Introduction

What is EKS?

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed Kubernetes service that makes it easy for you to run Kubernetes on AWS and on-premises. Amazon EKS lets you create, update, scale, and terminate nodes for your cluster with a single command. These nodes can also leverage Amazon EC2 Spot Instances to reduce costs.

What is Terraform?

Terraform is an open-source IaC software tool that provides a consistent command line interface (CLI) workflow to manage hundreds of cloud services.

Prerequisites

AWS Account(Free Tier)

AWS CLI

Terraform

Kubectl

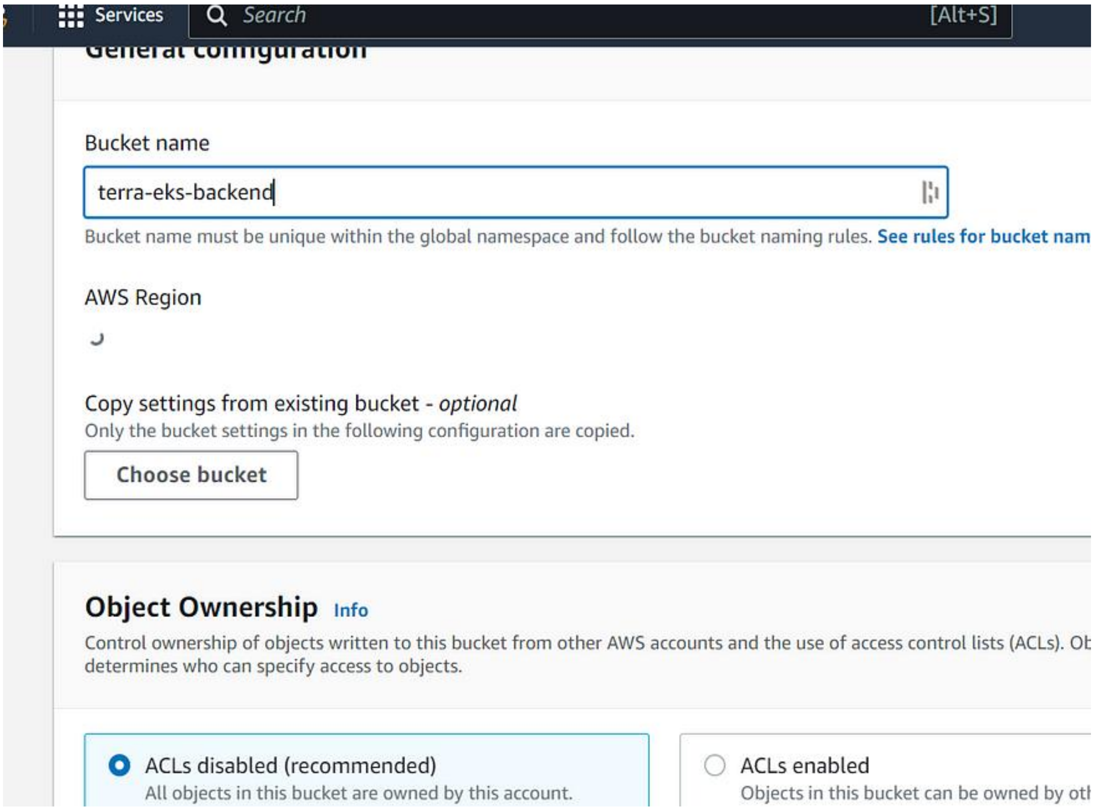
VScode

Let's get started.

Warning!!! You might incur costs in your AWS account by doing this

Head to your AWS account and navigate to the S3 section. This is where we will store our tf.state file as it is considered best practice to keep our state file in a Remote Location. The primary purpose of Terraform state is to store bindings between objects in a remote system and resources declared in your configuration and I am going to keep mine in Amazon S3.

Create an S3 Bucket. You can choose whatever name suits your purpose, I am naming mine *terra-eks-backend*. You can provision your bucket in any region that suits your purpose but I am provisioning mine in the us-east-1



General configuration

Bucket name

terra-eks-backend

Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket nam

AWS Region

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

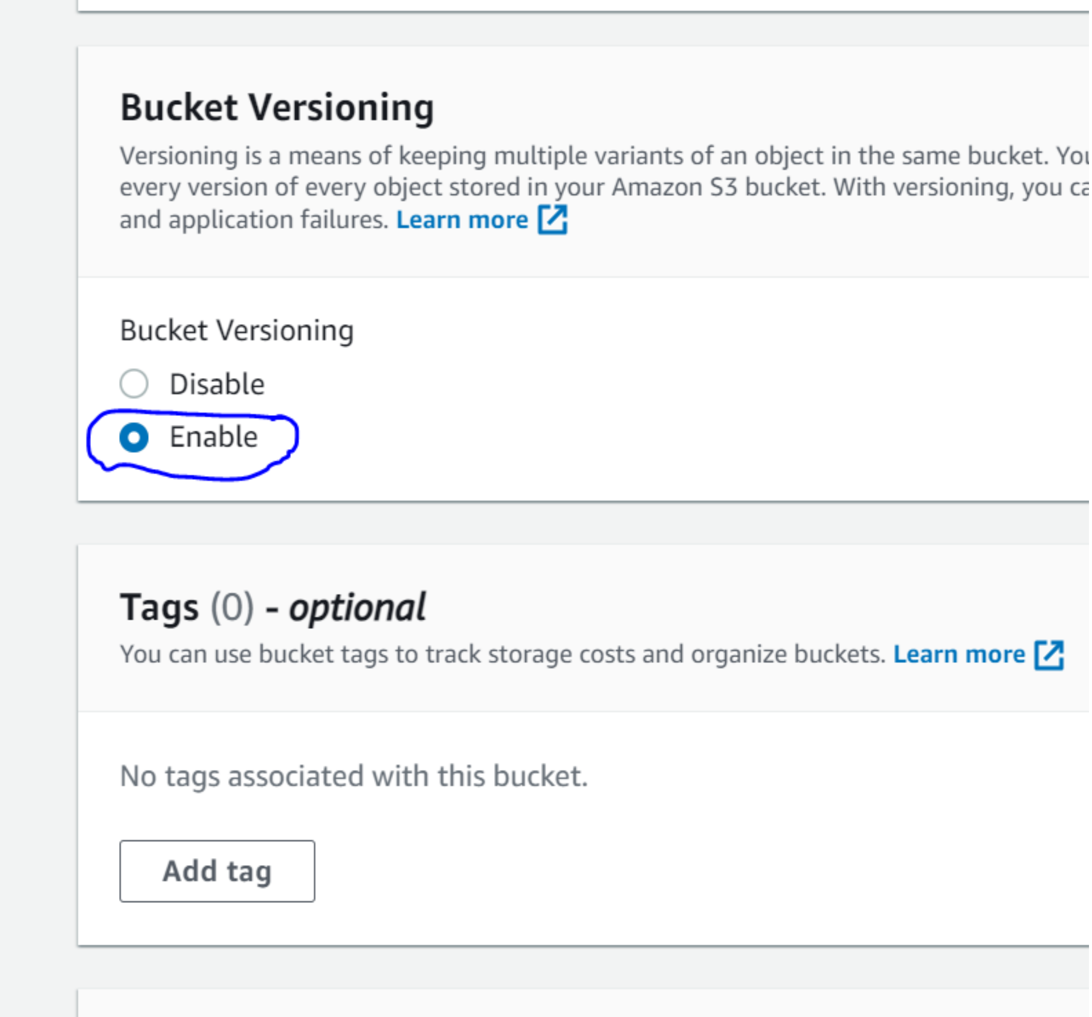
Object Ownership

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Ob
determines who can specify access to objects.

● ACLs disabled (recommended)
All objects in this bucket are owned by this account.

○ ACLs enabled
Objects in this bucket can be owned by ot

It is highly recommended that you enable Bucket Versioning on the S3 bucket to allow for state recovery in the case of accidental deletions and human error.



Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. Yo
every version of every object stored in your Amazon S3 bucket. With versioning, you ca
and application failures. Learn more

Bucket Versioning

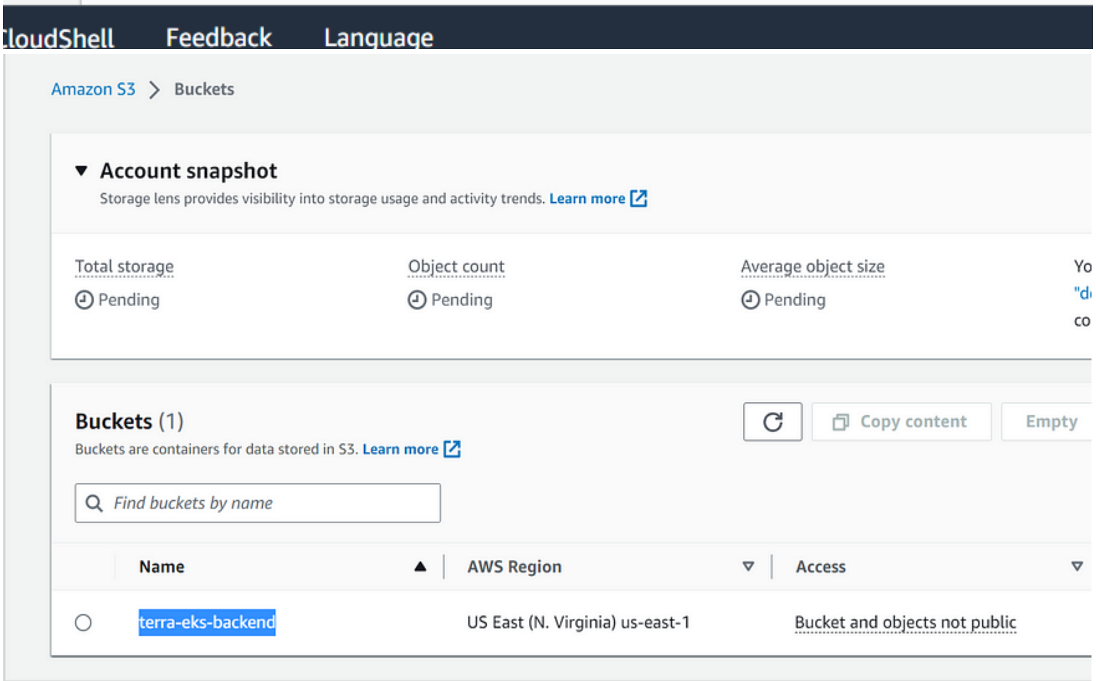
○ Disable

[✓ ● Enable]

Tags (0) - *optional*

You can use bucket tags to track storage costs and organize buckets. Learn more

No tags associated with this bucket.



Account snapshot

Storage lens provides visibility into storage usage and activity trends. Learn more

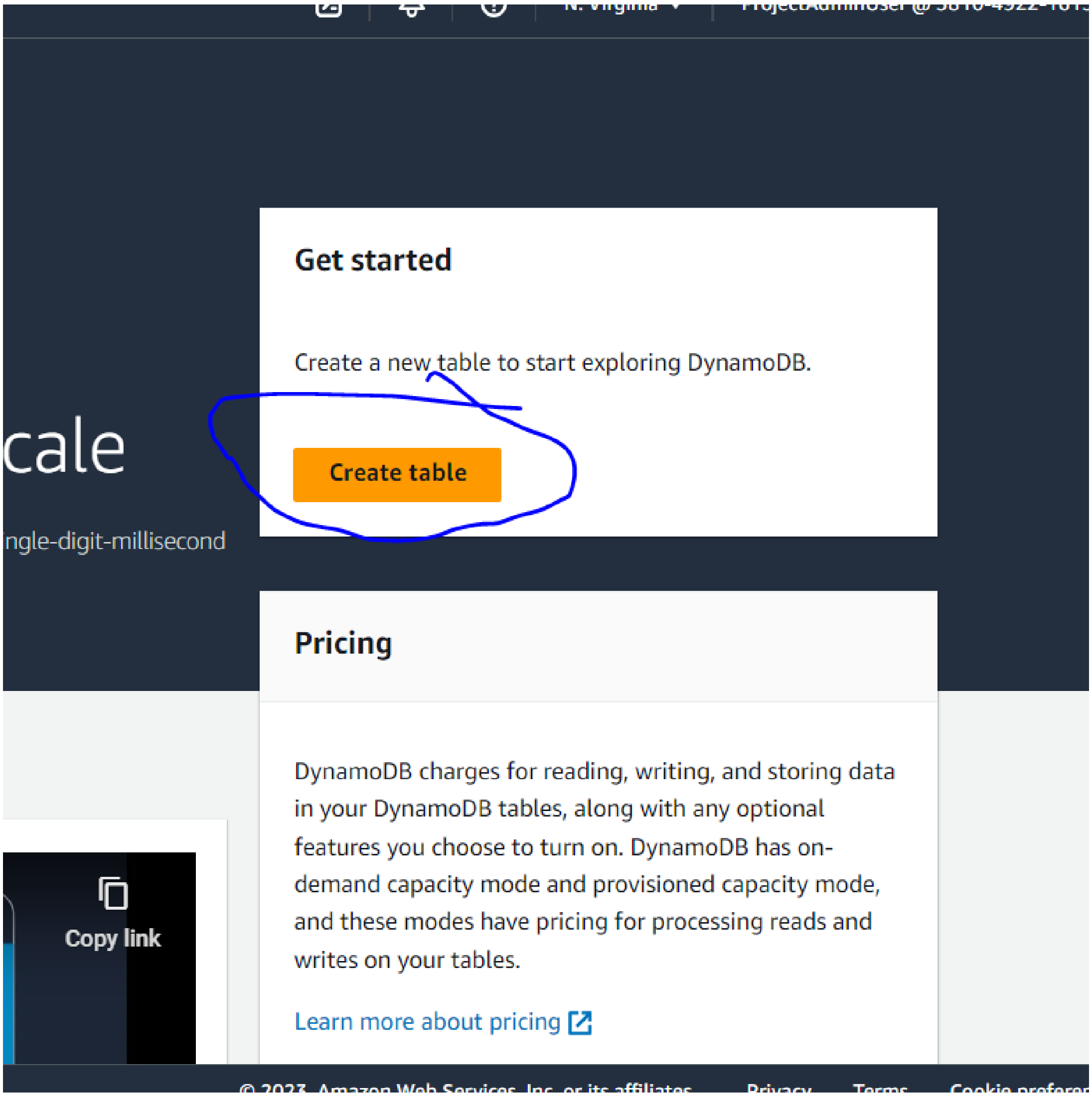
Total storage: Pending
Object count: Pending
Average object size: Pending

Buckets (1)

Buckets are containers for data stored in S3. Learn more

	Name	AWS Region	Access
○	terra-eks-backend	US East (N. Virginia) us-east-1	Bucket and objects not public

To make my tf.state file consistent I am going to enable State-Locking. File locking is a data management feature that allows only one user or process access to a file at any given time. It restricts other users from changing the file while it is being used by another user or process. In other to achieve this I need to create a Dynamo DB Table so let's head to Dynamo DB in our AWS account and create a Dynamo DB table



Get started

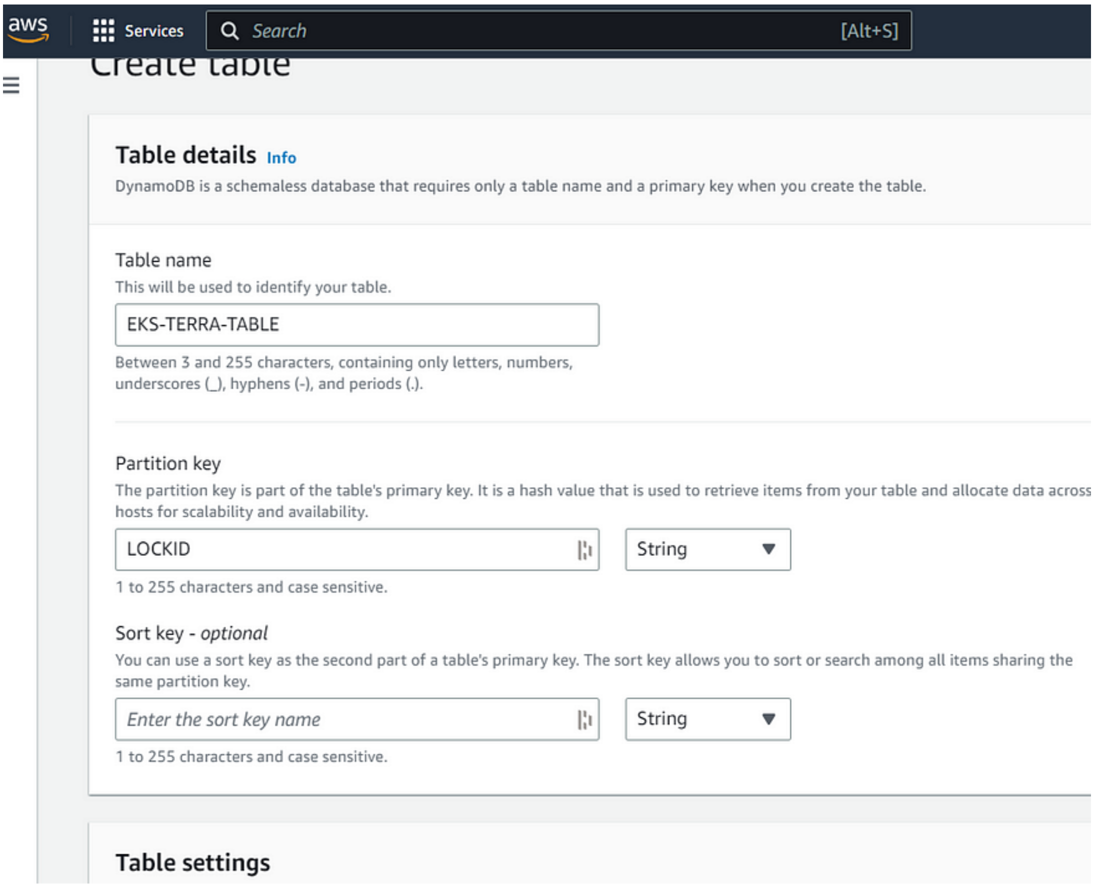
Create a new table to start exploring DynamoDB.

Pricing

DynamoDB charges for reading, writing, and storing data in your DynamoDB tables, along with any optional features you choose to turn on. DynamoDB has on-demand capacity mode and provisioned capacity mode, and these modes have pricing for processing reads and writes on your tables.

Learn more about pricing

© 2023, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferen



Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

EKS-TERRA-TABLE

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

LOCKID | String

1 to 255 characters and case sensitive.

Sort key - *optional*

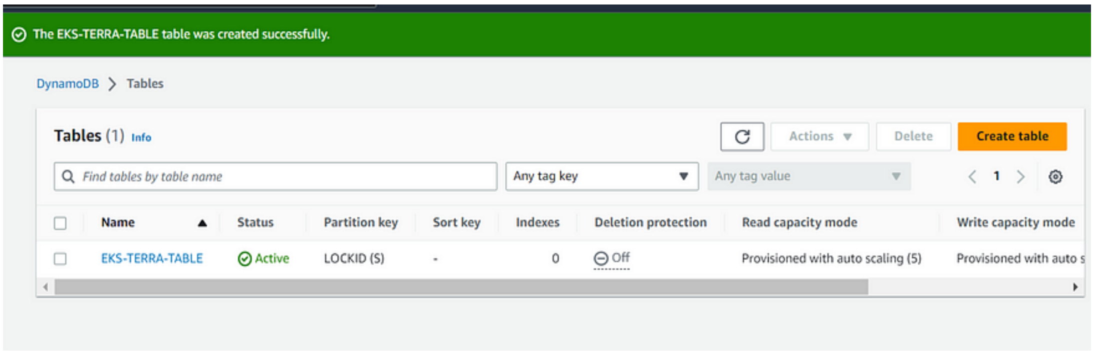
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name | String

1 to 255 characters and case sensitive.

Table settings

You can choose any name for your DynamoDB Table but it is important to note that the partition key is case sensitive. I named the Partition Key LOCKID because this will enable Dynamo DB to lock and release the file. Scroll down the page and click on Create Table



The EKS-TERRA-TABLE table was created successfully.

Tables (1) Info

☐	Name	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mode	Write capacity mode
☐	EKS-TERRA-TABLE	Active	LOCKID (S)	-	0	Off	Provisioned with auto scaling (5)	Provisioned with auto s

CREATE TERRAFORM FILES

eks-backend-terra.tf

```
terraform {


  backend "s3" {
    bucket = "BUCKET_NAME"

    key    = "backend/FILE_NAME_TO_STORE_STATE.tfstate"

    region = "us-east-1"

    dynamodb_table = "dynamoDB_TABLE_NAME"

  }

}
```

Provider.tf

Name of provider *AWS*

Our source will be defined as *hashicorp/aws*. This is a short abbreviation for `registry.terraform.io/hashicorp/aws`

The version is set to ~>4.66.1

Region is us-east-1

```
terraform {

  required_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "4.66.1"
```

```
        }

    }

}

provider "aws" {

    region = "us-east-1"

}
```

Vpc.tf

```
resource "aws_vpc" "main" {

    cidr_block = "10.0.0.0/16"

    tags = {

        Name = "PC-VPC"

    }

}
```

The vpc.tf contains codes to create a new Vpc. The CIDR block is 10.0.0.0/16 and I have tagged the name of the VPC as PC-VPC

Subnets.tf

```
resource "aws_subnet" "public-1" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true
  tags = {
    Name = "public-sub-1"
  }
}
resource "aws_subnet" "public-2" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.2.0/24"
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = true
```

```
    tags = {
        Name = "public-sub-2"
    }
}
```

EKS requires a minimum of two subnets to function so this is creating two public subnets in two availability zones namely us-east-1a and us-east-1b respectively. I have also set *map public IP on launch = true*. This will assign our subnets with public IPV4 addresses. I have given the CIDR range of 10.0.1.0/24 to public-1 and 10.0.2.0/24 to public-2.

Internetgw.tf

```
resource "aws_internet_gateway" "gw" {
    vpc_id = aws_vpc.main.id
    tags = {
        Name = "main"
    }
}
```

This will create and attach the internet gateway to the Vpc created

Rout.tf

```
resource "aws_route_table" "rtb" {
  vpc_id = aws_vpc.main.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.gw.id
  }
  tags = {
    Name = "MyRoute"
  }
}
resource "aws_route_table_association" "a-1" {
  subnet_id      = aws_subnet.public-1.id
  route_table_id = aws_route_table.rtb.id
}
```

```
resource "aws_route_table_association" "a-2" {
  subnet_id      = aws_subnet.public-2.id
  route_table_id = aws_route_table.rtb.id
}
```

This will create the Route table. The route table has been associated with the two subnets (public-1,public-2). An internet gateway id has also been associated with the Route table and the Vpc id has been assigned to the Route table.

Sg.tf

```
resource "aws_security_group" "allow_tls" {
  name        = "allow_tls"
  description = "Allow TLS inbound traffic"
  vpc_id      = aws_vpc.main.id
  ingress {
    description = "TLS from VPC"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
```

```
        cidr_blocks = ["0.0.0.0/0"]
    }
    egress {
        from_port   = 0
        to_port     = 0
        protocol    = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }
    tags = {
        Name = "allow_tls"
    }
}
```

This will create the security group attached to the created Vpc with both ingress and egress rules. Ingress rules allow incoming(inbound) connection with our Vpc while egress rule allows outgoing (outbound) connection.

iam_role.tf

```
resource "aws_iam_role" "master" {
  name = "ed-eks-master"

  assume_role_policy = <<POLICY

{

  "Version": "2012-10-17",

  "Statement": [

    {

      "Effect": "Allow",

      "Principal": {

        "Service": "eks.amazonaws.com"

      },

      "Action": "sts:AssumeRole"

    }

  ]

}

POLICY
```

```
}
resource "aws_iam_role_policy_attachment" "AmazonEKSClusterPolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.master.name
}
resource "aws_iam_role_policy_attachment" "AmazonEKSServicePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSServicePolicy"
  role       = aws_iam_role.master.name
}
resource "aws_iam_role_policy_attachment" "AmazonEKSVPCResourceController" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
  role       = aws_iam_role.master.name
}
resource "aws_iam_role" "worker" {
  name = "ed-eks-worker"
  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
POLICY
}
```

```
resource "aws_iam_policy" "autoscaler" {
    name   = "ed-eks-autoscaler-policy"
    policy = <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeTags",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:SetDesiredCapacity",
                "autoscaling:TerminateInstanceInAutoScalingGroup",
                "ec2:DescribeLaunchTemplateVersions"
```

```
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
EOF
}
resource "aws_iam_role_policy_attachment" "AmazonEKSWorkerNodePolicy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  role       = aws_iam_role.worker.name
}
resource "aws_iam_role_policy_attachment" "AmazonEKS_CNI_Policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  role       = aws_iam_role.worker.name
```

```
}
resource "aws_iam_role_policy_attachment" "AmazonSSMManagedInstanceCore" {
    policy_arn = "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
    role       = aws_iam_role.worker.name
}
resource "aws_iam_role_policy_attachment" "AmazonEC2ContainerRegistryReadOnly" {
    policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
    role       = aws_iam_role.worker.name
}
resource "aws_iam_role_policy_attachment" "x-ray" {
    policy_arn = "arn:aws:iam::aws:policy/AWSXRayDaemonWriteAccess"
    role       = aws_iam_role.worker.name
}
resource "aws_iam_role_policy_attachment" "s3" {
    policy_arn = "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
    role       = aws_iam_role.worker.name

}

resource "aws_iam_role_policy_attachment" "autoscaler" {

    policy_arn = aws_iam_policy.autoscaler.arn

    role       = aws_iam_role.worker.name

}

resource "aws_iam_instance_profile" "worker" {

    depends_on = [aws_iam_role.worker]

    name       = "ed-eks-worker-new-profile"

    role       = aws_iam_role.worker.name

}
```

This will create all the necessary IAM roles and Policies for the EKS cluster. It will be attached to the EKS cluster once the EKS Cluster is created. This will enable the EKS Cluster to have all the necessary permissions needed.

eks_cluster.tf

```
resource "aws_eks_cluster" "eks" {

    name     = "pc-eks"
    role_arn = aws_iam_role.master.arn

    vpc_config {

        subnet_ids = [aws_subnet.public-1.id, aws_subnet.public-2.id]

    }

    depends_on = [

        aws_iam_role_policy_attachment.AmazonEKSClusterPolicy,

        aws_iam_role_policy_attachment.AmazonEKSServicePolicy,

        aws_iam_role_policy_attachment.AmazonEKSVPCResourceController,

        aws_iam_role_policy_attachment.AmazonEKSVPCResourceController,

        #aws_subnet.pub_sub1,

        #aws_subnet.pub_sub2,

    ]

}
```

This will create the EKS Cluster. *depends_on =[* means that the EKS Cluster being created depends on the completion of the creation of the IAM roles. The two subnet IDs have also been mentioned here as well.

eks_node_group.tf

```
resource "aws_instance" "kubectl-server" {
    ami                         = "ami-06ca3ca175f37dd66"
    key_name                    = "EKSKEYPAIR"
    instance_type               = "t2.micro"
    associate_public_ip_address = true
    subnet_id                   = aws_subnet.public-1.id
    vpc_security_group_ids      = [aws_security_group.allow_tls.id]
    tags = {
        Name = "kubectl"
    }
}
resource "aws_eks_node_group" "node-grp" {
    cluster_name    = aws_eks_cluster.eks.name
    node_group_name = "pc-node-group"
```

```
node_role_arn   = aws_iam_role.worker.arn

subnet_ids      = [aws_subnet.public-1.id, aws_subnet.public-2.id]

capacity_type   = "ON_DEMAND"

disk_size       = "20"

instance_types  = ["t2.small"]

remote_access {

  ec2_ssh_key               = "EKSKEYPAIR"

  source_security_group_ids = [aws_security_group.allow_tls.id]

}

labels = tomap({ env = "dev" })

scaling_config {

  desired_size = 2

  max_size     = 3

  min_size     = 1

}
```

```
    update_config {

        max_unavailable = 1

    }

    depends_on = [

        aws_iam_role_policy_attachment.AmazonEKSWorkerNodePolicy,

        aws_iam_role_policy_attachment.AmazonEKS_CNI_Policy,

        aws_iam_role_policy_attachment.AmazonEC2ContainerRegistryReadOnly,

        #aws_subnet.pub_sub1,

        #aws_subnet.pub_sub2,

    ]

}
```

This will create 2 resources.

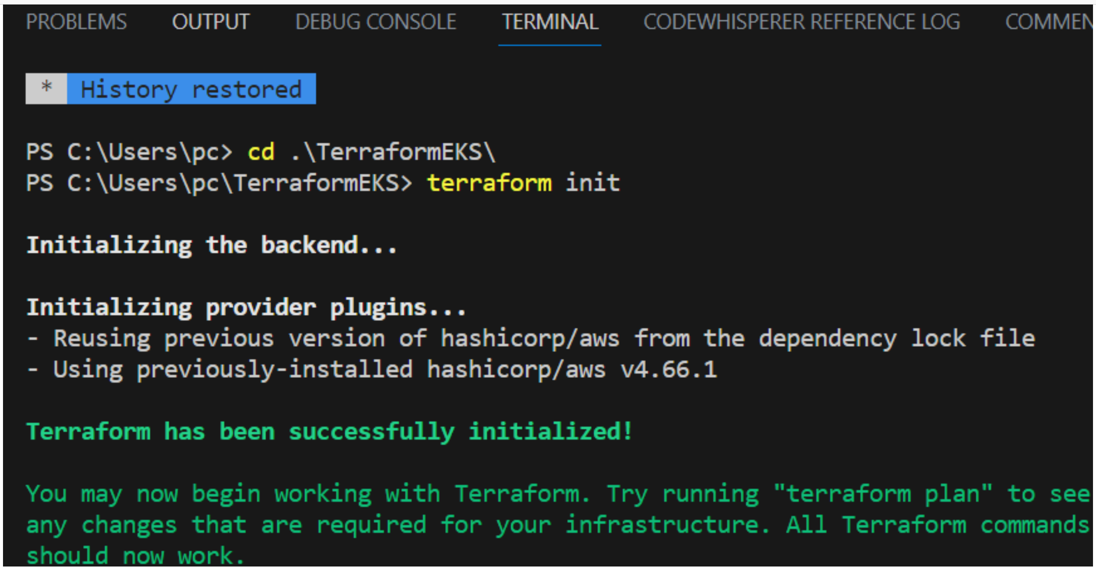
The first block will create an EC2 instance for the Kubectl server.

Note that you need a key pair in your AWS account in the region you are deploying this in other to be able to ssh into the kubectl server. If you do not have one then you will need to create one.

The second block will create the AWS EKS node group, capacity type is set to *ON DEMAND,* instance type is t2.small, and disk size is set to 20. Scaling Config, maximum size is 3, desired size is 2, and minimum size is set to 1.

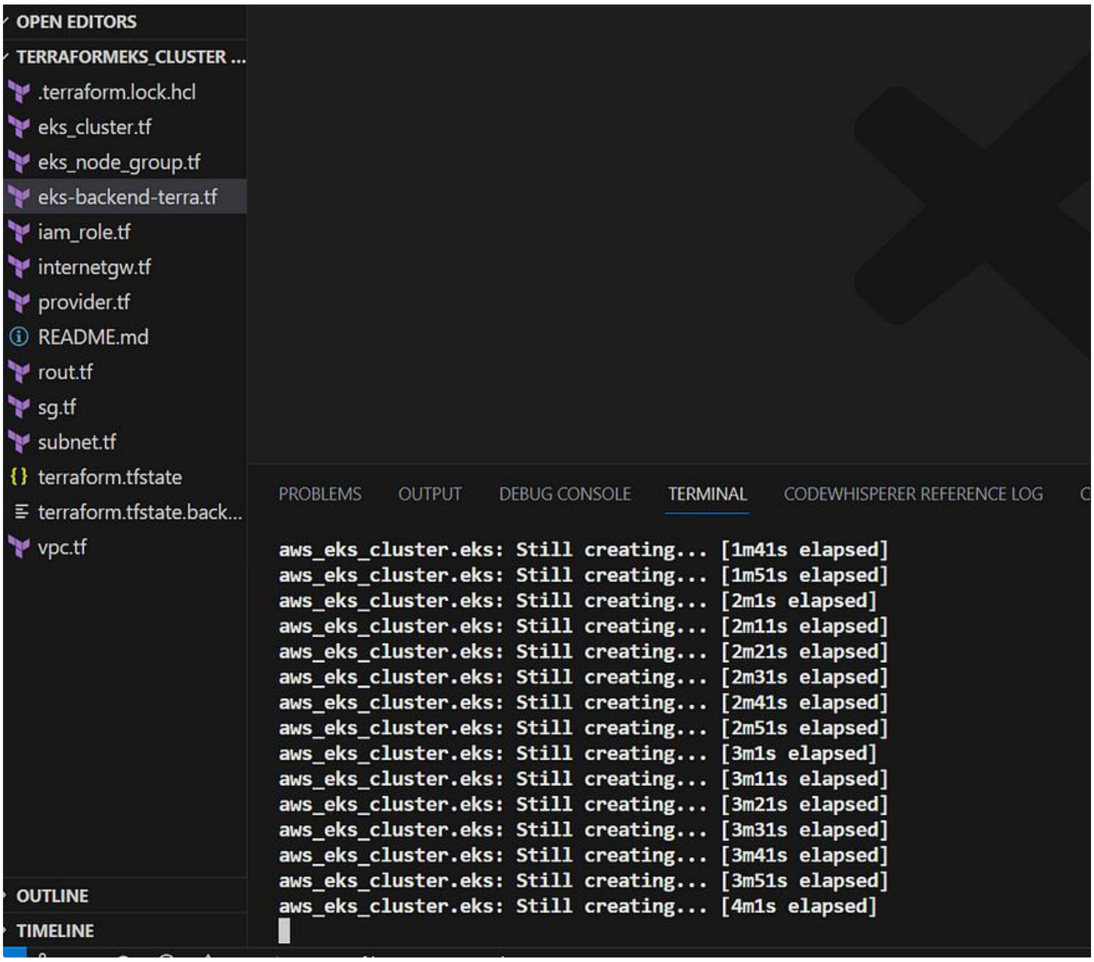
Now that the Terraform codes are ready its time to run our Terraform commands

```
terraform init
```



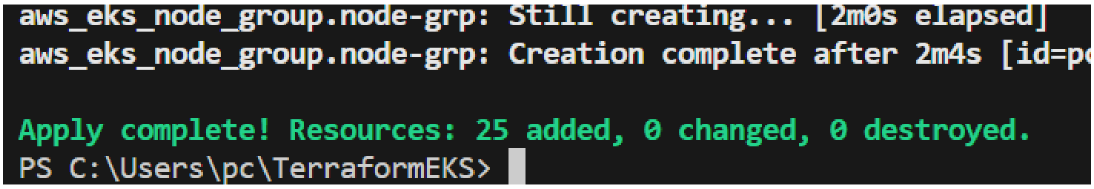
```
 * History restored

PS C:\Users\pc> cd .\TerraformEKS\
PS C:\Users\pc\TerraformEKS> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.66.1

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

```
terraform validate

terraform plan

terraform apply
```



```
aws_eks_cluster.eks: Still creating... [1m41s elapsed]
aws_eks_cluster.eks: Still creating... [1m51s elapsed]
aws_eks_cluster.eks: Still creating... [2m1s elapsed]
aws_eks_cluster.eks: Still creating... [2m11s elapsed]
aws_eks_cluster.eks: Still creating... [2m21s elapsed]
aws_eks_cluster.eks: Still creating... [2m31s elapsed]
aws_eks_cluster.eks: Still creating... [2m41s elapsed]
aws_eks_cluster.eks: Still creating... [2m51s elapsed]
aws_eks_cluster.eks: Still creating... [3m1s elapsed]
aws_eks_cluster.eks: Still creating... [3m11s elapsed]
aws_eks_cluster.eks: Still creating... [3m21s elapsed]
aws_eks_cluster.eks: Still creating... [3m31s elapsed]
aws_eks_cluster.eks: Still creating... [3m41s elapsed]
aws_eks_cluster.eks: Still creating... [3m51s elapsed]
aws_eks_cluster.eks: Still creating... [4m1s elapsed]
```

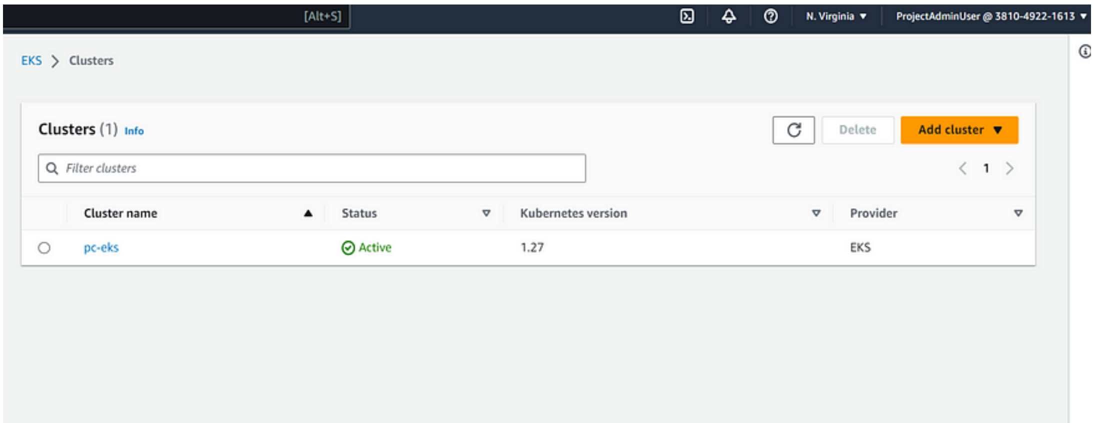
Terraform is creating the resources after running the terraform apply command.



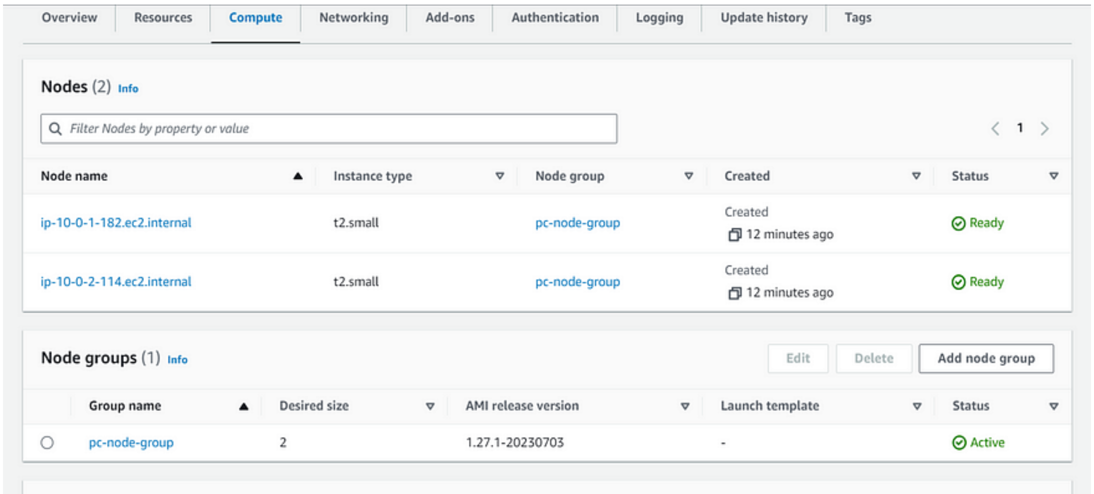
```
aws_eks_node_group.node-grp: Still creating... [2m0s elapsed]
aws_eks_node_group.node-grp: Creation complete after 2m4s [id=p

Apply complete! Resources: 25 added, 0 changed, 0 destroyed.
PS C:\Users\pc\TerraformEKS>
```

All resources have been created successfully. Let's check our AWS Account

EKS Cluster created



	Cluster name	Status	Kubernetes version	Provider
○	pc-eks	Active	1.27	EKS



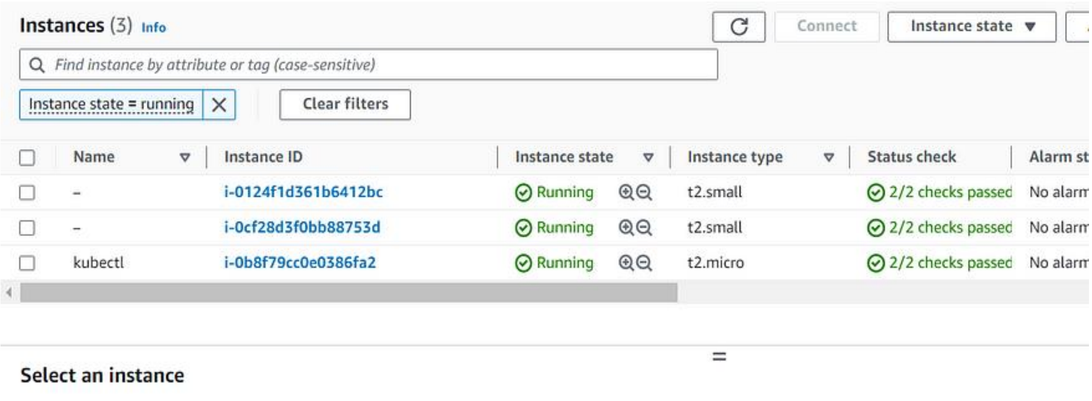
Nodes (2) Info

Node name	Instance type	Node group	Created	Status
ip-10-0-1-182.ec2.internal	t2.small	pc-node-group	Created 12 minutes ago	✓ Ready
ip-10-0-2-114.ec2.internal	t2.small	pc-node-group	Created 12 minutes ago	✓ Ready

Node groups (1) Info

	Group name	Desired size	AMI release version	Launch template	Status
○	pc-node-group	2	1.27.1-20230703	-	✓ Active

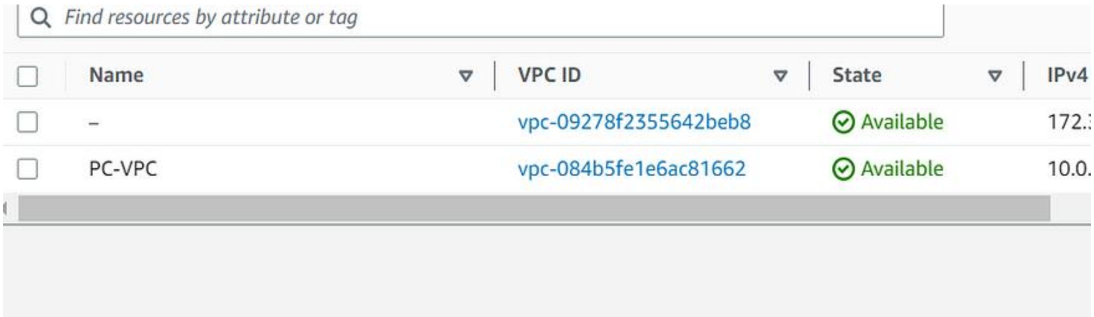
Let's check the Kubectl Server and the other instances created


Instances (3) Info

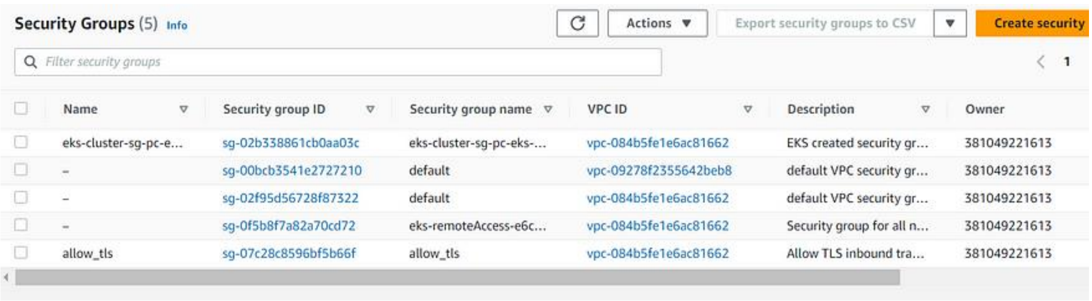
□	Name	Instance ID	Instance state	Instance type	Status check	Alarm st
□	–	i-0124f1d361b6412bc	✓ Running	t2.small	✓ 2/2 checks passed	No alarm
□	–	i-0cf28d3f0bb88753d	✓ Running	t2.small	✓ 2/2 checks passed	No alarm
□	kubectl	i-0b8f79cc0e0386fa2	✓ Running	t2.micro	✓ 2/2 checks passed	No alarm

Select an instance

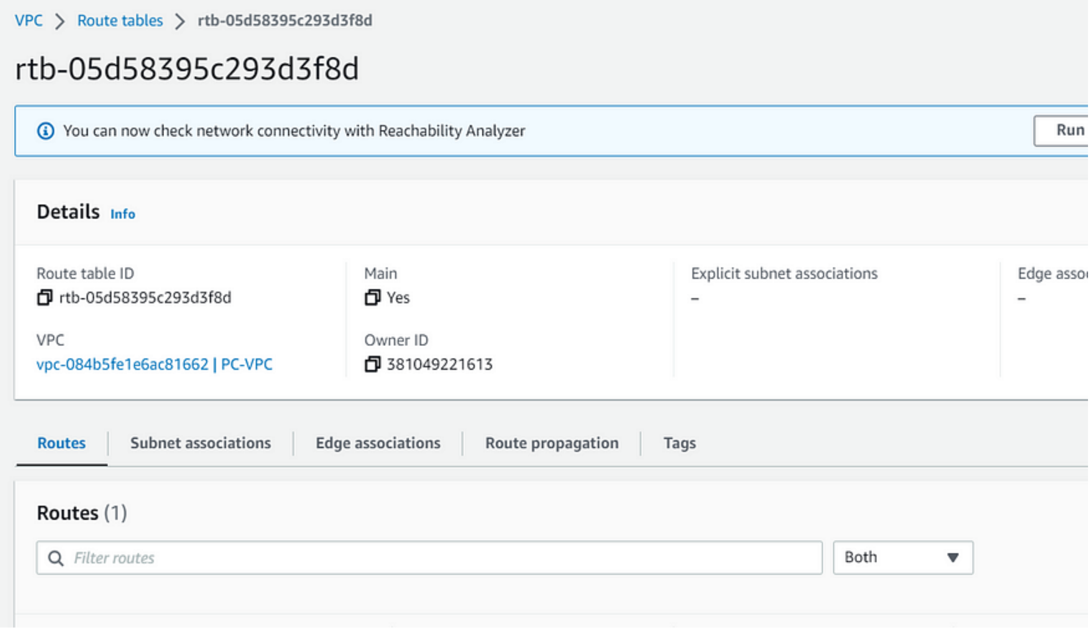
Let's check the VPC and Security Group, Route Table, and Subnets



□	Name	VPC ID	State	IPv4
□	–	vpc-09278f2355642beb8	✓ Available	172.3
□	PC-VPC	vpc-084b5fe1e6ac81662	✓ Available	10.0.


Security Groups (5) Info

□	Name	Security group ID	Security group name	VPC ID	Description	Owner
□	eks-cluster-sg-pc-e...	sg-02b338861cb0aa03c	eks-cluster-sg-pc-eks-...	vpc-084b5fe1e6ac81662	EKS created security gr...	381049221613
□	–	sg-00bcb3541e2727210	default	vpc-09278f2355642beb8	default VPC security gr...	381049221613
□	–	sg-02f95d56728f87322	default	vpc-084b5fe1e6ac81662	default VPC security gr...	381049221613
□	–	sg-0f5b8f7a82a70cd72	eks-remoteAccess-e6c...	vpc-084b5fe1e6ac81662	Security group for all n...	381049221613
□	allow_tls	sg-07c28c8596bf5b66f	allow_tls	vpc-084b5fe1e6ac81662	Allow TLS inbound tra...	381049221613



VPC > Route tables > rtb-05d58395c293d3f8d

rtb-05d58395c293d3f8d

You can now check network connectivity with Reachability Analyzer

Details Info

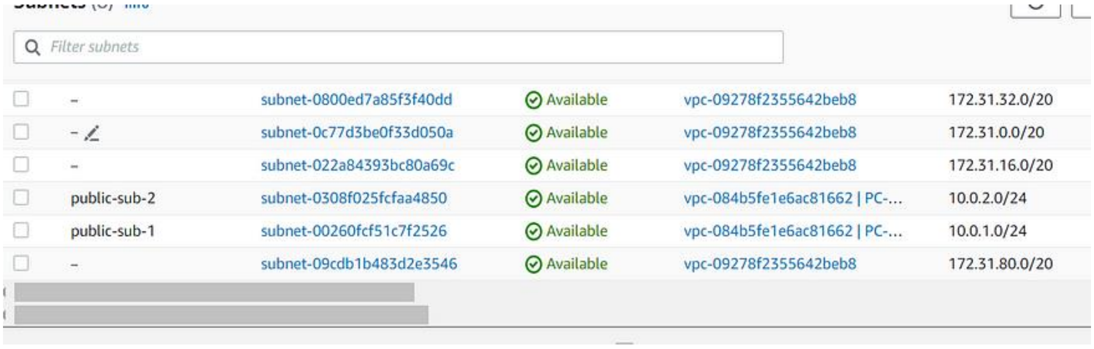
Route table ID	Main	Explicit subnet associations	Edge asso
rtb-05d58395c293d3f8d	Yes	–	–
VPC	Owner ID		
vpc-084b5fe1e6ac81662 | PC-VPC	381049221613		

Routes | Subnet associations | Edge associations | Route propagation | Tags

Routes (1)

Filter routes

Both



Filter subnets

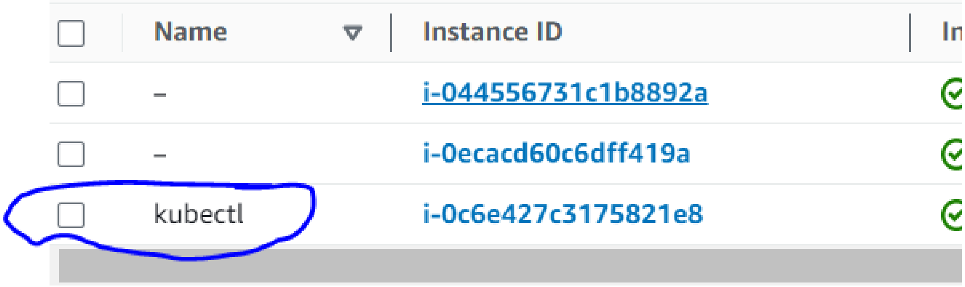
□	–	subnet-0800ed7a85f3f40dd	Available	vpc-09278f2355642beb8	172.31.32.0/20
□	–	subnet-0c77d3be0f33d050a	Available	vpc-09278f2355642beb8	172.31.0.0/20
□	–	subnet-022a84393bc80a69c	Available	vpc-09278f2355642beb8	172.31.16.0/20
□	public-sub-2	subnet-0308f025fcfaa4850	Available	vpc-084b5fe1e6ac81662 | PC-...	10.0.2.0/24
□	public-sub-1	subnet-00260fcf51c7f2526	Available	vpc-084b5fe1e6ac81662 | PC-...	10.0.1.0/24
□	–	subnet-09cdb1b483d2e3546	Available	vpc-09278f2355642beb8	172.31.80.0/20

Now that all resources have been provisioned by Terraform next step is to try to ssh into the Kubectl server.

I am using a newer version of Windows so I do not need to ssh with putty. If you are using an older version of Windows you will need to ssh into your instance using putty. You can learn how to do that here

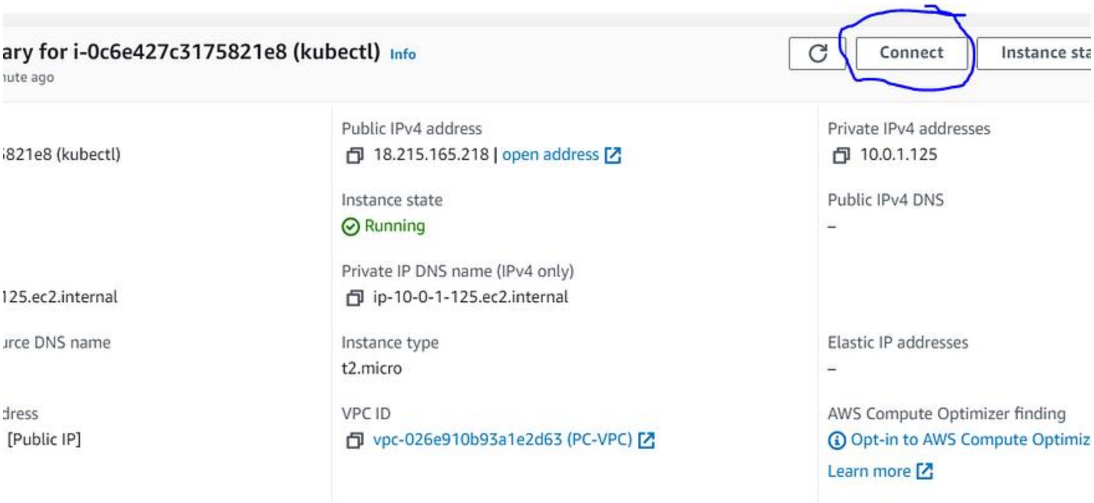
Connect to your Linux instance from Windows using PuTTY
Connect to your Linux instances from Windows using PuTTY, a free SSH client for Windows.
docs.aws.amazon.com

Click on the Instance Kubectl



□	Name	Instance ID	In
□	–	i-044556731c1b8892a	
□	–	i-0ecacd60c6dff419a	
□	kubectl	i-0c6e427c3175821e8	

On the next page click on connect



ary for i-0c6e427c3175821e8 (kubectl) Info
ute ago

5821e8 (kubectl)

Public IPv4 address
18.215.165.218 | open address

Private IPv4 addresses
10.0.1.125

Instance state
Running

Public IPv4 DNS
–

Private IP DNS name (IPv4 only)
ip-10-0-1-125.ec2.internal

125.ec2.internal

rce DNS name

Instance type
t2.micro

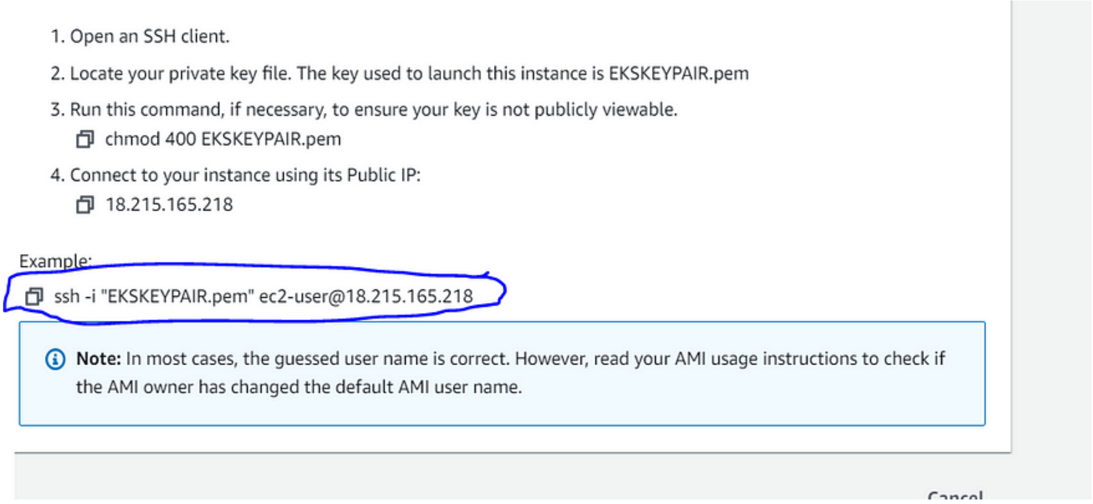
Elastic IP addresses
–

dress
[Public IP]

VPC ID
vpc-026e910b93a1e2d63 (PC-VPC)

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimiz
Learn more

Copy the ssh — i under Example



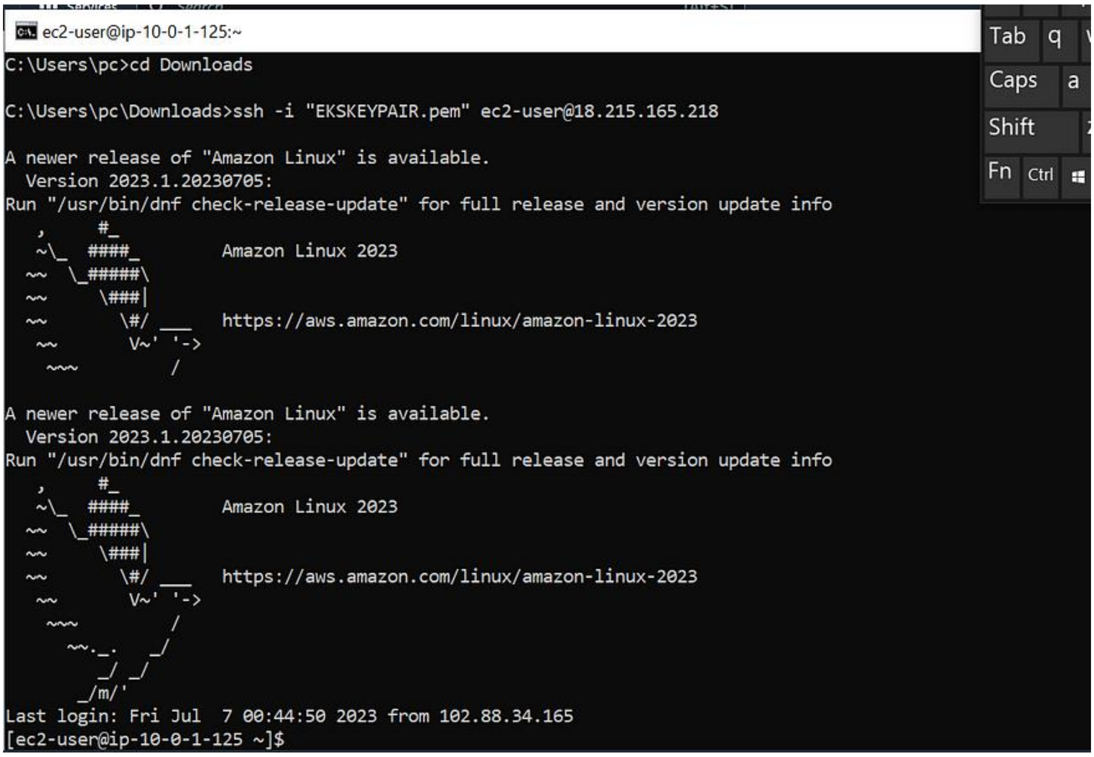
1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is EKSKEYPAIR.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
   chmod 400 EKSKEYPAIR.pem
4. Connect to your instance using its Public IP:
   18.215.165.218

Example:

```
ssh -i "EKSKEYPAIR.pem" ec2-user@18.215.165.218
```

Note: In most cases, the guessed user name is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

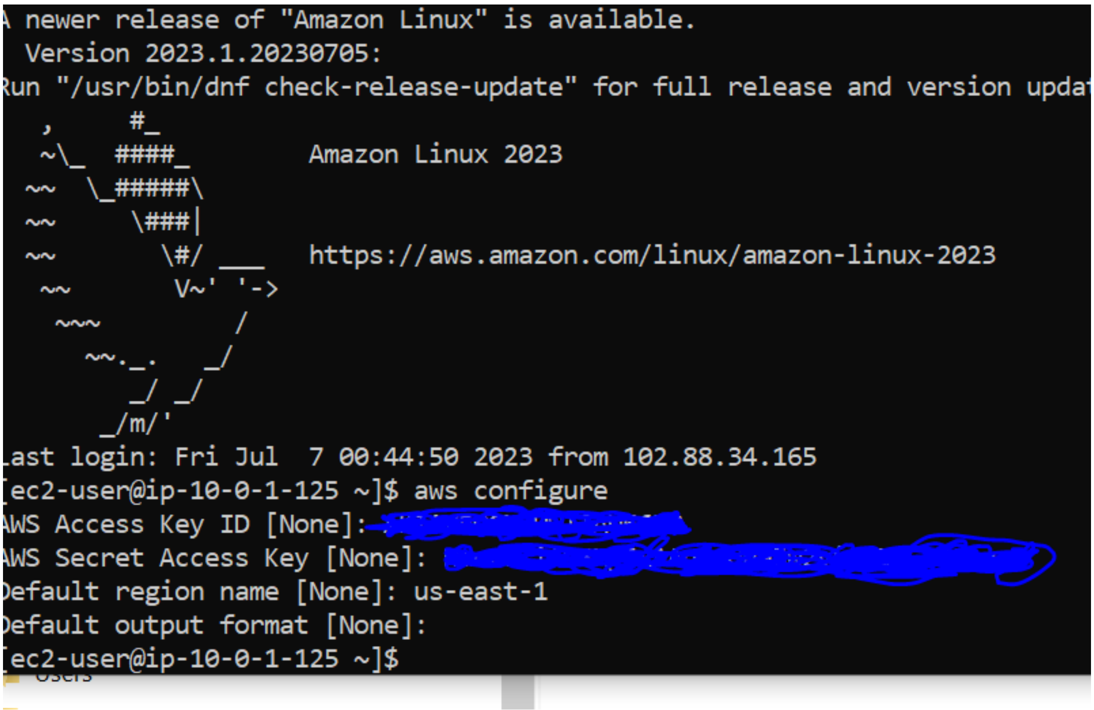
Now head to your cmd line on your machine and cd into the directory where your Keypair(.pem) is stored. In my case, it's stored in the Downloads directory. Yours could be different so take note. Once you are in the directory where your Keypair is stored paste in the ssh line you copied and press enter



```
C:\Users\pc>cd Downloads

C:\Users\pc\Downloads>ssh -i "EKSKEYPAIR.pem" ec2-user@18.215.165.218

A newer release of "Amazon Linux" is available.
  Version 2023.1.20230705:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /

A newer release of "Amazon Linux" is available.
  Version 2023.1.20230705:
Run "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Fri Jul  7 00:44:50 2023 from 102.88.34.165
[ec2-user@ip-10-0-1-125 ~]$
```

I had connected to mine earlier so there was no prompt but if you are connecting for the first time there will be a prompt. Just type yes and you will be connected to your ec2 instance.

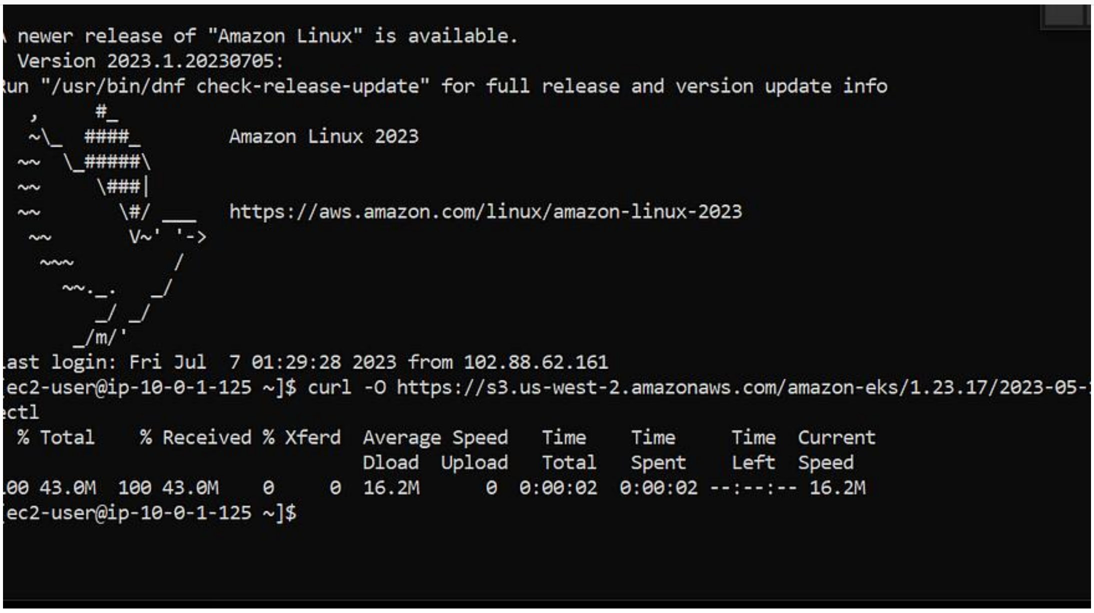
Let's install AWS CLI on the Kubectl Server



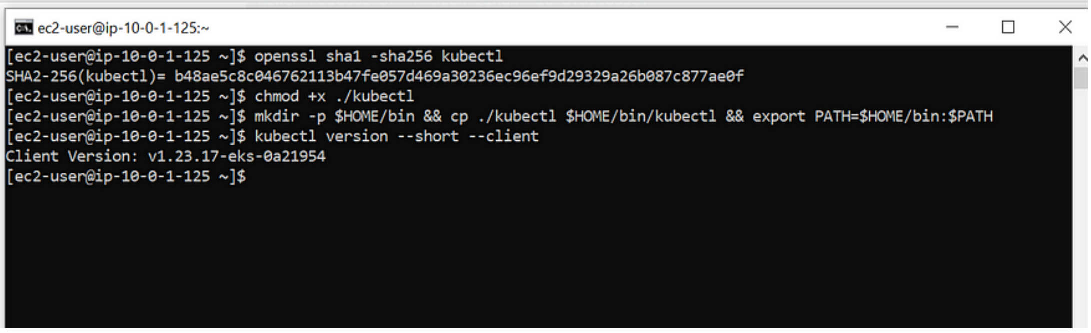
```
 newer release of "Amazon Linux" is available.
  Version 2023.1.20230705:
Run "/usr/bin/dnf check-release-update" for full release and version updat
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
ast login: Fri Jul  7 00:44:50 2023 from 102.88.34.165
ec2-user@ip-10-0-1-125 ~]$ aws configure
WS Access Key ID [None]: [redacted]
WS Secret Access Key [None]: [redacted]
efault region name [None]: us-east-1
efault output format [None]:
ec2-user@ip-10-0-1-125 ~]$
```

Next up is to set up Kubectl on the ec2 instance. For that, we will run these command

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.1/2023-04-19/bin/linux/amd64/kubectl

openssl sha1 -sha256 kubectl

chmod +x ./kubectl

mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH

kubectl version --short --client
```



```
 newer release of "Amazon Linux" is available.
 Version 2023.1.20230705:
un "/usr/bin/dnf check-release-update" for full release and version update info
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
ast login: Fri Jul  7 01:29:28 2023 from 102.88.62.161
ec2-user@ip-10-0-1-125 ~]$ curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.23.17/2023-05-
ctl
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                Dload  Upload   Total   Spent    Left  Speed
00 43.0M  100 43.0M    0     0  16.2M      0  0:00:02  0:00:02 --:--:-- 16.2M
ec2-user@ip-10-0-1-125 ~]$
```

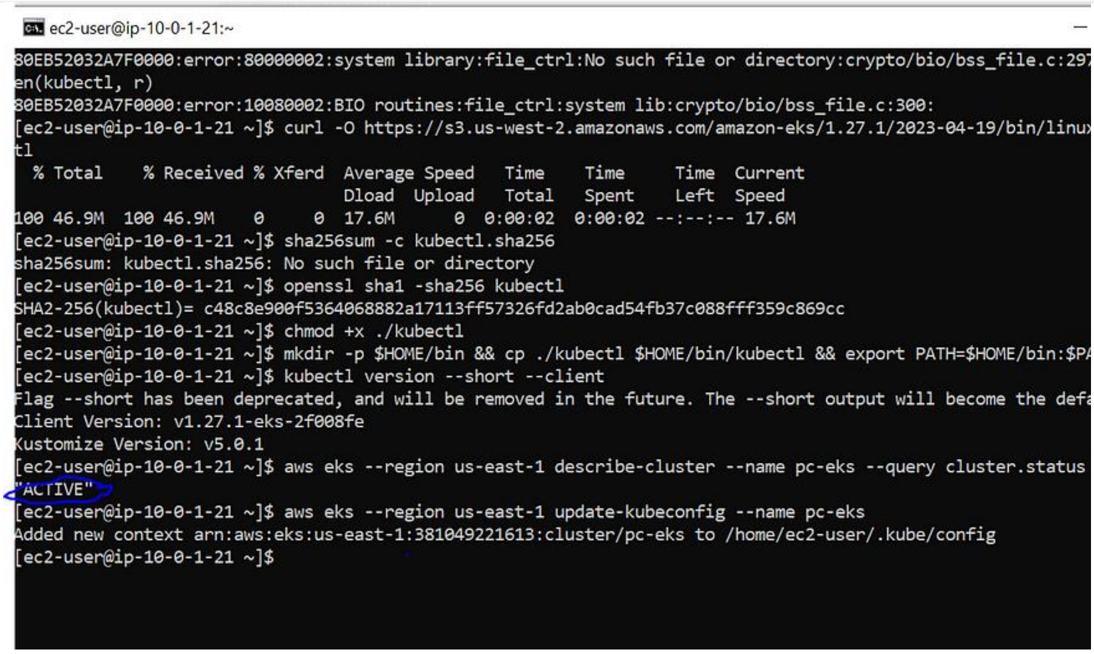


```
[ec2-user@ip-10-0-1-125 ~]$ openssl sha1 -sha256 kubectl
SHA2-256(kubectl)= b48ae5c8c046762113b47fe057d469a30236ec96ef9d29329a26b087c877ae0f
[ec2-user@ip-10-0-1-125 ~]$ chmod +x ./kubectl
[ec2-user@ip-10-0-1-125 ~]$ mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
[ec2-user@ip-10-0-1-125 ~]$ kubectl version --short --client
Client Version: v1.23.17-eks-0a21954
[ec2-user@ip-10-0-1-125 ~]$
```

Let's set up our EKS Cluster on the Kubectl Server. These two commands should do just that

```
aws eks --region us-east-1 describe-cluster --name pc-eks --query cluster.status
```

```
aws eks --region us-east-1 update-kubeconfig --name pc-eks
```



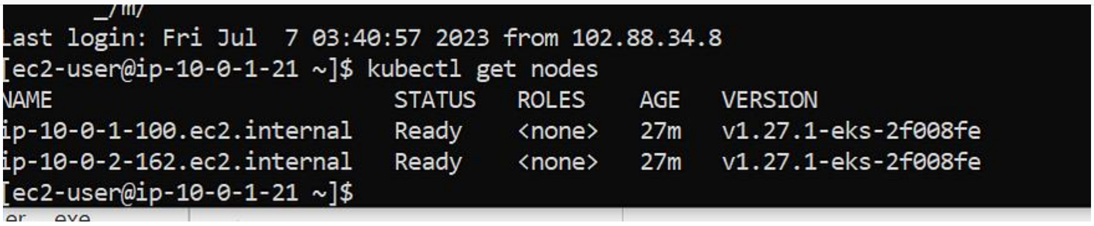
ec2-user@ip-10-0-1-21:~

```
80EB52032A7F0000:error:80000002:system library:file_ctrl:No such file or directory:crypto/bio/bss_file.c:297
en(kubectl, r)
80EB52032A7F0000:error:10080002:BIO routines:file_ctrl:system lib:crypto/bio/bss_file.c:300:
[ec2-user@ip-10-0-1-21 ~]$ curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.1/2023-04-19/bin/linux
tl
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 46.9M  100 46.9M    0     0  17.6M      0  0:00:02  0:00:02 --:--:-- 17.6M
[ec2-user@ip-10-0-1-21 ~]$ sha256sum -c kubectl.sha256
sha256sum: kubectl.sha256: No such file or directory
[ec2-user@ip-10-0-1-21 ~]$ openssl sha1 -sha256 kubectl
SHA2-256(kubectl)= c48c8e900f5364068882a17113ff57326fd2ab0cad54fb37c088fff359c869cc
[ec2-user@ip-10-0-1-21 ~]$ chmod +x ./kubectl
[ec2-user@ip-10-0-1-21 ~]$ mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PA
[ec2-user@ip-10-0-1-21 ~]$ kubectl version --short --client
Flag --short has been deprecated, and will be removed in the future. The --short output will become the defa
Client Version: v1.27.1-eks-2f008fe
Kustomize Version: v5.0.1
[ec2-user@ip-10-0-1-21 ~]$ aws eks --region us-east-1 describe-cluster --name pc-eks --query cluster.status
"ACTIVE"
[ec2-user@ip-10-0-1-21 ~]$ aws eks --region us-east-1 update-kubeconfig --name pc-eks
Added new context arn:aws:eks:us-east-1:381049221613:cluster/pc-eks to /home/ec2-user/.kube/config
[ec2-user@ip-10-0-1-21 ~]$
```

While the first command showed our Kubectl is active the second command updated the Kubectl server by adding my aws:arn to the Kubectl server config

Finally, we will run this last command

```
kubectl get nodes
```



```
Last login: Fri Jul  7 03:40:57 2023 from 102.88.34.8
[ec2-user@ip-10-0-1-21 ~]$ kubectl get nodes
NAME                          STATUS   ROLES    AGE   VERSION
ip-10-0-1-100.ec2.internal    Ready    <none>   27m   v1.27.1-eks-2f008fe
ip-10-0-2-162.ec2.internal    Ready    <none>   27m   v1.27.1-eks-2f008fe
[ec2-user@ip-10-0-1-21 ~]$
```

And here are the two nodes. With this, the second part of the week 4 challenge for the #10weeksofcloudops has been completed

Note: Run terraform destroy to remove all provisioned infrastructures from your AWS account so as not to incur unnecessary bills on your account. Also, remember to delete the S3 bucket and the DynamoDB table