Deploying a Microservices Application on Amazon EKS (Elastic Kubernetes Service)

Table of Contents

1. Problem Statement

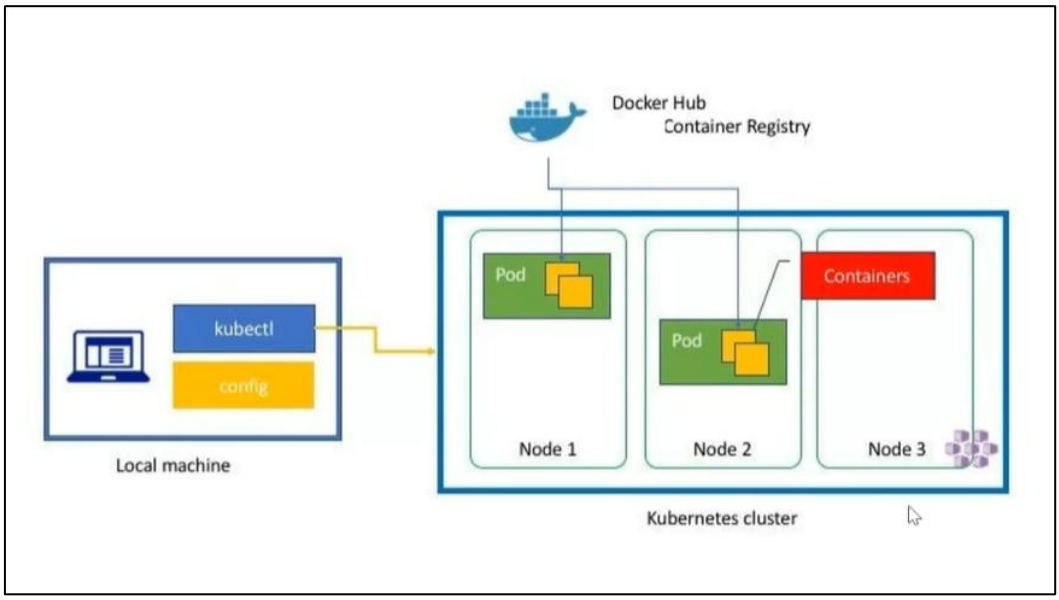
Ample Technologies is transforming its monolithic application into a microservices-based architecture. The application consists of one front-end service (Ruby) and two backend services (Node.js and Crystal). As an AWS Solutions Architect, I will deploy these services on AWS using Elastic Kubernetes Service (EKS), ensuring scalability and proper load balancing.

2. AWS Services Used:

- Amazon EKS
- Amazon EC2 (Node Groups)
- IAM (Roles and Policies)
- Elastic Load Balancer
- AWS CLI

3. Architecture Diagram

Below is the architecture diagram for the project:



4. Implementation Steps

Step 0: Install AWS CLI

Install AWS CLI from [AWS CLI Installation Guide]

```
C:\Users\reesa>aws --version
aws-cli/2.23.2 Python/3.12.6 Windows/11 exe/AMD64

C:\Users\reesa>
```

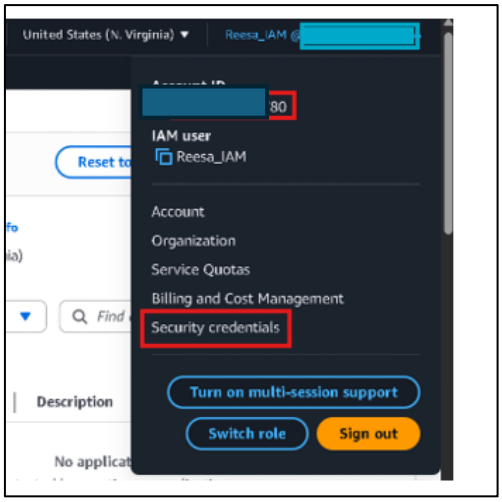
Step 1: Configure AWS CLI & Create EKS Cluster

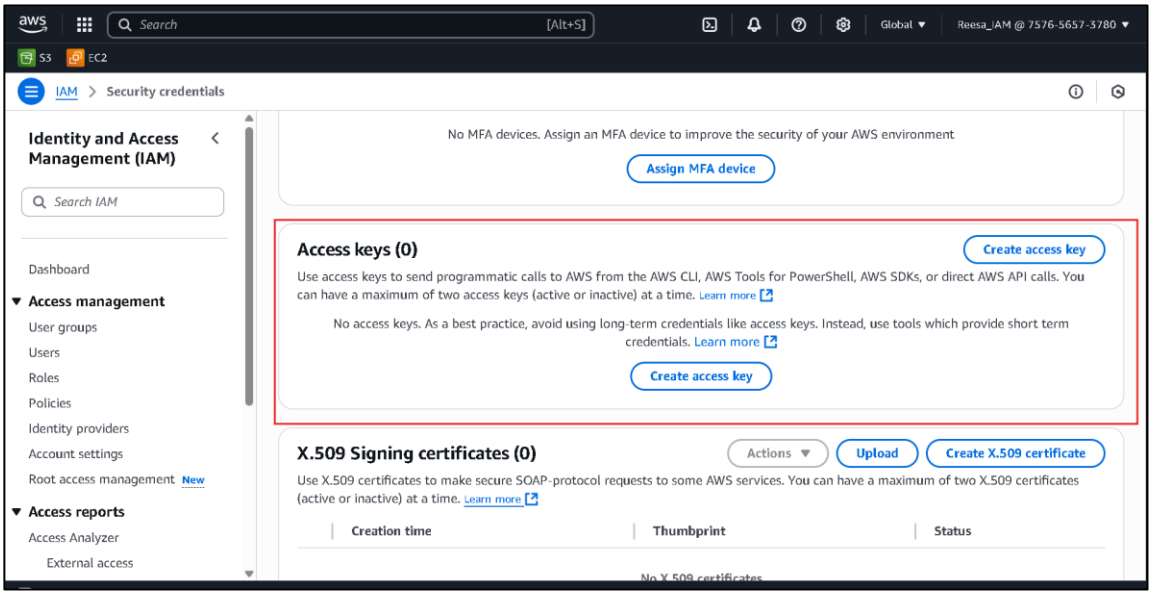
a. Run *aws configure*

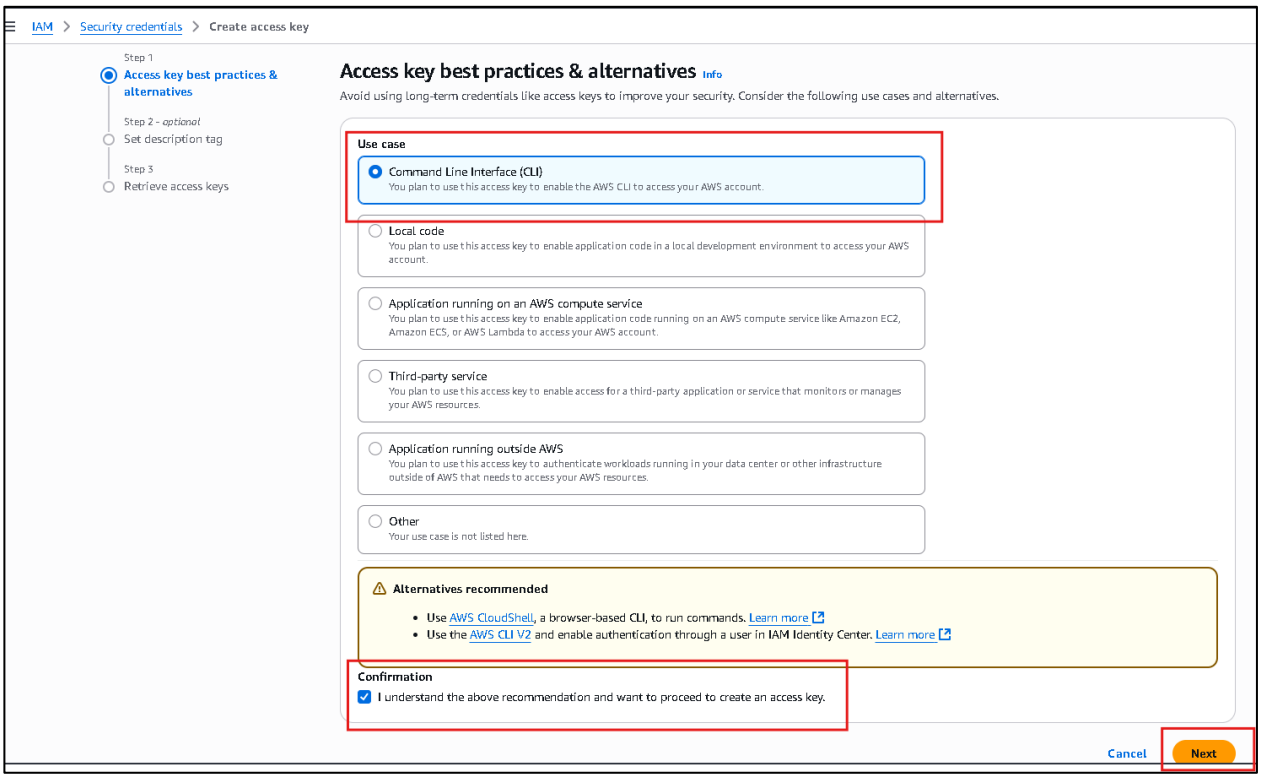
aws configure

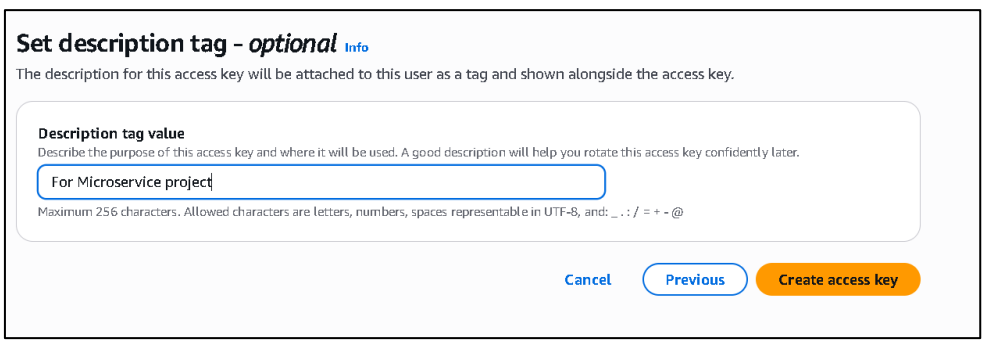
Before running , retrieve your **AWS Access Key ID** and **Secret Access Key** by following these steps:

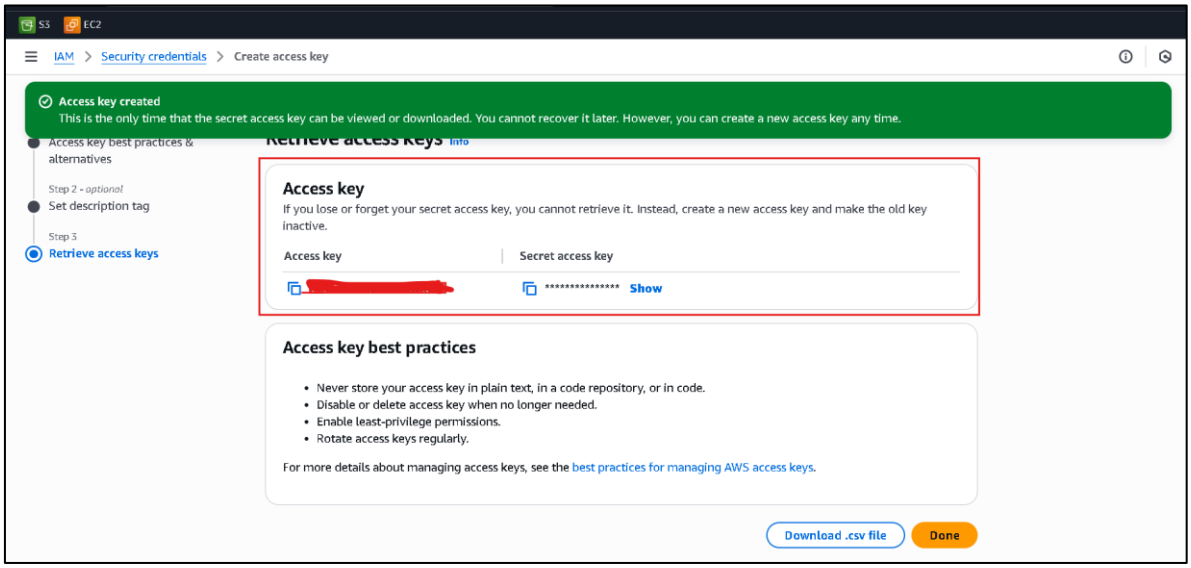
1. Go to AWS Console → IAM → Users
2. Click on your username
3. Navigate to the "**Security credentials**" tab
4. Click "**Create access key**"
5. Choose "**Command Line Interface (CLI)**" use case and click Next
6. Download the .csv file or copy the keys displayed




Search [Alt+S]
Global
Reesa_IAM @ 7576-5657-3780
S3
EC2
IAM > Security credentials
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Access Analyzer
External access
No MFA devices. Assign an MFA device to improve the security of your AWS environment
Assign MFA device
Access keys (0)
Create access key
Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more
No access keys. As a best practice, avoid using long-term credentials like access keys. Instead, use tools which provide short term credentials. Learn more
Create access key
X.509 Signing certificates (0)
Actions
Upload
Create X.509 certificate
Use X.509 certificates to make secure SOAP-protocol requests to some AWS services. You can have a maximum of two X.509 certificates (active or inactive) at a time. Learn more
Creation time
Thumbprint
Status
No X.509 certificates


IAM > Security credentials > Create access key
Step 1
Access key best practices & alternatives
Step 2 - optional
Set description tag
Step 3
Retrieve access keys
Access key best practices & alternatives Info
Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.
Use case
Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.
Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.
Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
Application running outside AWS
You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.
Other
Your use case is not listed here.
Alternatives recommended
Use AWS CloudShell, a browser-based CLI, to run commands. Learn more
Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. Learn more
Confirmation
I understand the above recommendation and want to proceed to create an access key.
Cancel
Next


Set description tag - optional Info
The description for this access key will be attached to this user as a tag and shown alongside the access key.
Description tag value
Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.
For Microservice project
Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . : / = + - @
Cancel
Previous
Create access key



Then, open your terminal and run:

```
aws configure
```

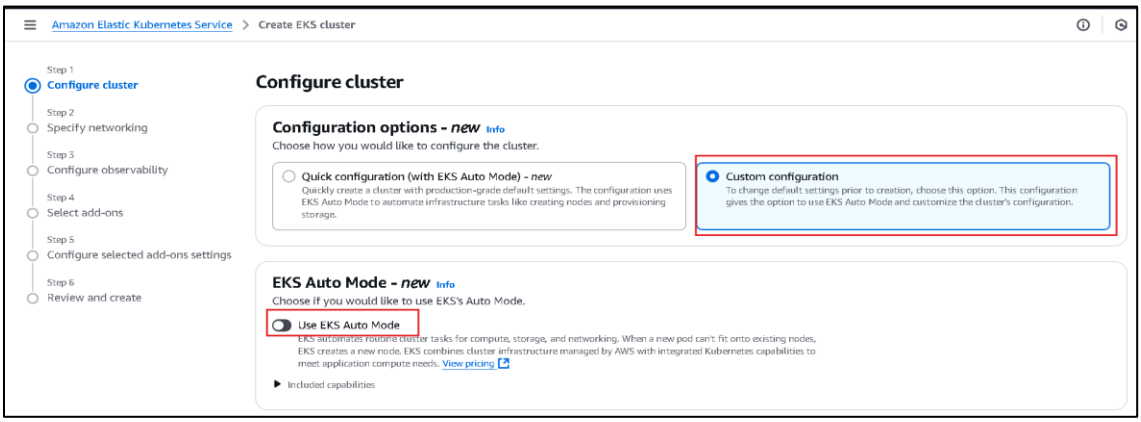
Enter the following:

- **AWS Access Key ID**
- **AWS Secret Access Key**
- **Region** (e.g., us-east-2)
- **Output format** (e.g., json)

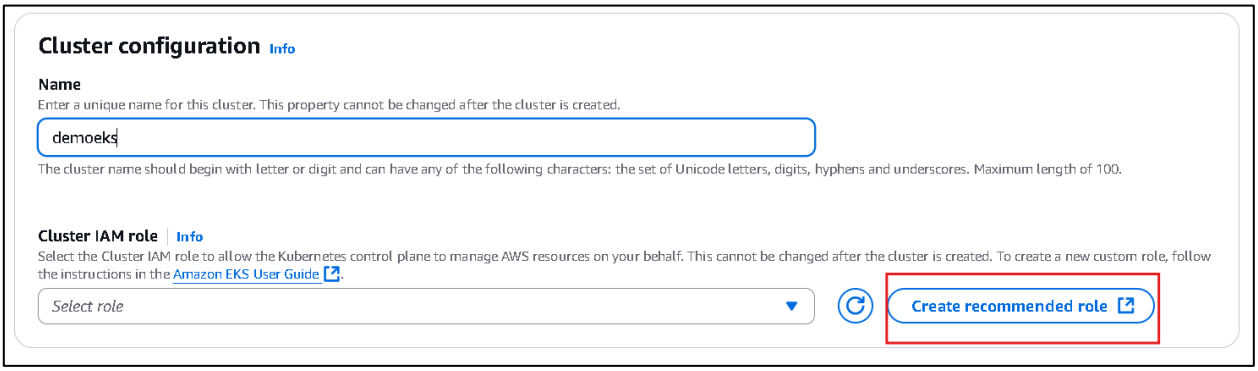
```
C:\Users\reesa>aws configure
AWS Access Key ID [****************7SSB]: [redacted]
AWS Secret Access Key [****************TFcr]: [redacted]
Default region name [us-east-2]: us-east-2
Default output format [json]: json

C:\Users\reesa>
```

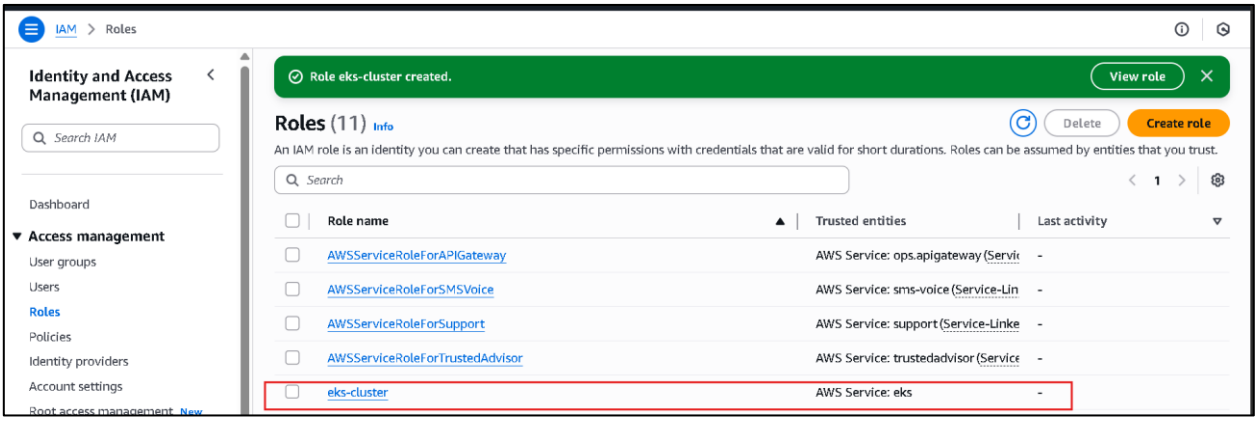
Step 2: Create EKS Cluster

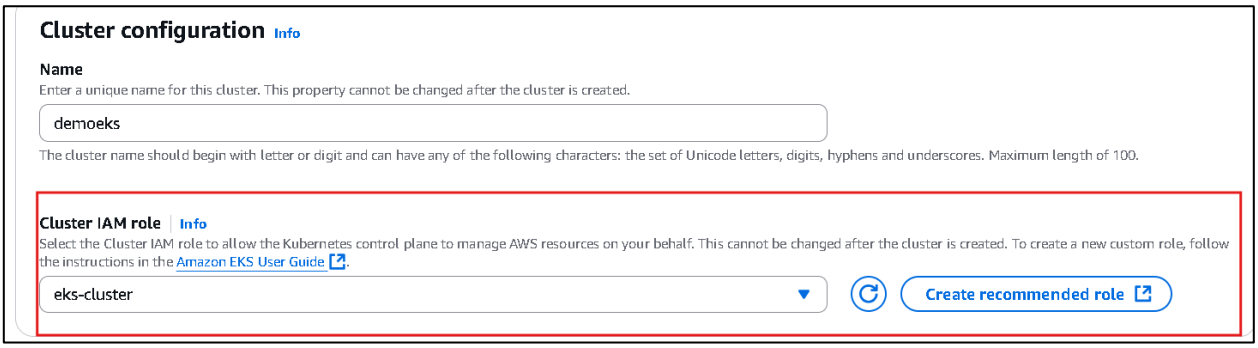
1. Navigate to the EKS console.
2. Click on "**Add cluster**" and select "**Create**".
3. Provide a name for your cluster (e.g., demoeks).



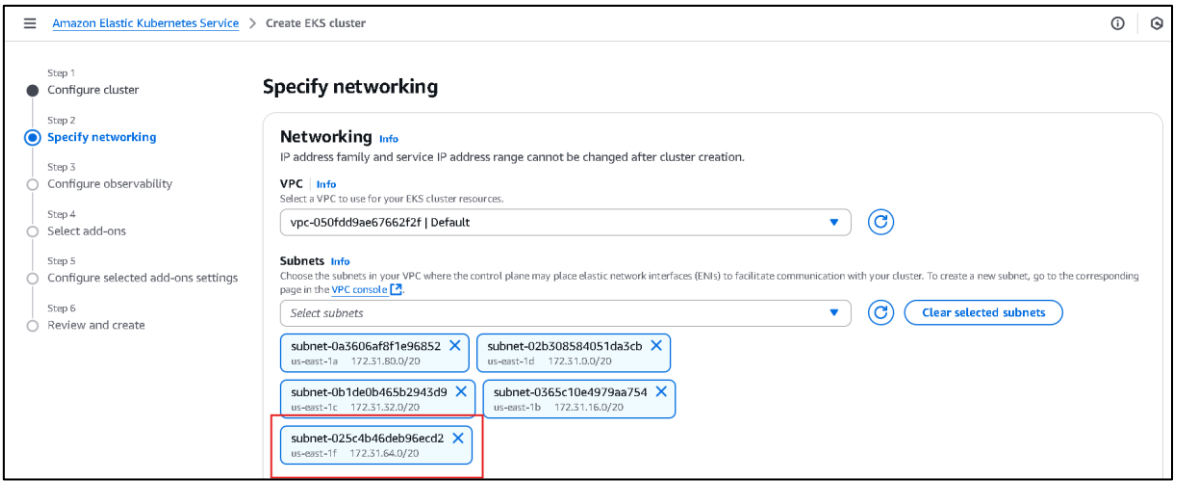
4. Select and create the IAM role as shown below.





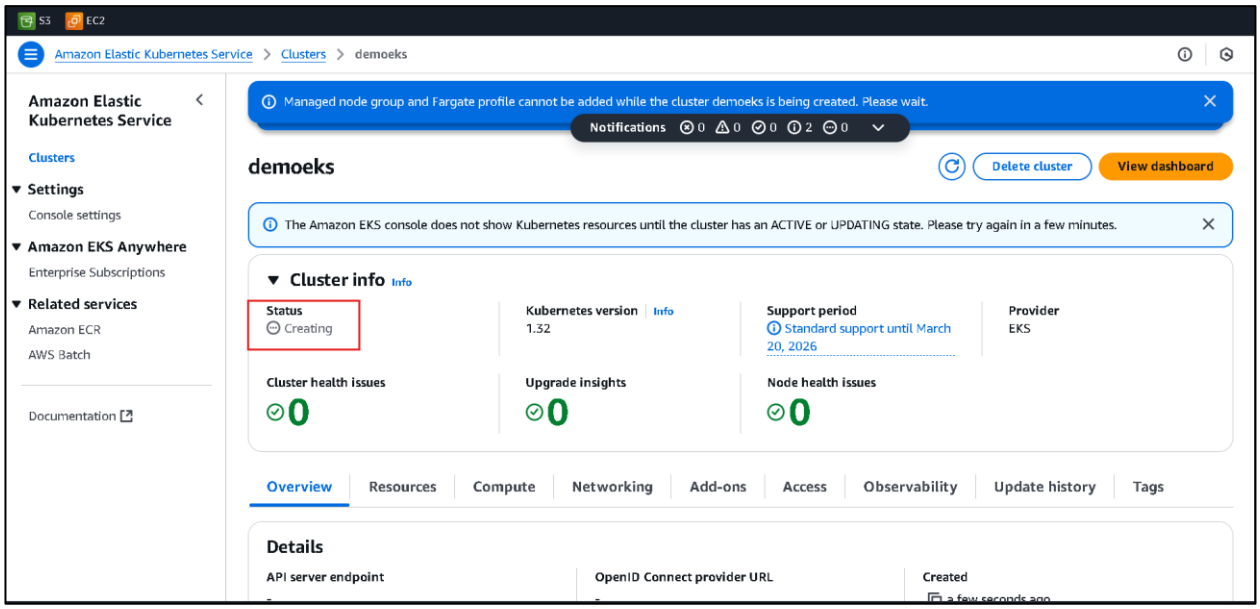


5. Configure networking settings as required.



The subnet in us-east-1f was removed during EKS cluster creation due to potential availability zone capacity issues or compatibility limitations. AWS occasionally restricts certain instance types or services in specific Availability Zones, so it's a best practice to use subnets in at least two validated and available AZs.

6. Create the cluster



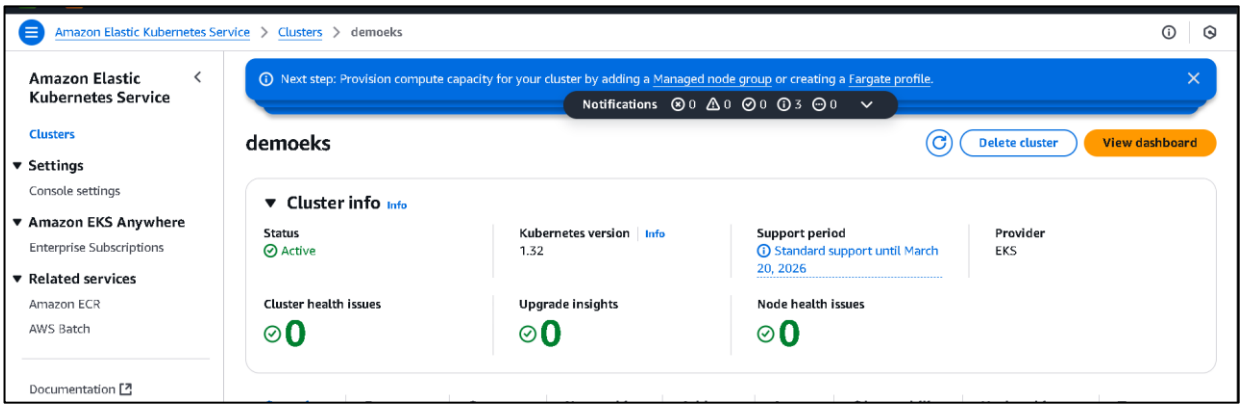
Step 3: Update kubeconfig

- Run the following command to update your kubeconfig file with the new cluster's context:

```
aws eks --region us-east-1 update-kubeconfig --
```

```
C:\Users\reesa>aws eks --region us-east-1 update-kubeconfig --name demoeks

Cluster status is CREATING

C:\Users\reesa>
```

That means your EKS cluster wasn't fully created yet, so the kubeconfig couldn't be updated or used.



```
C:\Users\reesa>aws eks --region us-east-1 update-kubeconfig --name demoeks
Added new context arn:aws:eks:us-east-1:757656573780:cluster/demoeks to C:\Users\reesa\.kube\config

C:\Users\reesa>
```

This means the kubeconfig is now successfully set up and you can interact with your EKS cluster using kubectl.

Step 4: Install kubectl

- If kubectl is not installed, follow these steps:

```
curl.exe -LO "https://dl.k8s.io/release/v1.29.0/bin/windows/amd64/kubectl.exe"

curl.exe -LO "https://dl.k8s.io/v1.29.0/bin/windows/amd64/kubectl.exe.sha256"

CertUtil -hashfile kubectl.exe SHA256

type kubectl.exe.sha256
```

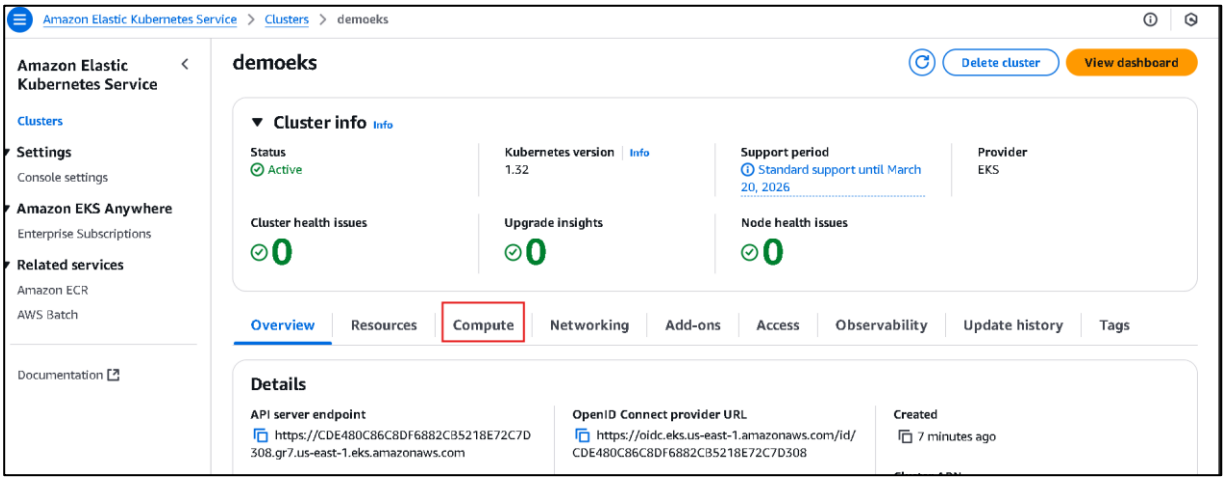
- Add the directory containing kubectl.exe to your system's PATH environment variable.
- Verify the installation:

```
kubectl version --client
```

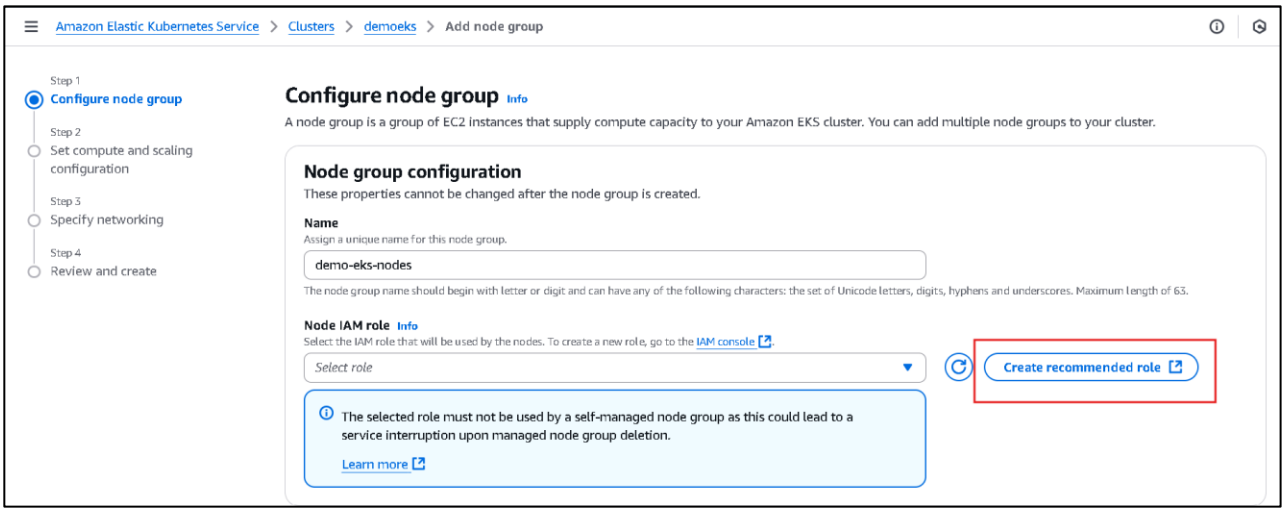
```
C:\Users\reesa>kubectl version --client
Client Version: v1.31.4
Kustomize Version: v5.4.2
```

Step 5: Create Node Group

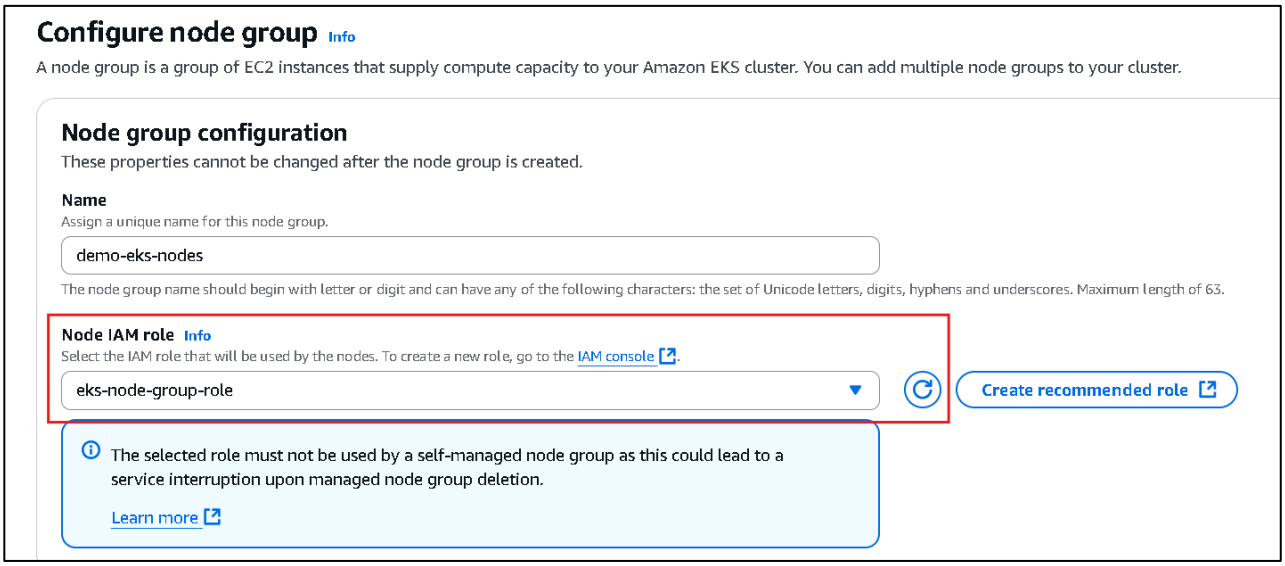
1. In the EKS console, navigate to your cluster.
2. Click on "**Add node group**".



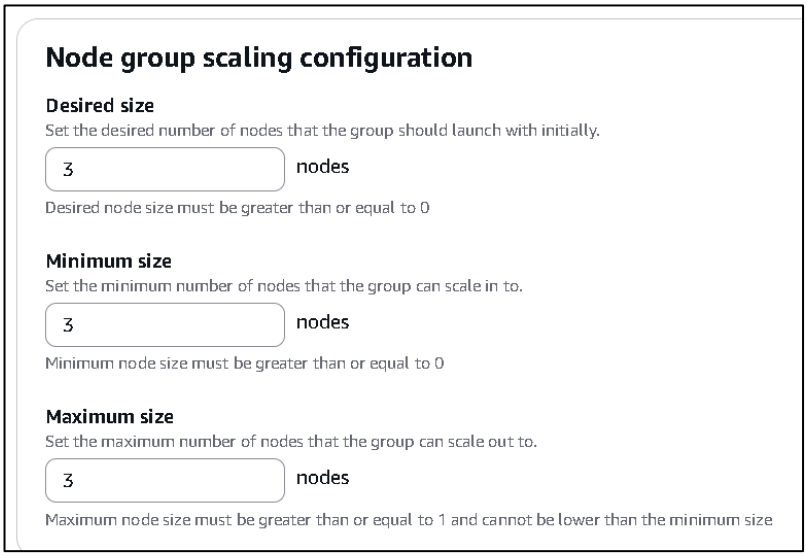
3. Provide a name for the node group.
4. Select and create an IAM role as shown below.



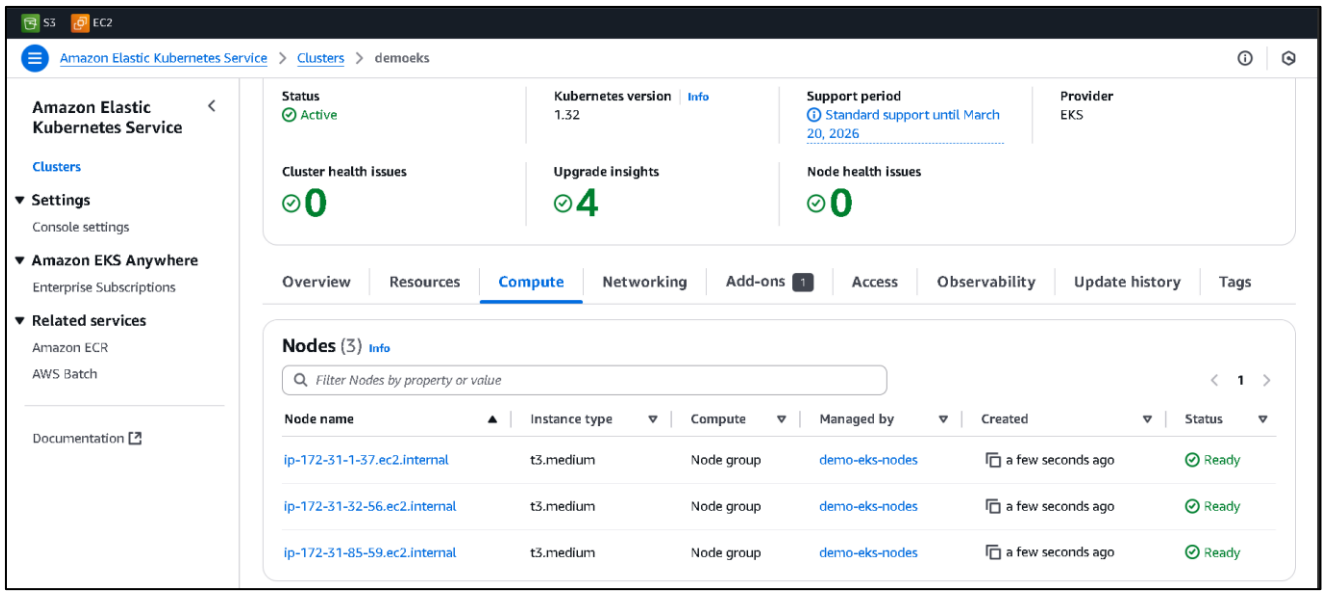
5. Refresh and add the created IAM role



6. Configure the following settings:
    - Instance type: t2.micro
    - Disk size: 5 GB
    - Minimum size: 3
    - Maximum size: 3
    - Desired size: 3



7. Allow remote access and select an existing SSH key pair.
8. Create the node group.



Step 6: Verify Node Group

- Run the following command to verify that the nodes are active:

```
kubectl get nodes
```

```
C:\Users\reesa>kubectl get nodes
NAME                          STATUS   ROLES    AGE    VERSION
ip-172-31-1-37.ec2.internal   Ready    <none>   109s   v1.32.3-eks-473151a
ip-172-31-32-56.ec2.internal  Ready    <none>   111s   v1.32.3-eks-473151a
ip-172-31-85-59.ec2.internal  Ready    <none>   110s   v1.32.3-eks-473151a

C:\Users\reesa>
```

Step 7: Deploy Microservices

This application consists of three microservices, each containerized and deployed independently on Amazon EKS. The Kubernetes manifests for these services are provided in the zipped folder available on my GitHub repository inside the 'Code' file (ecsdemo-nodejs, ecsdemo-crystal, and ecsdemo-frontend).

1. **Deploy ecsdemo-nodejs Service**
   - A Node.js-based backend API.
   - Exposes an internal service (`ClusterIP`) for communication within the Kubernetes cluster.
   - Does **not** require public access or a Load Balancer.
     - Navigate to the ecsdemo-nodejs/kubernetes directory.
     - Apply the deployment and service manifests:

```
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-nodejs\ku
bernetes>kubectl apply -f deployment.yaml
deployment.apps/ecsdemo-nodejs created

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-nodejs\ku
bernetes>kubectl apply -f service.yaml
service/ecsdemo-nodejs created

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-nodejs\ku
bernetes>
```

2. **Deploy ecsdemo-crystal Service**
   - A backend API written in Crystal programming language.
   - Also exposed internally via `ClusterIP`.
   - Used by the frontend to fetch or process data.

   1. Navigate to the ecsdemo-crystal/kubernetes directory.
   2. Apply the deployment and service manifests:

```
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\k
ubernetes>kubectl apply -f deployment.yaml
deployment.apps/ecsdemo-crystal created

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\k
ubernetes>kubectl apply -f service.yaml
service/ecsdemo-crystal created

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\k
ubernetes>
```

3. **Deploy ecsdemo-frontend Service**
   - The user-facing component built with Ruby.
   - Exposed publicly via a Kubernetes `LoadBalancer` service.
   - Automatically provisions an **Elastic Load Balancer (ELB)**, allowing access to the application from the internet using the provided DNS name.

1. Navigate to the ecsdemo-frontend/kubernetes directory.
2. Apply the deployment and service manifests:

```
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-frontend\
kubernetes>kubectl apply -f deployment.yaml
deployment.apps/ecsdemo-frontend created

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-frontend\
kubernetes>kubectl apply -f service.yaml
service/ecsdemo-frontend created

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-frontend\
kubernetes>
```

3. Verify the deployment and get the service details

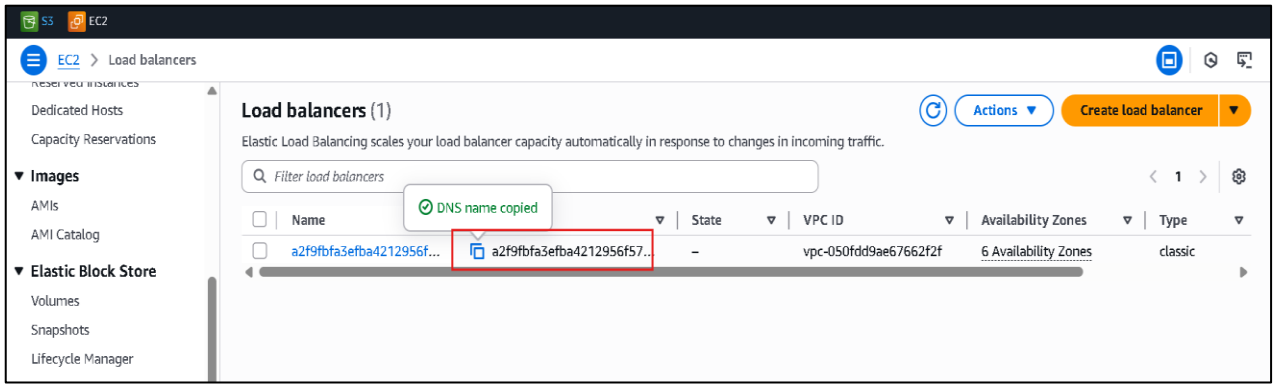
```
kubectl get pods
```

```
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\kubernetes>kub
ectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
ecsdemo-crystal-76796bbcf-2rnrl    1/1     Running   0          43s
ecsdemo-frontend-54d56d89fb-76sp9  1/1     Running   0          7m19s
ecsdemo-nodejs-6d679857c4-9cbpn    1/1     Running   0          2m19s
```

5. Testing & Validation

Step 1: Open the Application via Load Balancer

1. Go to the **EC2 Console → Load Balancers** (under **Load Balancing** in the left sidebar).
2. Click on the load balancer associated with your ecsdemo-frontend service.
3. Copy the **DNS name** of the load balancer.



4. Open an **Incognito window** in your browser and **paste the DNS name** in the address bar.

5. You should see the deployed application running.

Step 2: Scale Deployments

➢ Scale each deployment to 3 replicas:

```
kubectl scale deployment ecsdemo-nodejs --replicas=3
kubectl scale deployment ecsdemo-crystal --replicas=3
kubectl scale deployment ecsdemo-frontend --replicas=3
```

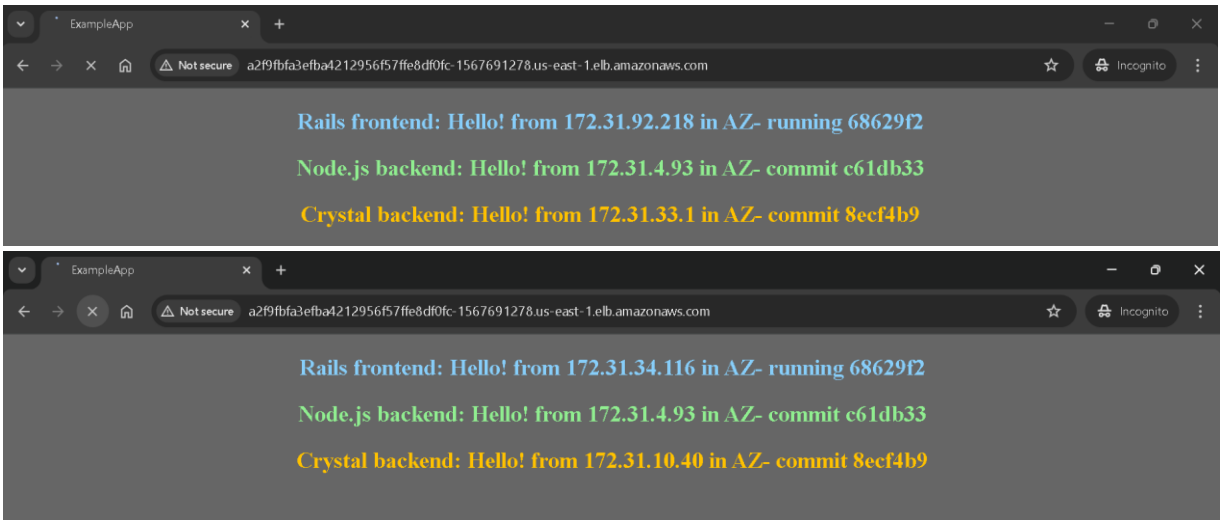
```
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\kubernetes> ku
bectl scale deployment ecsdemo-crystal --replicas=3
deployment.apps/ecsdemo-crystal scaled

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\kubernetes>get
 pods
'get' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\kubernetes>kub
ectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
ecsdemo-crystal-76796bbcf-2rnrl     1/1     Running   0          4m34s
ecsdemo-crystal-76796bbcf-4t528     1/1     Running   0          18s
ecsdemo-crystal-76796bbcf-zvbvr     1/1     Running   0          18s
ecsdemo-frontend-54d56d89fb-76sp9   1/1     Running   0          11m
ecsdemo-nodejs-6d679857c4-9cbpn     1/1     Running   0          6m10s

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\kubernetes>
```

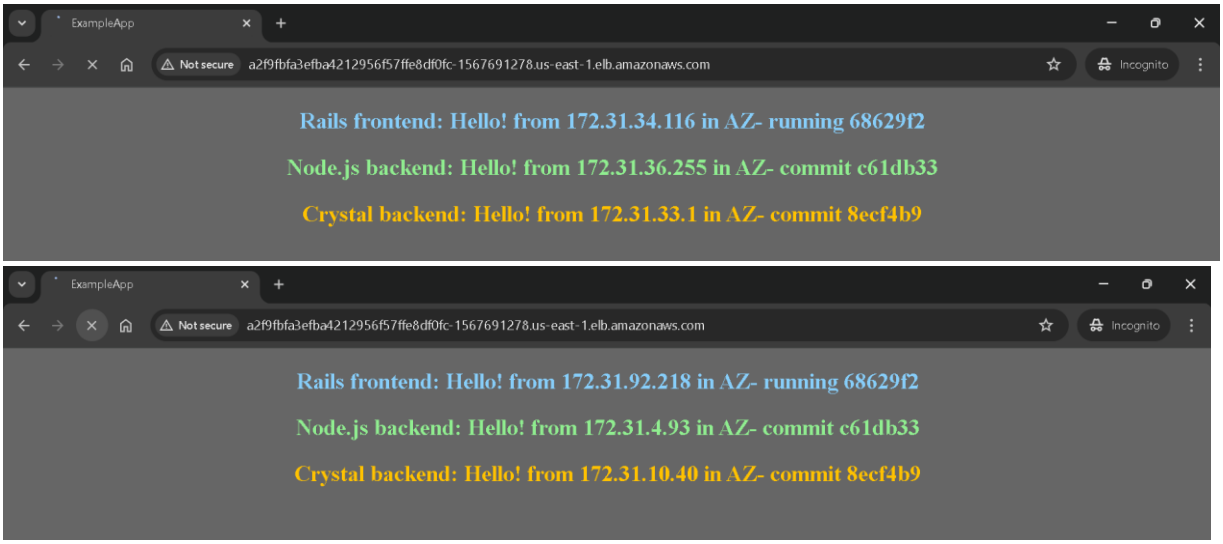
```
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-crystal\kubernetes>cd
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-frontend\kubernetes

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-frontend\kubernetes>ku
bectl scale deployment ecsdemo-frontend --replicas=3
deployment.apps/ecsdemo-frontend scaled

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-frontend\kubernetes>cd
 C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-nodejs\kubernetes

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-nodejs\kubernetes>kube
ctl scale deployment ecsdemo-nodejs --replicas=3
deployment.apps/ecsdemo-nodejs scaled
```

- After scaling the microservices using kubectl scale, Kubernetes automatically schedules additional pod replicas across the available worker nodes.

```
C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-nodejs\kubernetes>kube
ctl get pods
NAME                                READY   STATUS    RESTARTS   AGE
ecsdemo-crystal-76796bbcf-2rnrl     1/1     Running   0          7m54s
ecsdemo-crystal-76796bbcf-4t528     1/1     Running   0          3m38s
ecsdemo-crystal-76796bbcf-zvbvr     1/1     Running   0          3m38s
ecsdemo-frontend-54d56d89fb-76sp9   1/1     Running   0          14m
ecsdemo-frontend-54d56d89fb-kbkd5   1/1     Running   0          102s
ecsdemo-frontend-54d56d89fb-rsb7q   1/1     Running   0          102s
ecsdemo-nodejs-6d679857c4-9cbpn     1/1     Running   0          9m30s
ecsdemo-nodejs-6d679857c4-v8msk     1/1     Running   0          66s
ecsdemo-nodejs-6d679857c4-x2sqm     1/1     Running   0          66s

C:\Users\reesa\OneDrive\Desktop\AWS WORKS\DevOps Interview Questions\Kubernetes\6. Microservices\Codes\ecsdemo-nodejs\kubernetes>
```

- As a result, running will display multiple pods for each microservice, each with a **unique IP address** assigned by the Kubernetes network. This ensures **high availability** and **load distribution** across replicas.





6. Clean Up AWS Resources

To avoid incurring unnecessary charges, delete the following resources after the project is completed:

- **EKS Cluster**
- **Node Group**
- **IAM Roles and Policies** created for EKS
- **Load Balancer** associated with the frontend service
- **Key pairs**
- **Security Groups**
- **Access Keys**

7. Conclusion

This project successfully demonstrates the deployment of a real-world microservices-based architecture on Amazon EKS using Docker and Kubernetes. By containerizing and orchestrating three services — frontend, Node.js backend, and Crystal backend — the application was made scalable, modular, and cloud-native. Key AWS services like IAM, EC2, Load Balancer, and EKS were integrated to ensure secure, reliable, and high-performing deployments.

From configuring AWS CLI and Kubernetes tools to managing infrastructure and scaling deployments, this project offers a hands-on understanding of cloud-native DevOps practices. It showcases not only technical proficiency in using AWS and Kubernetes but also an ability to architect scalable, production-ready solutions aligned with modern application development standards.

This end-to-end experience strengthens your foundation in cloud engineering and DevOps, making you better equipped for roles involving cloud infrastructure automation and microservices deployment on AWS.