포팅메뉴얼

Status	백엔드
Assign	
Date	
속성	
Property	

기술스택

▼ 개발환경

OS

- window 10
- Ubuntu 20.04 LTS

IDE

- Android Studio Dolphin | 2021.3.1

Datebase

- MySQL workbench 5.7

Docker

- Docker 20.10.18

EC2 기본

▼ Docker 설치

```
# 오래된 버전 삭제하기
sudo apt-get remove docker docker-engine docker.io containerd runc

# repository 설정하기
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Docker의 Official GPG Key 를 등록
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

#stable repository 를 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 Engine 설치하기
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

```
# 설치확인
docker --version
```

▼ Jenkins 설치

- Docker로 Jenkins 설치(8282번 포트 개방)

```
docker run -d -p 8282:8080 -p 50000:50000 -v /var/jenkins:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /etc/localtime:/etc/localtime:ro -e TZ=Asia/Seoul \
--name jenkins -u root jenkins/jenkins:lts-jdk11
```

▼ jenkins 컨테이너 안에 docker 설치

- 컨테이너 형태로 설치된 젠킨스 안에서 docker 명령어를 실행하기 위해서 docker를 설치
- 설치를 위해 먼저 컨테이너 안으로 접근, 쉘을 실행.

```
docker exec -it jenkins bash
```

- 도커 설치 → 위의 **Docker 설치** 과정 참고

▼ Docker Image pull

- mysql 5.7

```
docker pull mysql:5.7
```

- node:16-alpine

```
docker pull node:16-alpine
```

- openjdk:8-jre-alpine

```
docker pull openjdk:8-jre-alpine
```

DB

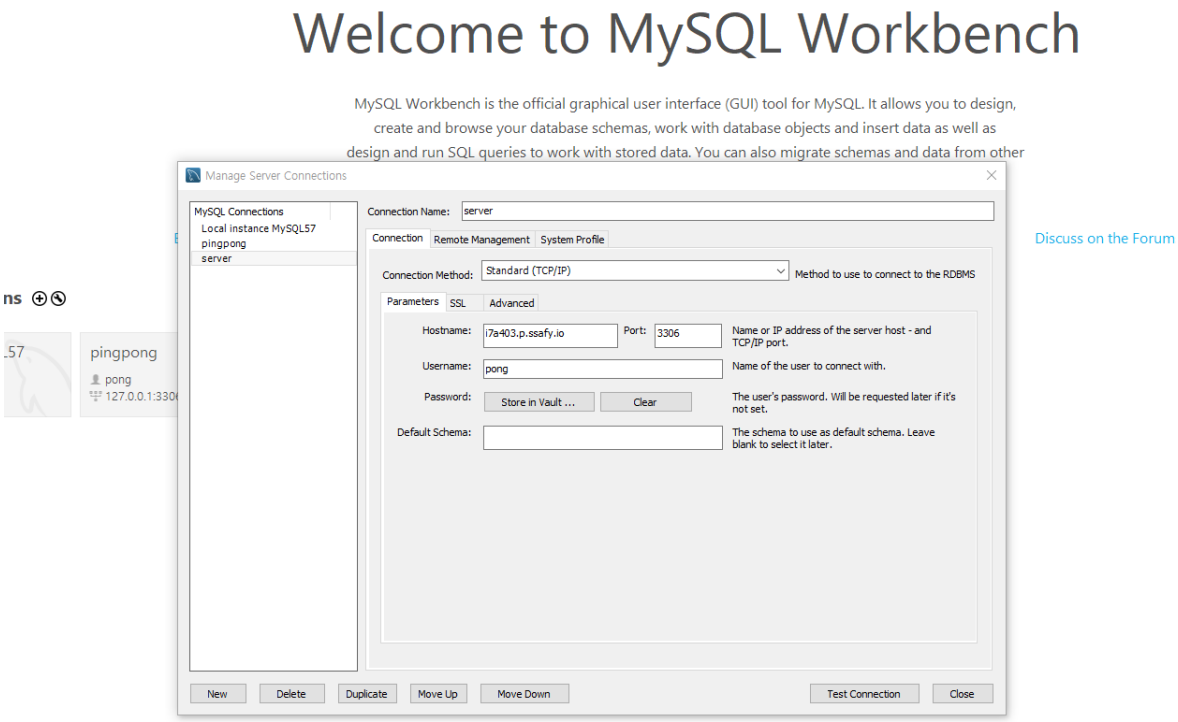
▼ mysql 컨테이너 실행(8283 포트 개방)

```
docker run --name mysql -d -p 8283:3306 mysql:5.7 \
 -e MYSQL_ROOT_PASSWORD=ssafydrdoca204! \
-e MYSQL_DATABASE=drdoc -e MYSQL_USER=drdoc -e MYSQL_PASSWORD=ssafya204drdoc \
--character-set-server=utf8 --collation-server=utf8_unicode_ci
```

```
docker run --name mysql -d -p 8283:3306 -e MYSQL_ROOT_PASSWORD=ssafydrdoca204! -e MYSQL_DATABASE=drdoc -e MYSQL_USER=drdoc -e MYSQ
```

<한줄버전 코드>

- workbench에서 연결 후 DB 관리 가능

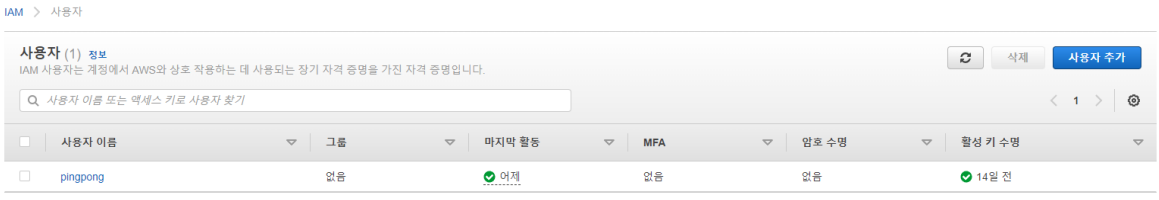


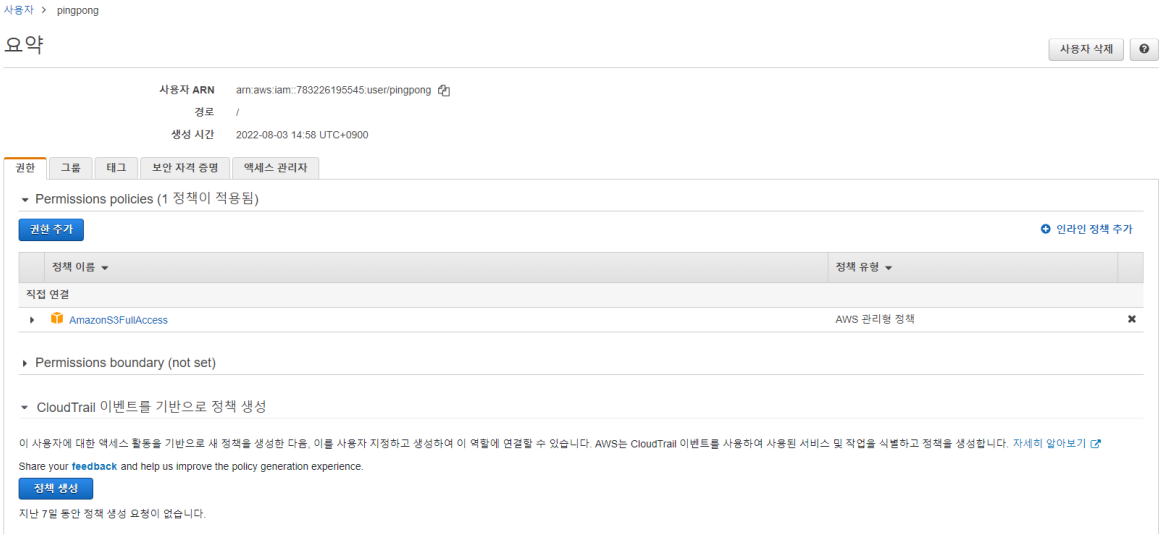
S3

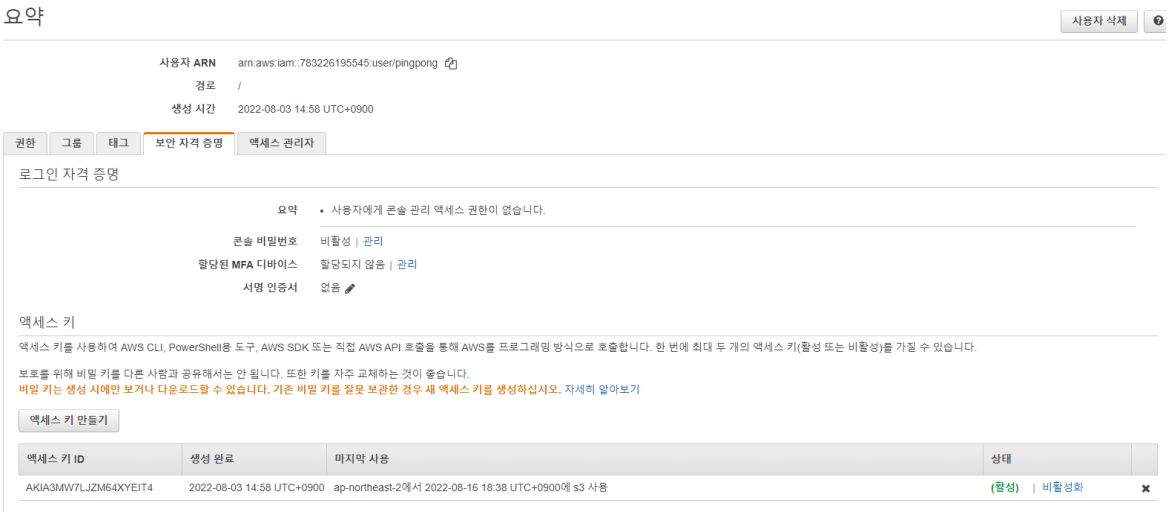
- 아마존 서버 컴퓨터의 저장공간만 빌리는 서비스. 파일 업로드, 다운로드가 가능하다.
- AWS 계정 필요

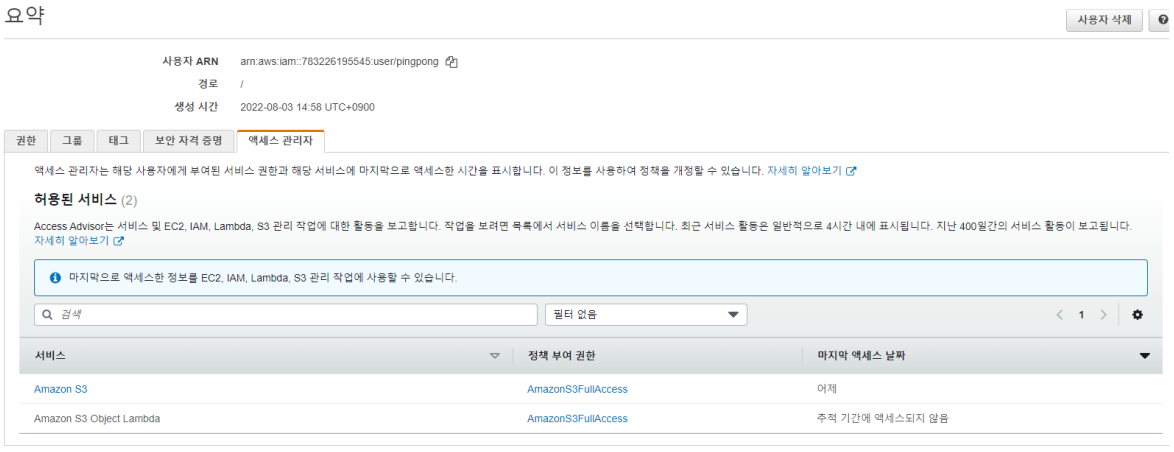
▼ IAM - 사용자

- 사용자 추가 클릭
- 사용자 이름 작성
- AWS 액세스 유형 선택 → 엑세스 키 선택 시 api 로 접근 가능한 key 발급



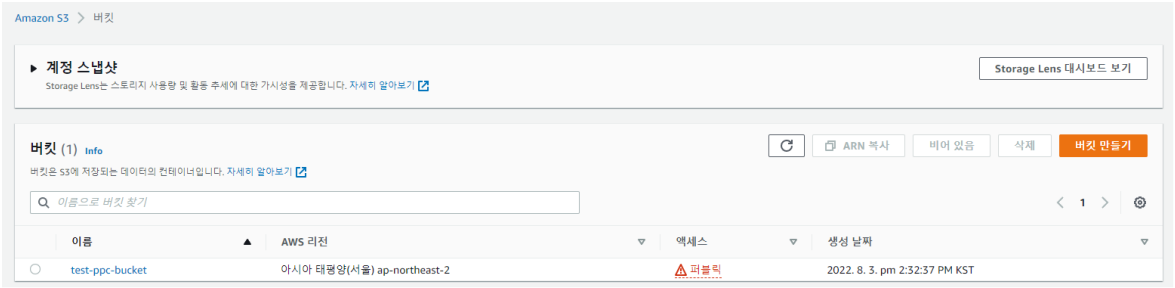


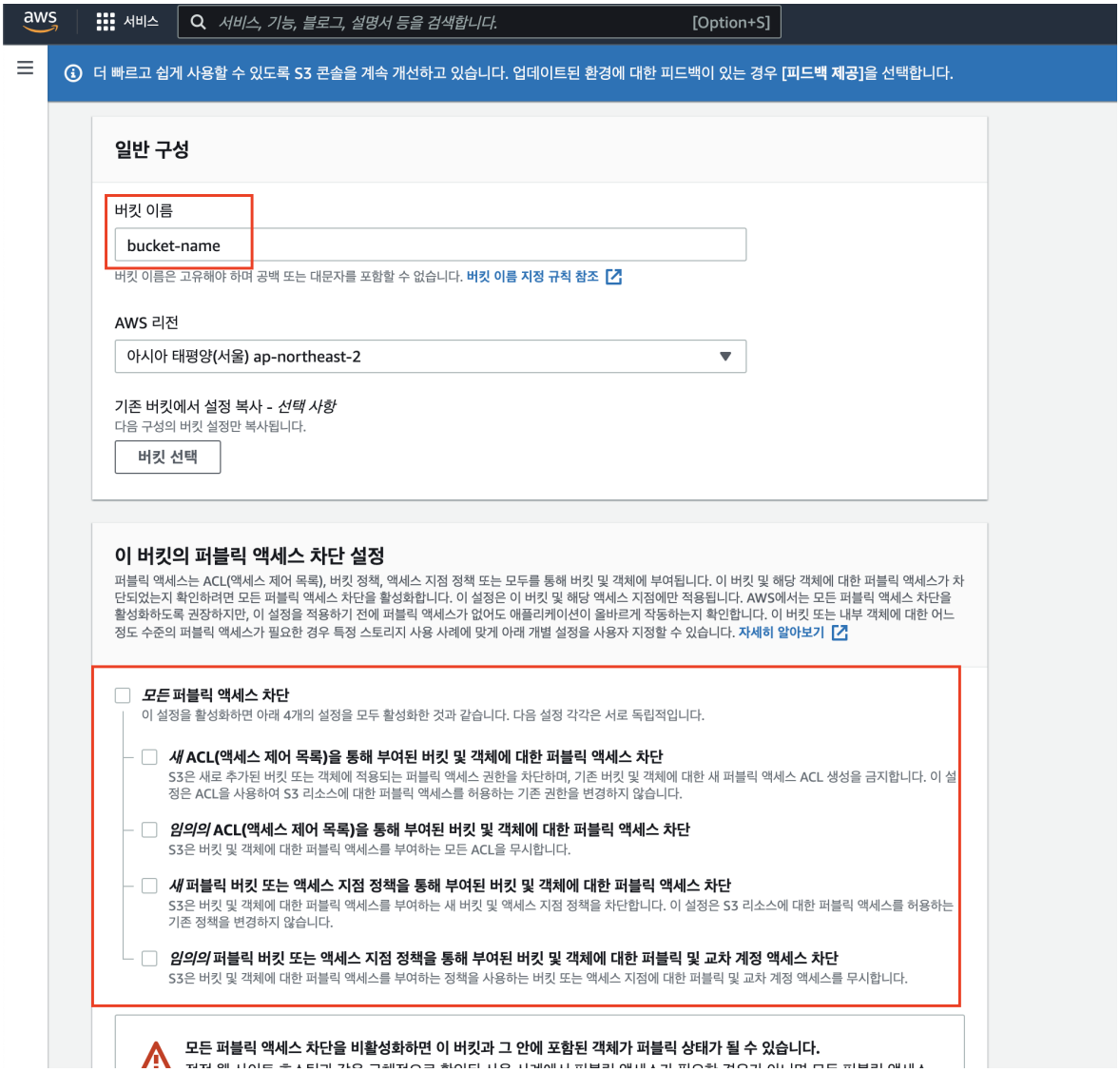




▼ S3 - bucket

- 버킷 만들기


Amazon S3 > 버킷
계정 스냅샷
Storage Lens는 스토리지 사용량 및 활동 추세에 대한 가시성을 제공합니다. 자세히 알아보기
Storage Lens 대시보드 보기
버킷 (1) Info
버킷은 S3에 저장되는 데이터의 컨테이너입니다. 자세히 알아보기
ARN 복사
비어 있음
삭제
버킷 만들기
이름으로 버킷 찾기
이름
AWS 리전
액세스
생성 날짜
test-ppc-bucket
아시아 태평양(서울) ap-northeast-2
퍼블릭
2022. 8. 3. pm 2:32:37 PM KST

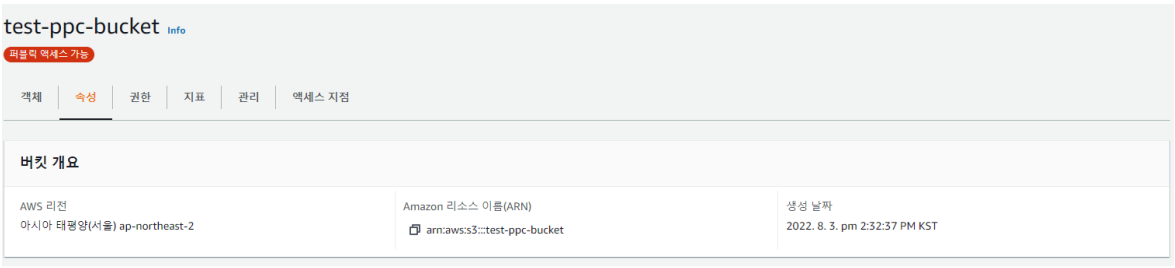

aws
서비스
서비스, 기능, 블로그, 설명서 등을 검색합니다.
[Option+S]
더 빠르고 쉽게 사용할 수 있도록 S3 콘솔을 계속 개선하고 있습니다. 업데이트된 환경에 대한 피드백이 있는 경우 [피드백 제공]을 선택합니다.
일반 구성
버킷 이름
bucket-name
버킷 이름은 고유해야 하며 공백 또는 대문자를 포함할 수 없습니다. 버킷 이름 지정 규칙 참조
AWS 리전
아시아 태평양(서울) ap-northeast-2
기존 버킷에서 설정 복사 - 선택 사항
다음 구성의 버킷 설정만 복사됩니다.
버킷 선택
이 버킷의 퍼블릭 액세스 차단 설정
퍼블릭 액세스는 ACL(액세스 제어 목록), 버킷 정책, 액세스 지정 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 이 버킷 및 해당 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 모든 퍼블릭 액세스 차단을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지점에만 적용됩니다. AWS에서는 모든 퍼블릭 액세스 차단을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 이 버킷 또는 내부 객체에 대한 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. 자세히 알아보기
모든 퍼블릭 액세스 차단
이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.
새 ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단
S3은 새로 추가된 버킷 또는 객체에 적용되는 퍼블릭 액세스 권한을 차단하며, 기존 버킷 및 객체에 대한 새 퍼블릭 액세스 ACL 생성을 금지합니다. 이 설정은 ACL을 사용하여 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 권한을 변경하지 않습니다.
임의의 ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 모든 ACL을 무시합니다.
새 퍼블릭 버킷 또는 액세스 지점 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 새 버킷 및 액세스 지점 정책을 차단합니다. 이 설정은 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 정책을 변경하지 않습니다.
임의의 퍼블릭 버킷 또는 액세스 지점 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 정책을 사용하는 버킷 또는 액세스 지점에 대한 퍼블릭 및 교차 계정 액세스를 무시합니다.
모든 퍼블릭 액세스 차단을 비활성화하면 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있습니다.

S3 퍼블릭 액세스 허용

정책

```
{
  "Id": "Policy1637493633297",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1637493415559",
      "Action": "s3:*",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::new-bucket2",
      "Principal": "*"
    }
  ]
}
```

S3 접근 정책 생성해서 버킷 정책으로 적용



▼ springboot 설정

aws.yml

```
#S3
cloud:
  aws:
    credentials:
      accessKey: <S3액세스 키>
      secretKey: <S3시크릿 키>
    s3:
      bucket: <s3 버켓 이름>
    region:
      static: ap-northeast-2 <S3 위치>
    stack:
      auto: false

spring:
  servlet:
    multipart:
      max-file-size: 10MB <용량제한>
      max-request-size: 10MB
```

Fast API 배포

▼ Fast API 서버 배포

참조 : https://www.vultr.com/docs/how-to-deploy-fastapi-applications-with-gunicorn-and-nginx-on-ubuntu-20-04/

```
// 1. 프로젝트 폴더 생성

// 2. 가상환경 생성
conda install -c anaconda python=3.7
conda create -n <가상환경이름> python=3.7.x
conda activate <가상환경이름>
// 기본 가상환경(base)인 경우 항상 activate된 경우가 있는데, 이 경우 자동 가상환경 실행을 취소한다.
conda config --set auto_activate_base false

// 3. 기본 dependencies 설치
pip install wheel
pip install fastapi[all]
```

```
// 4. 기본 Fast API 만들기
from fastapi import FastAPI
app = FastAPI()

@app.get("/hello")
async def home():
    return {"message": "Hello World"}

// 5. 배포 테스트
uvicorn <위에서 만든 .py 파일 이름>:app
// 다른 터미널에서 실행하여 확인
curl http://localhost:8000

// 6. gunicorn 설치
// gunicorn은 unix를 위한 파이썬 WSGI HTTP 서버다.
// 웹 어플리케이션을 관리하고 감시하기 더 쉽게 해준다.
pip install gunicorn

// 7. gunicorn을 이용한 배포
gunicorn -k uvicorn.workers.UvicornWorker --access-logfile ./gunicorn-access.log <FastAPI가 정의된 .py 파일 이름>:app --bind 0.0.0.0:8
// 방화벽 설정까지 마치면 8000번 포트를 통해 FastAPI를 호출할 수 있다

// 서버 종료
sudo kill -9 `sudo lsof -t -i:8000`
```

Backend Jenkins 자동배포

Jenkins

▼ 젠킨스 접속

- http://localhost:9090/로 접속 (위에서 포트를 9090으로 설정.)
- 젠킨스 초기 비밀번호 확인

```
docker logs jenkins
또는
sudo docker exec -it jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

- Plugin 다운로드 GitLab, generic WebhookTrigger, GitLab API, GitLab Authentication, Docker, Docker Comons, Docker Pipeline, Docker API

▼ Backend 프로젝트 생성(Free Style)

- 소스코드 관리: Git
  - Repository URL: https://lab.ssafy.com/s07-webmobile1-sub2/S07P12A403/
  - Credentials : gitlab id, password로 추가 or GitLab AccessToken을 발급받아 젠킨스 전체 설정에서 적용
- Branchs to build: */master
- Build Step: Execute shell

```
docker rm spring -f
cd backend
chmod +x gradlew
./gradlew bootJar
docker build . -t pingpongclass:latest
docker run -d -p 8080:8080 --link mysql-pingpong --name spring pingpongclass:latest
```

- 첫줄인 docker rm spring -f 는 첫 실행에서는 작성x (이전에 빌드된 이미지를 삭제하란 의미라 처음에는 삭제할게 없다고 빌드 실패가 뜰 수 있음.)
- chmod +x gadlew 빌드할 권한부여
- ./gradle bootJar jar 빌드 명령
- docker build . -t pingpongclass:latest: (중간에 ". " 넣는거 잊지말기) pingpongclass라는 이름으로 도커이미지를 빌드 명령.

- docker run -d -p 8080:8080 --link mysql-pingpong --name spring pingpongclass:latest: pingpongclass:latest를 8080 포트에서, mysql-pingpong 컨테이너와 연결, 컨테이너 이름은 spring으로 해서 작동 명령
- 주의) mysql-pingpong 컨테이너를 먼저 run 후에 be를 run해야됨

Dockerfile (방법 1)

```
FROM openjdk:8-jre-alpine
EXPOSE 8080
ARG JAR_FILE=build/libs/backend-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

▼ Backend 프로젝트 생성(Pipeline)

```
pipeline {
agent any
stages {
    stage ('clone') {
        steps {
            git branch: 'develop', credentialsId: 'SSAFY_git_global', url: '<https://oauth2:<gitlab 액세스 토큰>@lab.ssafy.com/s07
-ai-image-sub2/S07P22A204.git>'
            // 경로 이동
            sh "yes | cp -r ../BackEndDeploy/*.yml ./BackEnd/src/main/resources/"
            sh "yes | cp -r ../BackEndDeploy/Dockerfile ./BackEnd/"
        }
    }

    stage("Build") {
        steps {
            dir("BackEnd") {
            sh "chmod +x ./gradlew"
            sh "./gradlew clean build -x test"
            }
        }
    }

    stage("Deploy") {
        steps {
            echo "deploy.."
            dir("BackEnd") {
                sh "docker stop drdoc || true && docker rm drdoc || true"
                sh "docker build -t drdoc_backend:0.1 ."
                sh "docker run --name drdoc -d -p 8284:8080 --link mysql --rm drdoc_backend:0.1"
            }
        }
    }
}
```

}

```
pipeline {
    agent any
    stages {

        stage ('clone') {
            steps {
                git branch: 'develop', credentialsId: 'SSAFY_git_global', url: 'https://oauth2:PBpjyRqjhxRawy_stjLA@lab.ssafy.com/s
                sh "yes | cp -r ../BackEndDeploy/*.yml ./BackEnd/src/main/resources/"
                sh "yes | cp -r ../BackEndDeploy/Dockerfile ./BackEnd/"
            }
        }

        stage("Build") {
            steps {
                dir("BackEnd") {
                sh "chmod +x ./gradlew"
                sh "./gradlew clean build -x test"
                }
            }
        }
```

```
        stage("Deploy") {
            steps {
                echo "deploy.."
                dir("BackEnd") {
                    sh "docker stop drdoc || true && docker rm drdoc || true"
                    sh "docker build -t drdoc_backend:0.1 ."
                    sh "docker run --name drdoc -d -p 8284:8080 --link mysql --rm drdoc_backend:0.1"
                }
            }
        }
    }
}
```

Nginx

▼ docker container 설치

[SSL] Docker Nginx, Certbot & Let's Encrypt를 사용해 SSL 인증서 발급

EC2 ubuntu 18.04 Docker CE, CE-CLI 20.10.6 Docker Compose 1.29.1 docker-compose.yml init-letsencrypt.sh data/nginx/conf.d/app.conf data/nginx/conf.d/app.conf server { listen 80; listen [::]:80; server_name {domain}; // 등록한 도메인 추가 location /.well-known/acme-challenge/ { allow all; root

https://velog.io/@fordevelop/Docker-Nginx-Certbot-Lets-Encrypt%EB%A5%BC-%EC%82%AC%EC%9A%A9%ED%95%B4-SSL-%EC%9D%B8%EC%A6%9D%EC%84%9C-%EB%B0%9C%EA%B8%89


```
docker run --name nginx -p 80:80 -dt nginx
```

- 우선 nignx 설정파일 경로를 찾아서 접근(docker container 안에서 접근 등)

```
sudo su
# nginx.conf 파일 위치 찾기
find / -name nginx.conf
결과로 나온 위치 확인 후 해당 경로로 이동하는데 해당 경로는 nginx.conf 파일이있는 경로이고,
우리가 수정할 default.conf은 위에 나온경로/conf.d 아래있음

#directory이동
cd 경로/conf.d

# default.conf 수정
nano default.conf
```

default.conf 파일이 이미 있는 경우 복사 가능하다.

```
docker cp default.conf nginx:/etc/nginx/conf.d/default.conf
docker exec nginx nginx -s reload
```

▼ default.conf

- 기본 설정에 client 부분만 수정, 추가 (bold체 부분)

```
upstream client{
  server j7a204.p.ssafy.io:8284;
}

server {
    listen 80;
    listen [::]:80;
    server_name  j7a204.p.ssafy.io;

    location / {
        rewrite ^(.*) https://j7a204.a.ssafy.io:443$1 permanent;
    }
}

server{
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name  j7a204.p.ssafy.io;
    proxy_set_header Host $host;
```

```
    proxy_set_header X-Forwarded-For $remote_addr;

    #ssl config
    ssl_certificate /etc/letsencrypt/live/j7a204.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j7a204.p.ssafy.io/privkey.pem;

    #Proxy

    location /api {
        proxy_pass http://client;
    }

    #swagger 설정
    location /swagger-ui {
        proxy_pass http://client/swagger-ui;
    }
    location /swagger-resources {
        proxy_pass http://client/swagger-resources;
    }
    location /v2 {
        proxy_pass http://client/v2;
    }
    location /webjars {
        proxy_pass http://client/webjars;
    }
    location /configuration {
        proxy_pass http://client/configuration;
    }

    #영상 분석용 서버
    location /action {
        proxy_pass http://j7a204.p.ssafy.io:8000/action;

        proxy_connect_timeout   3000; #기본 60초
        proxy_send_timeout      3000; #기본 60초
        proxy_read_timeout      3000;
    }

    #안구질환  분석용 서버
    location /eye {
        proxy_pass http://j7a204.p.ssafy.io:8010/eye;

        proxy_connect_timeout   3000; #기본 60초
        proxy_send_timeout      3000; #기본 60초
        proxy_read_timeout      3000;
    }

    client_max_body_size 10M;
}
```

```
# conf 파일 수정 후 수정사항을 반영
$ docker container exec <container> nginx -s reload
<>에는 현재 가동중인 nginx 컨테이너명 또는 ID를 작성
확인이 필요하다면 docker ps 로 확인
```

방화벽

▼ 사용 포트

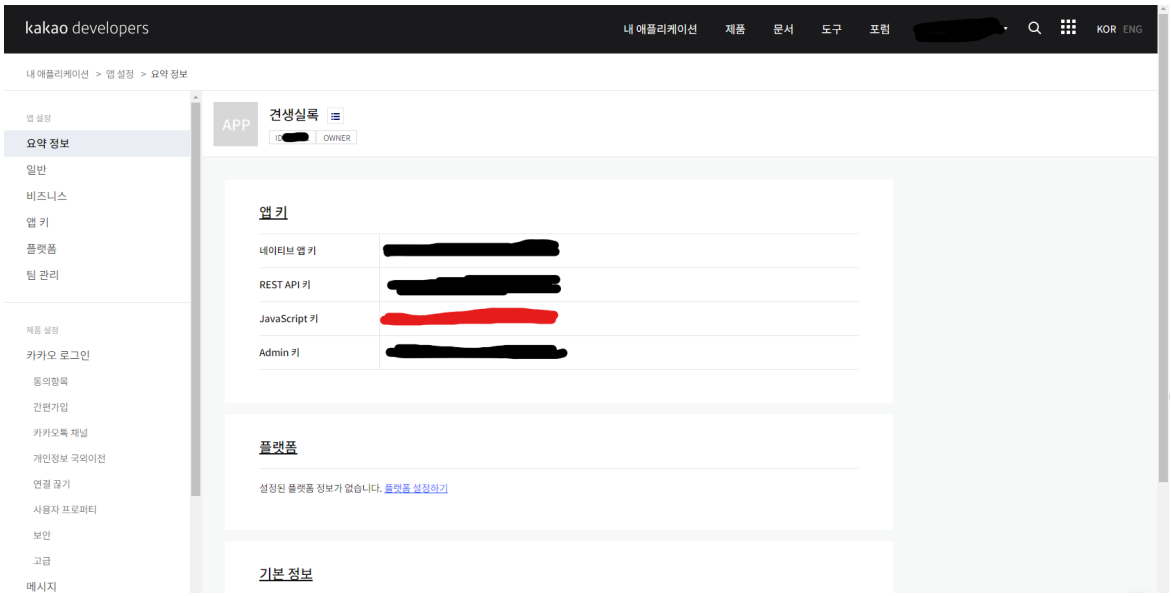
```
To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere //ssh
80/tcp                     ALLOW       Anywhere
443/tcp                    ALLOW       Anywhere
8000/tcp                   ALLOW       Anywhere // FAST API
8080/tcp                   ALLOW       Anywhere // 스웨거
8282/tcp                   ALLOW       Anywhere // 젠킨스
8283/tcp                   ALLOW       Anywhere // DB
8284/tcp                   ALLOW       Anywhere // nginx
50000/tcp                  ALLOW       Anywhere // 젠킨스 2

// OpenSSH                    ALLOW       Anywhere
// Nginx HTTP                 ALLOW       Anywhere
```

Flutter 카카오 맵 연동

▼

1. 카카오 디벨로퍼 가입 후 앱 생성
2. 앱 키 중 JavaScript 키 복사



3. 아래 내용으로 .env 파일 생성(위치: `<app dir>/.env`)

```
kakaoMapAPIKey=<발급받은 kakaoMapApi(JavaScript 키)>
// kakaoMapAPIKey=11111111111111111111111111111111
```

Flutter 구글 플레이스토어 수동배포

▼ Flutter Build(.aab app bundle 파일 생성)

1. (이미 있다면 생략 가능) 키스토어 파일(.jks)만들기

맥/리눅스:

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

윈도우:

```
keytool -genkey -v -keystore c:/Users/<유저명>/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias <참조할 이름>
```

2. `<app dir>/android/key.properties` 파일을 만들어 참조하기

```
storePassword=<password from previous step>
keyPassword=<password from previous step>
keyAlias=<생성하고 싶은 키 이름>
storeFile=<key store 파일 위치, 예) /Users/<user name>/key.jks>
```

▼ Gradle 수정하기 (`<app dir>/android/app/build.gradle` 파일을 수정)

```
// 이 위치에 코드를 삽입합니다.
   android {
```

`key.properties` 파일로부터 keystore 정보를 가져올 수 있도록 변경하세요.

```
    def keystoreProperties = new Properties()
    def keystorePropertiesFile = rootProject.file('key.properties')
    if (keystorePropertiesFile.exists()) {
        keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
    }

// 이 위치에 코드를 삽입합니다.
    android {
```

다음 코드를:

```
    buildTypes {
        release {
            // TODO: release 빌드 버전을 위한 서명 구성을 추가하세요.
            // 현재는 `flutter run --release`가 디버그용 키로 서명되어 동작합니다.
            signingConfig signingConfigs.debug
        }
    }
```

다음과 같이 변경하세요:

```
    signingConfigs {
        release {
            keyAlias keystoreProperties['keyAlias']
            keyPassword keystoreProperties['keyPassword']
            storeFile file(keystoreProperties['storeFile'])
            storePassword keystoreProperties['storePassword']
        }
    }
    buildTypes {
        release {
            signingConfig signingConfigs.release
        }
    }
```

이제 앱의 release 빌드에서 자동으로 서명이 될 것 입니다.

▼ Proguard 사용(난독화 및 축소)

Step 1 - Proguard 구성하기

Proguard Rule을 구성하기 위하여 `/android/app/proguard-rules.pro` 파일을 생성하고 다음과 같은 규칙을 추가합니다.

```
## Flutter wrapper
-keep class io.flutter.app.** { *; }
-keep class io.flutter.plugin.**  { *; }
-keep class io.flutter.util.**  { *; }
-keep class io.flutter.view.**  { *; }
-keep class io.flutter.**  { *; }
-keep class io.flutter.plugins.**  { *; }
-dontwarn io.flutter.embedding.**
```

이 설정은 Flutter 엔진 라이브러리를 보호합니다. 다른 라이브러리(예. Firebase)를 보호하기 위해서는 추가적으로 규칙을 작성해야합니다.

Step 2 - 난독화와 축소 사용하기

`/android/app/build.gradle` 파일을 열고 `buildTypes`가 정의된 부분을 찾으세요. `release` 설정 부분 안에 `minifiyEnabled` 와 `useProguard` 플래그가 true로 설정되어 있습니다. 여기에 step 1 에서 만든 Proguard 규칙 파일을 추가해야 합니다.

참고: 난독화와 축소 과정으로 인해 Android 앱의 컴파일 시간이 크게 늘어날 수 있습니다.

```
android {

    ...
```

```
    buildTypes {

        release {

            signingConfig signingConfigs.release

            minifyEnabled true
            useProguard true

            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'

        }
    }
}
```

▼ 앱 매니페스트 검토하기

올바른 앱 설정을 위하여 `<app dir>/android/app/src/main`에 있는 기본 앱 매니페스트 파일인 `AndroidManifest.xml`을 검토하고 올바른 값들을 포함하는지 확인하세요. 특히:

- `application`: 앱의 최종 이름을 반영하기 위해 `application`에 있는 `android:label`을 수정하세요.

▼ 빌드 구성 검토하기

올바른 빌드 구성을 위하여 `<app dir>/android/app`에 있는 기본 Gradle build file 파일인 `build.gradle`을 검토하고 올바른 값들을 포함하는지 확인하세요. 특히:

- `defaultConfig`:
  - `applicationId`: 고유한 최종 (Application Id)appid를 지정하세요.
  - `versionCode` & `versionName`: 내부 앱 버전 번호를 지정하고, 문자열 형태로 명시하세요. pubspec.yaml 파일에 `version` 속성을 설정하여 내부 앱 버전 번호를 문자열 형태로 지정할 수 있습니다. 버전 정보 지침에 대해서는 버전 문서를 참조하세요.
  - `minSdkVersion` & `targetSdkVersion`: 최소 API 레벨과 개발 대상 버전으로 지정한 지정 API 레벨을 명시하세요. 자세한 내용은 버전 문서의 API 레벨 영역을 참조하세요.

▼ release 앱 번들 빌드하기

여기서는 어떻게 release 앱 번들을 빌드하는지 소개합니다. 위 단락의 서명하기를 완료했다면, release 번들이 서명될 것입니다.

터미널:

```
1. cd <app dir>를 입력하세요(당신의 앱 디렉토리로 <app dir> 를 변경하세요.)
2. flutter build appbundle을 실행하세요.(flutter build 실행은 기본적으로 release 빌드 입니다.)
```

`<app dir>/build/app/outputs/bundle/release/app-release.aab`

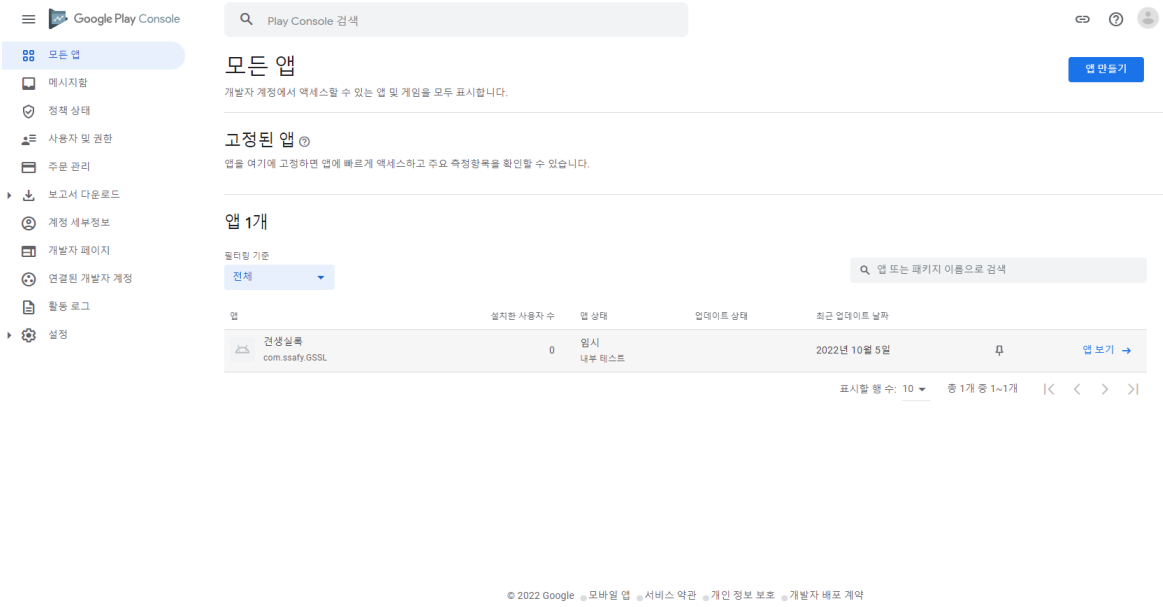
경로로 출력됩니다.

※ Flutter 빌드가 여러 번 되는 경우 빌드 정보를 변경하려면 `pubspec.yaml` 파일 내부의

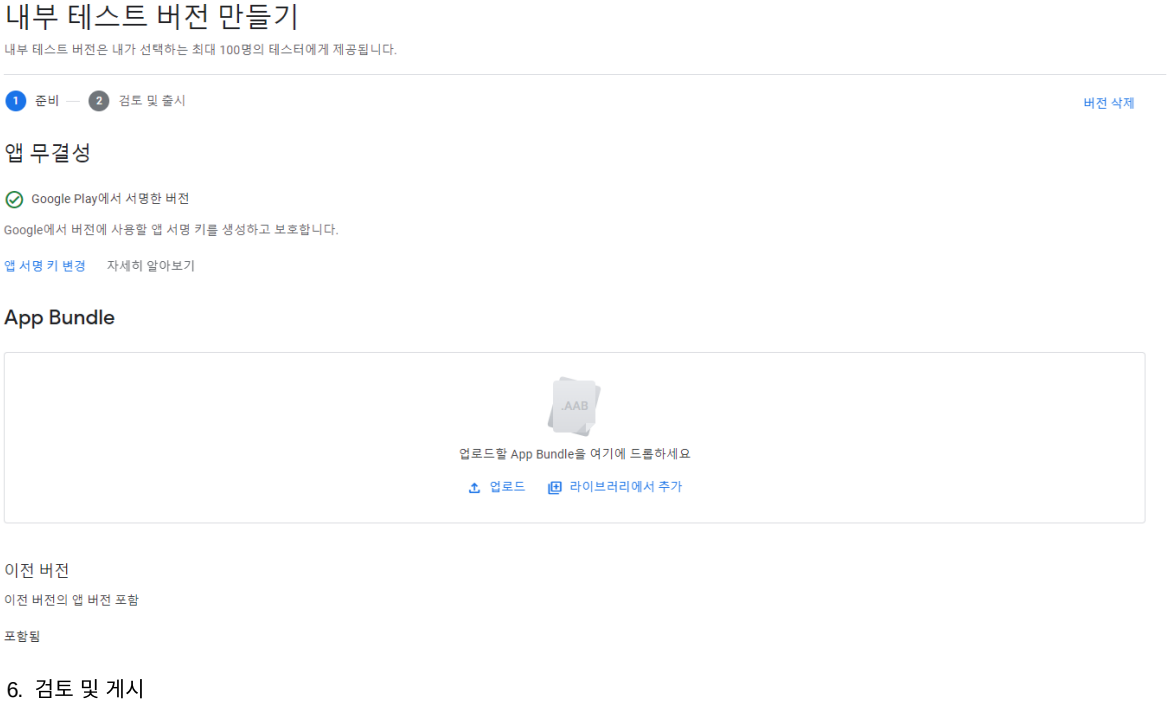
`version: 1.0.2+3` (빌드이름+빌드번호)와 같은 형식의 빌드 정보를 수정한 뒤 빌드해주세요.

▼ 구글 플레이스토어 설정

1. 구글 개발자 등록(25달러/평생)
2. 구글 플레이 콘솔 접속



3. [앱 만들기] 클릭 후 요구하는 양식에 맞춰 작성
4. 원하는 종류(프로덕션/공개 테스트/ 내부 테스트 등등...)의 트랙을 선택하여 정보를 제공
5. Flutter에서 빌드한 앱 번들 파일(.aab) 업로드



6. 검토 및 게시