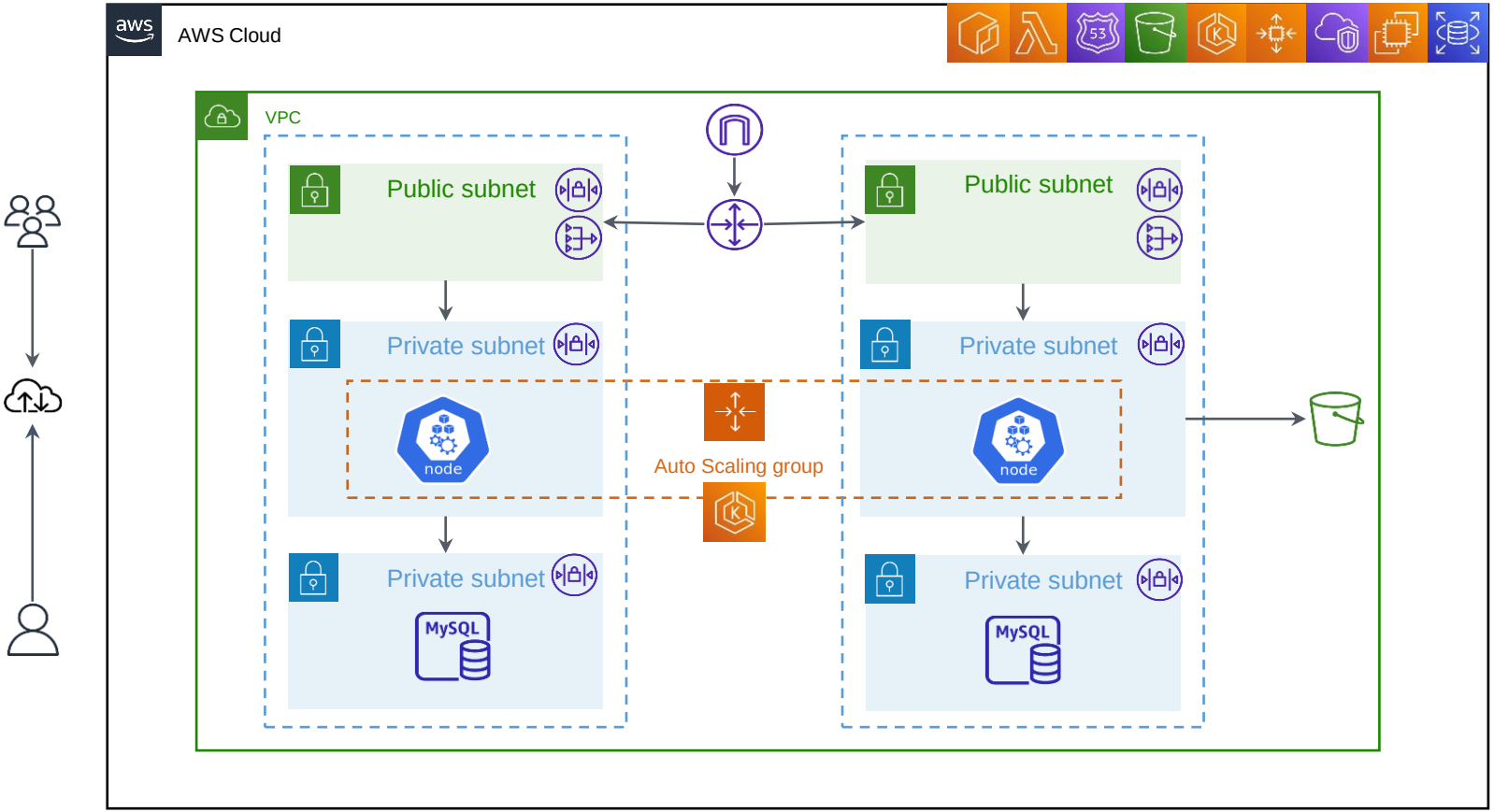
Django 3-Tier 구성

3조
임지원

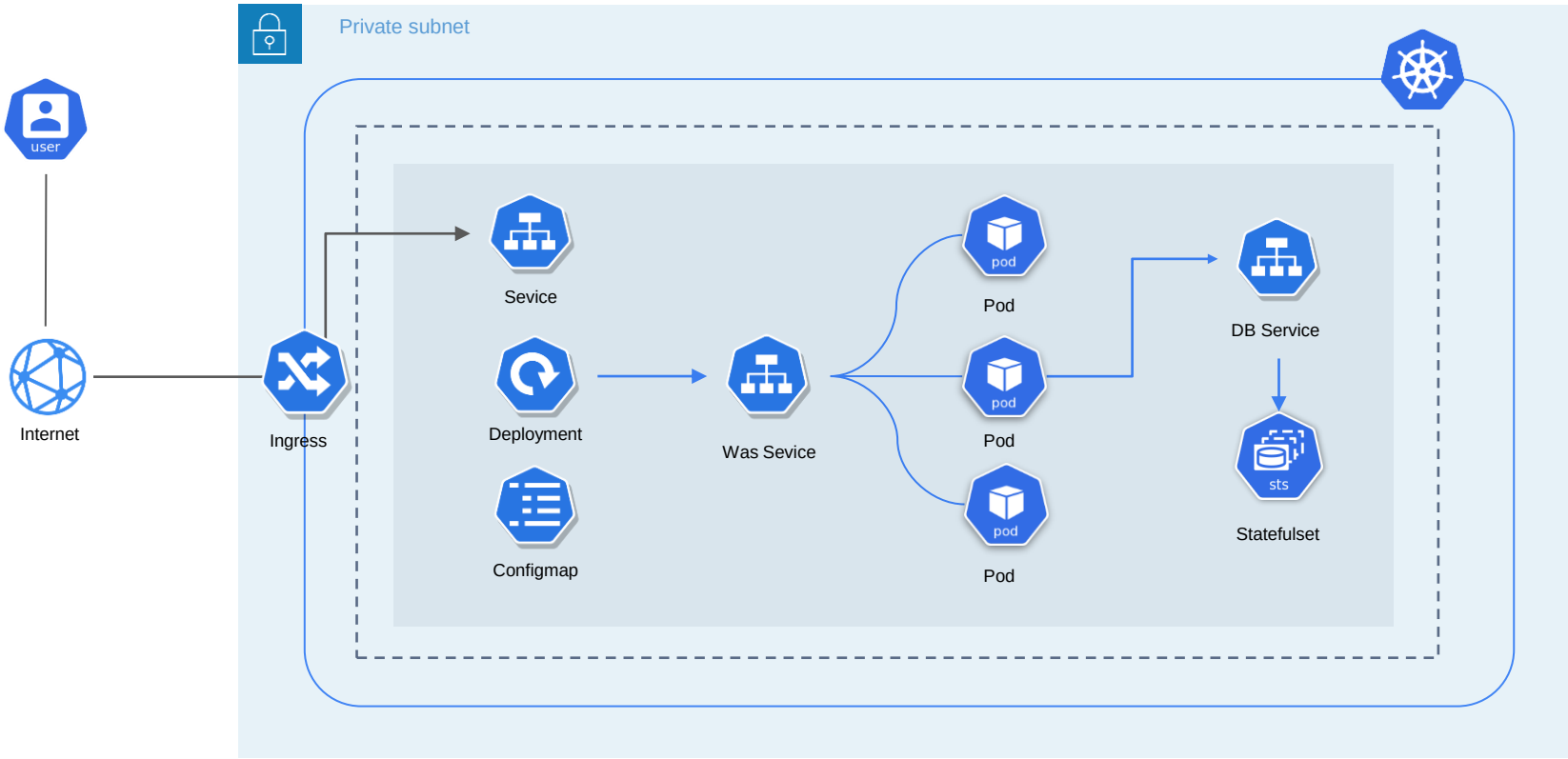
목차

1. **구성도**
2. **선행작업**
    1. Django 이미지 build
    2. Docker hub에 push
3. **3 tier 구성**
    1. DB tier 구성
    2. Was tier 구성
    3. Web tier 구성
4. **Ingress 구성**
5. **접속 확인**

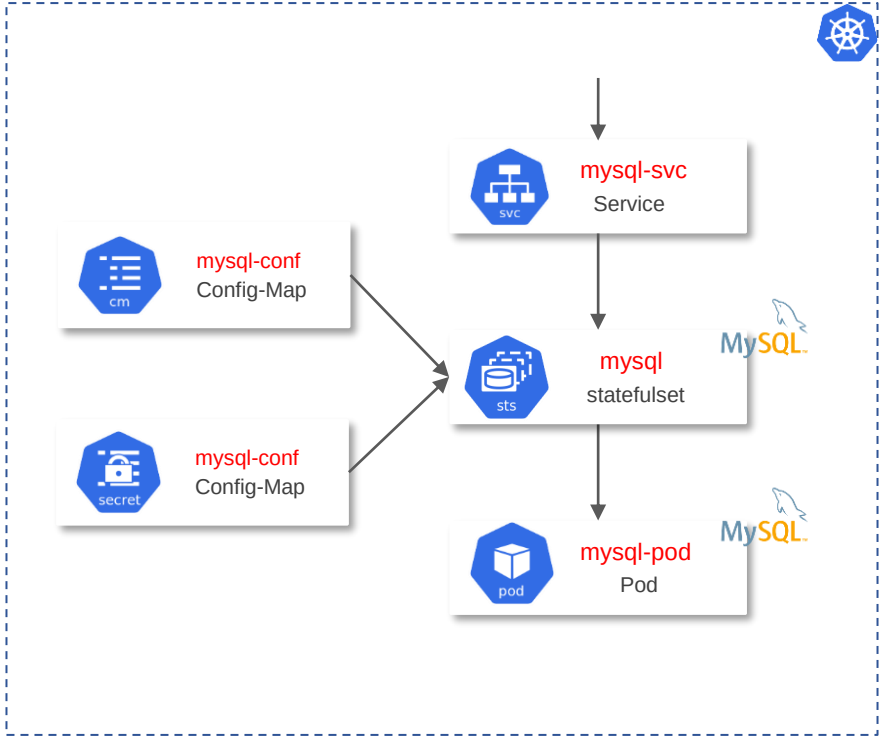
기본 AWS 구성도


AWS Cloud
VPC
Public subnet
Public subnet
Private subnet
Private subnet
node
node
Auto Scaling group
Private subnet
Private subnet
MySQL
MySQL

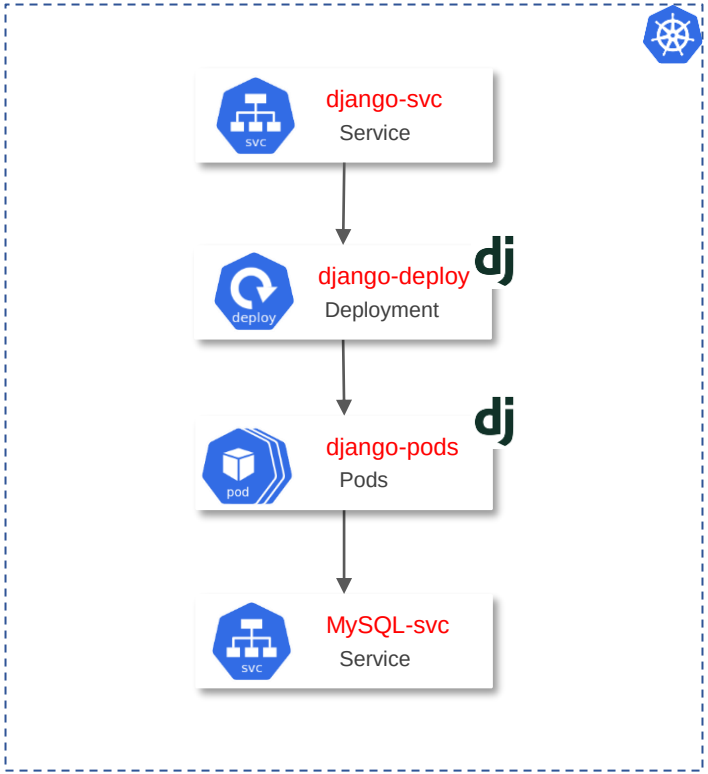
쿠버네티스 구성도


Private subnet
user
Internet
Ingress
Sevice
Deployment
Configmap
Was Sevice
pod
Pod
pod
Pod
pod
Pod
DB Service
sts
Statefulset

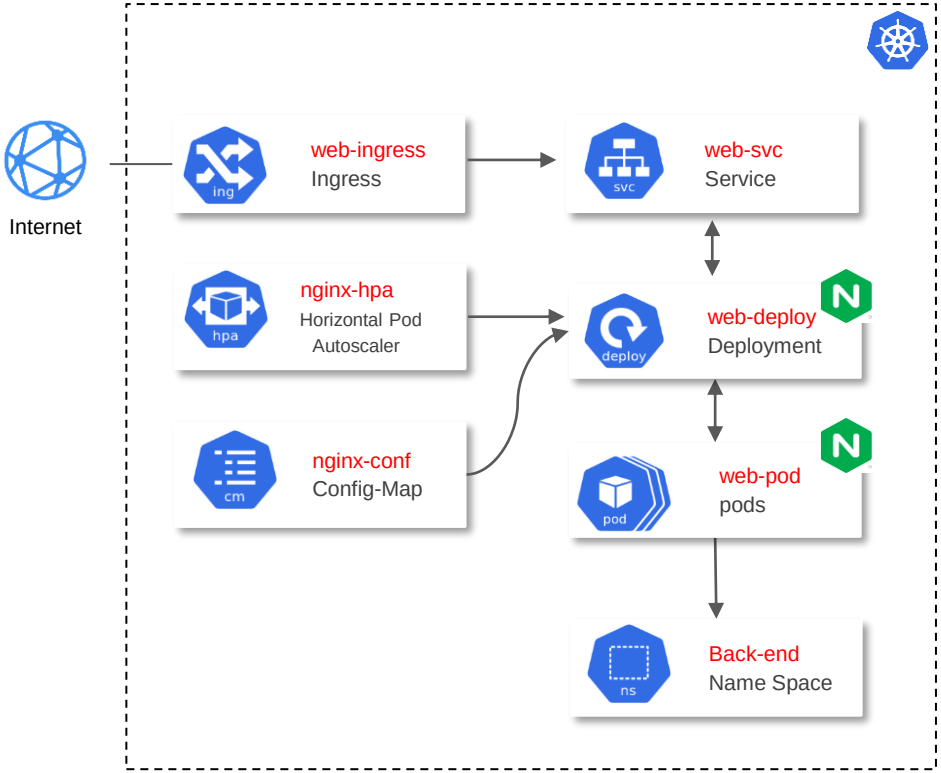
DB-tier 상세 구성도



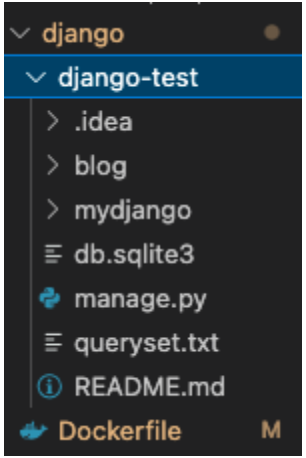
WAS-tier 상세 구성도



WEB-tier 상세 구성도



Django 이미지 빌드



도커 이미지 빌드를 위한 소스코드, 도커파일 구조

Dockerfile

```
FROM python:3.8

WORKDIR /usr/src/app

RUN pip install django==3.1.13
RUN pip install pymysql==1.0.2

COPY ./django-test .

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

EXPOSE 8000
```

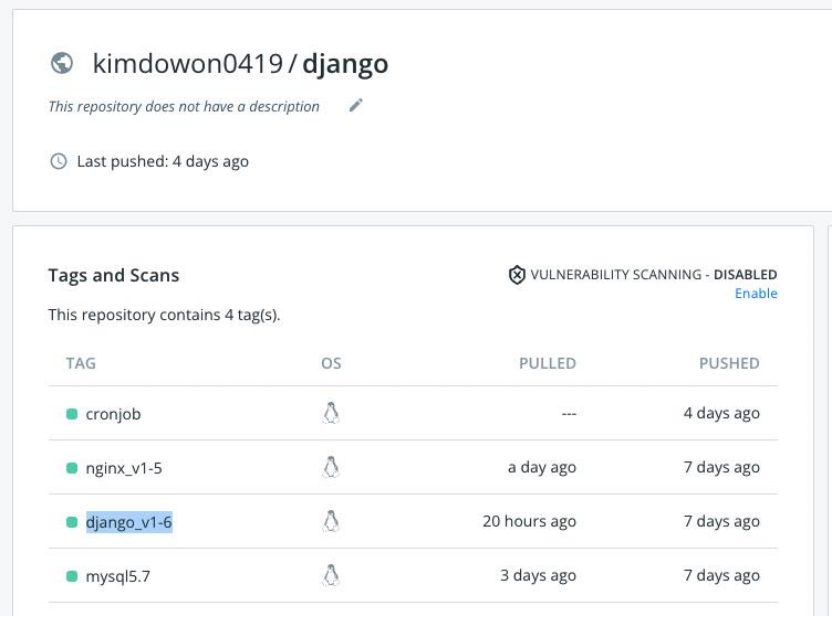
도커 이미지 빌드

```
$ docker build -t kimdowon0419/django:django_버전 .
```

Docker hub에 push

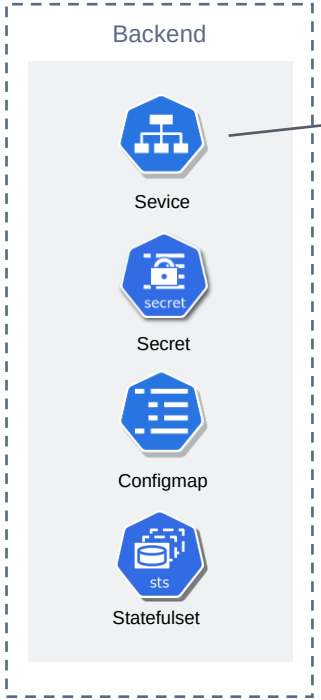
도커 이미지 빌드

$ docker push kimdowon0419/django:django_버전



DB tier 구성하기

Service Resource



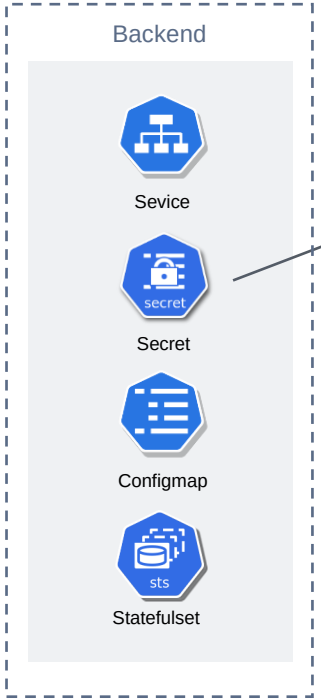
mysql-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-svc
  labels:
    app: mysql
spec:
  ports:
  - port: 3306
    name: mysql
  clusterIP: None # headless
  selector:
    app: mysql
```

실행 명령어 :
kubectl apply -f mysql-svc.yaml

DB tier 구성하기

Service Resource



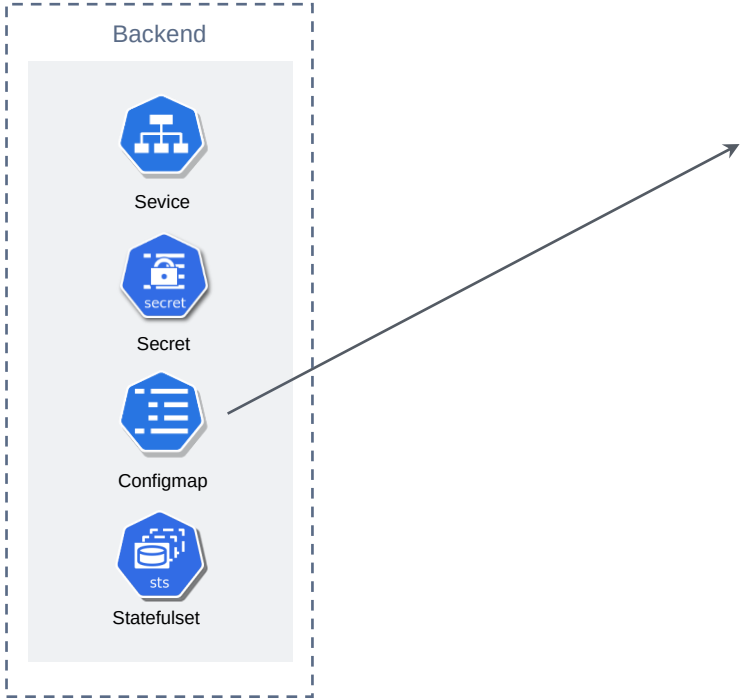
실행 명령어 :
kubectl apply -f mysql-pass.yaml

mysql-pass.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name:  mysql-pass
data:
   rootpass:  cXdlcjEyMzQ= #qwer1234 인코딩 된 password
   username:  cHl0aG9u # python
   userpass:  cHl0aG9u # python
type: Opaque
```

DB tier 구성하기

Service Resource



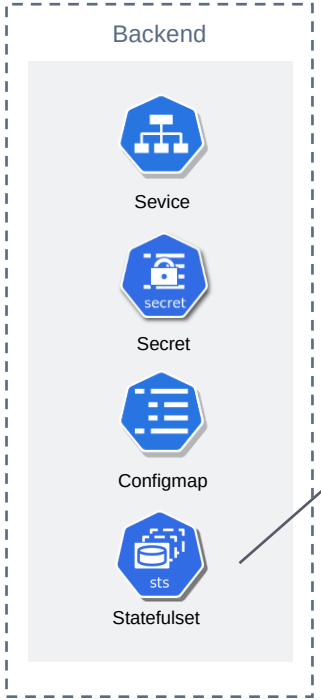
실행 명령어 :
kubectl apply -f mysql-conf.yaml

mysql-conf.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-cm
data:
  mysqldb: "django_db"
```

DB tier 구성하기

Service Resource



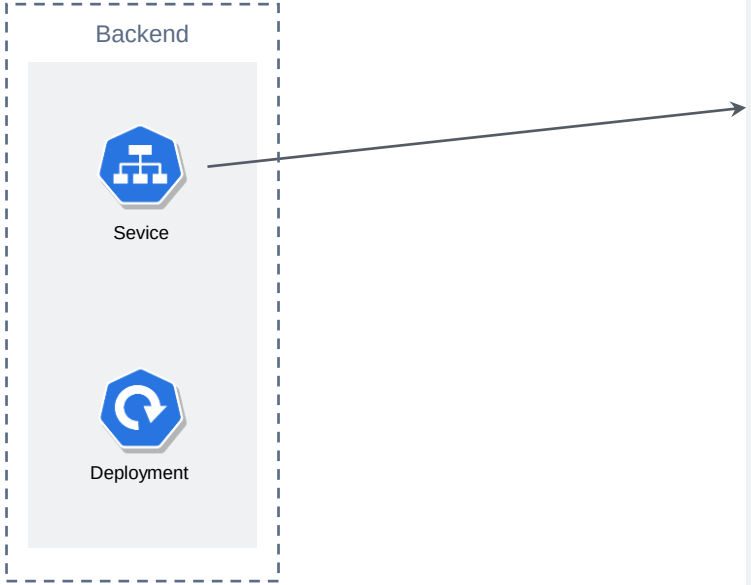
실행 명령어 :
kubectl apply -f mysql-state.yaml

mysql-state.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: "mysql-svc"
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: gymin97/django:mysql_v1
        ports:
        - containerPort: 3306
          name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: rootpass # qwer1234
        - name: MYSQL_DATABASE
          valueFrom:
            configMapKeyRef:
              name: mysql-cm
              key: mysqldb # django_db DB 생성
        - name: MYSQL_USER
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: username # python DB 계정
        - name: MYSQL_ROOT_HOST
          value: 'mysql-svc'
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: userpass # python DB 계정
volumeClaimTemplates:
- metadata:
    name: data
  spec:
    accessModes: [ "ReadWriteOnce" ]
    resources:
      requests:
        storage: 1Gi
```

WAS tier 구성하기

Service Resource



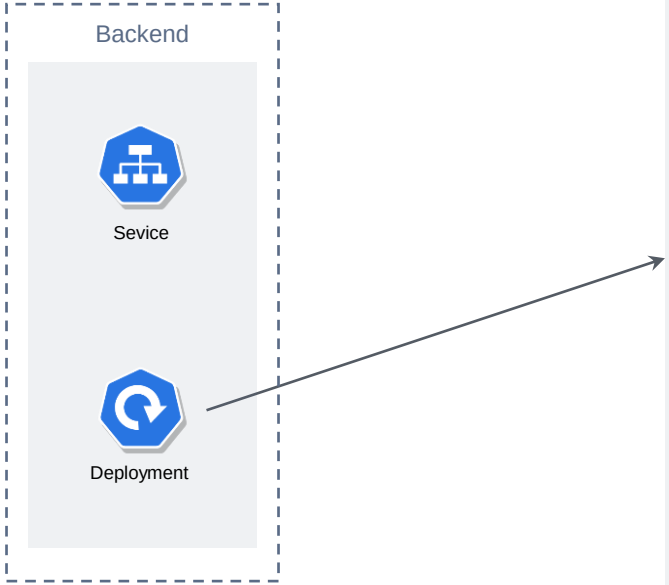
실행 명령어 :
kubectl apply -f was-svc.yaml

django-svc.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: django-svc
spec:
  selector:
    app: was
  ports:
  - port: 8000
    targetPort: 8000
```

WAS tier 구성하기

Deployment Resource



was server image → 빌드한 이미지 사용
image : ***hub계정/was:db-0.mydb***

실행 명령어 :

kubectl apply -f was-deploy.yaml

django-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: was
spec:
  replicas: 3
  selector:
    matchLabels:
      app: was
  template:
    metadata:
      labels:
        app: was
    spec:
      containers:
      - name: was
        image: gymin97/django:django_login
        imagePullPolicy: Always
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 8000
      initContainers:
      - name: init-mysql-svc
        image: busybox:1.28
        command: ['sh', '-c', "until nslookup mysql-0.mysql-svc.default.svc.cluster.local; do echo waiting for mysql-svc; sleep 2; done"]
```

Web tier 구성하기

Service Resource

nginx-svc.yaml

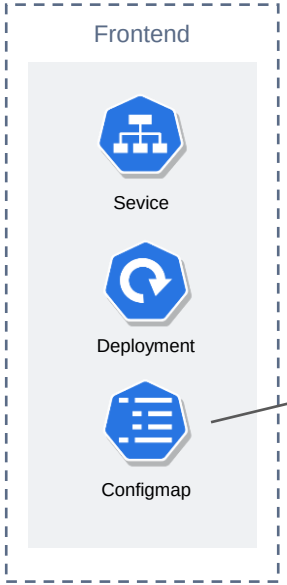
```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  selector:
    app: nginx
  ports:
  - nodePort: 32000
    port: 80
    targetPort: 80
  type: NodePort
```

Ingress를 하기 위해 type: NodePort로 한다

실행 명령어 :
kubectl apply -f nginx-svc.yaml

Web tier 구성하기

ConfigMap Resource



실행 명령어 :

kubectl apply -f nginx-conf.yaml

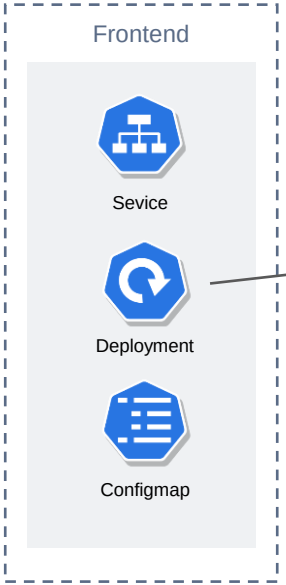
nginx-conf.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  nginx.conf: |
    user nginx;
    worker_processes auto;
    error_log /var/log/nginx/error.log;
    pid /run/nginx.pid;

    # Load dynamic modules. See /usr/share/nginx/README.dynamic.
    include /usr/share/nginx/modules/*.conf;

    events {
        worker_connections 1024;
    }

    http {
        log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                          '$status $body_bytes_sent "$http_referer" '
                          '"$http_user_agent" "$http_x_forwarded_for"';

        upstream backend {
        server django-svc:8000;
        }

        access_log  /var/log/nginx/access.log  main;

        sendfile            on;
        tcp_nopush          on;
        tcp_nodelay         on;
        keepalive_timeout   65;
        types_hash_max_size 2048;

        default_type        application/octet-stream;
```

was svc와
같은 이름 사용

Web tier 구성하기

Deployment Resource



실행 명령어 :

kubectl apply -f nginx-deploy.yaml

nginx-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: wonw/django:nginx_v1
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: "/etc/nginx/"
          name: nginx-conf
          readOnly: true
      volumes:
      - name: nginx-conf
        configMap:
          name: nginx-conf
```

HPA 위해서
resource 주기

nginx conf 이름 입력

Ingress controller 생성

eksctl utils associate-iam-oidc-provider --region ***본인리전*** --cluster ***클러스명*** --approve

curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.0/docs/install/iam_policy.json

eksctl create iamserviceaccount --cluster=***클러스명*** --namespace=kube-system --name=aws-load-balancer-controller --attach-policy-arn=arn:aws:iam::***계정 ID***:policy/AWSLoadBalancerControllerIAMPolicy --override-existing-serviceaccounts --approve

ingress Controller를 설치하기 전에 Cert-manager 설치
kubectl apply --validate=false -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml

curl -o v2_4_0_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.4.0/v2_4_0_full.yaml

v2_4_0_full.yaml 에서
your-cluster-name 부분을 ***내 클러스터명***으로

v2_4_0_full.yaml에서 다음 부분 제거

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
---
```

v2_4_0_full.yaml 실행

컨트롤러 설치 확인
kubectl get deployment -n kube-system aws-load-balancer-controller

Ingress

ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/scheme: internet-facing
    # SSL settings
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP":80}, {"HTTPS" :443}]'
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-east-2:118320467932:certificate/b6d
    alb.ingress.kubernetes.io/ssl-redirect: '443'
spec:
  rules:
  - host: web.oranda22.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: nginx-svc
            port:
              number: 80
```

aws Ingress Controller

ACM의 arn

ACM에 인증한 도메인 이름

nginx-svc명

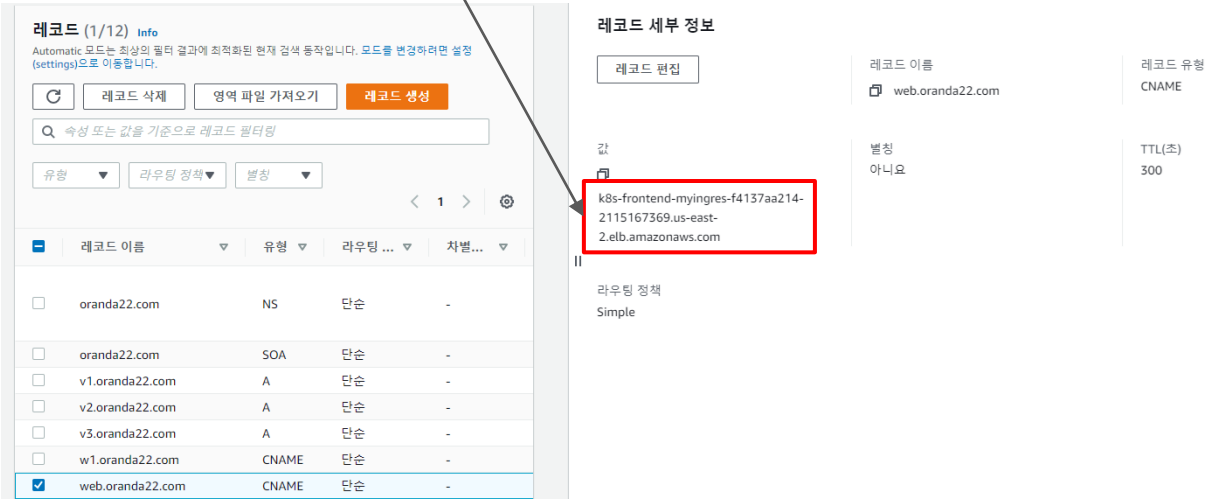
실행 명령어 :

kubectl apply -f ingress.yaml

Ingress

Route53

```
C:\multi-msa\msa-dev\03-project\statefulset-init-django-ingress-np-cw>kubectl get ingress -n frontend-ns
NAME        CLASS    HOSTS              ADDRESS                                                          PORTS   AGE
myingress   <none>   web.oranda22.com   k8s-frontend-myingres-f4137aa214-2115167369.us-east-2.elb.amazonaws.com   80      57s

C:\multi-msa\msa-dev\03-project\statefulset-init-django-ingress-np-cw>
```



AWS 콘솔 → Route53 → 레코드 수정

접속확인


Django Blog
web.oranda22.com
Django Blog
Blog(글)이 없습니다.
Page 1 of 1