# Setting Up an EKS Cluster and SSH Gateway on Kubernetes: Step-by-Step Guide

## Prerequisites

Before starting, ensure the following tools are installed and configured:

- AWS CLI

- kubectl

- eksctl

- Docker

Ensure your AWS account has the required IAM permissions for creating an EKS cluster, accessing ECR, and managing resources.

## Step 1: Create an EKS Cluster

Purpose: To set up a Kubernetes cluster on AWS to manage and deploy your SSH gateway.

Implementation:

1. Create the cluster using `eksctl`:

eksctl create cluster --name ssh-customer-cluster --region us-east-1 --nodes 3

2. Verify the cluster status:

aws eks describe-cluster --name ssh-customer-cluster --query "cluster.status"

3. Configure `kubectl` to connect to the cluster:

aws eks update-kubeconfig --region us-east-1 --name ssh-customer-cluster

4. Verify connectivity:

kubectl get nodes

## Step 2: Create a Storage Class

Purpose: To define how Kubernetes dynamically provisions storage for customer data.

File: `storageclass.yaml`

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

name: standard

provisioner: ebs.csi.aws.com

parameters:

type: gp2

reclaimPolicy: Delete

volumeBindingMode: Immediate

Implementation:

1. Apply the `StorageClass`:

kubectl apply -f storageclass.yaml

2. Verify it:

kubectl get storageclass

## Step 3: Create a Persistent Volume Claim (PVC)

Purpose: To allocate persistent storage for customer-specific data.

File: `pvc.yaml`

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: customer-pvc-1

namespace: customer-accounts

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 5Gi

storageClassName: standard

Implementation:

1. Create a namespace:

kubectl create namespace customer-accounts

2. Apply the PVC:

kubectl apply -f pvc.yaml

3. Verify the PVC:

kubectl get pvc -n customer-accounts

## Step 4: Build and Push the Docker Image

Purpose: To create a custom container image for the SSH gateway.

File: `Dockerfile`

FROM ubuntu:20.04

RUN apt-get update && apt-get install -y openssh-server mysql-client rsync

RUN mkdir /var/run/sshd

EXPOSE 22

CMD ["/usr/sbin/sshd", "-D"]

Implementation:

1. Build the Docker image:

docker build -t ssh-gateway:latest .

2. Tag the image for AWS ECR:

docker tag ssh-gateway:latest <your-account-id>.dkr.ecr.us-east-1.amazonaws.com/ssh-gateway:latest

3. Push the image to ECR:

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <your-account-id>.dkr.ecr.us-east-1.amazonaws.com

docker push <your-account-id>.dkr.ecr.us-east-1.amazonaws.com/ssh-gateway:latest

## Step 5: Deploy the SSH Gateway

Purpose: To run the SSH gateway pods on the Kubernetes cluster.

File: `ssh-gateway-customer1.yaml`

apiVersion: apps/v1

kind: Deployment

metadata:

name: ssh-gateway-customer1

namespace: customer-accounts

spec:

replicas: 1

selector:

matchLabels:

app: ssh-gateway

template:

metadata:

labels:

app: ssh-gateway

spec:

containers:

- name: ssh-gateway

image: <your-account-id>.dkr.ecr.us-east-1.amazonaws.com/ssh-gateway:latest

ports:

- containerPort: 22

volumeMounts:

- mountPath: "/customer-data"

name: customer-data

volumes:

- name: customer-data

persistentVolumeClaim:

claimName: customer-pvc-1

Implementation:

1. Apply the deployment:

kubectl apply -f ssh-gateway-customer1.yaml

2. Verify the pods:

kubectl get pods -n customer-accounts

## Step 6: Set Up Network Policy

Purpose: To restrict access to the SSH gateway pods.

File: `customer-network-policy-1.yaml`

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: customer-network-policy-1

namespace: customer-accounts

spec:

podSelector:

matchLabels:

app: ssh-gateway

ingress:

- from:

- ipBlock:

cidr: 192.168.79.0/32

ports:

- protocol: TCP

port: 22

Implementation:

1. Apply the network policy:

kubectl apply -f customer-network-policy-1.yaml

2. Verify the policy:

kubectl get networkpolicy -n customer-accounts

## Step 7: Add the SSH Key

Purpose: To configure key-based authentication for secure SSH access.

Implementation:

1. Access the pod:

kubectl exec -it <pod-name> -n customer-accounts -- /bin/bash

2. Configure the SSH key:

mkdir -p /root/.ssh

chmod 700 /root/.ssh

echo "<contents-of-customer\_1\_id\_rsa.pub>" >> /root/.ssh/authorized\_keys

chmod 600 /root/.ssh/authorized\_keys

3. Restart the SSH service:

service ssh restart

## Step 8: Test the Setup

Purpose: To verify the SSH gateway is functioning correctly.

Implementation:

1. Port forward the pod:

kubectl port-forward pod/<pod-name> 2222:22 -n customer-accounts

2. SSH into the pod:

ssh -i ~/.ssh/customer\_1\_id\_rsa root@localhost -p 2222

## Step 9: Monitor and Debug

Purpose: To ensure the system is running smoothly and resolve any issues.

Implementation:

1. Monitor pod logs:

kubectl logs -f <pod-name> -n customer-accounts

2. Check PVC usage:

kubectl get pvc -n customer-accounts

Add Permissions in the role created by EKS:

AmazonEBSCSIDriverPolicy