AWS EKS 웹 애플리케이션 구축하기

@Eeap
김수민

Contents

01 실습환경구축

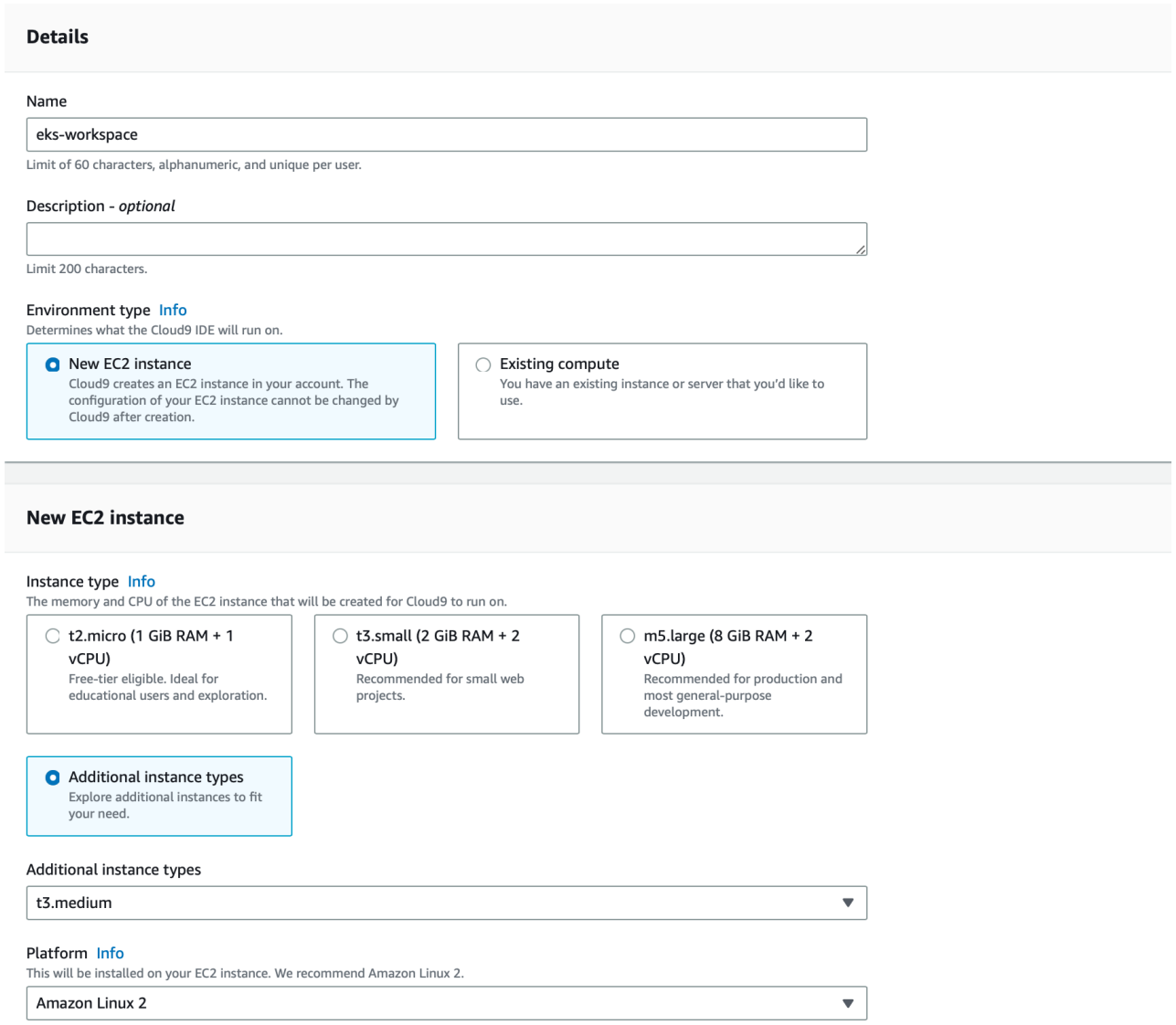
02 Amazon ECR

03 인그레스 컨트롤러 만들기

04 서비스 배포하기

05 AWS Fargate 사용하기

실습 환경 구축 – cloud9 구축



실습 환경 구축 – iam 생성(cloud9에서 사용)

역할 세부 정보

역할 이름
이 역할을 식별하는 의미 있는 이름을 입력합니다.

eksworkspace-admin

최대 64자입니다. 영숫자 및 '+=,.@-_' 문자를 사용하세요.

설명
이 역할에 대하여 간단한 설명을 추가합니다.

Allows EC2 instances to call AWS services on your behalf.

최대 1,000자입니다. 영숫자 및 '+=,.@-_' 문자를 사용하세요.

1단계: 신뢰할 수 있는 엔터티 선택 — 편집

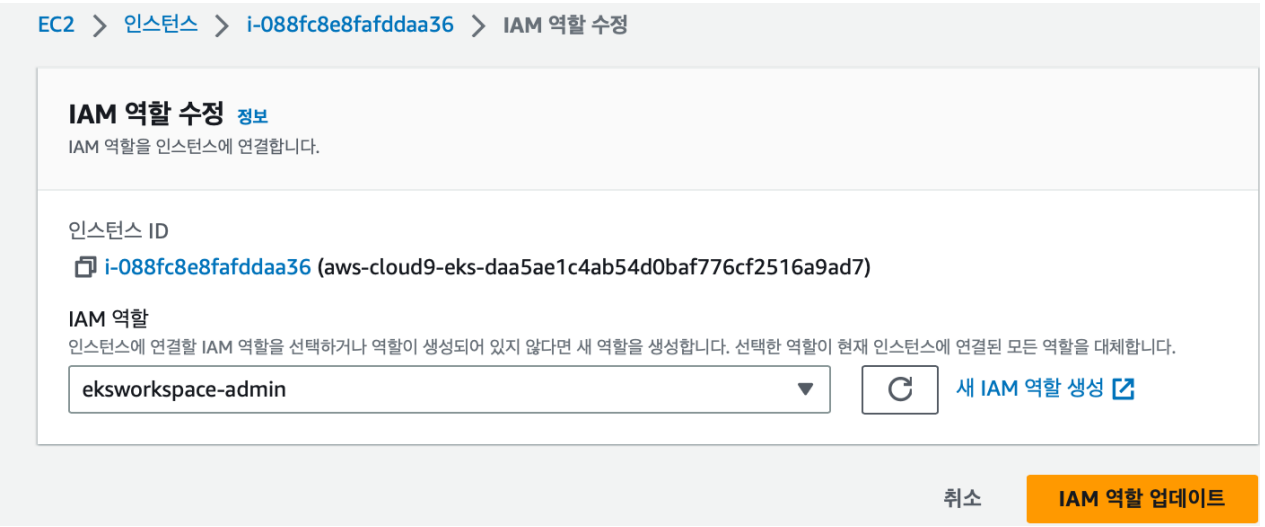
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
```

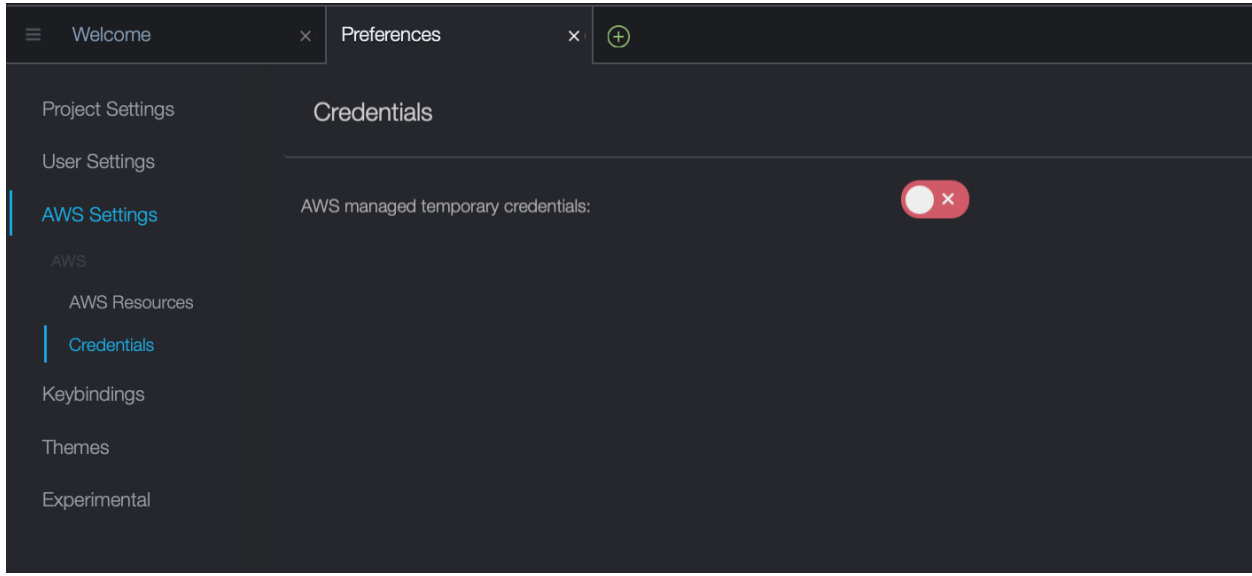
2단계: 권한 추가 — 편집

권한 정책 요약

정책 이름	유형	다음으로서 연결됨
AdministratorAccess	AWS 관리형 - 직무	권한 정책

실습 환경 구축 – iam 연결(ec2 instance에 역할 연결)






```
ec2-user:~/environment $ rm -vf ${HOME}/.aws/credentials
ec2-user:~/environment $ aws sts get-caller-identity --query Arn | grep eksworkspace-admin
"arn:aws:sts::231122965691:assumed-role/eksworkspace-admin/i-088fc8e8fafddaa36"
ec2-user:~/environment $
```

Cloud9에서는 IAM credentials을 동적으로 관리해서
비활성화해주고 우리가 연결한 역할이 잘되는지 테스트 필요!

실습 환경 구축 – aws cli 업데이트

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
export PATH=/usr/local/bin:$PATH
source ~/.bash_profile
```

아래의 명령어를 통해, 버전을 확인합니다. 이때 **2.x version**을 만족해야 합니다.

```
aws --version
```

aws cli 업데이트 필요(기존 버전은 1.x라서 2.x로 업데이트하기 위함)
aws cli는 aws service를 명령어로 구성할 수 있게 해주는 오픈 소스 툴

실습 환경 구축 – kubectl 설치

```
ec2-user:~/environment $ sudo curl -o /usr/local/bin/kubectl  \
>    https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.13/2022-10-31/bin/linux/amd64/kubectl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 44.4M  100 44.4M    0     0  42.0M      0  0:00:01  0:00:01 --:--:-- 42.0M
ec2-user:~/environment $ sudo chmod +x /usr/local/bin/kubectl
ec2-user:~/environment $ kubectl version --client=true --short=true
Client Version: v1.23.13-eks-fb459a0
ec2-user:~/environment $
```

Kubectl 설치는 eks 버전과 상응하도록 설치해줘야함. 그리고 맞게
설치도 잘되었는지 같이 확인!

실습 환경 구축 – 기타 라이브러리 설치(jq 등등...)

```
sudo yum install -y jq
```

```
sudo yum install -y bash-completion
```

```
curl -O https://bootstrap.pypa.io/get-pip.py
python3 get-pip.py --user
```

기타적인 json형식의 데이터를 다루는 커맨드라인 유틸리티나 kubectl
명령의 자동 완성 기능이나, pip의 업데이트를 진행

실습 환경 구축 – eksctl 설치

```
ec2-user:~/environment $ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
ec2-user:~/environment $ sudo mv -v /tmp/eksctl /usr/local/bin
'/tmp/eksctl' -> '/usr/local/bin/eksctl'
ec2-user:~/environment $ eksctl version
0.135.0
ec2-user:~/environment $
```

Eks를 관리하고 생성하기 위해 eksctl cli 툴을 다운로드 해서 해당 바이너리를 폴더로 옮기고 설치 여부 확인.

실습 환경 구축 – iam 연결(ec2 instance에 역할 연결)

```
ec2-user:~/environment $ export AWS_REGION=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')
ec2-user:~/environment $
ec2-user:~/environment $ echo "export AWS_REGION=${AWS_REGION}" | tee -a ~/.bash_profile
export AWS_REGION=us-west-2
ec2-user:~/environment $
ec2-user:~/environment $ aws configure set default.region ${AWS_REGION}
ec2-user:~/environment $ aws configure get default.region
us-west-2
ec2-user:~/environment $ export ACCOUNT_ID=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.accountId')
ec2-user:~/environment $
ec2-user:~/environment $ echo "export ACCOUNT_ID=${ACCOUNT_ID}" | tee -a ~/.bash_profile
```

```
ec2-user:~/environment $ wget https://gist.githubusercontent.com/joozero/b48ee68e2174a4f1ead93aaf2b582090/raw/2dda79390a10328df66e5f6162846017c682bef5/resize.sh
--2023-03-26 07:00:54--  https://gist.githubusercontent.com/joozero/b48ee68e2174a4f1ead93aaf2b582090/raw/2dda79390a10328df66e5f6162846017c682bef5/resize.sh
Resolving gist.githubusercontent.com (gist.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to gist.githubusercontent.com (gist.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1273 (1.2K) [text/plain]
Saving to: 'resize.sh'

100%[=========================================================================================================================================================>]

2023-03-26 07:00:55 (55.8 MB/s) - 'resize.sh' saved [1273/1273]

ec2-user:~/environment $ sh resize.sh
```

Region과 계정 변수 등록하고 cloud9의 디스크 사이즈 변경하는 script 실행

Amazon ECR 이용 – ECR에 이미지 올리고 사용하기

Amazon ECR(Elastic Container Registry)은 repository를 올리고 컨테이너 이미지를 올리는데 사용되는 서비스

AWS IAM을 사용하며 컨테이너 이미지에 액세스할 수 있는 사용자 및 리소스에 대한 권한 제어와 모니터링도 가능

```
git clone https://github.com/joozero/amazon-eks-flask.git
```

```
aws ecr create-repository \
--repository-name demo-flask-backend \
--image-scanning-configuration scanOnPush=true \
--region ${AWS_REGION}
```

```
aws ecr get-login-password --region ${AWS_REGION} | docker login --username AWS --password-stdin $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

```
cd ~/environment/amazon-eks-flask

docker build -t demo-flask-backend .
```

이미지 태그를 지켜야 하는지..?

```
docker tag demo-flask-backend:latest $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-flask-backend:latest
```

```
docker push $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-flask-backend:latest
```

Amazon ECR 이용 – 이미지 올리기

최신 버전의 AWS CLI 및 Docker가 설치되어 있는지 확인합니다. 자세한 내용은 Amazon ECR 시작하기을(를) 참조하세요.

다음 단계를 사용하여 이미지를 인증하고 리포지토리에 푸시합니다. Amazon ECR 자격 증명 헬퍼를 비롯한 추가 레지스트리 인증 방법은 레지스트리 인증을(를) 참조하십시오.

1. 인증 토큰을 검색하고 레지스트리에 대해 Docker 클라이언트를 인증합니다.

   AWS CLI 사용:

   ```
   aws ecr-public get-login-password --region us-east-1 | docker login --username AWS --password-stdin public.ecr.aws/s2n2b3r0
   ```

   참고: AWS CLI을(를) 사용하는 중 오류가 발생하면 최신 버전의 AWS CLI 및 Docker가 설치되어 있는지 확인하세요.

2. 다음 명령을 사용하여 도커 이미지를 빌드합니다. 도커 파일을 처음부터 새로 빌드하는 방법에 대한 자세한 내용은 여기 지침을 참조하십시오. 이미지를 이미 빌드한 경우에는 이 단계를 건너뛸 수 있습니다.

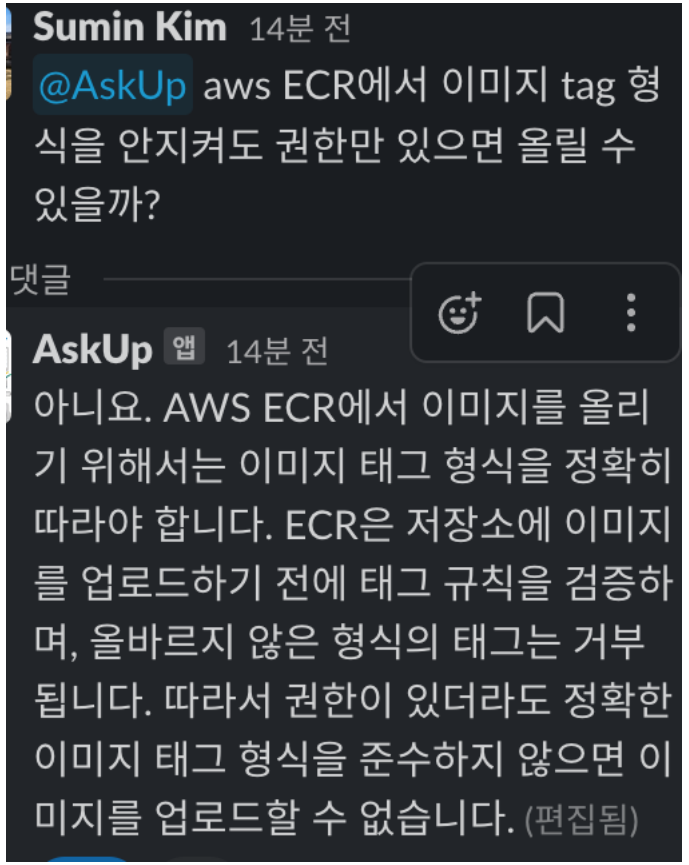
   ```
   docker build -t demo-flask-backend .
   ```

3. 빌드가 완료되면 이미지에 태그를 지정하여 이 리포지토리에 푸시할 수 있습니다.

   ```
   docker tag demo-flask-backend:latest public.ecr.aws/s2n2b3r0/demo-flask-backend:latest
   ```

4. 다음 명령을 실행하여 이 이미지를 새로 생성한 AWS 리포지토리로 푸시합니다.

   ```
   docker push public.ecr.aws/s2n2b3r0/demo-flask-backend:latest
   ```



eksctl을 사용해서 eks 클러스터 만들어보기

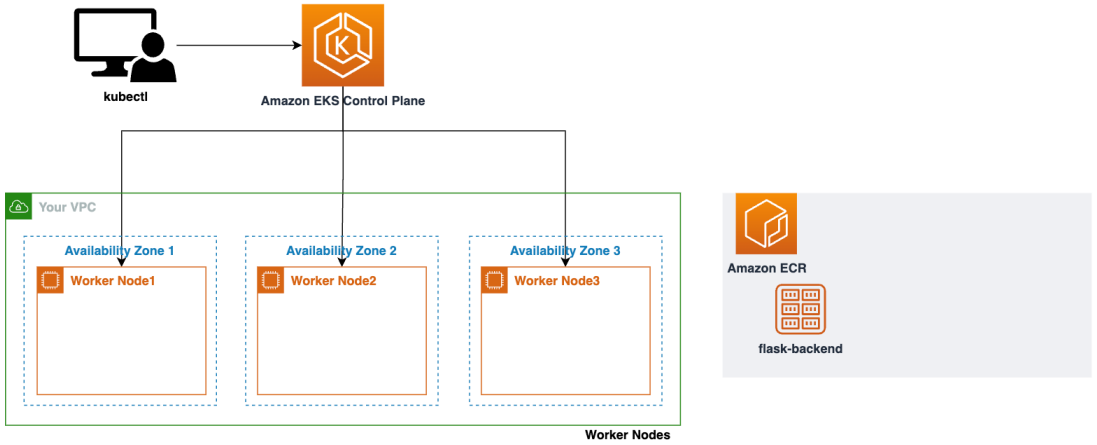
```
cd ~/environment
```

```
cat << EOF > eks-demo-cluster.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: eks-demo # 생성할 EKS 클러스터명
  region: ${AWS_REGION} # 클러스터를 생성할 리전
  version: "1.23"

vpc:
  cidr: "10.0.0.0/16" # 클러스터에서 사용할 VPC의 CIDR
  nat:
    gateway: HighlyAvailable

managedNodeGroups:
  - name: node-group # 클러스터의 노드 그룹명
    instanceType: m5.large # 클러스터 워커 노드의 인스턴스 타입
    desiredCapacity: 3 # 클러스터 워커 노드의 갯수
    volumeSize: 20  # 클러스터 워커 노드의 EBS 용량 (단위: GiB)
    privateNetworking: true
    ssh:
      enableSsm: true
    iam:
      withAddonPolicies:
        imageBuilder: true # Amazon ECR에 대한 권한 추가
        albIngress: true  # albIngress에 대한 권한 추가
        cloudWatch: true # cloudWatch에 대한 권한 추가
        autoScaler: true # auto scaling에 대한 권한 추가
        ebs: true # EBS CSI Driver에 대한 권한 추가

cloudWatch:
  clusterLogging:
    enableTypes: ["*"]

iam:
  withOIDC: true
EOF
```

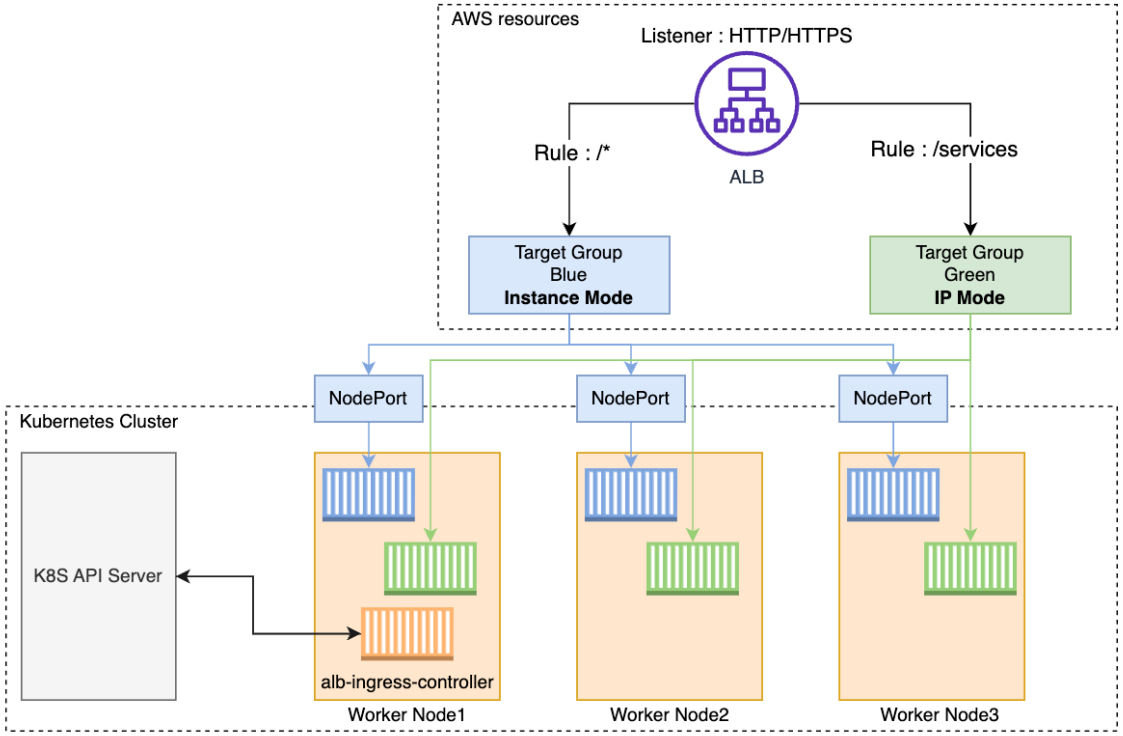
ssm 은 aws systems manager로서 aws 리소스를 업데이트, 관리 및 구성 작업을 용이하게 함.

OIDC는 Oauth같은 인증 관련.

```
eksctl create cluster -f eks-demo-cluster.yaml
```



인그레스 컨트롤러 만들기



Instance mode와 IP mode는 ALB 대상이 되는 리소스의 차이. 전자는 node를 후자는 파드를 대상으로 등록

Eks의 Load Balancer 컨트롤러는 클러스터에 인그레스 자원이 생성될 때에 ALB 및 필요한 자원이 생성되도록 트기러하는 컨트롤러로 Ingress의 경우엔 ALB로 프로비저닝되고 Service의 경우엔 NLB로 프로비저닝된다.

인그레스 컨트롤러 만들기

```
cd ~/environment

mkdir -p manifests/alb-ingress-controller && cd manifests/alb-ingress-controller

# 최종 폴더 위치
/home/ec2-user/environment/manifests/alb-ingress-controller
```

쿠버네티스가 직접 관리하는 사용자 계정인 service account에 IAM role을 사용하기 위해선 클러스터에 대한 IAM OIDC identity provider(인증)가 필요

```
eksctl utils associate-iam-oidc-provider \
    --region ${AWS_REGION} \
    --cluster eks-demo \
    --approve
```

curl -o iam-policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.4.4/docs/install/iam_policy.json

```
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam-policy.json
```

AWS Load Balancer Controller에 부여할 IAM 정책 생성 후 해당 정책을 이용해서 AWS Load Balancer Controller를 위한 ServiceAccount 생성

```
eksctl create iamserviceaccount \
    --cluster eks-demo \
    --namespace kube-system \
    --name aws-load-balancer-controller \
    --attach-policy-arn arn:aws:iam::$ACCOUNT_ID:policy/AWSLoadBalancerControllerIAMPolicy \
    --override-existing-serviceaccounts \
    --approve
```

인그레스 컨트롤러 만들기

```
kubectl apply --validate=false -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml
```

```
wget https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.4.4/v2_4_4_full.yaml
```

먼저 TLS인증서를 자동으로 프로비저닝하고 관리해주는 cert-manager를 설치를 하고 controller yaml파일을 다운로드한 후 cluster-name 같은 것을 변경해주고 ServiceAccount spec 부분을 지워주면 돼요!(sa를 이미 생성을 했기 때문)

```
kubectl apply -f v2_4_4_full.yaml
```

클러스터 내부에서는 필요한 기능들을 위해 실행되는 파드들을 Addon이라고 부르는데 이것들은 디플로이먼트, replication controller 등에 의해 관리. 그리고 해당 pod들이 사용하는 namespace가 kube-system이기 때문에 다음과 같이 –n옵션을 줘서 조회.

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

```
kubectl get sa aws-load-balancer-controller -n kube-system -o yaml
```

서비스 배포하기

```
cd ~/environment/manifests/
```

```
cat <<EOF> flask-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-flask-backend
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-flask-backend
  template:
    metadata:
      labels:
        app: demo-flask-backend
    spec:
      containers:
        - name: demo-flask-backend
          image: $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-flask-backend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
EOF
```

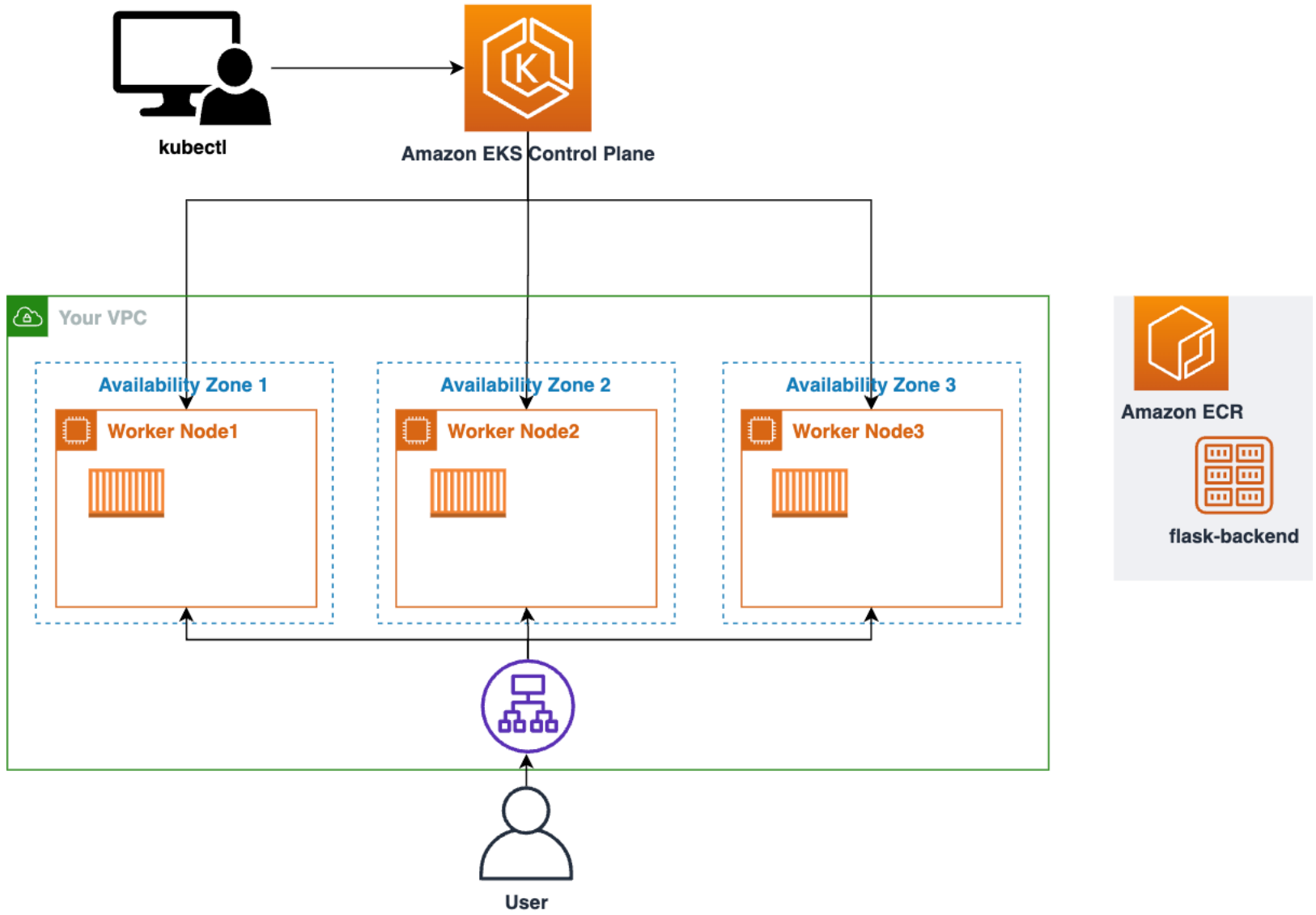
```
cat <<EOF> flask-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: demo-flask-backend
  annotations:
    alb.ingress.kubernetes.io/healthcheck-path: "/contents/aws"
spec:
  selector:
    app: demo-flask-backend
  type: NodePort
  ports:
    - port: 8080 # 서비스가 생성할 포트
      targetPort: 8080 # 서비스가 접근할 pod의 포트
      protocol: TCP
EOF
```

```
cat <<EOF> flask-ingress.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
    name: "flask-backend-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/group.name: eks-demo-group
      alb.ingress.kubernetes.io/group.order: '1'
spec:
    rules:
    - http:
        paths:
          - path: /contents
            pathType: Prefix
            backend:
              service:
                name: "demo-flask-backend"
                port:
                  number: 8080
EOF
```

```
kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
kubectl apply -f flask-ingress.yaml
```

```
echo http://$(kubectl get ingress/flask-backend-ingress -o
jsonpath='{.status.loadBalancer.ingress[*].hostname}')/contents/aws
```

서비스 배포하기


kubectl
Amazon EKS Control Plane
Your VPC
Availability Zone 1
Availability Zone 2
Availability Zone 3
Worker Node1
Worker Node2
Worker Node3
Amazon ECR
flask-backend
User

서비스 배포하기

```
cat <<EOF> nodejs-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-nodejs-backend
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-nodejs-backend
  template:
    metadata:
      labels:
        app: demo-nodejs-backend
    spec:
      containers:
        - name: demo-nodejs-backend
          image: public.ecr.aws/y7c9e1d2/joozero-repo:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 3000
EOF
```

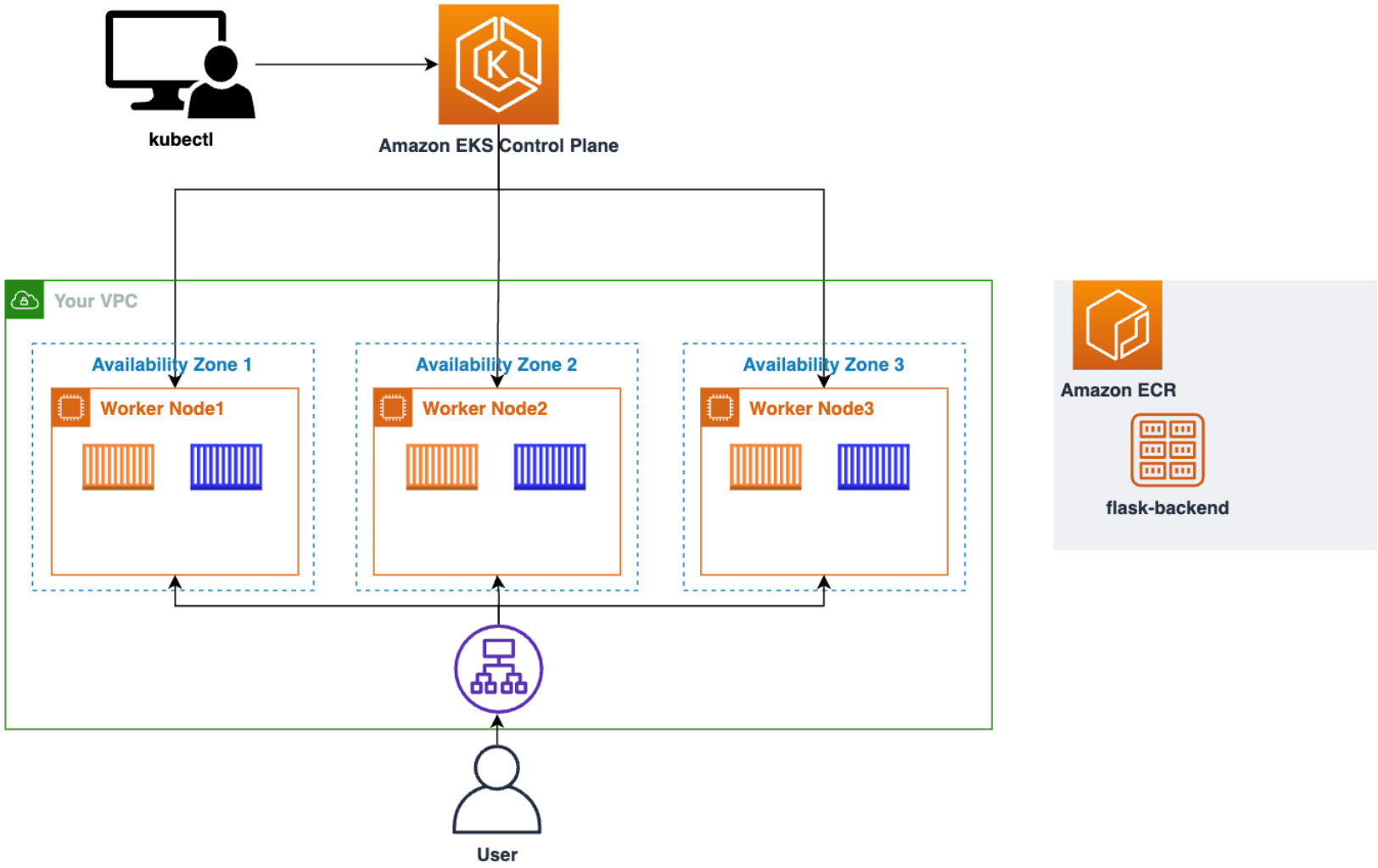
```
cat <<EOF> nodejs-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: demo-nodejs-backend
  annotations:
    alb.ingress.kubernetes.io/healthcheck-path: "/services/all"
spec:
  selector:
    app: demo-nodejs-backend
  type: NodePort
  ports:
    - port: 8080
      targetPort: 3000
      protocol: TCP
EOF
```

```
cat <<EOF> nodejs-ingress.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: "nodejs-backend-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/group.name: eks-demo-group
    alb.ingress.kubernetes.io/group.order: '2'
spec:
  rules:
  - http:
        paths:
          - path: /services
            pathType: Prefix
            backend:
              service:
                name: "demo-nodejs-backend"
                port:
                  number: 8080
EOF
```

```
kubectl apply -f nodejs-deployment.yaml
kubectl apply -f nodejs-service.yaml
kubectl apply -f nodejs-ingress.yaml
```

```
echo http://$(kubectl get ingress/nodejs-backend-ingress -o
jsonpath='{.status.loadBalancer.ingress[*].hostname}')/services/all
```

서비스 배포하기


kubectl
Amazon EKS Control Plane
Your VPC
Availability Zone 1
Availability Zone 2
Availability Zone 3
Worker Node1
Worker Node2
Worker Node3
Amazon ECR
flask-backend
User

서비스 배포하기

```
cd /home/ec2-user/environment
git clone https://github.com/joozero/amazon-eks-frontend.git
```

여기서 받은 파일 중 몇가지의 js파일에서 url을 앞에서 만든 ingress 주소로 변경해야함

```
aws ecr create-repository \
--repository-name demo-frontend \
--image-scanning-configuration scanOnPush=true \
--region ${AWS_REGION}
```

```
cd /home/ec2-user/environment/amazon-eks-frontend
npm install
npm run build
```

```
docker build -t demo-frontend .

docker tag demo-frontend:latest $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-frontend:latest
```

```
docker push $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-frontend:latest
```

서비스 배포하기

```
cd /home/ec2-user/environment/manifests

cat <<EOF> frontend-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-frontend
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-frontend
  template:
    metadata:
      labels:
        app: demo-frontend
    spec:
      containers:
        - name: demo-frontend
          image: $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-frontend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
EOF
```

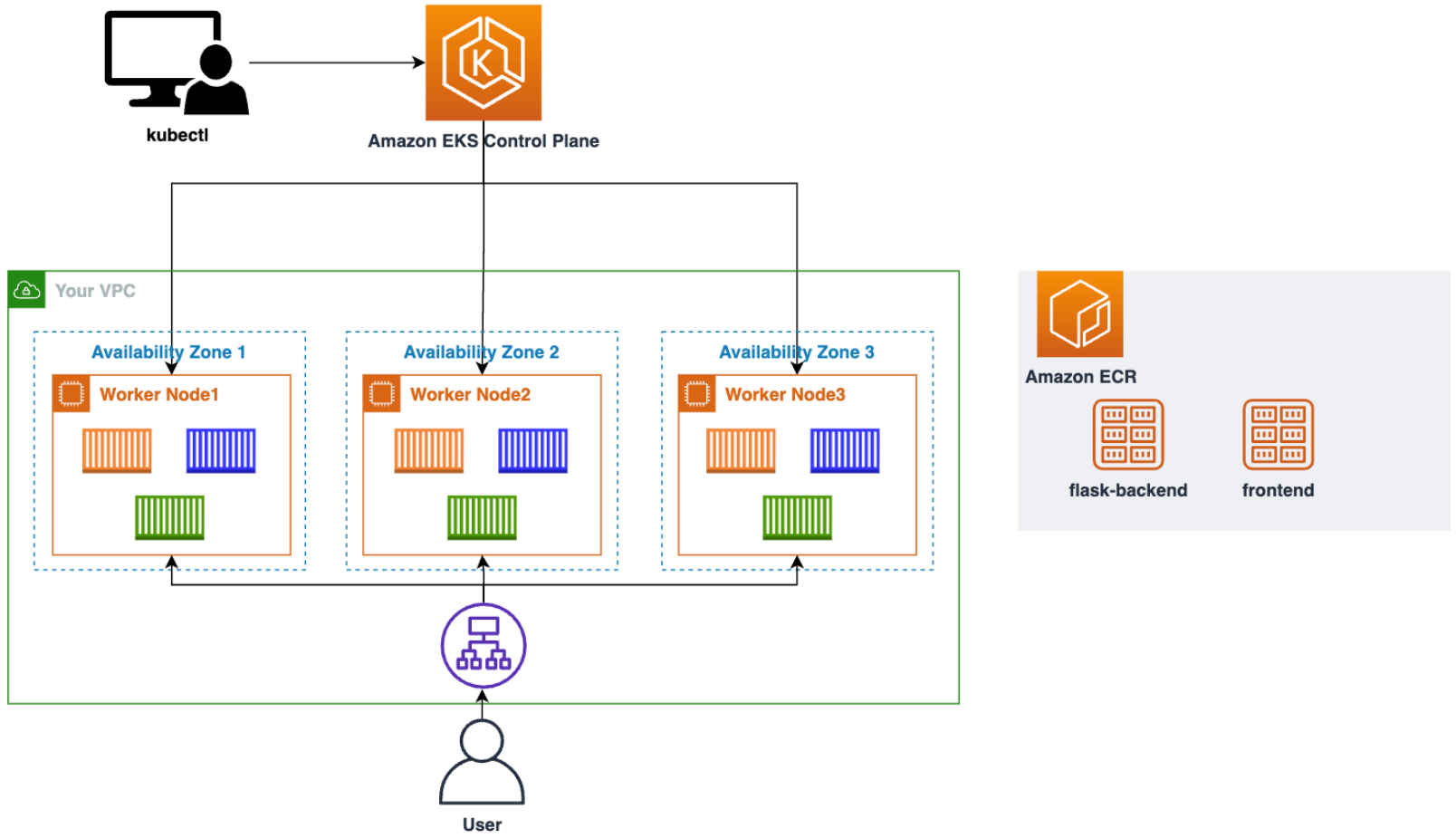
```
cat <<EOF> frontend-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: demo-frontend
  annotations:
    alb.ingress.kubernetes.io/healthcheck-path: "/"
spec:
  selector:
    app: demo-frontend
  type: NodePort
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF
```

```
cat <<EOF> frontend-ingress.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: "frontend-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/group.name: eks-demo-group
    alb.ingress.kubernetes.io/group.order: '3'
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: "demo-frontend"
                port:
                  number: 80
EOF
```

```
kubectl apply -f frontend-deployment.yaml
kubectl apply -f frontend-service.yaml
kubectl apply -f frontend-ingress.yaml
```

```
echo http://$(kubectl get ingress/frontend-ingress -o jsonpath='{.status.loadBalancer.ingress[*].hostname}')
```

서비스 배포하기


kubectl
Amazon EKS Control Plane
Your VPC
Availability Zone 1
Availability Zone 2
Availability Zone 3
Worker Node1
Worker Node2
Worker Node3
User
Amazon ECR
flask-backend
frontend

AWS Fargate 사용하기

AWS Fargate는 컨테이너를 실행시켜주는 서버리스 컴퓨팅 엔진으로 eks나 ecs에서 동작.
서버를 따로 프로비저닝하고 관리할 필교가 없어서
애플리케이션별로 리소스를 지정하고 비용한 지불하면 됨.

```
cat <<EOF> eks-demo-fargate-profile.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: eks-demo
  region: ${AWS_REGION}
fargateProfiles:
  - name: frontend-fargate-profile
    selectors:
      - namespace: default
        labels:
          app: frontend-fargate
EOF
```

Fargate로 pod을 배포하기 위해서는 pod을 생성하기 위한 조건인 fargate profile을 정의해야 함.

```
eksctl create fargateprofile -f eks-demo-fargate-profile.yaml
```

```
eksctl get fargateprofile --cluster eks-demo -o json
```


```
{
    "name": "frontend-fargate-profile",
    "podExecutionRoleARN": "arn:aws:iam::account-id:role/eksctl-eks-demo-test-
    "selectors": [
        {
            "namespace": "default",
            "labels": {
                "app": "frontend-fargate"
            }
        }
    ],
    "subnets": [
        "subnet-07e2d55650225419c",
        "subnet-0ac4a7fdbd803039c",
        "subnet-046a3dcfabce11b5f"
    ],
    "status": "ACTIVE"
}
```

AWS Fargate 사용하기

```
kubectl delete -f frontend-deployment.yaml
```

Front pod들을 삭제하고 fargate를 이용해서 pod을 재배포

```
cd /home/ec2-user/environment/manifests

cat <<EOF> frontend-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-frontend
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend-fargate
  template:
    metadata:
      labels:
        app: frontend-fargate
    spec:
      containers:
        - name: demo-frontend
          image: $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-frontend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
EOF
```

```
cat <<EOF> frontend-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: demo-frontend
  annotations:
    alb.ingress.kubernetes.io/healthcheck-path: "/"
spec:
  selector:
    app: frontend-fargate
  type: NodePort
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF
```

```
kubectl apply -f frontend-deployment.yaml
kubectl apply -f frontend-service.yaml
```

```
kubectl get nodes -l eks.amazonaws.com/compute-type=fargate
```

Fargate에서 사용하는 노드 – microVM Firecracker

Firecracker - 서버리스 컴퓨팅을 위한 오픈 소스 경량 가상화 기술 공개

by AWS Korea | on 03 12월 2018 | in Amazon EC2, AWS Re:Invent, Launch, News, Open Source | Permalink | Share

제가 가장 좋아하는 아마존 리더십 원칙 중 하나는 바로, 고객 집중(Customer Obsession) 원칙입니다. 저희는 2014년 AWS Lambda를 처음 공개하면서 인프라 관리가 필요하지 않도록 안전한 서버리스 환경을 개발자에게 제공하는 데 초점을 맞췄습니다. 원하는 수준의 애플리케이션 격리를 구현하기 위해 고객마다 전용 EC2 인스턴스를 사용했습니다.

이러한 접근 방식으로 보안 목표는 이루었지만, 한편으로는 Lambda를 관리하는 방식과 관련해 몇 가지 보완할 문제가 생겼습니다. 또한 당시에 다른 새로운 AWS 서비스들과 마찬가지로, 서버리스 모델에 대한 큰 그림이나 이에 대한 고객의 생각 및 고객이 실제로 Lambda를 어떻게 사용할 것인지를 파악하지 못했습니다. 뛰어난 고객 경험을 제공하는 동시에 시간이 지남에 따라 백엔드를 더욱 효율적으로 만드는 데 집중하는 과제를 안게 되었습니다

그리고 불과 4년만에 서버리스 모델은 우리 곁에 완전히 자리를 잡았습니다. 오늘날, Lambda는 매월 수십만 명의 활성 고객을 위해 수조 건의 실행 작업을 처리하고 있습니다. 작년에는 AWS Fargate를 출시하며 서버리스 환경의 장점을 컨테이너로 확장하면서, 이제 AWS 고객은 매주 수천만 개의 컨테이너를 실행할 수 있게 되었습니다.

서버리스 환경을 채택하는 고객이 늘어나면서 이제 효율성의 문제를 재고할 때가 왔습니다. 그리고 AWS의 발명과 단순화 원칙을 마음에 새기며 오늘날 컨테이너 및 기능에 맞게 설계된 가상 머신은 어떤 모습이어야 하는지를 자문해보았습니다!

Firecracker 소개

오늘 저는 KVM을 사용하는 새로운 가상화 기술인 Firecracker를 여러분에게 소개하고자 합니다. Firecracker를 통해 여러분은 가상화되지 않은 환경에서 1초도 되지 않는 시간 안에 경량 microVM(마이크로 가상 머신)을 시작할 수 있고, 컨테이너를 통해 제공하는 리소스 효율성과 기존 VM에서 제공하는 워크로드 격리 및 보안의 혜택을 그대로 활용할 수 있습니다.

다음은 Firecracker에 대해 알아야 할 핵심 사항입니다.

- **보안** - 항상 가장 중요한 우선순위입니다! Firecracker는 여러 수준의 격리와 보호를 사용하며, 공격 노출 영역을 최소화합니다.
- **고성능** - 현 시점 기준으로 125밀리초 안에 microVM을 시작할 수 있으며(2019년에는 더 빨라질 예정), 단기간 또는 일시적인 워크로드를 비롯한 여러 유형의 워크로드에 적합합니다.
- **검증된 실적** - Firecracker는 실전에서 검증되어 있습니다. 이미 AWS Lambda 및 AWS Fargate를 비롯한 사용량이 많은 여러 AWS 서비스가 Firecracker를 사용하고 있습니다.
- **낮은 오버헤드** - Firecracker는 microVM당 약 5MiB의 메모리를 사용합니다. 그리고 동일한 인스턴스에서 다양한 vCPU 및 메모리 구성 사양을 갖춘 안전한 수천 개의 VM을 실행할 수 있습니다.
- **오픈 소스** - Firecracker는 현재 진행 중인 오픈 소스 프로젝트입니다. 이미 검토와 PR(Pull Request)을 수락할 준비가 되었으며, 전 세계 기고자들과 협업할 수 있기를 기대하고 있습니다.

Firecracker는 미니멀리즘에 기반하여 제작되었습니다. crosvm에서 시작하여 오버헤드를 줄이고 안전한 멀티 테넌시를 활용하도록 최소 디바이스 모델을 설정했습니다. Firecracker는 스레드 보안을 보장하고 보안 취약성을 야기할 수 있는 여러 유형의 버퍼 오버런 오류를 방지하는 최신 프로그래밍 언어인 Rust로 작성되었습니다.

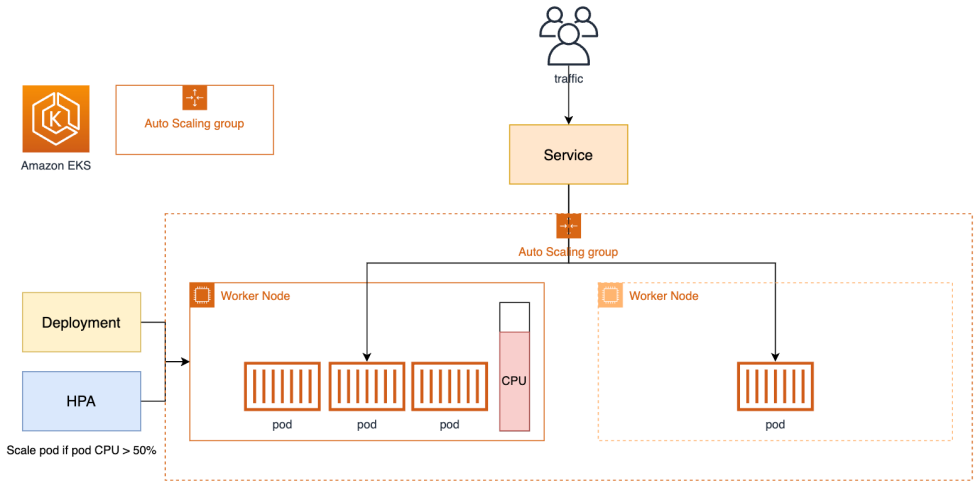
오토 스케일링

쿠버네티스 오토 스케일링을 사용해서 애플리케이션을 트래픽에 따라 탄력적으로 조절할 수 있음

- HPA(Horizontal Pod AutoScaler)
- Cluster Autoscaler

HPA는 CPU 사용량 또는 사용자 정의 메트릭을 관찰하여 파드 개수 조절

Cluster autoscaler는 워커 노드가 가득 차서 파드가 스케줄될 수 없는 경우 노드를 늘리는 방법




```
kubectl autoscale deployment demo-flask-backend --cpu-percent=30 --min=1 --max=5
```

```
cd /home/ec2-user/environment/manifests

cat <<EOF> flask-deployment.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-flask-backend
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo-flask-backend
  template:
    metadata:
      labels:
        app: demo-flask-backend
    spec:
      containers:
        - name: demo-flask-backend
          image: $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/demo-flask-backend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 8080
          resources:
            requests:
              cpu: 250m
            limits:
              cpu: 500m
EOF
```

or

```
cat <<EOF> flask-hpa.yaml
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: demo-flask-backend-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: demo-flask-backend
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 30
EOF
```

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

k8s metric server


thanks!