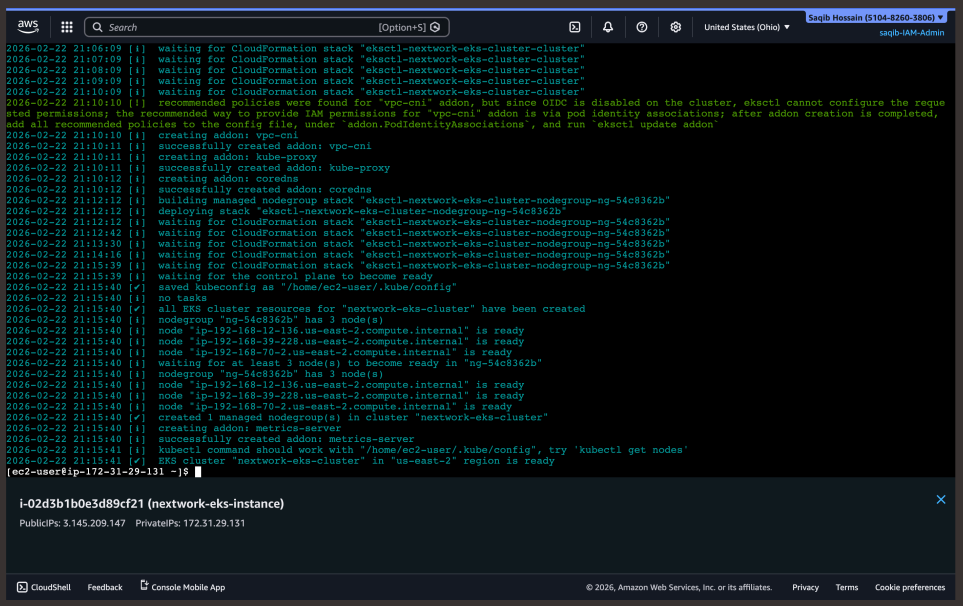
nextwork.org

Deploy Backend with Kubernetes

saqibh49@gmail.com




saqibh49@gmail.com
NextWork Student

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.

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.

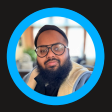
Backend code

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.

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

Manifest files

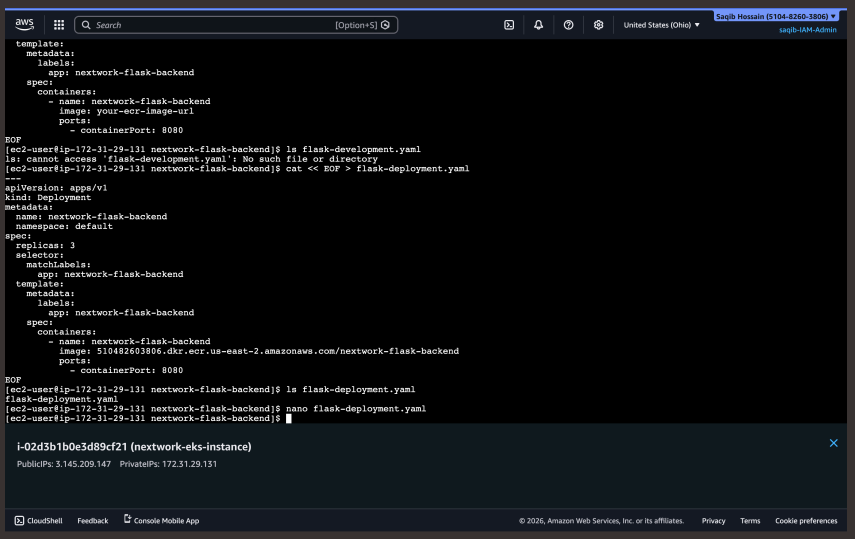
Kubernetes manifests are YAML files that define what you want Kubernetes to do — like which containers to run, how many replicas to have, and how to expose them. Manifests are helpful because they let you describe your entire deployment in a file that's easy to reuse, version control, and share, instead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

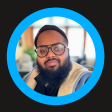
A Deployment manifest manages how my app runs on the cluster — things like which image to use, how many replicas to maintain, and how to handle updates or crashes. The container image URL in my Deployment manifest tells Kubernetes where to pull the app from, which in my case is my ECR repository.

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




saqibh49@gmail.com
NextWork Student

Backend Deployment!

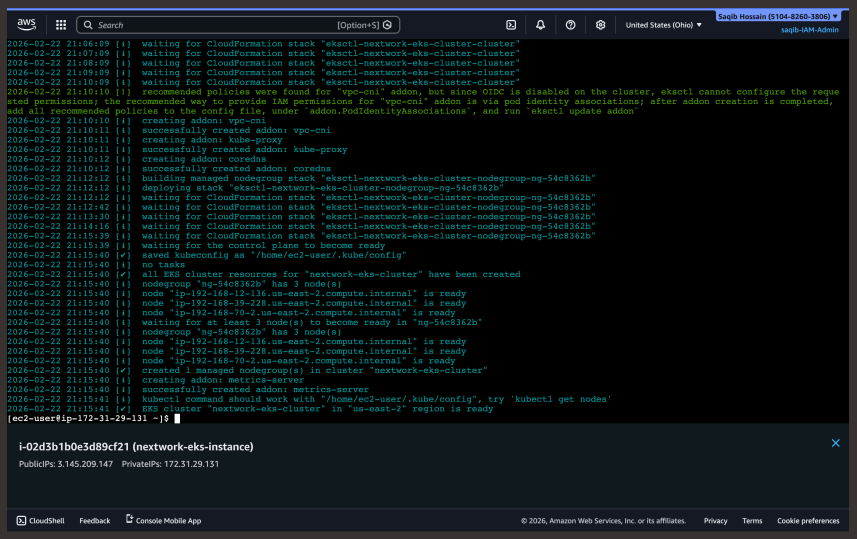
To deploy my backend application, I used kubectl to apply my Deployment and Service manifest files to the cluster. This told Kubernetes to pull my container image from ECR, spin up the pods, and expose the app through a NodePort so it's accessible from outside the cluster.

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

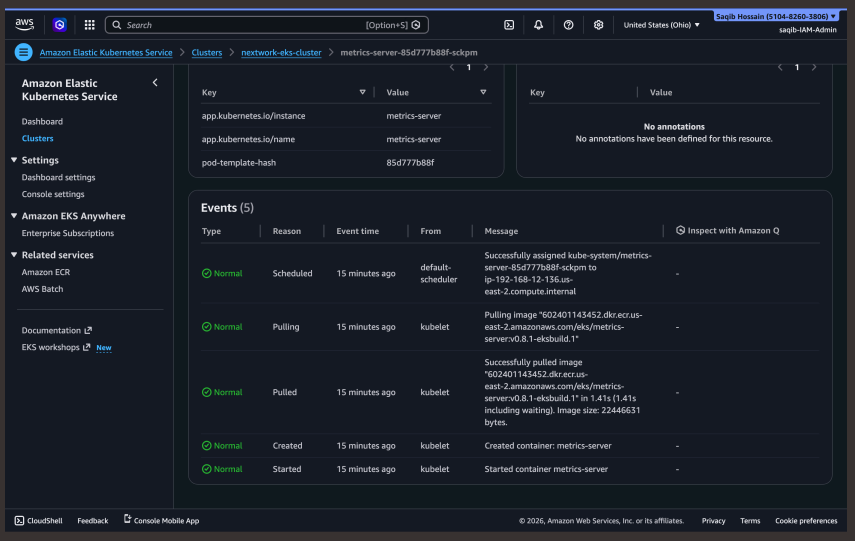


saqibh49@gmail.com
NextWork Student

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.



nextwork.org

The place to learn & showcase your skills

Check out nextwork.org for more projects