📋 CI/CD Code Walkthrough - Giải Thích Từng Dòng

🔄 CI Pipeline Analysis (.github/workflows/ci.yml)

📌 1. Workflow Definition & Triggers

```yaml
name: Simple CI

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

Giải thích:

- `name`: Tên hiển thị trong GitHub Actions tab
- `on.push.branches`: Trigger khi push code vào branch main
- `on.pull_request.branches`: Trigger khi tạo PR vào main
- **Tại sao main?**: Đây là branch chính, cần test trước khi merge

📌 2. Job 1: Code Structure Check

```yaml
jobs:
  code-check:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '18'

    - name: Check project structure
      run: |
        echo "📁 Checking project structure..."
        ls -la
        echo "✅ Project structure looks good!"
```

Line-by-line Analysis:

1. `code-check:` - Job name, có thể tùy chỉnh
2. `runs-on: ubuntu-latest` - Chọn OS cho runner (Ubuntu mới nhất)
3. `uses: actions/checkout@v4` - Action có sẵn để download source code

4. `uses: actions/setup-node@v4` - Action để cài Node.js
5. `node-version: '18'` - Version Node.js cần dùng
6. `run: |` - Chạy multi-line bash commands
7. `echo` **commands** - In thông báo cho người xem logs
8. `ls -la` - List files để verify structure

Tại sao cần job này?

- Verify code được download đúng
- Đảm bảo environment setup OK
- Base foundation cho các jobs khác

📌 3. Job 2: Database Testing

```yaml
  test-user-service:
    runs-on: ubuntu-latest
    needs: code-check

    services:
      postgres:
        image: postgres:13
        env:
          POSTGRES_PASSWORD: admin
          POSTGRES_USER: postgres
          POSTGRES_DB: userdb
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          - 5432:5432
```

Service Container Deep Dive:

1. `needs: code-check` - Job dependency, chờ code-check hoàn thành
2. `services:` - Khởi động containers phụ trợ
3. `postgres:` - Service name (có thể đặt tên tùy ý)
4. `image: postgres:13` - Docker image từ Docker Hub
5. `env:` - Environment variables cho container
   - `POSTGRES_PASSWORD: admin` - Password cho postgres user
   - `POSTGRES_USER: postgres` - Username (default)
   - `POSTGRES_DB: userdb` - Database name được tạo
6. `options:` - Docker run options
   - `--health-cmd pg_isready` - Command kiểm tra DB ready
   - `--health-interval 10s` - Check mỗi 10 giây
   - `--health-timeout 5s` - Timeout cho mỗi check
   - `--health-retries 5` - Retry 5 lần nếu fail
7. `ports: - 5432:5432` - Map container port 5432 → host port 5432

Tại sao cần health check?

- PostgreSQL cần time để start up
- Health check đảm bảo DB ready trước khi chạy tests

📌 4. Test Execution Steps

```
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '18'

    - name: Install dependencies
      working-directory: user-service
      run: |
        echo "📦 Installing user-service dependencies..."
        npm install --no-audit --no-fund

    - name: Wait for PostgreSQL
      run: |
        echo "⏳ Waiting for PostgreSQL..."
        timeout 60 bash -c 'until pg_isready -h localhost -p 5432; do
sleep 1; done'
        echo "✅ PostgreSQL is ready!"

    - name: Run tests
      working-directory: user-service
      env:
        NODE_ENV: test
        DATABASE_URL: postgres://postgres:admin@localhost:5432/userdb
      run: |
        echo "🧪 Running user-service tests..."
        npm test || echo "⚠ Tests failed but continuing..."
```

Step-by-Step Breakdown:

1. **Checkout & Setup** - Giống job trước
2. `working-directory: user-service` - Thay đổi thư mục làm việc
3. `npm install --no-audit --no-fund` - Cài dependencies
   - `--no-audit` - Skip security audit (faster)
   - `--no-fund` - Skip funding messages
4. **Wait for PostgreSQL**:
   - `timeout 60` - Giới hạn 60 giây
   - `until pg_isready` - Loop cho đến khi DB ready
   - `sleep 1` - Đợi 1 giây giữa các lần check
5. **Environment Variables**:

   - `NODE_ENV: test` - Set environment to test
   - `DATABASE_URL` - Connection string tới PostgreSQL
6. `npm test || echo` - Chạy test, nếu fail thì chỉ warning

📌 5. Job 3: Matrix Build Strategy

```yaml
build-check:
  runs-on: ubuntu-latest
  needs: code-check
  strategy:
    matrix:
      service: [user-service, gateway]

  steps:
  - name: Checkout code
    uses: actions/checkout@v4

  - name: Check if Dockerfile exists
    working-directory: ${{ matrix.service }}
    run: |
      if [ -f Dockerfile ]; then
        echo "✅ Dockerfile found for ${{ matrix.service }}"
        echo "🐳 Would build Docker image here..."
      else
        echo "⚠ No Dockerfile for ${{ matrix.service }}"
      fi

  - name: Simulate build success
    run: |
      echo "✅ Build simulation completed for ${{ matrix.service }}"
```

Matrix Strategy Analysis:

1. **`strategy.matrix.service: [user-service, gateway]`**
   - Tạo 2 jobs parallel: 1 cho user-service, 1 cho gateway
   - Mỗi job có biến `${{ matrix.service }}` khác nhau
2. **`working-directory: ${{ matrix.service }}`**
   - Job 1: working-directory = user-service
   - Job 2: working-directory = gateway
3. **`if [ -f Dockerfile ]`** - Bash conditional
   - Check file Dockerfile có tồn tại không
   - `-f` flag kiểm tra file existence
4. **`${{ matrix.service }}`** - GitHub Actions expression
   - Được thay thế bằng giá trị từ matrix

   - Job 1: user-service, Job 2: gateway

📦 CD Pipeline Analysis (.github/workflows/cd.yml)

📌 1. CD Triggers & Outputs

```
name: Simple CD Pipeline

on:
  push:
    branches: [ main ]
  workflow_dispatch:

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

Advanced Concepts:

1. `workflow_dispatch:` - Cho phép manual trigger từ GitHub UI
2. `outputs:` - Job có thể export data cho jobs khác
3. `steps.step-id.outputs.variable` - Reference đến output của step cụ thể

📌 2. Dynamic Service Detection

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

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

Output Mechanism:

1. `id: get-services` - Đặt ID cho step (bắt buộc để reference)
2. `services='["user-service", "gateway"]'` - JSON array format
3. `echo "services=$services" >> $GITHUB_OUTPUT` - Set output variable
4. `$GITHUB_OUTPUT` - Special file GitHub Actions dùng để store outputs

Tại sao dùng JSON array?

- Có thể dùng `fromJson()` function để convert thành matrix

- Flexible, có thể thêm/bớt services dễ dàng

📌 3. Conditional Logic với Secrets

```
      - 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
```

Bash Conditional Breakdown:

1. `if [[ ... ]]; then` - Bash conditional syntax
2. `-n "string"` - Test nếu string không empty
3. `${{ secrets.DOCKERHUB_USERNAME }}` - GitHub repository secret
4. `&&` - AND operator, cả 2 conditions phải true
5. `echo "should-deploy=true"` - Set boolean output
6. `else` - Fallback nếu không có credentials

Tại sao cần check secrets?

- Pipeline có thể chạy mà không có DockerHub setup
- Graceful degradation: build local thay vì fail completely

📌 4. Matrix Build với Dynamic Data

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

Advanced Matrix Strategy:

1. `needs: prepare-deploy` - Job dependency
2. `fromJson()` - Convert JSON string thành array
3. `needs.job-name.outputs.variable` - Access output từ job khác
4. **Result**: Matrix với [user-service, gateway] từ prepare-deploy job

📌 5. Docker Build Process

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

      - 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 Deep Dive:

1. `docker/setup-buildx-action@v3`
    - Setup advanced Docker builder
    - Supports multi-platform builds
    - Better caching capabilities
2. `if: needs.prepare-deploy.outputs.should-deploy == 'true'`
    - Conditional step execution
    - Chỉ login nếu có credentials
3. `docker/login-action@v3`
    - Pre-built action để authenticate DockerHub
    - Handles token management securely

📌 6. Image Metadata & Tagging

```
      - 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
```

Tagging Strategy Explained:

1. `images:` - Base image name (username/service)
2. `type=ref,event=branch` - Tag = branch name (VD: main)
3. `type=sha,prefix={{branch}}-` - Tag = branch-gitsha (VD: main-abc1234)
4. `type=raw,value=latest` - Fixed tag "latest"

Result: 3 tags cho mỗi image

- `username/user-service:main`
- `username/user-service:main-abc1234`
- `username/user-service:latest`

📌 7. Multi-Platform Build & Push

```
      - 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:

1. **`context: ./${{ matrix.service }}`** - Build context directory
2. **`file: ./${{ matrix.service }}/Dockerfile`** - Path tới Dockerfile
3. **`push:`** - Conditional push based on credentials
4. **`tags:`** - Use tags từ metadata action
5. **`platforms: linux/amd64,linux/arm64`** - Multi-architecture builds

Tại sao multi-platform?

- amd64: Intel/AMD processors (production servers)
- arm64: Apple Silicon, ARM servers (cost-effective)

🎯 Workflow Execution Flow

🔄 CI Execution Sequence

```
1. Developer pushes code to main
   ↓
2. GitHub detects push event
   ↓
3. CI workflow triggered
   ↓
4. Job: code-check (runs first)
   - Checkout code
   - Setup Node.js
   - Verify project structure
   ↓
5. Jobs: test-user-service + build-check (parallel)
   test-user-service:                build-check:
```

```
   - Start PostgreSQL container      - Check Dockerfile exists
   - Wait for DB ready              - Simulate build
   - Install dependencies           (runs for user-service + gateway)
   - Run tests
   ↓
6. All jobs complete → CI Success ✅
```

📦 CD Execution Sequence

```
1. CI Success (hoặc manual trigger)
   ↓
2. CD workflow triggered
   ↓
3. Job: prepare-deploy
   - Detect services to deploy: ["user-service", "gateway"]
   - Check DockerHub credentials → should-deploy: true/false
   ↓
4. Job: build-and-push (matrix strategy)
   Matrix creates 2 parallel jobs:

   user-service job:                 gateway job:
   - Setup Docker Buildx            - Setup Docker Buildx
   - Login DockerHub (if creds)     - Login DockerHub (if creds)
   - Extract metadata               - Extract metadata
   - Build user-service image       - Build gateway image
   - Push to DockerHub (if creds)   - Push to DockerHub (if creds)
   ↓
5. Job: deploy-staging
   - Deploy to staging environment
   - Use DockerHub images if available
   ↓
6. Job: deploy-production
   - Requires manual approval
   - Deploy to production
   - Use DockerHub images
```

💡 Key Learning Points

✅ CI Pipeline Teaches:

1. **Sequential vs Parallel Jobs**: `needs` keyword controls dependencies
2. **Service Containers**: PostgreSQL runs alongside main job
3. **Matrix Strategy**: One job definition → multiple executions
4. **Working Directory**: Commands execute in specific folders
5. **Environment Variables**: Pass config to applications

✅ CD Pipeline Teaches:

1. **Job Outputs**: Share data between jobs

2. **Conditional Logic**: Different behavior based on secrets
3. **Dynamic Matrix**: Build matrix from runtime data
4. **Docker Integration**: Build, tag, and push images
5. **Multi-Platform Builds**: Support different architectures

✅ Advanced Concepts:

1. **GitHub Actions Expressions**: `${{ }}` syntax
2. **Secrets Management**: Secure credential storage
3. **Action Marketplace**: Reusable community actions
4. **Environment Protection**: Manual approvals for production

🚀 Practical Implementation Tips

🎯 For Learning:

1. **Start Simple**: 1 job, 1 step, basic commands
2. **Add Complexity**: Services, matrix, outputs gradually
3. **Use Logs**: Every step logs to understand execution
4. **Test Locally**: Docker commands work on local machine first

🎯 For Production:

1. **Security**: Never hardcode secrets, use GitHub Secrets
2. **Efficiency**: Cache dependencies, use appropriate runners
3. **Reliability**: Health checks, retries, timeouts
4. **Monitoring**: Notifications, badges, status checks

🎓 **Kết luận**: CI/CD là automation của quy trình manual. Hiểu quy trình manual trước, sau đó automate từng bước!