Name: Pranav Pol Class:D15A Roll No.: 42

# Advance DevOps Practical Examination Case Study

## 1. 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.

### Key Feature and Application:

The key feature of this case study is using Kubernetes on AWS to manage scalable, fault-tolerant applications. Key aspects include:

- Auto-scaling: Kubernetes automatically adjusts resources based on demand, ensuring your application can handle increased traffic.
- Load balancing: It evenly distributes traffic across instances, preventing bottlenecks.
- Self-healing: Kubernetes detects failures and restarts containers or reassigns workloads to healthy nodes.
- Seamless updates: Rolling updates enable new versions of applications to be deployed without causing downtime.
- This case study is essential for anyone looking to understand Kubernetes in cloud-native environments, demonstrating how AWS and Kubernetes simplify deployment and management of applications at scale.

### Third-Year Project Integration :

For **Appointment Management System**, Kubernetes can be a powerful addition to improve scalability and reliability. Here's how it fits:

- **Auto-scaling:** Your system can handle more customers as salons grow. Kubernetes ensures that additional containers are automatically spun up as traffic increases, preventing crashes during peak usage times.
- **High availability:** Kubernetes’ self-healing ensures that if a pod (container) fails, another one takes over automatically, allowing continuous service availability.

- **Load balancing:** By using Kubernetes' LoadBalancer service, traffic is evenly distributed among your application's instances, improving response times and ensuring that no single server gets overwhelmed.
- **Rolling updates:** You can introduce new features (e.g., enhanced booking system or payment integrations) without downtime, keeping salon operations uninterrupted.

By integrating Kubernetes and EKS, your **Appointment Management System** becomes a robust, scalable solution capable of handling real-world salon operations, improving customer experience, and ensuring high uptime under load. This integration will enhance the reliability and efficiency of your project, preparing it for deployment in a production environment.

## 2. Implementation

### Step 1: Install and Configure `kubectl` using AWS Cloudshell

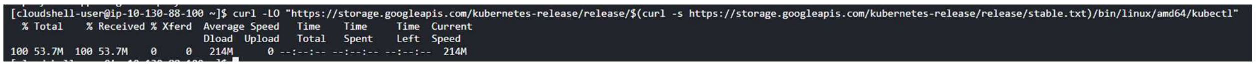
#### 1.1. Access AWS Cloudshell

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

#### 1.2. Download `kubectl`

In the AWS Cloudshell terminal, download the `kubectl` binary:

```
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-10-130-88-100 ~]$ 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   214M      0 --:--:-- --:--:-- --:--:--  214M
```

#### 1.3. Make `kubectl` Executable

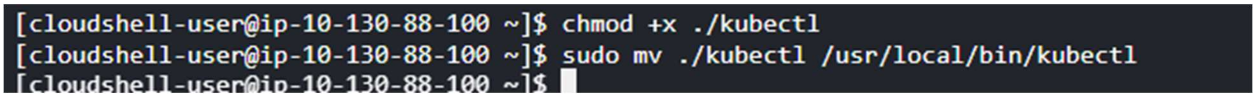
Set execute permissions for the binary:

```
chmod +x ./kubectl
```

### 1.4. Move `kubectl` to `/usr/local/bin`

Move the `kubectl` binary to the system path for easier access:

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

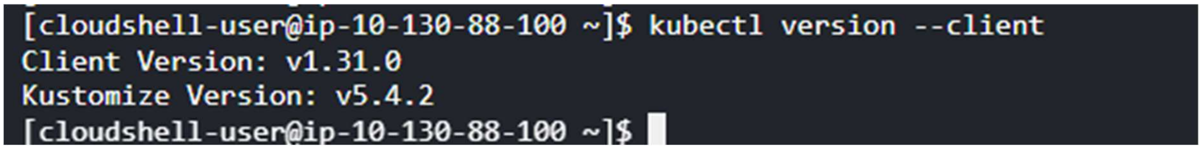


```
[cloudshell-user@ip-10-130-88-100 ~]$ chmod +x ./kubectl
[cloudshell-user@ip-10-130-88-100 ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
[cloudshell-user@ip-10-130-88-100 ~]$
```

### 1.5. Verify Installation

Verify that `kubectl` has been installed successfully:

```
kubectl version --client
```



```
[cloudshell-user@ip-10-130-88-100 ~]$ kubectl version --client
Client Version: v1.31.0
Kustomize Version: v5.4.2
[cloudshell-user@ip-10-130-88-100 ~]$
```

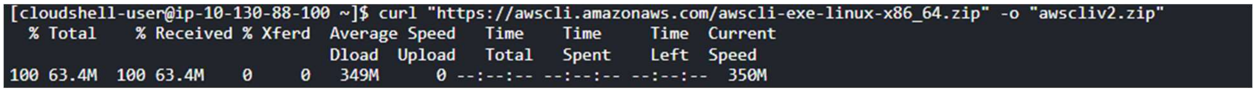
You should see output confirming the `kubectl` client version.

## Step 2: Configure AWS CLI

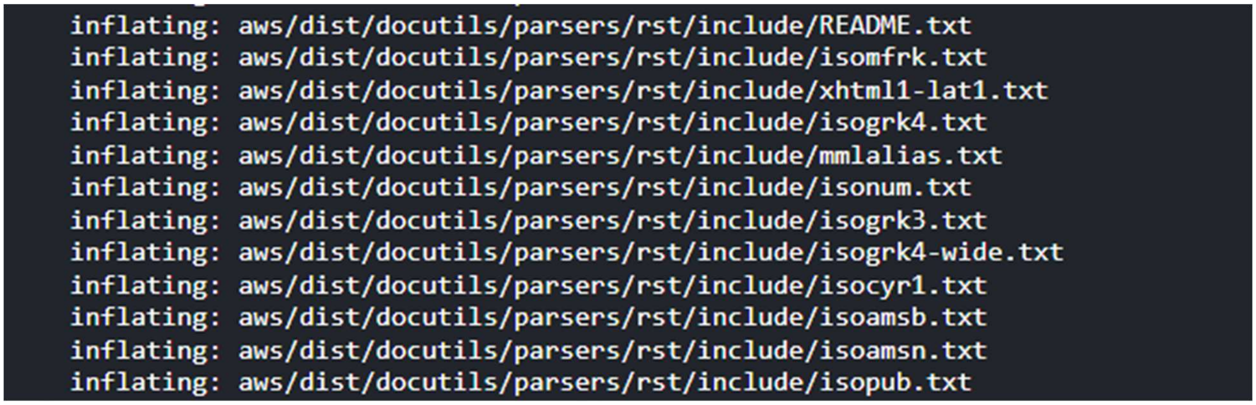
### 2.1. Install AWS CLI

To install the AWS CLI in AWS Cloudshell, run the following commands:

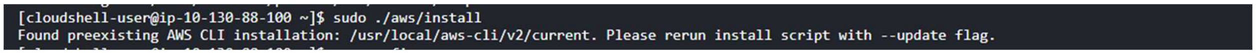
```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```



```
[cloudshell-user@ip-10-130-88-100 ~]$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 63.4M  100 63.4M    0     0   349M      0 --:--:-- --:--:-- --:--:--  350M
```



```
    inflating: aws/dist/docutils/parsers/rst/include/README.txt
    inflating: aws/dist/docutils/parsers/rst/include/isomfrk.txt
    inflating: aws/dist/docutils/parsers/rst/include/xhtml1-lat1.txt
    inflating: aws/dist/docutils/parsers/rst/include/isogrk4.txt
    inflating: aws/dist/docutils/parsers/rst/include/mmlalias.txt
    inflating: aws/dist/docutils/parsers/rst/include/isonum.txt
    inflating: aws/dist/docutils/parsers/rst/include/isogrk3.txt
    inflating: aws/dist/docutils/parsers/rst/include/isogrk4-wide.txt
    inflating: aws/dist/docutils/parsers/rst/include/isocyr1.txt
    inflating: aws/dist/docutils/parsers/rst/include/isoamsb.txt
    inflating: aws/dist/docutils/parsers/rst/include/isoamsn.txt
    inflating: aws/dist/docutils/parsers/rst/include/isopub.txt
```



```
[cloudshell-user@ip-10-130-88-100 ~]$ sudo ./aws/install
Found preexisting AWS CLI installation: /usr/local/aws-cli/v2/current. Please rerun install script with --update flag.
```

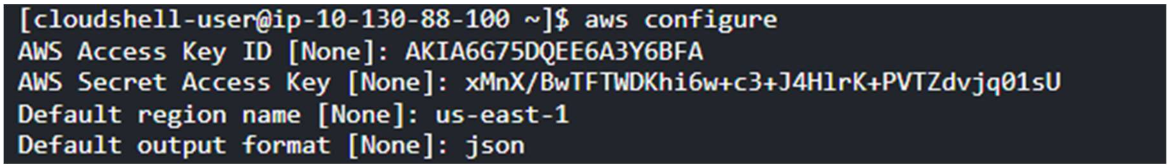
## 2.2. Configure AWS CLI

Once installed, configure the AWS CLI by entering your credentials:

```
aws configure
```

You will be prompted for the following details:

- **AWS Access Key ID**: Enter your access key ID.
- **AWS Secret Access Key**: Enter your secret access key.
- **Default region name**: Enter your preferred region (e.g., `us-east-1`).
- **Default output format**: Enter your preferred output format (e.g., `json`).



```
[cloudshell-user@ip-10-130-88-100 ~]$ aws configure
AWS Access Key ID [None]: AKIA6G75DQEE6A3Y6BFA
AWS Secret Access Key [None]: xMnX/BwTFTWDKhi6w+c3+J4HlrK+PVTZdvjq01sU
Default region name [None]: us-east-1
Default output format [None]: json
```

## Step-by-step guide for creating an **IAM User** with **Programmatic Access** in AWS and obtaining access key

### Step a: Log in to AWS Management Console

- Open the AWS Management Console.
- Sign in with your account credentials.

### Step b: Navigate to IAM Service

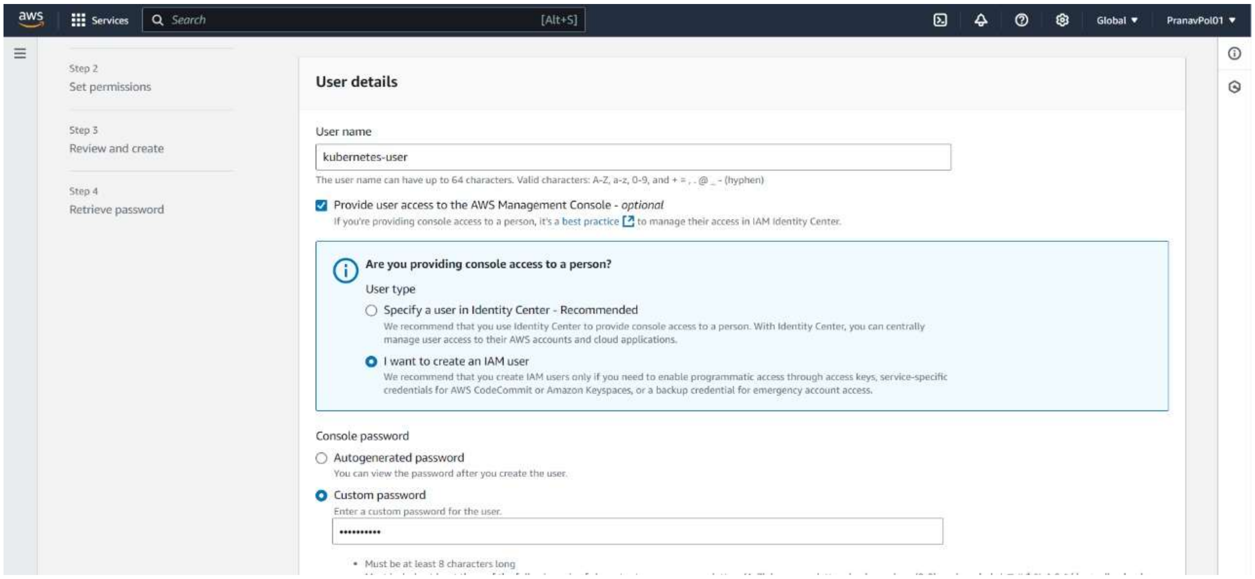
- In the **Services** menu, search for and select **IAM (Identity and Access Management)**.

### Step c: Create a New IAM User

1. In the IAM dashboard, click **Users** from the left-hand sidebar.
2. Click on the **Add user** button.

### Step d: Configure User Details

1. **User Name**: Enter a name for the new user (e.g., `kubernetes-user`).
2. **Access type**:
   - ○ Check the box for **Programmatic access** (this will generate an access key ID and secret access key for API, CLI, and SDK use).



Step 2
Set permissions

Step 3
Review and create

Step 4
Retrieve password

User details

User name

kubernetes-user

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☑ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

Are you providing console access to a person?

User type

○ Specify a user in Identity Center - Recommended
We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

● I want to create an IAM user
We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Console password

○ Autogenerated password
You can view the password after you create the user.

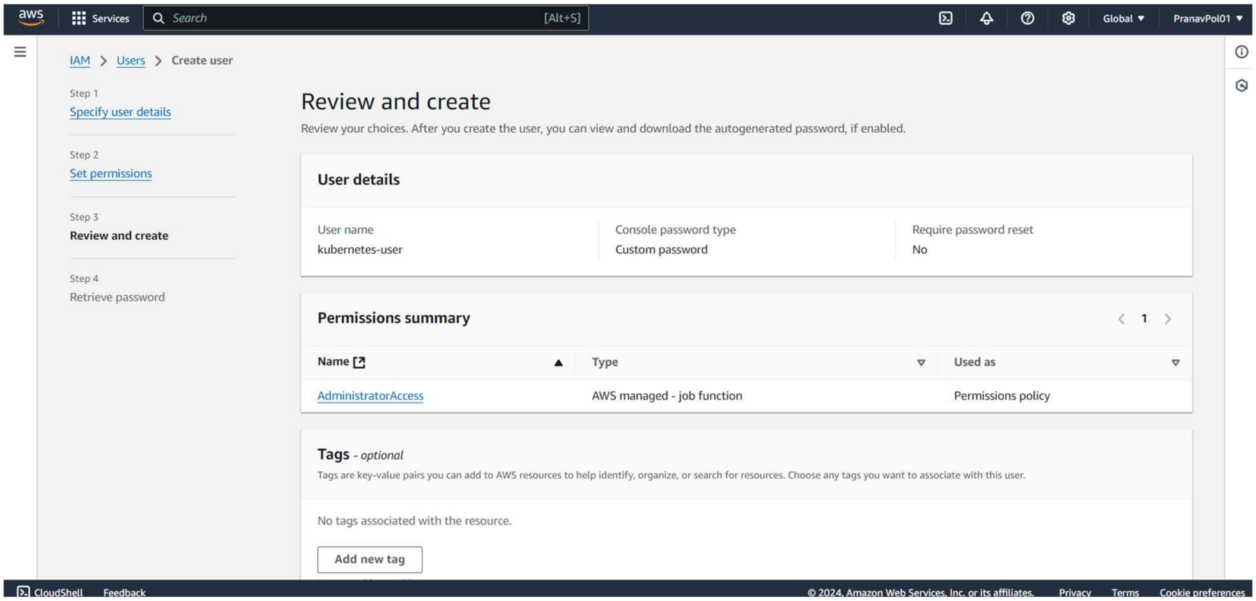
● Custom password
Enter a custom password for the user.

••••••••••

- Must be at least 8 characters long

### Step e: Set Permissions

1. Click **Next: Permissions**.
2. Choose how to set permissions for the user:
   - ○ **Attach policies directly**: If you want to assign predefined AWS policies, choose this option.
     - ■ Search for and select the **AdministratorAccess** policy for full access permissions.
     - ■ Alternatively, you can choose more restrictive policies if needed, such as `AmazonEKSFullAccess` for Kubernetes-related permissions.



Review and create

Review your choices. After you create the user, you can view and download the autogenerated password, if enabled.

User details

| User name | Console password type | Require password reset |
|---|---|---|
| kubernetes-user | Custom password | No |

Permissions summary

| Name | Type | Used as |
|---|---|---|
| AdministratorAccess | AWS managed - job function | Permissions policy |

Tags - *optional*

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

### Step f: Review and Create the User

1. Review the user details and the permissions you've assigned.
2. Once verified, click **Create user**.

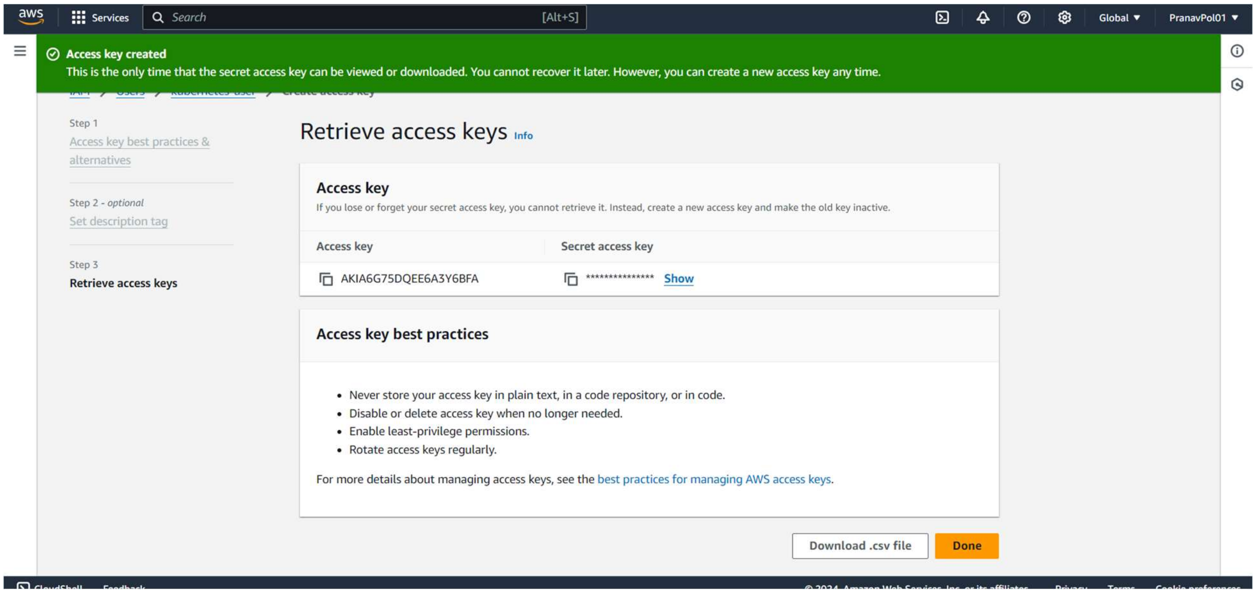
### Step g: Get Access Keys

Once the user is created, the **Access Key ID** and **Secret Access Key** for the user will be displayed.

- **Important**:
  - This is the only time you'll be able to see the **Secret Access Key**, so make sure to download the credentials as a `.csv` file or copy them to a secure location.
  - You can click **Download .csv** to save the access key information.

The access keys (Access Key ID and Secret Access Key) will be used for programmatic access, such as configuring the AWS CLI (`aws configure`) or using the SDK.

Pranav Pol D15A 42



Access key created

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

Step 1
Access key best practices & alternatives

Step 2 - *optional*
Set description tag

Step 3
Retrieve access keys

## Retrieve access keys Info

### Access key

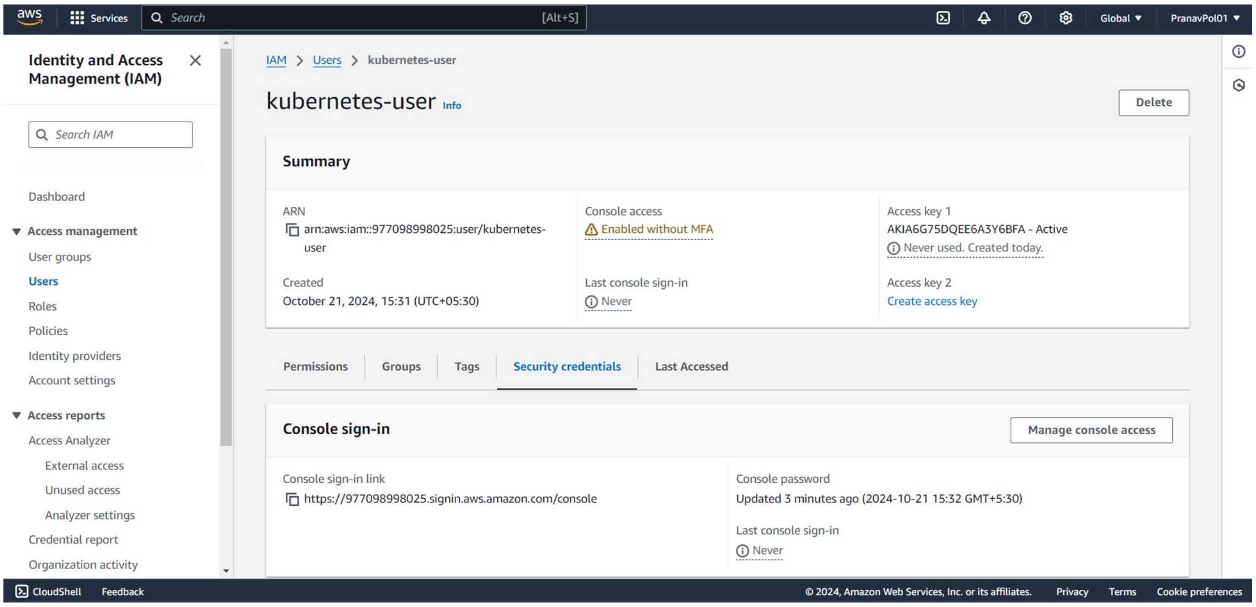
If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

| Access key | Secret access key |
|---|---|
| AKIA6G75DQEE6A3Y6BFA | *************** Show |

### Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the best practices for managing AWS access keys.



## kubernetes-user Info

### Summary

| ARN | Console access | Access key 1 |
|---|---|---|
| arn:aws:iam::977098998025:user/kubernetes-user | Enabled without MFA | AKIA6G75DQEE6A3Y6BFA - Active<br>Never used. Created today. |
| Created | Last console sign-in | Access key 2 |
| October 21, 2024, 15:31 (UTC+05:30) | Never | Create access key |

### Console sign-in

Console sign-in link
https://977098998025.signin.aws.amazon.com/console

Console password
Updated 3 minutes ago (2024-10-21 15:32 GMT+5:30)

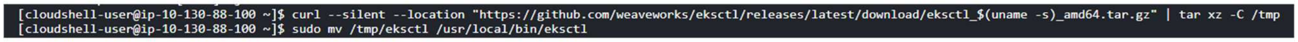
Last console sign-in
Never

## Step 3: Create a Kubernetes Cluster on AWS

### 3.1. Install eksctl

Install the `eksctl` command-line tool to manage Kubernetes clusters on AWS:

```
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
```



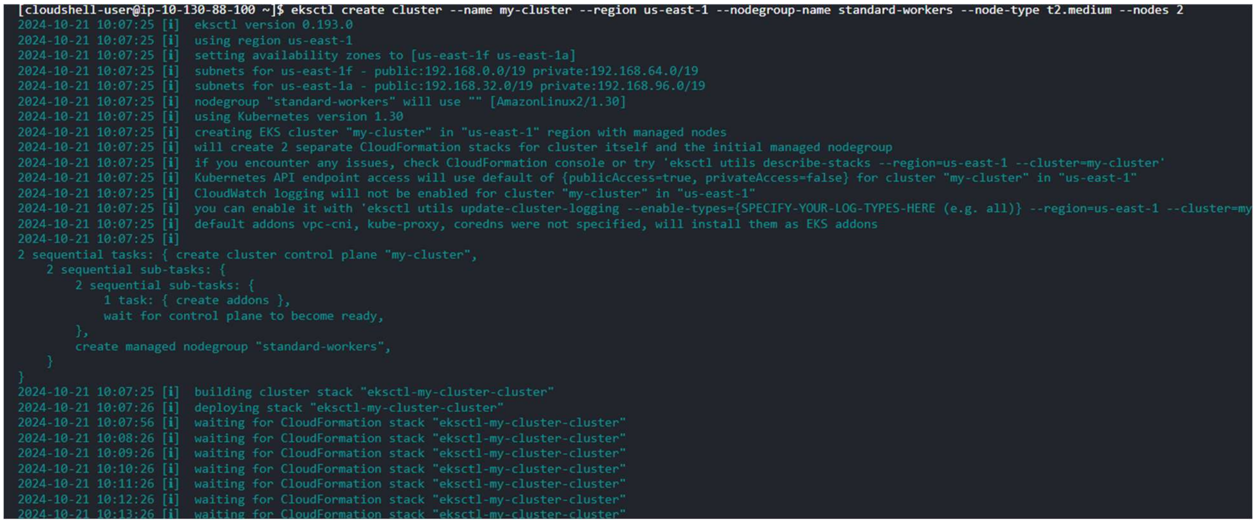
```
[cloudshell-user@ip-10-130-88-100 ~]$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
[cloudshell-user@ip-10-130-88-100 ~]$ sudo mv /tmp/eksctl /usr/local/bin/eksctl
```

### 3.2. Create a Kubernetes Cluster

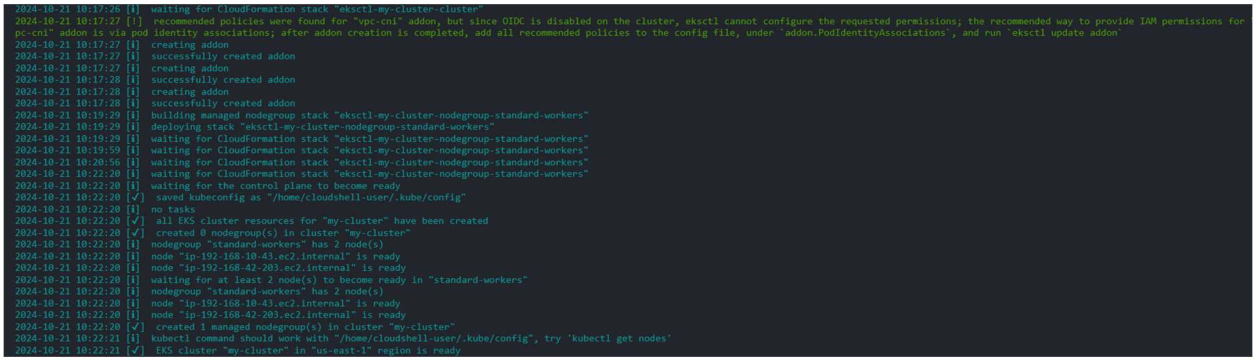
Use `eksctl` to create an Amazon EKS cluster with two worker nodes:

```
eksctl create cluster --name my-cluster --region us-east-1 --
nodegroup-name standard-workers --node-type t2.medium --nodes 2
```

The cluster creation may take several minutes.



```
[cloudshell-user@ip-10-130-88-100 ~]$ eksctl create cluster --name my-cluster --region us-east-1 --nodegroup-name standard-workers --node-type t2.medium --nodes 2
2024-10-21 10:07:25 [i]  eksctl version 0.193.0
2024-10-21 10:07:25 [i]  using region us-east-1
2024-10-21 10:07:25 [i]  setting availability zones to [us-east-1f us-east-1a]
2024-10-21 10:07:25 [i]  subnets for us-east-1f - public:192.168.0.0/19 private:192.168.64.0/19
2024-10-21 10:07:25 [i]  subnets for us-east-1a - public:192.168.32.0/19 private:192.168.96.0/19
2024-10-21 10:07:25 [i]  nodegroup "standard-workers" will use "" [AmazonLinux2/1.30]
2024-10-21 10:07:25 [i]  using Kubernetes version 1.30
2024-10-21 10:07:25 [i]  creating EKS cluster "my-cluster" in "us-east-1" region with managed nodes
2024-10-21 10:07:25 [i]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-10-21 10:07:25 [i]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=my-cluster'
2024-10-21 10:07:25 [i]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "my-cluster" in "us-east-1"
2024-10-21 10:07:25 [i]  CloudWatch logging will not be enabled for cluster "my-cluster" in "us-east-1"
2024-10-21 10:07:25 [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
2024-10-21 10:07:25 [i]  default addons vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons
2024-10-21 10:07:25 [i]
2 sequential tasks: { create cluster control plane "my-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",
    }
}
2024-10-21 10:07:25 [i]  building cluster stack "eksctl-my-cluster-cluster"
2024-10-21 10:07:26 [i]  deploying stack "eksctl-my-cluster-cluster"
2024-10-21 10:07:56 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
2024-10-21 10:08:26 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
2024-10-21 10:09:26 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
2024-10-21 10:10:26 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
2024-10-21 10:11:26 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
2024-10-21 10:12:26 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
2024-10-21 10:13:26 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
```



```
2024-10-21 10:17:26 [i]  waiting for CloudFormation stack "eksctl-my-cluster-cluster"
2024-10-21 10:17:27 [!]  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 "
pc-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`
2024-10-21 10:17:27 [i]  creating addon
2024-10-21 10:17:27 [i]  successfully created addon
2024-10-21 10:17:27 [i]  creating addon
2024-10-21 10:17:28 [i]  successfully created addon
2024-10-21 10:17:28 [i]  creating addon
2024-10-21 10:17:28 [i]  successfully created addon
2024-10-21 10:19:29 [i]  building managed nodegroup stack "eksctl-my-cluster-nodegroup-standard-workers"
2024-10-21 10:19:29 [i]  deploying stack "eksctl-my-cluster-nodegroup-standard-workers"
2024-10-21 10:19:29 [i]  waiting for CloudFormation stack "eksctl-my-cluster-nodegroup-standard-workers"
2024-10-21 10:19:59 [i]  waiting for CloudFormation stack "eksctl-my-cluster-nodegroup-standard-workers"
2024-10-21 10:20:56 [i]  waiting for CloudFormation stack "eksctl-my-cluster-nodegroup-standard-workers"
2024-10-21 10:22:20 [i]  waiting for CloudFormation stack "eksctl-my-cluster-nodegroup-standard-workers"
2024-10-21 10:22:20 [i]  waiting for the control plane to become ready
2024-10-21 10:22:20 [✓]  saved kubeconfig as "/home/cloudshell-user/.kube/config"
2024-10-21 10:22:20 [i]  no tasks
2024-10-21 10:22:20 [✓]  all EKS cluster resources for "my-cluster" have been created
2024-10-21 10:22:20 [✓]  created 0 nodegroup(s) in cluster "my-cluster"
2024-10-21 10:22:20 [i]  nodegroup "standard-workers" has 2 node(s)
2024-10-21 10:22:20 [i]  node "ip-192-168-10-43.ec2.internal" is ready
2024-10-21 10:22:20 [i]  node "ip-192-168-42-203.ec2.internal" is ready
2024-10-21 10:22:20 [i]  waiting for at least 2 node(s) to become ready in "standard-workers"
2024-10-21 10:22:20 [i]  nodegroup "standard-workers" has 2 node(s)
2024-10-21 10:22:20 [i]  node "ip-192-168-10-43.ec2.internal" is ready
2024-10-21 10:22:20 [i]  node "ip-192-168-42-203.ec2.internal" is ready
2024-10-21 10:22:20 [✓]  created 1 managed nodegroup(s) in cluster "my-cluster"
2024-10-21 10:22:21 [i]  kubectl command should work with "/home/cloudshell-user/.kube/config", try 'kubectl get nodes'
2024-10-21 10:22:21 [✓]  EKS cluster "my-cluster" in "us-east-1" region is ready
```

## Step 4: Deploy a Sample Application (Nginx Server)

### 4.1. Create Deployment YAML

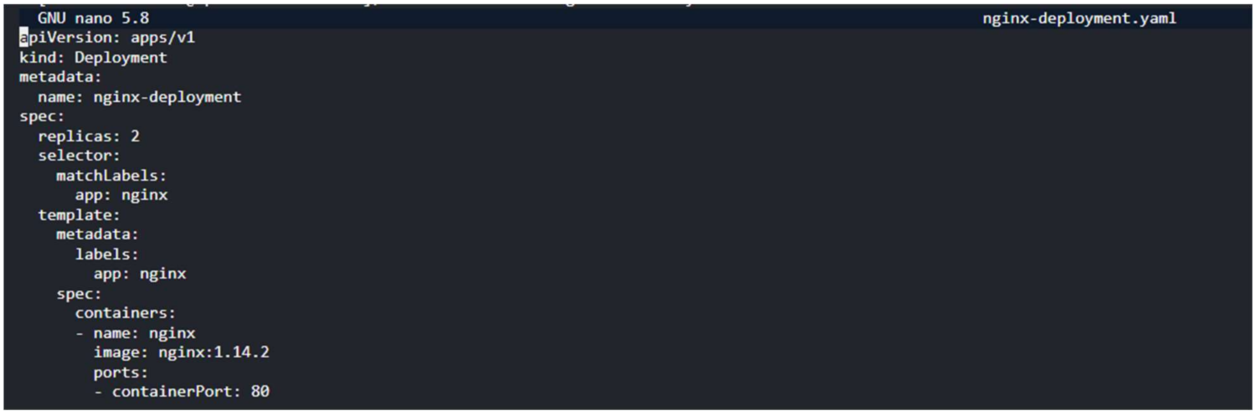
Create a YAML file to define your Nginx deployment. Open Cloudshell and use a text editor to create the file (e.g., nano):

```
nano nginx-deployment.yaml
```

```
[cloudshell-user@ip-10-130-88-100 ~]$ nano nginx-deployment.yaml
```

Paste the following content into the file:

```
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
```

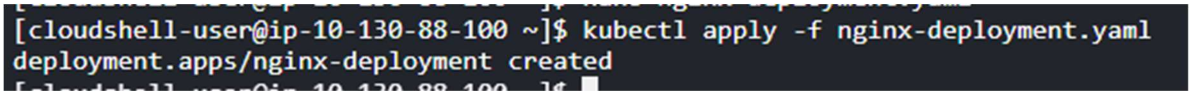


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        -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

### 4.2. Deploy the Application

Apply the deployment using `kubectl`:

```
kubectl apply -f nginx-deployment.yaml
```

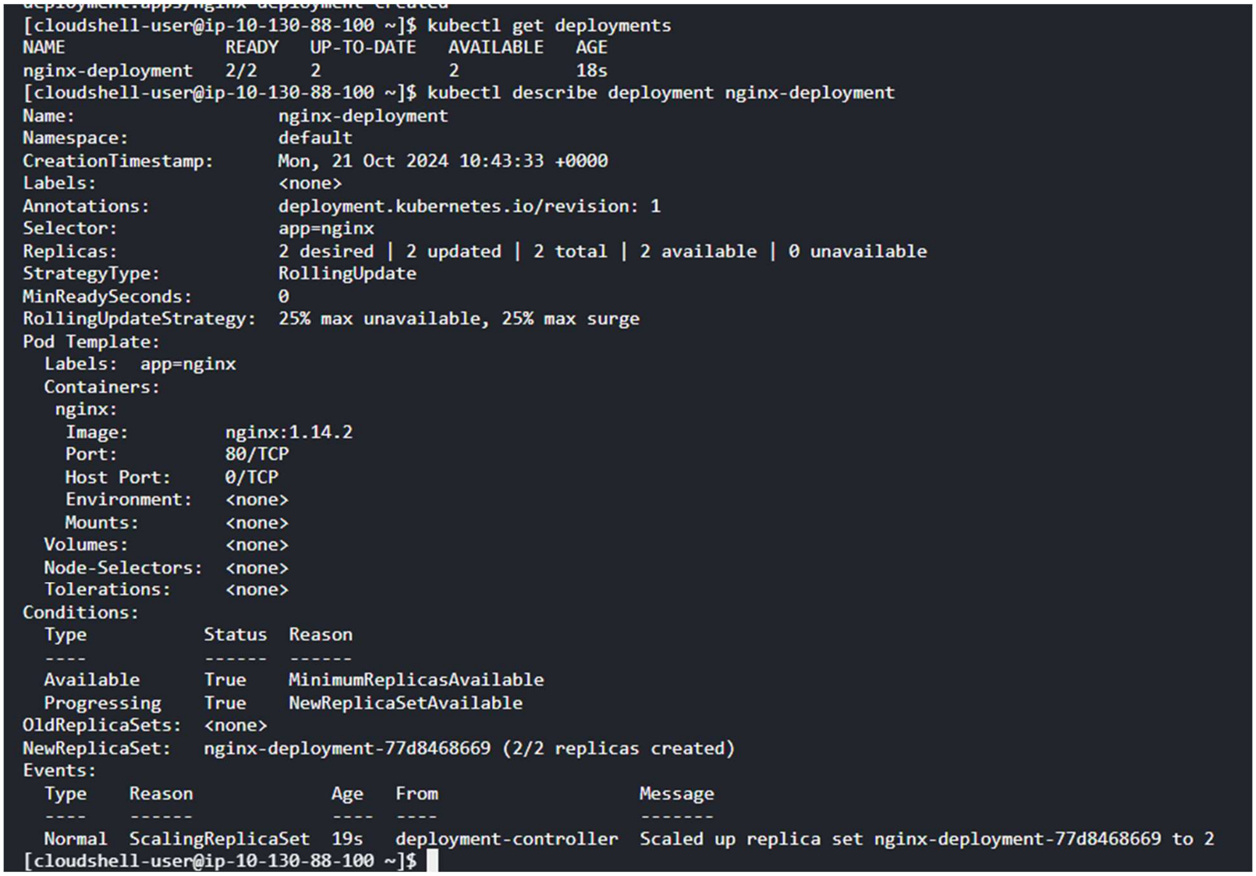


```
[cloudshell-user@ip-10-130-88-100 ~]$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created
[cloudshell-user@ip-10-130-88-100 ~]$
```

### 4.3. Verify Deployment

To verify the deployment, run:

```
kubectl get deployments
kubectl describe deployment nginx-deployment
```



```
deployment.apps/nginx-deployment created
[cloudshell-user@ip-10-130-88-100 ~]$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           18s
[cloudshell-user@ip-10-130-88-100 ~]$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Mon, 21 Oct 2024 10:43:33 +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  19s   deployment-controller  Scaled up replica set nginx-deployment-77d8468669 to 2
[cloudshell-user@ip-10-130-88-100 ~]$
```

## Step 5: Expose the Application Using a LoadBalancer

### 5.1. Create Service YAML

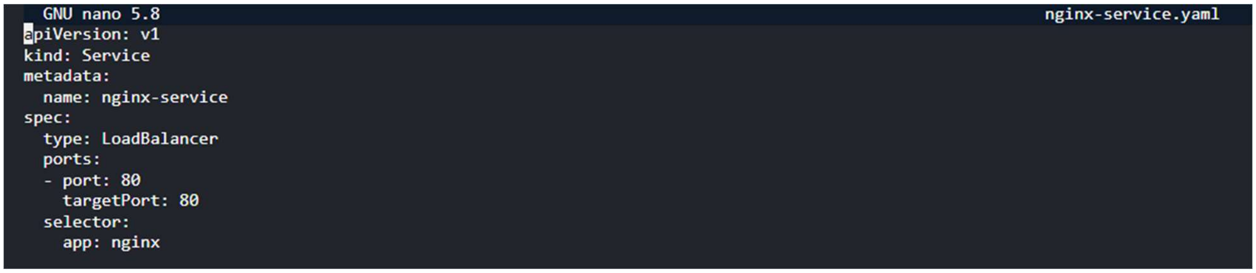
Create a YAML file for exposing the Nginx application as a LoadBalancer service:

```
nano nginx-service.yaml
```

Paste the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: LoadBalancer
  ports:
  - port: 80
```

```
    targetPort: 80
  selector:
    app: nginx
```

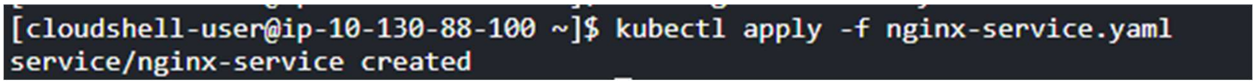


```
  GNU nano 5.8                                                                                   nginx-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
```

### 5.2. Apply the Service

Apply the service configuration to expose your application

```
kubectl apply -f nginx-service.yaml
```

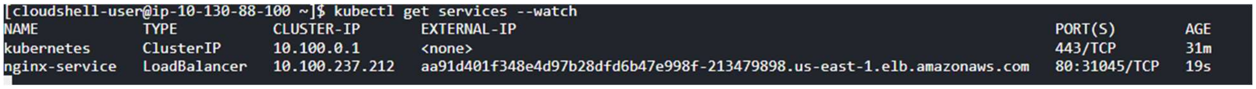


```
[cloudshell-user@ip-10-130-88-100 ~]$ kubectl apply -f nginx-service.yaml
service/nginx-service created
```

### 5.3. Get the External IP

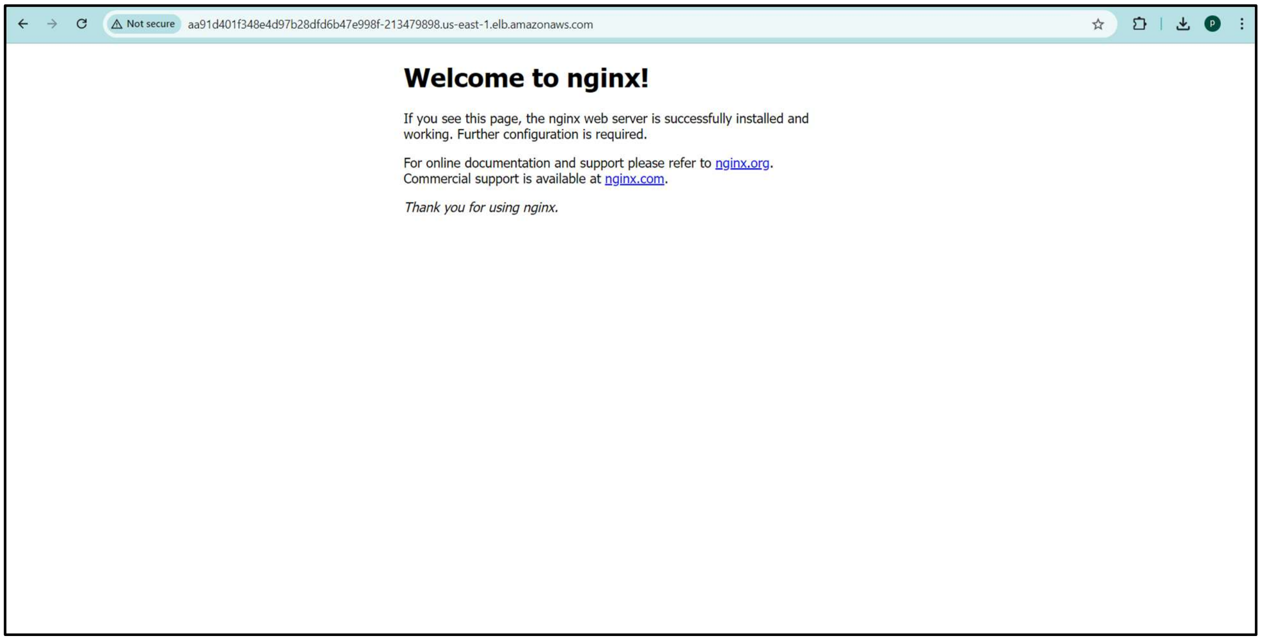
Use the following command to retrieve the external IP address assigned by the load balancer:

```
kubectl get services --watch
```



```
[cloudshell-user@ip-10-130-88-100 ~]$ kubectl get services --watch
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PORT(S)        AGE
kubernetes      ClusterIP      10.100.0.1       <none>                                                                    443/TCP        31m
nginx-service   LoadBalancer   10.100.237.212   aa91d401f348e4d97b28dfd6b47e998f-213479898.us-east-1.elb.amazonaws.com    80:31045/TCP   19s
```

The external IP may take a few minutes to appear. Once it does, you can access your Nginx application using the IP address in your web browser.



aa91d401f348e4d97b28dfd6b47e998f-213479898.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.*

## 3. Conclusion

This documentation outlines how to use AWS and Kubernetes to deploy a **Nginx Server** and **Appointment Management System**. By containerizing the application, deploying it on Amazon EKS, and exposing it with a LoadBalancer, the project becomes scalable, resilient, and ready for production-level traffic.