Vẽ và giải thích góc nhìn triển khai của Event-Driven Architecture được đề xuất trong bài lab 04? 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.

1. Góc nhìn Triển khai (Deployment View) của Event-Driven Architecture

1.1 Tổng quan Deployment Architecture

Góc nhìn triển khai mô tả cách các thành phần phần mềm được phân phối và triển khai trên hạ tầng vật lý/ảo hoá, bao gồm servers, containers, networks và dependencies.

Production Deployment Architecture:

```
graph TB
    subgraph "Frontend Layer"
        WEB[React App
Load Balanced]
    end

    subgraph "API Gateway"
        LB[Nginx Load Balancer
:80, :443]
    end

    subgraph "Microservices"
        US[User Service
:3001]
        QS[Quiz Service
:3000]
        AS[Answer Service
:3002]
        SS[Scoring Service
:3003]
    end

    subgraph "Event Streaming"
        KAFKA[Kafka Cluster
:9092]
        ZK[Zookeeper
:2181]
    end

    subgraph "Data Storage"
        PG[(PostgreSQL
:5432)]
        REDIS[(Redis
:6379)]
    end

    subgraph "Monitoring"
        MON[Kafdrop & pgAdmin]
```

```
    end

    %% Main flow
    WEB --> LB
    LB --> US
    LB --> QS
    LB --> AS
    LB --> SS

    %% Event connections
    US -.->|events| KAFKA
    QS -.->|events| KAFKA
    AS -.->|events| KAFKA
    SS -.->|events| KAFKA

    %% Data connections
    US --> PG
    QS --> PG
    QS --> REDIS
    AS --> PG
    SS --> PG

    %% Infrastructure
    KAFKA --> ZK
    MON --> KAFKA
    MON --> PG

    classDef frontend fill:#e8f5e8,stroke:#2e7d32,stroke-width:2px
    classDef service fill:#e1f5fe,stroke:#01579b,stroke-width:2px
    classDef data fill:#f3e5f5,stroke:#4a148c,stroke-width:2px
    classDef event fill:#fff3e0,stroke:#e65100,stroke-width:2px
    classDef infra fill:#f5f5f5,stroke:#424242,stroke-width:2px

    class WEB frontend
    class LB infra
    class US,QS,AS,SS service
    class PG,REDIS data
    class KAFKA,ZK event
    class MON infra
```

1.2 Container Deployment với Docker Swarm

Docker Swarm Stack Architecture:

```
graph TB
    subgraph "Docker Swarm Cluster"
        subgraph "Manager Nodes"
            M1[Manager Node 1
quiz-app-manager-1]
            M2[Manager Node 2
quiz-app-manager-2]
```

```
            M3[Manager Node 3
quiz-app-manager-3]
        end

        subgraph "Worker Nodes"
            W1[Worker Node 1
quiz-app-worker-1]
            W2[Worker Node 2
quiz-app-worker-2]
            W3[Worker Node 3
quiz-app-worker-3]
            W4[Worker Node 4
quiz-app-worker-4]
        end
    end

    subgraph "Service Distribution"
        subgraph "Frontend Services"
            FS[frontend-service
replicas: 2
ports: 3000:3000]
        end

        subgraph "Backend Services"
            USS[user-service
replicas: 2
ports: 3001:3001]
            QSS[quiz-service
replicas: 3
ports: 3000:3000]
            ASS[answer-service
replicas: 2
ports: 3002:3002]
            SSS[scoring-service
replicas: 2
ports: 3003:3003]
        end

        subgraph "Infrastructure Services"
            PGSVC[postgres-service
replicas: 1
ports: 5432:5432]
            RDSVC[redis-service
replicas: 1
ports: 6379:6379]
            KFSVC[kafka-service
replicas: 3
ports: 9092:9092]
            ZKSVC[zookeeper-service
replicas: 3
ports: 2181:2181]
        end

        subgraph "Proxy Services"
```

```
            NGSVC[nginx-service
replicas: 2
ports: 80:80,443:443]
        end
    end

    %% Node assignments
    M1 --> NGSVC
    M2 --> PGSVC
    M3 --> KFSVC

    W1 --> FS
    W1 --> USS
    W2 --> QSS
    W2 --> ASS
    W3 --> SSS
    W3 --> RDSVC
    W4 --> ZKSVC

    classDef manager fill:#ffebee,stroke:#c62828,stroke-width:2px
    classDef worker fill:#e8f5e8,stroke:#2e7d32,stroke-width:2px
    classDef service fill:#e1f5fe,stroke:#01579b,stroke-width:2px

    class M1,M2,M3 manager
    class W1,W2,W3,W4 worker
    class FS,USS,QSS,ASS,SSS,PGSVC,RDSVC,KFSVC,ZKSVC,NGSVC service
```

1.3 Kubernetes Deployment Architecture

Kubernetes Cluster Layout:

```
graph TB
    subgraph "Kubernetes Cluster"
        subgraph "Control Plane"
            API[API Server]
            ETCD[etcd]
            SCHED[Scheduler]
            CTRL[Controller Manager]
        end

        subgraph "quiz-app Namespace"
            subgraph "Frontend Deployment"
                FPOD1[frontend-pod-1]
                FPOD2[frontend-pod-2]
                FSVC[frontend-service
LoadBalancer]
            end

            subgraph "Backend Deployments"
                UPOD1[user-pod-1]
                UPOD2[user-pod-2]
```

```
                USVC[user-service
ClusterIP]

                QPOD1[quiz-pod-1]
                QPOD2[quiz-pod-2]
                QPOD3[quiz-pod-3]
                QSVC[quiz-service
ClusterIP]

                APOD1[answer-pod-1]
                APOD2[answer-pod-2]
                ASVC[answer-service
ClusterIP]

                SPOD1[scoring-pod-1]
                SPOD2[scoring-pod-2]
                SSVC[scoring-service
ClusterIP]
            end

            subgraph "Data StatefulSets"
                PGSTS[postgres-statefulset
replicas: 2]
                RDSTS[redis-statefulset
replicas: 2]
                KFSTS[kafka-statefulset
replicas: 3]
                ZKSTS[zookeeper-statefulset
replicas: 3]
            end

            subgraph "Storage"
                PV1[postgres-pv-1]
                PV2[postgres-pv-2]
                RV1[redis-pv-1]
                KV1[kafka-pv-1]
                KV2[kafka-pv-2]
                KV3[kafka-pv-3]
            end

            subgraph "Configuration"
                CM[ConfigMaps]
                SEC[Secrets]
            end
        end

        subgraph "Ingress"
            ING[nginx-ingress
*.quiz-app.com]
        end
    end

    %% Service connections
    FSVC --> FPOD1
```

```
    FSVC --> FPOD2
    USVC --> UPOD1
    USVC --> UPOD2
    QSVC --> QPOD1
    QSVC --> QPOD2
    QSVC --> QPOD3
    ASVC --> APOD1
    ASVC --> APOD2
    SSVC --> SPOD1
    SSVC --> SPOD2

    %% Storage bindings
    PGSTS --> PV1
    PGSTS --> PV2
    RDSTS --> RV1
    KFSTS --> KV1
    KFSTS --> KV2
    KFSTS --> KV3

    %% Configuration
    UPOD1 --> CM
    UPOD1 --> SEC
    QPOD1 --> CM
    QPOD1 --> SEC

    %% Ingress routing
    ING --> FSVC
    ING --> USVC
    ING --> QSVC
    ING --> ASVC
    ING --> SSVC

    classDef control fill:#ffebee,stroke:#c62828,stroke-width:2px
    classDef pod fill:#e1f5fe,stroke:#01579b,stroke-width:2px
    classDef storage fill:#f3e5f5,stroke:#4a148c,stroke-width:2px
    classDef config fill:#fff3e0,stroke:#e65100,stroke-width:2px

    class API,ETCD,SCHED,CTRL control
    class
FPOD1,FPOD2,UPOD1,UPOD2,QPOD1,QPOD2,QPOD3,APOD1,APOD2,SPOD1,SPOD2 pod
    class PGSTS,RDSTS,KFSTS,ZKSTS,PV1,PV2,RV1,KV1,KV2,KV3 storage
    class CM,SEC,ING config
```

2. Công cụ Triển khai

2.1 Container Orchestration Tools

A. Docker & Docker Compose

- **Docker Engine**: Container runtime
- **Docker Compose**: Multi-container management
- **Usage**: Local development, testing

B. Docker Swarm

- **Docker Swarm Mode**: Simple clustering
- **Benefits**: Easy setup, built-in load balancing
- **Usage**: Small to medium production deployments

C. Kubernetes

- **Kubernetes**: Enterprise orchestration
- **Helm**: Package management
- **kubectl**: CLI management
- **Usage**: Large scale production deployments

2.2 Infrastructure as Code

Terraform Example:

```
resource "aws_ecs_cluster" "quiz_app" {
  name = "quiz-app-cluster"
}

resource "aws_ecs_service" "quiz_service" {
  name            = "quiz-service"
  cluster         = aws_ecs_cluster.quiz_app.id
  desired_count   = 3
}
```

2.3 CI/CD Pipeline

Basic GitLab CI:

```
stages:
  - build
  - deploy

build:
  script:
    - docker build -t quiz-service .
    - docker push quiz-service

deploy:
  script:
    - kubectl apply -f k8s/
```

3. Các bước Triển khai Hệ thống

3.1 Development Environment Setup

Quick Setup Commands:

```
# 1. Clone và setup
git clone https://github.com/your-org/quiz-app-event-driven.git
cd quiz-app-event-driven

# 2. Setup environment
cp .env.example .env

# 3. Start infrastructure
docker-compose up -d postgres redis kafka

# 4. Start services
docker-compose up -d

# 5. Verify
curl http://localhost:3000/health
```

Basic Docker Compose:

```
# docker-compose.yml
version: '3.8'
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: quiz_app
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

  kafka:
    image: confluentinc/cp-kafka:7.0.1
    ports:
      - "9092:9092"
    environment:
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181

  user-service:
    build: ./services/user-service
    ports:
      - "3001:3001"
    environment:
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_app
```

```
  quiz-service:
    build: ./services/quiz-service
    ports:
      - "3000:3000"
    environment:
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_app
      KAFKA_BROKERS: kafka:9092

  frontend:
    build: ./frontend
    ports:
      - "80:80"
```

3.2 Staging Environment Deployment

Docker Swarm Setup:

```
# 1. Initialize Swarm
docker swarm init

# 2. Deploy services
docker stack deploy -c docker-stack.yml quiz-app

# 3. Verify
docker service ls
```

Basic Stack Configuration:

```
# docker-stack.yml
version: '3.8'
services:
  quiz-service:
    image: quiz-app/quiz-service:latest
    environment:
      NODE_ENV: staging
      DATABASE_URL: postgresql://postgres:password@postgres:5432/quiz_app
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: quiz_app
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
```

```
    deploy:
      replicas: 1
```

3.3 Production Kubernetes Deployment

Basic Kubernetes Setup:

```
# 1. Create namespace
kubectl create namespace quiz-app

# 2. Deploy services
kubectl apply -f k8s/

# 3. Verify
kubectl get pods -n quiz-app
```

Basic Kubernetes Manifest:

```
# k8s/quiz-service.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quiz-service
  namespace: quiz-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: quiz-service
  template:
    metadata:
      labels:
        app: quiz-service
    spec:
      containers:
      - name: quiz-service
        image: quiz-app/quiz-service:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          value: "postgresql://postgres:password@postgres:5432/quiz_app"
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: quiz-service
  namespace: quiz-app
spec:
  selector:
    app: quiz-service
  ports:
  - port: 3000
    targetPort: 3000
```

Helm Deployment:

```
# 1. Create basic values
cat > values.yml << EOF
replicaCount: 3
image:
  repository: quiz-app/quiz-service
  tag: latest
EOF

# 2. Deploy
helm upgrade --install quiz-app ./chart \
  --values values.yml \
  --namespace quiz-app
```

3.4 Monitoring & Observability Setup

Basic Monitoring Setup:

```
# monitoring/prometheus.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
    - job_name: 'quiz-services'
      static_configs:
      - targets: ['quiz-service:3000', 'user-service:3001']
```

Simple Health Checks:

```yaml
# monitoring/alerts.yml
groups:
- name: quiz-app-alerts
  rules:
  - alert: ServiceDown
    expr: up == 0
    for: 1m
    annotations:
      summary: "Service is down"

  - alert: HighErrorRate
    expr: rate(http_requests_total{status=~"5.."}[5m]) > 0.1
    for: 2m
    annotations:
      summary: "High error rate detected"
```

Kiến trúc triển khai Event-driven này đảm bảo **high availability**, **scalability**, và **maintainability** cho Quiz App system.