Deploy Nginx Ingress Controller on AWS EKS using Terraform and Helm

Overview

This document outlines the process of setting up an NGINX Ingress Controller on an AWS EKS cluster using **Terraform** and **Helm**. The NGINX Ingress Controller manages external access to services within the Kubernetes cluster and helps route traffic to the appropriate services based on defined rules.

Prerequisites

Before starting, ensure you have the following tools and services installed:

1. **Terraform** (>= 1.6.0)
2. **AWS CLI** (configured with IAM permissions)
3. **kubectl** (Kubernetes CLI)
4. **Helm** (Package manager for Kubernetes)
5. **An AWS EKS Cluster** (already set up)

Installation Instructions

Follow these steps to install the required dependencies on an Ubuntu-based system.

1. Install Terraform

```bash

# Update and upgrade system packages

sudo apt update && sudo apt upgrade -y

# Install necessary dependencies

sudo apt install -y gnupg software-properties-common curl unzip

# Add HashiCorp GPG key and repository
```

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

Update package lists and install Terraform

sudo apt update && sudo apt install -y terraform

```

#### 2. Install kubectl

```bash

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo chmod +x kubectl

sudo mv kubectl /usr/local/bin/

sudo kubectl version --client

```

3. Install Helm

```bash

curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

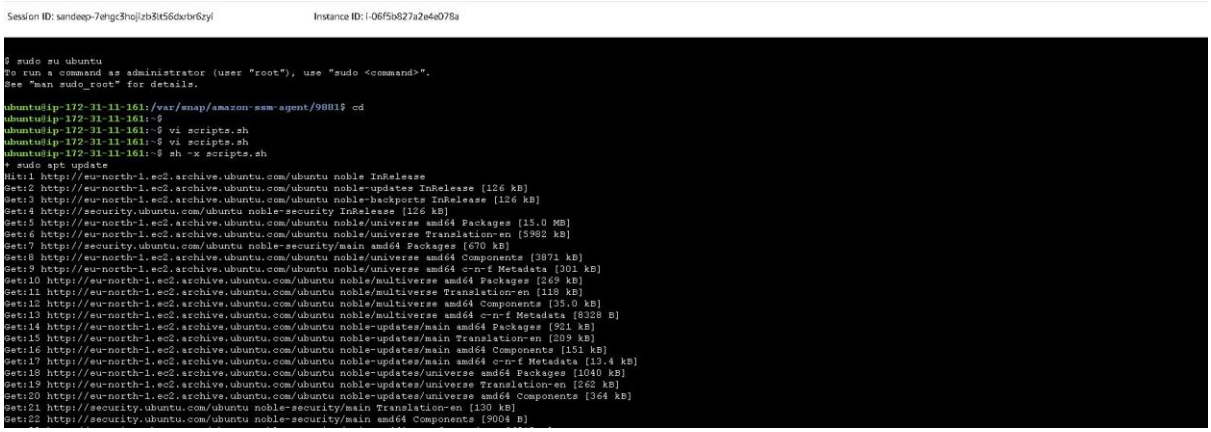
sudo helm version

```

4. Install AWS CLI
```

```bash
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

To automate the installation, you can use the provided `scripts.sh` script.



Configure AWS Credentials

Run the following command to configure AWS CLI:

```bash
aws configure
```

Step 1: Create Terraform Configuration Files

1. **Create Terraform Directory:**

```bash
mkdir -p terraform
cd terraform/
```

2. **Create Terraform Files:**
   - **main.tf:** Define the Virtual Private Cloud (VPC) setup.
   - **eks.tf:** Define the AWS EKS cluster configuration.
   - **iam.tf:** Define the IAM roles.
   - **outputs.tf:** Define outputs for the configuration.
   - **variables.tf:** Define variables for the configuration.

3. **Initialize Terraform:**

```bash
terraform init
```

Session ID: sandeep-7ehgc3hojizb3lt56dxrbr6zyi Instance ID: i-06f5b827a2e4e078a

```
ubuntu@ip-172-31-11-161:~$ mkdir -p terraform
ubuntu@ip-172-31-11-161:~$ cd terraform/
ubuntu@ip-172-31-11-161:~/terraform$ vi provider.tf
ubuntu@ip-172-31-11-161:~/terraform$ vi vpc.tf
ubuntu@ip-172-31-11-161:~/terraform$ vi eks.tf
ubuntu@ip-172-31-11-161:~/terraform$ vi node-group.tf
ubuntu@ip-172-31-11-161:~/terraform$ vi outputs.tf
ubuntu@ip-172-31-11-161:~/terraform$ terraform init
Initializing the backend...
Initializing modules...
Downloading registry.terraform.io/terraform-aws-modules/eks/aws 20.0.0 for eks...
- eks in .terraform/modules/eks
- eks.eks_managed_node_group in .terraform/modules/eks/modules/eks-managed-node-group
- eks.eks_managed_node_group.user_data in .terraform/modules/eks/modules/_user_data
- eks.fargate_profile in .terraform/modules/eks/modules/fargate-profile
Downloading registry.terraform.io/terraform-aws-modules/kms/aws 2.1.0 for eks.kms...
- eks.kms in .terraform/modules/eks.kms
- eks.self_managed_node_group in .terraform/modules/eks/modules/self-managed-node-group
- eks.self_managed_node_group.user_data in .terraform/modules/eks/modules/_user_data
Downloading registry.terraform.io/terraform-aws-modules/eks/aws 20.34.0 for eks_managed_node_group...
- eks_managed_node_group in .terraform/modules/eks_managed_node_group/modules/eks-managed-node-group
- eks_managed_node_group.user_data in .terraform/modules/eks_managed_node_group/modules/_user_data
Downloading registry.terraform.io/terraform-aws-modules/vpc/aws 5.0.0 for vpc...
- vpc in .terraform/modules/vpc
Initializing provider plugins...
- Finding hashicorp/null versions matching ">= 3.0.0"...
- Finding hashicorp/aws versions matching ">= 4.33.0, >= 5.0.0, >= 5.34.0, >= 5.83.0"...
- Finding hashicorp/tls versions matching ">= 3.0.0"...
- Finding hashicorp/time versions matching ">= 0.9.0"...
- Finding hashicorp/cloudinit versions matching ">= 2.0.0"...
- Installing hashicorp/aws v5.91.0...
- Installed hashicorp/aws v5.91.0 (signed by HashiCorp)
- Installing hashicorp/tls v4.0.6...
- Installed hashicorp/tls v4.0.6 (signed by HashiCorp)
- Installing hashicorp/time v0.13.0...
- Installed hashicorp/time v0.13.0 (signed by HashiCorp)
- Installing hashicorp/cloudinit v2.3.6...
- Installed hashicorp/cloudinit v2.3.6 (signed by HashiCorp)
- Installing hashicorp/null v3.2.3...
- Installed hashicorp/null v3.2.3 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
```

4. **Plan and Apply the Configuration:**

```bash
terraform plan
terraform apply -auto-approve
```

Session ID: sandeep-32l6az7rktt9qoa9rku7ccl764 Instance ID: i-06f5b827a2e4e078a

```
ubuntu@ip-172-31-11-161:~/terraform$ terraform apply -auto-approve
module.eks.module.eks_managed_node_group["general"].data.aws_caller_identity.current: Reading...
module.eks.module.eks_managed_node_group["general"].data.aws_iam_policy_document.assume_role_policy[0]: Reading...
module.eks.module.eks_managed_node_group["general"].data.aws_partition.current: Reading...
module.eks.data.aws_caller_identity.current: Reading...
module.eks.data.aws_partition.current: Reading...
module.eks.module.kms.data.aws_caller_identity.current[0]: Reading...
module.eks.module.kms.data.aws_partition.current[0]: Reading...
module.eks.data.aws_iam_policy_document.assume_role_policy[0]: Reading...
module.eks.data.aws_iam_policy_document.assume_role_policy[0]: Read complete after 0s [id=2764486067]
module.eks.module.kms.data.aws_partition.current[0]: Read complete after 0s [id=aws]
module.eks.data.aws_partition.current: Read complete after 0s [id=aws]
module.eks.module.kms.data.aws_caller_identity.current[0]: Read complete after 0s [id=794038256791]
module.eks.module.eks_managed_node_group["general"].data.aws_partition.current: Read complete after 0s [id=aws]
module.eks.module.eks_managed_node_group["general"].data.aws_iam_policy_document.assume_role_policy[0]: Read complete after 0s [id=2560088296]
module.eks.module.eks_managed_node_group["general"].data.aws_caller_identity.current: Read complete after 0s [id=794038256791]
```

Step 2: Install NGINX Ingress Controller using Helm

1. **Create a new directory for NGINX Ingress deployment:**

```bash
mkdir terraform-nginx-ingress && cd terraform-nginx-ingress
terraform init
terraform apply -auto-approve
```

2. **Update the kubeconfig file to access the EKS cluster:**

```bash
aws eks update-kubeconfig --region eu-north-1 --name stylish-threads-cluster
```

The output will confirm the addition of a new context to the kubeconfig file:

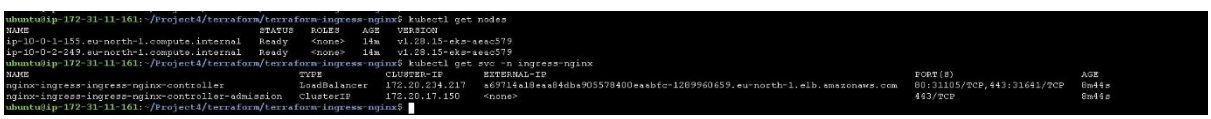
```
Added new context arn:aws:eks:eu-north-1:794038256791:cluster/stylish-threads-cluster to /home/ubuntu/.kube/config
```

3. **Get a token for accessing the EKS cluster:**

```bash
aws eks get-token --cluster-name stylish-threads-cluster
```

4. **Verify Cluster Access:**

```bash
kubectl get nodes
kubectl get svc -n ingress-nginx
```



At this point, the NGINX Ingress Controller service should be running with an external IP address.
Ingress Controller Service is running – The LoadBalancer has an external IP:

a69714a18eaa84dba905578400eaabfc-1289960659.eu-north-1.elb.amazonaws.com

Ports 80 and 443 are exposed.

Step 3: Test the Ingress Controller

Test the NGINX Ingress Controller by sending an HTTP request to the external IP address.

```bash
curl -I http://a69714a18eaa84dba905578400eaabfc-1289960659.eu-north-1.elb.amazonaws.com
```

You should see an HTTP 404 (which is normal since no ingress rules are defined yet).

Step 4: Deploy an NGINX Application Behind the Ingress Controller

1. Deploy NGINX Application

Create a deployment and service for the NGINX application.

nginx-deployment.yaml:

Apply the YAML file to create the deployment and service:

```bash

kubectl apply -f nginx-deployment.yaml

```

2. Create an Ingress Resource

Now, create an Ingress resource to route traffic to the NGINX service.

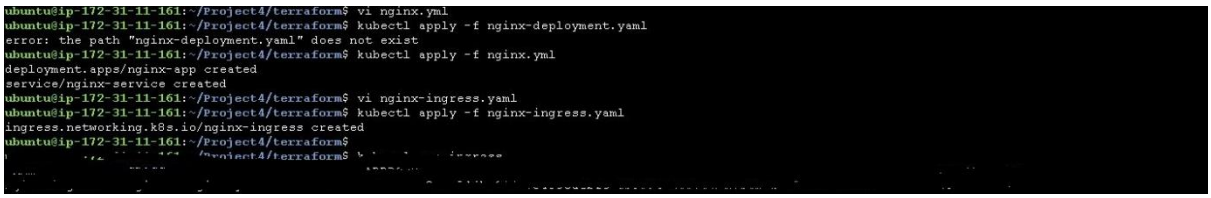
nginx-ingress.yaml:

Apply the Ingress resource:

```bash

kubectl apply -f nginx-ingress.yaml

```



Step 5: Verify Deployment

To verify the deployment, you can use the following commands:

1. **Test the Ingress:**

```bash

curl -I http://a69714a18eaa84dba905578400eaabfc-1289960659.eu-north-1.elb.amazonaws.com/

```

```

```
ubuntu@ip-172-31-11-161:~$ kubectl get ingress
NAME   CLASS   HOSTS   ADDRESS   PORTS   AGE
nginx-ingress   nginx   *   a69714a18eaa84dba905578400eaabfc-1289960659.eu-north-1.elb.amazonaws.com   80   18m
ubuntu@ip-172-31-11-161:~$ curl -I http://a69714a18eaa84dba905578400eaabfc-1289960659.eu-north-1.elb.amazonaws.com/
HTTP/1.1 200 OK
Date: Wed, 19 Mar 2025 14:47:15 GMT
Content-Type: text/html
Content-Length: 615
Connection: keep-alive
Last-Modified: Wed, 05 Feb 2025 11:06:32 GMT
ETag: "67a34638-267"
Accept-Ranges: bytes

ubuntu@ip-172-31-11-161:~$
```

2. **Access via Browser:**

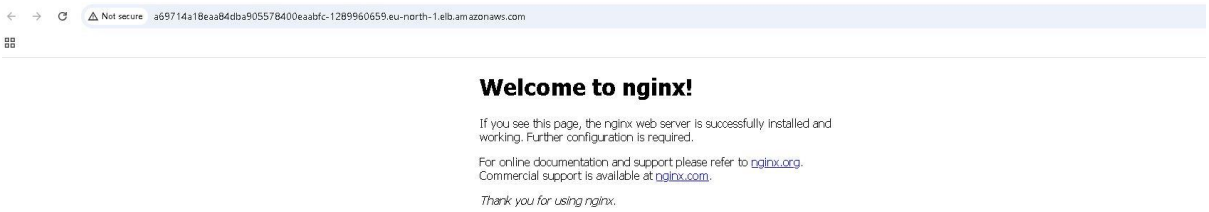
   Open a web browser and navigate to the external IP address:

   ```

   http://a69714a18eaa84dba905578400eaabfc-1289960659.eu-north-1.elb.amazonaws.com/

   ```

You should see the default NGINX welcome page if everything is set up correctly.




## Conclusion

This guide has provided detailed instructions to deploy the NGINX Ingress Controller on an AWS EKS cluster using Terraform and Helm. With the Ingress Controller successfully deployed, you can now manage external access to your Kubernetes services and route traffic based on defined Ingress rules.

```