



hikingdom Porting Manual



등산으로 완성하는 우리의 왕국, **hikingdom**

1. 개발 환경

형상 관리

- Gitlab

이슈 관리

- Jira

UI/UX

- Figma

협업 도구

- Discord
- Notion

IDE

- Visual Studio Code 1.75
- IntelliJ IDEA 2022.3.1
- DataGrip 2022.3.2

OS

- Windows 10
- Ubuntu 20.04 LTS

Database

- MySQL 8.0.33
- MongoDB 5.0.15
- Redis 7.0.8

Infra

- Jenkins
- docker compose v2.17.2
- nginx

Server

- AWS EC2 t3.large
- AWS S3

Front-end

- Node.js 18.14.2
- React 18.2.0
 - React Query 4.29.5
 - Recoil 0.7.7
- Sass 1.62.1
- three.js 0.152.2
- stomp 7.0.0

Back-end

- Java OpenJDK 11
- Spring Boot 2.7.9
 - Spring Data JPA
 - Spring Security
 - Spring Cloud Gateway 3.1.4
 - Spring Cloud Netflix (Eureka)

Android

- Kotlin 1.4.32
- Gradle JDK: JetBrains runtime 11.0.15
- Gradle 7.0.2
- Compile SDK version: API level 31

2. 배포 관련 환경 설정

2.1. Docker 설치 [공식 문서]

1. 이전 Docker 관련 라이브러리 삭제

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

2. Docker 레포지토리 설정

- **apt** 패키지 업데이트 및 기타 패키지 설치

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
```

- Docker official GPG 키 추가

```
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/ap
t/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

- 레포지토리 설정

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

3. Docker 설치

- **apt** 패키지 업데이트

```
sudo apt-get update
```

- 최신 Docker Engine, containerd, Docker Compose 설치

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

4. Docker 서비스 실행 및 부팅 시 자동 실행 설정

```
sudo systemctl start docker
sudo systemctl enable docker
```

5. Docker 그룹에 현재 계정 추가

```
sudo usermod -aG docker ${USER}
sudo systemctl restart docker
```

6. Docker 설치 확인

```
docker -v
```

2.2. Jenkins 설치 (Docker-in-Docker 방식)

1. `jenkins/jenkins:latest-jdk11` 이미지를 이용해 Docker 컨테이너 실행

```
docker run -d -p 9090:8080 -v /var/jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins jenkins/jenkins:latest-jdk11
```

2. Jenkins 내부에 Docker 설치

- `root` 계정으로 Jenkins 컨테이너 접속

```
docker exec -itu 0 jenkins bash
```

- Docker 설치 명령어 실행

```
apt-get update && \
apt-get -y install apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    zip \
    unzip \
    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
```

- Docker 그룹 변경

```
chown root:docker /var/run/docker.sock
```

- Jenkins 컨테이너에서 `exit` 한 후, 컨테이너 재실행

```
docker restart jenkins
```

3. Jenkins 초기 환경 설정

- Jenkins 실행 로그에서 `initialAdminPassword` 확인

```
docker logs -f jenkins
```

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

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

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

- Suggested Plugins 설치
- Gitlab, Docker 플러그인 설치

2.3. Jenkins Project 설정

1. Gitlab에서 Project Access Token 생성
 - Gitlab Repository > Settings > Access Tokens
2. Jenkins에서 Freestyle Project 생성
3. Jenkins Project에서 소스 코드 관리 설정

소스 코드 관리

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://gitlab+deploy-token-5178: @lab.ssafy.com/s08-final/S08P31A102.git

Credentials ?

- none -

Add ▾

고급 ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

main

Add Branch

- Repository URL에 Gitlab에서 발급 받은 Project Access Token 입력
- Branch에 해당하는 브랜치 입력

4. Jenkins Project에서 빌드 유발 설정

- 빌드 유발할 이벤트 설정

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://k8a102.p.ssafy.io:9090/project/main ?

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 설정 아래 Gitlab Webhook 설정 시 사용할 Secret token 발급

고급 ^

- ☒ 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 ?

Generate

5. Gitlab에서 Webhooks 설정

- Gitlab Repository > Settings > Webhooks
 - Jenkins에서 Gitlab Webhook URL과 Secret token 입력

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.

- ☐ Tag push events

A new tag is pushed to the repository.

6. Jenkins Project에서 Build Steps 추가

- Execute shell

```
cd backend
chmod +x ./gradlew
./gradlew clean build -x test -x asciidoctor

cd ..
docker image prune -f
```

```
docker-compose build
docker-compose up -d
```

2.4. 데이터베이스 관련 환경 설정

2.4.1. MySQL

1. Docker volume 생성

```
docker volume create mysql-volume
```

2. Docker 컨테이너 실행

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD= -it -d -p 3306:3306 -v mysql-volume:/var/lib/mysql mysql:latest
```

3. Docker 컨테이너 내부 접속

```
docker exec -it mysql bash
```

4. MySQL 환경 설정 진행

- root 계정으로 접속

```
mysql -u root -p
```

- 사용자 생성

```
create user '아이디'@'%' identified by '비밀번호';
```

- 권한 설정

```
grant all privileges on *.* to '아이디'@'%';
```

- 데이터베이스 생성

```
CREATE DATABASE 데이터베이스명;
```

- 한국 시간으로 시간대 설정

```
SET GLOBAL time_zone='Asia/Seoul';  
SET time_zone='Asia/Seoul';
```

2.4.2. redis

```
docker run --name redis -p 6379:6379 redis:alpine --requirepass [비밀번호]
```

2.4.3. mongoDB

```
docker run -d --network mongo-network --name mongod \\\n  -e MONGO_INITDB_ROOT_USERNAME=[사용자명] \\\n  -e MONGO_INITDB_ROOT_PASSWORD=[비밀번호] \\\n  -e MONGO_INITDB_DATABASE=[데이터베이스명] \\\n  -p 27017:27017 mongo:5.0.15
```

3. 주요 환경 설정 파일

3.1. Front-end 설정 파일

3.1.1. `.env.production` , `.env.local`

```
REACT_APP_API_BASE_URL=  
REACT_APP_KAKAOMAP_API_KEY=
```

3.2. Back-end 설정 파일

3.2.1. `discovery` 서버 `application.yml`

```
server:  
  port: 8761  
  
spring:  
  application:  
    name: discovery  
  
eureka:  
  client:  
    register-with-eureka: false  
    fetch-registry: false
```


3.2.2. **service** 서버 설정 파일

- **application.yml**

```
eureka:
  instance:
    instance-id: api-service-instance
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://${EUREKA_SERVER:localhost}:8761/eureka

spring:
  application:
    name: api-service
  jpa:
    hibernate:
      ddl-auto: update
      generate-ddl: true
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
    username:
    password:
  redis:
    host:
    port:
    password:
    lettuce:
      pool:
        max-active: 8
        max-idle: 8
        min-idle: 0
        max-wait: -1
  mail:
    host:
    port:
    username:
    password:
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true
  security:
    jwt:
      token:
        secret-key:
        expire-length:
        refresh-expire-length:
  cloud:
    aws:
      s3:
        bucket:
        region:
          static: ap-northeast-2 #Asia Pacific -> seoul
      stack:
        auto: false
      credentials:
        access-key:
```

```

    secret-key:
values:
  mail:
    setFrom:
  profile:
    default:
  password:
    possibleChars:
  asset:
    defaultAsset:

```

- **firebase 설정 json 파일** (`resources/firebase` 아래 위치)

```

{
  "type":
  "project_id":
  "private_key_id":
  "private_key":
  "client_email":
  "client_id":
  "auth_uri":
  "token_uri":
  "auth_provider_x509_cert_url":
  "client_x509_cert_url":
  "universe_domain":
}

```

3.2.3. `chat` 서버 `application.yml`

```

eureka:
  instance:
    instance-id: chat-service-instance
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://${EUREKA_SERVER:localhost}:8761/eureka

spring:
  application:
    name: chat-service
  jpa:
    hibernate:
      ddl-auto: update
      generate-ddl: true
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
    username:
    password:
  data:
    mongodb:
      host:
      port:
      authentication-database:
      username:
      password:
      database:

```

```

    auto-index-creation: true
security:
  jwt:
    token:
      secret-key:
      expire-length:
      refresh-expire-length:

```

3.2.4. `gateway` 서버 `application.yml`

```

eureka:
  instance:
    instance-id: gateway-service-instance
  client:
    fetch-registry: true
    register-with-eureka: true
    prefer-ip-address: true
    service-url:
      defaultZone: http://${EUREKA_SERVER:localhost}:8761/eureka

spring:
  application:
    name: gateway-service

cloud:
  gateway:
    default-filters:
      - DedupeResponseHeader=Access-Control-Allow-Origin Access-Control-Allow-Credentials
    globalcors:
      cors-configurations:
        '[/*]*':
          allowedOrigins:
            - "https://hikingdom.kr"
          allowedHeaders: "*"
          allowedMethods: "*"
          allowCredentials: true
    routes:
      - id: hiking-share-service
        uri: lb:ws://HIKING-SHARE-SERVICE
        predicates:
          - Path=/api/hiking/**
        filters:
          - RewritePath=/api/hiking/(?<segment>.*), /${segment}
      - id: api-service
        uri: lb://API-SERVICE
        predicates:
          - Path=/api/**
      - id: chat-service
        uri: lb://CHAT-SERVICE
        predicates:
          - Path=/chat/**

```

3.2.5. `batch` 서버 `application.yml`

```

server:
  port:

```

```
spring:
  jpa:
    hibernate:
      ddl-auto: update
      generate-ddl: true
    datasource:
      driver-class-name: com.mysql.cj.jdbc.Driver
      url:
      username:
      password:
```

3.3. Android 설정 파일

3.3.1. `local.properties`

- Android 프로젝트 최초 실행 시, `Gradle Scripts/local.properties` 에 아래 내용 추가

```
# Develop Server Url
DEV_URL=

# Production Server Url
PROD_URL=

KAKAOMAP_NATIVE_APP_KEY=
```

- `local.properties` 가 해당 경로에 없다면, 보이면 `File > Sync Project with Gradle Files` 실행

3.3.2. 카카오 지도 관련 설정 [공식 문서]

- `libDaumMapAndroid.jar` 추가
 - `Project` view 상태에서 `/app/libs/` 아래 복사
- `AndroidManifest.xml` 에 Permission 과 APP KEY 추가
 - Permission 추가

```
<uses-permission android:name="android.permission.INTERNET">
</uses-permission>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION">
</uses-permission>
```

- APP KEY 추가

```
<meta-data android:name="com.kakao.sdk.AppKey" android:value="XXXXXXXXXXXXXXXXXXXXXXX" />
```

4. 외부 서비스 관련 설정

4.1. Gmail

1. 구글 계정 생성
2. 구글 프로필 > 계정 설정 > 2단계 인증 활성화
3. 2단계 인증 활성화 이후 앱 비밀번호 발급
 - 앱 선택 : 기타(맞춤 이름)
 - 앱 비밀번호를 생성할 이름 입력 후 비밀번호 생성
4. 생성된 앱 비밀번호를 `application.yml` 에 입력

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: {google 메일 아이디}
    password: {google 계정 앱 key}
```

4. Gmail > 설정 > 빠른 설정/ 모든 설정 보기 > 전달 및 POP/IMAP > 이메일 보내기 설정

기본설정 라벨 받은편지함 계정 및 가져오기 필터 및 차단된 주소 전달 및 POP/IMAP 부가기능 채팅 및 Meet 고급 오프라인 테마

전달: 자세히 알아보기 전달 주소 추가

도움말: 필터를 만들면 메일 중 일부만 전달할 수도 있습니다.

POP 다운로드: 자세히 알아보기

1. 상태: 모든 메일에 대해 POP가 사용 설정되어 있습니다.

☐ 이미 다운로드 된 메일을 포함하여 모든 메일에 POP를 활성화 하기

☐ 지금부터 수신되는 메일에만 POP를 사용하기

☐ POP 사용 안함

2. POP로 메시지를 여는 경우 [Gmail 사본을 받은편지함에 보관하기]

3. 이메일 클라이언트 구성 (예: Outlook, Eudora, Netscape Mail) 설정 방법

IMAP 액세스: (IMAP를 사용하여 다른 클라이언트에서 Gmail에 액세스) 자세히 알아보기

상태: IMAP를 사용할 수 있습니다.

☒ IMAP 사용

☐ IMAP 사용 안함

IMAP에서 메일을 삭제된 것으로 표시하는 경우:

☒ 자동 삭제 사용 - 서버를 즉시 업데이트(기본값)

☐ 자동 삭제 사용 안함 - 클라이언트가 서버를 업데이트할 때까지 대기

메일이 삭제된 것으로 표시되고 마지막으로 표시된 IMAP 폴더에서 삭제된 경우:

☒ 메일 보관(기본값)

☐ 메일을 휴지통으로 이동

☐ 메일을 즉시 완전삭제

폴더 크기 제한

☒ IMAP 폴더에서 메일 수를 제한하지 않습니다(기본값).

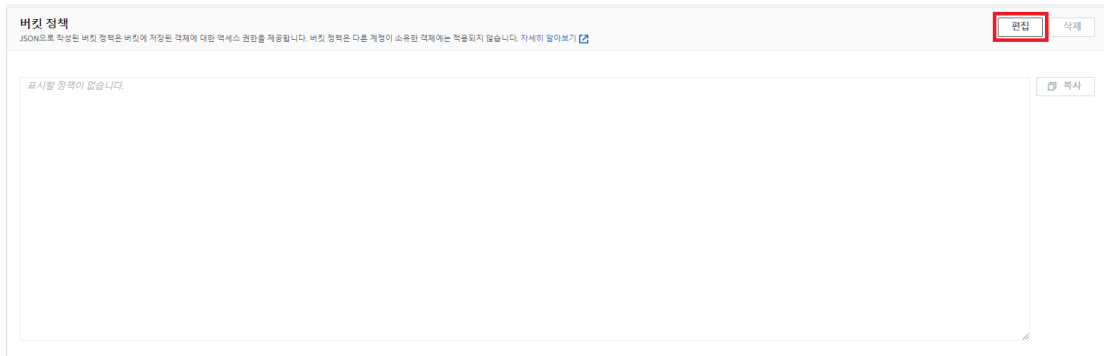
☐ 이만큼의 메일만 포함하도록 IMAP 폴더를 제한합니다. [1,000]

이메일 클라이언트 구성(예: Outlook, Thunderbird, iPhone) 설정 방법

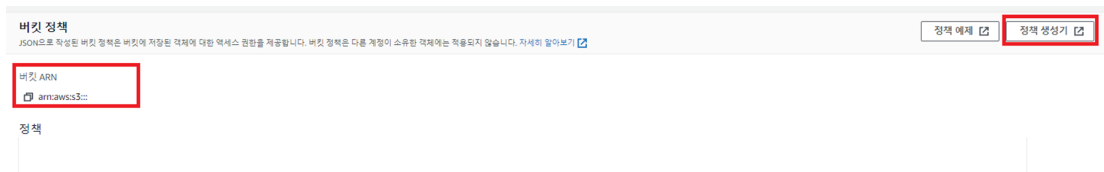
변경사항 저장 취소

4.2. Amazon S3

1. AWS 계정 설정
2. S3 > 버킷 > 버킷 생성
3. 버킷 권한 설정 및 정책 생성
 - 권한 > 버킷 정책 편집



- 정책 생성



- Action에 `GetObject`, `PutObject`, `DeleteObject` 3개를 체크하고 ARN에는 복사해둔 ARN값 입력

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an [IAM Policy](#), an [S3 Bucket Policy](#), an [SNS Policy](#), and a [Queue Policy](#).

Select Type of Policy S3 Bucket Policy

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See [a description of elements](#) that you can use in statements.

Effect ☒ Allow ☐ Deny

Principal *
Use a comma to separate multiple values.

AWS Service Amazon S3 ☐ All Services ("*")
Use multiple statements to add permissions for more than one service.

Actions 3 Action(s) Selected ☐ All Actions ("*")

Amazon Resource Name (ARN) arn:aws:s3::: ☐
ARN should follow the following format: `arn:aws:s3:::${BucketName}/${KeyName}`.
Use a comma to separate multiple values.

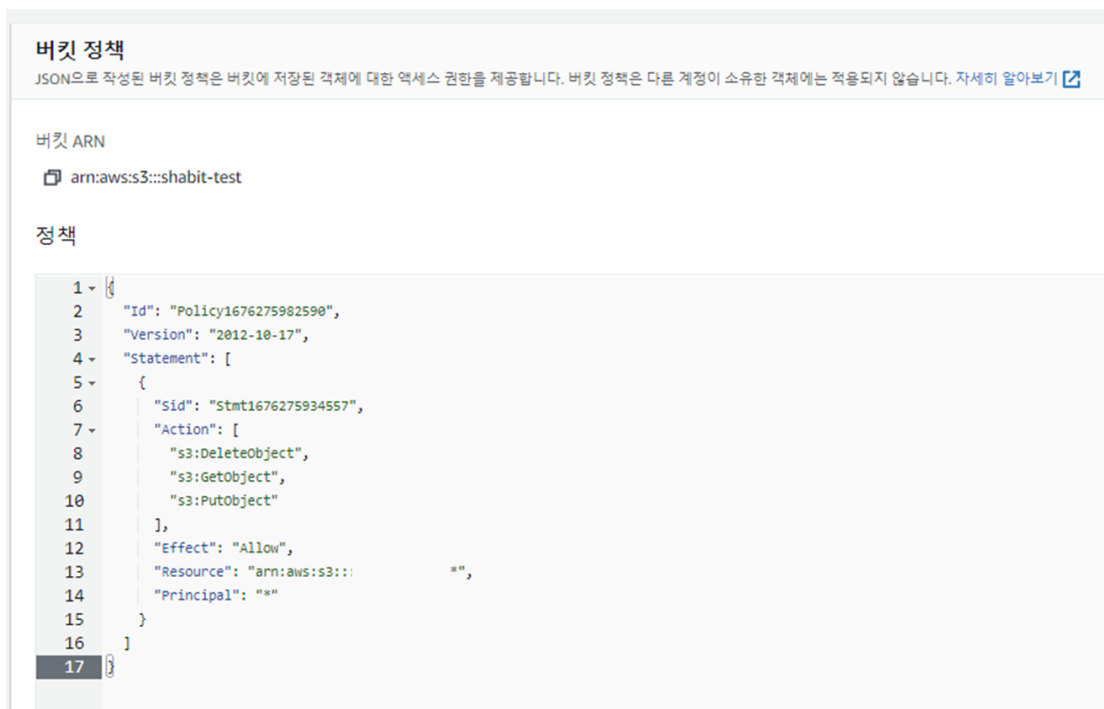
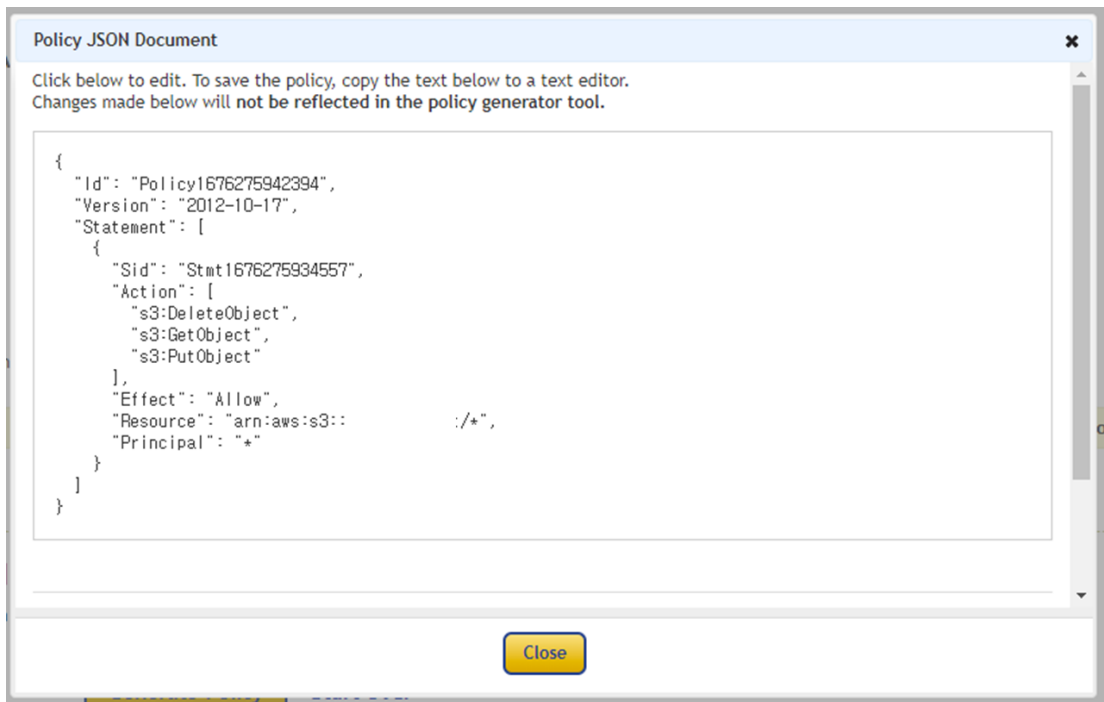
[Add Conditions \(Optional\)](#)

Add Statement

- ARN값을 입력하되 `/*` 값도 추가. ARN 값이 `arn:aws:s3:::test` 라 가정하면 `arn:aws:s3:::test/*` 라고 입력

- Generate Policy

- 생성된 정책을 복사한 뒤 정책 편집 부분에 붙여 넣은 후, 변경 사항 저장 버튼을 눌러 저장



4. 사용자 생성

사용자 세부 정보

사용자 이름

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

☒ 콘솔 액세스 활성화 - 선택 사항
사용자가 AWS 관리 콘솔에 로그인할 수 있도록 허용하는 암호를 활성화합니다.

콘솔 암호

☒ 자동 생성된 암호
사용자를 생성한 후 암호를 볼 수 있습니다.

☐ 사용자 지정 암호
사용자의 사용자 지정 암호를 입력합니다.

☐ 암호 표시

☒ 사용자는 다음 로그인 시 새 암호를 생성해야 합니다(권장).
사용자는 IAMUserChangePassword 정책을 자동으로 가져와 암호를 변경할 수 있도록 허용합니다.

[프로그래밍 방식의 액세스의 경우 사용자를 생성한 후 액세스 키를 생성할 수 있습니다. 자세히 알아보기](#)

5. 사용자 생성 및 Access Key 생성

- 보안 자격 증명 > 액세스 키 발급 (기타)

삭제

요약		
ARN arn:aws:iam:: :user/	콘솔 액세스 MFA 없이 활성화됨	액세스 키 1 활성화되지 않음
생성됨	마지막 콘솔 로그인 인 함	액세스 키 2 활성화되지 않음

권한 그룹 태그 **보안 자격 증명** 액세스 관리자

권한 정책 (1)
사용자에게 직접 연결된 정책을 통해 또는 그룹을 통해 권한을 정의합니다.

🔄 제거 권한 추가 ▼

6. 생성한 key 값을 application.yml 에 입력

```
cloud:
  aws:
    s3:
      bucket: 버킷 이름
      region:
        static: ap-northeast-2 # Asia Pacific -> seoul
      stack:
        auto: false
      credentials:
        access-key: S3 사용자 access-key
        secret-key: S3 사용자 secret-key
```