS Application Load Balancer

    alb.ingress.kubernetes.io/scheme: internet-facing

    alb.ingress.kubernetes.io/target-type: ip

spec:

  rules:

    - http:

        paths:

          - path: /api/

            pathType: Prefix

            backend:

              service:

                name: backend-service

                port:

                  number: 80

          - path: /

            pathType: Prefix

            backend:

              service:

                name: frontend-service

                port:

                  number: 80