Running Kubernetes on AWS (EKS)

Prajit Venkatachalam
https://www.linkedin.com/in/prajit-venkatachalam/

Prerequisites

- Access to a terminal application for running simple Unix commands in Windows, Linux, or macOS
- Access to a text editor like VS Code
- An active AWS account
- Working knowledge of Kubernetes

Three CLIs to Install

Installing AWS CLI

Command to install AWS CLI**: choco install awscli**

once installed, verify the installation by typing **aws**, if it doesn't work or shown any error, restart the PowerShell and retype the command **aws**

Installing Kubernetes CLI

What is Kube control

Kubectl is a Kubernetes command-line tool that enables you to manage and interact with Kubernetes clusters, whether they are set up on AWS, Azure, GCP, or even on local hardware like Raspberry Pi devices.

Command to install Kubernetes CLI: **choco install Kubernetes-cli**

```
PS C:\WINDOWS\system32> choco install kubernetes-cli
Chocolatey v2.3.0
Installing the following packages:
kubernetes-cli
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading kubernetes-cli 1.31.2... 100%

kubernetes-cli v1.31.2 [Approved]
kubernetes-cli package files install completed. Performing other installation steps.
The package kubernetes-cli wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): Y

Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client-windows-amd64.tar.gz to C:\ProgramData\chocolatey\lib\kubernetes-cli\tools...

C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
Extracting 64-bit C:\ProgramData\chocolatey\lib\kubernetes-cli\tools\kubernetes-client-windows-amd64.tar to C:\ProgramData\chocolatey\lib\kubernetes-cli\tools...
C:\ProgramData\chocolatey\lib\kubernetes-cli\tools
 ShimGen has successfully created a shim for kubectl-convert.exe
 ShimGen has successfully created a shim for kubectl.exe
 The install of kubernetes-cli was successful.
  Deployed to 'C:\ProgramData\chocolatey\lib\kubernetes-cli\tools'

Chocolatey installed 1/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\WINDOWS\system32>
```

Then verifying the installation by running the command: **kubectl**

Installing eksctl CLI

What is eksctl

The `eksctl` CLI is a command-line interface that enables the creation and management of Kubernetes clusters specifically with Amazon Elastic Kubernetes Service (EKS).

Command to install **choco install eksctl**

```
PS C:\WINDOWS\system32> choco install eksctl
Chocolatey v2.3.0
Installing the following packages:
eksctl
By installing, you accept licenses for the packages.
Downloading package from source 'https://community.chocolatey.org/api/v2/'
Progress: Downloading eksctl 0.194.0... 100%

eksctl v0.194.0 [Approved]
eksctl package files install completed. Performing other installation steps.
The package eksctl wants to run 'chocolateyInstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N]o/[P]rint): Y

eksctl is going to be installed in 'C:\ProgramData\chocolatey\lib\eksctl\tools'
Downloading eksctl 64 bit
  from 'https://github.com/eksctl-io/eksctl/releases/download/v0.194.0/eksctl_Windows_amd64.zip'
Progress: 100% - Completed download of C:\Users\vpraj\AppData\Local\Temp\chocolatey\eksctl\0.194.0\eksctl_Windows_amd64.zip (35.02 MB).
Download of eksctl_Windows_amd64.zip (35.02 MB) completed.
Hashes match.
Extracting C:\Users\vpraj\AppData\Local\Temp\chocolatey\eksctl\0.194.0\eksctl_Windows_amd64.zip to C:\ProgramData\chocolatey\lib\eksctl\tools...
C:\ProgramData\chocolatey\lib\eksctl\tools
 ShimGen has successfully created a shim for eksctl.exe
 The install of eksctl was successful.
  Deployed to 'C:\ProgramData\chocolatey\lib\eksctl\tools'

Chocolatey installed 1/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\WINDOWS\system32>
```

To verify, I ran the command: **eksctl**

Creating an EKS admin user group and user

I created a user group named `eks-admin` and assigned it the Administrator Access policy from the permissions tab, granting it full access to AWS resources. Next, I created a user and added them to the `eks-admin` group.

To configure AWS credentials in PowerShell, I used the command `**aws configure**`. I then entered the necessary details: AWS Access Key ID, AWS Secret Access Key, default region name, and output format.

Creating an EKS cluster with eksctl

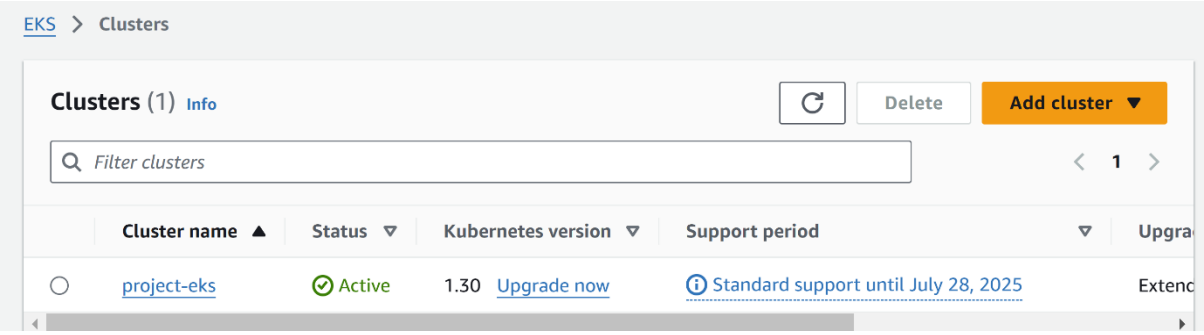
I started by creating a cluster.yaml file and populated it with essential configuration details, including API version, kind, metadata, and node groups. Then, I used the following command to create the EKS cluster:

eksctl create cluster -f "C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_02\02_04_Begin\cluster.yaml"

```
2024-11-13 19:13:46 [✔]  all EKS cluster resources for "project-eks" have been created
2024-11-13 19:13:46 [ℹ]  nodegroup "worker-node" has 2 node(s)
2024-11-13 19:13:46 [ℹ]  node "ip-192-168-18-202.ap-southeast-2.compute.internal" is ready
2024-11-13 19:13:46 [ℹ]  node "ip-192-168-56-38.ap-southeast-2.compute.internal" is ready
2024-11-13 19:13:46 [ℹ]  waiting for at least 2 node(s) to become ready in "worker-node"
2024-11-13 19:13:46 [ℹ]  nodegroup "worker-node" has 2 node(s)
2024-11-13 19:13:46 [ℹ]  node "ip-192-168-18-202.ap-southeast-2.compute.internal" is ready
2024-11-13 19:13:46 [ℹ]  node "ip-192-168-56-38.ap-southeast-2.compute.internal" is ready
2024-11-13 19:13:46 [✔]  created 1 nodegroup(s) in cluster "project-eks"
2024-11-13 19:13:46 [✔]  created 0 managed nodegroup(s) in cluster "project-eks"
2024-11-13 19:13:48 [ℹ]  kubectl command should work with "C:\\Users\\vpraj\\.kube\\config", try 'kubectl get nodes'
2024-11-13 19:13:48 [✔]  EKS cluster "project-eks" in "ap-southeast-2" region is ready
PS C:\WINDOWS\system32>
```

To make sure I can access the cluster using the CLI, I ran the command **kubectl get nodes**, which shown me the set of worker nodes.

Now I checked the AWS console to the see the eks cluster created.



Exploring the existing resources in my EKS cluster

To view the list of nodes in the cluster, I ran the command:

Kubectl get nodes

```
PS C:\WINDOWS\system32> kubectl get nodes
NAME                                             STATUS   ROLES    AGE     VERSION
ip-192-168-18-202.ap-southeast-2.compute.internal   Ready    <none>   4m56s   v1.30.4-eks-a737599
ip-192-168-56-38.ap-southeast-2.compute.internal    Ready    <none>   5m15s   v1.30.4-eks-a737599
```

To check which cluster is currently configured on my computer, I used:

Kubectl config get-contexts

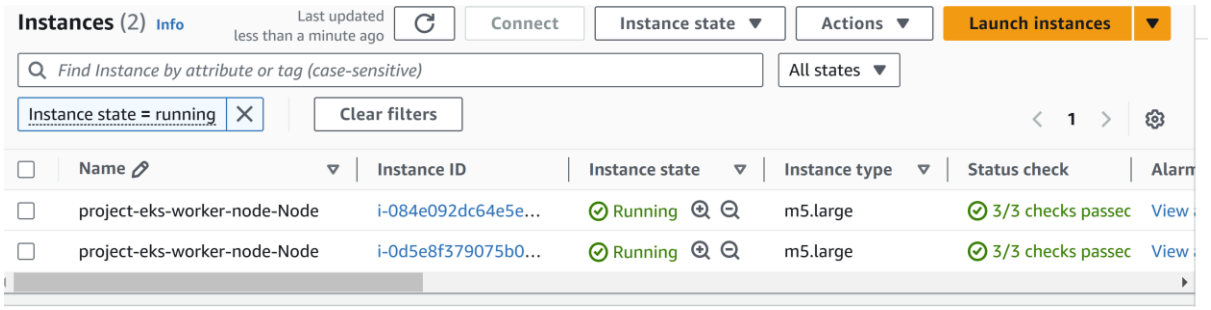
```
PS C:\WINDOWS\system32> kubectl config get-contexts
CURRENT   NAME                                          CLUSTER                          AUTHINFO                                      NAM
ESPACE
*         Prajit-IAM-Admin@project-eks.ap-southeast-2.eksctl.io   project-eks.ap-southeast-2.eksctl.io   Prajit-IAM-Admin@project-eks.ap-southeast-2.eksctl.io
PS C:\WINDOWS\system32>
```

To list all pods running in the cluster across all namespaces, I ran:

Kubectl get pods -A

```
PS C:\WINDOWS\system32> kubectl get pods -A
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE
kube-system   aws-node-hg8hk             2/2     Running   0          16m
kube-system   aws-node-pvrb9             2/2     Running   0          17m
kube-system   coredns-585d4c556b-4lvvp   1/1     Running   0          20m
kube-system   coredns-585d4c556b-fdcpl   1/1     Running   0          20m
kube-system   kube-proxy-7bs2f           1/1     Running   0          16m
kube-system   kube-proxy-v28t5           1/1     Running   0          17m
PS C:\WINDOWS\system32>
```

I verified the worker nodes created in the EKS cluster by checking the EC2 instances in the AWS Management Console.



Deploying an application to my EKS cluster

In this step, first I will create a Kubernetes namespace, deploy an application to the cluster, and create a service to prepare my application to be available to the internet.

Creating namespace

To create the namespace, I prepared a namespace.yaml file with the necessary elements, including kind, apiVersion, and metadata. I then executed the following command in PowerShell to apply the configuration:

kubectl apply -f "C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_01\namespace.yaml"

To verify that, I ran the command**: kubectl get ns**

```
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS> kubectl apply -f "C:
netes_on_AWS\Exercise Files\Chapter_03\03_01\namespace.yaml"
namespace/staging created
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS> kubectl get ns
NAME              STATUS   AGE
default           Active   4h11m
development       Active   3h
kube-node-lease   Active   4h11m
kube-public       Active   4h11m
kube-system       Active   4h11m
staging           Active   14s
```

Creating Deployment

To create the deployment, I wrote the configuration in a deployment.yaml file. I then applied it to the cluster with the following command:

kubectl apply -f "C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_01\deployment.yaml"

To verify all deployments within the staging namespace, I used:

kubectl deploy -n staging

```
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS> kubectl apply -f "C:\Users\vpraj\Downl
netes_on_AWS\Exercise Files\Chapter_03\03_01\deployment.yaml"
deployment.apps/pod-info created
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS> kubectl get deploy - staging
Error from server (NotFound): deployments.apps "-" not found
Error from server (NotFound): deployments.apps "staging" not found
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS> kubectl get deploy -n staging
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
pod-info   2/2     2            2           4m44s
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS>
```

Creating the service

What is Kubernetes NodePort service

The Kubernetes NodePort service acts as an external entry point for incoming requests to your application. I am using the NodePort service because it enables the use of an AWS load balancer to route traffic to the cluster. By exposing the service on a specific port across all nodes in the cluster, it provides a simple way to access the application externally. In this setup, AWS's load balancer can forward traffic to the NodePort, ensuring scalability and availability of the service across different worker nodes.

To create the service, I first developed the script for service.yaml with the necessary configurations. Then, I applied the configuration by running the following command:

kubectl apply -f "C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_01\service.yaml"

After creating the service, I verified that everything was running correctly by executing the command:

kubectl get all -n staging

```
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS> kubectl apply -f "C:\Users\vp
netes_on_AWS\Exercise Files\Chapter_03\03_01\service.yaml"
service/pod-service unchanged
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS> kubectl get all -n staging
NAME                            READY   STATUS    RESTARTS   AGE
pod/pod-info-59dd8d5699-4fkk4   1/1     Running   0          27m
pod/pod-info-59dd8d5699-prrng   1/1     Running   0          27m

NAME                  TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/pod-service   NodePort   10.100.192.97   <none>        80:30174/TCP   5m11s

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/pod-info   2/2     2            2           27m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/pod-info-59dd8d5699   2         2         2       27m
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS>
```

Making subnets discoverable

In this step, I will be adding the appropriate tags to private and public subnets.

Adding tags to Private and Public subnets

It is important to add tags to private and public subnets to make the AWS load balancer controller to find the subnets it needs to create an Elastic load balancer.

I added the following tags to two private subnets

kubernetes.io/cluster/lil-eks: shared

kubernetes.io/role/internal-elb: 1

I added the following tags to two public subnets

kubernetes.io/cluster/lil-eks: shared

kubernetes.io/role/elb: 1

Create and AWS IAM policy bound to a Kubernetes Service account

In this step, I will create an IAM policy to allow the Kubernetes service to manage and utilize the AWS Elastic Load Balancer service.

I created the IAM policy by writing the necessary configurations in an iam_policy.json file. This file includes the required permissions for the IAM service-linked role to interact directly with AWS services, the policy that links the role to the Elastic Load Balancer, and the specific actions and services the role is allowed to access.

I created the policy by running the following command:

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

Next, I enabled the IAM OIDC provider for the policy by running the command:
eksctl utils associate-iam-oidc-provider --cluster project-eks –approve

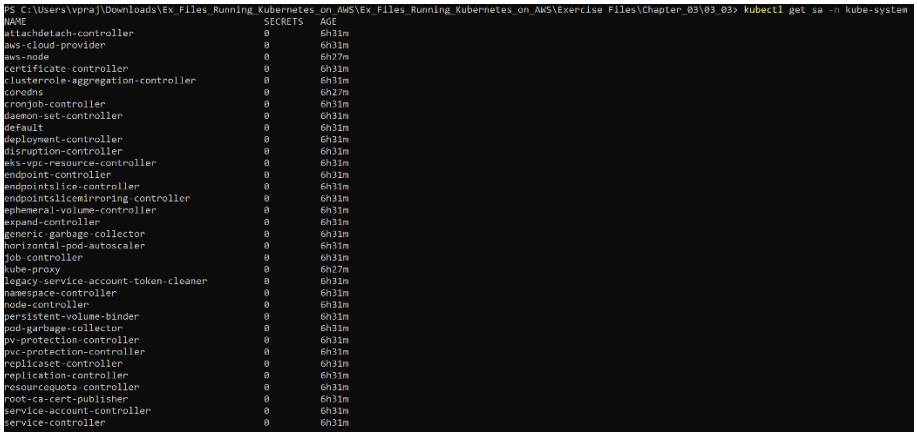
This step ensures that the EKS cluster can securely interact with AWS services through IAM roles.

After enabling OIDC, I created a Kubernetes service account and linked it to the IAM policy by executing the following command:

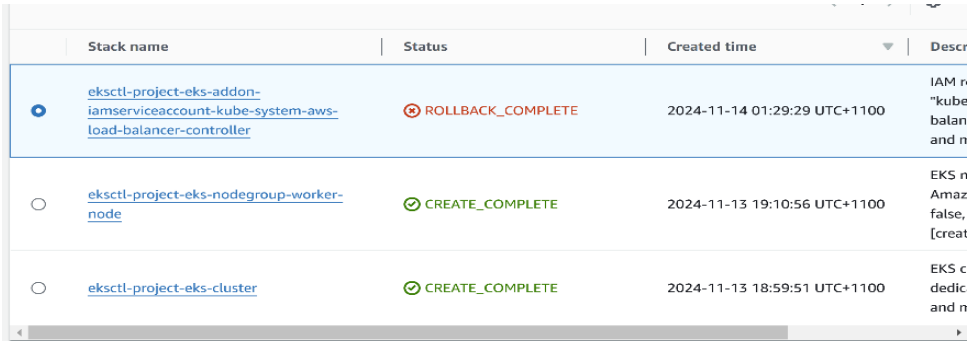
eksctl create iamserviceaccount `
--cluster=project-eks `
--name=aws-load-balancer-controller `
--namespace=kube-system `
--attach-policy-arn=arn:aws:iam:(account number of AWS**)**
:policy/AWSLoadBalancerControllerIAMPolicy `
--approve

Troubleshoot service account issues

When I ran the command **kubectl get sa -n kube-system** to check the service accounts, I noticed that the AWS Load Balancer Controller policy I had created was not listed. This indicated that something went wrong in AWS.



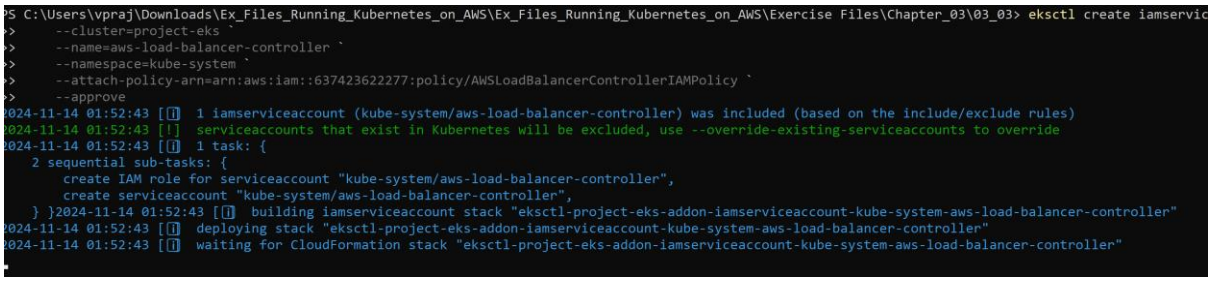
After investigating, I found that the issue was related to a failed CloudFormation stack. One of the stacks that were supposed to be created during the process did not complete successfully. This failure prevented the policy from being properly applied, which in turn caused the service account to not reflect the AWS Load Balancer Controller policy.



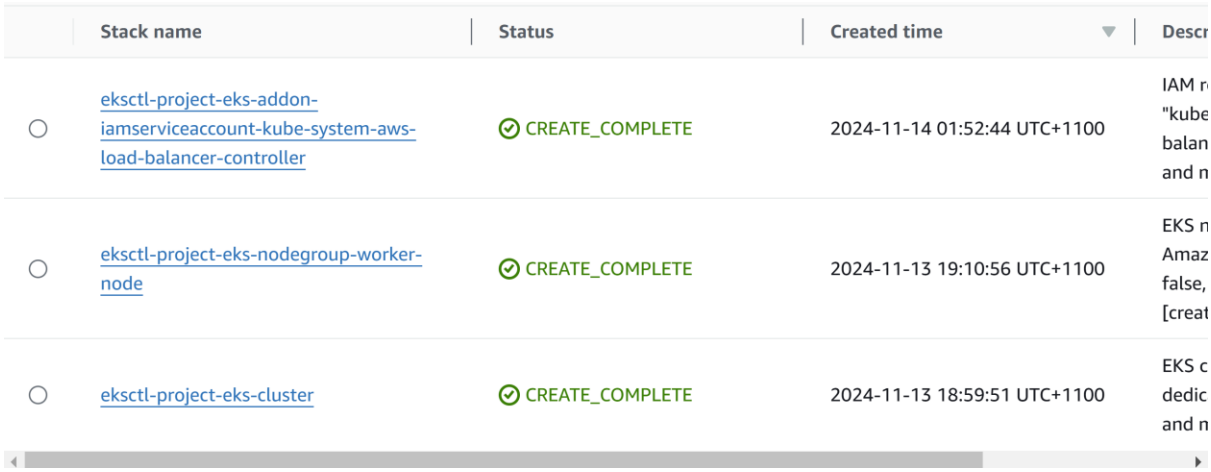
Now, I will delete the existing stack and then attempt to recreate the Kubernetes service account linked to the new IAM policy by running the following command:

eksctl create iamserviceaccount `

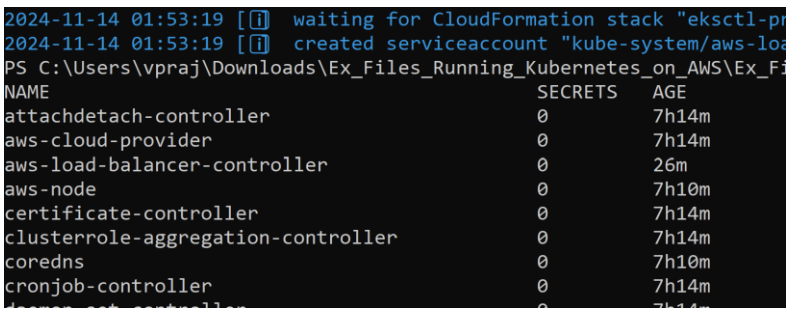
--cluster=project-eks `
--name=aws-load-balancer-controller `
--namespace=kube-system `
--attach-policy-arn=arn:aws:iam:(account number of AWS**)**
:policy/AWSLoadBalancerControllerIAMPolicy `
--approve



Now, I can see the stack being created successfully.



After deleting the stack and running the command, I verified that the aws -load-balancer-controller was created in the list.



Install the AWS load balancer controller

In this step, I will be installing both the cert-manager and the load balancer controller to integrate with the EKS cluster. To begin, I installed the cert-manager by executing the following command:

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

```
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_03> kubectl apply `
>>     --validate=false `
>>     -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
namespace/cert-manager created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrole.rbac.authorization.k8s.io/cert-manager-view created
clusterrole.rbac.authorization.k8s.io/cert-manager-edit created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-approve:cert-manager-io created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificatesigningrequests created
clusterrole.rbac.authorization.k8s.io/cert-manager-webhook:subjectaccessreviews created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-approve:cert-manager-io created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-controller-certificatesigningrequests created
clusterrolebinding.rbac.authorization.k8s.io/cert-manager-webhook:subjectaccessreviews created
role.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
role.rbac.authorization.k8s.io/cert-manager:leaderelection created
role.rbac.authorization.k8s.io/cert-manager-webhook:dynamic-serving created
rolebinding.rbac.authorization.k8s.io/cert-manager-cainjector:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager:leaderelection created
rolebinding.rbac.authorization.k8s.io/cert-manager-webhook:dynamic-serving created
service/cert-manager created
service/cert-manager-webhook created
deployment.apps/cert-manager-cainjector created
```

I verified the cert-manager pods created by running the command **kubectl get pods -n cert-manager**

```
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_K
ubernetes_on_AWS\Exercise Files\Chapter_03\03_03> kubectl get pods -n cert-manage
r
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-6bd59b759b-mfzpf              1/1     Running   0          51m
cert-manager-cainjector-8f4d48c6c-dzgh2    1/1     Running   0          51m
cert-manager-webhook-fdfb54b64-j2mnm       1/1     Running   0          51m
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_K
ubernetes_on_AWS\Exercise Files\Chapter_03\03_03>
```

What is Cert-manager

Cert Manager is a Kubernetes extension that streamlines the process of creating and managing TLS certificates, essential for enabling HTTPS. The AWS Load Balancer Controller relies on Cert Manager to secure the traffic entering and leaving your cluster through encryption.

What is AWS load balancer controller

The AWS Load Balancer Controller oversees the load balancer for the Kubernetes cluster, monitoring ingress events from the Kubernetes API server. When it detects ingress resources that meet specific criteria, it triggers the creation of corresponding AWS resources.

To set up the AWS Load Balancer Controller, I executed the following command:

kubectl apply -f "C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_05\load-balancer-controller.yaml"

To verify, I ran the command:

kubectl get deployment -n kube-system

```
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_03> kubectl apply -f "C:\Users\vpraj\
Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_05\load-balancer-controller.yaml"
customresourcedefinition.apiextensions.k8s.io/ingressclassparams.elbv2.k8s.aws created
customresourcedefinition.apiextensions.k8s.io/targetgroupbindings.elbv2.k8s.aws created
role.rbac.authorization.k8s.io/aws-load-balancer-controller-leader-election-role created
clusterrole.rbac.authorization.k8s.io/aws-load-balancer-controller-role created
rolebinding.rbac.authorization.k8s.io/aws-load-balancer-controller-leader-election-rolebinding created
clusterrolebinding.rbac.authorization.k8s.io/aws-load-balancer-controller-rolebinding created
service/aws-load-balancer-webhook-service created
deployment.apps/aws-load-balancer-controller created
certificate.cert-manager.io/aws-load-balancer-serving-cert created
issuer.cert-manager.io/aws-load-balancer-selfsigned-issuer created
mutatingwebhookconfiguration.admissionregistration.k8s.io/aws-load-balancer-webhook created
validatingwebhookconfiguration.admissionregistration.k8s.io/aws-load-balancer-webhook created
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_03> kubectl get deployment -n kube-sy
stem
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   1/1     1            1           28s
coredns                        2/2     2            2           137m
PS C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_03>
```

After installing the AWS Load Balancer Controller, I checked the AWS console to verify if the load balancer was created. However, I found that no load balancer had been created yet because there were no Kubernetes ingress objects available to trigger its creation.

Create an ingress

What is Ingress

Ingress is a resource in Kubernetes that manages the routing of HTTP and HTTPS traffic from outside the cluster to the services within the cluster. It defines rules for directing traffic to specific services based on the request path or host. The Ingress Controller is responsible for enforcing and fulfilling the ingress rules, ensuring that traffic is routed correctly to the appropriate services within the cluster.

Why do we need to create ingress class

To enable the AWS Load Balancer Controller to connect to the application ingress, it is essential to create an ingress resource. This is done by running the ingress-class.yaml script, which defines the configuration for IngressClass and IngressParams, specifying the resources required for the load balancer controller to function properly.

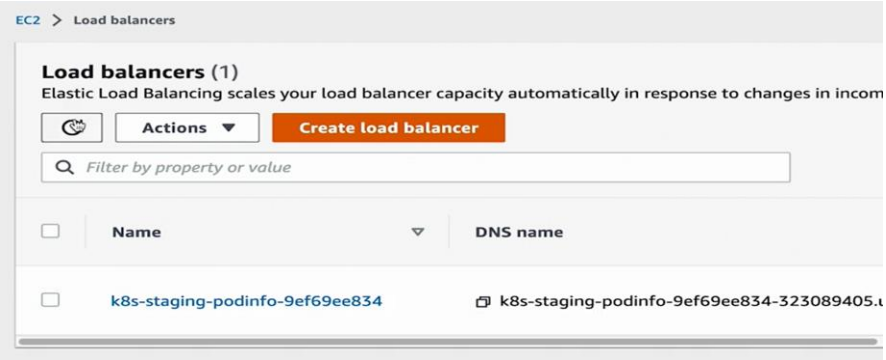
To create the ingress, I ran the following command:

kubectl apply -f "C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_05\ingress-class.yaml"

Afterward, I created the Kubernetes ingress resource for the application named "pod-info" under the staging namespace. To do this, I created the pod-info-ingress.yaml script, which includes the necessary configuration for the load balancer.I executed the creation of the ingress resource by running the following command:

Kubectl apply -f "C:\Users\vpraj\Downloads\Ex_Files_Running_Kubernetes_on_AWS\Ex_Files_Running_Kubernetes_on_AWS\Exercise Files\Chapter_03\03_06\pod-info-ingress.yaml"

Now, I see the load balance being created in the console.



I verified the application's availability on the internet by using the DNS name of the load balancer and accessing it through a web browser.