



CI/CD 포팅 매뉴얼

☰ Tags

1. 개발 환경
2. 설정파일 및 환경 변수 정보
3. SSL 인증서 발급 및 NGINX 설정
4. Docker 설치 관련 파일
6. mysql 설치(EC2)
7. Jenkins 설치
8. AWS S3
9. 배포
10. 사용 포트 목록

1. 개발 환경

- Server : 20.04.6 LTS (GNU/Linux 5.15.0-1051-aws x86_64)
- JDK : OpenJDK17
- Node.js : 20.11.0 LTS
- Nginx : 1.24.0
- MySQL : 8.0.36
- Jenkins : 2.426.2

2. 설정파일 및 환경 변수 정보

Spring

```
# Port
server:
  port: 8081

spring:
  # MySQL
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: ENC(f2wkijzdrQnxzxCZYH+lniBL7ndYX2J5EeEU0AUPmwthb0h6lc:
    username: ENC(E0lfe4WYsqJbH0JuHickcw==)
    password: ENC(chxyvNmguj+G1w3wCHN4TA==)

  # redis
  data:
    redis:
      host: ENC(9hP63pD3WAIIs+4sg6muJbuKU/Ie7BdR14pFduMfzpl8=)
      port: 6379
      password: ENC(kqyVDX/z7enC3fMN/f9twJnibSCyfdA0)

  # jpa
  jpa:
    database-platform: org.hibernate.dialect.MySQL57Dialect
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          format-sql: true

  # request multipart file
  servlet:
    multipart:
```

```

    max-file-size: 100MB
    max-request-size: 100MB

# S3
cloud:
  aws:
    s3:
      bucket: ENC(5+puSTEJWGaV9iB5TF79dT/1eV0HSqunB/EsigmI+14=)
      credentials:
        access-key: ENC(ipZe9KTXii9++3h2vSGstpVMeYK4PL6mLFB+0h+
        secret-key: ENC(u0Xn54tQBSwNr+uPjA+6YS+wV+q2/CkabsM6BwTl
      region: ap-northeast-2
    stack:
      auto: false # EC2의 Spring Cloud 자동 구성 해제

jasypt:
  encryptor:
    bean: jasyptStringEncryptor

DEFAULT_PROFILE_URL: ENC(jgqAgtjThvOP37bgwQv9hF+pBgojgdPM1/N7rCi

---

spring:
  data:
    mongodb:
      host: ENC(c/ECBd/OfzjAMwMHNkjajVsEViEIZvuZLOKKuZLrn18=)
      port: 27017
      database: ENC(ph20AB5KkD9X1/UJo1VMDBf8syZGBhIV)
      authentication-database: admin
      username: ENC(5uXWeieZVQXaVxHIhSzKhA==)
      password: ENC(6l+429SST+jR2bowaARaG6F2dFLzVjA0)

---

spring:

```

```
main:
  allow-bean-definition-overriding: true
```

```
---
```

Vue

```
VITE_API_BASE_URL="http://j10d111a.p.ssafy.io:8081"
```

3. SSL 인증서 발급 및 NGINX 설정

```
sudo apt-get install letsencrypt
sudo letsencrypt certonly --standalone -d <도메인>
```

```
upstream app {
  server backend:8081; # WAS 컨테이너의 이름
}

server {
  listen 80;
  listen 443 ssl;
  server_name i10d204.p.ssafy.io;

  root /usr/share/nginx/html; # 정적 파일이 위치한 디렉토리 지정

  location / {
    try_files $uri $uri/ /index.html; # 정적 파일 제공
  }

  location /api {
```

```

    proxy_pass http://app; # API 요청을 WAS로 전달
}

location /socket {
    proxy_pass http://app/socket; # 웹소켓 요청을 WAS로 전달
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_
}

location /swagger-ui {
    proxy_pass http://app/swagger-ui; # Swagger UI를 위한 프
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_
    proxy_set_header X-Forwarded-Proto $scheme;
}

# SSL 설정
ssl_certificate /etc/letsencrypt/live/i10d204.p.ssafy.io/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/i10d204.p.ssafy.io/privatekey.pem;

# HTTP 요청을 HTTPS로 리다이렉트
# if ($scheme = http) {
#     return 301 https://$server_name$request_uri;
# }
}

```

4. Docker 설치 관련 파일

docker 설치

```
sudo apt update
```

```
sudo apt install -y apt-transport-https ca-certificates curl  
software-properties-common
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sud  
o gpg --dearmor -o /usr/share/keyrings/docker-archive-keyrin  
g.gpg
```

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-ar  
chive-keyring.gpg] https://download.docker.com/linux/ubuntu  
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.  
d/docker.list > /dev/null
```

```
sudo apt update  
sudo apt install docker-ce
```

```
sudo systemctl start docker
```

```
sudo systemctl enable docker
```

```
docker --version
```

docker-compose 설치

```
sudo curl -L "https://github.com/docker/compose/releases/latest,
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

```
sudo docker-compose --version
```

dood(docker-out-of-docker)

원래 컨테이너 없애고 docker.sock 볼륨을 추가해서 다시 생성해야 한다. 도커 볼륨에 이전 정보들이 다 저장되어있어서 볼륨 삭제 안하고 컨테이너 생성할 때 그대로 쓰면 됨

도커 볼륨 : `/home/ubuntu/jenkins-data`

```
sudo docker rm -f jenkins
```

```
sudo docker run -d -p 8080:8080 \  
-v /home/ubuntu/jenkins-data:/var/jenkins_home \  
-v /var/run/docker.sock:/var/run/docker.sock \  
--name jenkins jenkins/jenkins:lts-jdk17
```

```
sudo docker exec -u 0 -it jenkins bash
```

```
``` jenkins container bash ```
```

```
apt-get update && \
apt-get -y install apt-transport-https \
ca-certificates \
curl \
gnupg2 \
software-properties-common && \
curl -fsSL https://download.docker.com/linux/debian/ \
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable" && \
apt-get update && \
apt-get -y install docker-ce-cli
```

```
``` jenkins container bash ```
```

```
docker --version
```

```
ls -l /var/run/docker.sock
```

```
# chmod 666 /var/run/docker.sock
```

6. mysql 설치(EC2)

```
sudo apt update  
sudo apt install mysql-server
```

```
sudo ufw allow mysql
```

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

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';  
  
FLUSH PRIVILEGES;
```

mysql 보안 설정

```
sudo mysql_secure_installation
```

1. 비밀번호 난이도를 설정할 것인지 : ☒ y

2. root 계정의 비밀번호를 수정할 것인지 : ☐
3. 익명 사용자를 제거할 것인지 : ☐
4. 원격으로 root의 mysql접속을 허용하지 않을 것인지 : ☐
5. Test 데이터베이스를 삭제할 것인지 : ☐
6. 권한테이블을 reload할 것인지 : ☐

외부 접속 허용

```
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

```
# bind-address            = 127.0.0.1
bind-address              = 0.0.0.0
```

```
sudo systemctl stop mysql
sudo systemctl start mysql
```

새로운 계정 생성

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

```
CREATE USER 'new_user'@'localhost' IDENTIFIED BY 'user_password'
```

```
GRANT ALL PRIVILEGES ON *.* TO 'new_user'@'localhost' WITH GRANT OPTION
```

```
FLUSH PRIVILEGES;
```

```
UPDATE mysql.user SET host='%' WHERE user='mefi123' AND host='localhost'
```

7. Jenkins 설치

jenkins container 생성 및 구동

```
cd /home/ubuntu && mkdir jenkins-data
```

```
sudo ufw allow 8080/tcp  
sudo ufw reload  
sudo ufw status
```

```
sudo docker run -d -p 8080:8080 -v /home/ubuntu/jenkins-data:/var/jenkins_home  
--name jenkins jenkins/jenkins:lts-jdk17
```

```
sudo docker logs jenkins
```

```
sudo docker stop jenkins  
sudo docker ps -a
```

환경 설정 변경 ★매우 중요★

```
cd /home/ubuntu/jenkins-data
```

```
mkdir update-center-rootCAs  
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCAs
```

```
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://updates.jenkins.io/update-center.json#'
```

```
sudo docker start jenkins
```

```
# 컨테이너 구동
```

```
sudo docker start jenkins
```

```
# 컨테이너 종료
```

```
sudo docker stop jenkins
```

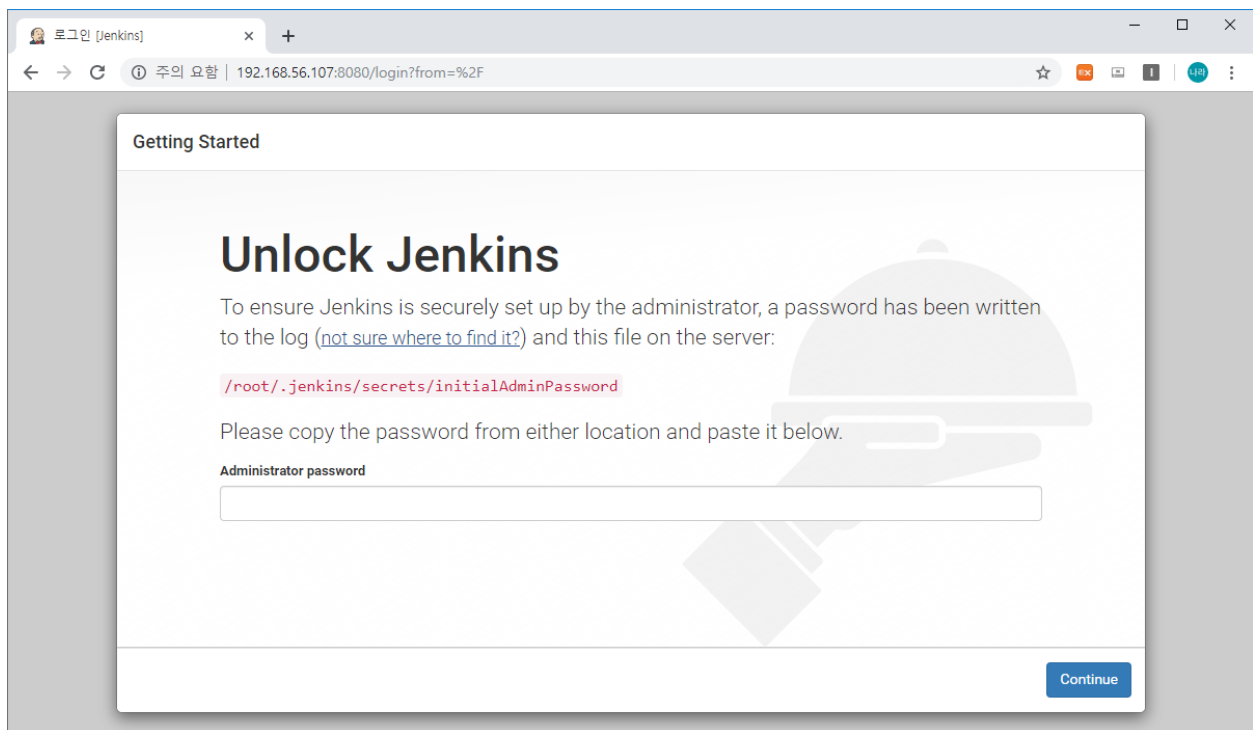
```
# 컨테이너 로그 확인
```

```
sudo docker logs jenkins
```

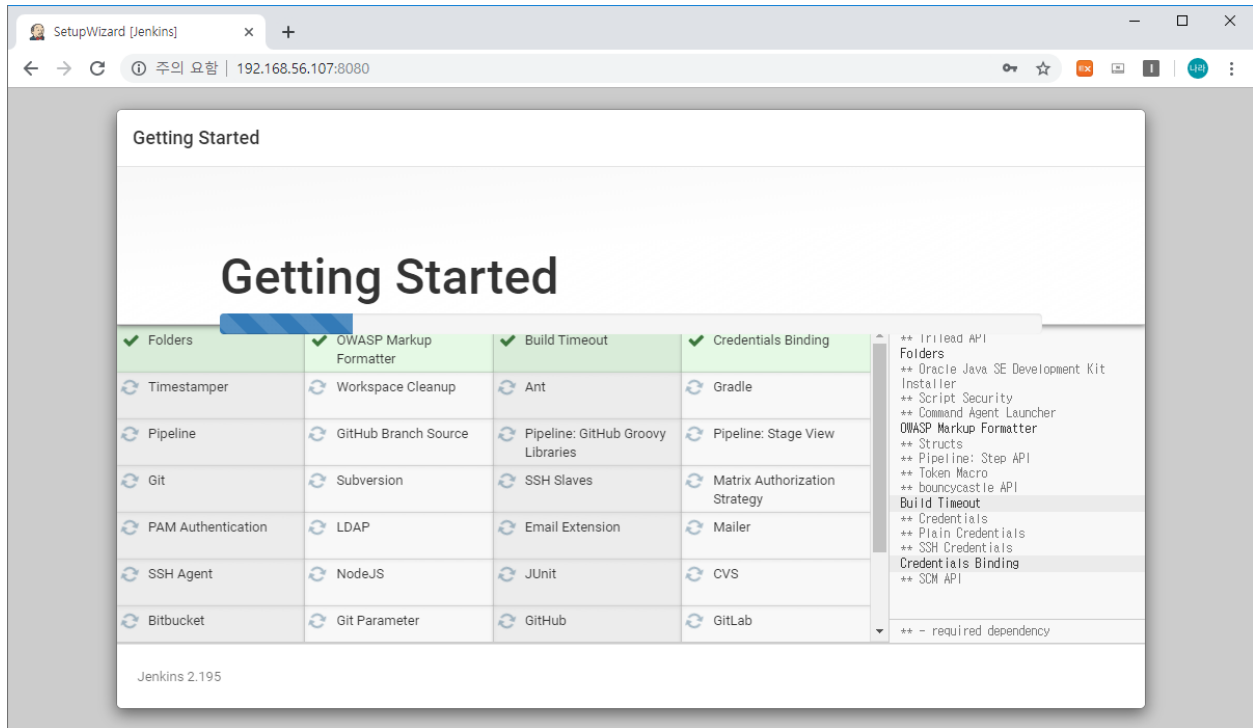
```
# 컨테이너 로그 실시간 확인(확인 중 Ctrl + C를 누르면 종료)
```

```
sudo docker logs -f jenkins
```

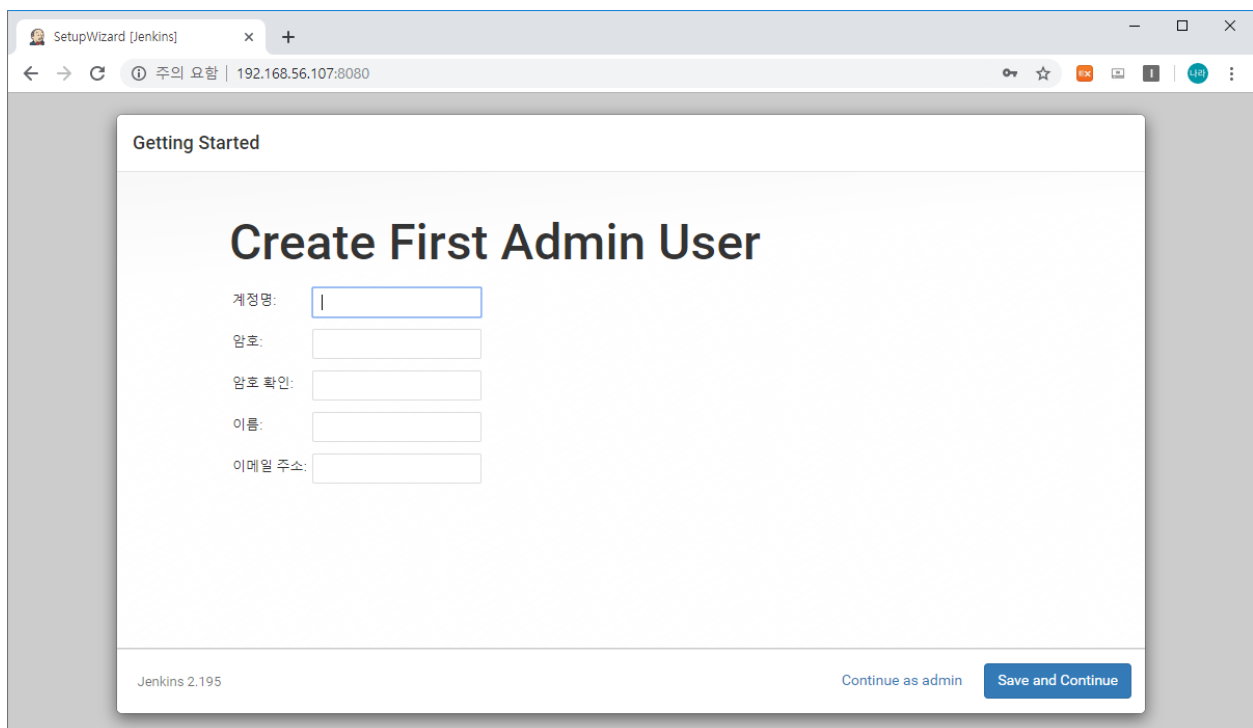
jenkins 초기 설정 진행



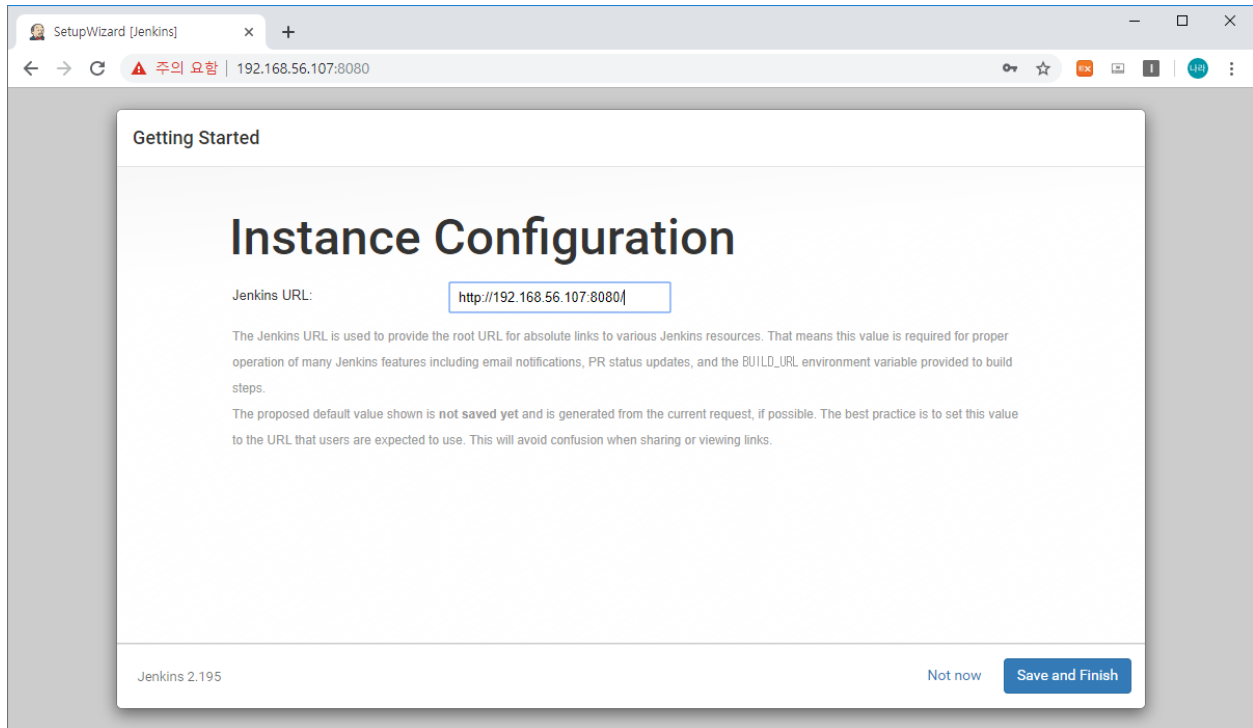
`http://<EC2 도메인네임>:8080` 으로 접속



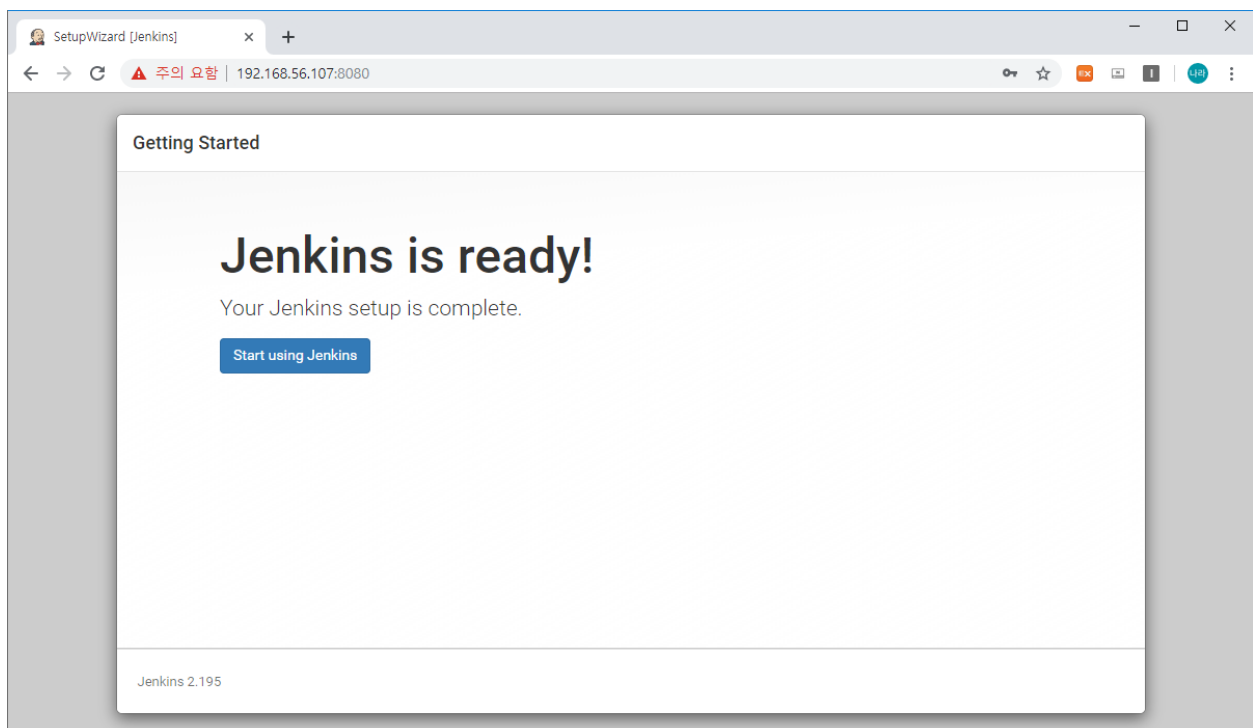
권장 플러그인 설치(만약 플러그인이 자동적으로 다 설치가 안되면 각각 설치 필요)



Admin User 생성



Jenkins Web UI에 접근하기 위한 Jenkins URL 입력(자동 입력됨)



초기설정 완료

8. AWS S3

IAM 계정

S3를 버킷을 사용하기 위해 IAM 계정을 생성한다. 이때 IAM 계정은 AWS 계정에 대한 공유 액세스,

Amazon EC2에서 실행되는 애플리케이션을 위한 보안 AWS 리소스 액세스를 지원한다.

IAM 계정 생성

STEP1) 사용자 이름 입력

STEP2) 권한 정책으로 AmazonS3FullAccess 선택하여 S3에 대한 모든 권한 소유

STEP3) 사용자 상세 - 보안 자격 증명 - 액세스 키 발급

※ 액세스 키는 다시 조회할 수 없으므로 반드시 저장하여 관리한다

S3 Bucket 생성

- Step1) AWS 리전 및 버킷 이름을 입력 (이때 리전에 관계없이 고유한 이름을 사용한다)
- Step2) 객체 소유권 설정 : 내 AWS 계정만 버킷에 접근하거나 사용할 수 있도록 설정한다
- Step3) 퍼블릭 액세스 차단 설정
 - 퍼블릭 액세스는 ACL, 버킷 정책, 액세스 지점 정책을 통해 버킷 및 객체에 부여된다
 - 액세스 차단을 활성화 하면 자원을 안전하게 보호할 수 있지만, 외부 접근이 불가능하다
 - 버킷에 저장된 파일에 접근하지 못하는 현상을 방지하고자, 모두 해제하고 진행한다
- Step4) 기본 암호화 : Amazon S3 관리형 키(SSE-S3)를 사용한 서버 측 암호화

버킷 정책

- 버킷 상세 - 권한 - 버킷 정책 : 객체 URL 접속 시 정상적으로 조회되는 것을 확인할 수 있다

```
// 모든 사용자에게 S3에 대한 DELETE, GET, PUT 권한 제공
{
  "Version": "2012-10-17",
  "Id": "Policy1705987522711",
  "Statement": [
    {
      "Sid": "Stmt1705987520867",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "{버킷 ARN}/*"
    }
  ]
}
```

9. 배포

docker-compose.yml

```
version: '3.8'

services:
  redis:
    image: redis:latest
    command: redis-server --appendonly yes --requirepass sample
```

```

ports:
  - "6379:6379"
volumes:
  - /home/ubuntu/db/redis:/var/lib/redis
mysql:
  image: mysql:latest
  environment:
    MYSQL_ROOT_PASSWORD: sample
    MYSQL_DATABASE: sample
  ports:
    - "3306:3306"
  volumes:
    - /home/ubuntu/db/mysql:/var/lib/mysql
mongo:
  image: mongo:latest
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin
    MONGO_INITDB_ROOT_PASSWORD: sample
  ports:
    - "27017:27017"
  volumes:
    - /home/ubuntu/db/mongo:/var/lib/mongo
backend:
  image: backend:latest
  ports:
    - "8080:8081"
  environment:
    - SPRING_PROFILES_ACTIVE=production
  depends_on:
    - mysql
    - redis
    - mongo

```

Dockerfile


```
FROM eclipse-temurin:17-jdk-alpine as builder
VOLUME /tmp
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java","-jar","/app.jar"]
```

```
FROM nginx:alpine
RUN rm -rf /etc/nginx/conf.d/default.conf
COPY /nginx/nginx.conf /etc/nginx/conf.d

RUN rm -rf /usr/share/nginx/html/*
COPY /dist /usr/share/nginx/html

EXPOSE 80
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Jenkins pipeline script

```
node {

    stage('Clone') {
        git branch: 'Dev', credentialsId: 'jenkins', url: 'http:

        sh "sed -i 's/abc/mefi/g' docker-compose.yml"
        sh "cat docker-compose.yml"
    }

    stage('Build'){
        dir('backend') {
            sh "chmod +x gradlew"
            sh "./gradlew clean build"
        }
    }
}
```

```

    }

    stage('Deploy') {
        // 이전에 실행된 컨테이너를 중지하고 삭제합니다.
        sh "docker stop mefi-backend || true"
        sh "docker rm mefi-backend || true"

        sh "docker stop mefi-frontend || true"
        sh "docker rm mefi-frontend || true"

        // 이전에 빌드된 이미지를 삭제합니다.
        sh "docker rmi dev_mefi-frontend || true"
        sh "docker rmi dev_mefi-backend || true"

        sh "docker ps -a || true"
        sh "docker images || true"

        sh "docker-compose build --no-cache"
        sh "docker-compose up -d"
    }
}

```

10. 사용 포트 목록

| 이름 | 내부 포트 | 외부 포트 |
|------------|-------|-------|
| http | 80 | |
| https | 443 | |
| Vue | 5173 | 5173 |
| SpringBoot | 8080 | 8081 |

| 이름 | 내부 포트 | 외부 포트 |
|---------|-------|-------|
| Jenkins | 9000 | 9000 |
| MySQL | 3306 | 3306 |