

# Memory After Service

## 포팅 매뉴얼

삼성청년SW아카데미 대전캠퍼스 7기

특화 프로젝트 빅데이터(분산)

B103(기억저장관리팀)

2022. 08. 22. ~ 2022. 10. 07.

강민성 김구연 김대원 이정건 최수연



# Environment

## 형상 관리

- Gitlab

## 이슈 관리

- Jira

## Communication

- Mattermost
- Webex
- Notion

## OS

- Windows 10

## UI/UX

- Figma

## 기타 편의 툴

- Postman 9.31.9
- Termius 7.48.0
- Bash shell script

## IDE

- IntelliJ (2022.1.3)
- Visual Studio Code (1.69.X)
- Android Studio Chipmunk(2021.2.1)

## Server

- AWS EC2 Instance
  - Ubuntu 20.04 LTS
- Hadoop Cluster

## Deployment

- Docker 20.10.18
- Jenkins 2.361.1

## DataBase

- MySQL (8.0.29)

## Mobile Application

- Java Open-JDK zulu 8.33.0.1
- Gradle 7.2.2
- Android sdk minimum : 26(8.0)
- PhilJay/MPAndroidChart(v3.1.0)

## Rest API Server

- Java Open-JDK zulu 8.33.0.1
- SpringBoot Gradle 2.7.2
  - Spring Data JPA
  - Lombok
  - Swagger 2.9.2

## Distributed processing

## Open API & External Library

- Hadoop Cluster
- PySpark 3.2.1-amzn-0
- Airflow 2.4.0

#### **Open API**

- NAVER CLOUD Simple & Easy Notification Service
- kakao login API

#### **Python Package**

- KoNLPy



# Amazon AWS

## Amazon Linux 2(Hadoop Cluster)

### [특화 프로젝트] 분산 클러스터(공용) 사용 안내

- 빅데이터(분산) 프로젝트 활용을 위해 하둡/스파크 공용 클러스터 제공
- 제공된 키(pem 파일)로 분산 클러스터 공용 서버에 접속하여 사용 가능
- 서버 접속

ex) `ssh -i J7B103T.pem j7b103@cluster.ssafy.io`

- 분산 클러스터는 공용 환경으로 `sudo` 권한은 제공되지 않음
- 하둡/스파크 구동 및 연산 용도로만 사용 가능
- 개방된 포트 : 22(TCP), 8890(TCP)
  - 필요한 포트가 있을 경우 포트 정보와 사용 목적을 실습코치에게 요청

- 기본 커맨드

```
# 사용자 홈 디렉토리 생성
$ hdfs dfs -mkdir -p .

# 다른 사용자가 내 디렉토리에 접근하지 못하도록 접근 권한 수정
$ hdfs dfs -chmod 0700 .

# 테스트 파일 생성
$ echo 'Hello SSAFY!' > a.txt

# DFS 업로드
$ hdfs dfs -put a.txt

# DFS 업로드 확인
$ hdfs dfs -cat a.txt

# MapReduce 예제 "GREP" 실행
$ hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples-3.2.1-amzn-7.jar \
  grep a.txt output 'SSAFY[a-z.!]+'

# MapReduce 실행 결과 확인
$ hdfs dfs -cat output/*

# 테스트 파일 삭제
$ hdfs dfs -rm a.txt
$ hdfs dfs -rm -r output
$ rm a.txt
```

```
# 홈 디렉토리 삭제
$ hdfs dfs -rmdir .
```

# Amazon EC2

## Deployment

**목표: Jenkins를 이용한 Spring Boot REST API 서버 자동 배포 시스템 구축**

1. Gitlab Push Event 발생
2. Jenkins에서 WebHook을 통해 자동으로 빌드를 실행
3. Jenkins에서 프로젝트 내부의 DockerFile를 이용하여 Docker Image 생성(\*.jar)
4. Jenkins에서 SSH 연결을 통해 AWS에 DockerContainer 생성
5. 외부에서 접속 확인(Postman 등)

## 시스템 구축 과정 설명

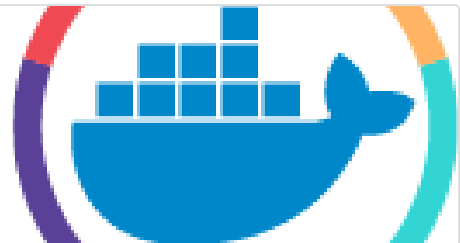
### Docker Install

아래의 링크 참고하여 설치

#### Install Docker Engine on Ubuntu

Estimated reading time: 10 minutes Docker Desktop for Linux Docker Desktop helps you build, share, and run containers easily on Mac and Windows as you do on Linux. We are excited to share that

 <https://docs.docker.com/engine/install/ubuntu/>



### Docker MySQL Image Install

```
$ sudo docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=<password> \
-d -p 3306:3306 mysql:latest
$ sudo docker exec -it mysql-container bash
```


```
bash-4.4# mysql -u root -p
```

```
mysql> create database IF NOT EXISTS `mas_db` collate utf8mb4_general_ci;
mysql> create user 'ssafy'@'%' identified by <password>;
mysql> grant all privileges on mas_db.* to 'ssafy'@'%';
mysql> flush privileges;
mysql> exit
```

```
bash-4.4# mysql -u ssafy -p
mysql> use mas_db;
```

### [Docker] MySQL설치 및 접속하기

Docker에 대한 설명은, 너무나도 훌륭한 글들이 많으므로 생략한다.  
Docker 공식 홈페이지에서 OS 및 환경에 맞는 Docker Desktop을 설치한다. 설치 후, Docker를 실행하면 작업표시줄에 고래선박(?) 아이콘이 생긴다.

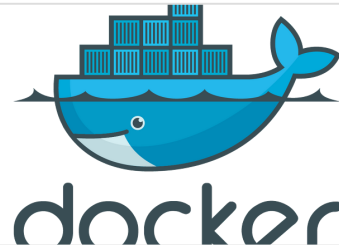
 [https://velog.io/@\\_nine/Docker-MySQL%EC%84%A4%EC%B9%98-%EB%B0%8F-%EC%A0%91%EC%86%8D%ED%95%98%EA%B8%B0](https://velog.io/@_nine/Docker-MySQL%EC%84%A4%EC%B9%98-%EB%B0%8F-%EC%A0%91%EC%86%8D%ED%95%98%EA%B8%B0)



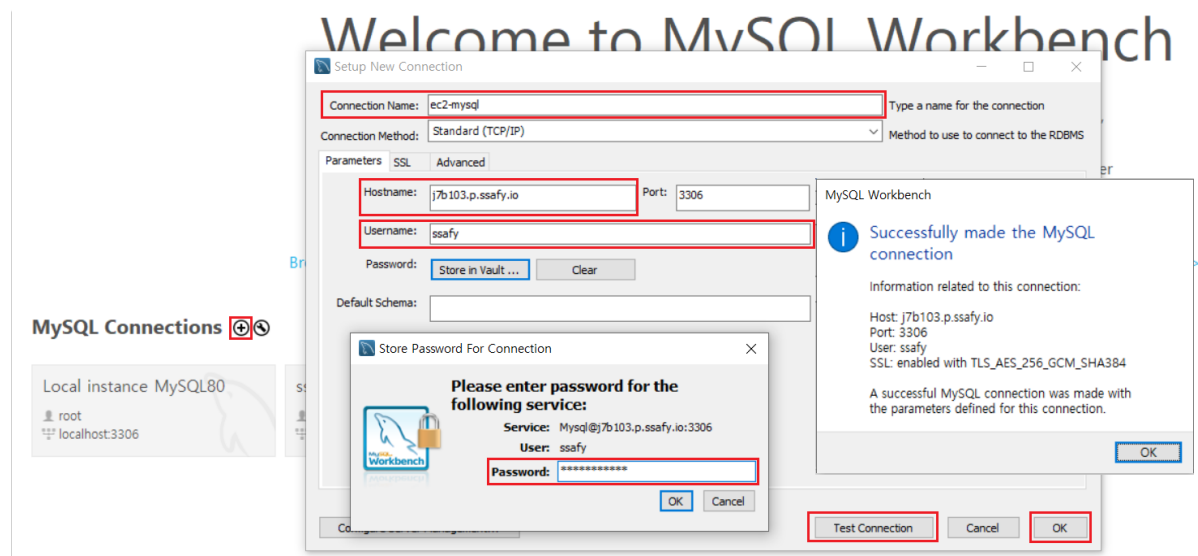
### [Docker] AWS EC2에 Docker로 MySQL 띄우기

MySQL을 클라우드 서버에 올려서 사용하고 싶어 Docker로 간편하게 MySQL 서버를 띄우는 법을 알아보겠다. AWS에 회원가입이 되어 있어야 하며 Docker의 설치가 되어 있어야 한다. Docker의 설치에 다음 포스팅을

 <https://mungiyu.tistory.com/23>



## TIP: MySQL Workbench에서 외부 MySQL DB 연결하기




## Jenkins Docker Container Install

### 1. Docker 이미지로 설치부터 참고

#### Jenkins AWS EC2 설치

Jenkins AWS EC2( Ubuntu20.04) 설치 설치명령어가 업데이트를 하면서 명령어가 다소 바껴질 수 있으니 install은 가이드문서에서 실제 확인하는 게 좋다. 젠킨스 Ubuntu 설치 가이드 문서

 <https://velog.io/@mooh2jj/Jenkins-AWS-EC2-%EC%84%A4%EC%B9%98>



# Jenkins

```
# docker-compose.yml
# 들여쓰기는 탭(tab)이 아닌 공백(space)을 사용해야 함 (tab을 사용할 경우 오류 발생)
```

```

version: '3'

services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins-container
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/jenkins:/var/jenkins_home
      - /usr/bin/docker:/usr/bin/docker
    ports:
      - "9090:8080"
      - "50000:50000"
    privileged: true
    user: root

# Description
# services : 컨테이너 서비스
# jenkins : 서비스 이름
# image : 컨테이너 생성시 사용할 image
# container_name : 컨테이너 이름
# volumes : Host(AWS)와 컨테이너 내부의 데이터를 연결하여 공유
# ports : 포트 매핑, AWS의 9090 포트와 컨테이너의 8080 포트를 연결
# privileged : 컨테이너 시스템의 주요 자원에 연결할 수 있게 함(Default: False)
# user : Jenkins에 접속할 계정(관리자는 root)

```

2. `$ sudo docker-compose up -d` 명령어로 컨테이너를 생성하고 `$ sudo docker ps -a` 로 확인
3. Browser에서 [j7b103.p.ssafy.io:9090](http://j7b103.p.ssafy.io:9090)에 접속

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

4. `$ sudo docker logs jenkins` 명령어로 Administrator password를 찾아 입력 후, `Install suggested plugins`를 클릭하여 기본 플러그인 자동 설치

```
*****
*****

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

a49f2e4a6f2c483c9e01e2f8adb1ae7e

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*****
*****
```

## 5. Jenkins 관리자 계정 생성

## 6. Plug-In Install

Jenkins 관리 > 플러그인 관리 > 설치 가능 탭 선택 후 키워드 검색하여 설치

GitLab 관련

- GitLab
- Generic Webhook Trigger
- Gitlab API
- GitLab Authentication

Docker 관련

- Docker
- Docker Commons
- Docker Pipeline
- Docker API

SSH 관련

- Publish Over SSH

## 7. Jenkins Item 생성

a. 메인 페이지에서 **새로운 Item** 클릭 후 **Freestyle project** 선택

b. Configuration

**Credentials** > **+ Add** > **Jenkins** 클릭



Username with password

Scope ?  
Global (Jenkins, nodes, items, all child items, etc)

Username ?  
pfcskms1997@naver.com

☐ Treat username as secret ?

Password ?  
\*\*\*\*\*

ID ?  
mas

- Username : GitLab 아이디
- Password : GitLab 비밀번호
- ID : Credential을 구별할 문자열(형식 없음)

## Configuration

- General
- 소스 코드 관리**
- 빌드 유발
- 빌드 환경
- Build Steps
- 빌드 후 조치

### 소스 코드 관리

☐ None

☒ Git ?

#### Repositories ?

Repository URL ?  
https://lab.ssfy.com/s07-bigdata-dist-sub2/S07P22B103

Credentials ?  
pfcskms1997@naver.com/\*\*\*\*\*  
- none -  
pfcskms1997@naver.com/\*\*\*\*\*

코딩...

Add Repository

특정 Branch의 소스를 빌드하고 싶을 경우는 아래와 같이 입력

## Configuration

- General
- 소스 코드 관리**
- 빌드 유발
- 빌드 환경
- Build Steps
- 빌드 후 조치

#### Branches to build ?

Branch Specifier (blank for 'any') ?  
\*/BackEnd

Add Branch

Repository browser ?  
gitlab

URL ?  
https://lab.ssfy.com/s07-bigdata-dist-sub2/S07P22B103/-/tree/BackEnd/BE/mas

Version ?  
0.4

- Branches Specifier : 타겟 브랜치 명
- URL : 타겟 프로젝트의 주소(브랜치 포함)
- Version : 프로젝트의 버전(아무 숫자나 입력해도 무관함)

**Configuration**

General

**소스 코드 관리**

빌드 유발

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://7b103.p.ssafy.io:9090/project/mas\_springboot ?

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

☒ Approved Merge Requests (EE-only)

☒ Comments

Comment (regex) for triggering a build ?

Jenkins please retry a build

**고급...**

☐ Generic Webhook Trigger ?

저장 Apply

위와 같이 체크 후 **고급** 버튼을 클릭하면 아래와 같은 화면에서 다음 작업을 진행

**Configuration**

- General
- 소스 코드 관리**
- 빌드 유발
- 빌드 환경
- Build Steps

☒ Enable [ci-skip]

☒ Ignore WIP Merge Requests

Labels that forces builds if they are added (comma-separated)

☒ Set build description to build cause (eg. Merge request or Git Push)

☐ Build on successful pipeline events

Pending build name for pipeline ?

☐ Cancel pending merge request builds on update

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

1c46b1e36048a677515de93685a589db

[Generate](#)

[Clear](#)

[저장](#) [Apply](#)

**Generate** 버튼을 클릭하여 Secret token을 발급 받아 저장(GitLab과 Webhook 연결 시 사용)

**Build**

[Add build step ^](#)

Filter

- Add a new template to all docker clouds
- Build / Publish Docker Image
- Execute Windows batch command
- Execute shell**
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Send files or execute commands over SSH
- Set build status to "pending" on GitHub commit
- Start/Stop Docker Containers

Build 탭에서 **Add build step** 를 클릭하고, **Execute shell** 을 선택하고 아래와 같이 입력 (테스트만 할 때에는 **pwd** 명령어만 입력하여 save하고, 빌드 잘 되는지 Console에서 확인)

```
docker image prune -a --force
mkdir -p /var/jenkins_home/images
cd /var/jenkins_home/workspace/mas_springboot/BE/mas
chmod +x gradlew # gradlew를 사용하여 프로젝트를 빌드하기 위해서는 권한 추가 필요
./gradlew build # Backend 프로젝트를 빌드하여 jar 파일 생성
```

```
docker build -t mas_springboot . # Dockerfile 실행하여 빌드
docker save mas_springboot > /var/jenkins_home/images/mas_springboot.jar

ls /var/jenkins_home/images
```

./gradlew build permission denied 발생은 chmod 명령어로 권한 수정을 하여 해결  
(Troubleshooting-1 참고)

## 8. GitLab Webhook 연결

프로젝트 Repository > Settings > Webhook 클릭

The screenshot shows the GitLab Webhook Settings page for project S07P22B103. The left sidebar contains a menu with 'Webhooks' highlighted. The main content area is titled 'Webhook Settings' and includes a search bar. Under the 'Webhooks' section, there is a description and a list of triggers. The 'URL' field is set to 'http://7b103.p.ssafy.io:9090/project/mas\_springboot/'. The 'Secret token' field is set to '1c46b1e36048a677515de93685a589db'. The 'Trigger' section has 'Push events' and 'Merge request events' selected. The 'SSL verification' checkbox is checked. The 'Add webhook' button is visible. At the bottom, the 'Project Hooks' table shows one hook with the URL 'http://7b103.p.ssafy.io:9090/project/mas\_springboot/' and buttons for 'Test', 'Edit', and 'Delete'.

URL에는 `http://서버도메인:9090/project/jenkins프로젝트이름/` 을 입력

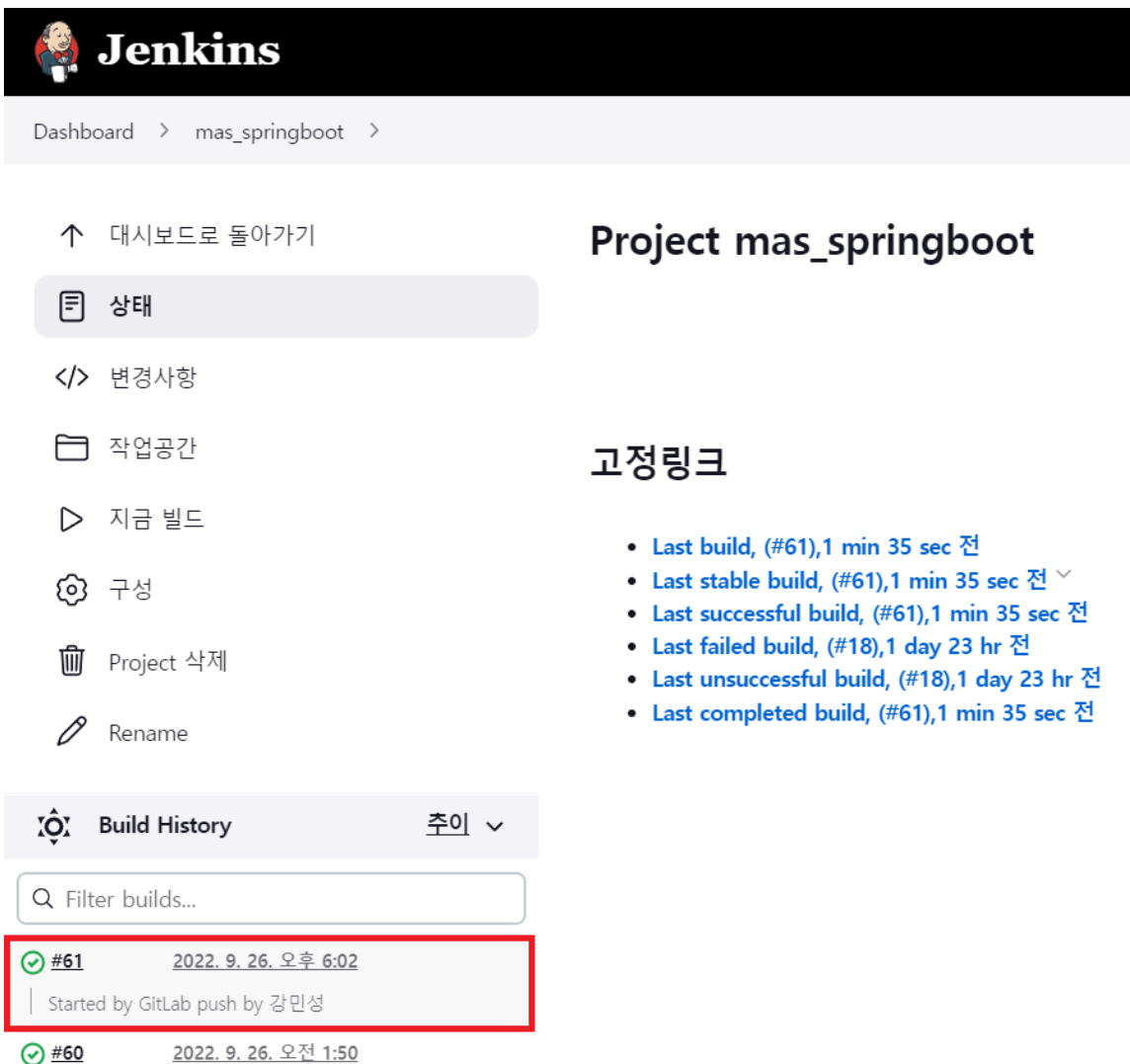
Secret token에는 위에서 발급받은 젠킨스 프로젝트의 secret token 입력

빌드 유발 Trigger로 **Push events** 와 **Merge request events** 를 설정합니다. 대상 Branch는 master으로 설정(테스트 BackEnd로 설정하여 진행함)

Add Webhook 버튼을 눌러 webhook을 생성하고, **Test** 버튼을 클릭하여 **Push events** 선택  
GitLab 화면 상단에 아래와 같이 응답이 나오면 성공

s07-bigdata-dist-sub2 > S07P22B103 > Webhook Settings

Hook executed successfully: HTTP 200



Dashboard > mas\_springboot >

↑ 대시보드로 돌아가기

Project mas\_springboot

상태

변경사항

작업공간

지금 빌드

구성

Project 삭제

Rename

Build History

추이

Filter builds...

✓ #61	2022. 9. 26. 오후 6:02
Started by GitLab push by 강민성	
✓ #60	2022. 9. 26. 오전 1:50

Jenkins에서도 방금의 테스트 결과를 확인 가능

## 9. Jenkins와 연결된 GitLab 프로젝트로 Docker Image 빌드

`$ sudo docker exec -it jenkins-container bash` 로 Jenkins 컨테이너의 bash shell에 접근

정상적으로 접속되면 아래의 명령어로 Docker 설치(EC2 서버에서 설치한 것과 동일한 과정이나 Linux debian 버전이므로 이에 맞도록 수정)

```
# 패키지 설치
$ apt update
$ apt-get install -y ca-certificates \
    curl \
    software-properties-common \
    apt-transport-https \
    gnupg \
    lsb-release

# gpg 키 다운로드
$ mkdir -p /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/
keyrings/docker.gpg

$ echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] ht
tps://download.docker.com/linux/debian \
    $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# Docker 설치
$ apt update
$ apt install docker-ce docker-ce-cli containerd.io docker-compose
```

## 10. Dockerfile 생성 및 Docker Image 생성

프로젝트의 최상단 디렉토리에 Dockerfile을 생성

처음 Dockerfile 작성할 때, 파일 경로를 잘못 적어주어 Docker COPY failed: no source files were specified 에러가 발생하였는데, 해결 방법은 아래의 [Troubleshooting-2](#)을 참고

```
FROM openjdk:8-jdk-alpine
ARG JAR_FILE=./build/libs/mas-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Jenkins에 접속하여 **구성** > **Build** > **Execute shell** 에서 명령어 입력(위의 7-b단계에서 입력하였으므로 생략)

**지금 빌드** 를 클릭하고 Console 가장 아래에 Finished: SUCCESS 확인

Jenkins 설치 시에 volume 설정을 하였으므로 호스트의 **/home/jenkins/images** 와 Jenkins 컨테이너의 **/var/jenkins\_home/images** 에 mas\_springboot.jar 파일이 생성되어 있음을 확인

## 11. 빌드한 Docker Image를 Jenkins에서 SSH 명령어로 컨테이너 생성

### a. Jenkins SSH 연결 설정(Publish over SSH)

**Jenkins 관리** > **시스템 설정** > **Publish over SSH** > SSH Servers의 **추가** 버튼 클릭

Dashboard > Jenkins 관리 > Configure System >

### SSH Servers

SSH Server

Name ?

B103\_EC2

Hostname ?

j7b103.p.ssafy.io

Username ?

ubuntu

Remote Directory ?

고급...

Test Configuration

저장

Apply

- Name : 원하는 이름 입력
- Hostname : EC2 도메인 네임 또는 IP주소 입력
- Username : EC2 유저 네임(ubuntu인 경우 ubuntu 입력)

Dashboard > Jenkins 관리 > Configure System >

☒ Use password authentication, or use a different key ?

Passphrase / Password ?

Concealed

Change Password

Path to key ?

Key ?

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAmMEVIZT18sV7eNCP3Gd9SPfh9QjZiu7YWYjAL7b2wkBvhBv
tRplGdbk8u5Q0v0Mw2CfeYnsByXzbq/6KwuMTuX6NqLAWxxjpqiCiaTHsuZKs9E
```

Use password authentication, or use a different key만 체크

AWS 인증 키(SSAFY에서 제공받은 pem 파일)를 VScode로 열어 복사하여 Key에 붙여 넣기

Proxy port ?

0

Proxy user ?

Proxy password

Concealed

Change Password

Success

Test Configuration

추가

저장

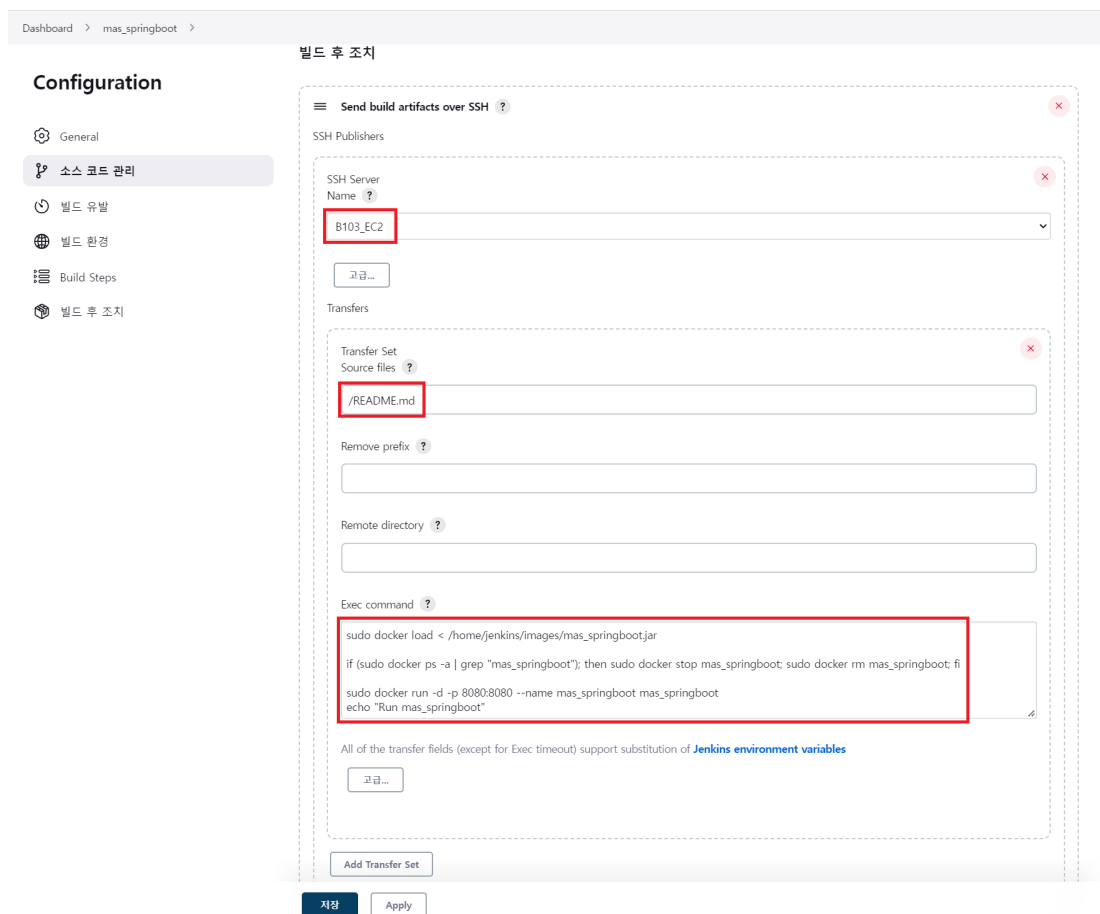
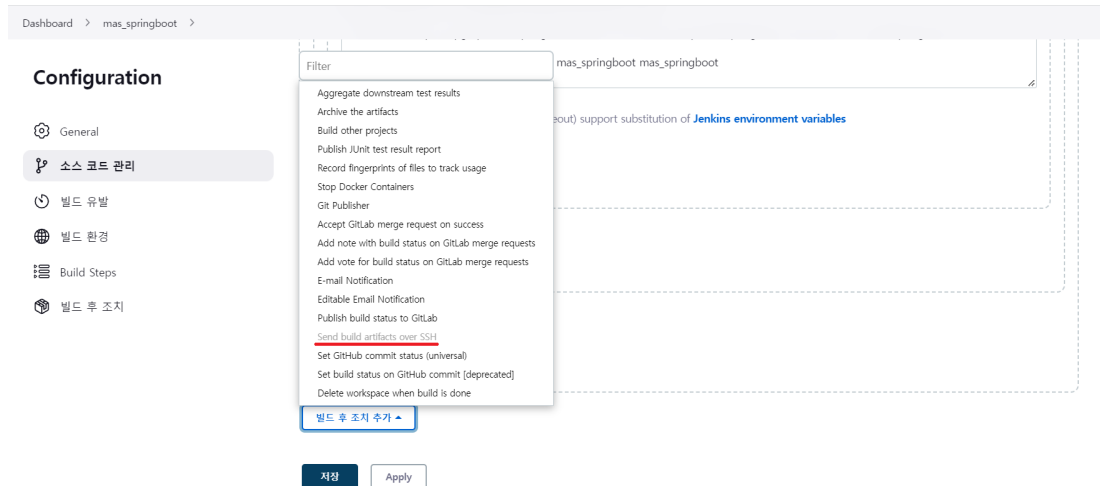
Apply

**Test Configuration** 버튼을 클릭했을 때, Success 메시지가 출력되는지 확인

**Ubuntu 18.xx 보다 높은 버전일 때 pem 키로 인증이 실패하는 경우는 아래의 GitHub CI/CD 매뉴얼 References를 참고**

b. Jenkins 빌드 후 조치로 SSH 명령어 전송(EC2에 Docker Container 생성)

Jenkins에서 **구성 > 빌드 후 조치 > 빌드 후 조치 추가 > Send build artifacts over SSH** 선택





- Name : 원하는 이름 입력
- Source files : 컨테이너에서 aws로 파일을 전송하는 부분. 의미가 없어도 필수 입력 사항이기 때문에 적당한 파일명 입력
- Exec command

```
# jar 파일을 압축 해제하여 docker 이미지로 등록
sudo docker load < /home/jenkins/images/mas_springboot.jar

# 해당하는 이름의 container가 존재하면 실행 중지 후 제거
if (sudo docker ps -a | grep "mas_springboot"); then sudo docker stop mas_springboot; sudo docker rm mas_springboot; fi

# mas_springboot라는 이름으로 container 생성 및 실행
# 8080 포트로 연결
# 전송받은 파일을 host와 공유하고, 이에 접근할 수 있도록 volume 설정
sudo docker run -d -p 8080:8080 -v /home/ubuntu/mas_server:/root --name mas_springboot mas_springboot


# 모든 과정이 정상적으로 완료되면 아래의 문자열 출력
echo "Run mas_springboot"
```


## 12. Build

Jenkins에서 **지금 빌드** 클릭 후 **Build History** 확인

↑ [대시보드로 돌아가기](#)

## Project mas\_springboot

 [상태](#)

 [변경사항](#)

 [작업공간](#)

 [지금 빌드](#)


 [구성](#)

 [Project 삭제](#)

 [Rename](#)

## 고정링크

- [Last build, \(#62\), 1 min 1 sec 전](#)
- [Last stable build, \(#62\), 1 min 1 sec 전](#)
- [Last successful build, \(#62\), 1 min 1 sec 전](#)
- [Last failed build, \(#18\), 2 days 1 hr 전](#)
- [Last unsuccessful build, \(#18\), 2 days 1 hr 전](#)
- [Last completed build, \(#62\), 1 min 1 sec 전](#)

 [Build History](#)

[추이](#) ▼

Q Filter builds...

 [#62](#) [2022. 9. 26. 오후 8:31](#)

 [#61](#) [2022. 9. 26. 오후 6:02](#)

| Started by GitLab push by 강민성

Dashboard > mas\_springboot > #62

```

Setting BUILD_CONTEXT to Docker daemon 42.0.0
Step 1/4 : FROM openjdk:8-jdk-alpine
8-jdk-alpine: Pulling from library/openjdk
Digest: sha256:94792824df2df33402f201713f932b58cb9de94a0cd524164a0f2283343547b3
Status: Downloaded newer image for openjdk:8-jdk-alpine
--> a3562aa0b991
Step 2/4 : ARG JAR_FILE=./build/libs/mas-0.0.1-SNAPSHOT.jar
--> Using cache
--> 8812674b8858
Step 3/4 : COPY ${JAR_FILE} app.jar
--> Using cache
--> d8e9761fb44d
Step 4/4 : ENTRYPOINT ["java","-jar","/app.jar"]
--> Using cache
--> 3dc9aef77414
Successfully built 3dc9aef77414
Successfully tagged mas_springboot:latest
+ docker save mas_springboot
+ ls /var/jenkins_home/images
mas_springboot.jar
SSH: Connecting from host [4e4a0844e6c4]
SSH: Connecting with configuration [B103_EC2] ...
SSH: EXEC: completed after 1,001 ms
SSH: Disconnecting configuration [B103_EC2] ...
SSH: Transferred 0 file(s)
Finished: SUCCESS

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
df3869b9185e	mas_springboot	"java -jar /app.jar"	6 minutes ago	Up 6 minutes	0.0.0.0:8080->8080/tcp, :::8080->8080/tcp	mas_s
pringboot	jenkins/jenkins:lts	"/usr/bin/tini -- /u..."	36 hours ago	Up 17 hours	0.0.0.0:50000->50000/tcp, :::50000->50000/tcp, 0.0.0.0:9090->8080/tcp, :::9090->8080/tcp	jenk1
4eab044e6c4	mysql:latest	"docker-entrypoint.s..."	3 days ago	Up 17 hours	0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp	mysql

Spring Boot 서버 container가 정상적으로 동작하고 있는데, Postman으로 REST API 요청을 보냈을 때, Error: connect ECONNREFUSED 에러가 출력됨(Troubleshooting-3과 Troubleshooting-4 참고)

## 번외: Shell Script(.container-shell.sh) 작성

AWS 서버 재부팅 후 container가 stop 상태일 때, `$ bash .container-shell.sh` 로 container 재 실행

```
#!/bin/sh

sudo docker restart jenkins-container
sudo docker restart mysql-container
sudo docker restart mas_springboot
sudo docker ps -a
```

## Troubleshooting

### 1. ./gradlew build permission denied

Windows에서 파일 생성 시 기본 권한이 644로 설정되기 때문에 이 환경에서 작업해서 소스를 push 할 경우 문제 발생

`$ chmod +x gradlew` 명령어로 실행 권한 추가

gradlew: Permission Denied

Thanks for contributing an answer to Stack Overflow! Please be sure to answer the question. Provide details and share your research! Asking for help, clarification, or responding to other

<https://stackoverflow.com/questions/17668265/gradlew-permission-denied>



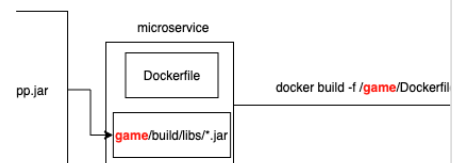
### 2. Docker COPY failed: no source files were specified

Docker COPY failed: no source files were specified 에러

오늘은 aws ecs fargat에 spring boot microservice를 code pipeline을 사용하여 배포하던 중 발생한 에러에 대해서 말해보려고 합니다.

docker 이미지를 build 하던 중, 계속 COPY failed: no source files

🔗 <https://bgpark.tistory.com/132>



### 3. Docker Container log 확인

\$ sudo docker logs <container\_name> 명령어로 로그를 확인

#### 4. [Postman] Error: connect ECONNREFUSED <ip:port>

Windows 로컬 환경에서 테스트 할 때 작성했던 Spring Boot 프로젝트의 application.yml 의 내용을 아래와 같이 수정(server:address: localhost를 제거한다.)

```
# 수정 전
server:
  port: 8080
  address: localhost

# 수정 후
server:
  port: 8080
```

## References

### GitHub - hjs101/CICD\_manual: CICD 매뉴얼

★ : 이 글은 싸피 깃을 이용하여 Docker, Jenkins를 이용하여 서버를 배포한 경험을 바탕으로 작성하였습니다. 글 내용 중 잘못 이해하고 있는 부분이 있을 수 있으니 어느 정도 감안해 주시길 부탁드립니다, 틀린 부분이 있다

[https://github.com/hjs101/CICD\\_manual](https://github.com/hjs101/CICD_manual)

### hjs101/ CICD\_manual

CICD 매뉴얼

Contributor 1 Issues 0 Stars 30 Forks 17

### CI/CD #8. Gitlab Webhook으로 Jenkins 빌드 유발하기

webhook 기술을 통해 Gitlab과 Jenkins Build 유발하기 앞선 포스팅을 통해 gitlab과 jenkins를 연동하고 gitlab저장소에 있는 소스를 기반으로 Maven 빌드하는 과정을 기술했었다. 이번 포스팅은 webhook기술에 대해

https://zunoxi.tistory.com/106



### Jenkins - Gitlab - 트리거된 Branch로 Build하기

Git Parameter 플러그인 설치 후 프로젝트 > 설정으로 들어가서 이 빌드는 매개변수가 있습니다 선택하면 아래 그림처럼 입력필드가 생긴다. 아래 내용 입력 후 저장하면 Build Now 메뉴가 Build with Parameters 메뉴로 변

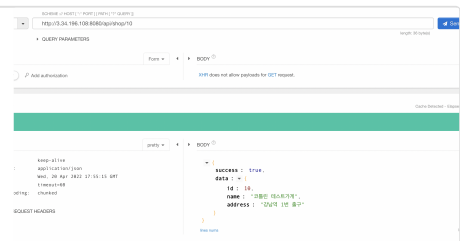
https://newbieloper.tistory.com/6



### [Docker] Spring 프로젝트를 Docker를 이용해서 배포해봅시다

이번 포스트에서는 Docker를 이용해서 저희가 예전에 작성했던 프로젝트를 올리는 방법에 대해서 다룰 예정입니다. 배포 방법이 달라졌따보니 아예 처음부터 설명을 다시 해드리겠습니다. 물론 ec2를 어떻게 만드는지는 설

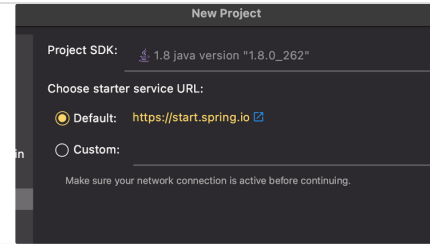
<https://velog.io/@18k7102dy/Docker-Spring-%ED%94%84%E B%A1%9C%EC%A0%9D%ED%8A%B8%EB%A5%BC-Docker%E B%A5%BC-%EC%9D%B4%EC%9A%A9%ED%95%B4%EC%84%9>



C-%EB%B0%B0%ED%8F%AC%ED%95%B4%EB%B4%85%EC%8B%9C%EB%8B%A4 Docker & Jenkins 도커와 젠킨스를 활용한 Spring Boot CI/CD 🤖

요약 : 도커에 Jenkins 띄운 후 Build Now를 클릭해 Github에서 소스코드를 가져와 Spring Boot 프로젝트 빌드, Docker이미지로 생성해 배포 이전 프로젝트에서는 빌드,배포를 쉘 스크립트로 진행했습니다. 현재 프로젝트에서는

📌 [https://velog.io/@hind\\_sight/Docker-Jenkins-%EB%8F%84%EC%B%B4%EC%99%80-%EC%A0%A0%ED%82%A8%EC%8A%A4%EB%A5%BC-%ED%99%9C%EC%9A%A9%ED%95%9C-Spring-Boot-CI](https://velog.io/@hind_sight/Docker-Jenkins-%EB%8F%84%EC%B%B4%EC%99%80-%EC%A0%A0%ED%82%A8%EC%8A%A4%EB%A5%BC-%ED%99%9C%EC%9A%A9%ED%95%9C-Spring-Boot-CI)



## Improvements

### docker-compose로 MySQL 세팅

Docker Compose로 MySQL/MariaDB 세팅하기 - 인하대학교 인트아이

Docker Compose를 이용하면 복잡한 DB 설치와 세팅 과정을 파일 하나로 간략화 할 수 있습니다. Docker와 Docker Compose가 설치되어 있어야 하며, 설치 후 아래와 같이 docker-compose.yml 를 만들어줍니다. docker-compose.yml: 그리고

📌 <https://int-i.github.io/sql/2020-12-31/mysql-docker-compose/>



### 무중단 배포

[SpringBoot, Nginx] 무중단 배포

이전에 스프링부트 프로젝트를 Travis CI를 활용하여 배포 자동화 환경을 구축하였기 때문에 Master 브랜치에 Push만 되면 자동으로 빌드와 테스트 그리고 배포까지 이루어진다. 그러나 배포하는 시간 동안 어플리케이션이

📌 <https://velog.io/@eeheaven/SpringBootNginx-%EB%AC%B4%EC%A4%91%EB%8B%A8-%EB%B0%B0%ED%8F%AC>

velog

Jenkins + CodeDeploy를 이용한 Spring boot 무중단 배포

위 환경은 AWS CodeDeploy + Jenkins를 이용한 Spring boot 자동화 + Jenkins CodeDeploy plugin 오류 해결 에서 설정가능하고 이 작업 이후로 이어진다. 혹시 sudo service nginx restart 를 했을때, 포트번호가 이미 사

📌 <https://velog.io/@seungju0000/Jenkins-CodeDeploy-%EB%A5%B4%EC%9D%B4%EC%9A%A9%ED%95%9C-Spring-boot-%EB%A4%91%EB%8B%A8-%EB%B0%B0%ED%8F%AC>

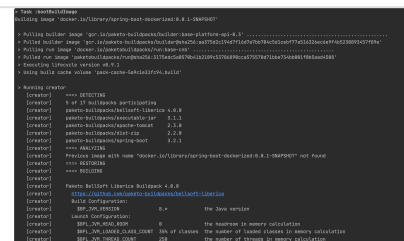


### Docker Image build 최적화

Spring boot docker image만들기 최적화 방법 (+ 새로운 gradle task)

Spring boot를 활용하여서 도커 이미지를 만들 때 단점을 이야기하며 그것에 대한 개선 방법에 대해 이야기합니다. gradle task: gradle 프로젝트의 작업 단위. gradle에서 제공하는 기본 task도 있으며 새로운 task를 만들거나 기존 task를

<https://baeji77.github.io/spring/dev/Spring-boot-gradle-build/>





# NAVER CLOUD PLATFORM - Simple & Easy Notification Service

## 인증키

공공기관용 금융클라우드

로그아웃 Languages

NAVER CLOUD PLATFORM 소개 서비스 솔루션 마켓플레이스 요금 고객지원 파트너 가이드센터 마이페이지

문의하기 콘솔

### 마이페이지

고객님의 서비스 이용내역을 확인하고 관리해 보세요  
나의계정 pfcskms1997@naver.com

서비스 이용 내역

#### 이용관리

- 서비스 이용 내역
- 서비스 이용 현황
- 프로모션 내역
- 청구 내역 추세

#### 계정관리

- 회원정보 변경
- 비밀번호 변경
- 파트너 관리
- 보안 설정
- SNS 연동
- 계정 변경
- 인증키 관리

#### 결제관리

- 결제수단 관리
- 크레딧 및 할인 관리
- 코인 관리

나의 문의내역

알림 관리 Update

솔루션 이용 현황

#### API 인증키 관리

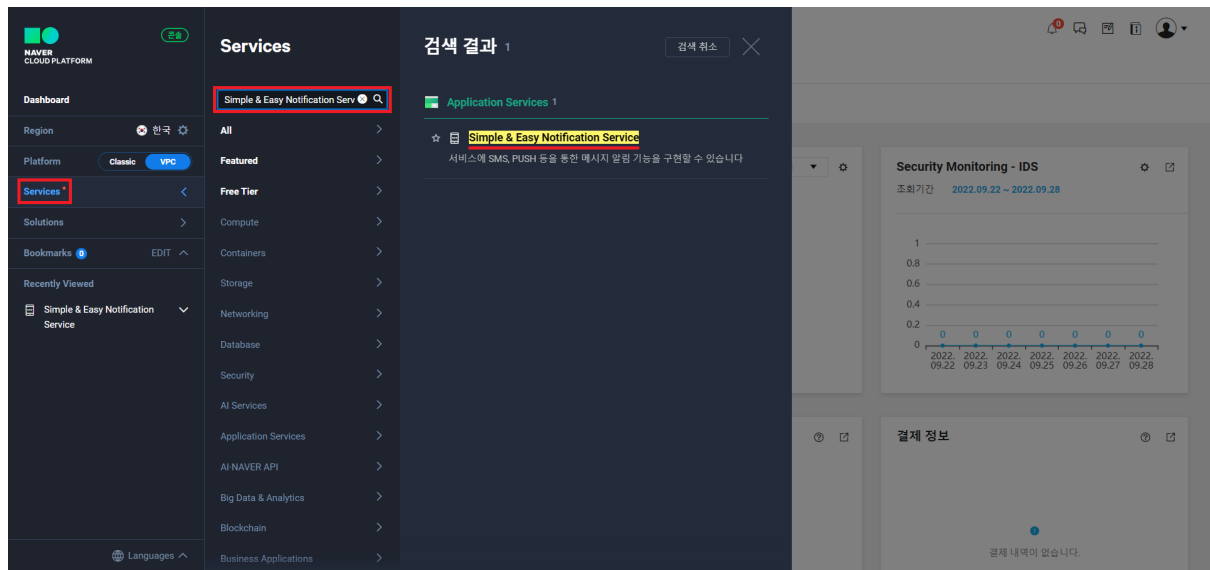
신규 API 인증키 생성

Access Key ID	Secret Key	생성일자	상태	관리
Access Key ID	보기	2022년 09월 20일	사용 중	사용 중지

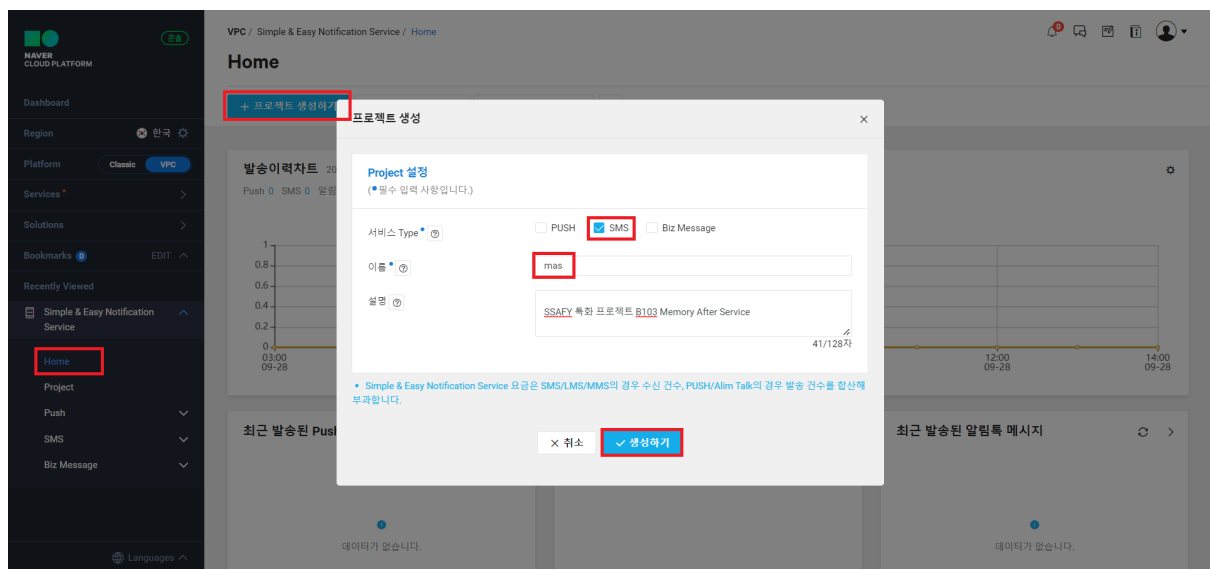
NAVER Cloud Platform의 [메인 페이지](#) > [마이페이지](#) > [인증키 관리](#) > [신규 API 인증키 생성](#)

추후 Spring Boot의 classpath(/resource/)에 있는 application-env.yml 구성할 때, naver-cloud-sms의 **accessKey**와 **secretKey**의 값으로 쓰이므로 Access Key ID와 Secret Key를 복사해두면 편함.

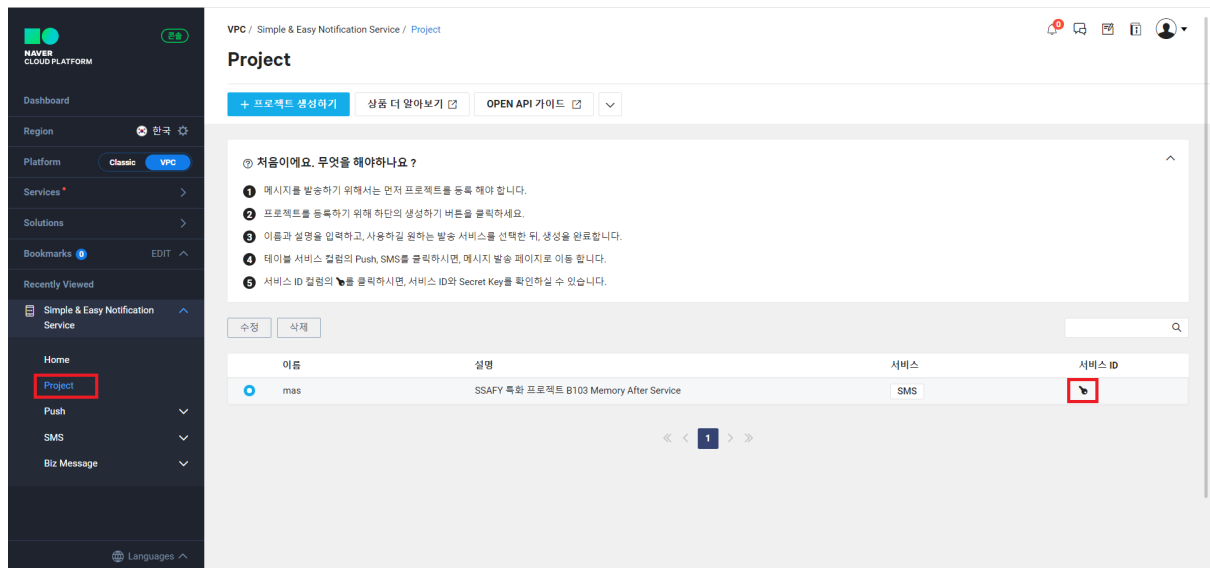
## Simple & Easy Notification Service(SENS)



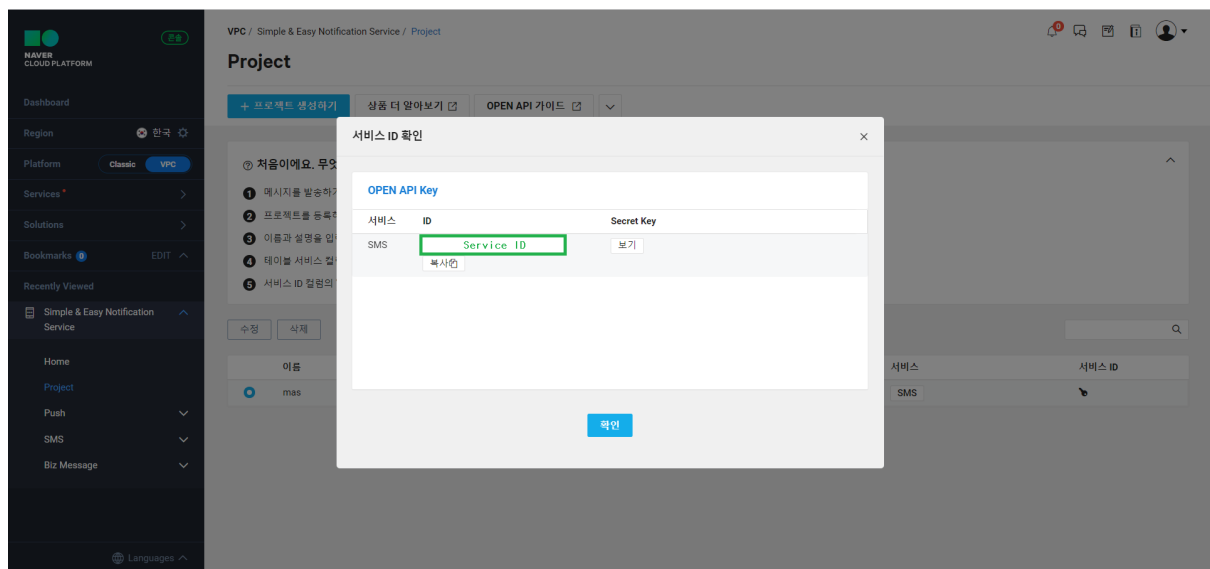
Naver Cloud Platform의 메인 페이지 > 콘솔 > Services > Simple & Easy Notification Service



Home > 프로젝트 생성 > Project 설정 > 생성하기



Project > key icon 클릭하면 아래처럼 Open API의 Service ID 확인이 가능



인증키와 동일하게 application-env.yml 구성할 때, naver-cloud-sms의 **serviceId**의 값으로 쓰임



# application-\*.yml Files Setting

## application-database.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://{host address}:{mysql port}/{db name}?serverTimezone=Asia/Seoul&
characterEncoding=UTF-8
    username: {username}
    password: {password}
```

## application-env.yml

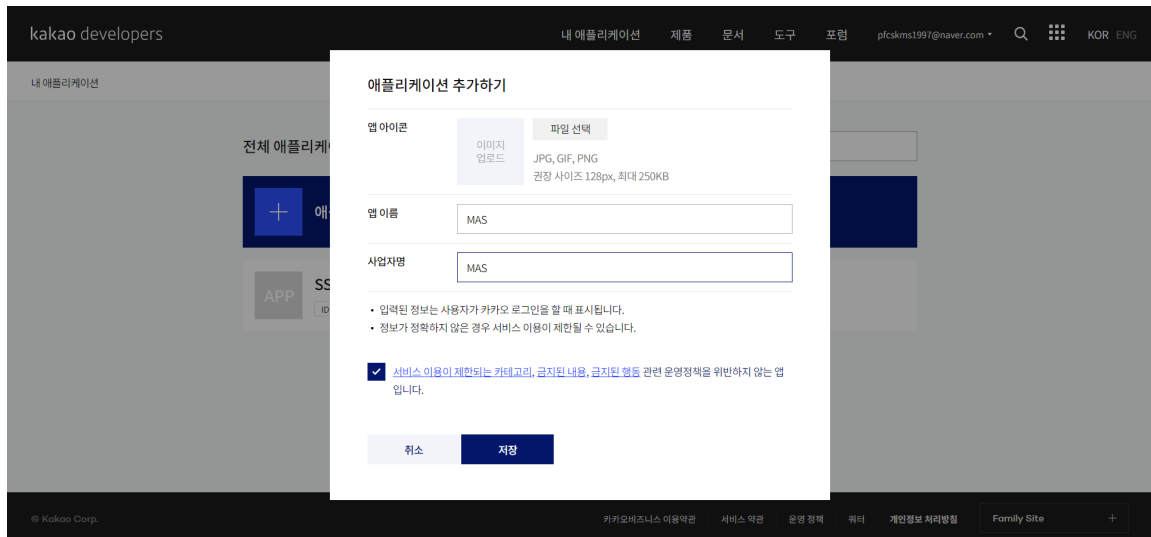
```
# Naver Cloud Simple & Easy Notification Service(SENS) 사용을 위한 data
naver-cloud-sms:
  accessKey: {Naver Cloud 회원 계정의 Access Key ID}
  secretKey: {Naver Cloud 회원 계정의 Secret Key}
  serviceId: {SENS Project의 서비스 ID}
  senderPhone: {발신자번호(하이픈 제외)}

# AES256 알고리즘으로 암호/복호화하기 위한 key
security-aes256:
  key: {32자리 정수}

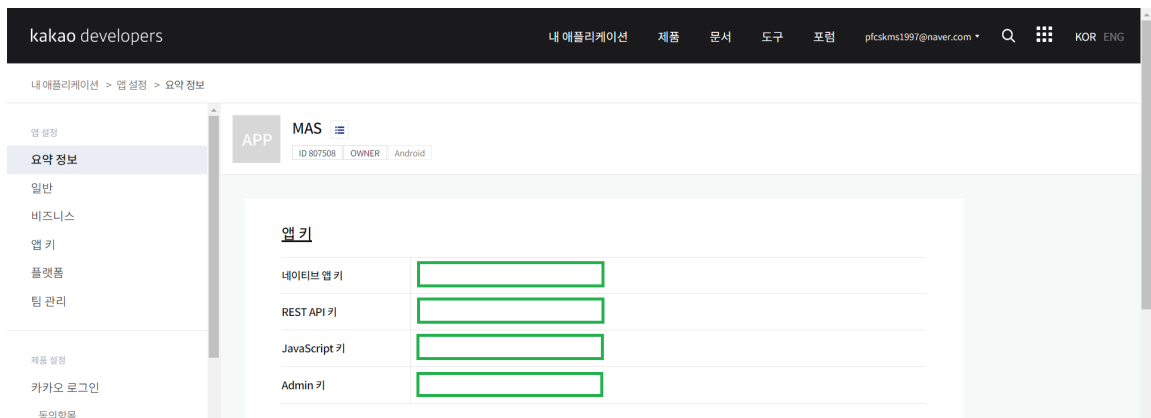
# SSH
#ssh-service:
#  username: {접속할 host username}
#  host: {접속할 host 주소}
#  port: {접속할 port 번호}
```

# Kakao Login API

1. Kakao Developers(<https://developers.kakao.com/>)에 접속 후 로그인
2. 시작하기 > 애플리케이션 추가하기



3. 생성한 애플리케이션에서 다음의 정보를 확인



4. AndroidManifest.xml에 코드 추가

<application> 내에 아래의 코드 추가

```
<meta-data
    android:name="com.kakao.sdk.AppKey"
    android:value="{네이티브 앱 키}" />
```



# Natural Language Processing

## Install KoNLPy

```
$ python3 -m pip install --upgrade pip
$ python3 -m pip install konlpy          # Python 3.x
```

## 품사 태깅 클래스 간 비교(<https://konlpy.org/ko/latest/morph/>)

수행 환경 : Quad Core Intel i7 CPU, Python 2.7, KoNLPy 0.4.1

### Time analysis

1. 로딩 시간: 사전 로딩을 포함하여 클래스를 로딩하는 시간.

- **Kkma** : 5.6988 secs
- **Komorán** : 5.4866 secs
- **Hannanum** : 0.6591 secs
- **okt** (previous **Twitter**): 1.4870 secs
- **Mecab** : 0.0007 secs

2. 실행시간: 10만 문자의 문서를 대상으로 각 클래스의 **pos** 메소드를 실행하는데 소요되는 시간.

- **Kkma** : 35.7163 secs
- **Komorán** : 25.6008 secs
- **Hannanum** : 8.8251 secs
- **okt** (previous **Twitter**): 2.4714 secs
- **Mecab** : 0.2838 secs

제공받은 Hadoop Cluster에서는 설치할 수 없는 Mecab을 제외하고, 가장 처리 시간이 빠른 Okt Class를 사용함

## Usage

```
from konlpy.tag import Okt
```



# Shell Script 작성

## BashScript.sh 설명

bash bashScript.sh [키 주소] [클러스터 주소] [데이터 경로] [데이터 파일 이름] [결과 경로] [결과 파일 이름]

- 키 주소
  - 하둡 클러스터에 들어가기 위한 .pem 키 파일 경로
  - `/home/ubuntu/bashScript/J7B103T.pem`
- 클러스터 주소
  - 하둡 클러스터에 들어가기 위한 주소
  - `j7b103@cluster.ssafy.io`
- 데이터 경로
  - 분석을 위한 카카오톡 데이터 경로
  - `~/mas_server/upload/계정명/시간_스트링`
- 데이터 파일 이름
  - 분석할 카카오톡 ZIP 파일
  - `Kakaotalk.zip`
- 결과 경로
  - 분석 결과가 있을 파일 경로
  - `~/mas_server/upload/`
- 결과 파일 이름
  - 분석 결과 파일 경로
  - `Data_koreanWordCount_result.csv`

```
#!/bin/bash

# 로그 출력용 함수
print_log() {
    if [ $? -eq 0 ];then
        echo -e " $1 Success!"
    fi
}
```

```

else
    echo -e " $1 Failed!"
    exit 9
fi
}

PRE_PROCESS="Data_Kakao_with_NLP.py"
WORD_COUNT="Data_koreanWordCount.py"

# 인자 확인
# bash bashScript.sh [키 주소] [클러스터 주소] [데이터 경로] [데이터 파일 이름] [결과 경로] [결과 파일 이름]
if [ $# -ne 6 ]
then
    echo "Usage: $0 [Key path] [cluster url] [data path] [data name] [result path] [result name]"
    exit 1
fi

# 연결 확인
ssh -i $1 $2 pwd
print_log "Connect to Hadoop Cluster from EC2 Instance"

# 데이터 파일 전송
scp -i $1 $3/$4 $2:$5/kakaotalk/$4
print_log "Copy Input data to Hadoop cluster from EC2 Instance"

# 파이썬 스크립트 2개 실행
# 1. rawData -> word
ssh -i $1 $2 "cd $5; python3 ${PRE_PROCESS} kakaotalk/$4"
print_log "Execute PySpark preprocess script in Hadoop cluster"

# 2. wordCount
ssh -i $1 $2 "cd $5; python3 ${WORD_COUNT} Data_Kakao_with_NLP_result.csv"
print_log "Execute PySpark wordcount script in Hadoop cluster"

# 결과 파일 가지고 오기
scp -i $1 $2:$5/$6 $3/$6
print_log "Copy Output data to EC2 Instance from Hadoop cluster"

# remove uploaded data
ssh -i $1 $2 "cd $4/result; rm -f *"
print_log "Delete personal data in Hadoop cluster"

```