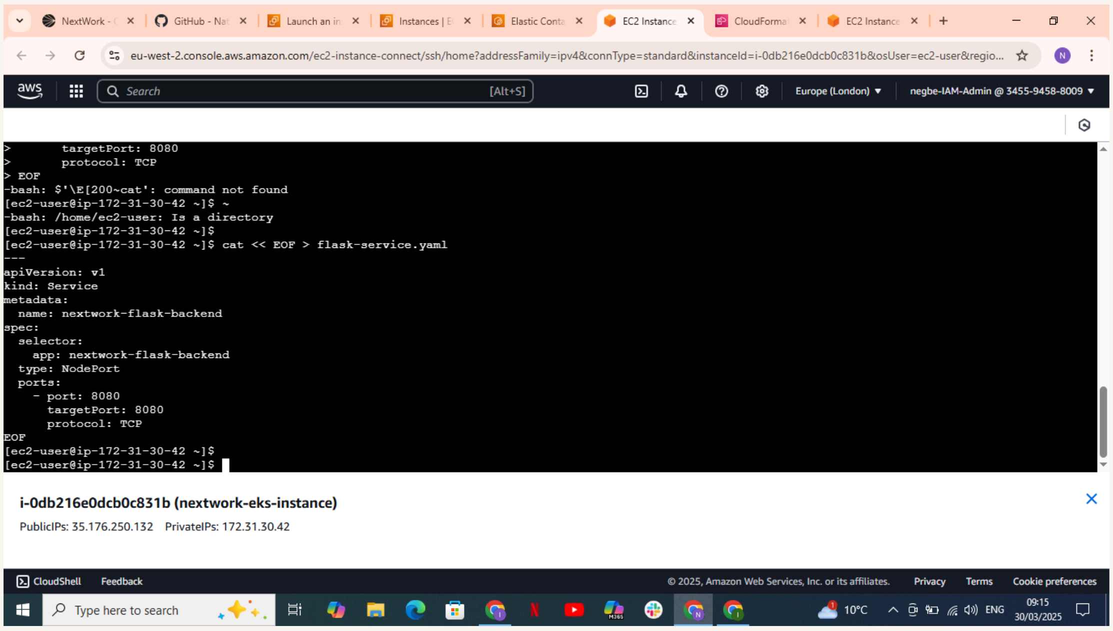
nextwork.org

Create Kubernetes Manifests

Negbe Pierre



eu-west-2.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-0db216e0dcb0c831b&osUser=ec2-user&regio...
Europe (London)
negbe-IAM-Admin @ 3455-9458-8009

```
>       targetPort: 8080
>       protocol: TCP
> EOF
-bash: $'\E[200~cat': command not found
[ec2-user@ip-172-31-30-42 ~]$ ~
-bash: /home/ec2-user: Is a directory
[ec2-user@ip-172-31-30-42 ~]$
[ec2-user@ip-172-31-30-42 ~]$ cat << EOF > flask-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: nextwork-flask-backend
spec:
  selector:
    app: nextwork-flask-backend
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
EOF
[ec2-user@ip-172-31-30-42 ~]$
[ec2-user@ip-172-31-30-42 ~]$
```

i-0db216e0dcb0c831b (nextwork-eks-instance)

PublicIPs: 35.176.250.132 PrivateIPs: 172.31.30.42

Introducing Today's Project!

In this project, I will create the key Kubernetes manifest files specifically a Deployment manifest and a Service manifest—because these files tell Kubernetes how to deploy and manage my containerized backend. After learning how to build and push a Docker image of the backend to Amazon ECR and how Kubernetes pulls that image for deployment, this next step is all about giving Kubernetes the actual instructions it needs to run the app. The Deployment manifest will define how many replicas (copies) of the backend to run and how to update them, ensuring availability and reliability. The Service manifest will expose the backend so it can receive traffic making it accessible to users or other services. These YAML configuration files are essential for controlling how the backend behaves once deployed in the cluster. By the end of this project, I'll have the foundation for a fully automated and managed Kubernetes deployment using AWS tools like EKS, ECR, and CloudFormation along with GitHub

Tools and concepts

I used Amazon EKS, eksctl, EC2, Docker, and ECR to set up a Kubernetes environment, build and store a container image, and deploy an application to a managed Kubernetes cluster. Key concepts include using manifests like Deployment and Service YAML files to tell Kubernetes how to run and expose applications, pushing Docker images to a container registry for accessibility, and creating replicas of Pods to ensure reliability and scalability.

Project reflection

to learn more on kurbenates

2 hours

Project Set Up

Kubernetes cluster

To set up today’s project, I launched a Kubernetes cluster. Steps I took to do this included spinning up an EC2 instance on AWS to serve as my control machine, installing eksctl to manage Kubernetes resources, and configuring AWS credentials on the instance. I then used eksctl to create the Kubernetes cluster in the eu-west-2 region, defining the cluster name, nodegroup, instance type, node count, and Kubernetes version to complete the setup.

Backend code

I retrieved the backend that I plan to deploy by cloning a GitHub repository using Git. This repository contains all the necessary backend code, including the application logic, Dockerfile, and dependencies listed in the requirements file. Backend is the part of the application that handles requests, processes data, and communicates with external services, ensuring the app behaves correctly behind the scenes

Container image

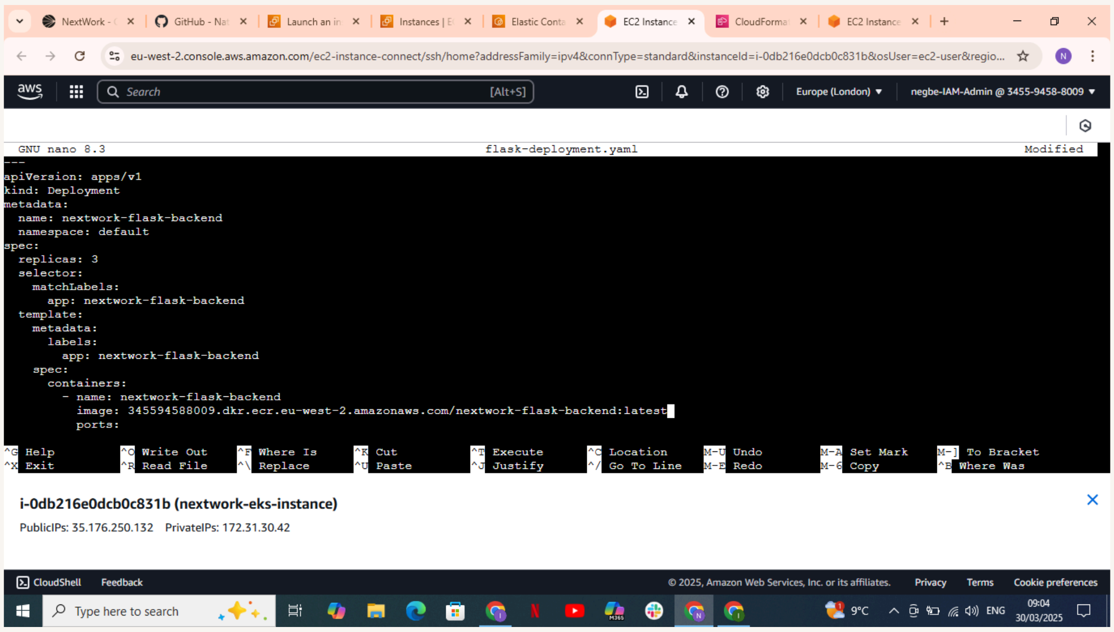
Once I cloned the backend code, I built a container image because Kubernetes needs an image to deploy applications. I used Docker to package the backend code, dependencies, and configuration into a single container image. I made sure Docker was running, then ran the docker build command from within the backend project directory. This created a reusable image that can run consistently across any environment Kubernetes assigns it to

I also pushed the container image to a container registry, because Kubernetes needs a place to pull the image from during deployment. To push the image to ECR, I created a repository in Amazon ECR, tagged the Docker image using the repository URI, authenticated Docker to ECR using the AWS CLI, and then pushed the image. This ensures the image is accessible by Kubernetes and helps with consistent deployments.

Manifest files

Kubernetes manifests are configuration files written in YAML or JSON that tell Kubernetes how to deploy and manage applications in a cluster. Manifests are helpful because they define the desired state of resources like deployments, services, and pods—such as which container image to run, how many replicas to use, and how to expose the app—making deployments consistent, repeatable, and easier to manage.

A Kubernetes deployment manages how many copies (replicas) of an application should be running and ensures they stay available and up to date. It handles rolling updates, restarts failed pods, and maintains the desired application state. The container image URL in my Deployment manifest tells Kubernetes exactly which image to use—in this case, from Amazon ECR—so it can pull and run the right version of the app on each pod



```
  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: 345594588009.dkr.ecr.eu-west-2.amazonaws.com/nextwork-flask-backend:latest
          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
```

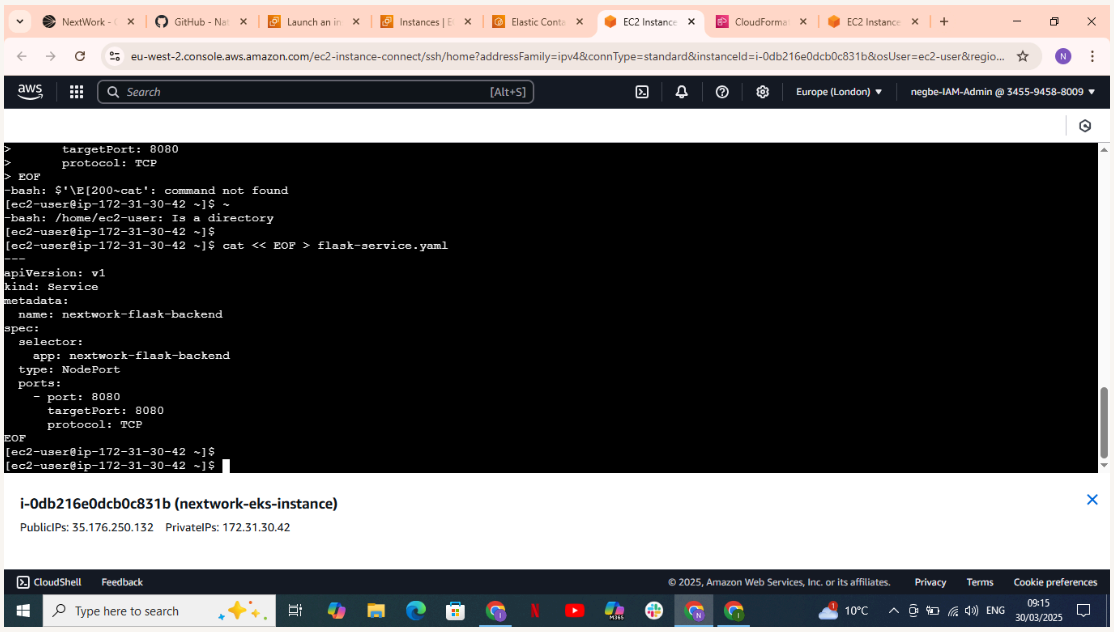
i-0db216e0dcb0c831b (nextwork-eks-instance)

PublicIPs: 35.176.250.132   PrivateIPs: 172.31.30.42

Service Manifest

A Kubernetes Service exposes a set of pods to the network so that other apps, users, or services can access them. You need a Service manifest to tell Kubernetes how to direct traffic to your app. It defines important details like the type of service (e.g., NodePort), the target port on the pod, and the port that users or other apps will use to connect. Without it, your app would run but not be reachable from outside the cluster

My Service manifest sets up a NodePort service that makes my application accessible externally. It targets pods labeled nextwork-flask-backend and routes traffic on port 8080 using the TCP protocol. This setup tells Kubernetes how to expose my app and send requests to the correct container running in the cluster.



```
>       targetPort: 8080
>       protocol: TCP
> EOF
-bash: $'\E[200~cat': command not found
[ec2-user@ip-172-31-30-42 ~]$ ~
-bash: /home/ec2-user: Is a directory
[ec2-user@ip-172-31-30-42 ~]$
[ec2-user@ip-172-31-30-42 ~]$ cat << EOF > flask-service.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: nextwork-flask-backend
spec:
  selector:
    app: nextwork-flask-backend
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
EOF
[ec2-user@ip-172-31-30-42 ~]$
[ec2-user@ip-172-31-30-42 ~]$
```

i-0db216e0dcb0c831b (nextwork-eks-instance)

PublicIPs: 35.176.250.132   PrivateIPs: 172.31.30.42

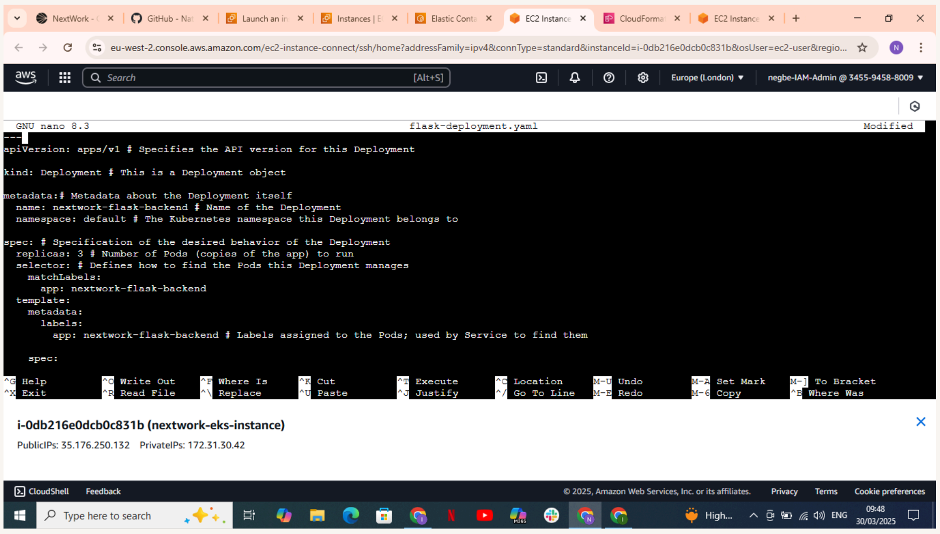
Deployment Manifest

Annotating the Deployment manifest helped me understand how Kubernetes deployments work because it broke down each line and showed me what it does. By adding comments, I could clearly see how the replicas field controls how many Pods run, how the selector and labels connect the Deployment to the right Pods, and how the container image is pulled from ECR. This made the structure and purpose of each section easier to follow, and it gave me a better understanding of how Kubernetes manages and scales applications behind the scenes.

A notable line in the Deployment manifest is the number of replicas, which means how many identical copies of the application should be running at all times in the cluster. This helps ensure availability—if one instance fails, others are still running to handle traffic. Pods are relevant to this because each replica runs inside a Pod. A Pod is the smallest deployable unit in Kubernetes and contains one or more containers. So when I set replicas: 3, Kubernetes will launch 3 separate Pods, each running my backend container to distribute the load and improve reliability.

Negbe Pierre
NextWork Student

One part of the Deployment manifest I still want to know more about is the selector and how it matches with the labels inside the template, because while I understand that they help connect the Deployment to the correct Pods, I'm still a little unsure about how strict or flexible that matching process is. I'd also like to know what happens if the labels don't match exactly — does the Deployment fail, or does Kubernetes try to fix it somehow?


eu-west-2.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-0db216e0dcb0c831b&osUser=ec2-user&regio...
Europe (London)
negbe-IAM-Admin @ 3455-9458-8009

```
  GNU nano 8.3                    flask-deployment.yaml                    Modified
---
apiVersion: apps/v1 # Specifies the API version for this Deployment

kind: Deployment # This is a Deployment object

metadata:# Metadata about the Deployment itself
  name: nextwork-flask-backend # Name of the Deployment
  namespace: default # The Kubernetes namespace this Deployment belongs to

spec: # Specification of the desired behavior of the Deployment
  replicas: 3 # Number of Pods (copies of the app) to run
  selector: # Defines how to find the Pods this Deployment manages
    matchLabels:
      app: nextwork-flask-backend
  template:
    metadata:
      labels:
        app: nextwork-flask-backend # Labels assigned to the Pods; used by Service to find them

    spec:
^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-0db216e0dcb0c831b (nextwork-eks-instance)
PublicIPs: 35.176.250.132   PrivateIPs: 172.31.30.42

Everyone should be in a job they love.

Check out nextwork.org for more projects