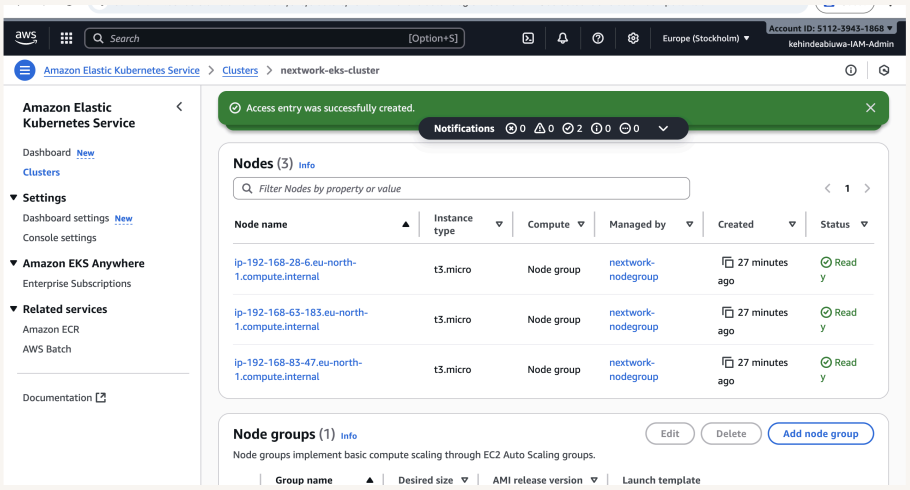
nextwork.org

Launch a Kubernetes Cluster

Kehinde Abiuwa



Kehinde Abiuwa
NextWork Student

Introducing Today's Project!

In this project, I will launch and connect to an EC2 instance, create a Kubernetes cluster, monitor its creation with CloudFormation, access it via an IAM access entry, and run a resilience test because I want hands-on mastery of building, securing, observing, and hardening cloud-native infrastructure on AWS.

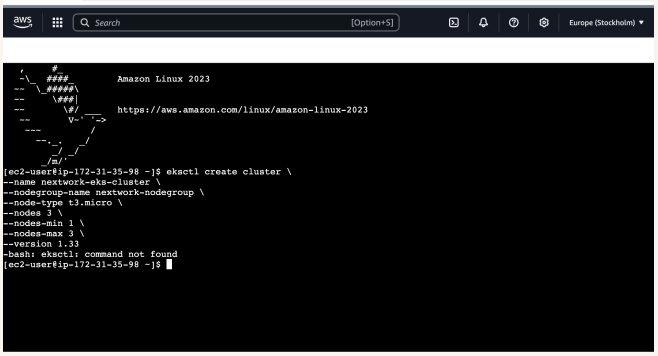
Kehinde Abiuwa
NextWork Student

What is Kubernetes?

Kubernetes is an open-source system that runs lots of containerized apps across many machines and automates the boring-but-critical stuff: where apps run, how they talk, how they scale, how they heal, and how they're updated—without you babysitting each server. Companies and developers use Kubernetes to: Keep apps reliable Scale automatically Ship faster, safer Run anywhere Use hardware efficiently Organize microservices Manage config & secrets Standardize platforms Secure multi-tenant envs Extend operations

I used eksctl to spin up an Amazon EKS cluster from my EC2 box. eksctl took my flags, built CloudFormation stacks, and created the EKS control plane, a managed node group, default VPC networking (since I didn't pass a custom VPC), IAM roles, and a kubeconfig so kubectl can talk to the cluster. The create cluster command I ran defined: Cluster name: nextwork-eks-cluster Region: eu-north-1 Kubernetes version: 1.33 Node group name: nextwork-nodegroup Instance type: t3.micro Capacity: desired 3 nodes, with autoscaling min 1 and max 3

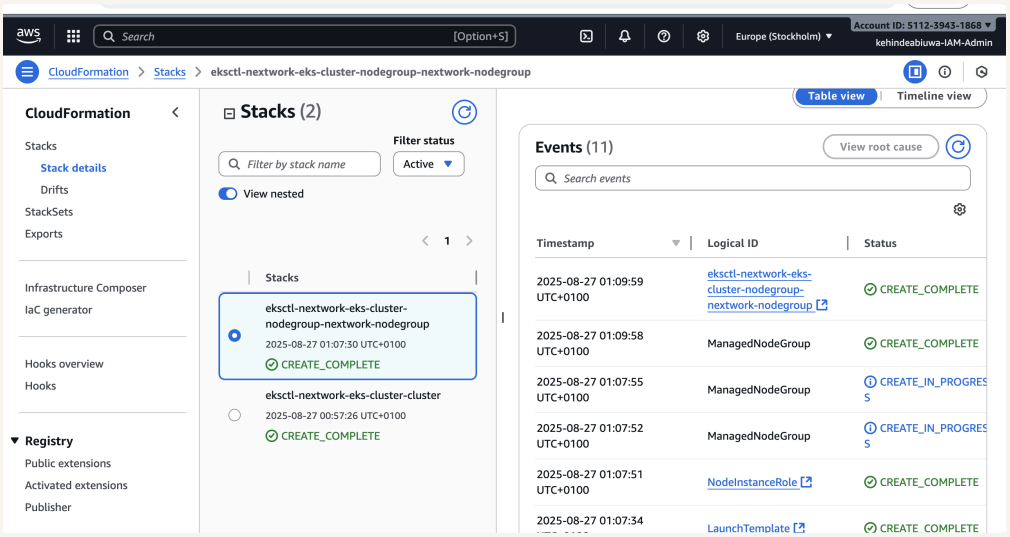
I initially ran into two errors while using eksctl. The first one was because eksctl wasn't installed/on my PATH, so the shell returned eksctl: command not found. I fixed it by downloading the binary, moving it to /usr/local/bin, and verifying with eksctl version. The second one was because my EC2 instance didn't have AWS credentials/permissions (no IAM instance role and/or region set). eksctl couldn't call EKS/CloudFormation/EC2 and would show "Unable to locate credentials" or "AccessDenied." I fixed it by attaching an appropriate IAM role (with EKS/CFN/EC2 + IAM PassRole rights) to the instance, setting the region, confirming with aws sts get-caller-identity, and re-running the command.


aws
Search
[Option+S]
Europe (Stockholm)
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023
[ec2-user@ip-172-31-35-98 ~]$ eksctl create cluster \
--name nextwork-eks-cluster \
--nodegroup-name nextwork-nodegroup \
--node-type t3.micro \
--nodes 3 \
--nodes-min 1 \
--nodes-max 3 \
--version 1.33
-bash: eksctl: command not found
[ec2-user@ip-172-31-35-98 ~]$

eksctl and CloudFormation

CloudFormation helped create my EKS cluster because eksctl builds everything through CloudFormation stacks. Those stacks declare all the AWS pieces the cluster needs—then CloudFormation creates them in order, tracks progress in the Events tab, and rolls back safely if something fails. Concretely, it stood up the EKS control plane, security groups, IAM roles/policies (cluster + node role), and the managed node group (with its launch template and autoscaling). It created VPC resources because I didn't provide an existing VPC/subnets. By default, eksctl asks CloudFormation to make a fresh VPC with public and private subnets across multiple AZs, route tables, an Internet Gateway, NAT Gateway + EIP (so private nodes can reach the internet for images/updates), and the necessary security group rules—giving the cluster sane, production-style networking out of the box.

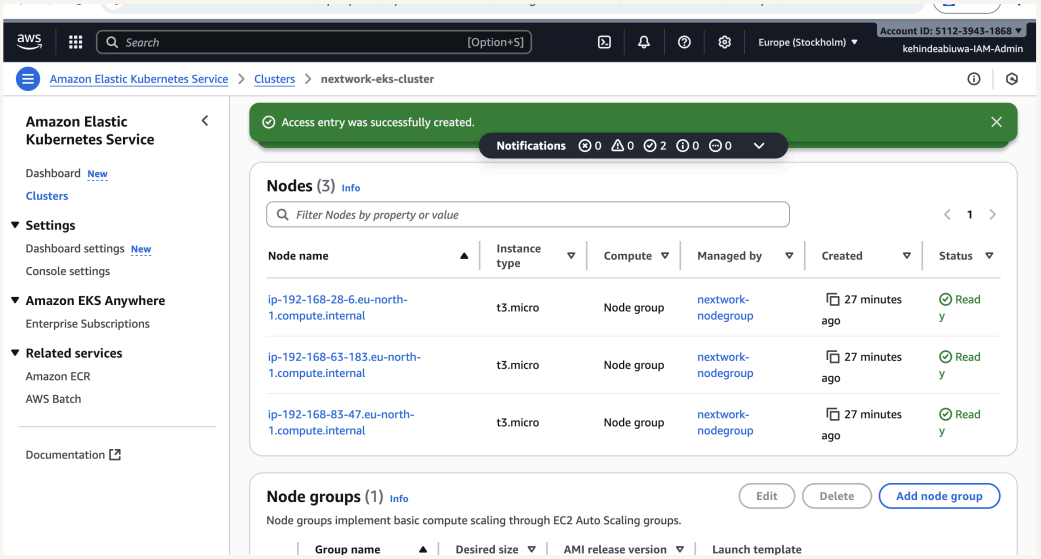
There was also a second CloudFormation stack for the managed node group (nextwork-nodegroup). eksctl always makes two stacks: one for the cluster/control plane and another for each node group. The node-group stack created the EC2 capacity (Auto Scaling group + Launch Template), the NodeInstanceRole (IAM), and the node security group/bootstrapping so the nodes can join the cluster. The difference between a cluster and node group is: Cluster → the EKS control plane (API server, etcd, cluster security groups/roles, endpoint). It can exist by itself but can't run Pods without workers. Node group → a set of worker nodes (EC2 instances) that register to the cluster and run your Pods; you can have many, each with its own instance types/size/spot settings, and you can scale/upgrade/delete them independently of the cluster.


aws
Search
[Option+S]
Europe (Stockholm)
Account ID: 5112-3943-1868
kehindeabiuwa-IAM-Admin
CloudFormation > Stacks > eksctl-nextwork-eks-cluster-nodegroup-nextwork-nodegroup
CloudFormation
Stacks
Stack details
Drifts
StackSets
Exports
Infrastructure Composer
IaC generator
Hooks overview
Hooks
Registry
Public extensions
Activated extensions
Publisher
Stacks (2)
Filter by stack name
Filter status
Active
View nested
Stacks
eksctl-nextwork-eks-cluster-nodegroup-nextwork-nodegroup
2025-08-27 01:07:30 UTC+0100
CREATE_COMPLETE
eksctl-nextwork-eks-cluster-cluster
2025-08-27 00:57:26 UTC+0100
CREATE_COMPLETE
Table view
Timeline view
Events (11)
View root cause
Search events
Timestamp
Logical ID
Status
2025-08-27 01:09:59 UTC+0100
eksctl-nextwork-eks-cluster-nodegroup-nextwork-nodegroup
CREATE_COMPLETE
2025-08-27 01:09:58 UTC+0100
ManagedNodeGroup
CREATE_COMPLETE
2025-08-27 01:07:55 UTC+0100
ManagedNodeGroup
CREATE_IN_PROGRESS
2025-08-27 01:07:52 UTC+0100
ManagedNodeGroup
CREATE_IN_PROGRESS
2025-08-27 01:07:51 UTC+0100
NodeInstanceRole
CREATE_COMPLETE
2025-08-27 01:07:34
LaunchTemplate
CREATE_COMPLETE

The EKS console

I had to create an IAM access entry in order to map my AWS identity (the EC2 instance role) to Kubernetes RBAC, so the cluster would actually authorize my kubectl calls (see nodes, deploy apps, etc.). AWS admin rights don't automatically carry into Kubernetes—that's why the console warned me. An access entry is an EKS-managed link between an IAM user/role and a set of EKS access policies (e.g., cluster-admin, read-only, namespace-scoped). EKS then creates/updates the underlying Kubernetes RBAC and aws-auth mapping for you. I set it up by opening EKS Console → Cluster → Access → Grant access, selecting the IAM role my EC2 uses, choosing a policy (lab: AmazonEKSClusterAdminPolicy; prod: least-privilege and/or namespace scope), and saving. Then I refreshed my EKS cluster console page and I could then see the nodes

It took about 20 minutes to create my cluster. Since I'll create this cluster again in the next project of this series, maybe this process could be sped up if I reuse the same cluster (or an existing VPC), start with a single node or Fargate, and use a supported version—so CloudFormation has less to build and the nodes join faster


aws
Search
[Option+S]
Europe (Stockholm)
Account ID: 5112-3943-1868
kehindeabiuwa-IAM-Admin
Amazon Elastic Kubernetes Service
Clusters
nextwork-eks-cluster
Amazon Elastic Kubernetes Service
Dashboard New
Clusters
Settings
Dashboard settings New
Console settings
Amazon EKS Anywhere
Enterprise Subscriptions
Related services
Amazon ECR
AWS Batch
Documentation
Access entry was successfully created.
Notifications
Nodes (3) Info
Filter Nodes by property or value
Node name
Instance type
Compute
Managed by
Created
Status
ip-192-168-28-6.eu-north-1.compute.internal
t3.micro
Node group
nextwork-nodegroup
27 minutes ago
Ready
ip-192-168-63-183.eu-north-1.compute.internal
t3.micro
Node group
nextwork-nodegroup
27 minutes ago
Ready
ip-192-168-83-47.eu-north-1.compute.internal
t3.micro
Node group
nextwork-nodegroup
27 minutes ago
Ready
Node groups (1) Info
Edit
Delete
Add node group
Node groups implement basic compute scaling through EC2 Auto Scaling groups.
Group name
Desired size
AMI release version
Launch template

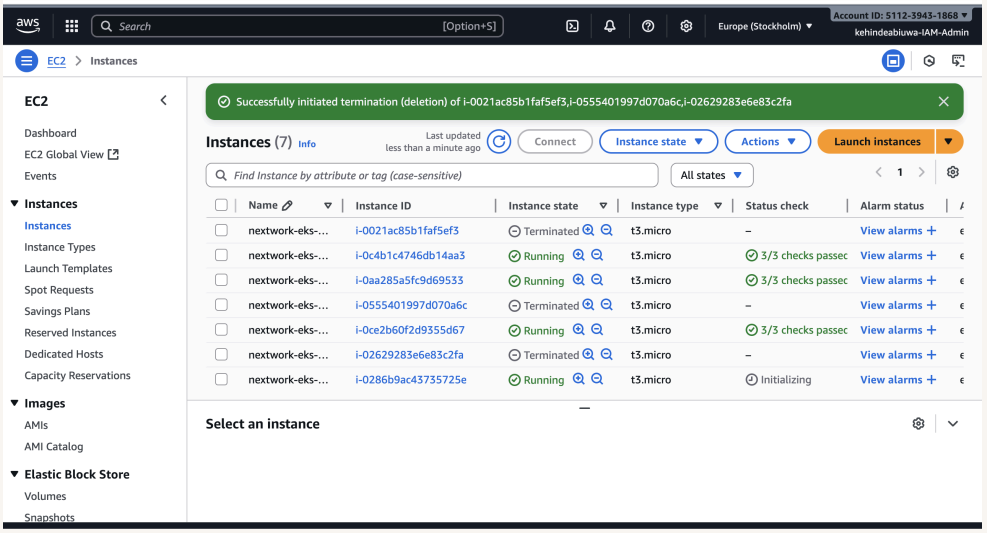
EXTRA: Deleting nodes

Did you know you can find your EKS cluster's nodes in Amazon EC2? This is because when you use managed (or self-managed) node groups, the "nodes" are just regular EC2 instances running the EKS-optimized AMI (or Bottlerocket) in your VPC. EKS manages the control plane for you, but it tells Auto Scaling to launch worker EC2s in your account—so they appear in the EC2 console like any other instance.

Desired size means the current target number of nodes in your EKS managed node group/Auto Scaling group. The ASG keeps launching or terminating EC2s to stay at this number (so if you kill a node, it replaces it to get back to the desired count). Minimum and maximum sizes are helpful for setting the floor and ceiling the group can scale between: Minimum = the baseline you always want running (e.g., enough nodes for HA across AZs and to fit DaemonSets/system pods). Maximum = the cap to prevent runaway scale and control costs/quotas.

When I deleted my EC2 instances, Kubernetes marked those nodes NotReady and the Auto Scaling Group for the managed node group launched replacement EC2 instances to bring the node count back to the desired size. Once the new instances booted and joined the cluster, everything returned to steady state. This is because EKS worker nodes are just EC2 instances managed by an Auto Scaling Group, and Kubernetes controllers keep the pod count at the desired state. If there isn't enough capacity while replacements are starting, some pods will sit Pending briefly.

Kehinde Abiuwa
NextWork Student



The place to learn & showcase your skills

Check out nextwork.org for more projects