HnH

K-Digital Training

클라우드(MSA) 서비스 개발 프로젝트

일석삼조

김경민 김도원 이병헌 임지원

목차

01

프로젝트 개요

주제 선정 배경

"Healthy Pleasure"

: 건강 관리(health)가
즐거워진다 (pleasure)

HnH

: Healthy & Happy

: 건강한 음식으로 즐거운 건강관리 !

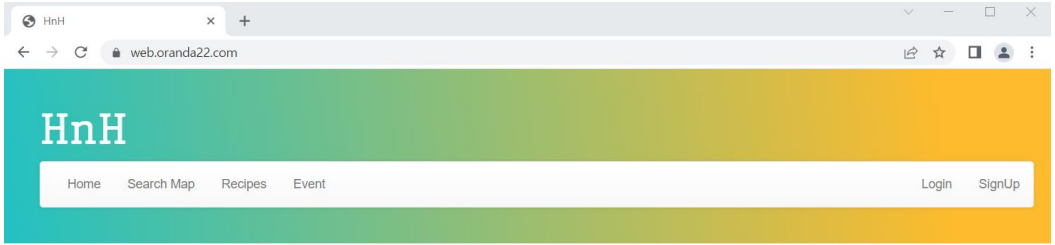
주요 서비스 기능

사용자 위치 기반 식당 검색

건강 레시피 검색

정보 공유 및 밥친구 게시판

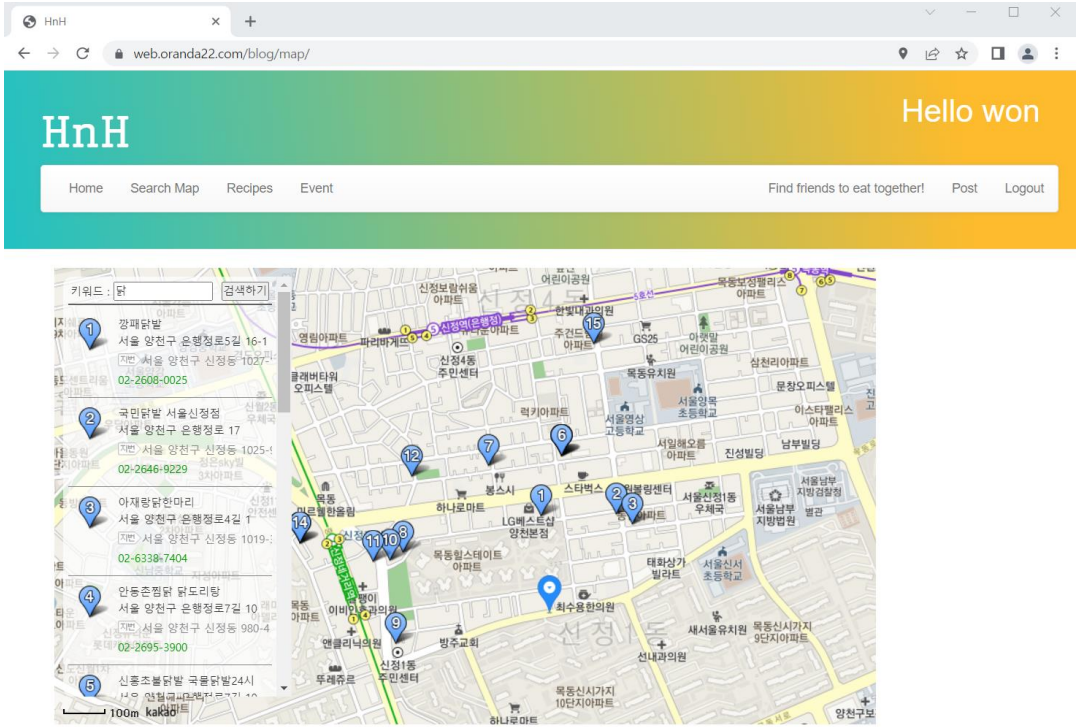
주요 서비스 기능


HnH
web.oranda22.com
HnH
Home
Search Map
Recipes
Event
Login
SignUp
글이 없습니다.
Page 1 of 1


HnH
web.oranda22.com
HnH
Hello won
Home
Search Map
Recipes
Event
Find friends to eat together!
Post
Logout
글이 없습니다.
Page 1 of 1

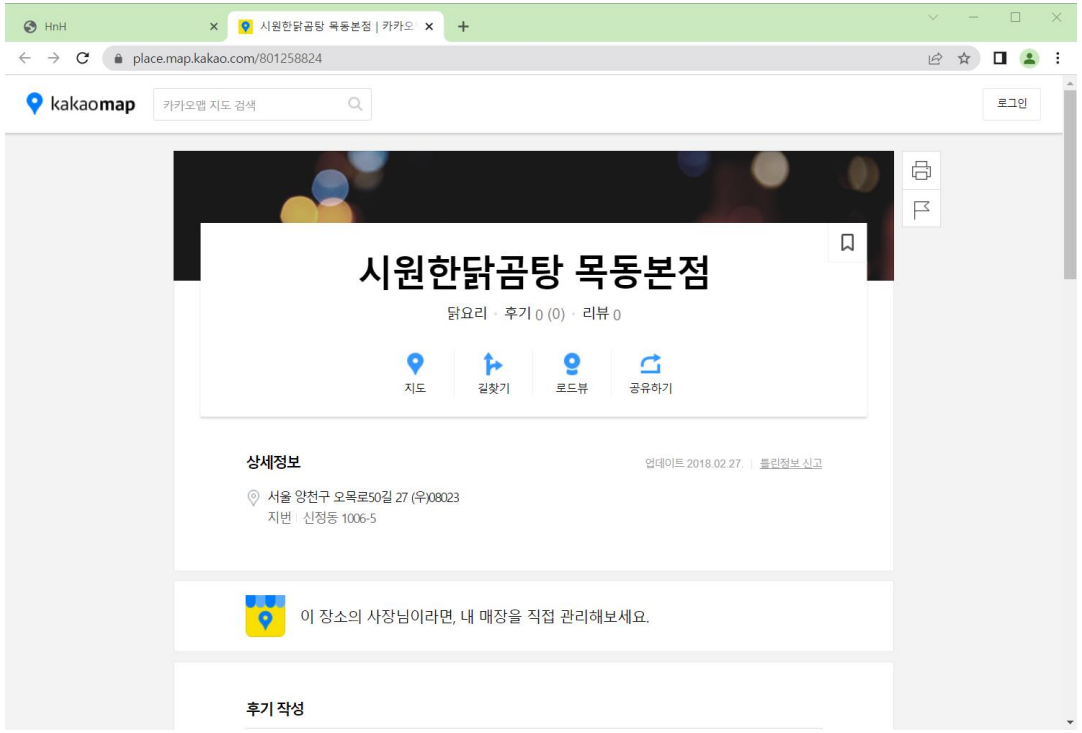
주요 서비스 기능

사용자 위치 기반 식당 검색



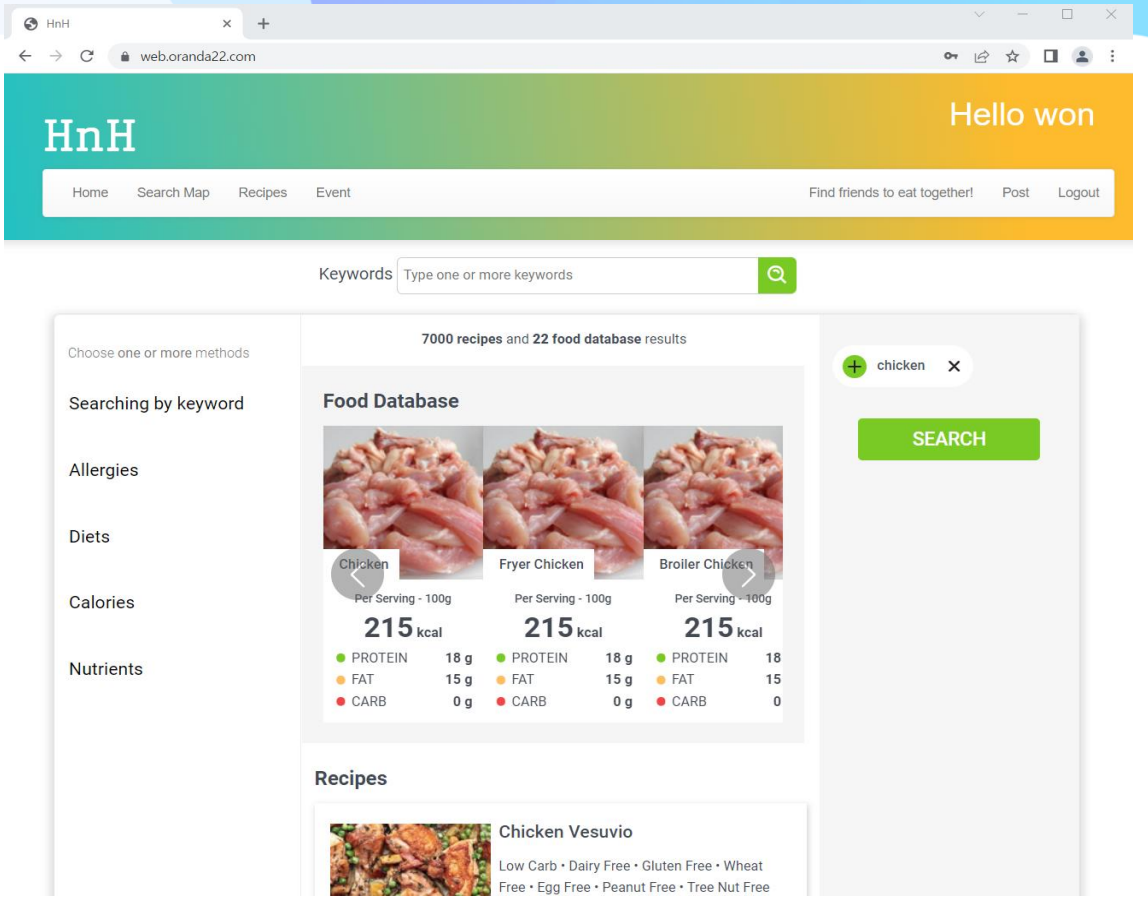
주요 서비스 기능

사용자 위치 기반 식당 검색



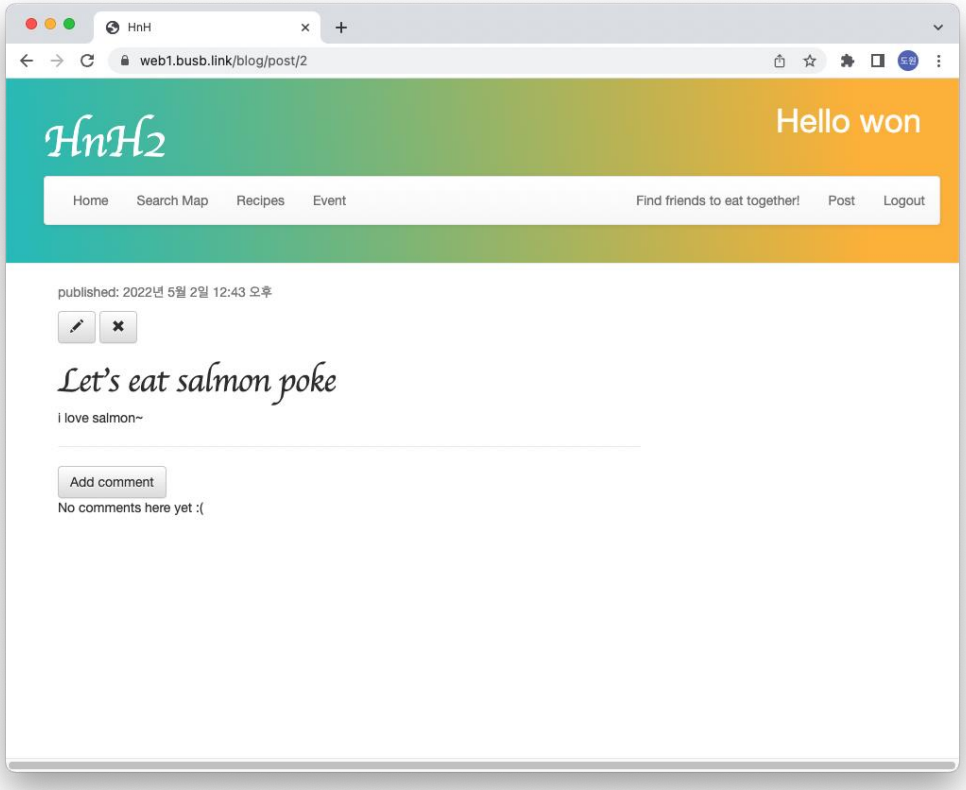
주요 서비스 기능

건강 레시피 검색

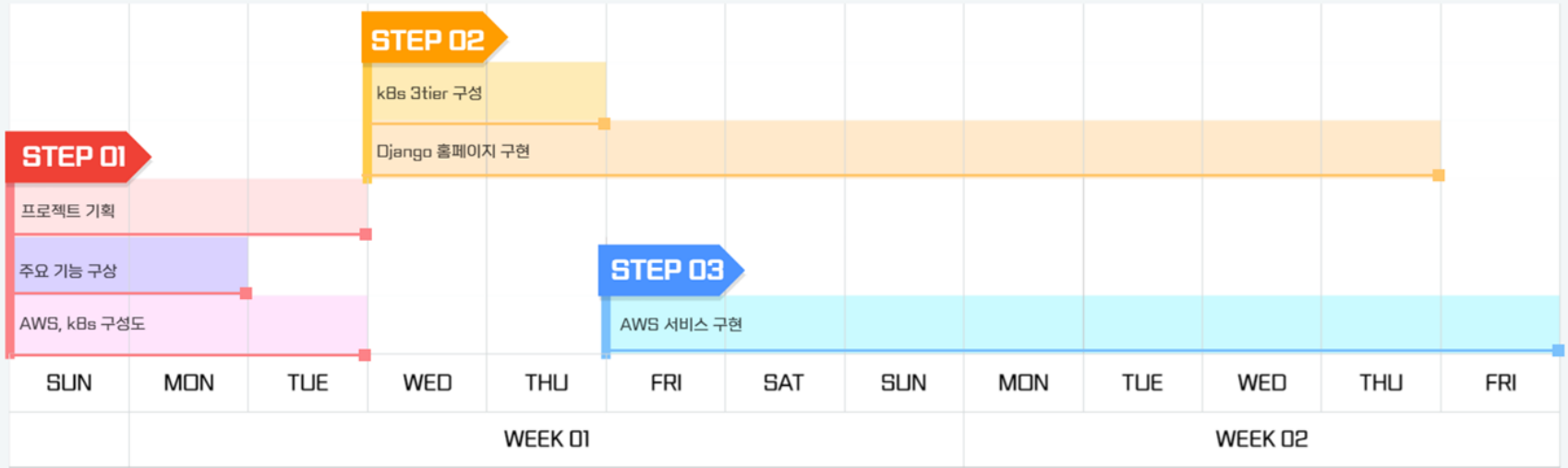


주요 서비스 기능

정보 공유 및 밥친구 게시판



프로젝트 작업 일정


STEP 02
k8s 3tier 구성
Django 홈페이지 구현
STEP 01
프로젝트 기획
주요 기능 구상
AWS, k8s 구성도
STEP 03
AWS 서비스 구현
SUN
MON
TUE
WED
THU
FRI
SAT
SUN
MON
TUE
WED
THU
FRI
WEEK 01
WEEK 02

프로젝트 도구


Amazon CloudWatch


Amazon SNS


AWS WAF


AWS Certificate Manager (ACM)


Elastic LoadBalancer

Route 53


Amazon RDS


Amazon S3


Amazon Elastic Container Registry (ECR)


Amazon EC2


Amazon EKS


AWS Lambda

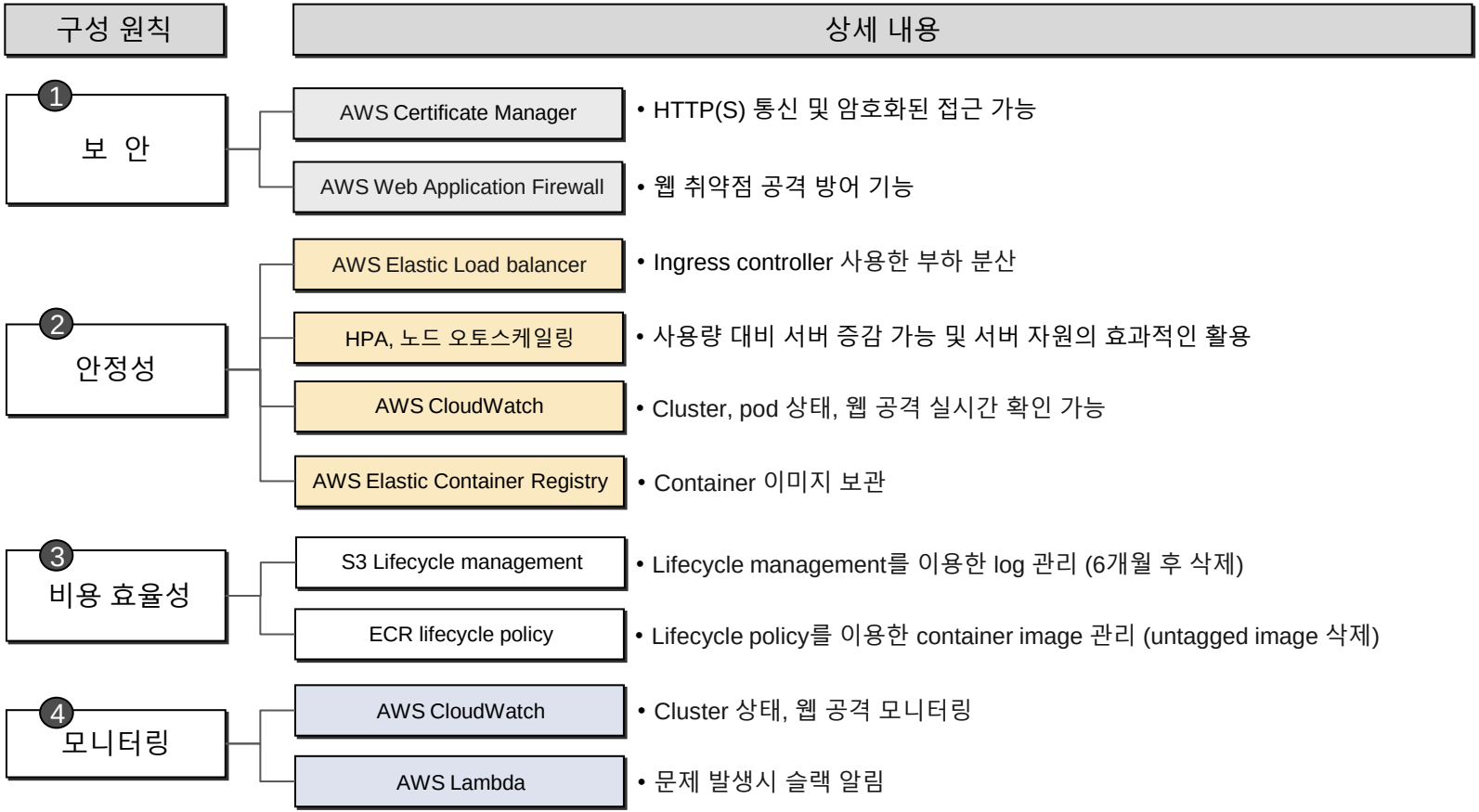

GitHub Actions

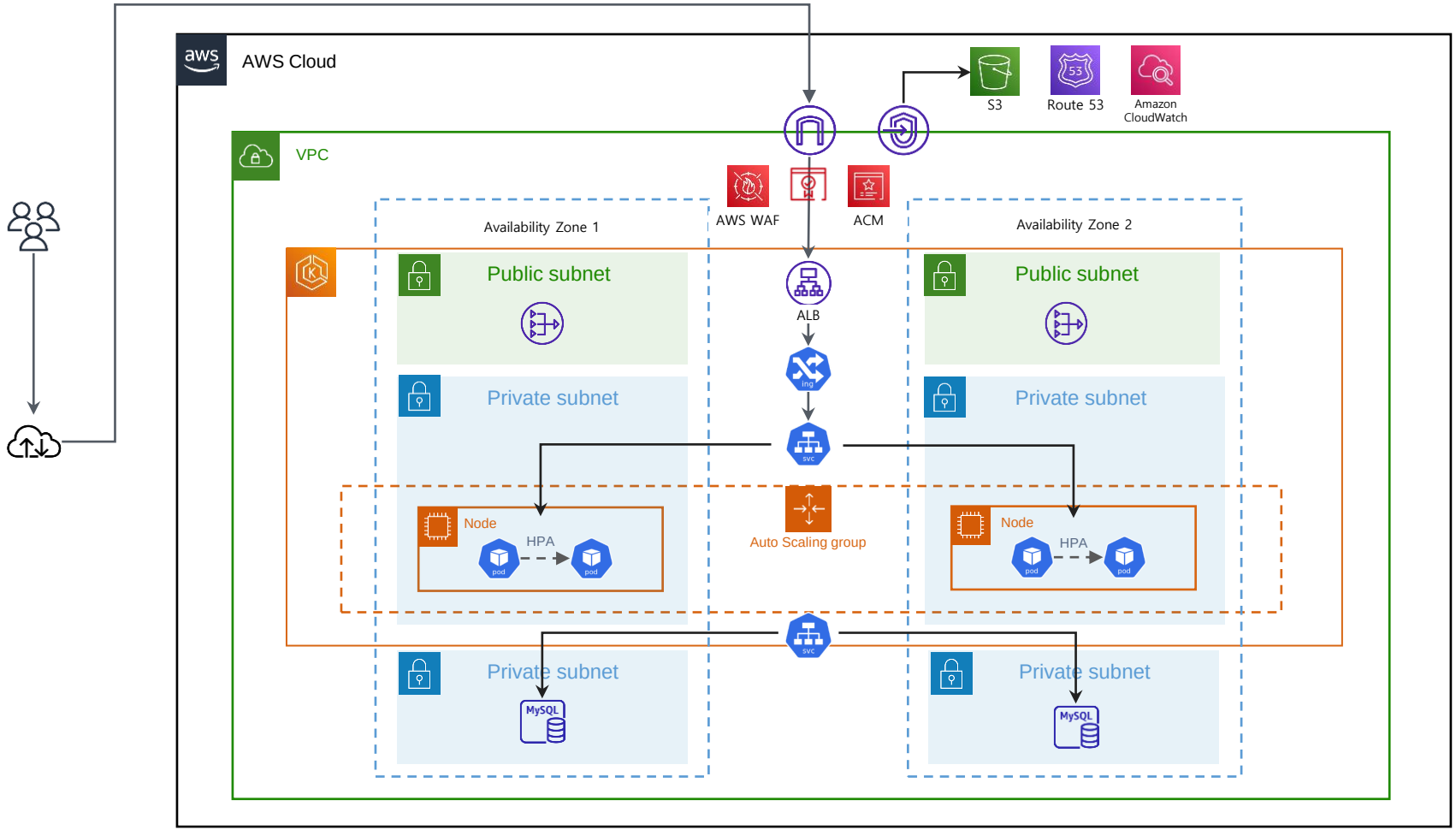
NGINX django slack

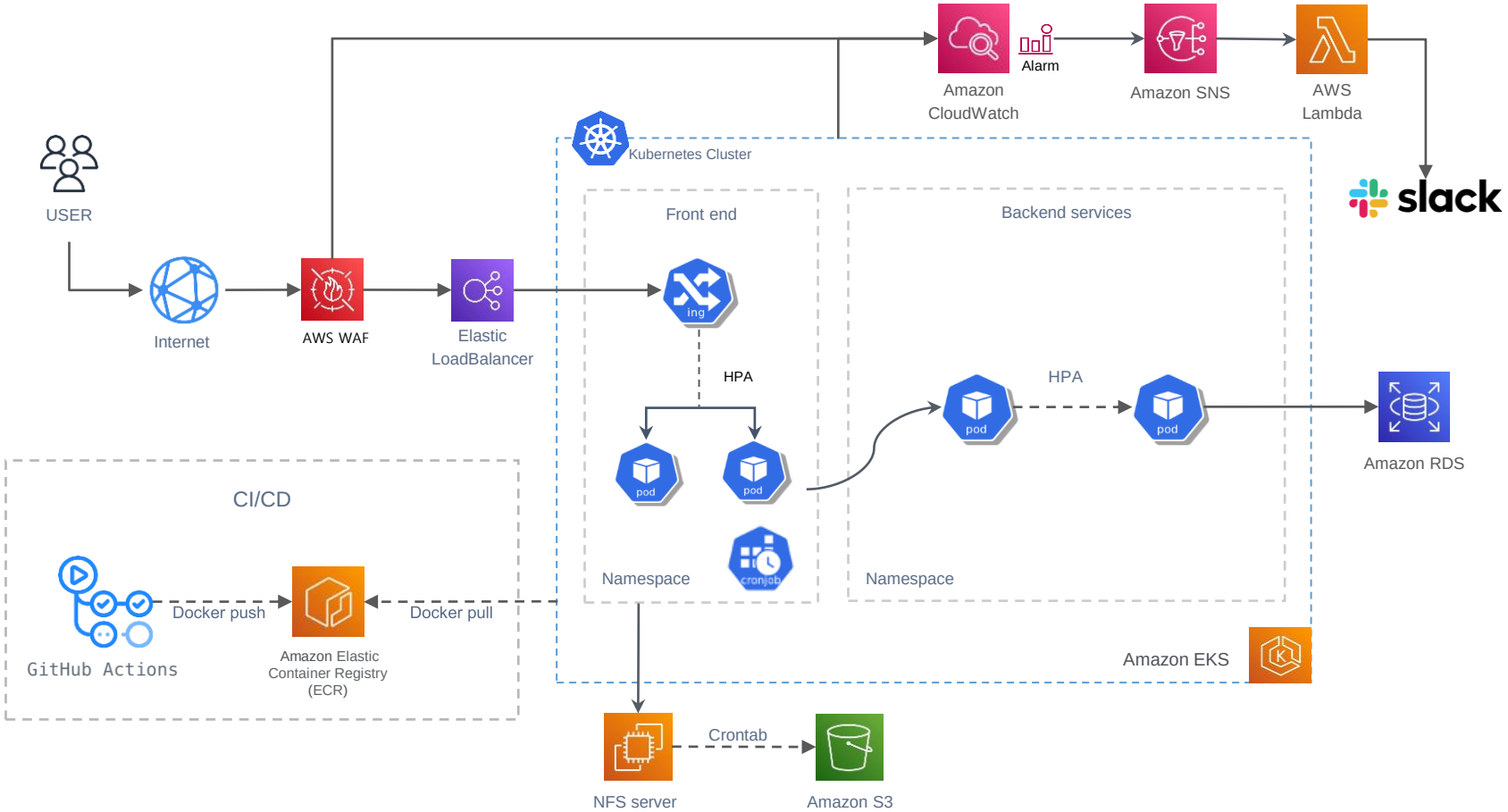
02

Architecture

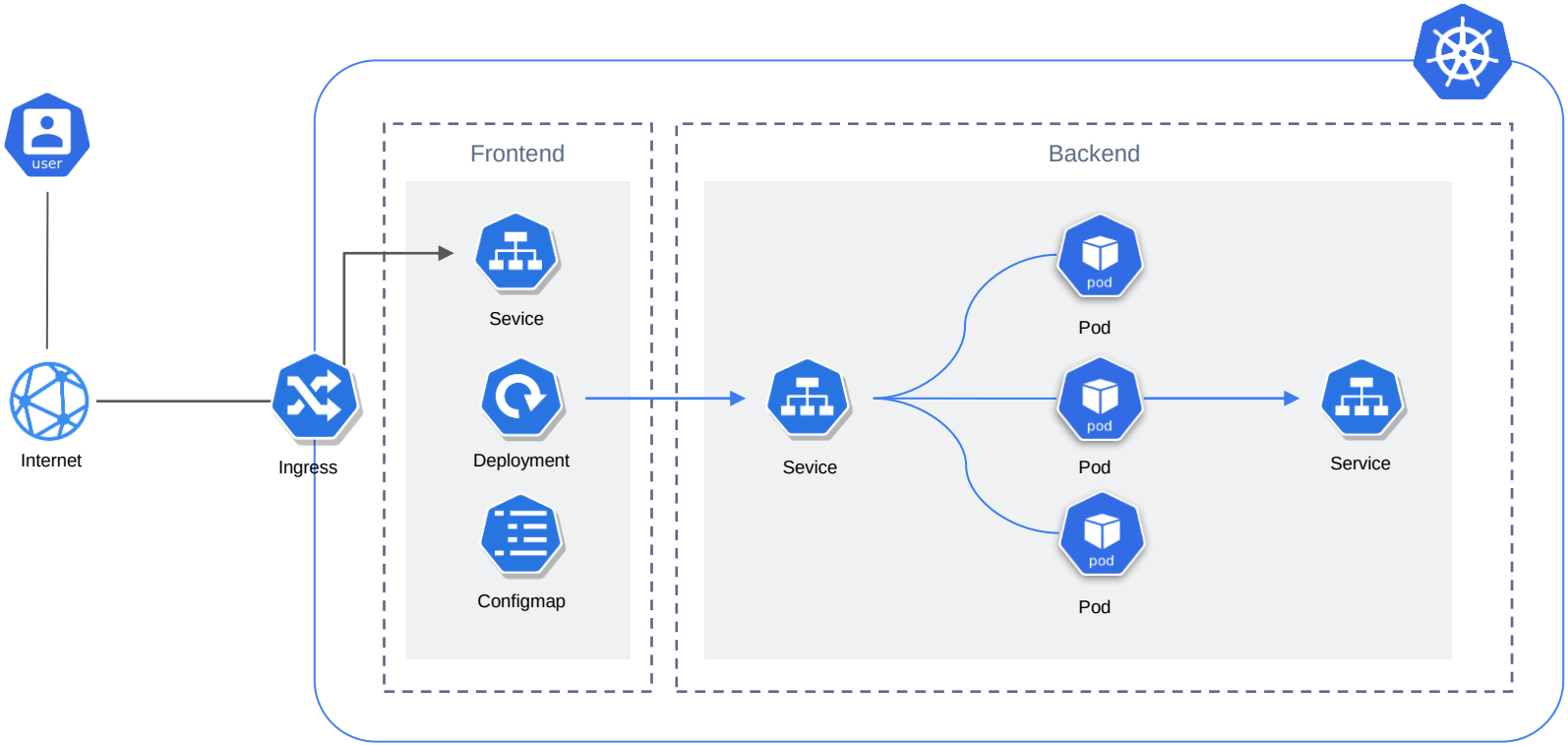
Architecture 구성 원칙




AWS Cloud
S3
Route 53
Amazon CloudWatch
VPC
AWS WAF
ACM
Availability Zone 1
Availability Zone 2
Public subnet
Public subnet
ALB
Private subnet
Private subnet
Node
HPA
Auto Scaling group
Node
HPA
Private subnet
Private subnet
MySQL
MySQL


Amazon CloudWatch
Alarm
Amazon SNS
AWS Lambda
slack
USER
Kubernetes Cluster
Front end
Backend services
Internet
AWS WAF
Elastic LoadBalancer
ing
HPA
HPA
pod
pod
pod
pod
Amazon RDS
CI/CD
cronjob
Namespace
Namespace
Docker push
Docker pull
GitHub Actions
Amazon Elastic Container Registry (ECR)
Amazon EKS
Crontab
NFS server
Amazon S3

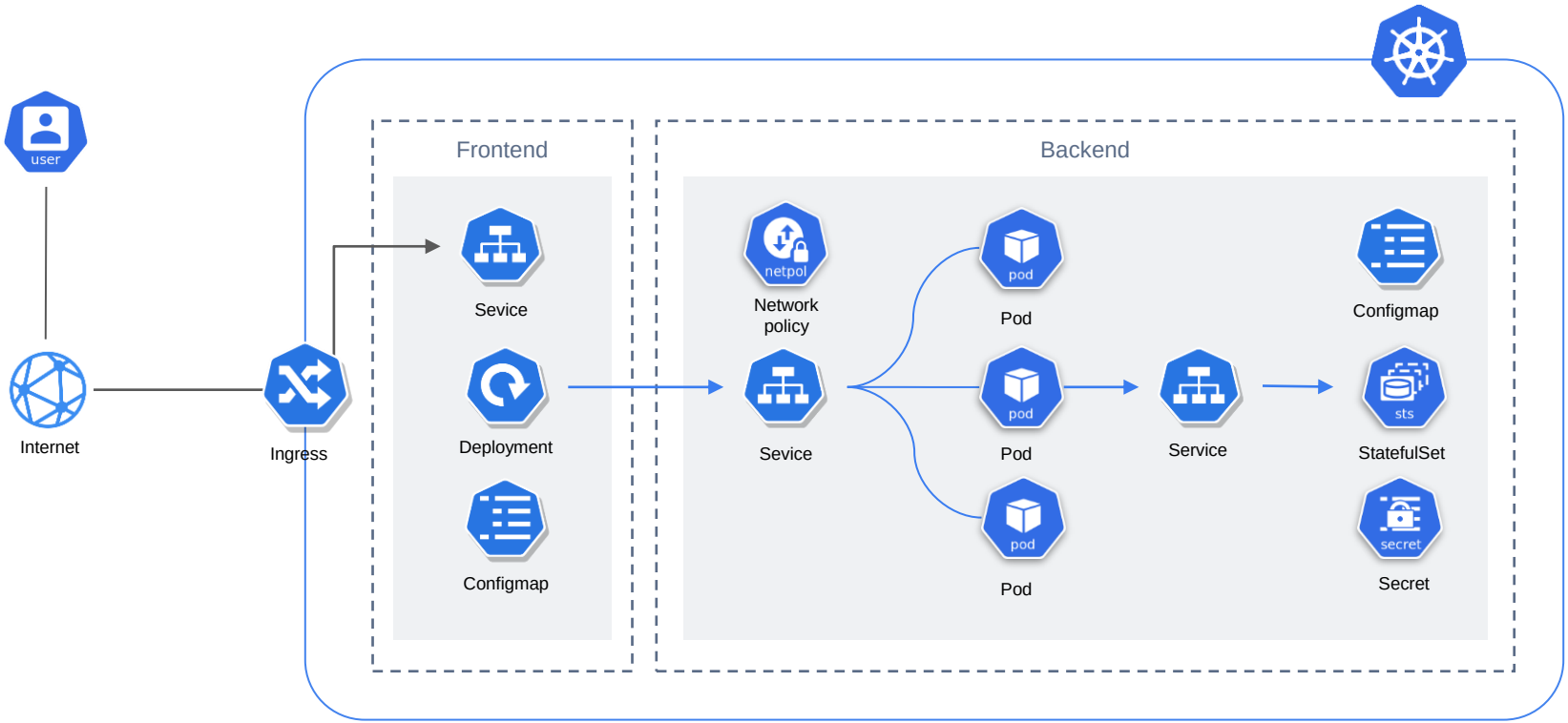
K8S Architecture



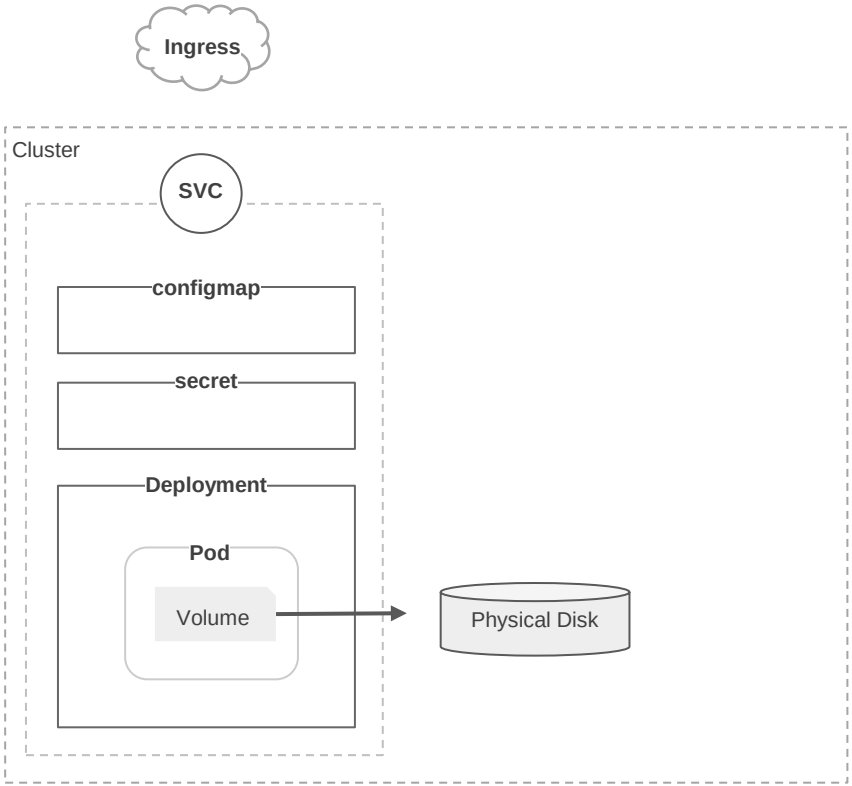
03

개발 환경에서의 구현 기술

K8S Architecture Full Container 환경으로 구성



Kubernetes resource



Ingress
외부에서 쿠버네티스 내부로 들어오는 네트워크 요청을 처리

Service
쿠버네티스 클러스터 외부로 애플리케이션 노출
(Type: NodePort/ClusterIP/LoadBalancer)

Deployment
Pod와 ReplicaSet에 대한 선언적 업데이트 제공

Statefulset
PV, PVC를 이용하여 pod의 고유성을 보장

ConfigMap
컨테이너에 필요한 환경 설정을 컨테이너와 분리해 외부볼륨으로 제공 (하드코딩 지양)

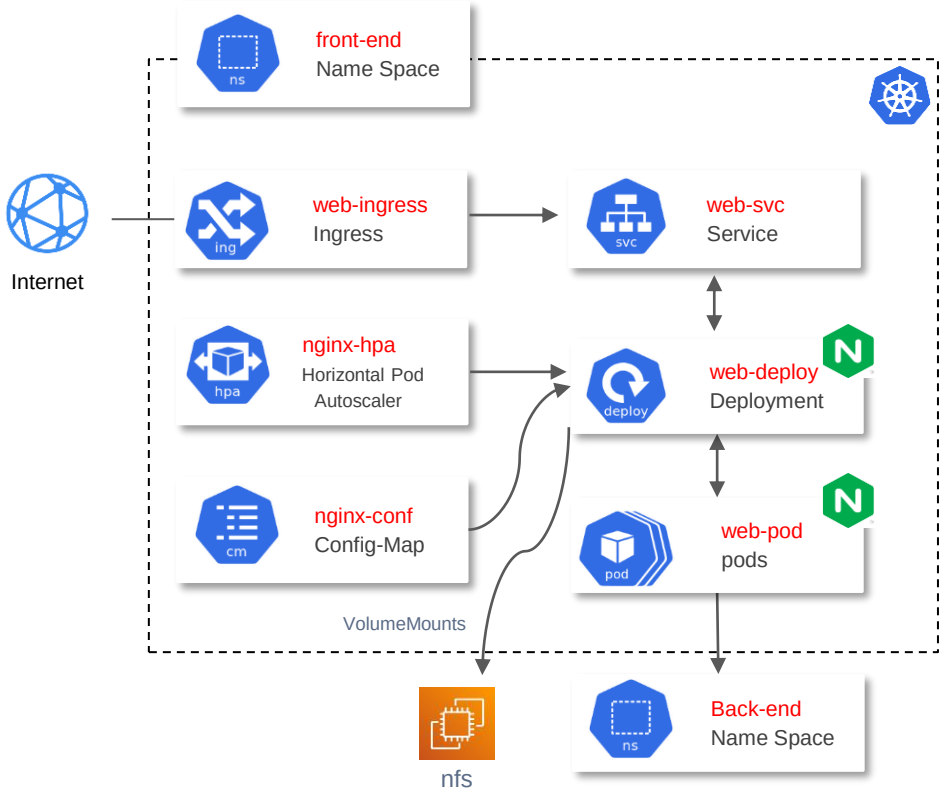
Secret
보안을 신경 써야 하는 설정 정보는 Secret으로 전달 (Base64 인코딩 값 입력)

Namespace
단일 클러스터 내에서의 리소스 그룹 격리 메커니즘을 제공

Cronjob
Job을 크론 형식으로 쓰여진 주어진 일정에 따라 주기적으로 동작하며 백업 용도로 사용

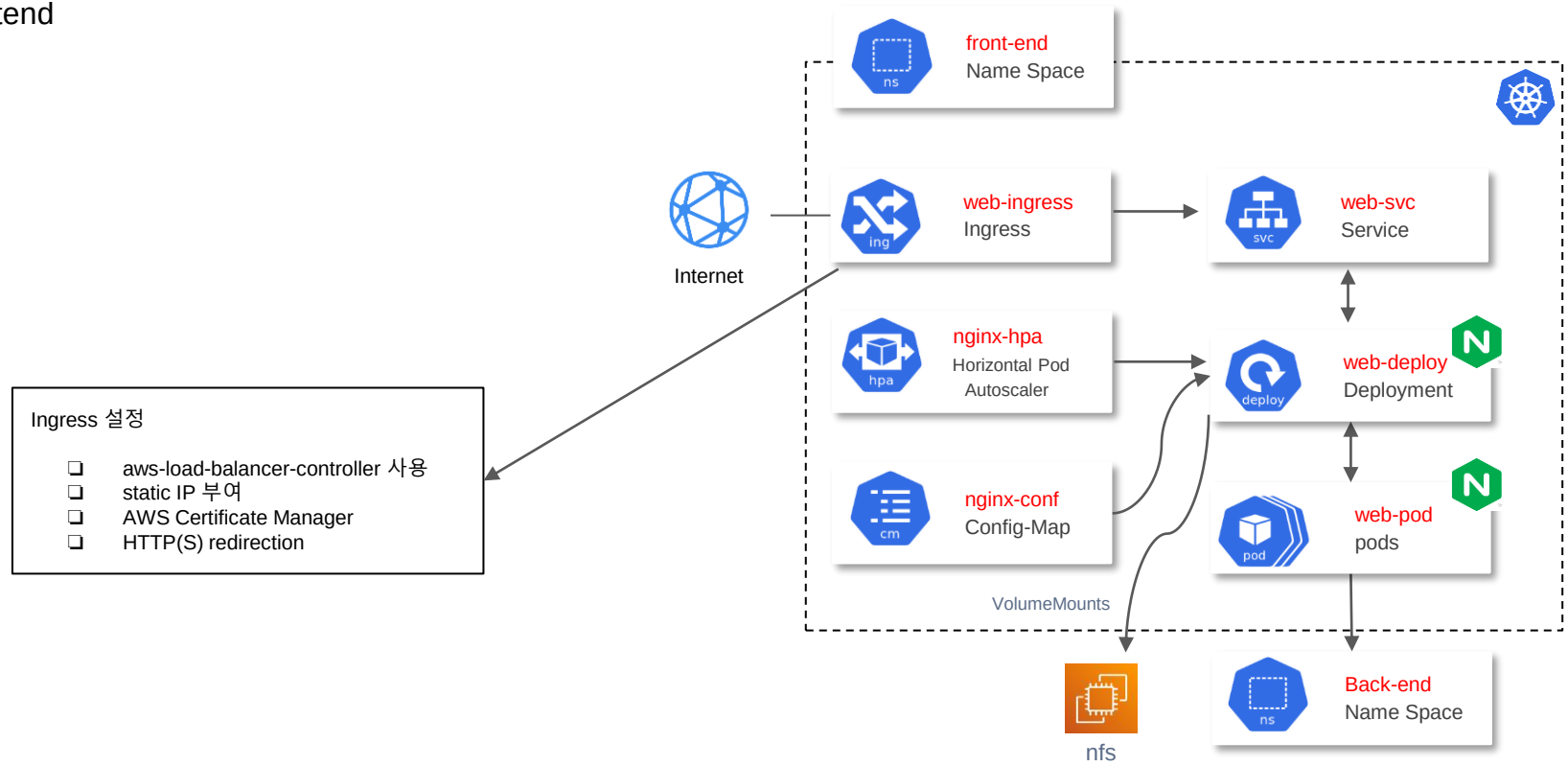
인프라 상세 구성

Frontend



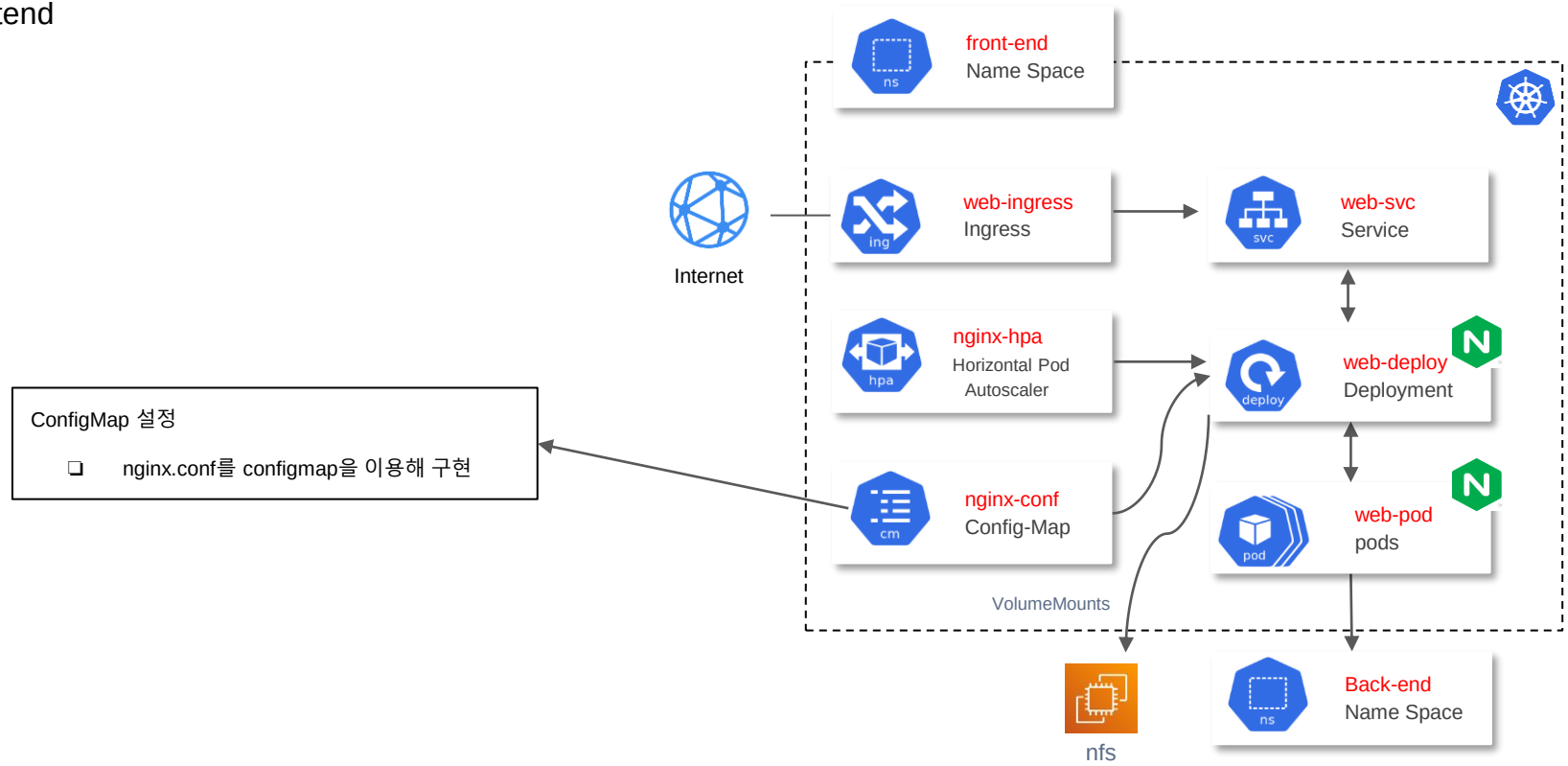
인프라 상세 구성

Frontend



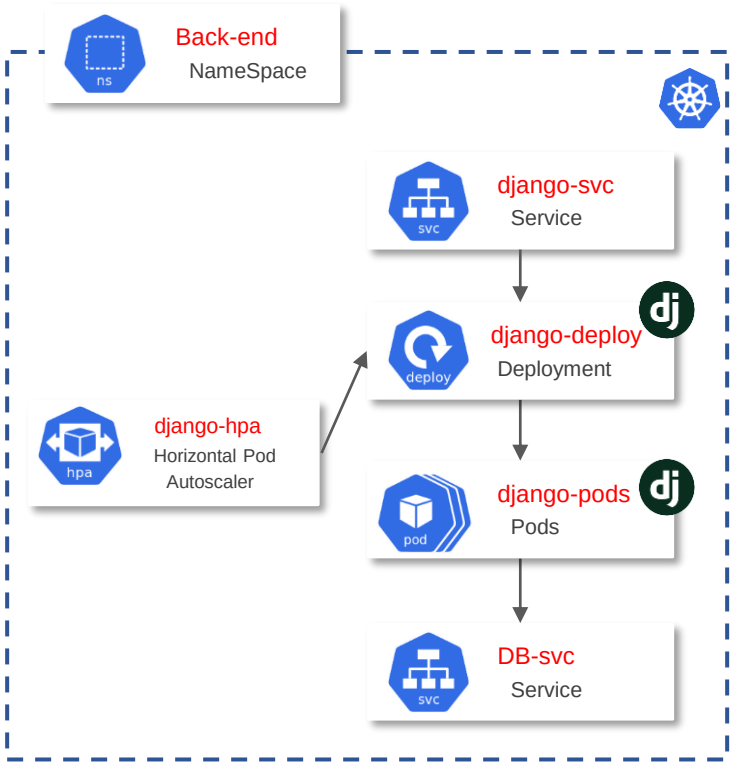
인프라 상세 구성

Frontend



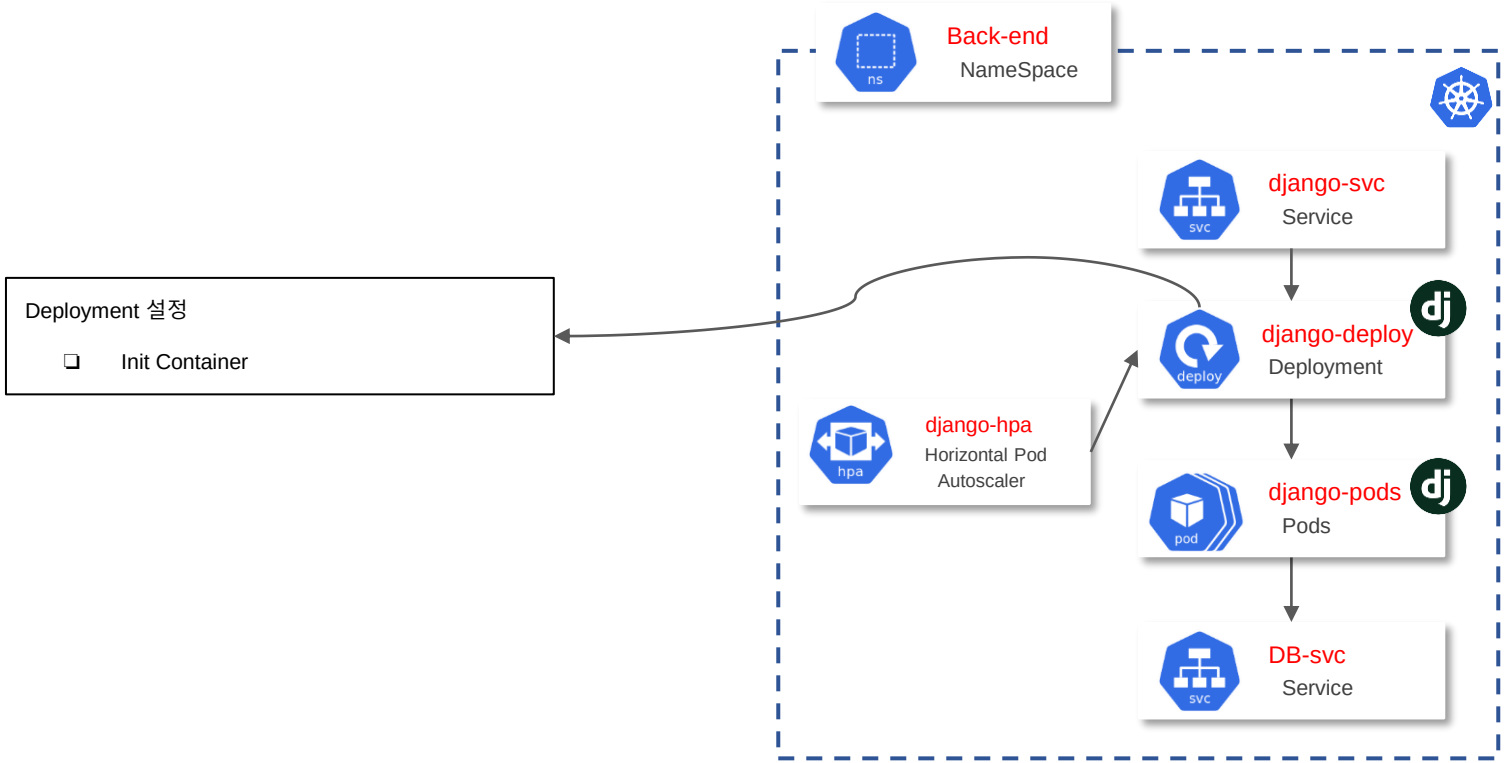
인프라 상세 구성

Backend



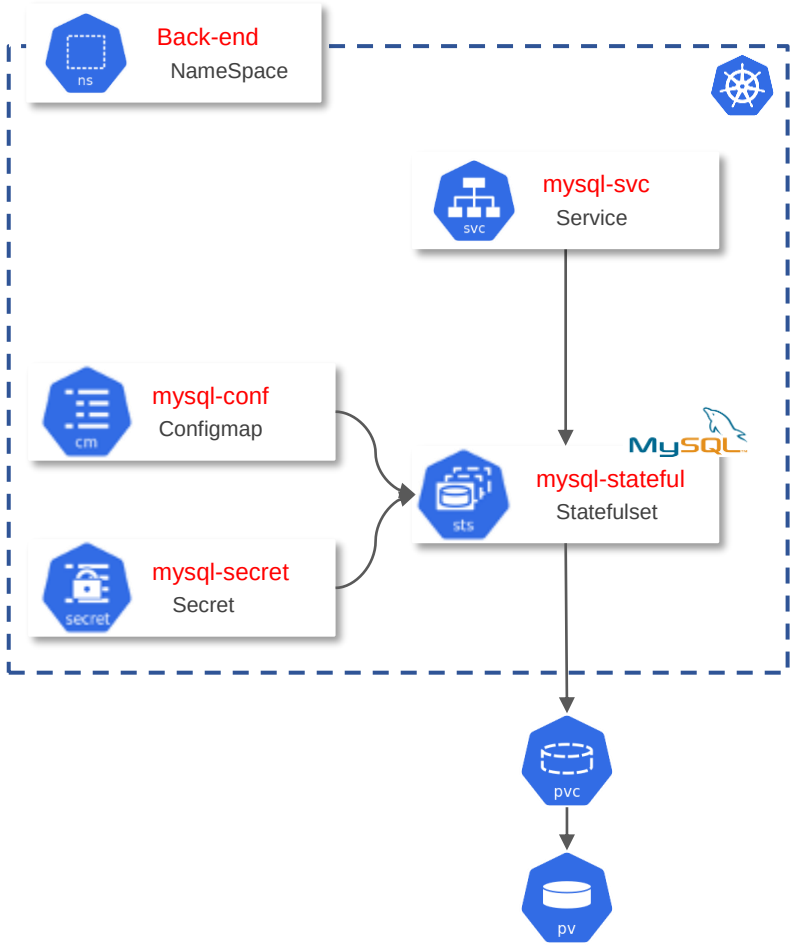
인프라 상세 구성

Backend



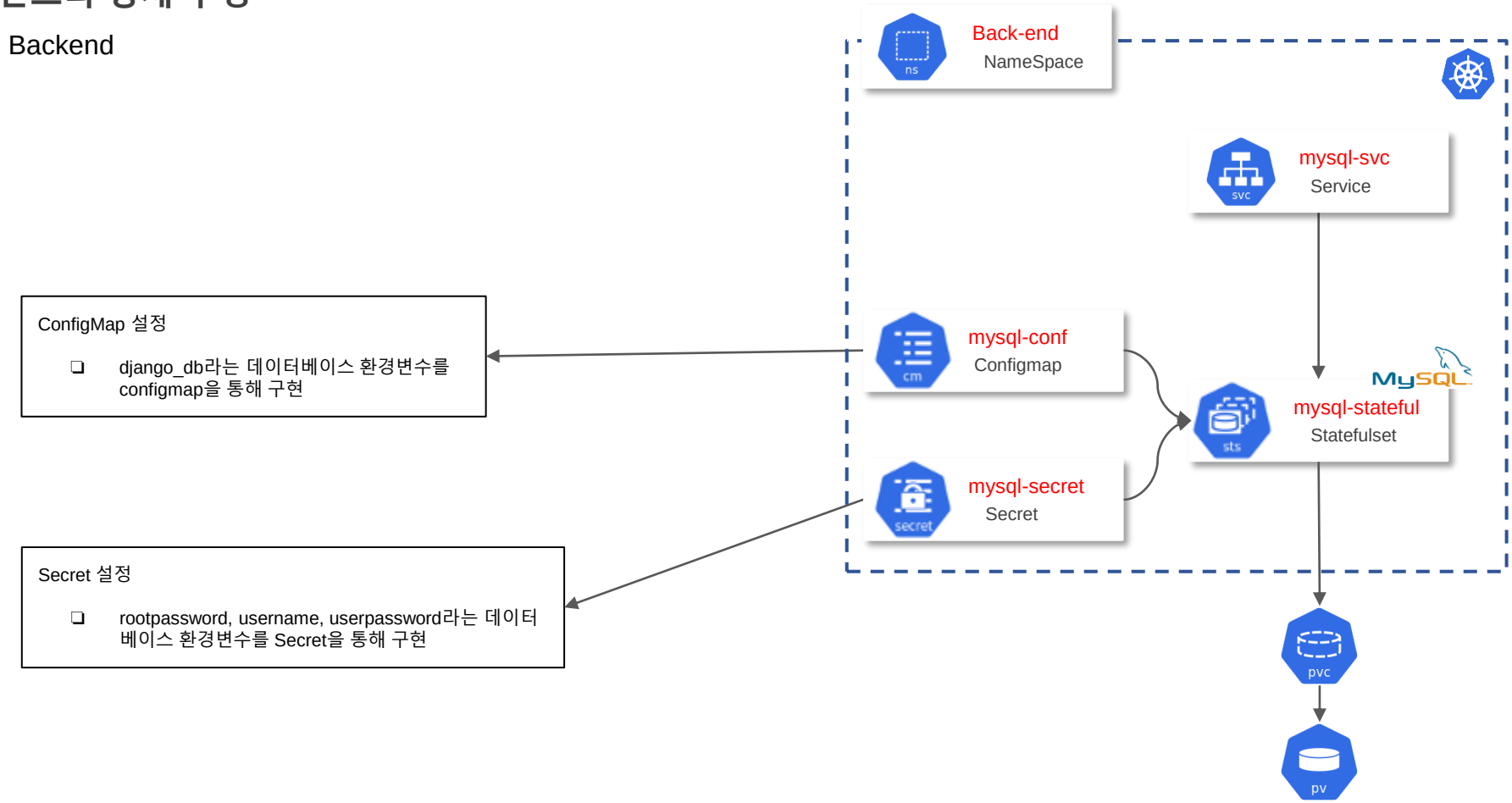
인프라 상세 구성

Backend



인프라 상세 구성

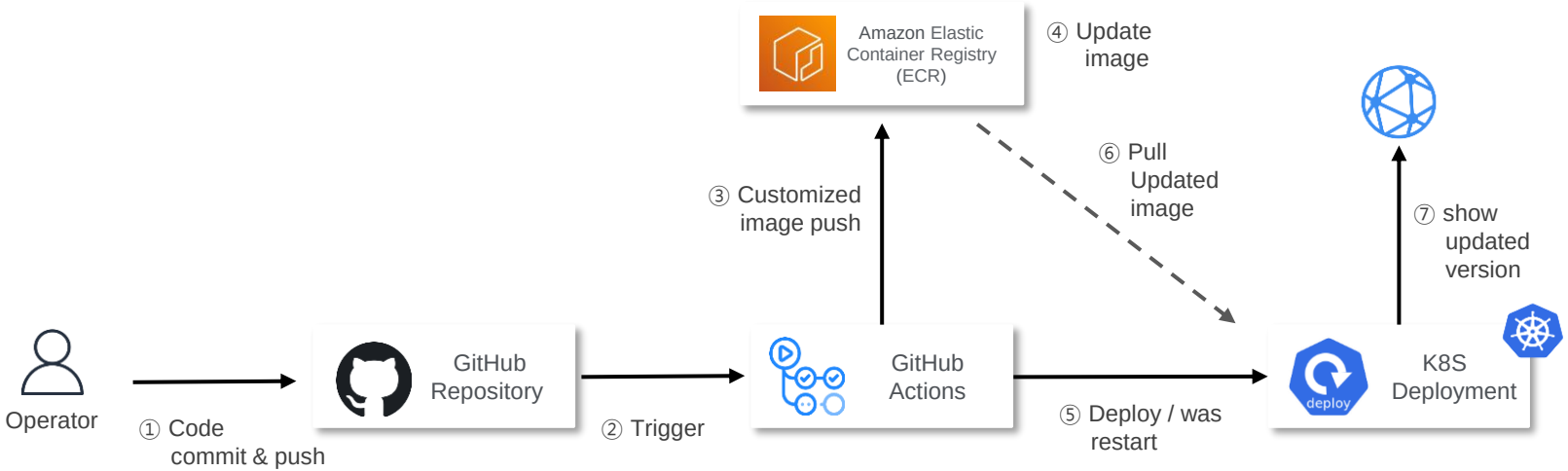
Backend



04

지속 통합 및 배포 (CI/CD)

CI/CD 파이프라인

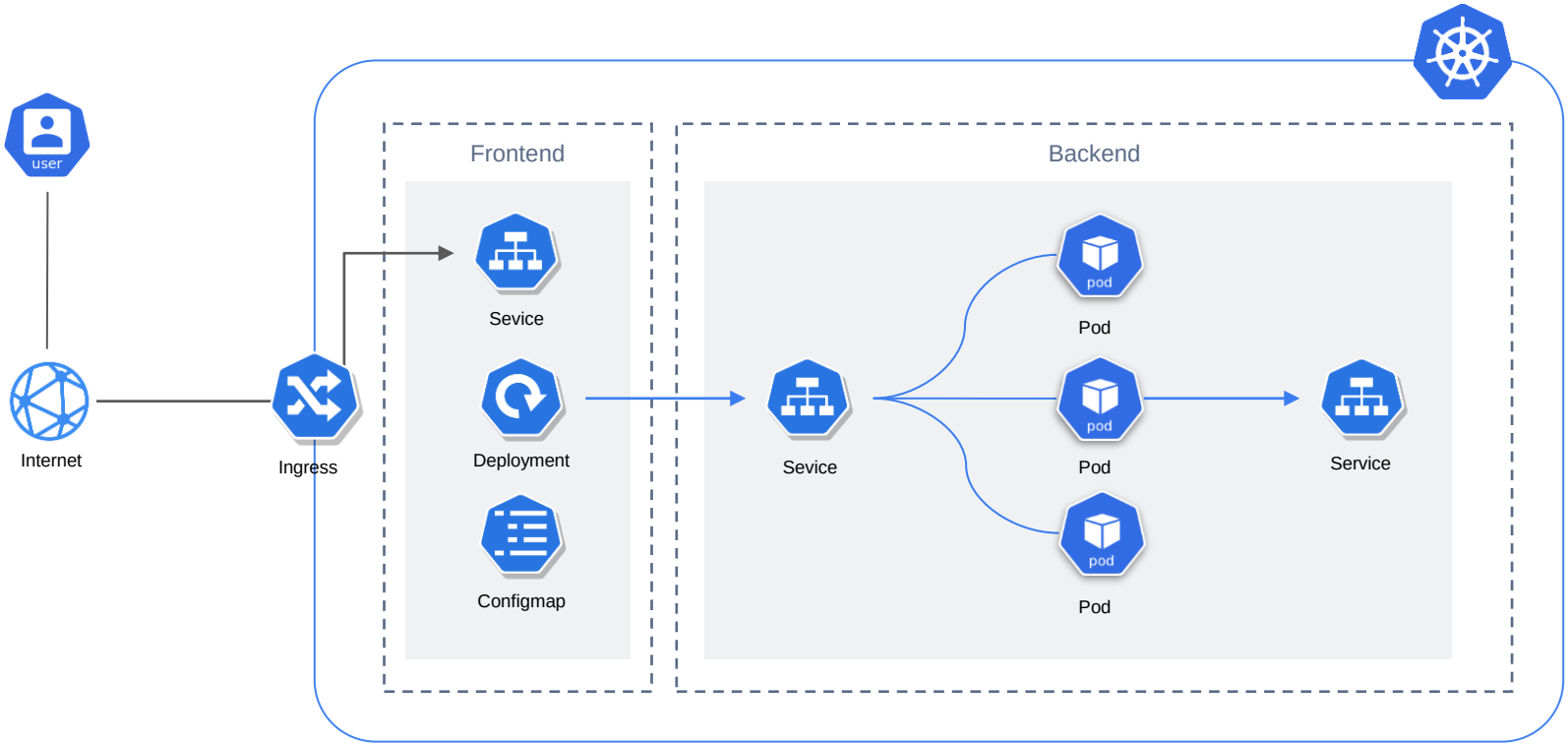

Amazon Elastic Container Registry (ECR)
④ Update image
⑥ Pull Updated image
③ Customized image push
⑦ show updated version
Operator
① Code commit & push
GitHub Repository
② Trigger
GitHub Actions
⑤ Deploy / was restart
deploy
K8S Deployment

(Demo 1) CI/CD 업데이트 데모 영상

05

Product 환경에서의 구현 기술

K8S Architecture

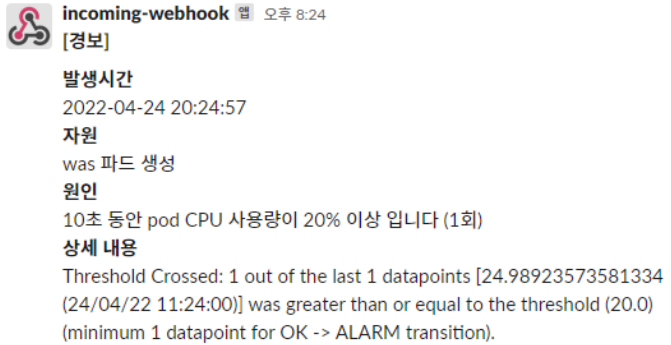


부록

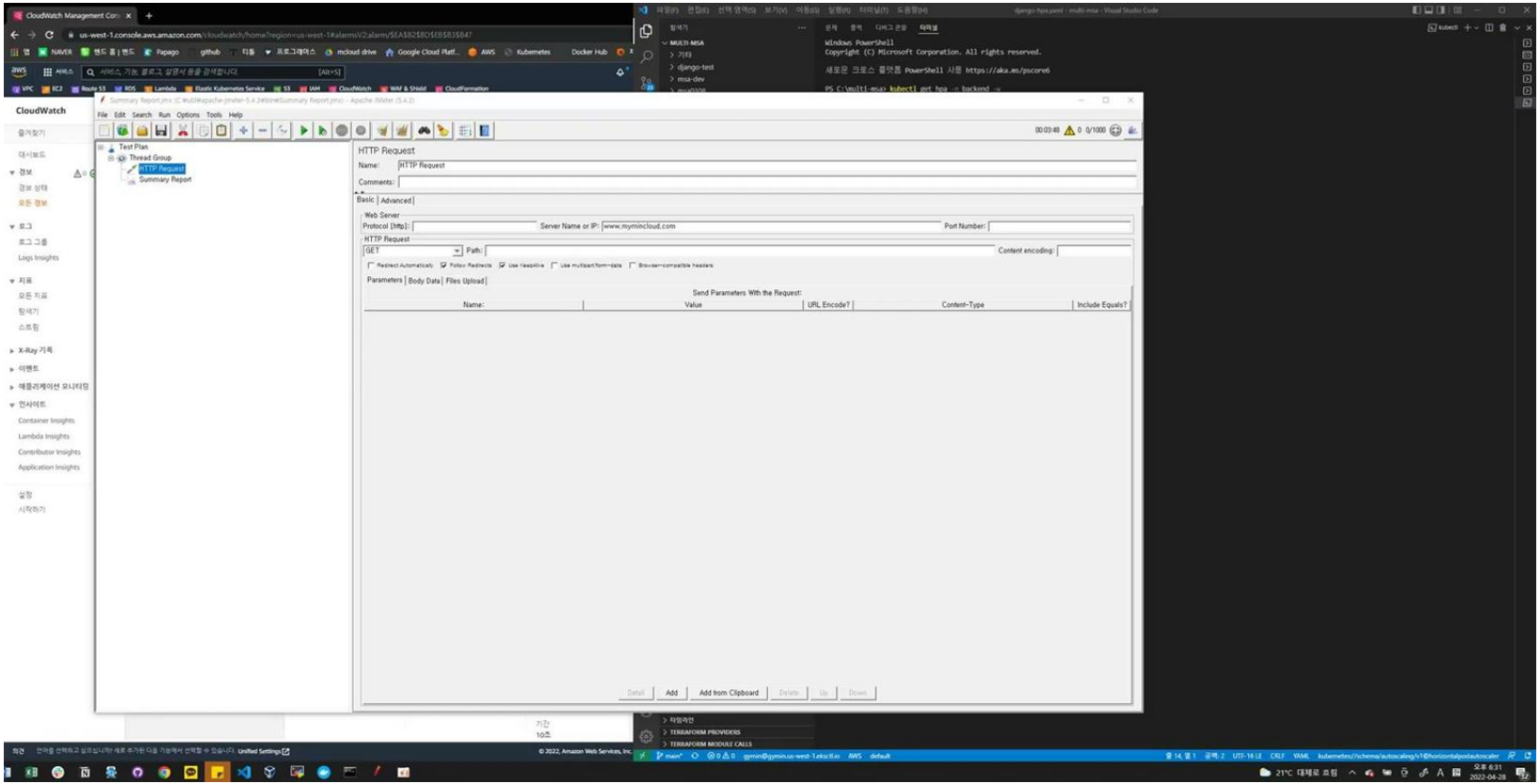
CloudWatch를 이용한 모니터링 - 부하 발생

Lambda 함수

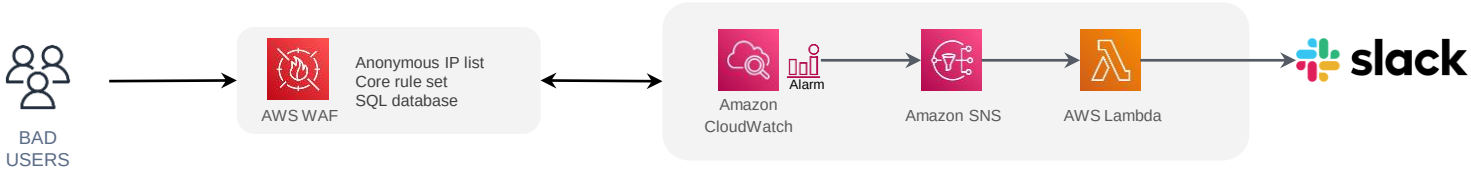
```
import time
from datetime import datetime, timedelta

def lambda_handler(event, context):
    logger.info("Event: " + str(event))
    message = json.loads(event['Records'][0]['Sns']['Message'])
    logger.info("Message: " + str(message))

    alarm_name = message['AlarmName']
    resource_id = message['Trigger']['Dimensions'][0]['value']
    description = message['AlarmDescription']

    th = message['Trigger']['Threshold']
    compar_state = message['Trigger']['ComparisonOperator']
    sec = message['Trigger']['Period']
    count = message['Trigger']['EvaluationPeriods']
    detail_reason = message['NewStateReason']
    # thd= message['NewStateReason'].split(':')[0]


    state_time = message['StateChangeTime'][:19]  # 시간 (형식 '%Y-%m-%dT%H:%M:%S')
    kst_time = datetime.strptime(state_time, '%Y-%m-%dT%H:%M:%S') - timedelta(hours=-9) #KST 시간 변환

    all_des = resource_id+' '+description # was 파드 생성 - cloudwatch 설정 변경에 따라 수정 가능

    cause = 'pod CPU 사용량이 '+ str(int(th))+'% '+compar_state
    mention = str(sec)+'초 동안 '+ cause + ' ('+str(count)+'회)'

    slack_message = {
          'text': "*[%s]*\n\n*발생시간*\n%s \n*자원*\n%s\n *원인*\n%s\n*상세 내용*\n%s" \
          % (alarm_name, str(kst_time), all_des, mention ,detail_reason )
       }
```



(Demo 2) HPA 데모

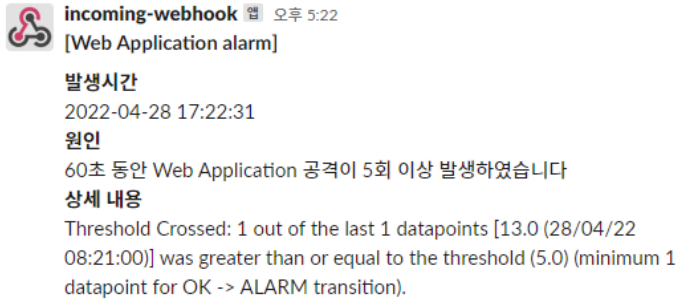

CloudWatch
Test Plan
Thread Group
HTTP Request
Summary Report
HTTP Request
Name: HTTP Request
Comments:
Basic
Advanced
Web Server
Protocol [http]:
Server Name or IP: www.myminicloud.com
Port Number:
HTTP Request
GET
Path:
Content encoding:
Parameters
Body Data
Files Upload
Send Parameters With the Request:
Name:
Value
URL Encode?
Content-Type
Include Equals?
Detail
Add
Add from Clipboard
Delete
Up
Down
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
새로운 크로스 플랫폼 PowerShell 사용 https://aka.ms/pscore6
TERRAFORM PROVIDERS
TERRAFORM MODULE CALLS

CloudWatch를 이용한 모니터링 - Web Application 공격

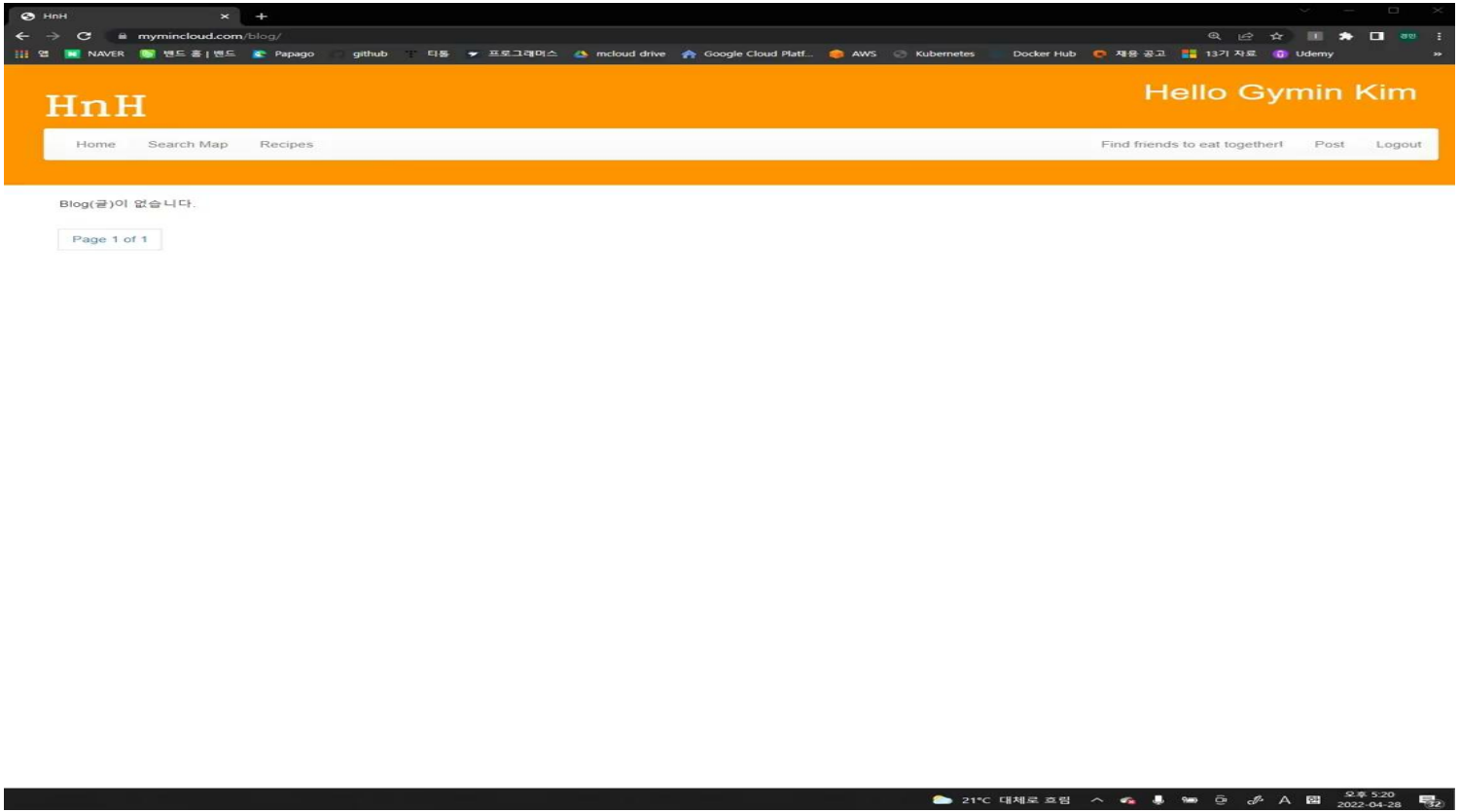


Lambda 함수

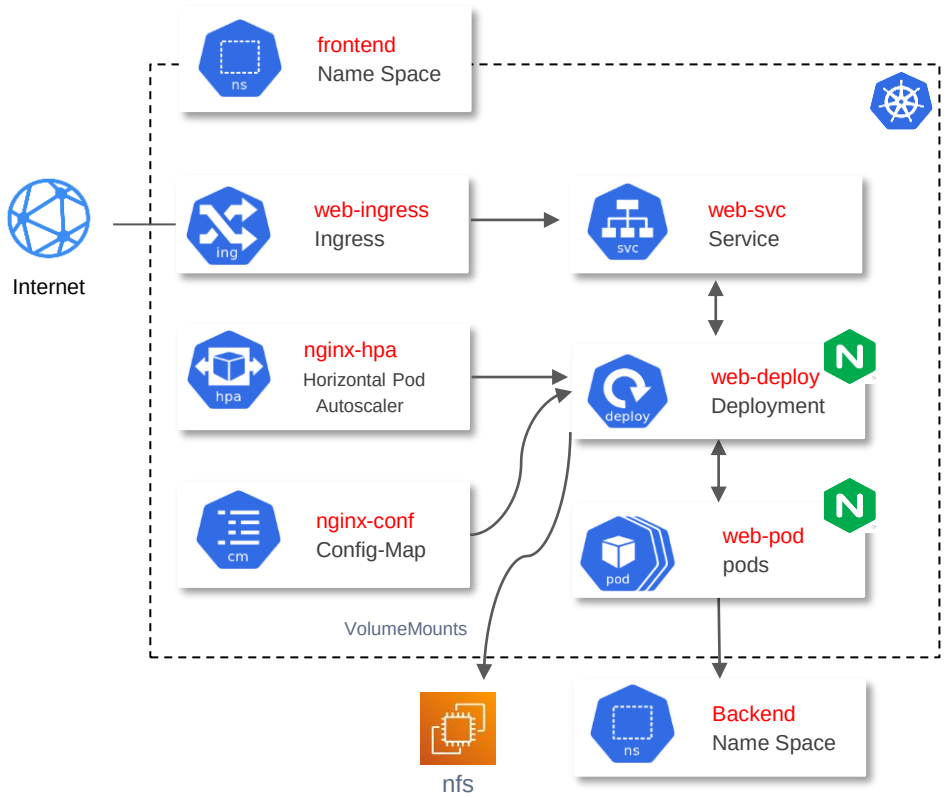
```
import time
from datetime import datetime, timedelta


def lambda_handler(event, context):
    logger.info("Event: " + str(event))
    message = json.loads(event['Records'][0]['Sns']['Message'])
    logger.info("Message: " + str(message))

    alarm_name = message['AlarmName']
    state_time = message['StateChangeTime'][:19]
    kst_time = datetime.strptime(state_time, '%Y-%m-%dT%H:%M:%S') - timedelta(hours=-9) #KST 시간 변환
    sec = message['Trigger']['Period']
    count = message['Trigger']['Threshold']
    detail_reason = message['NewStateReason']
    cause = str(sec)+'초 동안 Web Application 공격이 '+ str(int(count))+'회 이상 발생하였습니다'

    slack_message = {
          'text': "*[%s]*\n\n*발생시간*\n%s \n *원인*\n%s\n*상세 내용*\n%s" \
          % (alarm_name, str(kst_time), cause ,detail_reason )
       }

    req = Request(HOOK_URL, json.dumps(slack_message).encode('utf-8'))
    try:
       response = urlopen(req)
       response.read()
       logger.info("Message posted")
    except HTTPError as e:
       logger.error("Request failed: %d %s", e.code, e.reason)
    except URLError as e:
       logger.error("Server connection failed: %s", e.reason)
```



(Demo 3) Web Application 공격 데모



인프라 상세 구성 - frontend



구성

① **Ingress**
- aws Ingress controller
  - Route 53 으로 host, domain
  - Amazon Certificate Manager

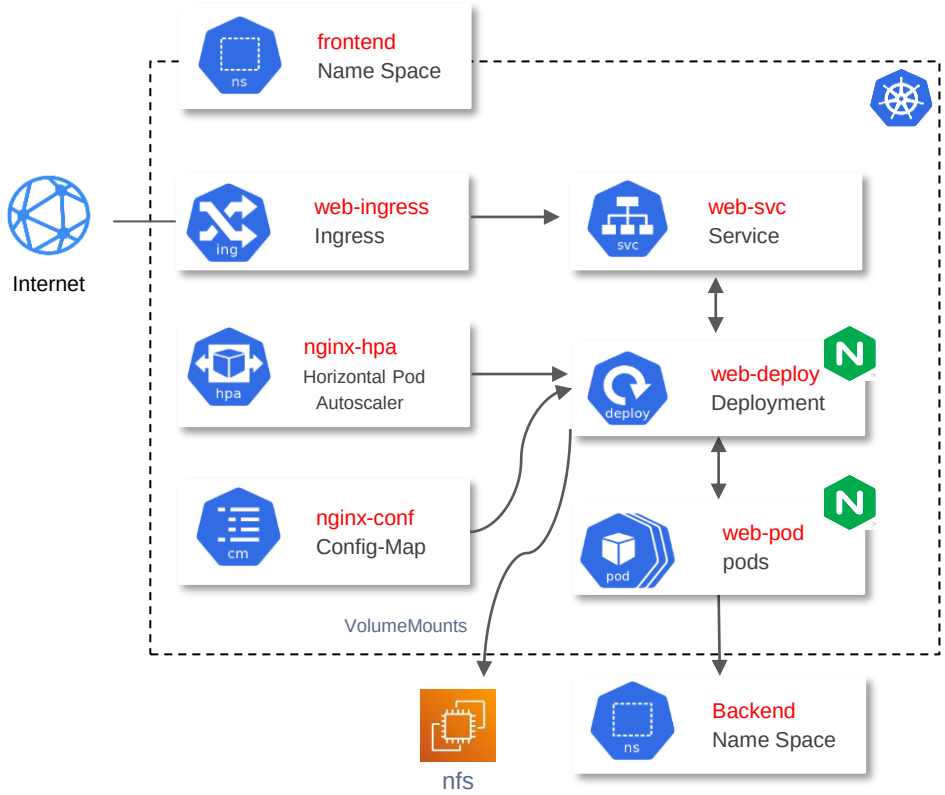
② **Service**
- NodePort type

③ **Deployment**
- replicas로 생성
- VolumeMounts로 nfs에 log 기록
- hpa로 서버 과부하 시 pod 증가

④ **Horizontal Pod Autoscaler**
- 서버 과부하시 pod 수 증가

⑤ **nginx-conf**
- was-svc 접속을 위한 conf 파일

인프라 상세 구성 - frontend



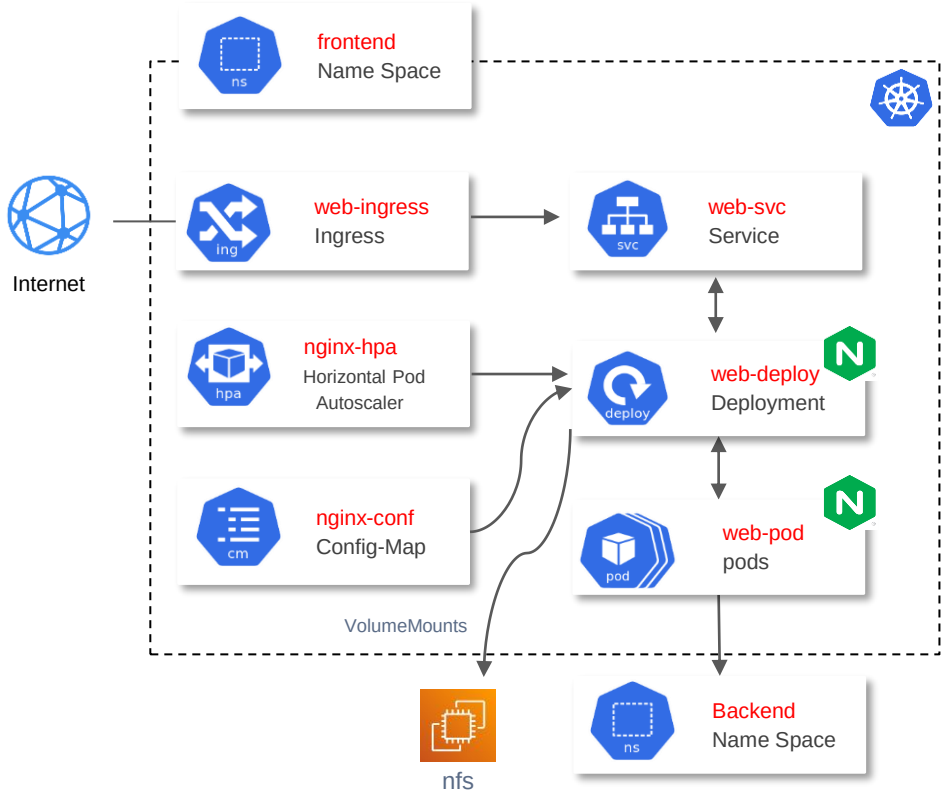
frontend-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name:  frontend-ns
  labels:
    tier: web
```

nginx-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  namespace: frontend-ns
spec:
  selector:
    app: nginx
  ports:
  - nodePort: 30001
    port: 80
    targetPort: 80
  type: NodePort
```

인프라 상세 구성 - frontend

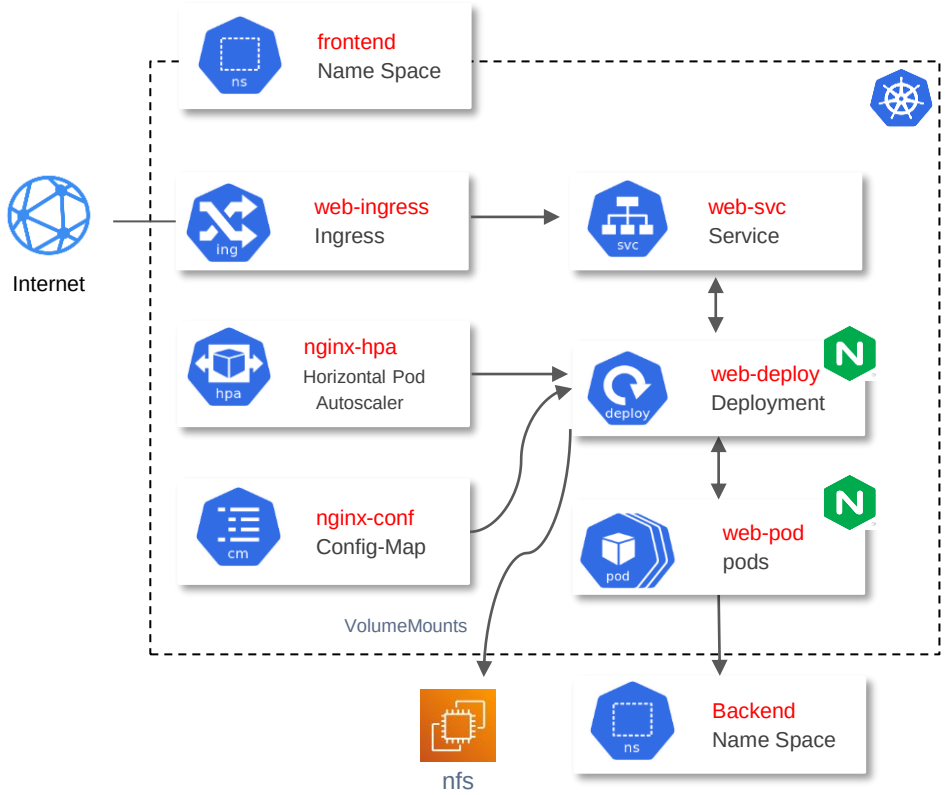


ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myingress
  namespace: frontend-ns
  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-
1:118320467932:certificate/56c80934-7489-4fd3-8bc1-42ddbc271300
    alb.ingress.kubernetes.io/ssl-redirect: '443'
spec:
  rules:
  - host: web1.busb.link
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: nginx-svc
            port:
              number: 80
```

→ **ingress ip가 등록된 도메인**

인프라 상세 구성 - frontend

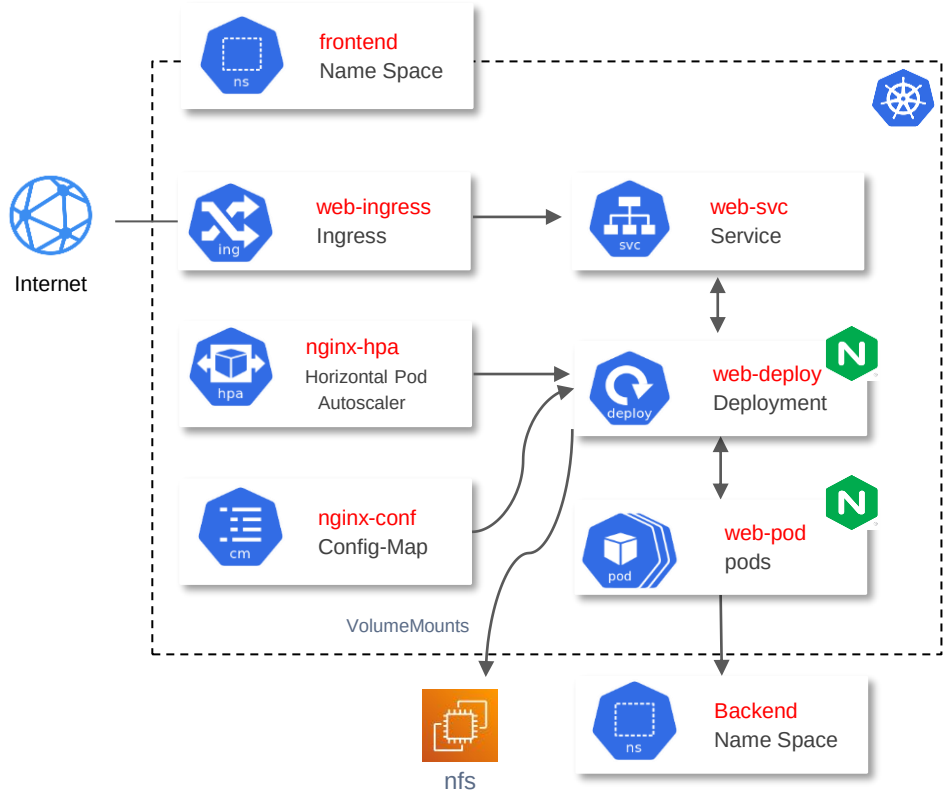


Nginx-conf.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
  namespace: frontend-ns
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.backend-ns: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;
        # Load modular configuration files from the /etc/nginx/conf.d directory.
        # See http://nginx.org/en/docs/ngx_core_module.html#include
        # for more information.
        include /etc/nginx/conf.d/*.conf;
        server {
            listen       80 default_server;
            listen       [::]:80 default_server;
            server_name  _;
            root         /usr/share/nginx/html/;
            # Load configuration files for the default server block.
            include /etc/nginx/default.d/*.conf;
            location / {
            proxy_pass http://backend;
            }
            error_page 404 /404.html;
                location = /40x.html {
            }
            error_page 500 502 503 504 /50x.html;
                location = /50x.html {
            }
        }
    }
```

django-svc 리소스와 연결하기 위한 설정

인프라 상세 구성 - frontend

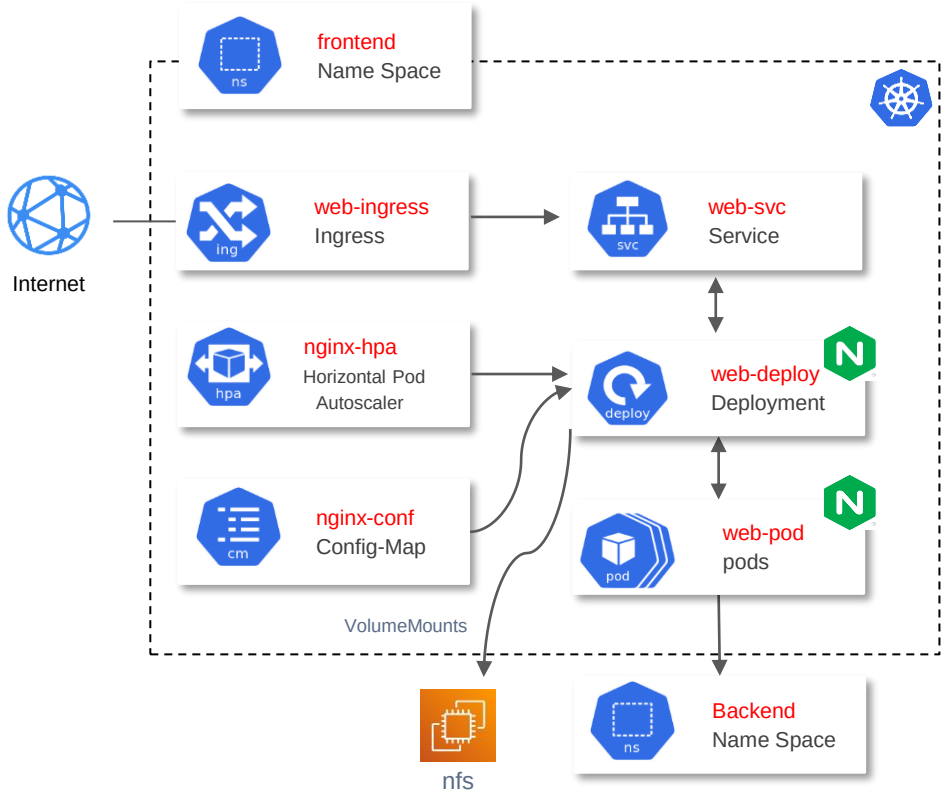


Nginx-deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: frontend-ns
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: kimdowon0419/django:nginx_v1-5
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /var/log/nginx
          name: log
        - mountPath: /etc/nginx/
          name: nginx-conf
      volumes:
       - name: log
         nfs:
           path: /log
           server: 192.168.58.52
      - name: nginx-conf
        configMap:
          name: nginx-conf
```

nfs 서버에 접속 로그 폴더를 마운트 하기 위한 설정

인프라 상세 구성 - frontend



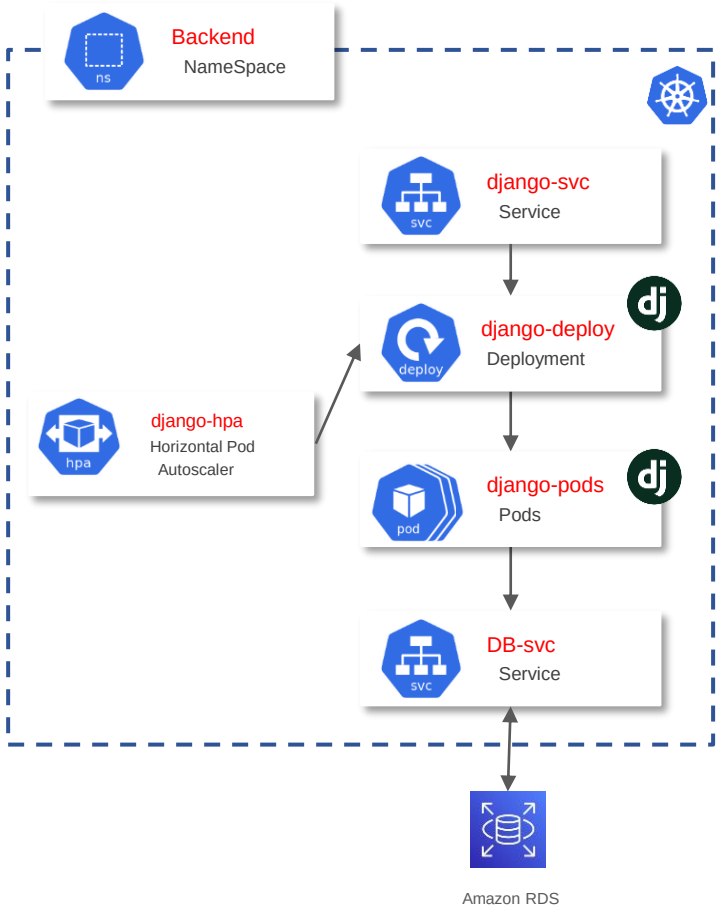
Nginx-hpa.yaml

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
  namespace: frontend-ns
spec:
  maxReplicas: 10
  minReplicas: 3
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  targetCPUUtilizationPercentage: 30
```

생성되는
최대 pod의 개수: 10
최소 pod의 개수: 3

각 pod cpu의 사용률이 30%이 넘어가면 pod의 개수를 늘리는 설정

인프라 상세 구성 - Backend



구성

① **django-svc**
django deployment를 위한 SVC
type: Cluster IP

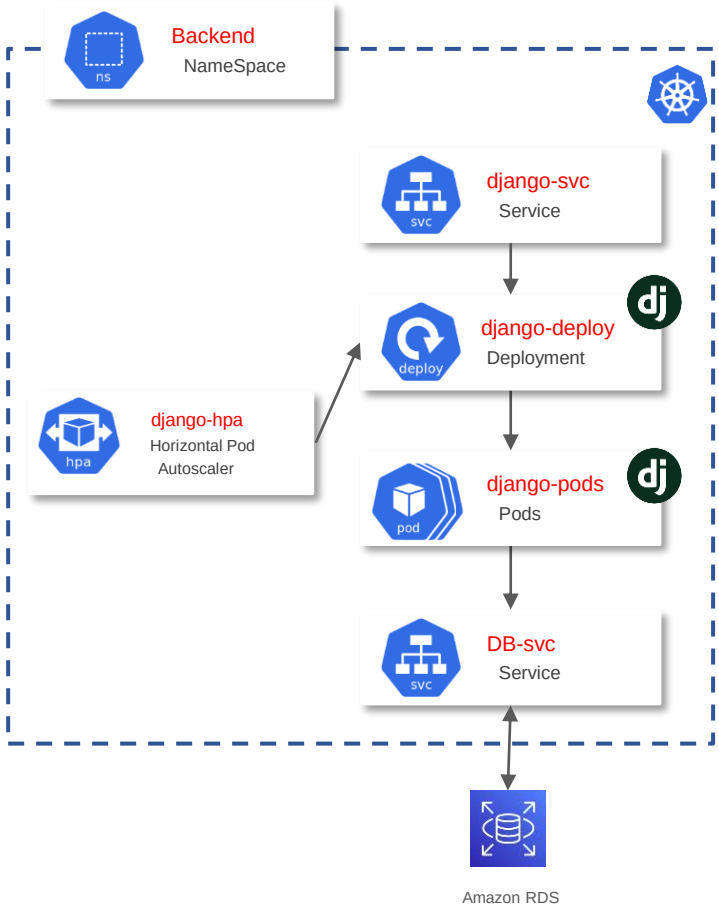
② **django-deployment**
django pods 관리

③ **django-pods**
was로 django 사용

④ **DB-SVC**
Database service
type: Cluster IP

⑤ **Horizontal Pod Autoscaler**
서버 과부하시 pod 수 증가

인프라 상세 구성 - Backend



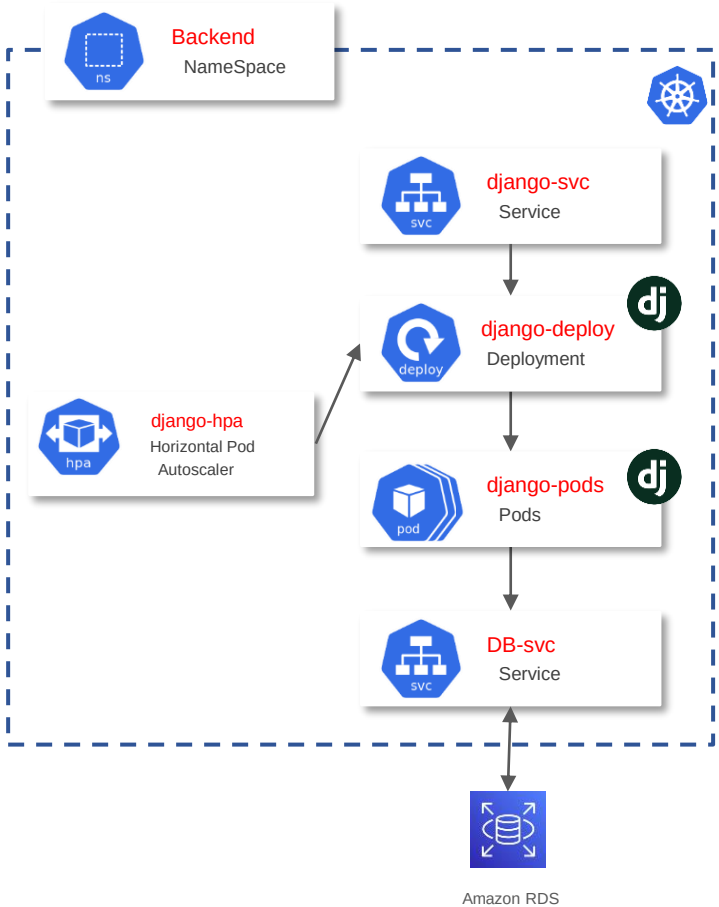
backend-ns.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: backend
  labels:
    tier: was
```

django-svc.yaml

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

WAS-DB



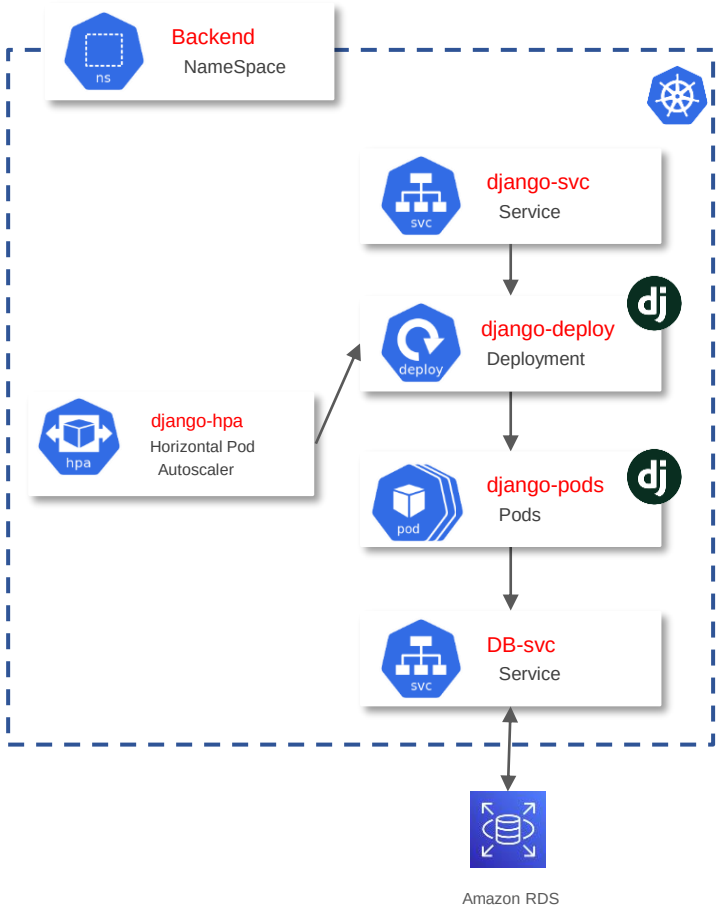
django-deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: was
  namespace: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: django
  template:
    metadata:
      labels:
        app: django
    spec:
      containers:
      - name: was
        image : ${ ECR image URI }
        imagePullPolicy: Always
        resources:
          limits:
            memory: "128Mi"
            cpu: "500m"
        ports:
        - containerPort: 8000
```

ECR private image 사용

HPA 위해 resource 설정

WAS-DB



mysql-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-svc
  namespace: backend
spec:
  type: ExternalName
  externalName: ${ private RDS end point }
  ports:
  - port: 3306
    targetPort: 3306
```

RDS 사용을 위해 externalName 사용

감사합니다