

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:

```
[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
           % Done   0     0     0    0      0      0      0      0
100  10.7MB 100 10.7MB    0     0  10.7MB/s    0     0      0      0
```

3.Make kubectl Executable

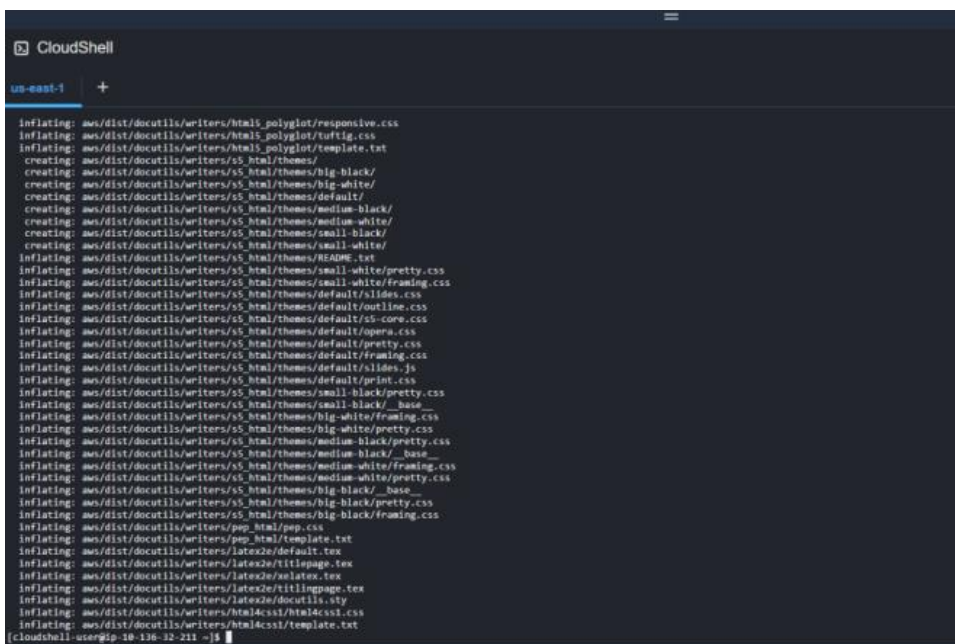
```
[cloudshell-user@ip-10-136-32-211 ~]$ chmod +x ./kubectl
[cloudshell-user@ip-10-136-32-211 ~]$ sudo mv ./kubectl /usr/local/bin/kubectl
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl version --client
Client Version: v1.31.0
Kustomize Version: v5.4.2
[cloudshell-user@ip-10-136-32-211 ~]$
```

Step 2: 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
us-east-1 +
Inflating: aws/dist/docutils/writers/html5_polyglot/responsive.css
Inflating: aws/dist/docutils/writers/html5_polyglot/tufte.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/_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/ppp_html/ppp.css
Inflating: aws/dist/docutils/writers/ppp_html/template.txt
Inflating: aws/dist/docutils/writers/latex2e/default.tex
Inflating: aws/dist/docutils/writers/latex2e/titlingpage.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

```
[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 ~]$
```

3.To Create Access Keys in AWS Management Console:

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

The screenshot shows the AWS IAM console interface. On the left is a navigation sidebar with 'Identity and Access Management (IAM)' selected. The main content area displays the details for the user 'cloudshell-admin'. The 'Summary' section shows the ARN as 'arn:aws:iam::022499028207:user/cloudshell-admin', console access as 'Enabled without MFA', and creation date as 'October 22, 2024, 21:03 (UTC+05:30)'. It also lists two access keys: 'Access key 1' (AKIAQKPIL5TXR4ZBFAU3) and 'Access key 2'. Below this, the 'Security credentials' tab is active, showing the 'Console sign-in' section with a console sign-in link and a console password updated 1 minute ago.

```
[cloudshell-user@ip-10-136-32-211 ~]$ kubectl create cluster --name my-new-cluster --region us-east-1 -n podgroup-name standard-workers --node-type t2.medium --nodes 2
2024-10-22 18:04:20 [I] executing version 0.192.0
2024-10-22 18:04:20 [I] using region us-east-1
2024-10-22 18:04:20 [I] setting availability zones to [us-east-1a us-east-1b]
2024-10-22 18:04:20 [I] subnet for us-east-1a = public:192.168.0.0/19 private:192.168.0.0/19
2024-10-22 18:04:20 [I] subnet for us-east-1b = public:192.168.0.0/19 private:192.168.0.0/19
2024-10-22 18:04:20 [I] subgroup "standard-workers" will use "" [awsnetifcaleni3.0]
2024-10-22 18:04:20 [I] using Kubernetes version 1.30
2024-10-22 18:04:20 [I] creating EKS Cluster "my-new-cluster" in "us-east-1" region with managed nodes
2024-10-22 18:04:20 [I] will create 2 separate CloudFormation stacks for cluster itself and the initial managed subgroup
2024-10-22 18:04:20 [I] if you encounter any issues, check CloudFormation console or try "kubectl get describe stack --region=us-east-1 -c cluster=my-new-cluster"
2024-10-22 18:04:20 [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 18:04:20 [I] Cloudwatch logging will not be enabled for cluster "my-new-cluster" in "us-east-1"
2024-10-22 18:04:20 [I] you can enable it with "kubectl update-cluster-logging --enable-types=[SPECIFY-YOUR-TYPES-HERE (e.g. all)] --region=us-east-1 -c cluster=my-new-cluster"
2024-10-22 18:04:20 [I] default address spec(eni), kube-proxy, coredns were not specified, will install them as EC2 add-on
2024-10-22 18:04:20 [I]
Sequential tasks: { create cluster control plane "my-new-cluster".
3 sequential sub-tasks: {
1 essential sub-tasks: {
1 task: { create add-on },
wait for control plane to become ready,
},
create managed subgroup "standard-workers",
}
}
2024-10-22 18:04:20 [I] building cluster stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] building stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] deploying stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-cluster"
2024-10-22 18:04:20 [I] recommended policies were found for "spec(eni)" add-on, but since DNS is disabled on the cluster, kubectl cannot configure the requested permissions; the recommended way to provide IAM permissions for "spec(eni)" add-on is via identity associations; after add-on creation is completed, and all recommended policies to the config file, run "kubectl update add-on"
2024-10-22 18:04:20 [I] creating add-on
2024-10-22 18:04:20 [I] successfully created add-on
2024-10-22 18:04:20 [I] creating add-on
2024-10-22 18:04:20 [I] successfully created add-on
2024-10-22 18:04:20 [I] creating add-on
2024-10-22 18:04:20 [I] successfully created add-on
2024-10-22 18:04:20 [I] building new-managed-subgroup-stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] deploying stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] waiting for CloudFormation stack "kctl-my-new-cluster-subgroup-standard-workers"
2024-10-22 18:04:20 [I] waiting for the control plane to become ready
2024-10-22 18:04:20 [I] saved kubeconfig as "/home/cloudshell-user/.kube/config"
2024-10-22 18:04:20 [I] no tasks
2024-10-22 18:04:20 [I] ✓ all EKS cluster resources for "my-new-cluster" have been created
2024-10-22 18:04:20 [I] ✓ created a subgroup(s) in cluster "my-new-cluster"
2024-10-22 18:04:20 [I] subgroup "standard-workers" has 2 node(s).
2024-10-22 18:04:20 [I] note "ip-10-100-100-22-145.ec2.internal" is ready
2024-10-22 18:04:20 [I] note "ip-100-100-55-175.ec2.internal" is ready
2024-10-22 18:04:20 [I] waiting for at least 2 node(s) to become ready in "standard-workers"
2024-10-22 18:04:20 [I] subgroup "standard-workers" has 2 node(s).
2024-10-22 18:04:20 [I] note "ip-100-100-22-145.ec2.internal" is ready
2024-10-22 18:04:20 [I] note "ip-100-100-55-175.ec2.internal" is ready
2024-10-22 18:04:20 [I] ✓ created 1 managed subgroup(s) in cluster "my-new-cluster"
2024-10-22 18:04:20 [I] subnet command should work with "/home/cloudshell-user/.kube/config", try "kubectl get nodes"
2024-10-22 18:04:20 [I] ✓ EKS cluster "my-new-cluster" in "us-east-1" region is ready
[cloudshell-user@ip-10-136-32-211 ~]$
```

Step 4: Deploying Nginx Application

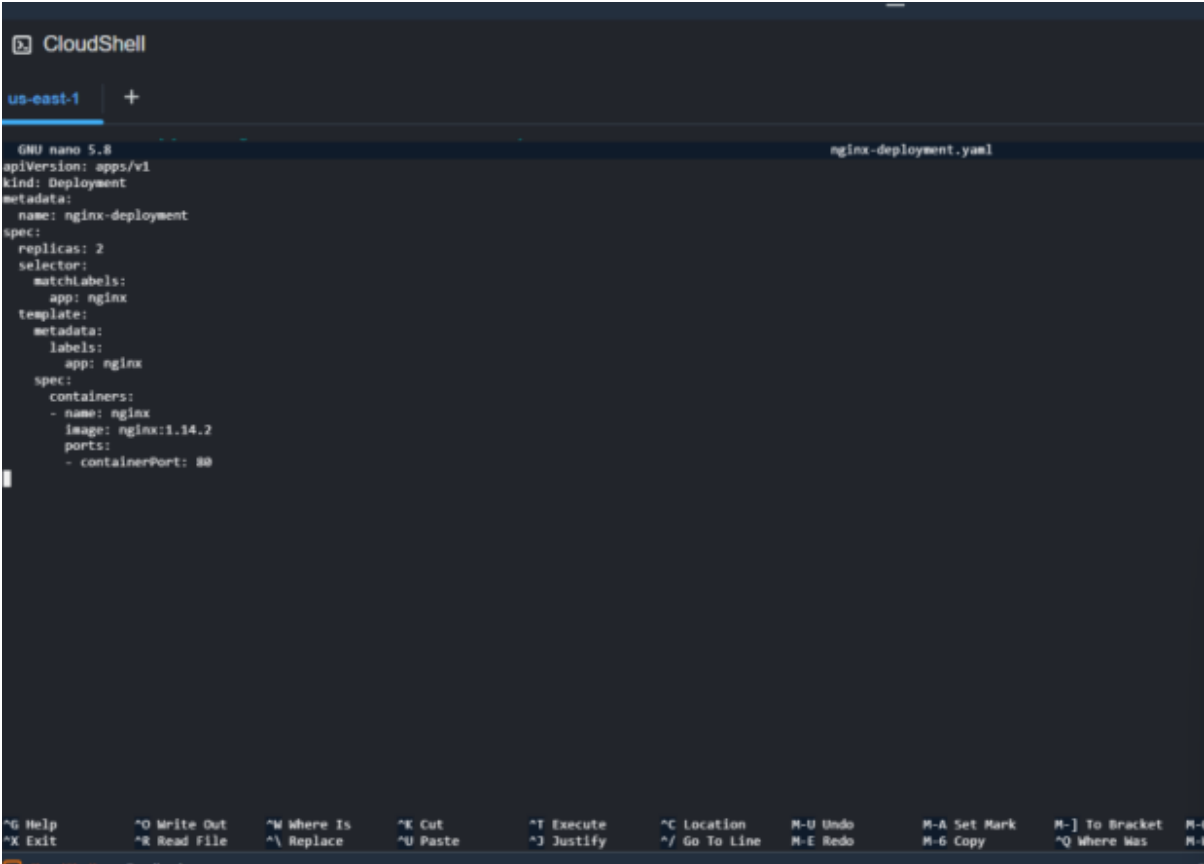
1. Create Deployment YAML

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 ~]$
```



```
GNU nano 5.8 nginx-deployment.yaml
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
```

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
```

```
[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
RollingUpdate: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
    nginx:
      Image:          nginx:1.34.2
      Port:           80/TCP
      Host Port:      80/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
```

Step 5: Expose the Application Using a LoadBalancer

1. Paste the following code:

yaml

Copy code a

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: LoadBalancer

ports: - port: 80

targetPort: 80

selector:

app: nginx

```
CloudShell
us-east-1 +
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. Retrieve the external ip

```
[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  80:31779/TCP 15s
```



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.

