



포팅 매뉴얼

Docker 설치

Docker 설치 전 필요한 패키지 설치

Docker에 대한 GPC Key 인증 진행

Docker 레포지토리 등록

패키지 리스트 갱신

Docker 패키지 설치

Docker 일반 유저에게 권한 부여

Nginx 설치

Host OS 설치

SSL 설정 (CertBot)

CertBot 다운로드

SSL 인증서 발급

Nginx Docker Container 설치

Nginx 컨테이너 실행

NginX OS 재시작 후 자동실행 설정

Reverse Proxy 설정

nginx.conf

/sites-available/default, /sites-enable/default

Nginx 실행

Jenkins 설치

Docker Container 받기 (jdk17)

Docker 실행

페이지 설정

플러그인 설치할 항목

Credential 설정

Jenkins OS 재시작 후 자동실행 설정

Docker jenkins 에서 Host Docker 접근 권한 부여

Frontend Pipeline

Backend Pipeline

MySQL 설치

Redis 설치

도커 관리 시 Command

도커 컨테이너의 로그 보기

- [도커 컨테이너 접속](#)
- [서버 포트 개방](#)
 - [BackEnd](#)
 - [FrontEnd](#)
 - [NginX \(서버\)](#)
 - [ufw 명령어](#)
- [AWS EKS\(Elastic Kubernetes Service\)](#)
 - [IAM 관리자 설정](#)
 - [eksctl, kubectl, awscli 설치](#)
 - [eksctl](#)
 - [kubectl](#)
 - [awscli](#)
 - [버전 확인](#)
 - [VPC 설정](#)
 - [VPC Flow log를 통해 접속 로그 확인](#)
 - [Cluster 생성](#)
 - [Cluster IAM 역할 생성](#)
 - [생성 페이지 \(AWS Management Console\)](#)
 - [OIDC 공급자 설정](#)
 - [IAM - 자격 증명 공급자 생성](#)
 - [Node Group 권한 설정](#)
 - [Node Group 생성](#)
 - [Node group 구성](#)
 - [Node 개수 지정](#)
 - [네트워크 지정](#)
 - [노드 자원 사용량 확인](#)
 - [AWS LoadBalancer](#)
 - [AWS LoadBalancer Controller 설치](#)
 - [LoadBalancer Role](#)
 - [LoadBalancer 설정](#)
 - [Listener 규칙 설정](#)
 - [모니터링](#)
 - [Kubernetes 설정 파일](#)
 - [Deployment.yaml](#)
 - [Ingress.yaml](#)
 - [Service.yaml](#)
 - [AWSCLI 명령어](#)
 - [계정 확인](#)
 - [계정 등록](#)
 - [Kubectl 명령어](#)
 - [배포](#)
 - [삭제](#)

[로그](#)
[배포된 설정 파일 보기](#)
[이벤트 로그](#)
[프로젝트 설정 파일](#)
[Backend](#)
[Frontend](#)



EKS로 프론트엔드 애플리케이션 배포 관리, Docker Container로 백엔드 애플리케이션 배포 관리

Docker 설치

Docker 설치 전 필요한 패키지 설치

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

Docker에 대한 GPG Key 인증 진행

- OK가 떴다면 정상적으로 등록이 되었다는 뜻이다

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | s  
udo apt-key add -
```

Docker 레포지토리 등록

- AMD64 계열

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

패키지 리스트 갱신

```
sudo apt-get -y update
```

Docker 패키지 설치

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

Docker 일반 유저에게 권한 부여

```
sudo usermod -aG docker ubuntu
```

Nginx 설치

Host OS 설치

```
sudo apt-get -y install nginx
```

SSL 설정 (CertBot)

CertBot 다운로드

- snap을 이용해서 다운로드 한다
- `sudo apt-get -y install certbot` 명령어를 이용하면 다음과 같은 오류가 발생하기 때문이다
 - **The requested nginx plugin does not appear to be installed**

```
sudo snap install --classic certbot
```

SSL 인증서 발급

- nginx 사용시
 - `-d` : 등록할 도메인 Host 주소를 입력한다

```
sudo certbot --nginx -d <my-domain>
```

- 인증서가 발급되면 다음과 같은 곳에 `fullchain.pem` 과 `privkey.pem` 이 발급된다
 - 인증서는 발급하려는 도메인마다 디렉토리 위치가 다르니 주의
 - 인증서는 90일마다 갱신해야 하지만, `/etc/cron.d` 에 자동으로 갱신되는 스크립트가 설치중 기록된다
 - 갱신되는지 테스트를 하고 싶다면 다음 명령어를 입력한다
 - `sudo certbot renew --dry-run`

Nginx Docker Container 설치

- latest 버전 설치

```
sudo docker pull nginx:latest
```

Nginx 컨테이너 실행

```
docker run -d -p 80:80 -p 443:443 -e TZ=Asia/Seoul --name nginx -u root nginx:latest
```

- run : 이미지와 함께 컨테이너 생성
- -d : 백그라운드에서 실행되도록 하는 옵션
- -e TZ=Asia/Seoul : 환경변수 설정 (컨테이너 내부 시간대를 Asia/Seoul로 지정)
- --name : 컨테이너 이름 지정
- -p : 컨테이너 포트 연결

NginX OS 재시작 후 자동실행 설정

- docker-nginx.service 등록

```
sudo vim /etc/systemd/system/docker-nginx.service
```

```
[Unit]
Description=docker-nginx
Wants=docker.service
After=docker.service

[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start nginx
ExecStop=/usr/bin/docker stop nginx

[Install]
WantedBy=multi-user.target
```

- docker 서비스 활성화

```
sudo systemctl enable docker
```

- docker 서비스 시작

```
sudo systemctl start docker
```

- docker-nginx 서비스 활성화

```
sudo systemctl enable docker-nginx.service
```

- docker-nginx 서비스 시작

```
sudo systemctl start docker-nginx.service
```

Reverse Proxy 설정

nginx.conf

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
    # multi_accept on;
}

http {
    ##
    # Basic Settings
```

```

##

sendfile on;
tcp_nopush on;
tcp_nodelay on;
keepalive_timeout 65;
types_hash_max_size 2048;
# server_tokens off;

# server_names_hash_bucket_size 64;
# server_name_in_redirect off;

include /etc/nginx/mime.types;
default_type application/octet-stream;

##
# SSL Settings
##

ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Drop
ping SSLv3, ref: POODLE
ssl_prefer_server_ciphers on;

##
# Logging Settings
##

access_log /var/log/nginx/access.log;
error_log /var/log/nginx/error.log;

proxy_set_header Connection '';
proxy_http_version 1.1;
##
# Gzip Settings
##

gzip on;

```



```

##
# Virtual Host Configs
##

include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*; # sites-available 의 파일이 sites-enabled로
}

```

/sites-available/default, /sites-enabled/default

```

server {
    listen 80;
    server_name moomul.kr;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    server_name moomul.kr;

    ssl_certificate /etc/letsencrypt/live/moomul.kr/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/moomul.kr/privatekey.pem;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass http://moomul-application-load-balancer-1172084242.ap-northeast-2.elb.amazonaws.com; # Proxy to AWS LoadBalancer
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}

```

```
        proxy_set_header X-NginX-Proxy true;
    }

    location /api {
        proxy_pass http://moomul.kr:8089;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_read_timeout 86400;
    }
}
```

Nginx 실행

- 상태 보기

```
sudo systemctl status nginx
```

- 재시작

```
sudo systemctl restart nginx
```

- 종료

```
sudo systemctl stop nginx
```

Jenkins 설치



CI/CD 파이프라인을 작성해 Gitlab과 연동, Gitlab에서 이벤트 발생 시 웹훅을 통해 서버에 빌드

Docker Container 받기 (jdk17)

```
docker pull jenkins/jenkins:jdk17
```

Docker 실행

```
docker run -d --env JENKINS_OPTS=--httpPort=8080 -v /etc/localtime:/etc/localtime:ro -e TZ=Asia/Seoul -p 8080:8080 -v /jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -v /usr/local/bin/docker-compose:/usr/local/bin/docker-compose --name jenkins -u root jenkins/jenkins:latest
```

- 8080 포트로 설정

페이지 설정

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Continue

```
docker exec -it jenkins /bin/bash
```

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

비밀번호를 입력 후 설정 진행하면 된다.

플러그인 설치할 항목

```
# ssh 커맨드 입력에 사용  
SSH Agent
```

```
# docker 이미지 생성에 사용  
Docker  
Docker Commons  
Docker Pipeline  
Docker API
```

```
# 웹훅을 통해 브랜치 merge request 이벤트 발생시 Jenkins 자동 빌드  
에 사용
```

Generic Webhook Trigger

```
# 타사 레포지토리 이용시 사용 (GitLab, Github 등)
```

GitLab

GitLab API













GitLab Authentication

GitHub Authentication

```
# Node.js 빌드시 사용, for React Application
```

NodeJS

Credential 설정

T	P	Store ↓	Domain	ID	Name
		System	(global)	seungwoo-gitlab	wskyard96/***** (Gitlab credential)
		System	(global)	moomul-token	coderyard/***** (dockerhub access token)
		System	(global)	ubuntu-a107	ubuntu (ubuntu ssh)
		System	(global)	jenkins-seungwoo	GitLab API token (gitlab api key for connect to jenkins)
		System	(global)	moomul-dockerhub	coderyard/*****
		System	(global)	eks-admin	AKIART7UDDCJ3F5JHFFI (IAM for eks)

- Gitlab : 이벤트 발생 시 웹훅을 받아오기 위해 필요
- DockerHub Token : DockerHub Repository에 연결해 이미지를 올리고, 받아오기 위해 사용
- ubuntu : 서버와 연결하기 위해 사용
- Gitlab API Token : Gitlab과 연동, 웹훅 사용 시 필요
- eks-admin : EKS Cluster에 접근해 이미지를 파드에 배포하기 위해 사용
 - Pod : 쿠버네티스의 최소 배포 단위

Jenkins OS 재시작 후 자동실행 설정

```
sudo vim /etc/systemd/system/docker-jenkins.service
```

위 파일 속에 넣기

```
[Unit]
Description=docker-jenkins
Wants=docker.service
After=docker.service

[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start jenkins
ExecStop=/usr/bin/docker stop jenkins

[Install]
WantedBy=multi-user.target
```

docker service 활성화

```
sudo systemctl enable docker
```

docker service 시작

```
sudo systemctl start docker
```

docker-jenkins service 활성화

```
sudo systemctl enable docker-jenkins.service
```

docker-jenkins service 시작

```
sudo systemctl start docker-jenkins.service
```

Docker jenkins 에서 Host Docker 접근 권한 부여

```
docker exec -it jenkins /bin/bash

groupadd -f docker
usermod -aG docker jenkins
chown root:docker /var/run/docker.sock
```

Frontend Pipeline

```
pipeline {
    agent any
    tools {nodejs "nodejs-20.12.2"}
    environment {
        imageName = "coderyard/moomul-fe" // docker hub
        registryCredential = 'moomul-token' // docker h
        dockerImage = ''
        releaseServerAccount = 'ubuntu' // ssh 연결 시 사
        releaseServerIPAddr = '172.26.5.75' // 서버 ip a
        releaseServerUri = 'moomul.kr' // 서비스 url
        containerName = 'moomul-fe' // 컨테이너 이름
    }
    stages {
```

```

        stage('Git Clone') { // 프로젝트 소스파일 clone
            steps {
                git branch: 'FE/develop',
                    credentialsId: 'seungwoo-gitlab', // GitLab Access Token
                    url: 'https://lab.ssafy.com/s10-final/S10P31A107.git' // clone 주소
            }
        }

        stage('.env copy') {
            steps {
                sh "mkdir -p ./client"
                // 경로가 없다면 생성

                sh "cp -f ../env/k8s/.env ./client/.env"; // env 설정파일 복사
                sh "cp -f ../env/k8s/Dockerfile-fe ./client/Dockerfile"; // Dockerfile 설정파일 복사
                sh "cp -f ../env/k8s/nginx.conf ./client/nginx.conf"; // nginx.conf 설정파일 복사
            }
        }

        stage('Node Build') { // 프로젝트 build
            steps {
                dir ('./client') {
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }

        stage('Image Build & DockerHub Push') { // 빌드된 파일 도커 이미지화 & 도커허브로 업로드
            steps {
                dir('./client') {
                    script {
                        docker.withRegistry('', registryCredential) {
                            sh "docker buildx create --use --name mybuilder"
                        }
                    }
                }
            }
        }
    }
}

```



```

        sh "pwd"
        sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:$BUILD_NUMBER --push ."
        sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:latest --push ."
    }
}
}
}

stage('DockerHub Pull') { // docker 이미지 가져옴
    steps {
        sshagent(credentials: ['ubuntu-a107']) {
            sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServerUri 'sudo docker pull $imageName:latest'"
        }
    }
}

stage('Service Start') { // 쿠버네티스 배포
    steps {
        withCredentials([
            $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'eks-admin',
            accessKeyVariable: 'AWS_ACCESS_KEY_ID',
            secretKeyVariable: 'AWS_SECRET_ACCESS_KEY'
        ]) {
            sh '''
                kubectl rollout restart deployment/moomul-fe-deployment -n dev
            '''
        }
    }
}

```

```

    }
    stage('Check Service') {
    steps {
        withCredentials([[
            $class: 'AmazonWebServicesCredentialsBi
nding',
            credentialsId: 'eks-admin',
            accessKeyVariable: 'AWS_ACCESS_KEY_ID',
            secretKeyVariable: 'AWS_SECRET_ACCESS_K
EY'
        ]]) {
            // 배포된 서비스 확인
            sh "kubectl get svc -n dev"
        }
    }
}
}
}

```



EKS를 이용해 파드들을 배포한다. 배포 시엔 AWS IAM 권한이 필요하다.

Backend Pipeline

```

pipeline {
    agent any

    environment {
        imageName = "coderyard/moomul-be" // docker 허브에 등
        록할 jar파일 이미지 이름
        registryCredential = 'moomul-token' // docker 허브 c
        redential 키
    }
}

```

```

    dockerImage = ''

    containerName = 'moomul-be' // 서버에 등록될 container
이름

    releaseServerAccount = 'ubuntu' // ssh로 서버 접
속 시 사용 할 사용자 이름
    releaseServerUri = 'moomul.kr' // 서버 도메인
    releaseServerIPAddr = '172.26.5.75' // 서버 ip addre
ss
    releasePort = '8089'
    containerPort = '8085' // container 포트포워딩 정보
}

    stages {
        stage('Git Clone') { // 프로젝트를 git clone
            steps {
                git branch: 'BE/develop',
                credentialsId: 'seungwoo-gitlab', // GitLab
Access Token
                url: 'https://lab.ssafy.com/s10-final/S10P3
1A107.git' // clone 주소
            }
        }

        stage('application.yml copy') { // 따로 관리 중인 설정
파일을 프로젝트 내로 복사
            steps {
                script {
                    // Jenkins 작업 공간 내에서의 경로로 변경
                    sh "mkdir -p ./server/src/main/resource
s" // 경로가 없다면 생성
                    sh "cp -f ../env/application.yml ./serv
er/src/main/resources/application.yml"; // 설정파일 복사
                    sh "cp -f ../env/Dockerfile-be ./serve
r/Dockerfile"; // 설정파일 복사
                    sh "cp -f ../env/bedev.env ./server/";
// 설정파일 복사

```

```

    }
  }
}

stage('rm integration test') { // 따로 관리 중인 설정
  파일을 프로젝트 내로 복사
  steps {
    script {
      // Jenkins 작업 공간 내에서의 경로로 변경
      sh "rm -r ./server/src/test"
    }
  }
}

stage('Jar Build') { // 프로젝트 파일 빌드
  steps {
    dir ('./server') {
      sh 'chmod +x ./gradlew'
      sh './gradlew clean build'
    }
  }
}

stage('Image Build & DockerHub Push') { // 빌드된 파일
  도커 이미지화 & 도커허브로 업로드
  steps {
    dir('./server') {
      script {
        docker.withRegistry('', registryCredential) {
          sh "docker buildx create --use
          --name mybuilder"
          sh "pwd"
          sh "docker buildx build --platform linux/amd64,linux/arm64 -t $imageName:$BUILD_NUMBER --push ."
          sh "docker buildx build --platform

```

```

    from linux/amd64,linux/arm64 -t $imageName:latest --push ."
    }
  }
}

stage('Before Service Stop') { // 서비스를 다시 컨테이
너로 가져오기 전, 기존 컨테이너 삭제
  steps {
    sshagent(credentials: ['ubuntu-a107']) {
      sh '''
        if test "`ssh -o StrictHostKeyChecking=
no $releaseServerAccount@$releaseServerUri "docker ps -aq -
-filter ancestor=$imageName:latest"`"; then
          ssh -o StrictHostKeyChecking=no $releas
eServerAccount@$releaseServerUri "docker stop $(docker ps -
aq --filter ancestor=$imageName:latest)"
          ssh -o StrictHostKeyChecking=no $releas
eServerAccount@$releaseServerUri "docker rm -f $(docker ps
-aq --filter ancestor=$imageName:latest)"
          ssh -o StrictHostKeyChecking=no $releas
eServerAccount@$releaseServerUri "docker rmi $imageName:lat
est"
        fi
      '''
    }
  }
}

stage('DockerHub Pull') { // docker 이미지 가져옴
  steps {
    sshagent(credentials: ['ubuntu-a107']) {
      sh "ssh -o StrictHostKeyChecking=no $re
leaseServerAccount@$releaseServerUri 'sudo docker pull $ima
geName:latest'"
    }
  }
}

```

```

    }

    stage('Service Start') { // docker 컨테이너 만들고 실행
        steps {
            sshagent(credentials: ['ubuntu-a107']) {
                sh '''
                    ssh -o StrictHostKeyChecking=no $r
eleaseServerAccount@$releaseServerUri "sudo docker run -i -
e TZ=Asia/Seoul -e SPRING_PROFILES_ACTIVE=prod --name $cont
ainerName -p $releasePort:$containerPort --env-file /.env -
d $imageName:latest"
                '''
            }
        }
    }

    stage('Service Check') { // 연결 체크
        steps {
            sshagent(credentials: ['ubuntu-a107']) {
                sh '''
                    #!/bin/bash

                    for retry_count in \$(seq 20)
                    do
                        if curl -s "http://$releaseServer
Uri:$releasePort" > /dev/null
                        then
                            break
                        fi
                        echo "The server is not alive ye
t. Retry health check in 5 seconds..."
                        sleep 5
                    done
                '''
            }
        }
    }
}

```

```

    post { // 빌드 성공 여부 MM으로 전송
        success {
            script {
                def Author_ID = sh(script: "git show -s --p
retty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s -
-pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'good',
                    message: "백엔드 서버 빌드 성공: ${env.JOB_NAM
E} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<
${env.BUILD_URL}|Details>)",
                    endpoint: 'https://meeting.ssafy.com/hooks/
rdtfbZR66bn1xcjgd53m87hgbe',
                    channel: 'jenkins'
                )
            }
        }
        failure {
            script {
                def Author_ID = sh(script: "git show -s --p
retty=%an", returnStdout: true).trim()
                def Author_Name = sh(script: "git show -s -
-pretty=%ae", returnStdout: true).trim()
                mattermostSend (color: 'danger',
                    message: "백엔드 서버 빌드 실패: ${env.JOB_NAM
E} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<
${env.BUILD_URL}|Details>)",
                    endpoint: 'https://meeting.ssafy.com/hooks/
rdtfbZR66bn1xcjgd53m87hgbe',
                    channel: 'jenkins'
                )
            }
        }
    }
}

```

MySQL 설치

- mysql latest 버전 설치

```
sudo docker pull mysql:latest
```

- root 비밀번호 설정 후 진행

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=<설정할 비밀번호> -e MYSQL_CHARSET=utf8 -e MYSQL_COLLATION=utf8_general_ci -v /path/to/my.cnf:/etc/mysql/conf.d/my.cnf -d -p 3306:3306 mysql:latest
```

Redis 설치

- redis alpine 버전 설치

```
sudo docker pull redis:alpine
```

- 컨테이너 실행

```
docker run --name redis --net host -v /data/redis:/data -d redis:alpine
```

도커 관리 시 Command

도커 컨테이너의 로그 보기

```
docker logs -f <container-name>
```

도커 컨테이너 접속

```
docker exec -it <container-name> /bin/bash # /bin/bash가 안  
된다면 /bin/sh or sh
```

서버 포트 개방

BackEnd

- Spring Boot Application : 8089:8080

FrontEnd

- AWS ELB에 연결

NginX (서버)

- 백엔드 서버 : 443:8089

ufw 명령어

```
ufw status # 현재 개방되어 있는 포트 확인
```

```
ufw allow <port-number> # 접속 허용할 포트 번호
```

AWS EKS(Elastic Kubernetes Service)



AWS의 Kubernetes Managed Service, 쿠버네티스를 더 효율적으로 관리할 수 있게 해준다.

IAM 관리자 설정

앞으로 EKS를 이용할 때 사용할 IAM 권한을 부여받은 사용자를 생성

 AdministratorAccess	AWS 관리형 - 직무	직접
 AmazonEC2FullAccess	AWS 관리형	직접
 AmazonVPCFullAccess	AWS 관리형	직접
 AWSCloudFormationFullAccess	AWS 관리형	직접

다음과 같이 정책을 연결하고 추가적으로

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Custom 정책을 만들어 연결한다.

eksctl, kubectl, awscli 설치

| 터미널에서 EKS를 관리할 수 있도록 하는 툴

eksctl

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
```

kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
kubectl version --client
```

awscli

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

버전 확인

```
aws --version
kubectl version --client
eksctl version
```

VPC 설정

| aws management의 콘솔에서 VPC 선택

- 보안 그룹 설정
 - 클러스터로 들어오는 트래픽에 대한 포트를 열어줘야 한다.
- VPC를 생성 후 서브넷을 연결

VPC Flow log를 통해 접속 로그 확인

- s3 버킷을 생성 후 VPC Flow log를 저장하도록 설정한다.

버킷을 지정해 flow log가 저장되는 곳에 가보면 날짜별로 Flow log가 생성되어 있는데, 다운로드 받아서 확인하면 된다. 예시는 다음과 같다.

```
111644186771_vpc...
보기 최신 지우기 다시 로드 공유
172.31.9.243 32019 172.31.42.76 55884 4 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 55275 172.31.38.103 30011 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.23.77 27626 172.31.9.243 30987 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.78.3 20175 172.31.9.243 2086 2 6 REJECT i-0456a67a832607430 1716011060
1716011092
172.31.73.250 55705 172.31.9.243 59407 2 6 REJECT i-0456a67a832607430 1716011060
1716011092
172.31.109.202.125 56100 172.31.9.243 60177 2 6 REJECT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 40323 172.31.43.107 30012 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.42.76 64884 172.31.9.243 30987 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 21902 172.31.43.107 30012 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.28.3 443 172.31.9.243 45768 0 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 45768 172.31.28.3 443 0 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 56162 172.31.43.107 30012 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 58277 172.31.38.103 30011 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.23.77 2604 172.31.9.243 32019 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 32019 172.31.23.77 2604 4 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.42.76 14348 172.31.9.243 32019 2 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.9.243 32019 172.31.42.76 14348 4 6 ACCEPT i-0456a67a832607430 1716011060
1716011092
172.31.3.155 63640 172.31.9.243 30987 2 6 ACCEPT i-0456a67a832607430 1716011060
```

Cluster 생성

Cluster IAM 역할 생성

myAmazonEKSClusterRole

정보

삭제

요약

편집

생성 날짜

May 02, 2024, 12:40 (UTC+09:00)

ARN

arn:aws:iam::111644186771:role/myAmazonEKSClusterRole

마지막 활동

30분 전

최대 세션 지속 시간

1시간

권한

신뢰 관계

태그

엑세스 관리자

세션 취소

권한 정책 (1) 정보

시뮬레이션

제거

권한 추가

최대 10개의 관리형 정책을 연결할 수 있습니다.

검색

모든 유형

< 1 >

정책 이름

유형

연결된 엔터티

+

AmazonEKSClusterPolicy

AWS 관리형

2

다음과 같이 클러스터의 역할을 지정해주는 권한 정책 연결

생성 페이지 (AWS Management Console)

포팅 매뉴얼

30

클러스터 구성 정보

이름

이 클러스터의 고유 이름을 입력합니다. 클러스터가 생성된 후에는 이 속성을 변경할 수 없습니다.

클러스터 이름은 문자 또는 숫자로 시작해야 하며 유니코드 문자 세트, 숫자, 하이픈 및 밑줄을 사용할 수 있습니다. 최대 길이는 100자입니다.

Kubernetes 버전 정보

이 클러스터의 Kubernetes 버전을 선택합니다.

i Kubernetes 1.29 버전은 2025년 3월 23일에 표준 지원이 종료됩니다. 해당 날짜 이전에 클러스터를 최신 버전으로 업데이트하지 않으면 자동으로 추가 지원이 시작됩니다. 추가 지원 미리보기가 종료된 후에는 추가 지원 중인 버전의 클러스터에 추가 요금이 부과됩니다. [자세히 알아보기](#)

클러스터 서비스 역할 정보

Kubernetes 컨트롤 플레인 사용자 대신하여 AWS 리소스를 관리하도록 허용하는 IAM 역할을 선택합니다. 클러스터가 생성된 후에는 이 속성을 변경할 수 없습니다. 새 역할을 생성하려면 [Amazon EKS 사용 설명서](#)의 지침을 따르세요.



Create a role in
IAM console

버전은 1.29버전 사용, 위에서 만든 서비스 역할 설정

클러스터 액세스 정보

IAM 보안 주체가 이 클러스터에 액세스하는 방법을 제어합니다.

부트스트랩 클러스터 관리자 액세스 정보

클러스터를 생성하는 IAM 보안 주체가 Kubernetes 클러스터 관리자 액세스 권한이 있는지 여부를 선택합니다.

☒ 클러스터 관리자 액세스 허용

IAM 보안 주체에 대한 클러스터 관리자 액세스를 허용합니다.

☐ 클러스터 관리자 액세스 허용 안 함

IAM 보안 주체에 대한 클러스터 관리자 액세스를 허용하지 않습니다.

클러스터 인증 모드 정보

클러스터가 인증된 IAM 보안 주체에 사용할 소스를 구성합니다.

☐ EKS API

클러스터가 인증된 IAM 보안 주체를 EKS 액세스 항목 API에서만 소싱합니다.

☒ EKS API 및 ConfigMap

클러스터가 인증된 IAM 보안 주체를 EKS 액세스 항목 API와 aws-auth ConfigMap 모두에서 소싱합니다.

☐ ConfigMap

클러스터가 인증된 IAM 보안 주체를 aws-auth ConfigMap에서만 소싱합니다.

네트워킹 정보

클러스터 생성 후에는 IP 주소 패밀리 및 서비스 IP 주소 범위를 변경할 수 없습니다.

VPC 정보

EKS 클러스터 리소스에 사용할 VPC를 선택합니다. 새 VPC를 생성하려면 [VPC 콘솔](#) (으)로 이동합니다.

vpc-0cc20370b22190193 기본값

서브넷 정보

클러스터와의 원활한 통신을 위해 제어 플레인 이 탄력적 네트워크 인터페이스(ENI)를 배치할 수 있는 서브넷을 VPC 내에서 선택합니다. 새 서브넷을 생성하려면 [VPC 콘솔](#)의 해당 페이지로 이동합니다.

서브넷 선택

subnet-0017c50403f1c7024
ap-northeast-2d 172.31.48.0/20

subnet-057c54710f007510
ap-northeast-2a 172.31.0.0/20

subnet-00f90c03b740c1f1
ap-northeast-2b 172.31.16.0/20

subnet-01e0d995cc079dab7
ap-northeast-2c 172.31.32.0/20

Clear selected
subnets

생성한 VPC와 서브넷들 선택

보안 그룹 | 정보

컨트롤 플레인 서브넷에서 생성된 EKS 관리형 탄력적 네트워크 인터페이스에 적용할 보안 그룹을 선택합니다. 새 보안 그룹을 생성하려면 [VPC 콘솔](#)의 해당 페이지로 이동합니다.

보안 그룹 선택



클러스터 IP 주소 패밀리 선택 | 정보

클러스터의 Pod 및 서비스에 대한 IP 주소 유형을 지정합니다.

☒ IPv4

☐ IPv6

☐ Kubernetes 서비스 IP 주소 범위 구성 | 정보

클러스터 서비스가 IP 주소를 수신할 범위를 지정합니다.

클러스터 엔드포인트 액세스 | 정보

Kubernetes API 서버 엔드포인트에 대한 액세스 권한을 구성합니다.

☐ 퍼블릭

VPC 외부에서 클러스터 엔드포인트에 액세스할 수 있습니다. 작업자 노드 트래픽은 엔드포인트에 연결하기 위해 VPC를 벗어납니다.

☒ 퍼블릭 및 프라이빗

VPC 외부에서 클러스터 엔드포인트에 액세스할 수 있습니다. 엔드포인트에 대한 작업자 노드 트래픽은 VPC 내에 유지됩니다.

☐ 프라이빗

클러스터 엔드포인트는 VPC를 통해서만 액세스할 수 있습니다. 엔드포인트에 대한 작업자 노드 트래픽은 VPC 내에 유지됩니다.

▶ 고급 설정

취소

이전

다음

보안 그룹 선택 후 클러스터 엔드포인트 액세스 → 퍼블릭 선택

<p>CoreDNS 정보 <input checked="" type="checkbox"/></p> <p>클러스터 내에서 서비스 검색을 활성화합니다.</p> <p>카테고리 networking</p> <p>🟢 기본적으로 설치됨</p>	<p>kube-proxy 정보 <input checked="" type="checkbox"/></p> <p>클러스터 내에서 포드 네트워킹을 활성화합니다.</p> <p>카테고리 networking</p> <p>🟢 기본적으로 설치됨</p>	<p>Amazon VPC CNI 정보 <input checked="" type="checkbox"/></p> <p>클러스터 내에서 포드 네트워킹을 활성화합니다.</p> <p>카테고리 networking</p> <p>🟢 기본적으로 설치됨</p>
<p>Amazon EKS Pod Identity 에이전트 정보 <input checked="" type="checkbox"/></p> <p>EKS Pod Identity 에이전트를 설치하고 EKS Pod Identity를 사용하여 Kubernetes 서비스 계정을 통해 포드에 AWS IAM 권한을 부여합니다.</p> <p>카테고리 security</p>	<p>Amazon GuardDuty EKS 런타임 모니터링 정보 <input type="checkbox"/></p> <p>클러스터 내에 EKS 런타임 모니터링 추가 기능을 설치합니다. Amazon GuardDuty 내에서 EKS 런타임 모니터링을 활성화해야 합니다.</p> <p>카테고리 security</p>	

Add on 플러그인 설치 페이지, 위와 같이 기본적으로 선택되어 있는 네 가지 선택

이후 버전이 나오는데, 다음 페이지를 참고해서 버전이 맞는 것을 찾아보자.

현재 프로젝트에선 `kubernetes v1.29` 사용

Amazon EKS 네트워킹 - Amazon EKS

이 문서는 Amazon Elastic Kubernetes Service(Amazon EKS)에 대한 공식 Amazon Web Services(AWS) 설명서입니다. Amazon EKS는 자체 Kubernetes 클러스터를 설치 및 운영할 필요 없이 AWS

 https://docs.aws.amazon.com/ko_kr/eks/latest/userguide/eks-networking.html



생성하게 되면 20분 정도 걸린다.

test-cluster

클러스터 정보

상태

생성 중

Kubernetes 버전

1.29

Support period

2025년 3월 23일까지 표준 지원

공급자

EKS

개요

리소스

컴퓨팅

네트워킹

추가 기능

액세스

관찰성

업그레이드 인사이트

업데이트 기록

태그

세부 정보

API 서버 엔드포인트

인증 기관

OpenID Connect 공급자 URL

클러스터 IAM 역할 ARN

생성됨

a few seconds ago

클러스터 ARN

arn:aws:eks:ap-northeast-2:111644186771:cluster/test-cluster

플랫폼 버전

OIDC 공급자 설정



Open ID Connect의 약자로, 계정의 AWS 리소스를 사용할 수 있는 권한을 부여

위의 클러스터 생성이 완료되면 OpenID Connect 공급자 URL이 뜰 것이다.

IAM - 자격 증명 공급자 생성

자격 증명 공급자 추가 정보

공급자 구성

공급자 유형 정보

☐ SAML

AWS 계정과 Shibboleth 또는 Active Directory Federation Services와 같은 SAML 2.0 호환 자격 증명 공급자 간에 신뢰를 설정합니다.

☒ OpenID Connect

AWS 계정과 Google 또는 Salesforce와 같은 자격 증명 공급자 서비스 간에 신뢰를 설정합니다.

공급자 URL

인증 요청에 대한 보안 OpenID Connect URL을 지정합니다.

최대 255자입니다. URL은 "https"로 시작해야 합니다.

대상 정보

앱의 자격 증명 공급자가 발행한 클라이언트 ID를 지정합니다.

최대 255자입니다. 영숫자 또는 '._-/ ' 문자를 사용하세요.

위에서 확인한 URL을 공급자 URL에 넣는다.

아래 칸엔 다음과 같이 기입한다.

`sts.amazonaws.com`

추후 역할 생성시 필요하므로 반드시 등록해놓는다.

Node Group 권한 설정



쿠버네티스의 파드는 최소 배포 단위라고 하였다. 파드는 노드에서 관리된다.
(Worker Node)

역할 생성 시 웹 자격 증명에 이전 단계에서 등록한 OIDC 공급자를 넣는다.

신뢰할 수 있는 엔터티 선택
정보

신뢰할 수 있는 엔터티 유형

☐ AWS 서비스
EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.

☐ AWS 계정
사용자 또는 서드 파티에 속한 다른 AWS 계정의 엔터티가 이 계정에서 작업을 수행하도록 허용합니다.

☒ 웹 자격 증명
지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.

☐ SAML 2.0 연동
기업 디렉터리에서 SAML 2.0과 연동된 사용자가 이 계정에서 작업을 수행할 수 있도록 허용합니다.

☐ 사용자 지정 신뢰 정책
다른 사용자가 이 계정에서 작업을 수행할 수 있도록 사용자 지정 신뢰 정책을 생성합니다.

웹 자격 증명

지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.

자격 증명 공급자

공급자 유형 선택

새로 생성

Node Group 권한을 생성할 때 다음과 같은 권한 정책을 기입한다.

권한 정책 (3) 정보

시뮬레이션
제거
권한 추가

최대 10개의 관리형 정책을 연결할 수 있습니다.

검색

필터링 기준 유형
모든 유형

<input type="checkbox"/>	정책 이름	유형	연결된 엔터티
<input type="checkbox"/>	AmazonEC2ContainerRegistryRea...	AWS 관리형	2
<input type="checkbox"/>	AmazonEKS_CNI_Policy	AWS 관리형	5
<input type="checkbox"/>	AmazonEKSWorkerNodePolicy	AWS 관리형	3

그리고, 역할 생성 후 신뢰관계를 다음과 같이 편집한다.



Node는 EC2 인스턴스로 생성되기 때문에, 관련 권한을 넣어주는 것이다.

Node Group 생성

EKS 관리 페이지 - 컴퓨팅 - 노드 그룹 추가

Node group 구성

노드 그룹 구성

노드 그룹을 생성한 후에는 이러한 속성을 변경할 수 없습니다.

이름
이 노드 그룹에 대한 고유한 이름을 할당합니다.

Test-node-group

노드 그룹 이름은 문자 또는 숫자로 시작해야 하며 유니코드 문자 세트, 숫자, 하이픈 및 밑줄을 포함할 수 있습니다. 최대 길이는 63자입니다.

노드 IAM 역할 정보
노드에서 사용할 IAM 역할을 선택합니다. 새 역할을 생성하려면 [IAM 콘솔](#)(으)로 이동합니다.

myAmazonEKSNoderole ▼

↻

ⓘ 관리형 노드 그룹 삭제 시 서비스가 중단될 수 있기 때문에 선택한 역할은 자체 관리형 노드 그룹에서 사용하지 않아야 합니다.

[자세히 알아보기](#) [🔗](#)

이전 단계에서 생성한 Node group 역할을 넣는다.

컴퓨팅 및 조정 구성 설정

노드 그룹 컴퓨팅 구성

노드 그룹을 생성한 후에는 이러한 속성을 변경할 수 없습니다.

AMI 유형 정보

노드에 대한 EKS 최적화 Amazon Machine Image를 선택합니다.

Amazon Linux 2 (AL2_x86_64)

용량 유형

이 노드 그룹에 대한 용량 구매 옵션을 선택합니다.

On-Demand

인스턴스 유형 정보

이 노드 그룹에 대해 선호하는 인스턴스 유형을 선택합니다.

인스턴스 유형 입력

t3.medium

vCPU: 2 vCPUs Memory: 4 GiB Network: Up to 5 Gigabit Max ENI: 3 Max IPs: 18

디스크 크기

각 노드에 연결되는 EBS 볼륨의 크기를 선택합니다.

20

GiB

노드에 할당 될 On-demand 자원 유형을 선택한다.

On-demand란, 사용한 만큼 비용을 지불하는 것이다.

Node 개수 지정

노드 그룹 조정 구성

원하는 크기

그룹에서 처음에 시작할 노드 수를 설정합니다.

노드

원하는 노드 크기는 0보다 크거나 같아야 함

최소 크기

그룹에서 축소할 수 있는 최소 노드 수를 설정합니다.

노드

최소 노드 크기는 0보다 크거나 같아야 함

최대 크기

그룹에서 확장할 수 있는 최대 노드 수를 설정합니다.

노드

최대 노드 크기는 1보다 크거나 같아야 하며 최소 크기보다 작을 수 없음



원하는 크기의 노드 수, 최소, 최대 크기의 노드를 지정한다.

파드의 수가 너무 많아졌을 때 어떻게 관리할 지 지정한다.

네트워크 지정

노드 그룹 네트워크 구성

노드 그룹을 생성한 후에는 이러한 속성을 변경할 수 없습니다.

서브넷 정보

노드가 실행될 VPC의 서브넷을 지정합니다. 새 서브넷을 생성하려면 [VPC 콘솔](#)의 해당 페이지로 이동합니다.

서브넷 선택

subnet-02476531374137021 ✕

subnet-054f651342f823540 ✕

subnet-03f30a33b7c3e1f1 ✕

subnet-01cc4333cc373dab7 ✕



Clear selected
subnets

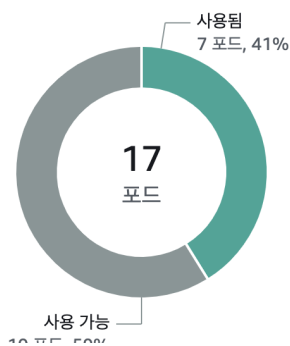
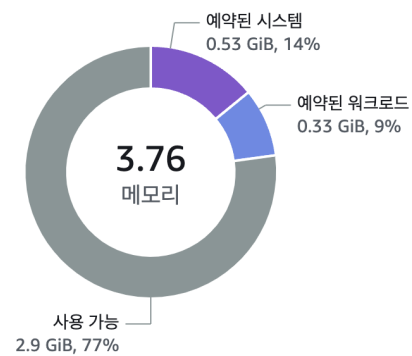
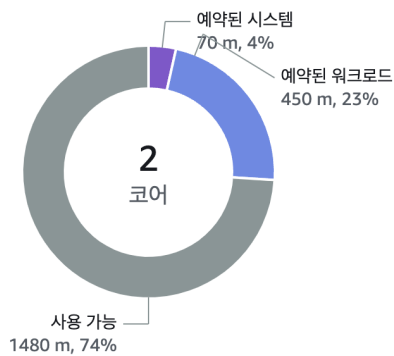
☐ 노드에 대한 원격 액세스 허용 정보

사용할 서브넷을 지정한다.

이후 생성 버튼을 누르면, 지정한 개수에 맞게 노드들이 생성되고 노드 그룹에 속할 것이다.

노드 자원 사용량 확인

노드 그룹에 속한 노드로 이동하면 다음과 같이 자원의 사용량을 판단할 수 있다.



AWS LoadBalancer

AWS LoadBalancer Controller 설치

- IAM OIDC 공급자와 Cluster 연결

```
eksctl utils associate-iam-oidc-provider --region <your-region> --cluster <your-cluster-name> --approve
```

- helm으로 AWS LoadBalancer Controller 설치

```
helm repo add eks https://aws.github.io/eks-charts

helm repo update

helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  --set clusterName=<your-cluster-name> \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  --namespace kube-system
```

LoadBalancer Role

다음과 같이 custom 관리 정책 생성한다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
    }
  ]
}
```

```

        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:AWSServiceName": "elasticloadbalancing.amazonaws.com"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeAccountAttributes",
            "ec2:DescribeAddresses",
            "ec2:DescribeAvailabilityZones",
            "ec2:DescribeInternetGateways",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcPeeringConnections",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeInstances",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeTags",
            "ec2:GetCoipPoolUsage",
            "ec2:DescribeCoipPools",
            "elasticloadbalancing:DescribeLoadBalancers",
            "elasticloadbalancing:DescribeLoadBalancerAttributes",
            "elasticloadbalancing:DescribeListeners",
            "elasticloadbalancing:DescribeListenerCertificates",
            "elasticloadbalancing:DescribeSSLPolicies",
            "elasticloadbalancing:DescribeRules",
            "elasticloadbalancing:DescribeTargetGroups",
            "elasticloadbalancing:DescribeTargetGroupAttributes",
            "elasticloadbalancing:DescribeTargetHealth"
        ]
    }
]

```

```

h",
    "elasticloadbalancing:DescribeTags",
    "elasticloadbalancing:DescribeTrustStores"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "cognito-idp:DescribeUserPoolClient",
    "acm:ListCertificates",
    "acm:DescribeCertificate",
    "iam:ListServerCertificates",
    "iam:GetServerCertificate",
    "waf-regional:GetWebACL",
    "waf-regional:GetWebACLForResource",
    "waf-regional:AssociateWebACL",
    "waf-regional:DisassociateWebACL",
    "wafv2:GetWebACL",
    "wafv2:GetWebACLForResource",
    "wafv2:AssociateWebACL",
    "wafv2:DisassociateWebACL",
    "shield:GetSubscriptionState",
    "shield:DescribeProtection",
    "shield:CreateProtection",
    "shield>DeleteProtection"
  ],
  "Resource": "*"
},
{
  "Effect": "Allow",
  "Action": [
    "ec2:AuthorizeSecurityGroupIngress",
    "ec2:RevokeSecurityGroupIngress"
  ],
  "Resource": "*"
},
{

```

```

        "Effect": "Allow",
        "Action": [
            "ec2:CreateSecurityGroup"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateTags"
        ],
        "Resource": "arn:aws:ec2:*:*:security-group/*",
        "Condition": {
            "StringEquals": {
                "ec2:CreateAction": "CreateSecurityGroup"
            },
            "Null": {
                "aws:RequestTag/elbv2.k8s.aws/cluster":
                    "false"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateTags",
            "ec2>DeleteTags"
        ],
        "Resource": "arn:aws:ec2:*:*:security-group/*",
        "Condition": {
            "Null": {
                "aws:RequestTag/elbv2.k8s.aws/cluster":
                    "true",
                "aws:ResourceTag/elbv2.k8s.aws/cluster": "false"
            }
        }
    }
}

```

```

    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:RevokeSecurityGroupIngress",
        "ec2>DeleteSecurityGroup"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:ResourceTag/elbv2.k8s.aws/cluste
r": "false"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing:CreateTargetGroup"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:RequestTag/elbv2.k8s.aws/cluster":
"false"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:CreateListener",
        "elasticloadbalancing>DeleteListener",
        "elasticloadbalancing:CreateRule",
        "elasticloadbalancing>DeleteRule"
      ],

```

```

        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:AddTags",
            "elasticloadbalancing:RemoveTags"
        ],
        "Resource": [
            "arn:aws:elasticloadbalancing:*:*:targetgroup/*/*/*",
            "arn:aws:elasticloadbalancing:*:*:loadbalancer/net/*/*/*",
            "arn:aws:elasticloadbalancing:*:*:loadbalancer/app/*/*/*"
        ],
        "Condition": {
            "Null": {
                "aws:RequestTag/elbv2.k8s.aws/cluster": "true",
                "aws:ResourceTag/elbv2.k8s.aws/cluster": "false"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:AddTags",
            "elasticloadbalancing:RemoveTags"
        ],
        "Resource": [
            "arn:aws:elasticloadbalancing:*:*:listener/net/*/*/*/*",
            "arn:aws:elasticloadbalancing:*:*:listener/app/*/*/*/*",
            "arn:aws:elasticloadbalancing:*:*:listener-rule/net/*/*/*/*",

```

```

        "arn:aws:elasticloadbalancing:*:*:listener-
rule/app/*/*/*"
    ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:ModifyLoadBalancerAtt
ributes",
            "elasticloadbalancing:SetIpAddressType",
            "elasticloadbalancing:SetSecurityGroups",
            "elasticloadbalancing:SetSubnets",
            "elasticloadbalancing>DeleteLoadBalancer",
            "elasticloadbalancing:ModifyTargetGroup",
            "elasticloadbalancing:ModifyTargetGroupAttr
ibutes",
            "elasticloadbalancing>DeleteTargetGroup"
        ],
        "Resource": "*",
        "Condition": {
            "Null": {
                "aws:ResourceTag/elbv2.k8s.aws/cluste
r": "false"
            }
        }
    },
    {
        "Effect": "Allow",
        "Action": [
            "elasticloadbalancing:AddTags"
        ],
        "Resource": [
            "arn:aws:elasticloadbalancing:*:*:targetgro
up/*/*",
            "arn:aws:elasticloadbalancing:*:*:loadbalan
cer/net/*/*",
            "arn:aws:elasticloadbalancing:*:*:loadbalan
cer/app/*/*"
        ]
    }
]

```



```

    ],
    "Condition": {
      "StringEquals": {
        "elasticloadbalancing:CreateAction": [
          "CreateTargetGroup",
          "CreateLoadBalancer"
        ]
      },
      "Null": {
        "aws:RequestTag/elbv2.k8s.aws/cluster":
"false"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "elasticloadbalancing:RegisterTargets",
      "elasticloadbalancing:DeregisterTargets"
    ],
    "Resource": "arn:aws:elasticloadbalancing:*:*:t
argetgroup/*/*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "elasticloadbalancing:SetWebAcl",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:AddListenerCertificat
es",
      "elasticloadbalancing:RemoveListenerCertifi
cates",
      "elasticloadbalancing:ModifyRule"
    ],
    "Resource": "*"
  }
]
}

```

해당 정책을 넣어 IAM 역할 생성 후, 다음과 같이 신뢰관계를 편집한다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "<OIDC 공급자 arn>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<OIDC 공급자>:sub": "system:serviceaccount:kube-system:aws-load-balancer-controller",
          "<OIDC 공급자>:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

이후, 생성된 로드밸런서에 해당 역할을 부여해주면 된다.

LoadBalancer 설정

| EC2 - 로드 밸런서 이동

AWS LoadBalancer Controller 설치 후 아래의 Ingress 설정 파일 배포 시 콘솔에 로드 밸런서가 보일 것이다.

Listener 규칙 설정

- 80 포트에서 트래픽 받도록 설정

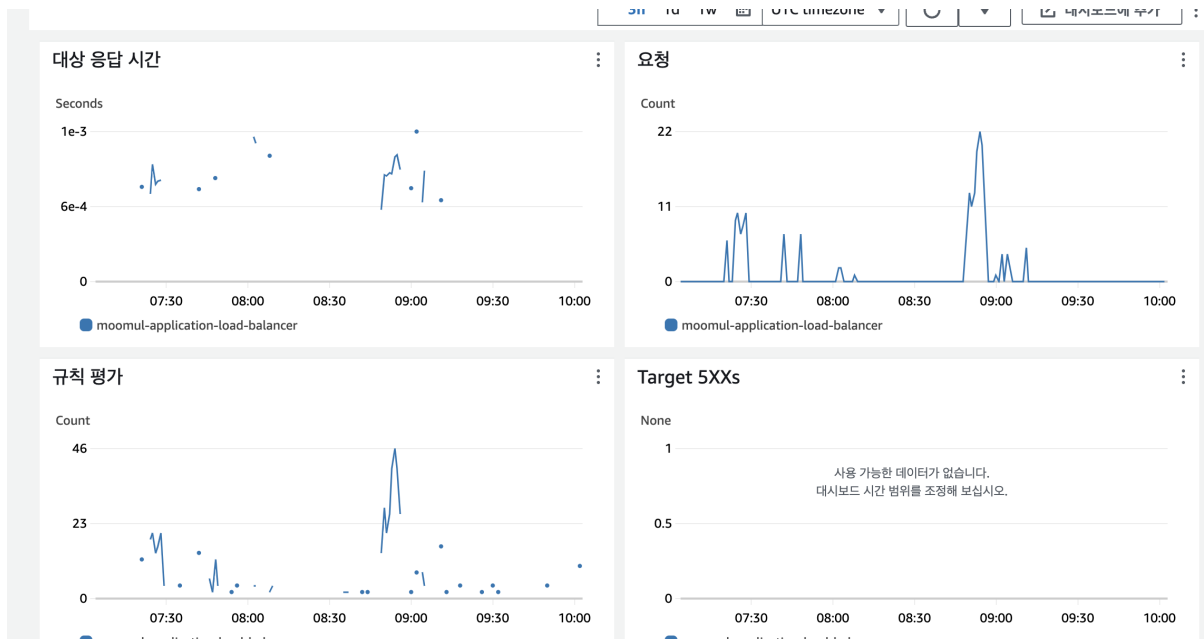
<input type="checkbox"/>	이름 태그	우선 순위	조건(인 경우)	작업(다음 수행)	ARN
규칙 1 우선 순위가 있는 규칙					
<input type="checkbox"/>	-	1	<ul style="list-style-type: none"> • HTTP 호스트 헤더는 moomul-application-load-balancer-1172084242.ap-northeast-2.elb.amazonaws.com, AND • 경로 패턴은 /api 또는 /api/*입니다 	대상 그룹으로 전달 <ul style="list-style-type: none"> • k8s-dev-moomulbe-0c6c65555b: 1 (100%) • 그룹 수준 고정성: 끄 	
<input type="checkbox"/>	-	2	<ul style="list-style-type: none"> • HTTP 호스트 헤더는 moomul-application-load-balancer-1172084242.ap-northeast-2.elb.amazonaws.com, AND • 경로 패턴은 /*입니다 	대상 그룹으로 전달 <ul style="list-style-type: none"> • k8s-dev-moomulfe-cee6a77172: 1 (100%) • 그룹 수준 고정성: 끄 	
<input type="checkbox"/>	기본값	마지막(기본값)	다른 규칙이 적용되지 않는 경우	고정 응답 반환 <ul style="list-style-type: none"> • 응답 코드: 404 • 응답 본문 • 응답 콘텐츠 유형: text/plain 	

이와 같이 구성할 수 있으며, 구체적인 순으로 우선 순위가 결정된다.

대상 그룹엔 이전에 생성했던 노드들에 각각의 레이블을 지정해줬는데, Frontend, Backend로 구분해 빠져나갈 수 있도록 하였다.

이후 보안 그룹을 설정해 원하는 포트로 트래픽을 받을 수 있게 한다.

모니터링



Kubernetes 설정 파일

Deployment.yaml

```

apiVersion: apps/v1 # k8s 버전 정의
kind: Deployment # 정의하려는 k8s 리소스의 유형 지정
metadata: # 리소스의 메타 데이터 지정
  name: moomul-fe-deployment # deployment 리소스의 이름을 지정
spec: # 리소스의 세부 사항 지정
  replicas: 2 # Deployment 리소스가 생성할 파드의 개수 지정
  strategy:
    type: RollingUpdate # RollingUpdate 파드 교체 방식
    rollingUpdate:
      maxSurge: 25% # 동시에 업데이트 될 최대 파드 수
      maxUnavailable: 25% # 동시에 사용할 수 없는 파드의 최대 수
  selector: # Deployment 리소스가 관리할 파드를 선택하기 위한 라벨
    매치 선택터를 지정
    matchLabels:
      app: moomul-fe
  template: # Pod의 템플릿 지정
    metadata: # Pod 템플릿의 메타 데이터를 지정

```

```

labels:
  app: moomul-fe
spec: # 파드의 사양 지정
  containers: # 파드에 포함될 컨테이너 목록 정의
    - name: moomul-fe # 컨테이너의 이름을 지정
      image: coderyard/moomul-fe:latest # 컨테이너가 사용할 이미지 지정
      ports: # 컨테이너가 사용할 포트 지정
        - containerPort: 80
      resources:
        requests:
          memory: "64Mi"
          cpu: "250m"
        limits:
          memory: "128Mi"
          cpu: "500m"
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: role
                    operator: In
                    values:
                      - frontend

```

- 사용할 자원 지정 및 배포 방식 지정
- 어느 레이블의 노드에 할당할 지 affinity 지정



Rolling Update 방식을 사용, Kubernetes의 무중단 배포 방식

```
root@ip-172-26-5-75:/etc/nginx/sites-enabled# kubectl get pod -n dev
```

NAME	READY	STATUS	RESTARTS	AGE
moomul-be-deployment-5df7b9ff94-hlmwp	1/1	Running	0	23h
moomul-be-deployment-5df7b9ff94-pcqc	1/1	Running	0	23h
moomul-fe-deployment-6b5c77bbf4-89lqb	1/1	Running	0	30m
moomul-fe-deployment-7cd8b4f478-69bzj	0/1	ContainerCreating	0	2s
moomul-fe-deployment-7cd8b4f478-lf2j7	1/1	Running	0	6s
mysql-deployment-df4f7cf66-xf9r8	1/1	Running	0	116m
redis-deployment-65d8857849-pgzzw	1/1	Running	0	23h

하나씩 교체되며 업데이트 버전으로 변경된다.

```
kubectl apply -f <deployment.yaml name> -n <namespace-name>
```

Ingress.yaml

```
apiVersion: networking.k8s.io/v1 # Ingress의 리소스 버전 정의
kind: Ingress # k8s의 리소스 유형을 나타냄, 클러스터의 서비스에 대한
외부 액세스를 관리하는 API 객체로, 일반적으로 HTTP 이용
metadata: # Ingress 리소스의 메타 데이터
  name: moomul-ingress # Ingress 리소스의 이름 지정
  namespace: dev
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/load-balancer-name: moomul-application-load-balancer
    alb.ingress.kubernetes.io/target-type: instance
    alb.ingress.kubernetes.io/subnets: <설정된 subnet을 ', '로
구분하여 기입>
spec: # Ingress 리소스의 원하는 상태 지정
  ingressClassName: alb
  rules: # 들어오는 요청에 대한 라우팅 규칙 정의
    - host: <내 로드 밸런서의 DNS 이름> # 해당 라우팅 규칙이 적용되는
호스트 이름 지정
      http: # http에 특화된 라우팅 규칙 지정
        paths: # 경로 기반의 라우팅 규칙을 정의
```

```

- path: / # path "/"가 프론트엔드 서비스로 라우팅되어야 함을
지정
  pathType: Prefix # 경로의 일치 유형을 지정, 프리픽스 매칭
  으로 "/"로 시작하는 모든 경로가 일치
  backend: # 이 경로와 일치하는 요청이 전달될 백엔드 서비스를
지정
    service: # 백엔드 서비스 이름 지정, moomul-fe
      name: moomul-fe-service
    port: # 백엔드 서비스의 포트를 지정
      number: 80 # 80 포트로 받음

```

```
kubectl apply -f <ingress.yaml name> -n <namespace-name>
```

Service.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: moomul-fe-service
spec:
  type: NodePort
  selector:
    app: moomul-fe
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80

```

- 80번 포트로 포트 포워딩

```
kubectl apply -f <service.yaml name> -n <namespace-name>
```

AWSCLI 명령어

계정 확인

```
aws sts get-caller-identity
```

계정 등록

```
aws configure
```

Kubectl 명령어

배포

```
kubectl apply -f <setting-file-name.yaml> -n <namespace-name>
```

삭제

```
kubectl delete -f <setting-file-name.yaml> -n <namespace-name>
```

로그

```
kubectl logs -f <pod-name> -n <namespace-name> -- /bin/bash
```


배포된 설정 파일 보기

```
kubectl get <category> -n <namespace-name>
# pod, svc(service), deployment, pv(persistence volume), pv
c(pv claim), ingress 등 확인 가능
```

이벤트 로그

```
kubectl get events -n <namespace-name>
```

프로젝트 설정 파일

Backend

- application.yml

```
spring:
  datasource:
    driver-class-name: ${DB_DRIVER}
    url: ${DB_URL}
    username: ${DB_USERNAME}
    password: ${DB_PASSWORD}
  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        default_batch_fetch_size: 100
  data:
    redis:
      host: ${REDIS_HOST}
```

```

    port: ${REDIS_PORT}
  servlet:
    multipart:
      max-file-size: 100MB
      max-request-size: 100MB
  logging:
    level:
      root: info
  cloud:
    aws:
      credentials:
        access-key: ${S3_ACCESS_KEY}
        secret-key: ${S3_SECRET_KEY}
      s3:
        bucket: ${S3_BUCKET}
      region:
        static: ${S3_REGION}
        auto: false
      stack:
        auto: false
  decorator:
    datasource:
      p6spy:
        enable-logging: true
  server:
    port: 8085

```

Frontend

- .env

```

REACT_APP_API_SERVER=https://moomul.kr/api

```