



최종 포팅 매뉴얼

🕒 생성일	@2023년 5월 18일 오전 11:04
≡ 태그	산출물

개발 환경

[형상관리](#)
[이슈관리](#)
[OS](#)
[Communication](#)
[IDE](#)
[기타 툴](#)
[DataBase](#)
[Recommend System](#)
[Server](#)
[Front-End](#)
[Back-End](#)

EC2 세팅

[EC2](#)
[AWS\(EC2\) 접속](#)
[EC2 시간대 변경](#)
[ifconfig 사용을 위해 net-tools 설치](#)
[S3 설치](#)
[S3 설정](#)
[AWS RDS](#)
[RDS 생성 및 관리](#)
[RDS ReadOnlyReplica 생성 및 관리](#)
[MYSQL Workbench 연결](#)
[Docker](#)
[Docker Install](#)
[sudo 없이 docker 사용](#)
[Redis](#)
[Redis 설치 및 Container 구동](#)
[Redis DB 확인](#)
[JPA-Redis 연결](#)
[Jenkins](#)
[Jenkins image 다운로드 및 실행](#)
[Docker in docker 설치](#)
[Jenkins 플러그인 설치 및 Pipeline Project 실행](#)
[Jenkinsfile 설정](#)

Kubernetes

[CRI-O 설치](#)
[Kubernetes 설치](#)
[Calico 설치](#)
[Kubernetes Worker Node 연결을 위한 설정](#)
[Kubernetes SSL 인증 설정](#)
[Ingress-nginx, Ingress.yaml 설정](#)
[ArgoCD 설정](#)
[YAML 파일 설정 \(Gitops 설정\)](#)

기타 자료

[Kubernetes 명령어](#)
[jenkinsPipeline 기본 명령어](#)
[보안 설정](#)

개발 환경

형상관리

- GitLab

이슈관리

- Jira

OS

- Windows 10

기타 툴

- Postman

Server

- AWS EC2
 - Ubuntu 20.04 LTS
 - Kubernetes
 - Docker version 23.0.1
 - Jenkins 2.387.1
- AWS S3
- AWS RDS

Communication

- Mattermost
- Notion
- Figma
- ERD Cloud

DataBase

- MySQL
- Redis

Front-End

- Node.js v18.14.2
- React v18.2.0
- Recoil v0.7.7
- JavaScript ES5
- styled-components v5.3.8
- react-router-dom v6.8.2

IDE

- IntelliJ IDEA 2020.3.3
- Spring-Tool-Suite-4 4.17.1
- Visual Studio Code 1.75.1(user setup)
- MySQL WorkBench version 8.0.32 build 2612062 CE(64 bits)

Recommend System

- Numpy
- Pandas

Back-End

- Java 11
- Gradle 7.11
- Spring Boot 2.7.10
- JPA Hibernate
- Python 3.10
- Django 3.2.13

EC2 세팅

EC2

AWS(EC2) 접속

```
# Putty를 이용한 접속
SSH : PORT 22 를 이용한 포트 접속
Credential : pem key 생성 > PuttyGen을 통하여 ppk로 변경 (사용)
```

EC2 시간대 변경

```
sudo timedatectl set-timezone Asia/Seoul
```

ifconfig 사용을 위해 net-tools 설치


```
# instasll net-tools
sudo apt install net-tool

# ifconfig 사용 가능
ifconfig
```

S3 설치

S3 설정

- IAM 설정 : Amazon S3FullAccess를 통해 IAM 생성 (보안을 위하여 반드시 필요함)
- 버킷 생성 : Amazon S3 Bucket 생성 > ACL 활성화 > Public Access 차단 해제
- 버킷 정책 생성

- 

amazon

webservices

AWS Policy Generator

The AWS Policy Generator is a tool that enables you to create policies that control access to Amazon Web Services (AWS) products and resources. For more information about creating policies, see [key concepts in using AWS Identity and Access Management](#). Here are sample policies.

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 IAM Resource Policy, a VPC Resource Policy, and an SQS Queue Policy.

Select Type of Policy: select Policy Type

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: 모든 가능한 IAM
Use a console to select from multiple details.

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

Action: select Action All Actions (*) 필요한 Action 선택

Amazon Resource Name (ARN): 적용할 버킷의 ARN
ARN should follow the format: arn:aws:iam::[AWSAccountId]:role/[role-name].
Use a console to select from multiple details.

Add Condition (Optional): Add Statement

You added the following statements. Click the [button](#) below to generate a policy.

Principal(s)	Effect	Action	Resource	Condition
모든 사용자	allow	<ul style="list-style-type: none"> s3:GetObject s3:PutObject s3:DeleteObject s3:ListBucket s3:PutObjectVersion s3:DeleteObjectVersion s3:ListBucketVersions s3:PutObjectVersionTagging s3:DeleteObjectVersionTagging 	amazon-s3	None

Step 3: Generate Policy

A policy is a document (written in the Access Policy Language) that acts as a container for one or more statements.

Generate Policy
Start Over

- ```
build.gradle
implementation 'io.awspring.cloud:spring-cloud-starter-aws:2.4.4'

application.yml
cloud:
 aws:
 stack:
 auto: false
 region:
 static: ap-northeast-2
 credentials:
 access-key: THREE_S_ACCESSKEY
 secret-key: THREE_S_SECRETKEY
 s3:
 bucket: THREE_S_BUCKET
```

- ```
#### AWS
AWS_ACCESS_KEY_ID = os.environ.get('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY = os.environ.get('AWS_SECRET_ACCESS_KEY')
AWS_REGION = os.environ.get('AWS_REGION')

## S3 Storages
AWS_STORAGE_BUCKET_NAME = os.environ.get('AWS_STORAGE_BUCKET_NAME')
DEFAULT_FILE_STORAGE = os.environ.get('DEFAULT_FILE_STORAGE')
AWS_S3_CUSTOM_DOMAIN = ''
AWS_S3_OBJECT_PARAMETERS = {
    # 'ContentType' : 'image/jpeg'
}
```

3

- MYSQL DB를 외부에서 받고자 할 때 특정 포트만을 연결하도록 하는 AWS 보안 방식
- 보통은 Public Port 번호를 받으면 된다. (예시 : Naver에서 “내 IP” 검색 등)
- Inbound / outbound에서 특정 포트 연결
- RDS의 Subnet Router Table에도 특정 포트를 열어놓아야함.
 - Router Table : 특정 포트에 패킷을 전달할 수 있게끔 연결해주는 AWS 연결 방식
 - Router는 보통 OSI 3계층에서 연결 (AWS에서는 Router를 Router Table로 관리)
- Public Access : Yes 를 통하여 외부에서 사용 필요
 - Public Access는 해당 VPC 외부에서도 받을 수 있을 것인지를 확인하는 방식

RDS ReadOnlyReplica 생성 및 관리

- 기존의 RDS에서 “읽기 전용”으로만 따로 Database를 생성 가능
- 데이터베이스 특성 상 CUD보다 Read가 압도적으로 많기에 해당 Traffic을 분할하기 위하여 Replica 사용
- AWS에서는 하나의 DB에 최대 5개의 Replica 생성이 가능

```
# RDS 생성방식
AWS Instance > RDS 관리 > Replica 생성
```

MYSQL Workbench 연결

- Hostname : AWS MYSQL의 Endpoint
- Username : MYSQL 생성 시 사용한 root_user
- Password : MYSQL 생성 시 사용한 password
- port : 3306 (default)

Docker

Docker Install

```
sudo apt-get update

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

sudo mkdir -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

# install 확인
sudo docker version
```

sudo 없이 docker 사용

```
# 현재 계정을 docker group에 포함시킨다.
sudo usermod -aG docker ${USER}

# 다시 로그인하기
# 루트 계정으로 접속
sudo su -
# 다시 원래의 계정으로 로그인
su - ubuntu
```

```
# group에 docker가 제대로 들어갔는지 확인 -> docker가 보이면 성공
groups ubuntu

# 확인
docker ps
```

Redis

Redis 설치 및 Container 구동

```
docker pull redis
docker run --name redis -d -p 6379:6379 redis
```

Redis DB 확인

```
# redis db에 들어감
redis cli

# redis의 key 값 확인
keys * # 모든 key, value 값 확인
get [key값] # 특정 key의 value 확인하기
```

JPA-Redis 연결

```
implementation 'org.springframework.boot:spring-boot-starter-data-redis'

redis:
  lettuce:
    pool:
      max-active: 10
      max-idle: 10
      min-idle: 2
    port: 6379
    host: [도메인 이름] or public ip
```

Jenkins

Jenkins image 다운로드 및 실행

```
# Docker Hub에서 jenkins/jenkins:lts 이미지 pull
sudo docker pull jenkins/jenkins:lts-jdk11

# jenkins 이미지 다운 및 Container 설정
sudo docker run -u 0 -d -p 8080:8080 -p 50000:50000 -v /var/jenkins:/var/jenkins_home -e JENKINS_OPTS="--httpPort=8080 --httpsPort=-1"

# 처음 접속시 키값 확인
# /var/jenkins_home/secrets/initialAdminPassword에서 확인 가능
sudo docker logs jenkins
```

Docker in docker 설치

```
# Jenkins Docker 내부 접속
docker exec -it jenkins /bin/bash

# OS 확인
lsb_release -a

# docker 설치 (Jenkins 내부 OS에 따라 다르게 작동)
apt-get update

apt-get -y install \
  apt-transport-https \
  ca-certificates \
  curl \
```

```

gnupg \
lsb-release

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

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

apt-get update

apt-get -y install docker-ce docker-ce-cli containerd.io

```

Jenkins 플러그인 설치 및 Pipeline Project 실행

- 플러그인 설치
 - Docker 관련 plugin (Dockerhub)
 - Pipeline 관련 plugin
 - Gitlab 관련 plugin
 - Gradle 관련 plugin
- Pipeline Project 실행
 - New project > pipeline project
 - Github 연결

Configure

General

Advanced Project Options

Pipeline

설명

[Plain text] [미리보기](#)

☐ Do not allow concurrent builds

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

☒ GitHub project

Project url [?](#)

<https://lab.ssfy.com/s08-final/S08P31A207.git/>

- Build Triggers > Merge일 때만으로 한정 (Push는 이후 yaml 파일 변경 시 필요하므로)

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://k8a207.p.ssafy.io:8080/project/watchify> ?

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

- main에서만 Accepted main에 올라올 때만 / Secret Token 발급

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 ?

Include

Exclude

- ☐ Filter branches by regex ?

Secret token ?

Generate

Clear

- Pipeline 설정

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssafty.com/s08-final/S08P31A207.git/

Credentials ?

[REDACTED]

Add +

그룹 ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add +

Script Path ?

Jenkinsfile

Jenkinsfile 설정

```

pipeline {
    environment{
        repository = "runtogether/watchify" // 공통되게 사용할 변수 사용
        DOCKERHUB_CREDENTIALS = credentials('Dockerhub-jenkins') // jenkins에 등록해 놓은 docker hub credentials 이름
    }
    agent any
    stages {
        stage('Frontend Build') {
            steps {
                echo 'Frontend Building'
                script {
                    def BUILD_NUMBER = currentBuild.number // script 안에서만 가능하다. (플러그인 필요)
                    sh 'docker build -t $repository:frontend$BUILD_NUMBER ./frontend' // frontend 파일 생성
                    sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin' // docker hub 로그인
                    sh 'docker push $repository:frontend$BUILD_NUMBER' //docker push
                }
            }
        }

        stage('BACKEND Build'){
            steps{
                echo 'BACKEND Building'
                script {
                    def BUILD_NUMBER = currentBuild.number
                    // BACKEND에서 JENKINS Global 변수 설정된 것을 사용
                    // 파일에서 설정을 변경한 후 Image를 생성 필요
                    dir('BACKEND/watchify/src/main/resources'){
                        sh """
                            sed -i 's/DB_USER/"$DB_USER"/g' application.yml
                            sed -i 's/DB_PW/"$DB_PW"/g' application.yml
                            sed -i 's/DB_HOST/"$DB_HOST"/g' application.yml
                            sed -i 's/SERVER_HOST/"$SERVER_HOST"/g' application.yml
                        """
                    }

                    dir('BACKEND/watchify/src/main/resources'){
                        sh """
                            sed -i 's/THREE_S_ACCESSKEY/"$THREE_S_ACCESSKEY"/g' application-dev.yml
                            sed -i 's/THREE_S_SECRETKEY~'"$THREE_S_SECRETKEY"'~g' application-dev.yml
                            sed -i 's/THREE_S_BUCKET/"$THREE_S_BUCKET"/g' application-dev.yml
                            sed -i 's/FCM_JSON/"$FCM_JSON"/g' application-dev.yml
                        """
                    }
                }
            }
        }
    }
}

```



```

    """
}

dir('BACKEND/watchify') { // 해당 directory로 들어가기 위해서는 cd는 안되고 대신 dir를 사용해야한다.
    sh 'chmod +x gradlew'
    sh './gradlew clean build -x test'
}
// 이미지를 빌드
sh 'docker build -t $repository:backend$BUILD_NUMBER ./BACKEND/watchify'
// Dockerhub Login
sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin' // docker hub로 로그인
// Dockerhub로 올린다.
sh 'docker push $repository:backend$BUILD_NUMBER'

// 다시 원래대로 돌려놓는다.
dir('BACKEND/watchify/src/main/resources'){
    sh """
        sed -i 's/"$DB_USER"/DB_USER/g' application.yml
        sed -i 's/"$DB_PW"/DB_PW/g' application.yml
        sed -i 's/"$DB_HOST"/DB_HOST/g' application.yml
        sed -i 's/"$SERVER_HOST"/SERVER_HOST/g' application.yml
    """
}

dir('BACKEND/watchify/src/main/resources'){
    sh """
        sed -i 's/"$THREE_S_ACCESSKEY"/THREE_S_ACCESSKEY/g' application-dev.yml
        sed -i 's-"$THREE_S_SECRETKEY"~'$THREE_S_SECRETKEY'-g' application-dev.yml
        sed -i 's/"$THREE_S_BUCKET"/THREE_S_BUCKET/g' application-dev.yml
        sed -i 's/"$FCM_JSON"/FCM_JSON/g' application-dev.yml
    """
}
}

}

stage('ReadOnlyBACKEND Build'){
    steps{
        echo 'ReadOnlyBACKEND Building'
        script {
            def BUILD_NUMBER = currentBuild.number

            dir('ReadOnlyBackend/watchify/src/main/resources'){
                sh """
                    sed -i 's/DB_USER/"$DB_USER"/g' application.yml
                    sed -i 's/DB_PW/"$DB_PW"/g' application.yml
                    sed -i 's/READONLYDB_HOST/"$READONLYDB_HOST"/g' application.yml
                    sed -i 's/SERVER_HOST/"$SERVER_HOST"/g' application.yml
                """
            }

            dir('ReadOnlyBackend/watchify/src/main/resources'){
                sh """
                    sed -i 's/THREE_S_ACCESSKEY/"$THREE_S_ACCESSKEY"/g' application-dev.yml
                    sed -i 's-THREE_S_SECRETKEY~'$THREE_S_SECRETKEY'-g' application-dev.yml
                    sed -i 's/THREE_S_BUCKET/"$THREE_S_BUCKET"/g' application-dev.yml
                    sed -i 's/FCM_JSON/"$FCM_JSON"/g' application-dev.yml
                """
            }

            dir('ReadOnlyBackend/watchify') { // 해당 directory로 들어가기 위해서는 cd는 안되고 대신 dir를 사용해야한다.
                sh 'chmod +x gradlew'
                sh './gradlew clean build -x test'
            }

            sh 'docker build -t $repository:readonlybackend$BUILD_NUMBER ./ReadOnlyBackend/watchify'
            sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin' // docker hub로 로그인
            sh 'docker push $repository:readonlybackend$BUILD_NUMBER'

            dir('ReadOnlyBackend/watchify/src/main/resources'){
                sh """
                    sed -i 's/"$DB_USER"/DB_USER/g' application.yml
                    sed -i 's/"$DB_PW"/DB_PW/g' application.yml
                    sed -i 's/"$READONLYDB_HOST"/READONLYDB_HOST/g' application.yml
                    sed -i 's/"$SERVER_HOST"/SERVER_HOST/g' application.yml
                """
            }

            dir('ReadOnlyBackend/watchify/src/main/resources'){
                sh """
                    sed -i 's/"$THREE_S_ACCESSKEY"/THREE_S_ACCESSKEY/g' application-dev.yml
                    sed -i 's-"$THREE_S_SECRETKEY"~'$THREE_S_SECRETKEY'-g' application-dev.yml
                    sed -i 's/"$THREE_S_BUCKET"/THREE_S_BUCKET/g' application-dev.yml
                    sed -i 's/"$FCM_JSON"/FCM_JSON/g' application-dev.yml
                """
            }
        }
    }
}

```

```

    }
  }
}

stage('AI Build') {
  steps {
    echo 'AI Building'
    script {
      def BUILD_NUMBER = currentBuild.number
      // Django 보안 설정을 위하여 envfile에 들어가서 변경 > image 생성 > 다시 되돌리는 순서대로 진행
      dir('AI/watchifyAI'){
        sh """
          sed -i 's/DB_NAME/"$DB_NAME"/g' settings.py
          sed -i 's/DB_USER/"$DB_USER"/g' settings.py
          sed -i 's/DB_PW/"$DB_PW"/g' settings.py
          sed -i 's/DB_HOST/"$DB_HOST"/g' settings.py
          sed -i 's/DB_PORT/"$DB_PORT"/g' settings.py
        """
      }
      sh 'docker build -t $repository:ai$BUILD_NUMBER ./AI' // frontend 파일 생성
      sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin' // docker hub 등록
      sh 'docker push $repository:ai$BUILD_NUMBER' //docker push
      dir('AI/watchifyAI'){
        sh """
          sed -i 's/"$DB_NAME"/DB_NAME/g' settings.py
          sed -i 's/"$DB_USER"/DB_USER/g' settings.py
          sed -i 's/"$DB_PW"/DB_PW/g' settings.py
          sed -i 's/"$DB_HOST"/DB_HOST/g' settings.py
          sed -i 's/"$DB_PORT"/DB_PORT/g' settings.py
        """
      }
    }
  }
}

stage('Gitops Dir') {
  steps {
    echo "Gitops Dir"
    script{
      def BUILD_NUMBER = currentBuild.number
      def credentialId = env.CREDENTIAL_ID
      def gitId = env.GIT_USER
      def gitpassword = env.GIT_PASSWORD
      // main에서 Pull을 받아야 그 이후에 main에서 version build가 가능
      withCredentials([usernamePassword(credentialsId: credentialId, passwordVariable: gitpassword, usernameVariable: gitId)]){
        sh 'git remote set-url origin https://sdc00035:diligent0924!@lab.ssafy.com/s08-final/S08P31A207.git'
        sh 'git stash'
        sh 'git switch main'
        sh 'git pull origin main'
      }
      // kubfiles 내의 yaml 파일을 최신 버전으로 변경
      dir("kubefiles"){
        // yaml파일을 현재 build 숫자로 변경
        sh """
          sed -i 's/watchify:frontend\\([[:^:]]*\\)/watchify:frontend${BUILD_NUMBER}/g' my-service.yaml
          git add my-service.yaml
          git commit -m 'Update my-service tag to frontend${BUILD_NUMBER}'
        """
        sh """
          sed -i 's/watchify:backend\\([[:^:]]*\\)/watchify:backend${BUILD_NUMBER}/g' back-service.yaml
          git add back-service.yaml
          git commit -m 'Update back-service tag to backend${BUILD_NUMBER}'
        """
        sh """
          sed -i 's/watchify:ai\\([[:^:]]*\\)/watchify:ai${BUILD_NUMBER}/g' ai-service.yaml
          git add ai-service.yaml
          git commit -m 'Update back-service tag to ai${BUILD_NUMBER}'
        """
        sh """
          sed -i 's/watchify:readonlybackend\\([[:^:]]*\\)/watchify:readonlybackend${BUILD_NUMBER}/g' readonlyback-service.yaml
          git add readonlyback-service.yaml
          git commit -m 'Update back-service tag to readonlyback-service${BUILD_NUMBER}'
        """
      }
      // git main에 실제로 올린다. ( main )
      withCredentials([usernamePassword(credentialsId: credentialId, passwordVariable: gitpassword, usernameVariable: gitId)]){
        sh 'git remote set-url origin https://sdc00035:diligent0924!@lab.ssafy.com/s08-final/S08P31A207.git'
        sh 'git switch main'
        sh 'git pull origin main'
        sh 'git push origin main'
      }
    }
    echo 'git OK'
  }
}
}

```

```
}  
}  
}
```

Kubernetes

CRI-O 설치

- CRI-O를 사용할 때 필요한 리눅스 커널 모듈 설치

```
cat <<EOF | sudo tee /etc/modules-load.d/crio.conf  
overlay  
br_netfilter  
EOF
```

- 디스크 오버레이 파일시스템 모듈 / CNI 플러그인에서 사용되는 네트워크 필터링 기능

```
sudo modprobe overlay # 디스크 오버레이 파일시스템 모듈  
sudo modprobe br_netfilter # CNI 플러그인에서 사용되는 네트워크 필터링 기능
```

- 호스트 시스템에 필요한 네트워크 관련 설정 추가

```
cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf # 파일 등록  
net.bridge.bridge-nf-call-iptables = 1 #  
net.ipv4.ip_forward = 1 #  
net.bridge.bridge-nf-call-ip6tables = 1  
EOF  
  
sudo sysctl --system # 파일 시스템을 적용시킨다.
```

- CRI-O 설치

```
sudo -i  
export OS=xUbuntu_20.04 # OS 버전  
export VERSION=1.26 # cri-o 버전  
  
echo "deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/ /" > /etc/apt/sources.list.d/cri-o.list  
curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/Release.key | apt-key add -  
  
echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /" | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION:$OS:Release.key  
curl -L "https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key" | sudo apt-key add -  
  
apt-get update  
  
apt-get install cri-o cri-o-runc  
  
sudo systemctl daemon-reload  
sudo systemctl enable crio --now  
sudo systemctl status crio
```

Kubernetes 설치

- 구글 클라우드의 공개 사이닝 키를 다운

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

- 쿠버네티스 apt 리포지토리를 추가

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

- apt 패키지 색인을 업데이트하고, kubelet, kubeadm, kubectl을 설치하고 해당 버전을 고정

```
sudo apt-get update
sudo apt-get install -y kubelet=1.26.2-00 kubeadm=1.26.2-00 kubectl=1.26.2-00
sudo apt-mark hold kubelet kubeadm kubectl
```

- CRI-O 구성 및 수정

```
sudo vi /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

[Service]
Environment="KUBELET_EXTRA_ARGS=--container-runtime=remote --cgroup-driver=systemd --runtime-request-timeout=15m --container-runtime-e
```

- 설정 적용을 위해 kubectl 재시작

```
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

- Ubuntu 설정 변경

```
# ~/.bashrc에서는 ubuntu에서 단축키로 사용할 것들을 추가해서 넣을 수 있다.
echo "alias k='kubectl'" >> ~/.bashrc
source ~/.bashrc
```

- 편리한 모듈 설치

```
sudo git clone https://github.com/ahmetb/kubectx /opt/kubectx
sudo ln -s /opt/kubectx/kubectx /usr/local/bin/kubectx
sudo ln -s /opt/kubectx/kubens /usr/local/bin/kubens
```

- Kubernetes Master node 생성

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16
```

- Master node 설정 파일 생성

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# 루트계정자라면 해당 부분을 사용해야함
export KUBECONFIG=/etc/kubernetes/admin.conf
```

Calico 설치

- Calico 다운로드

```
curl https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml -O
```

- Calico 변경

```
vi calico.yaml

~~~
livenessProbe:
  exec:
    command:
      - /bin/calico-node
      - -felix-live
      - -bird-live # --> 이 부분을 삭제합니다.
  periodSeconds: 10
  initialDelaySeconds: 10
  failureThreshold: 6
readinessProbe:
  exec:
```

```

      command:
        - /bin/calico-node
        - -felix-ready
        - -bird-ready # --> 이 부분을 삭제합니다.
    ~~~~
kind: ConfigMap
apiVersion: v1
metadata:
  name: calico-config
  namespace: kube-system
data:
  # Typha is disabled.
  typha_service_name: "none"
  calico_backend: "vxlan" # 이 부분을 수정한다.
  ~~~~
# Enable IPIP
- name: CALICO_IPV4POOL_IPIP
  value: "Never" # IP-IP 모드 비활성화
- name: CALICO_IPV4POOL_VXLAN
  value: "Always"
  ~~~~

```

- Calicoctl 설치

```

cd /usr/local/bin/

sudo curl -O -L https://github.com/projectcalico/calicoctl/releases/download/v3.20.6/calicoctl

sudo chmod +x calicoctl

export DATASTORE_TYPE=kubernetes

export KUBECONFIG=~/.kube/config

```

Kubernetes Worker Node 연결을 위한 설정

- Master Node의 Access Token 및 Hash값 생성
- 다른 기기에서 워커노드 생성 (다른 VPC여도 연동이 되는 것인지에 대해 확인 필요)
- kube-apiserver port 설정 파일

```

# kube-apiserver의 기본 포트가 50000 - 60000인데 이렇게 쓰긴 싫으니까 포트를 강제로 열기 위한 설정 파일

sudo vi /etc/kubernetes/manifests/kube-apiserver.yaml

~~~~
spec:
  containers:
    - command:
      - --service-node-port-range=1-65535 # 추가
  ~~~~

```

- 하나의 EC2 안에서 Master/worker node 생성

```

# 만약 MasterNode에 Worker Node를 같이 쓰고 싶다면
# 단일노드 생성시
## Taints 확인
kubectl describe node <master-node-name> | grep Taints

## Taints가 존재할 경우 제거
kubectl taint node <master-node-name> node-role.kubernetes.io/control-plane:NoSchedule-

```

Kubernetes SSL 인증 설정

- cert-manager 설치

```

k apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.11.1/cert-manager.yaml

```

- cluster-issuer.yaml 생성 및 적용

```

1. cluster-issuer.yaml 생성
vim cluster-issuer.yaml

2. 아래 코드 작성
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme: # Automated Certificate Management Environment
    # 어떤 acme 서버를 사용할 지 지정 (아래 예제는 let's encrypt의 CA)
    server: https://acme-v02.api.letsencrypt.org/directory
    # 사용자 이메일 주소 기재
    email: 본인이메일
    # 사용자의 개인키를 저장할 Secret 리소스 이름을 지정
    privateKeySecretRef:
      name: letsencrypt-prod
    # 도메인 주소에 대한 소유권 증명을 위한 방법 선택
    solvers:
      - http01: # http 요청을 통한 도메인 주소 소유권 증명 방법 사용
        ingress: # 이 때 사용할 ingress 컨트롤러 지정
          class: nginx # ingress 컨트롤러 타입 기재

```

- certificate.yaml 생성 및 적용

```

apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-certificate
  namespace: default
spec:
  secretName: my-certificate # TLS 키 이름을 지정합니다.
  duration: 2160h # 90d
  renewBefore: 360h # 15d
  issuerRef:
    name: letsencrypt-prod # issuer.yaml 의 이름과 일치해야 합니다.
    kind: ClusterIssuer
  dnsNames: # DNS 이름 설정
    - k8a207.p.ssafy.io

```

Ingress-nginx, Ingress.yaml 설정

- Ingress-nginx 설치

```
curl -o ingress-nginx.yaml https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.6.4/deploy/static/provider/baremet
```

- Ingress-nginx 설정

```

Ingress-nginx.yaml 파일 내 진입
ports:
  - appProtocol: http
    name: http
    port: 80
    protocol: TCP
    targetPort: http
    nodePort: 80 # 이 부분 추가
  - appProtocol: https
    name: https
    port: 443
    nodePort: 443 # 이 부분 추가
    protocol: TCP
    targetPort: https
selector:
  app.kubernetes.io/component: controller
  app.kubernetes.io/instance: ingress-nginx
  app.kubernetes.io/name: ingress-nginx
type: NodePort # 이 부분 추가

kubectl apply -f ingress-nginx.yaml

```

- Ingress.yaml 설정

```

# ingress 작성 시 만약 url이 여러개라면 따로 써줘야한다.
apiVersion: networking.k8s.io/v1

```

```

kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  tls:
    - hosts:
        - k8a207.p.ssafy.io
      secretName: my-certificate
  rules:
    - host: k8a207.p.ssafy.io
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: my-service
                port:
                  number: 3000
  ---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress-oauth2
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  tls:
    - hosts:
        - k8a207.p.ssafy.io
      secretName: my-certificate
  rules:
    - host: k8a207.p.ssafy.io
      http:
        paths:
          - path: /oauth2
            pathType: Prefix
            backend:
              service:
                name: back-service
                port:
                  number: 8080
... (위와 같은 방식으로 사용)

```

ArgoCD 설정

- namespace 생성

```
kubectl create namespace argocd
```

- argocd 설치파일 생성

```
curl https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml -o argo.yaml
```

- argocd Service 및 pods 설치

```
kubectl -n argocd apply -f argo.yaml # 해당 namespace를 기준으로 argo.yaml 설치
```

- argocd.yaml 파일 설정 변경

```

vi argo.yaml

spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 8080
      nodePort: 8090
    - name: https
      port: 443

```

```

protocol: TCP
targetPort: 8080
nodePort: 8091
selector:
  app.kubernetes.io/name: argocd-server
type: NodePort
---
```

- argocd 사이트 들어가기

```

# 비밀번호 확인
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo
(참고로 id는 admin(default))
```

- Gitlab 비밀번호 변경 / 연동

```

UserInfo > 비밀번호 변경
Repository > gitlab 연동 (만약 연동 안된다면 ufw 에러임)
Application > gitops파일 연결(kubefiles -> 미리 생성해야함)
```

YAML 파일 설정 (Gitops 설정)

- 파일 생성 시 주의점

```

# Ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  tls:
  - hosts:
    - k8a207.p.ssafy.io
    secretName: my-certificate
  rules:
  - host: k8a207.p.ssafy.io
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service # 해당 부분과 같은 Servicename 필요
            port:
              number: 3000 # 같은 ContainerPort를 반드시 사용
```

- 파일 생성 방법 (Service와 pod를 같이 생성 예시)

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ai-service
  labels:
    app: ai-service
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ai-service
  template:
    metadata:
      labels:
        app: ai-service
    spec:
      containers:
      - name: ai-service
        image: docker.io/runtogether/watchify:ai275
      imagePullSecrets:
      - name: dockerhub
      ports:
      - containerPort: 8100
      env: # 이 부분은 만약 env에 연결한다면 반드시 필요함
```



```

- name: DB_HOST
  valueFrom: # 이부분은 Kubernetes Secret으로 암호화
    secretKeyRef:
      name: db-secret
      key: DB_HOST
- name: DB_PORT
  valueFrom:
    secretKeyRef:
      name: db-secret
      key: DB_PORT
- name: DB_NAME
  valueFrom:
    secretKeyRef:
      name: db-secret
      key: DB_NAME
- name: DB_USER
  valueFrom:
    secretKeyRef:
      name: db-secret
      key: DB_USER
- name: DB_Password
  valueFrom:
    secretKeyRef:
      name: db-secret
      key: DB_PASSWORD
---
apiVersion: v1
kind: Service
metadata:
  name: ai-service # 이 부분이 ingress.yaml과 같아야한다.
spec:
  type: ClusterIP
  selector:
    app: ai-service
  ports:
    - protocol: TCP
      port: 8100

```

기타 자료

Kubernetes 명령어

```

# 다양한 서비스를 확인하는 방법
k get nodes : 실제 node(EC2)를 확인한다.
k get pods : 파드들을 확인한다.
k get svc : 서비스를 확인한다.
k get crd : crd를 확인한다.
k get replica : 레플리카를 확인한다.

# 구체적인 정보를 확인하기
k logs [pod 이름] : pod의 구체적인 에러를 확인할 수 있다.
kubectl pod <pod 이름> -n <네임스페이스> : Event에서 구체적인 Error를 확인할 수 있음
kubens or k get namespace : namespace를 확인할 수 있다.

# deployment 실행하기
k apply -f [00.yaml] : deployment 실행 (Replica, Service, pod 확인 가능)
k delete -f [00.yaml] : deployment 삭제 (전체 삭제)

# 예러 시 확인하는 방법 ( finalizers Error )
kubectl get namespace [namespace] -o yaml # 해당 namespace에 대하여 구체적인 확인
kubectl patch crd applications.argoproj.io -p '{"metadata":{"finalizers": []}}' --type=merge # finalizer 제거

```

jenkinsPipeline 기본 명령어

```

# 다양한 명령어
echo : print()
script{} : Groovy 언어를 사용하겠다. => 변수, 파일 열기 등 다양한 것이 가능하다.
sh : CNI를 위해 사용하는 것을 의미한다.

# 현재 빌드 번호를 보여주는 것(숫자를 증가시킬 때 중요한 역할을 할 듯)
def buildNumber = currentBuild.number

# 경로 바꾸는 방법
cd ~ 는 Jenkins에서는 절대 안먹힌다.
dir('BACKEND/watchify') {
  sh 'chmod +x gradlew'
  sh './gradlew clean build -x test'
}

```

```
}  
이것과 같이 dir로 경로를 지정할 수 있다.
```

보안 설정

- Kubernetes Secret (pod 설정)

```
Kubernetes에서 민감한 부분의 경우 Secret을 통하여 관리할 수 있다.  
k get secrets : secrets을 확인한다.  
  
# Secret yaml 파일 작성  
apiVersion: v1  
kind: Secret  
metadata:  
  name: db-secret  
type: Opaque  
data:  
  DB_HOST: [문자 => base64 변환]  
  DB_PORT: [문자 => base64 변환]  
  DB_NAME: [문자 => base64 변환]  
  DB_USER: [문자 => base64 변환]  
  DB_PASSWORD: [문자 => base64 변환]  
  
# Secret namespace 생성  
k create namespace secrets  
  
# Secret 확인  
k get secrets  
  
# Apply 적용  
k apply -n secrets -f 'aa.yaml'  
  
# 사용 예시  
  
- name: DB_HOST  
  valueFrom:  
    secretKeyRef: # Secret 형태를 들고온다.  
      name: db-secret # 어떤 Secrets?  
      key: DB_HOST # key를 가져온다.  
  
#####  
# Argocd에 Secret이 먹히지 않는 예외 발생  
=> Secret은 같은 namespace에서만 먹힐 수 있다. (아마 대부분 default일 것이다.)
```

- Dockerhub Private을 통한 보안 설정

```
1. Dockerhub private로 변경  
2. kubernetes secret key 추가 (주의: 띄어쓰기 금지!)  
kubectl create secret docker-registry <SECRET_NAME> \  
  --docker-server= https://index.docker.io/v1/ \  
  --docker-username=<DOCKERHUB_USERNAME> \  
  --docker-password=<DOCKERHUB_PASSWORD> \  
  --docker-email=<DOCKERHUB_EMAIL>  
3. 각각의 yaml 파일들 들어간다. (주의 : 라인선이 무조건 맞아야한다.)  
spec:  
  containers:  
    - name: my-container  
      image: my-image  
  imagePullSecrets: # 추가  
    - name: <SECRET_NAME> # 추가
```

- Jenkins Global variable을 이용한 보안 설정

```
Django의 설정파일이나 JPA의 설정 파일을 gitlab에 올리지 않고 진행할 수 있을까?  
=> Jenkins에서 자동 배포 시 Global variable을 이용하여 노출을 피하자.  
=> Jenkins 내부 variable은 내부에서만 볼 수 있기에 관리자를 제외하고는 볼 수 없다.  
=> 또한, github/gitlab과 같은 경우에도 env파일에 변수 형태로 들어가기 때문에 보안에 좋다.
```

1. Jenkins 접속
2. Jenkins > Jenkins 관리 > 시스템 설정 > Global Properties > Key-value 목록
3. key-value 목록 추가 (여기에 민감 정보를 포함하여 넣는다.)
4. Jenkinsfile에서 해당 파일에 들어가서 변수를 입력하여 넣는다.
5. 상황에 따라서 다시 돌려보아야한다.

```
-----  
보안 관련 예외  
jenkins Global variable로 사용할 때 만약 variable 안에 "/"가 포함된 경우
```

```
=> sed i " ~ "의 경우에는 "/"를 기준으로 나누는 문법이므로 기본적인 것으로 사용 X
sed -i 's/THREE_S_ACCESSKEY/"$THREE_S_ACCESSKEY"/g' application-dev.yml (X)
sed -i 's~THREE_S_SECRETKEY~'"$THREE_S_SECRETKEY""~g' application-dev.yml (O)
```