

pc용 도커 설치

1. 호스트 시스템에 적합한 설치 프로그램 다운로드

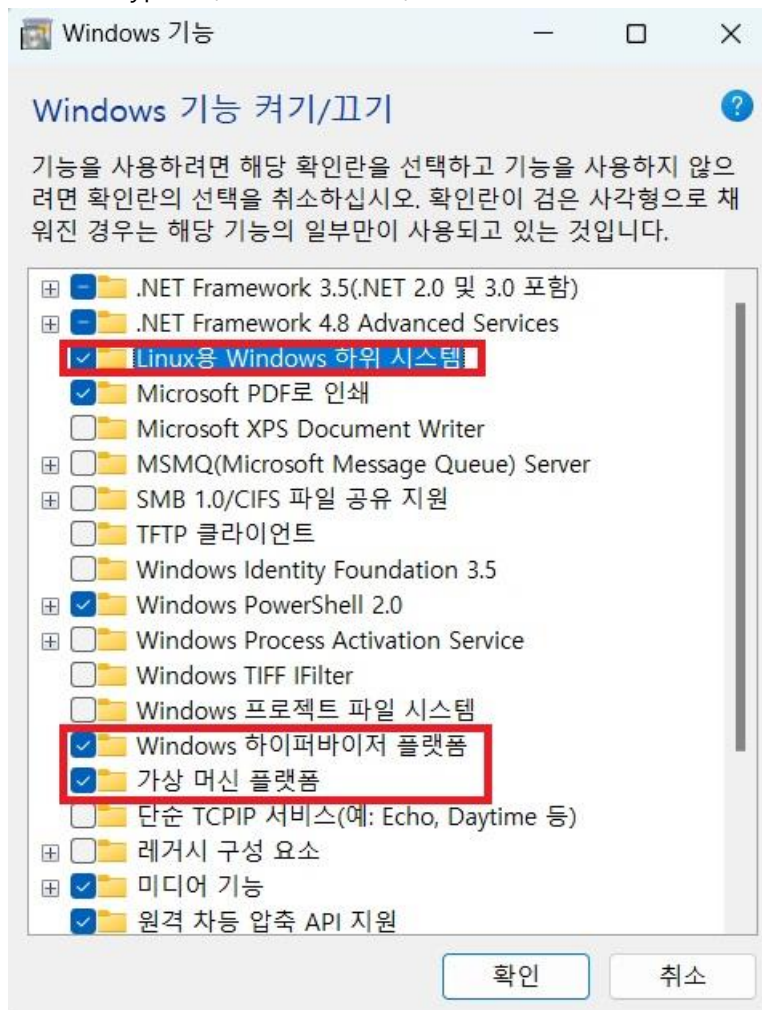
<https://docs.docker.com/get-started/get-docker/>

2. 설치전 pc 설정

제어판 -> 프로그램 -> 윈도우 기능 켜기/끄기 에서

윈도우 pro, enterprise 에디션:

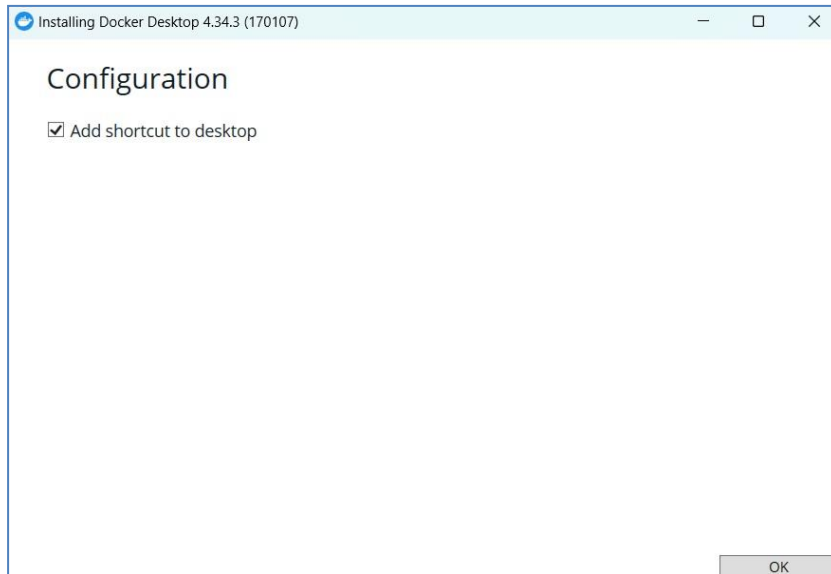
Hyper-V(하위 모든 항목) / Linux용 Windows 하위 시스템 / 가상머신 플랫폼 항목 체크



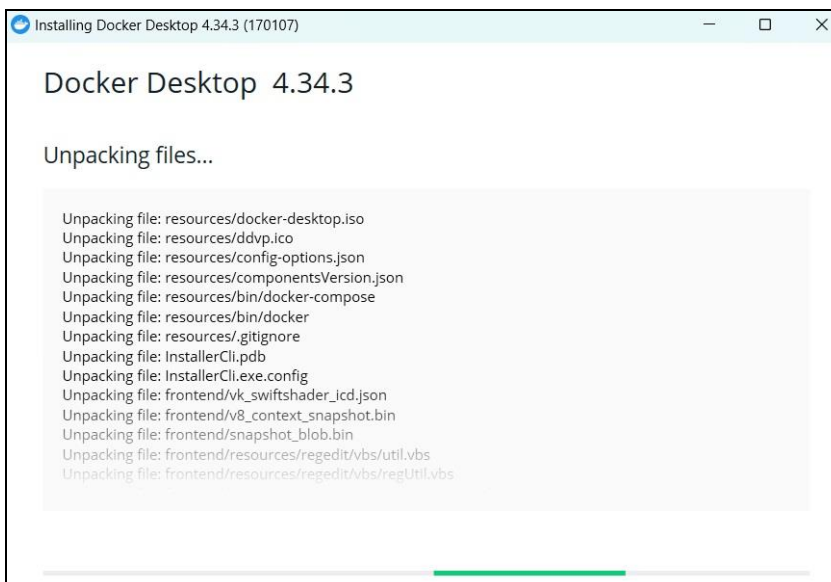
<그림은 윈도우 home 에디션>

3. 위 확인 버튼 누르고 컴퓨터 재부팅

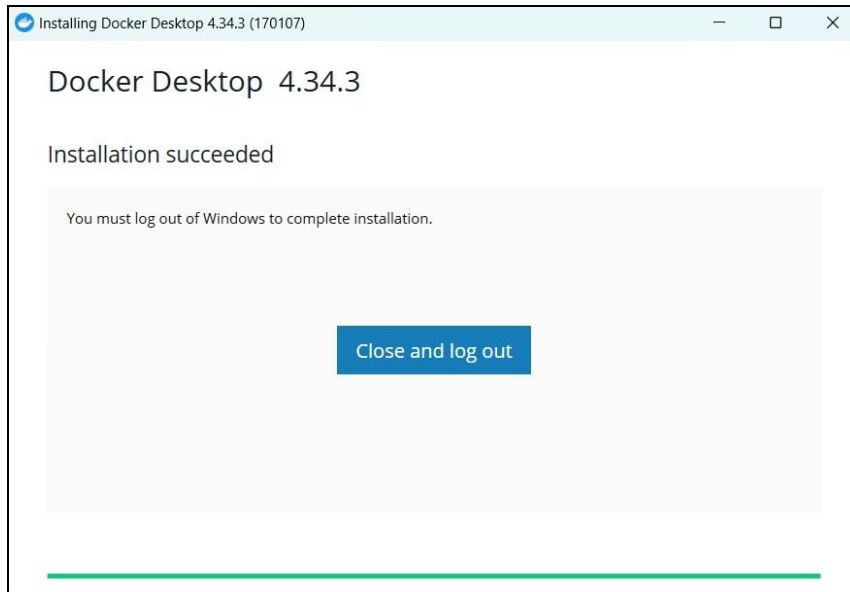
4. 도커 인스톨 프로그램 실행



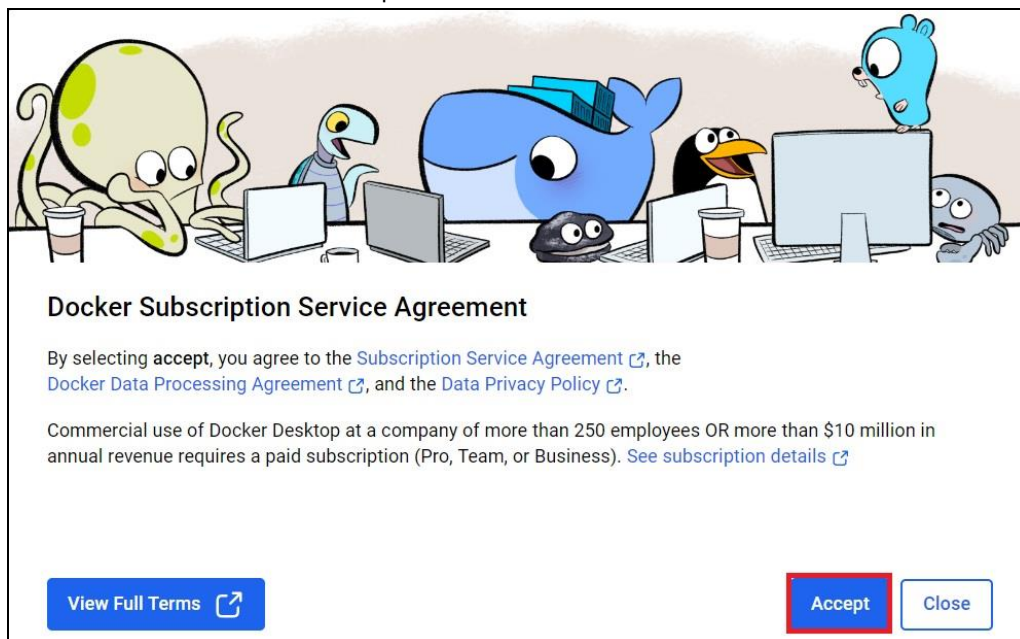
ok 버튼 클릭하면 아래 그림처럼 설치 진행됨




설치 완료 시 close 버튼 클릭



close 버튼 클릭하면 컴퓨터 재부팅됨.
재부팅 뒤 아래 화면에서 accept 선택



아래처럼 기본 설정 선택



Finish setting up Docker Desktop

version 4.34.3 (170107)

Complete the installation of Docker Desktop.

☒

Use recommended settings (requires administrator password)
Docker Desktop automatically sets the necessary configurations that work for most developers.

☐

Use advanced settings
You manually set your preferred configurations.

Finish

로그인. 계정 없으면 생성

Welcome to Docker

[Skip](#)

Work


Personal


Email address


이메일작성

Continue


Or





Create an account 

도커 로그인



Enter Your Password

Enter your password for Docker to continue to Docker Desktop


[Edit](#)


Password*

[Forgot password?](#)

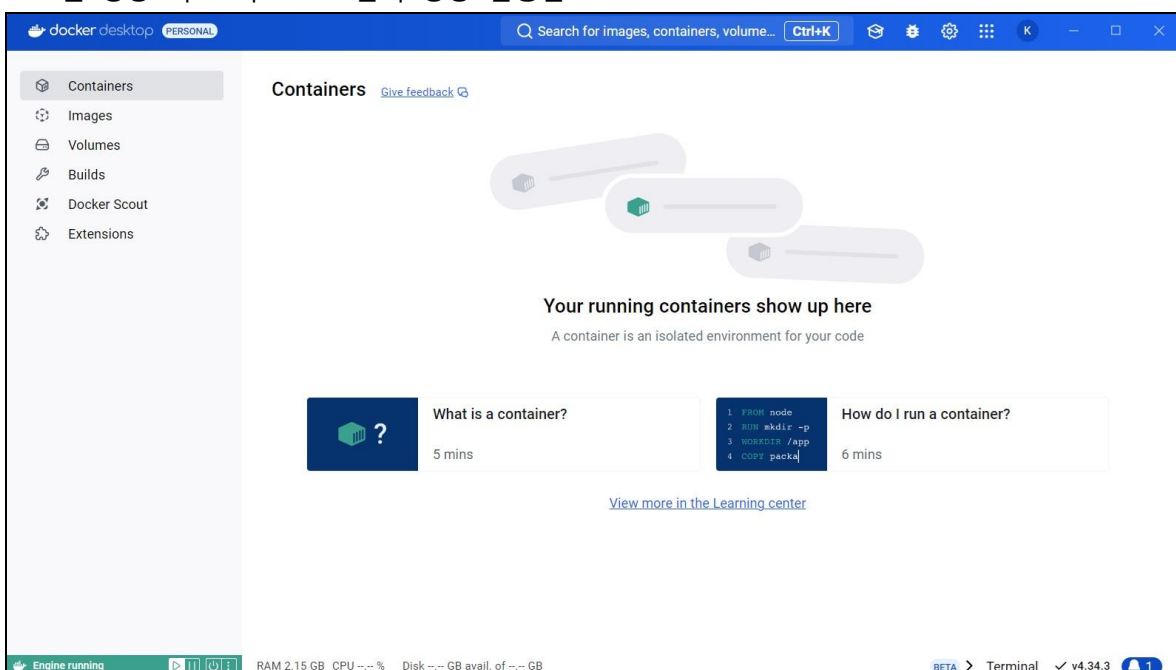
Continue

OR

 Continue with Google

 Continue with GitHub

로그인 성공 시 도커 프로그램이 정상 실행됨



커맨드 창에서 도커 실행하기

윈도우 검색 창에 cmd 입력하여 커맨드 창 실행 후 다음 도커 명령을 실행한다.

1. 도커 버전 확인

```
C:\Users\Wkingn>docker --version
```

```
Docker version 27.2.0, build 3ab4256
```

2. 현재 실행중인 컨테이너 목록 확인

```
C:\Users\Wkingn>docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

=> 실행중인 컨테이너가 없다.

3. 도커 기본 레지스트리에서 제공하는 hello-world 실행

```
C:\Users\Wkingn>docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
```

```
latest: Pulling from library/hello-world
```

```
c1ec31eb5944: Download complete
```

```
Digest: sha256:d211f485f2dd1dee407a80973c8f129f00d54604d2c90732e8e320e5038a0348
```

```
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

=> 호스트에는 hello-world 이미지가 없으므로 다운로드하여 실행하는 것을 출력 메시지를 통해 알 수 있다.

* 만약 위 명령어 실행 시 인증 문제가 발생하면 로그인 먼저 한다.

-- 로그인

C:\Users\kingn>**docker login**

Authenticating with existing credentials...

Stored credentials invalid or expired

Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to <https://hub.docker.com/> to create one.

You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using SSO. Learn more at <https://docs.docker.com/go/access-tokens/>

Username (kingnuna): 계정 입력

Password: 패스워드 입력

Login Succeeded

4. 실행중인 컨테이너 다시 확인.

C:\Users\kingn>**docker ps -a**

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
361a36daa331	hello-world	"/hello"	4 minutes ago	Exited (0) 4 minutes ago

recurring_kapitsa

=> 이 프로그램은 "Hello from Docker!" 메시지를 간단히 출력하고 종료하므로 ps 명령 실행 시 이미 종료된 상태이기 때문에 목록에 뜨지 않는다. -a 옵션을 주면 종료된 컨테이너도 출력됨

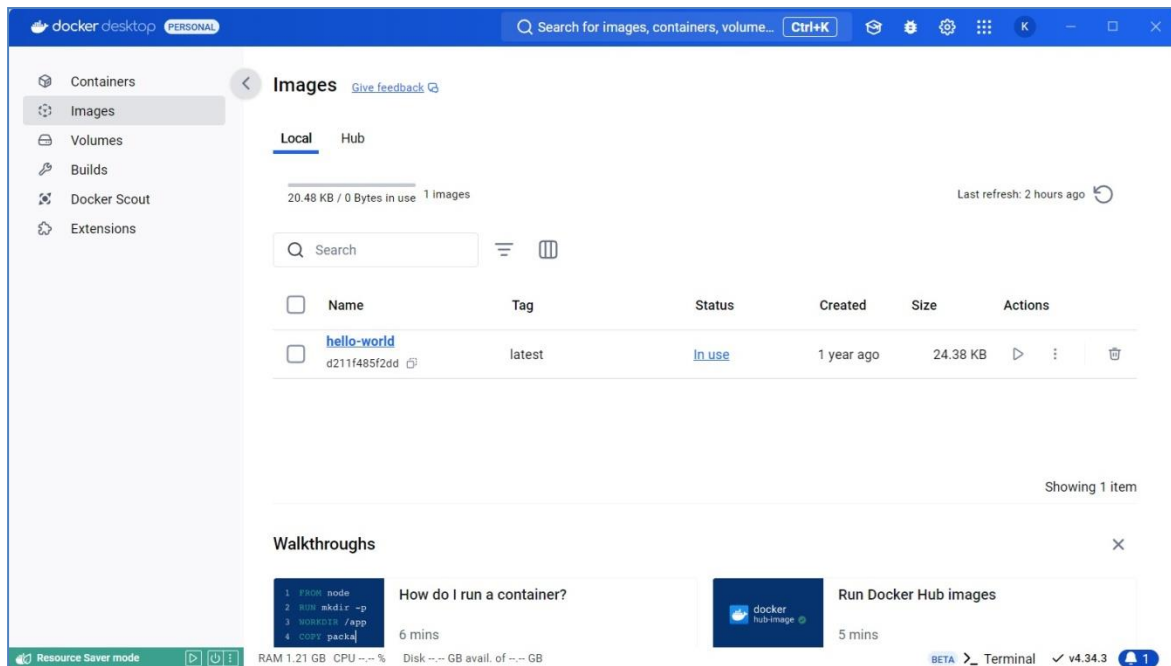
5. 다운받은 도커 이미지 확인

C:\Users\kingn>**docker image ls**

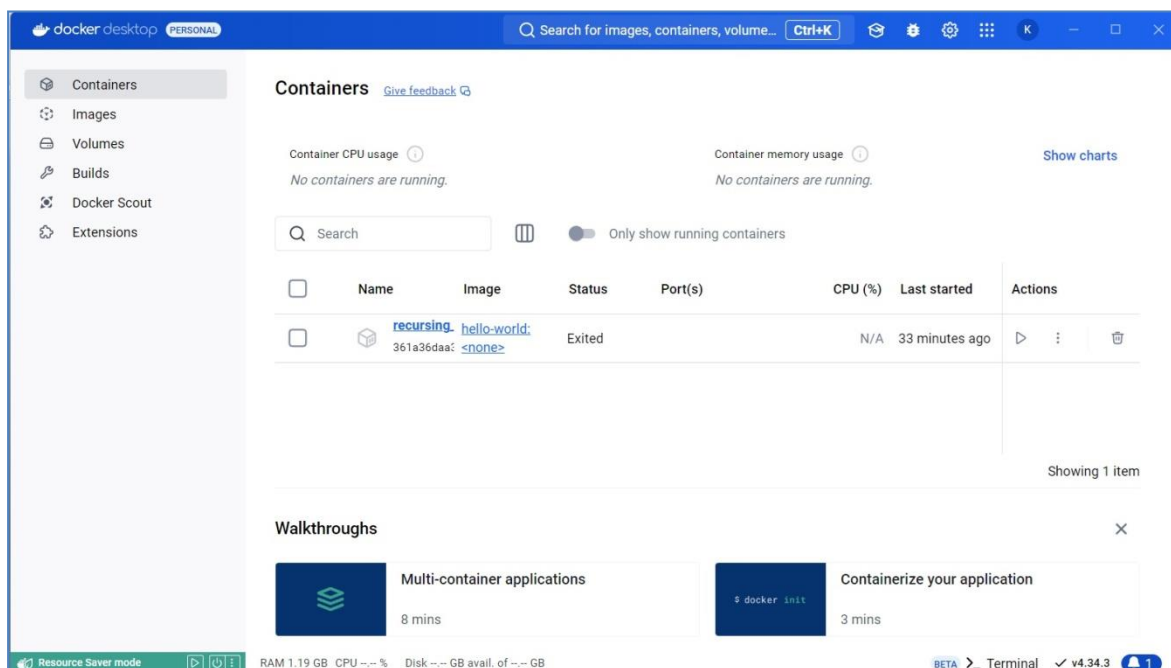
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	d211f485f2dd	17 months ago	24.4kB

=> hello-world 이미지가 목록에 뜬다

6. 컨테이너 목록을 ui 프로그램으로 확인



7. 이미지 목록을 ui 프로그램으로 확인



도커 기본 명령어

컨테이너의 ip등 상세정보확인

docker inspect 컨테이너id

#실행중인 컨테이너 목록

docker ps

#컨테이너 목록확인(실행종료된 컨테이너도 보여줌)

docker container ls -a

#이미지 목록

docker images

#도커 로그인

docker login

#도커 로그아웃

docker logout

#실행중인 컨테이너 종료

docker stop 컨테이너ID

#컨테이너 실행

docker container start <컨테이너명>

#컨테이너 삭제

docker container rm <컨테이너ID>

#이미지 삭제

docker image rm <이미지명:태그명>

실습1

이클립스에서 직접 빌드하여 생성된 war를 도커 이미지로 생성하고 실행한다.

- spring boot <hello world> 프로그램 생성. 진짜 hello world만 출력하는 웹 어플리케이션

- 이클립스 프로젝트 우클릭 > run as > maven build > goal:package 입력 후 런 실행
=> 프로젝트의 target 폴더에 war 파일 생성됨

- 아무 위치에 작업할 폴더 생성하고 war파일복사 및 확장자 없는 Dockerfile이름의 파일 생성
Dockerfile에 다음 내용 입력

```
FROM openjdk:17
COPY *.war app.war
```

```
CMD ["java","-jar","/app.war"]
```

- cmd 프로그램 실행하여 도커 이미지 빌드
C:\Users\Wkingn\Desktop\WEX1>docker build -t app1 .
- 이미지 목록 확인
C:\Users\Wkingn\Desktop\WEX1>docker images
C:\Users\Wkingn\Desktop\WEX1>docker image ls
- 이미지 컨테이너로 실행
포트지정: <host> : <container>
C:\Users\Wkingn\Desktop\WEX1> docker run -d -p 8081:8081 app1
- 웹 브라우저에서 실행해본다.

실습2

이미지 레이어링으로 빌드 단계를 빌드 단계와 실행단계 이렇게 2개로 나눔.

- 실습1의 spring boot 프로젝트 루트 디렉토리에 Dockerfile 생성하고 다음 내용 입력

```
FROM eclipse-temurin:21.0.2_13-jdk-jammy AS builder
#WORKDIR /doc-app1
COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY ./src ./src
RUN ./mvnw clean install

FROM eclipse-temurin:21.0.2_13-jre-jammy AS final
#WORKDIR /doc-app1
COPY --from=builder target/*.war app.war
ENTRYPOINT ["java", "-jar", "app.war"]
```

- cmd 프로그램 실행하여 도커 이미지 빌드
C:\Users\Wkingn\workspace\doc-app1>docker build -t app1 .
- 이미지 컨테이너로 실행

```
C:\Users\Wkingn\workspace\doc-app1>docker run -d -p 8081:8081 app1
7deef6a5a014214b35c7174b4b28e70c457eb71b2ae8a455849d66807202da03
```

실습3

컴포즈로 여러 컨테이너를 묶어서 제어해본다.

- 이클립스에서 데이터베이스(mysql) 연동된 spring boot jpa 프로젝트 생성
- 프로젝트 루트에 Dockerfile 생성

```
FROM eclipse-temurin:21.0.2_13-jdk-jammy AS builder
#WORKDIR /doc-app1
COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY ./src ./src
RUN ./mvnw -Dmaven.test.skip=true clean install

FROM eclipse-temurin:21.0.2_13-jre-jammy AS final
#WORKDIR /doc-app1
COPY --from=builder target/*.war app_db.war
ENTRYPOINT ["java", "-jar", "app_db.war"]
```

- 프로젝트 루트에 컴포즈 파일(docker-compose.yml) 생성

```
version: '3'

services:
  database:
    container_name: mysql_db
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: world
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 1234
      TZ: 'Asia/Seoul'
```

```
ports:
  - "3306:3306"
volumes:
  - ./mysql/conf.d:/etc/mysql/conf.d # MySQL 설정 파일 위치
command:
  - "mysqld"
  - "--character-set-server=utf8mb4"
  - "--collation-server=utf8mb4_unicode_ci"
networks:
  - test_network
```

```
application:
  container_name: db_app
  restart: on-failure
  build:
    context: ./
    dockerfile: Dockerfile
```

```
ports:
  - "8081:8081"
```

```
environment:
  SPRING_DATASOURCE_URL:
```

```
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
```

```
  SPRING_DATASOURCE_USERNAME: "root"
```

```
  SPRING_DATASOURCE_PASSWORD: "1234"
```

```
depends_on:
  - database
```

```
networks:
  - test_network
```

```
networks:
  test_network:
```

- cmd(커맨드) 창에서 컴포즈 명령 실행

```
C:\Users\Wkingn\workspace\doc-app2>docker-compose up
```

```
time="2024-10-13T23:16:10+09:00" level=warning msg="C:\Users\Wkingn\workspace\doc-app2\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
```

```
✓                                     database                                     Pulled
12.0s
✓                                     eba3c26198b7                                     Download                                     complete
3.8s
...
mysql_db | 2024-10-13T14:16:37.429526Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld
(mysqlld 8.0.39) initializing of server in progress as process 80
mysql_db | 2024-10-13T14:16:37.458871Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization
has started.
mysql_db | 2024-10-13T14:16:38.353591Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization
has ended.
db_app |
db_app | . ____ _
db_app | /WW / __'__ _(_)__ _ WW WW WW
db_app | (( )W__|'_|'|'_ W/_'| WW WW WW
db_app | WW/ __)| |)| | | | | (| | ) ) ) )
db_app | ' |__|_|_|_|_|_W_| | / / / /
db_app | =====|_|=====|_|=/ / / / /
db_app |
db_app | :: Spring Boot :: (v3.3.4)
db_app |
db_app | 2024-10-13T14:16:38.587Z INFO 1 --- [doc-app2] [ main]
com.example.demo.DocApp2Application : Starting DocApp2Application v0.0.1-SNAPSHOT
using Java 21.0.2 with PID 1 (/app_db.war started by root in /)
...
```

- aws에 도커 설치

```
ubuntu@ip-172-31-7-221:~$ sudo apt install docker.io
```

- 도커 실행/중지

```
$ sudo service docker stop
```

- 도커 엔진 삭제

```
$ sudo apt-get purge -y docker-engine docker docker.io docker-ce docker-ce-cli
$ sudo apt-get autoremove -y --purge docker-engine docker docker.io docker-ce
```

- 도커 컨테이너, 이미지, 볼륨, 사용자가 만든 파일 등 전부 삭제

```
$ sudo rm -rf /var/lib/docker /etc/docker
$ sudo rm /etc/apparmor.d/docker
$ sudo groupdel docker
$ sudo rm -rf /var/run/docker.sock
```

실습4

간단한 구조의 앱 테스트

1) 이미지에 태그명 추가

실습2에서 생성한 도커 이미지(app1)에 태그 추가

```
C:\WUsers\WUSER>docker tag app1 kingnuna/app1:v1 #username/app name:태그명
```

2) repository에 이미지 올리기

도커 사이트 로그인 -> 헤더에 Repositories 메뉴 선택 -> create repository 버튼 클릭하여 새 repository(app1) 생성

아래 명령어로 태그 지정한 이미지를 repository에 올린다.

```
C:\WUsers\WUSER>docker push kingnuna/app1:v1 #username/app name:태그명
```

도커 사이트에서 repository에 잘 올라갔는지 확인한다.

3) aws에서 이미지 다운로드

```
$ sudo docker pull kingnuna/app1:v1 #username/app name:태그명
```

다운로드 확인

```
$ sudo docker images
```

4) 이미지 컨테이너로 실행

```
$ sudo docker run -d -p 8081:8081 kingnuna/app1:v1
```

실습5

db(mysql) 연동 및 컴포즈 예제 실습

이클립스에서 doc-app2 프로젝트 application.properties 파일의 db 설정 부분 다음과 같이 작성

```
# mysql set
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
#spring.datasource.url=jdbc:mysql://mysql_db:3306/world?serverTimezone=UTC&characterEncoding=UTF-8&useSSL=false&allowPublicKeyRetrieval=true
#spring.datasource.username=root
#spring.datasource.password=1234
```

=> 수정 뒤 다시 빌드.

빌드 전 이전 컨테이너(doc-app2, mysql_db) 종료, 삭제 후 컴포즈 업을 실행한다.

앱 이미지(doc-app2-application) 도 삭제해야 컴포즈시 새로 빌드된 이미지가 실행됨

C:\Users\WUSER>docker-compose up

1) 이미지에 태그명 추가

실습3에서 생성한 도커 이미지(doc-app2-application)에 태그 추가

```
C:\Users\WUSER>docker tag doc-app2-application kingnuna/doc-app2:v1
#username/app name:태그명
```

2) repository에 이미지 올리기

도커 사이트 로그인 -> 헤더에 Repositories 메뉴 선택 -> create repository 버튼 클릭하여 새 repository(doc-app2) 생성

아래 명령어로 태그 지정한 이미지를 repository에 올린다.

```
C:\Users\WUSER>docker push kingnuna/doc-app2:v1 #username/app name:태그명
```

도커 사이트에서 repository에 잘 올라갔는지 확인한다.

3) 컴포즈 파일 업로드

다음과 같이 컴포즈파일 수정한 뒤 파일을 aws에 올린다.

```
version: '3'

services:
  database:
    container_name: mysql_db
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: world
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 1234
      TZ: 'Asia/Seoul'
    ports:
```

```

- "3306:3306"
volumes:
  - ./mysql/conf.d:/etc/mysql/conf.d # MySQL 설정 파일 위치
command:
  - "mysqld"
  - "--character-set-server=utf8mb4"
  - "--collation-server=utf8mb4_unicode_ci"
networks:
  - test_network

application:
  container_name: db_app
  # restart: on-failure
  image: kingnuna/doc-app2:v1
  # build:
  # context: ./
  # dockerfile: Dockerfile
  ports:
    - "8081:8081"
  environment:
    SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
    SPRING_DATASOURCE_USERNAME: "root"
    SPRING_DATASOURCE_PASSWORD: "1234"
  depends_on:
    - database
  networks:
    - test_network

networks:
  test_network:

```

C:\Users\USER>scp -i Downloads/ubuntuservkey.pem C:\Users\kingn\workspace\doc-app2\docker-compose.yml ubuntu@43.203.128.242:~

4) aws에서 이미지 다운로드

\$ sudo docker pull kingnuna/doc-app2:v1 #username/app name:태그명

다운로드 확인


```
$ sudo docker images
```

5) 컴포즈 실행

```
$ sudo docker-compose up
```

컴포즈를 다시 실행 시 이전 컨테이너(doc-app2, mysql_db) 종료, 삭제 후 컴포즈 업을 실행한 다. 앱 이미지(doc-app2-application) 도 삭제해야 컴포즈시 새로 빌드된 이미지가 실행됨

```
$ sudo docker stop <컨테이너ID>
```

```
$ sudo docker container rm <컨테이너ID>
```

```
$ sudo docker image rm <이미지명:태그명>
```

```
$ sudo docker ps -a
```

실습6

volume을 사용한 파일업로드 실습

1) 이클립스에서 doc-app3 프로젝트 application.properties 파일의 db 설정 부분과 업로드 설정 을 다음과 같이 작성

```
# mysql set
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

#multipart
spring.servlet.multipart.location=/img/
spring.servlet.multipart.max-file-size=5MB
```

2) 도커 파일 작성

파일명: Dockerfile

위치: 프로젝트 루트

```
FROM eclipse-temurin:21.0.2_13-jdk-jammy AS builder
#WORKDIR /doc-app1
COPY .mvn/ .mvn
COPY mvnw pom.xml ./
RUN ./mvnw dependency:go-offline
COPY ./src ./src
#RUN ./mvnw clean install
RUN ./mvnw -Dmaven.test.skip=true clean install

FROM eclipse-temurin:21.0.2_13-jre-jammy AS final
#WORKDIR /doc-app1
```

```
COPY --from=builder target/*.war app_db.war
ENTRYPOINT ["java", "-jar", "app_db.war"]
```

3) 컴포즈 파일 작성

파일명: docker-compose.yml

위치: 프로젝트 루트

```
version: '3'

services:
  database:
    container_name: mysql_db
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: world
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 1234
      TZ: 'Asia/Seoul'
    ports:
      - "3306:3306"
    volumes:
      - ./mysql/conf.d:/etc/mysql/conf.d # MySQL 설정 파일 위치
    command:
      - "mysqld"
      - "--character-set-server=utf8mb4"
      - "--collation-server=utf8mb4_unicode_ci"
  # healthcheck:
  #   test: ["CMD", "echo 'SELECT version();' | mysql"]
  #   timeout: 20s
  #   retries: 10
  networks:
    - test_network

  application:
    container_name: doc-app3
    restart: on-failure
    build:
      context: ./
```

```

    dockerfile: Dockerfile
    ports:
      - "8081:8081"
    volumes:
      - ./img:/img
    environment:
      SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
      SPRING_DATASOURCE_USERNAME: "root"
      SPRING_DATASOURCE_PASSWORD: "1234"
    depends_on:
      - database
    networks:
      - test_network

networks:
  test_network:

```

4) 컴포즈 실행

컴포즈 여러 번 실행 시 이전 컨테이너(doc-app3, mysql_db) 종료, 삭제 후 컴포즈 업을 실행한다.

앱 이미지(doc-app3-application) 도 삭제해야 컴포즈시 새로 빌드된 이미지가 실행됨

```
C:\Users\WUSER>docker-compose up
```

5) doc-app3, mysql_db 컨테이너가 실행되면 웹 브라우저에서 "localhost:8081"로 실행한다.

추가 메뉴에서 이름과 사진 입력하고 추가하면 목록에서 이름과 이미지가 떠야함

업로드된 이미지는 프로젝트 루트(컴포즈파일 위치와 동일한 위치)에 img 폴더가 생기고 그 안에 저장된다.

6) 이미지에 태그명 추가

4)에서 생성된 도커 이미지(doc-app3-application)에 태그 추가

```
C:\Users\WUSER>docker tag doc-app3-application kingnuna/doc-app3:v1 #username/app
name:태그명
```

7) repository에 이미지 올리기

도커 사이트 로그인 -> 헤더에 Repositories 메뉴 선택 -> create repository 버튼 클릭하여 새 repository(doc-app3) 생성

아래 명령어로 태그 지정한 이미지를 repository에 올린다.

C:\Users\WUSER>docker push kingnuna/doc-app3:v1

#username/app name:태그명

도커 사이트에서 repository에 잘 올라갔는지 확인한다.

8) 컴포즈 파일 업로드

다음과 같이 컴포즈파일 수정한 뒤 파일을 aws에 올린다.

```
version: '3'

services:
  database:
    container_name: mysql_db
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: world
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 1234
      TZ: 'Asia/Seoul'
    ports:
      - "3306:3306"
    volumes:
      - ./mysql/conf.d:/etc/mysql/conf.d # MySQL 설정 파일 위치
    command:
      - "mysqld"
      - "--character-set-server=utf8mb4"
      - "--collation-server=utf8mb4_unicode_ci"
  # healthcheck:
  #   test: ["CMD", "echo 'SELECT version();'| mysql"]
  #   timeout: 20s
  #   retries: 10
  networks:
    - test_network
  application:
    container_name: doc-app3
    restart: on-failure
    image: kingnuna/doc-app3:v1
    # build:
    #context: ./
```

```

#dockerfile: Dockerfile
ports:
  - "8081:8081"
volumes:
  - ./img:/img
environment:
  SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
  SPRING_DATASOURCE_USERNAME: "root"
  SPRING_DATASOURCE_PASSWORD: "1234"
depends_on:
  - database
networks:
  - test_network

networks:
  test_network:

```

```

C:\Users\WUSER>scp -i Downloads/ubuntukey.pem C:\Users\kingn\workspace\doc-
app3\docker-compose.yml ubuntu@43.203.128.242:~/doc-app3

```

9) aws에서 이미지 다운로드

```

$ sudo docker pull kingnuna/doc-app3:v1 #username/app name:태그명

```

다운로드 확인

```

$ sudo docker images

```

10) 컴포즈 실행

```

$ sudo docker-compose up

```

컴포즈를 다시 실행 시 이전 컨테이너(doc-app3, mysql_db) 종료, 삭제 후 컴포즈 업을 실행한
다. 앱 이미지(doc-app2-application) 도 삭제해야 컴포즈시 새로 빌드된 이미지가 실행됨

```

$ sudo docker stop <컨테이너ID>

```

```

$ sudo docker container rm <컨테이너ID>

```

```

$ sudo docker image rm <이미지명:태그명>

```

```

$ sudo docker ps -a

```

11) 웹 브라우저에서 "aws주소:8081"로 실행

파일 업로드 확인. img 폴더는 컴포즈 파일과 동일한 위치에 생성되고 그 안에 업로드한 이미지가 저장됨

```
ubuntu@ip-172-31-14-94:~/doc-app3$ ls
docker-compose.yml  img  mysql
ubuntu@ip-172-31-14-94:~/doc-app3$ cd img
ubuntu@ip-172-31-14-94:~/doc-app3/img$ ls
bird-8007868_640.jpg
```

< CI / CD >

CI/CD는 지속적 통합(Continuous Integration) 및 지속적 제공/배포(Continuous Delivery/Deployment)를 의미하며, 소프트웨어 개발 라이프사이클을 간소화하고 가속화하는 것을 목표로 한다.



지속적 통합이란?

CI/CD에서 "CI"는 지속적 통합을 의미하며, 지속적 통합은 코드 변경 사항을 다시 공유, 분기를 빈번하게 병합하는 것을 용이하게 하는 개발자용 자동화 프로세스다. 이러한 업데이트가 이루어지면 병합된 코드 변경 사항의 신뢰성을 보장하기 위해 자동화된 테스트 단계가 실행된다.

현대적인 애플리케이션 개발에서는 여러 개발자들이 동일한 애플리케이션의 각기 다른 기능을 동시에 작업할 수 있도록 하는 것을 목표로 한다. 그러나 팀에서 날("병합의 날을 정해 모든 분기 소스 코드를 병합하는 경우, 결과적으로 반복적인 수작업에 많은 시간을 소모하게 된다. 예로 한 개발자가 애플리케이션을 변경하는 경우 다른 개발자들이 동시에 적용하는 여러 변경 사항들과 충돌할 가능성이 있기 때문이다.

이렇게 동시에 개발 중인 애플리케이션의 분기가 너무 많아 상호 충돌할 가능성이 있는 문제에 대한 해결책으로 CI를 고려할 수 있다.

성공적인 CI란 개발자가 애플리케이션에 적용한 변경 사항들이 병합된 후 이러한 변경 사항이 애플리케이션을 손상시키지 않도록 자동으로 애플리케이션을 빌드하고 다양한 수준의 자동화된 테스트(일반적으로 단위 테스트와 통합 테스트)를 실행하여 해당 변경 사항을 검증하는 것이다. 즉, 클래스와 기능에서부터 전체 애플리케이션을 구성하는 다양한 모듈에 이르기까지 모든 것을 테스트한다. 자동화된 테스트를 통해 새 코드와 기존 코드 간 충돌이 발견되는 경우 CI를 적용하면 해

당 버그를 빠르게, 자주 수정하기가 더 용이해진다.

CI/CD에서 "CD"란?

CI/CD의 "CD"는 지속적 제공(Continuous Delivery) 또는 지속적 배포(Continuous Deployment)를 의미하며 이 두 용어는 상호 교환하여 사용된다. 두 가지 모두 파이프라인의 추가 단계를 자동화하는 것이지만 자동화가 얼마나 많이 진행되고 있는지를 나타내기 위해 별도로 사용될 수도 있다. 지속적 제공과 지속적 배포 중 어느 것을 선택할지는 개발 팀과 운영 팀의 위험 허용 범위와 구체적인 요구 사항에 따라 달라진다.

지속적 제공이란?

지속적 제공은 CI에서 빌드와 단위 및 통합 테스트를 자동화한 다음 검증된 코드를 리포지토리로 배포하는 것을 자동화한다. 따라서 효과적인 지속적 제공 프로세스를 마련하려면 CI가 개발 파이프라인에 이미 구축되어 있어야 한다.

일반적으로 지속적 제공이란 개발자의 애플리케이션 변경 사항이 자동으로 버그 테스트를 거치고 리포지토리(예: GitHub, 컨테이너 레지스트리)로 업로드 된다는 것을 의미한다. 이후 리포지토리에서 운영 팀이 변경 사항을 라이브 프로덕션 환경으로 배포할 수 있다. 그 결과 개발 팀과 비즈니스 팀 간 가시성 및 의사 소통 부족 문제가 해결될 수 있다. 이를 위한 지속적 제공의 목표는 언제나 프로덕션 환경으로 배포할 준비가 되어 있는 코드베이스를 갖추고 새로운 코드를 배포하는데 필요한 노력을 최소화하는 것이다.

지속적 배포란?

CI/CD 파이프라인의 최종 단계는 지속적 배포다. 지속적 배포는 지속적 제공의 확장으로, 개발자의 변경 사항을 리포지토리에서 프로덕션으로 릴리스하는 것을 자동화하여 고객이 사용할 수 있도록 하는 것을 말한다. CD는 애플리케이션 제공 속도를 저하시키는 수동 프로세스로 인한 운영 팀의 업무 과다 문제를 해결한다.

실제 사례에서 지속적 배포란 개발자가 애플리케이션에 변경 사항을 작성한 후 몇 분 이내에 자동화된 테스트를 통과하고 클라우드 애플리케이션을 자동으로 실행할 수 있는 것을 의미한다. 이를 통해 사용자 피드백을 지속적으로 수신하고 통합하는 일이 훨씬 수월하다. 이러한 연결된 모든 CI/CD 사례는 애플리케이션 배포의 리스크를 줄여주므로 애플리케이션 변경 사항을 한꺼번에 릴리스하지 않고 작은 단위로 세분화하여 더욱 손쉽게 릴리스할 수 있다.

그러나 프로덕션 이전의 파이프라인 단계에는 수동 게이트가 없으므로 지속적 배포는 잘 설계된 테스트 자동화에 크게 의존한다. 따라서 CI/CD 파이프라인에서 다양한 테스트 및 릴리스 단계를 수용하기 위해 자동화된 테스트를 작성해야 하므로 지속적 배포에는 많은 선행 투자가 필요하다.

CI/CD와 DevOps 비교

CI/CD는 개발 팀과 운영 팀 간 협업 촉진을 목표로 하는 DevOps 방법론의 필수적인 부분이다.

CI/CD와 DevOps는 모두 코드 통합 프로세스를 자동화하는 데 중점을 두어 사용자에게 가치를 제공할 수 있는 프로덕션 환경에서 새 기능, 개선 사항 요청, 버그 수정 등의 아이디어가 개발에서 배포 단계로 이동하는 프로세스를 가속화한다.

실습7

git action을 통한 ci/cd 환경을 구축한다.

1) 깃허브 로그인. 새 repository 생성하고 doc-app3 프로젝트를 깃에 push 한다.

프로젝트 루트에 Dockerfile 포함해야함. 내용은 다음과 같다.

```
FROM openjdk:17
ARG JAR_FILE=*.war
COPY ${JAR_FILE} app.war
ENTRYPOINT ["java","-jar","/app.war"]
```

- 컴포즈 파일 업로드

다음과 같이 컴포즈파일 수정한 뒤 파일을 aws에 올린다.

```
version: '3'

services:
  database:
    container_name: mysql_db
    image: mysql:8.0
    restart: always
    environment:
      MYSQL_DATABASE: world
      MYSQL_ROOT_HOST: '%'
      MYSQL_ROOT_PASSWORD: 1234
      TZ: 'Asia/Seoul'
    ports:
      - "3306:3306"
    volumes:
      - ./mysql/conf.d:/etc/mysql/conf.d # MySQL 설정 파일 위치
    command:
      - "mysqld"
      - "--character-set-server=utf8mb4"
      - "--collation-server=utf8mb4_unicode_ci"
#   healthcheck:
#     test: ["CMD", "echo 'SELECT version();'| mysql"]
#     timeout: 20s
```



```

#       retries: 10
      networks:
        - test_network
application:
  container_name: doc-app3
  restart: on-failure
  image: kingnuna/doc-app3:v1
  #   build:
  #context: ./
  #dockerfile: Dockerfile
  ports:
    - "8081:8081"
  volumes:
    - ./img:/img
  environment:
    SPRING_DATASOURCE_URL:
jdbc:mysql://mysql_db:3306/world?useSSL=false&allowPublicKeyRetrieval=true
    SPRING_DATASOURCE_USERNAME: "root"
    SPRING_DATASOURCE_PASSWORD: "1234"
  depends_on:
    - database
  networks:
    - test_network

networks:
  test_network:

```

- aws에 위 컴포즈 파일을 저장할 디렉토리 생성

```
~$ sudo mkdir doc-app3
```

- 윈도우 콘솔에서 다음 명령으로 생성한 doc-app3 디렉토리에 컴포즈 파일 업로드

```
C:\Users\USER>scp -i Downloads\ubuntuservkey.pem C:\Users\kingn\workspace\doc-app3\docker-compose.yml ubuntu@43.203.128.242:~/doc-app3
```

2) 사용할 도커 username / password / aws 키값을 변수로 저장

- 깃 허브 상위 메뉴 settings -> 왼쪽의 Security 항목 중 Secrets and variables -> Actions 선택

- New repository secret 버튼 클릭

Name: DOCKERHUB_USERNAME

Secret: 도커유저네임입력

Name: DOCKERHUB_PASSWORD

Secret: 도커유저 패스워드 입력

Name: PRIVATE_KEY

Secret: aws ec2의 pem키값. 이 파일을 메모장에서 열어 그 내용 복사하여 넣는다.

3) 상위 Actions 메뉴 클릭하고

choose a workflow에서 검색창에 maven 입력

하위 위젯 중 Java with Maven 의 Configure 버튼 클릭

- 생성된 maven.yml 파일에 다음과 같이 작성

```
# This workflow will build a Java project with Maven, and cache/restore any dependencies to
improve the workflow execution time
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven
```

```
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.
```

```
name: Java CI with Maven
```

```
on:
```

```
  push:
```

```
    branches: [ "main" ]
```

```
  pull_request:
```

```
    branches: [ "main" ]
```

```
jobs:
```

```
  build:
```

```
    runs-on: ubuntu-latest
```

```
    steps:
```

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

```
      - name: Set up JDK 17
```

```
        uses: actions/setup-java@v4
```

```
        with:
```

```
          java-version: '17'
```

```
    distribution: 'temurin'
    cache: maven
  - name: Setup MySQL
    uses: mirromutth/mysql-action@v1.1
    with:
      host port: 3306
      container port: 3306
      mysql database: 'world'
      mysql user: 'test'
      mysql password: '1234'
  - name: Build with Maven
    env:
      SPRING_DATASOURCE_URL: jdbc:mysql://localhost:3306/world
      SPRING_DATASOURCE_USERNAME: 'test'
      SPRING_DATASOURCE_PASSWORD: '1234'
    run: mvn -B package --file doc-app3/pom.xml && cp doc-app3/target/*.war doc-app3
    #run: mvn -Dmaven.test.skip=true -B package --file doc-app3/pom.xml && cp doc-
app3/target/*.war doc-app3

# Docker 이미지 빌드
- name: docker image build
  run: cd doc-app3 && docker build -t kingnuna/doc-app3:v2 .

# DockerHub 로그인
- name: docker login
  uses: docker/login-action@v2
  with:
    username: ${ secrets.DOCKERHUB_USERNAME }
    password: ${ secrets.DOCKERHUB_PASSWORD }

# Docker Hub 이미지 푸시
- name: docker Hub push
  run: docker push kingnuna/doc-app3:v2

# aws에 접속하여 도커허브에서 이미지 다운받아 실행
- name: Deploy
  uses: appleboy/ssh-action@master
  with:
    host: 54.180.127.89      #aws ip
```

```
port: 22          #ssh 포트
username: ubuntu  #ec2 사용자. aws에서 whoami 명령 실행으로 확인. 명령라인
맨 앞에서 써있음
key: ${ secrets.PRIVATE_KEY } # pem 키. ec2 생성시 public 키 생성하여 다운로드 받은
~.pem 파일을 메모장으로 열어 그 안의 모든 내용 복사해서 변수 값으로 등록
# 도커 작업
script: |
  sudo docker container stop doc-app3 #컨테이너 종료
  sudo docker container rm doc-app3  #컨테이너 삭제
  sudo docker rmi kingnuna/doc-app3:v2 #이미지 삭제
  sudo docker pull kingnuna/doc-app3:v2 # 이미지 다운로드
  cd doc-app3
  sudo docker-compose up
```

- 작성이 완료되면 오른쪽 상단의 commit changes.. 녹색 버튼 클릭

4) Actions의 workflow에서 작업상황 확인

예러나면 maven.yml 파일 수정

=> 프로젝트의 새 내용이 깃허브로 push될 때마다(main만) 통합 및 빌드, 테스트, 도커 허브로 자동 업로드, aws 접속하여 도커 허브에서 이미지 다운로드 및 실행까지 자동 처리된다.