

배포 명령어 정리

EC2 Docker 환경 세팅 및 Jenkins 설치

Docker 설치(Ubuntu 20.04 LTS)

옛 버전 Docker 삭제

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

Repository 설정

- Repository를 이용하기 위한 패키지 설치

```
# 1. apt 패키지 매니저 업데이트
$ sudo apt-get update

# 2. 패키지 설치
$ sudo apt-get install ca-certificates curl gnupg lsb-release

# 3. Docker 공식 GPG Key 등록
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# 4. Stable Repository 등록
$ echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Docker 엔진 설치

```
# 패키지 매니저 최신화
$ sudo apt-get update
$ sudo apt-get upgrade

# Docker 엔진 설치
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Jenkins 설치 & 세팅

Jenkins 이미지 다운로드

- jenkins 공식 도커 이미지를 최신버전으로 로컬에 다운로드 받기

```
$ sudo docker pull jenkins/jenkins:ls
```

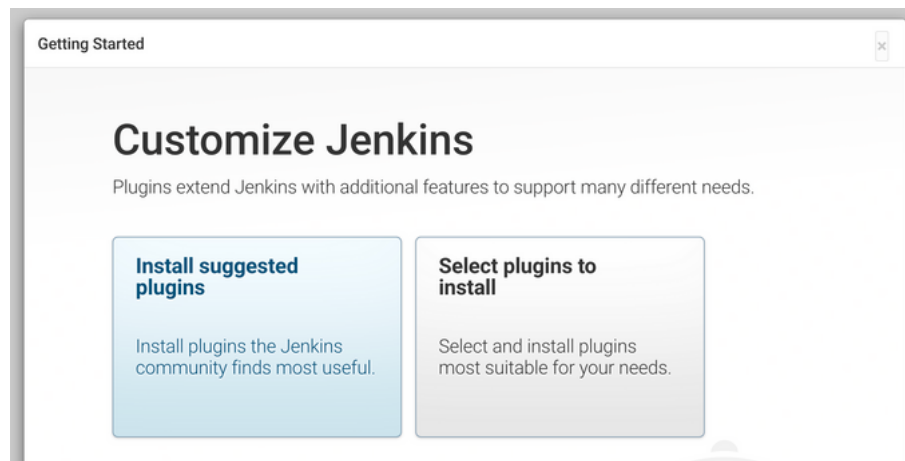
Jenkins 컨테이너 실행

- 다음 명령을 통해 다운로드 받은 젠킨스 이미지를 컨테이너로 띄울 수 있음

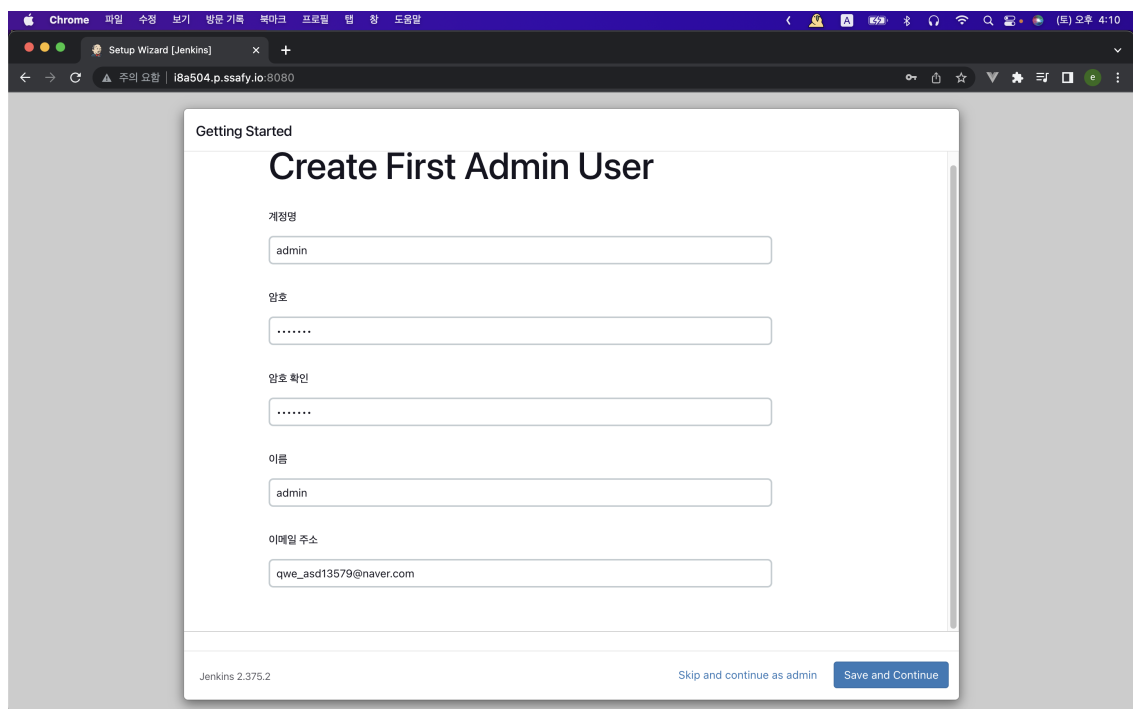
```
$ sudo docker run -d -p 8080:8080 --name jenkins-server --restart=on-failure -v jenkins_home:/var/jenkins_home -u root jenkins/jenkins
```

- d** : 컨테이너를 데몬으로 띄웁니다.
- p 8080:8080** : 컨테이너 외부와 내부 포트를 포워딩합니다. 좌측이 호스트 포트, 우측이 컨테이너 포트입니다.
- v /jenkins:/var/jenkins_home** : 도커 컨테이너의 데이터는 컨테이너가 종료되면 휘발됩니다. 도커 컨테이너의 데이터를 보존하기 위한 여러 방법이 존재하는데, 그 중 한 방법이 볼륨 마운트입니다. 이 옵션을 사용하여 젠킨스 컨테이너의 `/var/jenkins_home` 이라는 디렉토리를 호스트의 `/jenkins` 와 마운트하고 데이터를 보존할 수 있습니다.
- name jenkins** : 도커 컨테이너의 이름을 설정합니다.

2. Install suggested plugins(권장사항 설치) 클릭



3. 어드민 계정 생성



4. 접속 포트 설정

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps. The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.361.2

Not now

Save and Finish

설치완료

EC2에 MariaDB 설치 & GitLab 연결

EC2 운영체제

```
ubuntu@ip-172-26-5-168:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04 LTS"
```

Ubuntu 20.04 LTS

1. Linux 계열 3가지 : redhat (CentOS, redhat enterprise 등), devian, ubuntu
2. Ubuntu에서 **apt(devian, ubuntu)**가 **yum(redhat)**과 동일
3. **sudo**는 root 권한으로 실행하라는 의미

• MariaDB 서버 설치

```
$ sudo apt install mariadb-server
$ sudo apt install mariadb-client
# mariadb-client : 클라이언트 모듈은 MariaDB에 접속하기 위한 다양한 도구가 포함
```

• 외부 접속 허용하기

```
# MariaDB 설정 경로로 이동
$ cd /etc/mysql/mariadb.conf.d/

# server 설정 파일을 문서 편집기로 엮
$ sudo vi 50-server.cnf
# bind-address 항목을 0.0.0.0으로 변경. (# 주석 처리도 가능함)

# 설정 파일 리로드를 위해 서비스를 재시작
$ sudo service mariadb restart
```

- MariaDB 계정 생성

```
# MariaDB에 접속
sudo mariadb

# (dbname)으로 데이터베이스 생성
CREATE DATABASE (dbname);

# (userID) 계정을 (password) 비밀번호로 모든 IP주소('%')를 통하여 접근 가능하게 생성
CREATE USER '(userID)'@'%' IDENTIFIED BY '(password)'
# ex) create user 'admin'@'%' identified by 'danim1234';

# (dbname) 아래 있는 전체 테이블(*)에 대하여 생성된 (userID) 계정에 전체 권한을 줌
GRANT ALL PRIVILEGES ON (dbname).* TO '(userID)'@'%'
# ex) grant all privileges on *.* to 'admin'@'%'

# 현재 설정된 유저 권한을 적용 함
FLUSH PRIVILEGES;

# MariaDB 접속 종료
EXIT;
```

- 권한 확인

```
MariaDB [(none)]> show grants for 'admin'@'%';
+-----+
| Grants for admin@% |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'admin'@'%' IDENTIFIED BY PASSWORD '*CE7D2A8815774A3933110D6FB8A71E8E5AC341F2' |
+-----+
1 row in set (0.000 sec)
```

MariaDB 기본 동작 확인

MariaDB의 기본 동작을 확인하기 위해서 터미널에서 mysql (또는 mariadb)를 실행하고 MariaDB console에서 명령어를 확인합니다. Root 계정의 패스워드를 설정하지 않아서 실제 계정의 권한을 그대로 MariaDB에서 동일하게 인식됩니다. 권한 문제를 피하기 위해서 터미널에서 sudo 계정을 실행합니다. MariaDB에서 명령어 끝에 ; (세미콜론)을 붙이고 엔터치면 실행 결과를 출력합니다.

```
$ sudo mysql 또는 $ sudo mariadb
```

Maria DB 콘솔에서 명령어를 실행합니다.

```
# 현재 database를 확인합니다.
```

```
MariaDB [(none)]> show databases;
```

```
# 설치된 Maria DB 버전을 확인합니다.
```

```
MariaDB [(none)]> select version();
```

```
# db 선택
```

```
MariaDB [(none)]> use mysql;
```

```
# mysql db 안에 있는 tables을 보여 달라
```

```
MariaDB [(mysql)]> show tables;
```

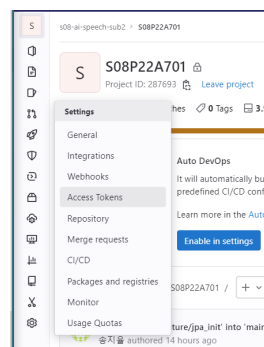
```
# sql 구문 사용
```

```
MariaDB [(mysql)]> select * from OOO where **** ;
```

GitLab - Jenkins 연동 및 Pipeline 구축

순서 :




1. **GitLab Access Tokens** 생성 (**Maintainer** 권한 있어야함) 및 토큰 값 저장해야함.




2. Dashboard > Jenkins 관리 > Manage Credentials > Add credentials 에서 **2-a, 2-b** 생성
 - a. Add credentials(**GitLab API token**)에서 **Access Tokens**를 사용하여 Credentials 생성
 - b. Add credentials(**Username with password**) **gitlab 계정**을 이용하여 Credentials 생성
 - i. kind : username password


- ii. username : **gitlab 계정 id**
- iii. Password : **gitlab 계정 pw**
- iv. ID, Description 생략 가능..

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	0a773872-e15b-4dcb-b7b9-4abe6a35cef3	GitLab API token
		System	(global)	db9ece91-bdb9-4801-bc07-076bcfef2d8b	2hojune@naver.com/*****

Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

 Add credentials

3. Dashboard > Jenkins 관리 > 시스템 설정 > 밑으로 스크롤 → **Gitlab** 부분에서
 - a. **Connection name** : 프로젝트 명(마음대로 설정)
 - b. **Gitlab host URL** : <https://lab.ssafy.com> (프로젝트 주소가 아님, **Gitlab 서버주소**)
 - c. **Credentials** : 2-a에서 생성한 **Credentials** 선택
 - d. 하단의 Test Connection 클릭하여 연결 여부 확인 (Success 출력 확인)

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

A name for the connection

danim

Gitlab host URL

The complete URL to the Gitlab server (e.g. <http://gitlab.mydomain.com>)

<https://lab.ssafy.com>

Credentials

API Token for accessing Gitlab

GitLab API token

+ Add

고급 ▾

Success

Test Connection

4. Dashboard > + 새로운 item > Pipeline 선택

5. Build Triggers에서 **Build when a change is pushed to GitLab. GitLab webhook URL** 체크박스 선택

a. **GitLab webhook URL** 저장 (ex. <http://j8a701.p.ssafy.io:8080/project/First-Danim-Project>)

b. 스크롤 후 고급 클릭 후, Generate 버튼 클릭

c. 생성된 **Secret token** 저장 (ex. 9d050224f3a95180a2f73511f56b937f)

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://j8a701.p.ssafy.io:8080/project/First-Danim-Project> ?

Enabled GitLab triggers

☒ Push Events

☐ Push Events in case of branch delete

☒ Opened Merge Request Events

☐ Build only if new commits were pushed to Merge Request ?

☐ Accepted Merge Request Events

☐ Closed Merge Request Events

Rebuild open Merge Requests

Never

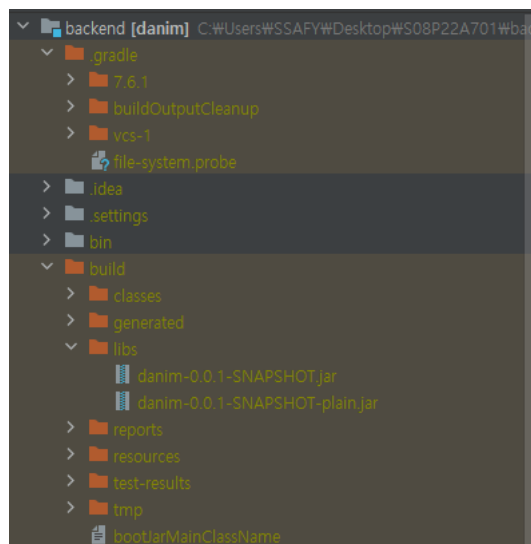
6. GitLab으로 돌아와서, Webhooks 클릭
7. 저장한 **GitLab webhook URL, Secret token**을 입력하고 Trigger의 Push events에 브랜치 명을 입력 (ex. main). Add webhook 버튼 클릭
 - a. 이후 Pipeline script를 클릭 시 script 직접 입력
 - b. Pipeline script from SCM 클릭 시, Pipeline script가 Jenkinsfile로 자동 생성되는듯..?
 - i. SCM : Git
 - ii. Repository URL : 프로젝트 주소 입력 (ex. <https://lab.ssafy.com/s08-ai-speech-sub2/S08P22A701>)
 - iii. Credentials : **2-b**에서 생성한 **Credentials** 선택
8. Apply > 저장 클릭 후 빌드 테스트

SpringBoot Dockerfile 생성

Gradle 빌드 결과물 경로 확인

상단 메뉴 **view** → **Tool Windows** → **Gradle**

danim → **Tasks** → **build** → **build** 더블클릭



Spring Dockerfile

```
# openjdk-11-jdk 가져오기
FROM openjdk:11-jdk

# 5000번 포트 노출
ENV PORT 5000
EXPOSE $PORT

# 빌드후 생성된 jar 파일을 컨테이너 내부에 복사
COPY /build/libs/danim-0.0.1-SNAPSHOT.jar

# jar 파일 실행(= BE 서버 실행)
ENTRYPOINT ["java", "-jar", "app.jar"]
```

FLASK Dockerfile

```
FROM python:3.8-slim

COPY . /app
WORKDIR /app
```

```

RUN echo server will be running on 4000

RUN pip3 install flask
RUN pip3 install numpy
RUN pip3 install pydub
RUN pip3 install requests
RUN pip3 install response
RUN pip3 install requests_toolbelt

RUN apt-get update && \
  apt-get install -y ffmpeg && \
  apt-get clean && \
  rm -rf /var/lib/apt/lists/*

CMD ["python3", "main.py"]

```

SpringBoot 원격 서버 & Maria DB 설정

<DB 접속 정보>

- host: j8a701.p.ssafy.io
- username: admin
- password: danim1234
- port: 3306

설정값

application.yml

```

spring:
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://j8a701.p.ssafy.io:3306/Danim
    username: admin
    password: danim1234

  jpa:
    open-in-view: true
    hibernate:
      ddl-auto: create # 실행할 때마다 스키마 초기화
    properties:
      hibernate:
        format_sql: true # hibernate가 생성한 쿼리를 정리해준다
      jdbc:
        lob:
          non_contextual_creation: true
        use-new-id-generator-mapping: false

```

```

spring:
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://j8a701.p.ssafy.io:3306/Danim
    username: admin
    password: danim1234

  jpa:
    open-in-view: true
    hibernate:
      ddl-auto: create
    properties:
      hibernate:
        format_sql: true
      jdbc:
        lob:
          non_contextual_creation: true
        use-new-id-generator-mapping: false

```

\$ cd /var/jenkins_home/workspace/First-Danim-Project/backend/src/main/resources/application.yml

<접속 에러시 확인 사항>

1. DB 가동여부 확인
2. MariaDB 접속 계정 ID&PW 일치여부, 권한 설정 확인
3. MariaDB 접속 가능 호스트 ip 확인(in 원격 서버)
 - 위치 : vi /etc/mysql/mariadb.conf.d/50-server.cnf
 - 설정

```

bind-address            = 127.0.0.1

위 부분 주석처리 or 0.0.0.0 으로 변경

```

4. 3306 포트 방화벽 허용 설정(*ufw*)

```
sudo ufw allow 3306
```

5. *JDBC 커넥터* 버전 확인
6. *application.yml* 확인

```

server:
  port: 80
spring:
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://j8a701.p.ssafy.io:3306/Danim
    username: admin
    password: danim1234

  jpa:
    open-in-view: true
    hibernate:
      ddl-auto: create
    properties:
      hibernate:
        format_sql: true
      jdbc:
        lob:
          non_contextual_creation: true
        use-new-id-generator-mapping: false

```

7. SpringBoot - MariaDB dependency 추가 확인 (**build.gradle** 파일)

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '2.7.9'
    id 'io.spring.dependency-management' version '1.0.15.RELEASE'
}

group = 'com.danim'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    // https://mvnrepository.com/artifact/org.json/json ,카카오 로그인시 jsonobject사용을 위해 추가
    implementation group: 'org.json', name: 'json', version: '20230227'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    testImplementation 'org.springframework.security:spring-security-test'
}

tasks.named('test') {
    useJUnitPlatform()
}
```

EC2 서버 배포

과정

- Dashboard > Pipeline item 추가 및 구성
- Pipelinescript에 의해 jenkins-server에 .jar 파일 생성
- EC2 상에서 backend-server를 컨테이너로 생성하고 싶으므로, jar 파일을 EC2로 전달
- .jar 파일을 구동하기 위한 EC2 dockerfile 생성
- Pipeline script에 의해 danim-backend-server 이미지 및 컨테이너 생성

Dashboard > Pipeline item 추가 및 구성

Dashboard > 새로운 Item > Pipeline 선택

Build Triggers

> ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://i8a701.p.ssafy.io:8080/project/First-Danim-Project>
선택

>> 고급 클릭 후 **Generate** 버튼 클릭 (Secret token 저장)

빌드된 Jar 파일을 EC2 서버로 전달

GitLab Project Clone, Build Gradle Project가 성공적으로 수행됐다면,
컨테이너 (jenkins-server)의 build/libs 경로에 실행 가능한 .jar 파일이 생성됨
경로 : /var/jenkins_home/workspace/First-Danim-Project/backend/build/libs

Jenkins Publish Over SSH 플러그인 설치

Dashboard > Jenkins 관리 > 시스템 설정 > Publish over SSH 탭으로 이동

Key 텍스트 필드에 aws pem 키를 복사해서 붙여넣어준다.



하단의 SSH Servers 추가 버튼을 클릭해 내용을 작성

SSH Servers

SSH Server
✕

Name ?

Hostname ?

Username ?

Remote Directory ?

고급 ▼

- **Name** : 해당 설정의 이름을 지정
- **Hostname** : SSH를 이용해 접속할 도메인 이름 (클라우드 공용 IP 주소 : 서버 도메인 주소)
- **Username** : SSH 접속시 필요한 유저 이름
- **Remote Directory** : .Jar 파일 전송 시 Root 경로로 잡히는 경로

모든 정보를 입력한 후 Test Configuration을 누르면 Success 와 함께 제대로 연동됨을 확인

EC2 상에 dockerfile 생성

```
ubuntu@ip-172-26-5-168:~$ vi backend/dockerfile

# openjdk-11-jdk 가져오기
FROM openjdk:11-jdk

WORKDIR /home/ubuntu/backend

# 빌드 후 생성된 jar 파일을 컨테이너 내부에 복사
COPY danim-0.0.1-SNAPSHOT.jar .

# jar 파일 실행
CMD java -jar danim-0.0.1-SNAPSHOT.jar
```

Pipeline Script 작성하기

생성했던 Pipeline 아이템 > Pipeline 항목 아래의 [Pipeline Syntax](#) 클릭

Pipeline

Definition

Pipeline script

Script ?

```
1 pipeline {  
2   agent any  
3  
4   stages {  
5     stage('Catch Webhook') {  
6       steps {  
7         echo 'Webhook executed!'  
8       }  
9     }  
10    stage('GitLab Project Clone') {  
11      steps {  
12        git branch: 'main', credentialsId: 'db9ece91-bdb9-  
13      }  
14    }  
15    stage('Build Gradle Project') {  
16      steps {  
17        sh '''  
18
```

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

Snippet Generator의 Steps에서 **sshPublisher: Send build artifacts over SSH**를 선택

Sample Step

sshPublisher: Send build artifacts over SSH

sshPublisher ?

SSH Publishers

SSH Server

Name ?

danim-dev

고급

Transfers

Transfer Set

Source files ?

build/libs/*.jar

Remove prefix ?

build/libs

Remote directory ?

/backend

Exec command ?

- **Name** : 이전 Publish Over ssh에서 새로 추가한 서버를 선택한다.
- **Source files** : gradlew bootJar 명령어로 빌드한 Jar 실행파일이 저장된 경로를 작성한다. bootJar 명령어로 build 시 build/libs 경로에 파일이 생성된다.
- **Remove prefix** : Source files 경로의 하위폴더를 지운다. 하위 폴더를 지움으로 써 *.jar 파일만 전송할 수 있게 해준다.
- **Remote Directory** : 젠킨스 서버가 전달하는 Jar 파일을 저장할 디렉토리 (클라우드 내 사용자가 직접 생성한 디렉토리)
- **Exec command** : 파일 전송이후 해당 경로의 쉘 스크립트를 실행해 작성한 명령어를 실행할 수 있다.

작성한 후 아래 **Generate**를 통해 pipeline script를 복사한 후 Pipeline에 새로운 Stage를 만들고 steps에 붙여넣음.

최종 Pipeline script

```
pipeline {
    agent any

    stages {
        stage('Catch Webhook') {
            steps {
                echo 'Webhook executed!'
            }
        }
    }
}
```



```

    }
  }
  stage('GitLab Project Clone') {
    steps {
      git branch: 'main', credentialsId: 'db9ece91-bdb9-4801-bc07-076bcfef2d8b', url: 'https://lab.ssaify.com/s08-ai-speech-s
    }
  }
  stage('Build Gradle Project') {
    steps {
      sh '''
        rm -rf First-Danim-Project
        rm -rf First-Danim-Project@tmp
        cd backend;
        chmod +x gradlew;
        ./gradlew clean build;
        '''
    }
  }
  stage('Jenkins To EC2 delivery *.jar file') {
    steps {
      sshPublisher(
        publishers: [
          sshPublisherDesc(
            configName: 'danim-dev',
            transfers: [
              sshTransfer(
                cleanRemote: false,
                excludes: '',
                execCommand: '''
                  sudo docker stop danim-backend-server || true;
                  sudo docker rmi popurach/danim-backend || true;
                  sudo docker build -t popurach/danim-backend ./backend;
                  sudo docker push popurach/danim-backend;
                  sudo docker run --rm --privileged --name danim-backend-server -itd -p 80:80 popurach/danim-bac
                  sudo docker container prune -f;
                  sudo docker image prune -f;
                ''',
                execTimeout: 120000,
                flatten: false,
                makeEmptyDirs: false,
                noDefaultExcludes: false,
                patternSeparator: '[, ]+',
                remoteDirectory: '/backend',
                remoteDirectorySDF: false,
                removePrefix: 'backend/build/libs',
                sourceFiles: 'backend/build/libs/danim-0.0.1-SNAPSHOT.jar'
              )
            ],
            usePromotionTimestamp: false,
            useWorkspaceInPromotion: false,
            verbose: false
          )
        ]
      )
    }
  }
}

```

execCommand 설명

```
# 현재 실행되고 있는 danim-backend-server 컨테이너가 있다면 중지
sudo docker stop danim-backend-server || true;

# 현재 popurach/danim-backend 이미지가 있다면 삭제
sudo docker rmi popurach/danim-backend || true;

# ec2 상의 ./backend 경로에 있는 dockerfile을 이용하여 이미지를 build
# 이미지의 이름은 popurach/danim-backend라고 태깅
sudo docker build -t popurach/danim-backend ./backend;

# docker hub에 push (백업)
# (ec2 shell에서 docker hub에 로그인 docker login 선행 필요)
sudo docker push popurach/danim-backend;

# popurach/danim-backend 이미지를 사용하여 danim-backend-server 컨테이너 실행
sudo docker run --rm --privileged --name danim-backend-server -itd -p 80:80 popurach/danim-backend;

# dangling 컨테이너, 이미지 삭제
sudo docker container prune -f;
sudo docker image prune -f;
```

dangling 이미지, 컨테이너 삭제

```
# 중지된 모든 컨테이너 삭제
$ docker container prune

# 태깅이 되지 않은 모든 이미지(dangling image) 삭제
$ docker image prune

# 사용되지 않는 도커 네트워크 삭제
$ docker network prune

# 컨테이너에서 사용하지 않는 모든 도커 볼륨 삭제
$ docker volume prune

# 모든 리소스 (컨테이너, 이미지, 네트워크 등) 중 사용하지 않는 리소스 삭제
$ docker system prune -a
```

- -a 명령을 사용하면 dangling 이미지 이외에도 사용하지 않는 이미지까지 삭제하므로, 주의한다.

도커 로그 관리

EC2 상의 도커 컨테이너들은 다양한 방법으로 로그를 남기는데, 시간이 지나면서 **감당할 수 없는 로그가 쌓이게 됨**. 따라서 주기적으로 **로그를 관리**해줘야 함.

도커 로그 위치 및 용량 확인

```
# sudo 권한 필요
ubuntu@ip-172-26-5-168:~$ sudo su

# 도커 로그 디렉토리
ubuntu@ip-172-26-5-168:~$ cd /var/lib/docker/containers
root@ip-172-26-5-168:~$ ls

# $docker ps -> wordpress id와 동일한 것이 해당 컨테이너의 로그 디렉토리
# 젠킨스 컨테이너 로그 디렉토리
cd7af97652995b1ca556b37ae31da7d06450996ca71b952ea5a68c3ca6857e7f
# 스프링 컨테이너 로그 디렉토리
d96ac3e68f785fd0296635f6e45805594676ca8ebc12cf1037ae0dfd4c4fc21d

# container 용량 확인
root@ip-172-26-5-168:~$ du -hs *
284K    cd7af97652995b1ca556b37ae31da7d06450996ca71b952ea5a68c3ca6857e7f
64K     d96ac3e68f785fd0296635f6e45805594676ca8ebc12cf1037ae0dfd4c4fc21d
```

logrotate 설정하기

```
# etc/logrotate.d/docker-container 파일 생성
root@ip-172-26-5-168:~$ vi etc/logrotate.d/docker-container

# etc/logrotate.d/docker-container 파일 내용
/var/lib/docker/containers/*/*.log {
    rotate 7
    daily
    compress
    size=1M
    missingok
    dailycompress
    copytruncate
}
```

rotate 7 : 백업하는 개수. 최대 7개의 파일을 보관 (log.1, log.2, ... , log.7)

daily : 로그 회전 주기 (yearly : 매 년, monthly : 매 월, weekly : 매 주, daily : 매일)

compress : 로그 파일 압축

size=1M : 크기가 1메가 바이트를 넘으면 실행

missingok : 로그 파일이 없는 경우 에러 없이 다음 rotate 실행

delaycompress : 로그 압축 시 백업한 로그를 재백업할 때 압축

copytruncate : log.1 처럼 log를 나눠서 실행