nextwork.org

Create Kubernetes Manifests

Louis Moyo

```
EOF
[ec2-user@ip-172-31-23-249 manifests]$ nano flask-deployment.yaml
[ec2-user@ip-172-31-23-249 manifests]$ 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-172-31-23-249 manifests]$
```

L
Louis Moyo
NextWork Student

Introducing Today's Project!

In this project, I will create Kubernetes manifest files because they are the instructions that tell Kubernetes how to run and manage my backend app. Specifically, I'll write a Deployment manifest to define how the backend containers should be created and scaled, and a Service manifest to expose the backend so users can access it. This step is essential because manifests turn the container image I built in the last project into a running, accessible app inside my Kubernetes cluster.

Tools and concepts

I used Amazon EKS, eksctl, EC2, Docker, and Amazon ECR to deploy my backend. Key concepts included writing Kubernetes manifests (Deployment and Service files), understanding how Pods and replicas work, and learning how Kubernetes manages applications automatically.

Project reflection

I chose to do this project today because I wanted hands-on experience with Kubernetes deployment on AWS. Something that would make learning with NextWork even better is having more real-world troubleshooting examples, since that's where I learn the most.

L

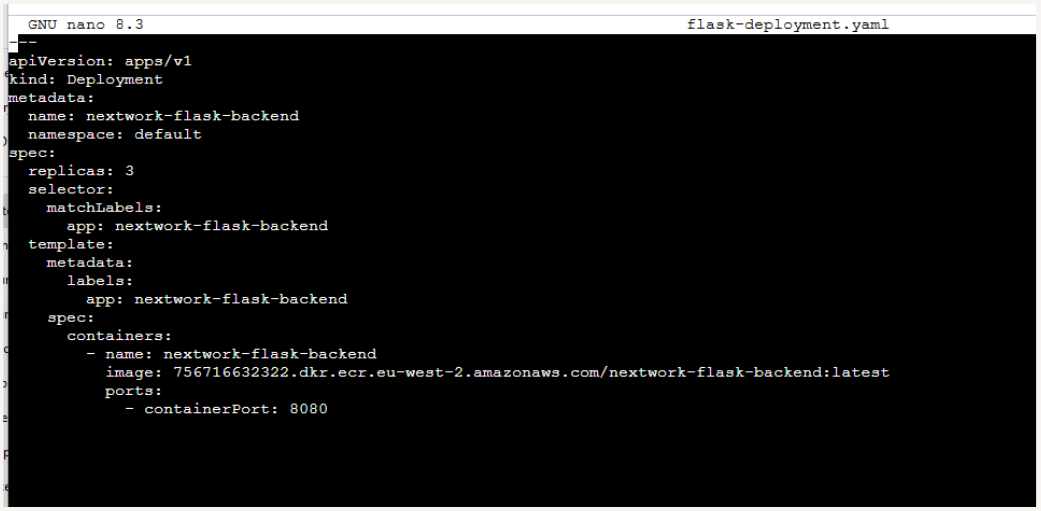
Louis Moyo
NextWork Student

This project took me approximately 1 hour. The most challenging part was troubleshooting ECR repository and region mismatches. My favourite part was creating and annotating the Kubernetes manifests, because it helped me clearly understand how Kubernetes turns instructions into a running app.

Manifest files

Kubernetes manifests are YAML/JSON files that describe the desired state of resources (Pods, Deployments, Services, etc.) in a cluster. They're helpful because they make deployments repeatable and version-controlled, let Kubernetes create/heal/scale your app automatically, and keep configuration clear, auditable, and shareable across environments.

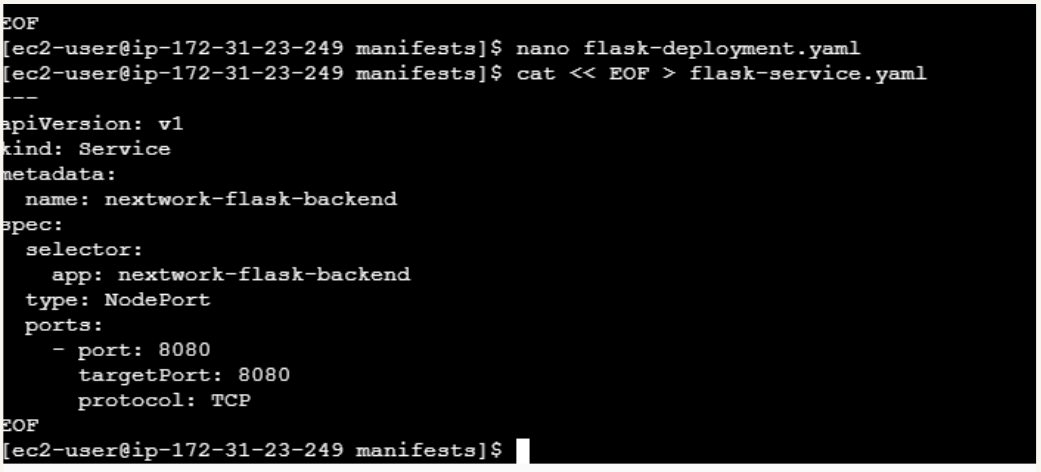
A Kubernetes deployment manages how many copies (pods) of my application should run, ensures they stay healthy, and restarts them if they fail. The container image URL in my Deployment manifest tells Kubernetes which version of my backend to use so it can create and maintain the right pods automatically.


GNU nano 8.3
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: 756716632322.dkr.ecr.eu-west-2.amazonaws.com/nextwork-flask-backend:latest
ports:
- containerPort: 8080

Service Manifest

A Kubernetes Service exposes my backend app so other apps or users can reach it. It makes sure traffic is sent to the right Pods, even if Pods are restarted or replaced. You need a Service manifest to reliably route requests to your app.

My Service manifest sets up a stable endpoint that points to my backend Deployment. It specifies the app label to match Pods, chooses a port to listen on, and ensures requests are routed to the running backend containers.



Deployment Manifest

Annotating the Deployment manifest helped me understand how each section works together to deploy and manage applications in Kubernetes, because writing comments forced me to break down the meaning of apiVersion, kind, metadata, spec, and containers step by step.

A notable line in the Deployment manifest is the number of replicas, which means how many identical Pods Kubernetes should run. Pods are relevant to this because they're the smallest deployable unit in Kubernetes - each replica is a Pod running my backend, and Kubernetes makes sure the right number are always healthy and available.

One part of the Deployment manifest I still want to know more about is how labels and selectors interact in larger projects, because while I see how they connect pods and services here, I'd like to understand how they scale when multiple apps share the same cluster.

```
cat << EOF > flask-deployment.yaml
---
apiVersion: apps/v1   # API version for Deployments
kind: Deployment      # Resource type is Deployment
metadata:
  name: nextwork-flask-backend   # Name of the Deployment
  namespace: default             # Namespace in cluster
spec:                            # Desired state specification
  replicas: 3                    # Number of Pod copies
  selector:                      # Defines which Pods this Deployment manages
    matchLabels:                 # Match Pods by label
      app: nextwork-flask-backend   # Label key/value pair
  template:                      # Pod template definition
    metadata:
      labels:  # Tags for identifying resources
        app: nextwork-flask-backend   # Pod label for selector
    spec:                        # Pod specification
      containers: # List of containers to run in each Pod
```

The place to learn & showcase your skills

Check out nextwork.org for more projects