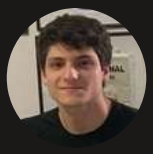
nextwork.org

Deploy Backend with Kubernetes

Nicolás Aversa

```
[ec2-user@ip-172-31-47-51 manifests]$ 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-172-31-47-51 manifests]$
```



Nicolás Aversa
NextWork Student

Introducing Today's Project!

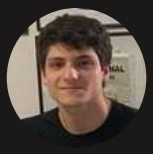
In this project, I will set up the backend of an app for deployment and install kubectl to deploy it on a Kubernetes cluster.

Tools and concepts

I used Kubernetes, ECR, kubectl to deploy a Flask backend on AWS. Key concepts include using manifests (YAML files) to define the app setup, containers to package the app, and kubectl commands to control everything. The system automatically keeps the app running and scales it when needed.

Project reflection

This project took me approximately 2 hours. The most challenging part was understanding what Pods are. My favourite part was when I verified that my backend application was successfully deployed in EKS.



Nicolás Aversa
NextWork Student

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 orchestrate containerized applications, ensuring scalability, high availability, and efficient resource management.

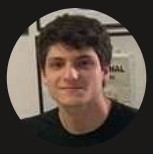
Backend code

I retrieved backend code by configuring Git with my GitHub credentials and cloning the backend application repository. Pulling code is essential to this deployment because I'm copying all the code and resources onto my EC2 instance so I can build, run, and deploy the backend part of my project.

Container image

Once I cloned the backend code, I built a container image because this container image lets Kubernetes set up multiple, identical containers so my application runs consistently across different environments. Without an image, it would be difficult for Kubernetes to distribute, scale, or manage the application.

I also pushed the container image to a container registry, which is a place to securely store, share, and deploy container images. ECR facilitates scaling for my deployment because it provides a fast, reliable, and secure way for Kubernetes to access the exact container image needed to spin up identical instances of my application across multiple nodes in the cluster.



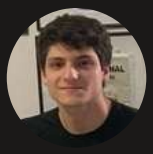
Nicolás Aversa
NextWork Student

Manifest files

Kubernetes manifests are a set of instructions that tells Kubernetes how to run your app. Manifests are helpful because without manifests, Kubernetes wouldn’t know how to set up and manage your app automatically. This means you'd have to manually configure each container every time you deploy, which would be confusing, error-prone, and hard to repeat. Manifests make deployment simple, clear, and consistent.

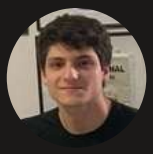
A Deployment manifest manages the desired state of your application within a Kubernetes cluster, defining how your containerized workload should be deployed, updated, and maintained. The container image URL in my Deployment manifest tells Kubernetes where to pull the container image from when it deploys my application.

A Service resource exposes your app to the outside world or other parts of your system. My Service manifest sets up a Service that routes external traffic to port 8080 on the groups of containers labeled nextwork-flask-backend; it makes them accessible from outside the cluster. This Service lets me access my backend using my node's IP and a port number.



Nicolás Aversa
NextWork Student


```
[ec2-user@ip-172-31-42-217 manifests]$ nano flask-deployment.yaml
[ec2-user@ip-172-31-42-217 manifests]$ cat 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: 034362037253.dkr.ecr.sa-east-1.amazonaws.com/nextwork-flask-backend:latest
          ports:
            - containerPort: 8080
[ec2-user@ip-172-31-42-217 manifests]$
```



Nicolás Aversa
NextWork Student

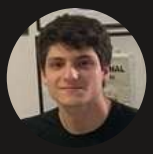
Backend Deployment!

To deploy my backend application, I installed kubectl, run a command to give myself the permission to use kubectl, and run kubectl apply -f flask-deployment.yaml kubectl apply -f flask-service.yaml.

kubectl

kubectl is the command-line tool for interacting with Kubernetes resources (e.g. , Deployment or Service resources) once your cluster is up and running. I need this tool to deploy applications and manage resources within the cluster. I can't use eksctl for the job because eksctl is just for setting up and managing the cluster itself, while kubectl is for managing what runs inside it.

```
[ec2-user@ip-172-31-47-51 manifests]$ 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-172-31-47-51 manifests]$
```



Nicolás Aversa
NextWork Student

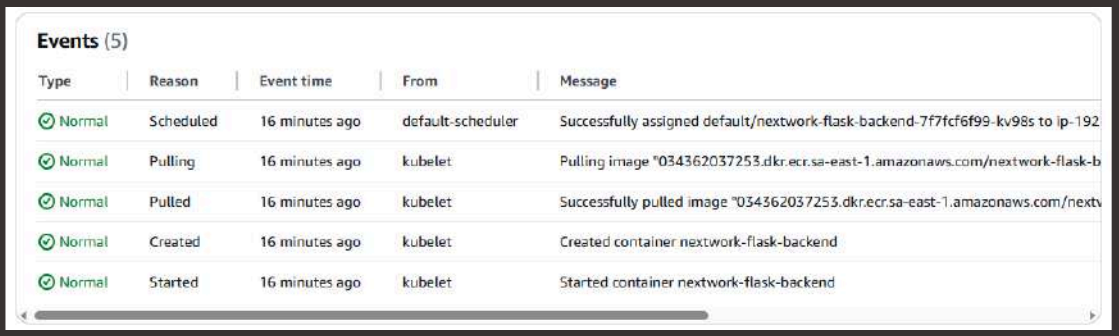
Verifying Deployment

My extension for this project is to use the EKS console to see my cluster nodes in action. I had to set up IAM access policies because AWS permissions alone don't automatically carry over to Kubernetes — Kubernetes has its own way of managing access within a cluster. Even if I have AdministratorAccess in AWS, Kubernetes will only let you into different parts of the cluster if you have permissions under its own system too. I set up access by using a terminal command where I provide my IAM User's ARN.

Once I gained access into my cluster's nodes, I discovered pods running inside each node. Pods are the smallest deployable units in Kubernetes. Containers in a pod share the same network space and storage so they can communicate and share data more efficiently.

The EKS console shows you the events for each pod, where I could see the steps that happened when Kubernetes was creating my pod. This validated that my backend application was successfully deployed.

Nicolás Aversa
NextWork Student



Events (5)

Type	Reason	Event time	From	Message
Normal	Scheduled	16 minutes ago	default-scheduler	Successfully assigned default/nextwork-flask-backend-7f7fcf6f99-kv98s to ip-192
Normal	Pulling	16 minutes ago	kubelet	Pulling image "034362037253.dkr.ecr.sa-east-1.amazonaws.com/nextwork-flask-b
Normal	Pulled	16 minutes ago	kubelet	Successfully pulled image "034362037253.dkr.ecr.sa-east-1.amazonaws.com/nextv
Normal	Created	16 minutes ago	kubelet	Created container nextwork-flask-backend
Normal	Started	16 minutes ago	kubelet	Started container nextwork-flask-backend

NextWork.org

Everyone should be in a job they love.

Check out nextwork.org for more projects