포팅매뉴얼

목차

I. 빌드 및 배포

1. 실행환경

1. Server: AWS EC2 Ubuntu 20.04.6 LTS
2. Visual Studio Code: 1.81
3. IntelliJ IDEA: 2023.1.3 (Ultimate Edition) 17.0.7+10-b829.16 amd64
4. JVM: Zulu OpenJDK 11
5. Docker: 24.0.5
6. Node.js: 18.16.1
7. MySQL: 8.0.34
8. Jenkins: 2.416
9. Cloud Storage: AWS S3
10. OCR API: Naver Cloud Clova OCR

11. Python: 3.11.5

2. 설정 파일 목록 (경로 및 파일은 배포 과정 중 생성 될 수 있음)

1. React
    a. Dockerfile : ./frontend
2. Spring Boot
    a. Dockerfile : ./backend/namani
    b. application.properties : ./backend/namani/src/main/resources
    c. application-prod.properties : ./backend/namani/src/main/resources
    d. application-local.properties : ./backend/namani/src/main/resources
3. Fast API
    a. Dockerfile : ./ML
4. Docker
    a. docker-compose.yml : (Docker의 Jenkins 컨테이너 내) /var/jenkins_home
    b. .env : (EC2 인스턴스 내) /var/jenkins_home/workspace/Namani/config
5. Nginx
    a. default.conf : (EC2 인스턴스 내) /etc/nginx/sites-available

3. 설정 파일 및 환경 변수 정보

1. React
    a. Dockerfile :

```
FROM node:18.16.1
# http -> https 시 웹소켓 오류 때문에
ENV WDS_SOCKET_PORT 0
WORKDIR /app
COPY package.json .
RUN npm install
# 현재 디렉토리의 모든 파일을 도커 컨테이너 워킹 디렉토리에 저장
COPY . .
EXPOSE 3000
CMD ["npm","start"]
```

2. Spring Boot
    a. Dockerfile :

```
FROM adoptopenjdk/openjdk11
EXPOSE 8081
ARG JAR_FILE=build/libs/namani-0.0.1-SNAPSHOT.jar
ADD ${JAR_FILE} app.jar
# Tomcat에서 blocking issue 때문에
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

    b. application.properties :

```
server.servlet.context-path=/api
server.port=8081
spring.profiles.default=prod
spring.jackson.time-zone=Asia/Seoul
spring.jackson.serialization.fail-on-empty-beans=false
#jwt
jwt.secret=secretKey =${JWT_SECRET_KEY}
#S3
```

```
cloud.aws.credentials.accessKey=${S3_ACCESS_KEY}
cloud.aws.credentials.secretKey=${S3_SECRET_KEY}
cloud.aws.stack.auto=false

# AWS S3 Service bucket
cloud.aws.s3.bucket=${BUCKET_NAME}
cloud.aws.region.static=ap-northeast-2

# AWS S3 Bucket URL
cloud.aws.s3.bucket.url=https://s3.ap-northeast-2.amazonaws.com/${BUCKET_NAME}

#multipartfile max file size setting
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB

# ClovaOCR Keys
clova.secret=${CLOVA_SECRET}
clova.api.url=${CLOVA_API_URL}
```

c. application-prod.properties :

```
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=${LOCAL_DATASOURCE_URL}
spring.datasource.username=${LOCAL_DATASOURCE_USERNAME}
spring.datasource.password=${LOCAL_DATASOURCE_PASSWORD}
spring.jpa.hibernate.ddl-auto=update
#jpa query log
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.highlight_sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
#logger setting
logging.level.root=info
logging.level.com.ssafy.namani=debug
```

d. application-local.properties :

```
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=${PROD_DATASOURCE_URL}
spring.datasource.username=${PROD_DATASOURCE_USERNAME}
spring.datasource.password=${PROD_DATASOURCE_PASSWORD}
spring.jpa.hibernate.ddl-auto=update
#jpa query log
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.highlight_sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
#logger setting
logging.level.root=info
logging.level.com.ssafy.namani=debug
```

3. Fast API

a. Dockerfile :

```
FROM python:3.11
WORKDIR /fast

COPY ./requirements.txt /fast/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /fast/requirements.txt

COPY . /fast
EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

4. Docker

a. docker-compose.yml :

```
version: '3.8'
services:
  backend:
    image: ${dockerhub username}/backend:latest
    build:
      context: ./workspace/Namani/backend/namani
```

```
    dockerfile: Dockerfile

frontend:
  image: ${dockerhub username}/frontend:latest
  environment:
    - WDS_SOCKET_PORT=0
  build:
    context: ./workspace/Namani/frontend
    dockerfile: Dockerfile

ml:
  image: ${dockerhub username}/ml:latest
  build:
    context: ./workspace/Namani/ML
    dockerfile: Dockerfile
  ports:
    - 8000:8000
  volumes:
    - ./:/fast
```

b. .env :

```
PROD_DATASOURCE_URL=${your mysql or mariadb url}
PROD_DATASOURCE_USERNAME=${your db username}
PROD_DATASOURCE_PASSWORD=${your db password}
WDS_SOCKET_PORT=0
JWT_SECRET_KEY=${your jwt secret key}
S3_ACCESS_KEY=${your s3 access key}
S3_SECRET_KEY=${your s3 secret key}
CLOVA_API_URL=${your naver cloud clova api url}
CLOVA_SECRET=${your naver cloud clova secret key}
BUCKET_NAME=${your s3 bucket name}
apikey=${your open-ai api key}
```

5. Nginx

a. default.conf :

```
server {

        autoindex_localtime on;

        listen 443 ssl;
        ssl_certificate /etc/letsencrypt/live/j9a410.p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/j9a410.p.ssafy.io/privkey.pem;

        server_name j9a410.p.ssafy.io;

        proxy_set_header Host $host;
        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 X-Forwarded-Proto https;
        proxy_headers_hash_bucket_size 512;
        proxy_redirect off;

        # Websockets
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        location / {
                proxy_pass http://localhost:3000;
                proxy_connect_timeout 300s;
                proxy_read_timeout 600s;
                proxy_send_timeout 600s;
                proxy_buffer_size          128k;
                proxy_buffers              4 256k;
                proxy_busy_buffers_size    256k;
                #mixed error
                add_header Content-Security-Policy "upgrade-insecure-requests";
        }

        location /api {
                proxy_pass http://localhost:8081;
        }

        location /ws {
                proxy_pass http://localhost:3000;
```

```
        }

        location /ml {
                proxy_pass http://localhost:8000;
        }

}
server {
        listen 80;
        server_name j9a410.p.ssafy.io;

        return 301 https://$host$request_uri;
}
```

6. Jenkins Pipeline Script

```
pipeline {
    agent any
    tools{
        gradle 'gradle'
        nodejs 'NodeJS 18.16.1'
    }
    environment {
        DOCKER_REPO_BACK = "${dockerhub username}/backend"
        DOCKER_REPO_FRONT = "${dockerhub username}/frontend"
        DOCKER_REPO_ML = "${dockerhub username}/ml"
        DOCKER_CREDENTIAL = credentials('dockerhub-credential')
        BACK_CONTAINER = "pendi-back"
        FRONT_CONTAINER = "pendi-front"
        ML_CONTAINER = "pendi-ml"
        SSH_CONN = "ubuntu@j9a410.p.ssafy.io"
        ENV_DIR = "./config/.env"
        BACK_PORT = "8081"
        FRONT_PORT = "3000"
        ML_PORT = "8000"
    }
    stages{
        stage('Git Clone') {
            steps {
                sh "#### Git Clone Start ####"
                git branch: 'master', credentialsId: 'gitlabID', url: 'https://lab.ssafy.com/s09-fintech-finance-sub2/S09P22A410.g
                sh "#### Git Clone Success ####"

            }
        }
        stage('Build Backend'){
            steps{
                dir('backend/namani'){
                    sh "echo '#### Build Backend Start ####'"
                    sh "chmod +x gradlew"
                    sh "./gradlew clean compileJava bootJar"
                    sh "echo '#### Build Backend Success ####'"

                }

            }
        }

        stage('Build Frontend'){
            steps{
                sh "echo '#### Build Frontend Start ####'"
                dir('frontend'){
                    sh "npm install serve"
                    sh "CI=false npm run build"
                }
                sh "echo '#### Build Frontend Success ####'"
            }
        }


        stage('Build Images'){
            steps{
                sh "echo '#### Build Image Start ####'"
                sh "docker compose build"
                sh "echo '#### Build Image Success ####'"
            }

        }


        stage('Push Images'){
```

```
            steps{
                sh "echo '#### Push Images Start ####'"
                sh "echo $DOCKER_CREDENTIAL_PSW | docker login -u $DOCKER_CREDENTIAL_USR --password-stdin"
                sh "docker push pohangman/backend:latest"
                sh "docker push pohangman/frontend:latest"
                sh "docker push pohangman/ml:latest"
                sh "echo '#### Push Images Success ####'"
            }

        }

        stage('Deploy Backend Server'){
            steps{
                sh "echo '#### Deploy Backend Server Start ####'"
                sshagent(credentials:['ec2-user']){
                    sh "pwd"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker rm -f pendi-back'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker rmi -f $DOCKER_REPO_BACK'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker pull $DOCKER_REPO_BACK'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'echo y | docker image prune'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker images'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker run -d --name pendi-back --env-file $ENV_DIR -p $BACK_PO
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker ps'"
                }
                sh "echo '#### Deploy Backend Server Success ####'"
            }
        }

        stage('Deploy Frontend Server'){
            steps{
                sh "echo '#### Deploy Frontend Server Start ####'"
                sshagent(credentials:['ec2-user']){
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker rm -f pendi-front'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker rmi -f $DOCKER_REPO_FRONT'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker pull $DOCKER_REPO_FRONT'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'echo y | docker image prune'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker images'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker run -d --name pendi-front --env-file $ENV_DIR -p $FRONT_
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker ps'"
                }
                sh "echo '#### Deploy Frontend Server Success ####'"
            }
        }

        stage('Deploy ML Server'){
            steps{
                sh "echo '#### Deploy ML Server Start ####'"
                sshagent(credentials:['ec2-user']){
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker rm -f pendi-ml'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker rmi -f $DOCKER_REPO_ML'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker pull $DOCKER_REPO_ML'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'echo y | docker image prune'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker images'"
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker run -d --name pendi-ml --env-file $ENV_DIR -p $ML_PORT:8
                    sh "ssh -o StrictHostKeyChecking=no $SSH_CONN 'docker ps'"
                }
                sh "echo '#### Deploy ML Server Succcess ####'"
            }
        }
    }
}
```

4. 포트 개방

```
sudo apt-get install ufw

sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https

sudo ufw allow 8080
sudo ufw allow 8080/tcp
sudo ufw allow 3000
sudo ufw allow 3000/tcp
sudo ufw allow 8081
sudo ufw allow 8081/tcp
sudo ufw allow 8000
sudo ufw allow 8000/tcp
sudo ufw allow 80
sudo ufw allow 20
```

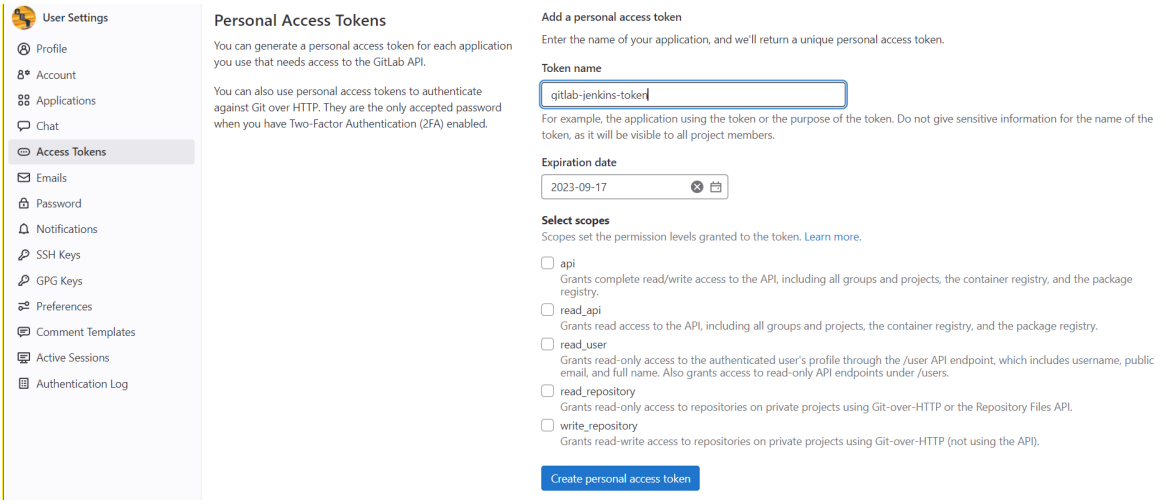
```
sudo ufw allow 80/tcp
sudo ufw allow 22/tcp
sudo ufw allow 443/tcp
```

5. SSL 인증서 발급

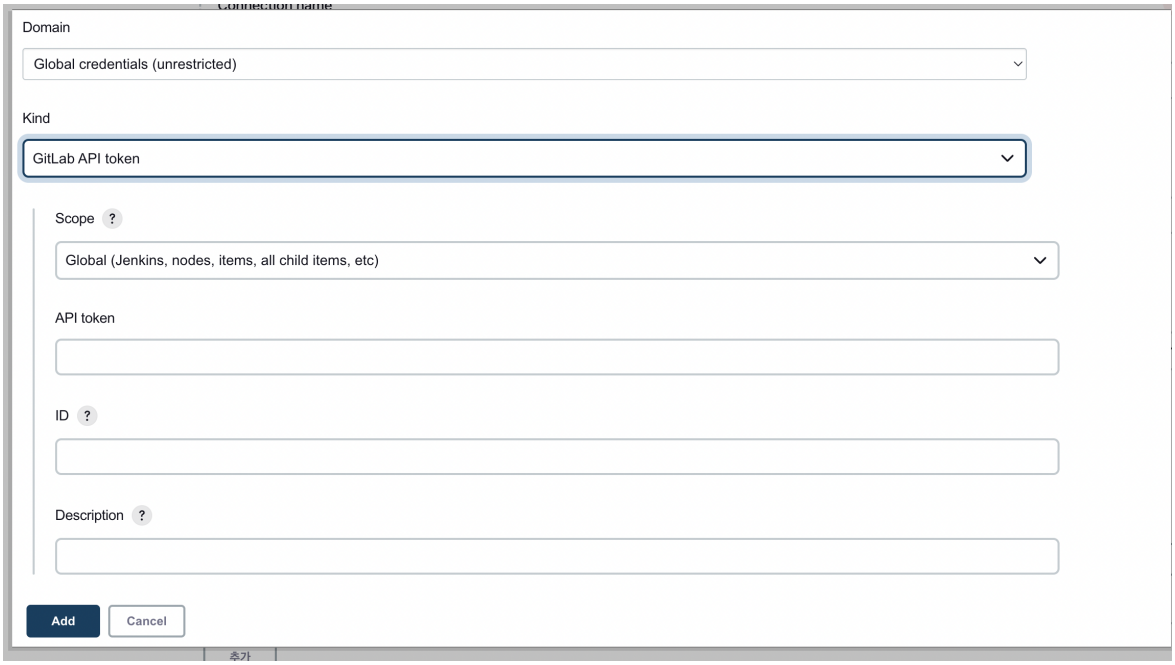
```
sudo apt-get install letsencrypt
sudo apt-get install certbot python3-certbot-nginx
sudo certbot --nginx
# 이메일 입력
# 약관 동의 - Y
# 이메일 발송 동의 - Y or N
# 도메인 입력
```

6. Jenkins에 Gitlab Webhook 연결

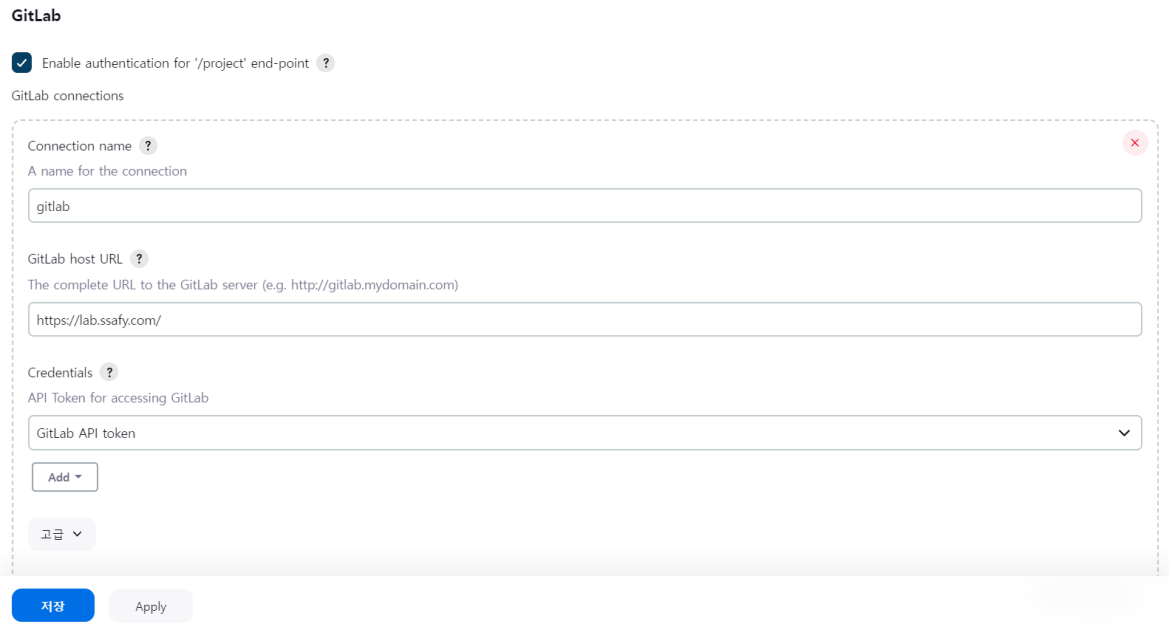
1. Jenkins 대시보드>Jenkins 관리>Plugins
2. Gitlab 플러그인 설치
3. Gitlab API Token 발급



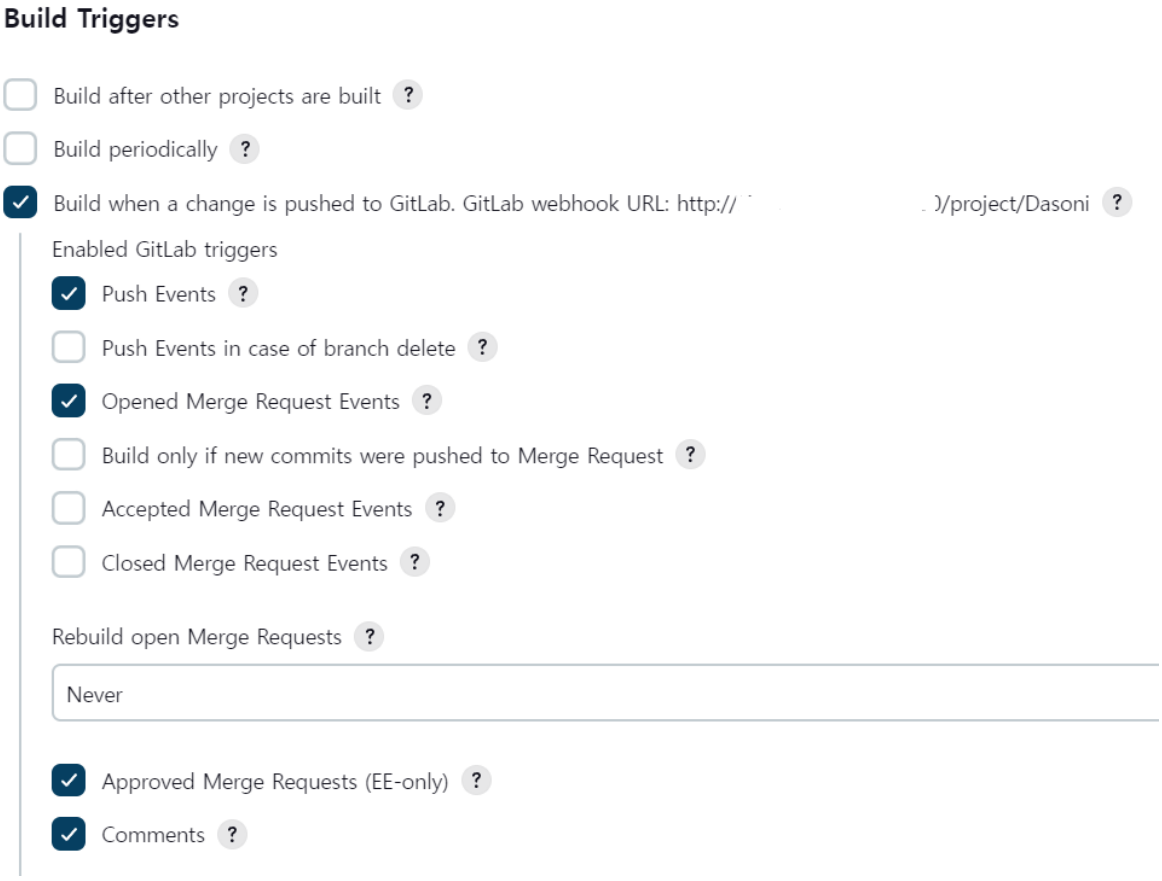
4. 발급 받은 Token을 Jenkins에 등록



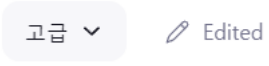
5. Jenkins관리>시스템 설정으로 이동 후, gitlab 경로와 위에서 등록한 API Token Credential을 사용하여 Gitlab과 Jenkins를 연동



6. 파이프라인 프로젝트 생성 및 Webhook 경로 확인
   - CI/CD를 수행할 Pipeline 아이템 생성

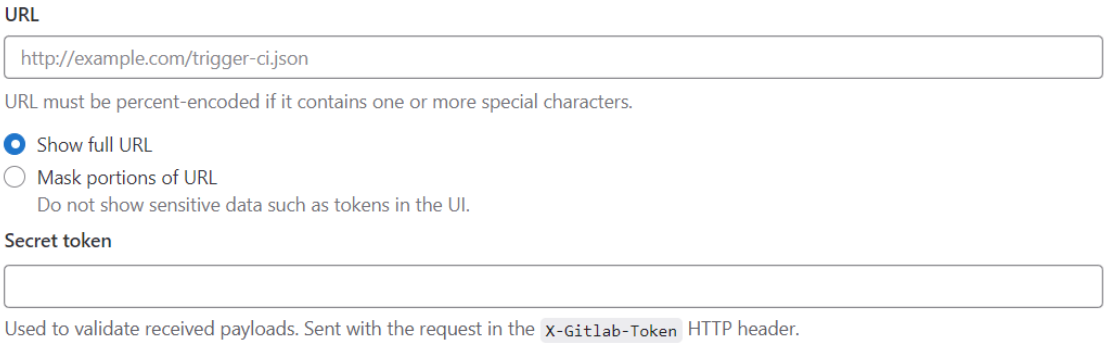


7. Pipeline Secret Token 발급
   - Build Triggers 메뉴 아래 '고급' 탭 클릭

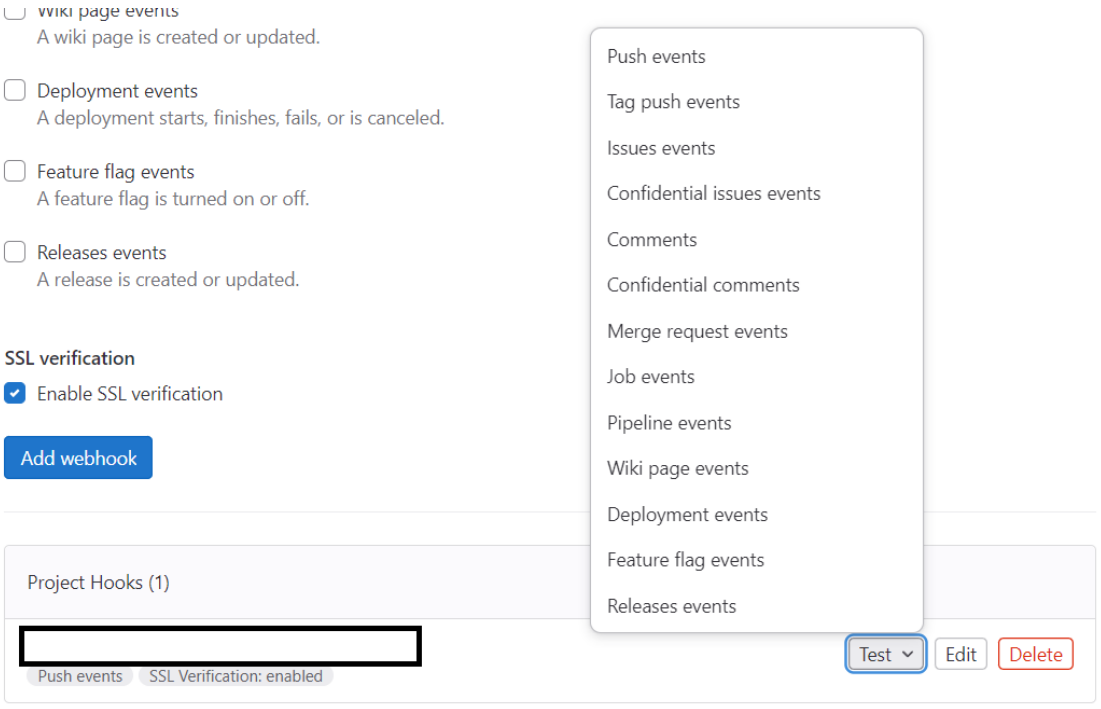


   - Generate를 통해 Secret Token 발급 (꼭, 따로 저장하도록 해야함)

8. Repository Webhook Event 설정
   - Gitlab의 Repository에서 Settings → Webhooks
   - Webhook Event는 Repository 별로 설정해주어야 함
   - 앞서 기억해둔 URL과 Secret을 입력해주고 원하는 Trigger를 체크해준다



9. 등록 확인
   - 등록 후, Test를 통해 연동 확인 테스트를 할 수 있다

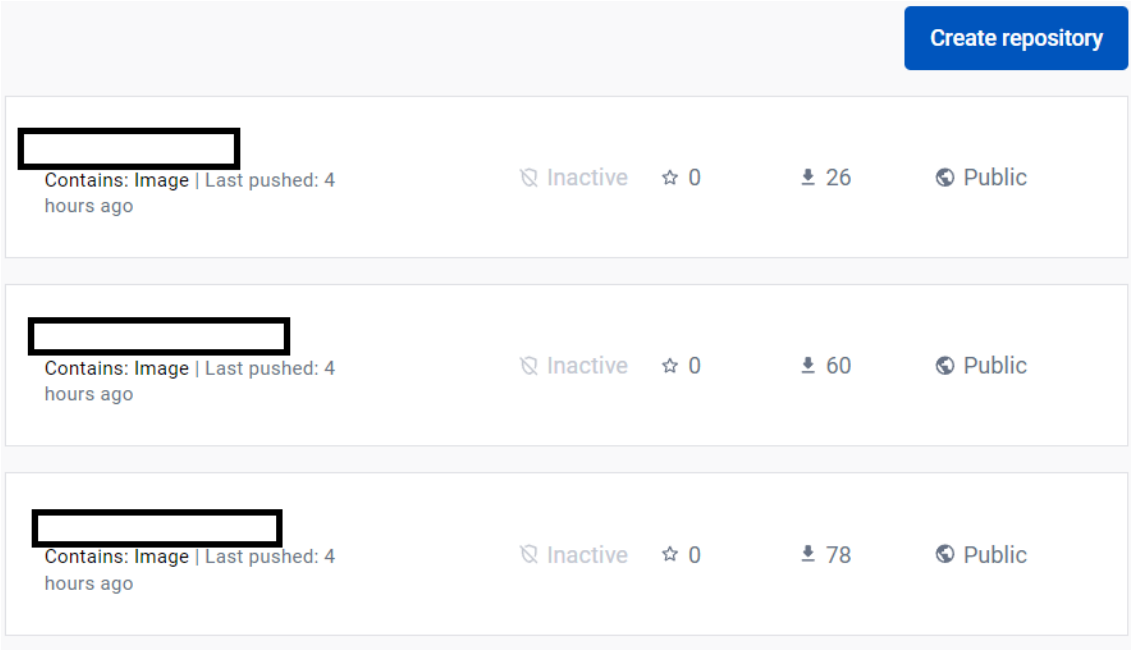


   - Test를 하고 나서 이와 같이 표시되면 연동이 성공 된 것이다

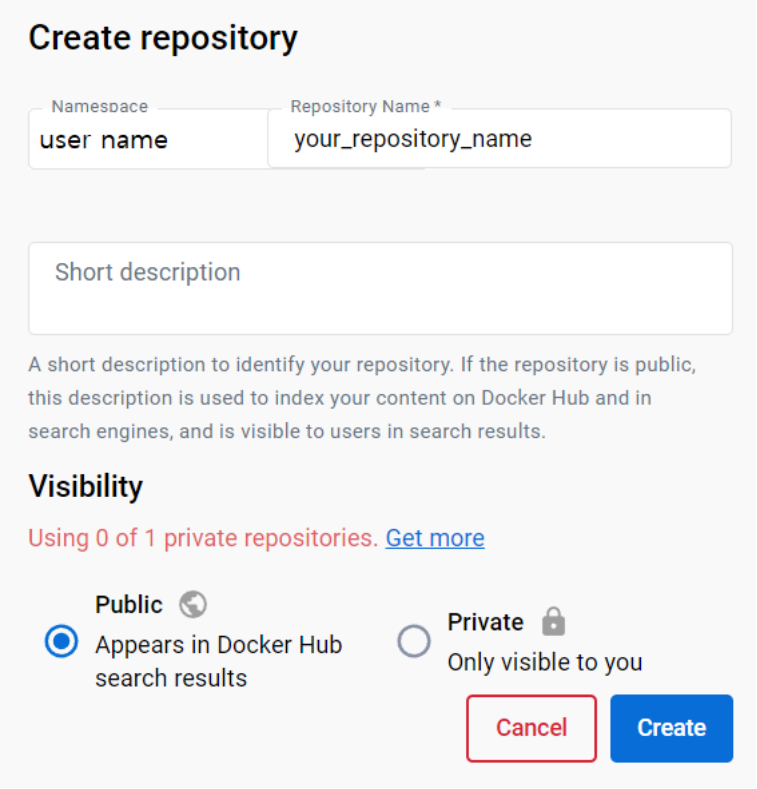
II. 외부 서비스

1. Dockerhub

1. Dockerhub Repository 만들기
   a. Dockerhub 로그인 후, 우측 상단 Create repository 클릭

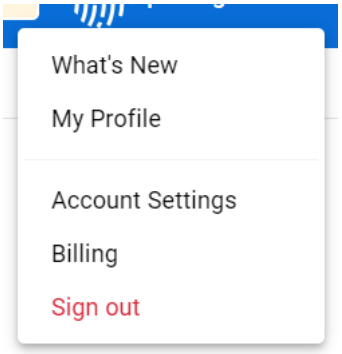


b. Repository Name을 채우고 Create (Private는 Free Tier 에선 하나 밖에 생성하지 못함)

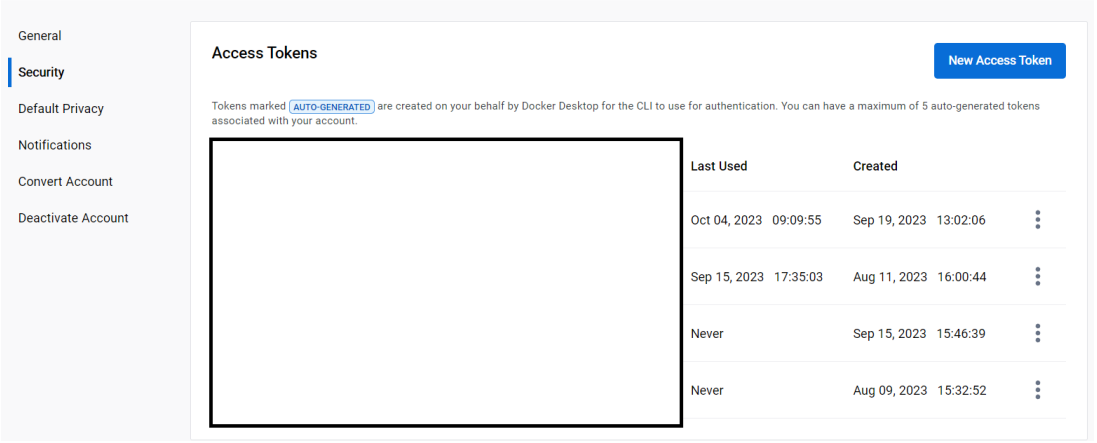


c. Jenkins Pipeline Script에서 Dockerhub를 사용할 수 있도록 Dockerhub Access Token 발급 및 등록

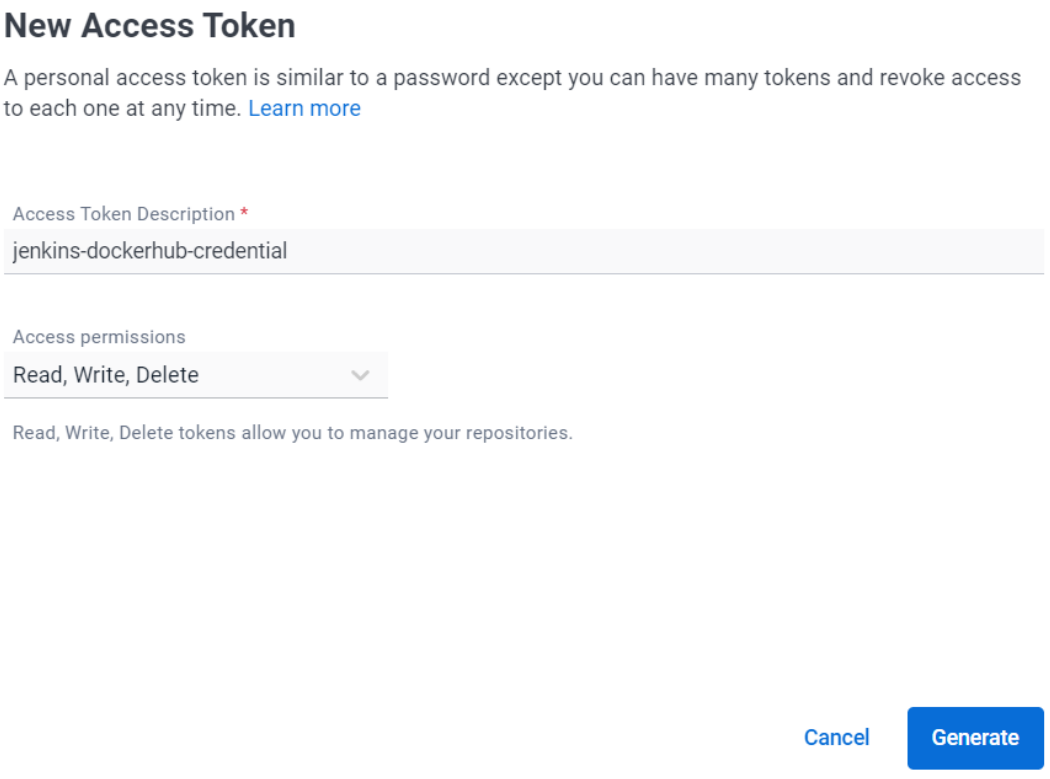
i. Account Settings 클릭



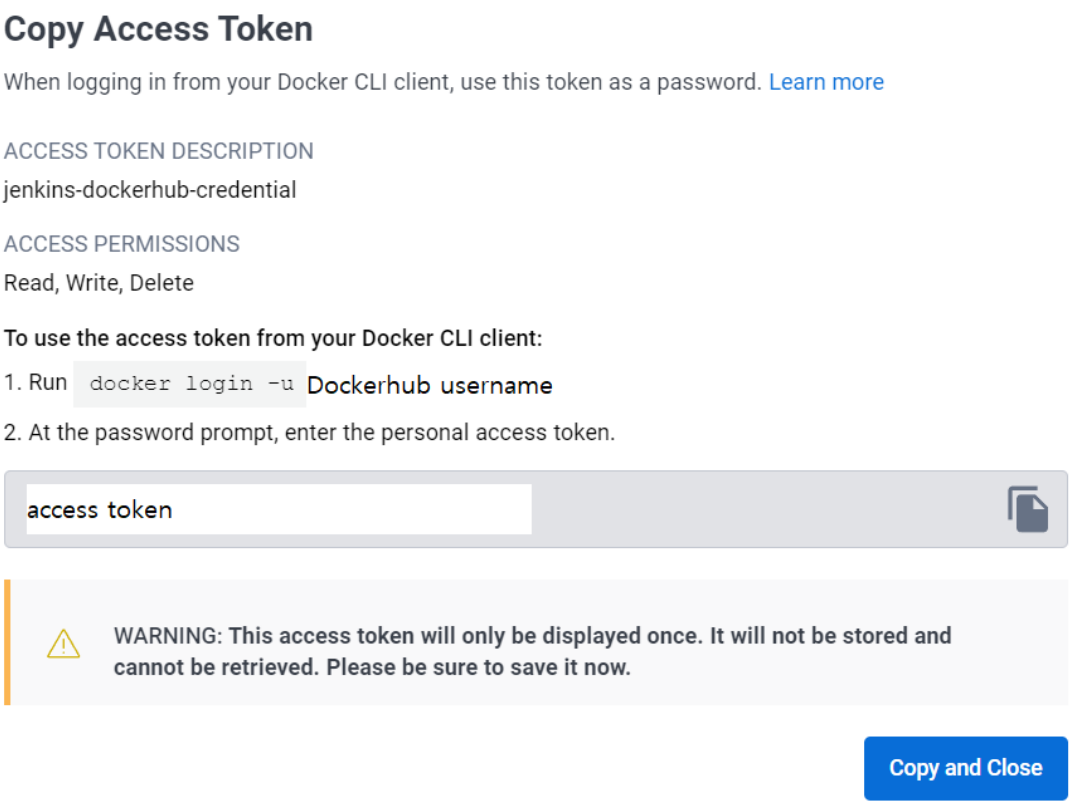
ii. Security 메뉴 클릭 후, New Access Token 버튼 클릭



iii. Access Token Description 작성 후, 부여하고 싶은 액세스 권한 설정 후, Generate 클릭

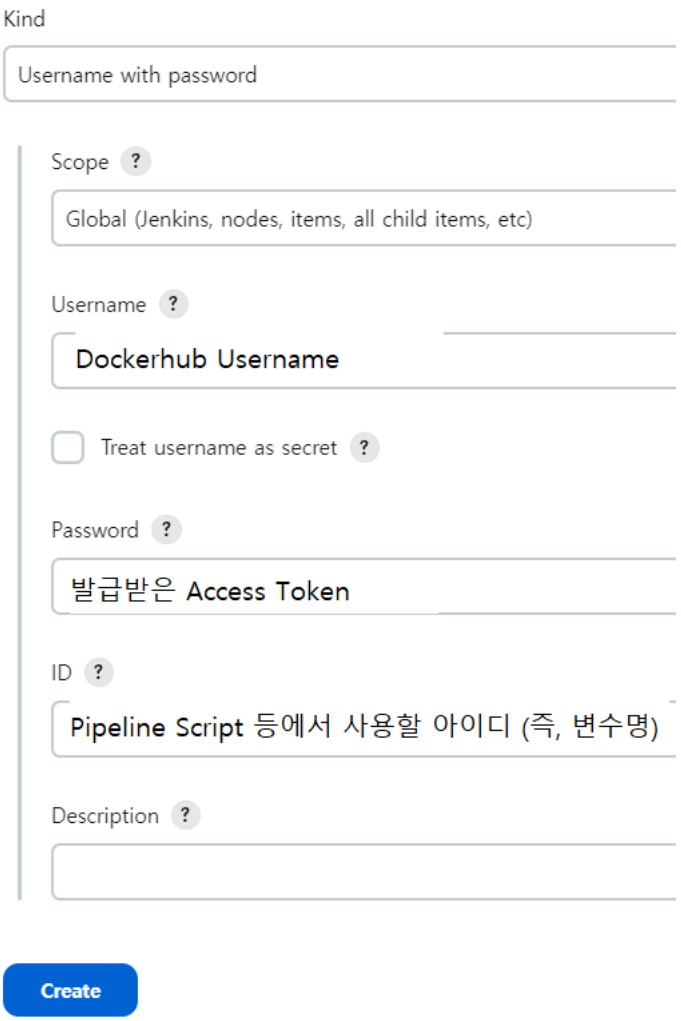


iv. 발급된 Access Token을 꼭 저장해두도록 한다



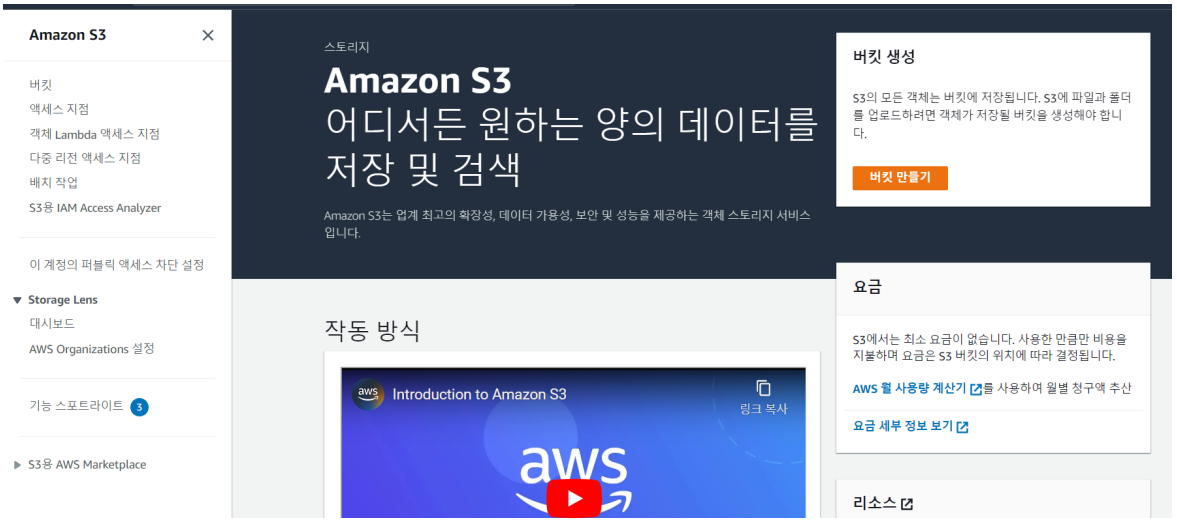
v. Jenkins의 Dashboard>Jenkins관리>Credentials 에서 Add credentials>Username with password 선택

vi. 필요한 내용을 채운 뒤, Create 클릭 후 사용



2. AWS S3

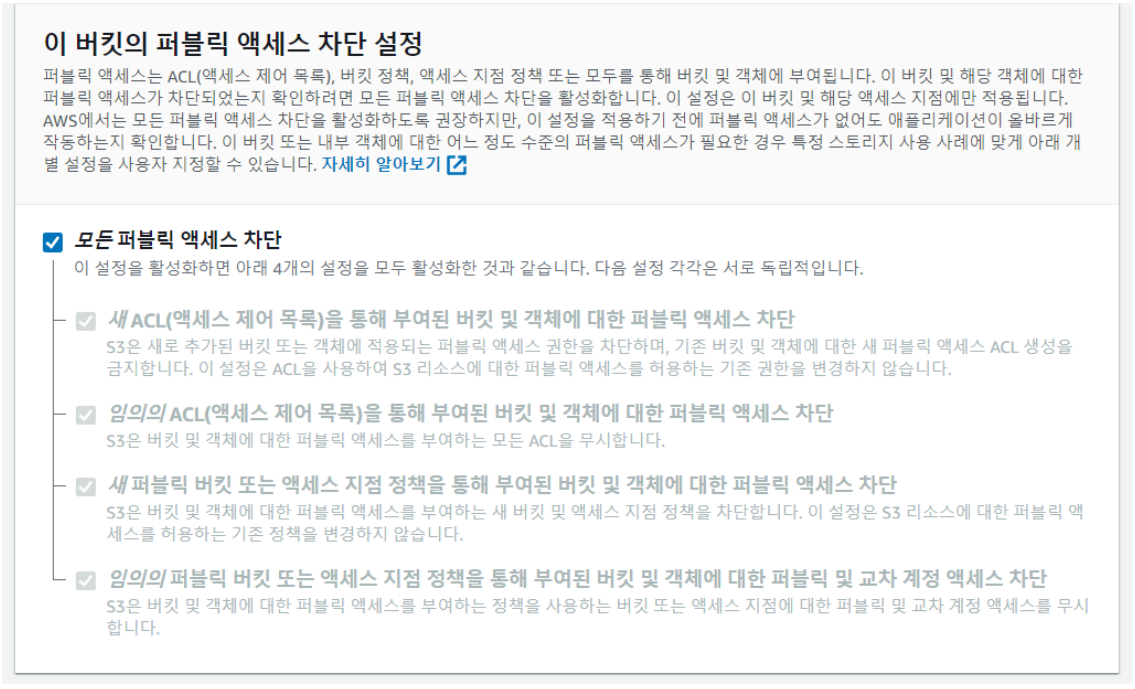
1. `https://aws.amazon.com/ko/` 로 접속
2. 로그인 후, 서비스 창에서 S3 선택
3. 버킷 만들기 클릭



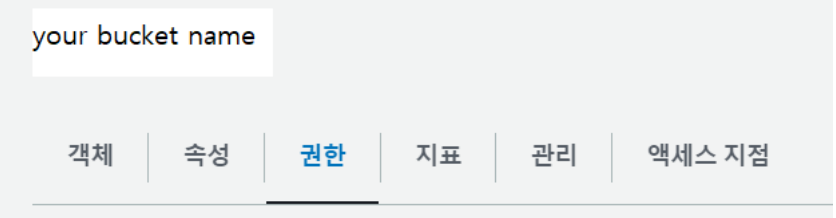
4. 버킷 이름과 리전 설정

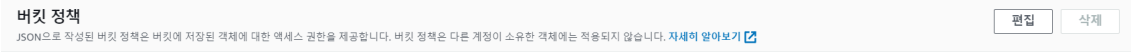
a. 버킷 이름은 고유한 값이어야 함

5. 퍼블릭 엑세스 설정 후 버킷 만들기 클릭
   - 외부에 S3를 공개할 경우 모든 퍼블릭 액세스 차단에 체크를 해제, 공개하지 않는다면 체크

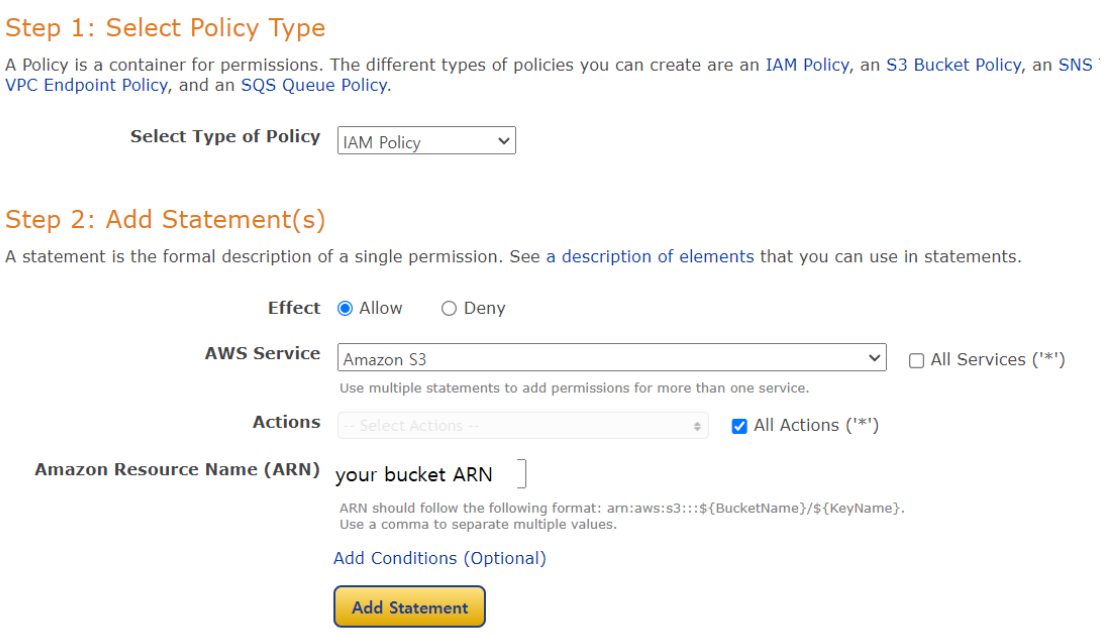


6. 버킷 정책 생성
   a. 생성된 버킷 클릭 → 권한 메뉴 클릭 → 버킷 정책 메뉴의 편집 클릭



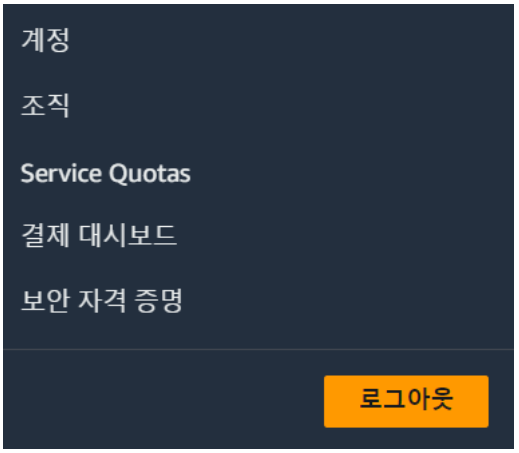


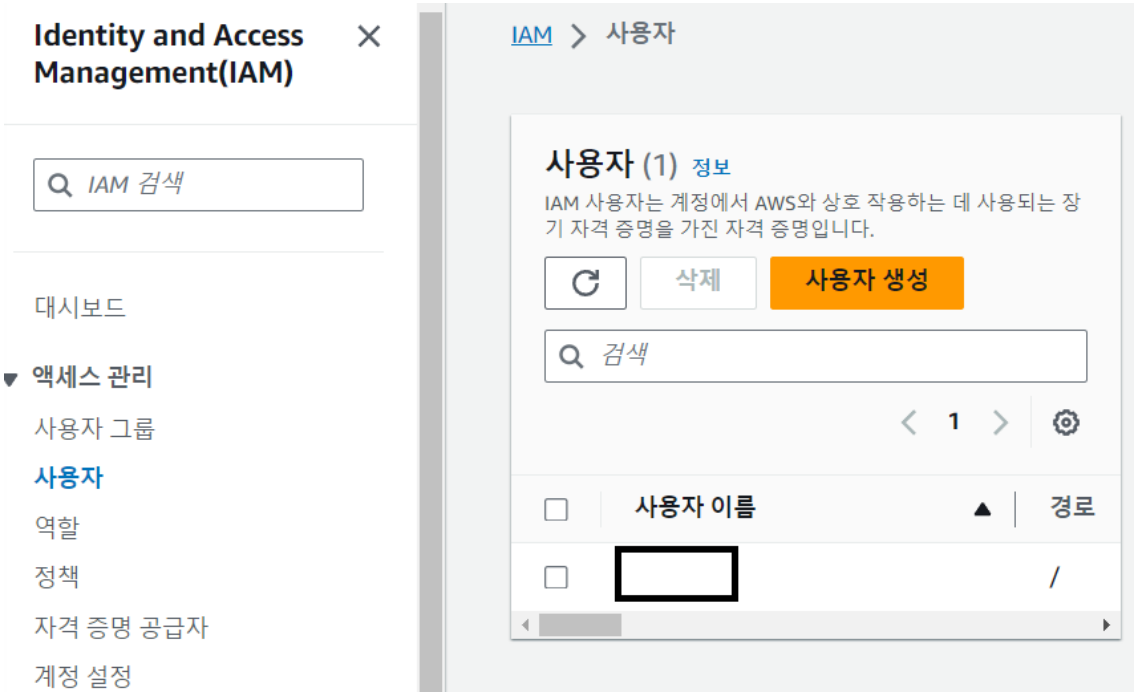
b. 버킷 ARN을 복사한 뒤, AWS 정책 생성기로 접속(http://awspolicygen.s3.amazonaws.com/policygen.html) 하여 정책 타입과 상태를 추가한 뒤, 정책 생성



c. 생성된 정책을 복사 후, 버킷의 정책란에 붙여 넣기 후, 변경 사항 저장

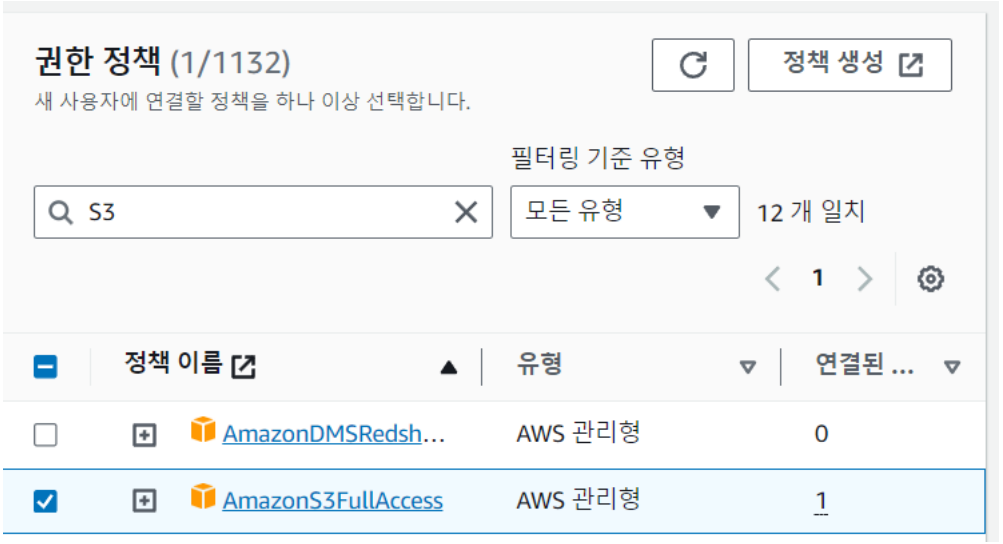
7. Spring Boot 서버에서 S3를 사용할 수 있도록, IAM에서 사용자 등록
   a. 보안 자격 증명 → 액세스 관리 하위 사용자 → 사용자 생성





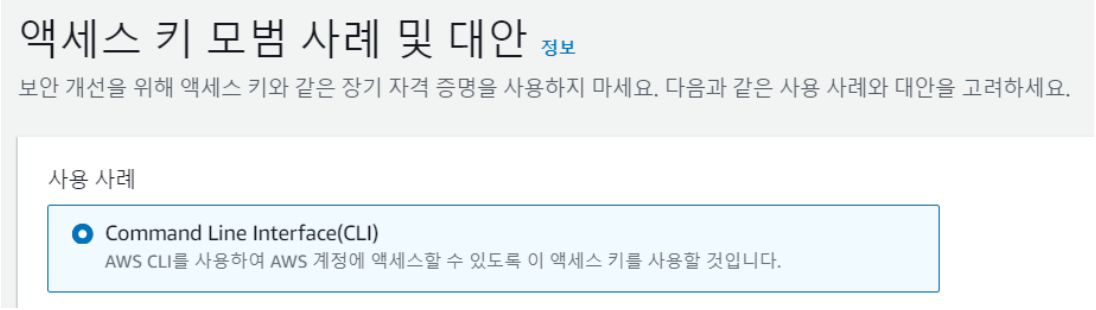
b. 사용자 이름 작성 후 다음

c. 권한 설정에서 직접 정책 연결 선택 후, AmazonS3FullAccess 검색 후 선택 → 다음 클릭 → 사용자 생성 클릭



d. 생성한 사용자 클릭 → 보안 자격 증명 → 액세스 키 → 액세스 키 만들기

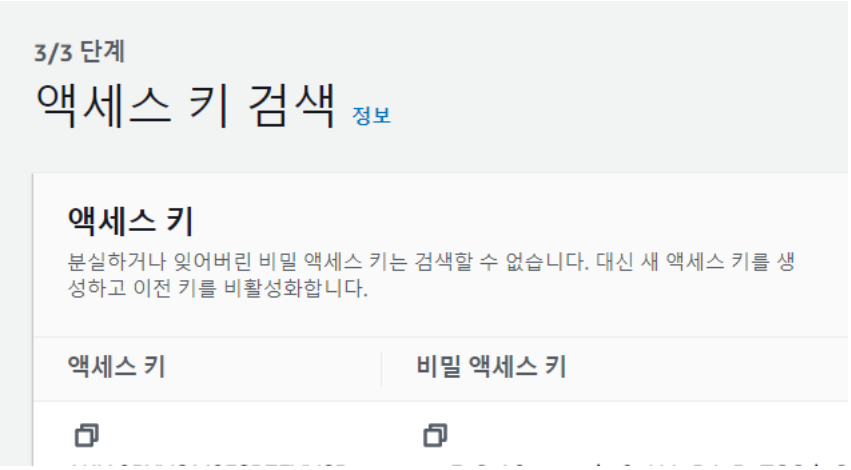
i. CLI 선택 후 다음



ii. 설명 태그 값 입력 후 액세스 키 생성

iii. 액세스 키와 비밀 액세스 키 반드시 다른 곳에 저장

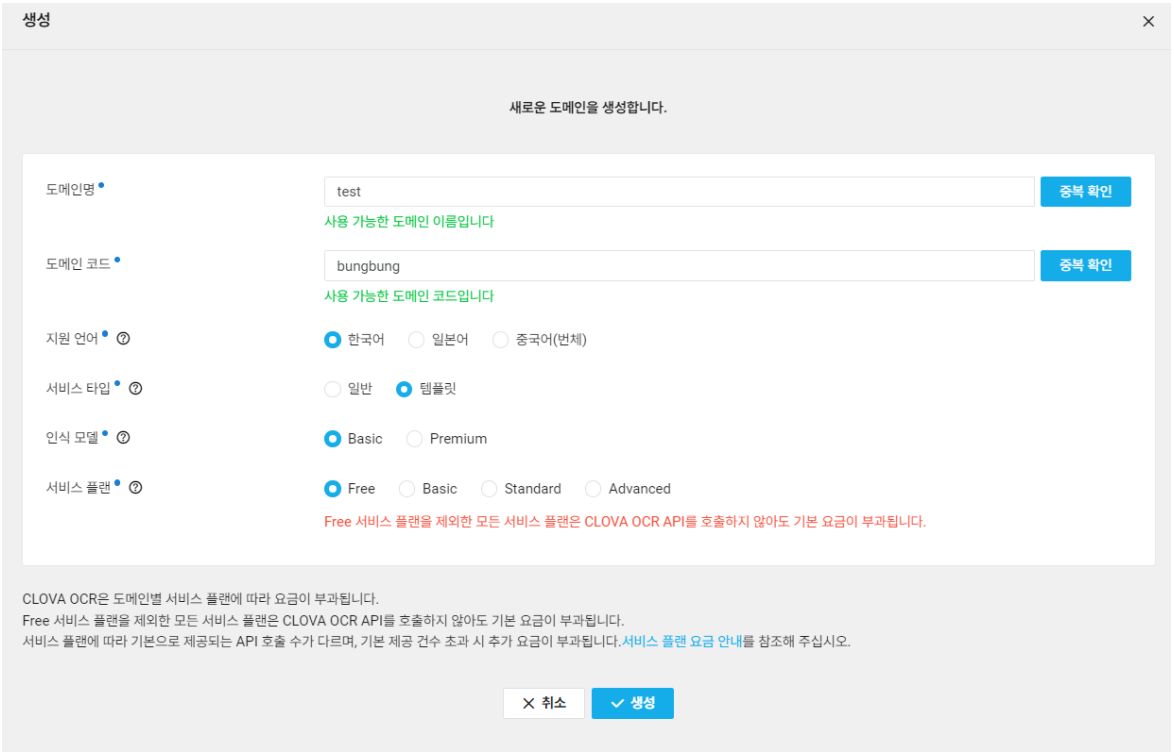
- 액세스 키는 다시 확인 가능하지만 비밀 액세스 키는 다시 확인하기 힘듬



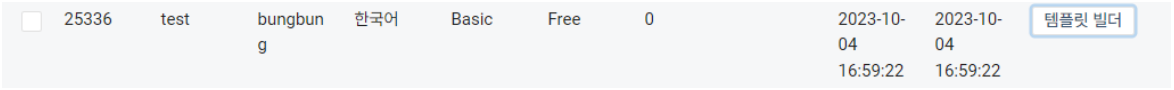
e. 발급받은 액세스 키와 비밀 액세스 키를 사용

3. Naver Cloud Clova OCR

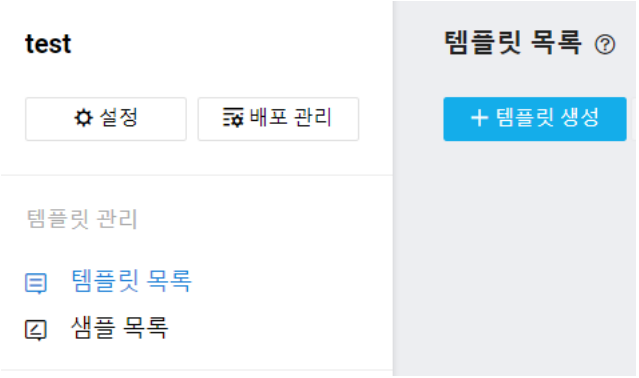
1. Naver Cloud Platform Console 접속
2. 도메인 생성 → 고유의 도메인 명과 도메인 코드 입력



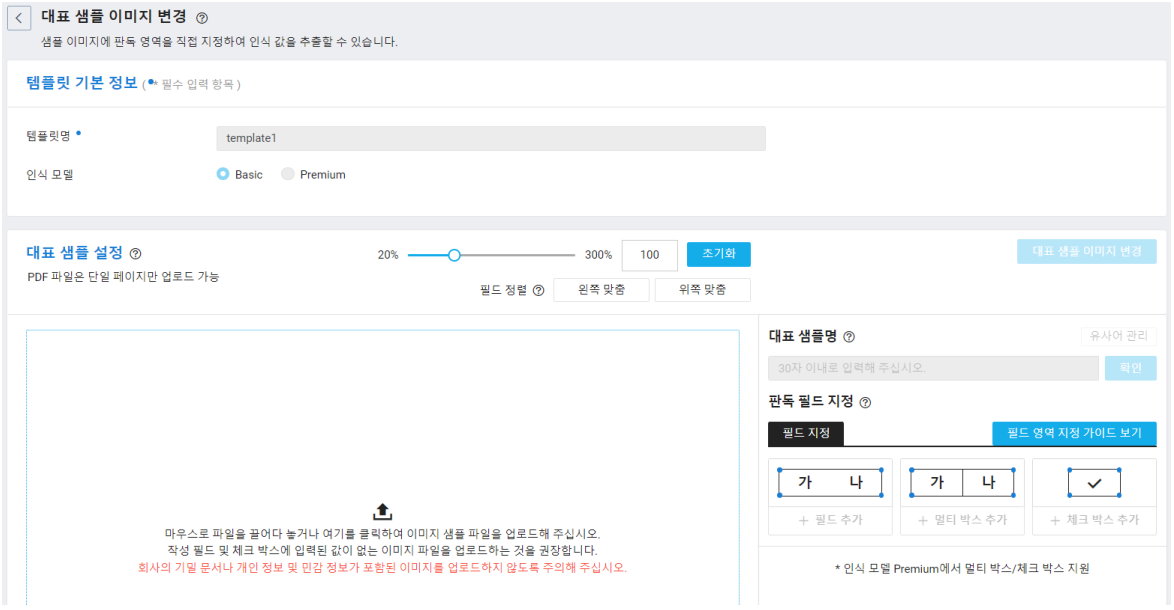
3. 생성 후, 콘솔에서 생성한 도메인의 템플릿 빌더 클릭



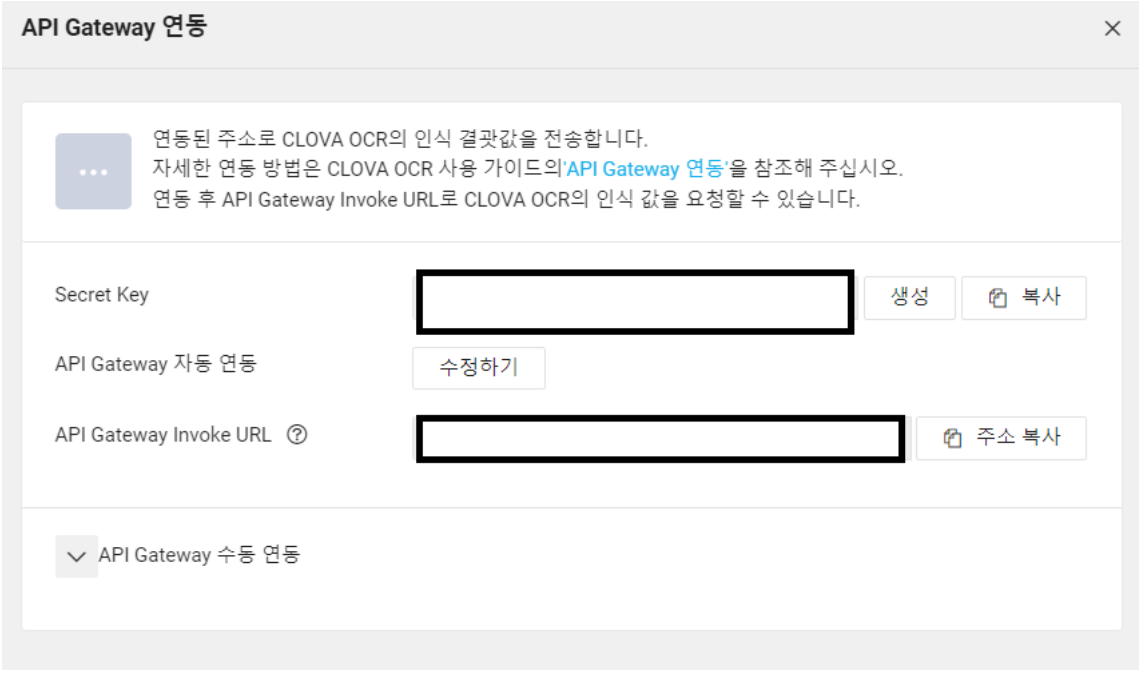
4. 템플릿 목록 → 템플릿 생성



5. 템플릿 명을 입력 → 대표 이미지 등록 → 판독 필드 지정 → 저장



6. 설정 → API Gateway 연동 → Secret Key 복사 → API 게이트웨이 자동 연동 → API URL 복사
   - 창을 닫더라도 다시 확인할 수 있음

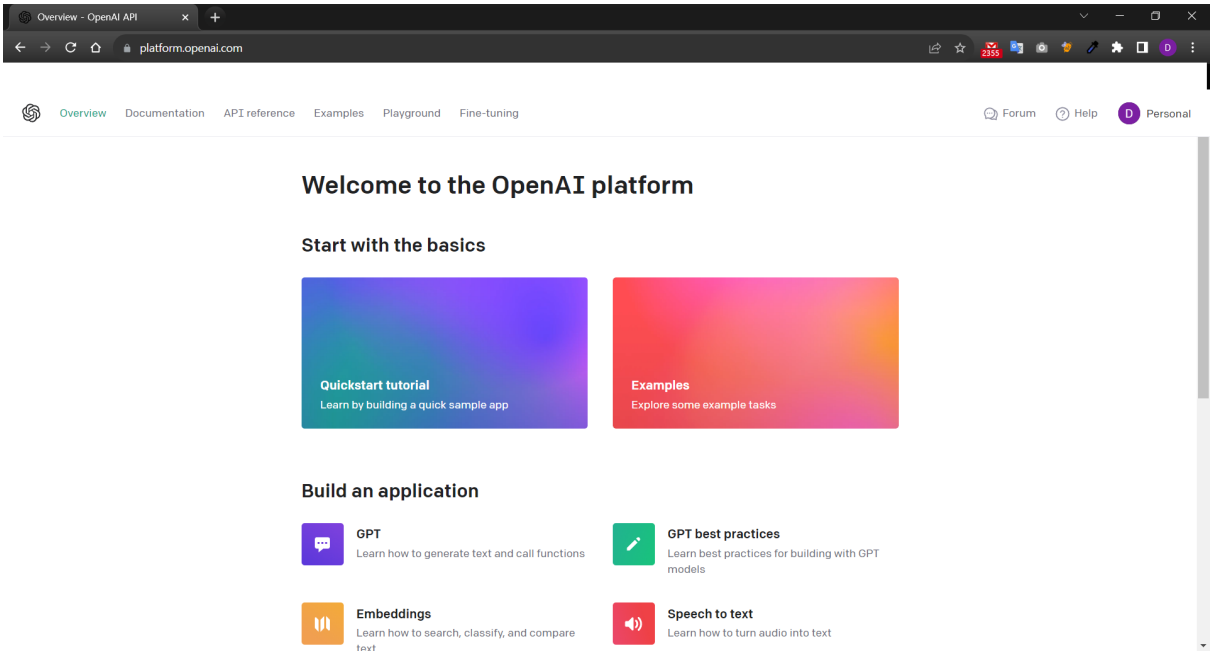


7. 베타 배포 후 테스트를 거쳐 테스트가 제대로 된다면 서비스 배포 후 사용

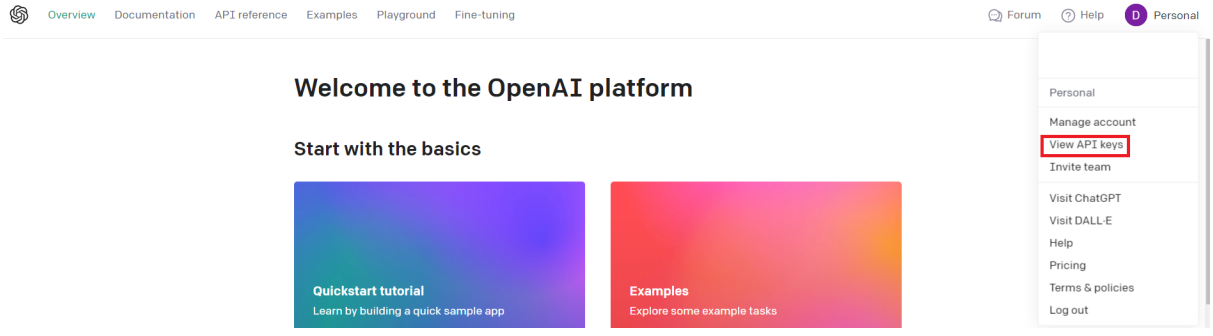
8. Request와 Response의 양식을 공식문서를 통해 참조
    - https://api.ncloud-docs.com/docs/ai-application-service-ocr-ocr

4. OpenAI ChatGPT

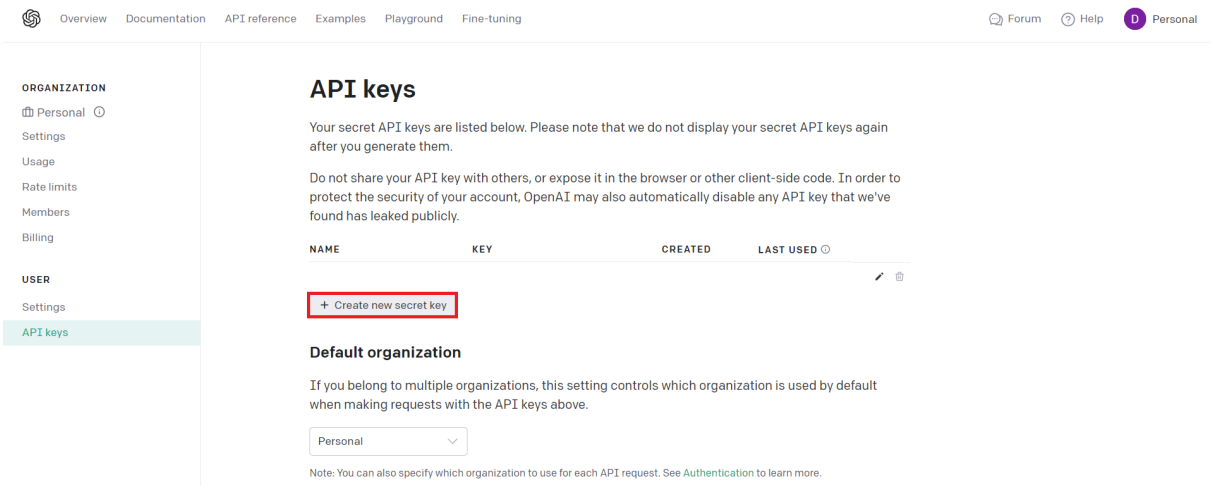
1. platform.openai.com 접속



2. **View API keys 클릭**



3. **Create new secret key**



4. ML디렉토리에 .env 파일 생성 후 'apykey' 변수 초기화