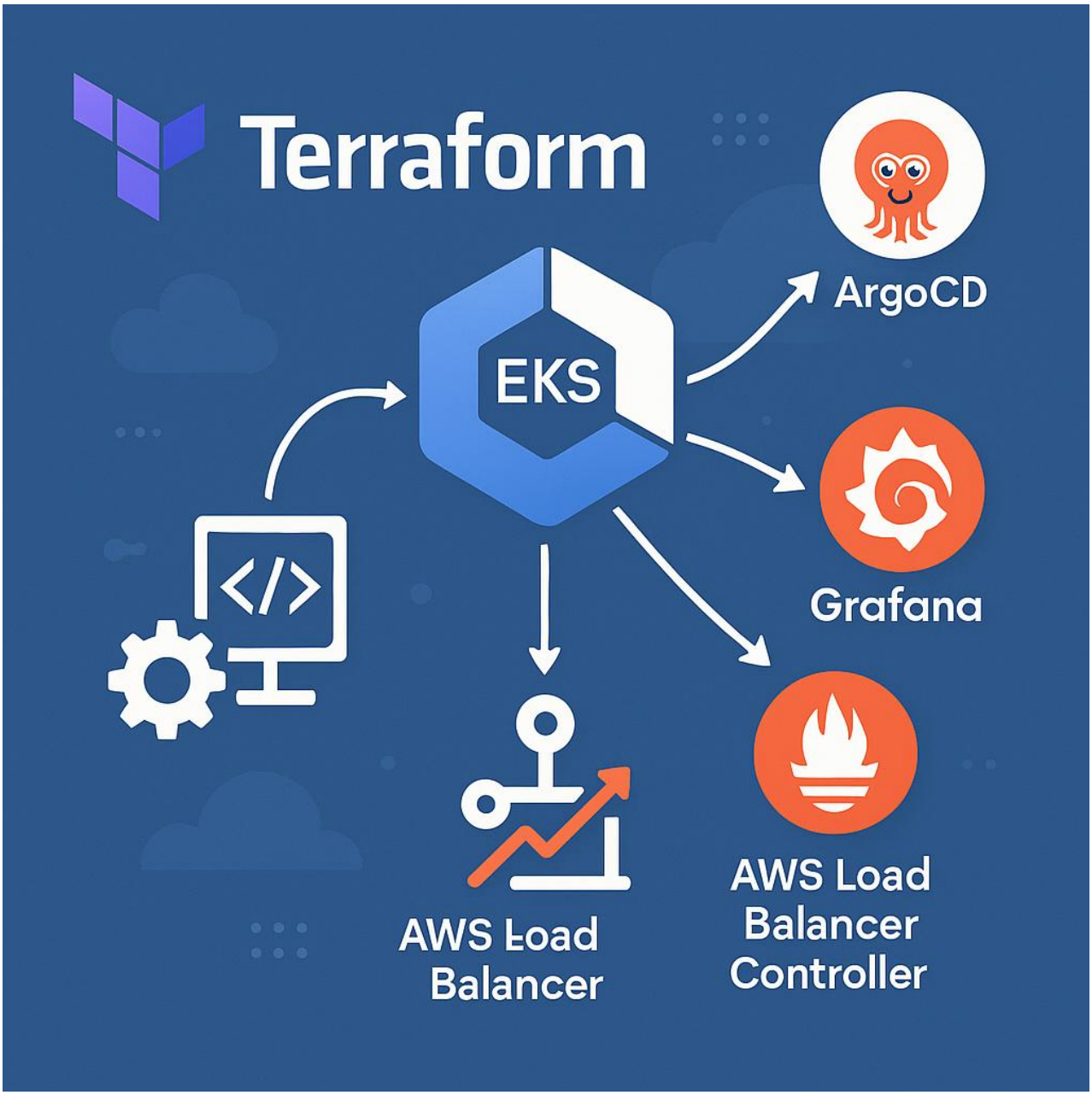
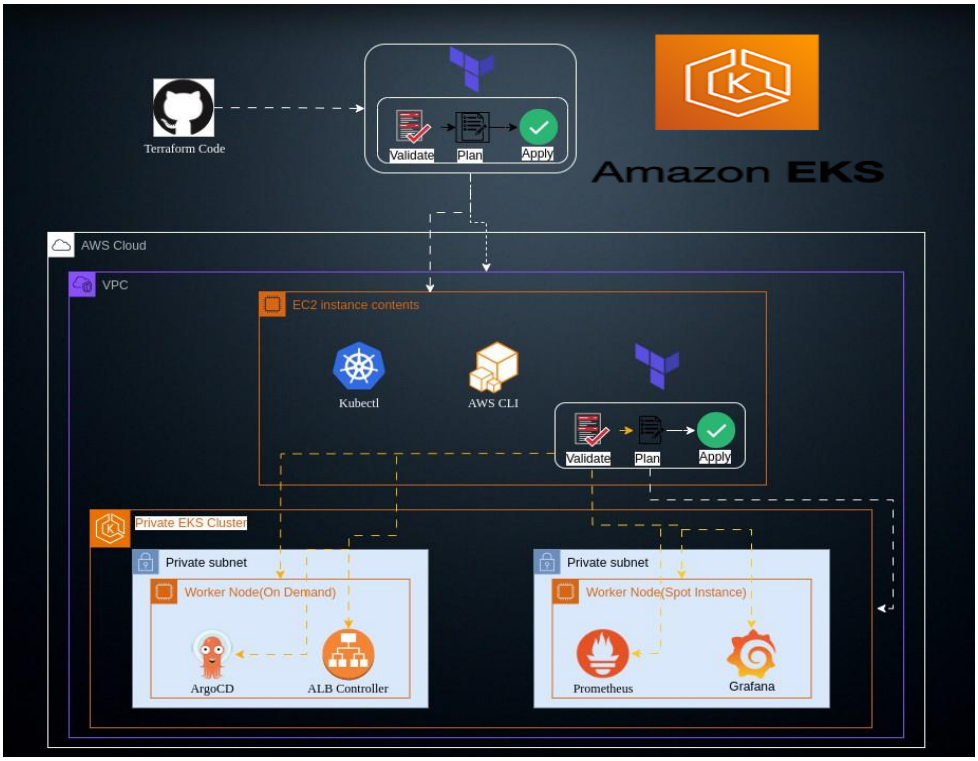


Configure ArgoCD, Promel'heus, Grafana &

AWS Load Balancer Conl'roller on EKS Clusl'er using Terraform



Introduction

In today's DevOps-driven world, automating infrastructure deployment is crucial for maintaining efficiency and scalability. Setting up a secure and robust EKS (Elastic Kubernetes Service) cluster, complete with essential tools like ArgoCD, Prometheus, and Grafana, requires careful planning and execution. This guide will walk you through the entire process, from configuring your environment to deploying your infrastructure using Terraform, ensuring that your private EKS cluster is up and running smoothly with all the necessary resources.

Why Use Terraform for Configuring ArgoCD, Prometheus, and Other Tools?

Configuring tools like ArgoCD, Prometheus, and Grafana through Terraform offers several key advantages:

1. **Infrastructure as Code (IaC)**: Terraform allows you to define your entire infrastructure as code, making it versionable, repeatable, and easier to manage. This approach minimizes the risk of manual configuration errors and ensures consistency across environments.

2. **Automated and Scalable Deployments:** Terraform automates the deployment process, enabling you to scale your infrastructure efficiently. Whether you're deploying in a development, staging, or production environment, Terraform ensures that all resources are provisioned and configured correctly with minimal manual intervention.

3. **Easier Maintenance and Updates:** With Terraform, updating or modifying configurations for tools like ArgoCD and Prometheus is straightforward. You can manage changes centrally, apply updates consistently, and track modifications through version control, reducing the complexity of managing dynamic infrastructures.

4. **Improved Security and Compliance:** By using Terraform, you can enforce security policies and compliance standards across your infrastructure. Terraform configurations can be audited, and changes can be tracked, helping to maintain a secure and compliant environment.

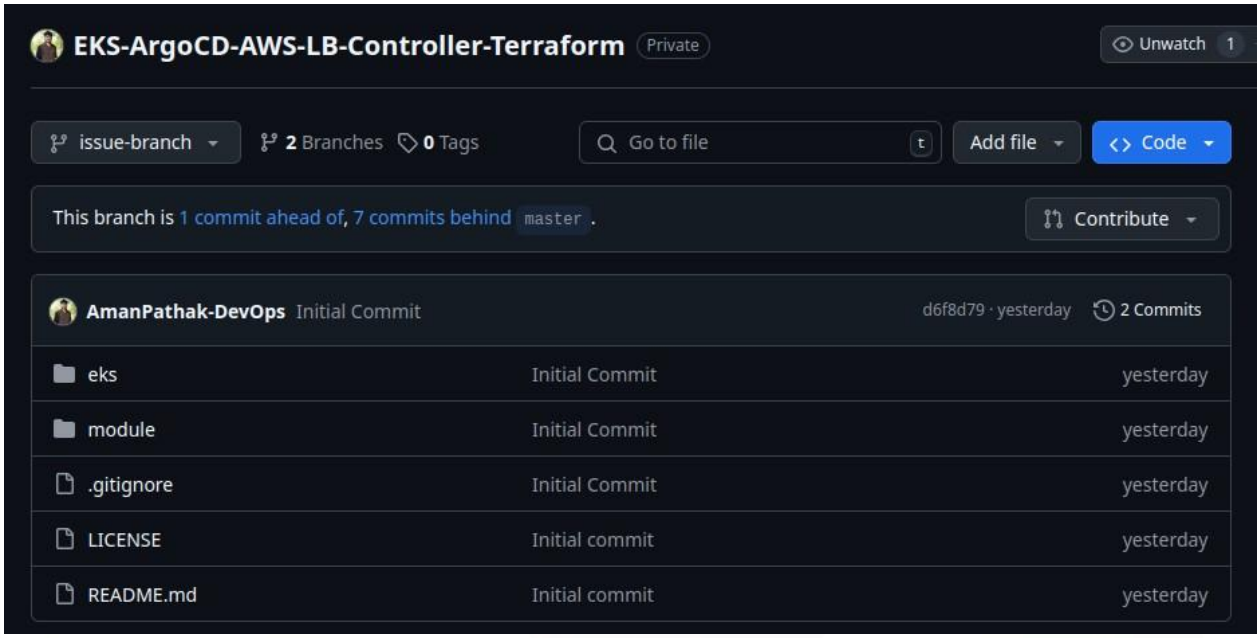
Prerequisites

Before diving into the deployment process, ensure that you have the following prerequisites in place:

1. **AWS Account:** You should have access to an AWS account with sufficient permissions to create and manage resources like VPCs, EC2 instances, and EKS clusters.
2. **Terraform Installed:** Terraform must be installed on your local machine or the server from which you will manage the infrastructure.
3. **Git:** Ensure Git is installed for cloning the necessary repositories.
4. **AWS CLI:** The AWS CLI tool should be installed and configured with appropriate credentials for deploying resources on AWS.
5. **Basic Knowledge of Terraform and Kubernetes:** Familiarity with Terraform and Kubernetes is essential to follow along with the steps and understand the infrastructure you're deploying.

Demonstration

So, I have created two branches Issue-branch and the other one is default named as master. Issue branch is to let you understand, why you can’t deploy argoCD, Prometheus, and any other resources in your EKS Cluster.



This is because your Cluster is Private. You need to be in the same network to enter your cluster and make any changes or configure something.
So, the Issue branch will create everything related to Cluster only such as EKS Cluster, Node groups, EKS Add-ons, etc. But when it tries to deploy argocd and other resources inside your cluster It will throw the error.
You can check the error in the below snippet.

```
module.eks.aws_eks_addon.eks-addons["3"]: Still creating... [20s elapsed]
module.eks.aws_eks_addon.eks-addons["0"]: Still creating... [30s elapsed]
module.eks.aws_eks_addon.eks-addons["3"]: Still creating... [30s elapsed]
module.eks.aws_eks_addon.eks-addons["2"]: Still creating... [30s elapsed]
module.eks.aws_eks_addon.eks-addons["3"]: Still creating... [40s elapsed]
module.eks.aws_eks_addon.eks-addons["0"]: Still creating... [40s elapsed]
module.eks.aws_eks_addon.eks-addons["2"]: Still creating... [40s elapsed]
module.eks.aws_eks_addon.eks-addons["2"]: Creation complete after 48s [id=dev-medium-eks-cluster:kube-proxy]
module.eks.aws_eks_addon.eks-addons["0"]: Still creating... [50s elapsed]
module.eks.aws_eks_addon.eks-addons["3"]: Still creating... [50s elapsed]
module.eks.aws_eks_addon.eks-addons["0"]: Creation complete after 51s [id=dev-medium-eks-cluster:vpc-cni]
module.eks.aws_eks_addon.eks-addons["3"]: Creation complete after 59s [id=dev-medium-eks-cluster:aws-ebs-csi-driver]
data.aws_eks_cluster.eks-cluster: Reading...
data.aws_eks_cluster.eks-cluster: Read complete after 0s [id=dev-medium-eks-cluster]
kubernetes_namespace.argocd: Creating...
kubernetes_service_account.example: Creating...
kubernetes_service_account.example: Still creating... [10s elapsed]
kubernetes_namespace.argocd: Still creating... [10s elapsed]
kubernetes_namespace.argocd: Still creating... [20s elapsed]
kubernetes_service_account.example: Still creating... [20s elapsed]
kubernetes_service_account.example: Still creating... [30s elapsed]
kubernetes_namespace.argocd: Still creating... [30s elapsed]

Error: Post "https://16D6FBF9A7E7FC3417BDE71A310D655C.gr7.us-east-1.eks.amazonaws.com/api/v1/namespaces": dial tcp [64:ff9b::a10:8dfd]:443: i/o timeout

  with kubernetes_namespace.argocd,
  on argocd.tf line 1, in resource "kubernetes_namespace" "argocd":
   1: resource "kubernetes_namespace" "argocd" {


Error: Post "https://16D6FBF9A7E7FC3417BDE71A310D655C.gr7.us-east-1.eks.amazonaws.com/api/v1/namespaces/default/serviceaccounts": context deadline exceeded

  with kubernetes_service_account.example,
  on kubernetes.tf line 70, in resource "kubernetes_service_account" "example":
  70: resource "kubernetes_service_account" "example" {


Releasing state lock. This may take a few moments...
aman-pathak@aman:~/Projects/EKS-ArgoCD-AWS-LB-Controller-Terraform/eks$ git branch
* issue-branch
  master
```

P.S.: The EKS Cluster was configured as it was not visible in above snippet.

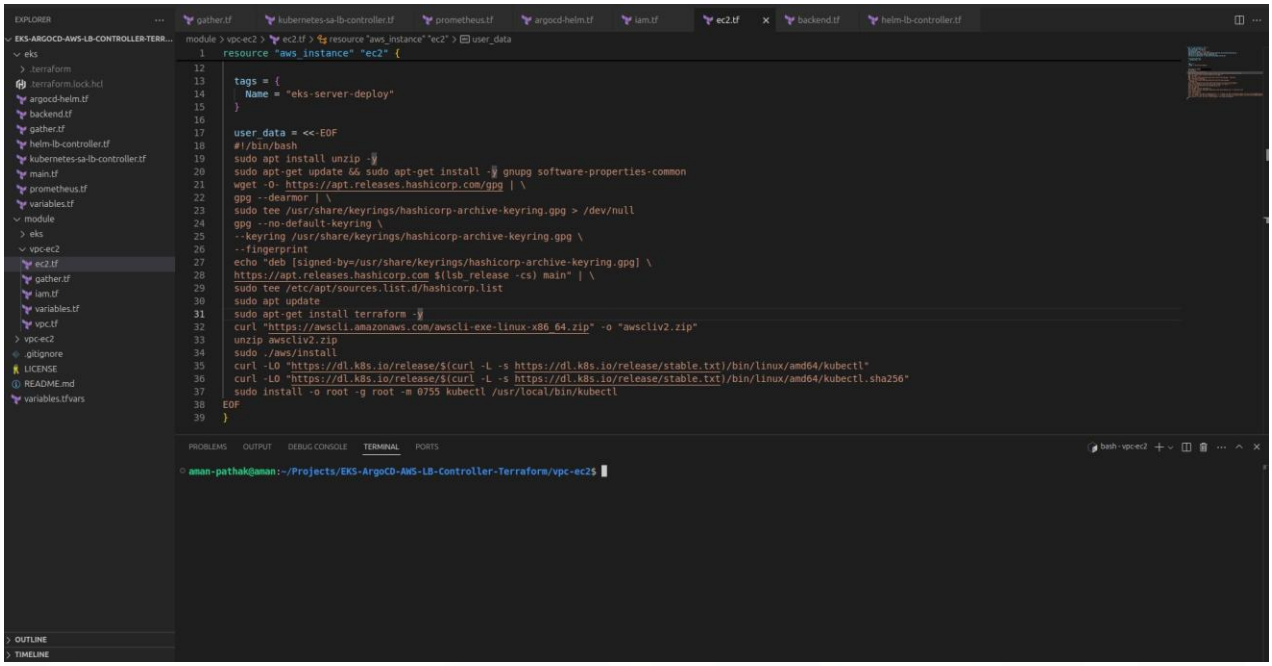
```
module.eks.aws_eks_node_group.ondemand-node: Still destroying... [id=dev-medium-eks-cluster:dev-medium-eks-cluster-on-demand-nodes, 2m30s elapsed]
module.eks.aws_eks_node_group.spot-node: Destruction complete after 2m31s
module.eks.aws_eks_node_group.ondemand-node: Still destroying... [id=dev-medium-eks-cluster:dev-medium-eks-cluster-on-demand-nodes, 2m40s elapsed]
module.eks.aws_eks_node_group.ondemand-node: Destruction complete after 2m42s
module.eks.aws_iam_role.eks-nodegroup-role[0]: Destroying... [id=dev-medium-eks-cluster-nodegroup-role-2377]
module.eks.aws_eks_cluster.eks[0]: Destroying... [id=dev-medium-eks-cluster]
module.eks.aws_iam_role.eks-nodegroup-role[0]: Destruction complete after 3s
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 10s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 20s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 30s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 40s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 50s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m0s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m10s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m20s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m30s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m40s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m50s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m0s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m10s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m20s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m30s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m40s elapsed]
module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m50s elapsed]
module.eks.aws_eks_cluster.eks[0]: Destruction complete after 2m51s
module.eks.aws_iam_role.eks-cluster-role[0]: Destroying... [id=dev-medium-eks-cluster-role-2377]
module.eks.aws_subnet.private-subnet[0]: Destroying... [id=subnet-00204cfe95bcd3cd1]
module.eks.aws_subnet.private-subnet[2]: Destroying... [id=subnet-0fde9b85844aa5a3a]
module.eks.aws_subnet.private-subnet[1]: Destroying... [id=subnet-07b6d658ba103d26f]
module.eks.aws_security_group.eks-cluster-sg: Destroying... [id=sg-0c8e072f99d14c4a3]
module.eks.aws_iam_role.eks-cluster-role[0]: Destruction complete after 2s
module.eks.random_integer.random_suffix: Destroying... [id=2377]
module.eks.random_integer.random_suffix: Destruction complete after 0s
module.eks.aws_subnet.private-subnet[0]: Destruction complete after 3s
module.eks.aws_subnet.private-subnet[2]: Destruction complete after 3s
module.eks.aws_subnet.private-subnet[1]: Destruction complete after 4s
module.eks.aws_security_group.eks-cluster-sg: Destruction complete after 4s
module.eks.aws_vpc.vpc: Destroying... [id=vpc-0230937f80bc75d5e]
module.eks.aws_vpc.vpc: Destruction complete after 2s
Releasing state lock. This may take a few moments...

Destroy complete! Resources: 40 destroyed.
```

How to solve this?

To deploy anything inside your EKS Cluster, we need an instance or server to deploy those configurations that have the same VPC.
So, we are going to deploy our VPC and one EC2 server first. Then from the ec2 server, we will deploy everything related to the EKS Cluster.

In the master branch of the repository, you will find two modules. The first module is vpc-ec2 which we need to apply first.

So, clone the repository with the master branch and navigate to the vpc-ec2 directory.



Run the below commands to deploy VPC(Other networking resources) & EC2 Server on AWS.

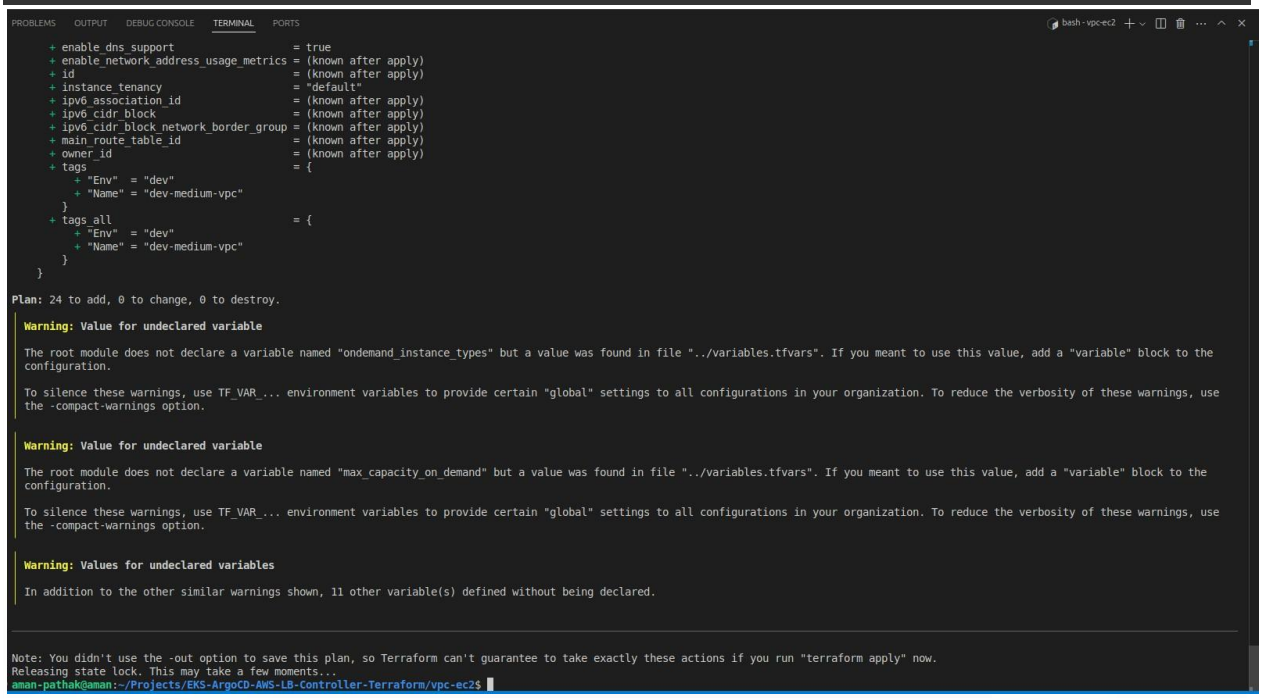
```
> terraform init
```

```
aman-pathak@aman:~/Projects/EKS-ArgoCD-AWS-LB-Controller-Terraform/vpc-ec2$ terraform init
Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing modules...
- vpc-ec2 in ../module/vpc-ec2
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.49.0"...
- Finding hashicorp/kubernetes versions matching "2.31.0"...
- Installing hashicorp/kubernetes v2.31.0...
- Installed hashicorp/kubernetes v2.31.0 (signed by HashiCorp)
- Installing hashicorp/aws v5.49.0...
- Installed hashicorp/aws v5.49.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
aman-pathak@aman:~/Projects/EKS-ArgoCD-AWS-LB-Controller-Terraform/vpc-ec2$
```

```
> terraform validate
```

```
aman-pathak@aman:~/Projects/EKS-ArgoCD-AWS-LB-Controller-Terraform/vpc-ec2$ terraform validate
Success! The configuration is valid.

aman-pathak@aman:~/Projects/EKS-ArgoCD-AWS-LB-Controller-Terraform/vpc-ec2$
```

> terraform plan -var-file=../variables.tfvars



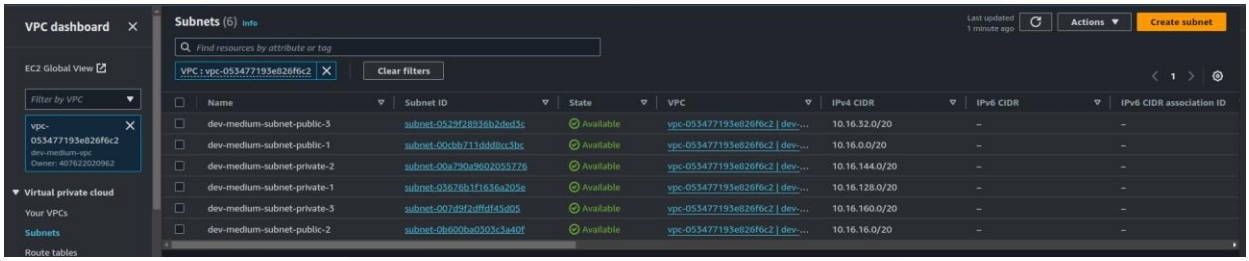
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    bash - vpc-ec2

      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Env"  = "dev"
          + "Name" = "dev-medium-vpc"
        }
      + tags_all                             = {
          + "Env"  = "dev"
          + "Name" = "dev-medium-vpc"
        }
    }

Plan: 24 to add, 0 to change, 0 to destroy.

  Warning: Value for undeclared variable

  The root module does not declare a variable named "ondemand_instance_types" but a value was found in file "../variables.tfvars". If you meant to use this value, add a "variable" block to the
  configuration.

  To silence these warnings, use TF_VAR_... environment variables to provide certain "global" settings to all configurations in your organization. To reduce the verbosity of these warnings, use
  the -compact-warnings option.

  Warning: Value for undeclared variable

  The root module does not declare a variable named "max_capacity_on_demand" but a value was found in file "../variables.tfvars". If you meant to use this value, add a "variable" block to the
  configuration.

  To silence these warnings, use TF_VAR_... environment variables to provide certain "global" settings to all configurations in your organization. To reduce the verbosity of these warnings, use
  the -compact-warnings option.

  Warning: Values for undeclared variables

  In addition to the other similar warnings shown, 11 other variable(s) defined without being declared.


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
Releasing state lock. This may take a few moments...
aman-pathak@aman:~/Projects/EKS-ArgoCD-AWS-LB-Controller-Terraform/vpc-ec2$
```

> terraform apply -auto-approve -var-file=../variables.tfvars

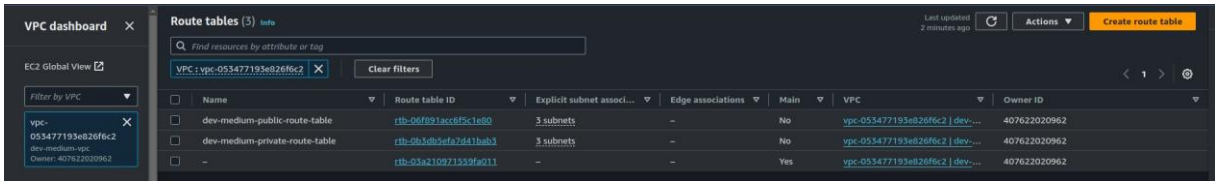
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    bash - vpc-ec2

module.vpc-ec2.aws_instance.ec2: Still creating... [10s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [20s elapsed]
module.vpc-ec2.aws_instance.ec2: Still creating... [20s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [30s elapsed]
module.vpc-ec2.aws_instance.ec2: Still creating... [30s elapsed]
module.vpc-ec2.aws_instance.ec2: Creation complete after 37s [id=i-0d8a3bc509812b65b]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [40s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [50s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [1m0s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [1m10s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [1m20s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [1m30s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Still creating... [1m40s elapsed]
module.vpc-ec2.aws_nat_gateway.ngw: Creation complete after 1m50s [id=nat-0e920b64e401eb765]
module.vpc-ec2.aws_route_table.private-rt: Creating...
module.vpc-ec2.aws_route_table.private-rt: Creation complete after 3s [id=rtb-0b3db5efa7d41bab3]
module.vpc-ec2.aws_route_table_association.private-rt-association[1]: Creating...
module.vpc-ec2.aws_route_table_association.private-rt-association[0]: Creating...
module.vpc-ec2.aws_route_table_association.private-rt-association[2]: Creating...
module.vpc-ec2.aws_route_table_association.private-rt-association[0]: Creation complete after 2s [id=rtbassoc-0a46e70188fbf044c]
module.vpc-ec2.aws_route_table_association.private-rt-association[2]: Creation complete after 2s [id=rtbassoc-0de5af7a0ddd86803]
module.vpc-ec2.aws_route_table_association.private-rt-association[1]: Creation complete after 2s [id=rtbassoc-00599d70a94f9b8fc]

  Warning: Value for undeclared variable

  The root module does not declare a variable named "max_capacity_on_demand" but a value was found in file "../variables.tfvars". If you meant to use this value, add a "variable" block to the
  configuration.

  To silence these warnings, use TF_VAR_... environment variables to provide certain "global" settings to all configurations in your organization. To reduce the verbosity of these warnings, use
  the -compact-warnings option.

  Warning: Value for undeclared variable

  The root module does not declare a variable named "min_capacity_spot" but a value was found in file "../variables.tfvars". If you meant to use this value, add a "variable" block to the
  configuration.

  To silence these warnings, use TF_VAR_... environment variables to provide certain "global" settings to all configurations in your organization. To reduce the verbosity of these warnings, use
  the -compact-warnings option.

  Warning: Values for undeclared variables

  In addition to the other similar warnings shown, 11 other variable(s) defined without being declared.

Releasing state lock. This may take a few moments...

Apply complete! Resources: 24 added, 0 changed, 0 destroyed.
aman-pathak@aman:~/Projects/EKS-ArgoCD-AWS-LB-Controller-Terraform/vpc-ec2$
```

VPC & other services created through the above commands
Here are the snippets
VPC

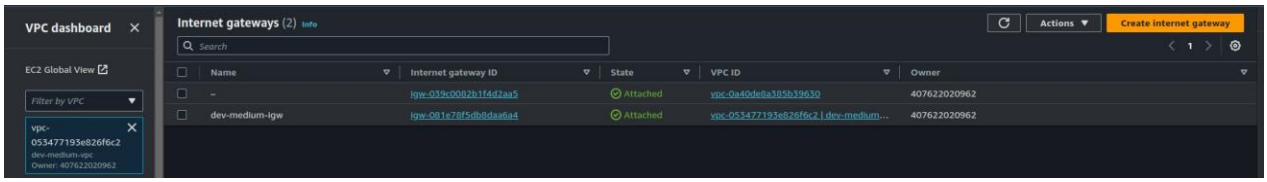
Subnets



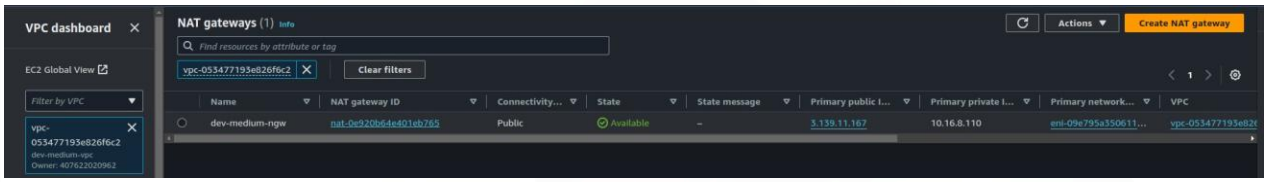
Route tables



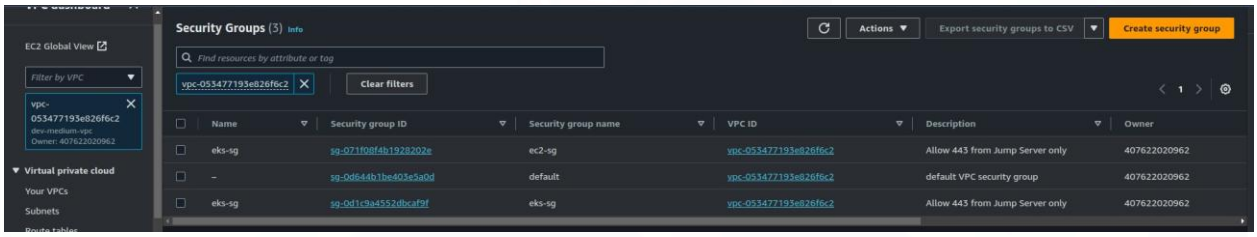
Internet Gateway



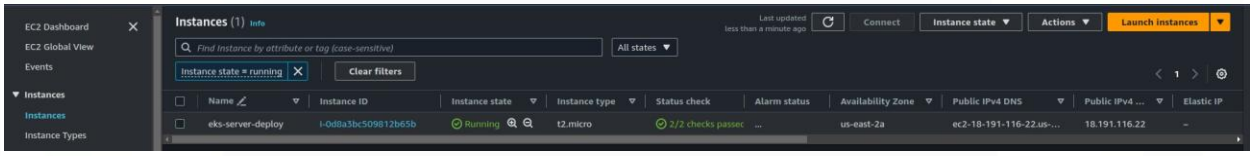
NAT Gateway



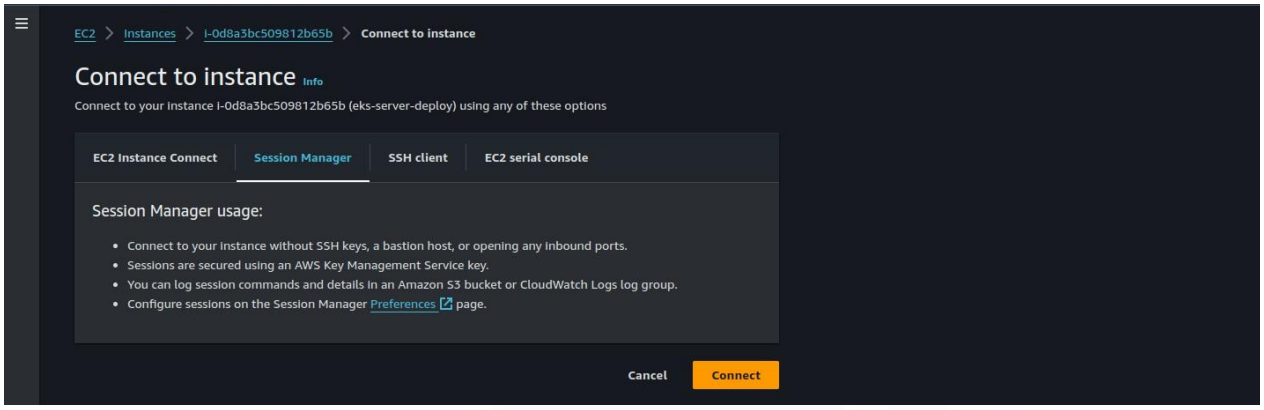
Security Group



EC2-Instance



So, everything is created according to our requirements.
Now log in to the Server by selecting the created ec2 instance and click on **Connect**
You can log in without the Pem file as we did not follow that.
If Session Manager is struggling to connect to the instance, you can use EC2 Instance Connect to login to the server.



I am logged In to my server and switched to the Ubuntu user with the help of the below command

```
> sudo su ubuntu
```

We need to install a few tools as our pre-requisites. We installed it through user data in the Terraform script
Run the below command to validate whether it's installed or not. If not, wait for 5-10 minutes,

```
> terraform version
aws --version
kubectl version
```

Once all the tools are installed then, we need to configure AWS CLI as we need to deploy our infrastructure over AWS Cloud.
Run the below command to configure CLI and make sure you have sufficient permission with your credentials. For demonstration, you can utilize Administration Access user keys
P.S.: Don't use the same access keys as you will have your keys in your AWS Account

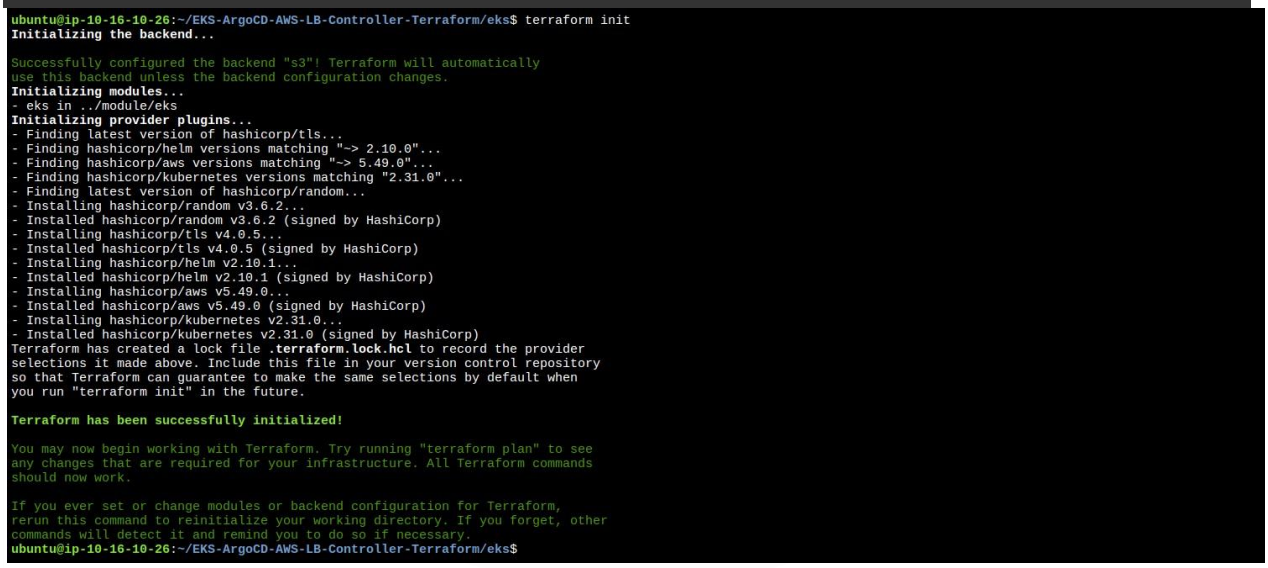
```
> aws configure
```

Now, clone the same repository where our Terraform code is present for EKS

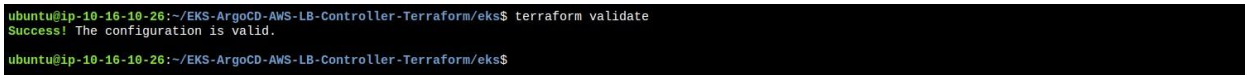
```
> git clone
https://github.com/AmanPathak-DevOps/EKS-ArgoCD-AWS-LB-Controlle
r-Terraform.git
```

Now, we are ready to deploy our EKS Cluster, and other tools through Terraform. Navigate to the eks directory and run the below commands to deploy it

```
> terraform init
```

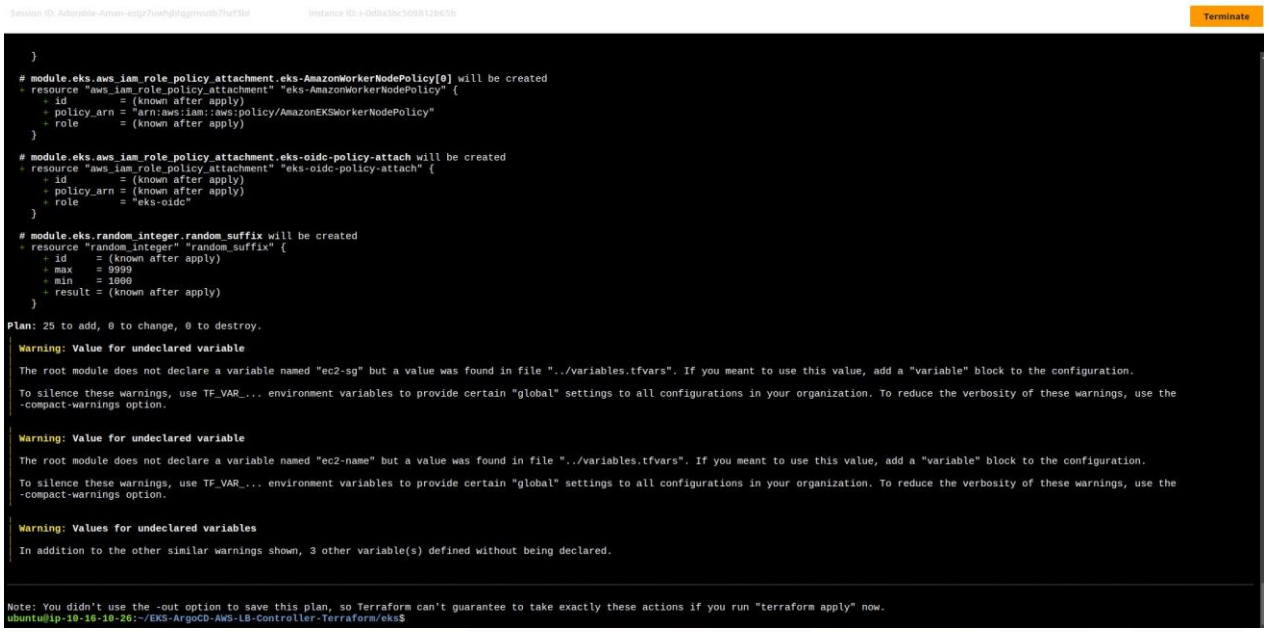


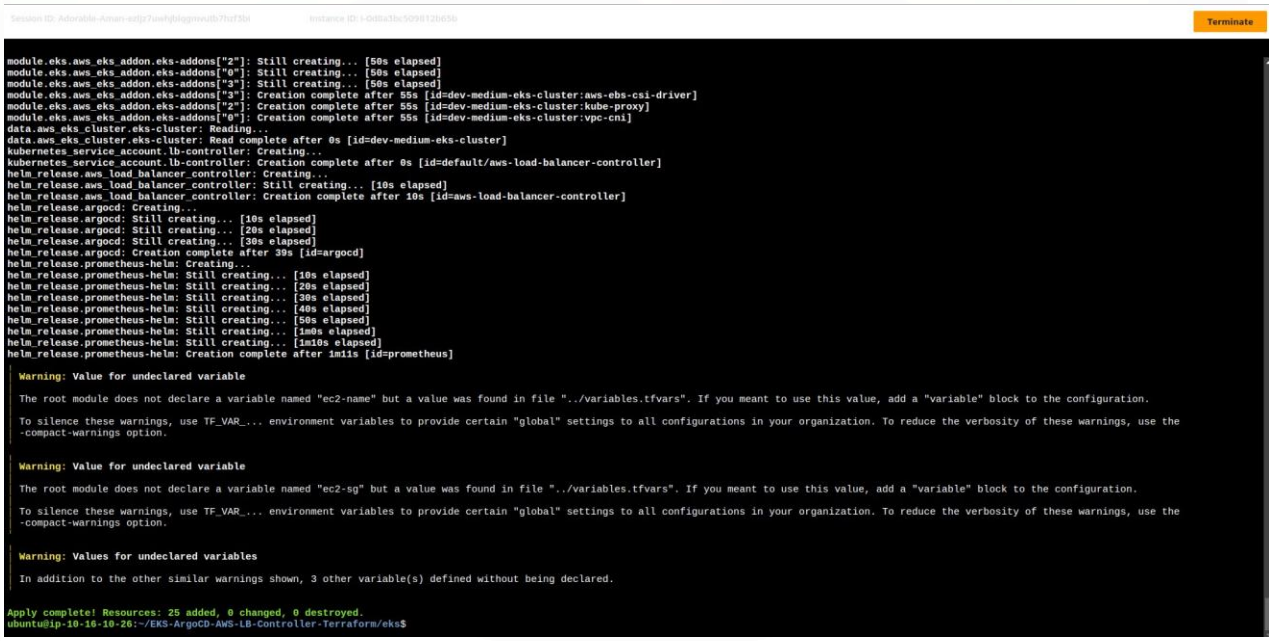

```
> terraform validate
```



```
> terraform apply -auto-approve -var-file=../variables.tfvars
```

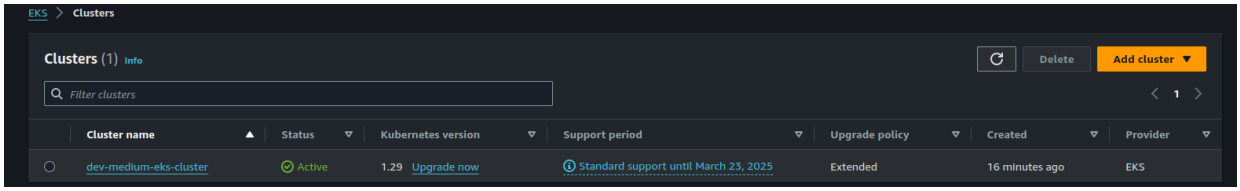
```
> terraform plan -var-file=../variables.tfvars
```



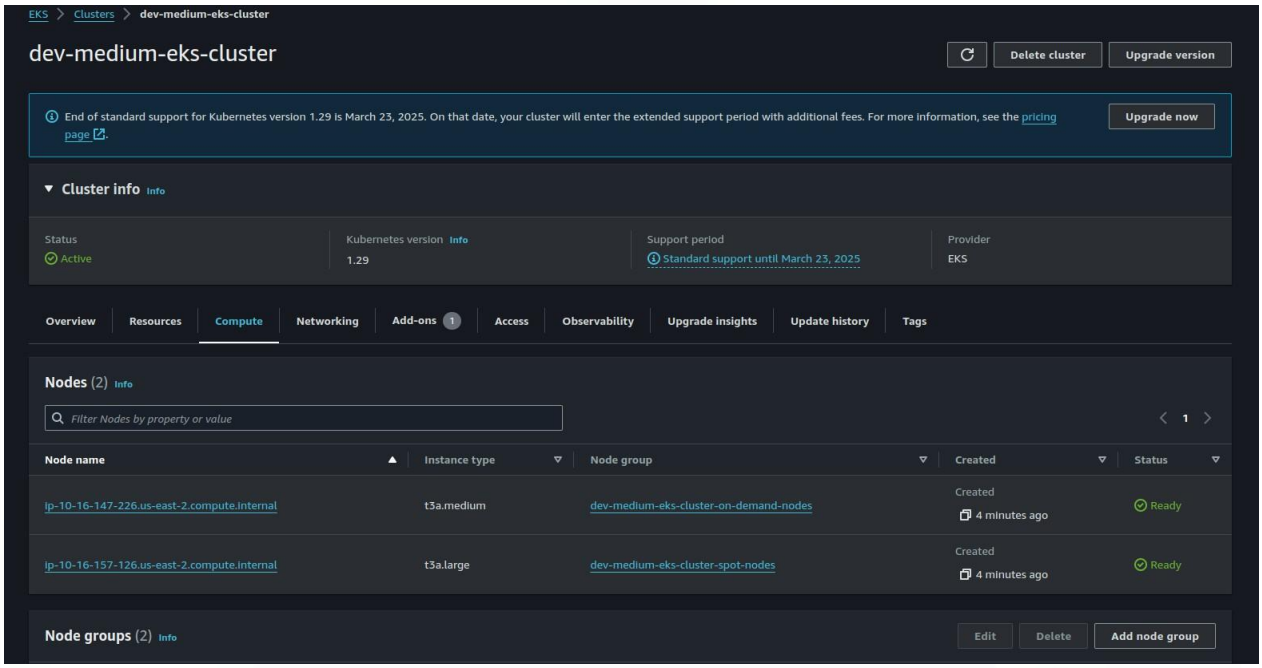


EKS Cluster & other services created through the above commands
Here are the snippets

EKS Cluster

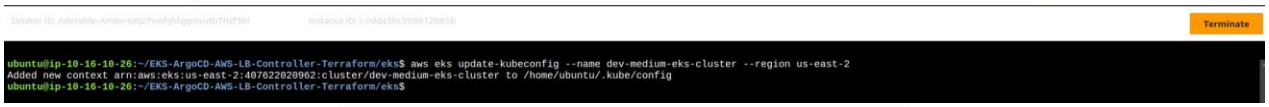


Node Groups



To validate whether the other Kubernetes resources have been created or not.
We need to update the kubconfig on our same ec2 server
Run the below command to update the kubeconfig

```
> aws eks update-kubeconfig --name dev-medium-eks-cluster
--region us-east-2
```

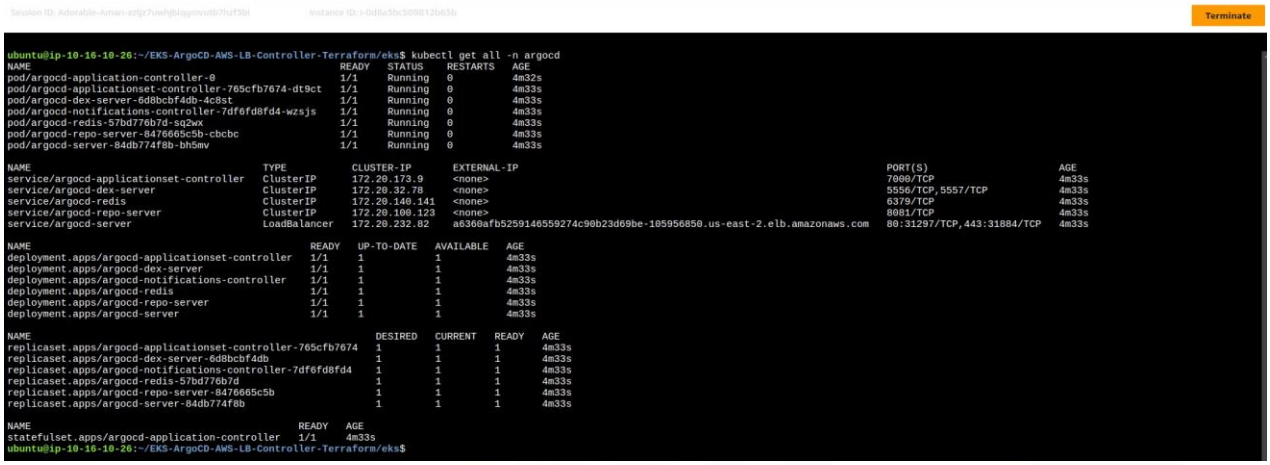


Run the below command to validate whether it's our cluster or not

```
> kubectl get nodes
```

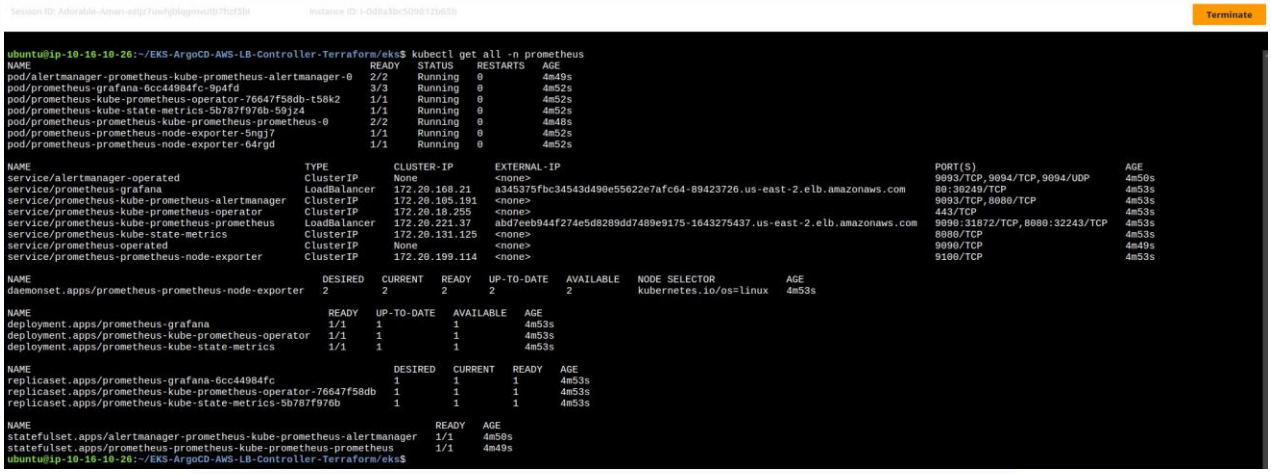
Check whether the resources are running in argocd namespace or not

```
> kubectl get all -n argocd
```



Check whether the resources are running in Prometheus namespace or not

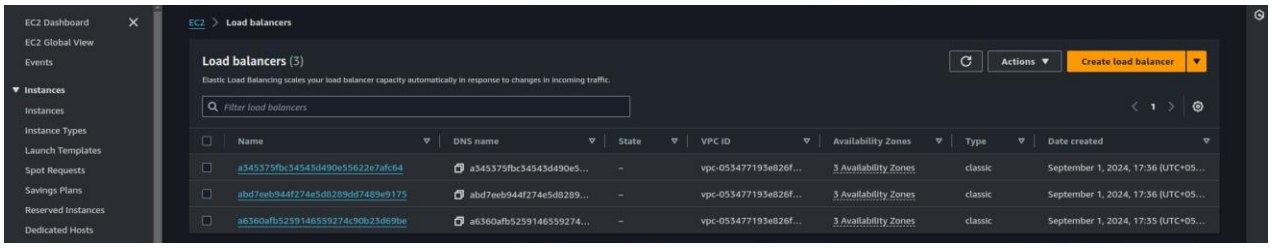
```
> kubectl get all -n prometheus
```



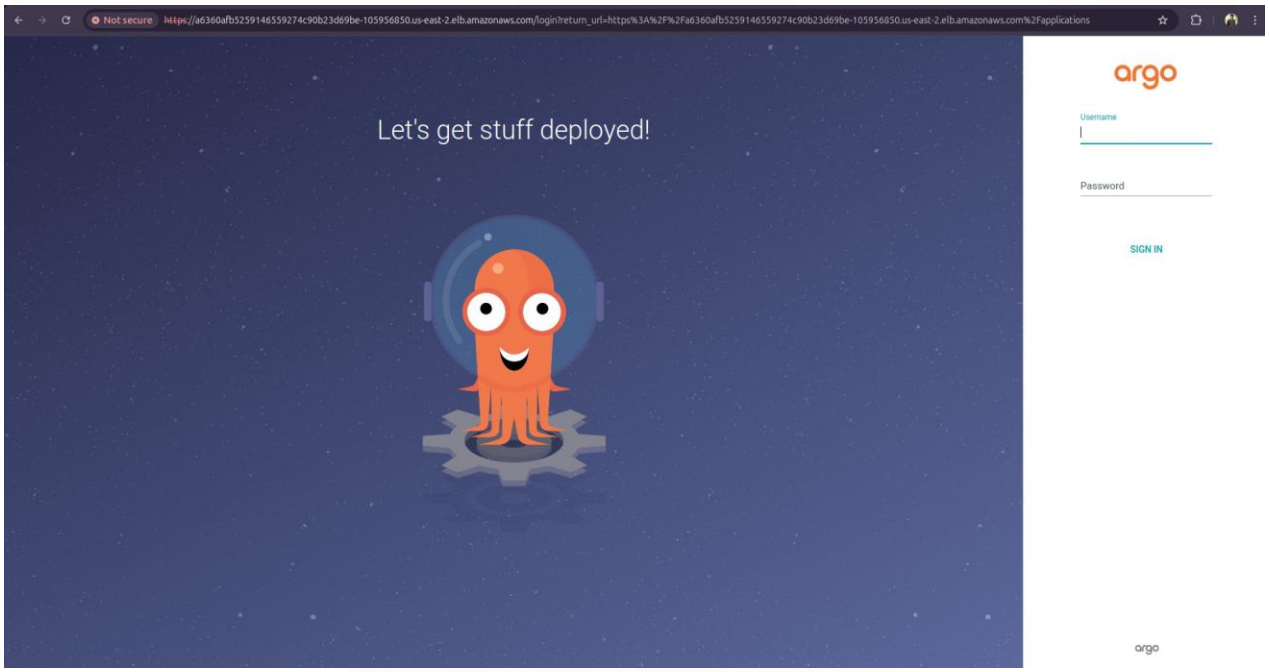
Check whether the AWS Load Balancer controller pods are running in a aws-loadbalancer-controller namespace or not

```
> kubectl get all -n aws-loadbalancer-controller
```

So, we have configured the AWS Load Balancer Controller, ArgoCD, Prometheus, and Grafana through Terraform.
According to the terraform script, we have updated the service type for ArgoCD, Prometheus, and Grafana from ClusterIP to LoadBalacer.
Hence, navigate to Load Balancer on your AWS account.
You can check which LB is for which tool through the kubectl command that we ran in the above steps.



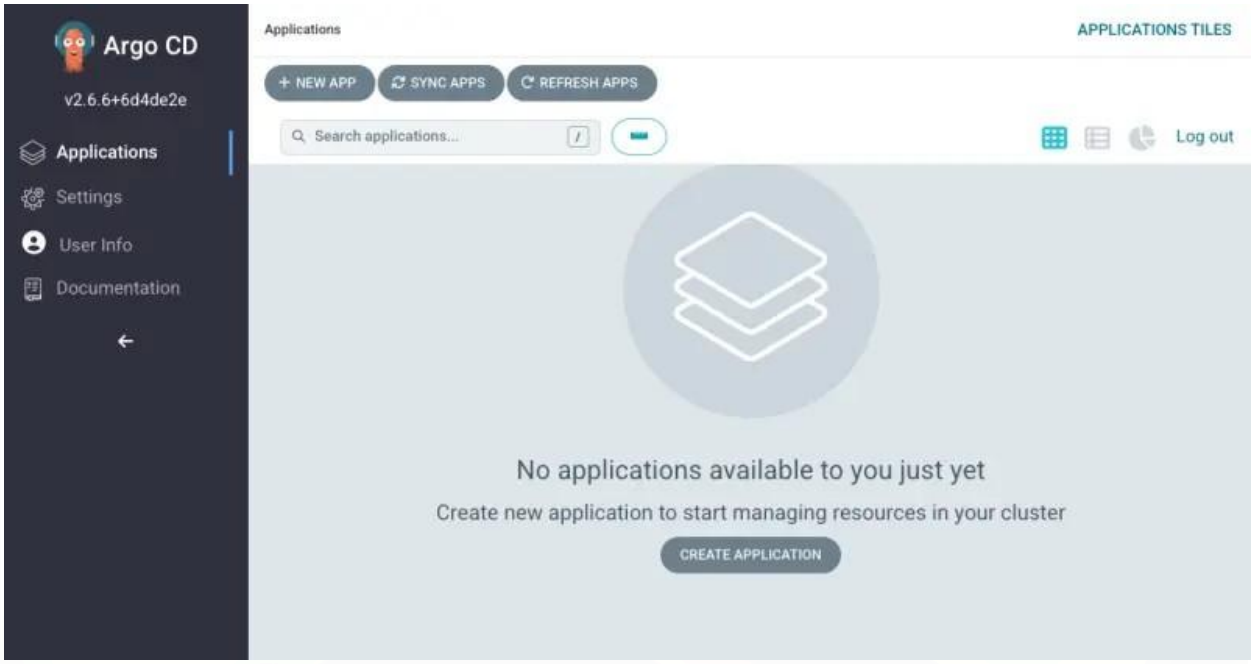
So, let's try to access argoCD first. Copy the DNS and paste it into your favorite browser



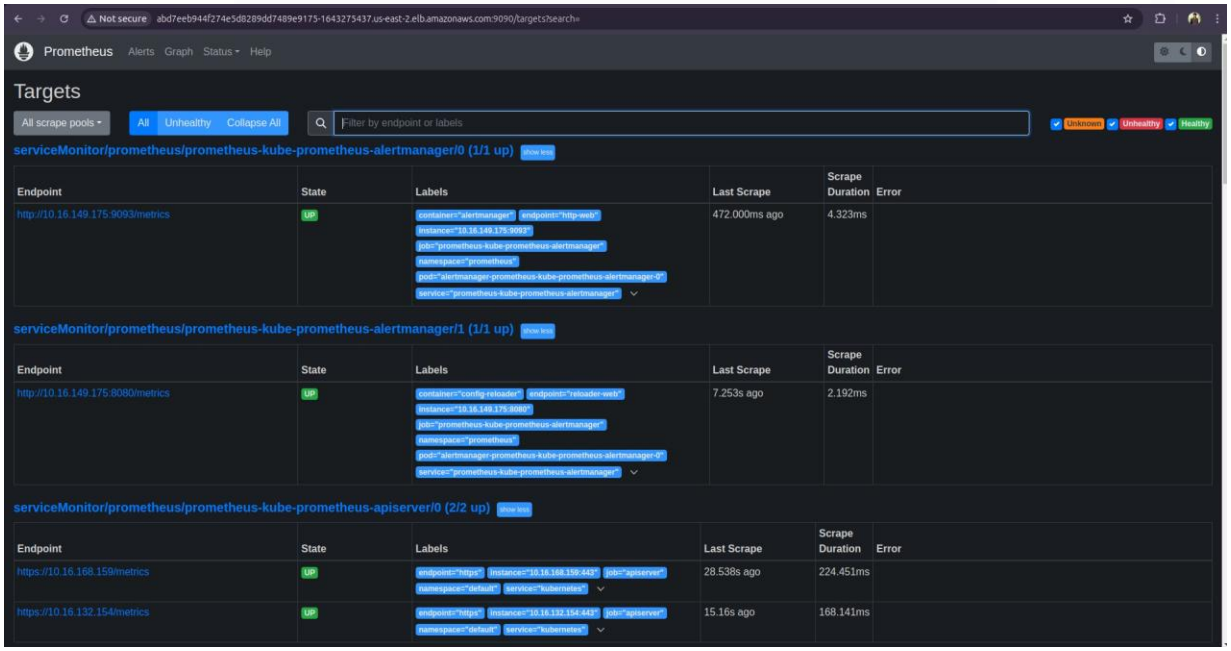
The username is admin but it needs the password which we don't know. So, we need to run a command on our ec2 server to get the password for the login

```
kubectl -n argocd get secret argocd-initial-admin-secret -o
jsonpath="{.data.password}" | base64 -d
```

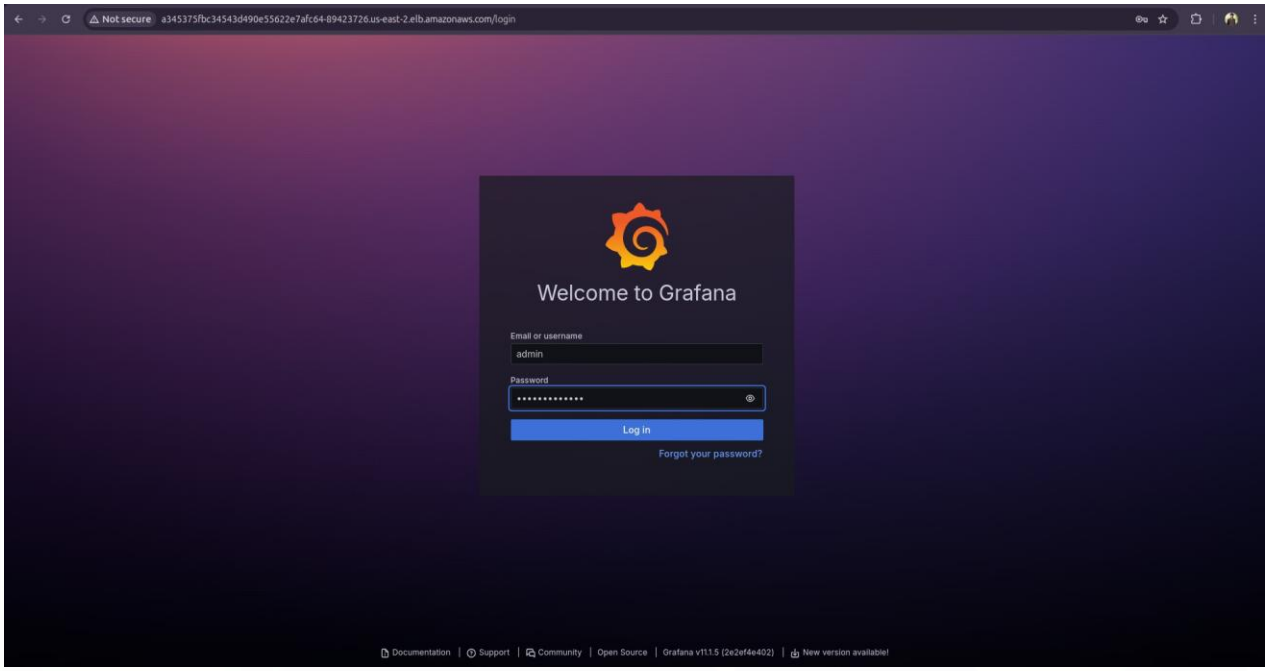
It is accessible



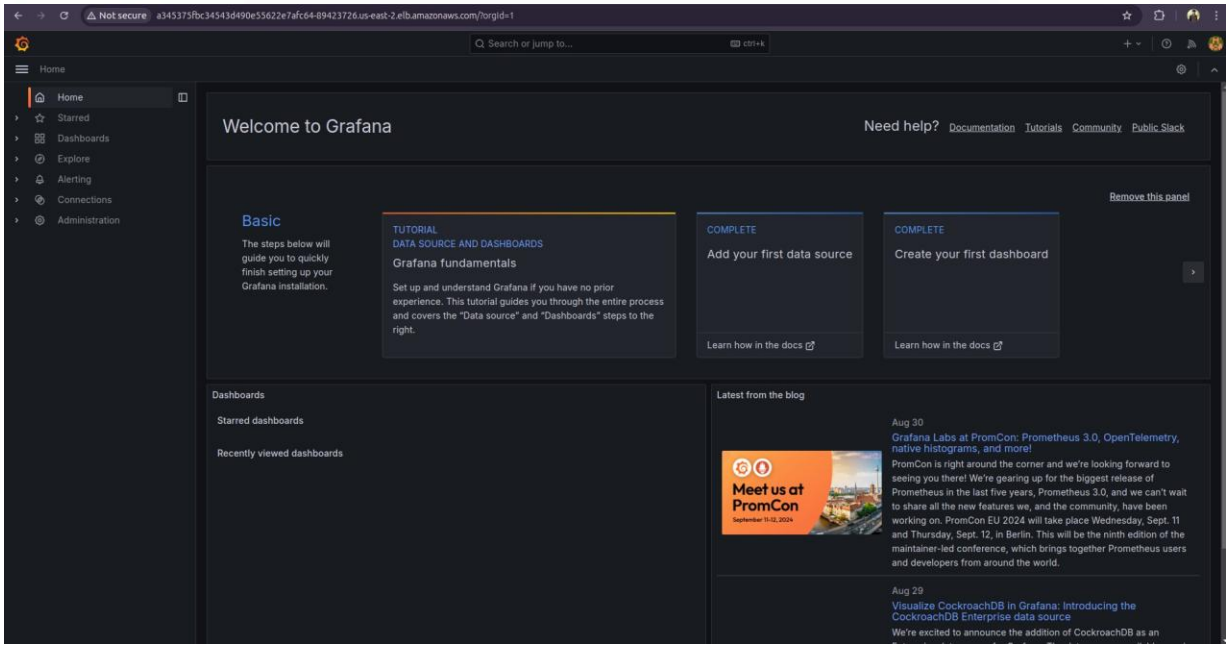
Now, let’s try to access Prometheus. Copy the DNS and paste it into your favorite browser with port 9090.
It is accessible



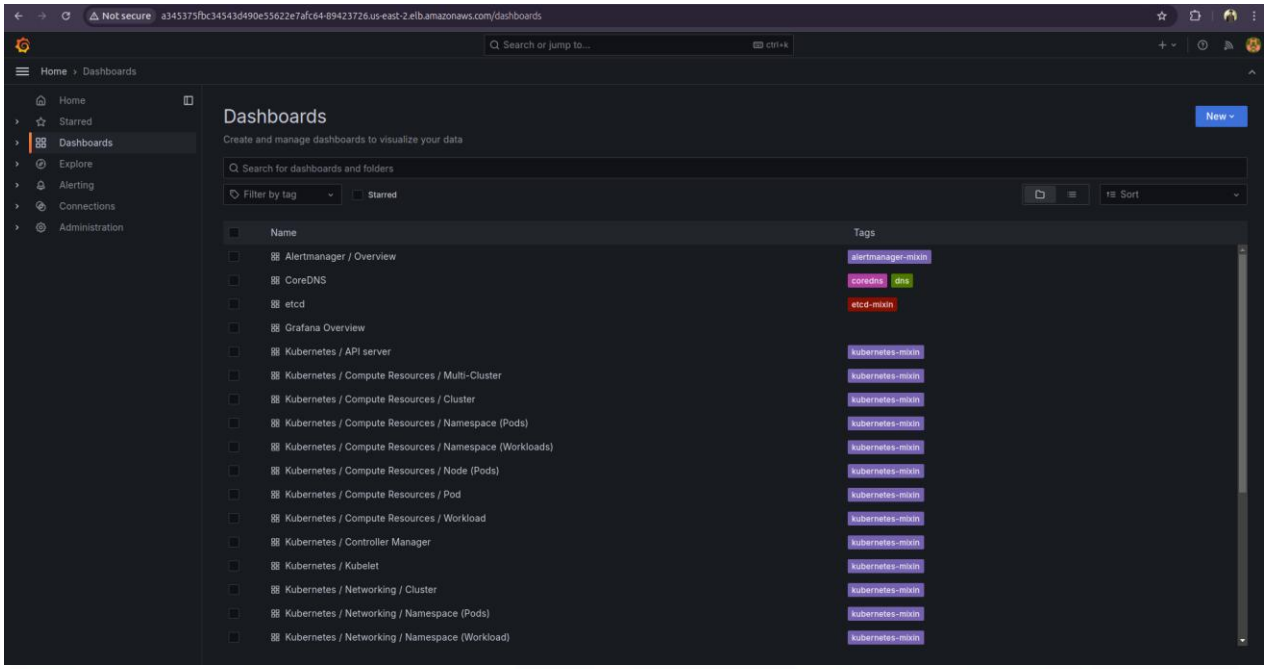
In the end, let’s try to access our Grafana dashboard. Copy the DNS and paste it into your favorite browser
It is accessible.
The username is admin and the password is prom-operator for the Grafana login



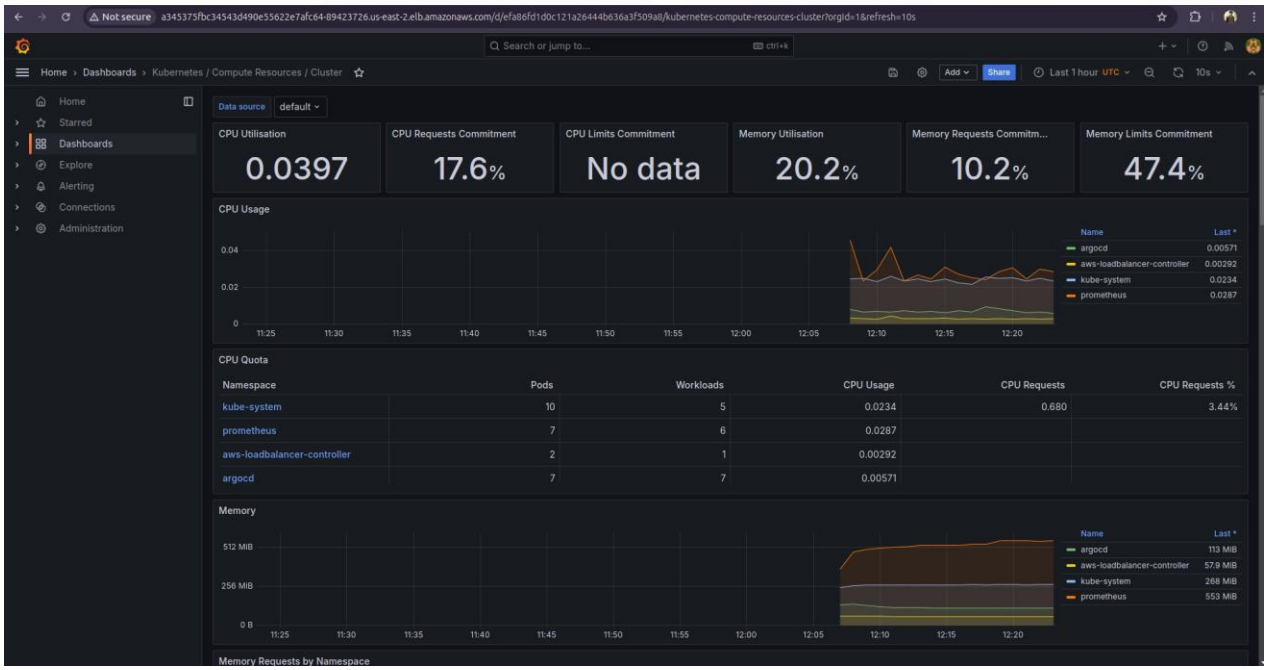
After login, the Grafana dashboard



There are a lot of dashboards already imported. You can click any of them to get insight about the EKS Cluster resources accordingly.



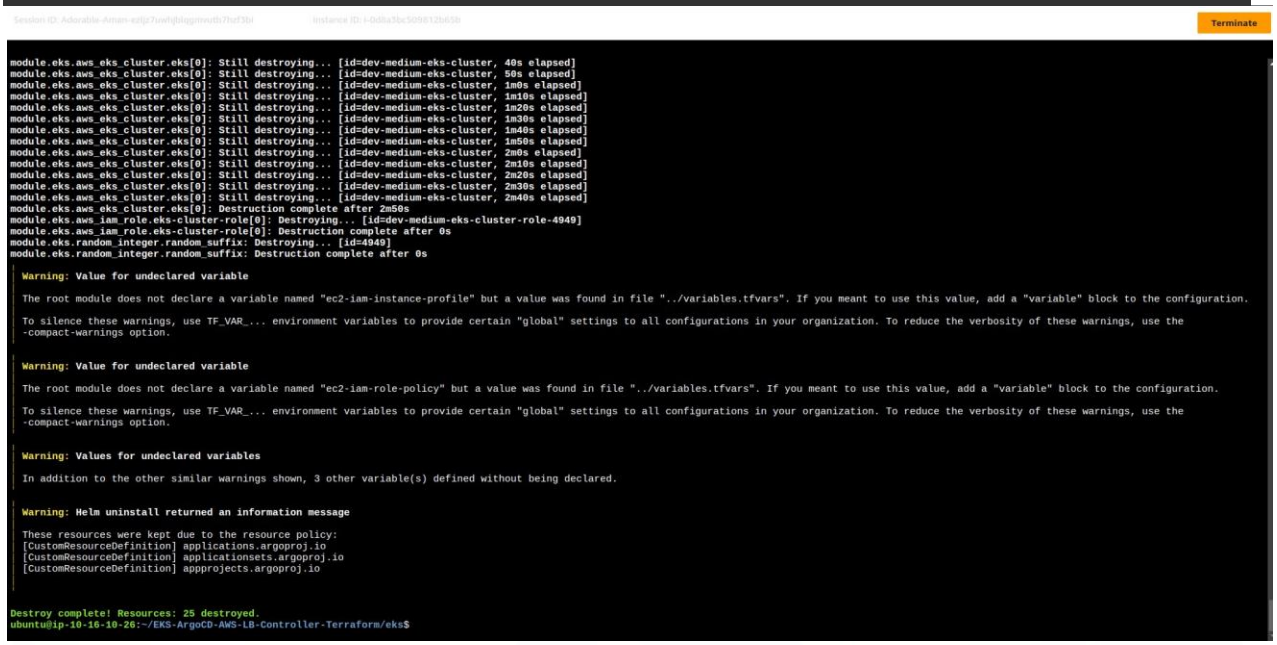
Here is one of them



So, we have completed our demonstration for today.
Hope you learn something new today.
To prevent the cost of cloud. Don’t forget to destroy all the resources.

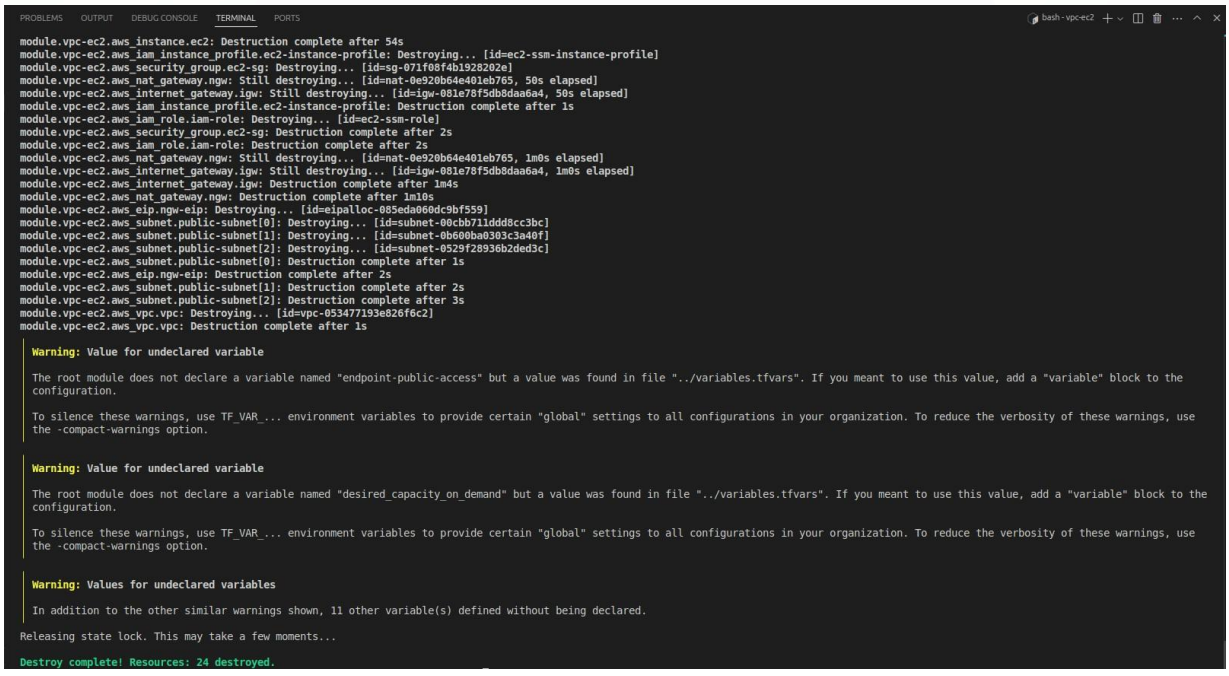
To do that, we need to run EKS Cluster first.

```
> terraform destroy -auto-approve -var-file=../variables.tfvars
```



Now, destroy the vpc and an ec2 server. To do it, run the below command on your local server

```
> terraform destroy -auto-approve -var-file=../variables.tfvars
```



I would recommend you to read the eks terraform files to get a better understanding of the resources.

Conclusion

In this comprehensive guide, we've explored how to deploy a private EKS cluster on AWS and configure essential Kubernetes tools such as ArgoCD, Prometheus, and Grafana using Terraform. By following these steps, you can efficiently manage your infrastructure and ensure that your applications are running smoothly in a secure, scalable environment. Remember to clean up your resources after the demonstration to avoid unnecessary costs. Continuous learning and hands-on practice are key to mastering these DevOps practices, so keep experimenting with different configurations and tools to enhance your skills.