ADVANCE DEVOPS CASESTUDY

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 large-scale 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. In this case, we will use Nginx as a sample application to deploy on the Kubernetes cluster.

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.
- Lightweight: Requires minimal resources while handling large-scale traffic efficiently.

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.

Advantages of Using Kubernetes and EKS for this Deployment:

- Simplified Management: EKS manages the Kubernetes control plane, while you focus on deploying Nginx and other applications.
- Scalability: With a few commands, you can scale Nginx across multiple servers to handle increased traffic.
- Automated Failover: Kubernetes ensures that your application remains available, even if some nodes fail.
- Cost Efficiency: By automatically scaling down when traffic is low, Kubernetes can help reduce costs on cloud resources.

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

1. Access AWS CloudShell: Log in to the AWS Management Console and open CloudShell 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.
3. Configure AWS CLI: Set up AWS CLI with your credentials, which allows you to communicate with AWS services.
4. 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.
5. Deploy Nginx: Create a YAML file defining the Nginx deployment, specifying how many replicas (instances) you want and their configuration.
6. 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: Setting Up kubectl in AWS CloudShell

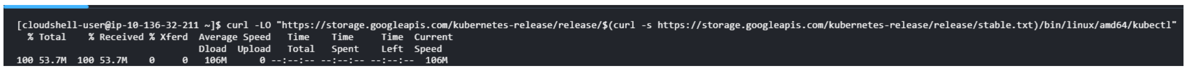
1. **Accessing CloudShell**

   Log into the AWS Management Console and click the CloudShell icon, which appears on the top-right corner. This opens a terminal interface directly within the browser, offering a secure environment to execute commands.

2. **Downloading kubectl**

   Execute the following command to download the latest version of kubectl, the command-line tool for Kubernetes management:

   - curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl



```
[cloudshell-user@ip-10-136-32-211 ~]$ curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 53.7M  100 53.7M    0     0   106M      0 --:--:-- --:--:-- --:--:--  106M
```

3. **Making kubectl Executable, moving it to the system path And veryfying**

   Write following commands to modify the permissions to make kubectl executable:

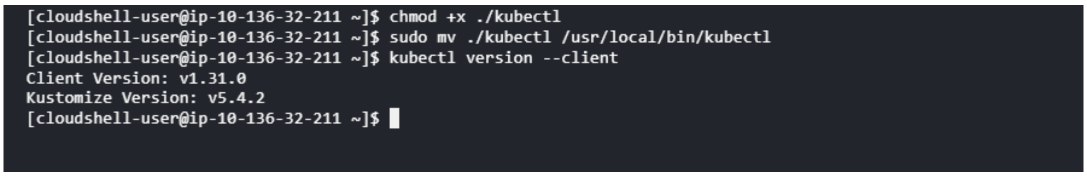
   - chmod +x ./kubectl

   Move the binary to a directory included in your system’s path to make it accessible from anywhere:

   - sudo mv ./kubectl /usr/local/bin/kubectl

   Verify the successful installation of kubectl:

   - kubectl version --client



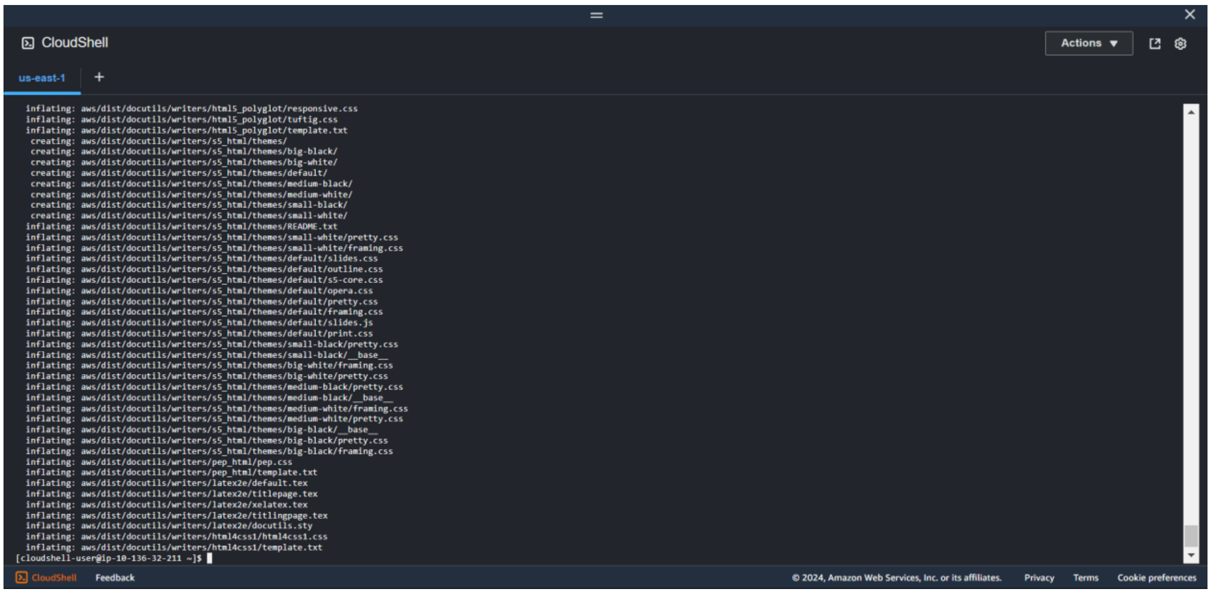
```
[cloudshell-user@ip-10-136-32-211 ~]$ chmod +x ./kubectl
[cloudshell-user@ip-10-136-32-211 ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl version --client
Client Version: v1.31.0
Kustomize Version: v5.4.2
[cloudshell-user@ip-10-136-32-211 ~]$ 
```

Step 2: Configuring AWS CLI

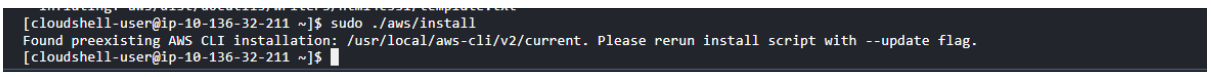
1. Installing AWS CLI

If the AWS CLI isn't already installed in CloudShell, install it by running:

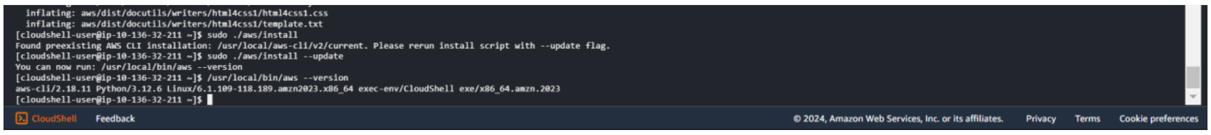
- curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
  unzip awscliv2.zip
- sudo ./aws/install



```
 inflating: aws/dist/docutils/writers/html5_polyglot/responsive.css
 inflating: aws/dist/docutils/writers/html5_polyglot/tuftig.css
 inflating: aws/dist/docutils/writers/html5_polyglot/template.txt
  creating: aws/dist/docutils/writers/s5_html/themes/
  creating: aws/dist/docutils/writers/s5_html/themes/big-black/
  creating: aws/dist/docutils/writers/s5_html/themes/big-white/
  creating: aws/dist/docutils/writers/s5_html/themes/default/
  creating: aws/dist/docutils/writers/s5_html/themes/medium-black/
  creating: aws/dist/docutils/writers/s5_html/themes/medium-white/
  creating: aws/dist/docutils/writers/s5_html/themes/small-black/
  creating: aws/dist/docutils/writers/s5_html/themes/small-white/
 inflating: aws/dist/docutils/writers/s5_html/themes/README.txt
 inflating: aws/dist/docutils/writers/s5_html/themes/small-white/pretty.css
 inflating: aws/dist/docutils/writers/s5_html/themes/small-white/framing.css
 inflating: aws/dist/docutils/writers/s5_html/themes/default/slides.css
 inflating: aws/dist/docutils/writers/s5_html/themes/default/outline.css
 inflating: aws/dist/docutils/writers/s5_html/themes/default/s5-core.css
 inflating: aws/dist/docutils/writers/s5_html/themes/default/opera.css
 inflating: aws/dist/docutils/writers/s5_html/themes/default/pretty.css
 inflating: aws/dist/docutils/writers/s5_html/themes/default/framing.css
 inflating: aws/dist/docutils/writers/s5_html/themes/default/slides.js
 inflating: aws/dist/docutils/writers/s5_html/themes/default/print.css
 inflating: aws/dist/docutils/writers/s5_html/themes/small-black/pretty.css
 inflating: aws/dist/docutils/writers/s5_html/themes/small-black/__base__
 inflating: aws/dist/docutils/writers/s5_html/themes/big-white/framing.css
 inflating: aws/dist/docutils/writers/s5_html/themes/big-white/pretty.css
 inflating: aws/dist/docutils/writers/s5_html/themes/medium-black/pretty.css
 inflating: aws/dist/docutils/writers/s5_html/themes/medium-black/__base__
 inflating: aws/dist/docutils/writers/s5_html/themes/medium-white/framing.css
 inflating: aws/dist/docutils/writers/s5_html/themes/medium-white/pretty.css
 inflating: aws/dist/docutils/writers/s5_html/themes/big-black/__base__
 inflating: aws/dist/docutils/writers/s5_html/themes/big-black/pretty.css
 inflating: aws/dist/docutils/writers/s5_html/themes/big-black/framing.css
 inflating: aws/dist/docutils/writers/pep_html/pep.css
 inflating: aws/dist/docutils/writers/pep_html/template.txt
 inflating: aws/dist/docutils/writers/latex2e/default.tex
 inflating: aws/dist/docutils/writers/latex2e/titlepage.tex
 inflating: aws/dist/docutils/writers/latex2e/xelatex.tex
 inflating: aws/dist/docutils/writers/latex2e/titlingpage.tex
 inflating: aws/dist/docutils/writers/latex2e/docutils.sty
 inflating: aws/dist/docutils/writers/html4css1/html4css1.css
 inflating: aws/dist/docutils/writers/html4css1/template.txt
[cloudshell-user@ip-10-136-32-211 ~]$
```



```
[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/html4css1.css
 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 ~]$
```

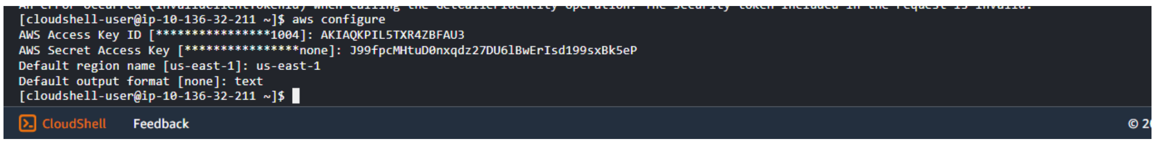
2. Setting Up AWS CLI

After installation, configure the CLI by providing AWS credentials:

- aws configure

enter:

1. AWS Access Key ID
2. AWS Secret Access Key
3. Default region (e.g., us-west-2)
4. Output format (e.g., json)



```
[cloudshell-user@ip-10-136-32-211 ~]$ aws configure
AWS Access Key ID [****************1004]: AKIAQKPIL5TXR4ZBFAU3
AWS Secret Access Key [****************none]: J99fpcMHtuD0nxqdz27DU6lBwErIsd199sxBk5eP
Default region name [us-east-1]: us-east-1
Default output format [none]: text
[cloudshell-user@ip-10-136-32-211 ~]$
```

To Create Access Keys in AWS Management Console:

- Navigate to IAM:

In the search bar, type "IAM" and select IAM (Identity & Access Management) from the services.

- Create a New IAM User:

On the left sidebar, click on Users.
Click on Add users.

- Set User Details:

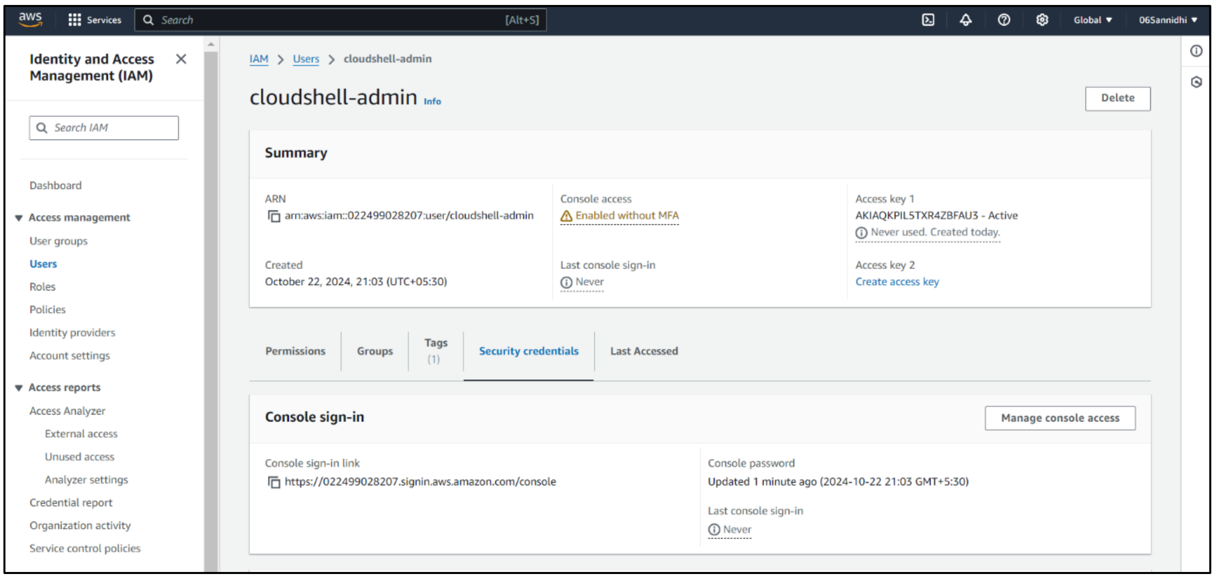
Username: Enter a name for your new user (e.g., Cloudshell-admin).
Access Type: Select Programmatic access (this creates an access key ID and secret access key for the AWS CLI and SDKs).

- Attach Permissions:

On the next screen, choose Attach existing policies directly.
Select AdministratorAccess to give the user full access to AWS services.

- Complete the Setup:

Continue through the steps and create the user.
After the user is created, you will see the Access Key ID and Secret Access Key.
Download these credentials or save them securely.



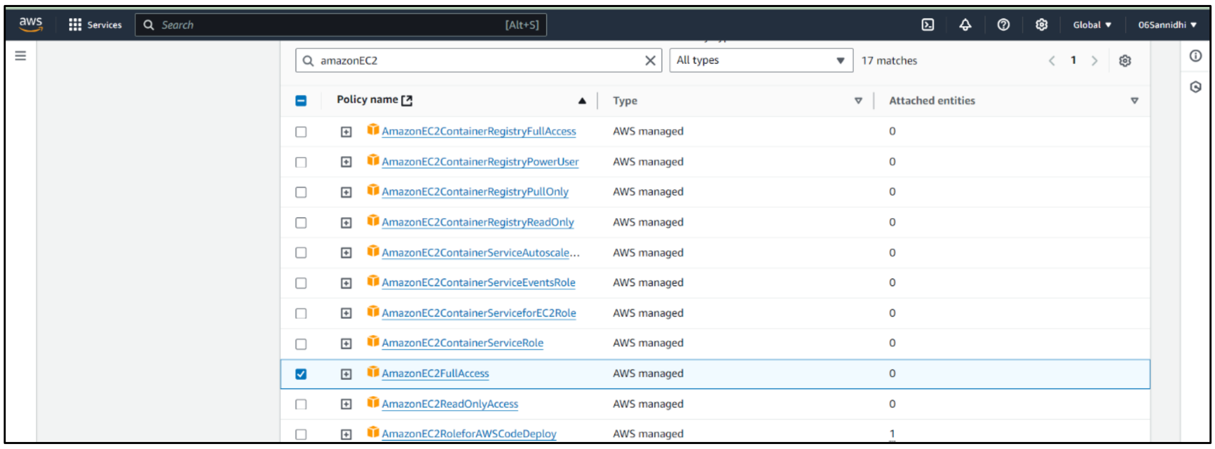
cloudshell-admin Info

Summary

ARN: arn:aws:iam::022499028207:user/cloudshell-admin
Console access: Enabled without MFA
Access key 1: AKIAQKPIL5TXR4ZBFAU3 - Active
Never used. Created today.
Created: October 22, 2024, 21:03 (UTC+05:30)
Last console sign-in: Never
Access key 2: Create access key

Console sign-in

Console sign-in link: https://022499028207.signin.aws.amazon.com/console
Console password: Updated 1 minute ago (2024-10-22 21:03 GMT+5:30)
Last console sign-in: Never



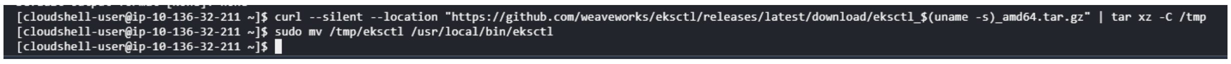
	Policy name	Type	Attached entities
☐	AmazonEC2ContainerRegistryFullAccess	AWS managed	0
☐	AmazonEC2ContainerRegistryPowerUser	AWS managed	0
☐	AmazonEC2ContainerRegistryPullOnly	AWS managed	0
☐	AmazonEC2ContainerRegistryReadOnly	AWS managed	0
☐	AmazonEC2ContainerServiceAutoscale...	AWS managed	0
☐	AmazonEC2ContainerServiceEventsRole	AWS managed	0
☐	AmazonEC2ContainerServiceforEC2Role	AWS managed	0
☐	AmazonEC2ContainerServiceRole	AWS managed	0
✓	AmazonEC2FullAccess	AWS managed	0
☐	AmazonEC2ReadOnlyAccess	AWS managed	0
☐	AmazonEC2RoleforAWSCodeDeploy	AWS managed	1

Step 3: Create an Amazon EKS Cluster

1. Installing eksctl

Install eksctl, the CLI tool used to interact with Amazon EKS:

- curl --silent --location https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
- sudo mv /tmp/eksctl /usr/local/bin/eksctl

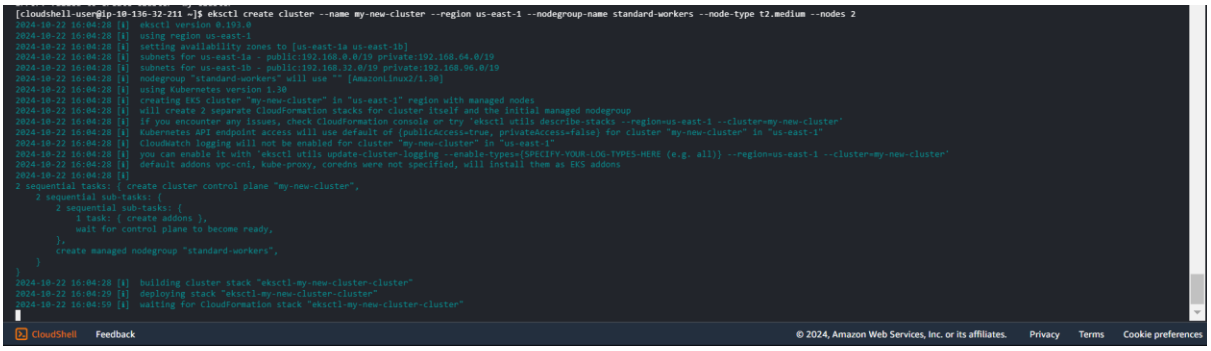


```
[cloudshell-user@ip-10-136-32-211 ~]$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
[cloudshell-user@ip-10-136-32-211 ~]$ sudo mv /tmp/eksctl /usr/local/bin/eksctl
[cloudshell-user@ip-10-136-32-211 ~]$ 
```

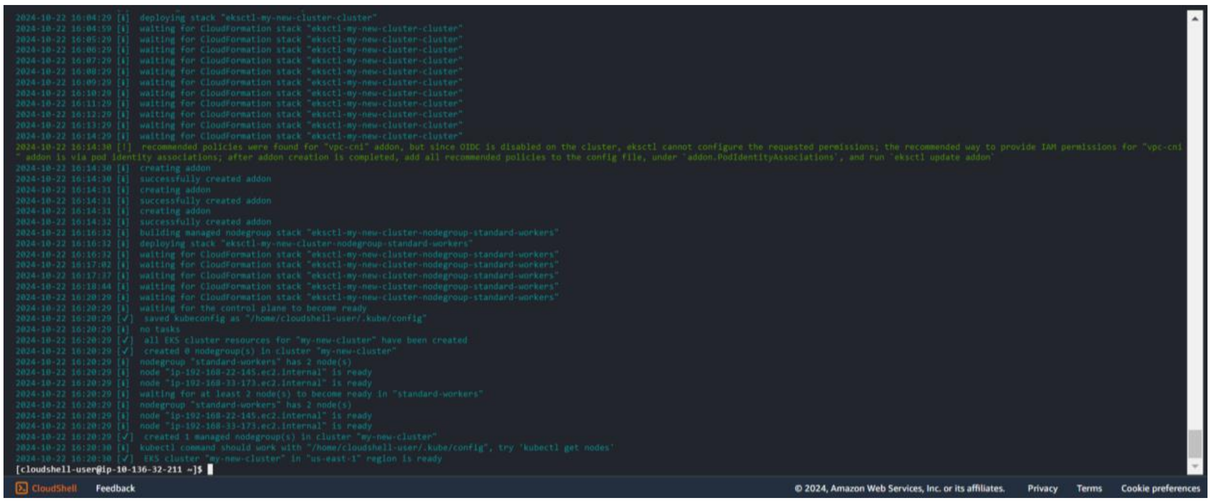
2. Creating a Kubernetes Cluster

Use eksctl to create a cluster with two worker nodes:

- eksctl create cluster --name my-new-cluster --region us-west-2 --nodegroup-name standard-workers --node-type t2.medium --nodes 2



```
[cloudshell-user@ip-10-136-32-211 ~]$ eksctl create cluster --name my-new-cluster --region us-east-1 --nodegroup-name standard-workers --node-type t2.medium --nodes 2
2024-10-22 16:04:28 [ℹ]  eksctl version 0.193.0
2024-10-22 16:04:28 [ℹ]  using region us-east-1
2024-10-22 16:04:28 [ℹ]  setting availability zones to [us-east-1a us-east-1b]
2024-10-22 16:04:28 [ℹ]  subnets for us-east-1a - public:192.168.0.0/19 private:192.168.64.0/19
2024-10-22 16:04:28 [ℹ]  subnets for us-east-1b - public:192.168.32.0/19 private:192.168.96.0/19
2024-10-22 16:04:28 [ℹ]  nodegroup "standard-workers" will use "" [AmazonLinux2/1.30]
2024-10-22 16:04:28 [ℹ]  using Kubernetes version 1.30
2024-10-22 16:04:28 [ℹ]  creating EKS cluster "my-new-cluster" in "us-east-1" region with managed nodes
2024-10-22 16:04:28 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-10-22 16:04:28 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=my-new-cluster'
2024-10-22 16:04:28 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "my-new-cluster" in "us-east-1"
2024-10-22 16:04:28 [ℹ]  CloudWatch logging will not be enabled for cluster "my-new-cluster" in "us-east-1"
2024-10-22 16:04:28 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-1 --cluster=my-new-cluster'
2024-10-22 16:04:28 [ℹ]  default addons vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons
2024-10-22 16:04:28 [ℹ]
2 sequential tasks: { create cluster control plane "my-new-cluster",
    2 sequential sub-tasks: {
        2 sequential sub-tasks: {
            1 task: { create addons },
            wait for control plane to become ready,
        },
        create managed nodegroup "standard-workers",
    }
}
2024-10-22 16:04:28 [ℹ]  building cluster stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:04:29 [ℹ]  deploying stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:04:59 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
```

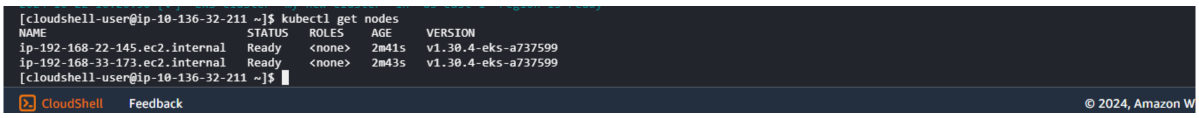


```
2024-10-22 16:04:29 [ℹ]  deploying stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:04:59 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:05:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:06:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:07:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:08:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:09:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:10:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:11:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:12:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:13:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:14:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-cluster"
2024-10-22 16:14:30 [!]  recommended policies were found for "vpc-cni" addon, but since OIDC is disabled on the cluster, eksctl cannot configure the requested permissions; the recommended way to provide IAM permissions for "vpc-cni
" addon is via pod identity associations; after addon creation is completed, add all recommended policies to the config file, under `addon.PodIdentityAssociations`, and run `eksctl update addon`
2024-10-22 16:14:30 [ℹ]  creating addon
2024-10-22 16:14:30 [ℹ]  successfully created addon
2024-10-22 16:14:31 [ℹ]  creating addon
2024-10-22 16:14:31 [ℹ]  successfully created addon
2024-10-22 16:14:31 [ℹ]  creating addon
2024-10-22 16:14:32 [ℹ]  successfully created addon
2024-10-22 16:16:32 [ℹ]  building managed nodegroup stack "eksctl-my-new-cluster-nodegroup-standard-workers"
2024-10-22 16:16:32 [ℹ]  deploying stack "eksctl-my-new-cluster-nodegroup-standard-workers"
2024-10-22 16:16:32 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-nodegroup-standard-workers"
2024-10-22 16:17:02 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-nodegroup-standard-workers"
2024-10-22 16:17:37 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-nodegroup-standard-workers"
2024-10-22 16:18:44 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-nodegroup-standard-workers"
2024-10-22 16:20:29 [ℹ]  waiting for CloudFormation stack "eksctl-my-new-cluster-nodegroup-standard-workers"
2024-10-22 16:20:29 [ℹ]  waiting for the control plane to become ready
2024-10-22 16:20:29 [✓]  saved kubeconfig as "/home/cloudshell-user/.kube/config"
2024-10-22 16:20:29 [ℹ]  no tasks
2024-10-22 16:20:29 [✓]  all EKS cluster resources for "my-new-cluster" have been created
2024-10-22 16:20:29 [✓]  created 0 nodegroup(s) in cluster "my-new-cluster"
2024-10-22 16:20:29 [ℹ]  nodegroup "standard-workers" has 2 node(s)
2024-10-22 16:20:29 [ℹ]  node "ip-192-168-22-145.ec2.internal" is ready
2024-10-22 16:20:29 [ℹ]  node "ip-192-168-33-173.ec2.internal" is ready
2024-10-22 16:20:29 [ℹ]  waiting for at least 2 node(s) to become ready in "standard-workers"
2024-10-22 16:20:29 [ℹ]  nodegroup "standard-workers" has 2 node(s)
2024-10-22 16:20:29 [ℹ]  node "ip-192-168-22-145.ec2.internal" is ready
2024-10-22 16:20:29 [ℹ]  node "ip-192-168-33-173.ec2.internal" is ready
2024-10-22 16:20:29 [✓]  created 1 managed nodegroup(s) in cluster "my-new-cluster"
2024-10-22 16:20:30 [ℹ]  kubectl command should work with "/home/cloudshell-user/.kube/config", try 'kubectl get nodes'
2024-10-22 16:20:30 [✓]  EKS cluster "my-new-cluster" in "us-east-1" region is ready
[cloudshell-user@ip-10-136-32-211 ~]$ 
```

3. Verifying the Cluster

Once the cluster is created, verify that it's up and running by listing the available nodes:

- kubectl get nodes



```
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl get nodes
NAME                             STATUS   ROLES    AGE     VERSION
ip-192-168-22-145.ec2.internal   Ready    <none>   2m41s   v1.30.4-eks-a737599
ip-192-168-33-173.ec2.internal   Ready    <none>   2m43s   v1.30.4-eks-a737599
[cloudshell-user@ip-10-136-32-211 ~]$ 
```

Step 4: Deploying Nginx Application

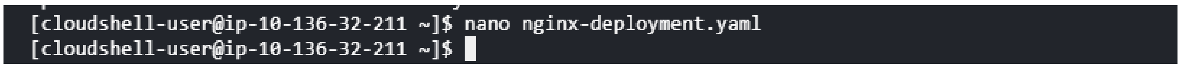
1. Creating a Deployment YAML

Create a YAML file that defines the Nginx deployment. In CloudShell, use a text editor such as nano to create the file:

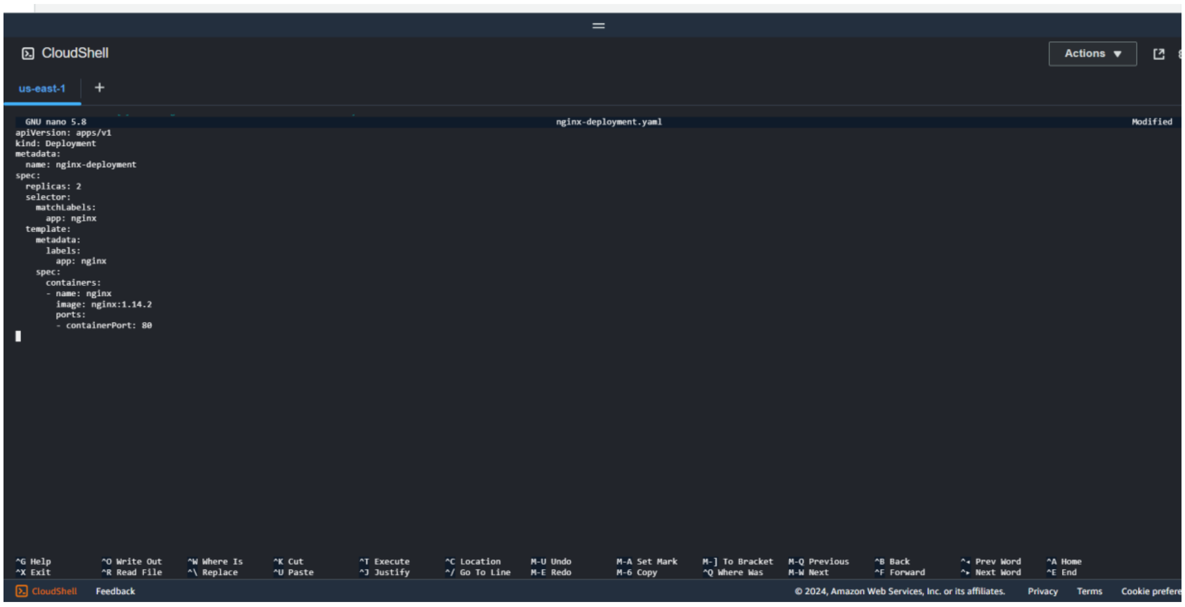
- nano nginx-deployment.yaml
- Paste the following content into the file:

yaml
Copy code

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```



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



```
  GNU nano 5.8                                  nginx-deployment.yaml                                  Modified
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80


^G Help     ^O Write Out  ^W Where Is  ^K Cut    ^T Execute  ^C Location    M-U Undo  M-A Set Mark  M-] To Bracket  M-Q Previous  ^B Back     ^← Prev Word  ^A Home
^X Exit     ^R Read File  ^\ Replace   ^U Paste  ^J Justify  ^/ Go To Line  M-E Redo  M-6 Copy      ^Q Where Was    M-W Next      ^F Forward  ^→ Next Word  ^E End
```

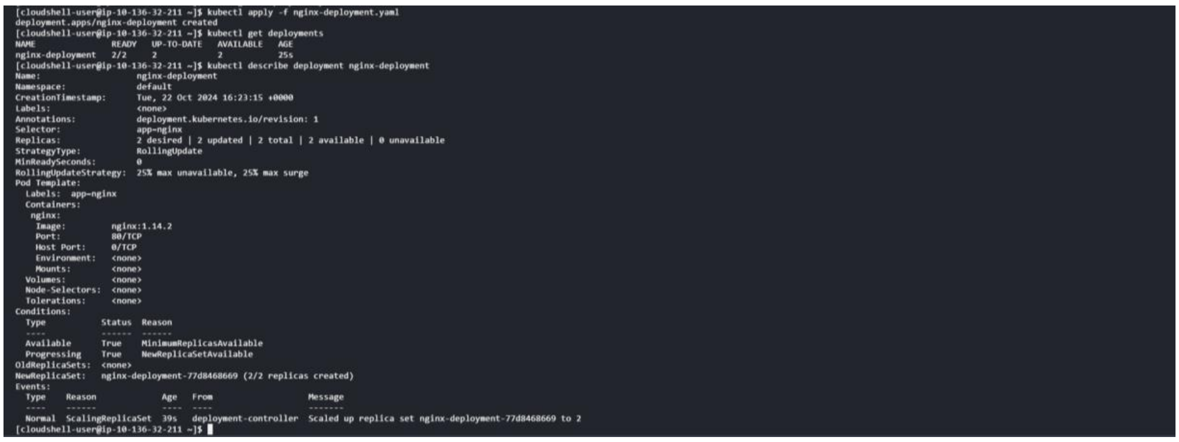
2. **Deploying the Application**

   Apply the configuration to deploy Nginx:
   - kubectl apply -f nginx-deployment.yaml

3. **Verifying the Deployment**

   Ensure the Nginx deployment was created successfully:
   - kubectl get deployments

   Use the describe command to view details:
   - kubectl describe deployment nginx-deployment



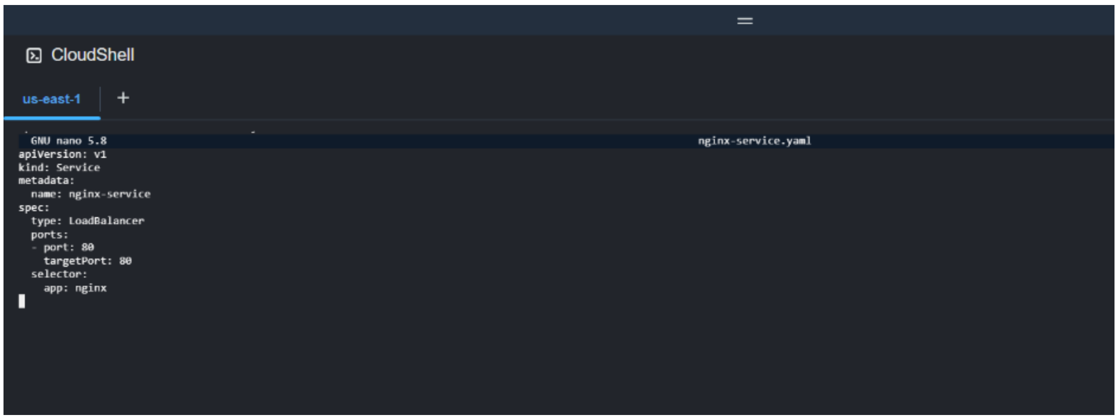
```
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           25s
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Tue, 22 Oct 2024 16:23:15 +0000
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.14.2
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
  Node-Selectors: <none>
  Tolerations:    <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-77d8468669 (2/2 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  39s   deployment-controller  Scaled up replica set nginx-deployment-77d8468669 to 2
[cloudshell-user@ip-10-136-32-211 ~]$
```

Step 5: Exposing the Nginx Application Using a LoadBalancer

1. **Creating a Service YAML**

   Now create another YAML file to expose the Nginx deployment via a LoadBalancer service:
   - nano nginx-service.yaml
   - Paste this configuration:

```
yaml
Copy code
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
```



```
  GNU nano 5.8                                                                     nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
```

2. **Applying the Service**

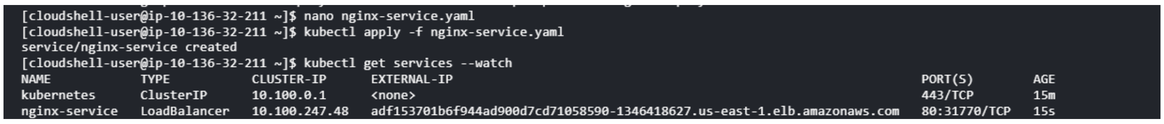
   Apply the LoadBalancer service:

- kubectl apply -f nginx-service.yaml

3. **Retrieving the External IP**

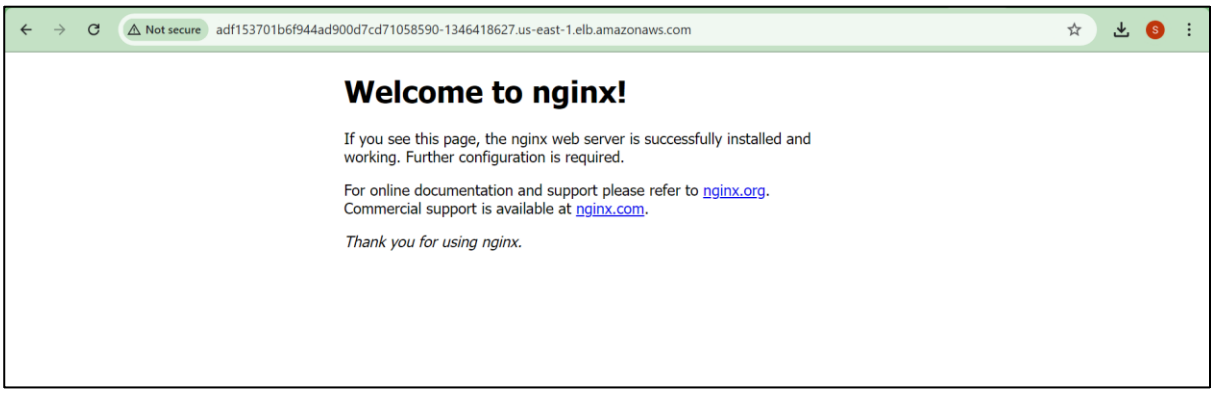
   To access the application, retrieve the external IP assigned to the LoadBalancer:

- kubectl get services –watch



```
[cloudshell-user@ip-10-136-32-211 ~]$ nano nginx-service.yaml
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl apply -f nginx-service.yaml
service/nginx-service created
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl get services --watch
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
kubernetes      ClusterIP      10.100.0.1      <none>                                                                     443/TCP        15m
nginx-service   LoadBalancer   10.100.247.48   adf153701b6f944ad900d7cd71058590-1346418627.us-east-1.elb.amazonaws.com   80:31770/TCP   15s
```

It may take a few minutes for the IP address to become available. Once assigned, access the Nginx application by entering the external IP into your web browser.


Not secure
adf153701b6f944ad900d7cd71058590-1346418627.us-east-1.elb.amazonaws.com

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Conclusion

Through this experiment, we successfully set up a Kubernetes cluster on AWS and deployed a simple Nginx application. The application was exposed to the internet using a LoadBalancer, allowing us to access it via a web browser. This experiment demonstrated the practical use of Kubernetes in managing containerized applications and how managed services like Amazon EKS simplify Kubernetes cluster management.

Challenges Faced

- While creating the Kubernetes cluster, one of the difficulties encountered was configuring the eksctl tool, as it requires appropriate permissions and a well-defined IAM role in AWS.

- The process of obtaining the external IP address for the LoadBalancer took more time than expected due to AWS's dynamic provisioning of resources.