



Deploy Backend with Kubernetes



saqibh49@gmail.com

The screenshot shows a terminal session in AWS CloudShell. The user is deploying a Kubernetes cluster named "nextwork-eks-cluster". The session starts with the creation of a CloudFormation stack, followed by the configuration of addons like "vpc-cni", "kube-proxy", and "coredns". Nodes are then deployed, and the cluster becomes ready. Finally, a metrics server is created and successfully added to the cluster.

```
aws | ⚡ | Search | [Option+R] | ⓘ | United States (Ohio) | Saqib Hossain (S104-8260-3806) | saqib-JAM-Admin

2026-02-22 21:06:09 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-cluster"
2026-02-22 21:07:09 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-cluster"
2026-02-22 21:08:09 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-cluster"
2026-02-22 21:09:09 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-cluster"
2026-02-22 21:10:09 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-cluster"
2026-02-22 21:10:10 [i] recommended policies were found for "vpc-cni" addon, but since OIDC is disabled on the cluster, eksctl cannot configure the required permissions. the recommended way to provide IAM permissions for "vpc-cni" addon is via pod identity associations; after addon creation is completed, add the recommended policies to the config file, under 'addon.PodIdentityAssociations', and run 'eksctl update addon'
2026-02-22 21:10:10 [i] creating addon: vpc-cni
2026-02-22 21:10:11 [i] successfully created addon: vpc-cni
2026-02-22 21:10:11 [i] creating addon: kube-proxy
2026-02-22 21:10:12 [i] successfully created addon: kube-proxy
2026-02-22 21:10:13 [i] creating addon: coredns
2026-02-22 21:10:13 [i] successfully created addon: coredns
2026-02-22 21:12:12 [i] building managed nodegroup stack "eksctl-nextwork-eks-cluster-nodegroup-ng-54c8362b"
2026-02-22 21:12:12 [i] deploying stack "eksctl-nextwork-eks-cluster-nodegroup-ng-54c8362b"
2026-02-22 21:12:13 [i] waiting for Cloudformation stack "eksctl-nextwork-eks-cluster-nodegroup-ng-54c8362b"
2026-02-22 21:12:45 [i] waiting for Cloudformation stack "eksctl-nextwork-eks-cluster-nodegroup-ng-54c8362b"
2026-02-22 21:13:30 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-nodegroup-ng-54c8362b"
2026-02-22 21:14:16 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-nodegroup-ng-54c8362b"
2026-02-22 21:15:39 [i] waiting for CloudFormation stack "eksctl-nextwork-eks-cluster-nodegroup-ng-54c8362b"
2026-02-22 21:15:39 [i] waiting for the cluster to become healthy
2026-02-22 21:15:40 [i] saved kubeconfig as "/home/ec2-user/.kube/config"
2026-02-22 21:15:40 [i] no tasks
2026-02-22 21:15:40 [i] all EKS cluster resources for "nextwork-eks-cluster" have been created
2026-02-22 21:15:40 [i] nodegroup "ng-54c8362b" has 3 nodes(s)
2026-02-22 21:15:40 [i] node "ip-192-168-12-136.us-east-2.compute.internal" is ready
2026-02-22 21:15:40 [i] node "ip-192-168-39-228.us-east-2.compute.internal" is ready
2026-02-22 21:15:40 [i] node "ip-192-168-70-2.us-east-2.compute.internal" is ready
2026-02-22 21:15:40 [i] nodegroup "ng-54c8362b" has 3 nodes(s) and is now ready in ng-54c8362b
2026-02-22 21:15:40 [i] nodegroup "ng-54c8362b" has 3 node(s)
2026-02-22 21:15:40 [i] node "ip-192-168-12-136.us-east-2.compute.internal" is ready
2026-02-22 21:15:40 [i] node "ip-192-168-39-228.us-east-2.compute.internal" is ready
2026-02-22 21:15:40 [i] node "ip-192-168-70-2.us-east-2.compute.internal" is ready
2026-02-22 21:15:40 [i] created 1 managed nodegroup(s) in cluster "nextwork-eks-cluster"
2026-02-22 21:15:40 [i] creating addon: metrics-server
2026-02-22 21:15:41 [i] successfully created addon: metrics-server
2026-02-22 21:15:41 [i] kubectl command should work with "/home/ec2-user/.kube/config", try 'kubectl get nodes'
2026-02-22 21:15:41 [i] kubectl command should work with "nextwork-eks-cluster" in "us-east-2" region is ready
[e2c-user@ip-172-31-29-131 ~]$
```

i-02d3b1b0e3d89cf21 (nextwork-eks-instance)
PublicIPs: 3.145.209.147 PrivateIPs: 172.31.29.131

CloudShell Feedback Console Mobile App © 2026, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

saqibh49@gmail.com

NextWork Student

nextwork.org

Introducing Today's Project!

In this project, I will set up a backend app, install kubectl, and deploy it to my Kubernetes cluster. I'm doing this project because this is where everything comes together — I've built images, written manifests, and now it's time to actually deploy and see my app running on Kubernetes.

Tools and concepts

I used Kubernetes, ECR, kubectl, eksctl, Docker, EC2, Git, and IAM to deploy a containerized backend application to an EKS cluster. Key concepts include using manifests to define how the app runs and gets exposed, using kubectl to apply those manifests, pushing images to ECR for the cluster to pull from, and verifying the deployment through the EKS console.

Project reflection

This project took me approximately 2 hours. The most challenging part was figuring out the errors I got when creating my cluster. I found that there were old resources in my CloudFormation blocking my new clusters so I deleted those first and things started going more smoothly. My favourite part was seeing my app actually running on Kubernetes after all the setup — knowing that every piece from the Docker image to the manifests to the cluster was working together made it all worth it.

saqibh49@gmail.com

NextWork Student

nextwork.org

Project Set Up

Kubernetes cluster

To set up today's project, I launched a Kubernetes cluster. The cluster's role in this deployment is to be the environment where my containerized backend actually runs — it manages the nodes, schedules the containers, and makes sure everything stays up and accessible.

Backend code

I retrieved backend code by installing Git on my EC2 instance and cloning the repository from GitHub. Pulling code is essential to this deployment because without the source code, I can't build the Docker image or access the manifest files that tell Kubernetes how to run my app.

Container image

Once I cloned the backend code, I built a container image because Kubernetes runs applications inside containers, so the app needs to be packaged into an image first. Without an image, Kubernetes wouldn't know what to deploy — it needs that image as the blueprint for spinning up containers across the cluster.

saqibh49@gmail.com

NextWork Student

nextwork.org

I also pushed the container image to a container registry, which is a central storage location for container images that services like Kubernetes can pull from. ECR facilitates scaling for my deployment because whenever Kubernetes needs to spin up more containers to handle traffic, it can quickly pull the same image from ECR without me having to manually distribute it to each node.

saqibh49@gmail.com

NextWork Student

nextwork.org

Manifest files

Kubernetes manifests are YAML files that define what you want Kubernetes to do — like which containers to run, how many replicas to have, and how to expose them. Manifests are helpful because they let you describe your entire deployment in a file that's easy to reuse, version control, and share, instead of running a bunch of manual commands every time.

A Deployment manifest manages how my app runs on the cluster — things like which image to use, how many replicas to maintain, and how to handle updates or crashes. The container image URL in my Deployment manifest tells Kubernetes where to pull the app from, which in my case is my ECR repository.

A Service resource exposes my application so it can actually receive traffic from users or other services. My Service manifest sets up a NodePort Service that routes traffic on port 8080 to my backend containers, opening a port on each node so the app is accessible from outside the cluster.

saqibh49@gmail.com

NextWork Student

nextwork.org

```
aws [Option+S] Search United States (Ohio) Saqib Hossain (S104-8260-3806) saqib-JAM-Admin
[ec2-user@ip-172-31-29-131 nextwork-flask-backend]$ ls flask-deployment.yaml
ls: cannot access flask-deployment.yaml: No such file or directory
[ec2-user@ip-172-31-29-131 nextwork-flask-backend]$ cat << EOF > flask-deployment.yaml
-->
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nextwork-flask-backend
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nextwork-flask-backend
  template:
    metadata:
      labels:
        app: nextwork-flask-backend
    spec:
      containers:
        - name: nextwork-flask-backend
          image: your-ecr-image-url
          ports:
            - containerPort: 8080
EOF
[ec2-user@ip-172-31-29-131 nextwork-flask-backend]$ ls flask-deployment.yaml
ls: cannot access flask-deployment.yaml: No such file or directory
[ec2-user@ip-172-31-29-131 nextwork-flask-backend]$ nano flask-deployment.yaml
[ec2-user@ip-172-31-29-131 nextwork-flask-backend]$ i-02d3b1b0e3d9cf21 (nextwork-eks-instance)
PublicIPs: 3.145.209.147 PrivateIPs: 172.31.29.131

CloudShell Feedback Console Mobile App © 2026, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences
```

saqibh49@gmail.com

NextWork Student

nextwork.org

Backend Deployment!

To deploy my backend application, I used kubectl to apply my Deployment and Service manifest files to the cluster. This told Kubernetes to pull my container image from ECR, spin up the pods, and expose the app through a NodePort so it's accessible from outside the cluster.

kubectl

kubectl is the command-line tool for interacting with Kubernetes clusters. I need this tool to apply my manifests and manage my deployments, pods, and services. I can't use eksctl for the job because eksctl is specifically for creating and managing EKS clusters themselves — once the cluster is up, kubectl is what you use to actually work with what's running inside it.

saqibh49@gmail.com

NextWork Student

nextwork.org

The screenshot shows a terminal window titled 'aws' with the user 'Saqib Hossain (S104-8260-3806)' and 'saqib-JAM-Admin'. The terminal displays a log of commands run on an AWS CloudFormation stack named 'eksctl-nextnetwork-eks-cluster-cluster'. The log includes:

- Waiting for CloudFormation stack 'eksctl-nextnetwork-eks-cluster-cluster'.
- Creating add-on: vpc-cni.
- Creating add-on: kube-proxy.
- Successfully created add-on: kube-proxy.
- Successfully created add-on: coredns.
- Building managed nodegroup stack 'eksctl-nextnetwork-eks-cluster-nodegroup-ng-54c8362b'.
- Waiting for CloudFormation stack 'eksctl-nextnetwork-eks-cluster-nodegroup-ng-54c8362b'.
- Waiting for the control plane to become ready.
- No tasks running.
- All AWS CloudFormation resources in 'nextnetwork-eks-cluster' have been created.
- Node group 'ng-54c8362b' has 3 node(s).
- Node 'ip-192-168-12-136.us-east-2.compute.internal' is ready.
- Node 'ip-192-168-39-228.us-east-2.compute.internal' is ready.
- Node 'ip-192-168-70-2.us-east-2.compute.internal' is ready.
- Waiting for at least 3 node(s) to become ready in 'ng-54c8362b'.
- Node group 'ng-54c8362b' has 3 node(s).
- Node 'ip-192-168-12-136.us-east-2.compute.internal' is ready.
- Node 'ip-192-168-39-228.us-east-2.compute.internal' is ready.
- Node 'ip-192-168-70-2.us-east-2.compute.internal' is ready.
- Creating add-on: metrics-server.
- Successfully created add-on: metrics-server.
- Execution command should work with '/home/ec2-user/.kube/config', try 'kubectl get nodes'.
- EKS cluster 'nextnetwork-eks-cluster' in 'us-east-2' region is ready.

The terminal prompt is '| ec2-user@ip-172-31-29-131 ~ \$|'

i-02d3b0e3d9cf21 (nextnetwork-eks-instance)

PublicIPs: 3.145.209.147 PrivateIP: 172.31.29.131

© 2026, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

saqibh49@gmail.com

NextWork Student

nextwork.org

Verifying Deployment

Once I gained access into my cluster's nodes, I discovered pods running inside each node. Pods are the smallest units in Kubernetes — they're basically wrappers around one or more containers that get deployed together on the same node. Containers in a pod share the same network and storage, meaning they can talk to each other using localhost and access the same files without any extra configuration.

The EKS console shows you the events for each pod, where I could see the steps Kubernetes took to get my pod running — like scheduling it to a node, pulling the image from ECR, creating the container, and starting it up. This validated that my entire deployment pipeline worked end to end, from building the image to having it running live on my cluster.

The screenshot shows the AWS EKS Cluster Events page. The left sidebar includes options for Dashboard, Clusters, Settings (Dashboard settings, Console settings), Amazon EKS Anywhere (Enterprise Subscriptions), Related services (Amazon ECR, AWS Batch), Documentation, and EKS workshops. The main content area shows the following details:

- Metrics Server Configuration:** A table with three rows:
 - app.kubernetes.io/instance: metrics-server
 - app.kubernetes.io/name: metrics-server
 - pod-template-hash: 85d777b88f
- No Annotations:** A message stating "No annotations have been defined for this resource."
- Events (5):** A table showing five events:
 - Normal - Scheduled:** 15 minutes ago, default-scheduler, Message: "Successfully assigned kube-system/metrics-server-85d777b88f-sckpm to ip-192-168-12-136.us-east-2.compute.internal"
 - Normal - Pulling:** 15 minutes ago, kubelet, Message: "Pulling image \"602401143452.dkr.ecr.us-east-2.amazonaws.com/eks/metrics-server:v0.8.1-eksbuild.1\" in 141s (1.41s including waiting). Image size: 22446531 bytes"
 - Normal - Pulled:** 15 minutes ago, kubelet, Message: "Successfully pulled image \"602401143452.dkr.ecr.us-east-2.amazonaws.com/eks/metrics-server:v0.8.1-eksbuild.1\" in 141s (1.41s including waiting). Image size: 22446531 bytes"
 - Normal - Created:** 15 minutes ago, kubelet, Message: "Created container: metrics-server"
 - Normal - Started:** 15 minutes ago, kubelet, Message: "Started container metrics-server"



nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects

