



A506 포팅메뉴얼

구축 환경

서비스 아키텍처

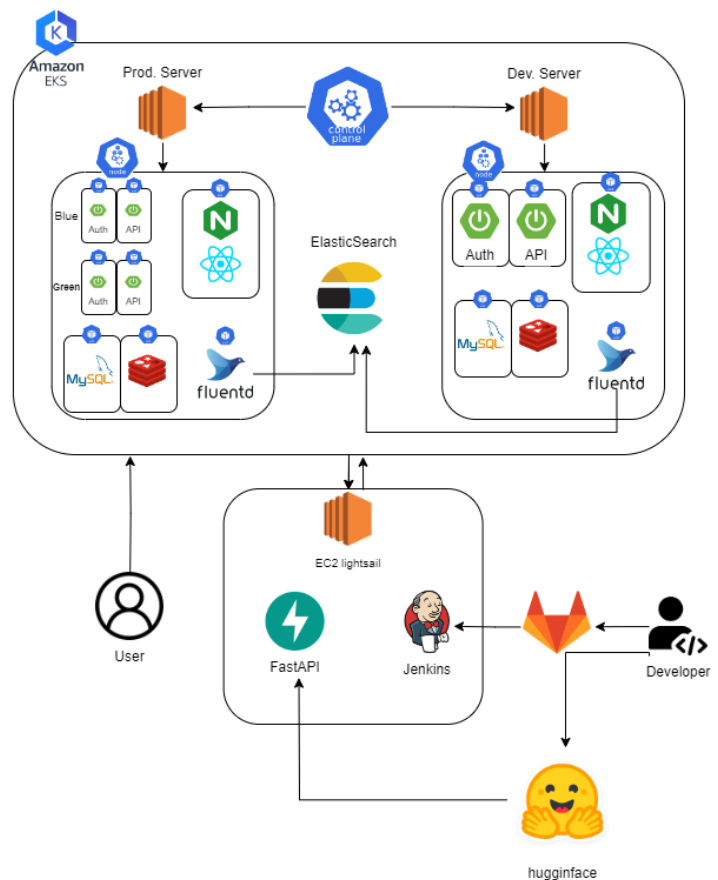
- 공통
 - fluentd (노드 데몬셋)
 - elasticsearch
 - kibana

노드1

- 개발서버
 - 인증인가 서버
 - api 서버
- 개발 db
 - mysql
 - redis

노드 2

- 프론트 서버
 - nginx
- 배포서버
 - 인증인가 서버
 - api 서버
 - blue
 - green



- 배포 db
 - mysql
 - redis

lightsail

- 모델 추론 서버
- Jenkins
- 판례 크롤링 조회 서버
- kibana, elasticsearch ?

▼ kubectl 주요 명령어 모음

- kube 모든 리소스 조회

```
kubectl get all -n <namespace>
```

- kube 리소스 조회

```
kubectl get {리소스 종류} {리소스 이름} -n {네임스페이스}
```

-n 말고 --all-namespaces 붙이면 모든 네임스페이스 조회

- 파드 describe

```
kubectl describe pod -l app=${앱 이름} -n {네임스페이스}
```

- 파드 로그 조회

```
kubectl logs -f [파드 이름] -n {네임스페이스}
```

- kube 리소스 지우기

```
kubectl delete -n {네임스페이스 명} {리소스 종류} {리소스 이름}
```

- 네임스페이스 생성

```
kubectl create namespace {이름}
```

- 롤링업데이트 트리거

```
kubectl edit deployment [deployment_name]
```

- patch 명령

```
kubectl patch {리소스} {리소스명} -n {네임스페이스} -p "{내용}"
```

필요한 작업들

▼ Lightsail 초기 설정

- 시간 맞추기
- 미러서버
- 방화벽 풀기
- 도커 설치
- 젠킨스 이미지 띄우기
- AWS CLI 설치

```
sudo apt install awscli
aws --version
```

- 해당 명령어로 configuration 정보 입력 (AWS 액세스 키 발급 받기)

```
aws configure
# 1. AWS Access Key ID [None]:
# 2. AWS Secret Access Key [None]:
# 3. Default region name [None]: 가용영역에서 확인
# 4. Default output format [None]: json
```

- kubectl 설치

```
curl -LO "[https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)](https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"
```

- kubectl 쓰기권한추가

```
chmod +x ./kubectl
```

- kubectl 이동

```
sudo mv ./kubectl /usr/local/bin
```

- 설치확인

```
kubectl version
```

- eksctl 설치

- Aws EKS Clusters 를 생성하기 위해 eksctl을 설치한다.

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

- eksctl 이동

```
sudo mv /tmp/eksctl /usr/local/bin
```

- 설치 확인

```
eksctl version
```

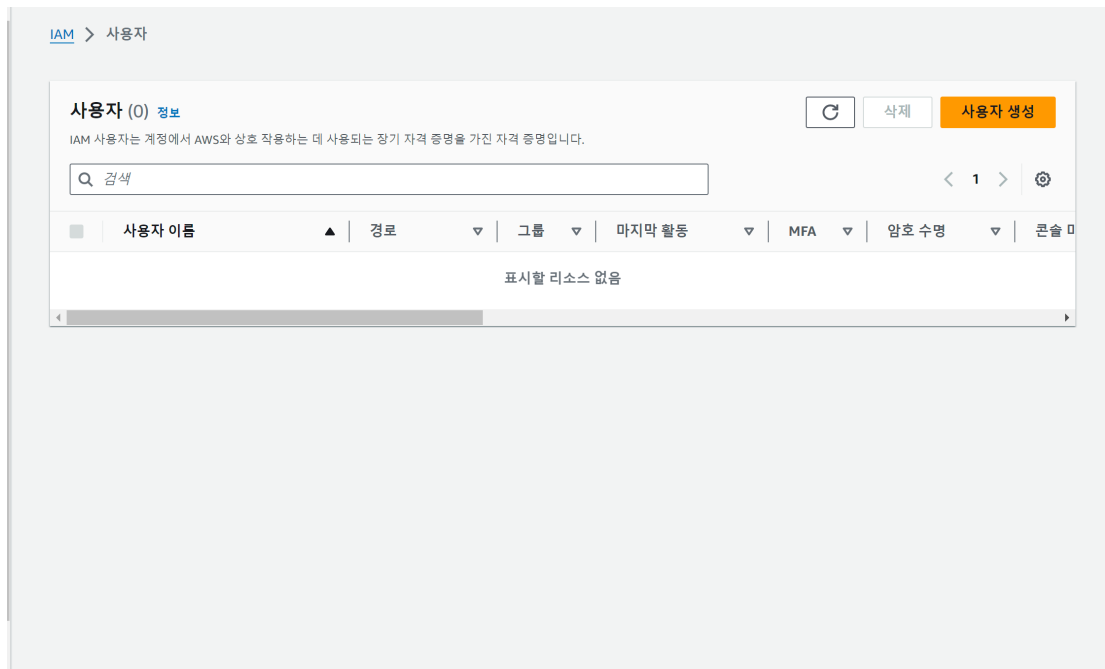
- eksctl 활용해서 EKS Cluster 생성

```
eksctl create --name 클러스터 이름 --version 버전 --region 지역명(한국은 ap-northeast-2) --nodegroup-name 노드그룹 이름 --node-type 노드타입 --nodes 노드개수 --nodes-min 지정
```

할 노드 최소개수 --nodes-max 지정할 노드 최대개수 --ssh-access --ssh-public-key 워커노드접속에 사용할 키 --managed

▼ AWS 액세스키 생성

- AWS IAM에서 사용자 탭에 접속하여 사용자 등록하기



- 사용자 그룹 생성 후 등록하기

사용자 그룹 생성

×

사용자 그룹을 생성하고 그룹에 연결할 정책을 선택합니다. 그룹을 사용하여 직무, AWS 서비스 액세스 또는 사용자 지정 권한별로 사용자 권한을 관리하는 것이 좋습니다. [자세히 알아보기](#)

사용자 그룹 이름
이 그룹을 식별하는 의미 있는 이름을 입력합니다.

aws-accesskey

최대 128자입니다. 영숫자 및 '+', '@', '-' 문자를 사용하세요.

권한 정책 (924)

↺

정책 생성 ↗

필터링 기준 유형

🔍 검색

모든 ... ▼

< 1 2 3 4 5 6 7 ... 47 >

⚙️

<input type="checkbox"/>	정책 이름 ↗	유형 ▼	다음... ▼	설명
<input type="checkbox"/>	AdministratorAccess	AWS 관리형 - ...	없음	Provides full access to AWS service
<input type="checkbox"/>	AdministratorAcce...	AWS 관리형	없음	Grants account administrative pern
<input type="checkbox"/>	AdministratorAcce...	AWS 관리형	없음	Grants account administrative pern
<input type="checkbox"/>	AlexaForBusinessD...	AWS 관리형	없음	Provide device setup access to Alex
<input type="checkbox"/>	AlexaForBusinessF...	AWS 관리형	없음	Grants full access to AlexaForBusin
<input type="checkbox"/>	AlexaForBusinessG...	AWS 관리형	없음	Provide gateway execution access t
<input type="checkbox"/>	AlexaForBusinessLi...	AWS 관리형	없음	Provide access to Lifesize AVS devic

취소

사용자 그룹 생성

- 생성된 사용자 눌러서 액세스 키 추가하기

skajd 정보

요약

ARN arn:aws:iam::654654554890:user/skajd	콘솔 액세스 비활성화된	액세스 키 1 액세스 키 만들기
생성됨 April 24, 2024, 09:28 (UTC+09:00)	마지막 콘솔 로그인 -	

권한

그룹

태그

보안 자격 증명

액세스 관리자

액세스 키 (0) 액세스 키 만들기

액세스 키를 사용하여 AWS CLI, AWS Tools for PowerShell, AWS SDK 또는 직접 AWS API 호출을 통해 AWS에 프로그래밍 방식 호출을 전송합니다. 한 번에 최대 두 개의 액세스 키(활성 또는 비활성)를 가질 수 있습니다. [자세히 알아보기](#)

액세스 키가 없습니다. 액세스 키와 같은 장기 보안 인증을 사용하지 않는 것이 모범 사례입니다. 대신 단기 보안 인증을 제공하는 도구를 사용하세요. [자세히 알아보기](#)

액세스 키 만들기

액세스 키

분실하거나 잊어버린 비밀 액세스 키는 검색할 수 없습니다. 대신 새 액세스 키를 생성하고 이전 키를 비활성화합니다.

액세스 키	비밀 액세스 키
AKIAZQ3DTYMFLUA2CAVA	[REDACTED] 숨기기

만들어서 따로 기록해둔다 !

- 이후 사용자 그룹 정책을 설정한다 (full access 로 설정함)

[IAM](#) > [사용자 그룹](#) > aws-accesskey

aws-accesskey 정보 삭제

요약 편집

사용자 그룹 이름 aws-accesskey	생성 시간 April 24, 2024, 09:27 (UTC+09:00)	ARN arn:aws:iam::654654554890:group/aws-accesskey
----------------------------	--	--

사용자 (1) **권한** 액세스 관리자

권한 정책 (1) 정보 시뮬레이션 제거 권한 추가 ▼

최대 10개의 관리형 정책을 연결할 수 있습니다.

필터링 기준 유형

검색 모든 유형 ▼ < 1 > ⚙

<input type="checkbox"/>	정책 이름 ?	유형 ▲ ▼	연결된 엔터티 ▼
<input type="checkbox"/>	AdministratorAccess	AWS 관리형 - 직무	1

▼ AWS CLI 설정

- EC2에서 `eksctl` 을 사용하기 위해 AWS CLI를 설치한다.
 - `sudo apt install awscli`
 - `aws --version`
- AWS CLI 설정

- AWS CLI 가 AWS 환경과 소통하고 인증할 수 있도록 설정
- `aws configure` 명령어를 입력하시르면 아래 내용을 입력해야한다.

```
1. AWS Access Key ID [None]:
2. AWS Secret Access Key [None]:
3. Default region name [None]: 입력 x
4. Default output format [None]: json
```

▼ Kubectl 설치

- kubectl 설치

```
curl -LO "https://dl.k8s.io/release/v1.29.0/bin/linux/amd64/kubectl"
```

- kubectl 쓰기권한추가

```
chmod +x ./kubectl
```

- kubectl 이동

```
sudo mv ./kubectl /usr/local/bin
```

- 설치확인

```
kubectl version
```

▼ eksctl 설치

- Aws EKS Clusters 를 생성하기 위해 eksctl을 설치한다.

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

- eksctl 이동


```
sudo mv /tmp/eksctl /usr/local/bin
```

- 설치 확인

```
eksctl version
```

▼ EKS

```
eksctl create --name 클러스터 이름 --version 버전 --region  
지역명(한국은 ap-northeast-2) --nodegroup-name 노드그룹이름 --  
node-type 노드타입 --nodes 노드개수 --nodes-min 지정할 노드 최  
소개수 --nodes-max 지정할 노드 최대개수 --ssh-access --ssh-pub  
lic-key 워커노드접속에 사용할 키 --managed
```

```
eksctl create cluster --name mycluster --region ap-north  
east-2 --nodegroup-namemygroup --node-type t2.xlarge --n  
odes 2 --nodes-min 2 --nodes-max 2 --managed
```

```
aws eks --region ap-northeast-2 update-kubeconfig --name  
mycluster
```

- EKS 서비스 어카운트 : Jenkins에서 EKS에 접근하기 위한 계정, 일종의 크레덴셜

▼ Jenkins 초기 설정 및 GitLab 연동

- 플러그인

```
# 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 빌드에 사용

Generic Webhook Trigger

타사 레포지토리 이용시 사용 (GitLab, Github 등)

GitLab

GitLab API

GitLab Authentication

Node.js 빌드시 사용

NodeJS

Jenkins Stage View (파이프라인 시각화)

Pipeline: Stage View Plugin

AWS 관련

AWS Global Configuration

AWS Credentials

AWS ECR

K8s


Kubernetes Plugin

Kubernetes CLI

▼ Jenkins AWS 연동

Jenkins Pipeline을 이용하여 Docker Image를 ECR로 Push

안녕하세요. 오늘 글은 지난 시간에 이어서 Jenkins Pipeline을 이용하여 Docker Image Build 후 AWS ECR로 Push 하는 방법에 대해서 다뤄볼까 합니다. 준비물 : AWS Accesss Key, AWS IAM(ECR

 <https://teichae.tistory.com/entry/Jenkins-Pipeline을-이용한-Docker-Image를-ECR로-Push>



푸시 명령 보기

- AWS IAM 생성 후 젠킨스 크레덴셜 등

New credentials

Kind
AWS Credentials

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

ID ?
aws-credential

Description ?

Access Key ID ?

Secret Access Key

Create

▼ FastApi 젠킨스 설정 + 파이프라인

```
pipeline {
    agent any

    stages {
        stage('github clone') {
            steps {
                git branch: 'back-ai-deploy',
                    credentialsId: 'gitlab-credential',
                    url: 'https://lab.ssafy.com/s10-final/s
            }
        }

        stage('Containers Restart') {
            steps {
                script {
                    sh 'docker-compose down'
                    sh 'docker-compose up --build -d'
                }
            }
        }
    }
}
```

▼ FastApi 도커 설정

- Dockerfile

```
FROM python:latest

WORKDIR /app/

COPY ./*.py /app/
COPY ./env /app/
COPY ./requirements.txt /app/

RUN pip install -r requirements.txt

CMD uvicorn --host=0.0.0.0 --port 8000 main:app
```

- 도커 명령어

```
docker build --tag inferServer .
```

```
docker run -d -p 8000:8000 --name inferServer inferServer
```

- Docker-Compose

```
version: "3.8"

services:
  app:
    build:
      context: ./
      dockerfile: Dockerfile
    container_name: inferServer
    ports:
      - "8000:8000"
    restart: on-failure
    volumes:
      - ./fastapi:/model
```

▼ 개발서버 쿠버네티스 매니페스트

▼ Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dev-ingress
  namespace: development
  annotations:
    kubernetes.io/ingress.class: "alb"
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP
S": 443}]'
    alb.ingress.kubernetes.io/certificate-arn: "arn:a
ws:acm:ap-northeast-2:654654554890:certificate/ec3eb1
f1-9f69-4a39-8bc6-79b9217110d9"
spec:
  tls:
    - hosts:
      - test.hellolaw.kr
  rules:
    - host: test.hellolaw.kr
    - http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: nginx-service
              port:
                number: 80
        - path: /api/
          pathType: Prefix
          backend:
            service:
              name: back-api-service
              port:
                number: 8082
```

```

- path: /auth/
  pathType: Prefix
  backend:
    service:
      name: back-auth-service
      port:
        number: 8099
- path: /kibana/
  pathType: Prefix
  backend:
    service:
      name: kibana
      port:
        number: 5601

```

▼ 프론트

▼ Service

```

apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: development
spec:
  type: ClusterIP
  selector:
    app: hellolaw-front
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80

```

▼ deployment

```

apiVersion: apps/v1
kind: Deployment
metadata:

```

```

name: hellolaw-front
namespace: development
spec:
  replicas: 1
  revisionHistoryLimit: 0 # 보유할 ReplicaSet의 최대
개수
  selector:
    matchLabels:
      app: hellolaw-front
  template:
    metadata:
      labels:
        app: hellolaw-front
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExe
cution:
          nodeSelectorTerms:
            - matchExpressions:
                - key: role
                  operator: In
                  values:
                    - development
      containers:
        - name: hellolaw-front
          image: 654654554890.dkr.ecr.ap-northeast-
2.amazonaws.com/hellolaw-front:latest
          ports:
            - containerPort: 3000

```

▼ 백 api

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellolaw-api

```

```

    namespace: development
spec:
  replicas: 1
  revisionHistoryLimit: 0
  selector:
    matchLabels:
      app: hellolaw-api
  template:
    metadata:
      labels:
        app: hellolaw-api
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: role
                    operator: In
                    values:
                      - development
      containers:
        - name: backend
          image: 654654554890.dkr.ecr.ap-northeast-2.amazonaws.com/hellolaw-api:dev
          imagePullPolicy: Always
          ports:
            - containerPort: 8082
          env:
            - name: SPRING_PROFILES_ACTIVE
              value: "dev"
            - name: MYSQL_HOST
              value: mysql-service
            - name: DB_USER
              value: hellolaw_dev
            - name: DB_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret

```



```

        key: password
    ---

apiVersion: v1
kind: Service
metadata:
  name: back-api-service
  namespace: development
spec:
  selector:
    app: hellolaw-api
  ports:
    - protocol: TCP
      port: 8082
      targetPort: 8082

```

▼ 백 auth

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellolaw-auth
  namespace: development
spec:
  replicas: 1
  revisionHistoryLimit: 0 # 보유할 ReplicaSet의 최대 개수
  selector:
    matchLabels:
      app: hellolaw-auth
  template:
    metadata:
      labels:
        app: hellolaw-auth
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:

```

```

        nodeSelectorTerms:
          - matchExpressions:
              - key: role
                operator: In
                values:
                  - development
containers:
- name: backend
  image: 654654554890.dkr.ecr.ap-northeast-2.amazonaws.com/hello-law:dev
  ports:
    - containerPort: 8099
  env:
    - name: SPRING_PROFILES_ACTIVE
      value: "dev"
    - name: MYSQL_HOST
      value: mysql-service
    - name: DB_USER
      value: hellolaw_dev
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysql-secret
          key: password
---

apiVersion: v1
kind: Service
metadata:
  name: back-auth-service
  namespace: development
spec:
  selector:
    app: hellolaw-auth
  ports:
    - protocol: TCP

```

```
port: 8099
targetPort: 8099
```

▼ 배포서버 쿠버네티스 매니페스트

▼ Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: prod-ingress
  namespace: production
  annotations:
    kubernetes.io/ingress.class: "alb"
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP
S": 443}]'
    alb.ingress.kubernetes.io/certificate-arn: "arn:a
ws:acm:ap-northeast-2:654654554890:certificate/ec3eb1
f1-9f69-4a39-8bc6-79b9217110d9"
spec:
  tls:
    - hosts:
      - hellolaw.kr
  rules:
    - host: hellolaw.kr
    - http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: nginx-service
              port:
                number: 80
        - path: /api/
          pathType: Prefix
          backend:
```

```

      service:
        name: back-api-service
        port:
          number: 8082
    - path: /auth/
      pathType: Prefix
      backend:
        service:
          name: back-auth-service
          port:
            number: 8099

```

▼ 프론트

▼ deployment

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellolaw-front
  namespace: production
spec:
  replicas: 1
  revisionHistoryLimit: 2 # 보유할 ReplicaSet의 최대
개수 <- 버전 롤백 위해서 여러개 설정
  selector:
    matchLabels:
      app: hellolaw-front
  template:
    metadata:
      labels:
        app: hellolaw-front
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:

```

```

        nodeSelectorTerms:
        - matchExpressions:
          - key: role
            operator: In
            values:
            - production
    containers:
    - name: hellolaw-front
      image: 654654554890.dkr.ecr.ap-northeast-2.amazonaws.com/hellolaw-front:latest
      imagePullPolicy: Always
      ports:
      - containerPort: 3000
  ---
  apiVersion: v1
  kind: Service
  metadata:
    name: nginx-service
    namespace: production
  spec:
    type: ClusterIP
    selector:
      app: hellolaw-front
    ports:
    - protocol: TCP
      port: 80
      targetPort: 80

```

▼ 백 api

▼ service

```

apiVersion: v1
kind: Service

```

```

metadata:
  name: back-api-service
  namespace: production
spec:
  selector:
    app: hellolaw-api
    color: blue
  ports:
    - protocol: TCP
      port: 8082
      targetPort: 8082

```

▼ blue

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellolaw-api-blue
  namespace: production
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hellolaw-api
      color: blue
  template:
    metadata:
      labels:
        app: hellolaw-api
        color: blue
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: role

```

```

        operator: In
        values:
          - production
    containers:
      - name: backend
        image: 654654554890.dkr.ecr.ap-northeast-2.amazonaws.com/hellolaw-back-api:latest
        imagePullPolicy: Always
        ports:
          - containerPort: 8082
        env:
          - name: SPRING_PROFILES_ACTIVE
            value: "prod"
          - name: MYSQL_HOST
            value: mysql-service
          - name: DB_USER
            value: hellolaw_prod
          - name: DB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: password

```

▼ green

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellolaw-api-green
  namespace: production
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hellolaw-api
      color: green
  template:
    metadata:

```

```

    labels:
      app: hellolaw-api
      color: green
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: role
                    operator: In
                    values:
                      - production
            containers:
              - name: backend
                image: 654654554890.dkr.ecr.ap-northeast-2.amazonaws.com/hellolaw-back-api:latest
                imagePullPolicy: Always
                ports:
                  - containerPort: 8082
                env:
                  - name: SPRING_PROFILES_ACTIVE
                    value: "prod"
                  - name: MYSQL_HOST
                    value: mysql-service
                  - name: DB_USER
                    value: hellolaw_prod
                  - name: DB_PASSWORD
                    valueFrom:
                      secretKeyRef:
                        name: mysql-secret
                        key: password

```

```

//blue-green.yml
spec:
  selector:

```



```
app: hellolaw-api
color: blue
```

▼ 백 auth

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellolaw-auth
  namespace: production
spec:
  replicas: 1
  revisionHistoryLimit: 0 # 보유할 ReplicaSet의 최대 개
수
  selector:
    matchLabels:
      app: hellolaw-auth
  template:
    metadata:
      labels:
        app: hellolaw-auth
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecut
ion:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: role
                    operator: In
                    values:
                      - production
      containers:
        - name: backend
          image: 654654554890.dkr.ecr.ap-northeast-2.am
azonaws.com/hellolaw-back-auth:latest
          imagePullPolicy: Always
          ports:
```

```

      - containerPort: 8099
    env:
      - name: SPRING_PROFILES_ACTIVE
        value: "prod"
      - name: MYSQL_HOST
        value: mysql-service
      - name: DB_USER
        value: hellolaw_prod
      - name: DB_PASSWORD
        valueFrom:
          secretKeyRef:
            name: mysql-secret
            key: password

---

apiVersion: v1
kind: Service
metadata:
  name: back-auth-service
  namespace: production
spec:
  selector:
    app: hellolaw-auth
  ports:
    - protocol: TCP
      port: 8099
      targetPort: 8099

```

▼ 프론트 CICD

쿠버네티스에서 nginx에 리액트 앱을 감싼 형태를 컨테이너 1개로 배포.

0. SCM

1. Dockerfile 이용해서 프론트 도커라이징

2. ECR에 푸쉬

3. kubectl로 해당 이미지 띄우기. (kubectl apply -f ~~)

▼ Dockerfile

```
FROM node:18-alpine as build

RUN npm install -g pnpm

WORKDIR /app

COPY package.json ./
COPY pnpm-lock.yaml ./

RUN pnpm install

COPY . .

RUN pnpm build

#####

FROM nginx:stable-alpine

RUN rm -rf /etc/nginx/conf.d/default.conf

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

COPY --from=build /app/dist /usr/share/nginx/html

COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 3000

CMD ["nginx", "-g", "daemon off;"]
```

▼ 파이프라인

```
pipeline {
    agent any
```

```

environment{
    REGION = 'ap-northeast-2'
    EKS_API = 'https://F73C5F6407E7C0DFD2AF34C773B41
    EKS_CLUSTER_NAME = 'mycluster'
    EKS_JENKINS_CREDENTIAL_ID = 'kubectl-credential'
    ECR_PATH = '654654554890.dkr.ecr.ap-northeast-2.
    IMAGE_NAME = 'hellolaw-front'
    AWS_CREDENTIAL_ID = 'aws-credential'

}
stages {
    stage('MMAalert-start') {
        steps {
            sh '''
                curl -d '{"text":"## Front-Dev Deploy
            '''
        }
    }
    stage('github clone') {
        steps {
            git branch: 'front-develop',
                credentialsId: 'gitlab-credential',
                url: 'https://lab.ssafy.com/s10-fina
        }
    }
    stage('Docker Build') {
        steps {
            script{
                sh '''
                    docker build -t ${IMAGE_NAME}:${E
                    docker tag ${IMAGE_NAME}:${BUILD_
                '''
            }
        }
    }
    stage('Push to ECR'){
        steps{

```

```

        script{
            docker.withRegistry("https://${ECR_F
                docker.image("${IMAGE_NAME}:${BU
                docker.image("${IMAGE_NAME}:late
            }
        }
    }

}

stage('deployment download'){
    steps{
        script{
            withCredentials([file(credentialsId:
                sh 'cp $deployment ./deployment.
            }
        }
    }
}

stage('Deploy to k8s'){
    steps{
        script{
            sh """
                sed -i 's|latest|${BUILD_NUMBER}
                kubectl apply -f deployment.yml
            """
        }
    }
}

stage('Frontend Health Check') {
    steps {
        script {
            def attempts = 10
            def healthCheckPassed = false

            for (int i = 0; i < attempts; i++) {

```


▼ 백 CICD

▼ Dockerfile

```
FROM openjdk:17-jdk-slim

WORKDIR /app

ARG JAR_FILE=build/libs/*.jar

COPY ${JAR_FILE} app.jar

EXPOSE 8082

ENTRYPOINT ["java", "-jar", "app.jar"]
```

- 파이프라인

▼ 개발서버 예시

```
pipeline {
    agent any

    environment{
        REGION = 'ap-northeast-2'
        EKS_API = 'https://F73C5F6407E7C0DFD2AF34C773
        EKS_CLUSTER_NAME = 'mycluster'
        EKS_JENKINS_CREDENTIAL_ID = 'kubect1-credenti
        ECR_PATH = '654654554890.dkr.ecr.ap-northeast
        IMAGE_NAME = 'hellolaw-back-auth'
        AWS_CREDENTIAL_ID = 'aws-credential'
    }

    stages {
        stage('MMAAlert-start') {
            steps {
                sh '''
                    curl -d '{"text":"## Back-Auth-Dev
                    '''
```

```

    }
  }
  stage('github clone') {
    steps {
      git branch: 'back-auth-develop',
        credentialsId: 'gitlab-credential',
        url: 'https://lab.ssafy.com/s10-f
    }
  }
  stage('application.yml download') {
    steps {
      withCredentials([file(credentialsId:
        script {
          dir('back-auth'){
            if (fileExists('src/main/
              sh 'rm src/main/resou
            }
            sh 'mkdir -p src/main/res
            sh 'cp $yamlfile src/main/
          }
        }
      ]
    }
    withCredentials([file(credentialsId:
      script {
        dir('back-auth'){
          if (fileExists('src/main/
            sh 'rm src/main/resou
          }
          sh 'mkdir -p src/main/res
          sh 'cp $yamlfile src/main/
        }
      ]
    }
  }
}
stage('Jar Build') {
  steps {
    dir('back-auth'){

```



```

        sh 'chmod +x ./gradlew'
        sh './gradlew clean bootJar'
        sh 'pwd'
    }
}
stage('Docker Build') {
    steps {
        script{
            dir('back-auth') {
                sh '''
                    docker build -t ${IMAGE_NAME}
                    docker tag ${IMAGE_NAME}:
                '''
            }
        }
    }
}
stage('Push to ECR'){
    steps{
        script{
            dir('back-auth'){
                docker.withRegistry("https://
                docker.image("${IMAGE_NAME}
                docker.image("${IMAGE_NAME}
            }
        }
    }
}

stage('deployment download'){
    steps{
        script{
            withCredentials([file(credentials
            sh 'cp $deployment ./deployme
        }
    }
}

```

```

    }
}

stage('Deploy to k8s'){
    steps{
        script{
            sh """
                sed -i 's|latest|${BUILD_NUMBER}|' deployment.yaml
                kubectl apply -f deployment.yaml
            """
        }
    }
}

stage('Health Check') {
    steps {
        script {
            def attempts = 10
            def healthCheckPassed = false

            for (int i = 0; i < attempts; i++) {
                // curl을 사용하여 헬스 체크 수행
                def response = sh script: "curl -s http://localhost:8080/health"
                if (response.contains("health check passed")) {
                    echo "Health check passed"
                    healthCheckPassed = true
                    sh '''
                        curl -d '{"text":"## Back to work"}' http://localhost:8080/health
                    '''
                    break
                }
                // Jenkins에서 슬립 함수 사용
                sleep time: 1, unit: 'SECONDS'
            }

            if (!healthCheckPassed) {
                echo "Health check failed"
            }
        }
    }
}

```

```
// Jenkins 파이프라인에서 에러 발생 시
sh '''
    curl -d '{"text":"## Back
    ...
error("Deployment failed due
}
}
}
}
}
}
```

▼ 배포서버 예시 1 (rollout)

```

pipeline {
    agent any

    environment{
        REGION = 'ap-northeast-2'
        EKS_API = 'https://F73C5F6407E7C0DFD2AF34C773'
        EKS_CLUSTER_NAME = 'mycluster'
        EKS_JENKINS_CREDENTIAL_ID = 'kubect1-credential'
        ECR_PATH = '654654554890.dkr.ecr.ap-northeast'
        IMAGE_NAME = 'hellolaw-back-auth'
        AWS_CREDENTIAL_ID = 'aws-credential'

    }

    stages {
        stage('MMAalert-start') {
            steps {
                sh '''
                    curl -d '{"text":"## Back-Auth-Proc
                    '''
            }
        }

        stage('deployment download'){
            steps{

```

```

        script{
            withCredentials([file(credentials
                sh 'cp $deployment ./deployme
            }
        }
    }
}

stage('Deploy to k8s'){
    steps{
        script{
            sh """
                kubectl apply -f deployment.y
                kubectl rollout restart deplc

            """
        }
    }
}

stage('Health Check') {
    steps {
        script {
            def attempts = 10
            def healthCheckPassed = false

            for (int i = 0; i < attempts; i++)
                // curl을 사용하여 헬스 체크 수행
                def response = sh script: "cu
                if (response.contains("health
                    echo "Health check passed
                    healthCheckPassed = true
                    sh '''
                    curl -d '{"text":"## Back
                    '''
                    break
                }
        }
    }
}

```

```
// Jenkins에서 슬립 함수 사용
sleep time: 1, unit: 'SECONDS'
}

if (!healthCheckPassed) {
    echo "Health check failed"
    // Jenkins 파이프라인에서 에러 발
    sh '''
        curl -d '{"text":"## Back
        '''
    error("Deployment failed due
}
}
}
}
}
```

▼ 배포서버 예시 2 (blue-green)

```
pipeline {
    agent any

    environment{
        REGION = 'ap-northeast-2'
        EKS_API = 'https://F73C5F6407E7C0DFD2AF34C
773B41A24.gr7.ap-northeast-2.eks.amazonaws.com'
        EKS_CLUSTER_NAME = 'mycluster'
        EKS_JENKINS_CREDENTIAL_ID = 'kubectl-crede
ntial'
        ECR_PATH = '654654554890.dkr.ecr.ap-northe
ast-2.amazonaws.com'
        IMAGE_NAME = 'hellolaw-back-api'
        AWS_CREDENTIAL_ID = 'aws-credential'

    }
    stages {
        stage('MMAalert-start') {
```

```

        steps {
            sh '''
                curl -d '{"text":"## Back-API-Pr
od Deploy Start :anya1::anya1::anya1:  }' -H "Co
ntent-Type: application/json" -X POST https://meet
ing.ssafy.com/hooks/ya5sm8fy4brz9eprbxxmghqu8e
            '''
        }
    }

    stage('deployment download'){
        steps{
            script{
                withCredentials([file(credenti
alsId: 'prod-back-api-blue', variable: 'deploymen
t'))] {
                    sh 'rm -rf blue-deploymen
t.yml'
                    sh 'cp $deployment blue-de
ployment.yml'
                }
                withCredentials([file(credenti
alsId: 'prod-back-api-green', variable: 'deploymen
t'))] {
                    sh 'rm -rf green-deploymen
t.yml'
                    sh 'cp $deployment green-d
eployment.yml'
                }
            }
        }
    }

    stage('Determine Active Version') {
        steps {
            script {
                // 파드의 현재 상태를 쿠버네티스에서
확인
                env.ACTIVE_VERSION = sh(scrip

```

```

t: 'kubectl get service back-api-service -n produc
tion -o jsonpath="{.spec.selector.color}"', return
Stdout: true).trim()
        env.TARGET_VERSION = env.ACTIV
E_VERSION == 'blue' ? 'green' : 'blue'
    }
}
}

stage('Deploy to k8s'){
    steps{
        script{
            sh """
            kubectl apply -f ${TARGET_VERS
ION}-deployment.yml
            PATCH_CONTENT=\$(cat ${ACTIVE_
VERSION}-${TARGET_VERSION}.yaml)
            kubectl patch svc back-api-ser
vice -n production -p "\${PATCH_CONTENT}"
            """
        }
    }
}
stage('Health Check') {
    steps {
        script {
            def attempts = 15
            def healthCheckPassed = false

            for (int i = 0; i < attempts;
i++) {
                // curl을 사용하여 헬스 체크 수
                행

                def response = sh script:
                "curl -s https://hellolaw.kr/api/health", returnSt
                dout: true

                if (response.contains("hea
lth")) {

```

```
sed"
    healthCheckPassed = true
    sh '''
        curl -d '{"text":"## Back-Api-Prod Deploy Success :anya_wakuwaku_2::anya_wakuwaku_2::anya_wakuwaku_2:"}' -H "Content-Type: application/json" -X POST https://meeting.ssafy.com/hooks/ya5sm8fy4brz9eprbxxmghqu8e
        ...
    break
}
// Jenkins에서 슬립 함수 사용
sleep time: 1, unit: 'SECONDS'
NDS'
}

if (!healthCheckPassed) {
    echo "Health check failed"
    // Jenkins 파이프라인에서 에러 발생시키기
    sh '''
        curl -d '{"text":"## Back-Api-Prod Deploy Failed :anya_crying::anya_crying::anya_crying:"}' -H "Content-Type: application/json" -X POST https://meeting.ssafy.com/hooks/ya5sm8fy4brz9eprbxxmghqu8e
        ...
        error("Deployment failed due to health check failure.")
    }
}
}
```

▼ TLS 적용

- ACM에서 구입한 도메인 인증받아야함 (Route 53 썼으면 자동으로 찾아준다)

도메인 (1)

Route 53에서 레코드 생성

CSV로 내보내기

< 1 >

도메인	상태	갱신 상태	유형	CNAME 이름	CNAME 값
*.hellolaw.kr	성공	-	CNAME	<div><div></div><div>_2635ff7f9406d5228d31c5ef5399d5d7.hellolaw.kr.</div></div>	<div><div></div><div>_14c40b82ed8a0e92c51ec4a2870a7804.mhbtsbpdnt.acm-validations.aws.</div></div>

레코드 개수 : 2개

최근 업데이트 : 2024-04-30 12:03:57

타입	호스트	값/위치
CNAME	<div><div>_2635ff7f940.hellolaw.kr</div></div>	<div><div>_14c40b82ed8a0e92c51ec4a2870a7804.mhbtsbpdnt.acm-validations.aws.</div></div>

1. ACM에 도메인 인증 요청 생성
2. DNS 관리 사이트 → CNAME 이름에서 도메인 앞부분을 호스트로, CNAME 값을 값/위치 항목에 쓰고 등록하기
3. 기다리면 인증 된다.
4. 발급된 ARN 주소를 Ingress에 입력하여 사용하기

- ingress yml에서

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-ingress
  namespace: default
  annotations:
    kubernetes.io/ingress.class: "alb"
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":
443}]'
    alb.ingress.kubernetes.io/certificate-arn: <your-acm
-certificate-arn>
spec:
  rules:
    - host: myapp.example.com
      http:

```

```

paths:
  - path: /
    pathType: Prefix
    backend:
      service:
        name: myservice
        port:
          number: 80

```

▼ OICP 생성

- IAM OIDC provider

```

eksctl utils associate-iam-oidc-provider \
  --region ap-northeast-2 \ # Your Region
  --cluster eks \ # Your Cluster Name
  --approve

```

- Policy 생성

- policy 문서 다운로드

```

curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/main/docs/install/iam_policy.json

```

- policy 등록

```

aws iam create-policy --policy-name alb-controller-iam-policy --policy-document file://iam_policy.json

```

- ServiceAccount 생성

```

eksctl create iamserviceaccount --cluster {cluster_name} --namespace kube-system --name aws-load-balancer-controller --attach-policy-arn arn:aws:iam::$ACCOUNT_ID:policy/{policy_name} --override-existing-serviceaccounts --approve

```

```
eksctl create iamserviceaccount --cluster mycluster -
-namespace kube-system --name aws-load-balancer-contr
oller --attach-policy-arn arn:aws:iam::skajd1:policy/
alb-controller-iam-policy --override-existing-service
accounts --approve
```

▼ ALB Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: aws-load-balancer-controller
  namespace: development # 개발서버
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aws-load-balancer-controller
  template:
    metadata:
      labels:
        app: aws-load-balancer-controller
    spec:
      serviceAccountName: iamserviceaccount # 이 서비스 계
      정은 development 네임스페이스에 있어야 함
      containers:
        - name: aws-load-balancer-controller
          image: amazon/aws-alb-ingress-controller:v2.3.
1
          args:
            - --cluster-name=mycluster
            - --ingress-class=alb
            - --watch-namespace=development # 이 컨트롤러
      가 development 네임스페이스만 관찰하도록 설정
```

▼ 서비스어카운트 생성

- eksctl 생성

```
eksctl create iamserviceaccount --cluster mycluster --names
```

▼ db 연동

MYSQL

- 먼저 PVC 설정 해야한다! (도커의 볼륨 설정과 유사)

▼ EBS-CLI Driver(pvc 볼륨 자동 생성해주는 애)

- iam 생성

```
eksctl create iamserviceaccount \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --cluster mycluster \
  --role-name AmazonEKS_EBS_CSI_DriverRole \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/se
  --approve
```

- eksctl driver 설치

```
eksctl create addon --name aws-ebs-csi-driver --cl
```

▼ MySQL 매니피스트

▼ 개발서버 deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: development
spec:
  replicas: 1
  selector:
    matchLabels:
```

```

    app: mysql
template:
  metadata:
    labels:
      app: mysql
  spec:
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
            - matchExpressions:
                - key: role
                  operator: In
                  values:
                    - development
    containers:
      - name: mysql
        image: mysql:5.6
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: password
        ports:
          - containerPort: 3306
        volumeMounts:
          - name: mysql-storage
            mountPath: /var/lib/mysql
          - name: initdb
            mountPath: /docker-entrypoint-initdb.d
    volumes:
      - name: mysql-storage
        persistentVolumeClaim:
          claimName: mysql-pvc
      - name: initdb
        configMap:
          name: mysql-initdb

```

```

---

apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  namespace: development
spec:
  ports:
    - port: 3306
      targetPort: 3306
  selector:
    app: mysql

---

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
  namespace: development
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 10Gi

---

apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  namespace: development
type: Opaque

```

```
data:
  password: aGVsbG9sYXdfZGV2
```

▼ config

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-initdb
  namespace: development
data:
  init.sql: |
    CREATE DATABASE IF NOT EXISTS hellolaw CHARACTER
    SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci;

    CREATE USER 'hellolaw_dev'@'%' IDENTIFIED BY 'hellolaw_dev';
    GRANT ALL PRIVILEGES ON hellolaw.* TO 'hellolaw_dev'@'%';
    FLUSH PRIVILEGES;
```

데이터베이스 생성할 때 인코딩 맞춰주자 꼭 !

여기서 mysql 5.7이 아니라, 5.6을 사용했는데, 5.7은 생성된 볼륨을 인식 못하는 버그가 있었다. 다운그레이드 하니까 해결 됐음! ㅋㅋ

REDIS

▼ Redis

▼ 개발서버 deployment

```
apiVersion: v1
kind: Secret
metadata:
  name: redis-secret
  namespace: development
type: Opaque
data:
  redis-password: aGVsbG9sYXdfZGV2 # 'hellolaw_dev'의 비밀번호
```

```

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  namespace: development
spec:
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: role
                    operator: In
                    values:
                      - development
      containers:
        - name: redis
          image: redis:6.2.6
          env:
            - name: REDIS_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: redis-secret
                  key: redis-password
          args: ["--requirepass", "${REDIS_PASSWORD}"]
          ports:
            - containerPort: 6379

```



```

---


apiVersion: v1
kind: Service
metadata:
  name: redis-service
  namespace: development
spec:
  ports:
    - port: 6379
      targetPort: 6379
  selector:
    app: redis

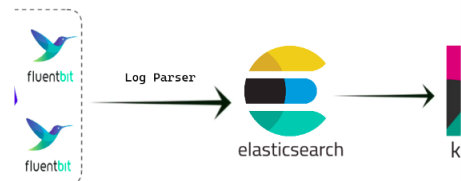
```

▼ Fluentd 로 로깅관리

로그 수집을 위한 AWS EFK Stack 구축

로그 수집의 배경 쿠버네티스는 기본적으로 Pod가 정상상태가 아니라면 Pod를 kill하고 새로 생성한다. 하지만 운영자 입장에서는 Pod가 죽은 원인을 알아야 한다. 하지만, Pod가 죽으면 로그까지

 <https://devpoong.tistory.com/101>



▼ 전체 리소스

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
cert-manager	pod/cert-manager-cainjector-5c7c44654d-sz	244	1/1	Running	0 12d
cert-manager	pod/cert-manager-d7d7c546f-znf4b	1/1	Running	0	12d
cert-manager	pod/cert-manager-webhook-554cd5b9d5-kxlv2	1/1	Running	0	12d
development	pod/hellolaw-api-6f9c46c9cd-7zgp2	1/1	Running	0	3h34m
development	pod/hellolaw-auth-5884944877-xq7kp	1/1	Running	0	3h45m

```

development    pod/hellolaw-front-55c6577f97-vnkdz
1/1    Running    0            21h
development    pod/mysql-6b7487f75f-pw9pw
1/1    Running    0            25h
development    pod/redis-69f4db98d8-h7h54
1/1    Running    0            9d
kube-system    pod/aws-load-balancer-controller-76db544c
b-j8jpb 1/1    Running    0            9d
kube-system    pod/aws-node-mswmg
2/2    Running    0            13d
kube-system    pod/aws-node-txckq
2/2    Running    0            13d
kube-system    pod/coredns-f94fb47d9-m2h5c
1/1    Running    0            13d
kube-system    pod/coredns-f94fb47d9-t6trj
1/1    Running    0            13d
kube-system    pod/ebs-csi-controller-796cc8f5b5-4hmqx
6/6    Running    0            9d
kube-system    pod/ebs-csi-controller-796cc8f5b5-bxz85
6/6    Running    0            9d
kube-system    pod/ebs-csi-node-5r5bp
3/3    Running    0            9d
kube-system    pod/ebs-csi-node-zgmcg
3/3    Running    0            9d
kube-system    pod/kube-proxy-clqcb
1/1    Running    0            13d
kube-system    pod/kube-proxy-rshq8
1/1    Running    0            13d
production     pod/hellolaw-api-blue-647f85955-8925s
1/1    Running    0            33m
production     pod/hellolaw-api-blue-647f85955-s528h
1/1    Running    0            33m
production     pod/hellolaw-api-green-5968f77f4b-tfb92
1/1    Running    0            33m
production     pod/hellolaw-api-green-5968f77f4b-thlrh
1/1    Running    0            33m
production     pod/hellolaw-auth-74db5469fc-2rnsd
1/1    Running    0            13m

```

```

production      pod/hellolaw-front-75d464bc45-88hv4
1/1      Running    0          18h
production      pod/mysql-b7bfb574-xrclt
1/1      Running    0          5h1m
production      pod/redis-d9d9674cd-q467x
1/1      Running    0          5h31m

```

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
cert-manager	service/cert-manager	ClusterIP	10.100.168.143	<none>	9402/TCP	12d
cert-manager	service/cert-manager-webhook	ClusterIP	10.100.92.228	<none>	443/TCP	12d
default	service/kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	13d
development	service/back-api-service	ClusterIP	10.100.55.27	<none>	8082/TCP	9d
development	service/back-auth-service	ClusterIP	10.100.241.25	<none>	8099/TCP	9d
development	service/mysql-service	LoadBalancer	10.100.1.126	a08f14ea73e2e4504b53f82484df43e9-778954246.ap-northeast-2.elb.amazonaws.com	306:32599/TCP	9d
development	service/nginx-service	ClusterIP	10.100.111.20	<none>	80/TCP	9d
development	service/redis-service	ClusterIP	10.100.1.162	<none>	6379/TCP	9d
kube-system	service/aws-load-balancer-webhook-service	ClusterIP	10.100.41.136	<none>	443/TCP	10d

```

kube-system      service/kube-dns
ClusterIP        10.100.0.10      <none>
53/UDP,53/TCP,9153/TCP    13d
logging          service/elasticsearch-svc
ClusterIP        10.100.48.220    <none>
9200/TCP,9300/TCP        4d20h
production       service/back-api-service
ClusterIP        10.100.253.234   <none>
8082/TCP          47m
production       service/back-auth-service
ClusterIP        10.100.241.179   <none>
8099/TCP          18h
production       service/mysql-service
LoadBalancer     10.100.35.186    a2aa13d3d99e34db5a6fce13
7bb0692c-1011795548.ap-northeast-2.elb.amazonaws.com    3
306:31157/TCP      21h
production       service/nginx-service
ClusterIP        10.100.200.55    <none>
80/TCP           27h
production       service/redis-service
ClusterIP        10.100.185.219   <none>
6379/TCP         25h

```

NAMESPACE	NAME	DESI
RED	CURRENT	READY
UP-TO-DATE	AVAILABLE	AGE
LECTOR	AGE	SE
kube-system	daemonset.apps/aws-node	2
2	2	2
13d		<none>
kube-system	daemonset.apps/ebs-csi-node	2
2	2	2
o/os=linux	9d	kubernetes.i
kube-system	daemonset.apps/ebs-csi-node-windows	0
0	0	0
o/os=windows	9d	kubernetes.i
kube-system	daemonset.apps/kube-proxy	2
2	2	2
13d		<none>

NAMESPACE	NAME	READY	UP-TO-DATE	AVAILABLE	AGE
cert-manager	deployment.apps/cert-manager	1/1	1	1	12d
cert-manager	deployment.apps/cert-manager-cainjector	1/1	1	1	12d
cert-manager	deployment.apps/cert-manager-webhook	1/1	1	1	12d
development	deployment.apps/hellolaw-api	1/1	1	1	22h
development	deployment.apps/hellolaw-auth	1/1	1	1	6d5h
development	deployment.apps/hellolaw-front	1/1	1	1	13d
development	deployment.apps/mysql	1/1	1	1	25h
development	deployment.apps/redis	1/1	1	1	9d
kube-system	deployment.apps/aws-load-balancer-controller	1/1	1	1	9d
kube-system	deployment.apps/coredns	2/2	2	2	13d
kube-system	deployment.apps/ebs-csi-controller	2/2	2	2	9d
production	deployment.apps/hellolaw-api-blue	2/2	2	2	33m
production	deployment.apps/hellolaw-api-green	2/2	2	2	33m
production	deployment.apps/hellolaw-auth	1/1	1	1	18h
production	deployment.apps/hellolaw-front	1/1	1	1	27h
production	deployment.apps/mysql	1/1	1	1	5h1m
production	deployment.apps/redis	1/1	1	1	5h31m

NAMESPACE	NAME
DESIRED	CURRENT
READY	AGE
cert-manager	replicaset.apps/cert-manager-cainjector-5c7c44654d
1	1
1	12d
cert-manager	replicaset.apps/cert-manager-d7d7c546f
1	1
1	12d
cert-manager	replicaset.apps/cert-manager-webhook-554cd5b9d5
1	1
1	12d
development	replicaset.apps/hellolaw-api-6f9c46c9cd
1	1
1	3h34m
development	replicaset.apps/hellolaw-auth-5884944877
1	1
1	3h45m
development	replicaset.apps/hellolaw-front-55c6577f97
1	1
1	21h
development	replicaset.apps/mysql-6b7487f75f
1	1
1	25h
development	replicaset.apps/redis-69f4db98d8
1	1
1	9d
kube-system	replicaset.apps/aws-load-balancer-controller-76db544cb
1	1
1	9d
kube-system	replicaset.apps/coredns-f94fb47d9
2	2
2	13d
kube-system	replicaset.apps/ebs-csi-controller-796cc8f5b5
2	2
2	9d
production	replicaset.apps/hellolaw-api-blue-647f85955
2	2
2	33m
production	replicaset.apps/hellolaw-api-green-5968f77f4b
2	2
2	33m
production	replicaset.apps/hellolaw-auth-676b5658f7
0	0
0	3h51m
production	replicaset.apps/hellolaw-auth-6f94466657
0	0
0	3h49m
production	replicaset.apps/hellolaw-auth-74db5469fc
1	1
1	13m
production	replicaset.apps/hellolaw-front-56456948b
0	0
0	18h
production	replicaset.apps/hellolaw-front-75d464bc45
1	1
1	18h

```
production      replicaset.apps/hellolaw-front-7667ff674d
0              0              0              27h
production      replicaset.apps/mysql-b7bfb574
1              1              1              5h1m
production      replicaset.apps/redis-d9d9674cd
1              1              1              5h31m
```