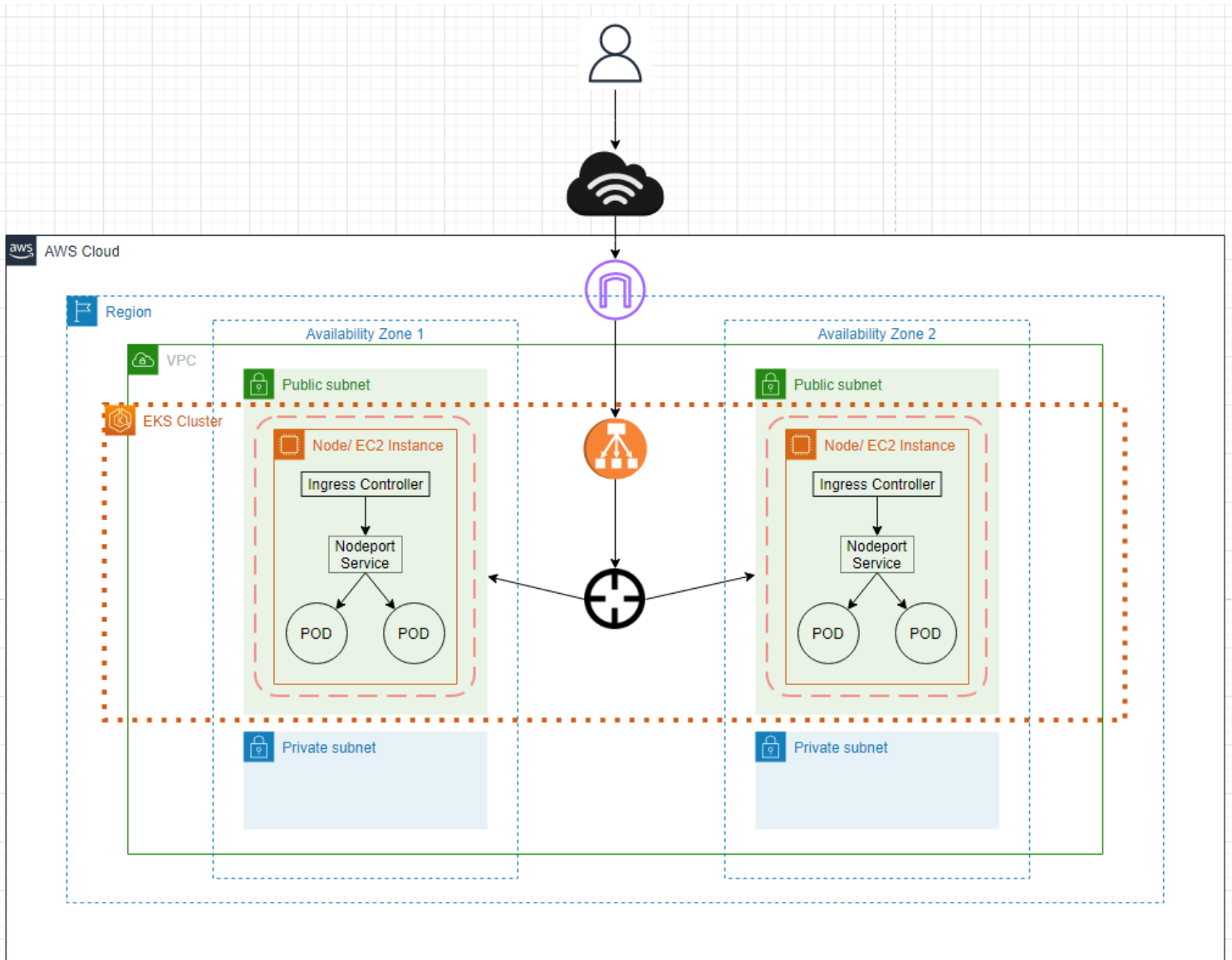




## Amazon EKS



For this assignment, we had to use Kubernetes and Amazon Elastic Kubernetes Service (EKS) to deploy an NGINX application on the Cloud. EKS is a cloud-based container management service that natively integrates with Kubernetes to deploy our application. The NGINX application that is being deployed has a simple web page that was created using a Docker Image.

To start, we will have to create an EKS Cluster on our local machine using AWS Command Line Interface (CLI). This will create a CloudFormation stack on AWS that will create all resources needed such as VPCs, EC2s, security groups, load balancers, etc. Once the cluster is created, we configured the application by creating a YAML file. YAML files are essentially configuration files that we can edit to access certain applications and link multiple applications. YAML files are simply infrastructure as code for Kubernetes. In this assignment, we created the NGINX deployment and a NodePort service in one YAML file. In Kubernetes, deployments give instructions on how the pod is configured. Services, on the other hand, give access to pods. Pods are a bundle of containers needed to run an application.

Once we have created the deployment and NodePort service in one YAML file, we need to then create the ingress controller, which is simply a load balancer for Kubernetes. Ingress controllers can also be seen as a reverse proxy that forwards traffic to nodes. The ingress controller will talk to the NodePort service on a specific path and port. The NodePort service will then send traffic to the Pods that have our NGINX application.

When the application is configured, we need to set up OpenID. The EKS Cluster we created is in charge of creating resources on AWS such as elastic load balancers. We need to allow the cluster to authenticate itself. Setting up OpenID will give the cluster permissions.

Along with OpenID, we need to create a role and bind that role to the ingress controller. We will create a YAML file with the specific roles. The Role Based Action Control will allow our EKS Cluster to create resources such as the load balancer. The YAML file will have rules of what the cluster is allowed to do such as create, get, list, etc.

We then need to download and create an IAM policy that will create a service link role on AWS. This Role will allow different access to certain services on AWS and certain permission that is binded to the ingress controller on the cluster. In other words, this will allow our ingress control to create everything it needs to on AWS. When the policy is created, you will need to attach the policy to the EKS Cluster.

You will then need to create a service account for the cluster. Our cluster will inherit the policy. Overall at this stage, we are binding the policy that we created, which gives access to create resources on AWS, to the ingress controller and cluster on AWS.

We then need to create a certificate manager for the ingress controller. This will create the certificates which are needed for the communication between the elastic load balancer and the Ingress controller. By doing so, our application is secure by having an encrypted connection.

Finally, we need to make the load balancer controller by downloading a YAML file and editing it to match our cluster name. Once edited, create that YAML file. We can then deploy our application by creating the two YAML files we created earlier. Once it's deployed we can access our NGINX webpage. The DNS name will be under the AWS load balancer service.

# Task 1

---

## Create a cluster using...

eksctl create cluster --name **mycluster03**

## Create an Nginx deployment yaml file with...

nano nginx.yaml

## Paste the following inside of it...

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.19.6
          ports:
            - name: http
              containerPort: 80
```

---

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-nodeport
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
```

targetPort: 80

**Create an ingress controller yaml file with...**

nano nginx-ingress.yaml

**Paste the following inside of it...**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: simple-ingress

annotations:

kubernetes.io/ingress.class: alb

alb.ingress.kubernetes.io/scheme: internet-facing

alb.ingress.kubernetes.io/target-type: instance

spec:

rules:

- http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: nginx-service-nodeport

port:

number: 80

## Task 2

---

### Describe the cluster

aws eks describe-cluster --name **mycluster03** --query "cluster.identity.oidc.issuer" --output text

```
ricardo@ricardo-VirtualBox:~/Documents/EKS$ aws eks describe-cluster --name mycluster03 --query "cluster.identity.oidc.issuer" --output text
https://oidc.eks.us-east-1.amazonaws.com/id/807B90B160B63010E889EAE3929DAC18
```

### Check the open-id provider list

aws iam list-open-id-connect-providers

```
ricardo@ricardo-VirtualBox:~/Documents/EKS$ aws iam list-open-id-connect-providers
{
  "OpenIDConnectProviderList": []
}
```

### Sign up for an openID provider which is IAM

eksctl utils associate-iam-oidc-provider --cluster mycluster03 --approve

```
ricardo@ricardo-VirtualBox:~/Documents/EKS$ eksctl utils associate-iam-oidc-provider --cluster mycluster03 --approve
2021-10-23 11:55:14 [i] eksctl version 0.70.0
2021-10-23 11:55:14 [i] using region us-east-1
2021-10-23 11:55:14 [i] will create IAM Open ID Connect provider for cluster "mycluster03" in "us-east-1"
2021-10-23 11:55:14 [✓] created IAM Open ID Connect provider for cluster "mycluster03" in "us-east-1"
```

### Check the open-id provider list again

aws iam list-open-id-connect-providers

```
ricardo@ricardo-VirtualBox:~/Documents/EKS$ aws iam list-open-id-connect-providers
{
  "OpenIDConnectProviderList": [
    {
      "Arn": "arn:aws:iam::[REDACTED]:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/807B90B160B63010E889EAE3929DAC18"
    }
  ]
}
```

## Task 3

---

**We are going to download a file and save it to rbac-role.yaml (role based action control)**

```
curl -o rbac-role.yaml
```

```
https://raw.githubusercontent.com/RobinNagpal/kubernetes-tutorials/master/06_tools/007_alb_ingress/01_eks/rbac-role.yaml
```

```
ricardo@ricardo-VirtualBox:~/Documents/EKS$ curl -o rbac-role.yaml https://raw.githubusercontent.com/RobinNagpal/kubernetes-tutorials/master/06_tools/007_alb_ingress/01_eks/rbac-role.yaml
% Total    % Received % Xferd Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 1163  100 1163    0     0  7006      0 --:--:-- --:--:-- --:--:--  7006
```

**Create the rbac-role.yaml file. This will create a role and bind it to the ingress controller.**

```
kubectl apply -f rbac-role.yaml
```

**Shows different roles that was created for your cluster.**

```
kubectl get serviceaccount
```

**Download the IAM policy. This will create the role on AWS. This allows the ingress controller to create everything its needed on aws.**

```
curl -o iam_policy.json
```

```
https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.3.0/docs/install/iam_policy.json
```

**Create the AWS policy by running the following**

**(Make sure you're in the same directory as the IAM policy that was downloaded)...**

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

**Then run the following. Change the ARN number and clustername. ...**

```
eksctl create iamserviceaccount --cluster=mycluster03 --namespace=kube-system
--name=aws-load-balancer-controller
--attach-policy-arn=arn:aws:iam::252544977596:policy/AWSLoadBalancerControllerIAMPolicy
--override-existing-serviceaccounts --approve
```

**Create the certificate manager for the ingress controller using...**

```
kubectl apply --validate=false -f
```

```
https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml
```

## Task 4

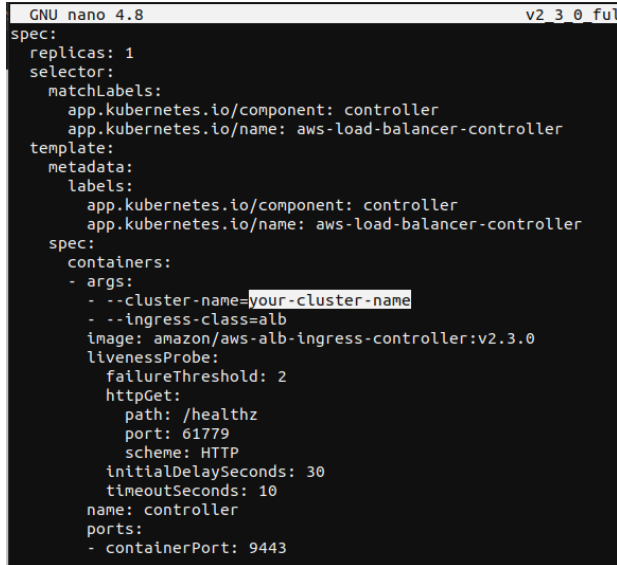
---

**Make the load balancer controller by downloading and running the following**

```
curl -Lo v2_3_0_full.yaml
```

```
https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.3.0/v2_3_0_full.yaml
```

**Edit the file and replace your-cluster-name with your cluster name**



```
GNU nano 4.8 v2_3_0_full
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/component: controller
      app.kubernetes.io/name: aws-load-balancer-controller
  template:
    metadata:
      labels:
        app.kubernetes.io/component: controller
        app.kubernetes.io/name: aws-load-balancer-controller
    spec:
      containers:
      - args:
        - --cluster-name=your-cluster-name
        - --ingress-class=alb
        image: amazon/aws-alb-ingress-controller:v2.3.0
        livenessProbe:
          failureThreshold: 2
          httpGet:
            path: /healthz
            port: 61779
            scheme: HTTP
          initialDelaySeconds: 30
          timeoutSeconds: 10
        name: controller
        ports:
        - containerPort: 9443
```

**Once changes, save the file and run the following command**

```
kubectl apply -f v2_3_0_full.yaml
```

**View the controller using...**

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

**Create the yaml file using...**

```
kubectl apply -f nginx.yaml
```

**Look at what was created**

```
kubectl get all
```

```

ricardo@ricardo-VirtualBox:~/Documents/EKS$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/sample-app-5f7fdb8854-rt42q     1/1     Running   0           3m26s
pod/sample-app-5f7fdb8854-zj9np     1/1     Running   0           3m26s

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
service/kubernetes                  ClusterIP     10.100.0.1   <none>        443/TCP        3h32m
service/nginx-service-nodeport      NodePort      10.100.110.27 <none>        80:31277/TCP   3m26s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/sample-app          2/2     2             2           3m28s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/sample-app-5f7fdb8854 2         2         2       3m28s

```

**Create the ingress controller yaml file using....**

`kubectl apply -f nginx-ingress.yaml`

**Look at what was created**

`kubectl describe ingress.networking.k8s.io -o wide`

**Go to AWS (make sure region is correct) -> EC2 -> Load balancer**

**Copy the DNS name and enter it into your browser URL**

The screenshot shows the AWS Management Console interface for a Load Balancer. The top navigation bar includes a search bar and a list of resources. The main content area shows the details of the Load Balancer 'k8s-default-simiplein-e6802a9da4'. The 'DNS name' field is highlighted with a red circle, showing the value 'k8s-default-simiplein-e6802a9da4-729160938.us-east-1.elb.amazonaws.com'.

**Result should be...**

The screenshot shows a web browser address bar with the URL 'k8s-default-simiplein-e6802a9da4-729160938.us-east-1.elb.amazonaws.com'. The address bar is dark, and the URL is displayed in white text.

## Welcome to nginx!

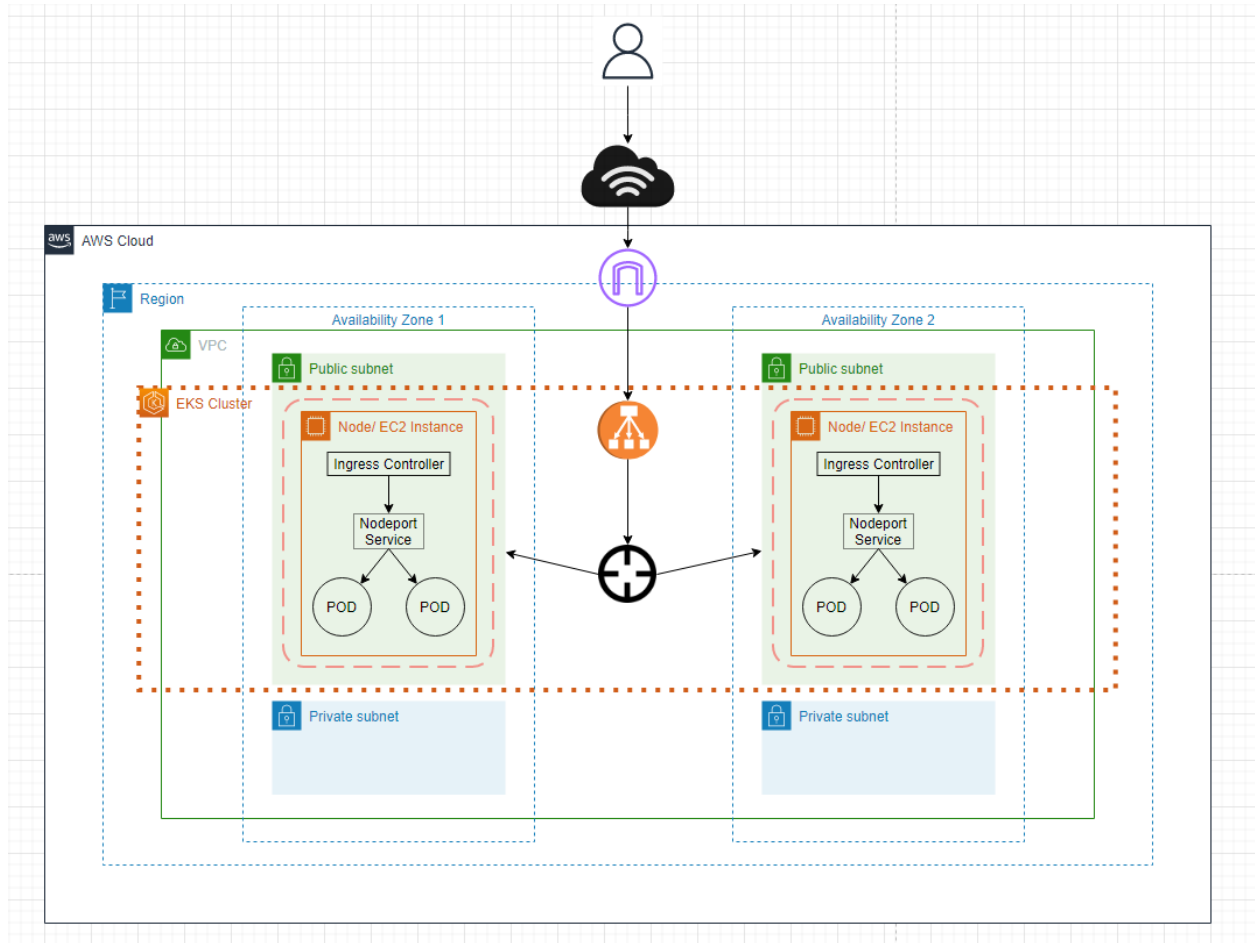
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](https://nginx.org).  
Commercial support is available at [nginx.com](https://nginx.com).

*Thank you for using nginx.*



# Topology Explanation



The flow of traffic for this assignment is simple. Users want to access our NGINX application. Users will then access the exposed elastic load balancer which will look at the target group to see where to send the traffic and at what port/path. Traffic is then sent to the EKS Cluster which has our ingress controller. The ingress controller then forwards traffic to the NodePort service which talks to our Pods. The Pods has packaged containers of our applications.