



PORTING MANUAL

목 차

I. 개발환경	3
1. 프로젝트 기술 스택	3
2. 설정 파일 목록	3
Frontend:	3
Backend:	3
Docker:	4
Nginx:	4
3. 설정 파일 및 환경 변수 정보	4
Frontend:	4
Backend:	6
Docker:	9
Nginx:	10
II. 빌드 및 배포	12
1. Docker 설치	12
2. SSL 인증서 발급	12
3. OpenVidu 배포	15
4. DB 및 Infra 배포	16
5. Backend CI/CD	19
6. Frontend CI/CD	29
III. 외부 서비스	31
1. Google SMTP	31
2. AWS S3 Bucket	33

I. 개발환경

1. 프로젝트 기술 스택

Server : AWS EC2 Ubuntu 20.04 LTS

Visual Studio Code : 1.75.1

IntelliJ IDEA : 2022.3.1 (Ultimate Edition) 17.0.5+1-b653.23 amd64

JVM : OpenJDK 11

Docker : 20.10.17

Node.js : 16.19.0

MariaDB : 10.10.2

Redis : 7.0.8

Nginx : 1.23.3

Jenkins : 2.375.2

Openvidu : 2.25.0

2. 설정 파일 목록

Frontend:

- **Dockerfile** : /jenkins/workspace/frontend/Frontend
- **Jenkinsfile** : /jenkins/workspace/frontend/Frontend
- **front.conf** : /jenkins/workspace/frontend/Frontend

Backend:

- **Dockerfile** : /jenkins/workspace/backend/Backend
- **Jenkinsfile** : /jenkins/workspace/backend/Backend
- **application.yml** : /jenkins/workspace/backend/Backend/src/main/resources

Docker:

- **docker-compose.yml** : /home/ubuntu

Nginx:

- **nginx.conf** : /home/ubuntu/nginx/conf.d

3. 설정 파일 및 환경 변수 정보

Frontend:

- **Dockerfile** :

```
FROM nginx:stable-alpine
WORKDIR /app
RUN mkdir ./build
ADD ./build ./build
RUN rm /etc/nginx/conf.d/default.conf
COPY ./front.conf /etc/nginx/conf.d
EXPOSE 3126
CMD ["nginx", "-g", "daemon off;"]
```

- **Jenkinsfile** :

```
pipeline {
    agent any

    tools {
        nodejs "nodejs"
    }

    stages {
        stage('React Build') {
            steps {
                dir('Frontend') {
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }
    }
}
```

```

    }
  }

  stage('Docker Build') {
    steps {
      dir('Frontend') {
        sh 'docker build -t banana-front:latest .'
      }
    }
  }

  stage('Deploy') {
    steps{
      sh 'docker rm -f front'
      sh 'docker run -d --name front -p 3126:3126 -u root
banana-front:latest'
    }
  }

  stage('Finish') {
    steps{
      sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
    }
  }
}

```

- front.conf :

```

server {
  listen 3126;
  location / {
    root /app/build;
    index index.html;
    try_files $uri $uri/ /index.html;
  }
}

```

Backend:

- Dockerfile :

```
FROM openjdk:11-jdk-slim
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8099
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- Jenkinsfile :

```
pipeline {
    agent any

    stages {
        stage('Gradle Build') {
            steps {
                dir('Backend') {
                    sh 'chmod +x gradlew'
                    sh './gradlew clean build -x test'
                }
            }
        }

        stage('Docker Build') {
            steps {
                dir('Backend') {
                    sh 'docker build -t banana-back:latest .'
                }
            }
        }

        stage('Deploy') {
            steps {
                sh 'docker rm -f back'
                sh 'docker run -d --name back -p 8099:8099 -u root
banana-back:latest'
            }
        }
    }
}
```

```

    stage('Finish') {
        steps{
            sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
        }
    }
}

```

- application.yml :

```

server:
  port: 8099
  servlet:
    encoding:
      charset: UTF-8
      enabled: true
      force: true

spring:
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://{서비스도메인}:3324/{schema이름}?serverTimezone=Asia/Seoul
    username: 계정명
    password: 비밀번호
  jpa:
    database-platform: org.hibernate.dialect.MariaDBDialect
    hibernate:
      ddl-auto: none
      use-new-id-generator-mappings: false
    properties:
      hibernate:
        format_sql: true
      open-in-view: false
  redis:
    database: 0
    host: 서비스도메인
    password: 비밀번호
    port: 6379
  mail:

```

```
host: smtp.gmail.com
port: 587
username: 이메일주소
password: 앱 비밀번호
properties:
  mail:
    smtp:
      starttls:
        enable: true
        required: true
      auth: true
      connection-timeout: 5000
      timeout: 5000
      write-timeout: 5000

logging.level:
  org.hibernate.SQL: debug

jwt:
  header: Authorization
  secret: 비밀키값
  access-token-valid-time: 1800
  refresh-token-valid-time: 604800

cloud:
  aws:
    s3:
      bucket: 버킷이름
      folder:
        default-profile: default-profile/
        profile: profile/
        art-image: art/
        art-thumbnail: art/thumbnail/
      credentials:
        access-key: IAM 액세스키값
        secret-key: IAM 시크릿키값
      region:
        static: ap-northeast-2
    stack:
      auto: false
```

Docker:

- docker-compose.yml :

```
version: "3.8"
services:
  mariadb:
    image: mariadb
    container_name: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: 비밀번호
    volumes:
      - ~/mariadb/databases:/var/lib/mysql/
    ports:
      - 3324:3306
    restart: unless-stopped

  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    volumes:
      - /usr/bin/docker:/usr/bin/docker
      - /var/run/docker.sock:/var/run/docker.sock
      - /jenkins:/var/jenkins_home
    ports:
      - 8093:8080
    privileged: true
    user: root
    restart: unless-stopped

  redis:
    image: redis
    container_name: redis
    ports:
      - 6379:6379
    command: redis-server --requirepass 비밀번호 --port 6379
    environment:
      - REDIS_REPLICATION_MODE=master
    volumes:
      - ~/redis/data:/data
    restart: unless-stopped
```

```

nginx:
  image: nginx
  container_name: nginx
  ports:
    - 80:80
    - 443:443
  volumes:
    - ~/nginx/conf.d:/etc/nginx/conf.d
    - ~/certbot/conf:/etc/letsencrypt
    - ~/certbot/www:/var/www/certbot
  restart: unless-stopped
  command: "/bin/sh -c 'while ;; do sleep 6h & wait $$(!); nginx -s reload; done
& nginx -g \"daemon off;\""

certbot:
  image: certbot/certbot
  container_name: certbot
  volumes:
    - ~/certbot/conf:/etc/letsencrypt
    - ~/certbot/www:/var/www/certbot
  restart: unless-stopped
  entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do certbot renew; sleep 12h &
wait $$(!); done

```

Nginx:

- **nginx.conf :**

```

server {
    listen 80;
    server_name 서비스도메인;
    server_tokens off;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

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

```

```

}
server {
    listen 443 ssl;
    server_name 서비스도메인;
    server_tokens off;
    access_log off;
    ssl_certificate /etc/letsencrypt/live/서비스도메인/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/서비스도메인/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass http://서비스도메인:3126/;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Host $server_name; proxy_set_header X-Real-IP
$remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme; proxy_set_header Upgrade
$http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_redirect off;
    }

    location /api/ {
        proxy_pass http://서비스도메인:8099/;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Host $server_name; proxy_set_header X-Real-IP
$remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme; proxy_set_header Upgrade
$http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_redirect off;
    }
}

```

II. 빌드 및 배포

1. Docker 설치

- Install Docker Engine on Ubuntu

```
sudo apt-get update
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

sudo mkdir -m 0755 -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

sudo apt-get update

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

2. SSL 인증서 발급

- 쉘 스크립트 생성 (init-letsencrypt.sh)

```
#!/bin/bash

if ! [ -x "$(command -v docker)" ] || ! [ -x "$(command -v compose)" ]; then
    echo 'Info: docker compose is not installed.' >&2
    exit 1
fi

domains=서비스도메인
rsa_key_size=4096
data_path="/home/ubuntu/certbot"
```

```

email="이메일" # Adding a valid address is strongly recommended
staging=0 # Set to 1 if you're testing your setup to avoid hitting request limits

if [ -d "$data_path" ]; then
    read -p "Existing data found for $domains. Continue and replace existing
certificate? (y/N) " decision
    if [ "$decision" != "Y" ] && [ "$decision" != "y" ]; then
        exit
    fi
fi

if [ ! -e "$data_path/conf/options-ssl-nginx.conf" ] || [ ! -e
"$data_path/conf/ssl-dhparams.pem" ]; then
    echo "### Downloading recommended TLS parameters ..."
    mkdir -p "$data_path/conf"
    curl -s
https://raw.githubusercontent.com/certbot/certbot/master/certbot-nginx/certbot_nginx/_internal/tls_configs/options-ssl-nginx.conf >
"$data_path/conf/options-ssl-nginx.conf"
    curl -s
https://raw.githubusercontent.com/certbot/certbot/master/certbot/certbot/ssl-dhparams.pem > "$data_path/conf/ssl-dhparams.pem"
    echo
fi

echo "### Creating dummy certificate for $domains ..."
path="/etc/letsencrypt/live/$domains"
mkdir -p "$data_path/conf/live/$domains"
docker compose run --rm --entrypoint "\
    openssl req -x509 -nodes -newkey rsa:$rsa_key_size -days 1\
        -keyout '$path/privkey.pem' \
        -out '$path/fullchain.pem' \
        -subj '/CN=localhost'" certbot
echo

echo "### Starting nginx ..."
docker compose up --force-recreate -d nginx
echo

echo "### Deleting dummy certificate for $domains ..."

```

```

docker compose run --rm --entrypoint "\
    rm -Rf /etc/letsencrypt/live/$domains && \
    rm -Rf /etc/letsencrypt/archive/$domains && \
    rm -Rf /etc/letsencrypt/renewal/$domains.conf" certbot
echo

echo "### Requesting Let's Encrypt certificate for $domains ..."
#Join $domains to -d args
domain_args=""
for domain in "${domains[@]"; do
    domain_args="$domain_args -d $domain"
done

# Select appropriate email arg
case "$email" in
    "") email_arg="--register-unsafely-without-email" ;;
    *) email_arg="--email $email" ;;
esac

# Enable staging mode if needed
if [ $staging != "0" ]; then staging_arg="--staging"; fi

docker compose run --rm --entrypoint "\
    certbot certonly --webroot -w /var/www/certbot \
    $staging_arg \
    $email_arg \
    $domain_args \
    --rsa-key-size $rsa_key_size \
    --agree-tos \
    --force-renewal" certbot
echo

echo "### Reloading nginx ..."
docker compose exec nginx nginx -s reload

```

- 쉘스크립트 실행

```
./init-letsencrypt.sh
```

3. OpenVidu 배포

- 기존 openvidu, kurento와 관련된 컨테이너가 존재하면 전부 삭제
- OpenVidu 서버를 On premises 방식으로 설치

```
# root 계정으로 전환
sudo su
cd /opt

curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_openvidu_latest.sh
| bash

cd openvidu

# 설정파일 수정
vi .env

# .env 파일 내에 다음 내용들을 찾아서 수정
DOMAIN_OR_PUBLIC_IP=서비스도메인
OPENVIDU_SECRET=OpenVidu비밀번호
CERTIFICATE_TYPE=letsencrypt
LETSencrypt_EMAIL= 인증서 발급 시 입력한 이메일
# 앞에 '#'을 지워서 주석 해제 후 작성
HTTP_PORT=8446
HTTPS_PORT=8447

# :wq! 입력후 엔터를 눌러 변경 내용 저장
```

- SSL 인증서 적용

```
# root 계정으로 전환이 안되어 있다면
sudo su

# 기존에 발급받은 인증서를 openvidu폴더로 복사
mkdir /opt/openvidu/certificates
cd ~/certbot/conf
cp -r * /opt/openvidu/certificates
```

- OpenVidu 실행

```
# openvidu 설치 위치로 이동
cd /opt/openvidu
./openvidu start
# Ctrl + C 눌러서 로그에서 빠져나옴
```

- 포트 개방

```
# ubuntu firewall에 설정한 https 포트 등록
sudo ufw allow 8447
```

- 종료 방법

```
# openvidu 설치 위치로 이동
cd /opt/openvidu
./openvidu stop
```

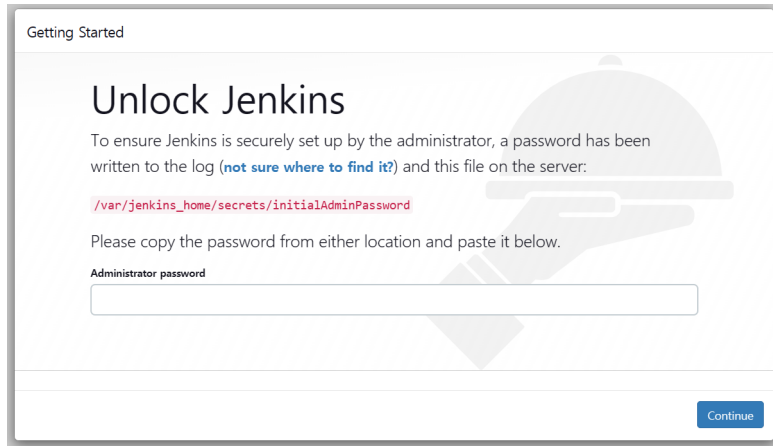
4. DB 및 Infra 배포

- docker-compose.yml 작성 : **I-3 설정 파일 및 환경 변수 정보**의 Docker 항목 참조
- docker compose 실행:

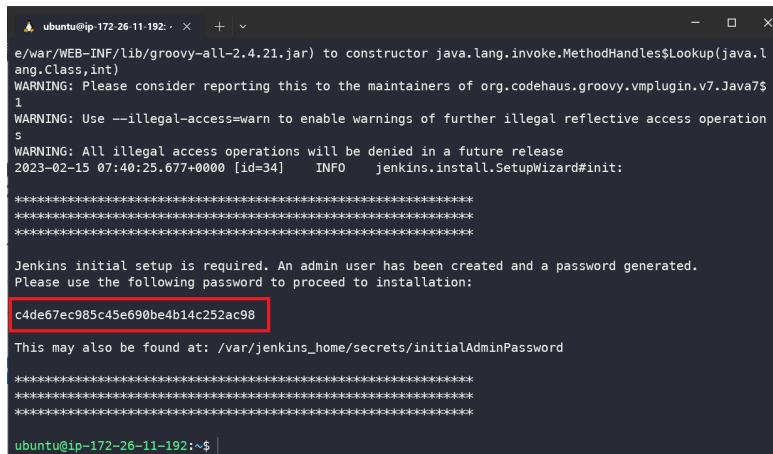
```
# docker-compose.yml파일이 있는 home디렉토리에서 실행
cd /home/ubuntu
docker compose up -d
```

- Nginx 설정 : **I-3 설정 파일 및 환경 변수 정보**의 Nginx 항목을 참조하여
/home/ubuntu/nginx/conf.d/nginx.conf 파일 생성 및 수정
- Jenkins 설정

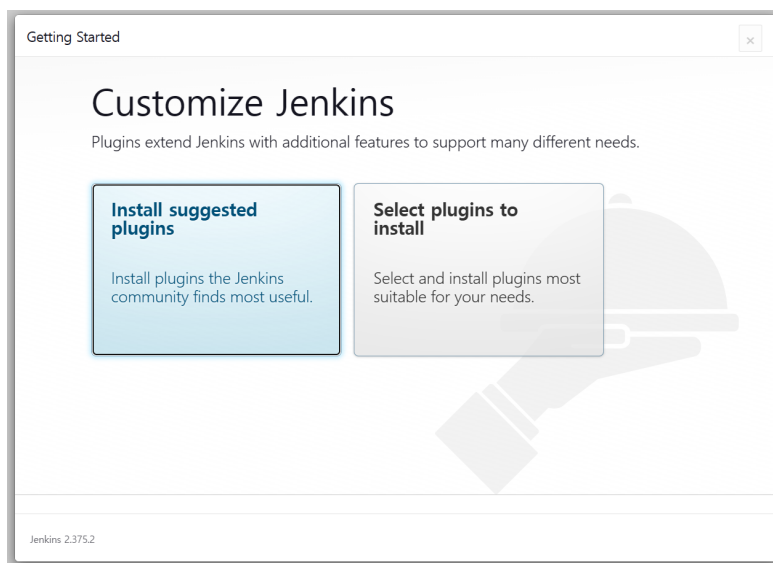
1. http://서비스도메인:8093으로 접속



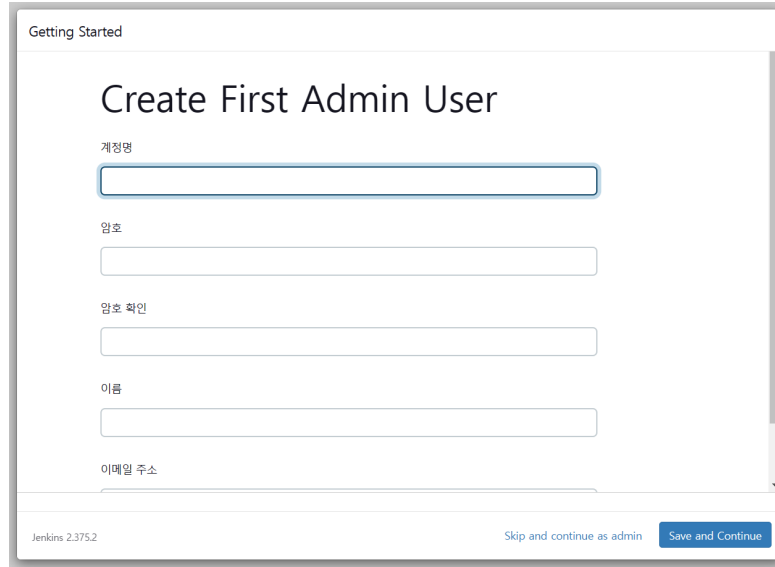
2. 서버 콘솔에서 docker logs jenkins를 입력하여 출력된 패스워드를 입력



3. Install suggested plugins를 클릭하고 플러그인들 설치



4. Jenkins에서 사용할 계정 생성

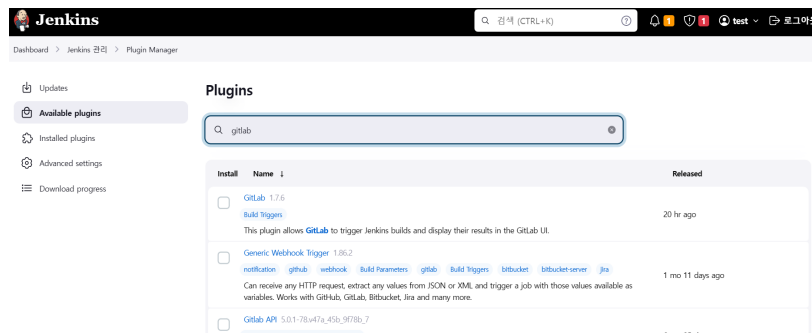


The image shows the 'Getting Started' screen in Jenkins with the title 'Create First Admin User'. It contains five input fields: '계정명' (Username), '암호' (Password), '암호 확인' (Confirm Password), '이름' (Name), and '이메일 주소' (Email address). At the bottom, there are two buttons: 'Skip and continue as admin' and 'Save and Continue'.

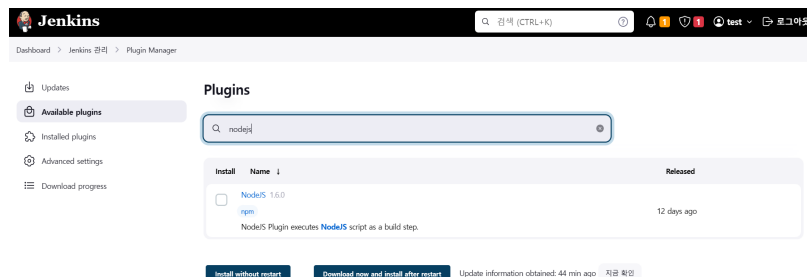
5. Jenkins 접속 URL 확인 후 Save and Finish 클릭

6. Jenkins 페이지 접속시에 생성한 계정으로 로그인

7. Jenkins 관리 - 플러그인 관리 - Available plugins에서 gitlab 플러그인 설치



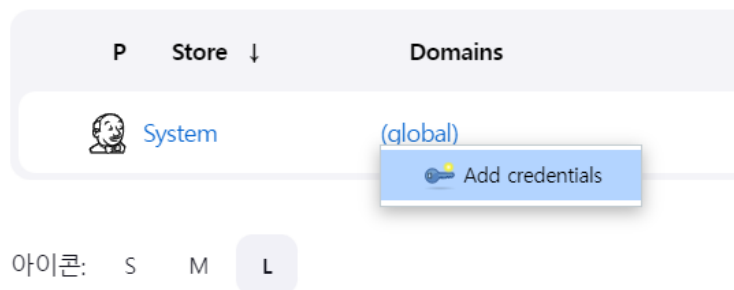
8. NodeJS 플러그인도 설치



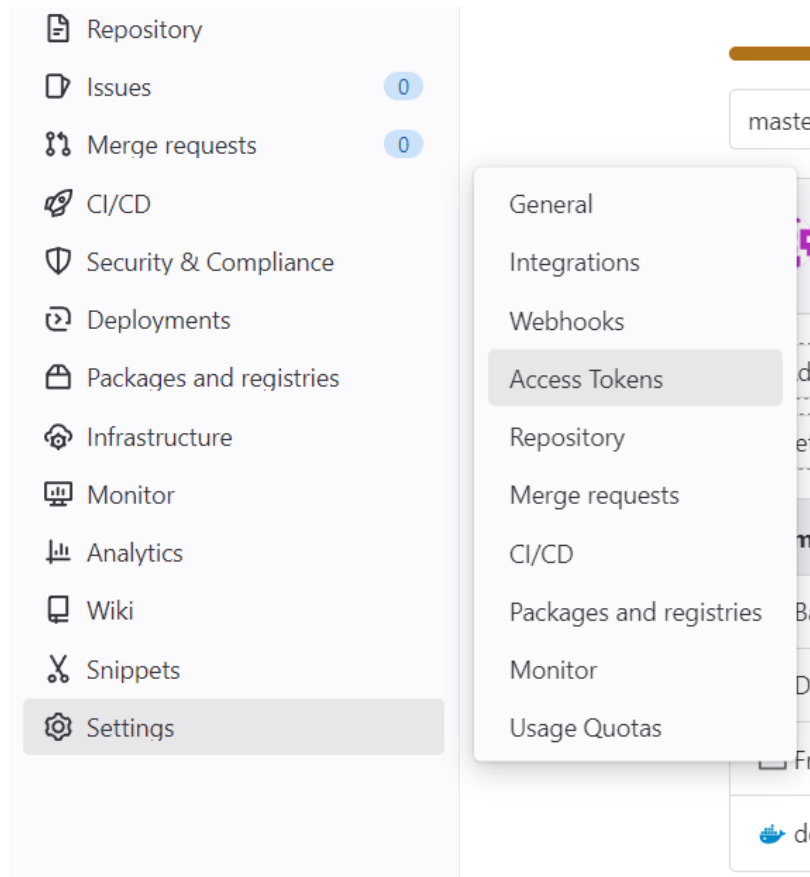
5. Backend CI/CD

- application.yml 작성 : **I-3 설정 파일 및 환경 변수 정보**의 Backend 항목을 참조하여 파일을 작성하고 ec2 머신의 /home/ubuntu 경로에 준비해 놓는다
- Dockerfile 작성 : **I-3 설정 파일 및 환경 변수 정보**의 Backend 항목을 참조하여 작성후 프로젝트의 Backend 폴더 최상위에 넣어서 GitLab Repository에 push한다
- Jenkinsfile 작성 : **I-3 설정 파일 및 환경 변수 정보**의 Backend 항목 참조하여 작성후 프로젝트의 Backend 폴더 최상위에 넣어서 GitLab Repository에 push한다
- GitLab 설정
 1. Jenkins 관리 - Manage Credentials로 이동
 2. Stores scoped to Jenkins의 Domains 부분에 (global)에 마우스를 올렸을때 나오는 화살표를 클릭하고 Add credentials 클릭

Stores scoped to Jenkins



3. GitLab 프로젝트 Repository에서 Settings - Access Tokens로 진입



4. Token name과 만료일을 임의로 설정하고 Select a role은 Maintainer로 설정, scopes는 read_api, read_repository를 체크하고 토큰을 생성한다

Project Access Tokens

Generate project access tokens scoped to this project for your applications that need access to the GitLab API. You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)

Add a project access token

Enter the name of your application, and we'll return a unique project access token.

Token name

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

Expiration date

Select a role

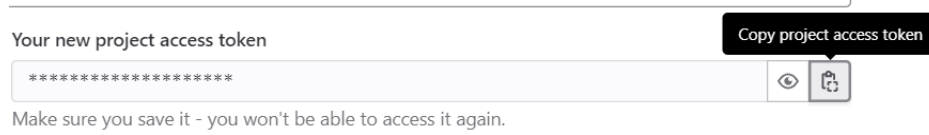
Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☐ api
Grants complete read and write access to the scoped project API, including the Package Registry.
- ☒ read_api
Grants read access to the scoped project API, including the Package Registry.
- ☒ read_repository
Grants read access (pull) to the repository.
- ☐ write_repository
Grants read and write access (pull and push) to the repository.

Create project access token

5. 위에 생성된 토큰을 복사한다 (미리 백업 해놓지 않으면 다시는 확인 불가)



6. 다시 Jenkins의 New credentials로 돌아와서 다음과 같이 설정 후 Create



- **Kind** : GitLab API token
- **Scope** : Global
- **API token** : gitlab에서 복사해온 값 붙여넣기
- **ID** : credential들을 구분하는 ID (임의로 설정)
- **Description** : 설명 (임의로 설정)

7. Jenkins 관리 - 시스템 설정으로 진입

8. Gitlab 부분에 추가를 누르고 다음과 같이 작성 후 저장

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name ✕

A name for the connection

Gitlab host URL

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

Credentials

API Token for accessing Gitlab

▼

+ Add

고급...

Test Connection

추가

- **Connection name** : 연결설정 이름 (임의로 작성)
- **Gitlab host URL** : gitlab 서버 도메인 입력 (https://lab.ssafy.com/)
- **Credentials** : 아까 만든 GitLab API Token Credentials 선택

9. Jenkins 관리 - Manage Credentials에서 Add credentials를 한번 더 클릭

10. New credentials에서 GitLab 로그인 계정을 다음과 같이 작성

New credentials

Kind

Username with password ▼

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?

☐ Treat username as secret ?

Password ?

ID ?

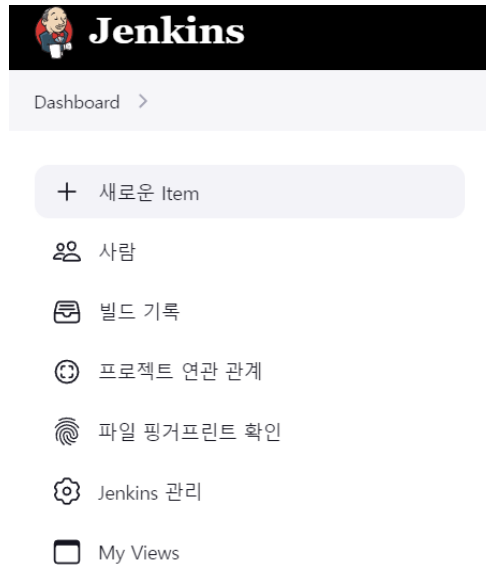
Description ?

Create

- **Kind** : Username with password 선택
- **Scope** : Global 선택
- **Username** : GitLab 계정 이름
- **Password** : GitLab 계정 비밀번호
- **ID** : credential들을 구분하는 ID (임의로 설정)
- **Description** : 설명 (임의로 설정)

- Pipeline item 생성

1. Jenkins 메인화면 좌측 사이드바에서 “새로운 아이템” 클릭



2. Item 이름을 작성하고 Pipeline을 선택한 후 OK

3. General에서 다음과 같이 설정

General Enabled

설명

[Plain text] [미리보기](#)

☐ Do not allow concurrent builds

☒ Do not allow the pipeline to resume if the controller restarts

☐ GitHub project

GitLab Connection

ssafy-gitlab

- “Do not allow the pipeline to resume if the controller restarts” 체크
 - GitLab Connection을 만들었던 설정으로 선택
- Webhook 설정 (Pipeline 설정 이어서)
1. Build Trigger에서 GitLab webhook 부분 체크, Push Event만 체크, webhook URL은 복사해 놓는다

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL **http://192.168.1.100:8081/project/backend** ?

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

2. 밑에 고급 버튼 클릭

Advanced configuration options for Jenkins:

- ☒ Approved Merge Requests (EE-only)
- ☒ Comments
- Comment (regex) for triggering a build [?](#)
Jenkins please retry a build
- ☐ GitHub hook trigger for GITScm polling [?](#)
- ☐ Poll SCM [?](#)

The '고급...' (Advanced) button is highlighted with a red box.

3. 밑에 Secret token을 Generate 버튼을 눌러 생성 후 복사해 놓는다

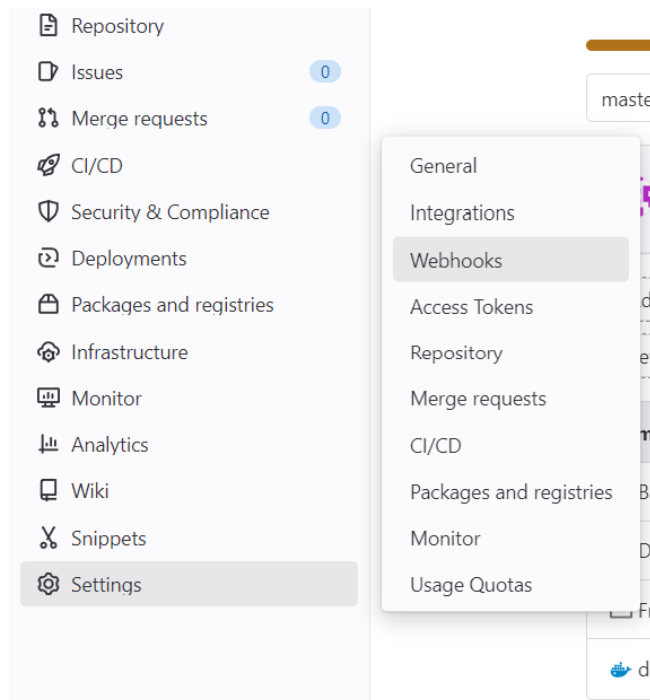
Secret token [?](#)

Generate

Clear

4. Webhook 설정 후에 테스트까지 하기위해 현재까지의 설정을 저장한다

5. GitLab 프로젝트 Repository에서 Settings - Webhooks에 진입



6. Webhooks에 다음과 같이 작성 후 밑에 Add webhook 클릭

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

URL

URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

Trigger

☒ Push events

Push to the repository.

SSL verification

☒ Enable SSL verification

Add webhook

- **URL** : 복사해왔던 URL에서 포트번호만 Jenkins 컨테이너에서 포트포워딩한 포트번호로 바꾼다
- **Secret token** : 복사해왔던 Secret token 붙여넣기
- **Trigger** : Push events만 체크
- **SSL verification** : “Enable SSL verification” 체크

7. 아래 생성된 webhook에서 Test - Push events를 눌러서 테스트

A feature flag is turned on or off.

☐ Releases events
A release is created or updated.

SSL verification

☒ Enable SSL verification

Add webhook

Project Hooks (2)

Push Events Tag Push Events SSL Verification: enabled

Test Edit Delete

Push events
Tag push events
Issues events
Confidential issues events
Note events
Confidential note events
Merge requests events
Job events
Pipeline events

8. 위에 HTTP 200이라고 뜬다면 성공

A screenshot of a webhooks configuration interface. At the top, a light blue notification bar with a close button (X) displays the message: "Hook executed successfully: HTTP 200". Below this is a search bar labeled "Search page". The main section is titled "Webhooks" and includes a brief description: "Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook." To the right, there are input fields for "URL" (containing "http://example.com/trigger-cijson") and "Secret token". A note below the URL field states: "URL must be percent-encoded if it contains one or more special characters."

- Pipeline script from SCM 설정

1. Pipeline - Definition을 “Pipeline script form SCM”으로 고르고 다음과 같이 설정한다

A screenshot of the "Pipeline" configuration page. The "Definition" dropdown is set to "Pipeline script from SCM". Below this, the "SCM" dropdown is set to "Git". Under the "Repositories" section, the "Repository URL" is set to "https://lab.ssfy.com/s08-webmobile1-sub2/S08P12A108" and the "Credentials" dropdown is set to "gitlab account". There is an "+ Add" button and a "고급..." (Advanced) link at the bottom of the repository configuration area.

- **Repository URL** : GitLab 프로젝트 Repository url을 입력
- **Credentials** : 만들어둔 GitLab 로그인 계정 선택

2. 아래 부분을 다음과 같이 작성 후 저장

Branches to build ?

Branch Specifier (blank for 'any') ?

*/dev-back

Add Branch

Repository browser ?

(자동)

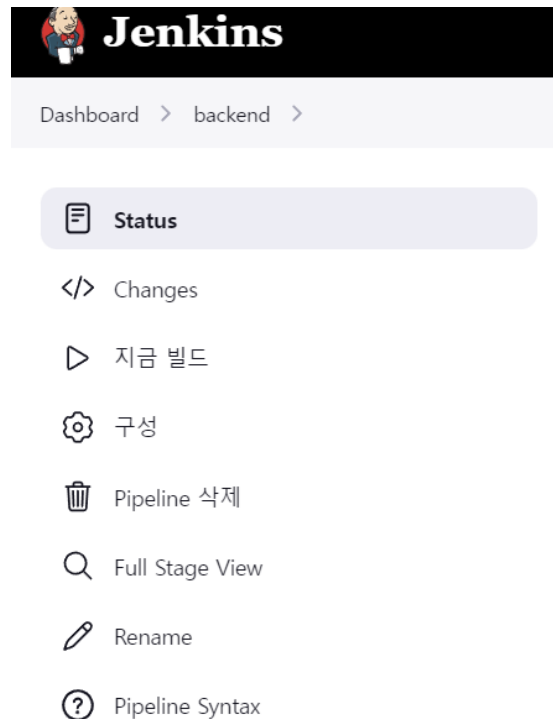
Additional Behaviours

Add ▾

Script Path ?

./Backend/Jenkinsfile

- **Branch Specifier** : Backend 코드가 배포용으로 최종 merge 되는 브랜치 선택
- **Script path** : 선택한 브랜치내에서 Backend의 Jenkinsfile의 경로 입력
- 초기 빌드로 git clone 후 application.yml 파일 이동



1. 좌측 사이드바에서 “지금 빌드”를 클릭한다
2. ec2 콘솔에서 `/jenkins/workspace/backend/` 경로로 이동하여 브랜치가 정상적으로 clone이 되었는지 확인

```
cd /jenkins/workspace/backend
```

```
ls
```

3. /jenkins/workspace/backend/Backend/src/main/ 경로로 이동하여 resources 폴더를 만들고 application.yml 파일을 이동시킨다

```
cd /jenkins/workspace/backend/Backend/src/main
```

```
mkdir resources
```

```
# /home/ubuntu 위치에 application.yml이 있다고 하면  
mv ~/application.yml ./resources
```

4. 이후 “지금 빌드”를 누르거나 GitLab에 push가 감지되면 자동으로 빌드&배포가 된다

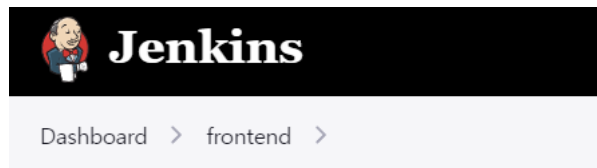
6. Frontend CI/CD

- front.conf 작성 : **I-3 설정 파일 및 환경 변수 정보**의 Frontend 항목을 참조하여 파일을 작성하고 ec2 머신의 /home/ubuntu 경로에 준비해 놓는다
- Dockerfile 작성 : **I-3 설정 파일 및 환경 변수 정보**의 Frontend 항목을 참조하여 작성후 프로젝트의 Frontend 폴더 최상위에 넣어서 GitLab Repository에 push한다
- Jenkinsfile 작성 : **I-3 설정 파일 및 환경 변수 정보**의 Frontend 항목 참조하여 작성후 프로젝트의 end 폴더 최상위에 넣어서 GitLab Repository에 push한다
- GitLab 설정 : Backend에서 설정했다면 pass
- Pipeline item 생성 : Backend와 과정 동일
- Webhook 설정 : Backend와 과정 동일

- Pipeline script from SCM 설정 : Backend와 과정과 대부분 동일하지만 아래 설정 주의

The screenshot shows the 'Branches to build' section of the Jenkins Pipeline configuration. It includes a 'Branch Specifier (blank for \'any\')' field with the value '*/dev-front'. Below this is an 'Add Branch' button. The 'Repository browser' is set to '(자동)'. There is an 'Additional Behaviours' section with an 'Add' button. At the bottom, the 'Script Path' is set to './Frontend/Jenkinsfile'.

- **Branch Specifier** : Frontend 코드가 배포용으로 최종 merge 되는 브랜치 선택
 - **Script path** : 선택한 브랜치내에서 Frontend의 Jenkinsfile의 경로 입력
- 초기 빌드로 git clone 후 front.conf 파일 이동
 1. 좌측 사이드바에 “지금 빌드”를 클릭한다



- Status
- Changes
- 지금 빌드
- 구성
- Pipeline 삭제
- Full Stage View
- Rename
- Pipeline Syntax

2. ec2 콘솔에서 `/jenkins/workspace/frontend/` 경로로 이동하여 브랜치가 정상적으로 clone이 되었는지 확인

```
cd /jenkins/workspace/frontend  
ls
```

3. `/jenkins/workspace/frontend/Frontend/` 경로로 `front.conf` 파일 이동

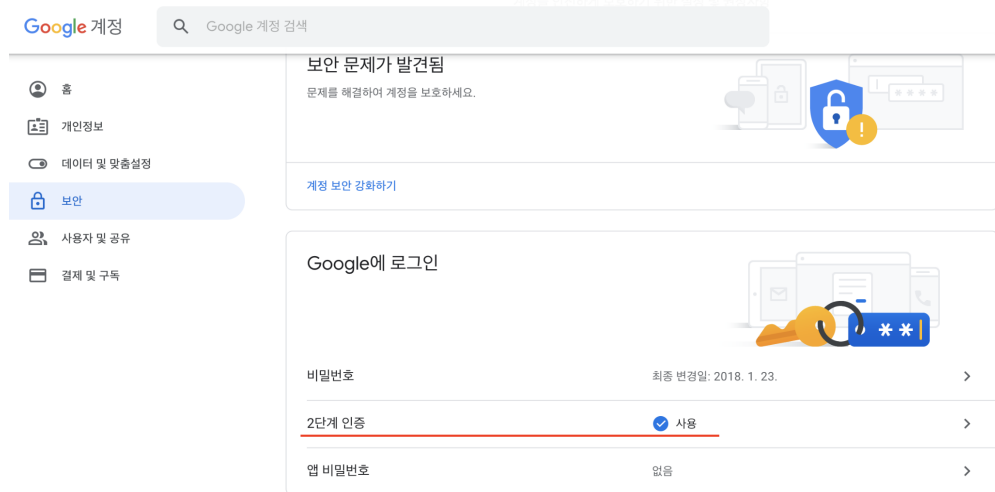
```
# /home/ubuntu 위치에 front.conf가 있다고 하면  
mv ~/front.conf /jenkins/workspace/frontend/Frontend
```

4. 이후 “지금 빌드”를 누르거나 GitLab에 push가 감지되면 자동으로 빌드&배포가 된다

III. 외부 서비스

1. Google SMTP

- Google 계정 설정
 1. Google 계정 로그인
 2. Google 계정 관리 > 보안 > Google에 로그인 > 2단계 인증 설정



3. Google 계정 관리 > 보안 > Google에 로그인 > 앱 비밀번호 설정

← 앱 비밀번호

앱 비밀번호를 사용하면 2단계 인증을 지원하지 않는 기기의 앱에서 Google 계정에 로그인할 수 있습니다. 비밀번호를 한 번만 입력하면 기억할 필요가 없습니다. [자세히 알아보기](#)

앱 비밀번호가 없습니다.

앱 비밀번호를 생성할 앱 및 기기를 선택하세요.

메일

▼

Windows 컴퓨터

▼

생성

4. 생성된 앱 비밀번호를 복사해 놓는다

생성된 앱 비밀번호

Windows 컴퓨터용 앱 비밀번호

terf

사용 방법

1. '메일' 앱을 엽니다.
2. '설정' 메뉴를 엽니다.
3. '계정'을 선택한 뒤 내 Google 계정을 선택합니다.
4. 비밀번호를 위에 표시된 16자리 비밀번호로 교체합니다.

일반적인 비밀번호와 마찬가지로 이 앱 비밀번호는 Google 계정에 대한 완전한 액세스 권한을 부여합니다. 비밀번호를 기억하지 않아도 되므로 적어 놓거나 다른 사용자와 공유하지 마세요.

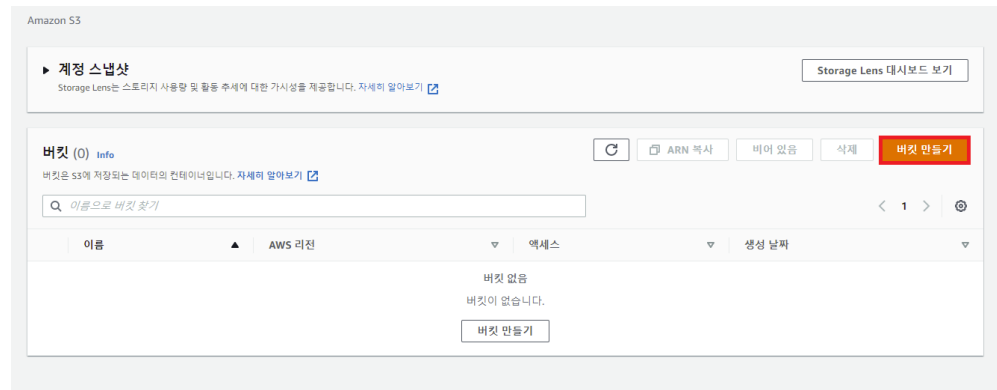
[자세히 알아보기](#)

확인

- application.yml에 설정 추가 : **I-3 설정 파일 및 환경 변수 정보**의 Backend 항목에서 application.yml에 spring.mail 부분 참조

2. AWS S3 Bucket

- AWS 프리티어 계정 생성
- S3 버킷 생성
 1. Amazon S3 서비스로 이동
 2. 버킷 만들기 클릭



3. 버킷 이름 및 AWS리전 설정



4. 객체 소유권을 “ACL활성화됨 - 버킷 소유자 선호”로 설정

객체 소유권 Info
다른 AWS 계정에서 이 버킷에 작성한 객체의 소유권 및 액세스 제어 목록(ACL)의 사용을 제어합니다. 객체 소유권은 객체에 대한 액세스를 지정할 수 있는 사용자를 결정합니다.

☐ **ACL 비활성화됨(권장)**
이 버킷의 모든 객체는 이 계정이 소유합니다. 이 버킷과 그 객체에 대한 액세스는 정책을 통해서만 지정됩니다.

☒ **ACL 활성화됨**
이 버킷의 객체는 다른 AWS 계정에서 소유할 수 있습니다. 이 버킷 및 객체에 대한 액세스는 ACL을 사용하여 지정할 수 있습니다.

객체 소유권
☒ **버킷 소유자 선호**
이 버킷에 작성된 새 객체가 bucket-owner-full-control 삽입 ACL을 지정하는 경우 새 객체는 버킷 소유자가 소유합니다. 그렇지 않은 경우 객체 라이터가 소유합니다.
☐ **객체 라이터**
객체 라이터는 객체 소유자로 유지됩니다.

새 객체에 대해서만 객체 소유권을 적용하려면 버킷 정책이 객체 업로드에 bucket-owner-full-control 삽입 ACL을 요구하도록 지정해야 합니다. 자세히 알아보기

ACL 활성화와 관련 향후 권한 변경 사항
2023년 4월부터는 S3 콘솔을 사용하여 버킷을 생성할 때 ACL을 활성화하기 위해 s3:PutBucketOwnershipControls 권한이 있어야 합니다. 자세히 알아보기

5. 퍼블릭 액세스 차단 설정을 다음과 같이 한다

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 지정 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단**
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 새 버킷 및 액세스 지정 정책을 차단합니다. 이 설정은 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 정책을 변경하지 않습니다.

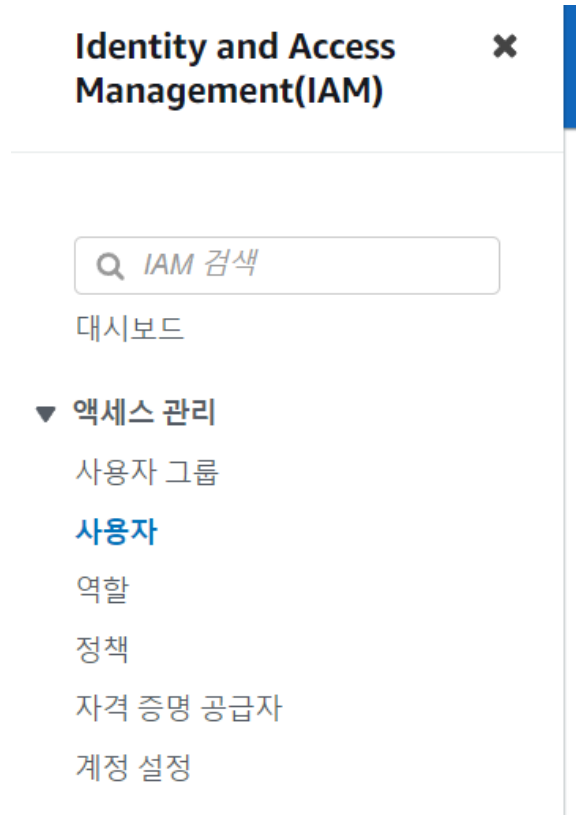
☒ **임의의 퍼블릭 버킷 또는 액세스 지정 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단**
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 정책을 사용하는 버킷 또는 액세스 지정에 대한 퍼블릭 및 교차 계정 액세스를 무시합니다.

모든 퍼블릭 액세스 차단을 비활성화하면 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있습니다.
정적 웹 사이트 호스팅과 같은 구체적으로 확인된 사용 사례에서 퍼블릭 액세스가 필요한 경우가 아니면 모든 퍼블릭 액세스 차단을 활성화하는 것이 좋습니다.

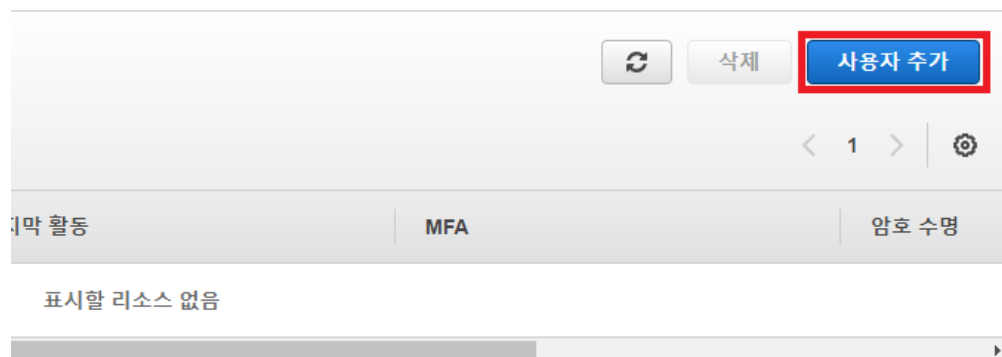
☒ **현재 설정으로 인해 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있음을 알고 있습니다.**

- IAM 사용자 추가

1. Identity and Access Management(IAM) 서비스로 이동
2. 좌측 메뉴에서 “액세스 관리 - 사용자” 클릭



3. 우상단 “사용자 추가” 클릭



4. 사용자 이름 입력 후 “다음”

사용자 세부 정보 지정

사용자 세부 정보

사용자 이름

사용자 이름은 최대 64자까지 가능합니다. 유효한 문자: A-Z, a-z, 0-9 및 +, -, @, _ (<하이픈>)

☐ AWS Management Console에 대한 사용자 액세스 권한 제공 - 선택 사항

사용자에게 콘솔 액세스 권한을 제공하는 경우 IAM Identity Center에서 액세스를 관리하는 것은 모범 사례입니다.

☒ 이 IAM 사용자를 생성한 후 액세스 키 또는 AWS CodeCommit이나 Amazon Keyspaces에 대한 서비스별 보안 인증 정보를 통해 프로그래밍 방식 액세스를 생성할 수 있습니다. 자세히 알아보기

취소

다음

5. 권한 설정에서 “직접 정책 연결”을 고르고 ‘S3’를 검색해서 “AmazonS3FullAccess” 체크

권한 설정

기존 그룹에 사용자를 추가하거나 새 그룹을 생성합니다. 직무별로 사용자의 권한을 관리하려면 그룹을 사용하는 것이 좋습니다. 자세히 알아보기

권한 옵션

☐ 그룹에 사용자 추가
기존 그룹에 사용자를 추가하거나 새 그룹을 생성합니다. 그룹을 사용하여 직무별로 사용자 권한을 관리하는 것이 좋습니다.

☐ 권한 복사
기존 사용자의 모든 그룹 멤버십, 연결된 관리형 정책 및 인라인 정책을 복사합니다.

☒ 직접 정책 연결
관리형 정책을 사용자에게 직접 연결합니다. 사용자에게 연결하는 대신, 정책을 그룹에 연결한 후 사용자를 적절한 그룹에 추가하는 것이 좋습니다.

권한 정책 (1/1043)

새 사용자와 연결할 정책을 하나 이상 선택합니다.

Q 텍스트 또는 값을 기준으로 빠르게 필터링

11 개 일치

S3 X

필터 지우기

<input type="checkbox"/>	정책 이름	유형	연결된 엔터티
<input type="checkbox"/>	AmazonDMSRedshiftS3Role	AWS 관리형	0
<input checked="" type="checkbox"/>	AmazonS3FullAccess	AWS 관리형	1
<input type="checkbox"/>	AmazonS3OutpostsAmazonS3OutpostsExecution	AWS 관리형	0

6. 검토 후 “사용자 생성” 클릭

검토 및 생성

선택 사항을 검토합니다. 사용자를 생성한 후 자동 생성된 암호를 보고 다운로드할 수 있습니다(활성화된 경우).

사용자 세부 정보

사용자 이름

test

콘솔 암호 유형

None

암호 재설정 필요

아니요

권한 요약

< 1 >

이름	유형	다음과 같이 사용
AmazonS3FullAccess	AWS 관리형	권한 정책

태그 - 선택 사항

태그는 리소스를 식별, 구성 또는 검색하는 데 도움이 되도록 AWS 리소스에 추가할 수 있는 키 값 페어입니다. 이 사용자와 연결된 태그를 선택합니다.

리소스와 연결된 태그가 없습니다.

새 태그 추가

최대 50개의 태그를 더 추가할 수 있습니다.

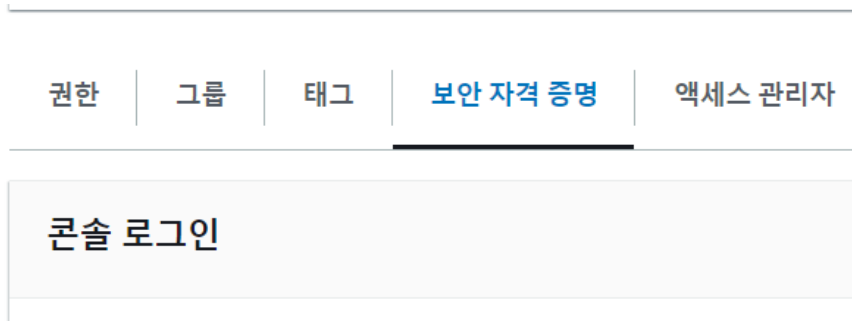
취소

이전

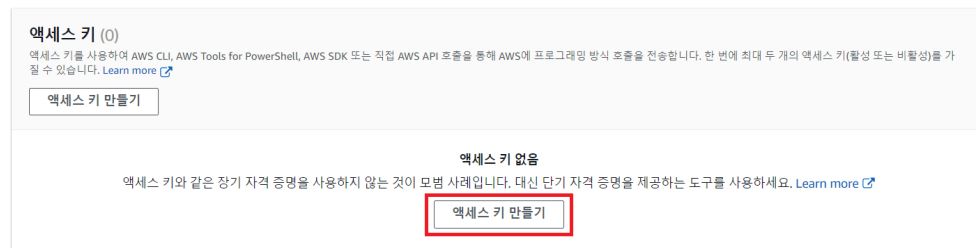
사용자 생성

7. 생성된 사용자를 목록에서 클릭하여 상세 정보로 진입

8. “보안 자격 증명” 탭 클릭



9. “액세스 키” 부분에서 “액세스 키 만들기” 클릭



10. “액세스 키 모범 사례 및 대안”에서 “AWS 외부에서 실행되는 어플리케이션”을 체크

액세스 키 모범 사례 및 대안

보안 개선을 위해 액세스 키와 같은 장기 자격 증명을 사용하지 마세요. 다음과 같은 사용 사례와 대안을 고려하세요.



11. 설명 입력 후 “액세스 키 만들기” 클릭

설명 태그 설정 - 선택 사항

이 액세스 키에 대한 설명은 이 사용자에게 태그로 연결되고, 액세스 키와 함께 표시됩니다.

설명 태그 값



이 액세스 키의 용도와 사용 위치를 설명합니다. 좋은 설명은 나중에 이 액세스 키를 자신있게 교체하는 데 유용합니다.

최대 256자까지 가능합니다. 허용되는 문자는 문자, 숫자, UTF-8로 표현할 수 있는 공백 및 _ . / = + - @입니다.

[취소](#) [이전](#) [액세스 키 만들기](#)

12. 생성된 액세스 키와 비밀 액세스 키를 복사해 놓는다 (application.yml에 입력)

액세스 키 검색

액세스 키	
분실하거나 잊어버린 비밀 액세스 키는 검색할 수 없습니다. 대신 새 액세스 키를 생성하고 이전 키를 비활성화합니다.	
액세스 키	비밀 액세스 키
 AKIA3E5X98XYM27SECN	 ***** 표시

- application.yml에 설정 추가 : **I-3 설정 파일 및 환경 변수 정보**의 Backend 항목에서 application.yml에 cloud.aws 부분 참조