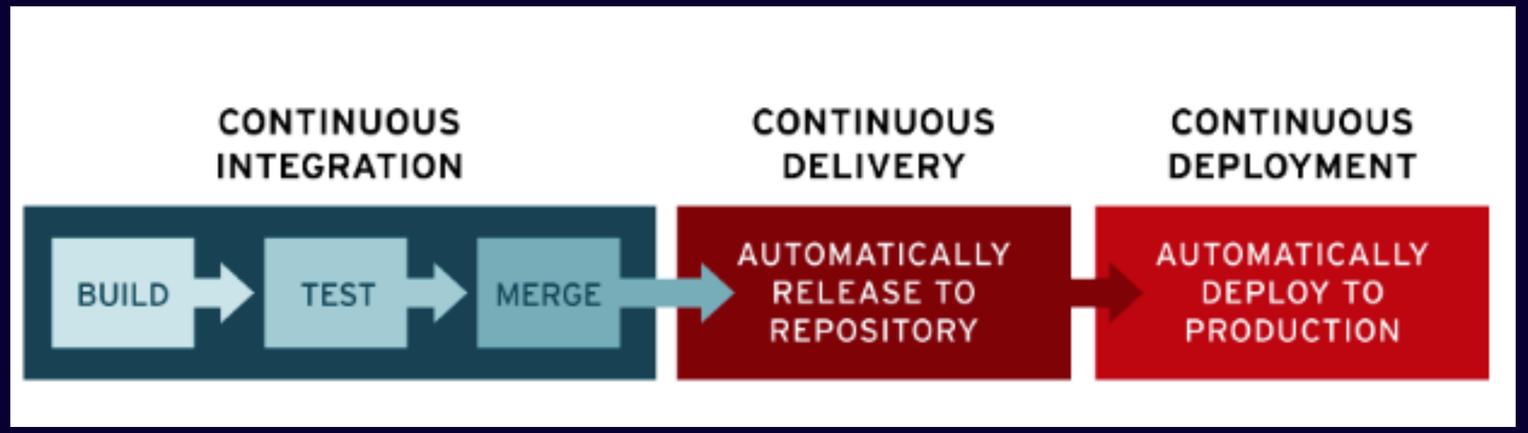
빅데이터를 지탱하는 기술 week7

CI/CD with Github Actions

2024.01.08

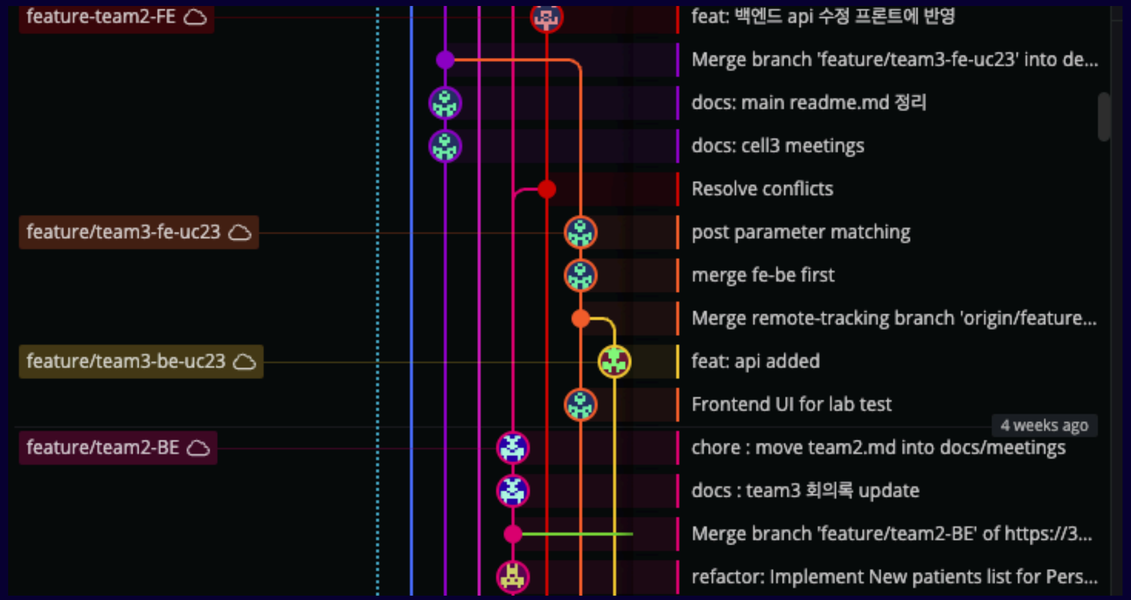
8기 엄소은

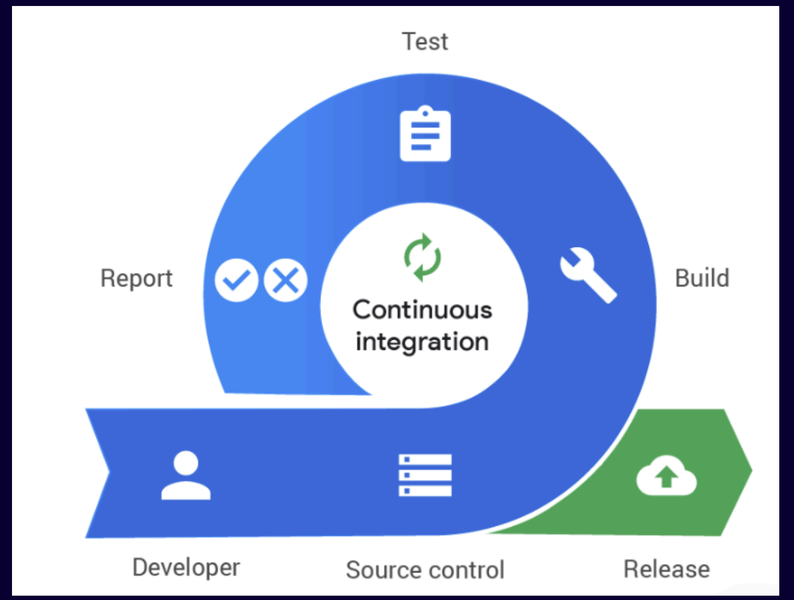
CI/CD 란 ?



- **CI (Continuous Integration)** : 어플리케이션의 새로운 코드 변경 사항이 정기적으로 빌드 및 테스트 되어 공유 레포지토리에 통합되는 것
- **CD (Continuous Delivery/Development)**
  - Delivery : 공유 레포지토리로 자동으로 release 되는 것
  - Deployment : 프로덕션 레벨까지 자동으로 배포 되는 것

Continuous Integration





- 다수의 개발자가 형상관리 툴 (Git) 툴을 공유하여 사용하는 환경 – 기능 추가 시 마다 커밋하고, 레포지토리 버전을 업데이트 한다.
- 이때 공유 레포에 수많은 커밋이 쌓이게 된다. 그럴 때마다 수동으로 빌드 / 테스트 / 머지 해야 한다면 ?
- 매번 머지 잘 되었는지, 날라간 커밋은 없는지, 테스트 코드는 잘 작동하는지 확인하는 것이 상당히 까다롭다.. ~~(본인 경험 🤯)~~
- **CI 는 자동화된 빌드 & 테스트를 제공**하여 소스코드의 충돌을 원천적으로 방어한다.

Continuous Delivery / Deployment

- Continuous Delivery:
  - CI의 빌드 자동화, 유닛 및 통합 테스트 수행 후, 이어지는 지속적 프로세스에서는 유효한 (=에러없는) 코드를 레포지토리에 자동으로 릴리스한다.
  - 코드 변경 사항 병합부터 프로덕션에 적합한 빌드 제공에 이르는 모든 단계에는 테스트 자동화와 코드 릴리스 자동화가 포함된다.
  - 이 프로세스를 완료하면 운영팀이 더욱 빠르고 쉽게 어플리케이션을 배포할 수 있다.
- Continuous Deployment:
  - 어플리케이션을 프로덕션으로 릴리스하는 작업을 자동화한다.
  - 개발자가 코드 작성 -〉 클라우드 어플리케이션 자동으로 실행

🤯 CI/CD 적용 전

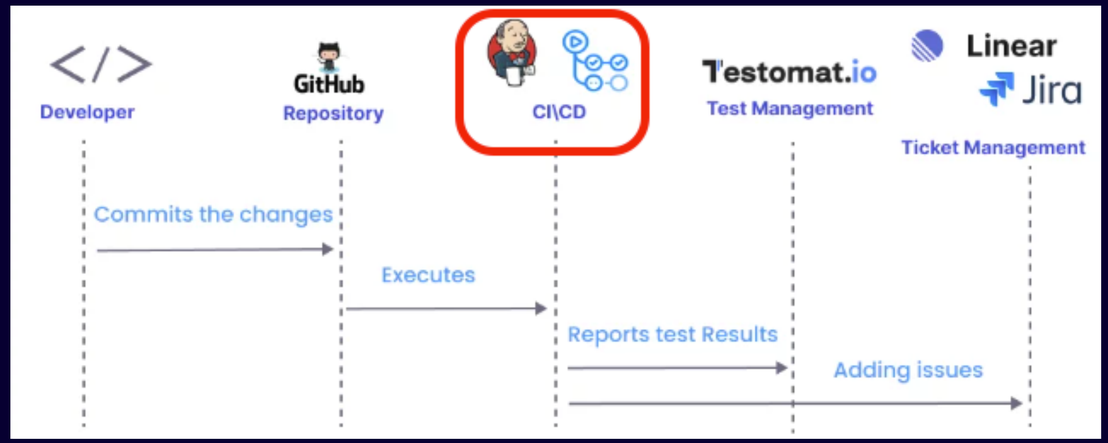
1. 개발자들이 개발하여 본인의 로컬에서 코드를 수정한다.
2. 각자의 feature 브랜치에 코드를 push 한다. (소스코드끼리 충돌해도 눈치채지 못한다 – ~~바극의 서막..~~)
3. 각자의 코드를 git 에 올리고 통합한다.
4. 에러가 발생했지만 어느 부분에서 에러가 났는지 모르므로 전체 커밋 히스토리를 뒤지고, 소스코드를 디버깅하면서 겨우겨우 코드를 수정한다.
5. 1~4 의 단계를 계속 반복한다.
6. 에러가 모두 해결 되었으면 배포한다.

🥰 CI/CD 적용 후

1. 개발자들이 개발하여 feature 브랜치에 코드를 push 한다.
2. git push 를 통해 trigger 되어 CI 서버에서 자동으로 build, test, lint 를 실행하고 결과를 전송한다.
3. 개발자들은 결과를 받고 에러가 난 부분이 있으면 에러를 수정하고 코드를 main 브랜치에 머지한다.
4. main 브랜치에 코드를 머지하고 build, test 가 정상적으로 수행이 되었다면 CI 서버에서 자동으로 배포까지 한다.

CI/CD Tools

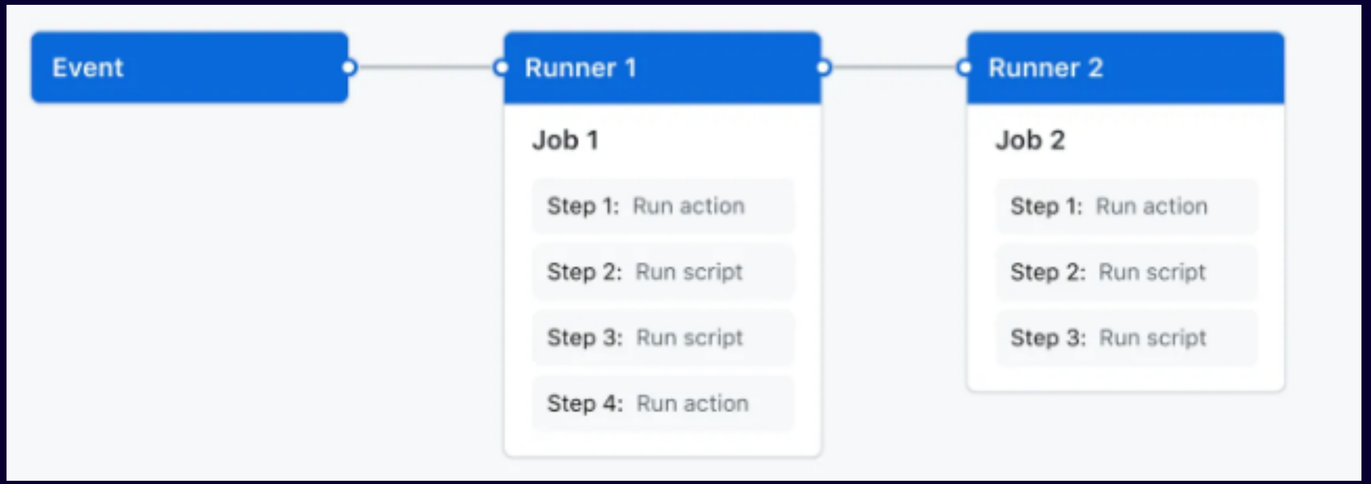
- Jenkins
- **Github Actions**
- CircleCI
- TravisCI



	Jenkins	GitHub Actions
Software model	Open source	Open source
Hosting	On-premise and cloud-based	On-premise and cloud-based
Supported OS	Linux/Windows/MacOS	Linux/Windows/MacOS
Ease of use and Setup	Medium	Easy to use
Installation	Required	Not required
Scalability	Highly scalable	Limited by GitHub's infrastructure
Control	Doesn't provide full control over CI\CD pipelines	Full control over CI\CD pipelines
Paid plan details	Free to use, requires a dedicated administrator	Both free and paid plans with varying features and usage limits.

What are GitHub Actions ?

- Understading GitHub Actions (GitHub Docs)
- GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production
- You can configure a GitHub Actions workflow to be triggered when an event occurs in your repository, such as a pull request being opened or an issue being created.



GitHub Actions 의 구성요소

- **Workflows**
  - A workflow is a configurable automated process that will run one or more jobs. Workflows are defined by a **YAML file** checked in to your repository and will run when **triggered by an event** in your repository, or they can be triggered manually, or at a defined schedule.
  - .github/workflows 에 정의
- **Events**
  - An event is a specific activity in a repository that **triggers a workflow run**.
  - e.g.) someone creates a pull request, opens an issue, or pushes a commit to a repository, work on a schedule, posting to a REST API …
- **Jobs**
  - A job is a set of steps in a workflow that is executed on the same runner.
  - Each step is either a shell script that will be executed, or an action that will be run.
  - Steps are executed in order and are dependent on each other. Since each step is executed on the same runner, you can share data from one step to another.

GitHub Actions 의 구성요소

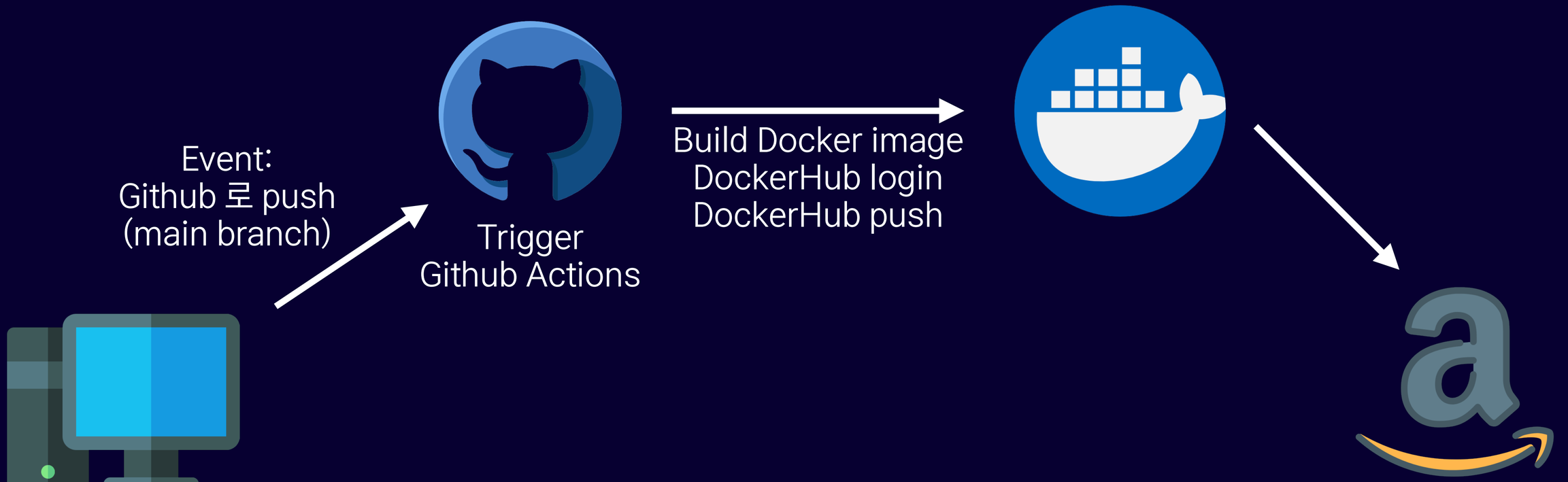
- **Actions**
  - An action is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task.
- **Runners**
  - A runner is a server that runs your workflows when they're triggered.
    (GitHub 에서 제공해주는 서버– Ubuntu Linux, Windows, MacOS)

Example Workflow

- Example workflow

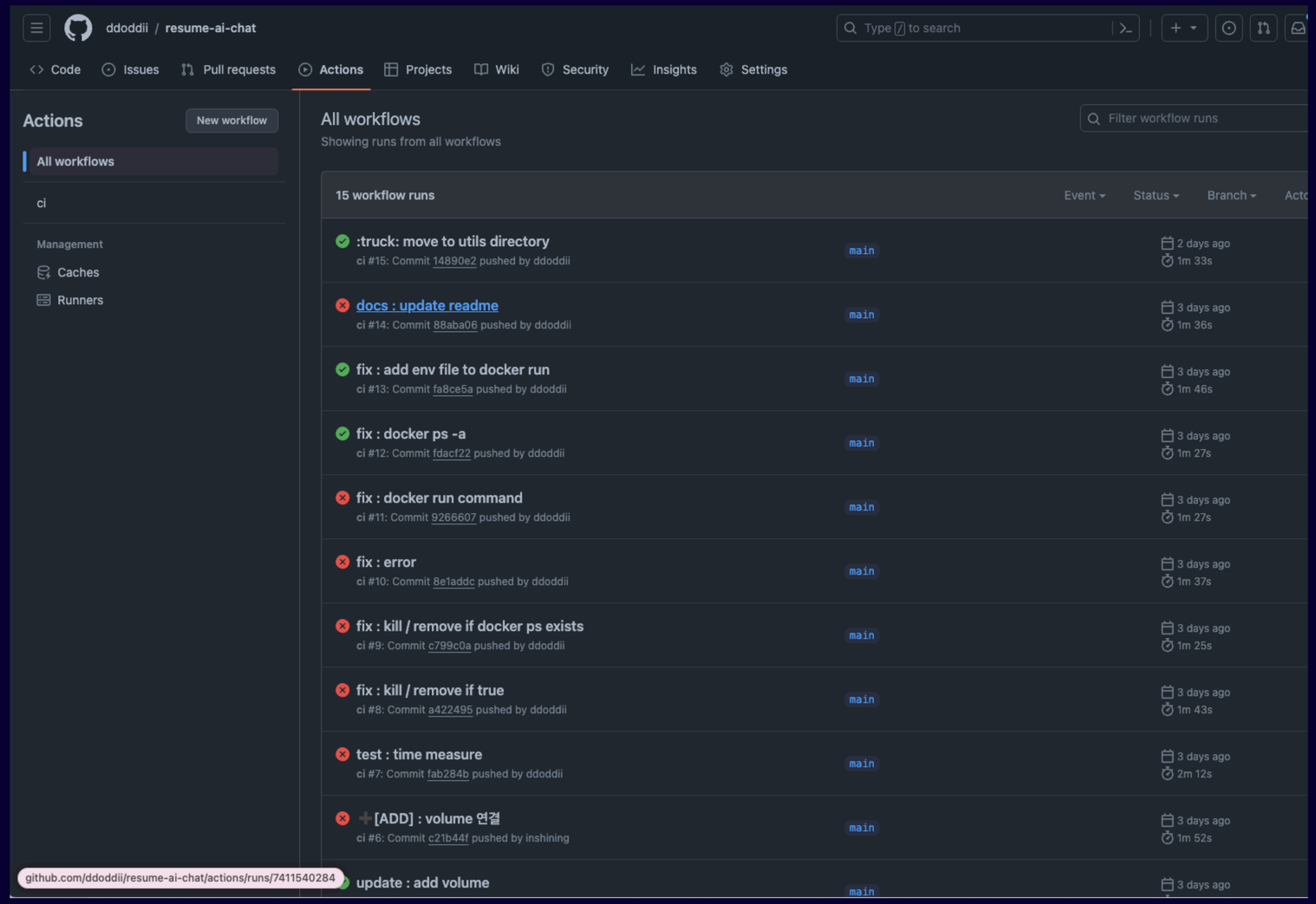
```yaml
name: ci

on:
  push:
    branches:
      - "main"

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      -
        name: Checkout
        uses: actions/checkout@v4
      -
        name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      -
        # Build and push to dockerhub
        name: Build and push
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./Dockerfile
          push: true
          tags: ${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.PROJECT_NAME }}:latest

      # EC2 인스턴스 접속 및 애플리케이션 실행
      -
        name: Application Run
        uses: appleboy/ssh-action@v0.1.6
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.EC2_KEY }}

          script: |
            sudo docker ps -a --filter "name=${{ secrets.PROJECT_NAME }}" | xargs -r docker stop
            sudo docker ps -a --filter "name=${{ secrets.PROJECT_NAME }}" | xargs -r docker kill
            sudo docker ps -a --filter "name=${{ secrets.PROJECT_NAME }}" | xargs -r docker rm -f
            sudo docker rmi ${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.PROJECT_NAME }}
            sudo docker pull ${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.PROJECT_NAME }}

            sudo docker run -p  ${{ secrets.PORT }}:${{ secrets.PORT }} \
            --env-file ${{ secrets.ENV }} \
            --name ${{ secrets.PROJECT_NAME }} \
            -v /home/ubuntu/resume_ai_chat.db:/app/resume_ai_chat.db \
            -d ${{ secrets.DOCKERHUB_USERNAME }}/${{ secrets.PROJECT_NAME }}
```

Example Workflow



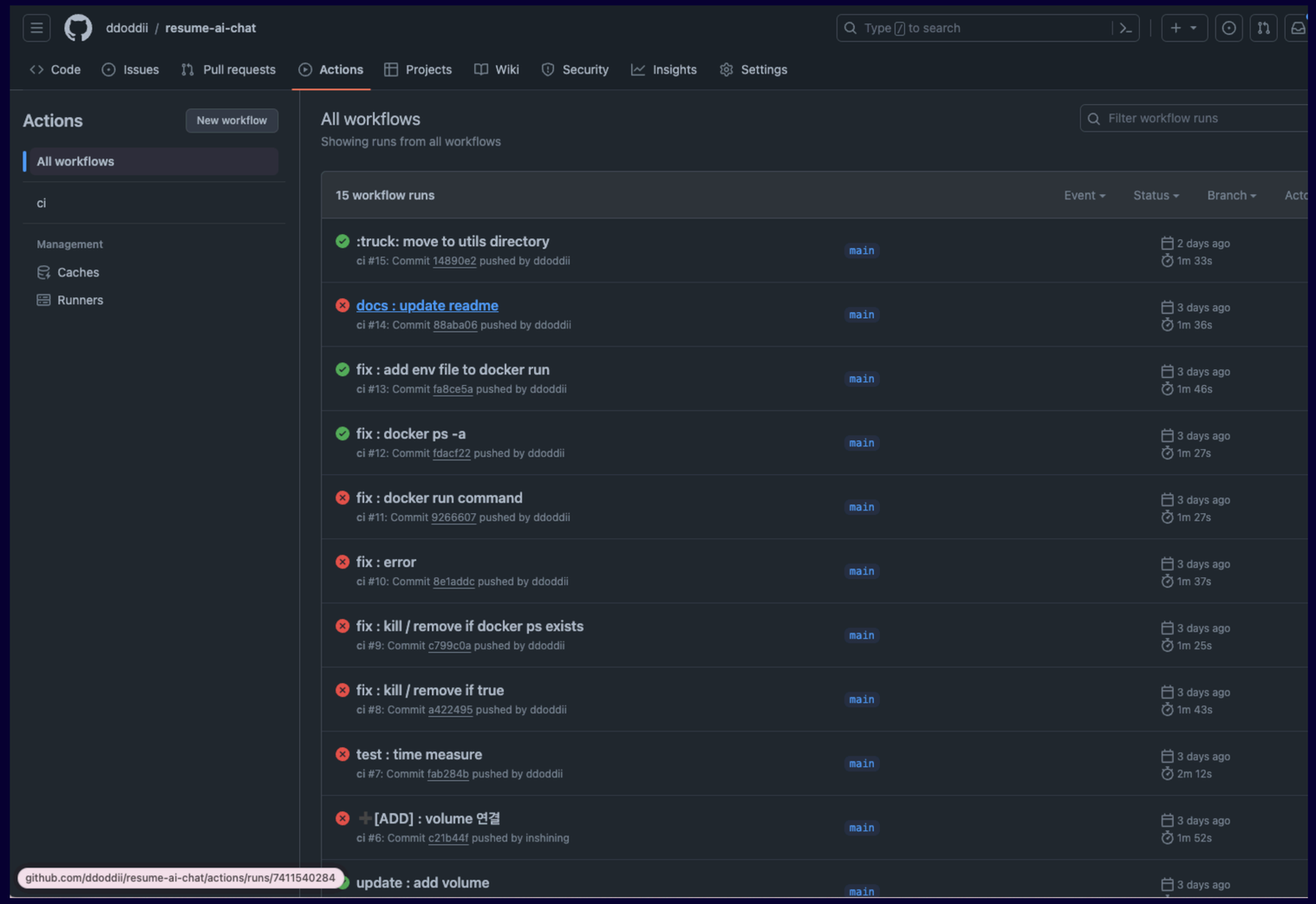
Example Workflow

- Results in GitHub Actions



Example Workflow

- Results in GitHub Actions



과제

- week6 코드 기반 GitHub actions workflow 만들어보기