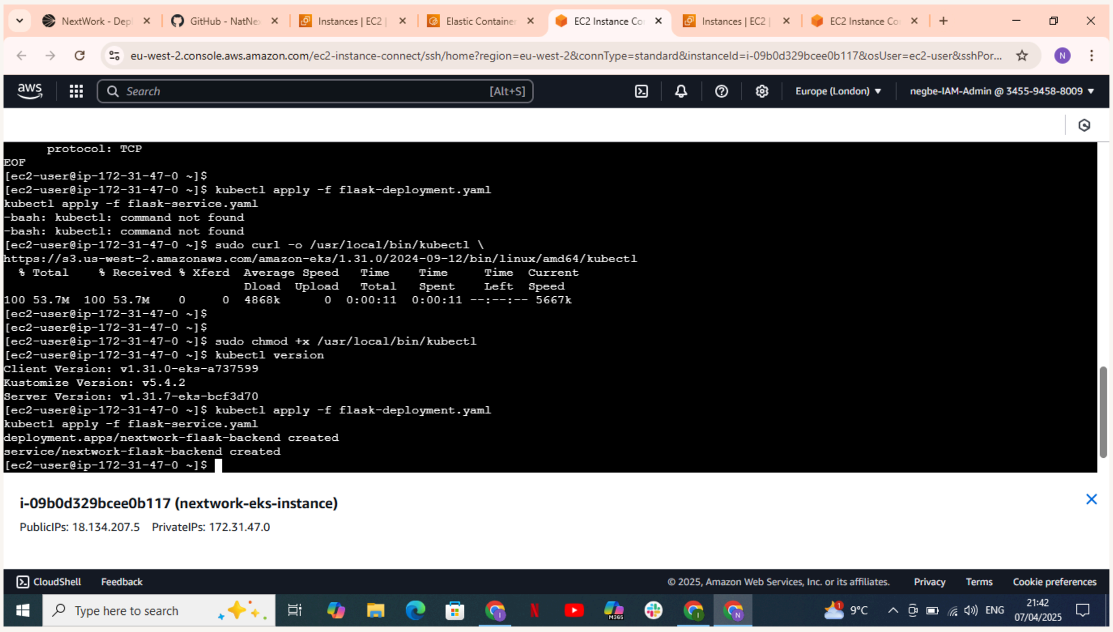
nextwork.org

Deploy Backend with Kubernetes

N Negbe Pierre


eu-west-2.console.aws.amazon.com/ec2-instance-connect/ssh/home?region=eu-west-2&connType=standard&instanceId=i-09b0d329bcee0b117&osUser=ec2-user&sshPor...
Europe (London)
negbe-IAM-Admin @ 3455-9458-8009
CloudShell
Feedback

```
     protocol: TCP
EOF
[ec2-user@ip-172-31-47-0 ~]$
[ec2-user@ip-172-31-47-0 ~]$ kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
-bash: kubectl: command not found
-bash: kubectl: command not found
[ec2-user@ip-172-31-47-0 ~]$ sudo curl -o /usr/local/bin/kubectl \
https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.0/2024-09-12/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  4868k      0  0:00:11  0:00:11 --:--:-- 5667k
[ec2-user@ip-172-31-47-0 ~]$
[ec2-user@ip-172-31-47-0 ~]$
[ec2-user@ip-172-31-47-0 ~]$ sudo chmod +x /usr/local/bin/kubectl
[ec2-user@ip-172-31-47-0 ~]$ kubectl version
Client Version: v1.31.0-eks-a737599
Kustomize Version: v5.4.2
Server Version: v1.31.7-eks-bcf3d70
[ec2-user@ip-172-31-47-0 ~]$ kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
deployment.apps/nextwork-flask-backend created
service/nextwork-flask-backend created
[ec2-user@ip-172-31-47-0 ~]$
```

i-09b0d329bcee0b117 (nextwork-eks-instance)

PublicIPs: 18.134.207.5 PrivateIPs: 172.31.47.0

© 2025, Amazon Web Services, Inc. or its affiliates.

Introducing Today's Project!

In this project, I will deploy the backend of an application on a Kubernetes cluster using AWS EKS because it simulates a real-world DevOps workflow and demonstrates how to manage containerized applications in a scalable, production-ready environment. Here’s what we’re doing and why: Set up the backend for deployment Because this is the core of the app that will run on the cloud infrastructure. Install kubectl (Kubernetes command-line tool) Because we use it to interact with the Kubernetes cluster — to deploy, inspect, and manage applications. Deploy the backend using EKS (Elastic Kubernetes Service) Because EKS is a managed Kubernetes service on AWS, making it easier to run Kubernetes without managing the control plane. Track the deployment Because monitoring and troubleshooting are essential to ensure that the app is running as expected in the cluster. This project ties together GitHub, Docker, EC2, ECR, and Kubernetes giving you hands-on practice

Tools and concepts

I used Kubernetes, ECR, kubectl, and eksctl to containerize and deploy an application using Amazon EKS. Key concepts include using manifests (Deployment and Service) to define how apps should run and be exposed, storing container images in ECR for easy cluster-wide access, and managing the deployment lifecycle with kubectl. I also configured IAM roles to grant permissions and validated deployment through the EKS console.

Project reflection

2hours

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 serve as the control center where Kubernetes manages containerized applications — specifically the backend of our app. It handles critical tasks like scheduling, scaling, networking, and updating containers. Once the backend application is deployed, the EKS cluster ensures it runs smoothly across nodes (groups of EC2 instances) and stays highly available and responsive to traffic and performance needs.

Backend code

I retrieved backend code by cloning a GitHub repository that contains the backend logic for the application. Pulling code is essential to this deployment because it provides the server-side functionality that the application needs to process user requests, manage data, and connect to databases or other services. Without this code, there would be nothing to containerize, push to ECR, or deploy on the EKS cluster.

Container image

Once I cloned the backend code, I built a container image because Kubernetes doesn't work with raw source code—it needs a pre-built container image to run the app consistently across different environments. Without an image, it would be difficult for Kubernetes to create and manage multiple identical copies (pods) of the backend application.

I also pushed the container image to a container registry, which is Amazon ECR (Elastic Container Registry), because ECR provides a secure and scalable location to store my Docker images. This makes it easy for Kubernetes (via EKS) to pull the image whenever it needs to create or update pods. ECR facilitates scaling for my deployment because all nodes in my Kubernetes cluster can consistently pull the latest version of the image from a centralized location, ensuring my app runs the same way across different environments and instances

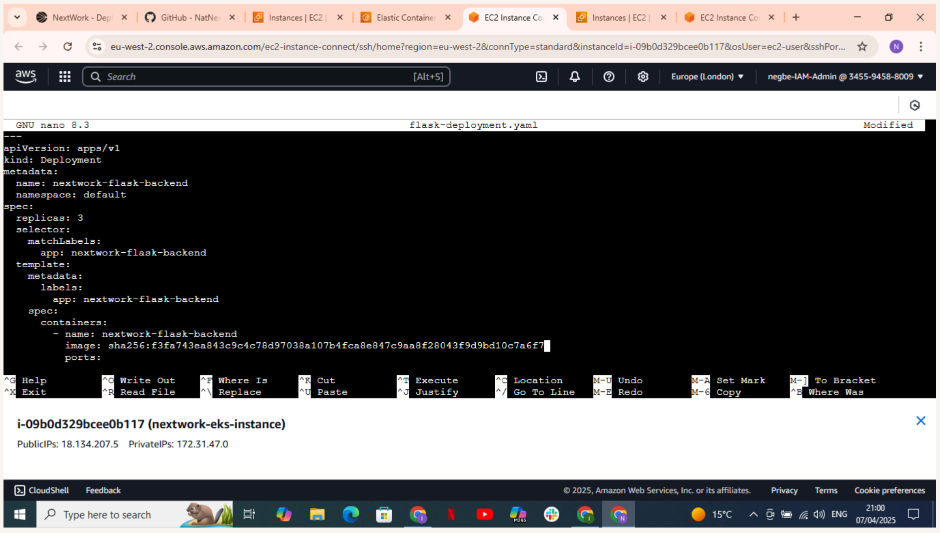
Manifest files

Kubernetes manifests are a set of instructions that tell Kubernetes how to run your app. Manifests are helpful because they describe everything Kubernetes needs to know—such as which container image to use, how many replicas to create, and how much memory to allocate. Without manifests, you'd have to manually configure each deployment every time, which would be confusing and error-prone. Manifests make the deployment process repeatable, consistent, and easier to manage.

A Deployment manifest manages how many copies (pods) of an application should run, and ensures they stay running. It defines the desired state of your app—for example, which container image to use, how many replicas to create, and how to update them. The container image URL in my Deployment manifest tells Kubernetes exactly where to pull the backend app from—usually from a container registry like Amazon ECR—so it can deploy the right version of the app across the cluster

A Service resource exposes an application to internal or external traffic, making it reachable across the network. My Service manifest sets up a NodePort type service that routes external traffic on port 8080 to the app running on the Kubernetes cluster. This ensures users can access the backend even from outside the cluster by connecting through the exposed port

Negbe Pierre
NextWork Student



eu-west-2.console.aws.amazon.com/ec2-instance-connect/ssh/home?region=eu-west-2&connType=standard&instanceId=i-09b0d329bcee0b117&osUser=ec2-user&sshPor...

```
  GNU nano 8.3                    flask-deployment.yaml                    Modified
---
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: sha256:f3fa743ea843c9c4c78d97038a107b4fca8e847c9aa8f28043f9d9bd10c7a6f7
          ports:
^G Help       ^O Write Out  ^F Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^B Where Was
```

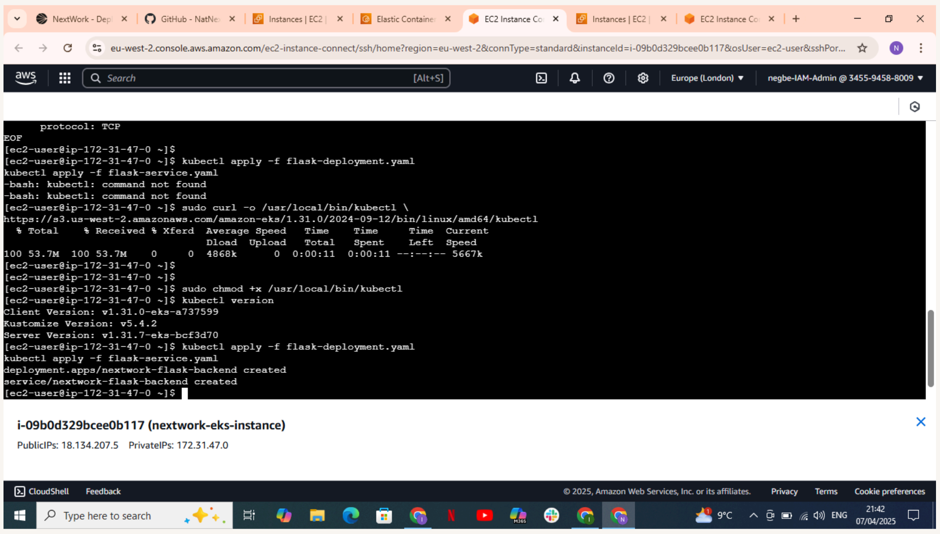
i-09b0d329bcee0b117 (nextwork-eks-instance)
PublicIPs: 18.134.207.5 PrivateIPs: 172.31.47.0

Backend Deployment!

To deploy my backend application, I first installed the kubectl CLI tool since it wasn’t pre-installed. Then, I used kubectl apply -f flask-deployment.yaml to deploy my backend container and kubectl apply -f flask-service.yaml to expose it. These manifest files told Kubernetes how to run and expose my app on the EKS cluster.

kubectl

kubectl is the command-line tool for interacting with Kubernetes resources like Deployments and Services. I need this tool to apply my manifest files and manage the app once the cluster is running. I can't use eksctl for the job because eksctl is mainly for setting up or deleting the EKS cluster, not for managing or deploying apps within it.



```
     protocol: TCP
EOF
[ec2-user@ip-172-31-47-0 ~]$
[ec2-user@ip-172-31-47-0 ~]$ kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
-bash: kubectl: command not found
-bash: kubectl: command not found
[ec2-user@ip-172-31-47-0 ~]$ sudo curl -o /usr/local/bin/kubectl \
https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.0/2024-09-12/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  4868k      0  0:00:11  0:00:11 --:--:-- 5667k
[ec2-user@ip-172-31-47-0 ~]$
[ec2-user@ip-172-31-47-0 ~]$
[ec2-user@ip-172-31-47-0 ~]$ sudo chmod +x /usr/local/bin/kubectl
[ec2-user@ip-172-31-47-0 ~]$ kubectl version
Client Version: v1.31.0-eks-a737599
Kustomize Version: v5.4.2
Server Version: v1.31.7-eks-bcf3d70
[ec2-user@ip-172-31-47-0 ~]$ kubectl apply -f flask-deployment.yaml
kubectl apply -f flask-service.yaml
deployment.apps/nextwork-flask-backend created
service/nextwork-flask-backend created
[ec2-user@ip-172-31-47-0 ~]$
```

i-09b0d329bcee0b117 (nextwork-eks-instance)

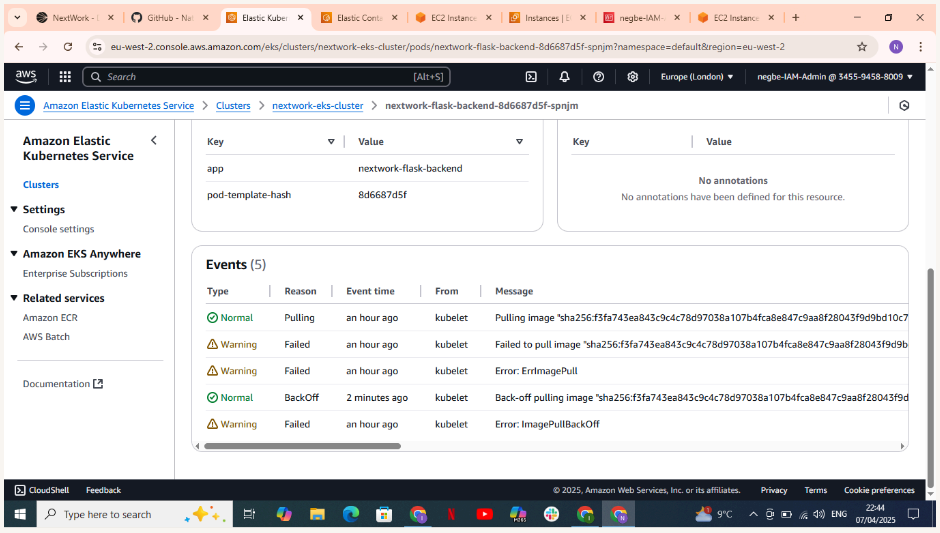
PublicIPs: 18.134.207.5 PrivateIPs: 172.31.47.0

Verifying Deployment

My extension for this project is to use the EKS console to investigate and verify the progress of my backend deployment visually. I had to set up IAM access policies because Amazon EKS requires proper authentication to allow users to interact with the cluster and view resources like nodes, deployments, and services. I set up access by running the eksctl create iamidentitymapping command with my IAM user ARN and region to grant myself admin-level permissions

Once I gained access into my cluster's nodes, I discovered pods running inside each node. Pods are the smallest deployable units in Kubernetes and they bundle one or more containers together so they can operate as a single unit. You can't deploy containers on their own—they must be in a pod. Containers in a pod share the same network space and storage, which allows them to communicate and exchange data more efficiently.

The EKS console shows you the events for each pod, where I could see that the pod was trying to pull the container image but failed. It displayed errors like ErrImagePull and ImagePullBackOff, meaning Kubernetes couldn't fetch the image from the registry. This validated that there was an issue with the image URL or permissions in my deployment manifest, confirming that the backend wasn't running successfully due to image pull errors.



Amazon Elastic Kubernetes Service > Clusters > nextwork-eks-cluster > nextwork-flask-backend-8d6687d5f-spnjm

Key	Value
app	nextwork-flask-backend
pod-template-hash	8d6687d5f

Key	Value

No annotations
No annotations have been defined for this resource.

Events (5)

Type	Reason	Event time	From	Message
Normal	Pulling	an hour ago	kubelet	Pulling image "sha256:f3fa743ea843c9c4c78d97038a107b4fca8e847c9aa8f28043f9d9bd10c7
Warning	Failed	an hour ago	kubelet	Failed to pull image "sha256:f3fa743ea843c9c4c78d97038a107b4fca8e847c9aa8f28043f9d9b
Warning	Failed	an hour ago	kubelet	Error: ErrImagePull
Normal	BackOff	2 minutes ago	kubelet	Back-off pulling image "sha256:f3fa743ea843c9c4c78d97038a107b4fca8e847c9aa8f28043f9d
Warning	Failed	an hour ago	kubelet	Error: ImagePullBackOff

eu-west-2.console.aws.amazon.com/eks/clusters/nextwork-eks-cluster/pods/nextwork-flask-backend-8d6687d5f-spnjm?namespace=default&region=eu-west-2
Europe (London)
negbe-IAM-Admin @ 3455-9458-8009
Amazon Elastic Kubernetes Service
Clusters
Settings
Console settings
Amazon EKS Anywhere
Enterprise Subscriptions
Related services
Amazon ECR
AWS Batch
Documentation
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Everyone should be in a job they love.

Check out nextwork.org for more projects