Vanshika Ambwani D15A 02

ADVANCE DEVOPS CASE STUDY

INTRODUCTION

Case Study Overview: This case study focuses on deploying and managing a Kubernetes environment on AWS, using tools like kubectl and eksctl for cluster management. The deployment involves creating an EKS (Elastic Kubernetes Service) cluster, deploying an Nginx web server, and exposing it using a LoadBalancer to demonstrate real-world Kubernetes operations. These steps illustrate best practices for scaling and managing containerized applications in a cloud environment.

AIM

The objective of this project is to set up a Kubernetes cluster on AWS using the AWS Cloud9 IDE, deploy a sample Nginx application using kubectl, and verify its deployment by accessing it through a LoadBalancer or NodePort.

THEORY

1. Kubernetes
   Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps ensure that the application remains available even in cases of system failure, making it an essential tool for managing largescale applications.

   Advantages of Kubernetes:
   • Scalability: Automates the scaling of applications.
   • Fault Tolerance: Automatically replaces or restarts failed containers.
   • Portability: Applications can be run consistently across different environments.
   • Efficiency: Optimizes resource utilization by distributing workloads dynamically

2. Amazon Elastic Kubernetes Service (EKS) Amazon EKS is a managed Kubernetes service that simplifies running Kubernetes clusters on AWS. It automates much of the operational overhead, allowing developers to focus on deploying their applications instead of managing the underlying infrastructure.
   Advantages of EKS:
   • Managed Service: Reduces the need to manage Kubernetes control plane and nodes manually.
   • AWS Integration: Seamlessly integrates with AWS services like IAM, CloudWatch, and load balancers.
   • High Availability: Offers reliability through AWS's scalable and secure infrastructure.

3. Nginx Nginx is a popular, high-performance web server commonly used for serving web pages, load balancing, and acting as a reverse proxy.
   Advantages of Nginx:
   • High Performance: Known for efficiently handling large numbers of concurrent connections.
   • Flexibility: Can be used for load balancing, caching, and reverse proxying.

Why Use Kubernetes to Deploy Nginx on EKS?

• Automation: You can automate the deployment, scaling, and management of Nginx instances across multiple servers, ensuring high availability and performance.

• Load Balancing: Kubernetes' built-in service features, like the LoadBalancer, allow you to automatically distribute traffic among Nginx instances.

• Portability and Flexibility: Kubernetes can run the same Nginx container on any environment (local, cloud, hybrid), making deployments consistent and portable

In this case study we are following the below procedure to deploy simple nginx application:

1. Access AWS Cloud Shell: Log in to the AWS Management Console and open Cloud Shell to use AWS's command-line interface. 2. Install Kubernetes CLI (kubectl):Download and install kubectl in CloudShell, which lets you control and manage Kubernetes clusters.

2. Configure AWS CLI: Set up AWS CLI with your credentials, which allows you to communicate with AWS services.

3. Create a Kubernetes Cluster:Use the eksctl tool to create a Kubernetes cluster on AWS with two worker nodes. This cluster is where your applications will run.

4. Deploy Nginx: Create a YAML file defining the Nginx deployment, specifying how many replicas (instances) you want and their configuration.

5. Expose Nginx with LoadBalancer: Use a LoadBalancer service to expose the Nginx application to the internet, allowing you to access it through an external IP address.

IMPLEMENTATION:

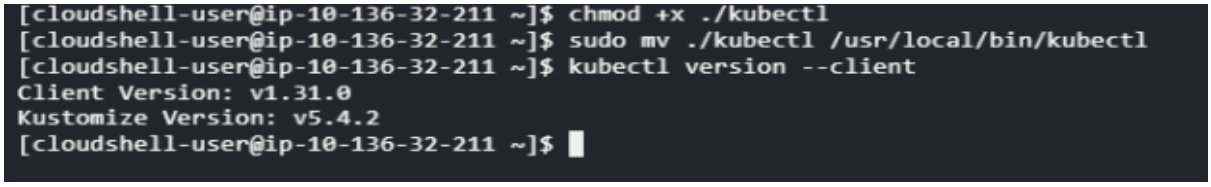
Step 1: Install and configure kubectl using AWS Cloudshell)

1.Accessing CloudShell:

Sign in to the AWS Management Console.
Click on the Cloudshell icon (located at the top right corner).

2. Install kubectl:

3.Make kubectl Executable

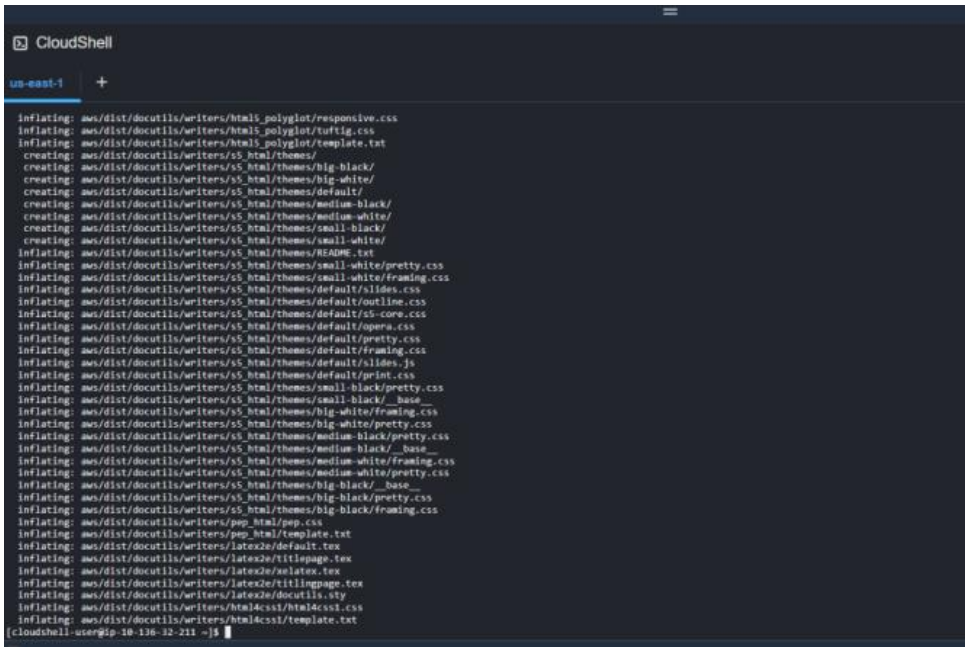


Step 2: Install AWS CLI

1.Installing AWS CLI:

install by running

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



```
[cloudshell-user@ip-10-136-32-211 ~]$ sudo ./aws/install
Found preexisting AWS CLI installation: /usr/local/aws-cli/v2/current. Please rerun install script with --update flag.
[cloudshell-user@ip-10-136-32-211 ~]$
```

```
  inflating: aws/dist/docutils/writers/html4css1/template.txt
[cloudshell-user@ip-10-136-32-211 ~]$ sudo ./aws/install
Found preexisting AWS CLI installation: /usr/local/aws-cli/v2/current. Please rerun install script with --update flag.
[cloudshell-user@ip-10-136-32-211 ~]$ sudo ./aws/install --update
You can now run: /usr/local/bin/aws --version
[cloudshell-user@ip-10-136-32-211 ~]$ /usr/local/bin/aws --version
aws-cli/2.18.11 Python/3.12.6 Linux/6.1.109-118.189.amzn2023.x86_64 exec-env/CloudShell exe/x86_64.amzn.2023
[cloudshell-user@ip-10-136-32-211 ~]$
```

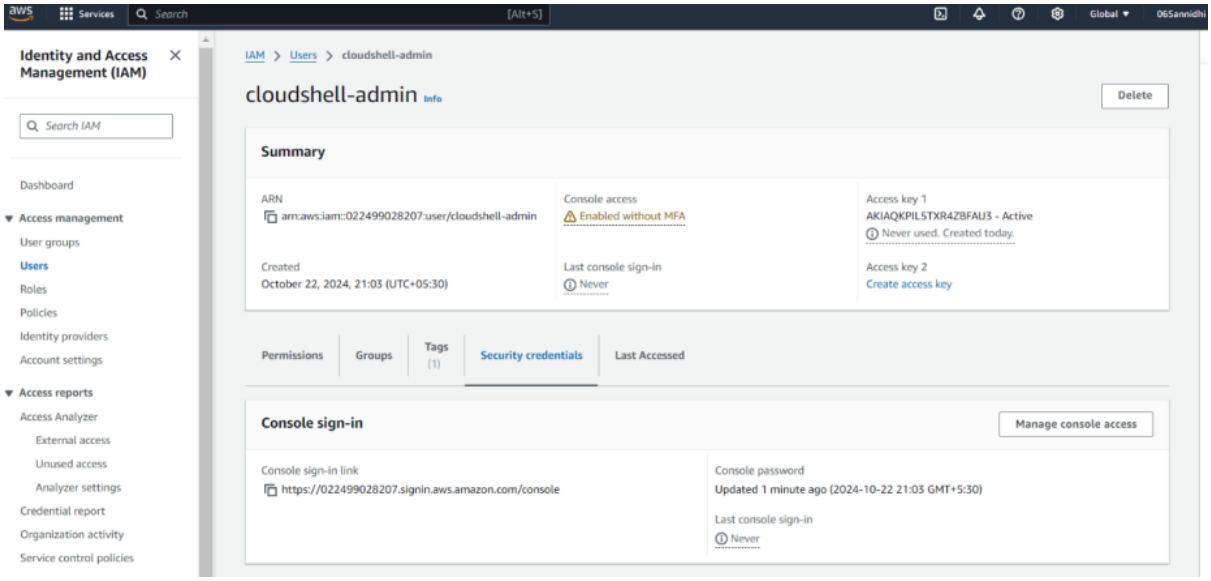
2. Setting up AWS CLI

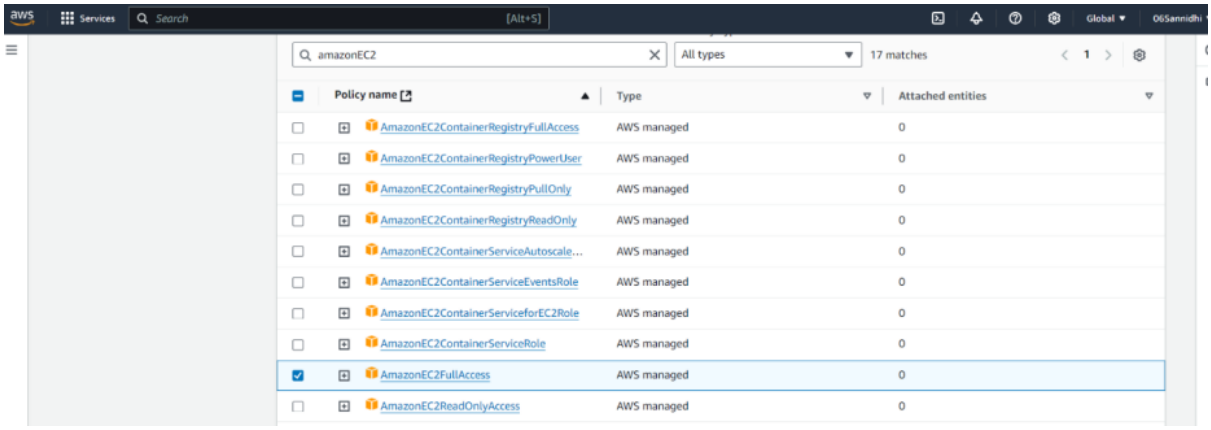
```
[cloudshell-user@ip-10-136-32-211 ~]$ aws configure
AWS Access Key ID [****************1004]: AKIAQKPIL5TXR4ZBFAU3
AWS Secret Access Key [****************none]: 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
Default region name [us-east-1]: us-east-1
Default output format [none]: text
[cloudshell-user@ip-10-136-32-211 ~]$
```

CloudShell Feedback

3.To Create Access Keys in AWS Management Console:

- Navigate to IAM
- Set User Details
- Attach Permissions
- Complete the Setup



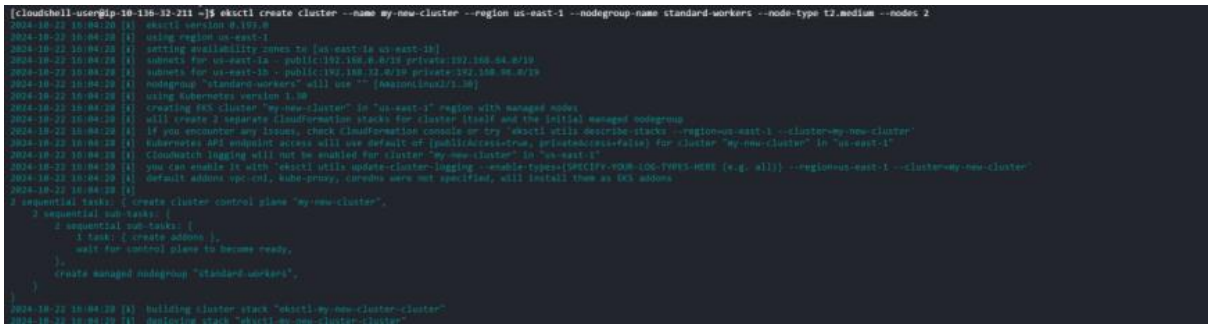


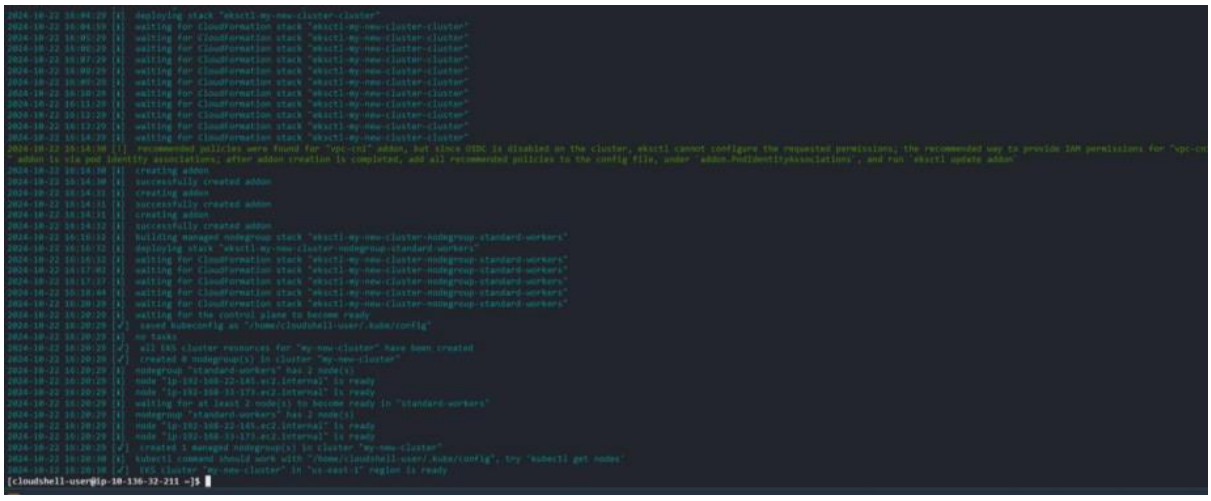
Step 3: Create an Amazon EKS Cluster

1. Installing eksctl

```
[cloudshell-user@ip-10-136-32-211 ~]$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)
[cloudshell-user@ip-10-136-32-211 ~]$ sudo mv /tmp/eksctl /usr/local/bin/eksctl
[cloudshell-user@ip-10-136-32-211 ~]$
```

2. Create a Kubernetes Cluster





Step 4: Deploying Nginx Application

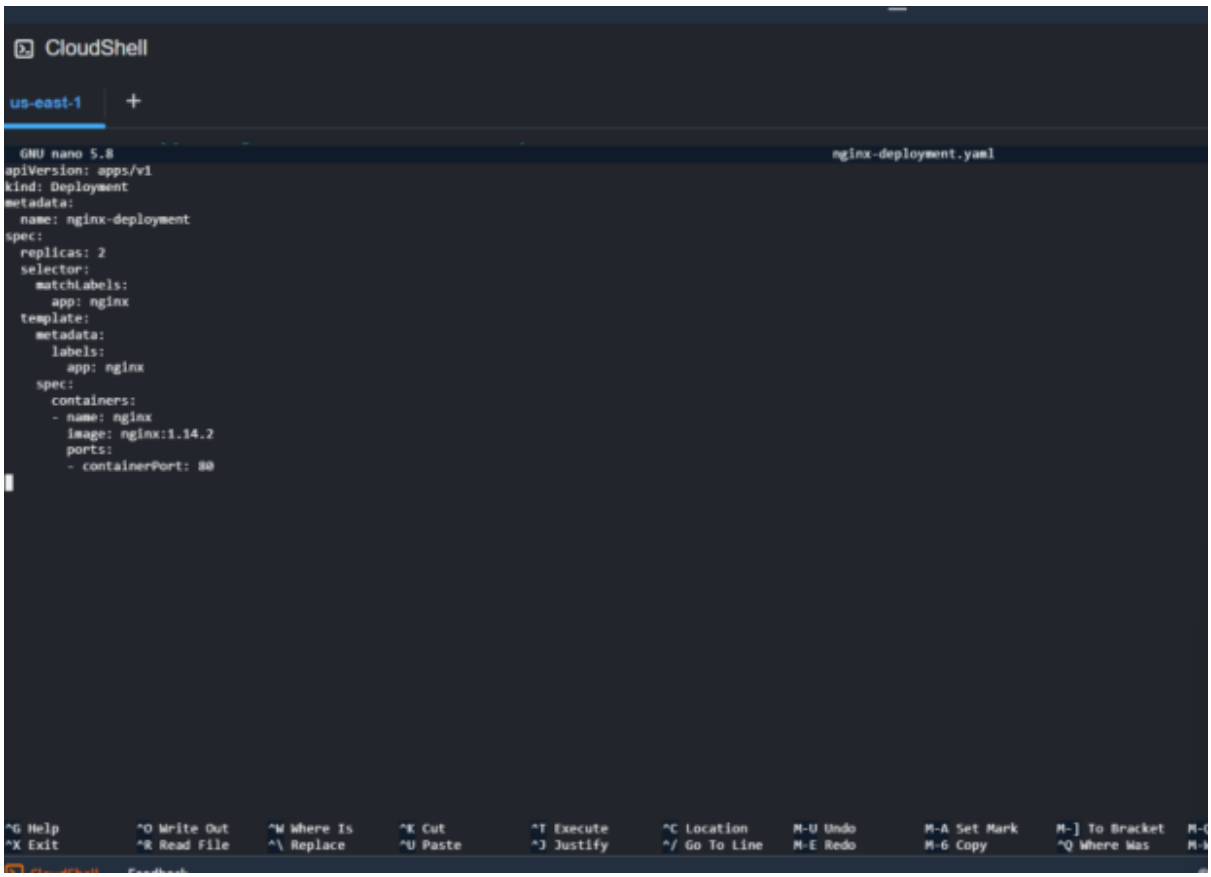
1. Create Deployment YAML

Create a YAML file to define your Nginx deployment.

Open Cloudshell and use a text editor to create the file (e.g., nano):

nano nginx-deployment.yaml

```
[cloudshell-user@ip-10-136-32-211 ~]$ nano nginx-deployment.yaml
[cloudshell-user@ip-10-136-32-211 ~]$
```



2. Deploy the Application

Apply the deployment using kubectl:

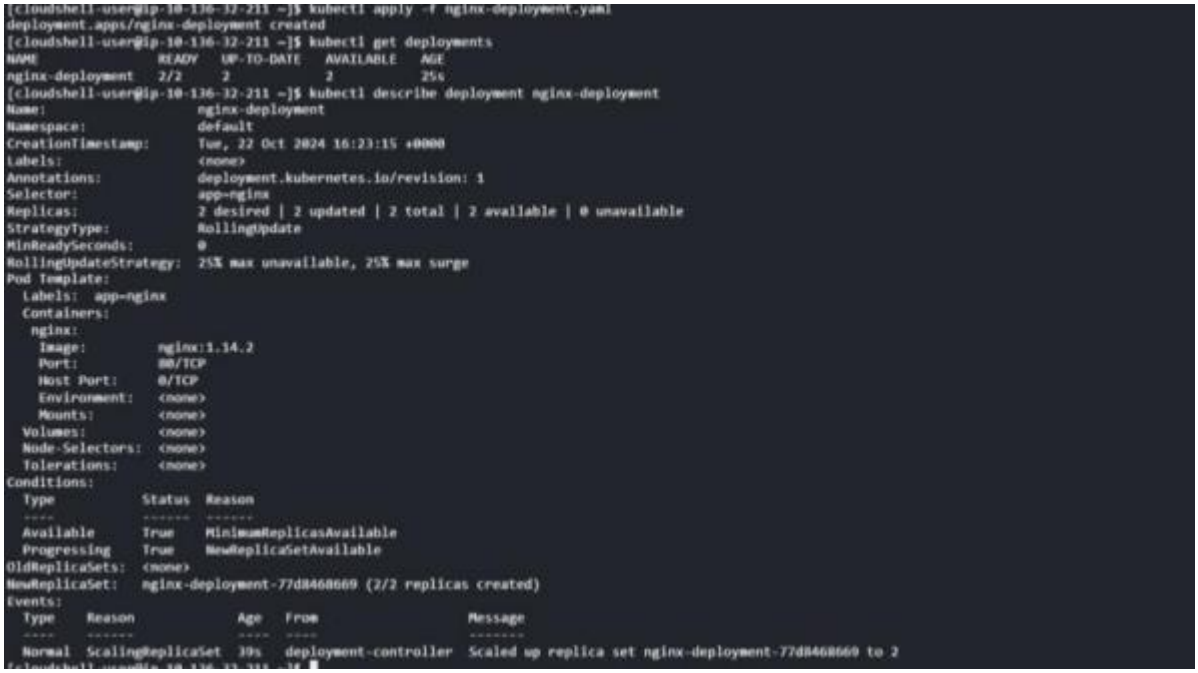
kubectl apply -f nginx-deployment.yaml

3. Verify Deployment

To verify the deployment, run:

kubectl get deployments

kubectl describe deployment nginx-deployment



Step 5: Expose the Application Using a LoadBalancer

1.Paste the following code:

yaml

Copy code a

piVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

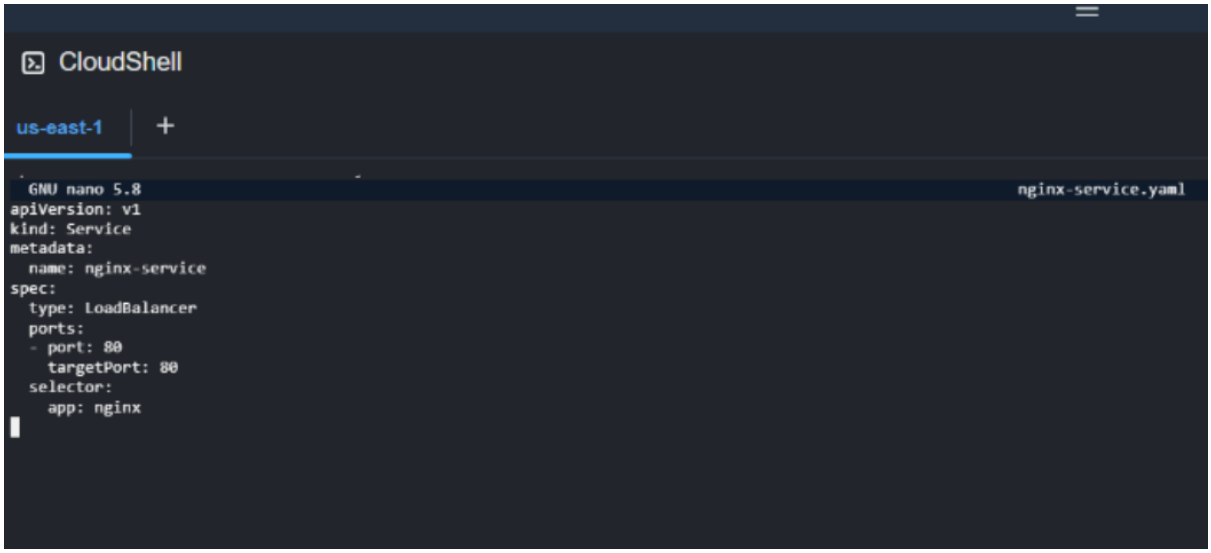
type: LoadBalancer

ports: - port: 80

targetPort: 80

selector:

app: nginx

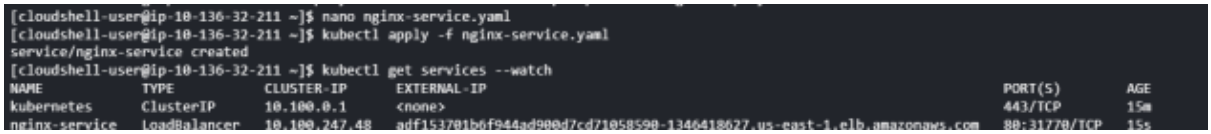


2.Applying the Service

Apply the LoadBalancer service:

kubectl apply -f nginx-service.yaml

3. Retrieve the external ip



CONCLUSION

This case study successfully demonstrates the setup of a Kubernetes cluster on AWS using Cloud9 IDE. We installed and configured kubectl, created an EKS cluster, deployed an Nginx application, and verified its accessibility using both NodePort and LoadBalancer services. Kubernetes simplifies application management and deployment, especially in cloud environments like AWS EKS.