SeniorNaver 포팅매뉴얼

1. 개발환경

1.1 Frontend

- React-Native CLI: 0.72.4
- ANDROID SDK & NDK VERSION
    - MIN_SDK: 26
    - COMPILE_SDK: 34
    - NDK_VERSION: 23.1.7779620
- Virtual Device
    - Pixel 7
    - API 33

1.2 Backend

- Java 11
    - java OpenJDK
    - Spring Boot
        - Spring Data JPA
        - oauth2
        - JWT

        - SpringSecurity
        - JUnit
        - Lombok
        - SpringDocs
    - Gradle 8.2.1

1.3 Server

- Ubuntu 20.04
- Docker 24.0.5
- Docker-Compose 1.25.0
- Jenkins 2.414.1

1.4 Database

- MySQL 8.0.33

1.5 UI/UX

- Figma

1.6 IDE

- Visual Studio Code
- IntelliJ IDEA

1.7 형상 / 이슈관리

- Gitlab
- Jira
- Notion

1.8 기타 툴

- S3 2.2.6
- mattermost 7.8.6
- postman 10.17.0

2. EC2 세팅

2.1 Docker, Docker-compose Engine 설치

```
Docker install
Sudo apt-get update
sudo apt-get install docker -y

Docker-compose install
sudo apt install docker-compose
sudo chmod +x /home/ubuntu/docker-compose.yml
```

2.2 EC2 포트 정리

Port번호	내용
22	여유포트
80	여유포트
5000	Flask(Docker)
6379	Redis(Docker)
8080	Spring Boot(Docker)

9000	Jenkins(Docker)

3.DB설정

- AWS RDS MySQL사용

```
spring:

  datasource:
    url: jdbc:mysql://database-1.cjfyk8ntjeyl.ap-northeast-2.rds.amazonaws.com:3306/seniorNaver?useSSL=false&serverTimezone=Asia/Seoul&
    username: seniorNaver
    password: d106seniorNaver
    driver-class-name: com.mysql.cj.jdbc.Driver

  main:
    allow-circular-references: true

  jpa:
    show-sql: true

    # Hibernate
    database: mysql
    hibernate:
      # ?? ?? ? create ? DB??? ???.
      ddl-auto: update
      # camelcase ??
      strategy: org.hibernate.cfg.ImprovedNamingStrategy
      physical-strategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
    # ddl-auto ?? ?? ??
      generate-ddl: true
    # SQL? ?? ??
    properties:
      hibernate:
        format_sql: true
      # LazyInitializationException ??
      enable_lazy_load_no_trans: true
```

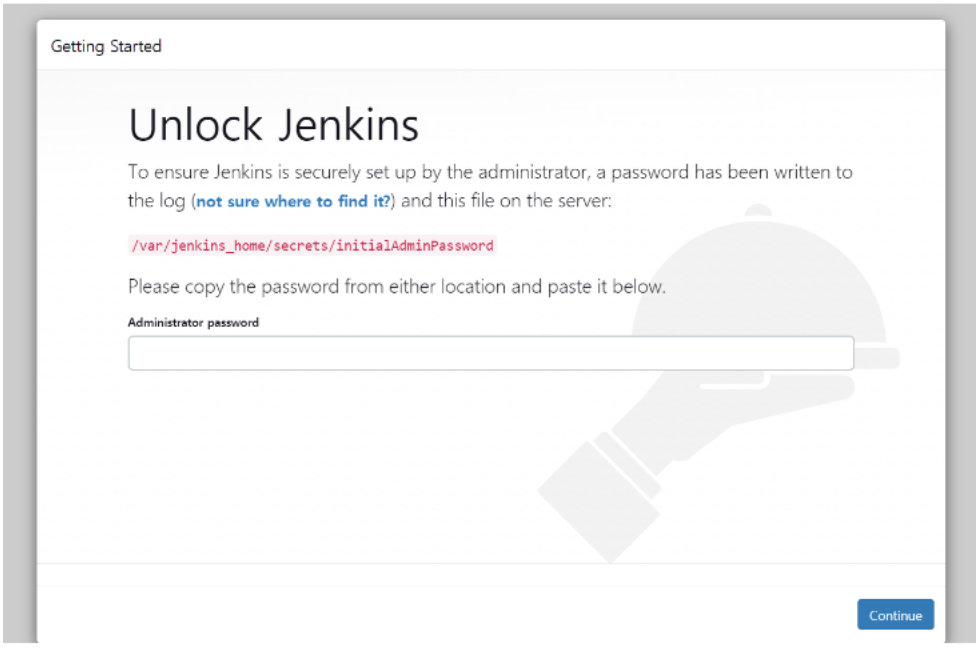
4.빌드 및 배포

4.1 Springboot

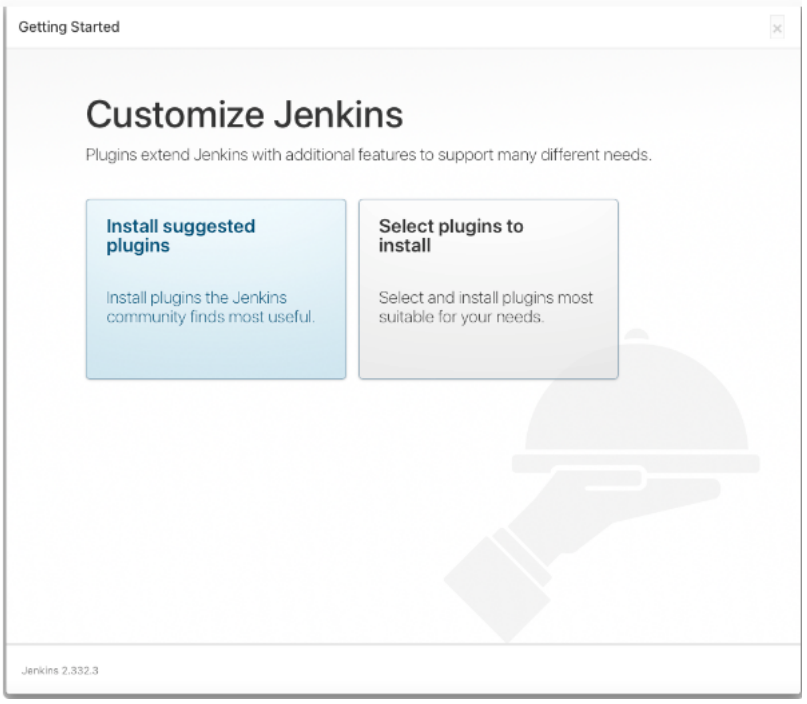
- docker-compose.yml파일을 만들고 여러개의 컨테이너를 실행할 수 있는 코드를 구현한다.
  위치는 /home/ubuntu/에 만들었다.
  - sudo vim docker-compose.yml파일을 만들고 아래의 코드를 입력한다. 포트는 사전에 열어둬야한다.

```
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:
    - 9000:8080
    privileged: true
    user: root
```

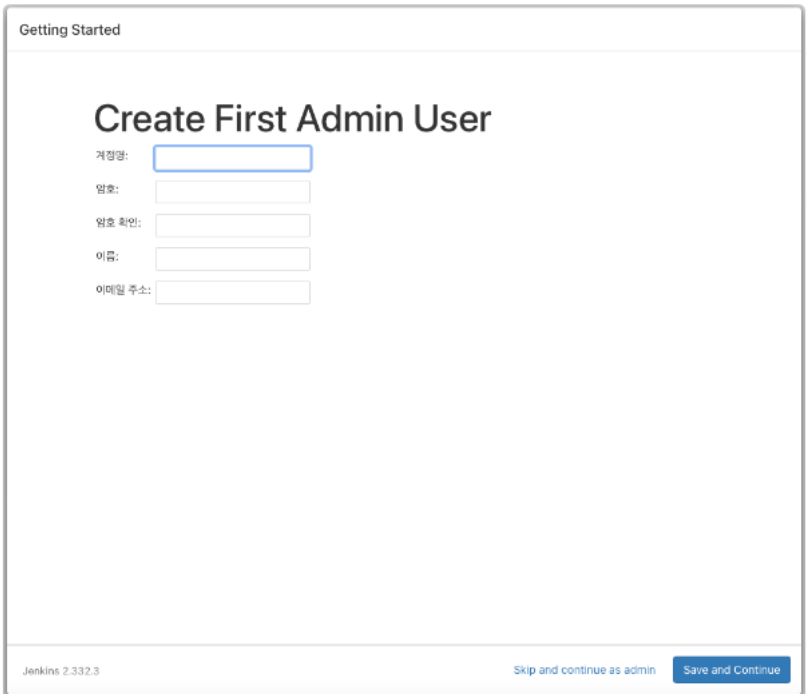
    - /var/run/docker.sock를 등록을 해야 jenkins 안에서도 docker를 실행할 수 있다.
    - `volumes` : 도커 컨테이너의 데이터는 컨테이너가 종료되면 휘발된다. 도커 컨테이너의 데이터를 보존하기 위해 사용한다. (`/var/jenkins_home` 이라는 디렉터리를 `/jenkins`와 마운트하고 데이터를 보존할 수 있다.)
  - docker-compose.yml이 존재하는 경로에서 sudo docker-compose up -d를 입력하여 실행한다. -d는 백그라운드에서 실행하겠다는 명령어다.
- docker ps 를 입력하여 컨테이너가 실행 되는지 확인한다. 만약 안보이면 docker ps -a로 확인한다. -a 명령어는 실행되지 않는 것도 볼 수 있는 명령어다. 확인후 도메인과 포트번호를 입력하여 들어간다.
- 입력한 도메인과 포트를 입력해 jenkins를 실행하고 다음 화면과 같이 나오며 성공이다.



- docker logs jenkins(name이다) 을 통해 비밀번호를 확인하고 빈칸에 넣는다.
- 젠킨스 플러그인을 전부 설치한다.



- 계정을 생성한다.



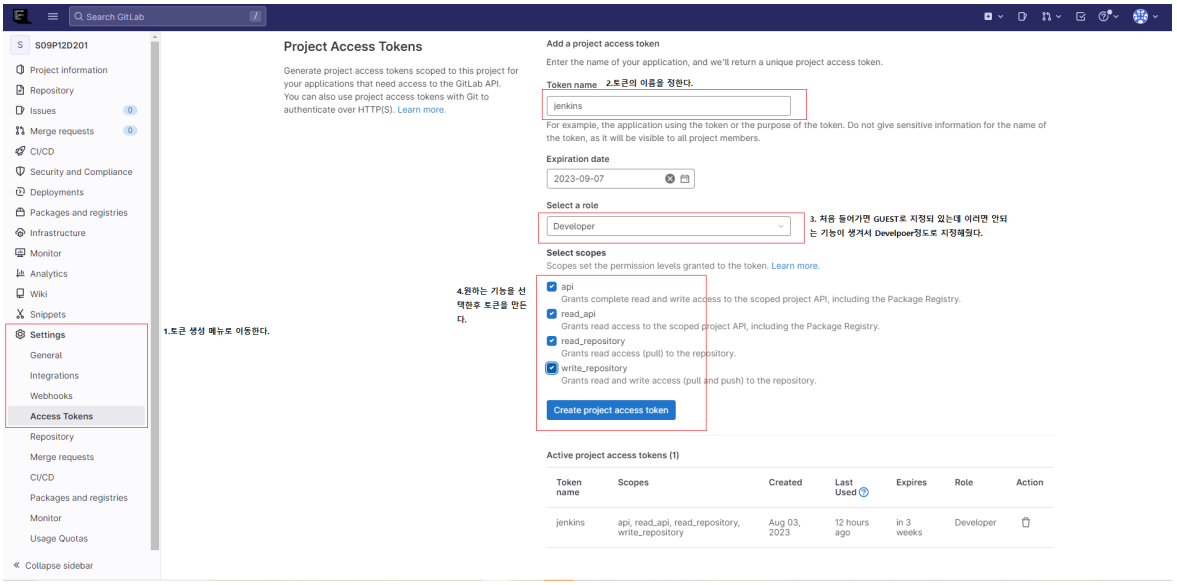
- 다시 서버로 돌아가서 jenkins 도커 컨테이너 안으로 들어간다.

```
sudo docker exec -it jenkins /bin/bash
```

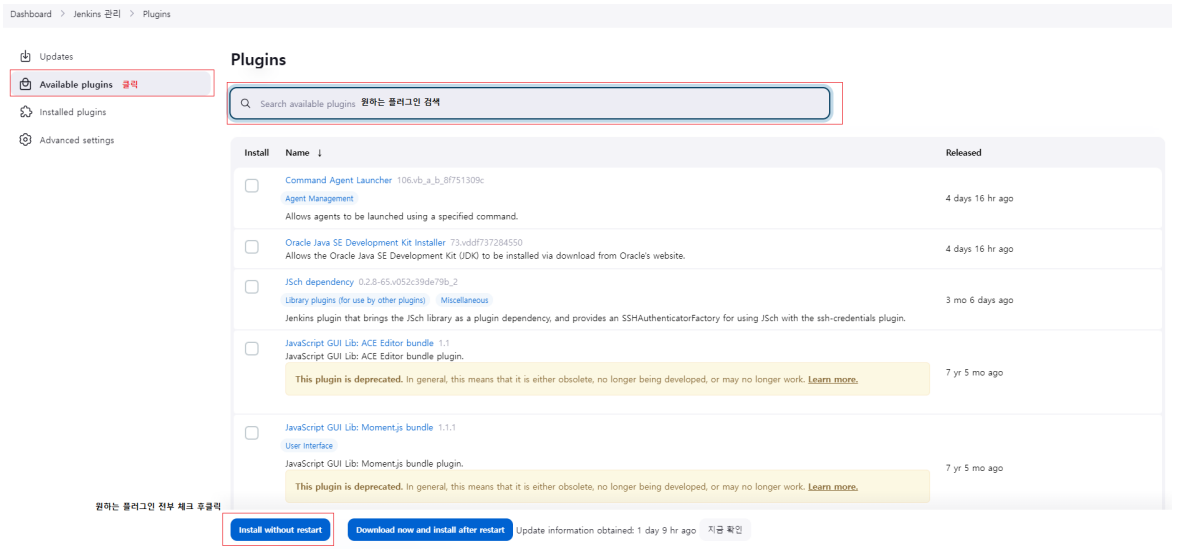
- 컨테이너 안에 들어가면 docker를 설치한다. 둘중 하나의 코드를 입력해 실행 후 확인 하기위해 안에서 docker ps를 입력한다.

```
docker exec -it ubuntu_jenkins_1 /bin/bash : container 접속
curl -fsSL https://get.docker.com -o get-docker.sh : docker 설치
sh get-docker.sh
혹은
apt-get update && \
apt-get -y install apt-transport-https \
     ca-certificates \
     curl \
     gnupg2 \
     software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
   $(lsb_release -cs) \
   stable" && \
apt-get update && \
apt-get -y install docker-ce
```

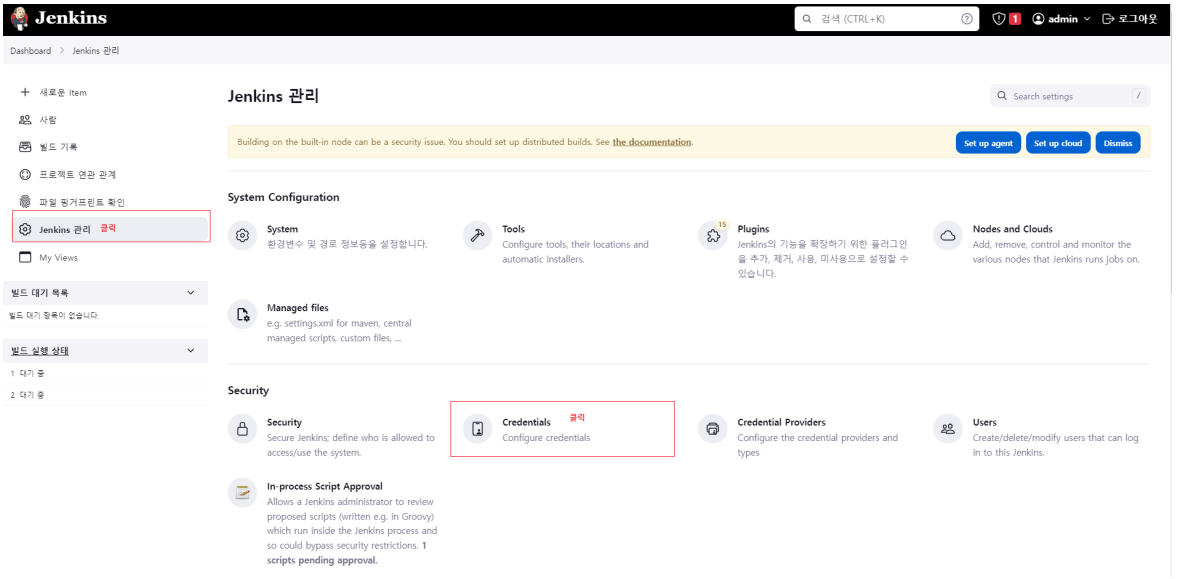
- gitlab을 사용해서 gitlab을 연결하기 위한 토큰을 생성을 한다.



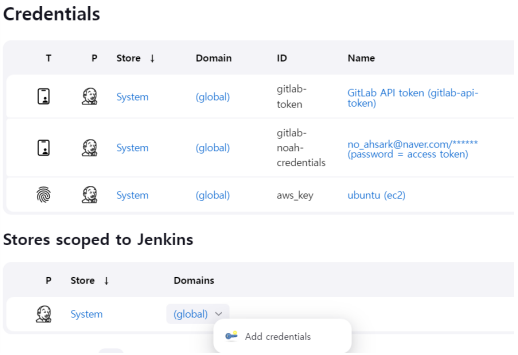
- plogins에 들어가서 available plubins에 들어가 gitlab과 ssh agent, docker를 설치한다.



- 젠킨스에 접속해서 로그인 후 jenkins관리로 이동 후 Credentials를 클릭한다.

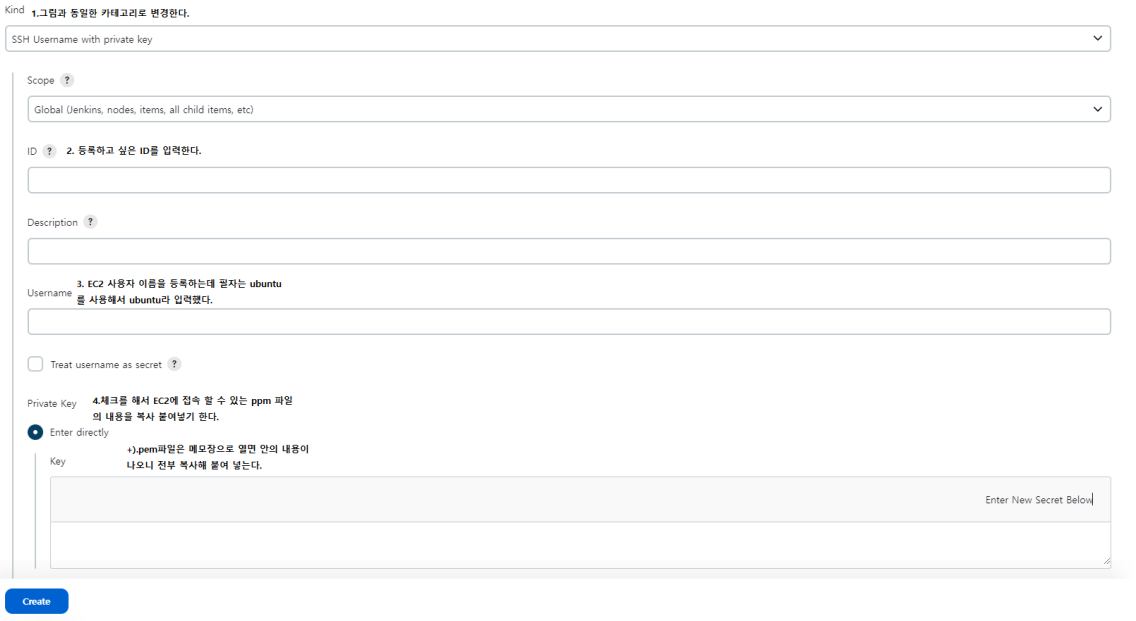


- 새로운 Credentials를 만들어준다.



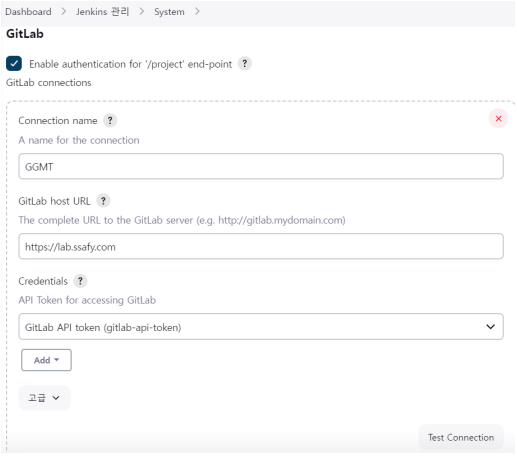
- 발급받은 토큰을 jenkins에 등록을 해준다.

- 이번엔 ssh에 연결할 수 있는 키를 등록한다.

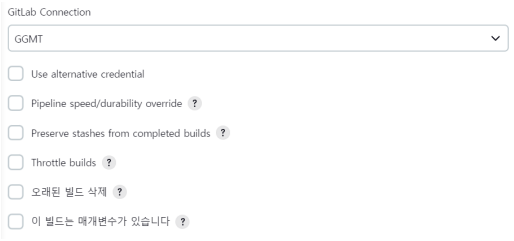


- 추가로 GitLab과의 connection을 위한GitLab API token도 등록해준다.

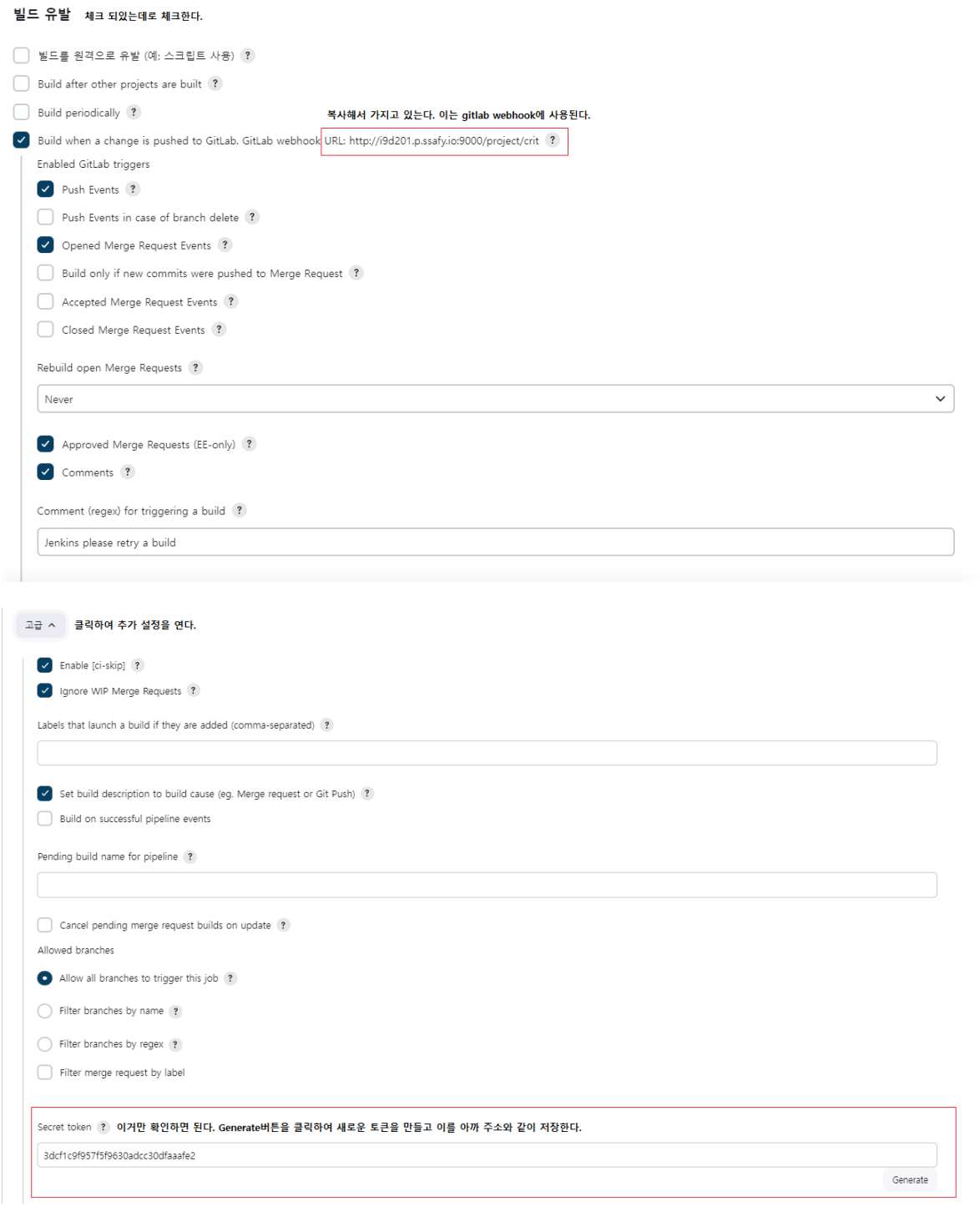
- Jenkins관리의 System에서 GitLab으로 진입하여 Connection을 설정해준다. URL은 도메인까지만 입력하며 Credentials는 등록해준 GitLab API token을 설정해준다. Test Connection을 눌러서 성공여부를 확인할수있다.



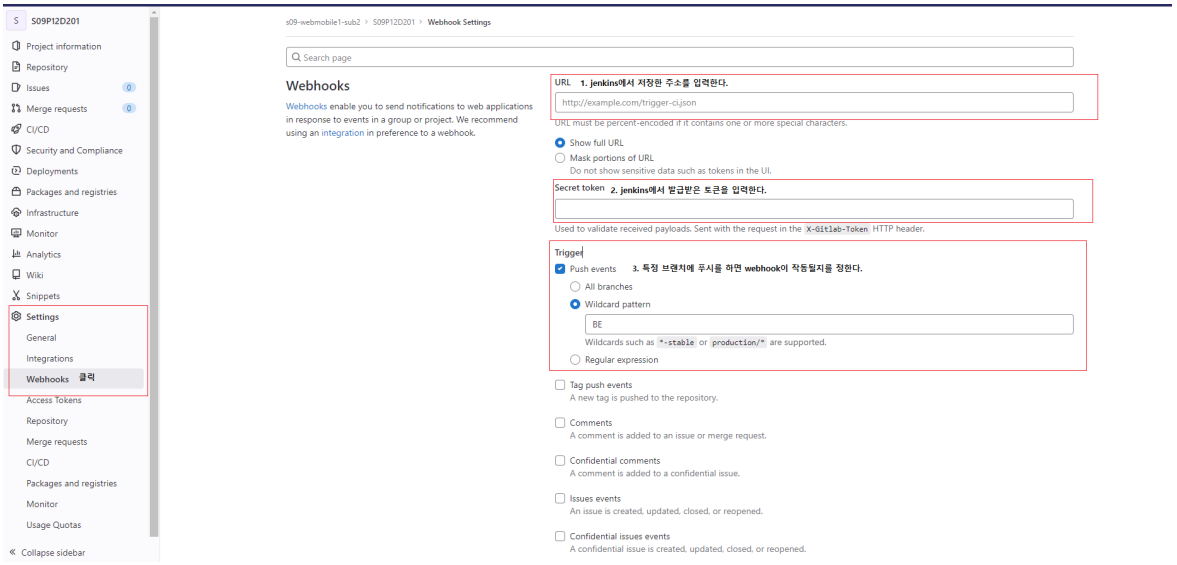
- 이제 메인 페이지에서 새로운 item을 클릭하고 Pipeline을 선택 후 원하는 이름을 설정 후 생성한다.
  - 생성 후 바로 입력해도 되고 나가서 언제든 해당 item의 구성을 클릭하여 수정할 수 있다.
- 구성의 GitLab Connection에서 설정해준 Connection을 선택한다.



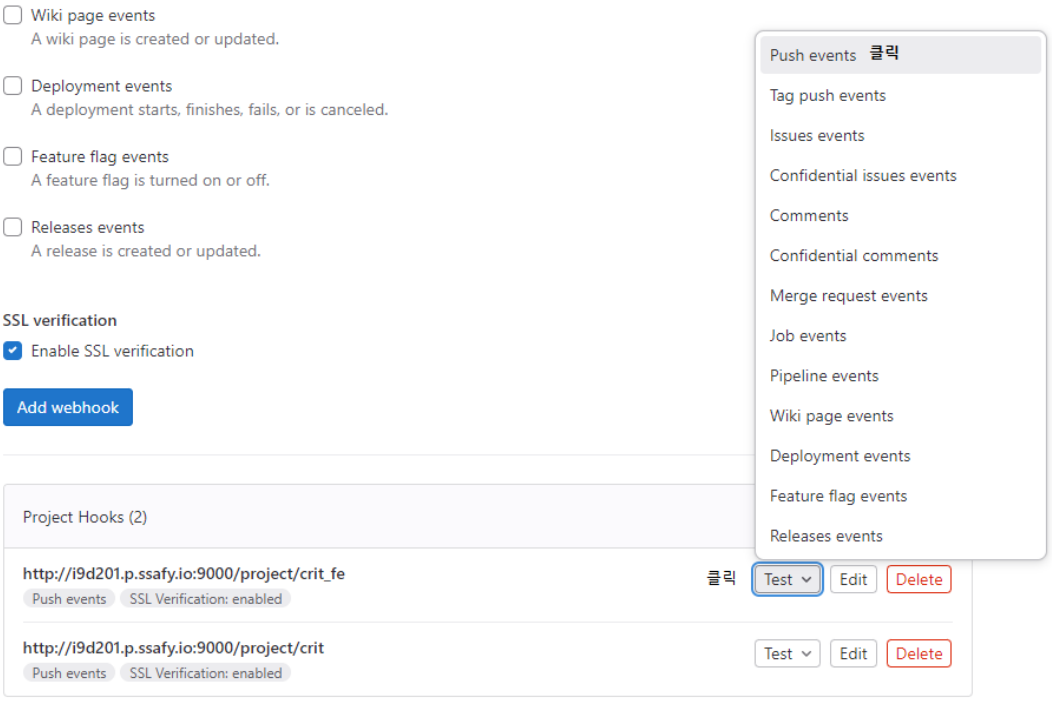
- 빌드 유발로 이동하여 webhook설정을 한다.



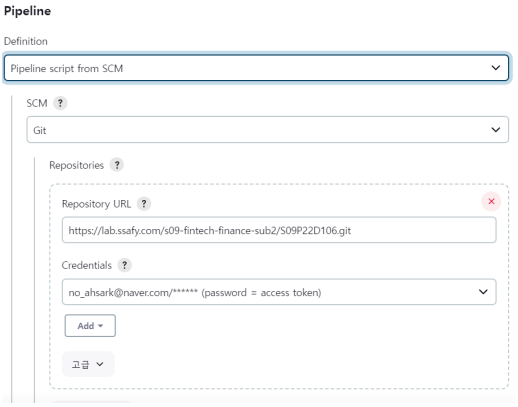
- gitlab에서 webhook을 등록하고 jenkins 와 연결한다.



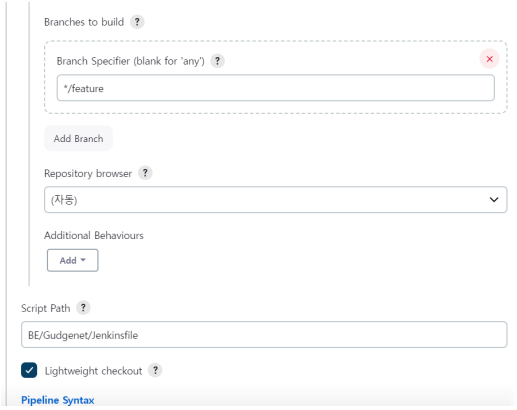
- 등록을 하면 밑에 나타나게 되는데 Test - push events를 클릭하면 테스트가 진행되고 위에 http 200이라는 말이 나오면 연결이 된 거다.



- Pipeline의 Definition을 Pipeline script from SCM으로 변경해준다. SCM을 Git으로 설정하고 프로젝트 Repository URL과 아까 설정한 Credentials을 설정해준다.



- 연동할 브랜치와 파이프라인 스크립트가 적혀있을 Jenkinsfile의 경로를 설정해준다.
  - Springboot의 경우 BE/Gudgenet/ 경로에 Jenkinsfile, Dockerfile, docker-compose를 작성해뒀다.



- Jenkinsfile의 내용

```
pipeline {
    agent any
    tools {
        gradle 'gradle'
    }
    stages {
        stage('Git Clone') {
            steps {
                dir('/var/jenkins_home/workspace/Gokk2_Senior'){
                    sh '''
                        echo delete existing project file
                    '''
                    deleteDir()
                }
                checkout scmGit(branches: [[name: '*/BE']],
                extensions: [submodule(recursiveSubmodules: true, reference: '', trackingSubmodules: true)],
                userRemoteConfigs: [
                    [credentialsId: 'submodule', url: 'https://lab.ssafy.com/kwmw0427/yml.git'],
                    [credentialsId: 'jenkins', url: 'https://lab.ssafy.com/s09-final/S09P31D105.git']])
                }
        }
        stage('BE-Build') {
            steps {
                dir("./BE/seniorNaver") {
                    sh "chmod +x ./gradlew"
                    sh "./gradlew clean build"
                }
            }
        }
        stage('Deploy') {
            steps {
                dir("./BE/seniorNaver") {
                    sh "docker-compose up -d --build"
                }
            }
        }
```

```
        stage('Clean Up') {
            steps {
                sh "docker image prune -f"
            }
        }
    }
}
```

- feature 브랜치를 클론해온다.
- chmod +x 로 gradlew에 읽기 권한을 주고 빌드한다.
- docker-compose 파일을 -d로 띄워 빌드한다.
- prune으로 사용하지 않고있는 이미지를 정리한다.

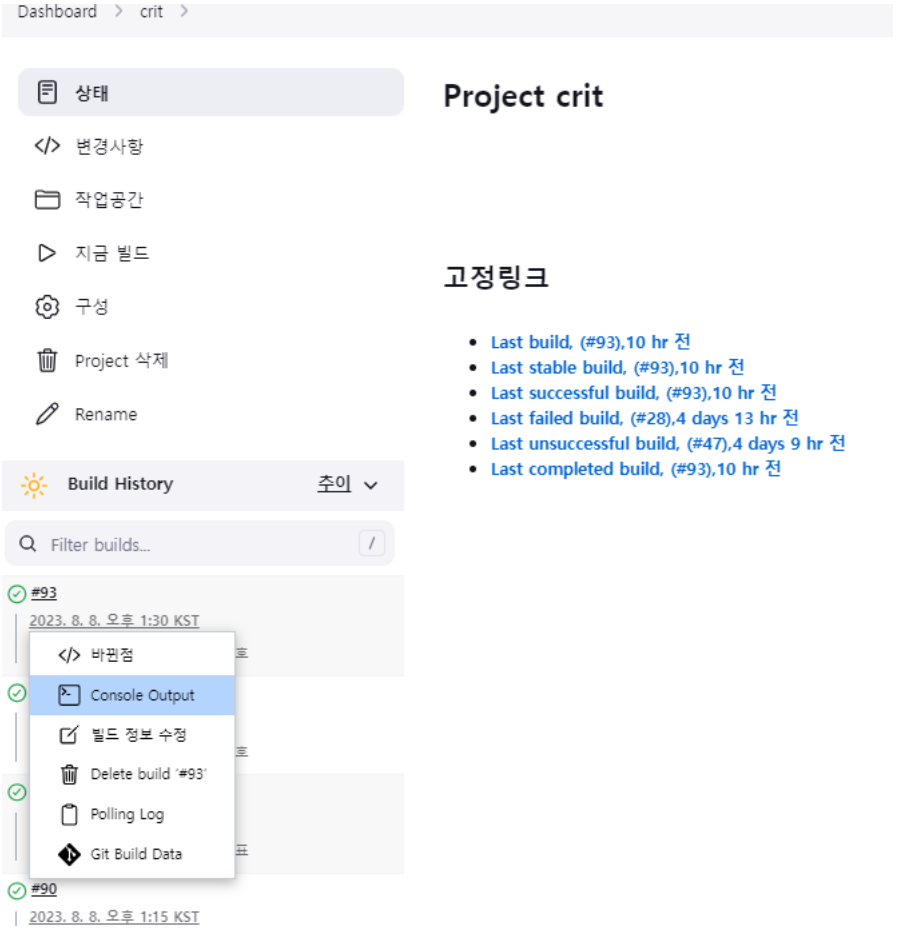
- docker-compose의 내용

```
version: "3"
services:
  spring:
    build:
      context: .
    ports:
      - 8080:9999 #로컬포트:컨테이너포트
```

  - build: context: .
    - 같은 경로의 Dockerfile을 빌드한다.
  - ports: 8080:9999
    - 외부에서 접속할 포트번호:컨테이너 내의 포트번호
- dockerfile의 내용

```
FROM openjdk:11-jdk

WORKDIR /app

COPY ./build/libs/*.jar application.jar

EXPOSE 9999

CMD ["java", "-jar", "application.jar"]
```

  - java 버전을 맞게 설정한다.
  - COPY로 jar파일을 복사하고 실행하여 배포한다.
- 이제 git의 연결해둔 브런치에 push,merge를 하거나 혹은 jenkins에서 지금빌드를 클릭하면 새로운 docker 컨테이너가 생성되며 배포가 된다.
  - 혹시 새로운 컨테이너가 바로 꺼진다면 서버에서 docker logs <해당 컨테이너의 이름> 을 입력하면 log를 확인할 수 있고 이를 통해 어디에서 오류가 났는지 확인이 가능하다.
  - 혹시 jenkins가 제대로 돌아가지 않으면 jenkins 내의 console로 확인이 가능하다.

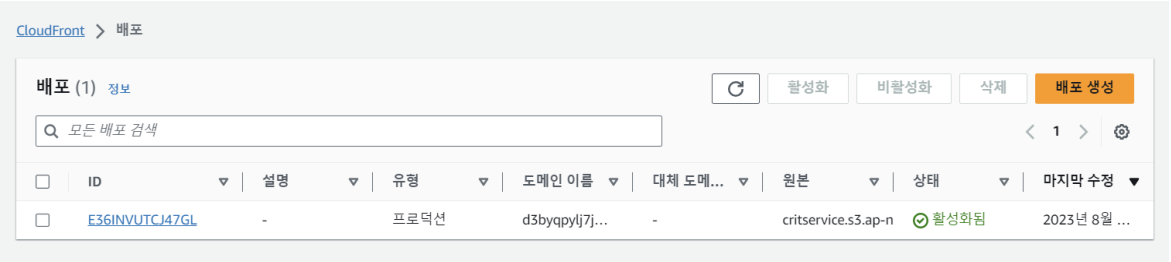


- 확인하고자 하는 빌드 번호를 클릭하거나 기다리면 나오는 오른쪽 화살표에 마우스를 대면 나오는 console Output을 클릭하면 해당 jenkins pipeline이 왜 실패했는지가 나오니 이를 보고 수정하면 된다.

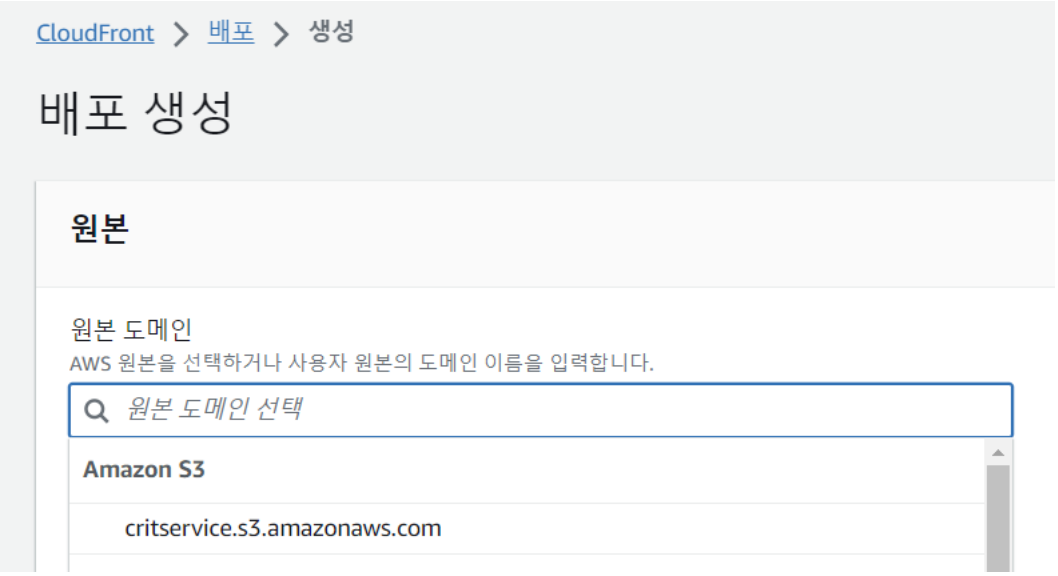
5. 외부서비스

5.1 S3 & cloudfront

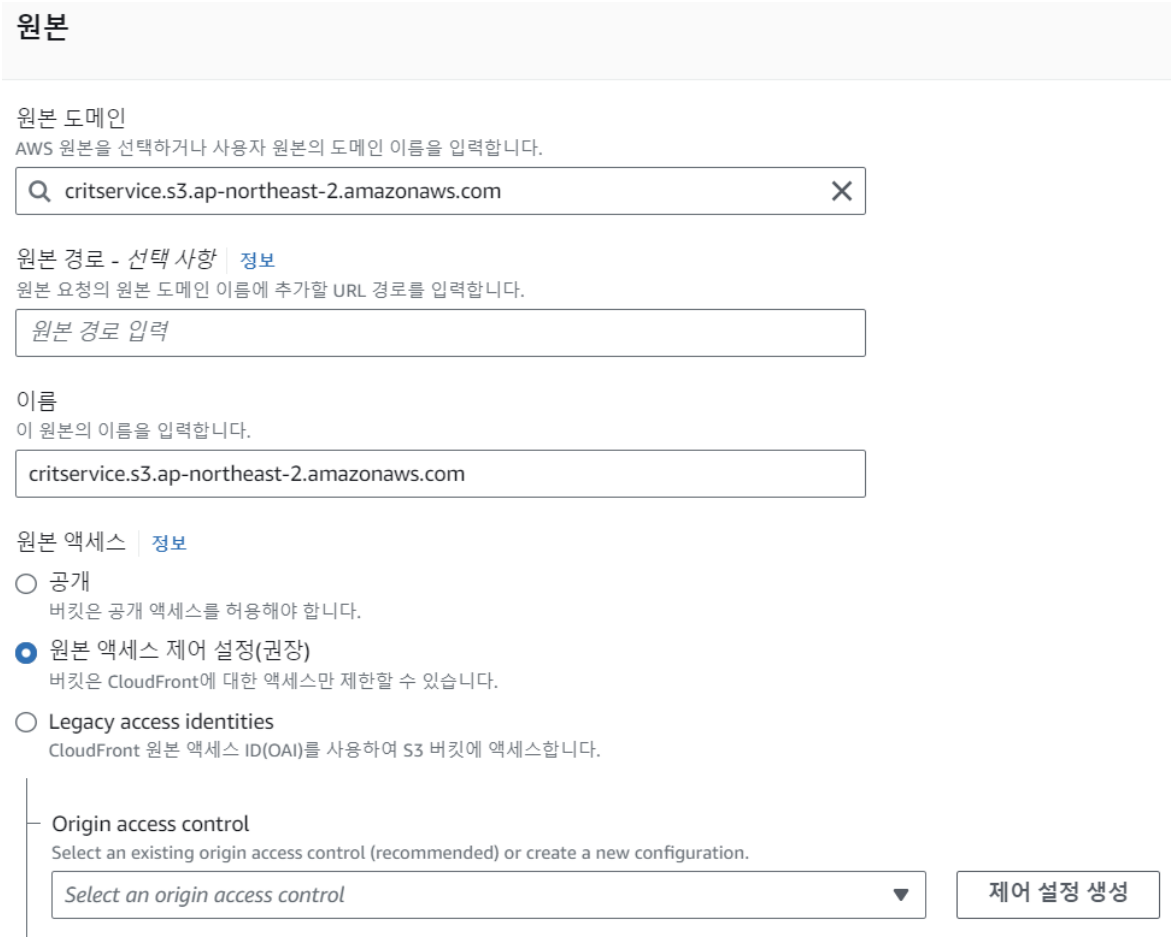
- aws에 로그인하여 CloudFront로 이동한다. 이후 배포 생성을 클릭



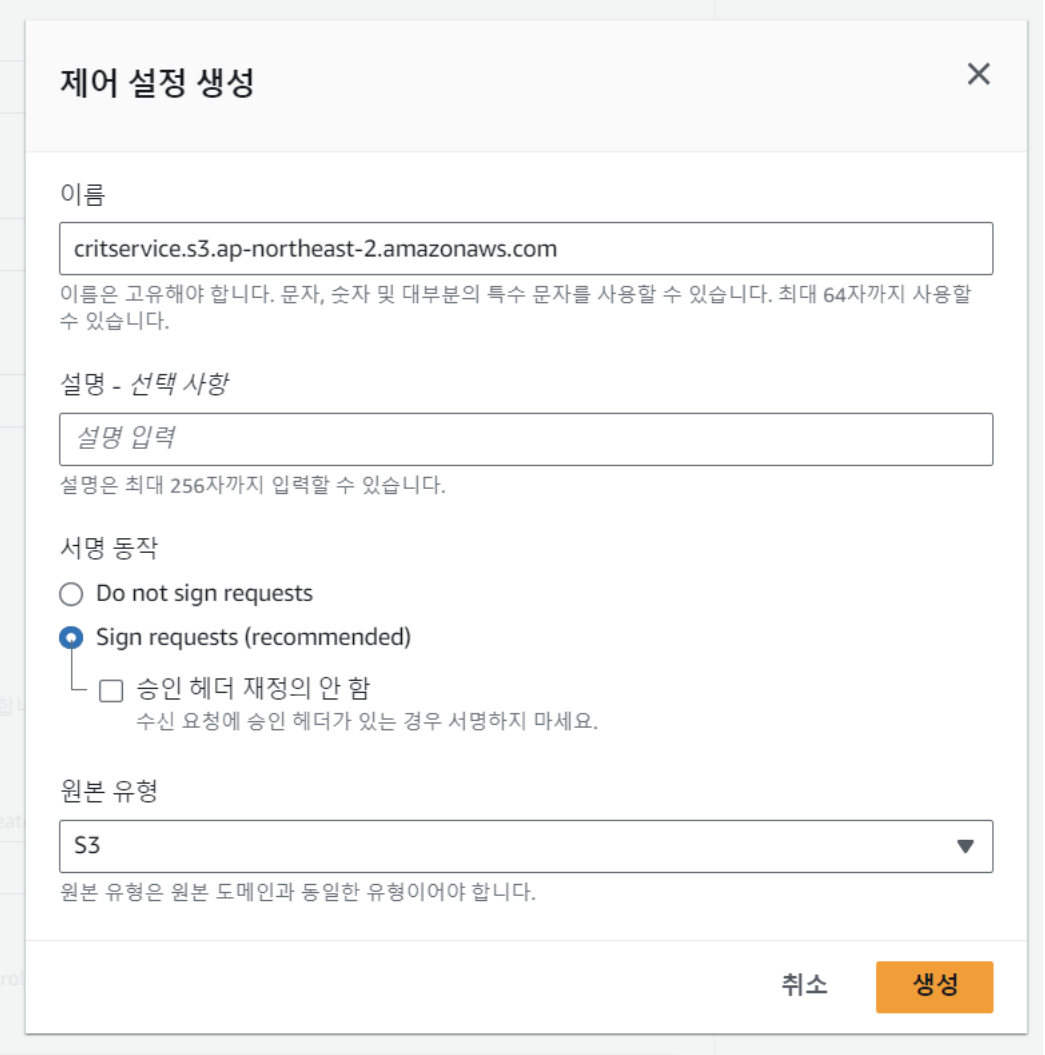
- 원본 도메인 선택을 클릭하여 나오는 s3주소를 클릭한다.



- 이후 나오는 원본 액세스를 원본 엑세스 제어 설정(권장)으로 체크한다.



- 이후 나오는 Origin access control에 있는 s3와 동일한 이름의 주소를 클릭한다. 혹시 에러가 나오거나 없는 경우 제어 설정 생성을 클릭 해준다.
  - 제어 설정 생성에 들어가서 특별히 건들 설정 없이 바로 생성 해주고 이를 사용해준다.



- 이후 건들 설정 없이 바로 배포생성 버튼을 클릭해준다.
- 이후 생성한 cloudfront를 클릭하면 배포 도메인 이름이 나오는데 이를 복사하여 자신이 사용하고 있던 S3주소와 바꾸기만 하면 사용이 된다.