LAB: Deploy NGINX App on an Amazon EKS Cluster

Hands-on walkthrough with commands, explanations, and junior DevOps tips

Report date: 2025-08-28 13:35:15Z (UTC)

# Overview

In this lab, we deployed a basic NGINX web server on an existing Amazon EKS cluster. We created a Deployment, exposed it via a LoadBalancer Service, verified the public endpoint, and scaled the Deployment. This report captures the exact commands, explanations, and notes.

# Prerequisites

- An AWS account and CLI credentials configured on your workstation.

- kubectl and eksctl installed.

- EKS cluster reachable (we created one during this lab).

- kubectl context pointed at the target EKS cluster.

# Step 0 — Tooling & Credentials (Summary)

We verified tooling (awscli, kubectl, eksctl) and credentials:

aws --version  
aws sts get-caller-identity  
kubectl version --client  
eksctl version

# Step 1 — Create the EKS Cluster

We created an EKS cluster in region us-east-2 with a managed node group and OIDC enabled. We also used a single NAT gateway to limit Elastic IP usage.

export AWS\_REGION=us-east-2  
export CLUSTER\_NAME=nginx-lab-us-east-2  
export K8S\_VERSION=1.29  
  
eksctl create cluster \  
 --name "${CLUSTER\_NAME}" \  
 --region "${AWS\_REGION}" \  
 --version "${K8S\_VERSION}" \  
 --managed \  
 --nodegroup-name "ng-1" \  
 --node-type "t3.medium" \  
 --nodes 2 \  
 --nodes-min 2 \  
 --nodes-max 3 \  
 --with-oidc \  
 --vpc-nat-mode Single  
  
kubectl get nodes -o wide

# Step 2 — Create NGINX Deployment

We created a Deployment named nginx-deployment using the official image.

kubectl create deployment nginx-deployment --image=nginx:latest  
kubectl get deployments  
kubectl get pods

Explanation: A Deployment manages a ReplicaSet which ensures the desired number of Pods are running. By default it creates 1 Pod.

# Step 3 — Expose via LoadBalancer Service

We exposed the Deployment with a Service of type LoadBalancer to obtain a public endpoint.

kubectl expose deployment nginx-deployment \  
 --type=LoadBalancer \  
 --port=80 \  
 --name=nginx-service  
  
kubectl get svc nginx-service -w

Once EXTERNAL-IP was assigned, we verified the endpoint with curl.

curl -I http://<EXTERNAL-IP>  
curl http://<EXTERNAL-IP>

For this run, the ELB DNS was: http://ab240c6d8a5b146c08b20a24b00be530-1646847293.us-east-2.elb.amazonaws.com

# Step 4 — Scale the Deployment

We scaled the Deployment to 3 replicas and watched the rollout complete.

kubectl scale deployment nginx-deployment --replicas=3  
kubectl rollout status deployment/nginx-deployment  
kubectl get pods -o wide

# Optional — Cleanup

To remove resources when done:

kubectl delete service nginx-service  
kubectl delete deployment nginx-deployment

# Appendix — YAML Manifests

Declarative manifests equivalent to what we created with imperative commands:

## k8s/deployment.yaml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: nginx-deployment  
 labels:  
 app: nginx  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: nginx  
 template:  
 metadata:  
 labels:  
 app: nginx  
 spec:  
 containers:  
 - name: nginx  
 image: nginx:latest  
 ports:  
 - containerPort: 80  
 readinessProbe:  
 httpGet:  
 path: /  
 port: 80  
 initialDelaySeconds: 5  
 periodSeconds: 10  
 livenessProbe:  
 httpGet:  
 path: /  
 port: 80  
 initialDelaySeconds: 10  
 periodSeconds: 20

## k8s/service.yaml

apiVersion: v1  
kind: Service  
metadata:  
 name: nginx-service  
 labels:  
 app: nginx  
spec:  
 type: LoadBalancer  
 ports:  
 - port: 80  
 targetPort: 80  
 protocol: TCP  
 name: http  
 selector:  
 app: nginx

# Tips for Junior DevOps Engineers

• Prefer declarative YAML (`kubectl apply -f`) for repeatability; use `kubectl create/expose --dry-run=client -o yaml` to generate manifests quickly.

• Add readiness/liveness probes to catch bad containers and ensure traffic only hits healthy Pods.

• Use managed node groups for simplified operations; enable OIDC to leverage IAM Roles for Service Accounts (IRSA).

• Service type LoadBalancer provisions an external LB per Service; consider an Ingress Controller to consolidate LBs.

• Use `kubectl rollout status` and `kubectl rollout history` to track Deployment changes; set resource requests/limits.

• Tag your public/private subnets correctly in EKS clusters to ensure LoadBalancers provision successfully.