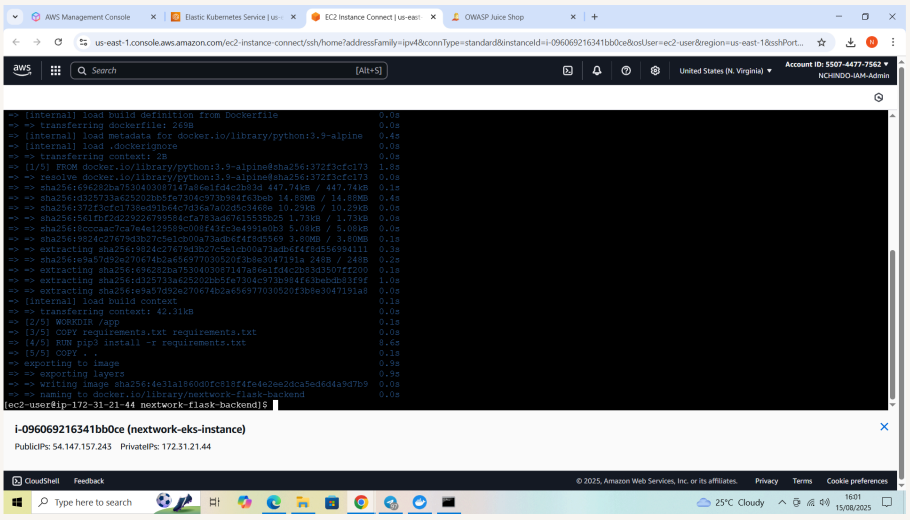
Set Up Kubernetes Deployment

Nchindo Boris



Nchindo Boris
NextWork Student

Introducing Today's Project!

'In this project, I will; - Clone a backend application from GitHub. - Build a Docker image of the backend. - Push your image to an Amazon ECR repository. - Troubleshoot installation and configuration errors

Tools and concepts

I used; - Amazon EKS - Git - Docker and Amazon ECR to set up and deploy the backend of my app. Key steps include; - Set up EC2 and EKS - Use Git to pull the code for the backend of your app. - Build a Docker image of the backend. - Push Docker image to ECR. - Troubleshoot installation and configuration issues.

Project reflection

This project took me 1 hour to complete. The most challenging part was troubleshooting the Docker permission errors and setting up the right IAM roles for the EC2 instance. My favourite part was building the container image and pushing it to Amazon ECR, seeing everything come together and knowing it would run consistently in Kubernetes, it felt good.

placeholder

Something new that I learnt from this experience was how to use eksctl to simplify creating and managing an Amazon EKS cluster. It was surprising to see how much it automates, especially around networking and node management, which would have taken many more manual steps otherwise.

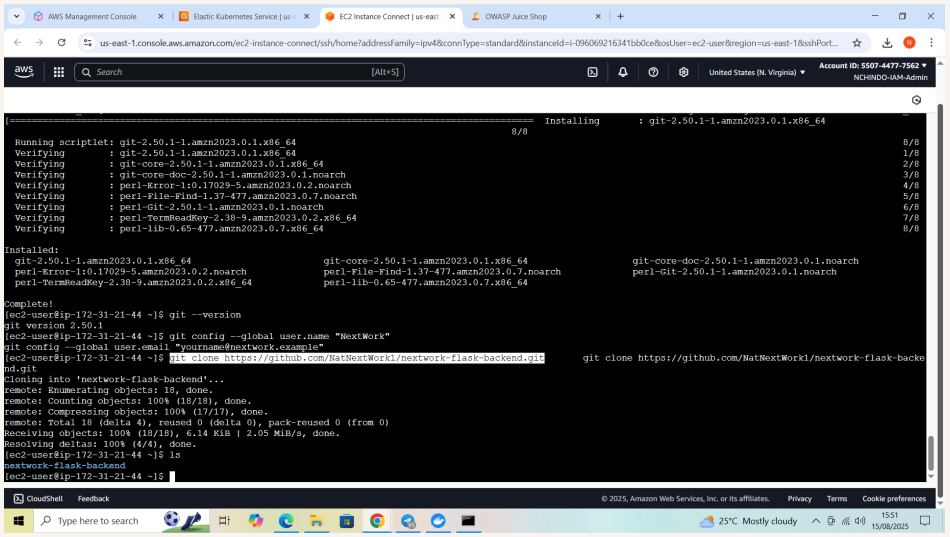
What I'm deploying

To set up today's project, I launched a Kubernetes cluster on AWS. - First, I logged into the AWS Management Console using my IAM Admin account. I then launched an EC2 instance and named it nextwork-eks-instance. - After connecting to the instance via EC2 Instance Connect, I installed eksctl, a tool that makes it easier to create AWS EKS clusters. - Next, I attached an IAM role called nextwork-eks-instance-role to the instance and gave it AdministratorAccess so it could create all the necessary AWS resources. - Finally, I ran a single eksctl command to create the Kubernetes cluster named nextwork-eks-cluster, along with a node group using small t2.micro instances.

I'm deploying an app's backend

Next, I retrieved the backend that I plan to deploy. An app's backend means the part of the application that runs on the server; it handles things like processing requests, connecting to databases, and sending responses to the frontend. I retrieved backend code by first installing Git on my EC2 instance, since it wasn't pre-installed. After setting up Git and configuring my user details, I copied the HTTPS URL from the GitHub repository and ran the git clone command. This downloaded a full copy of the nextwork-flask-backend repository onto my EC2 instance, giving me access to all the backend files I need for deployment.

Nchindo Boris
NextWork Student

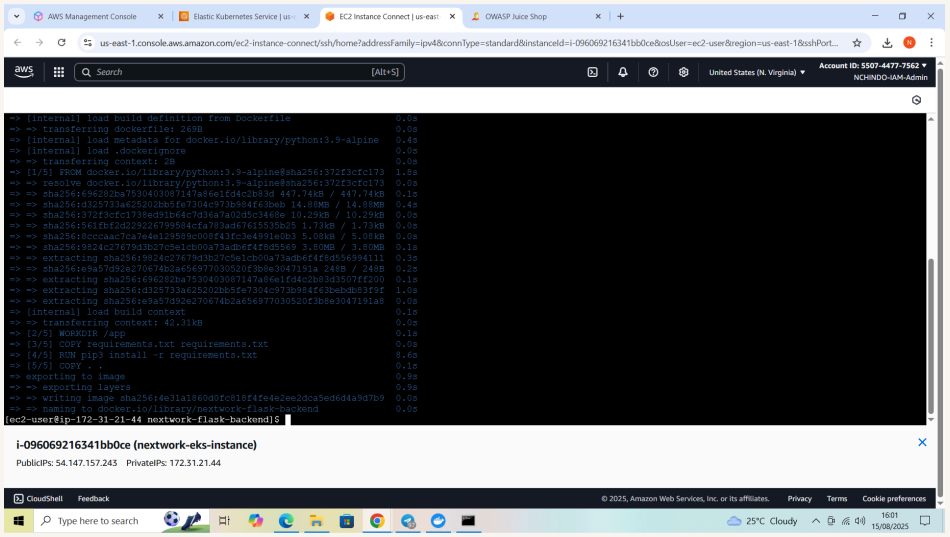


Building a container image

Once I cloned the backend code, my next step was to build a container image of the backend. This is because Kubernetes doesn't run raw code; it runs containers that are created from container images. By building an image using the Dockerfile in the project folder, I'm packaging the backend app along with all of its dependencies so it can run consistently in any environment. This image will later be used by Kubernetes to spin up identical containers across the cluster.

When I tried to build a Docker image of the backend, I ran into a permissions error because the user I'm logged in as, i.e. ec2-user, doesn't have the necessary rights to run Docker commands by default. Docker was installed for the root user, so when I ran the docker build command without sudo, it failed. This happened because Docker requires elevated privileges to build images and manage containers, and my current user didn't have those permissions yet.

To solve the permissions error, I added the ec2-user to the Docker group using the sudo usermod -a -G docker ec2-user command. The Docker group is a special group on Linux systems that gives users permission to run Docker commands without needing to use sudo every time. By adding my user to this group, I gave it the necessary access to build container images and work with Docker directly.

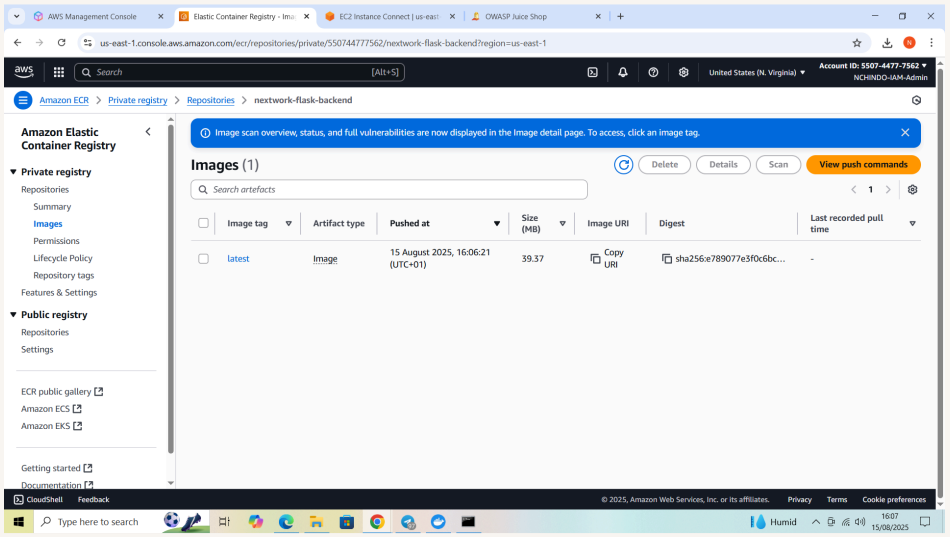

AWS Management Console
Elastic Kubernetes Service | us-
EC2 Instance Connect | us-east
OWASP Juice Shop
us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh/home?addressFamily=ipv4&connType=standard&instanceId=i-096069216341bb0ce&osUser=ec2-user&region=us-east-1&sshPort...
aws
Search
[Alt+S]
United States (N. Virginia)
Account ID: 5507-4477-7562
NCHINDO-IAM-Admin
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 268B 0.0s
=> [internal] load metadata for docker.io/library/python:3.9-alpine 0.4s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/5] FROM docker.io/library/python:3.9-alpine@sha256:372f3cfc173 1.8s
=> => resolve docker.io/library/python:3.9-alpine@sha256:372f3cfc173 0.0s
=> => sha256:696282ba7530403087147a86e1fd4c2b83d 447.74kB / 447.74kB 0.1s
=> => sha256:d325733a625202bb5fe7304c973b984f63beb 14.88MB / 14.88MB 0.4s
=> => sha256:372f3cfc1738ed91b64c7d36a7a02d5c3468e 10.29kB / 10.29kB 0.0s
=> => sha256:561fbf2d223226799584cfa783ad67615535b25 1.73kB / 1.73kB 0.0s
=> => sha256:8cccaac7ca7e4e129589c008f43fc3e4991e0b3 5.08kB / 5.08kB 0.0s
=> => sha256:9824c27679d3b27c5e1cb00a73adb6f4f8d5569 3.80MB / 3.80MB 0.1s
=> => extracting sha256:9824c27679d3b27c5e1cb00a73adb6f4f8d556994111 0.3s
=> => sha256:e9a57d92e270674b2a656977030520f3b8e3047191a 248B / 248B 0.2s
=> => extracting sha256:696282ba7530403087147a86e1fd4c2b83d3507ff200 0.1s
=> => extracting sha256:d325733a625202bb5fe7304c973b984f63bebdb83f9f 1.0s
=> => extracting sha256:e9a57d92e270674b2a656977030520f3b8e3047191a8 0.0s
=> [internal] load build context 0.1s
=> => transferring context: 42.31kB 0.0s
=> [2/5] WORKDIR /app 0.1s
=> [3/5] COPY requirements.txt requirements.txt 0.0s
=> [4/5] RUN pip3 install -r requirements.txt 8.6s
=> [5/5] COPY . . 0.1s
=> exporting to image 0.9s
=> => exporting layers 0.9s
=> => writing image sha256:4e31a1860d0fc818f4fe4e2ee2dca5ed6d4a9d7b9 0.0s
=> => naming to docker.io/library/nextwork-flask-backend 0.0s
[ec2-user@ip-172-31-21-44 nextwork-flask-backend]$
i-096069216341bb0ce (nextwork-eks-instance)
PublicIPs: 54.147.157.243 PrivateIPs: 172.31.21.44
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
25°C Cloudy
16:01
15/08/2025

Nchindo Boris
NextWork Student

Container Registry

I'm using Amazon ECR in this project to securely store and manage the container image of my backend app. ECR is a good choice for the job because it's fully integrated with AWS services like EKS, making it easy for Kubernetes to pull the images without complicated authentication. Plus, it offers features like image scanning for vulnerabilities, which helps keep the deployment secure and reliable.

Container registries like Amazon ECR are great for Kubernetes deployment because they provide a centralized, secure place to store container images that Kubernetes can easily access and pull from. This means the cluster can automatically get the latest version of your app without manually updating each node. Registries also help keep images organised with tags and enable features like vulnerability scanning and encryption, making deployments more consistent, scalable, and secure across all environments.



The place to learn & showcase your skills

Check out nextwork.org for more projects