A blue and white logo

Description automatically generated**Running Kubernetes on AWS (EKS)**

**Prajit Venkatachalam**

[**https://www.linkedin.com/in/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**

A computer screen shot of a program code

Description automatically generated

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**

A computer screen shot of text

Description automatically generated

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**

A computer screen shot of text

Description automatically generated

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

**Creating an EKS admin user group and user**

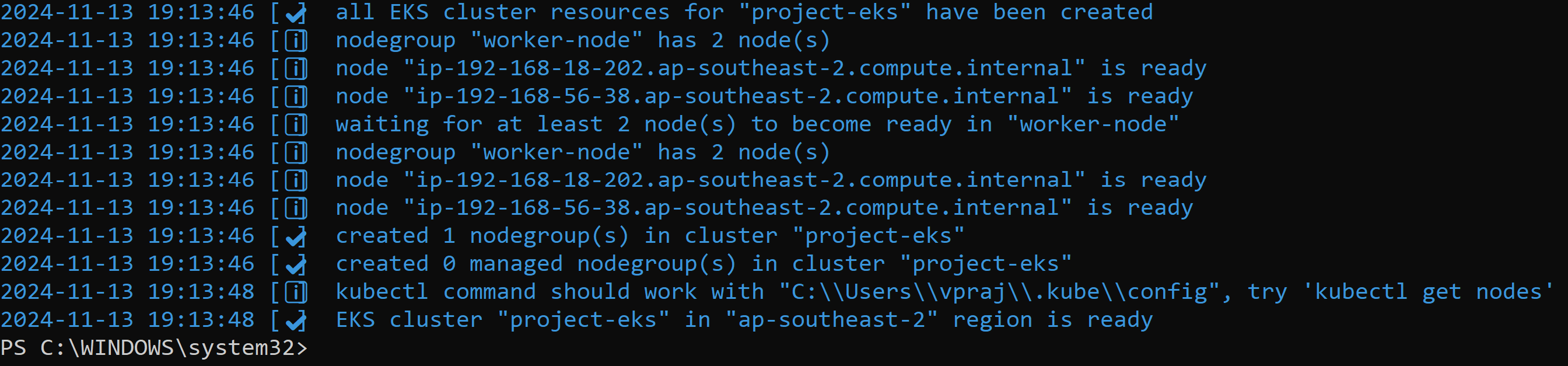
I created a user group named `eks-admin` and assigned it the Administrator Access policy from the permissions tab, granting it full access to AWS resources. Next, I created a user and added them to the `eks-admin` group.

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"**



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.

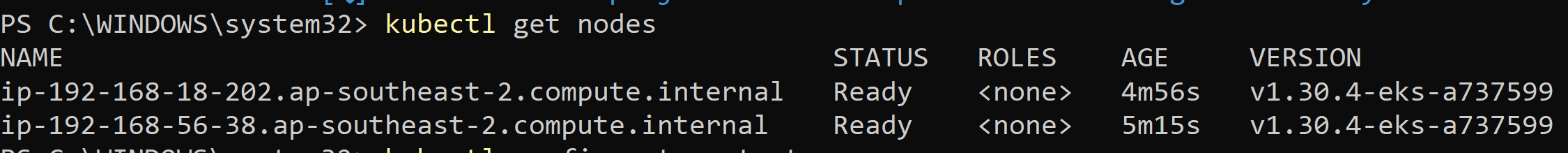
A screenshot of a computer

Description automatically generated

**Exploring the existing resources in my EKS cluster**

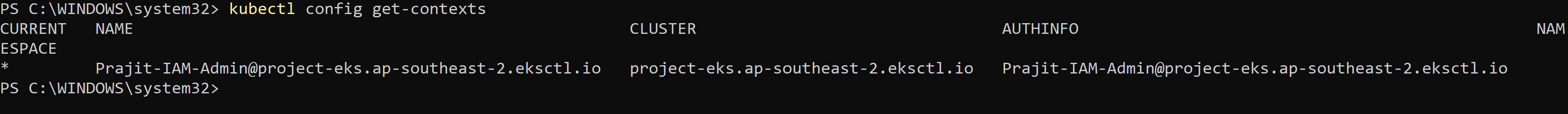
To view the list of nodes in the cluster, I ran the command:

**Kubectl get nodes**



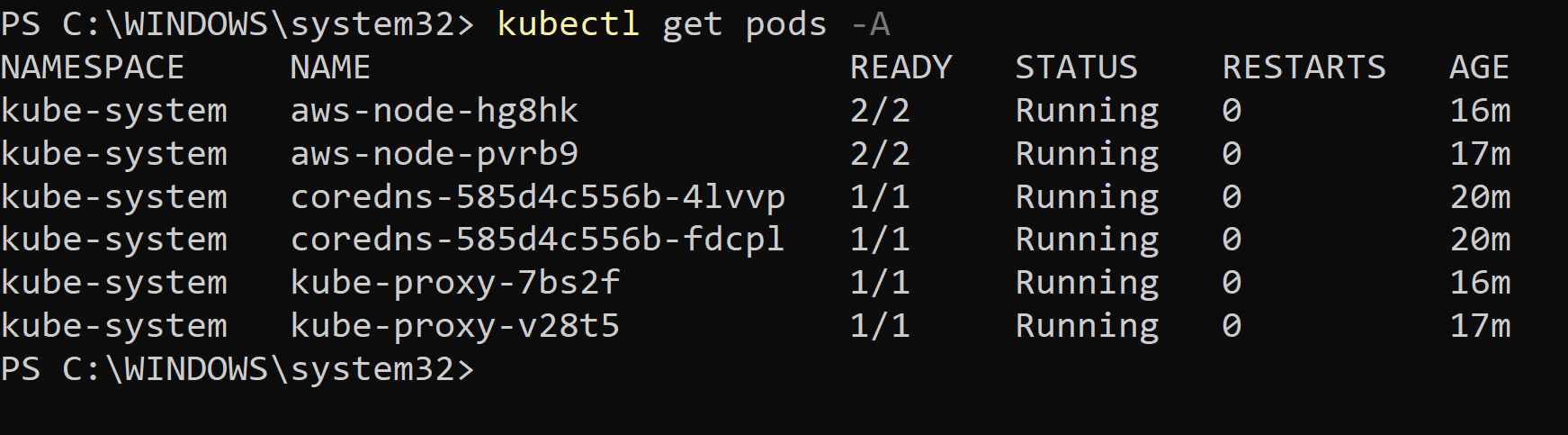
To check which cluster is currently configured on my computer, I used:

**Kubectl config get-contexts**



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

**Kubectl get pods -A**



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

A screenshot of a computer

Description automatically generated

**Deploying an application to my EKS cluster**

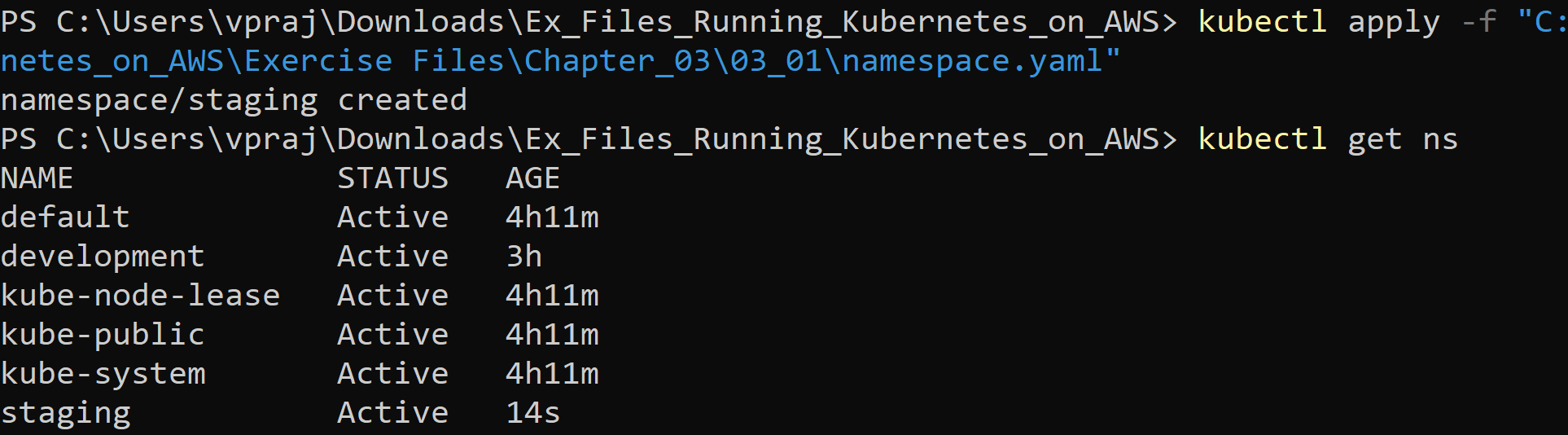
In this step, first I will create a Kubernetes namespace, deploy an application to the cluster, and create a service to prepare my application to be available to the internet.

**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**



**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**

**A screen shot of a computer code

Description automatically generated**

**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**

**A computer screen with white and blue text

Description automatically generated**

**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**](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.

A black screen with white text

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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**

A black screen with many colored lines

Description automatically generated with medium confidence

Now, I can see the stack being created successfully.

A screenshot of a computer

Description automatically generated

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

**A screen shot of a computer

Description automatically generated**

**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**](https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml)

A computer screen shot of a black screen

Description automatically generated

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

**A computer screen shot of a black background

Description automatically generated**

**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**

**A computer screen with white text

Description automatically generated**

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. **A screenshot of a computer

Description automatically generated**

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.

A blue square with white spots

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated