🚀 CI/CD Deep Dive - Giải Thích Cặn Kẽ

📖 1. CI/CD Là Gì?

🔄 CI (Continuous Integration) - Tích hợp liên tục

- **Mục đích**: Tự động kiểm tra code mỗi khi có thay đổi
- **Quy trình**: Code Push → Test → Build → Báo cáo
- **Lợi ích**: Phát hiện lỗi sớm, đảm bảo code quality

📦 CD (Continuous Deployment) - Triển khai liên tục

- **Mục đích**: Tự động deploy code đã test lên production
- **Quy trình**: CI Pass → Build Images → Deploy Staging → Deploy Production
- **Lợi ích**: Giảm thời gian release, ít lỗi production

🏗️ 2. GitHub Actions Architecture

📁 Cấu Trúc Thư Mục

```
.github/
└── workflows/
    ├── ci.yml        # CI Pipeline
    ├── cd.yml        # CD Pipeline
    └── security.yml  # Security Pipeline (optional)
```

⚙️ GitHub Actions Components

1. Workflow (File YAML)

- Định nghĩa toàn bộ pipeline
- Trigger conditions (khi nào chạy)
- Jobs và steps

2. Jobs (Công việc)

- Chạy trên một máy ảo (runner)
- Có thể chạy song song hoặc tuần tự
- Mỗi job có environment riêng

3. Steps (Bước)

- Từng hành động cụ thể trong job
- Có thể là command hoặc action có sẵn

4. Actions (Hành động có sẵn)

- Code tái sử dụng từ community
- VD: `actions/checkout@v4`, `docker/build-push-action@v5`

🔄 3. CI Pipeline - Code Analysis

Hãy xem file `.github/workflows/ci.yml`:

```
name: Simple CI Pipeline

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
```

🎯 Trigger Events

- **`on.push.branches`**: Chạy khi push code lên branch main
- **`on.pull_request`**: Chạy khi tạo PR vào main
- **`workflow_dispatch`**: Cho phép chạy manual

💼 Jobs Architecture

```
jobs:
  code-check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Check Project Structure
        run: |
          echo "🔍 Checking project structure..."
          ls -la
          echo "✅ Project structure verified"
```

Giải thích:

- **`runs-on: ubuntu-latest`**: Sử dụng máy ảo Ubuntu
- **`actions/checkout@v4`**: Download source code về runner
- **`run`**: Chạy shell commands

🧪 Test Job với Database

```
  test-user-service:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:13
```

```
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: testdb
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          - 5432:5432
```

Giải thích Services:

- `services`: Khởi động containers phụ trợ
- `postgres`: Database container cho testing
- `health-cmd`: Command kiểm tra database sẵn sàng
- `ports`: Map port 5432 để service connect

🔧 Node.js Setup & Testing

```
    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'
          cache: 'npm'
          cache-dependency-path: user-service/package-lock.json

      - name: Install Dependencies
        working-directory: user-service
        run: npm ci

      - name: Run Tests
        working-directory: user-service
        run: npm test
        env:
          DATABASE_URL:
postgresql://postgres:postgres@localhost:5432/testdb
```

Giải thích Testing:

- `actions/setup-node@v4`: Cài đặt Node.js version 18
- `cache: 'npm'`: Cache node_modules để build nhanh hơn
- `npm ci`: Cài đặt dependencies từ package-lock.json (faster than npm install)
- `working-directory`: Thay đổi thư mục làm việc
- `env.DATABASE_URL`: Biến môi trường cho database connection

🐋 Docker Build Check

```
  build-check:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        service: [user-service, gateway]
    steps:
      - uses: actions/checkout@v4

      - name: Check Dockerfile
        run: |
          if [ -f ${{ matrix.service }}/Dockerfile ]; then
            echo "✅ Dockerfile found for ${{ matrix.service }}"
          else
            echo "❌ Dockerfile missing for ${{ matrix.service }}"
            exit 1
          fi

      - name: Simulate Docker Build
        run: |
          echo "🐋 Building Docker image for ${{ matrix.service }}..."
          echo "FROM node:18-alpine" > temp-dockerfile
          echo "✅ Build simulation completed"
```

Giải thích Matrix Strategy:

- `strategy.matrix`: Chạy job cho nhiều values
- `matrix.service`: Biến chứa [user-service, gateway]
- Job sẽ chạy 2 lần: 1 cho user-service, 1 cho gateway
- `${{ matrix.service }}`: Reference đến giá trị hiện tại

📦 4. CD Pipeline - Deployment Deep Dive

File `.github/workflows/cd.yml`:

🚀 Multi-Job Workflow

```
jobs:
  prepare-deploy:
    runs-on: ubuntu-latest
    outputs:
      services: ${{ steps.get-services.outputs.services }}
      should-deploy: ${{ steps.check-secrets.outputs.should-deploy }}
```

Job Outputs:

- `outputs`: Chia sẻ data giữa các jobs

- `steps.step-id.outputs.variable`: Reference đến output của step

🔍 Dynamic Service Detection

```
    steps:
      - name: Get Services to Deploy
        id: get-services
        run: |
          services='["user-service", "gateway"]'
          echo "services=$services" >> $GITHUB_OUTPUT
          echo "🚀 Services to deploy: $services"
```

GitHub Output Mechanism:

- `id: get-services`: Đặt ID cho step
- `echo "services=$services" >> $GITHUB_OUTPUT`: Set output variable
- `$GITHUB_OUTPUT`: Special file để set outputs

🐋 DockerHub Integration Logic

```
      - name: Check DockerHub Secrets
        id: check-secrets
        run: |
          if [[ -n "${{ secrets.DOCKERHUB_USERNAME }}" && -n "${{
secrets.DOCKERHUB_TOKEN }}" ]]; then
            echo "should-deploy=true" >> $GITHUB_OUTPUT
            echo "✅ DockerHub credentials found"
          else
            echo "should-deploy=false" >> $GITHUB_OUTPUT
            echo "⚠ DockerHub credentials missing - will skip push"
          fi
```

Secrets & Conditional Logic:

- `secrets.DOCKERHUB_USERNAME`: GitHub repository secret
- `-n "string"`: Check if string is not empty
- **Conditional deployment**: Chỉ push khi có credentials

🏗 Build and Push Job

```
  build-and-push:
    runs-on: ubuntu-latest
    needs: prepare-deploy
    strategy:
      matrix:
        service: ${{ fromJson(needs.prepare-deploy.outputs.services) }}
```

Job Dependencies:

- `needs: prepare-deploy`: Chờ job prepare-deploy hoàn thành
- `fromJson()`: Convert string JSON thành array
- `needs.job-name.outputs.variable`: Access output từ job khác

🐳 Docker Build Process

```
    steps:
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Login to DockerHub
        if: needs.prepare-deploy.outputs.should-deploy == 'true'
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
```

Docker Actions:

- `docker/setup-buildx-action`: Setup advanced Docker builder
- `if: condition`: Conditional step execution
- `docker/login-action`: Authenticate với DockerHub

🏷️ Image Tagging Strategy

```
      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ secrets.DOCKERHUB_USERNAME }}/${{ matrix.service }}
          tags: |
            type=ref,event=branch
            type=sha,prefix={{branch}}-
            type=raw,value=latest
```

Metadata Action:

- `type=ref,event=branch`: Tag với branch name (VD: main)
- `type=sha`: Tag với Git SHA (VD: main-abc1234)
- `type=raw,value=latest`: Tag latest cố định

🚀 Multi-Platform Build

```
      - name: Build and push Docker image
        uses: docker/build-push-action@v5
```

```
        with:
          context: ./${{ matrix.service }}
          file: ./${{ matrix.service }}/Dockerfile
          push: ${{ needs.prepare-deploy.outputs.should-deploy == 'true'
}}
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          platforms: linux/amd64,linux/arm64
```

Build Configuration:

- `context`: Build context directory
- `file`: Path tới Dockerfile
- `push`: Conditional push based on credentials
- `platforms`: Multi-architecture builds (Intel + ARM)

🌐 Environment-Based Deployment

```
  deploy-staging:
    runs-on: ubuntu-latest
    needs: [prepare-deploy, build-and-push]
    environment: staging
```

GitHub Environments:

- `environment: staging`: Deploy vào environment staging
- Environment có thể có protection rules
- Có thể require manual approval

🎯 Production Deployment

```
  deploy-production:
    runs-on: ubuntu-latest
    needs: [prepare-deploy, build-and-push, deploy-staging]
    environment: production
    steps:
      - name: Deploy to Production
        run: |
          if [[ "${{ needs.prepare-deploy.outputs.should-deploy }}" ==
"true" ]]; then
            echo "✅ Using DockerHub images for production deployment"
            echo "docker pull ${{ secrets.DOCKERHUB_USERNAME }}/user-
service:latest"
            echo "docker-compose -f docker-compose.prod.yml up -d"
          else
            echo "⚠ Production deployment requires DockerHub setup"
            exit 1
          fi
```

Production Logic:

- **Sequential dependency**: Chờ staging deploy xong
- **Conditional deployment**: Require DockerHub cho production
- `exit 1`: Fail job nếu không có DockerHub

🔧 5. Code Structure Analysis

📁 Repository Structure

```
project/
├── .github/workflows/     # CI/CD definitions
├── user-service/
│   ├── Dockerfile         # Container definition
│   ├── package.json       # Dependencies & scripts
│   └── tests/            # Test files
├── gateway/
│   └── Dockerfile
└── docker-compose.yml    # Local development
```

📦 Package.json Scripts

```
{
  "scripts": {
    "test": "tap tests/**/*.test.js --coverage",
    "start": "node src/index.js",
    "dev": "nodemon src/index.js"
  }
}
```

CI Integration:

- `npm ci`: Cài đặt dependencies trong CI
- `npm test`: Chạy test suite
- `npm run build`: Build production code (nếu có)

🐳 Dockerfile Optimization

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./
RUN npm ci --only=production

COPY . .
```

```
EXPOSE 3000
CMD ["npm", "start"]
```

CI/CD Considerations:

- **Multi-stage builds**: Optimize image size
- **Cache layers**: Package.json copy trước source code
- **Production dependencies**: `--only=production`

🔄 6. Workflow Execution Flow

🎯 CI Trigger Flow

```
Developer Push Code
        ↓
GitHub detects push to main
        ↓
GitHub Actions starts CI workflow
        ↓
3 Jobs start parallel:
├── code-check (structure validation)
├── test-user-service (with PostgreSQL)
└── build-check (Dockerfile validation)
        ↓
All jobs complete → CI Success ✅
```

📦 CD Trigger Flow

```
CI Success
    ↓
CD Workflow starts
    ↓
prepare-deploy (check credentials)
    ↓
build-and-push (matrix: user-service, gateway)
├── Build Docker images
├── Push to DockerHub (if credentials)
    ↓
deploy-staging (automatic)
├── Use DockerHub images
├── Deploy to staging environment
    ↓
deploy-production (manual approval)
├── Require approval
├── Deploy to production
    ↓
deployment-summary
└── Report results
```

💡 7. Best Practices Implementation

🚀 Performance Optimization

1. **Caching Strategy**

```
- uses: actions/setup-node@v4
  with:
    cache: 'npm'
    cache-dependency-path: service/package-lock.json
```

2. **Parallel Jobs**

```
jobs:
  job1:    # Runs parallel
  job2:    # Runs parallel
  job3:
    needs: [job1, job2]  # Runs after job1 & job2
```

3. **Matrix Strategy**

```
strategy:
  matrix:
    service: [service1, service2]
    node-version: [16, 18]
# Creates 4 jobs: service1+node16, service1+node18, service2+node16,
service2+node18
```

🔒 Security Best Practices

1. **Secrets Management**

```
env:
  DATABASE_URL: ${{ secrets.DATABASE_URL }}
```

2. **Conditional Steps**

```
- name: Deploy to Production
  if: github.ref == 'refs/heads/main'
```

3. **Environment Protection**

```
environment: production  # Can require approvals
```

🎓 8. Key Learning Points

✅ CI Pipeline Ensures:

1. **Code Quality**: Tests pass before merge
2. **Build Validation**: Docker images can be built
3. **Fast Feedback**: Developers know immediately if something breaks

✅ CD Pipeline Provides:

1. **Automated Deployment**: No manual deployment steps
2. **Consistency**: Same process every time
3. **Rollback Capability**: Easy to revert if issues

✅ GitHub Actions Benefits:

1. **Integration**: Built into GitHub
2. **Marketplace**: Thousands of pre-built actions
3. **Free Tier**: 2000 minutes/month for public repos

🔧 9. Troubleshooting Common Issues

❌ CI Fails

```
# Check logs in GitHub Actions tab
# Common issues:
- Test failures → Fix tests
- Dependency issues → Update package.json
- Database connection → Check service config
```

❌ CD Fails

```
# DockerHub authentication
- Check DOCKERHUB_USERNAME secret
- Check DOCKERHUB_TOKEN validity
- Verify token permissions

# Build failures
- Check Dockerfile syntax
- Verify build context
```

❌ Environment Issues

```
# Production deployment
- Setup production environment in GitHub
- Add protection rules
- Configure required reviewers
```

🎉 Kết Luận

CI/CD với GitHub Actions giúp:

- ✅ **Tự động hóa** quy trình từ code → production
- ✅ **Đảm bảo chất lượng** với automated testing
- ✅ **Giảm rủi ro** với staging environment
- ✅ **Tăng tốc độ** development cycle

Quy trình hoàn chỉnh: Developer push code → CI tests → CD builds → Staging deploy → Production deploy với approval