How to use Docker, git

Ⅰ. Git Installation

- 협업을 위해 Git을 사용하려면 먼저 로컬 컴퓨터에 Git을 설치해야 합니다.

- Windows: Download Git for Windows (https://git-scm.com/downloads/win)
- macOS: Use Homebrew: 'brew install git'
- Linux: Use the package manager: 'sudo apt-get install git' or 'sudo yum install git'

Ⅱ. Initializing a Repository

- Git을 설치한 후, 프로젝트 디렉토리로 이동하여 Git을 초기화합니다.

- git init

Ⅲ. Basic Git Configuration

- 협업 전에 Git 사용자 이름과 이메일을 설정합니다.

- git config --global user.name "사용자 이름"
- git config --global user.email "youremail@example.com"

Ⅳ. SSH Connect

- ssh키를 생성해 github에 등록합니다. (SSH키 연결 확인법 : ssh -T git@github.com)

1. SSH Keygen

- git bash 열기
- 키 생성: ssh-keygen -t ed25519 -C "your_email@example.com"
- 경로 지정: Enter file in which to save the key (/home/you/.ssh/id_ed25519):
- passphrase 설정, passphrase 확인

2. SSH Agent

- git bash 열기
- SSH 에이전트 실행: eval "$(ssh-agent -s)"
- SSH 키 추가: ssh-add ~/.ssh/id_ed25519 (등록 확인: ssh-add -l)
- SSH 공개 키 복사: cat ~/.ssh/id_ed25519.pub

3. SSH Add

- github 로그인
- 오른쪽 상단 프로필 클릭, Settings 클릭
- SSH and GPG keys 클릭, New SSH key 클릭
- Title 입력, 복사한 SSH 공개 키 붙여넣고 Add SSH key

Ⅳ. Creating a Remote Repository & Connect

- GitHub나 GitLab과 같은 플랫폼에서 원격 저장소를 생성합니다.
- 이후 로컬 저장소를 원격 저장소에 연결합니다. (SSH로 연결 예정)

• git remote add origin git@gitlab.com:user/repository.git (원격 저장소 주인)

Ⅴ. Collaboration

- GitHub에 로그인해 New repository를 클릭해 원격 저장소를 만듭니다.
- 이후 팀원들과 협업을 할 경우 아래의 내용을 따라오면 됩니다.

• GitHub에서 해당 repository로 이동, Settings 탭으로 이동
• 좌측 메뉴에서 Collaborators and teams 클릭
• Manage access에서 Add people or Add teams로 팀원 초대.
• 초대에 수락한 팀원들은 repository에 접근 가능.

1. Clone

- 팀원들은 repository <> Code에서 주소 복사 후 저장소를 로컬에 클론.

• .git clone git@github.com:사용자명/저장소이름.git (SSH)

2. Branches

- 팀원들은 기능이나 버그 수정을 위한 새로운 브랜치 생성 가능.
- 협업 방식과 브랜치 전략에 따라 달라짐. (Git Flow, GiHub Flow)

• git checkout -b 브랜치명

* Branch 삭제

• git checkout main
• git branch -d 삭제할 브랜치명
• git push origin --delete feature-branch (원격 브랜치 삭제)

3. Push & Pull

- 팀원들은 작업을 마친 후 변경 사항을 커밋하고, 원격 저장소로 푸시합니다.
- 또한, 팀원들은 항상 최신 코드로 보컬 브랜치를 업데이트 해야합니다.

* Push

• git add .
• git commit -m "작업 내용"
• git push origin 브랜치명

* Pull

• git pull origin 브랜치명

Ⅵ. Conflict

- 협업 중 여러 팀원이 동일한 파일이나 같은 코드 부분을 수정했을 때 충돌이 발생합니다.
- 충돌 상황은 주로 git pull 또는 git merge 명령을 사용할 때 발생합니다.
- 팀원들은 자주 Pull을 받아 최신 상태를 유지, 작은 단위로 자주 커밋, 푸시 해야합니다.

1. 충돌 확인

- 충돌이 발생하면 Git이 자동으로 알려줍니다.

ex) 병합 시 충돌

Auto-merging 파일이름.txt
CONFLICT (content): Merge conflict in 파일이름.txt
Automatic merge failed; fix conflicts and then commit the result.

2. 충돌 파일 확인

- 충돌된 파일도 자동으로 표시해주며, 해당 파일을 열어보면 충돌 부분을 표시해줍니다.

ex) 충돌된 파일 형태

<<<<<<< HEAD
여기에는 내 로컬 브랜치의 변경 사항이 있습니다.
=======
여기에는 병합하려는 원격 브랜치의 변경 사항이 있습니다.
>>>>>>> 브랜치명

3. 충돌 해결

- 충돌을 해결하려면, 각 충돌된 부분을 검토한 후 어떻게 처리할지를 결정해야 합니다.
- google serach or ChatGPT

• **로컬 브랜치 변경 사항 유지**: 내 로컬 브랜치의 변경 사항 유지, 상대 브랜치의 변경 사항 무시

• **원격 브랜치 변경 사항 유지**: 상대 브랜치의 변경 사항 유지, 내 로컬 브랜치의 변경 사항 무시

• **두 변경 사항을 수동으로 병합**: 두 변경 사항을 모두 고려해 최종 코드를 수동으로 작성합니다.

4. 충돌 해결 확인

- 충돌 해결 후, Git 상태를 확인해 충돌이 해결되었는지 확입합니다.
- 이후 해결된 파일 push등 다시 작업.

• git status

Ⅶ. PR을 통한 병합

- 모든 작업이 완료되면 GitHub에서 main 브랜치로 병합하기 위해 PR을 만듭니다.

• repository에서 Pull Request(PR) 탭을 클릭, "New Pull Request"를 선택
• base branch -> main브랜치로 설정, compare branch -> 팀원이 작업한 브랜치를 선택
• 필요한 설명을 추가, Create Pull Request버튼을 눌러 PR을 생성
• 이후 코드 리뷰 및 검토 후 Merge Pull Request로 병합 완료. 병합 후 로컬 업데이트.

Ⅰ. Docker Installation

- Docker를 사용하려면 먼저 로컬 컴퓨터에 Docker를 설치해야 합니다.

• Download Docker Desktop: (https://www.docker.com/products/docker-desktop/)
• WSL2 기반 설치 권장 (관련 사이트 첨부)
• 이후 cmd나 터미널에서 docker --version으로 확인 가능

Ⅱ. Dockerfile 작성

- Dockerfile은 애플리케이션의 환경을 정의하는 파일입니다

1. FROM

- Dockerfile의 첫 번째 명령어로, 어떤 베이스 이미지를 사용할지 지정.

• **FROM python:3.9**

2. WORKDIR

- Docker 컨테이너 내부에서 작업 디렉토리를 지정하는 명령어.

• **WORKDIR /app**

3. COPY

- 이 명령은 로컬 파일 시스템의 파일을 Docker 컨테이너의 지정된 위치로 복사.
- 이는 애플리케이션 파일이나 의존성 파일을 Docker 컨테이너로 옮길 때 사용.

• **COPY . /app**

4. RUN

- Docker 이미지를 빌드할 때 실행될 명령어를 지정.
- 주로 패키지 설치나 시스템 설정과 같은 작업을 수행할 때 사용.

• **RUN pip install -r requirements.txt**

5. EXPOSE

- Docker 컨테이너가 외부에 노출할 포트를 지정.
- 이 명령어는 컨테이너와 외부 시스템 간의 네트워크 연결을 설정하는 데 사용.

• **EXPOSE 80**
• **docker run -p 8080:80 image_name (실제 포트 열기)**

6. CMD

- 컨테이너가 시작될 때 실행할 명령을 지정, 이 명령은 컨테이너의 기본 명령어가 됨.
- 한 개의 CMD 명령만 사용 가능, 여러 개의 CMD의 경우 마지막 명령만 적용.

• **CMD ["python", "app.py"]**

7. ENTRYPOINT
- CMD와 비슷하지만, CMD보다 우선순위가 높고 고정적인 명령을 지정할 때 사용.
- 변경할 수 없는 기본 실행 프로세스를 지정할 때 유용.

- **ENTRYPOINT ["python", "app.py"]**

8. ENV
- 환경 변수 설정에 사용.

- **ENV APP_ENV=production**

ETC. VOLUME, ADD 등

ex) Dockerfile 예시

```
# Python 이미지 사용
FROM python:3.9

# 작업 디렉토리 설정
WORKDIR /app

# 로컬 파일을 도커 컨테이너로 복사
COPY . /app

# 필요한 패키지 설치
RUN pip install -r requirements.txt

# Flask 앱 실행
CMD ["python", "app.py"]
```

Ⅲ. Docker Image Build
- Dockerfile을 작성했다면 이제 Docker 이미지를 빌드.
- Build시 오류 발생 가능 -> google search or ChatGPT.

- cd /path/to/your/project
- docker build -t your_image_name .
- docker images

Ⅳ. Docker Use
- docker run <이미지 이름> (-d: 백그라운드 실행, --name: 이름 지정)
- docker ps
- docker start <컨테이너 이름 또는 ID> (restart는 재시작)
- docker stop <컨테이너 이름 또는 ID> ***모두 중지: docker stop $(docker ps -q)**

- docker rm <컨테이너 이름 또는 ID> (중지된 컨테이너 삭제)
- docker images (이미지 조회)
- docker rmi <이미지 ID 또는 이름> (이미지 삭제) ***모든 이미지: docker image prune**
- docker logs <컨테이너 이름 또는 ID> (-f: 실시간 확인)
- docker exec -it <컨테이너 이름 또는 ID> /bin/bash (도커 내부 쉘 접속)
- docker system prune (도커 정리 -a: all, --volumes: 볼륨도)