Devi Suhitha Chundru

From EC2 to EKS: Automating Kubernetes Cluster Deployment

Launch a Kubernetes Cluster with Amazon EKS

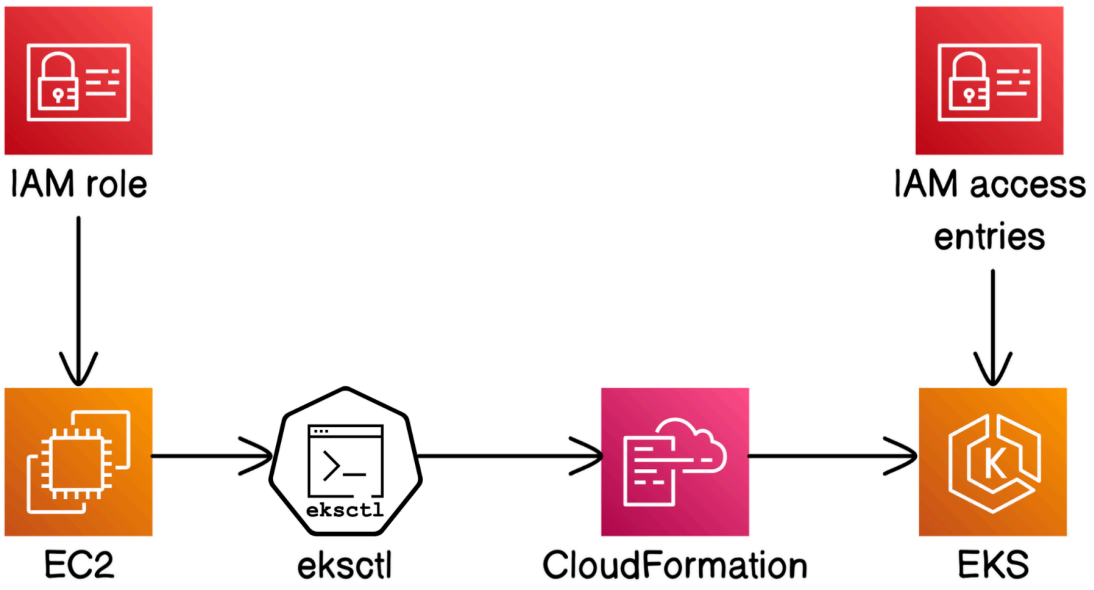
🎯 Project Objective

The objective of this project is to provision and deploy a fully functional Kubernetes cluster on AWS using Amazon EKS, eksctl, and supporting AWS services such as EC2, IAM, and CloudFormation. This hands-on implementation demonstrated the end-to-end process of setting up scalable container orchestration infrastructure by:

- **Launched** and configured an EC2 instance with the necessary IAM role
- **Automated** cluster creation using the eksctl command-line tool
- **Monitored** resource provisioning and stack creation via AWS CloudFormation
- **Set up IAM** access entries to enable secure interaction with the EKS cluster
- **Validated** the deployment and explored the cluster's resilience

Through this project, I developed a solid understanding of how Kubernetes operates within AWS and how to build cloud-native infrastructure using automation and best practices.

Architecture Diagram



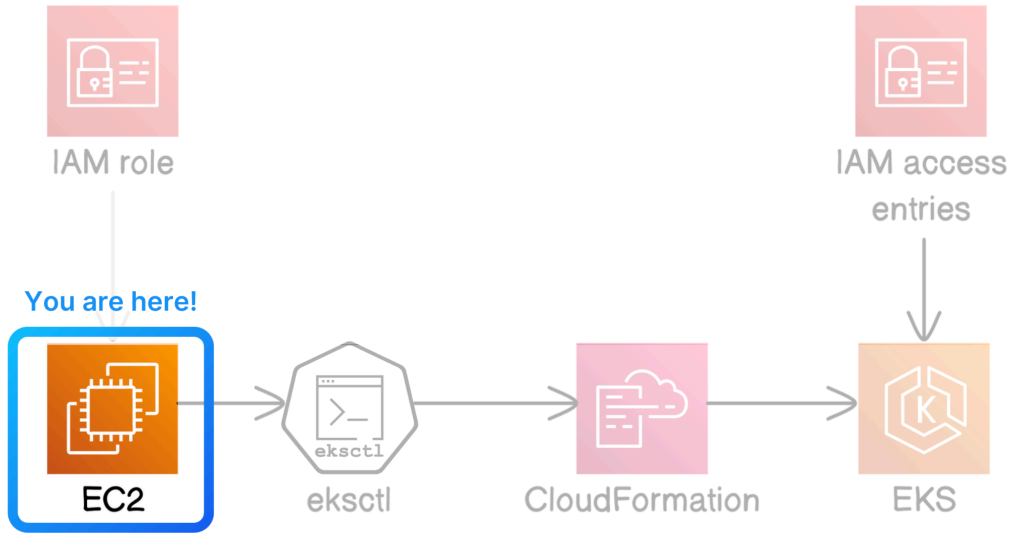
🛠️ Tools & Technologies Used

- **Amazon EKS** – I used Amazon EKS as the managed Kubernetes service to deploy and scale containerized applications.
- **Amazon EC2** – I launched an EC2 instance to serve as the compute environment for running eksctl and managing the cluster setup.
- **AWS CloudFormation** – I utilized AWS CloudFormation to automate the provisioning of required infrastructure components.
- **AWS IAM** – I configured IAM roles and policies to securely manage access to AWS services during the deployment process.
- **Kubernetes** – I worked with Kubernetes as the container orchestration platform to manage workloads and services within the cluster.
- **eksctl** – I used the eksctl CLI tool to automate the creation and configuration of the EKS cluster.
- **kubectl** – I installed and used kubectl to interact with and manage the Kubernetes cluster from the command line.

Step 1: Launched and Connected to an EC2 instance

In this step:

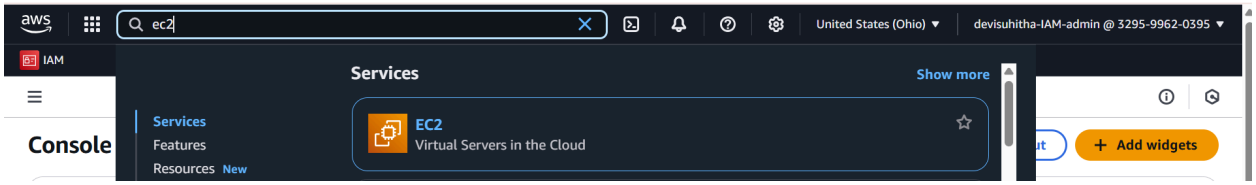
- I launched an EC2 instance and connected to my instance so that I can start sending commands.



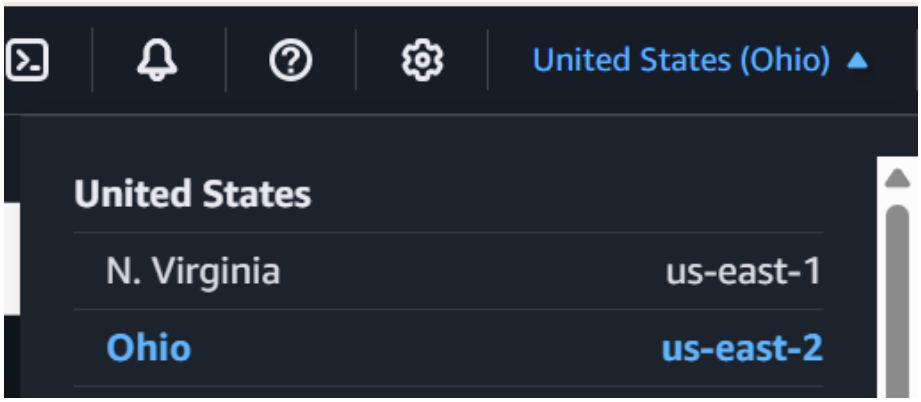
What I am about to build in this step!

Launch an EC2 Instance

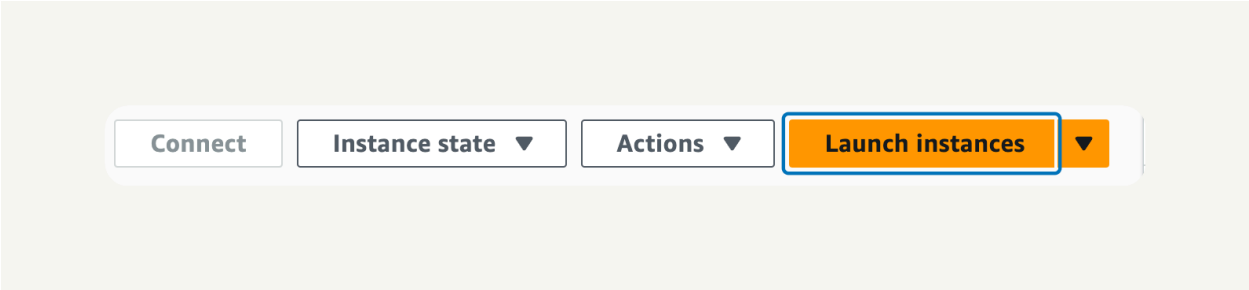
- Logged into my AWS Management Console as my IAM Admin user.
- Headed to EC2 in my AWS Management Console.



Selected **Instances** from the left hand navigation bar and checked that I'm using the **AWS Region** that's closest to me.



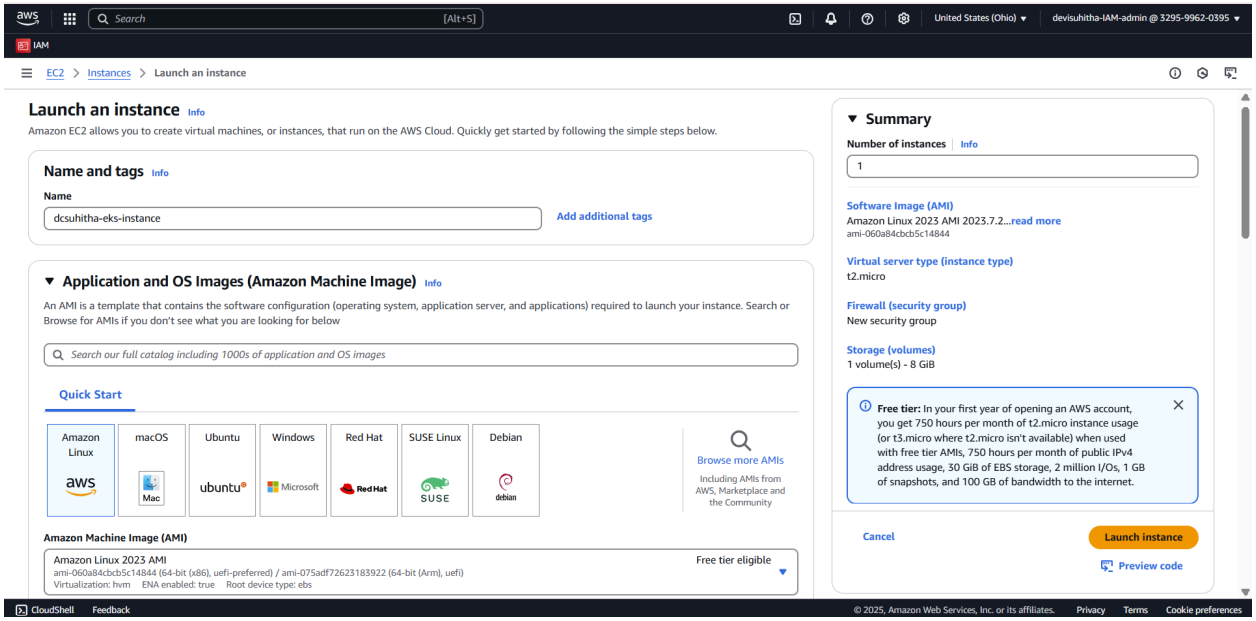
Selected **Launch instances** at the top right corner of my console.



- Named my EC2 instance dcsuhitha-eks-instance.
- For the **Amazon Machine Image,** selected **Amazon Linux 2023 AMI.**

What is AMI?

An **AMI** (**Amazon Machine Image**) is a template or blueprint used to create an EC2 instance. The image defines our EC2 instance's operating system and with the applications needed to launch the instance.

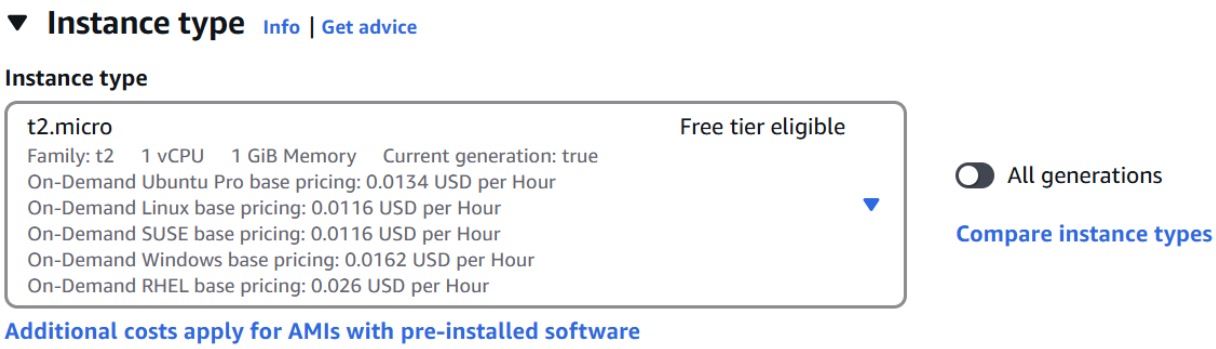


- For the **Instance type,** I have selected **t2.micro.**

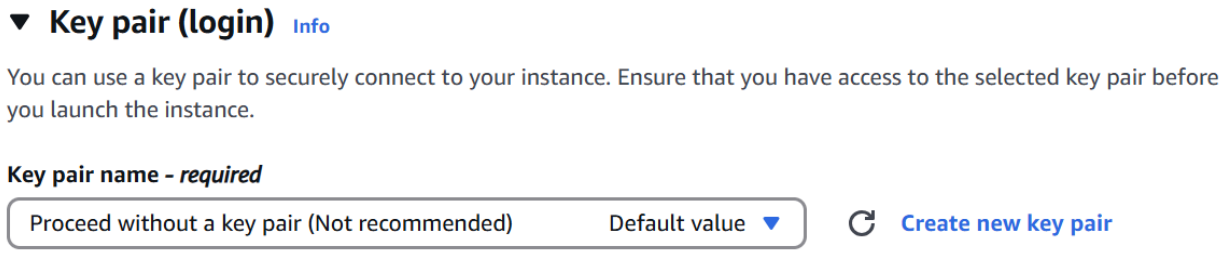
What is the instance type?

If AMIs give us pre-built software and operating systems, instance types cover the '**hardware**' components. CPU power, memory size, storage space and more!

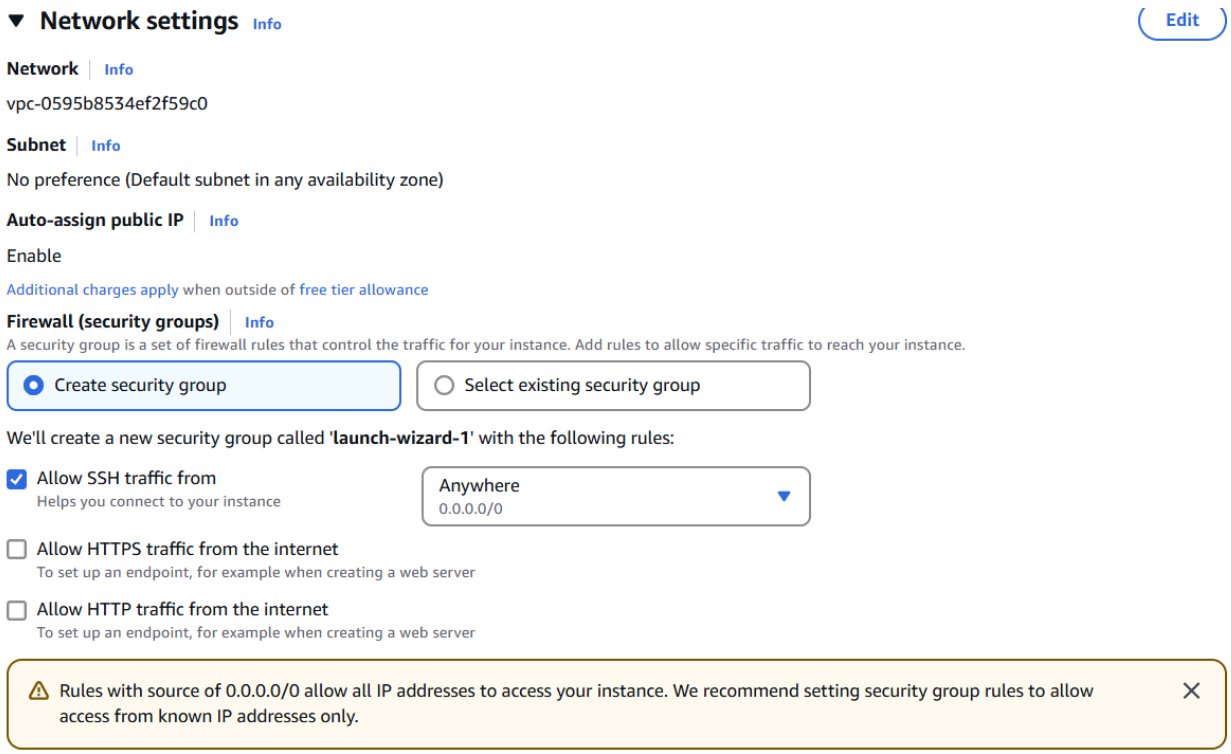
So, while the AMI decides what operating system our server runs, the instance type determines how fast and powerful it performs.



For the **Key pair (login)** panel, I have selected **Proceed without a key pair (not recommended).**



I haven't changed anything in the **Networking** section and keep the default security group.



- Selected **Launch Instances.**

Connected to my EC2 instance

Now that my EC2 instance is up and running, let's connect to it using EC2 Instance Connect.

What is EC2 Instance Connect?

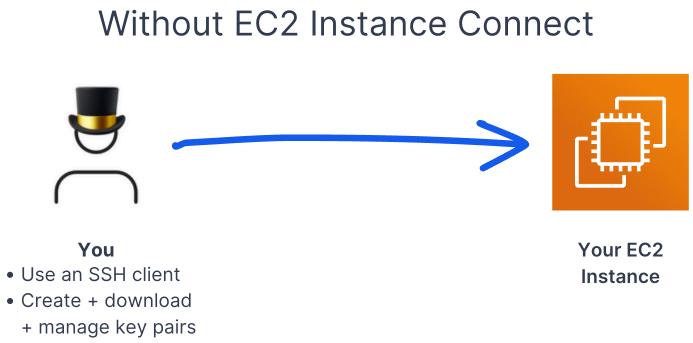
EC2 Instance Connect is a shortcut way to get direct SSH(Secure Shell) access to our EC2 instance. Instead of having to manage a key pair manually, Instance Connect connects us to our instance within our AWS Management Console - no download required!

Why am I using EC2 Instance Connect?

When I connect via SSH to an EC2 instance, I usually need to:

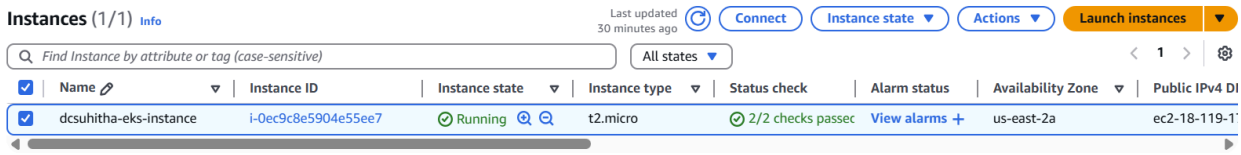
- Generate a key pair.
- Associate the public key with my EC2 instance.
- Securely store my private key on my local machine.
- Set up an SSH client (a software that can handle the SSH protocol, like Terminal on Mac/Linux or VS Code)
- Provide my private key and run SSH commands to set up an SSH connection to my EC2 instance.

EC2 Instance Connect let's skip all those steps and connect to my EC2 instances directly using the AWS Management Console.I'd use the usual, manual way when I have some development work to do, but Instance Connect is great because I'm just running a few commands in the terminal.

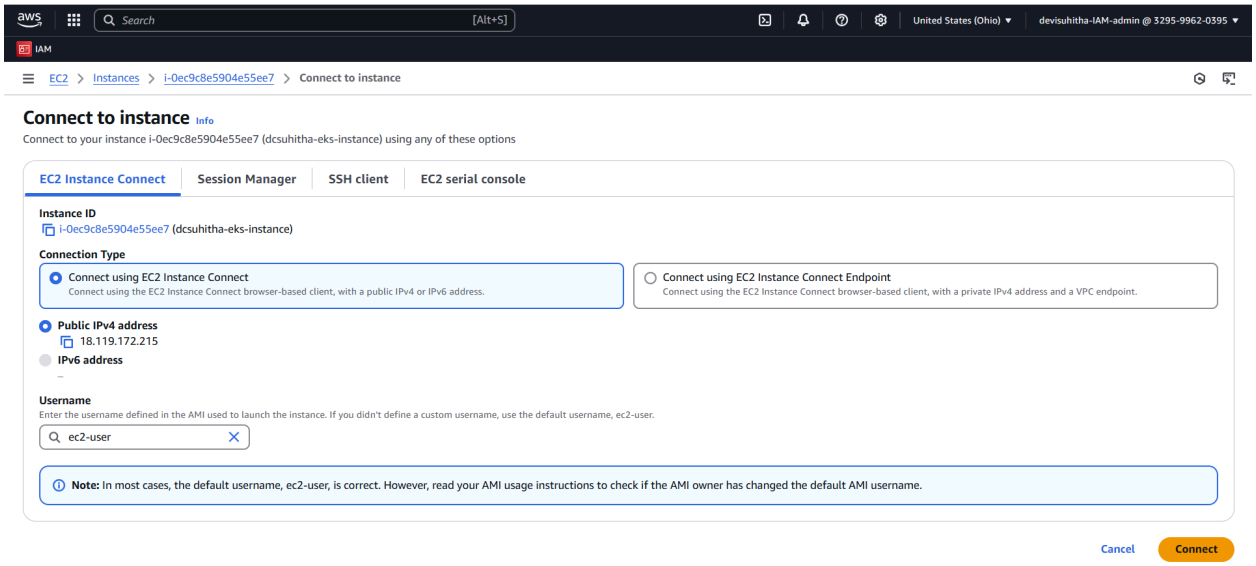


- I have Selected the instance I just launched.

- Selected the checkbox next to **dcsuhitha-eks-instance.** Selected the **Connect** button.



- Welcome to my instance connection page! This is where AWS gives me different options to connect with my EC2 instance.



What is the yellow warning banner about Port 22?
That warning banner is a **security reminder** from AWS.

Port 22 is the default port for SSH (Secure Shell) connections, which is what we use to connect securely to our EC2 instance.

Back when we set up our EC2 instance, we used the default security group which allows SSH access from **any** IP address. This opens your instance to the entire internet, which makes it more vulnerable to attacks.

- Selected **Connect** at the bottom of the page where this opened a terminal session directly in my browser in a new tab.

Step 2: Launched an EKS cluster (and get an error)

Now for the main event – I created my Kubernetes cluster using Amazon EKS!

I used the **`eksctl`** command-line tool within my EC2 instance terminal to automate the creation of my EKS cluster.

What is Kubernetes?

Kubernetes is a tool that helps us manage our running containers.

Once I've created containers (e.g. using Docker), Kubernetes helps keep them running smoothly by automatically handling tasks like load balancing, scaling, and restarting containers if they fail. It **doesn't** build the containers, that's Docker's job but it's great for **keeping them running** without us having to do it manually.

Why do people Use Kubernetes?

Kubernetes is a powerful tool for automating the management of containerized applications/running containers/container orchestration at scale. Without it, developers would need to:

- Manually start, stop, and monitor each container
- Scale applications up or down one container at a time
- Manage storage and networking for each container

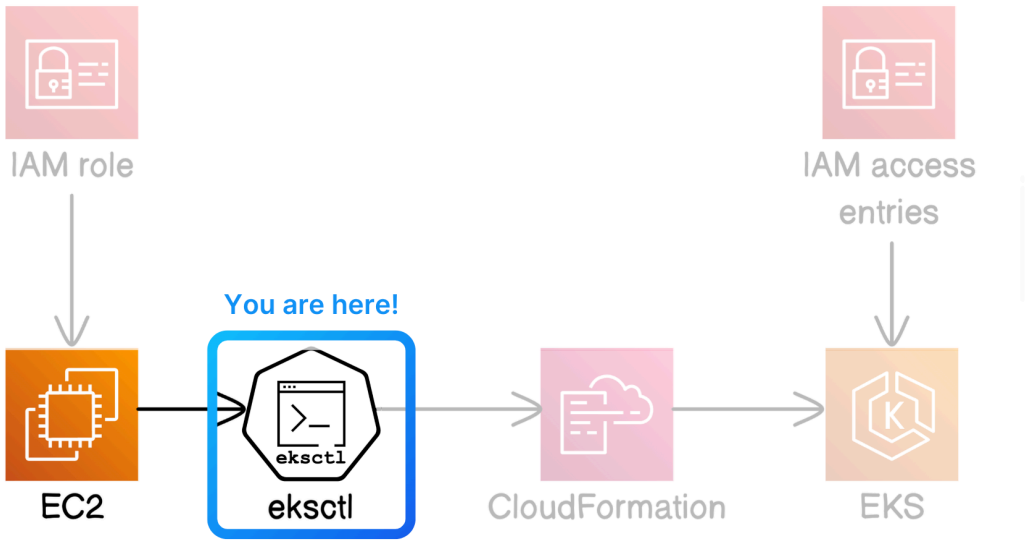
- Perform updates carefully to avoid downtime

As application complexity grows, managing containers manually becomes inefficient and error-prone. Kubernetes automates these tasks handling deployment, scaling, monitoring, and updates so teams can focus on delivering features rather than managing infrastructure. Companies and developers use kubernetes to deploy and manage large scale containerized applications.

What is Amazon EKS?

While Kubernetes makes it easier to work with containers, **Amazon EKS** makes it easier to work with Kubernetes itself!

Setting up Kubernetes from scratch can be quite a time consuming and complex thing to do, because we'd need to configure Kubernetes' networking, scaling, and security settings on our own. Amazon EKS handles all of these set up tasks for us and helps us integrate Kubernetes with other AWS services. **eksctl** is a tool specifically for working with EKS in the command line.



What I'm about to build in this step!

In this step,

- I attempted to create an EKS cluster (and ran into an error)!
- I installed a tool for creating Kubernetes clusters called **eksctl.**

I tried creating a cluster in my EC2 Instance Connect window, where I ran below command:

What does this command do?

This command is meant to set up a Kubernetes cluster!

I have likely seen an error message saying something like **eksctl not found.**

```
[ec2-user@ip-172-31-8-98 ~]$ eksctl create cluster \
> ^[[200~name dcsuhitha-eks-cluster \
> region ${AWS_REGION} \
> nodegroup-name dcsuhitha-nodegroup \
> node-type t3.micro \
> node 3 \
> nodes-min 1 \
> nodes-max 4
-bash: eksctl: command not found
```

This is expected! It means I haven't installed eksctl yet.

What is eksctl? → eksctl is an official AWS tool for managing Amazon EKS clusters in our terminal. It's much, much easier to use compared to setting up a Kubernetes cluster using the AWS CLI!

If we were using the AWS CLI, we'd have to create a lot of other components manually before getting to deploy a cluster.

Installed eksctl

- Ran these commands in my terminal:

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

sudo mv -v /tmp/eksctl /usr/local/bin

```
[ec2-user@ip-172-31-8-98 ~]$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
[ec2-user@ip-172-31-8-98 ~]$ sudo mv -v /tmp/eksctl /usr/local/bin
copied '/tmp/eksctl' -> '/usr/local/bin/eksctl'
removed '/tmp/eksctl'
```

What do these commands do?

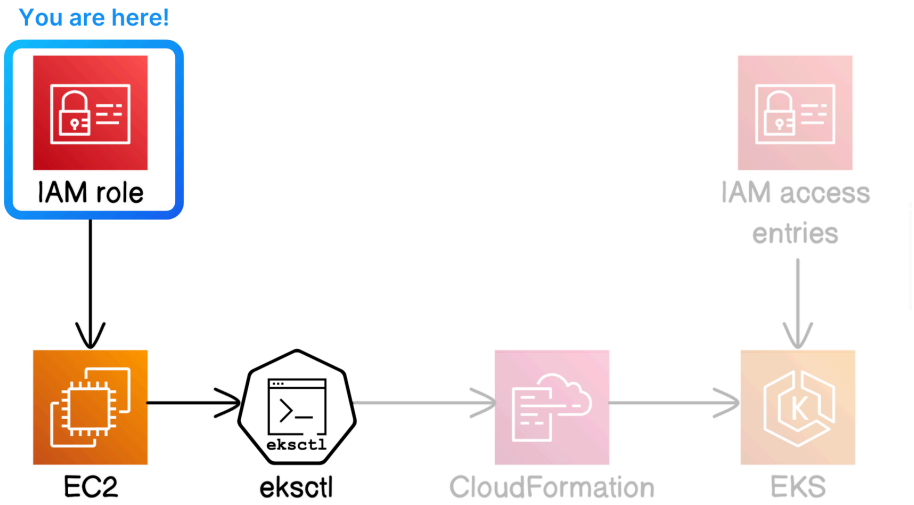
The commands here download and install **eksctl** on my EC2 instance. The first commands downloads the latest eksctl release from GitHub, then the second command moves it to a directory that lets me run eksctl from anywhere in my terminal.

- Checked that eksctl is installed correctly by running eksctl version. I should see the version number printed in the terminal.

```
[ec2-user@ip-172-31-8-98 ~]$ eksctl version
0.207.0
```

Step 3: Launched an EKS cluster (for real this time)

After installing **eksctl,** I proceeded to launch a Kubernetes cluster on EKS. I also configured the necessary permissions for the EC2 instance to run AWS commands.



What I'm about to build in this step!

In this step, I:

- Attempted to create an EKS cluster (and encountered an error again).

- Set up the IAM role for my EC2 instance.
- Successfully created an EKS cluster using **eksctl.**

What error did I expect to see when running eksctl create cluster?

I expected a permissions-related error, likely due to the EC2 instance not having the necessary IAM role to interact with EKS.

How did I approach solving it?

I reran the eksctl create cluster command to trigger the error and reviewed the error message. This confirmed the issue was related to missing permissions, and I proceeded to attach the appropriate IAM role to the EC2 instance.

Now that eksctl is installed, again I tried to create my cluster and ran this command.

Note: I made sure to replace `[YOUR-REGION]` in the last line of the command with my AWS region code (e.g., `us-west-2`). I ensured that the selected region matched the one where I had launched my EC2 instance.

```
eksctl create cluster \
--name dcsuhitha-eks-cluster \
--nodegroup-name dcsuhitha-nodegroup \
--node-type t2.micro \
--nodes 3 \
--nodes-min 1 \
--nodes-max 3 \
--version 1.31 \
--region [YOUR-REGION]
```

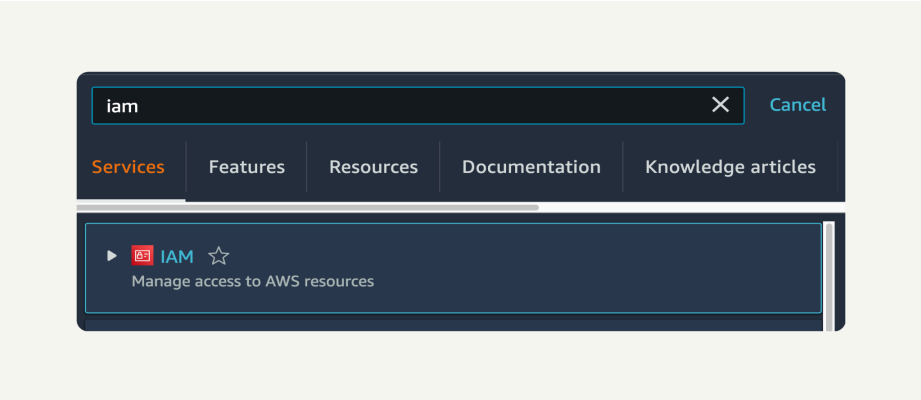
```
[ec2-user@ip-172-31-8-98 ~]$ eksctl create cluster \
> --name dcsuhitha-eks-cluster \
--nodegroup-name dcsuhitha-nodegroup \
--node-type t2.micro \
--nodes 3 \
--nodes-min 1 \
--nodes-max 3 \
--version 1.31 \
--region us-east-2
Error: checking AWS STS access – cannot get role ARN for current session: operation error STS: GetCallerIdenti
ty, get identity: get credentials: failed to refresh cached credentials, no EC2 IMDS role found, operation err
or ec2imds: GetMetadata, http response error StatusCode: 404, request to EC2 IMDS failed
```

What's the error this time?
My EC2 instance doesn't have the permission to create an EKS cluster. By default, I can think of my EC2 instance as another computer that isn't 'logged in' or has access to my AWS Account yet.

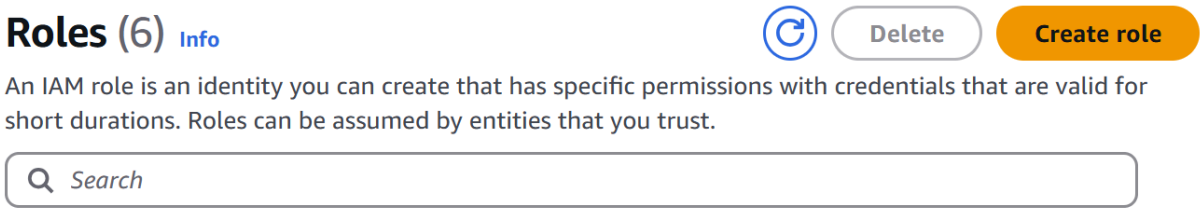
I need to **set up an IAM role** that lets my EC2 instance communicate with services like EKS to create the cluster.

Created an IAM role for my EC2 instance

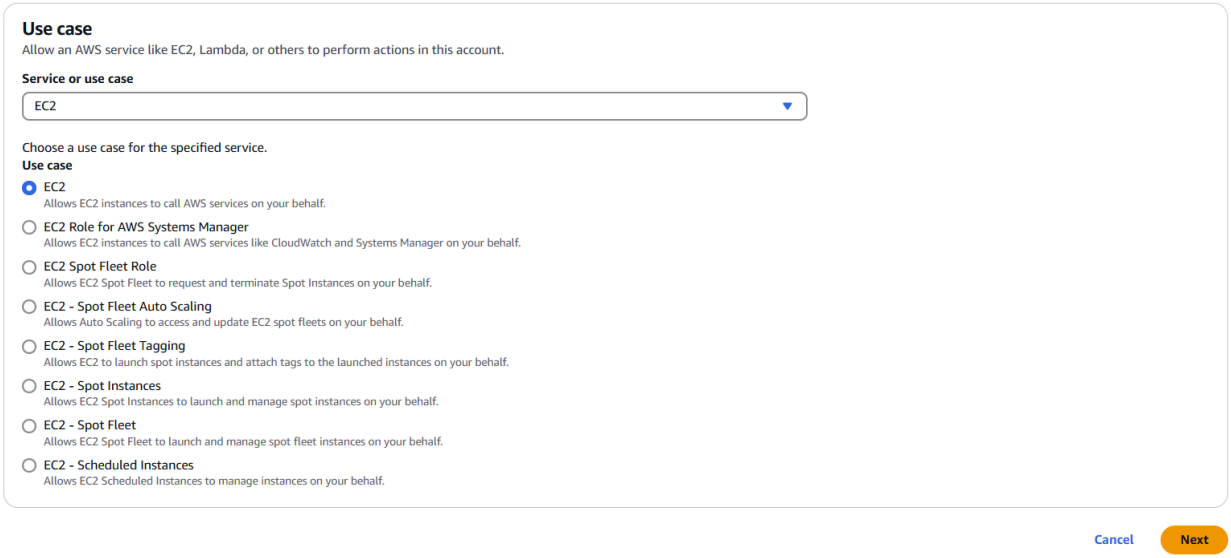
To resolve the permissions error, I set up a new **Role** in AWS IAM. I opened a new tab and navigated to the AWS IAM console to begin the process.



- Selected **Roles** from the left hand sidebar and create Role.



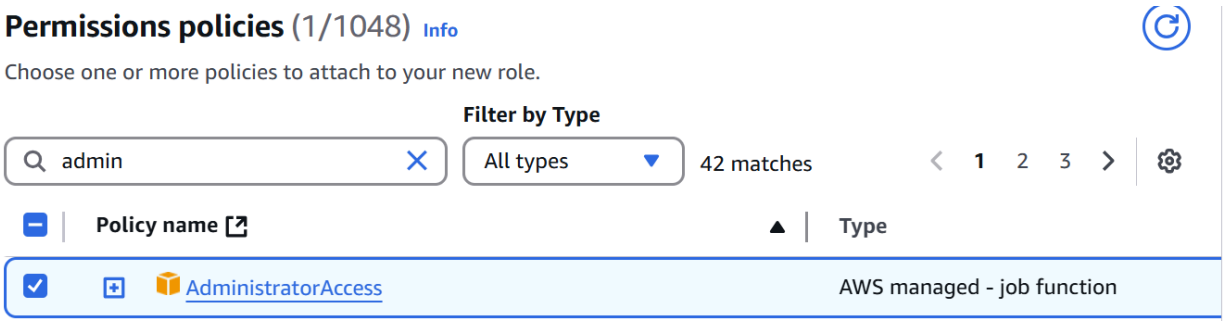
- Under the **Trusted entity type**, I selected **AWS service** to tell AWS that I'm setting up this role for an AWS service (Amazon EC2)**,** and for **Use case**, I chose **EC2**.



I clicked **Next**, and under **Permissions policies**, I gave my EC2 instance **AdministratorAccess** so it can access all necessary AWS services. I made sure the **AdministratorAccess** option was checked and clicked **Next**.

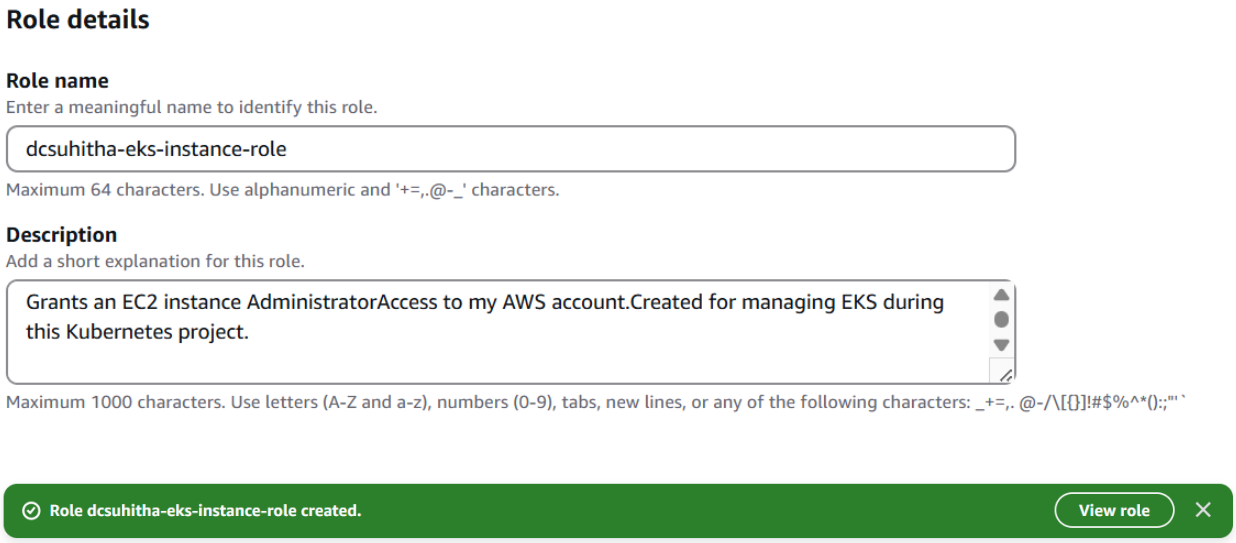
What does the AdministratorAccess permission policy mean?

A: It gives the EC2 instance full access to all AWS resources and services in my account. This is important because the instance needs wide permissions to deploy and manage the Kubernetes cluster using many different AWS services.



I named the role **dcsuhitha-eks-instance-role** and added a short description:

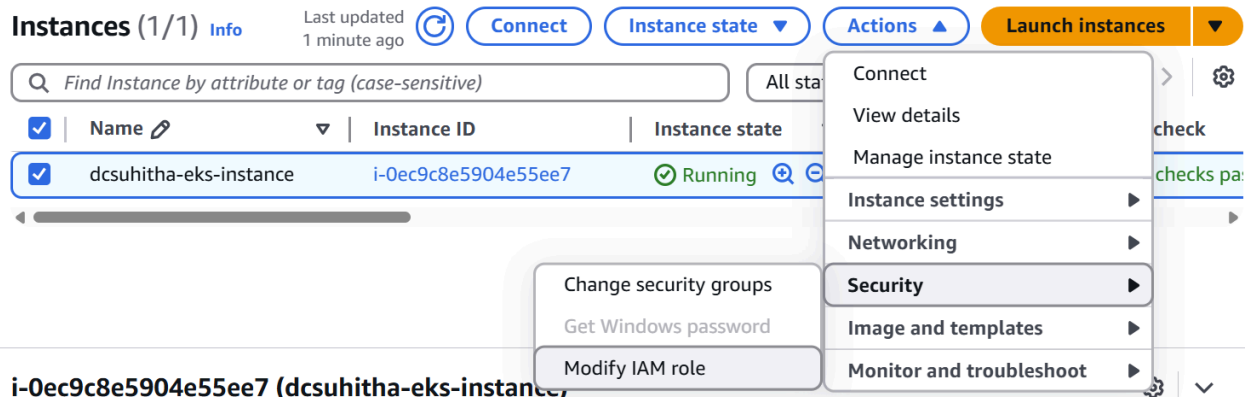
Grants an EC2 instance AdministratorAccess to my AWS account.Created for managing EKS during this Kubernetes project.



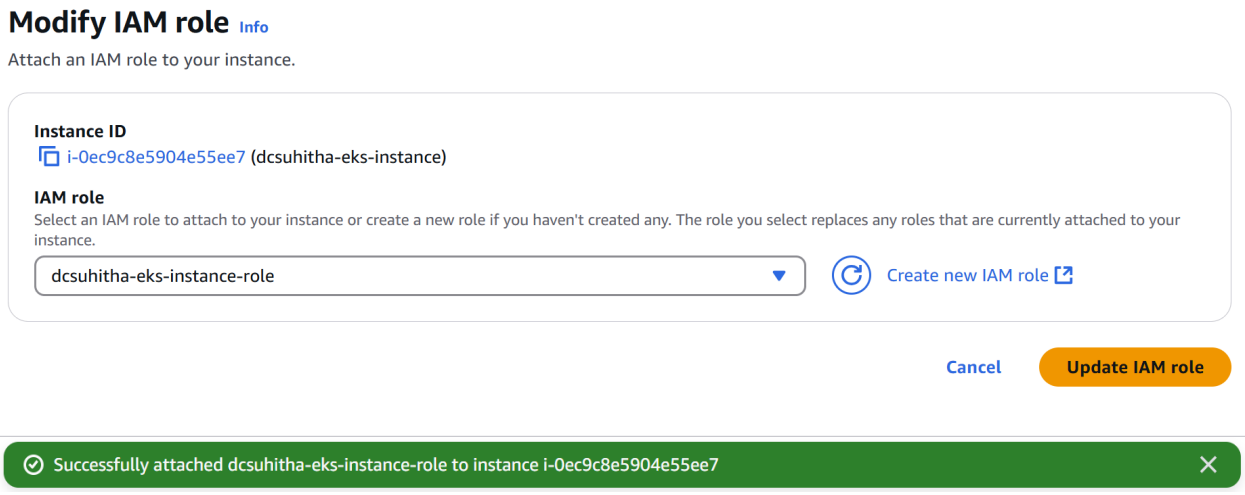
Attached IAM role to EC2 instance

With the new role created, I attached it to my EC2 instance:

- I went back to the Amazon EC2 console.
- From the left sidebar, I selected Instances.
- I checked the box next to my EC2 instance named dcsuhitha-eks-instance.
- Then, I clicked the Actions dropdown, navigated to Security → Modify IAM role, and selected the newly created role dcsuhitha-eks-instance-role to attach it.



Under IAM role, I selected my new dcsuhitha-eks-instance-role role and clicked **update IAM role.**



Created the EKS Cluster (Third Time)

- I headed back to the EC2 Instance Connect tab in my browser.
- Then, I ran the eksctl create cluster command one more time to kick off the creation of my Kubernetes cluster.

```
eksctl create cluster \
--name dcsuhitha-eks-cluster \
--nodegroup-name dcsuhitha-nodegroup \
--node-type t2.micro \
--nodes 3 \
--nodes-min 1 \
--nodes-max 3 \
--version 1.31 \
--region us-east-2
```

What does this command do? → The eksctl create cluster command performs several tasks to set up the Kubernetes environment:

- Set up an EKS cluster named dcsuhitha-eks-cluster
- Launch a managed node group called dcsuhitha-nodegroup
- Use t2.micro EC2 instances as worker nodes
- Starts the node group with 3 nodes, and enables auto-scaling between 1 (minimum) and 3 (maximum) nodes based on demand
- Deploys the cluster using Kubernetes version 1.31

```
[ec2-user@ip-172-31-8-98 ~]$ eksctl create cluster \
--name dcsuhitha-eks-cluster \
--nodegroup-name dcsuhitha-nodegroup \
--node-type t2.micro \
--nodes 3 \
--nodes-min 1 \
--nodes-max 3 \
--version 1.31 \
--region us-east-2
2025-04-21 00:40:48 [ℹ]  eksctl version 0.207.0
2025-04-21 00:40:48 [ℹ]  using region us-east-2
2025-04-21 00:40:48 [ℹ]  setting availability zones to [us-east-2a us-east-2c us-east-2b]
2025-04-21 00:40:48 [ℹ]  subnets for us-east-2a - public:192.168.0.0/19 private:192.168.96.0/19
2025-04-21 00:40:48 [ℹ]  subnets for us-east-2c - public:192.168.32.0/19 private:192.168.128.0/19
2025-04-21 00:40:48 [ℹ]  subnets for us-east-2b - public:192.168.64.0/19 private:192.168.160.0/19
2025-04-21 00:40:48 [ℹ]  nodegroup "dcsuhitha-nodegroup" will use "" [AmazonLinux2/1.31]
2025-04-21 00:40:48 [ℹ]  using Kubernetes version 1.31
2025-04-21 00:40:48 [ℹ]  creating EKS cluster "dcsuhitha-eks-cluster" in "us-east-2" region with managed nodes
2025-04-21 00:40:48 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2025-04-21 00:40:48 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-2
 --cluster=dcsuhitha-eks-cluster'
2025-04-21 00:40:48 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "dcsuhith
a-eks-cluster" in "us-east-2"
```

```
2025-04-21 00:40:48 [ℹ]  CloudWatch logging will not be enabled for cluster "dcsuhitha-eks-cluster" in "us-east-2"
2025-04-21 00:40:48 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all
)} --region=us-east-2 --cluster=dcsuhitha-eks-cluster'
2025-04-21 00:40:48 [ℹ]  default addons kube-proxy, coredns, metrics-server, vpc-cni were not specified, will install them as EKS addons
2025-04-21 00:40:48 [ℹ]
2 sequential tasks: { create cluster control plane "dcsuhitha-eks-cluster",
    2 sequential sub-tasks: {
        2 sequential sub-tasks: {
            1 task: { create addons },
            wait for control plane to become ready,
        },
        create managed nodegroup "dcsuhitha-nodegroup",
    }
}
2025-04-21 00:40:48 [ℹ]  building cluster stack "eksctl-dcsuhitha-eks-cluster-cluster"
2025-04-21 00:40:49 [ℹ]  deploying stack "eksctl-dcsuhitha-eks-cluster-cluster"
2025-04-21 00:41:19 [ℹ]  waiting for CloudFormation stack "eksctl-dcsuhitha-eks-cluster-cluster"
2025-04-21 00:41:49 [ℹ]  waiting for CloudFormation stack "eksctl-dcsuhitha-eks-cluster-cluster"
2025-04-21 00:42:49 [ℹ]  waiting for CloudFormation stack "eksctl-dcsuhitha-eks-cluster-cluster"
2025-04-21 00:43:49 [ℹ]  waiting for CloudFormation stack "eksctl-dcsuhitha-eks-cluster-cluster"
```

? Why did I run into errors while using eksctl?

I encountered two errors during the setup process with eksctl:

1. Missing installation – The first error occurred because eksctl was not installed on my EC2 instance.

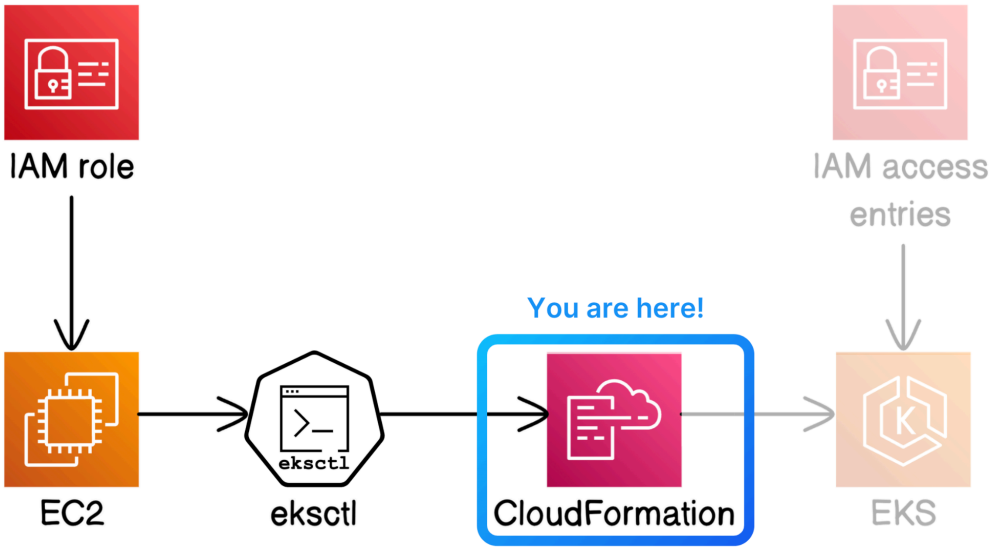
2. Insufficient permissions – The second error happened because the EC2 instance did not yet have the necessary IAM role to access AWS services and manage resources within my AWS account.

? How did I use eksctl?

I used **eksctl** to create a Kubernetes cluster using the command line. The 'eksctl create cluster' command I ran defined the name of the cluster, the name of the node group, and the node size settings. I also specified the AWS region and the EC2 instance type to be used for the worker nodes. This allowed me to quickly provision a fully managed EKS cluster with minimal configuration.

Step 4: Tracked How AWS Created My EKS Cluster

I took a tour around the AWS Management Console to see how AWS actually created a Kubernetes cluster. What happens under the hood when I ran eksctl create cluster?

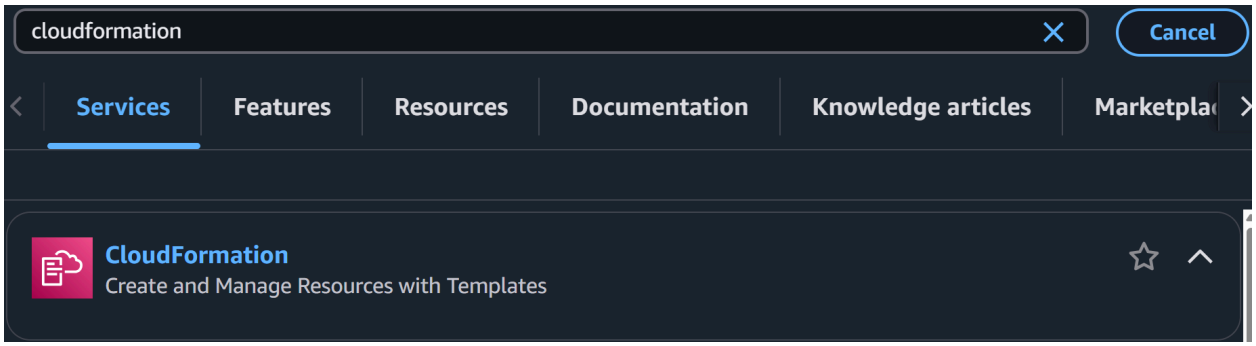


What I'm about to build in this step!

In this step, I

- Used **CloudFormation** to track how my cluster is getting created.

In a new tab, head to the CloudFormation console.



What is CloudFormation?

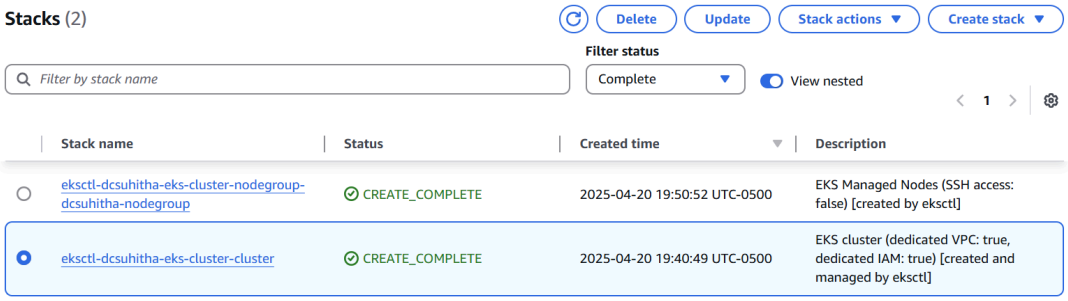
CloudFormation is AWS's service for setting up infrastructure as code. We write a **template** describing the resources we need (like an instruction manual), and CloudFormation handles creating and configuring those resources.

? Why am I in CloudFormation?

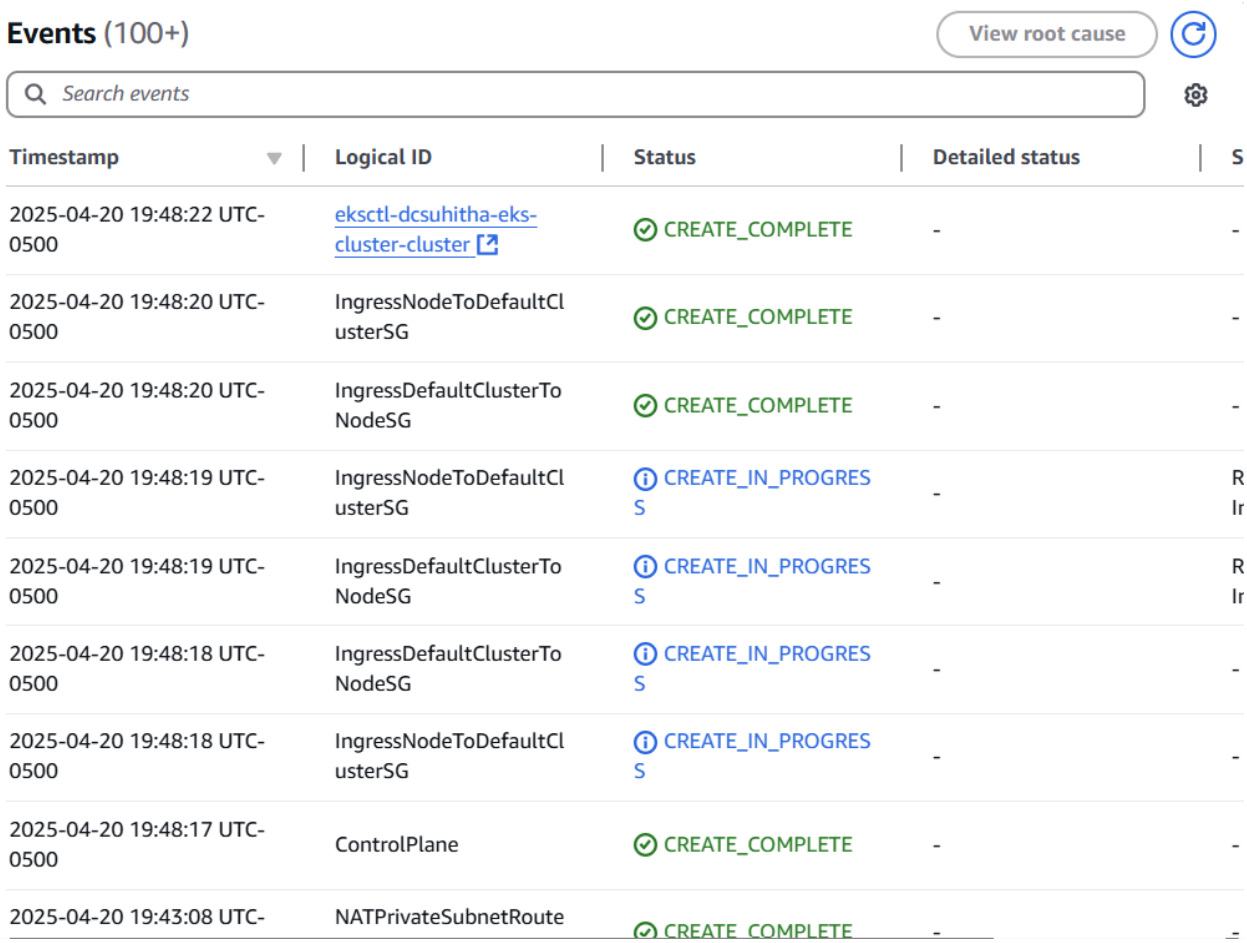
eksctl actually uses CloudFormation under the hood to create an EKS cluster.

When I ran the 'eksctl create cluster command', eksctl sets up a CloudFormation stack to automate the creation of all the necessary resources for the EKS cluster.

On the **Stacks** page in CloudFormation, I noticed a new stack in progress named **eksctl-dcsuhitha-eks-cluster-cluster**.



I selected the stack and used the **Events** tab to track the creation of each resource step by step.



Events (100+)

View root cause

Search events

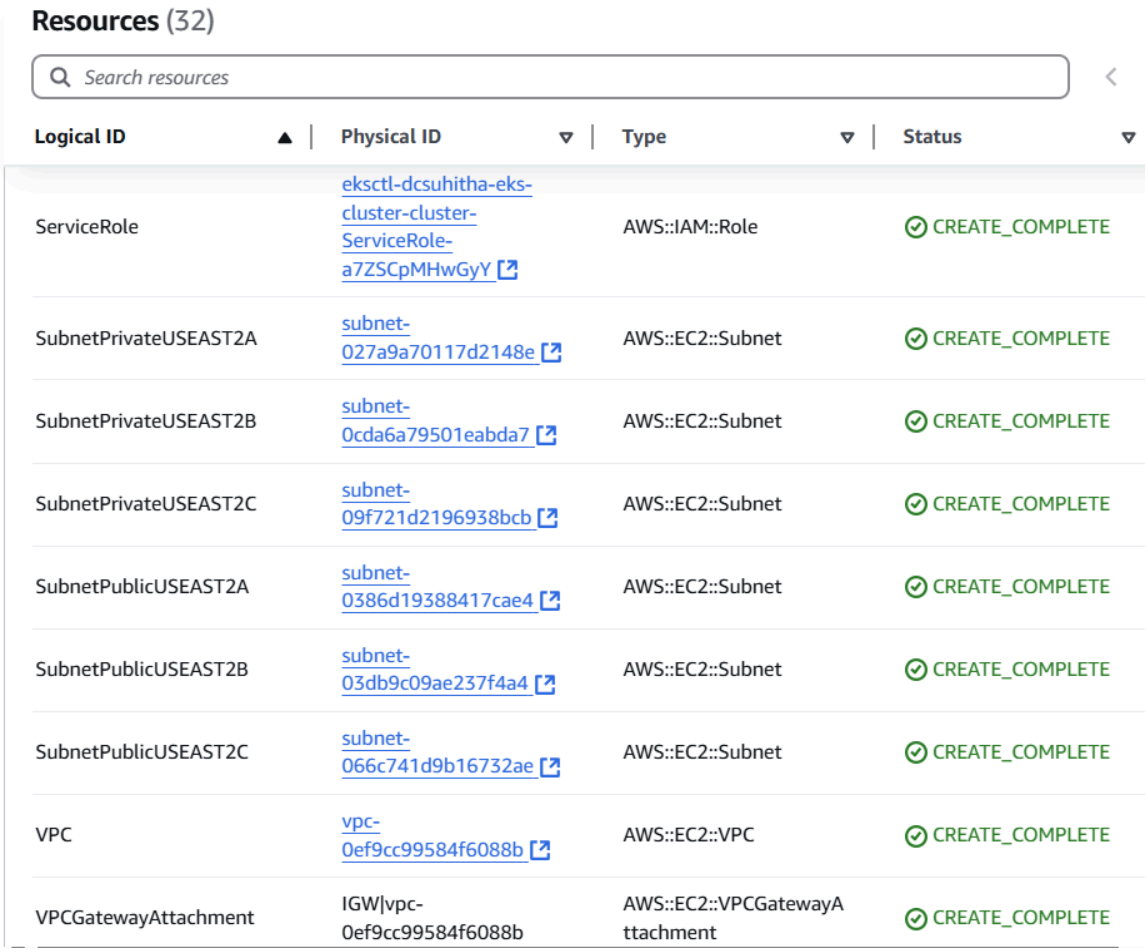
Timestamp	Logical ID	Status	Detailed status	S
2025-04-20 19:48:22 UTC-0500	eksctl-dcsuhitha-eks-cluster-cluster	CREATE_COMPLETE	-	-
2025-04-20 19:48:20 UTC-0500	IngressNodeToDefaultClusterSG	CREATE_COMPLETE	-	-
2025-04-20 19:48:20 UTC-0500	IngressDefaultClusterToNodeSG	CREATE_COMPLETE	-	-
2025-04-20 19:48:19 UTC-0500	IngressNodeToDefaultClusterSG	CREATE_IN_PROGRESS	-	R Ir
2025-04-20 19:48:19 UTC-0500	IngressDefaultClusterToNodeSG	CREATE_IN_PROGRESS	-	R Ir
2025-04-20 19:48:18 UTC-0500	IngressDefaultClusterToNodeSG	CREATE_IN_PROGRESS	-	-
2025-04-20 19:48:18 UTC-0500	IngressNodeToDefaultClusterSG	CREATE_IN_PROGRESS	-	-
2025-04-20 19:48:17 UTC-0500	ControlPlane	CREATE_COMPLETE	-	-
2025-04-20 19:43:08 UTC-	NATPrivateSubnetRoute	CREATE_COMPLETE	-	-

What are these Events about?

The **Events** tab gives us a timeline of each action CloudFormation is taking to set up our resources. It's a live update of what's happening, which can be helpful for tracking progress or identifying any issues that come up during creation.

We can even select the **Refresh** button at the top of the panel to watch new updates come through on our stack's resources.

- Now I selected the **Resources** tab. Woah! Lots of resources are getting created here.



Resources (32)

Search resources

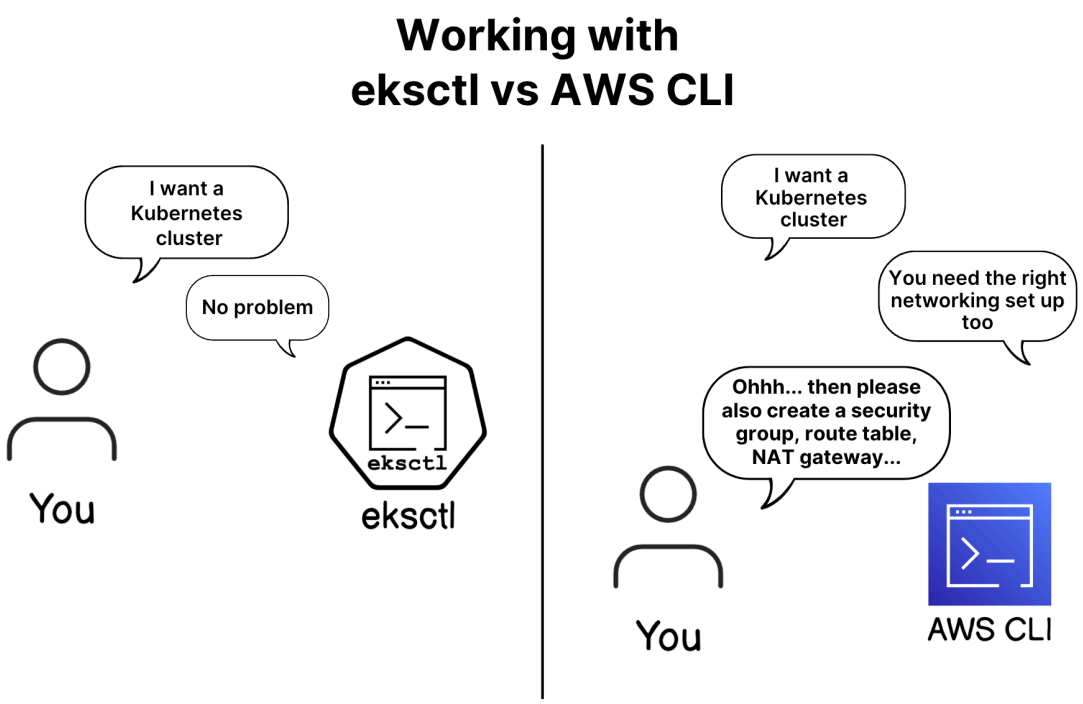
Logical ID	Physical ID	Type	Status
ServiceRole	eksctl-dcsuhitha-eks-cluster-cluster-ServiceRole-a7ZSCpMHwGyY	AWS::IAM::Role	CREATE_COMPLETE
SubnetPrivateUSEAST2A	subnet-027a9a70117d2148e	AWS::EC2::Subnet	CREATE_COMPLETE
SubnetPrivateUSEAST2B	subnet-0cda6a79501eabda7	AWS::EC2::Subnet	CREATE_COMPLETE
SubnetPrivateUSEAST2C	subnet-09f721d2196938bcb	AWS::EC2::Subnet	CREATE_COMPLETE
SubnetPublicUSEAST2A	subnet-0386d19388417cae4	AWS::EC2::Subnet	CREATE_COMPLETE
SubnetPublicUSEAST2B	subnet-03db9c09ae237f4a4	AWS::EC2::Subnet	CREATE_COMPLETE
SubnetPublicUSEAST2C	subnet-066c741d9b16732ae	AWS::EC2::Subnet	CREATE_COMPLETE
VPC	vpc-0ef9cc99584f6088b	AWS::EC2::VPC	CREATE_COMPLETE
VPCGatewayAttachment	IGW|vpc-0ef9cc99584f6088b	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE

? What are these resources?

In the Resources tab, I saw various networking components like VPC, subnets, route tables, security groups, and gateways. These set up a secure network for containers to communicate and access the internet.

Using **eksctl** automatically creates all these resources saving time compared to manually setting them up with the AWS CLI.

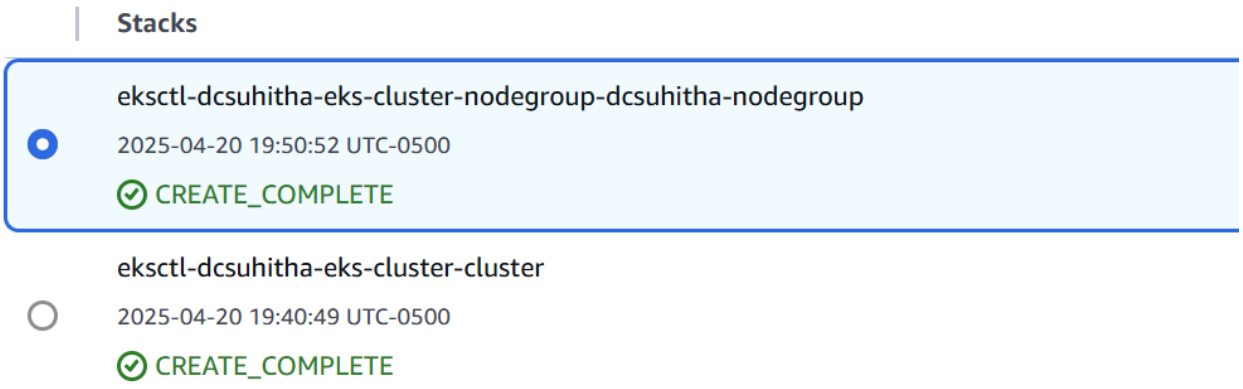
Why can't I just use my account's default VPC?
While I could use my AWS account's default VPC, I might need to manually set up new VPC resources and edit a few settings to make it work for a Kubernetes cluster. eksctl creates a whole new VPC for me to let me start fresh.



? How is CloudFormation involved in creating my EKS cluster?

CloudFormation helped create my EKS cluster because **eksctl** uses CloudFormation under the hood to provision all necessary AWS resources. **It created VPC resources because** using the default VPC could lead to compatibility and permission issues when setting up an EKS cluster.

I also noticed a **new stack** appearED on the **Stacks** page about **10–12 minutes** after running the **eksctl create cluster** command.

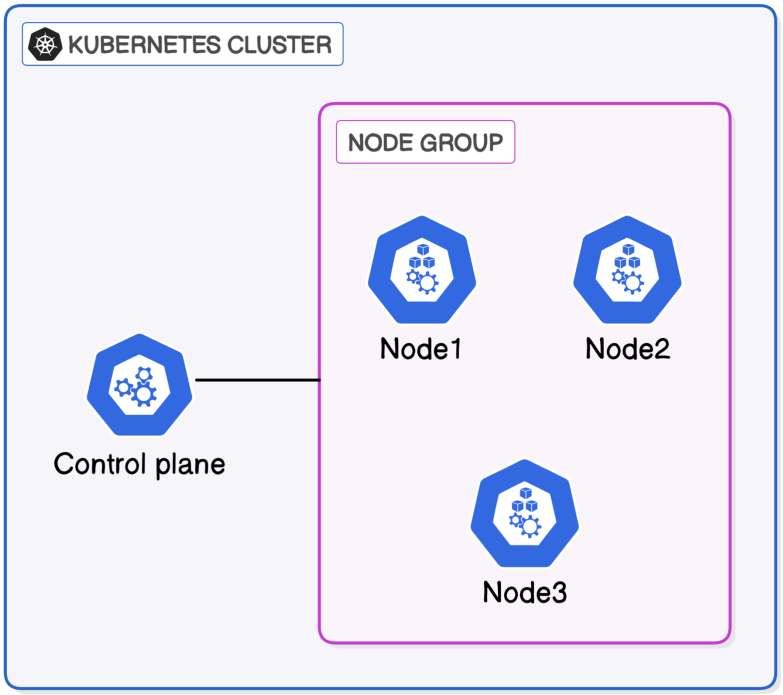


Why was there a second stack?

The second stack is specifically for my **node group**, which is a group of EC2 instances that will run my containers.

CloudFormation separates the core EKS **cluster** stack from the **node group** stack to make it easier to manage and troubleshoot each part independently, especially **if one half fails.**

What's the difference between a cluster and a node group?



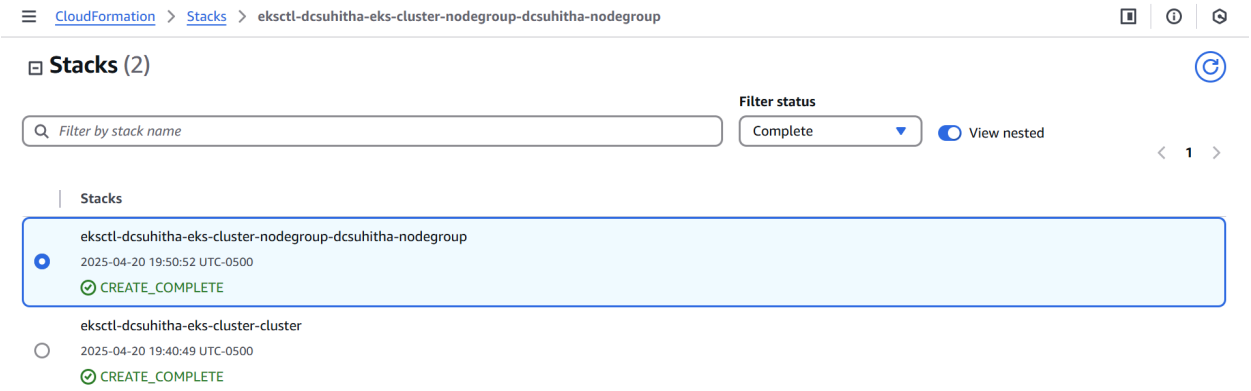
Think of a **cluster** as the environment that Kubernetes uses to run our containerized app. This cluster is made up of:

- **nodes** (the servers that actually run our containers)
- **control plane** (the brain that decides things like when to create or shut down containers)

Within our cluster, Nodes are grouped into **node groups**, which make it easier to manage multiple nodes at once. We can set things like instance type and resource limits for the whole group, making scaling and configuration much simpler.

I selected the **nodegroup stack** and opened the **Events** tab to see what was being deployed.

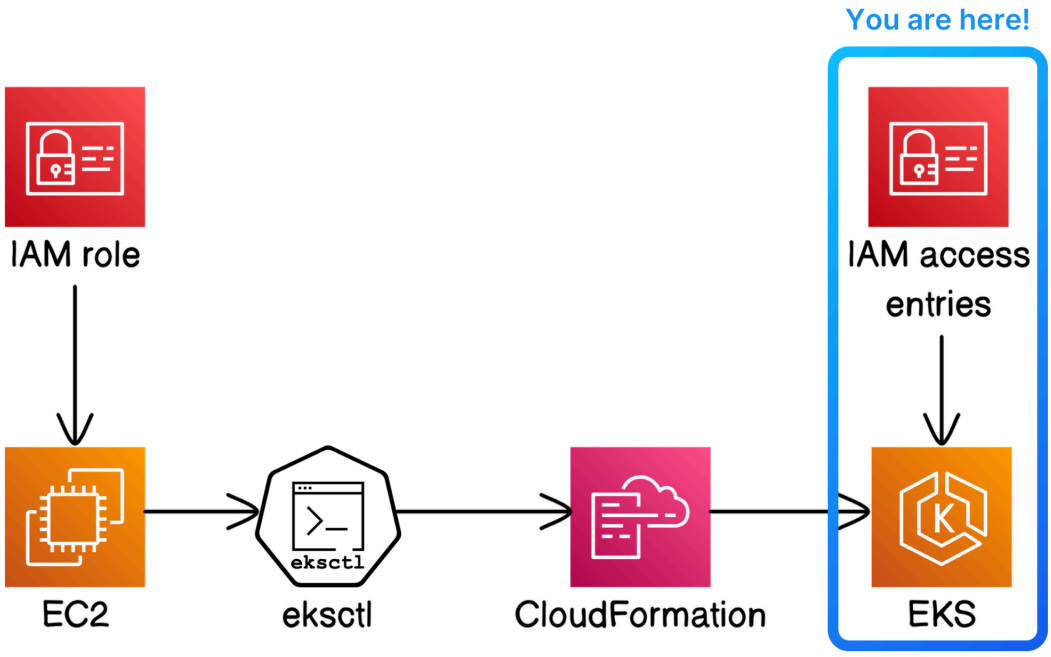
Then, I waited for the first stack (**eksctl-dcsuhitha-eks-cluster-cluster**) to reach the **CREATE_COMPLETE** status before moving forward.



Step 5: Access EKS from the Management Console

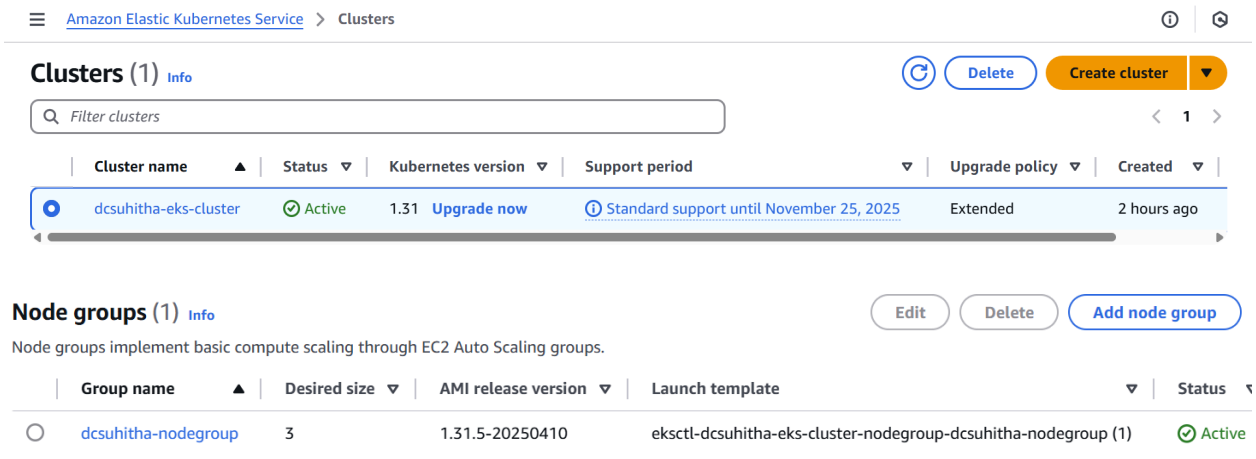
Now that the EKS cluster is ready, I:

- Opened the **EKS console** in the AWS Management Console.
- Granted myself the necessary permissions to interact with the cluster.



Accessed the EKS Console

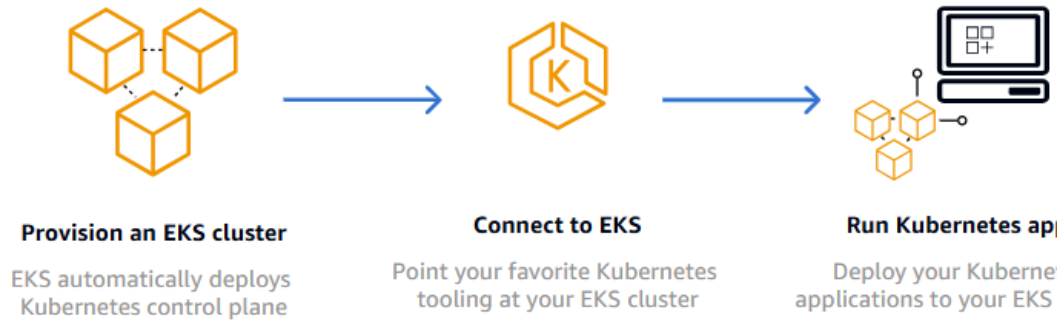
- I opened the EKS console in a new tab and selected the newly created dcsuhitha-eks-cluster.
- On the cluster's page, I navigated to the **Compute tab** and scrolled down to the Node groups panel where I could see the node group being created!



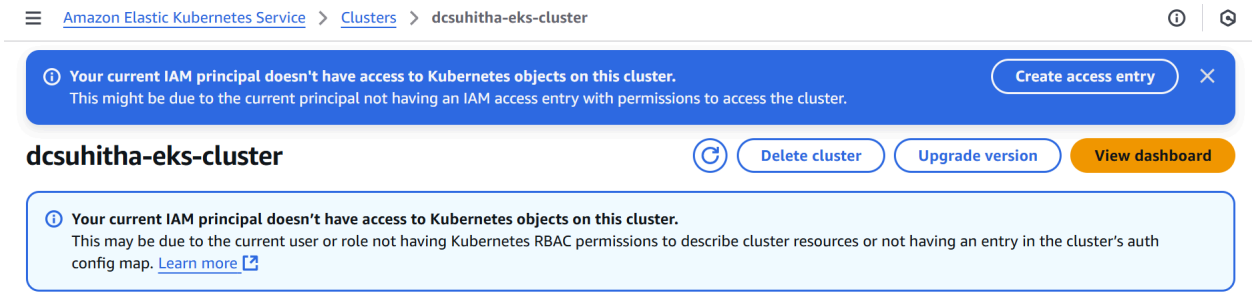
Elastic Kubernetes Service (Amazon EKS) → Fully managed Kubernetes cluster infrastructure

Amazon EKS is a managed service that makes it easy for us to use Kubernetes on AWS without needing to install and operate our own Kubernetes infrastructure.

How does it work?



Scrolled back up to the top of your cluster's page. Notice that there are two banners in blue.



What are these two banners saying?
The two blue banners in the EKS console are letting me know that while I have created a node group, I might not have the permission to see the nodes inside.

But my IAM User has AdministratorAccess - how could it not have permission?
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, which grants full access to AWS resources, Kubernetes will only let me into different parts of the cluster if I have permissions under its own system too.

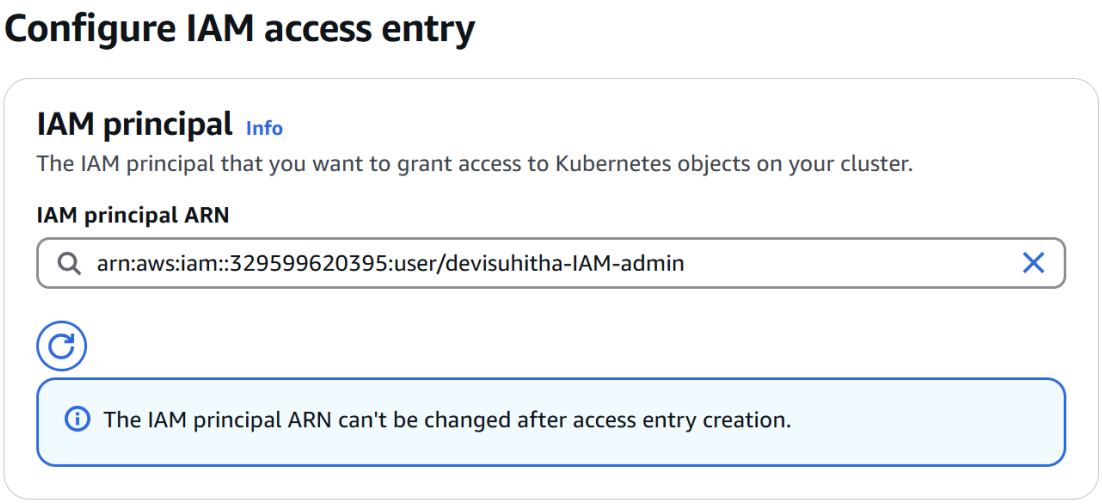
Set up my own access to my cluster

- Selected **Create access entry** at the blue banner at the top of the page.

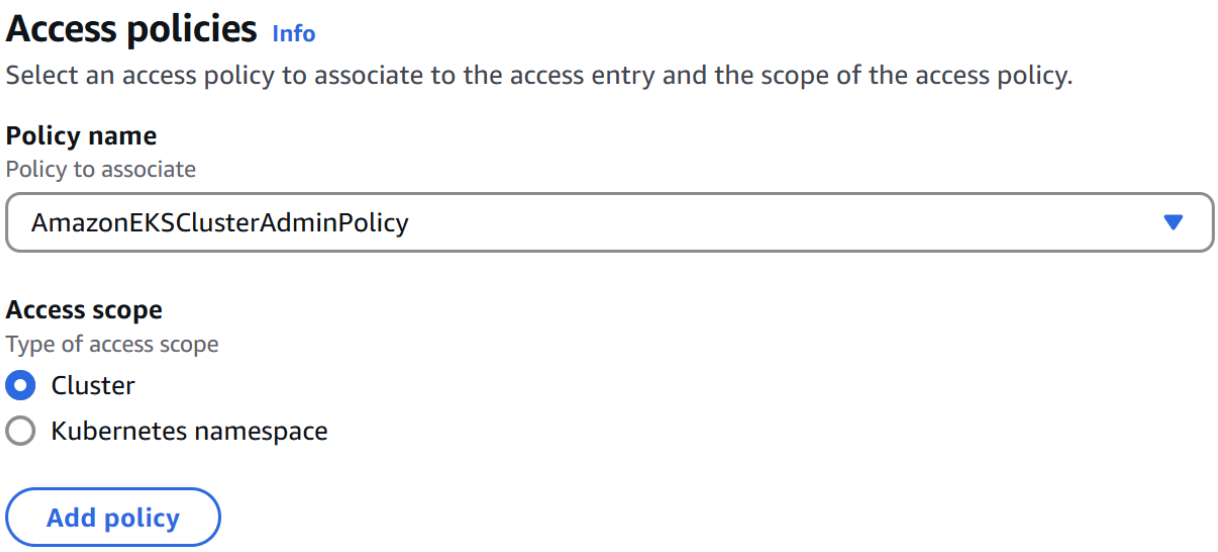
What are IAM access entries?
IAM access entries connect AWS IAM users with Kubernetes' Role-Based Access Control (RBAC), which is Kubernetes' system to manage access inside a cluster.

In short, the IAM Admin user (the one I'm using to open this Management Console) needs a specific access entry to get access to the nodes in my EKS cluster.

- I set up an IAM access entry by selecting my **IAM user's ARN** under **IAM Principal**. I double-checked the IAM Admin name in the top-right corner to ensure it matched my ARN before proceeding.



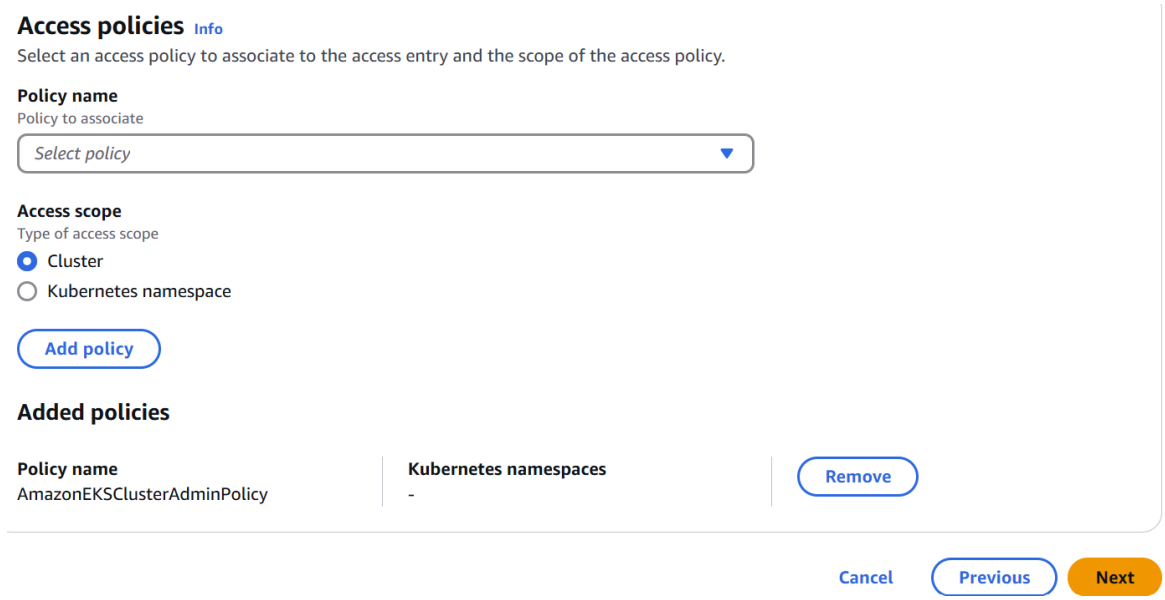
I selected **Next**, and under **Policy name**, I chose **AmazonEKSClusterAdminPolicy** to grant admin access to the EKS cluster.



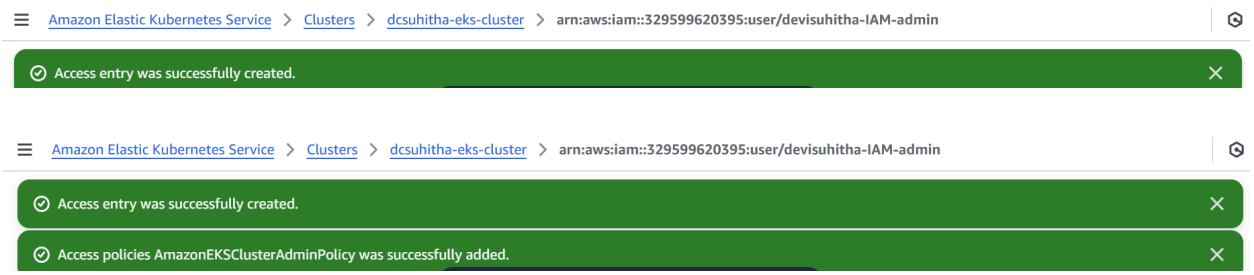
What does this policy do?

The **AmazonEKSClusterAdminPolicy** gives me full administrative rights over my EKS clusters. My IAM user will be able to see and control all parts of my EKS cluster - including all the nodes inside!

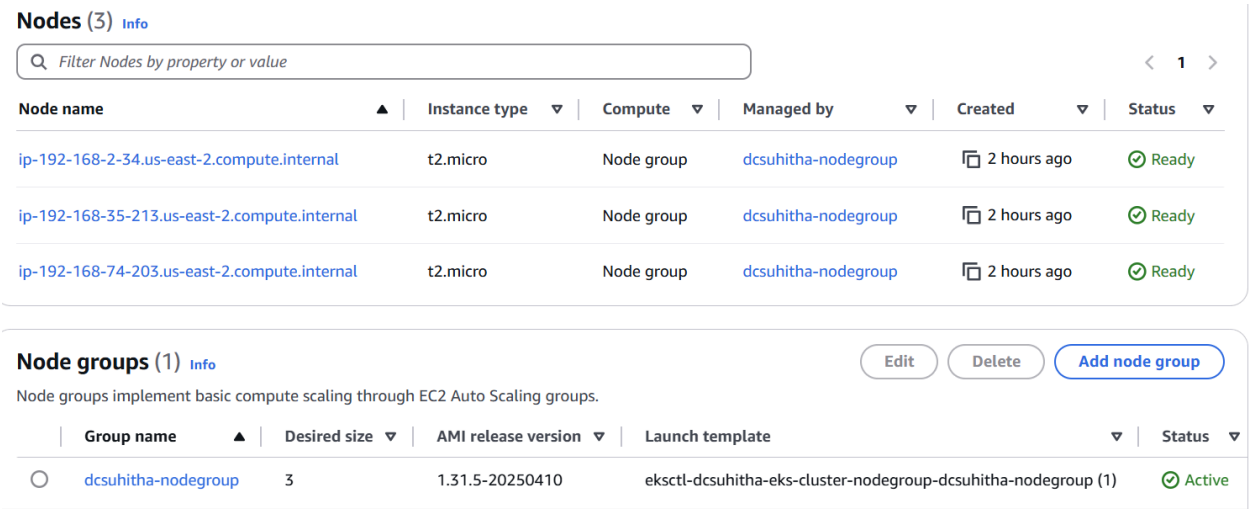
- I clicked **Add policy**, then selected **Next** to continue with the setup.



- Clicked Create. Then, the access entry was successfully created.



I clicked the **refresh** button on the console, and I could now see my nodes listed under the **node group** section. EKS console showing my cluster and its nodes.



Why are there three nodes but one node group?

The **node group** is a group of nodes that share the same configuration, like instance type and scaling settings. In this case, my node group has three nodes, which means there are three EC2 instances that will run containers like a single unit.

I successfully created my very first **Kubernetes cluster using Amazon EKS**.

This marks the completion of the core setup phase for my EKS project. I now have a fully functional Kubernetes environment running on AWS complete with a managed node group, networking resources, and IAM configurations.

I'll be repeating this process in future projects to build on this foundation. But before that... I took on a little extra challenge 👇

🕵️ My Secret Mission: Testing Cluster Resilience

Out of curiosity, I wanted to see how resilient my EKS cluster really was. So, I took on a **mini challenge:** delete some nodes and observe how Kubernetes reacts.

In this secret mission, I:

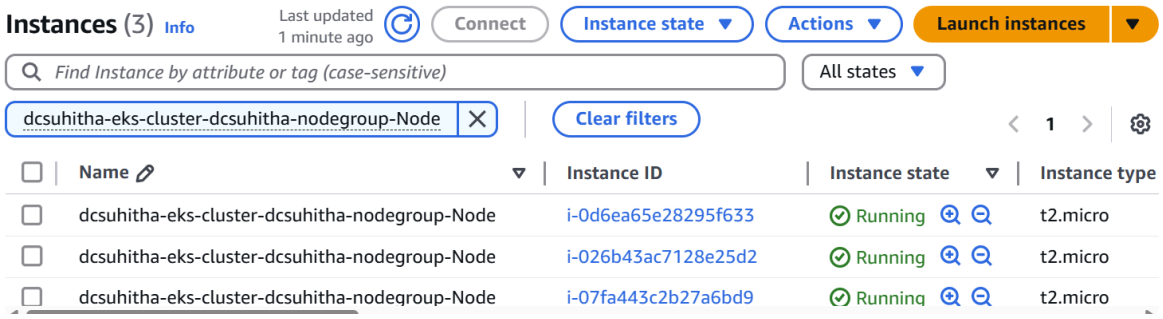
- Terminated EC2 instances in my Kubernetes cluster
- Observed how Kubernetes automatically replaced the terminated instances
- Documented the entire process to showcase the cluster's self-healing capabilities in my project documentation

I headed back into the EC2 console and terminated a node from my node group. Within minutes, Kubernetes (through the managed node group) spun up a replacement automatically. It was fascinating to watch the self-healing feature in action confirming that the **cluster stays stable even when something goes down.**

Terminating Nodes

- I navigated to the **EC2 service** in the AWS Management Console.

There, I located the EC2 instances that were part of my EKS cluster's node group. These instances had names similar to **dcsuhitha-eks-cluster-dcsuhitha-nodegroup-Node**.

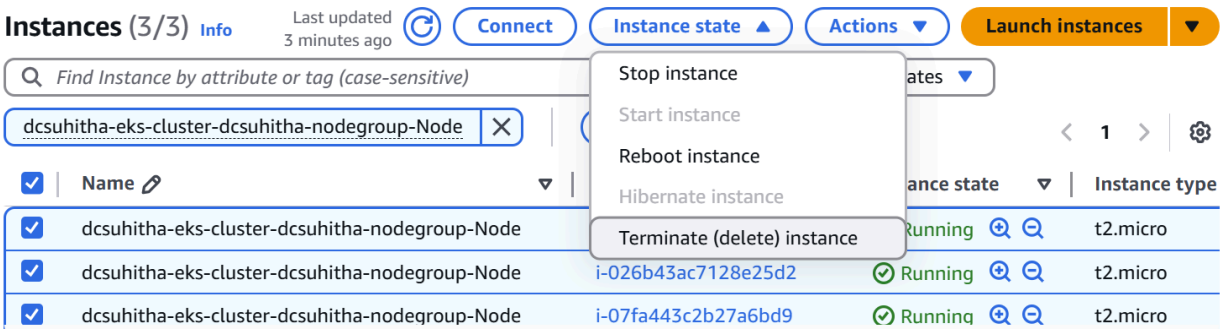


How are Kubernetes and EC2 instances related?

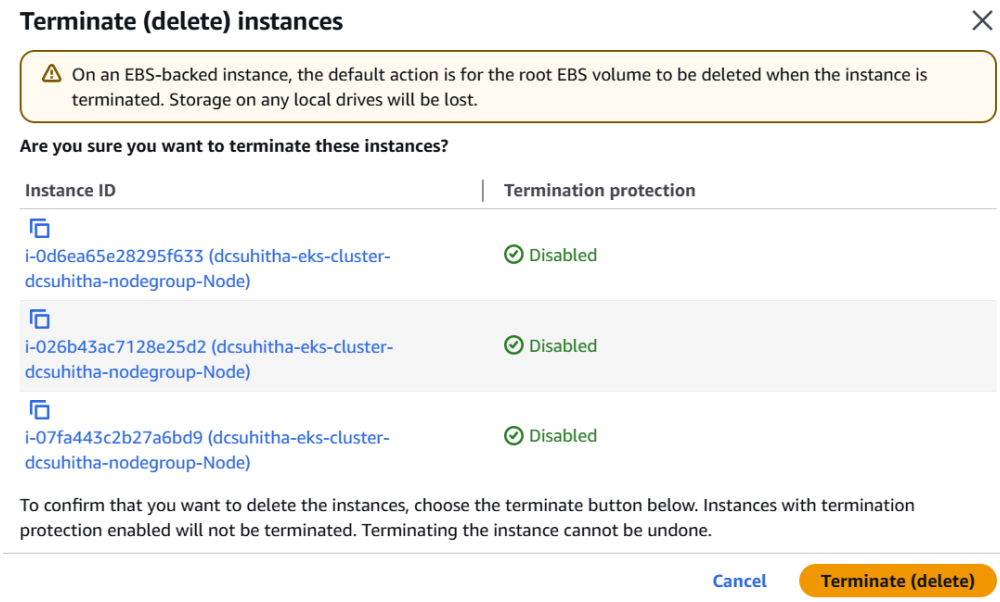
When I create **nodes** in a Kubernetes cluster on AWS, each node is actually an EC2 instance!

Kubernetes uses a generic term (node) because different cloud platforms use different cloud resources to be the node; **in AWS, we use EC2 instances as our nodes.**

- I selected all three nodes, clicked the **Instance state** dropdown, and chose the Terminate **(delete) instance** to remove them from the cluster.

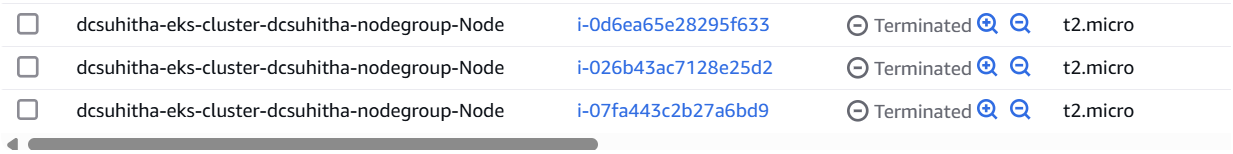


- Selected terminate delete.

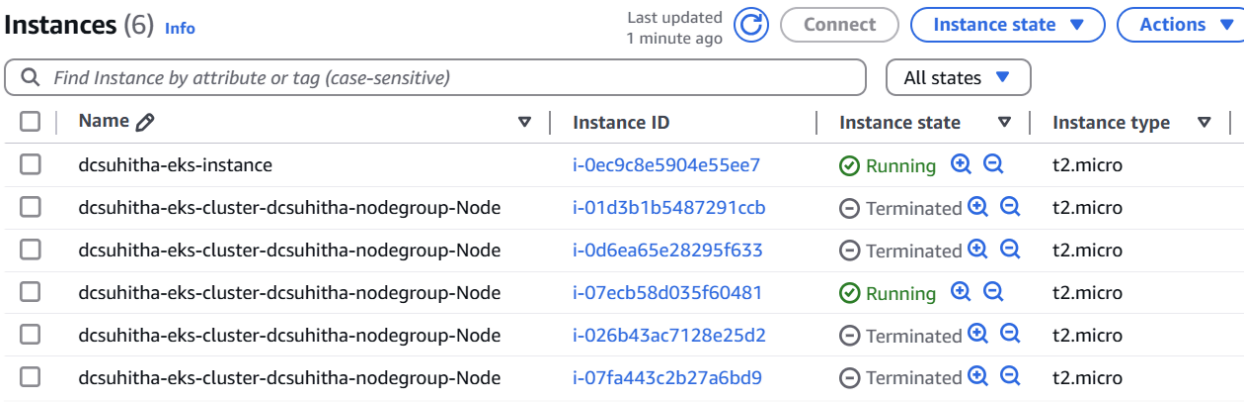


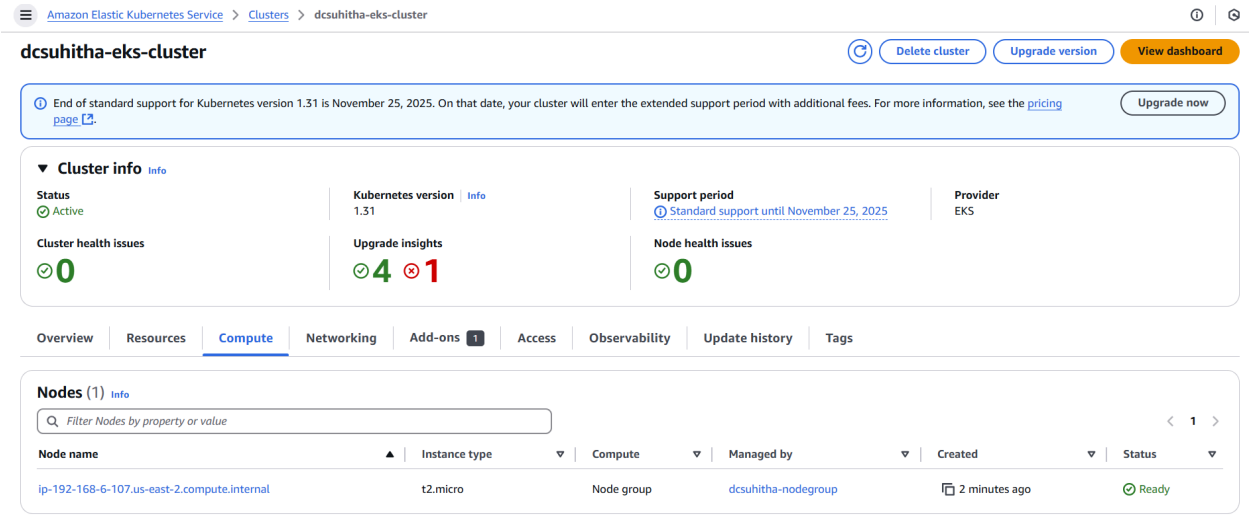
All three nodes showed the **Shutting-down** state initially.

After about a minute, their status changed to **Terminated**.



I then went back to the **EKS console**, refreshed the tab, and checked the **Node Groups** section to see if any new nodes were appearing. Kubernetes was already working to replace the terminated instances.





? Why Could I Find EKS Nodes in the EC2 Console?

Yes I was able to find my EKS cluster's nodes in the **EC2 console** because, in AWS, **each node in a Kubernetes cluster is actually an EC2 instance**. When I created the node group with eksctl, it launched EC2 instances to act as the worker nodes for my cluster.

? What Happened When I Deleted Nodes in a Node Group?

When I deleted the nodes in my node group, **Kubernetes automatically detected the change** and launched new EC2 instances(nodes) to keep the cluster running at its desired state (more on this in a second).As you can see in the above picture.

This is one of the **key benefits** of using Kubernetes – it makes sure that my application is always running, even if some nodes fail.

I selected my node group **dcsuhitha-nodegroup**.

In the **Details** panel, I observed that the **Desired size** was set to **3**, with a **Minimum size** of **1** and a **Maximum size** of **3** nodes. This configuration allows the node group to automatically scale within these limits based on demand.

Capacity type
On-Demand

Desired size
3 nodes

Minimum size
1 node

Maximum size
3 nodes

What Do Desired State, Minimum Size, and Maximum Size Mean?

- **Desired size** is the number of nodes I wanted to run in the node group under normal conditions.
- **Minimum size** is the lowest number of nodes needed to keep the application available, even during low demand.
- **Maximum size** is the highest number of nodes allowed in the group, enabling EKS to scale up during high demand.

Kubernetes uses these settings to **automatically adjust** the number of nodes based on the workload. I can update these values anytime to scale my cluster up or down as needed.

- Once a few minutes have passed since I've terminated my nodes, refresh the **EKS** console again. I noticed that there are new nodes again!

Nodes (3) Info

Filter Nodes by property or value

1

Node name	Instance type	Compute	Managed by	Created	Status
ip-192-168-44-35.us-east-2.compute.internal	t2.micro	Node group	dcsuhitha-nodegroup	13 minutes ago	Ready
ip-192-168-6-107.us-east-2.compute.internal	t2.micro	Node group	dcsuhitha-nodegroup	17 minutes ago	Ready
ip-192-168-92-122.us-east-2.compute.internal	t2.micro	Node group	dcsuhitha-nodegroup	15 minutes ago	Ready

I headed back to the **EC2 console** and refreshed the **Instances** page.

I noticed that **new EC2 instances** had been launched automatically to **replace the ones I had terminated**, confirming that Kubernetes restored the desired node count.