포팅메뉴얼 (1)

1. 프로젝트 기술 스택

A. Back-end

기술 스택 (버전) : Spring boot `2.7.3`, MySQL `8.0.30`, Nginx `1.18.0`, Jenkins `2.361.1`, AWS EC2, AWS S3

사용 툴 : IntelliJ `2021.2.4`, MobaXterm `22.0`, MySQL Workbench `8.0.20`, DataGrip `2022.2.4`, JDK `11.0.15.1`

B. Front-end

기술 스택 (버전) : React `18.2.0`, stomp `6.1.2` (node-sass `7.0.3`, mui `5.10.5`

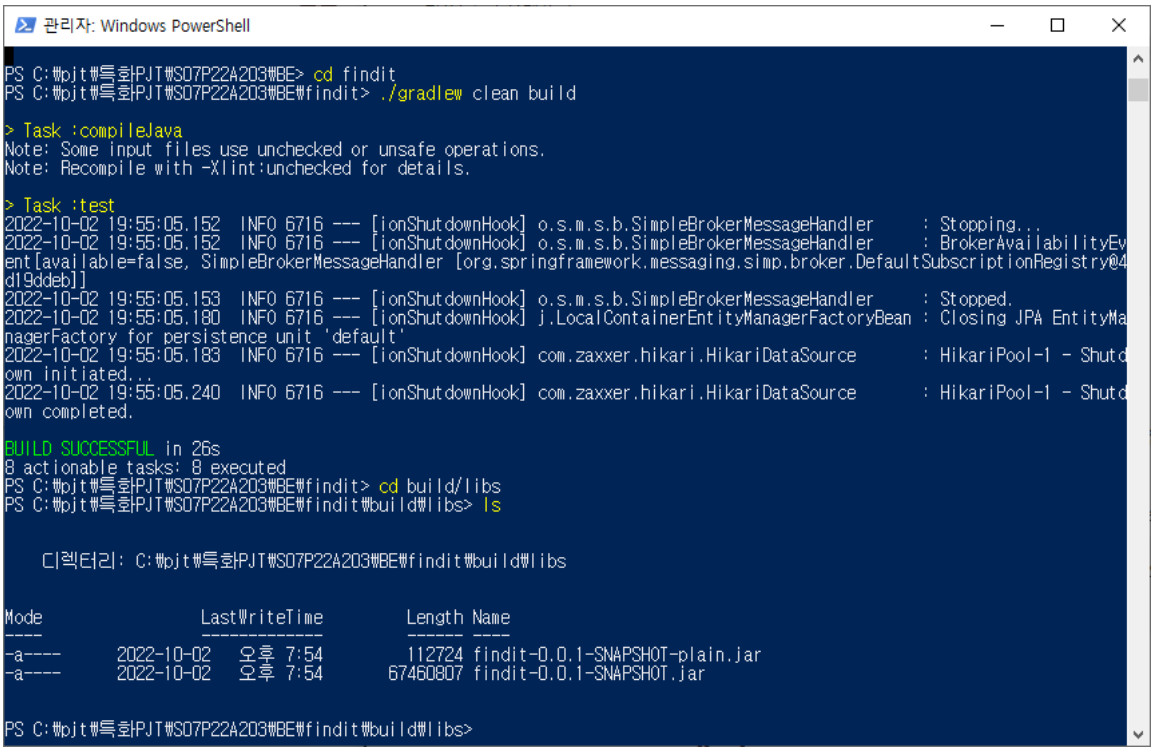
사용 툴 : VSCode `1.71.2`, Chrome

2. 빌드 방법

A. 백엔드 빌드 방법

1. Command Shell을 통해 프로젝트 폴더 안의 Back\sixback 폴더 안으로 이동한다.

2. `./gradlew clean build` 명령어를 통해 빌드한다.
3. 프로젝트 폴더 안의 Back\sixback\build\libs 안에 build 파일이 생성된다.



> 💡 주의! 빌드 하기 전에, appilcation.yaml 파일이 프로젝트 폴더 안의 Back\sixback\src\main\resources 폴더 안에 존재해야 한다.

B. 프론트엔드 빌드 방법

1. Node.js 환경에서 Front\soccer 디렉토리로 이동
2. `npm i` 를 통해 package-lock.json에 정의된 패키지를 다운로드
3. 해당 폴더에서 아래의 명령어를 입력하여 배포 버전 파일 생성

```
npm run build
```

4. soccer 디렉토리에 build 폴더가 생성됨
5. 생성된 build 폴더를 서버에 배포하여 사용

C. 배포 명령어 정리

1. 현재 실행 중인 서버 pid 확인

   ```
   ps -ef | grep java
   ```

   현재 실행중인 서버의 pid를 확인한다.

2. 실행 중인 서버 종료

   ```
   sudo kill -9 <pid>
   ```

만약 실행 중인 서버가 존재한다면, kill 명령어를 통해 종료한다.

3. 새로운 서버 백그라운드에서 실행

```
nohup java -jar sixback-0.0.1-SNAPSHOT.jar
```

BE 빌드 과정에서 생성된 빌드 파일의 경로로 이동해서 서버를 실행시킨다.

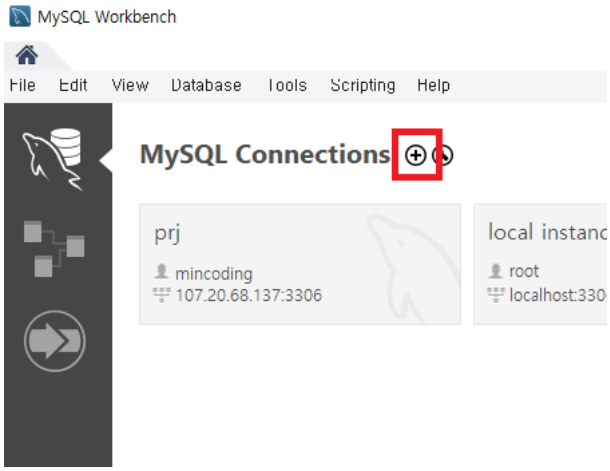
4. Nginx 재시작

```
sudo systemctl restart nginx
```

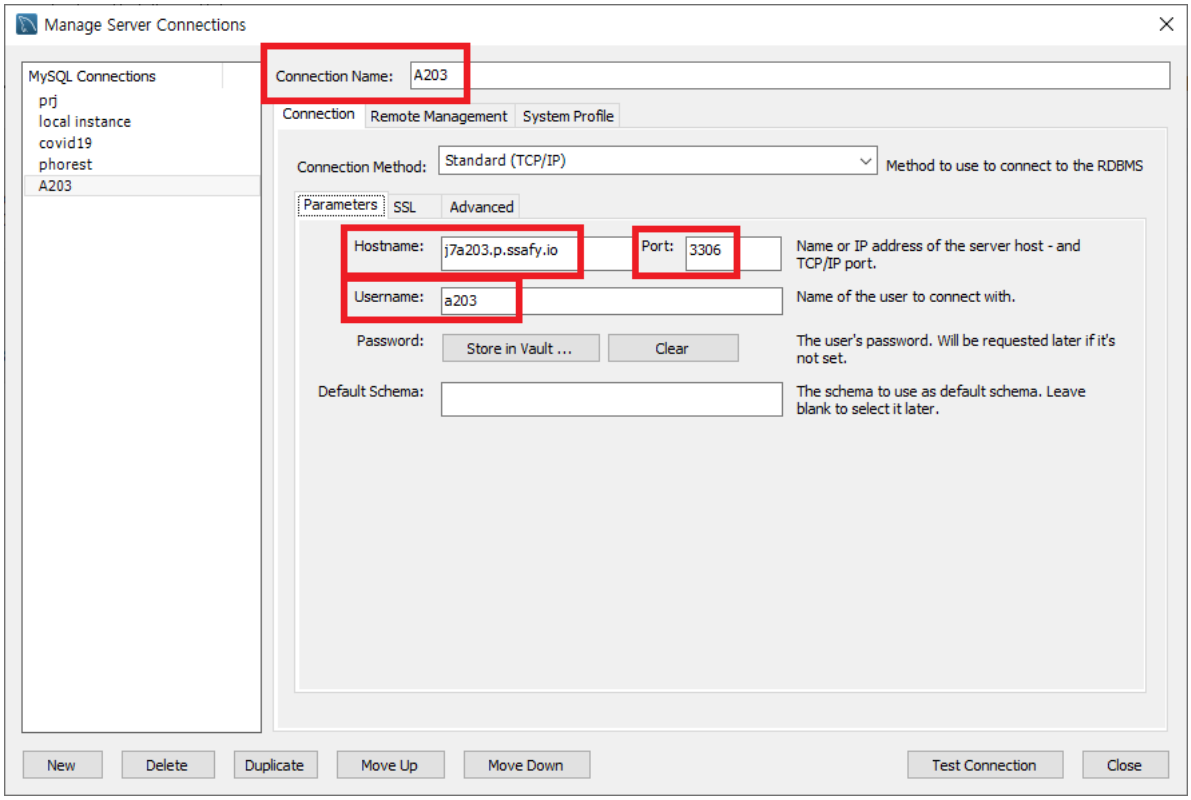
Nginx를 재시작한다.

3. DB 계정

A. MySQL WorkBench 추가하기



B. EC2 계정정보 넣기



- username : `a203` , password : `rhehdgnjs123`

  기존 root 계정이 아닌 별도의 a203 계정을 만들어서 진행했습니다.

4. 프로퍼트 정의

A. Git ignore 파일

```
spring:
  jpa:
    generate-ddl: 'true'
    hibernate:
      ddl-auto: update
    show-sql: 'false'
  profiles:
    include: key
    active: local
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://k7a203.p.ssafy.io:3306/six_back?useSSL=false&allowPublicKeyRetrieval=true&useUnicode=true&serverTimezone=Asia/Se
    username: a203
    password: rhehdgnjs123
    hikari:
      data-source-properties:
        rewriteBatchedStatements: 'true'
      maximum-pool-size: '20'
      pool-name: jpa-hikari-pool
            jdbc-url: ${spring.datasource.url}
            driver-class-name: ${spring.datasource.driver-class-name}
            username: ${spring.datasource.username}
            password: ${spring.datasource.password}

  data:
    mongodb:
      uri: mongodb://k7a203.p.ssafy.io:27017/six_back
      username: a203
      password: rhehdgnjs123

  redis:
```

```
    url: redis://k7a203.p.ssafy.io:6379
    port: '6379'

  security:
    oauth2:
      client:
        registration:
          google:
            client-id: 271554785368-punan5dpcia650n8mm462r3oqr4hkkko.apps.googleusercontent.com
            client-secret: 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
            scope: profile, email
          kakao:
            authorization-grant-type: authorization_code
            client-id: 9a7e7716d34ac02b96b4007c71bf8087
            client-secret: SQuA7enTnq5t3M1COo8aqP7p5yzq3DTv
            scope: account_email, profile_nickname, profile_image
            client-authentication-method: post
            # redirect-uri: http://localhost:8080/api/v1/login/oauth2/code/kakao
            redirect-uri: https://football-wellknown.com/api/v1/login/oauth2/code/kakao
            client-name: Kakao
        provider:
          kakao:
            token-uri: https://kauth.kakao.com/oauth/token
            user-name-attribute: id
            user-info-uri: https://kapi.kakao.com/v2/user/me
            authorization-uri: https://kauth.kakao.com/oauth/authorize

logging:
  level:
    com:
      amazonaws:
        util:
          EC2MetadataUtils: error

app:
  oauth2:
    # authorizedRedirectUris: http://localhost:3000/oauth/redirect
    authorizedRedirectUris: https://football-wellknown.com/oauth/redirect
  auth:
    refreshTokenExpiry: '604800000'
    tokenSecret: 926D96C90030DD58429D2751AC1BDBBC
    tokenExpiry: '1800000'

cloud:
  aws:
    s3:
      bucket:
        value: k7a203-ckeditor-image-upload
        url: https://s3.ap-northeast-2.amazonaws.com/k7a203-ckeditor-image-upload/
    region:
      static: ap-northeast-2
    credentials:
      accessKey: AKIAQXFVDSF7YR7GE27D
      secretKey: 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
    stack:
      auto: 'false'

cors:
  allowed-origins: http://localhost:3000,https://football-wellknown.com
  allowed-methods: GET,POST,PUT,DELETE,OPTIONS
  allowed-headers: '*'
  max-age: '3600'

sever:
  http:
    port: '8080'
server:
  port: '8080'
  servlet:
    context-path: /api/v1

#  ssl:
#    key-store: classpath:keystore.p12
#    key-store-type: PKCS12
#    key-store-password: Y1Ya6uu54UwkgwVAX6wV15iYfmvnk2mM

jwt:
  secret: adbonFDsgdGawfsdfWERPwasbdfbo342ruusdhvsdSGUIdsaff

SCHEDULER-SERVER: true


API-KEY: a67cd14e35bc9fe57a5134c095bc959c2b89956fc3c50d323b020ccef8a5bd05
```

1) application.yml 파일

- application.yml은 서버의 /var/jenkins/resources 폴더 안에 위치해야 한다.

5. EC2 설정

A. Docker 설치

참고 링크

1. 사전 패키지 설치

```
sudo apt-get update
sudo apt-get install -y ca-certificates \
    curl \
    software-properties-common \
    apt-transport-https \
    gnupg \
    lsb-release
```

2. gpg 키 다운로드

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

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

3. Docker 설치

`sudo apt update`

`sudo apt install docker-ce docker-ce-cli containerd.io docker-compose`

4. 젠킨스 사용을 위해 docker-compose.yml 작성

`vi docker-compose.yml`

```
version: '3'

services:
    jenkins:
        image: jenkins/jenkins:lts
        container_name: jenkins
        volumes:
            - /var/run/docker.sock:/var/run/docker.sock
            - /jenkins:/var/jenkins_home
        ports:
            - "9090:8080"
        privileged: true
        user: root
```

5. docker-compose.yml 실행

`sudo docker-compose up -d`

6. mysql 컨테이너

`sudo docker pull mysql`

```
sudo docker run -d -it -p 3306:3306 -v /var/mysql:/var/mysql --name mysql-container \
-e MYSQL_DATABASE=<> \
-e MYSQL_USER=<> \
-e MYSQL_PASSWORD=<> \
mysql
```

위 명령어를 간단하게 설명하면

- -d : 컨테이너를 데몬프로세스로 실행
- -it : 사용자가 입출력할 수 있고, 가상 터미널 환경에서 에뮬레이션
- -p : 호스트에서 선택한 포트
- —name : 컨테이너 이름
- -e : 환경변수
  - MYSQL_DATABASE : 사용할 데이터베이스
  - MYSQL_USER : 계정명
  - MYSQL_PASSWORD : 계정 비밀번호

7. redis 컨테이너

`sudo docker pull redis`

`sudo docker run -it -d -p 6379:6379 —-name redis-container redis`

8. mongo 컨테이너

`sudo docker pull mongo`

`sudo docker run -it -d -p 27017:27017 —-name mongodb-container mongo`

B. SSL 설정

1) Frontend

1. certbot 설치

   `sudo apt-get install certbot`

2. letsencrypt로 pem 키 생성

   `sudo certbot certonly --standalone -d <도메인>`

3. Nginx의 default.conf 변경

```
server {
    listen       80;
    listen  [::]:80;

    server_name _;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;

    server_name _;

    root   /usr/share/nginx/html;
    index  index.html index.htm;

    location / {
        try_files $uri $uri/ /index.html =404;
    }

    ssl_certificate /etc/letsencrypt/live/<도메인>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<도메인>/privkey.pem;
}
```

2) Backend

1. letsencrypt로 생성된 pem 키 위치로 이동

   `cd /etc/letsencrypt/live/<도메인>`

2. OpenSSL로 pkcs12키 생성

   `sudo openssl pkcs12 -export -inkey privkey.pem -in fullchain.pem -out keystore.p12`

3. Spring Boot에 코드 추가

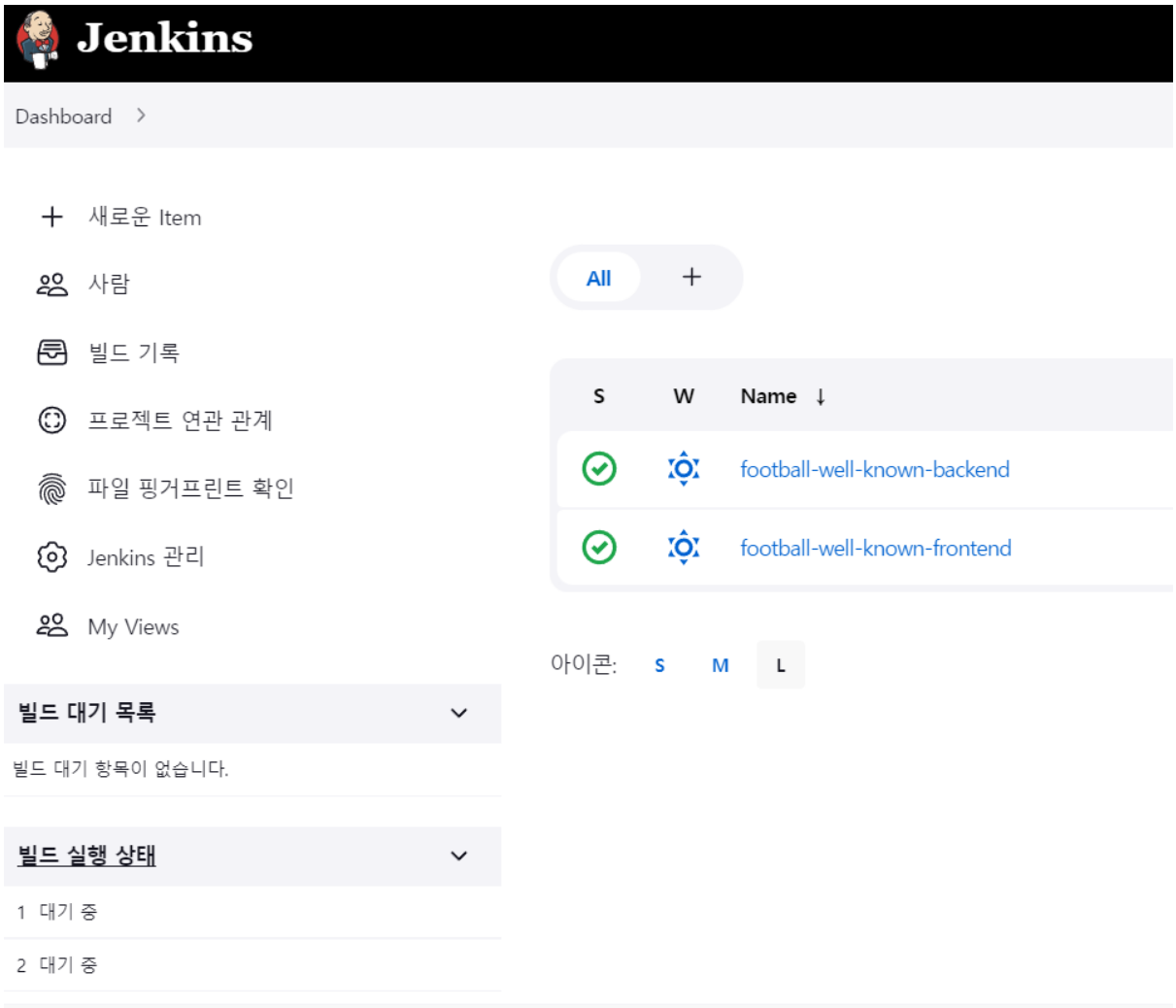
```
server:
  ssl:
    key-store: classpath:keystore.p12
    key-store-type: PKCS12
    key-store-password: <설정한 비밀번호>
```

```
server.ssl.key-store: classpath:keystore.p12
server.ssl.key-store-type: PKCS12
server.ssl.key-store-password: <설정한 비밀번호>
```

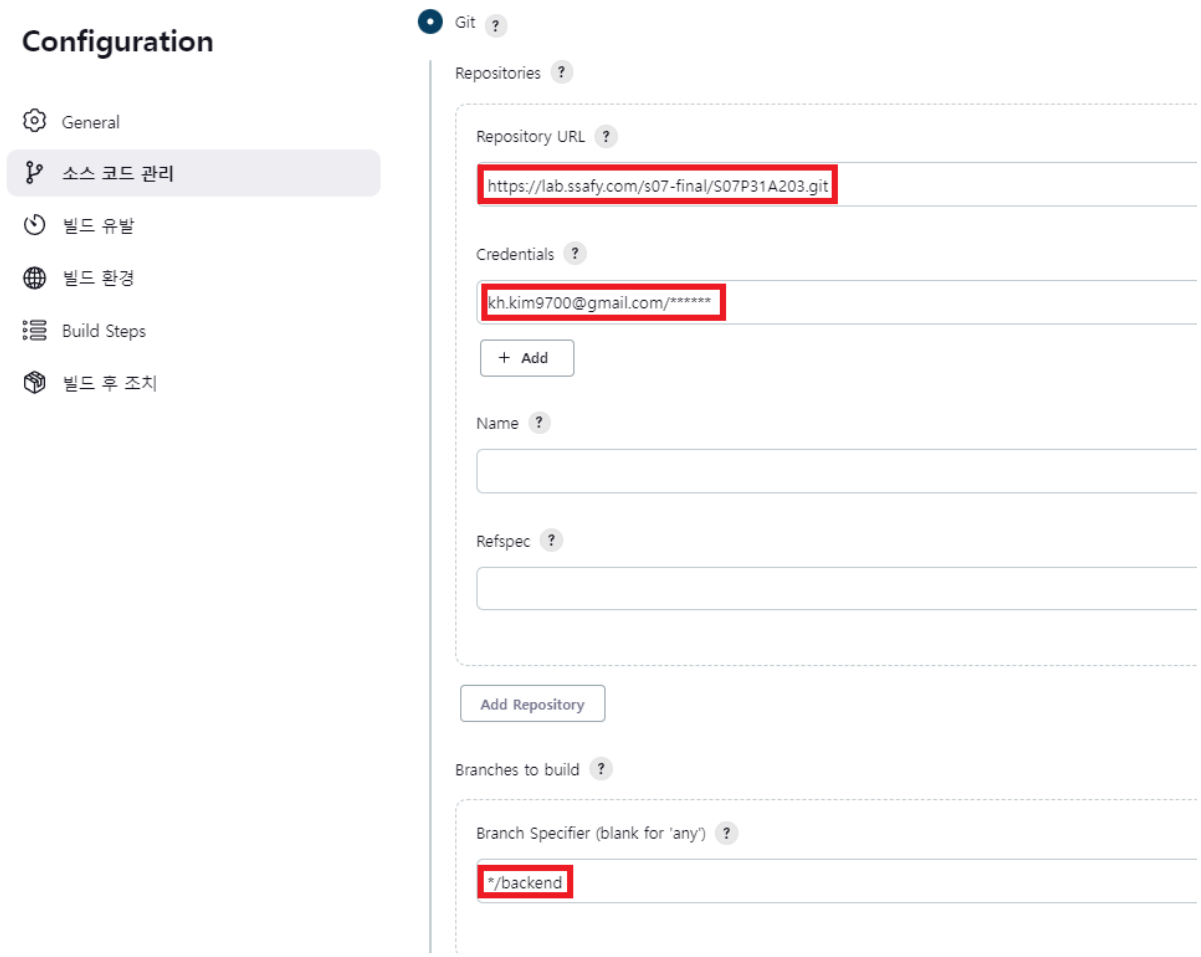
6. Jenkins 설정

Jenkins를 이용해 CICD 환경을 구축, 개발 과정 중 약 000번의 빌드와 배포를 진행하였습니다.

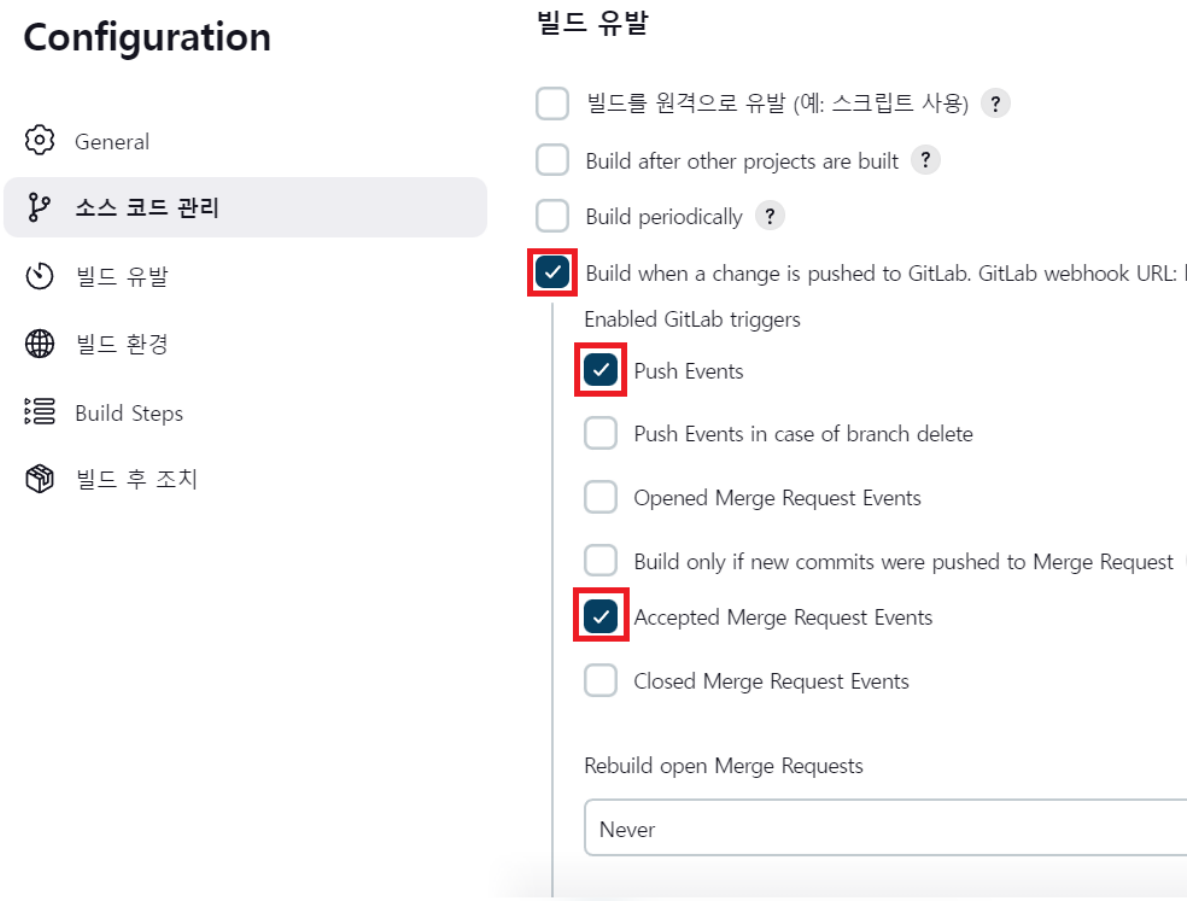
A. Jenkins 구성



- FE와 BE의 빌드를 따로 하기 위해서 Item을 둘로 나눠서 생성해줬습니다.

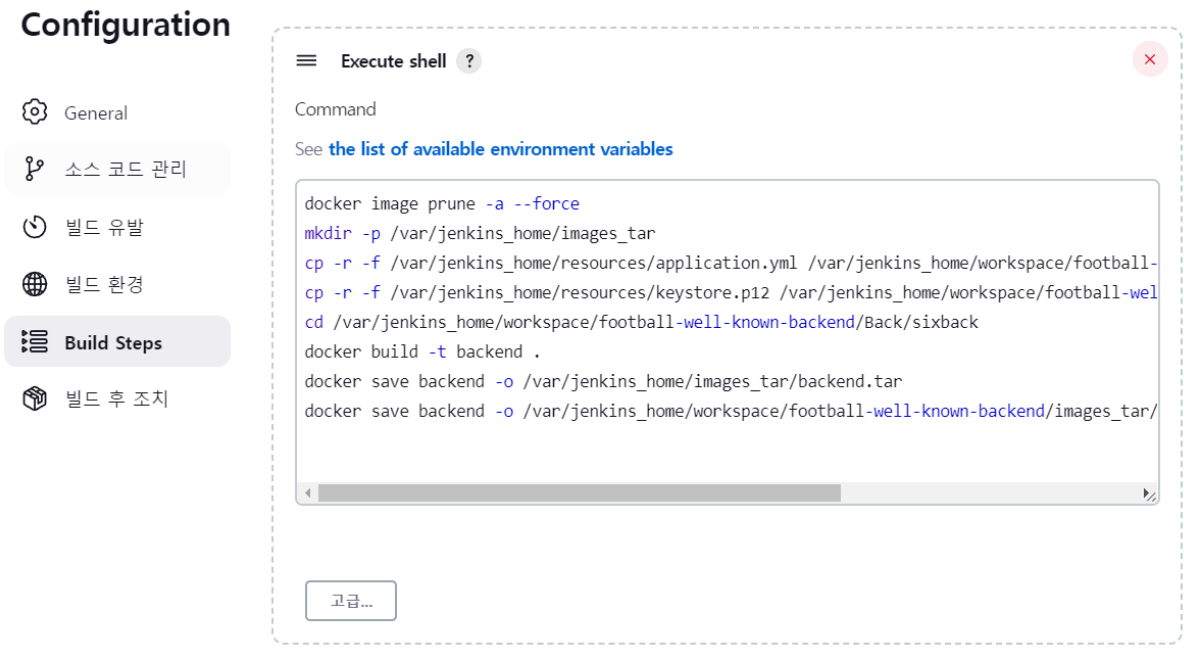


- 소스 코드 관리에서, Git을 선택하고 Repository에는 현재 개발하는 프로젝트의 Repository 주소를 입력합니다. Credentials에는 add를 통해 Gitlab에서 사용하는 아이디 비밀번호를 입력 한 후, 선택해줍니다. Branch Specifier에는 변화를 감지할 branch를 선택하는 곳입니다. 저희는 backend branch와 frontend branch를 선택했습니다.



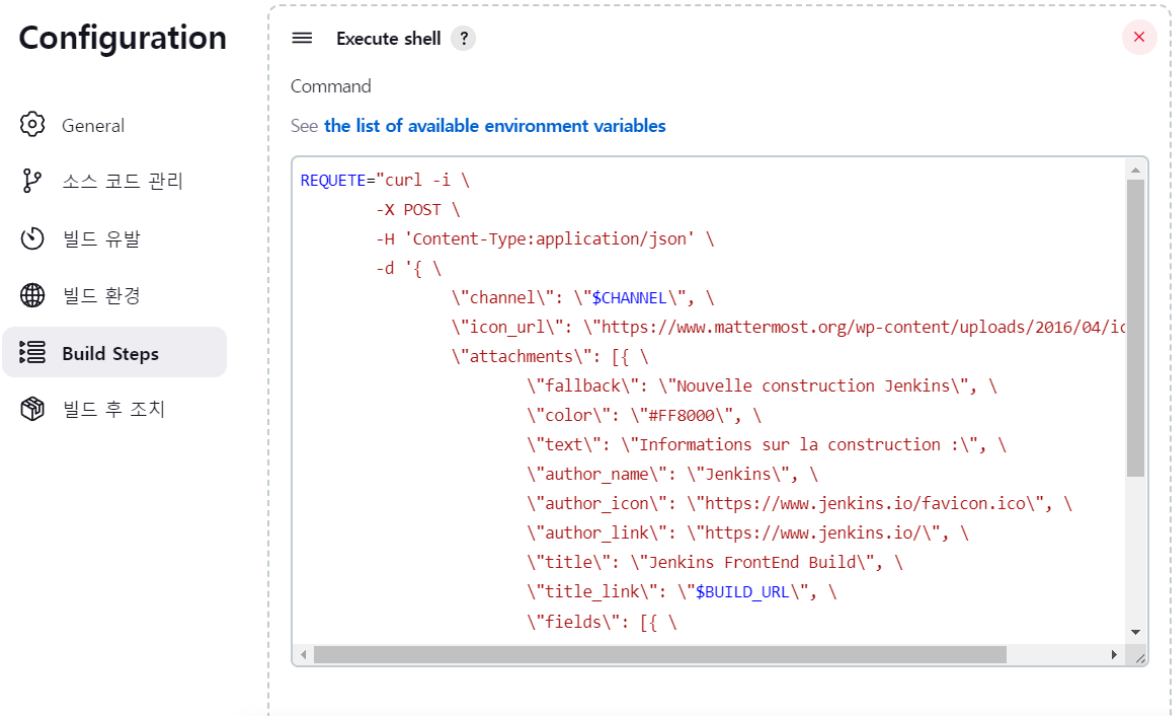
- 빌드 유발에서, webhook을 통해 빌드를 유발하기 위해 Build when a change is pushed to GitLab 부분을 체크해주었고, Push Events가 발생했을 때와 Merge Request Events가 허용되었을 때 빌드를 유발하였습니다.

- 빌드 유발의 고급 탭을 눌러서 나오는 Secret token을 Generate 한 후, 이후 GitLab webhook 설정에 사용하였습니다.

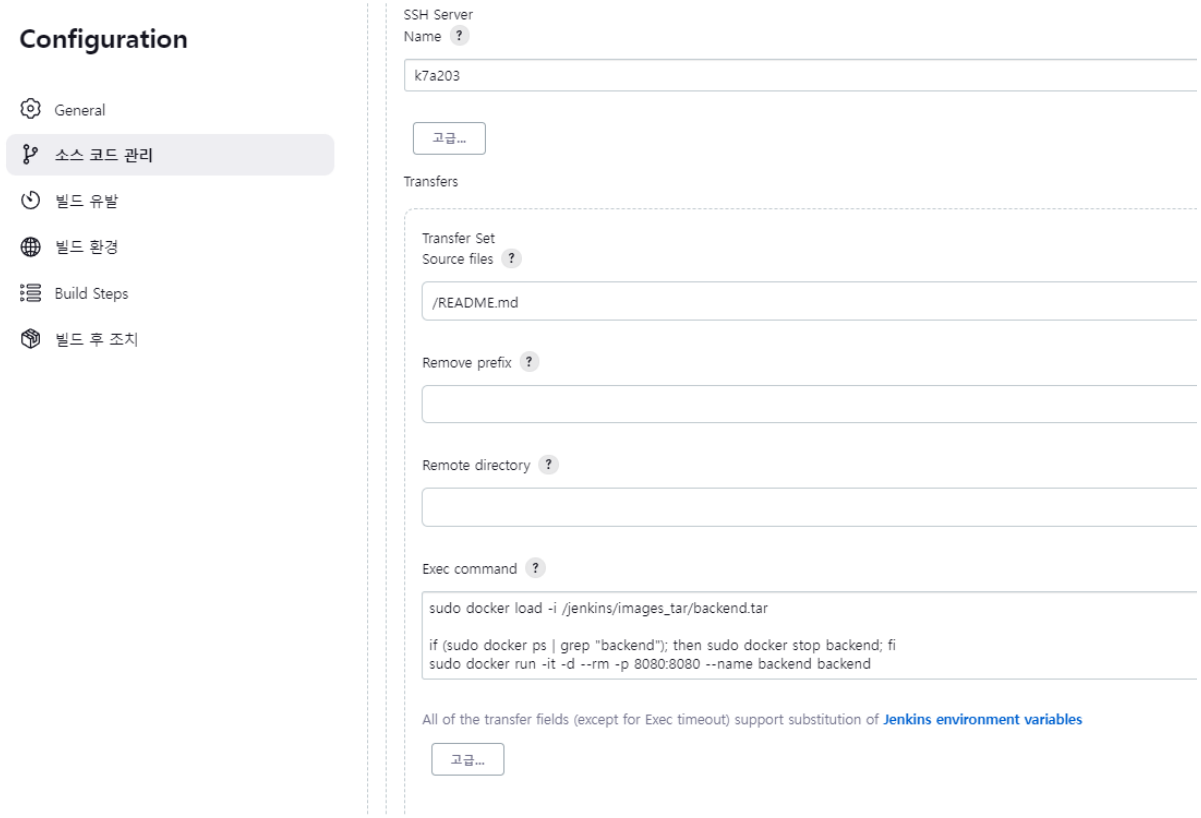


Backend

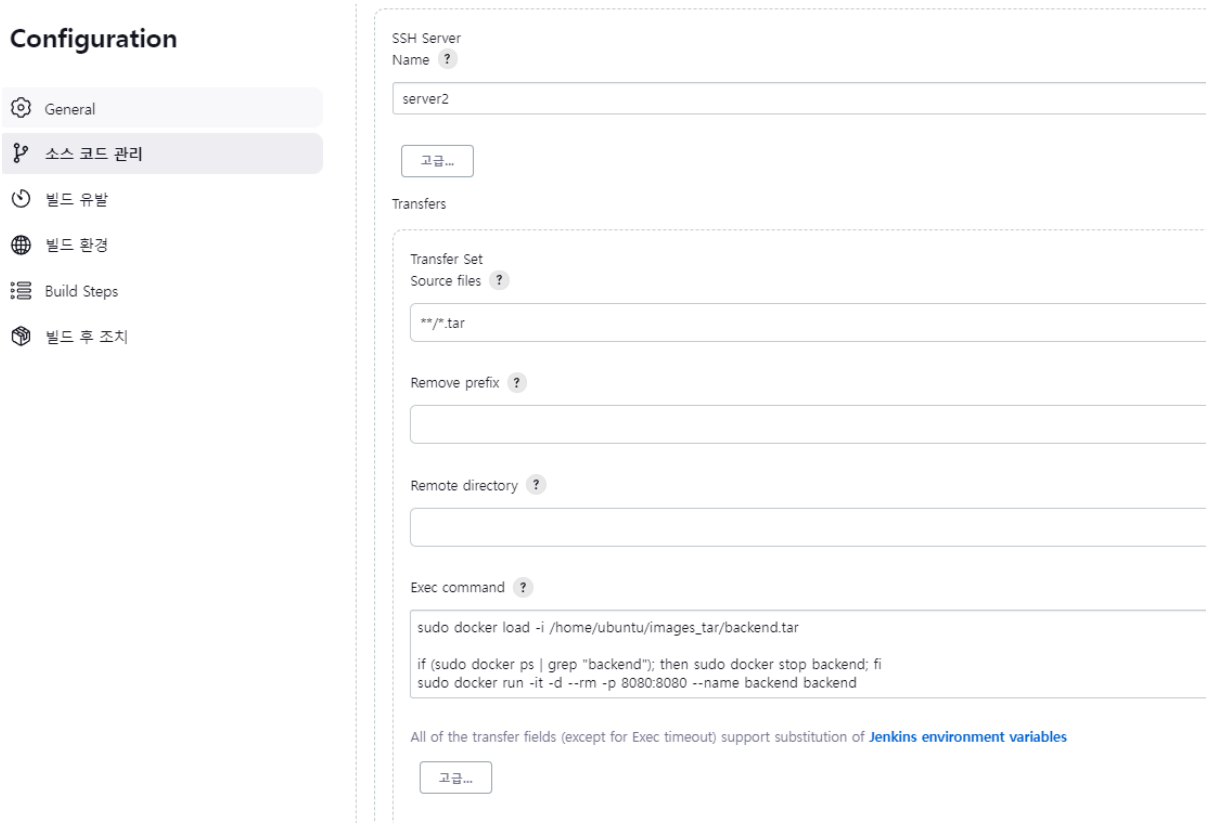
- Build 탭에서 Execute shell을 선택하고, 리눅스 명령어와 도커를 이용하셔 빌드하였습니다.



- 빌드가 완료된 후 Mattermost 채널로 알림을 보내도록 Execute shell에 curl을 이용하여 코드를 작성했습니다.



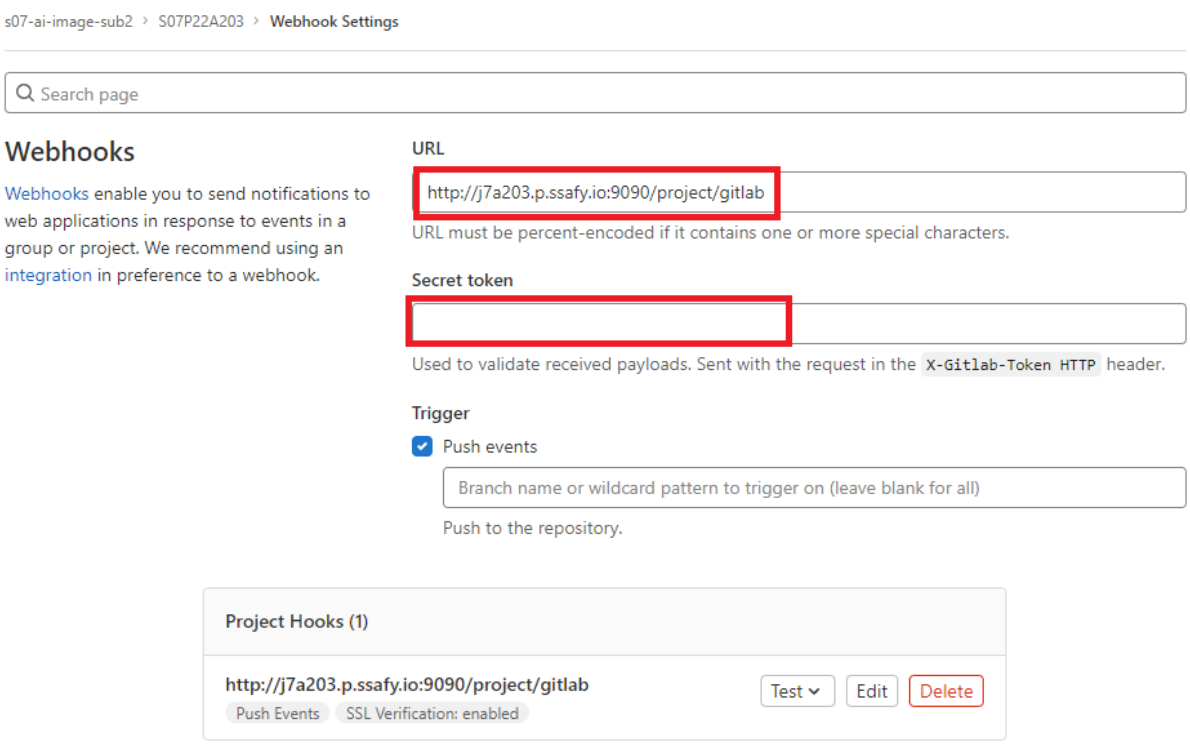
AWS-1



AWS-2

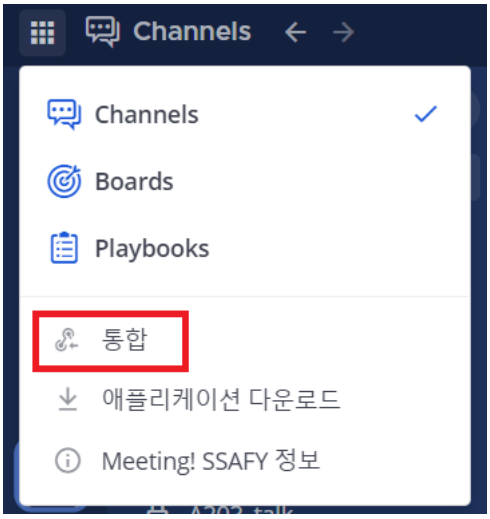
- SSH를 이용하여 서버 2대에 배포를 하였습니다.

B. GitLab 설정

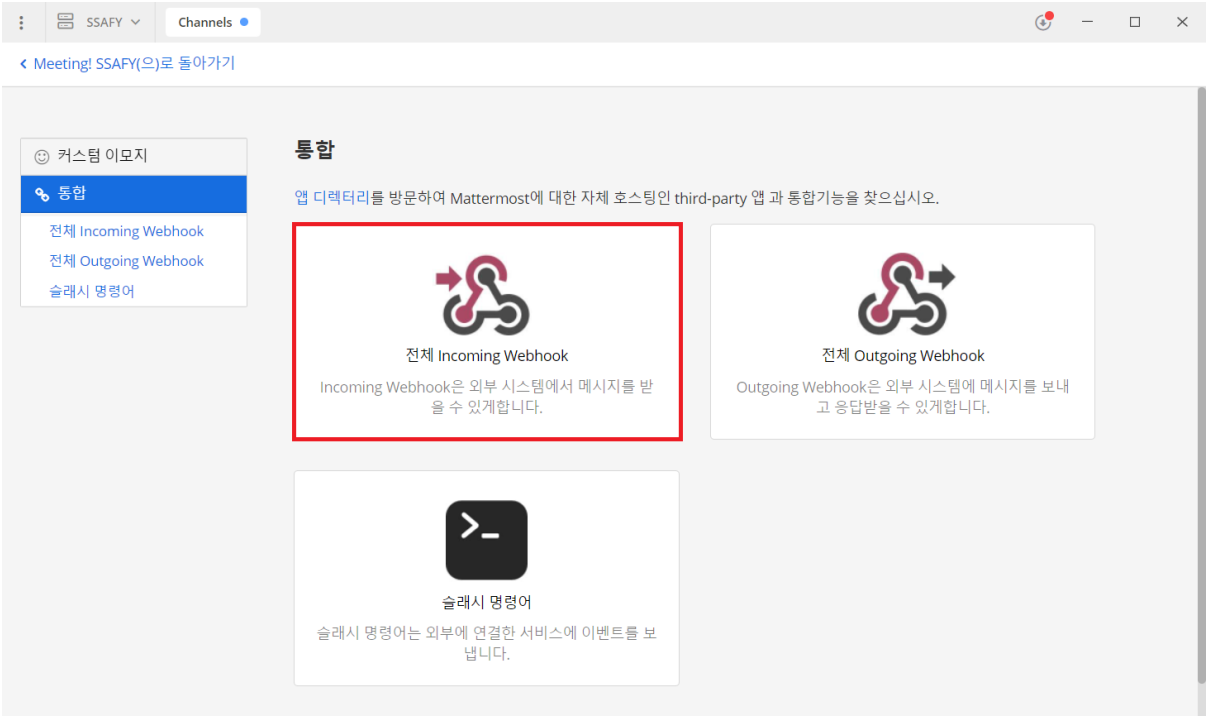


- GitLab repository의 설정의 Webhook 탭에서 URL과 jenkins에서 얻은 webhook을 위한 Secret token을 입력하고, Push event가 발생했을 때 web hook이 되도록 설정하였습니다.

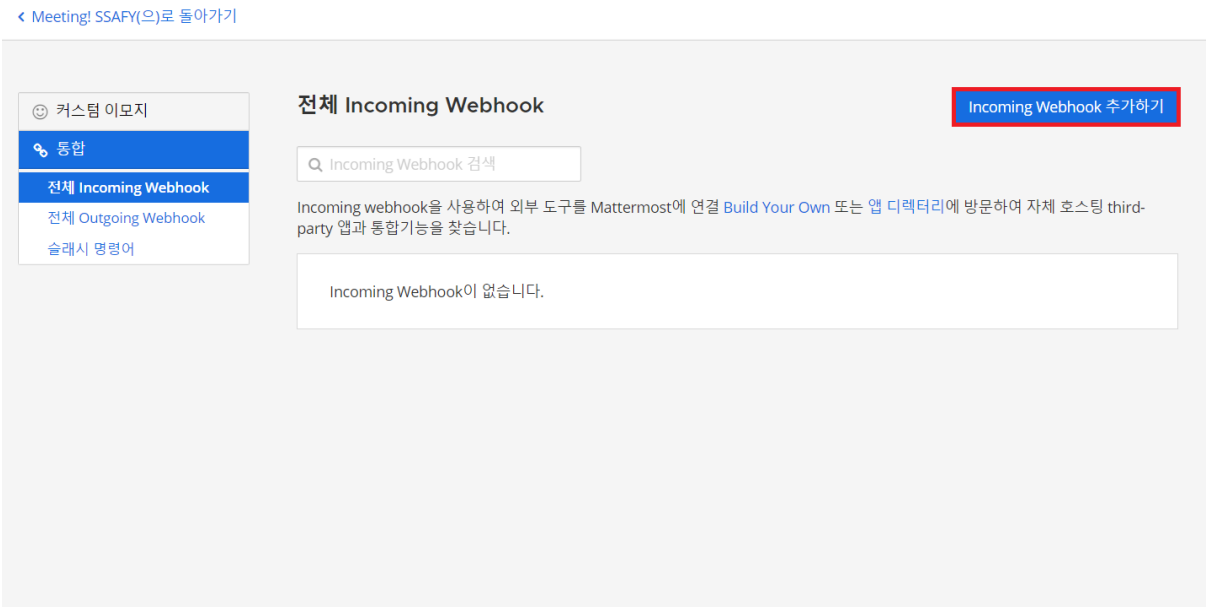
C. MatterMost 설정



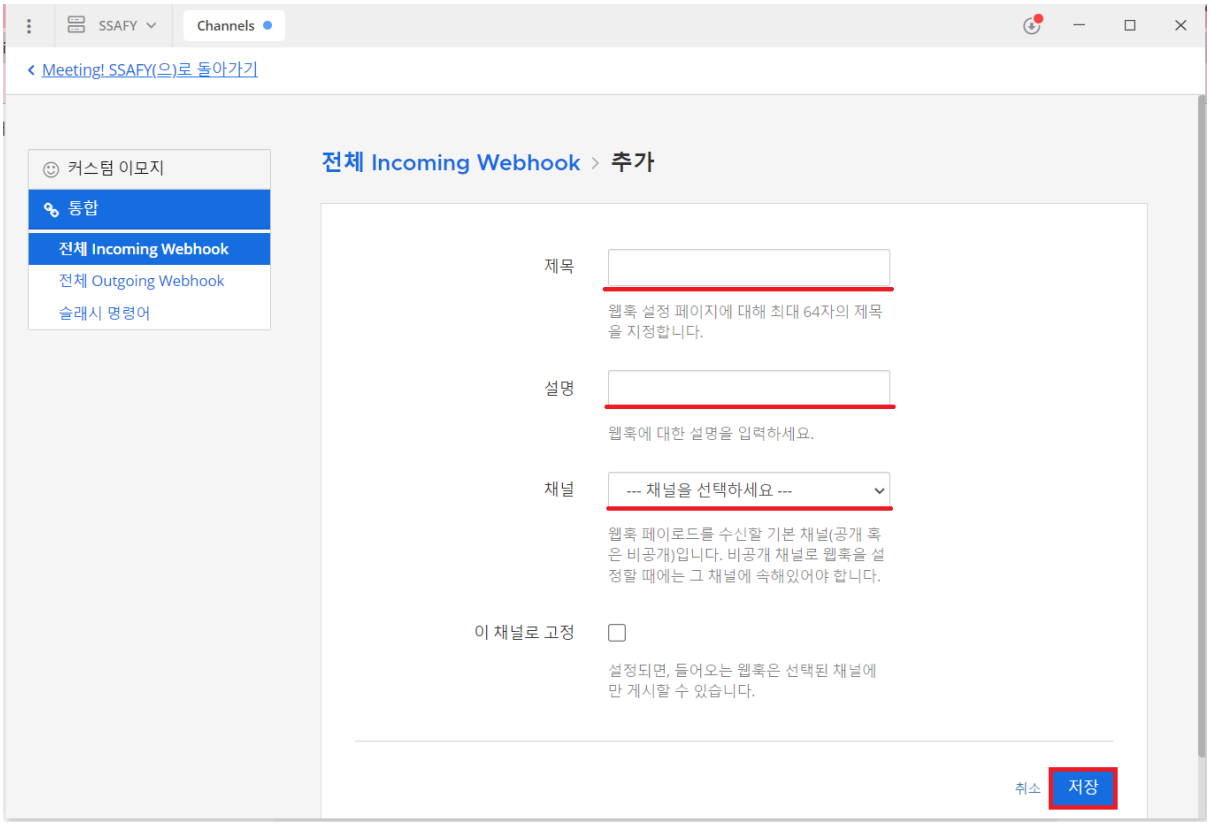
- Mattermost에 설정에서 통합을 선택합니다.



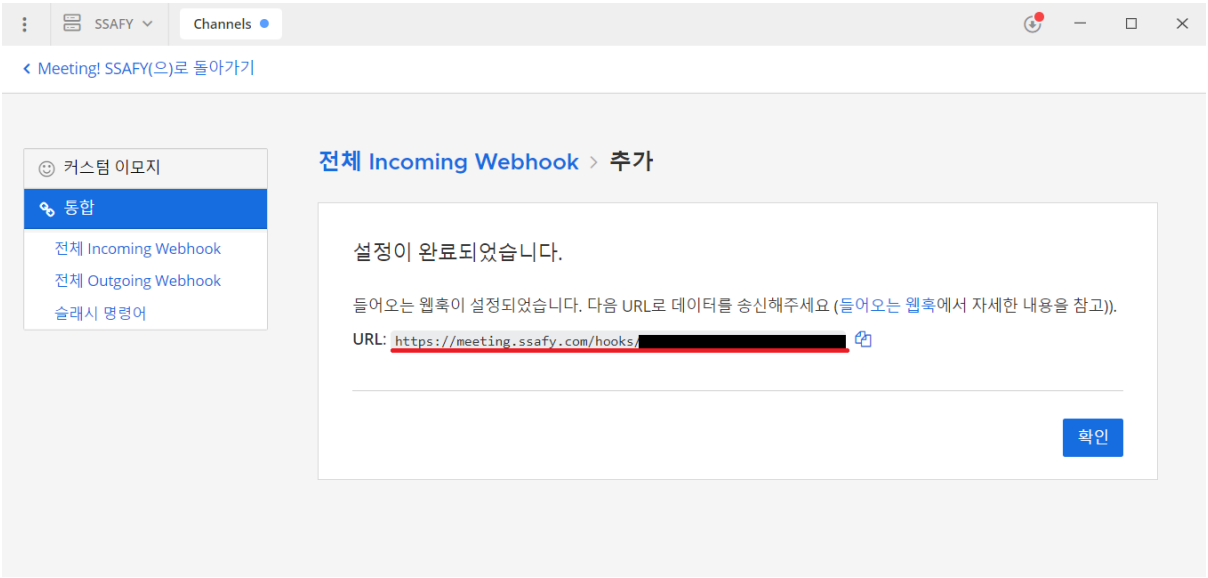
- Incoming Webhook을 선택합니다.



- Incoming Webhook 추가하기를 선택합니다.



- 제목, 설명을 작성하고 알림이 올 채널을 선택한 후, 저장을 선택합니다.



- 설정된 URL을 Execute shell의 Mattermost Webhook URL에 넣습니다.

D. Jenkins 빌드, 배포 명령어

1) Frontend

```
docker image prune -a --force
mkdir -p /var/jenkins_home/images_tar
```

```
cd /var/jenkins_home/workspace/football-well-known-frontend/Front/soccer
docker build -t frontend .
docker save frontend -o /var/jenkins_home/images_tar/frontend.tar
```

```
sudo docker rmi frontend
sudo docker load -i /jenkins/images_tar/frontend.tar

if (sudo docker ps | grep "frontend"); then sudo docker stop frontend; fi
sudo docker run -it -d --rm -p 80:80 -p 443:443 -v /etc/letsencrypt:/etc/letsencrypt --name frontend frontend
```

2) Backend

```
docker image prune -a --force
mkdir -p /var/jenkins_home/images_tar
cp -r -f /var/jenkins_home/resources/application.yml /var/jenkins_home/workspace/football-well-known-backend/Back/sixback/src/main/res
cp -r -f /var/jenkins_home/resources/keystore.p12 /var/jenkins_home/workspace/football-well-known-backend/Back/sixback/src/main/resour
cd /var/jenkins_home/workspace/football-well-known-backend/Back/sixback
docker build -t backend .
docker save backend -o /var/jenkins_home/images_tar/backend.tar
docker save backend -o /var/jenkins_home/workspace/football-well-known-backend/images_tar/backend.tar
```

```
sudo docker load -i /jenkins/images_tar/backend.tar

if (sudo docker ps | grep "backend"); then sudo docker stop backend; fi
sudo docker run -it -d --rm -p 8080:8080 --name backend backend
```

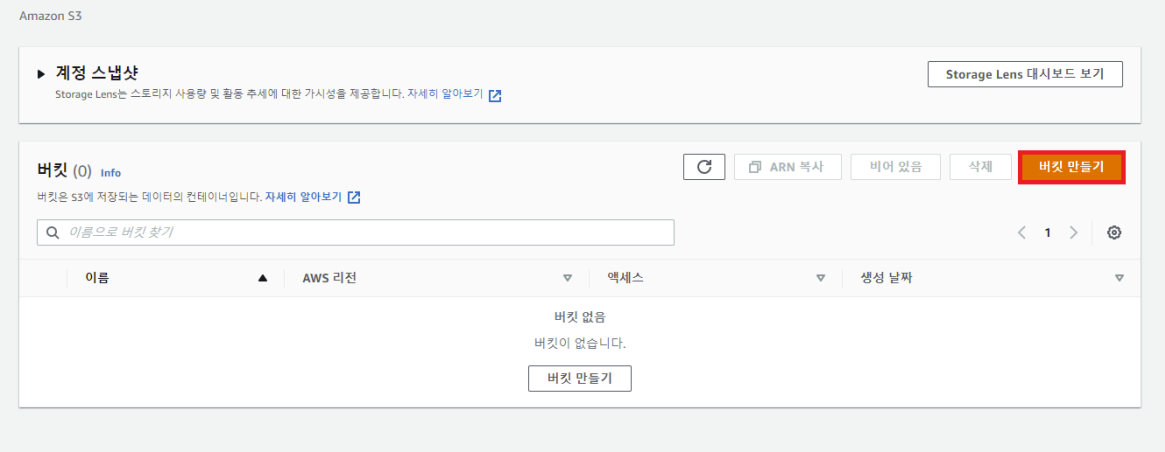
3) Mattermost

```
REQUETE="curl -i \
        -X POST \
        -H 'Content-Type:application/json' \
        -d '{ \
                \"channel\": \"$CHANNEL\", \
                \"icon_url\": \"https://www.mattermost.org/wp-content/uploads/2016/04/icon.png\", \
                \"attachments\": [{ \
                        \"fallback\": \"Nouvelle construction Jenkins\", \
                        \"color\": \"#FF8000\", \
                        \"text\": \"Informations sur la construction :\", \
                        \"author_name\": \"Jenkins\", \
                        \"author_icon\": \"https://www.jenkins.io/favicon.ico\", \
                        \"author_link\": \"https://www.jenkins.io/\", \
                        \"title\": \"Jenkins FrontEnd Build\", \
                        \"title_link\": \"$BUILD_URL\", \
                        \"fields\": [{ \
                                \"short\":false, \
                                \"title\":\"Details\", \
                                \"value\":\"$BUILD_URL\" \
                        }] \
        }] \
        }'\
        <Mattermost Webhook URL>"
 eval $REQUETE
```

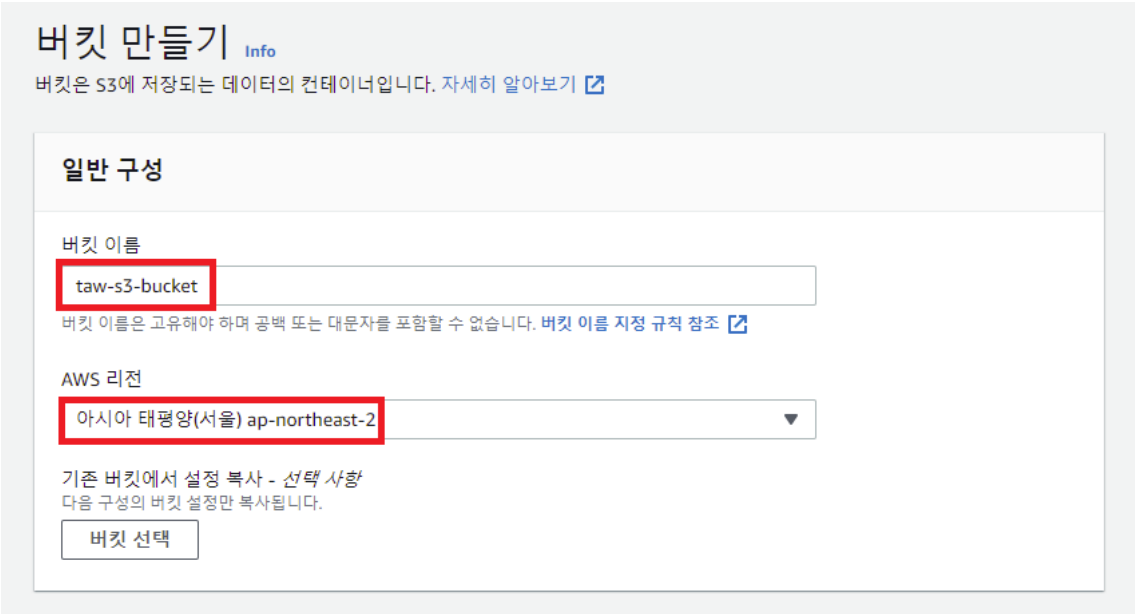
7. 외부 서비스

A. AWS S3

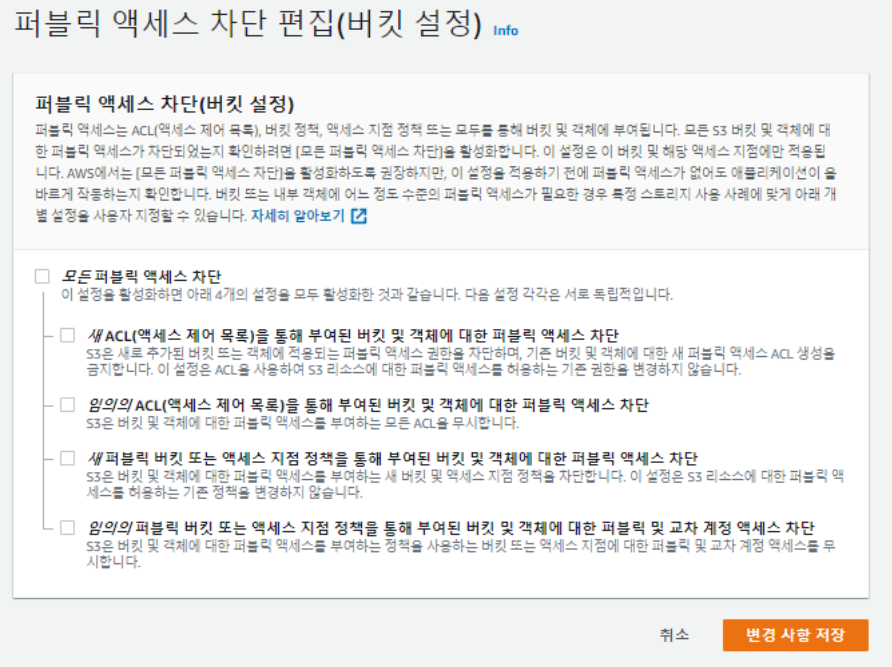
1. 버킷 만들기 클릭



2. 버킷 이름, 리전 입력



3. 퍼블릭 액세스 설정 : 체크 모두 해제

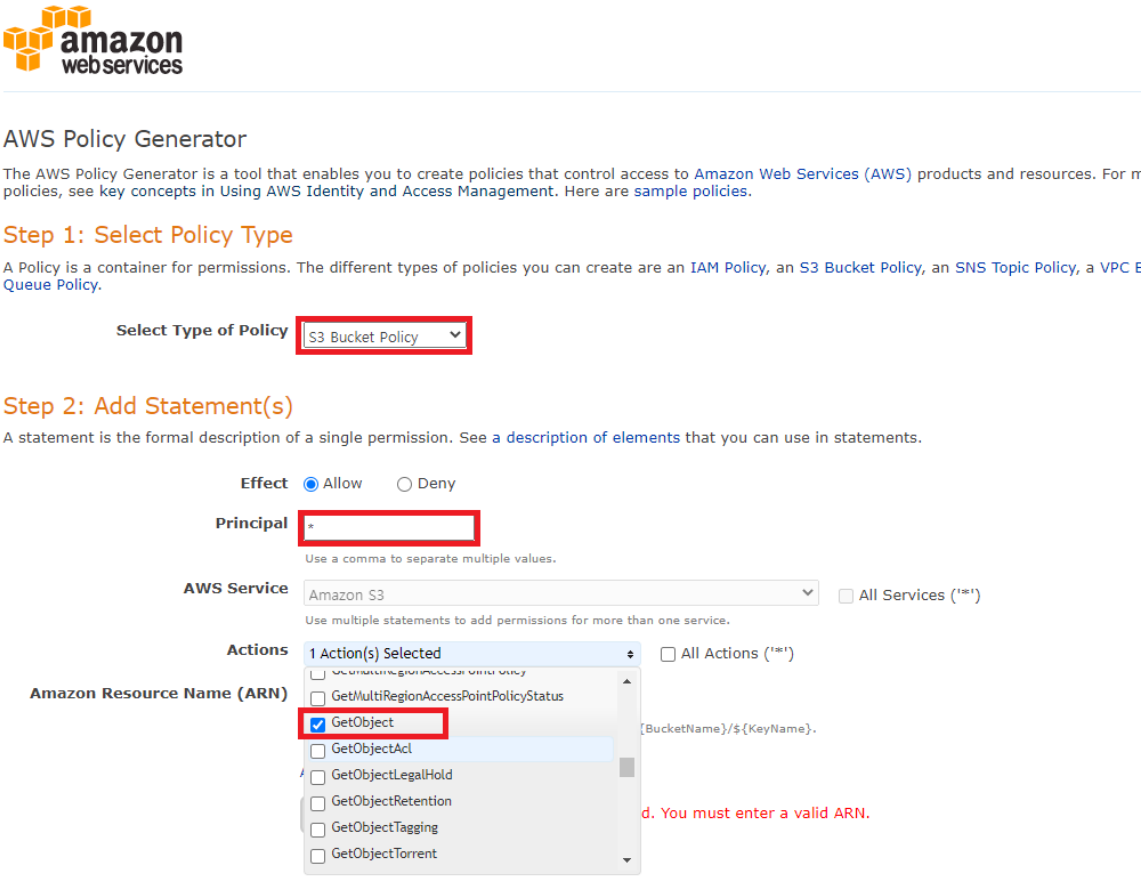


4. 버킷 생성 확인

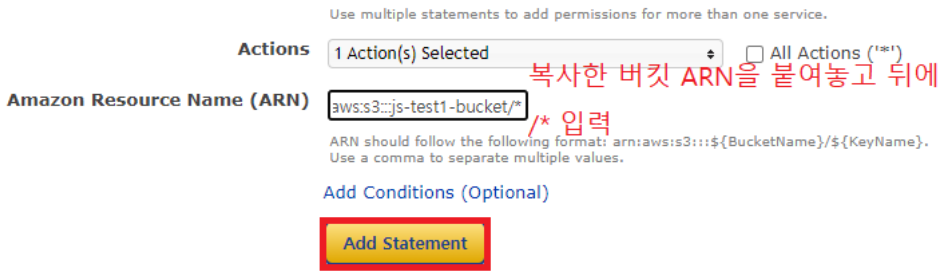
5. 버킷 정책 편집


버킷 정책 편집 Info
버킷 정책
JSON으로 작성된 버킷 정책은 버킷에 저장된 객체에 대한 액세스 권한을 제공합니다. 버킷 정책은 다른 계정이 소유한 객체에는 적용되지 않습니다. 자세히 알아보기
정책 예제
정책 생성기
2
버킷 ARN
arn:aws:s3:::js-test1-bucket
정책
1
1
외부 액세스 미리 보기

6. 버킷 정책 생성



7. 버킷 정책 생성(2) : 복사한 `버킷 ARN`을 붙여넣기 한 후에 `/*`을 추가 → Add Statement 클릭

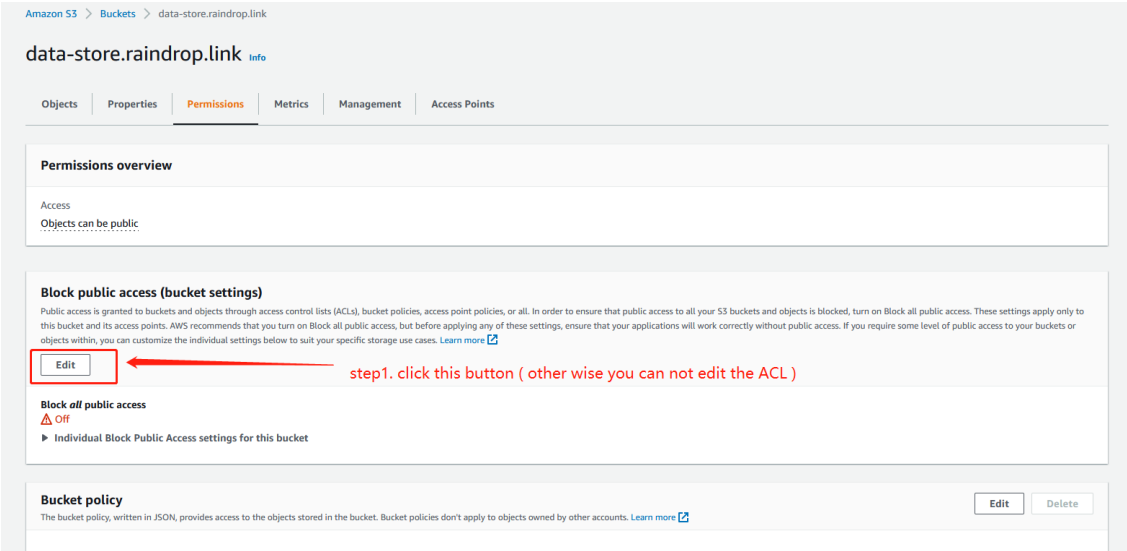


8. 버킷 정책 생성(3)

9. 버킷 정책 생성(4)

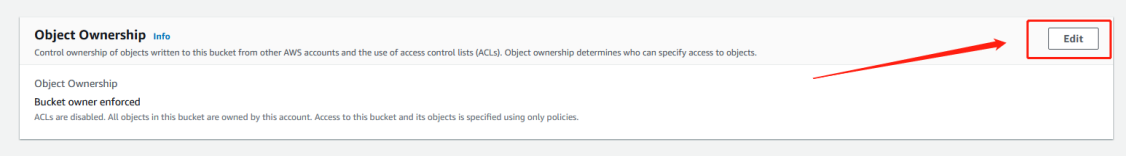
10. 버킷 정책 편집 적용

- ERROR : “The bucket does not allow ACLs”
  1. assume you have created the s3 bucket, in the list page

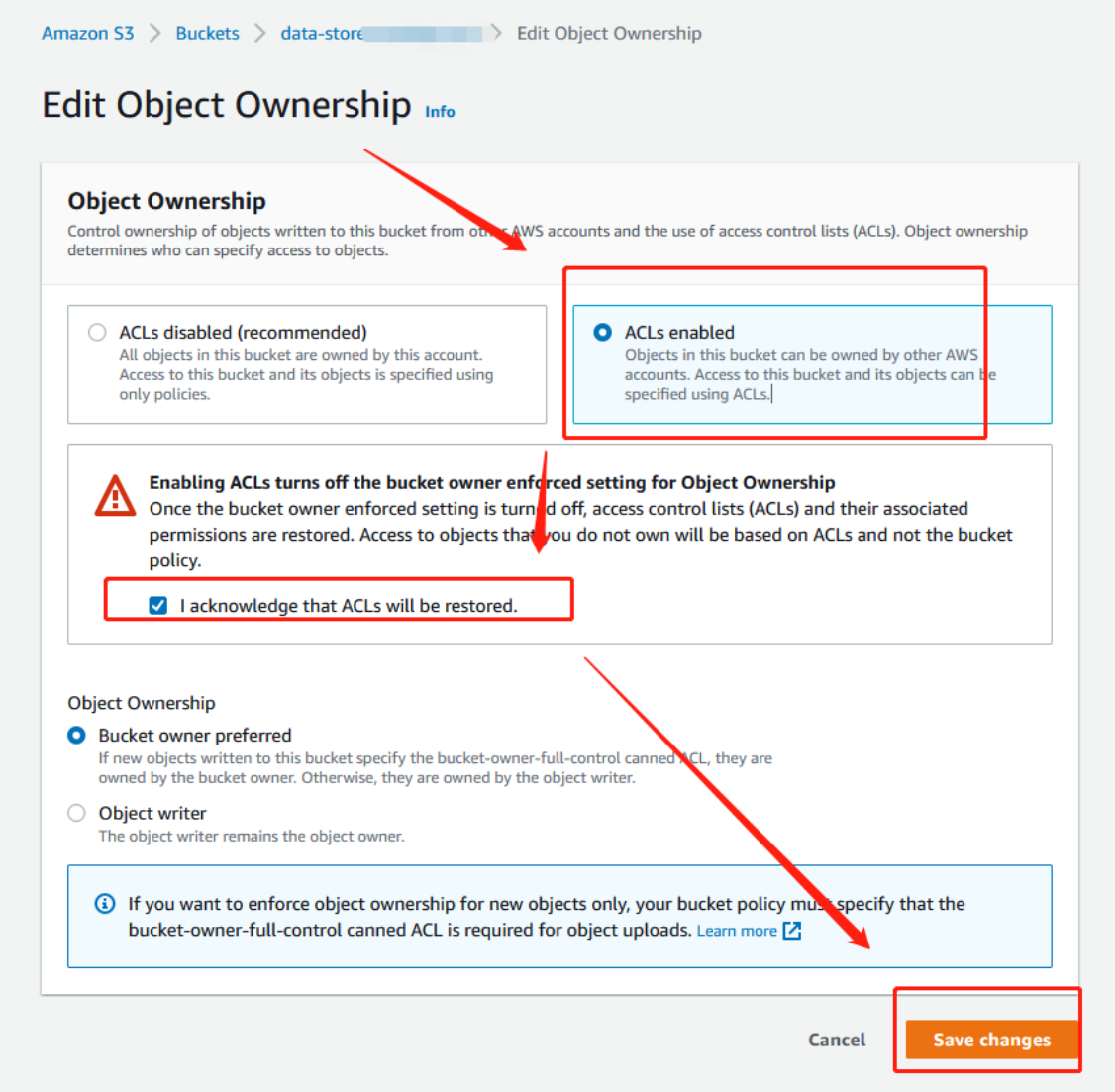


  2. don’t toggle the “block” options

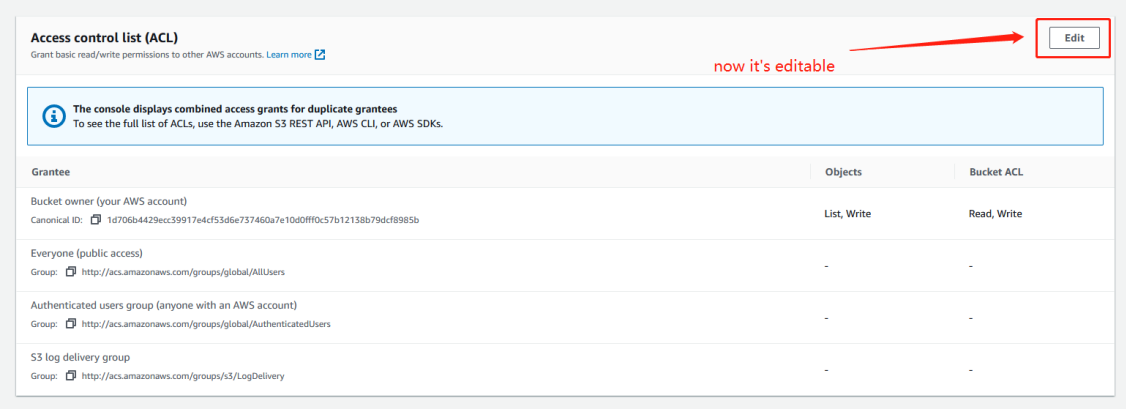
3. find the ownership, then click edit.



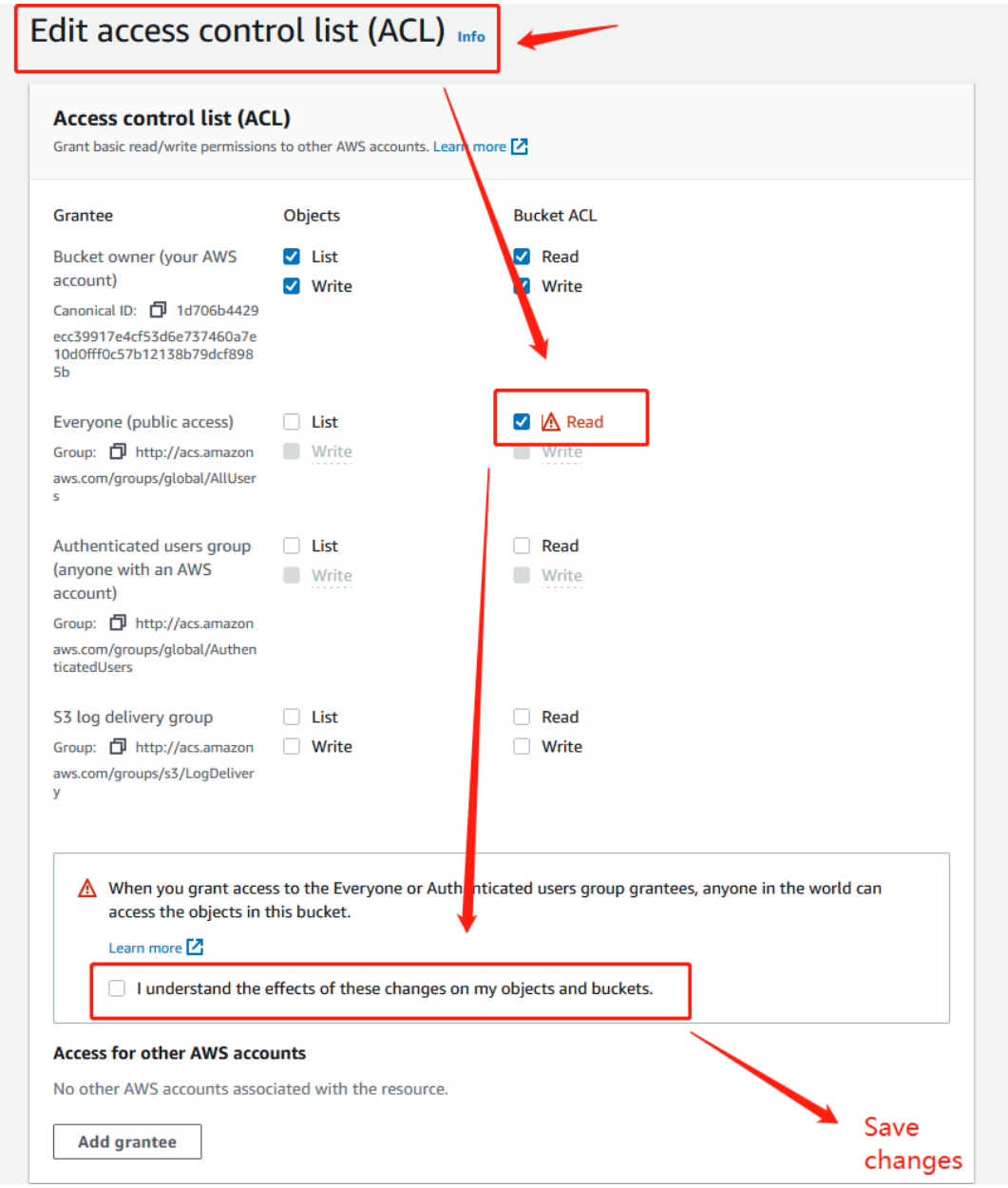
4. edit the object owner ship (ACLs enabled)



5. now the edit button for ACL is clickable.



6. toggle the permissions you want and save changes.



B. 소셜 로그인

구글 로그인

[Spring Boot] OAuth2 소셜 로그인 가이드 (구글, 페이스북, 네이버, 카카오)

웹 또는 앱 서비스에서 로그인을 구현하는 것은 간단하지 않은 일입니다. 로그인을 구현하기 위해서는 다양한 사전 지식들을 가지고 있어야합니다. 특히 세션이나 쿠키 등의 역할 등을 알아야하고, 보안적인 측면에서도 신경을 써주어야 합니다. 하지만 로그인을 구현하기 위해서 개발 시간을 단축시켜줄 수 있는 것이 있다면 어떨까요?

https://deeplify.dev/back-end/spring/oauth2-social-login#%EC%A0%84%EC%B2%B4-%EC%8B%9C%ED%80%80%EC%8A%A4-%EB%8B%A4%EC%9D%B4%EC%96%B4%EA%B7%B8%EB%9E%A8

DEEPLIFY

카카오 로그인

Kakao Developers

이 문서는 REST API를 사용한 로그인 구현 방법을 안내합니다. 이 문서에 포함된 기능 일부는 [도구] > [REST API 테스트]를 통해 사용해 볼 수 있습니다. 카카오 로그인 구현에 필요한 로그인 버튼 이미지는 [도구] > [리소스 다운로드]에서 제공합니다. 해당 로그인 버튼은 디자인 가이드를 참고하여 서비스 UI에 적합한 크기로 수정하여 사용할 수

https://developers.kakao.com/docs/latest/ko/kakaologin/rest-api#before-you-begin-process

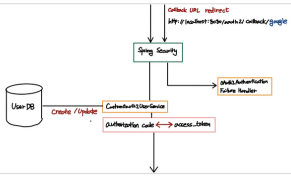
kakao developers

Spring Security

Spring Security OAuth2 Login Flow

www.callicoder.com/spring-boot-security-oauth2-social-login-part-2/ Spring Boot OAuth2 Social Login with Google, Facebook, and Github - Part 2 Integrate social login with Facebook, Google, and Github in your spring boot application using Spring Security's OAuth2 functionalities. You'll also add email and password

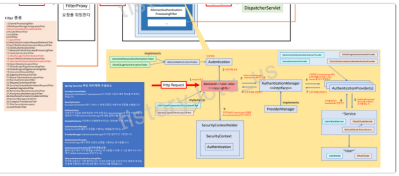
https://jyami.tistory.com/121



[Spring Security] OAuth 구글 로그인하기

목차 [이전 게시글] 꼭! 봐주세여 [Spring Security] 동작방법 및 Form, OAuth 로그인하기 (Feat.Thymeleaf 타임리프) 목차 Spring Security란? Spring을 사용할 때 애플리케이션에 대한 인증, 권한 부여 등의 보안 기능을 제공하는 프레임워크이다. 다양한 로그인 방법(Form태그, OAuth2, JWT...)에 대해 Spring이 어느정도 구현

https://lotuus.tistory.com/79



C. replication

1. 환경 생성
   a. (cmd창에서) cluster 생성

```
docker network create cluster --subnet-192.168.0.0/16
```

   b. mysql 이미지 생성

```
docker run -d -p 3306:3306 --net=cluster --name=master -v /var/mysql:/var/mysql --ip=192.168.0.2 -it -e MYSQL_ROOT_PASSWORD={
docker run -d -p 4306:3306 --net=cluster --name=slave1 -v /var/mysql:/var/mysql --ip=192.168.0.3 -it -e MYSQL_ROOT_PASSWORD={
docker run -d -p 5306:3306 --net=cluster --name=slave2 -v /var/mysql:/var/mysql --ip=192.168.0.4 -it -e MYSQL_ROOT_PASSWORD={
...
```

2. Master
   a. 계정 생성

```
CREATE USER '{유저명}'@'%' IDENTIFIED BY '{비밀번호}';
GRANT REPLICATION SLAVE ON *.* TO '{유저명}'@'%';
```

   b. 권한 부여

```
GRANT REPLICATION SLAVE ON *.* TO '{유저명}'@'%';
```

   c. LOG_FILE, LOG_POSITION 확인

```
SHOW MASTER STATUS;
```

   d. (bash창에서) dump file 생성

```
mysqldump -u root {DB명} -p > /var/mysql/dbdump.sql
```

3. Slave

a. (bash창에서) 백업

```
mysqldump -u root -p {DB명} < /var/mysql/dbdump.sql
```

b. server_id : Master와 다르게 설정

```
SHOW VARIABLES LIEK 'server_id';
SET GLOBAL server_id={숫자};
```

c. master 설정

```
CHANGE MASTER TO
    > MASTER_HOST='192.168.0.2',
    > MASTER_USER='{유저명}',
    > MASTER_PASSWORD='{비밀번호}',
    > MASTER_LOG_FILE='{LOG_FILE}'
    > MASTER_LOG_POS={LOG_POSITI},
    > GET_MASTER_PUBLIC_KEY=1;
```

d. 시작

```
START SLAVE;
```

e. 결과 확인

```
SHOW SLAVE STATUS;
```

ref :

https://velog.io/@max9106/DB-Spring-Replication

[MySQL] Replication 적용기 - 1

[Spring] Replication 적용기 - 2