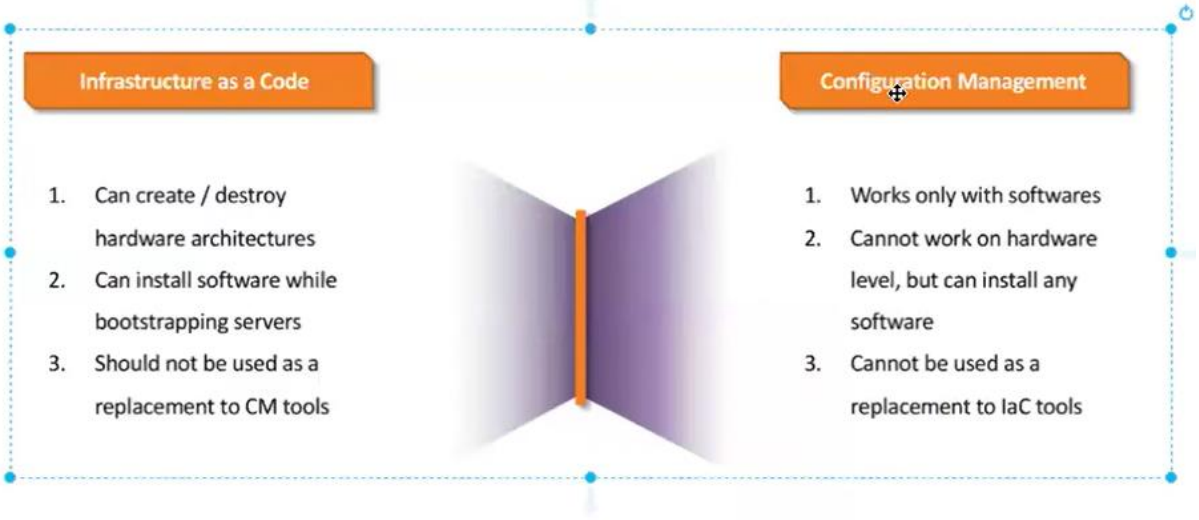
Terraform DAY-1

Agenda: Terraform

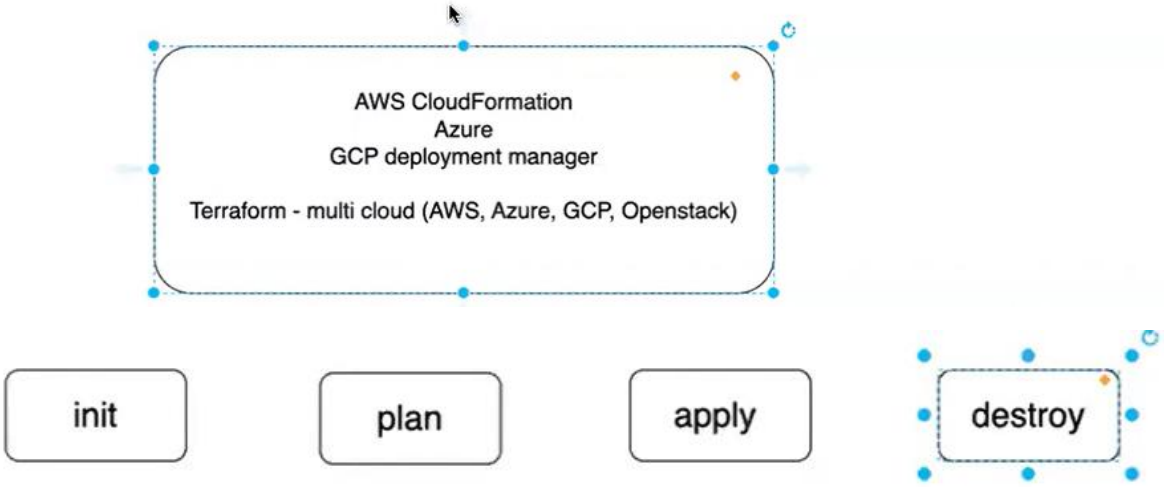
1. What is infrastructure as a code?
2. Infrastructure as a code vs Configuration Management
3. Intro to Terraform
4. Installing Terraform
5. Terraform Operations
6. Terraform code
7. Terraform Demo

immutability (0EWQw84Xngl

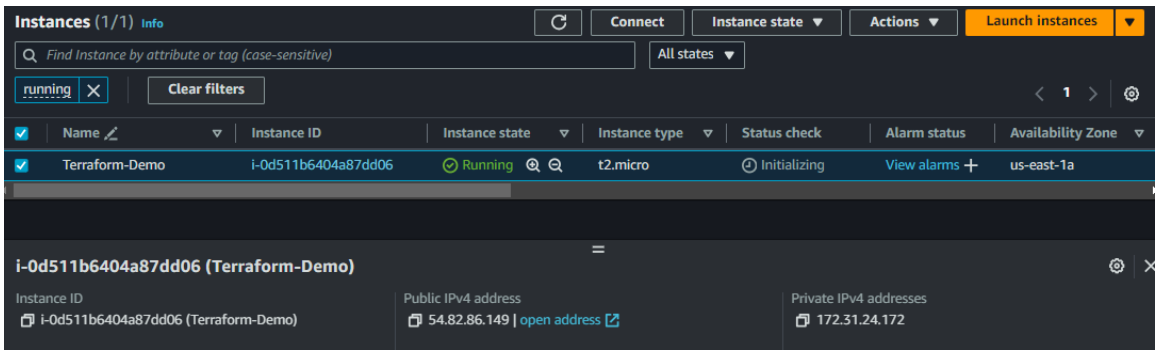


file -> configuration -> infrastructure will created

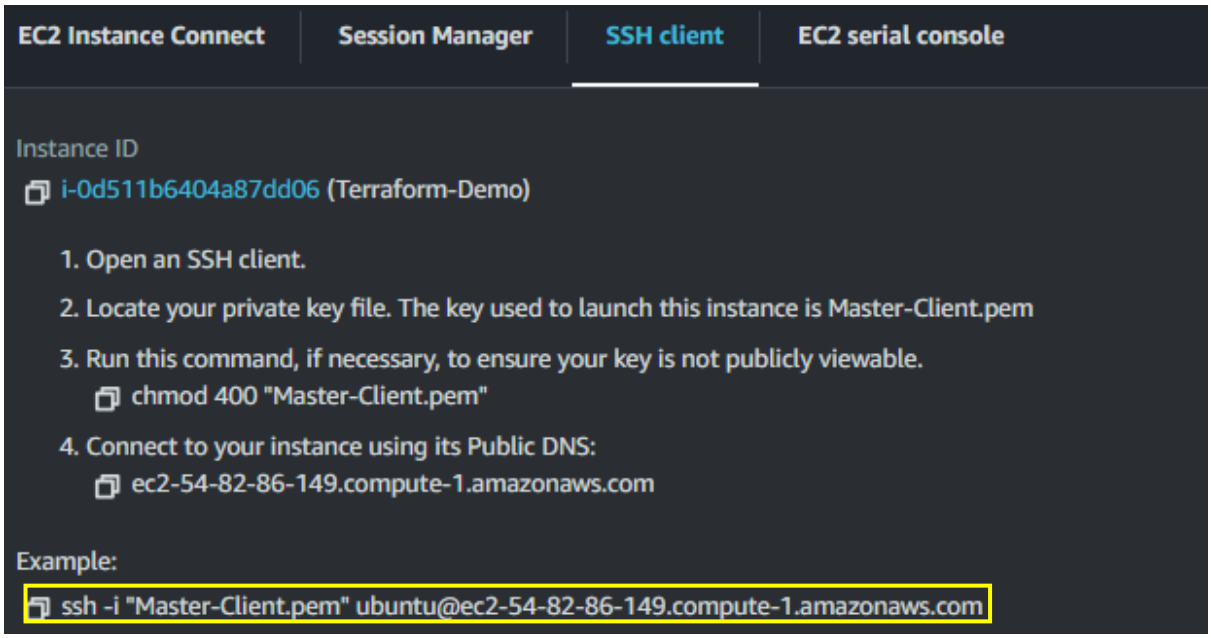
Terraform is an open-source infrastructure as code software tool created by HashiCorp. It enables users to define and provision a datacenter infrastructure using a high-level configuration language known as Hashicorp Configuration Language, or optionally JSON.



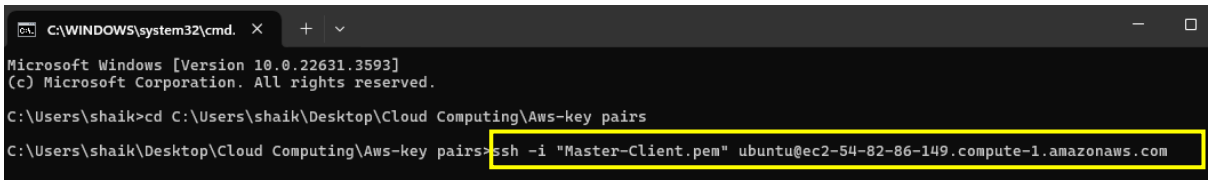
1. Launching Instances for Terraform task



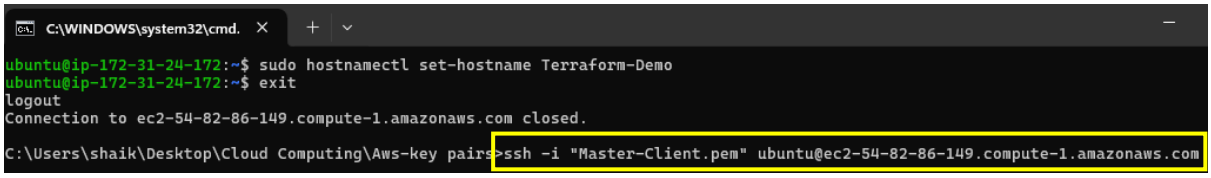
2. SSH-Client



3. Paste it on key pair path and connect the instances

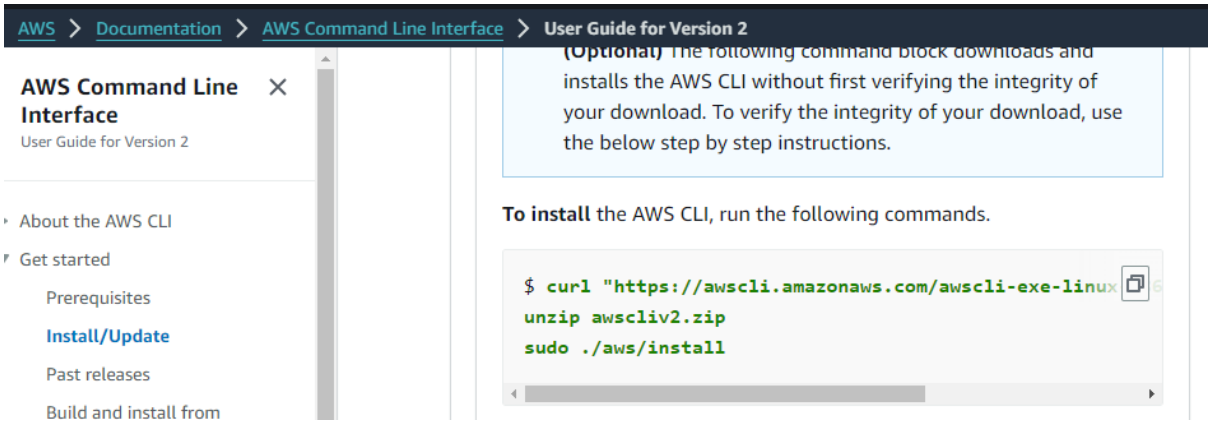


4. change the hostname name (Optional)

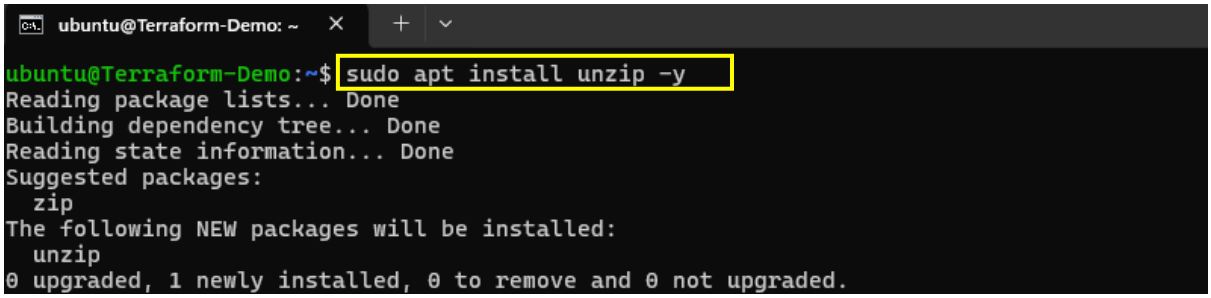


5. Installing the AWS CLI(Command Line Interface) and go through below the link

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html



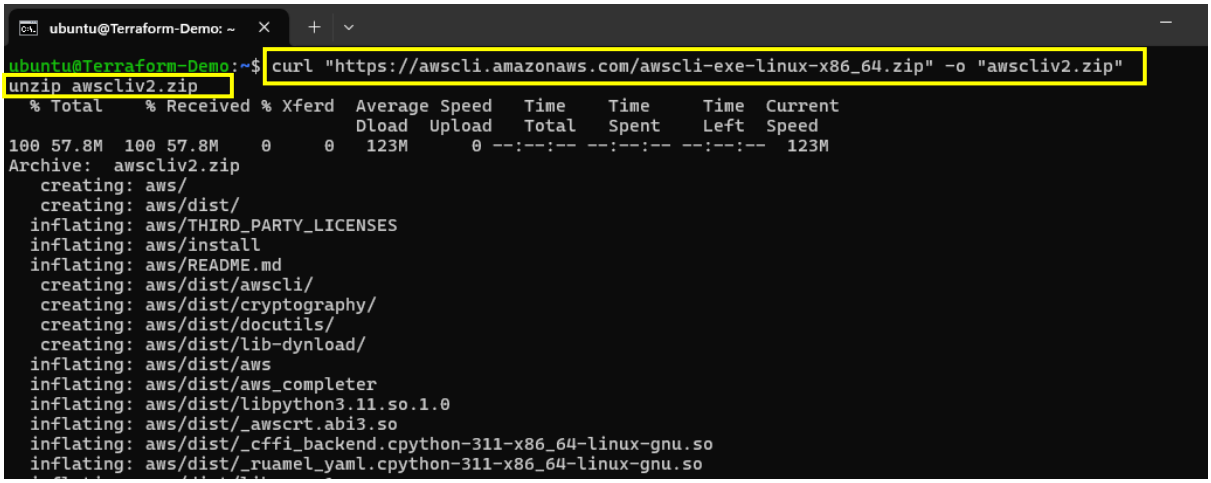
6. sudo apt install unzip -y



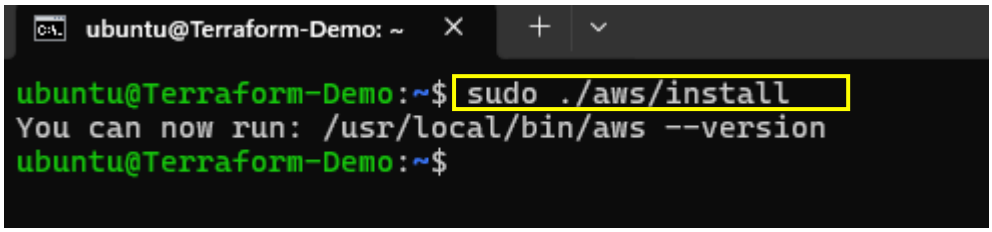
7. and then run below the commands to install AWS CLI

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

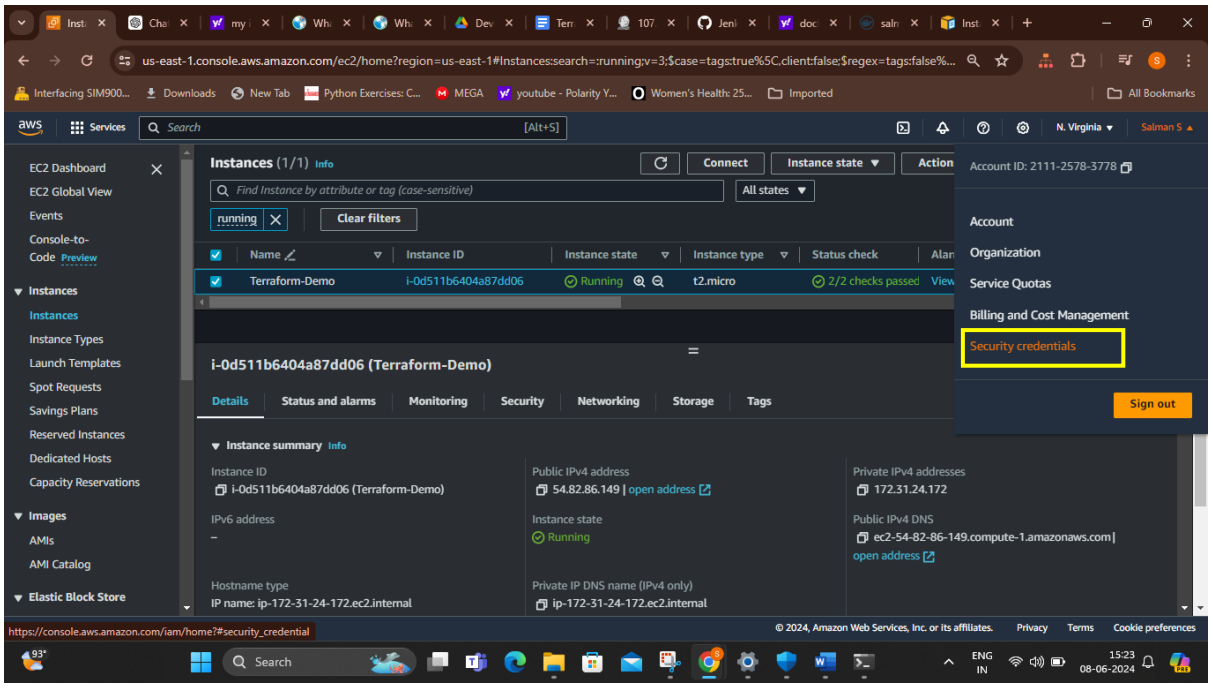
unzip awscliv2.zip



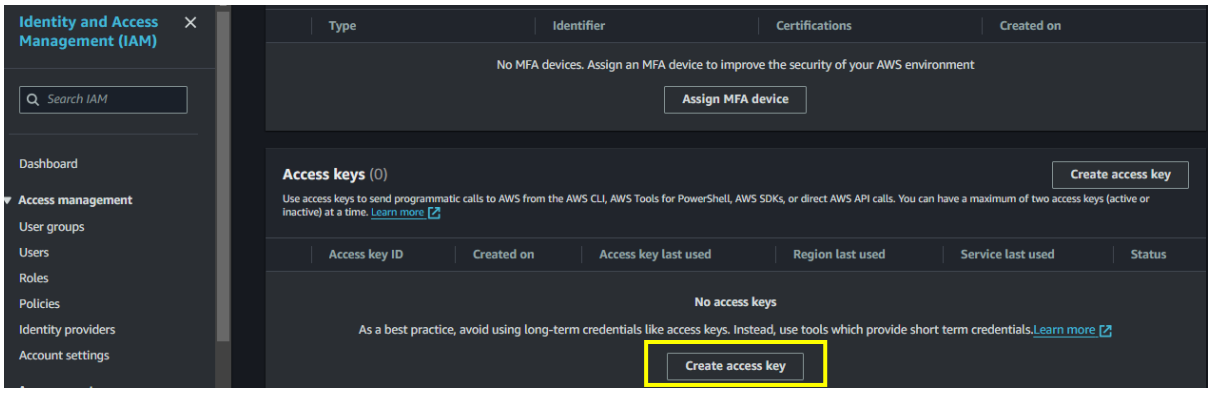
8. sudo ./aws/install



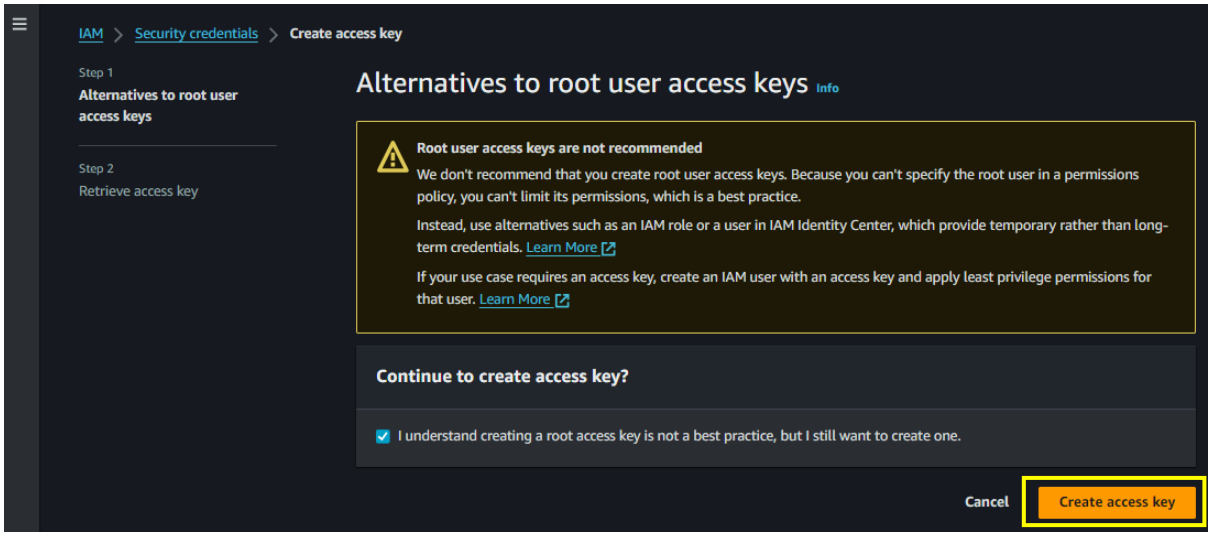
9. After Installing AWS CLI, Creating Security Credentials



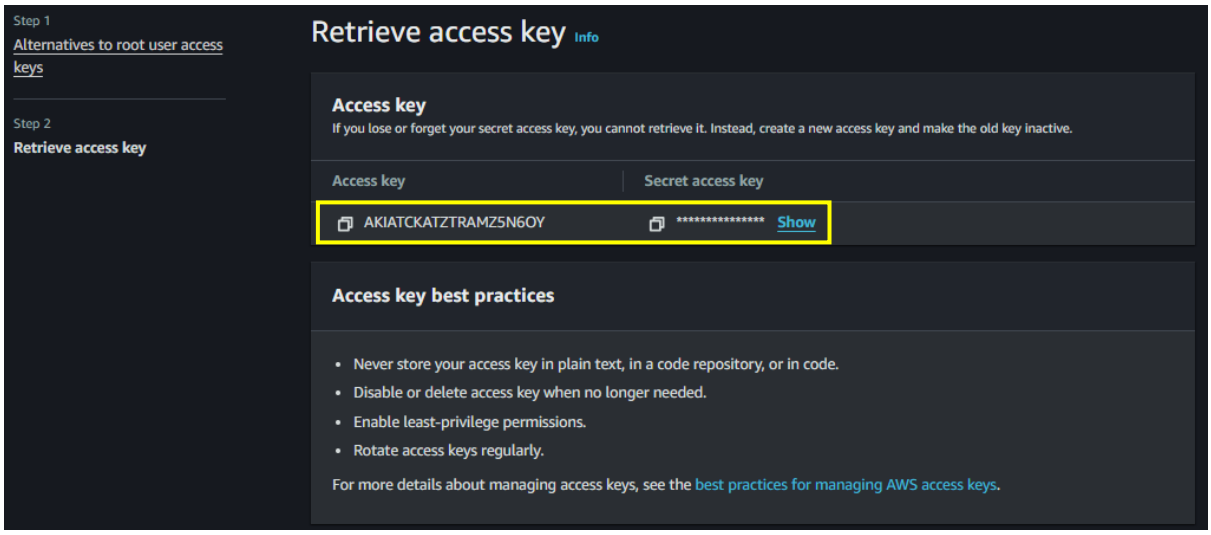
10. It will trigger to IAM Click on create access key



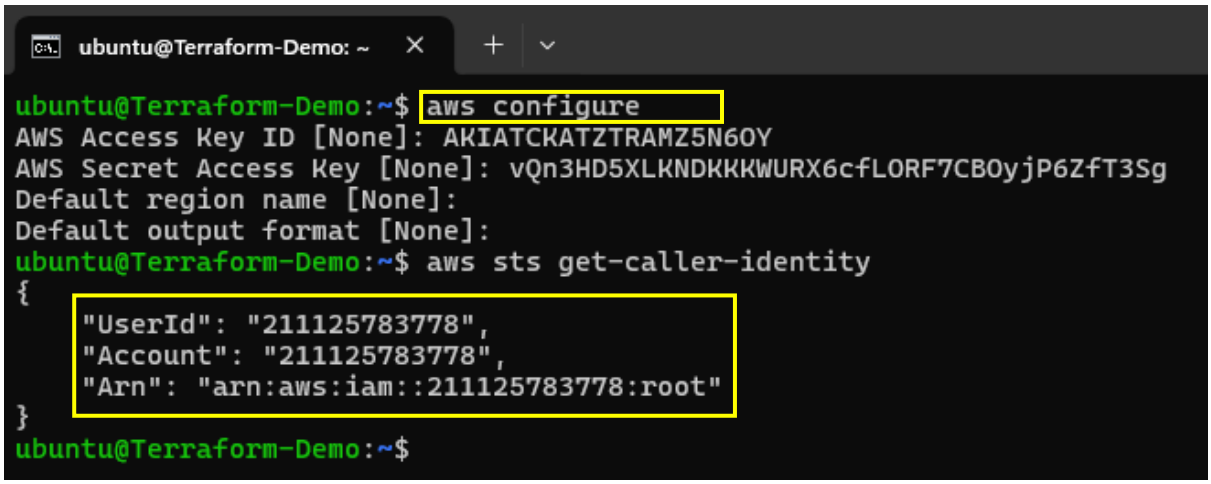
11. Create access key



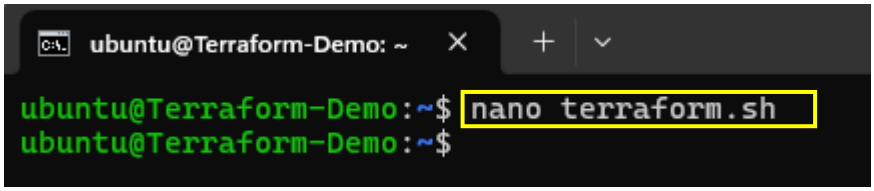
12. Copy the Credentials on local machine notepad.



13. After Installing AWS CLI, Now we can configure on aws and paste the credentials and it will connected to your AWS Account.



13. Now installing Terraform



14. sudo apt-get update && sudo apt-get install -y gnupg software-properties-common

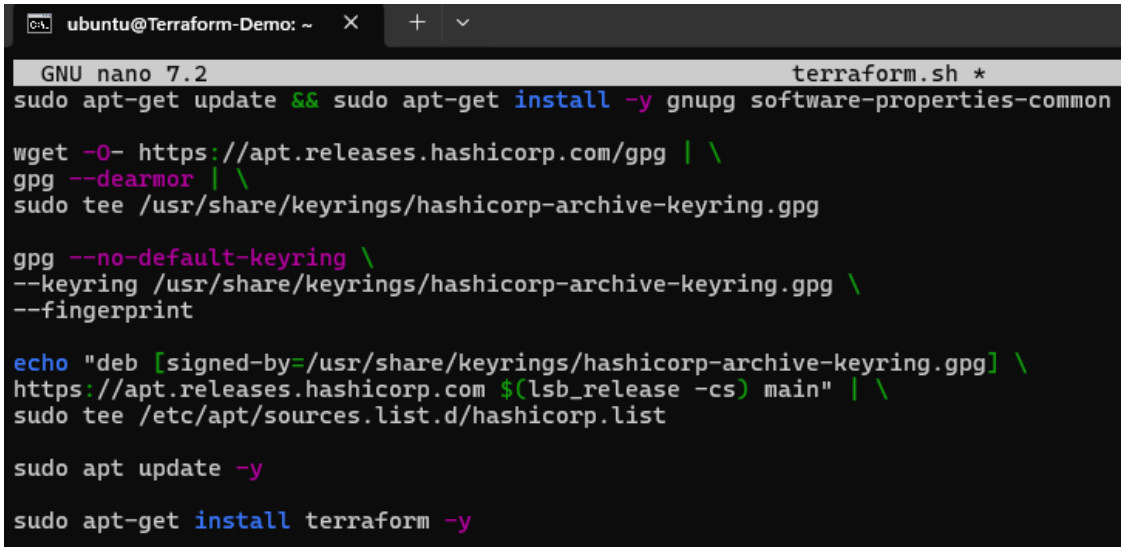
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg

gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint

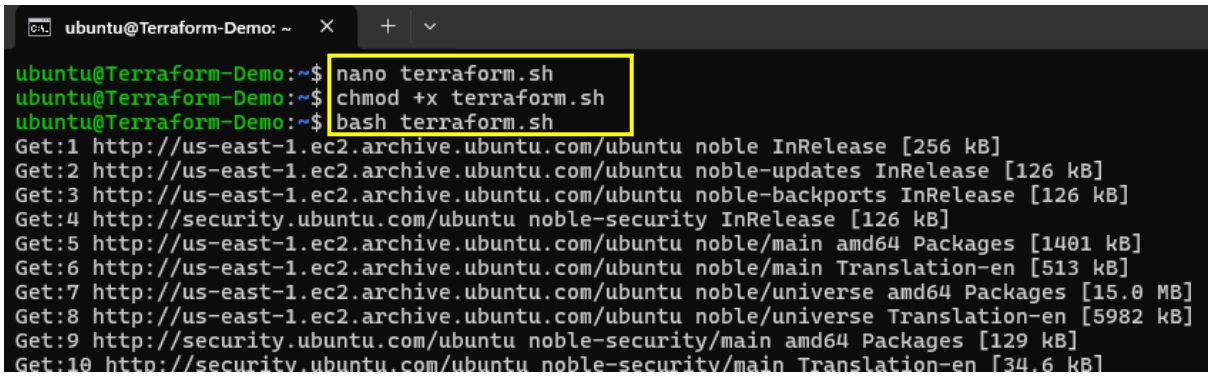
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update -y

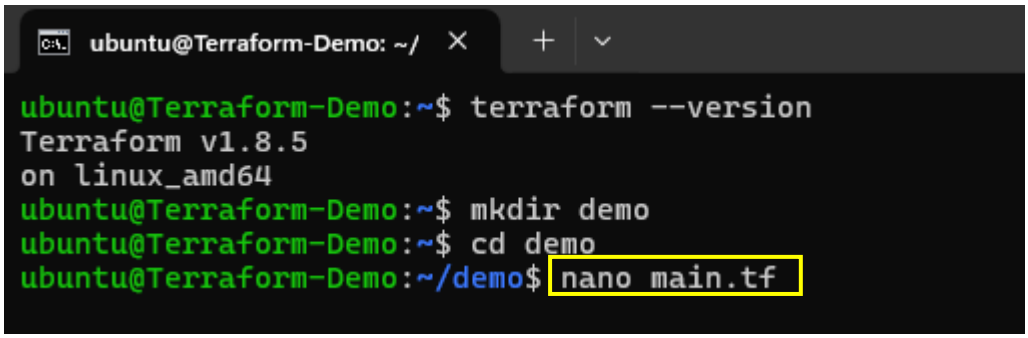
sudo apt-get install terraform -y



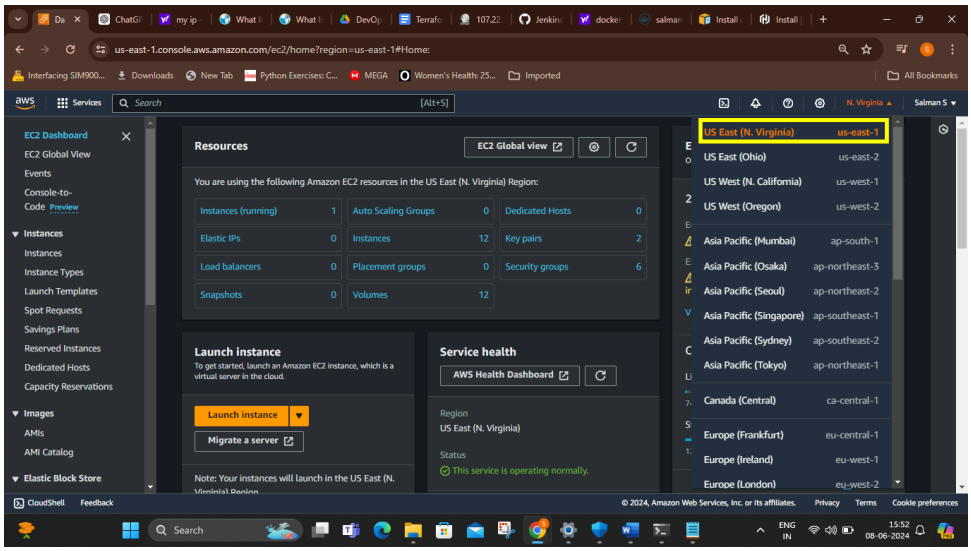
15. and Execute it.



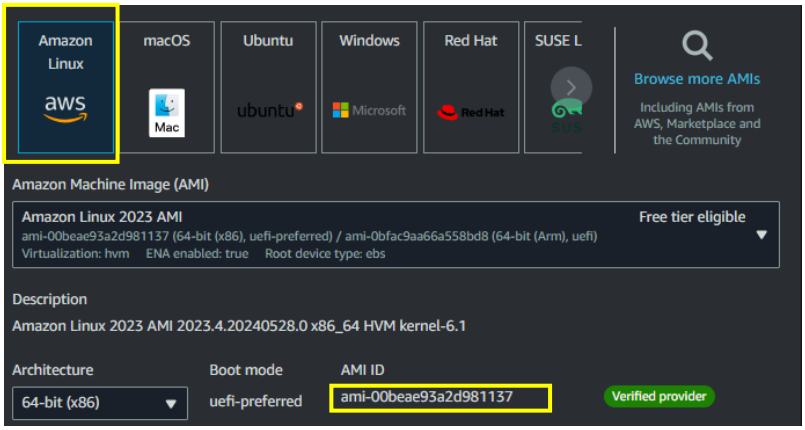
16. Checking that terraform is installed or not, and created demo folder and inside that creating main.tf terraform script file



17. Now I am launching instaces at North virginia(us-east-1)



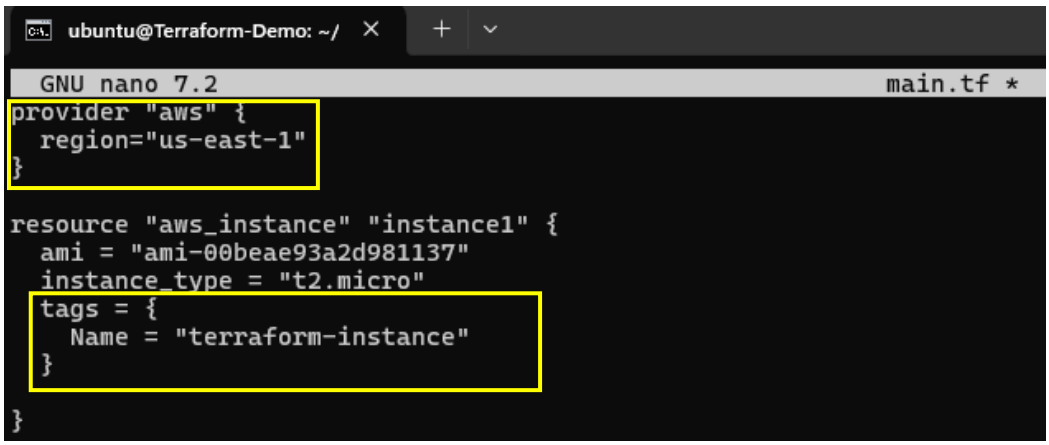
18. Copy AMI, I am here Launching Amazon Linux



19. Now this the code to Launching instances on AWS using terraform

Terraform we can Install infrastructure at different-different cloud providers

Provider : aws, region: us-east-1(N.V) resource “aws_instances” Name of Instances: terraform-instance, ami and instance type



- **Init:**
  - Initializes the Terraform workspace by downloading and installing required plugins mentioned in your Terraform configuration files (`.tf` files).
- **Plan:**
  - Reads your Terraform configuration and compares it to the current state of your infrastructure (if any exists).
  - Generates an execution plan that outlines the actions Terraform would take to create or modify resources to match your desired state as defined in the configuration.
- **Apply:**
  - Executes the plan generated in the previous step.
  - Creates, updates, or destroys resources as necessary to match your desired infrastructure state.
- **Destroy:**
  - Similar to `plan`, it creates a plan to destroy your infrastructure based on your Terraform configuration.
  - Once confirmed, it removes the resources managed by Terraform, essentially tearing down your infrastructure.

20. Now terraform init(Downloading and Installing required plugins)

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/demo$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.53.0...
- Installed hashicorp/aws v5.53.0 (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.
ubuntu@Terraform-Demo:~/demo$
```

21. We can see the its downloaded the things

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/demo$ cd .terraform/
ubuntu@Terraform-Demo:~/demo/.terraform$ ls
providers
ubuntu@Terraform-Demo:~/demo/.terraform$ cd providers
ubuntu@Terraform-Demo:~/demo/.terraform/providers$ ls
registry.terraform.io
ubuntu@Terraform-Demo:~/demo/.terraform/providers$ cd registry.terraform.io
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io$ ls
hashicorp
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io$ cd hashicorp
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp$ ls
aws
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp$ cd aws
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp/aws$ ;s
-bash: syntax error near unexpected token `;'
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp/aws$ ls
5.53.0
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp/aws$ cd 5.53.0
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp/aws/5.53.0$ ls
linux_amd64
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp/aws/5.53.0$ cd linux_amd64
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp/aws/5.53.0/linux_amd64$ ls
LICENSE.txt  terraform-provider-aws_v5.53.0_x5
ubuntu@Terraform-Demo:~/demo/.terraform/providers/registry.terraform.io/hashicorp/aws/5.53.0/linux_amd64$
```

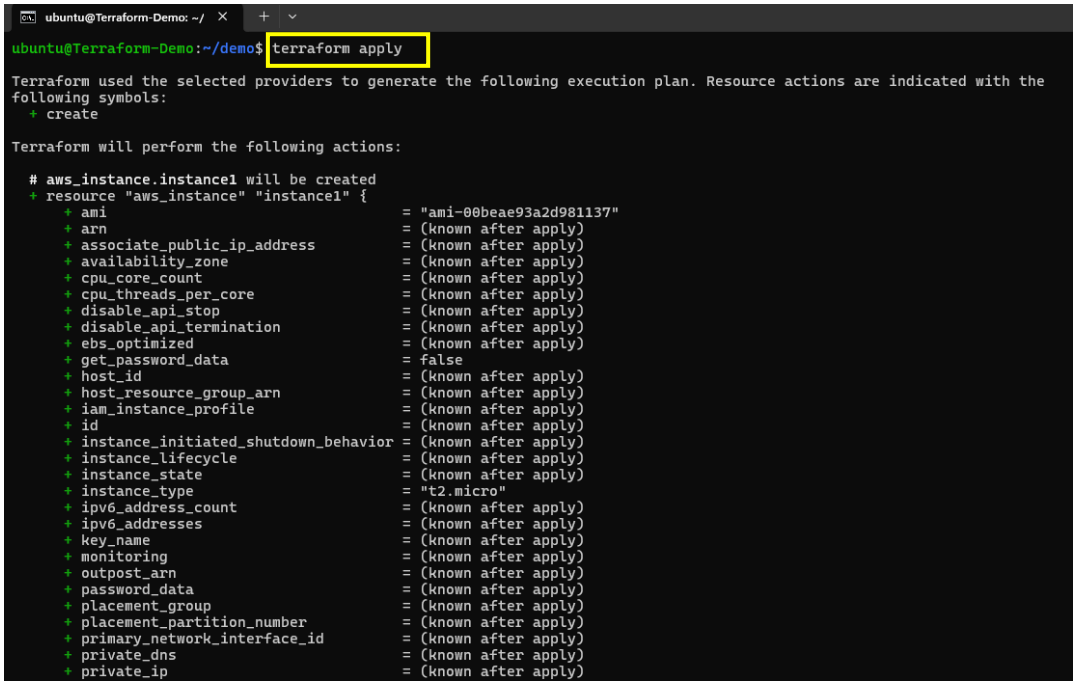
22. Now terraform plan(Reads our Configurations, generate and execute the plans)

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/demo$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.instance1 will be created
  + resource "aws_instance" "instance1" {
      + ami                                  = "ami-00beae93a2d981137"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
```

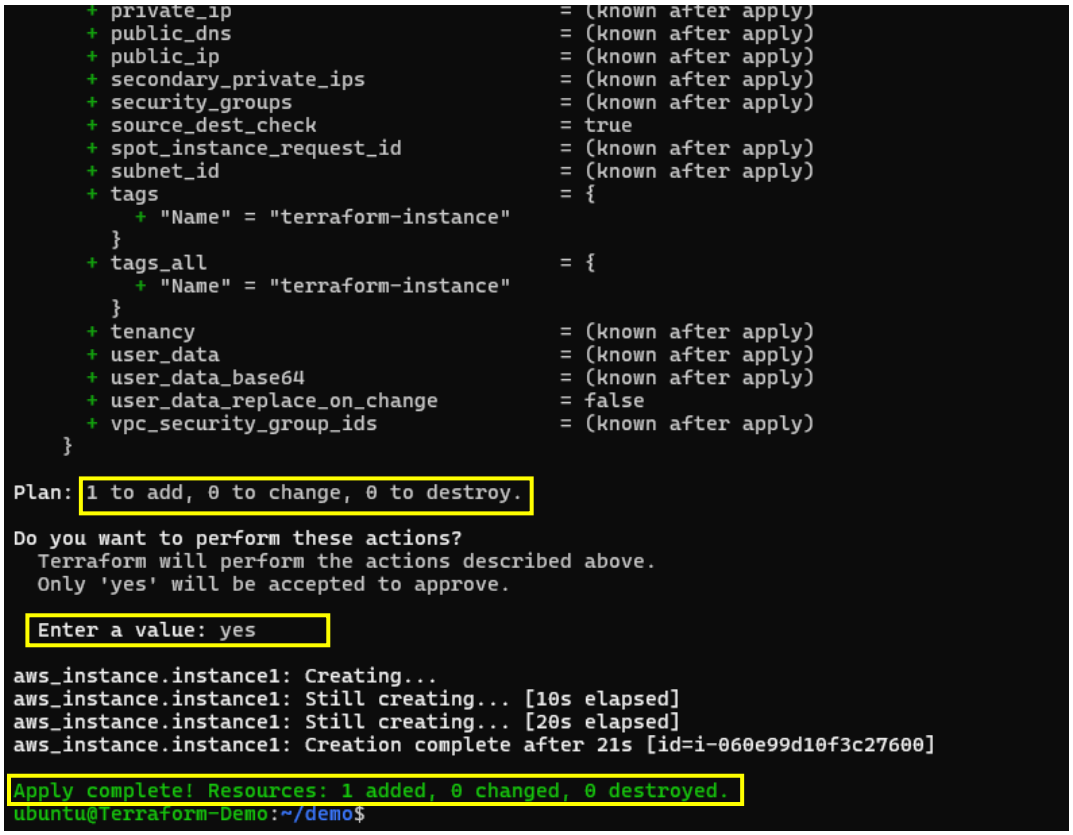
23. 1 to add there is no changes and there is no destroy and create. Because this is our first execution

```
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "terraform-instance"
        }
      + tags_all                             = {
          + "Name" = "terraform-instance"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

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.
ubuntu@Terraform-Demo:~/demo$
```

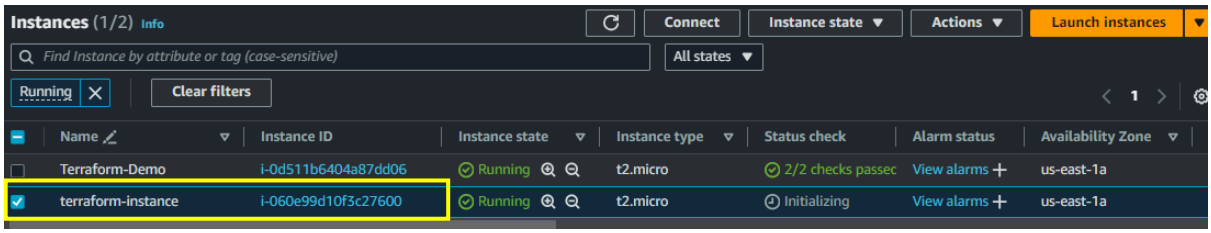
24. terraform apply now, it will going to install our infrastructure



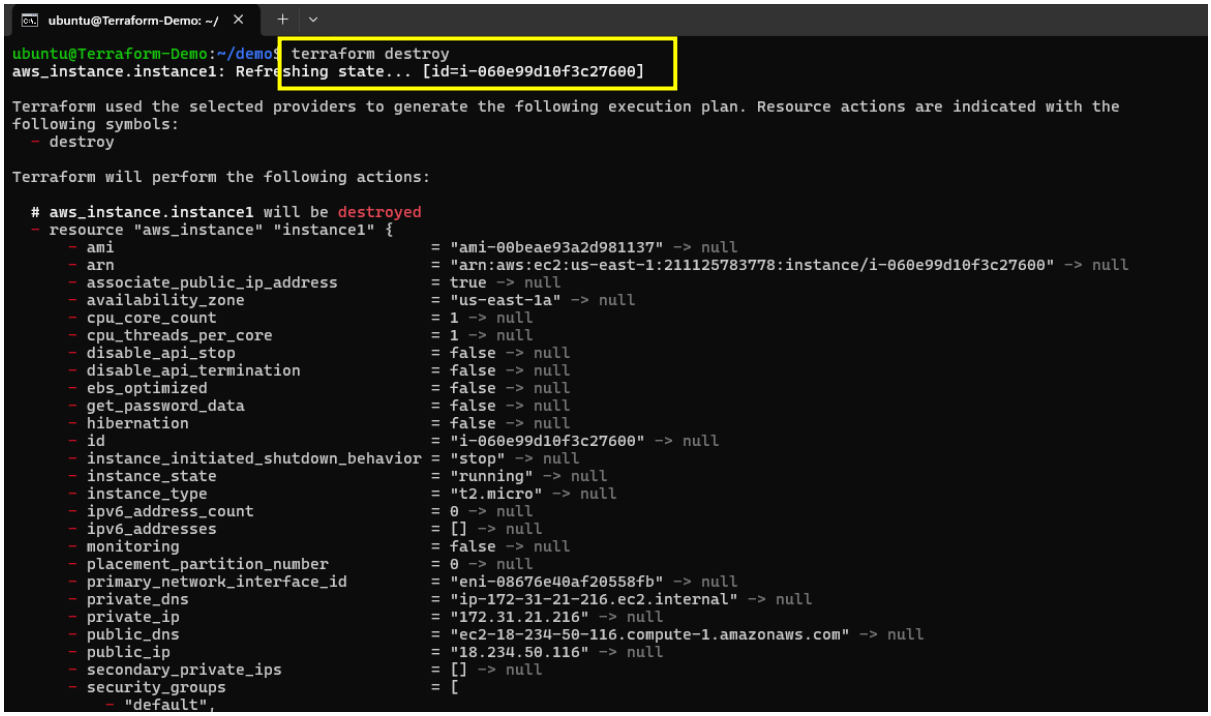
25. click on yes and it will start creating the infrastructure



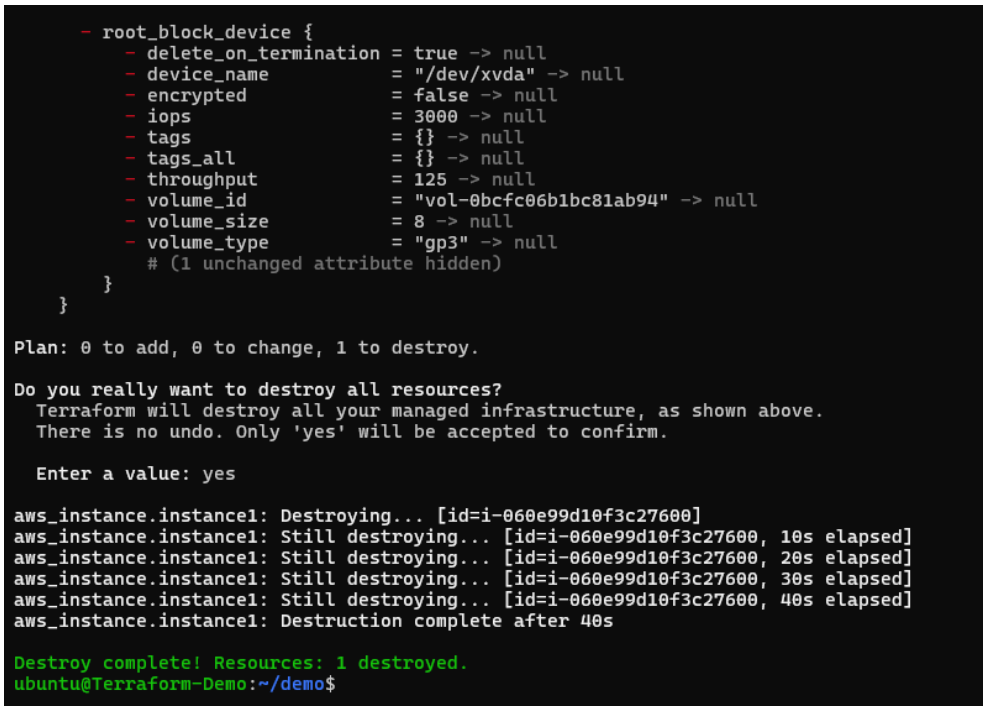
26. We can see the terraform-instances has been created, we given tag name terraform-instance



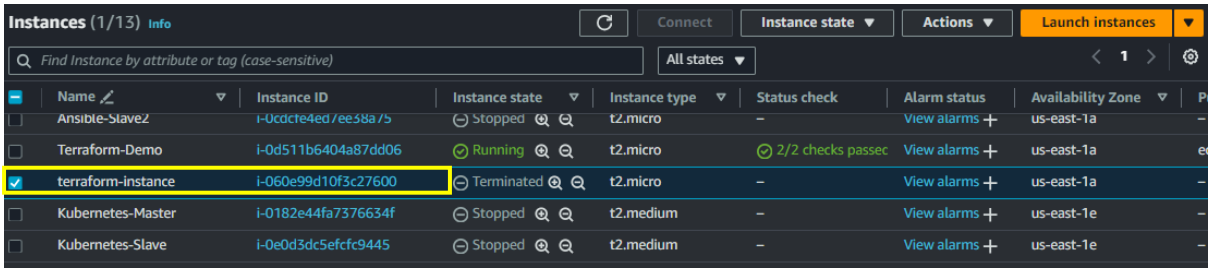
27. terraform destroy, means it will destroy the infrastructure



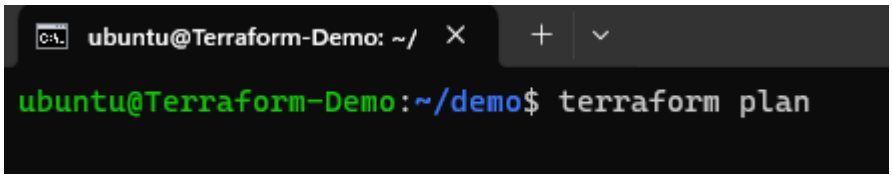
28. Previously 1 added now its 1 destroy.



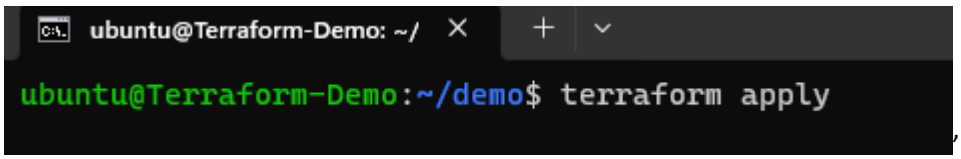
29. Now the terraform-instance is terminated.



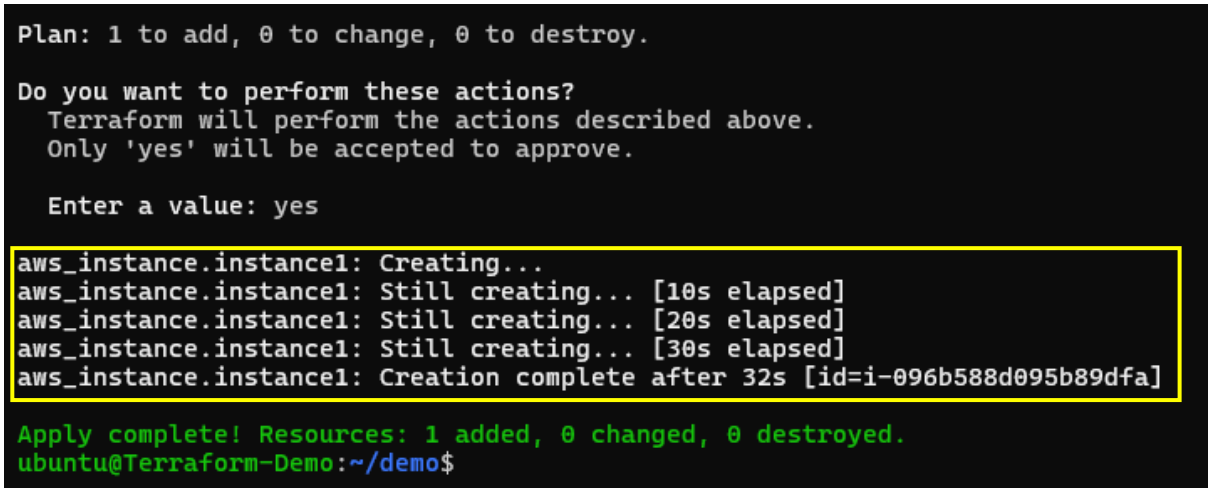
30. Again terraform plan and no need to terraform init, because we already initialize



31. again terraform apply



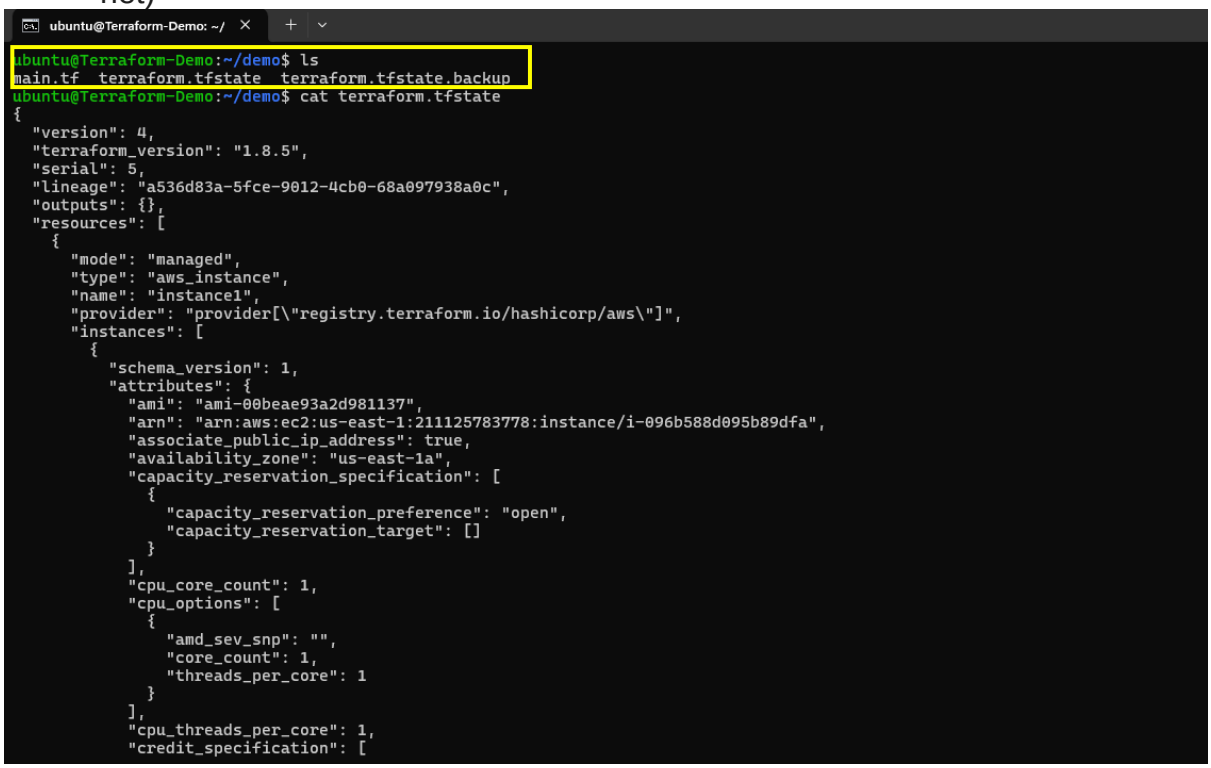
32. Again it will going to creating one instances



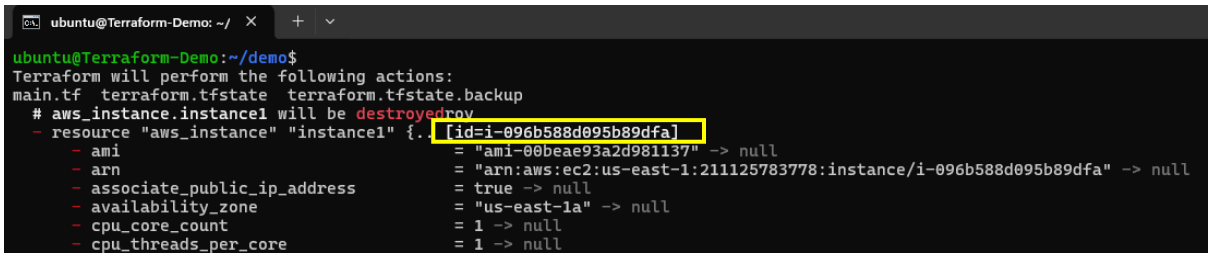
33. See Now its again running status

34. In Terraform:

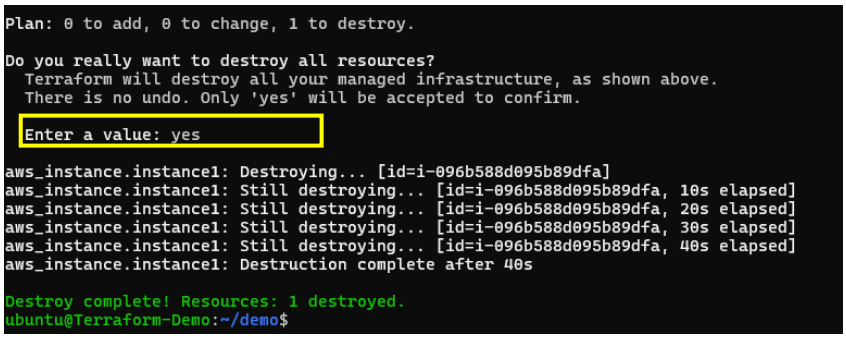
- **terraform.tfstate**: Stores the **current state** of your infrastructure managed by Terraform (resources created, configurations applied). (This is the main file)
- **terraform.tfstate.backup**: Acts as a **backup** of the `terraform.tfstate` file, created automatically before significant changes are made. (This is a safety net)



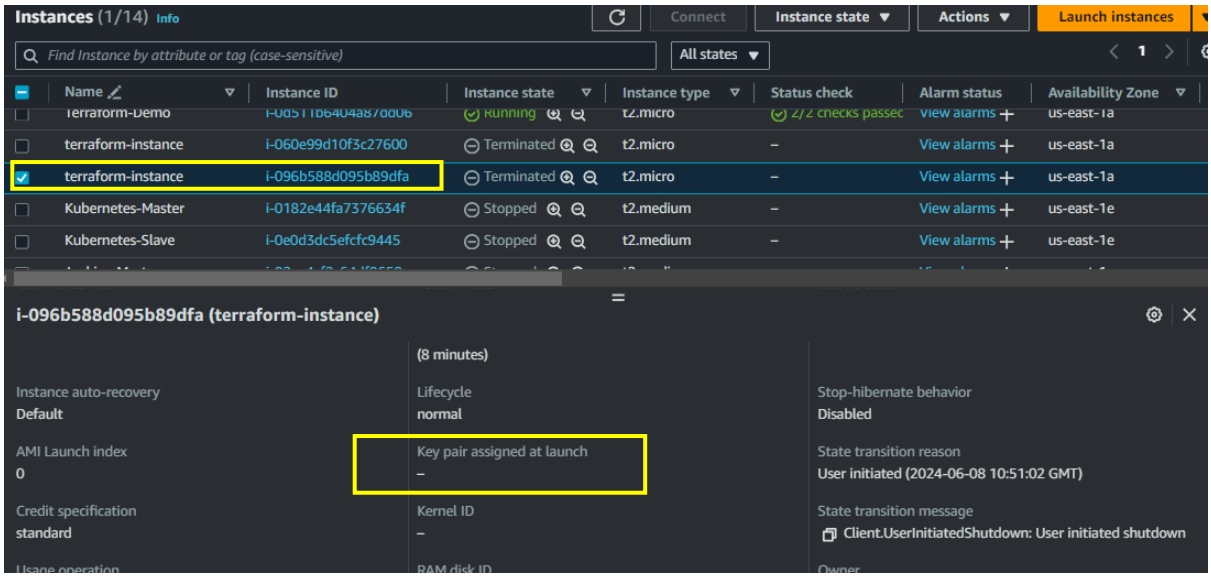
35. terraform destroy it will destroy your insfrastructure again



36. Now its destroying

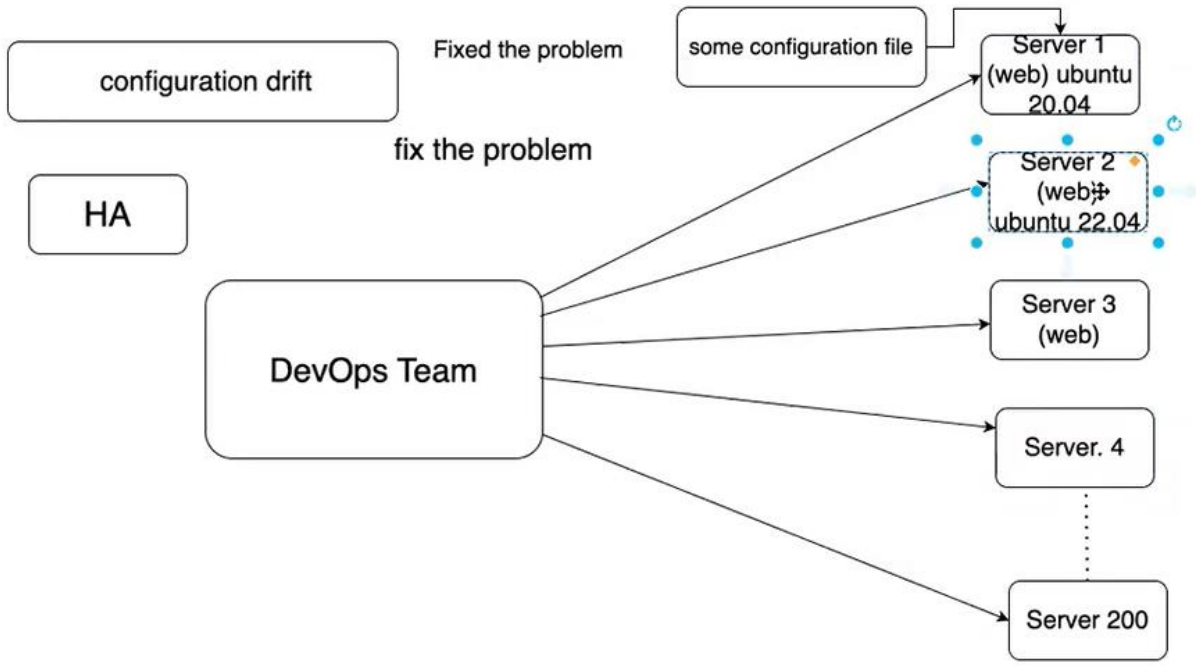


37. its terminated.



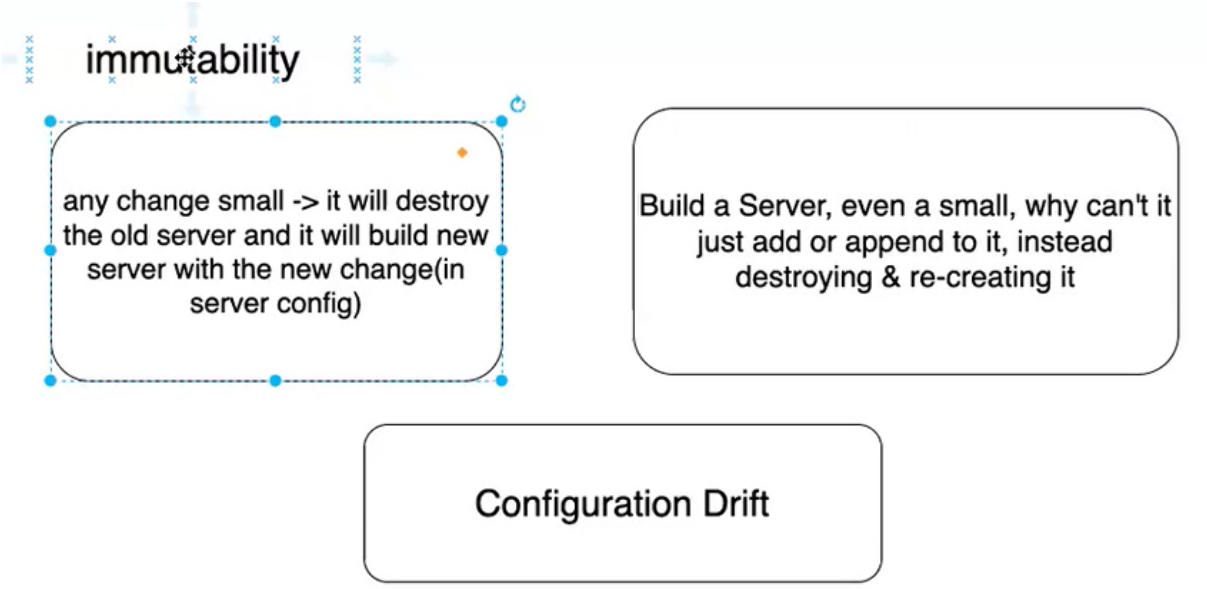
CONFIGURATION DRIFT:

Configuration drift refers to the gradual, unintended divergence between the actual state of deployed infrastructure and its intended state as defined in code. It typically occurs due to manual changes or updates made directly to resources outside of the defined configuration management system, like Terraform.

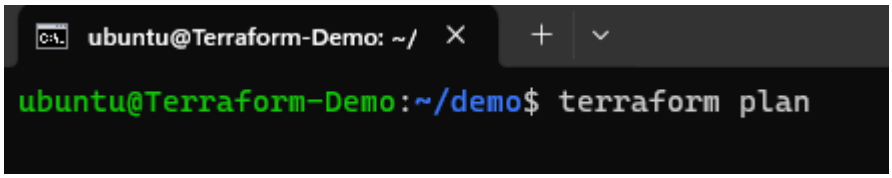


IMMUTABILITY:

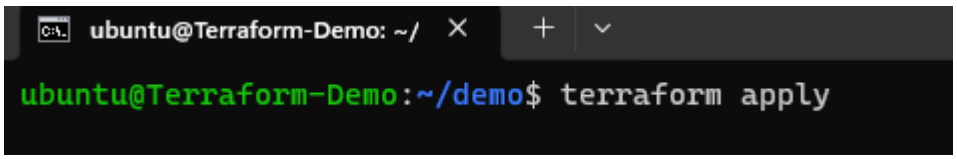
Immutability, in the context of Terraform, refers to the principle of never modifying infrastructure in-place. Instead, it involves creating new resources or replacing existing ones when changes are needed, ensuring consistency and predictability. This approach minimizes configuration drift and promotes reproducibility in infrastructure deployment.



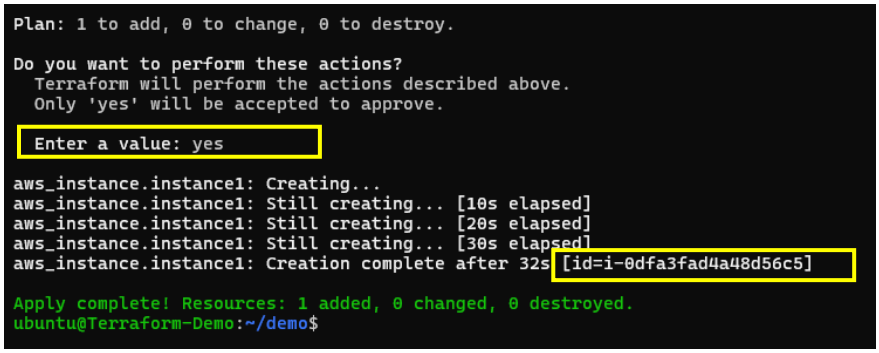
38. Again Terraform plan



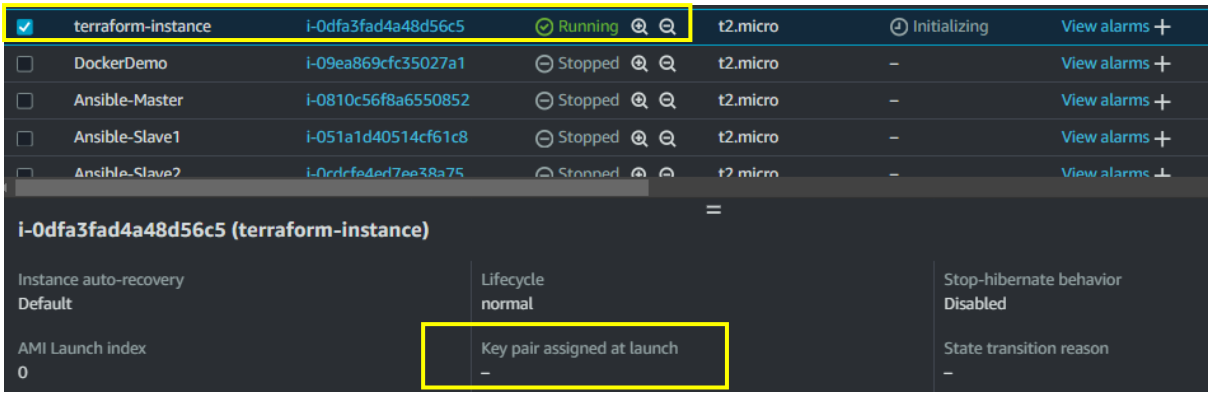
39. Again Terraform apply



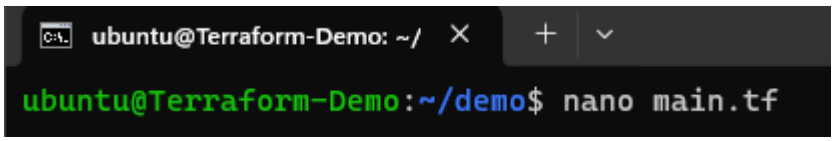
40. Now Again its creating the infrastructure



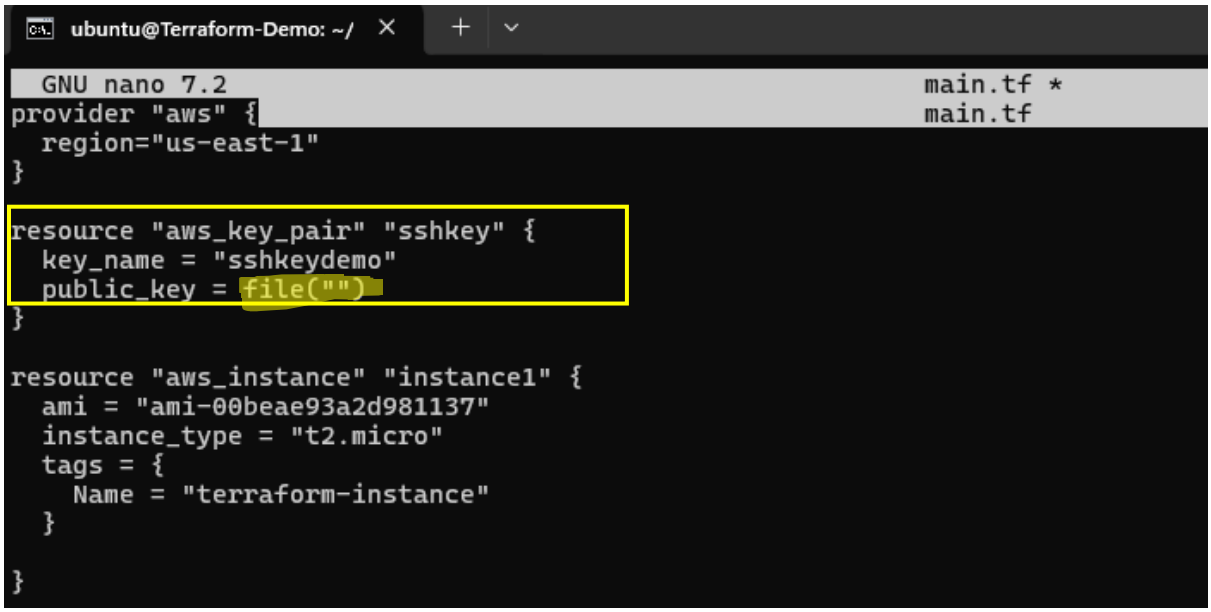
41. We can see, but there is no key-pair attached here



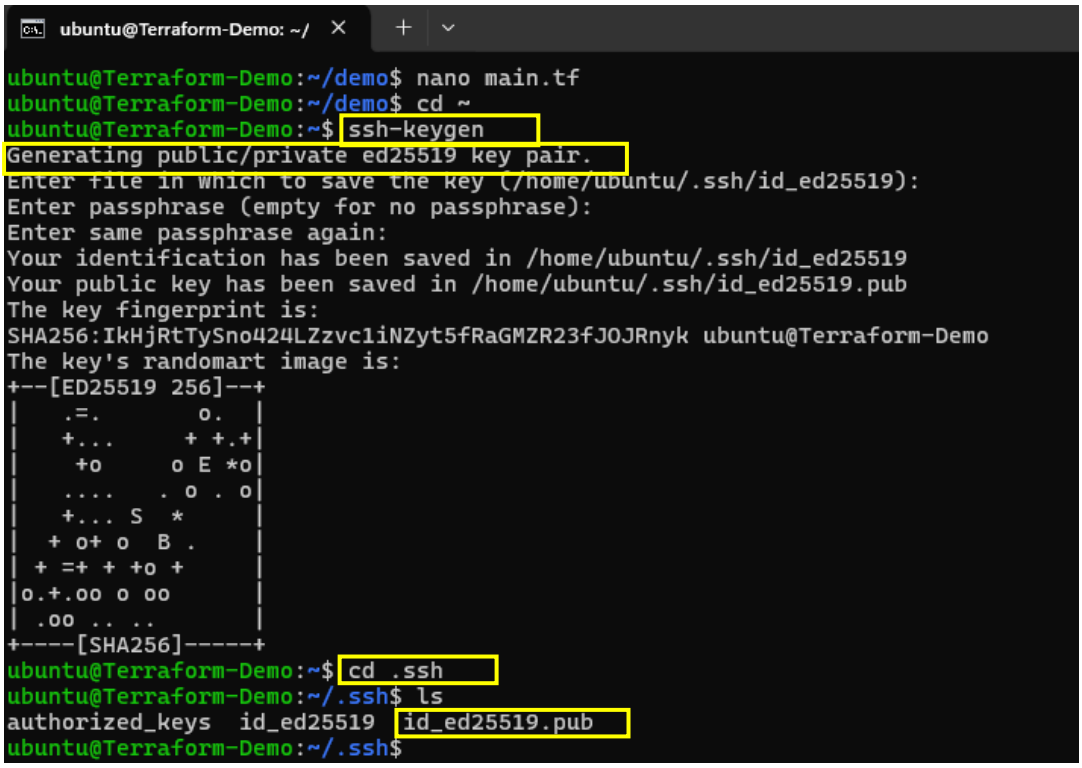
42. Now Again going to modifying the main.tf file



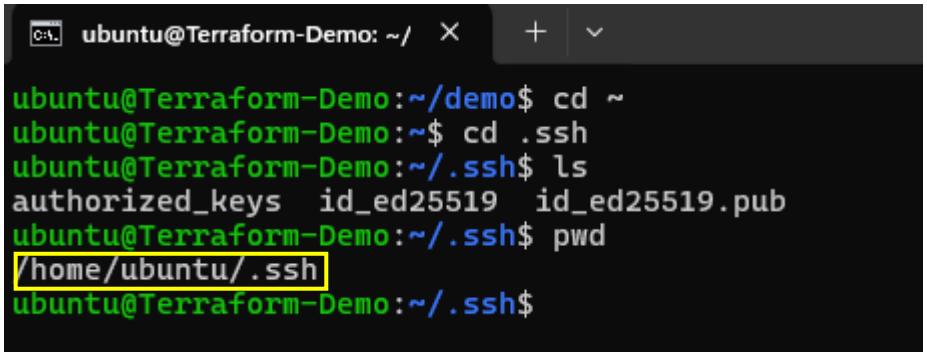
43. resource key-pair named as sshkeydemo and public-key leaving empty as of now and follow the upcoming steps



44. generating ssh-keygen and after that going inside the .ssh/

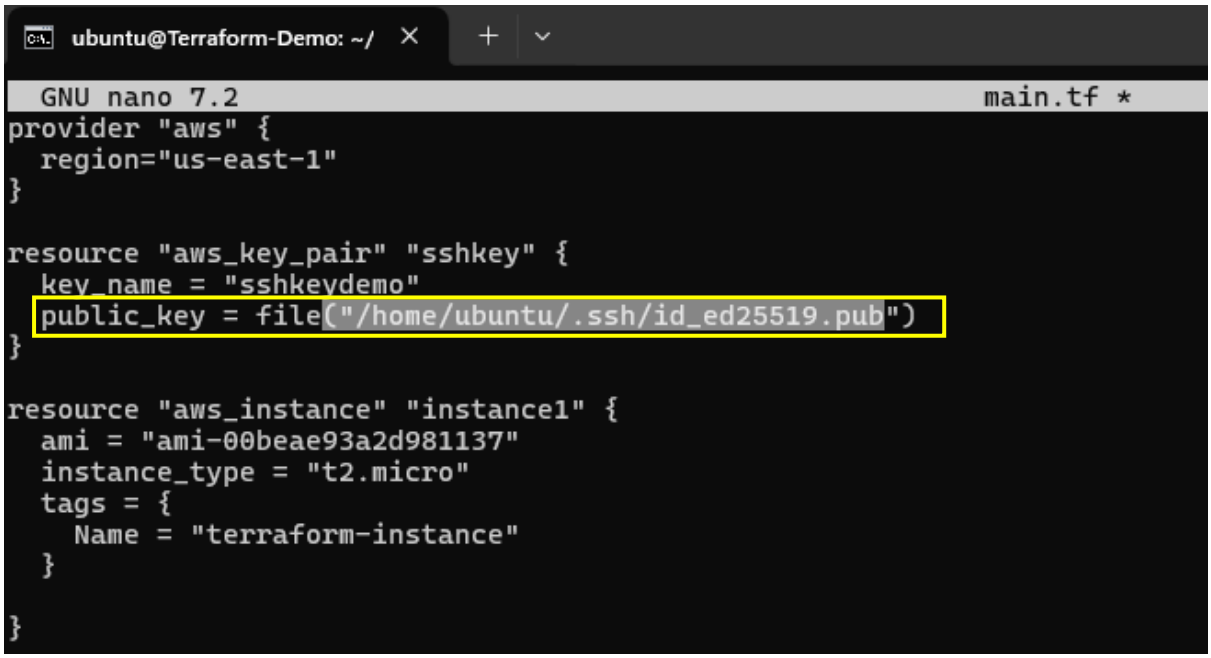


45. and copy the path of present working directory

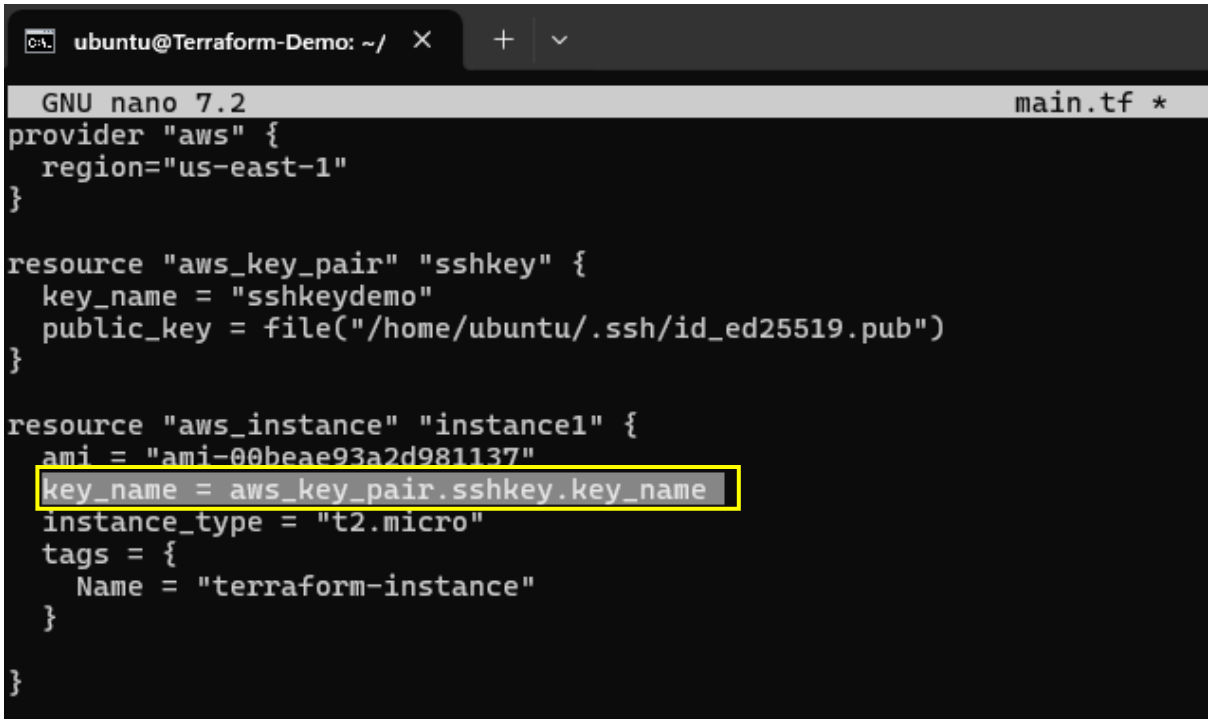


46. and come demo folder and go to main.tf file

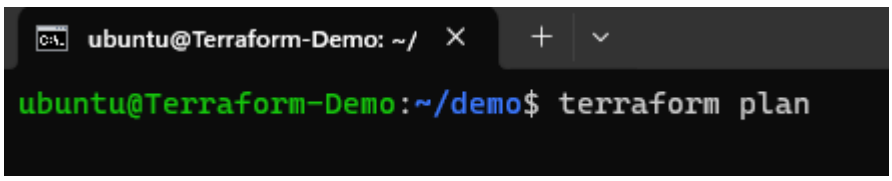
47. paste the path and public key file



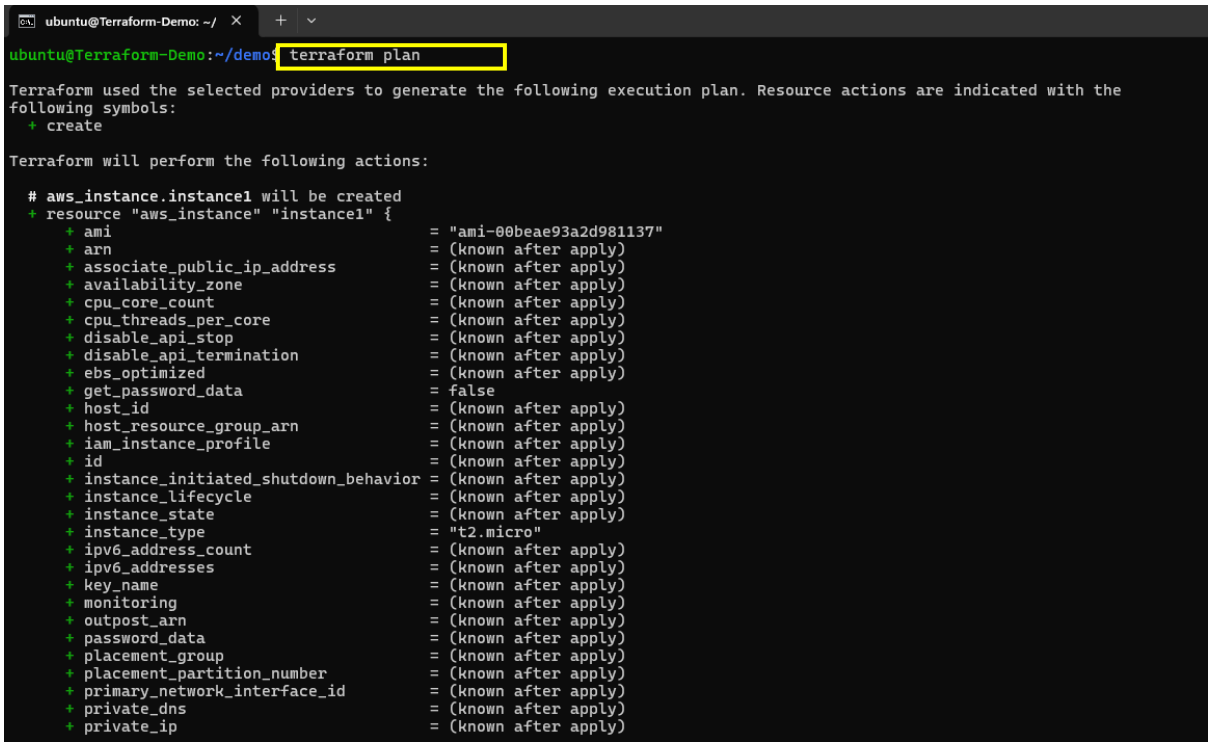
48. And paste key_name in resource part, and It will trigger to public_key, save and exit



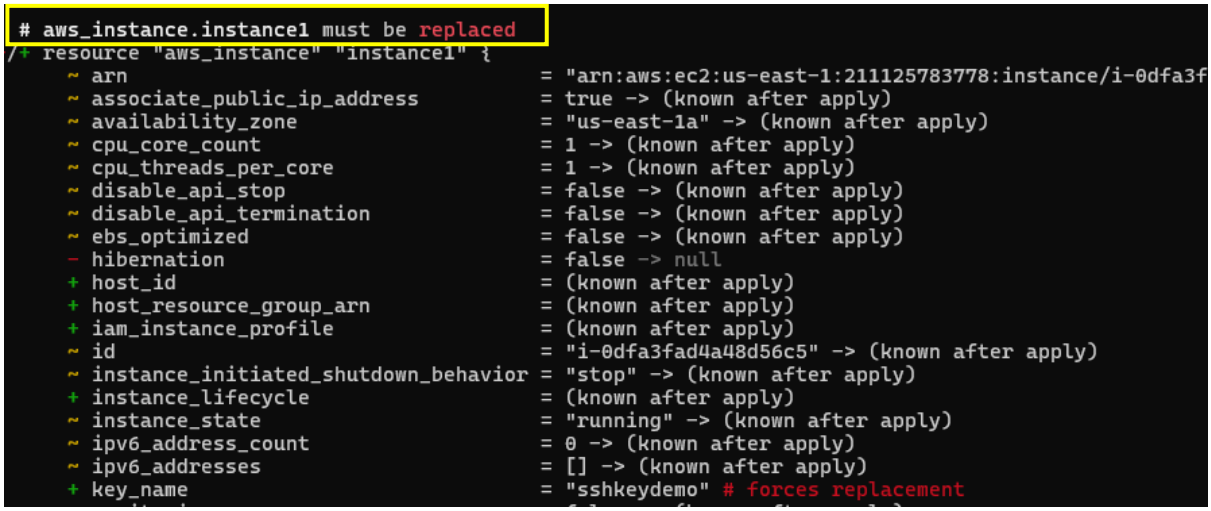
49. And Now terraform plan(Reads Our Configurations, generate and execute the plans)



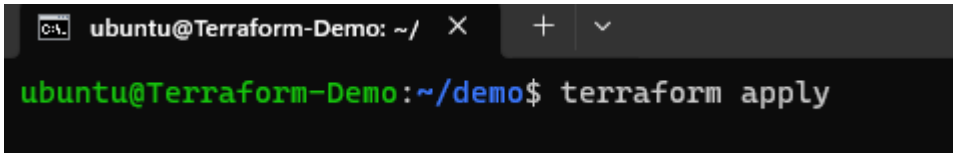
50. Now I Again its Executing the plan



51. Here are replaced



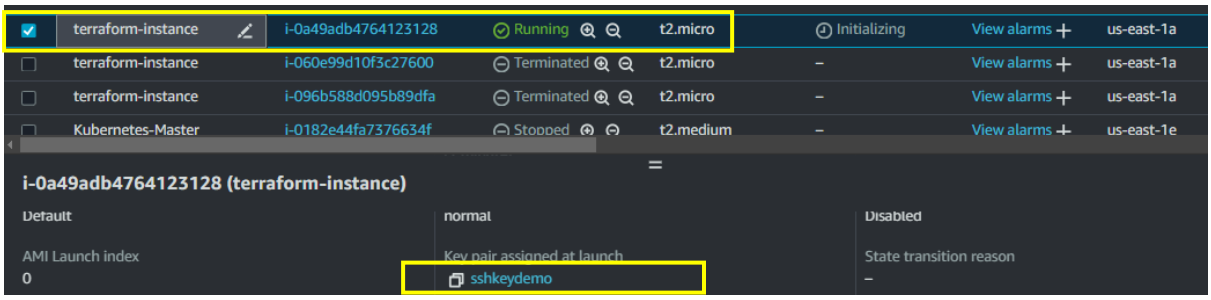
52. Terraform Apply



53. We can see it will adding 2 and 1 to destroy, that means its destroying the instances and adding the instances and key -pair

```
Plan: 2 to add, 0 to change, 1 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.instance1: Destroying... [id=i-0dfa3fad4a48d56c5]
aws_instance.instance1: Still destroying... [id=i-0dfa3fad4a48d56c5, 10s elapsed]
aws_instance.instance1: Still destroying... [id=i-0dfa3fad4a48d56c5, 20s elapsed]
aws_instance.instance1: Still destroying... [id=i-0dfa3fad4a48d56c5, 30s elapsed]
aws_instance.instance1: Still destroying... [id=i-0dfa3fad4a48d56c5, 40s elapsed]
aws_instance.instance1: Destruction complete after 40s
aws_key_pair.sshkey: Creating...
aws_key_pair.sshkey: Creation complete after 0s [id=sshkeydemo]
aws_instance.instance1: Creating...
aws_instance.instance1: Still creating... [10s elapsed]
aws_instance.instance1: Still creating... [20s elapsed]
aws_instance.instance1: Still creating... [30s elapsed]
aws_instance.instance1: Creation complete after 32s [id=i-0a49adb4764123128]

Apply complete! Resources: 2 added, 0 changed, 1 destroyed.
ubuntu@Terraform-Demo:~/demo$
```

54. We can see the key-pair attached



55. We Can see the Public Key-Pair

56. Again go to main.tf file

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~$ ls
aws  awscliv2.zip  demo  terraform.sh
ubuntu@Terraform-Demo:~$ cd demo
ubuntu@Terraform-Demo:~/demo$ ls
main.tf  terraform.tfstate  terraform.tfstate.backup
ubuntu@Terraform-Demo:~/demo$ nano main.tf
```

57. This time I am attaching private key pair and save it

```
ubuntu@Terraform-Demo: ~/
  GNU nano 7.2                                          main.tf *
provider "aws" {
  region="us-east-1"
}

resource "aws_instance" "instance1" {
  ami = "ami-00beae93a2d981137"
  key_name = "Master-Client"
  instance_type = "t2.micro"
  tags = {
    Name = "terraform-instance"
  }

}
```

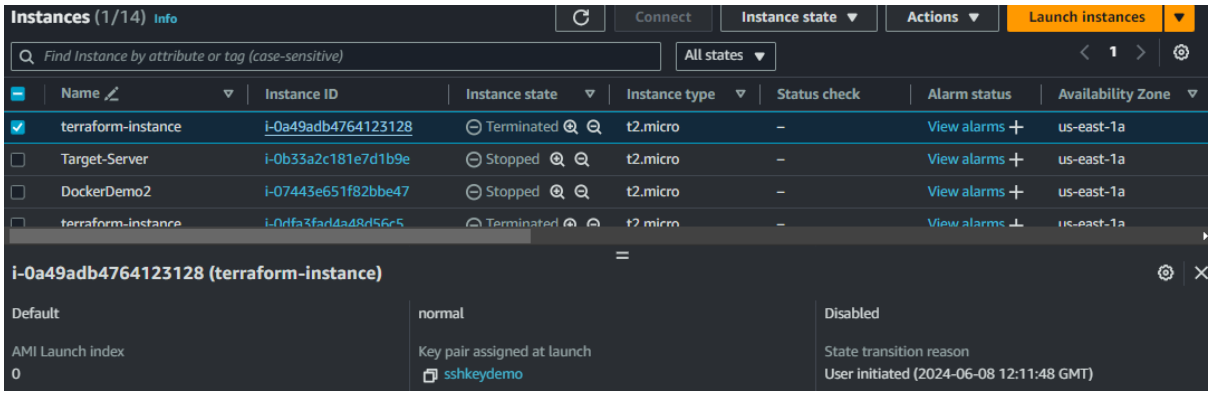
58. Now, I am last one destroying

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/demo$ terraform destroy
```

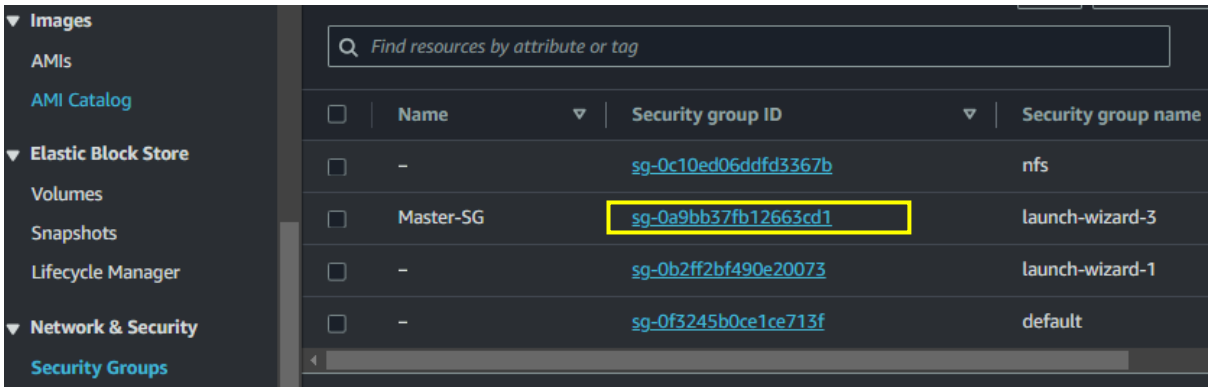
59. we can see the 2 destroys

```
Plan: 0 to add, 0 to change, 2 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.instance1: Destroying... [id=i-0a49adb4764123128]
aws_instance.instance1: Still destroying... [id=i-0a49adb4764123128, 10s elapsed]
aws_instance.instance1: Still destroying... [id=i-0a49adb4764123128, 20s elapsed]
aws_instance.instance1: Still destroying... [id=i-0a49adb4764123128, 30s elapsed]
aws_instance.instance1: Still destroying... [id=i-0a49adb4764123128, 40s elapsed]
aws_instance.instance1: Destruction complete after 40s
aws_key_pair.sshkey: Destroying... [id=sshkeydemo]
aws_key_pair.sshkey: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
ubuntu@Terraform-Demo:~/demo$
```

60. its terminated

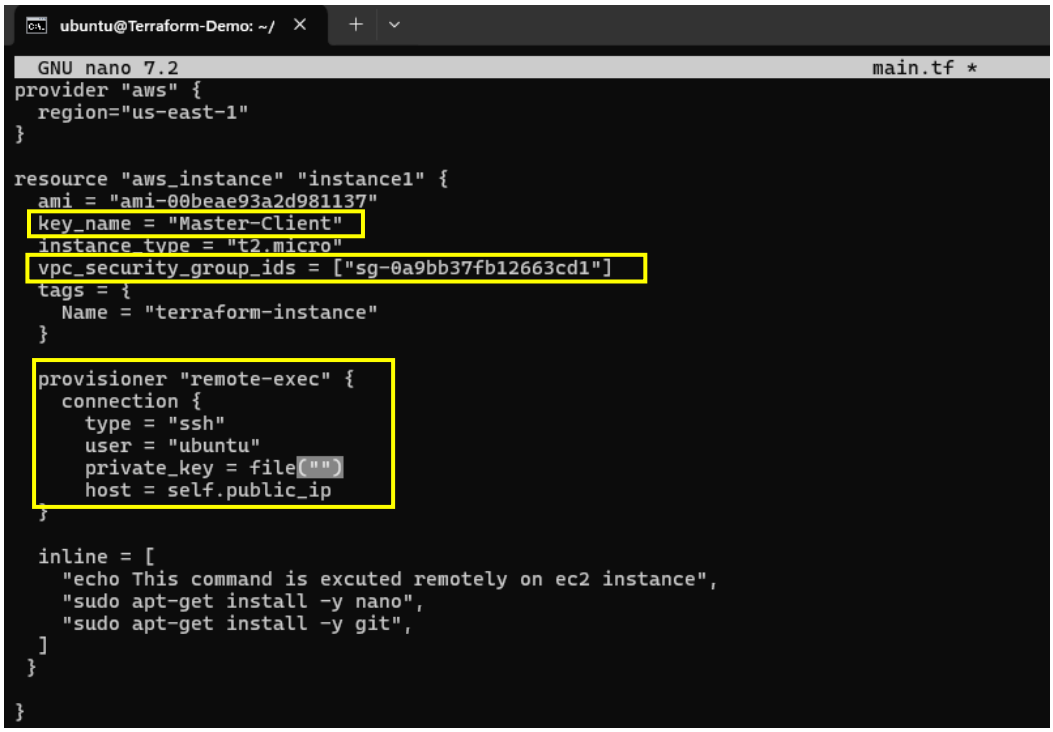


61. Now I am adding SG(Security Group)

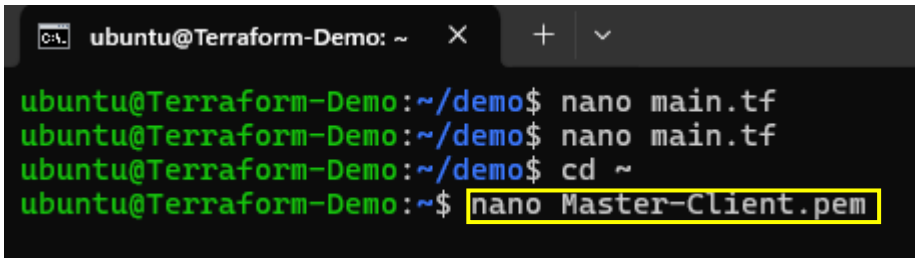


62. Here we can see the key_name and instance_type and vpc_security_group_ids and provisioner remote-exec connecting through type ssh and user ubuntu and private key and host = self.public_ip

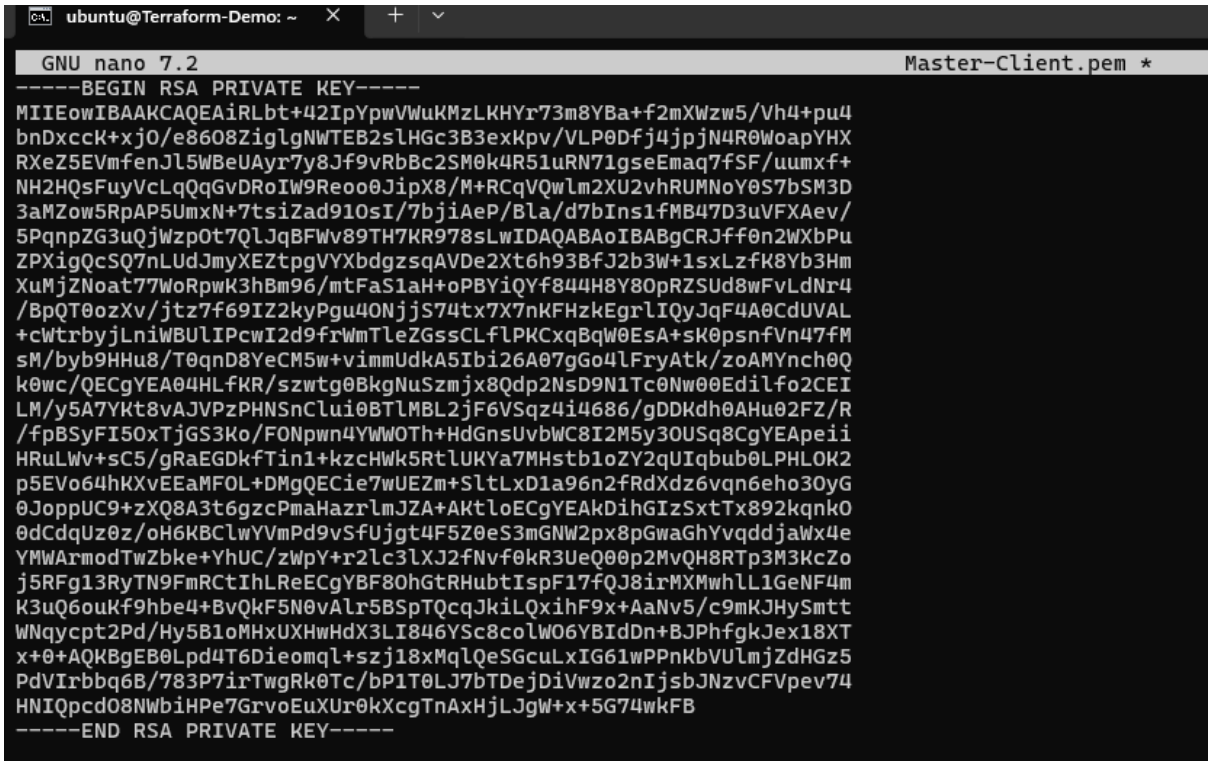
Inline in sided it will be print what we written in echo and it will install nano and git and save it.



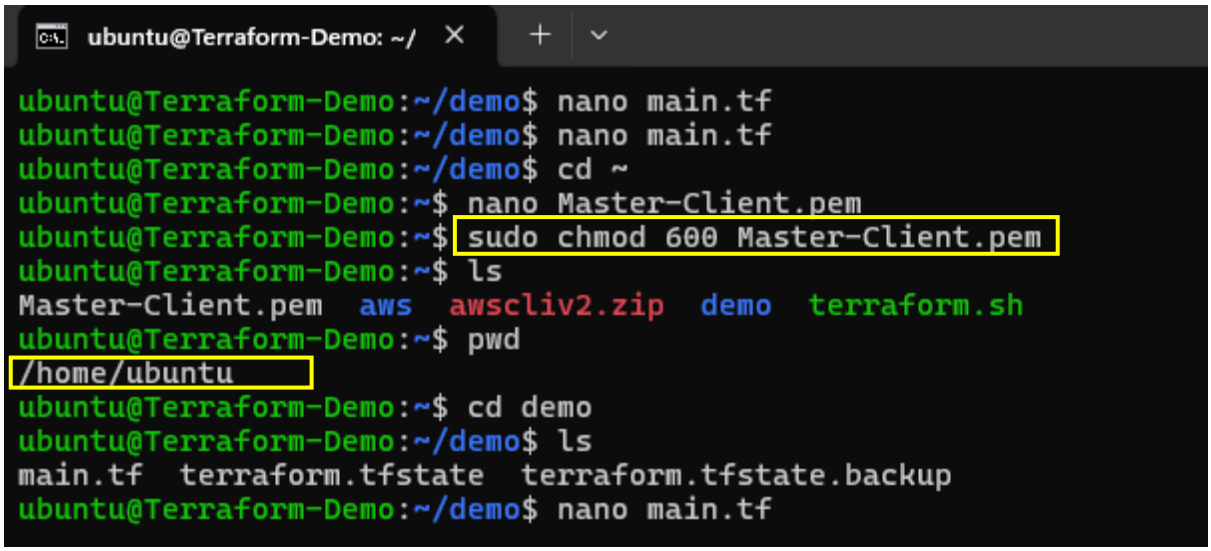
63. Come to home creating private key to connect remote server(exec)



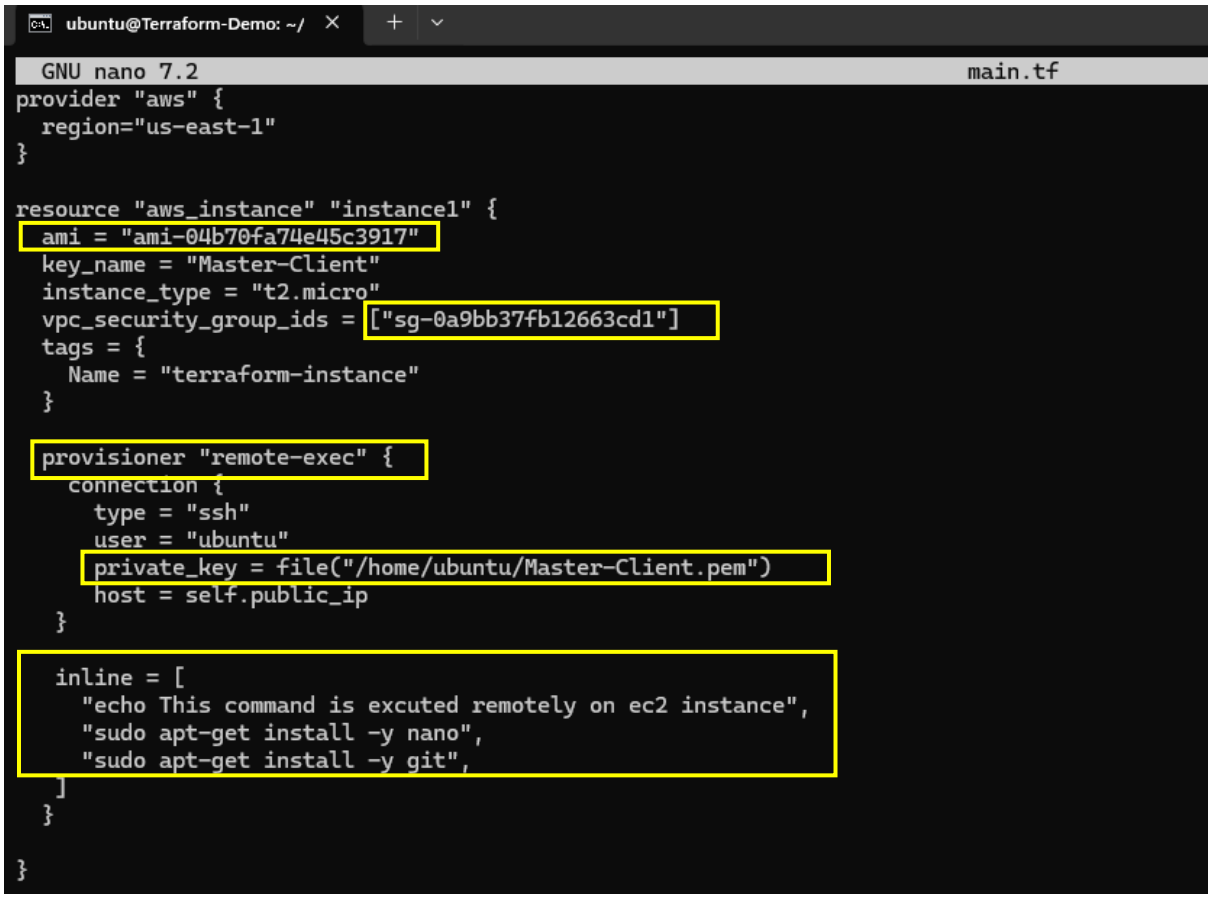
64. Go to local machine key pair location and copy that key pair and paste here



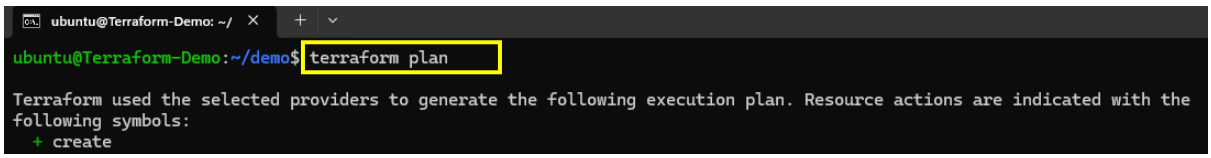
65. Provide permsion to key-pair to read the remote server



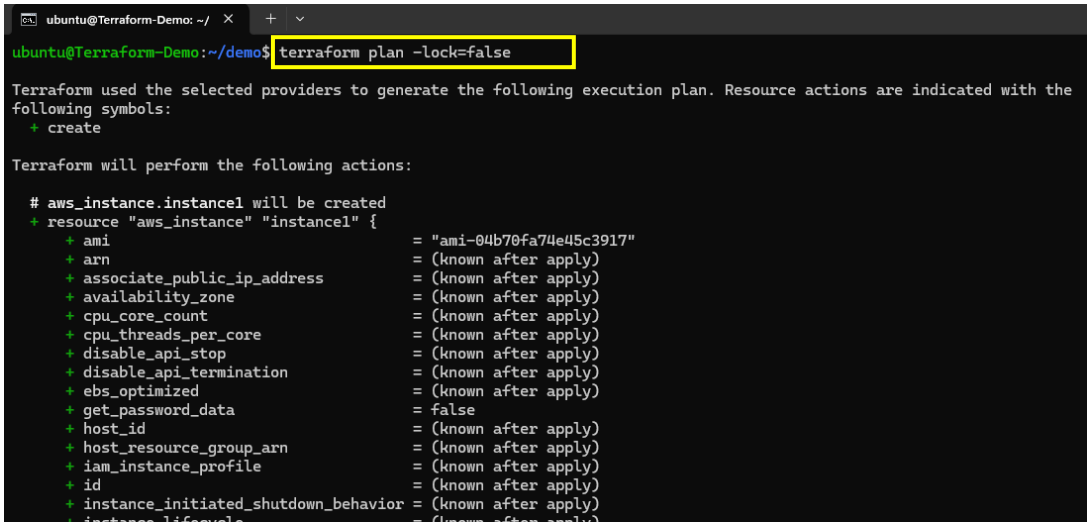
66. Now Came here paste the key-pair path in provisioner “remote-exec”, here changing AMI previously I am using Amazon AMI



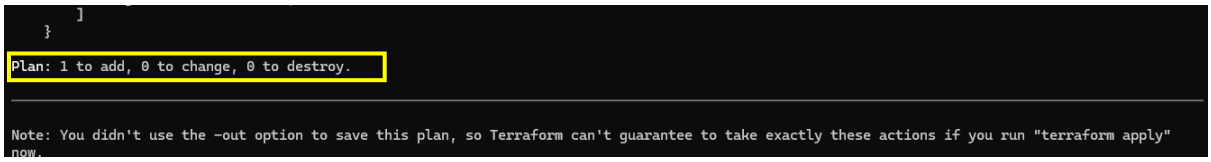
67. terraform plan for me its getting error so I use below the command



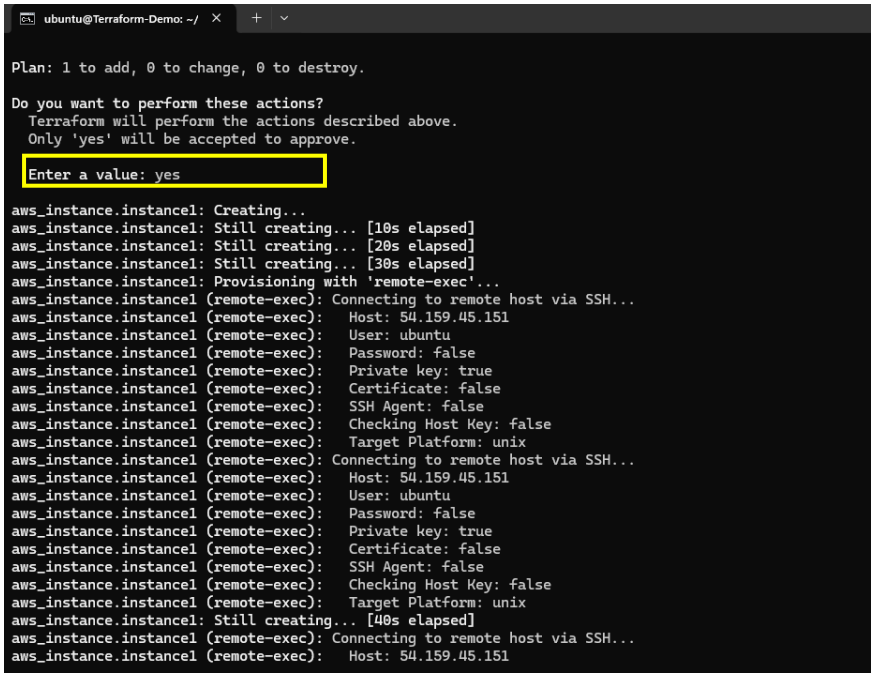
68. terraform plan -lock=false



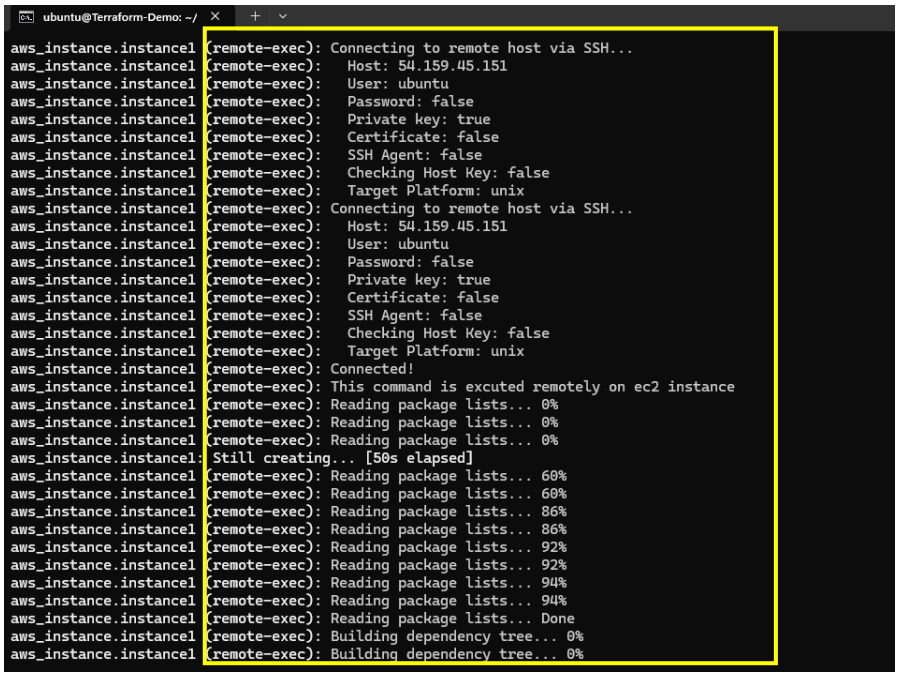
69. Now its plan to 1 to add



70. yes, and it will start create



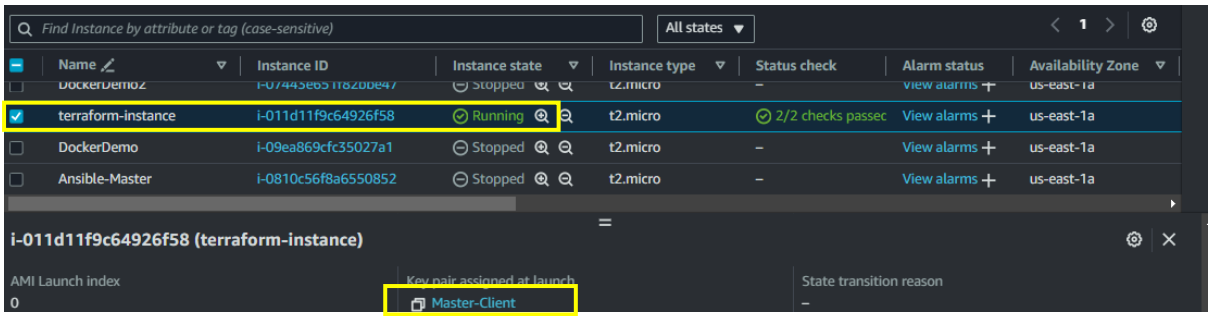
71. Now its connecting to remote host



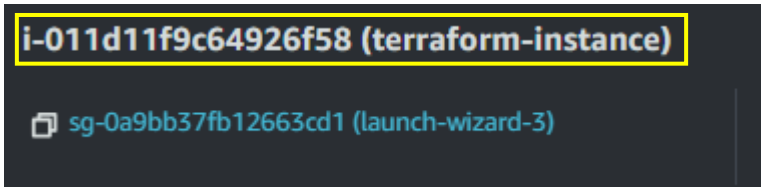
72. Creation completed

```
aws_instance.instance1 (remote-exec): Reading package lists... Done
aws_instance.instance1 (remote-exec): Building dependency tree... 0%
aws_instance.instance1 (remote-exec): Building dependency tree... 0%
aws_instance.instance1 (remote-exec): Building dependency tree... 50%
aws_instance.instance1 (remote-exec): Building dependency tree... 50%
aws_instance.instance1 (remote-exec): Building dependency tree... Done
aws_instance.instance1 (remote-exec): Reading state information... 0%
aws_instance.instance1 (remote-exec): Reading state information... 0%
aws_instance.instance1 (remote-exec): Reading state information... Done
aws_instance.instance1 (remote-exec): git is already the newest version (1:2.43.0-1ubuntu7).
aws_instance.instance1 (remote-exec): git set to manually installed.
aws_instance.instance1 (remote-exec): 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
aws_instance.instance1: Creation complete after 51s [id=i-011d11f9c64926f58]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
ubuntu@Terraform-Demo:~/demo$
```

73. We can the instance with Master-client key-pair which I gave that



74. We can see the security group also attached to terraform instance




```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~$ cd demo
ubuntu@Terraform-Demo:~/demo$ ls -la
total 32
drwxrwxr-x  3 ubuntu ubuntu 4096 Jun  8 13:06 .
drwxr-x--- 10 ubuntu ubuntu 4096 Jun  8 12:30 ..
drwxr-xr-x  3 ubuntu ubuntu 4096 Jun  8 10:29 .terraform
-rw-r--r--  1 ubuntu ubuntu 1377 Jun  8 10:29 .terraform.lock.hcl
-rw-rw-r--  1 ubuntu ubuntu  611 Jun  8 13:06 main.tf
-rw-rw-r--  1 ubuntu ubuntu 4877 Jun  8 13:12 terraform.tfstate
-rw-rw-r--  1 ubuntu ubuntu  181 Jun  8 13:12 terraform.tfstate.backup
ubuntu@Terraform-Demo:~/demo$
```

75. Now destroy the infrastructure

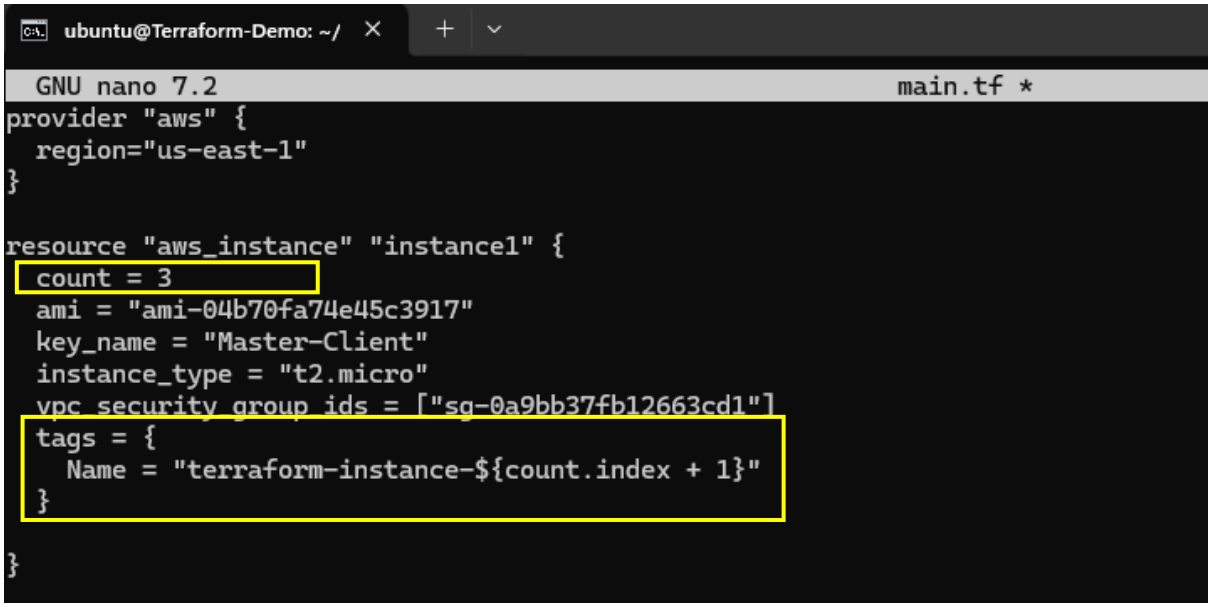
```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/demo$ terraform destroy
```

76. its destroyed

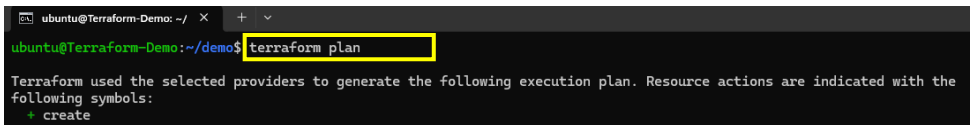
```
Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.instance1: Destroying... [id=i-011d11f9c64926f58]
aws_instance.instance1: Still destroying... [id=i-011d11f9c64926f58, 10s elapsed]
aws_instance.instance1: Still destroying... [id=i-011d11f9c64926f58, 20s elapsed]
aws_instance.instance1: Still destroying... [id=i-011d11f9c64926f58, 30s elapsed]
aws_instance.instance1: Still destroying... [id=i-011d11f9c64926f58, 40s elapsed]
aws_instance.instance1: Destruction complete after 40s

Destroy complete! Resources: 1 destroyed.
```

77. Now I again come to modification

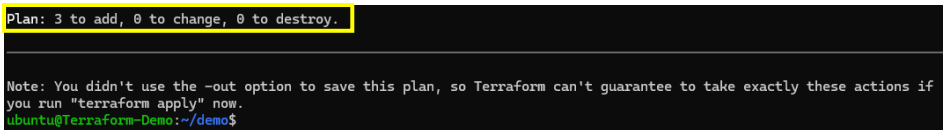
78. Now Creating 3 instances and here key_name = Master-Client which is available on us-east-1 it will took from there and in the tags goint to run $count.index + 1}



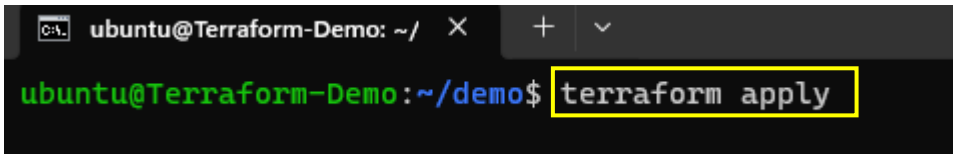
79. terraform plan



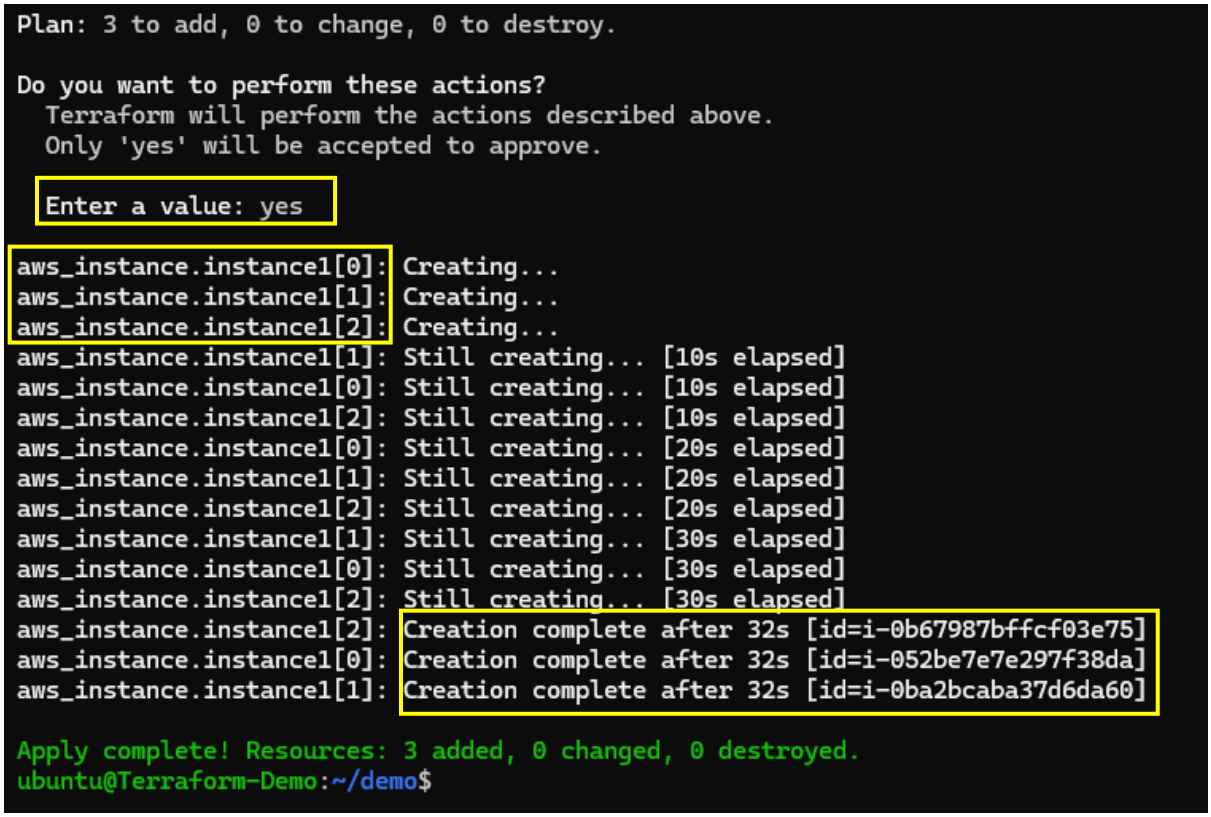
80. it will add 3



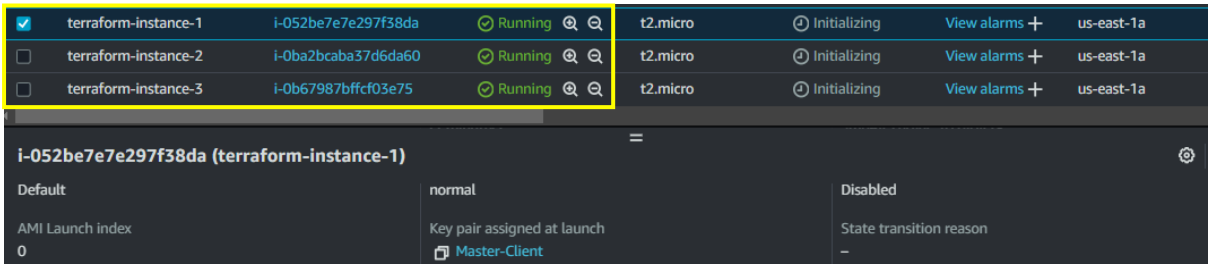
81. Now terraform apply



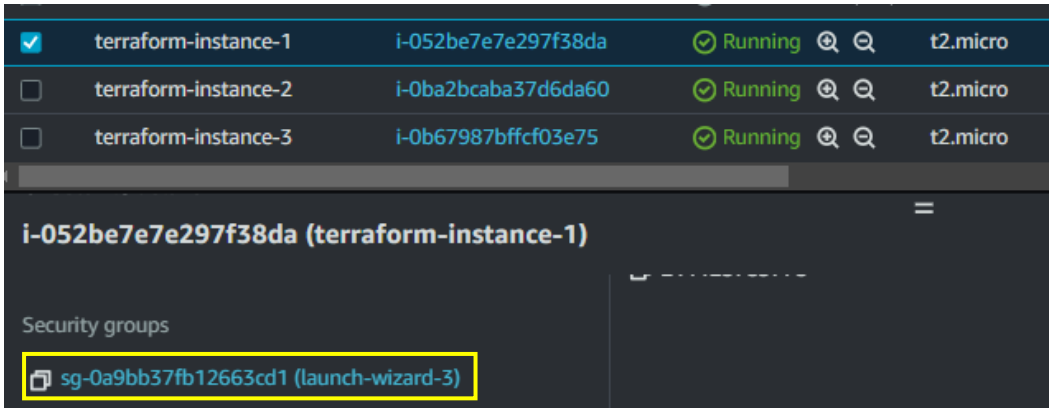
82. yes, and then it will create all three instances



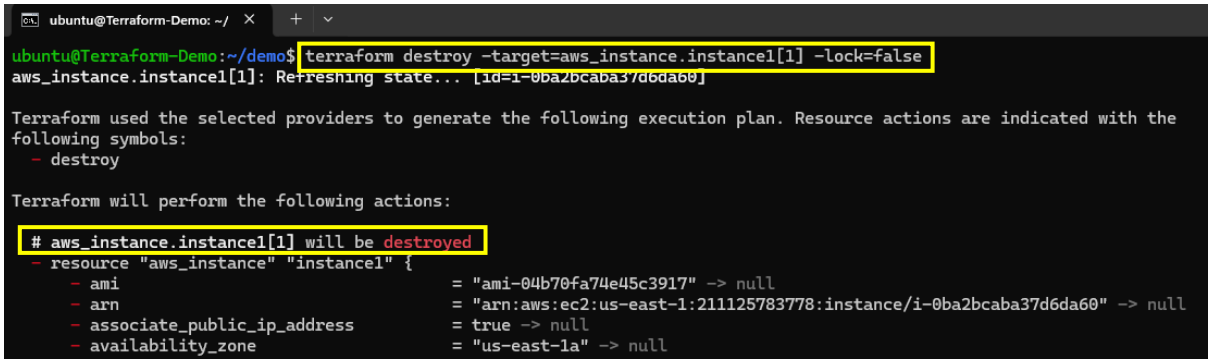
83. running three instances



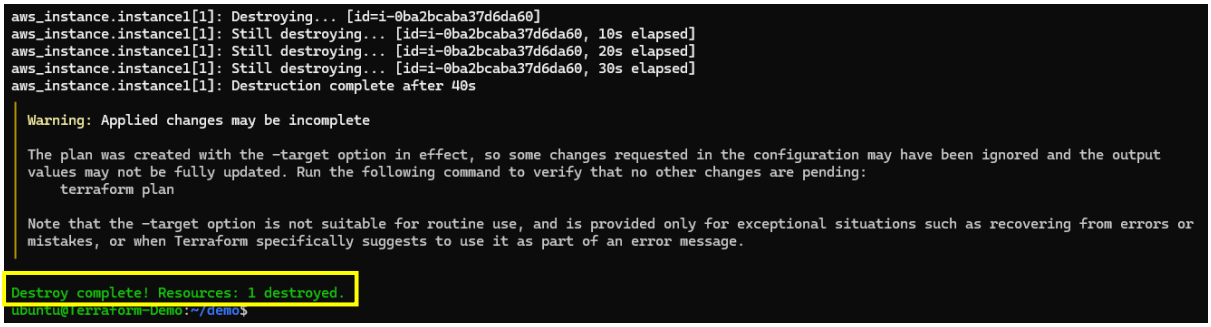
84. If we want to delete Particular instances



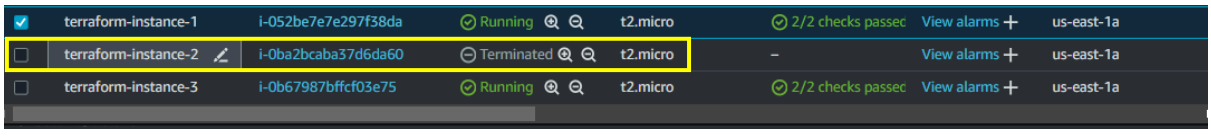
85. follow the command



86. it will destroying the instances



87. its destroyed. That particular instances and destroy all because day 1 completed



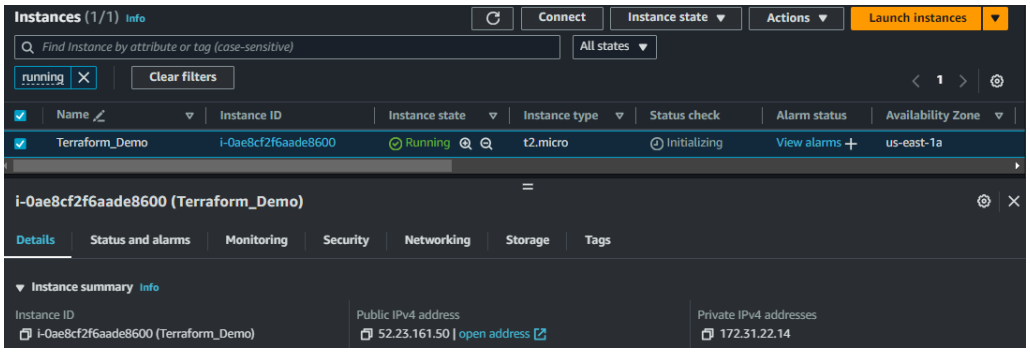
DAY-2

Agenda: Terraform

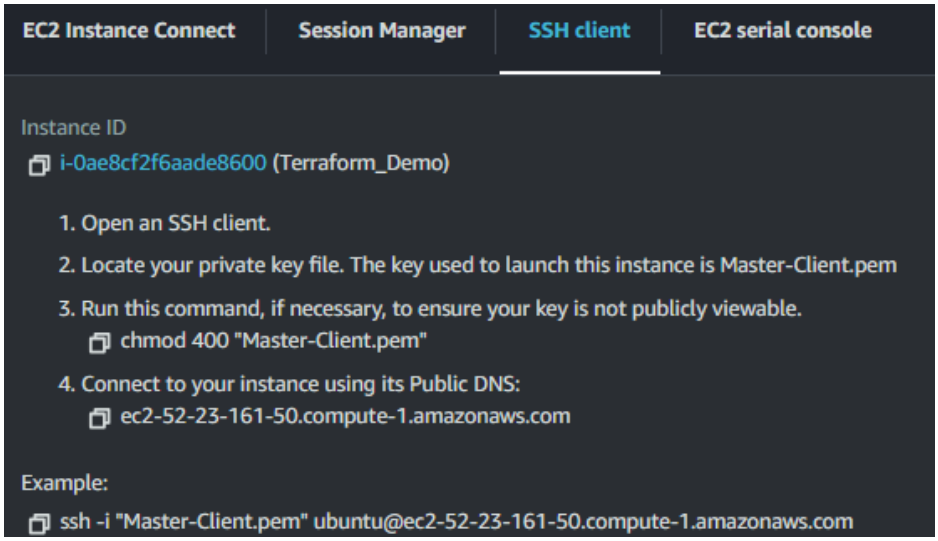
1. What is infrastructure as a code?
2. Infrastructure as a code vs Configuration Management
3. Intro to Terraform
4. Installing Terraform
5. Terraform Operations
6. Terraform code
7. Terraform Demo

immutability (0EWQw84Xngl

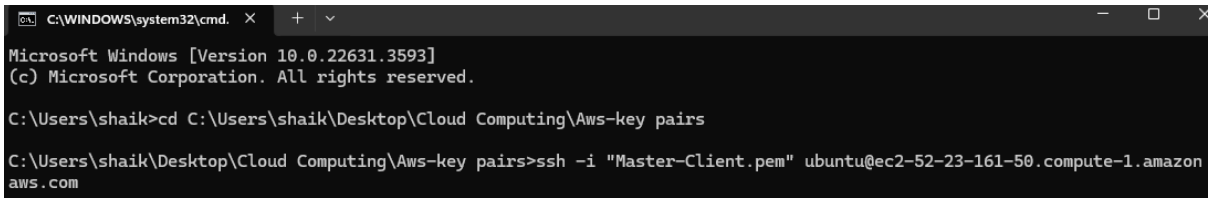
1. Launch Instances



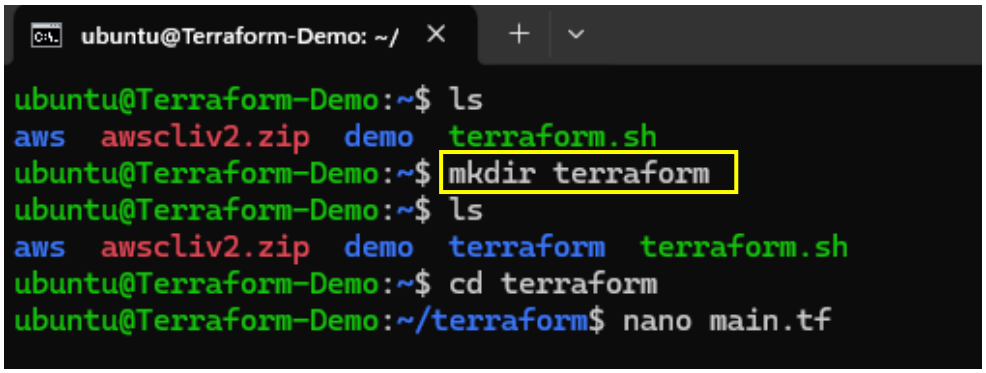
2. Copying the SSH-client



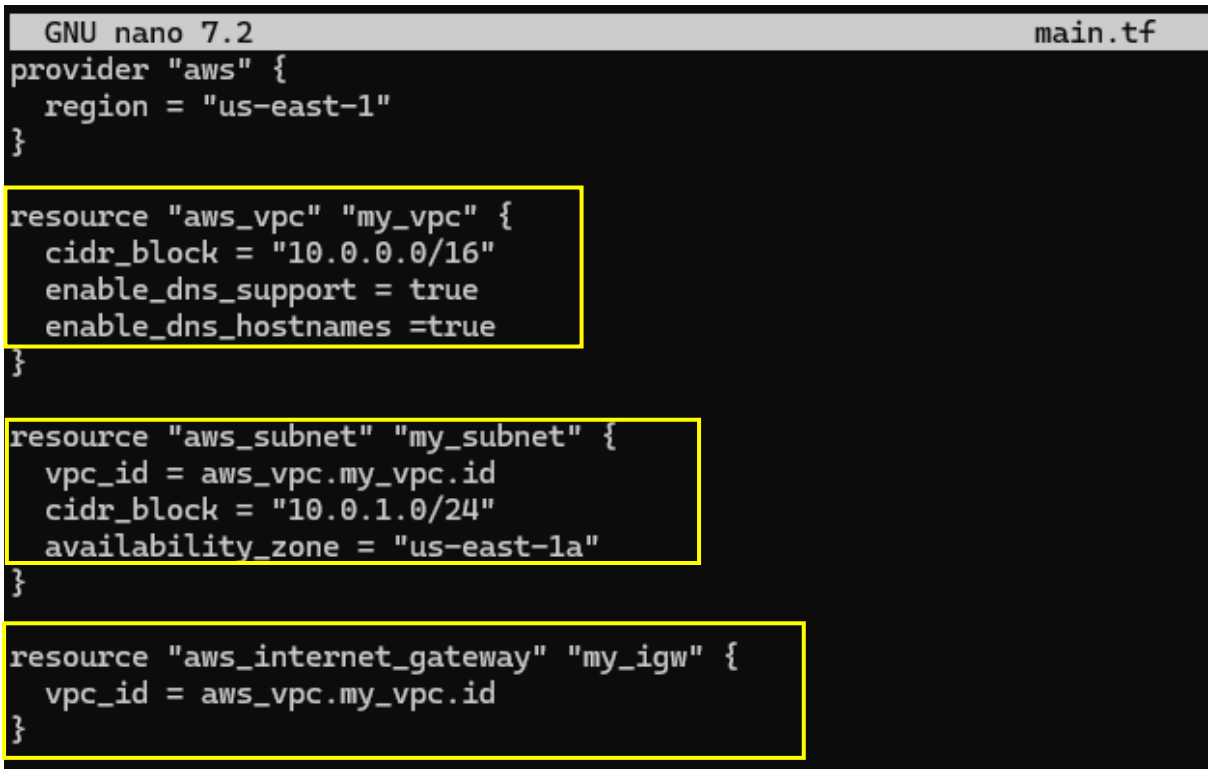
3. Paste and connect the instances



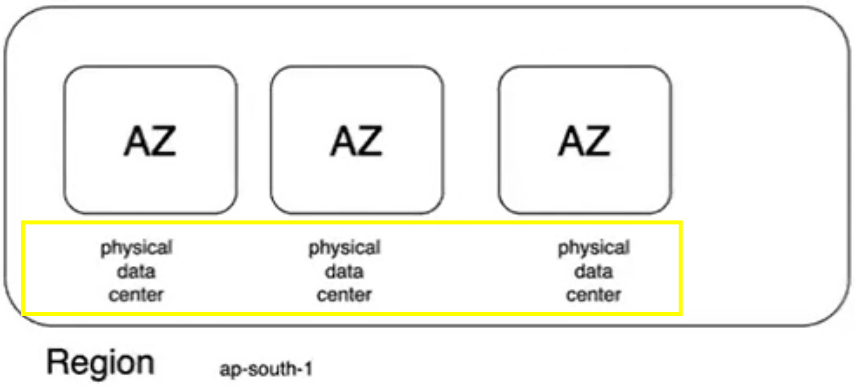
4. Now Creating Another folder Named as terraform

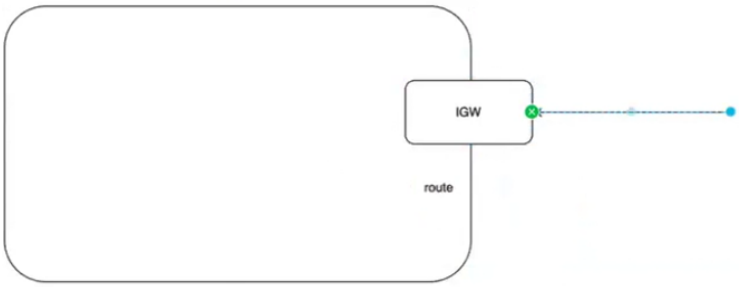


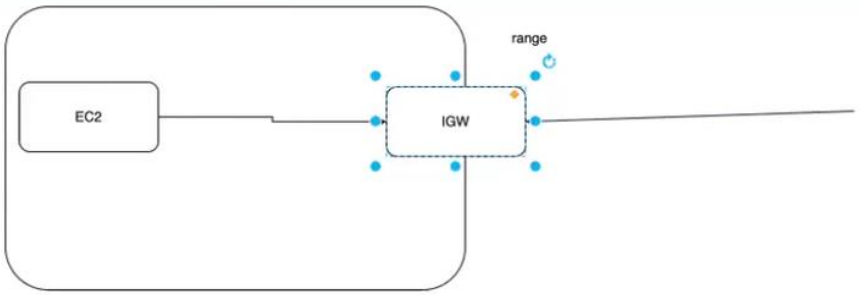
5. Now Creating Vpc and giving cidr_block range 10.0.0.0/16 and adding subnet resources



Creating resources in Availability Zone







6. Continue attaching Route Table and cidr to everyone, and adding subnet resources

```
resource "aws_route_table" "my_route_table" {
  vpc_id = aws_vpc.my_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
  }
}

resource "aws_route_table_association" "my_subnet_association" {
  subnet_id = aws_subnet.my_subnet.id
  route_table_id = aws_route_table.my_route_table.id
}
```

7. Now Allowing security groups are ssh and tcp port and particularly 22 and 80

```
resource "aws_security_group" "my_security_group" {
  vpc_id = aws_vpc.my_vpc.id

  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

8. and Adding instances Resources

```
resource "aws_instance" "my_instance" {
  ami = "ami-04b70fa74e45c3917"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.my_subnet.id
  key_name = "Master-Client"
  security_groups = [aws_security_group.my_security_group.id]

  user_data = file("")
  tags = {
    Name = "MyEC2Instance"
  }
}
```

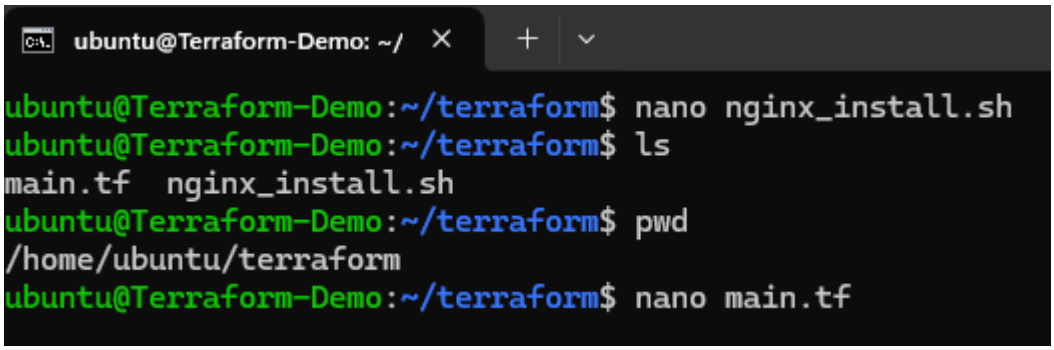
9. Now writing nginx_install.sh for user data

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/terraform$ nano nginx_install.sh
```

10. Write commands and save it.

```
ubuntu@Terraform-Demo: ~/
  GNU nano 7.2                                    nginx_install.sh *
#!/bin/bash
sudo apt update -y
sudo apt install nginx -y
sudo systemctl enable nginx -y
```

11. Now copy the path



12. Come into main.tf file and paste the path

```
resource "aws_instance" "my_instance" {
  ami = "ami-04b70fa74e45c3917"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.my_subnet.id
  key_name = "Master-Client"
  security_groups = [aws_security_group.my_security_group.id]

  user_data = file("/home/ubuntu/terraform/nginx_install.sh")
  tags = {
    Name = "MyEC2Instance"
  }
}
```

Follow the code:

Note: This is edited code in below I got an errors, but this is code is rectified one.

```
provider "aws" {
 region = "us-east-1"
}

resource "aws_vpc" "my_vpc" {
 cidr_block = "10.0.0.0/16"
 enable_dns_support = true
 enable_dns_hostnames = true
}

resource "aws_subnet" "my_subnet" {
```

```
  vpc_id = aws_vpc.my_vpc.id
  cidr_block = "10.0.6.0/24"
  availability_zone = "us-east-1a"
}

resource "aws_internet_gateway" "my_igw" {
  vpc_id = aws_vpc.my_vpc.id
}

resource "aws_route_table" "my_route_table" {
  vpc_id = aws_vpc.my_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.my_igw.id
  }
}

resource "aws_route" "my_route" {
  route_table_id = aws_route_table.my_route_table.id
  gateway_id = aws_internet_gateway.my_igw.id
  destination_cidr_block = "0.0.0.0/0"
}

resource "aws_route_table_association" "my_subnet_association" {
  subnet_id = aws_subnet.my_subnet.id
  route_table_id = aws_route_table.my_route_table.id
}

resource "aws_security_group" "my_security_group" {
  vpc_id = aws_vpc.my_vpc.id
```

```
  ingress {
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "my_instance" {
  ami = "ami-04b70fa74e45c3917"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.my_subnet.id
  key_name = "Master-Client"
  security_groups = [aws_security_group.my_security_group.id]

  user_data = file("/home/ubuntu/terraform/nginx_install.sh")
  tags = {
    Name = "MyEC2Instance"
  }
}
```

13. terraform init

```
ubuntu@Terraform-Demo:~/terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.53.0...
- Installed hashicorp/aws v5.53.0 (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.
```

14. terraform plan and we got an error

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/terraform$ ls -a
.  ..  .terraform  .terraform.lock.hcl  main.tf  nginx_install.sh
ubuntu@Terraform-Demo:~/terraform$ terraform plan

Error: Invalid combination of arguments

  with aws_route.my_route,
  on main.tf line 29, in resource "aws_route" "my_route":
  29: resource "aws_route" "my_route" {

"destination_ipv6_cidr_block": one of `destination_cidr_block,destination_ipv6_cidr_block,destination_prefix_list_id` must be specified


Error: Invalid combination of arguments

  with aws_route.my_route,
  on main.tf line 29, in resource "aws_route" "my_route":
  29: resource "aws_route" "my_route" {

"destination_prefix_list_id": one of `destination_cidr_block,destination_ipv6_cidr_block,destination_prefix_list_id` must be specified


Error: Invalid combination of arguments

  with aws_route.my_route,
  on main.tf line 29, in resource "aws_route" "my_route":
  29: resource "aws_route" "my_route" {

"destination_cidr_block": one of `destination_cidr_block,destination_ipv6_cidr_block,destination_prefix_list_id` must be specified
```

15. Use above the code

```
resource "aws_route" "my_route" {
  route_table_id = aws_route_table.my_route_table.id
  gateway_id = aws_internet_gateway.my_igw.id
  destination_cidr_block = "0.0.0.0/0"
}

resource "aws_route_table_association" "my_subnet_association" {
  subnet_id = aws_subnet.my_subnet.id
  route_table_id = aws_route_table.my_route_table.id
}
```

16. Now terraform init and plan

```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.53.0

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.
ubuntu@Terraform-Demo:~/terraform$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create
```

17. Now its Generating

```
Terraform will perform the following actions:

  # aws_instance.my_instance will be created
  + resource "aws_instance" "my_instance" {
      + ami                                  = "ami-04b70fa74e45c3917"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = "Master-Client"
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
```

```
    + private_ip                          = (known after apply)
    + public_dns                          = (known after apply)
    + public_ip                           = (known after apply)
    + secondary_private_ips               = (known after apply)
    + security_groups                     = (known after apply)
    + source_dest_check                   = true
    + spot_instance_request_id            = (known after apply)
    + subnet_id                           = (known after apply)
    + tags                                = {
        + "Name" = "MyEC2Instance"
      }
    + tags_all                            = {
        + "Name" = "MyEC2Instance"
      }
    + tenancy                             = (known after apply)
    + user_data                           = "e2f83e187ce0ce308f940d40cf54e4e7c59d12b5"
    + user_data_base64                    = (known after apply)
    + user_data_replace_on_change         = false
    + vpc_security_group_ids              = (known after apply)
  }

# aws_internet_gateway.my_igw will be created
+ resource "aws_internet_gateway" "my_igw" {
    + arn      = (known after apply)
    + id       = (known after apply)
    + owner_id = (known after apply)
    + tags_all = (known after apply)
    + vpc_id   = (known after apply)
  }

# aws_route.my_route will be created
+ resource "aws_route" "my_route" {
    + destination_cidr_block = "0.0.0.0/0"
    + gateway_id             = (known after apply)
    + id                     = (known after apply)
```

```
    + id                     = (known after apply)
    + instance_id            = (known after apply)
    + instance_owner_id      = (known after apply)
    + network_interface_id   = (known after apply)
    + origin                 = (known after apply)
    + route_table_id         = (known after apply)
    + state                  = (known after apply)
  }

# aws_route_table.my_route_table will be created
+ resource "aws_route_table" "my_route_table" {
    + arn              = (known after apply)
    + id               = (known after apply)
    + owner_id         = (known after apply)
    + propagating_vgws = (known after apply)
    + route            = [
        + {
            + cidr_block                = "0.0.0.0/0"
              # (12 unchanged attributes hidden)
          },
      ]
    + tags_all         = (known after apply)
    + vpc_id           = (known after apply)
  }

# aws_route_table_association.my_subnet_association will be created
+ resource "aws_route_table_association" "my_subnet_association" {
    + id             = (known after apply)
    + route_table_id = (known after apply)
    + subnet_id      = (known after apply)
  }

# aws_security_group.my_security_group will be created
+ resource "aws_security_group" "my_security_group" {
    + arn                    = (known after apply)
```

```
  + id                     = (known after apply)
  + ingress                = [
      + {
          + cidr_blocks      = [
              + "0.0.0.0/0",
            ]
          + from_port        = 22
          + ipv6_cidr_blocks = []
          + prefix_list_ids  = []
          + protocol         = "tcp"
          + security_groups  = []
          + self             = false
          + to_port          = 22
            # (1 unchanged attribute hidden)
        },
      + {
          + cidr_blocks      = [
              + "0.0.0.0/0",
            ]
          + from_port        = 80
          + ipv6_cidr_blocks = []
          + prefix_list_ids  = []
          + protocol         = "tcp"
          + security_groups  = []
          + self             = false
          + to_port          = 80
            # (1 unchanged attribute hidden)
        },
    ]
  + name                   = (known after apply)
  + name_prefix            = (known after apply)
  + owner_id               = (known after apply)
  + revoke_rules_on_delete = false
  + tags_all               = (known after apply)
  + vpc_id                 = (known after apply)
```

```
      + vpc_id                = (known after apply)
    }

  # aws_subnet.my_subnet will be created
  + resource "aws_subnet" "my_subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1a"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "10.0.1.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags_all                                       = (known after apply)
      + vpc_id                                         = (known after apply)
    }

  # aws_vpc.my_vpc will be created
  + resource "aws_vpc" "my_vpc" {
      + arn                                = (known after apply)
      + cidr_block                         = "10.0.0.0/16"
      + default_network_acl_id             = (known after apply)
      + default_route_table_id             = (known after apply)
      + default_security_group_id          = (known after apply)
      + dhcp_options_id                    = (known after apply)
      + enable_dns_hostnames               = true
      + 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_all                           = (known after apply)
    }

Plan: 8 to add, 0 to change, 0 to destroy.
```

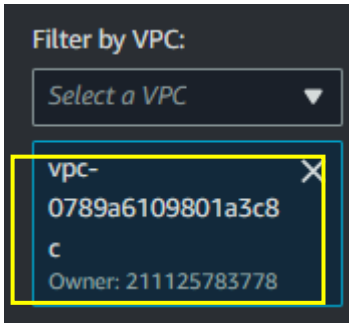
17. Again While doing terraform apply got an error so below the edited one. And above code is edited one

```
resource "aws_route_table" "my_route_table" {
  vpc_id = aws_vpc.my_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.my_igw.id
  }
}

resource "aws_route" "my_route" {
  route_table_id = aws_route_table.my_route_table.id
  gateway_id = aws_internet_gateway.my_igw.id
  destination_cidr_block = "0.0.0.0/0"
}
```

18. Now Its Creating all the 7 resources

```
aws_vpc.my_vpc: Creating...
aws_vpc.my_vpc: Still creating... [10s elapsed]
aws_vpc.my_vpc: Creation complete after 12s [id=vpc-0789a6109801a3c8c]
aws_internet_gateway.my_igw: Creating...
aws_route_table.my_route_table: Creating...
aws_security_group.my_security_group: Creating...
aws_subnet.my_subnet: Creating...
aws_internet_gateway.my_igw: Creation complete after 0s [id=igw-0f2a2f84e64cc3487]
aws_subnet.my_subnet: Creation complete after 1s [id=subnet-059193af925bce55a]
aws_security_group.my_security_group: Creation complete after 2s [id=sg-01067e31fbe6ab135]
aws_instance.my_instance: Creating...
aws_instance.my_instance: Still creating... [10s elapsed]
aws_instance.my_instance: Still creating... [20s elapsed]
aws_instance.my_instance: Still creating... [30s elapsed]
```

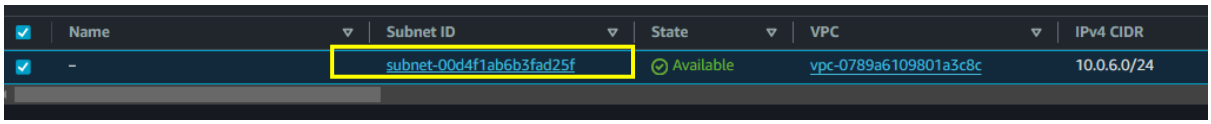
19. VPC ID



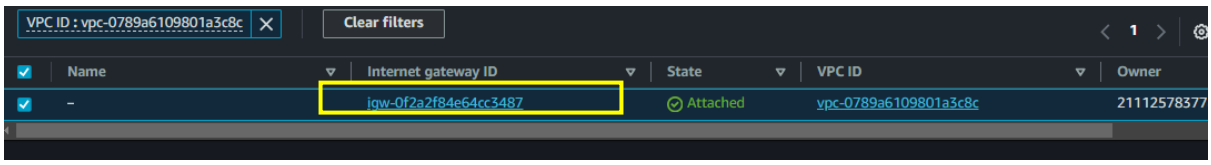
19. This are the generated ids

```
ubuntu@Terraform-Demo:~/terraform$ terraform apply
aws_vpc.my_vpc: Refreshing state... [id=vpc-0789a6109801a3c8c]
aws_subnet.my_subnet: Refreshing state... [id=subnet-00d4f1ab6b3fad25f]
aws_internet_gateway.my_igw: Refreshing state... [id=igw-0f2a2f84e64cc3487]
aws_security_group.my_security_group: Refreshing state... [id=sg-01067e31fbe6ab135]
aws_route_table.my_route_table: Refreshing state... [id=rtb-0d1fec63bb68b48fb]
aws_instance.my_instance: Refreshing state... [id=i-096b7d0174367166a]
aws_route_table_association.my_subnet_association: Refreshing state... [id=rtbassoc-0c8f6041d1a61fb15]
```

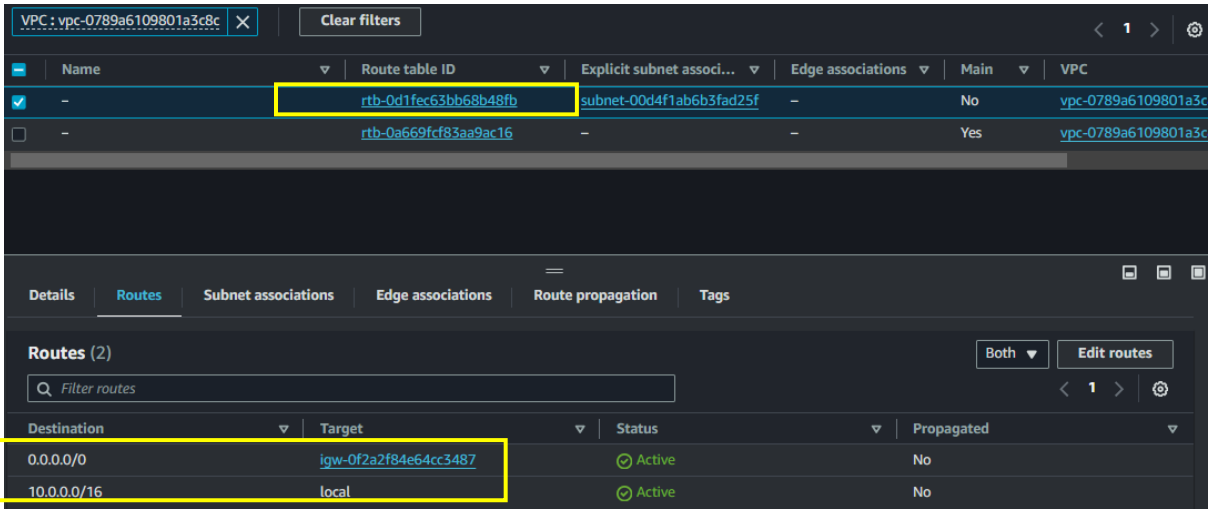
20. this is subnet id



21. And this is IGW-ID



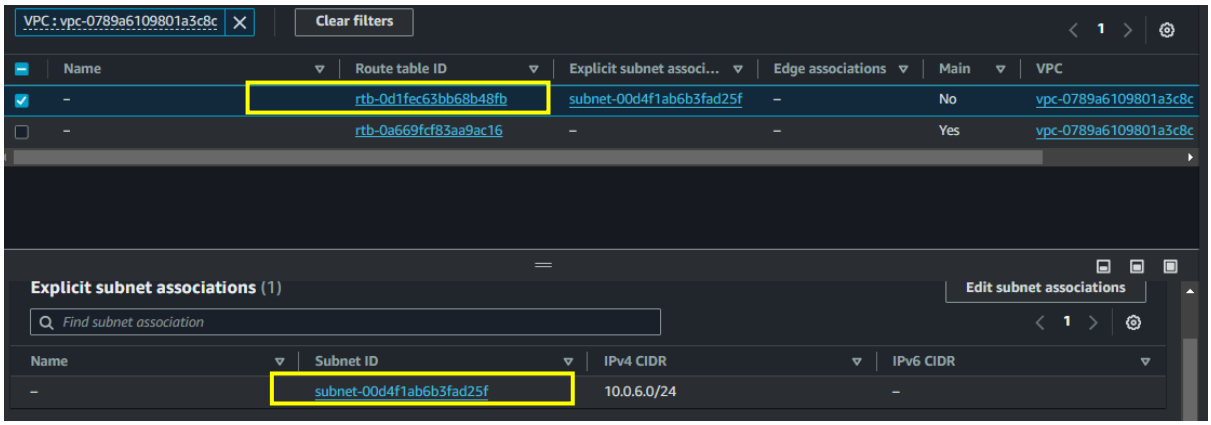
22. And this is Route Table ID and we can see the CIDR block and IGW attached



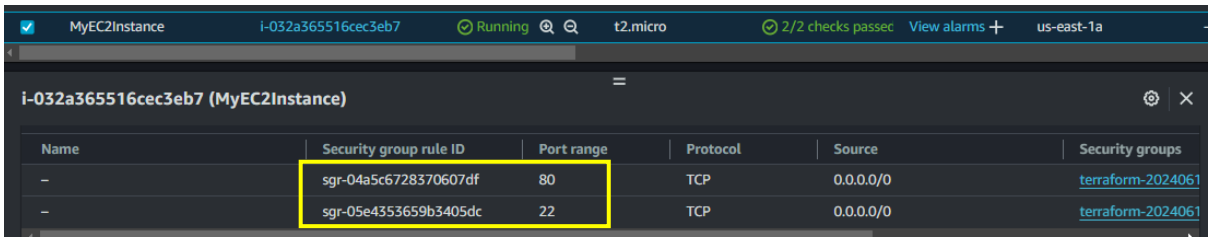
23. See the what we given cidr for vpc and cidr for subnet association, Follow Above the code

```
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support = true
  enable_dns_hostnames = true
}

resource "aws_subnet" "my_subnet" {
  vpc_id = aws_vpc.my_vpc.id
  cidr_block = "10.0.6.0/24"
  availability_zone = "us-east-1a"
}
```

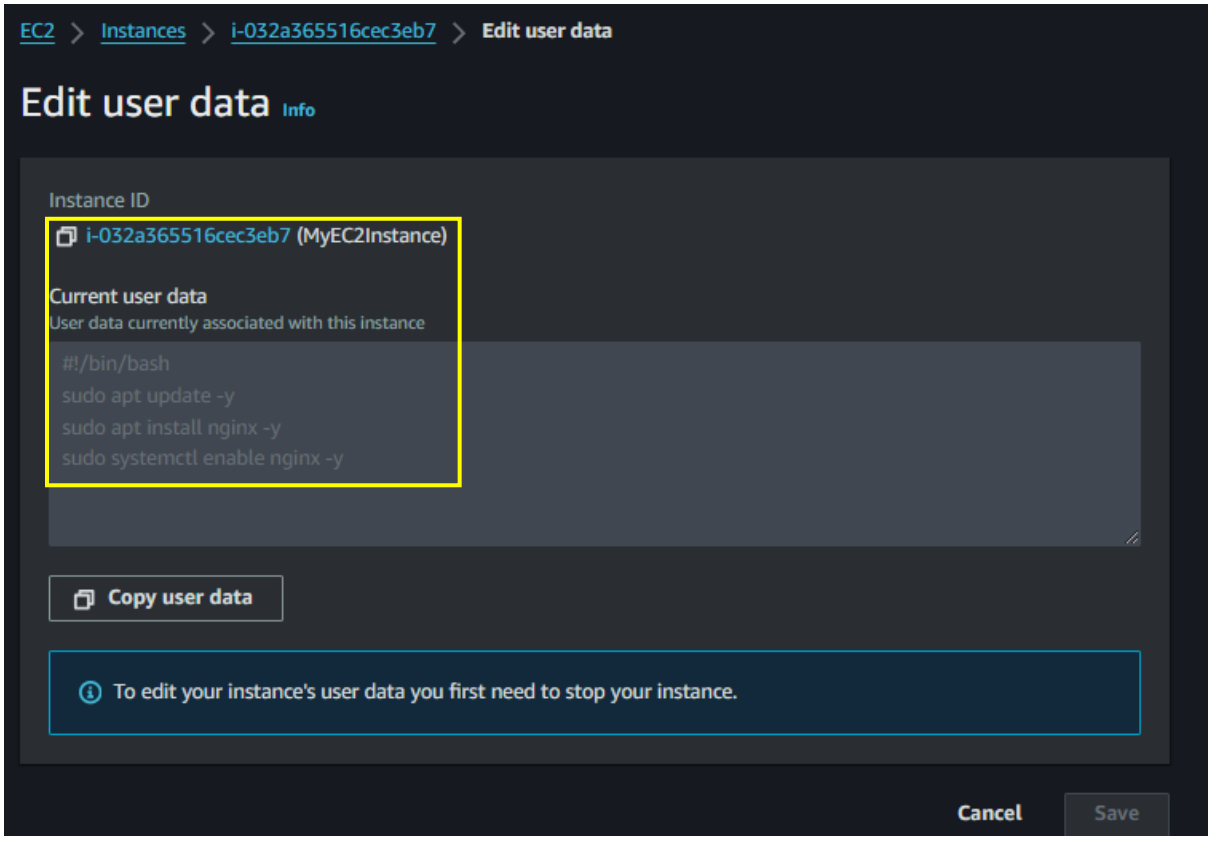
24. Here the subnet association cidr



25. we can see the given port and cidr source was everyone



26. user data



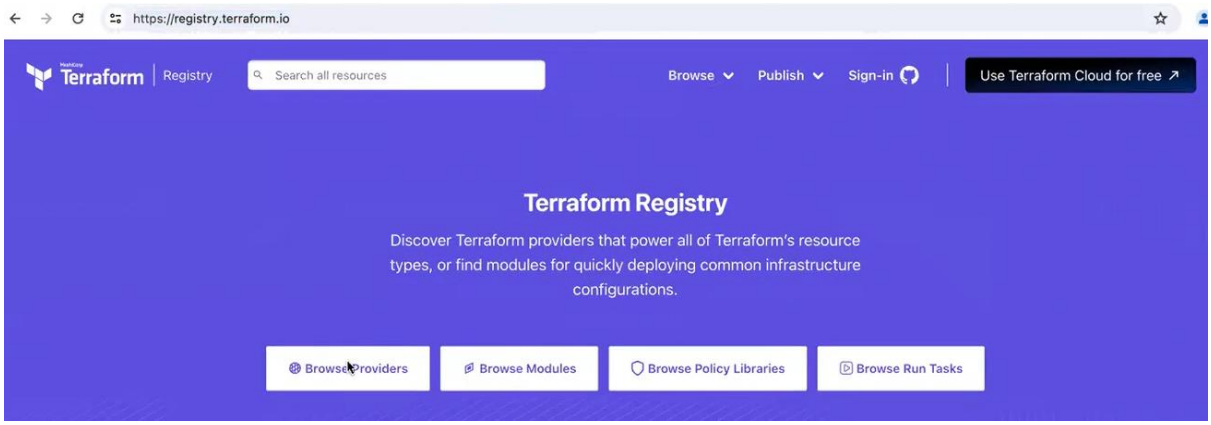
27. Now Destroying the infrastructure

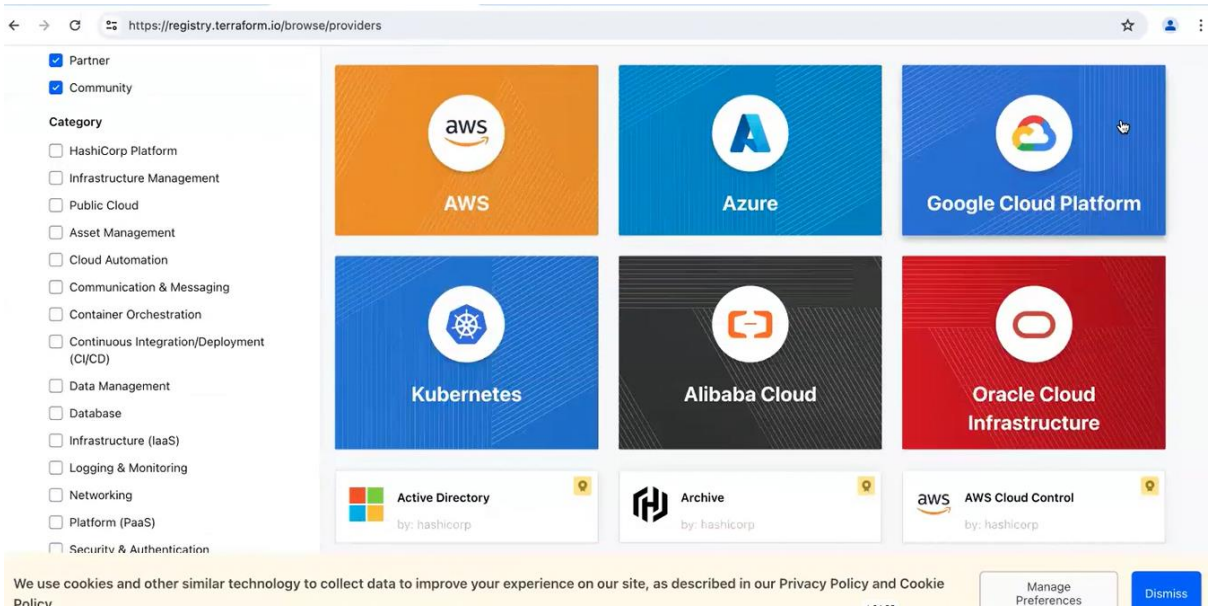
```
ubuntu@Terraform-Demo: ~/
ubuntu@Terraform-Demo:~/terraform$ terraform destroy
aws_vpc.my_vpc: Refreshing state... [id=vpc-0789a6109801a3c8c]
aws_security_group.my_security_group: Refreshing state... [id=sg-01067e31fbe6ab135]
aws_internet_gateway.my_igw: Refreshing state... [id=igw-0f2a2f84e64cc3487]
aws_subnet.my_subnet: Refreshing state... [id=subnet-00d4f1ab6b3fad25f]
aws_route_table.my_route_table: Refreshing state... [id=rtb-0d1fec63bb68b48fb]
aws_route_table_association.my_subnet_association: Refreshing state... [id=rtbassoc-0c8f6041d1a61fb15]
aws_instance.my_instance: Refreshing state... [id=i-032a365516cec3eb7]
```

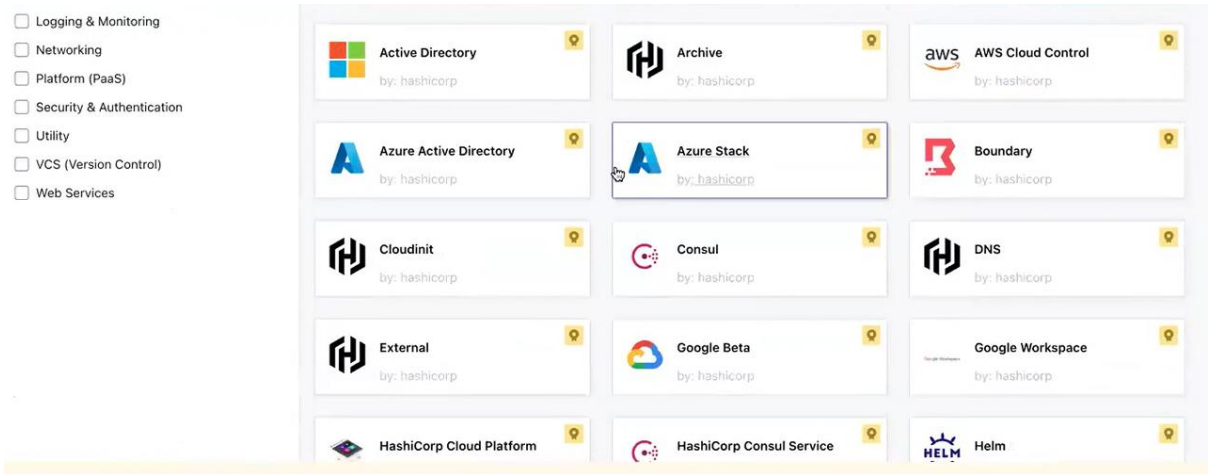
28. All Seven going to be destroying

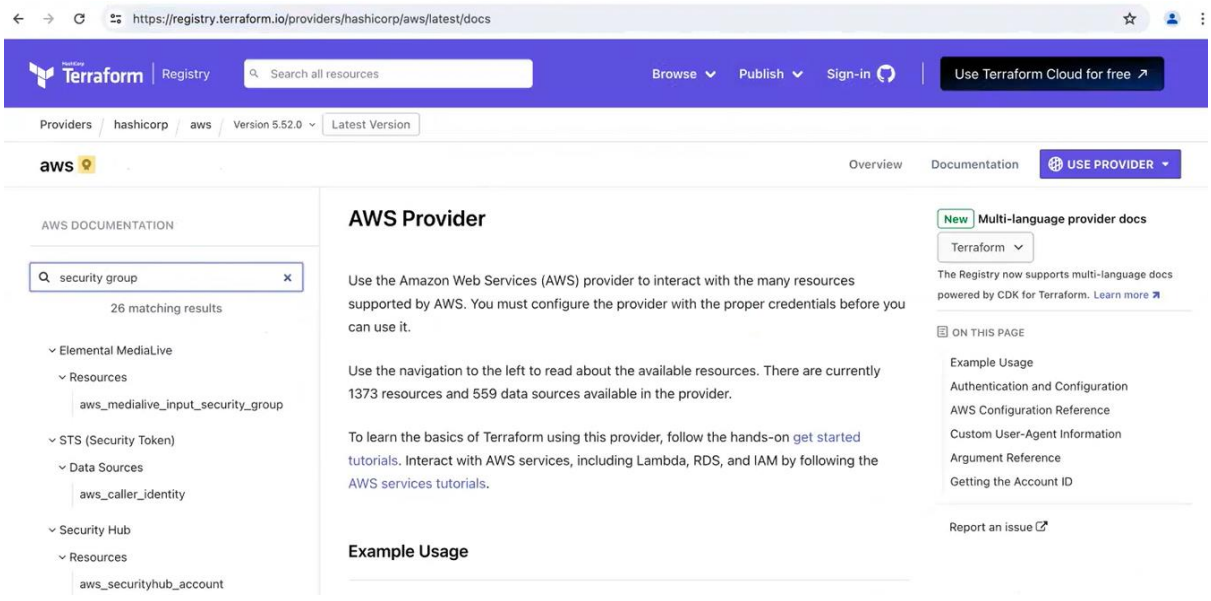
```
Plan: 0 to add, 0 to change, 7 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_route_table_association.my_subnet_association: Destroying... [id=rtbassoc-0c8f6041d1a61fb15]
aws_instance.my_instance: Destroying... [id=i-032a365516cec3eb7]
aws_route_table_association.my_subnet_association: Destruction complete after 1s
aws_route_table.my_route_table: Destroying... [id=rtb-0d1fec63bb68b48fb]
aws_route_table.my_route_table: Destruction complete after 0s
aws_internet_gateway.my_igw: Destroying... [id=igw-0f2a2f84e64cc3487]
aws_internet_gateway.my_igw: Destruction complete after 0s
aws_instance.my_instance: Still destroying... [id=i-032a365516cec3eb7, 10s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a365516cec3eb7, 20s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a365516cec3eb7, 30s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a365516cec3eb7, 40s elapsed]
aws_instance.my_instance: Still destroying... [id=i-032a365516cec3eb7, 50s elapsed]
aws_instance.my_instance: Destruction complete after 51s
aws_security_group.my_security_group: Destroying... [id=sg-01067e31fbe6ab135]
aws_subnet.my_subnet: Destroying... [id=subnet-00d4f1ab6b3fad25f]
aws_subnet.my_subnet: Destruction complete after 0s
aws_security_group.my_security_group: Destruction complete after 0s
aws_vpc.my_vpc: Destroying... [id=vpc-0789a6109801a3c8c]
aws_vpc.my_vpc: Destruction complete after 1s

Destroy complete! Resources: 7 destroyed.
ubuntu@Terraform-Demo:~/terraform$
```

Follow the below the links we can go to explore it




https://registry.terraform.io/browse/providers
Partner
Community
Category
HashiCorp Platform
Infrastructure Management
Public Cloud
Asset Management
Cloud Automation
Communication & Messaging
Container Orchestration
Continuous Integration/Deployment (CI/CD)
Data Management
Database
Infrastructure (IaaS)
Logging & Monitoring
Networking
Platform (PaaS)
Security & Authentication
AWS
Azure
Google Cloud Platform
Kubernetes
Alibaba Cloud
Oracle Cloud Infrastructure
Active Directory
by: hashicorp
Archive
by: hashicorp
AWS Cloud Control
by: hashicorp
We use cookies and other similar technology to collect data to improve your experience on our site, as described in our Privacy Policy and Cookie Policy.
Manage Preferences
Dismiss


Logging & Monitoring
Networking
Platform (PaaS)
Security & Authentication
Utility
VCS (Version Control)
Web Services
Active Directory
by: hashicorp
Archive
by: hashicorp
AWS Cloud Control
by: hashicorp
Azure Active Directory
by: hashicorp
Azure Stack
by: hashicorp
Boundary
by: hashicorp
Cloudinit
by: hashicorp
Consul
by: hashicorp
DNS
by: hashicorp
External
by: hashicorp
Google Beta
by: hashicorp
Google Workspace
by: hashicorp
HashiCorp Cloud Platform
HashiCorp Consul Service
Helm


https://registry.terraform.io/providers/hashicorp/aws/latest/docs
Terraform | Registry
Search all resources
Browse
Publish
Sign-in
Use Terraform Cloud for free
Providers / hashicorp / aws / Version 5.52.0 / Latest Version
aws
Overview
Documentation
USE PROVIDER
AWS DOCUMENTATION
security group
26 matching results
Elemental MediaLive
Resources
aws_medialive_input_security_group
STS (Security Token)
Data Sources
aws_caller_identity
Security Hub
Resources
aws_securityhub_account
AWS Provider
Use the Amazon Web Services (AWS) provider to interact with the many resources supported by AWS. You must configure the provider with the proper credentials before you can use it.
Use the navigation to the left to read about the available resources. There are currently 1373 resources and 559 data sources available in the provider.
To learn the basics of Terraform using this provider, follow the hands-on get started tutorials. Interact with AWS services, including Lambda, RDS, and IAM by following the AWS services tutorials.
Example Usage
New Multi-language provider docs
Terraform
The Registry now supports multi-language docs powered by CDK for Terraform. Learn more
ON THIS PAGE
Example Usage
Authentication and Configuration
AWS Configuration Reference
Custom User-Agent Information
Argument Reference
Getting the Account ID
Report an issue

Links:

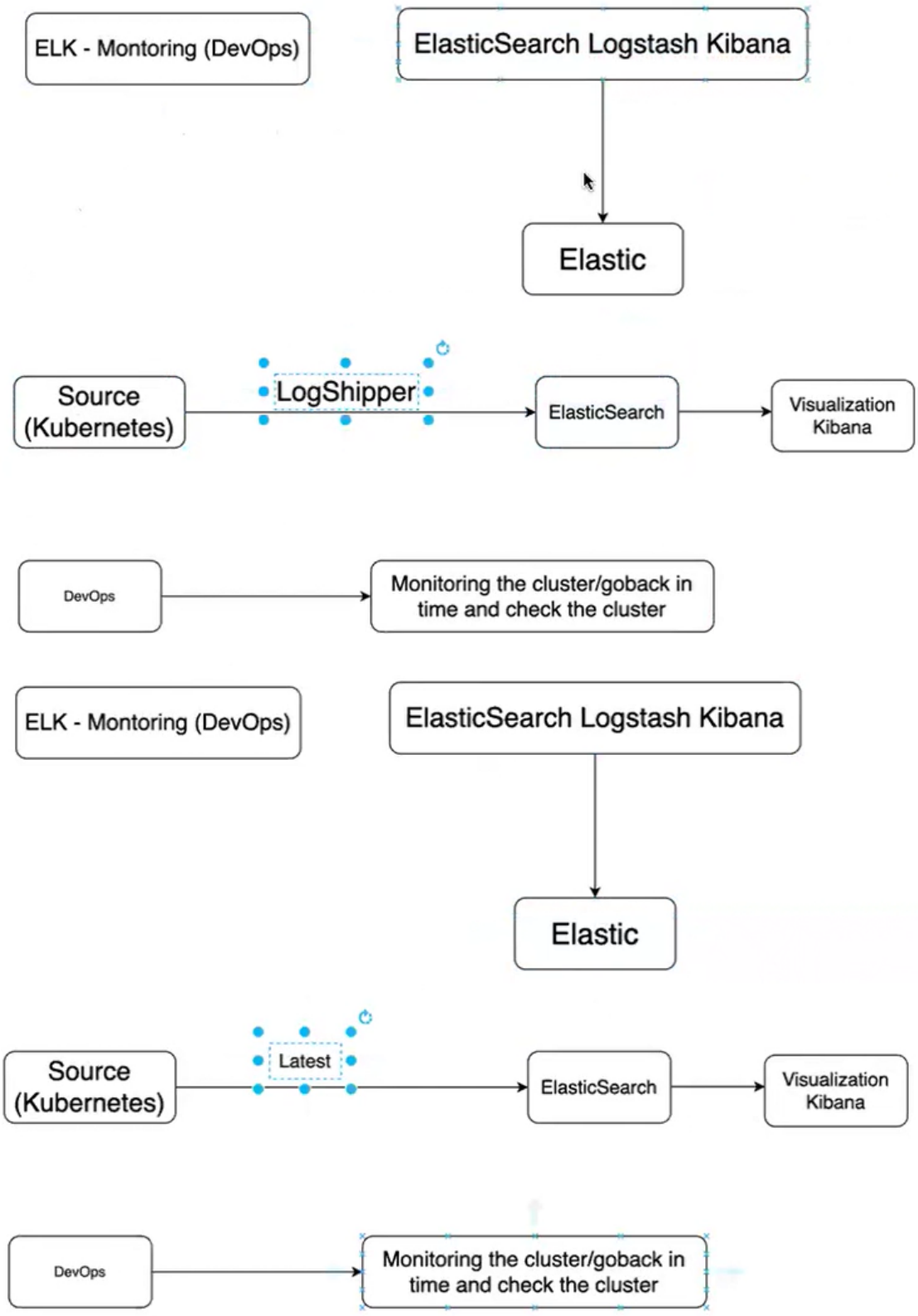
https://registry.terraform.io/

https://registry.terraform.io/browse/providers

Aws eg:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/service_discovery_instance

Go and Explore it.

ELK(ElasticSearch Logstash Kibana)



ELK in DevOps is like having a search engine for your system's logs. "ELK" stands for Elasticsearch, Logstash, and Kibana.

- Elasticsearch is like the big database where all your logs are stored.

- Logstash helps you collect, parse, and send your logs to Elasticsearch.

- Kibana is a visualization tool that lets you search, analyze, and visualize your logs in a user-friendly way.

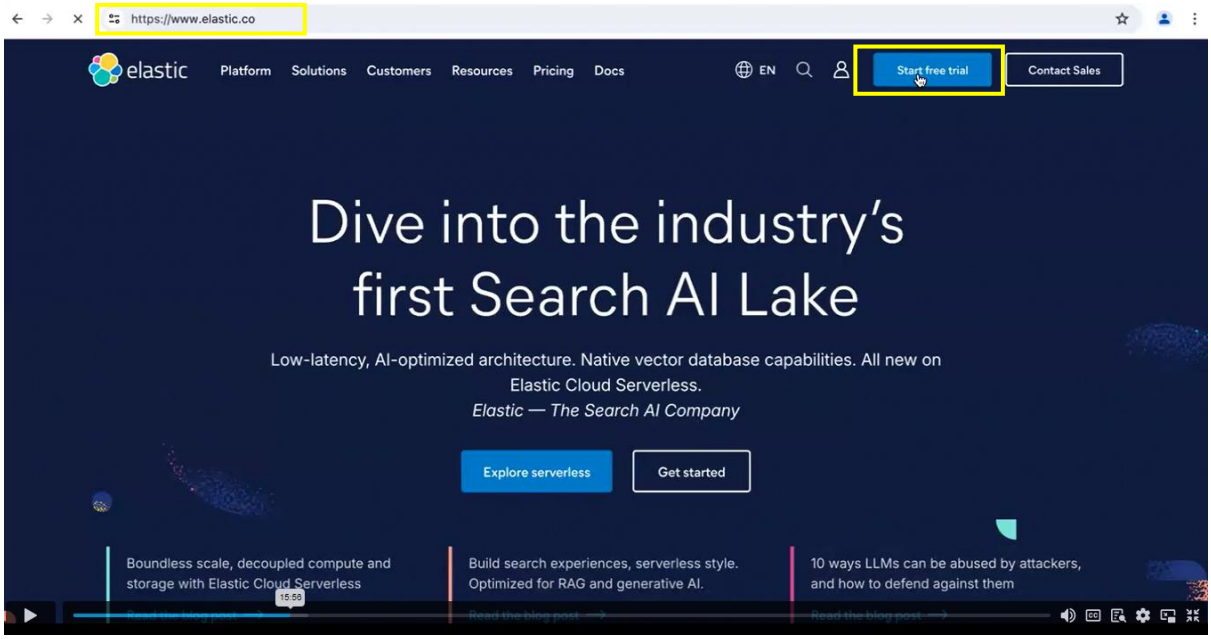
Together, they help DevOps teams manage and troubleshoot their systems by providing a centralized place to store, search, and analyze logs, helping them detect issues, monitor performance, and improve their systems' reliability.

Links:

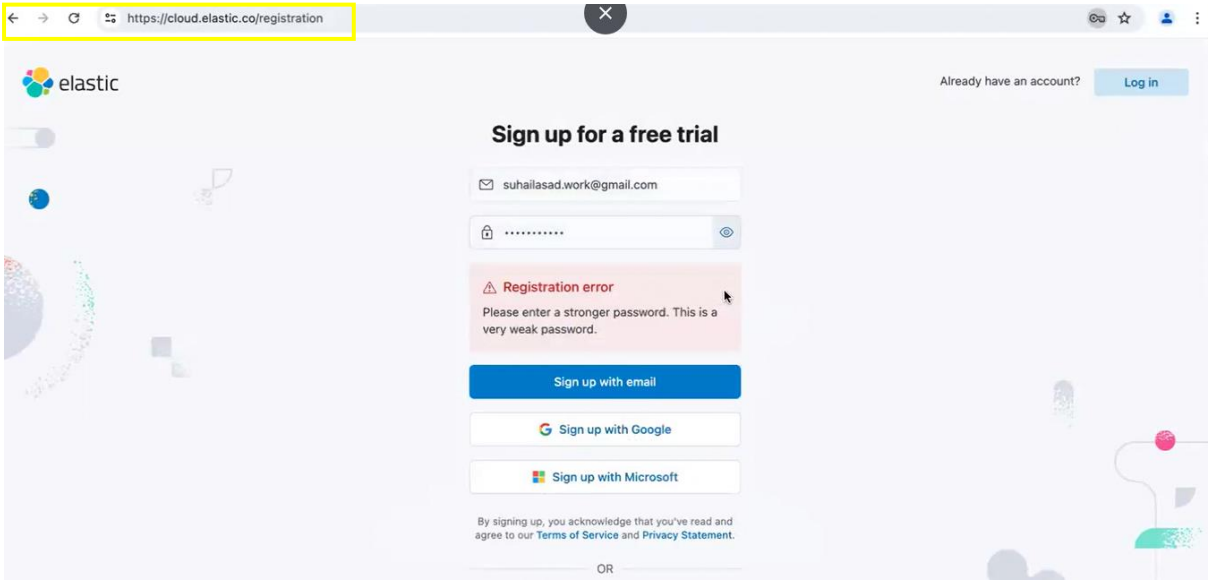
https://www.elastic.co/

is indeed the official website of Elastic, the company behind Elasticsearch, as well as the other components of the ELK stack (Logstash and Kibana). On this website, you can find information about their products, solutions, documentation, and resources related to search, logging, and analytics.

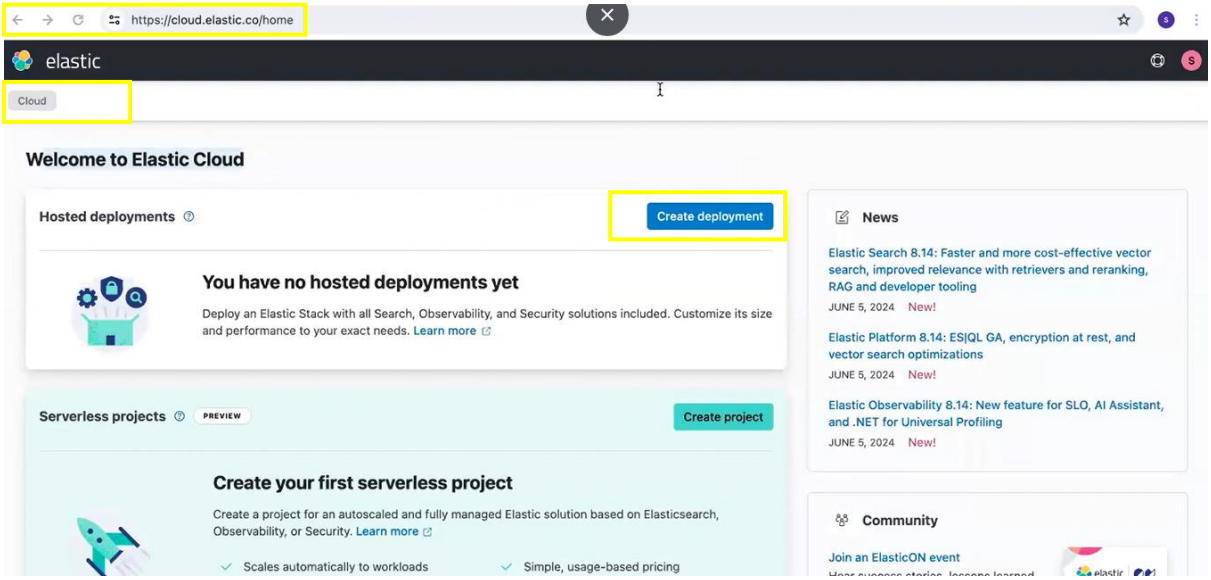
Click start free trial, if u want to practice.



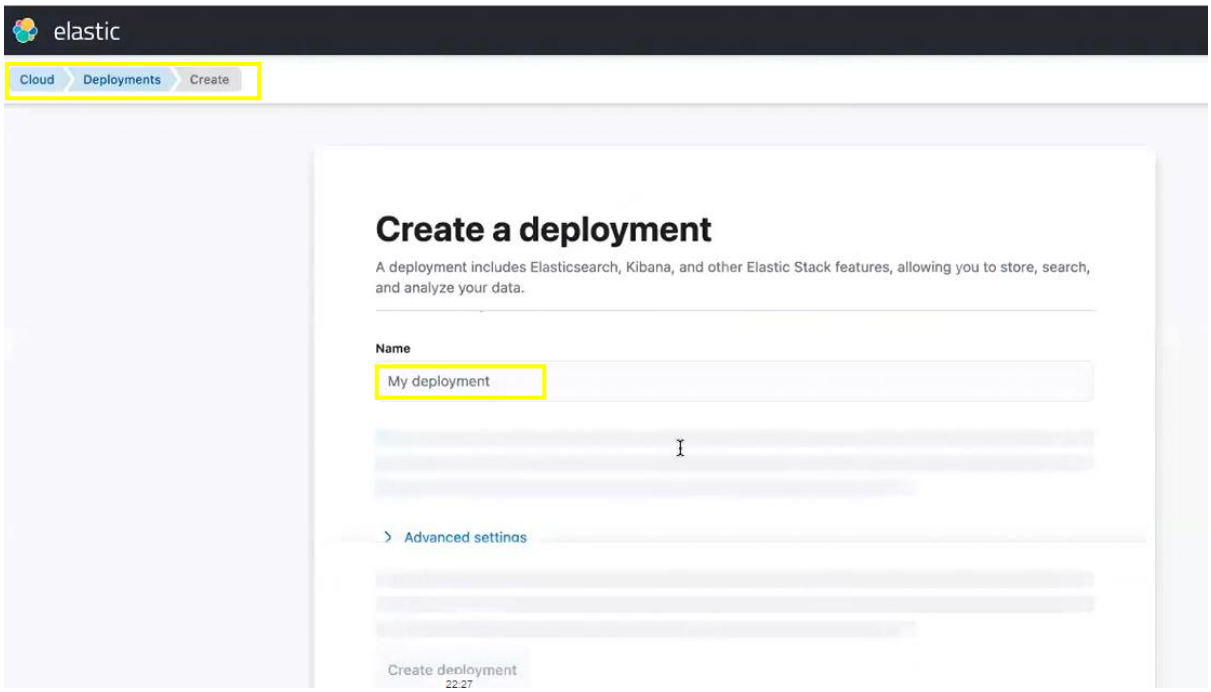
After that u will trigger here https://cloud.elastic.co/registration



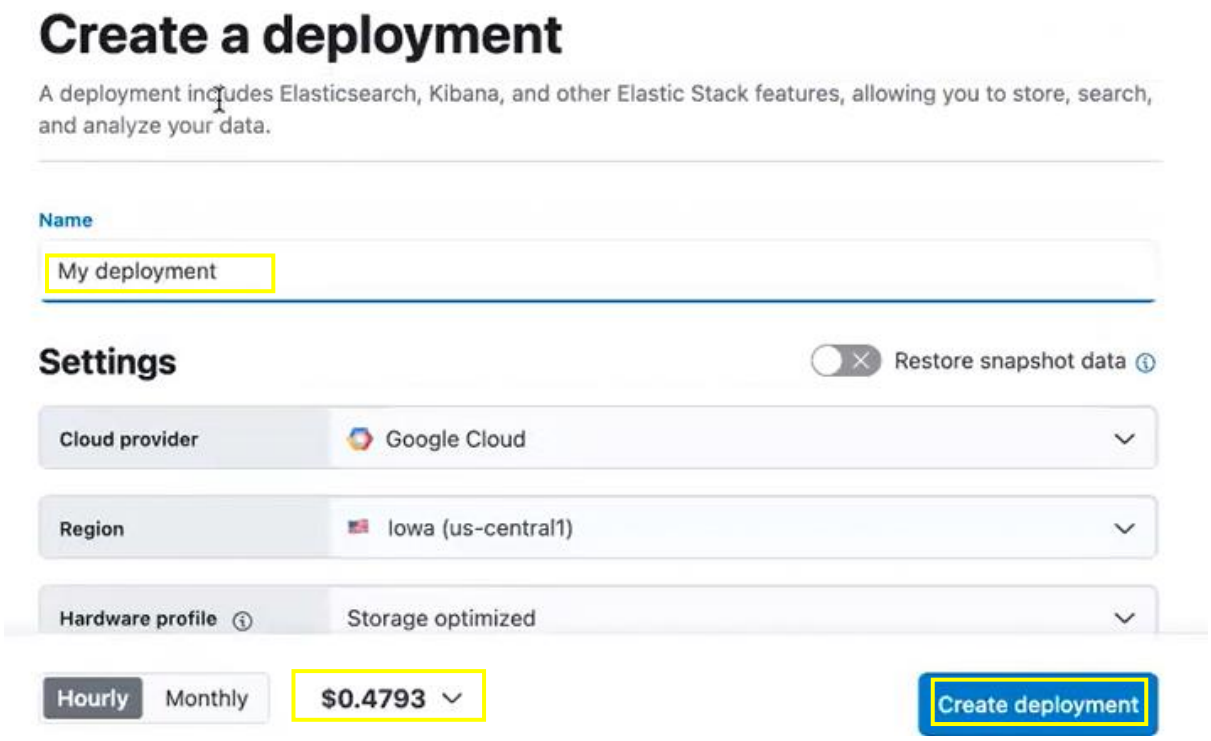
After Login home page in elastic search



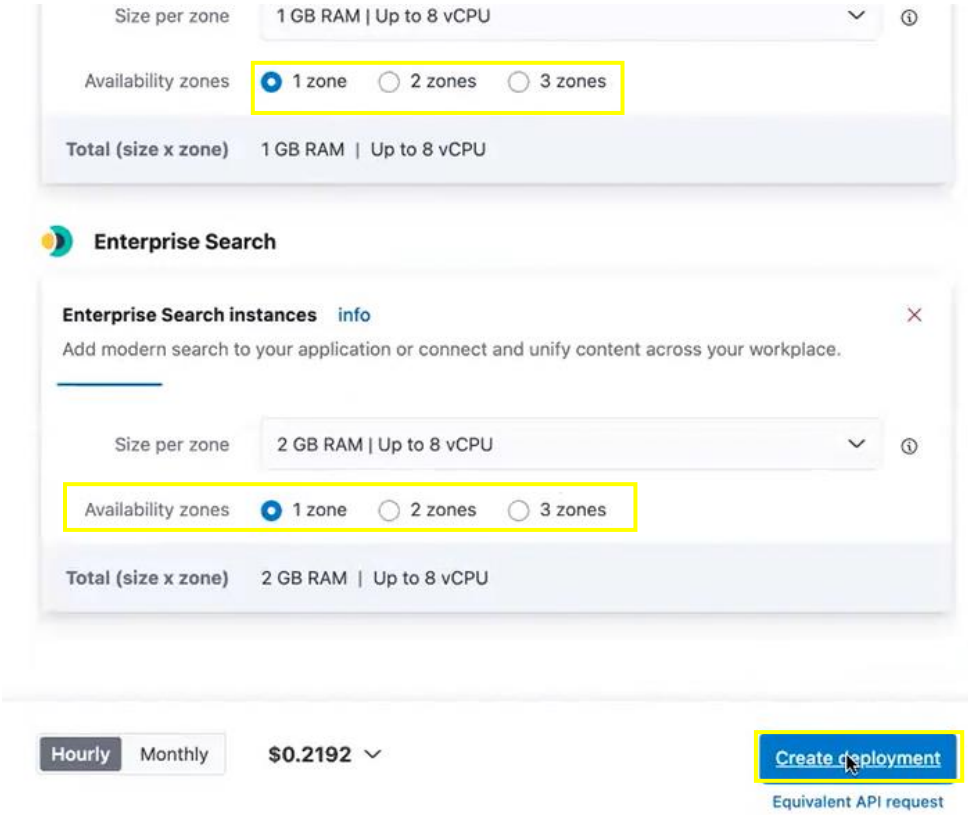
Create a deployment



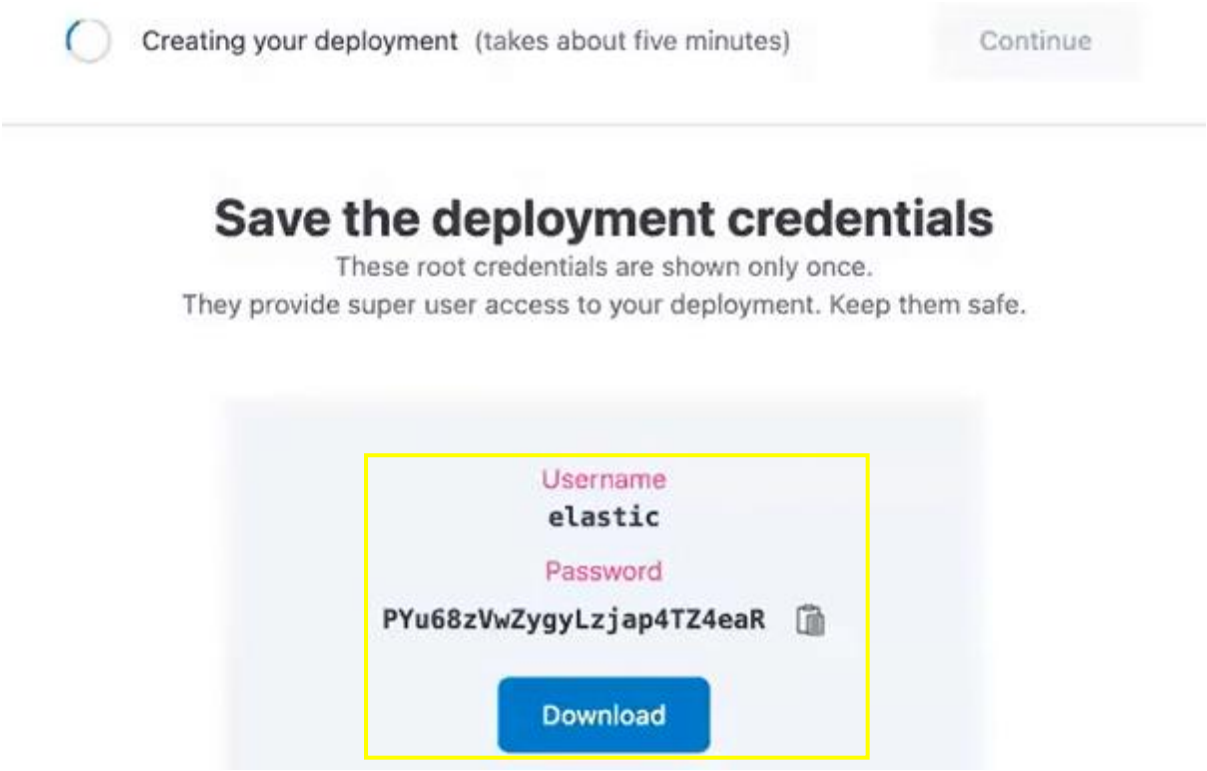
U can give any name and u can create it



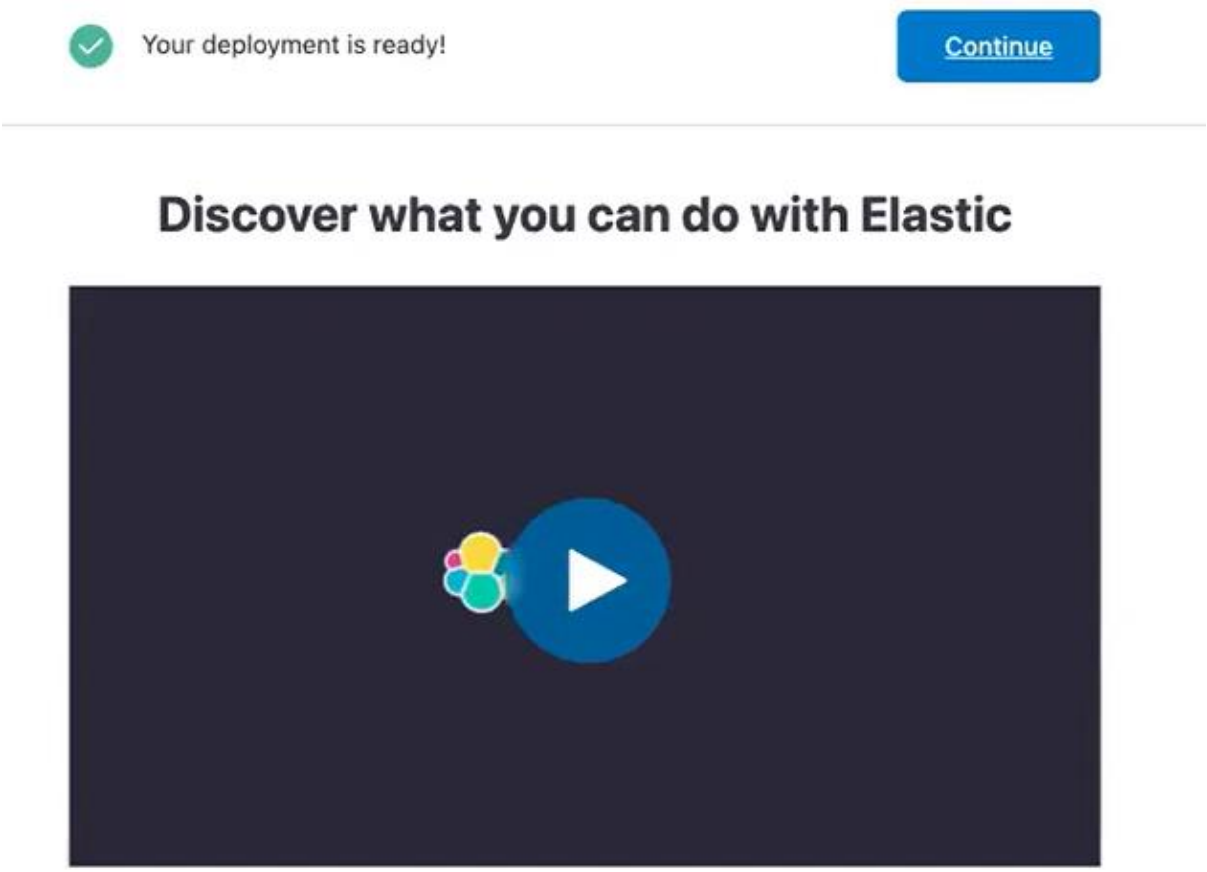
For 1 zone or 2 zone select it, click on create deployment



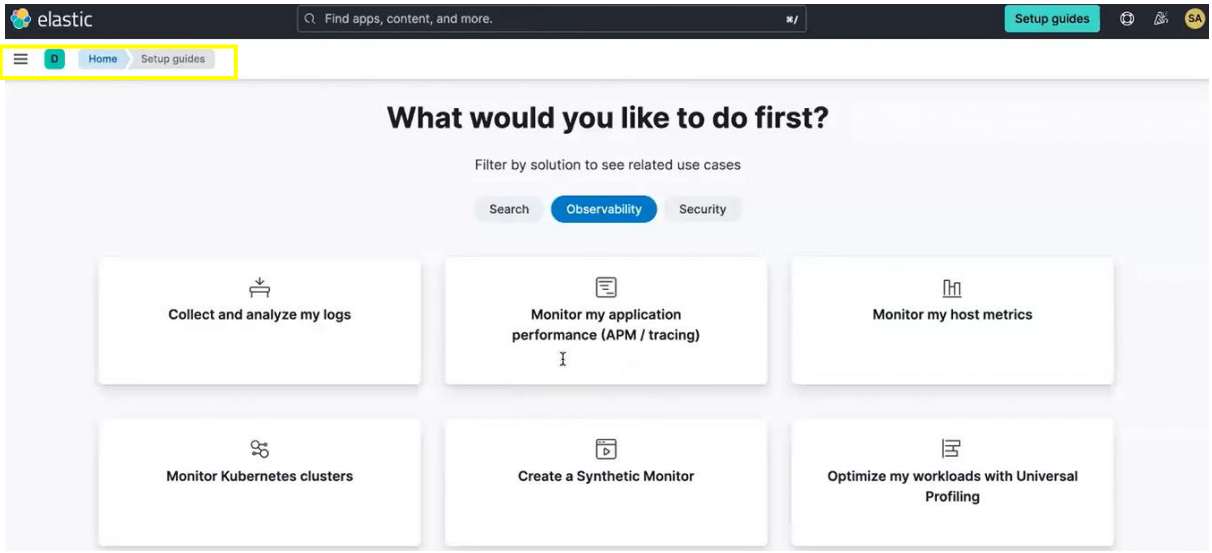
This is Username and password save it.



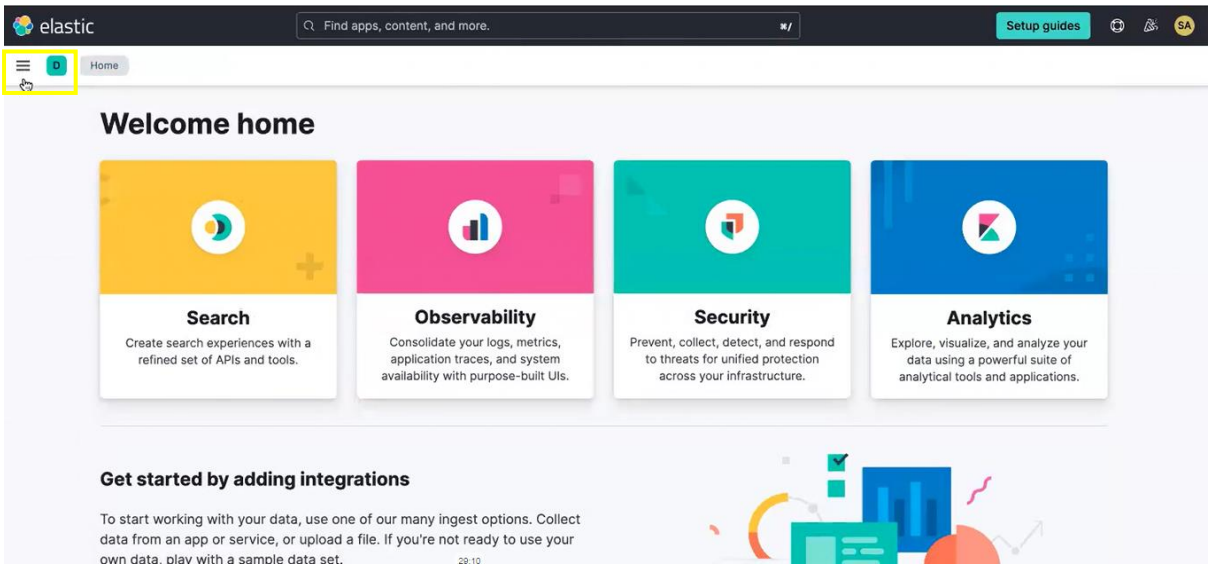
Continue



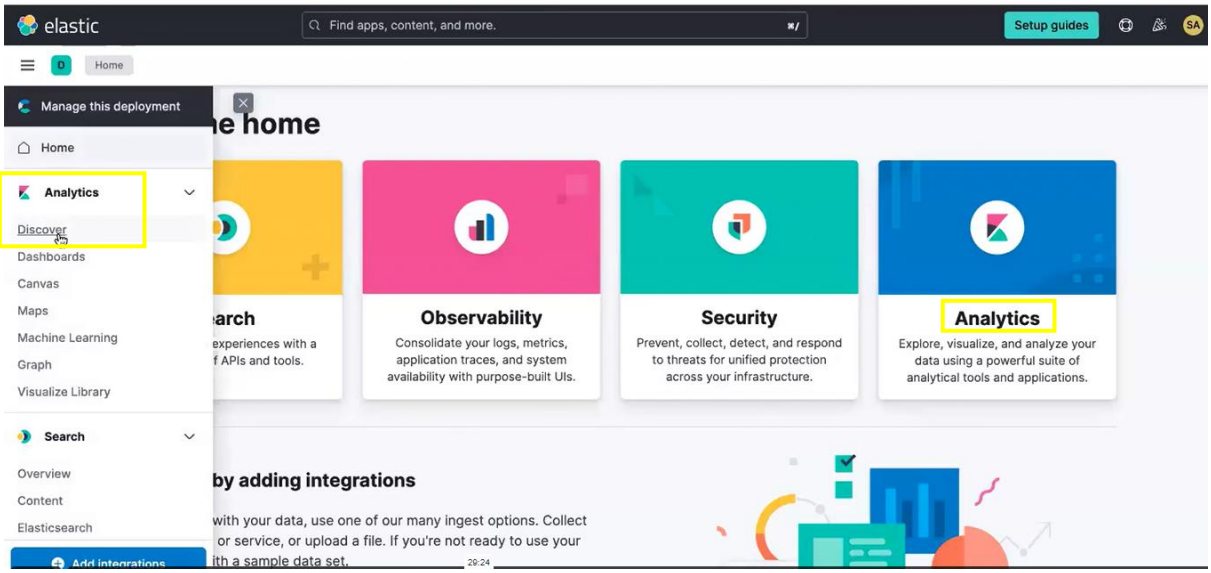
This is Dashboard of Elastic search



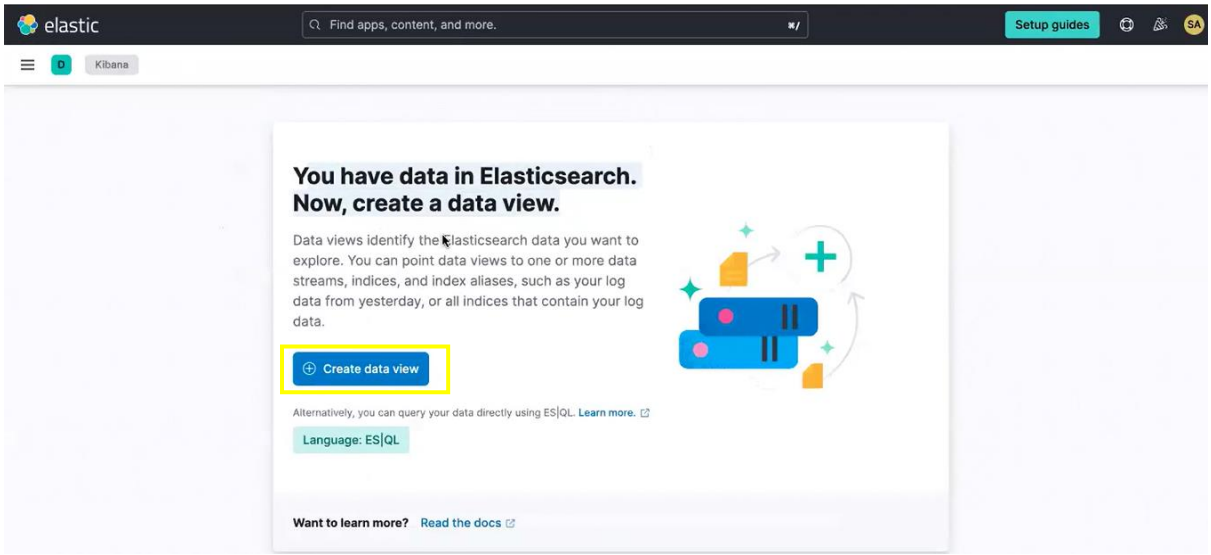
This is Home Page and click on 3 bar



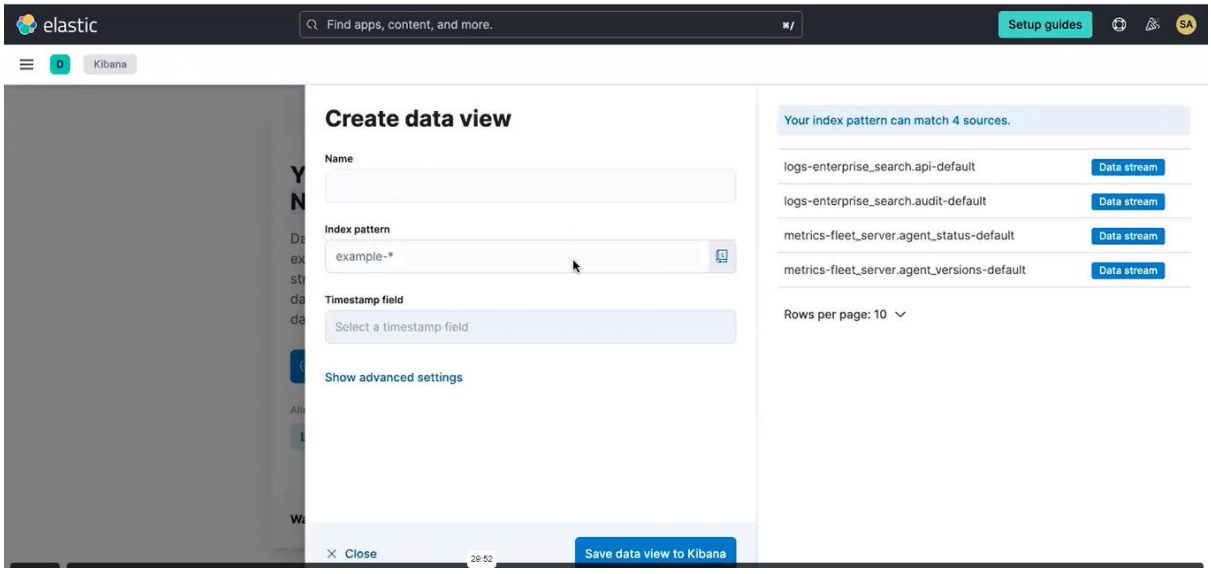
Click on discover

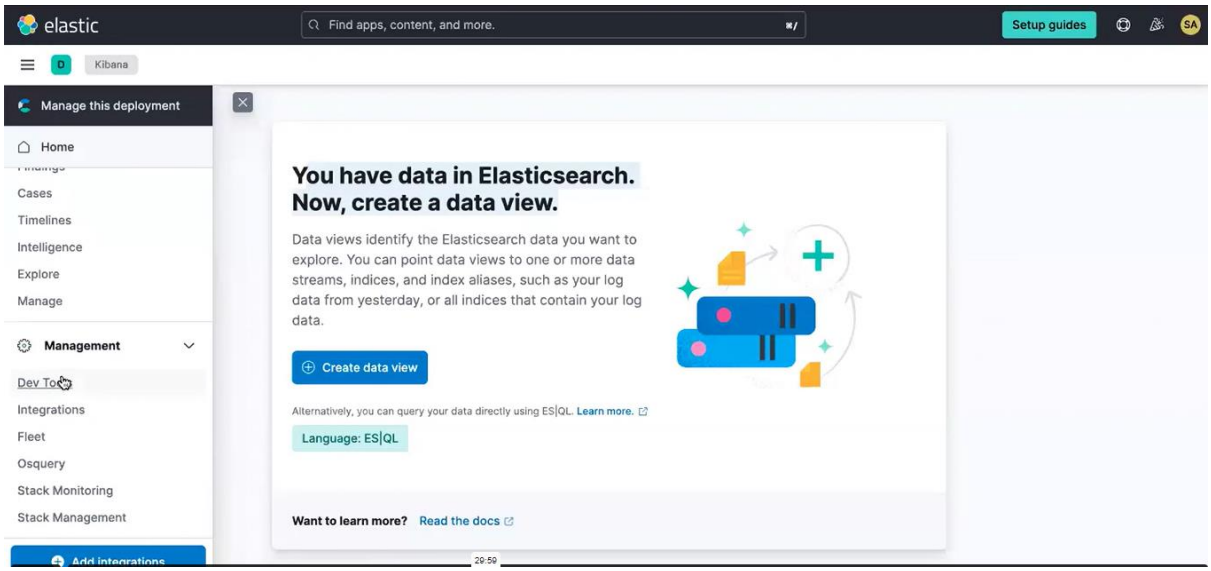


We have data in elastic search and click to create a data view

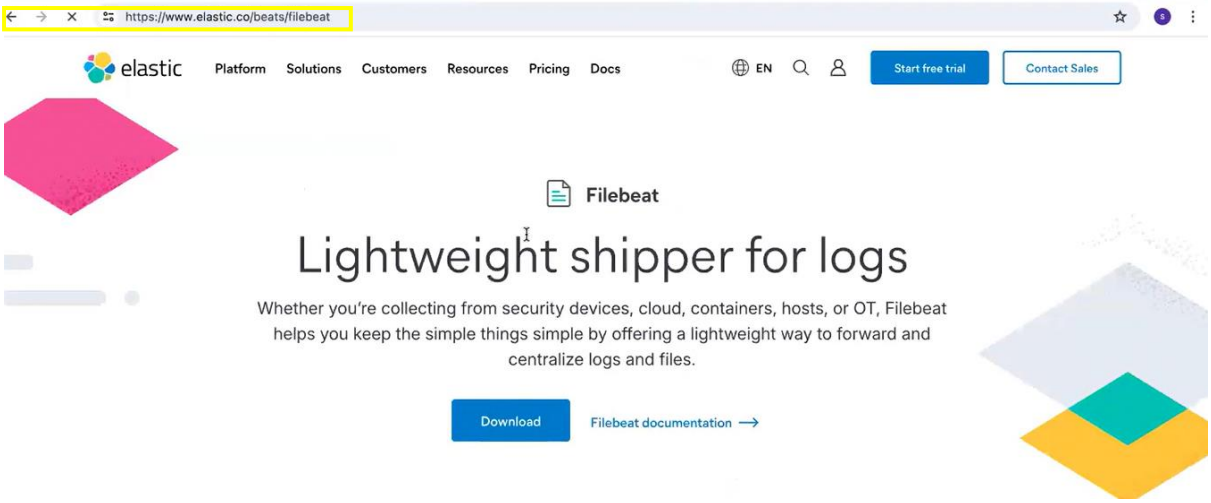


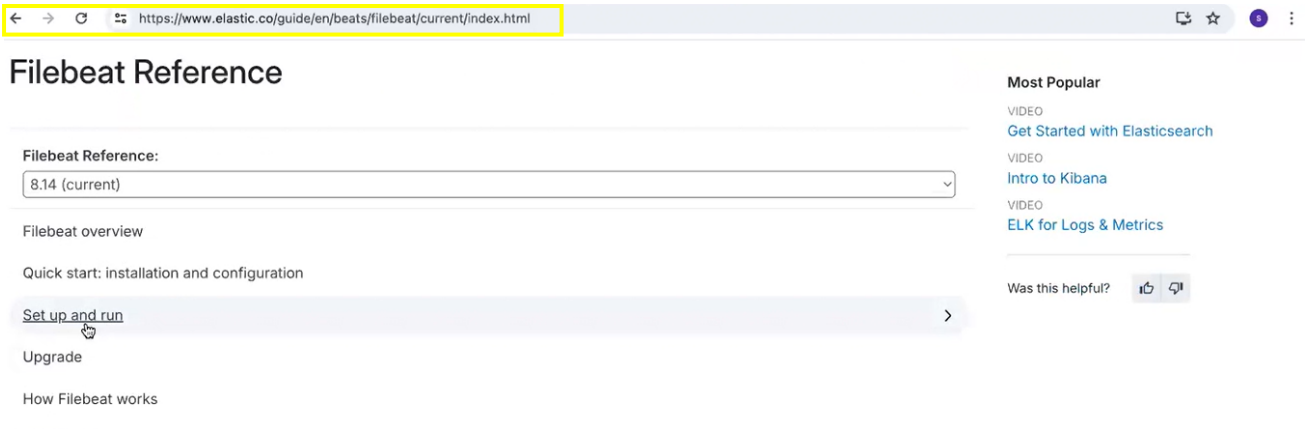
Remember this page and go back u need credentials to connect





The link https://www.elastic.co/beats/filebeat directs you to a specific page on the official Elastic website dedicated to Filebeat, which is a lightweight shipper for forwarding and centralizing log data. Elastic provides various "Beats" modules, each tailored for different purposes, and Filebeat is specifically designed for collecting and forwarding log files. On this page, you can find information about Filebeat, its features, use cases, documentation, and how to get started with it.

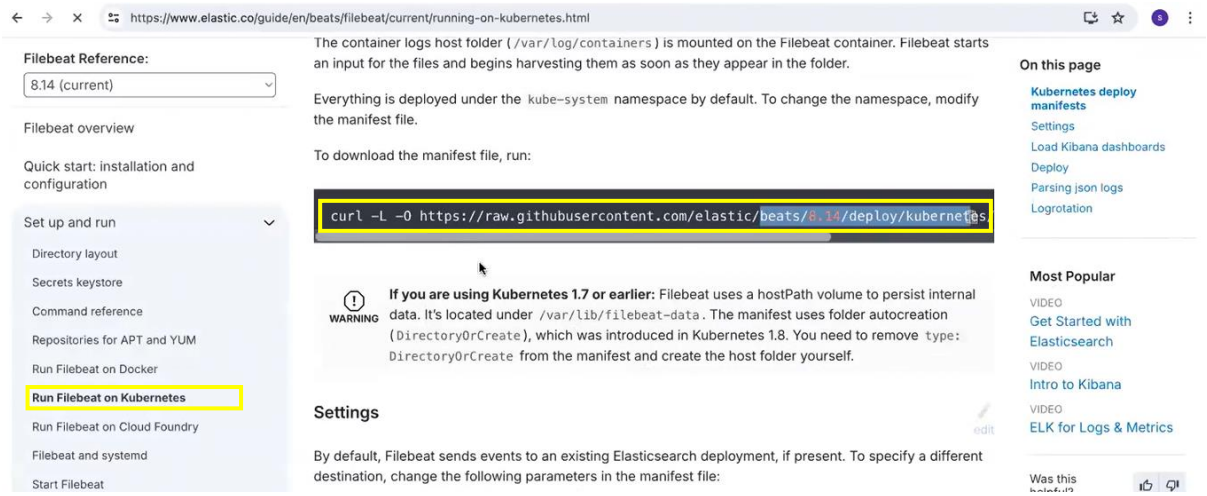




https://www.elastic.co/guide/en/beats/filebeat/current/running-on-kubernetes.html-visit page

curl -L -O https://raw.githubusercontent.com/elastic/beats/8.14/deploy/kubernetes/filebeat-kubernetes.yaml- for download

Installing filebeats on Kubernetes cluster



Installing filebeats on Kubernetes and ls we can the the filebeats

```
ubuntu@kubernetesmaster:~$ curl -L -O https://raw.githubusercontent.com/el
astic/beats/8.14/deploy/kubernetes/filebeat-kubernetes.yaml
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
rrent
                                 Dload  Upload   Total   Spent    Left  Sp
eed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
100  5981  100  5981    0     0  14223      0 --:--:-- --:--:-- --:--:-- 1
4240
ubuntu@kubernetesmaster:~$ ls
clusterip.yml   filebeat-kubernetes.yaml  nodeport.yml  replicaset.yml
deployment.yml  ingressrule.yml           pod.yml
ubuntu@kubernetesmaster:~$
```

Elastic and Elastic Cloud are both services provided by Elastic, but they serve different purposes:

1. Elastic (https://www.elastic.co/):

This is the main website for Elastic, the company. Here, you can find information about all the products and solutions offered by Elastic, including Elasticsearch, Kibana, Logstash, Beats, APM, and more. This website provides resources for developers, users, and businesses interested in using Elastic's products for search, logging, analytics, and observability.

2. Elastic Cloud (https://cloud.elastic.co/):

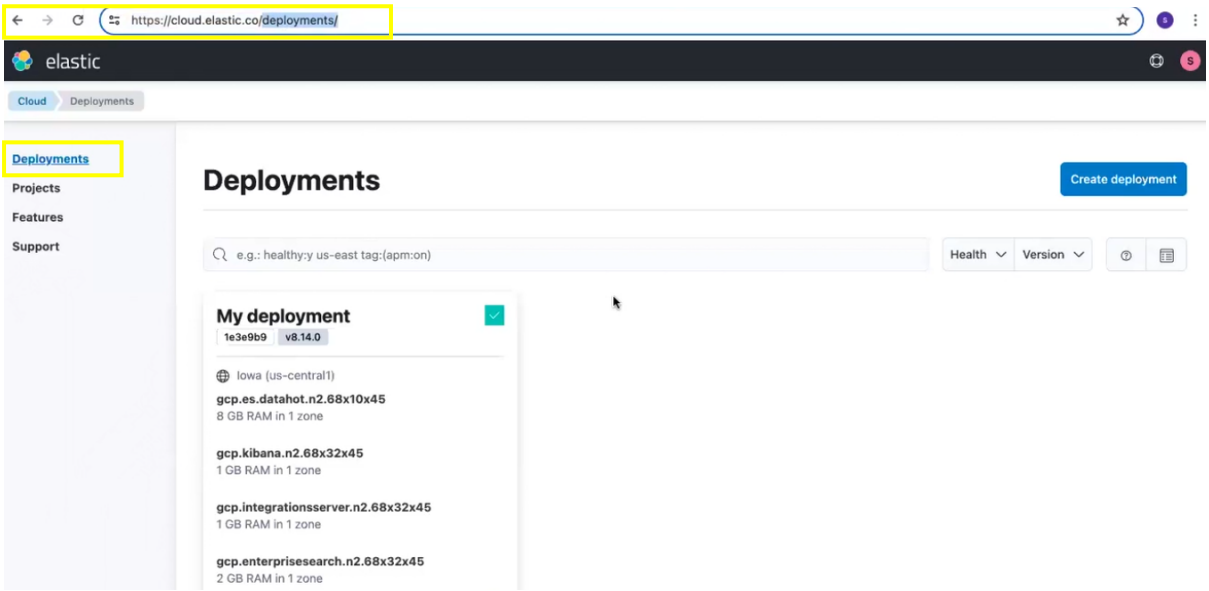
Elastic Cloud is a cloud-based service offered by Elastic. It provides managed Elasticsearch and Kibana deployments, allowing users to easily deploy, manage, and scale Elasticsearch clusters without the need to set up or maintain infrastructure. The Elastic Cloud website is specifically focused on the Elastic Cloud service, providing information and tools for managing deployments in the cloud.

While both websites are operated by Elastic, they cater to different audiences and offer distinct sets of services and information.

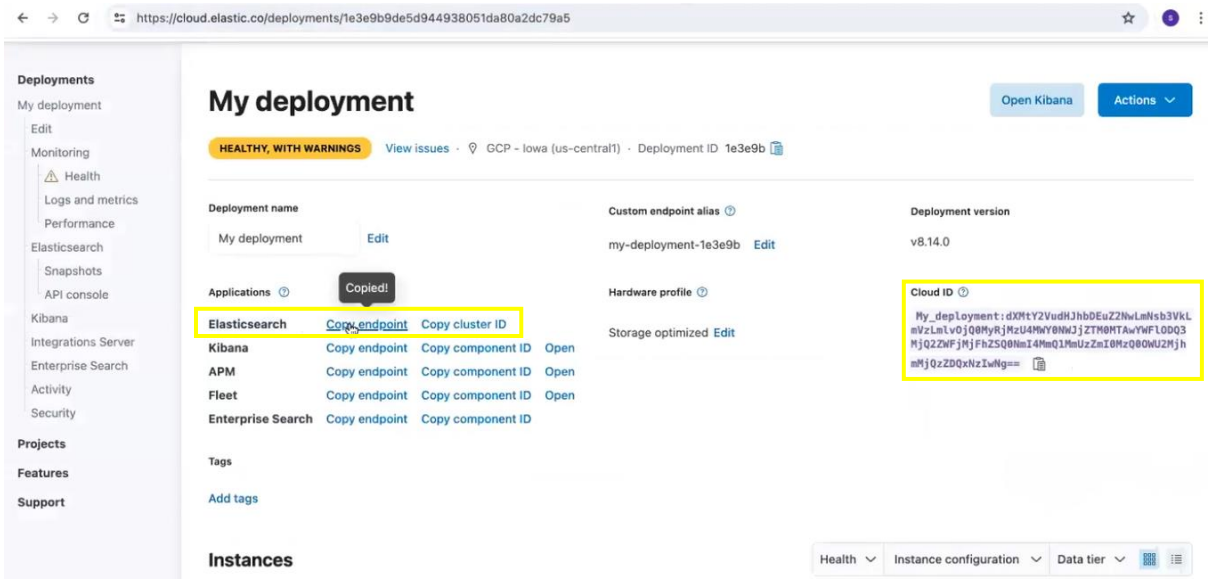
Now to go to cloud elastic website and click your deployment

The link https://cloud.elastic.co/deployments

directs you to the Elastic Cloud website, specifically to the "Deployments" page. Elastic Cloud is a cloud-based service provided by Elastic that offers managed Elasticsearch, Kibana, and other Elastic Stack products. On this page, users can manage their deployments, including creating, configuring, and monitoring Elasticsearch clusters and associated services through a user-friendly interface.



Copy the Elastic search endpoint



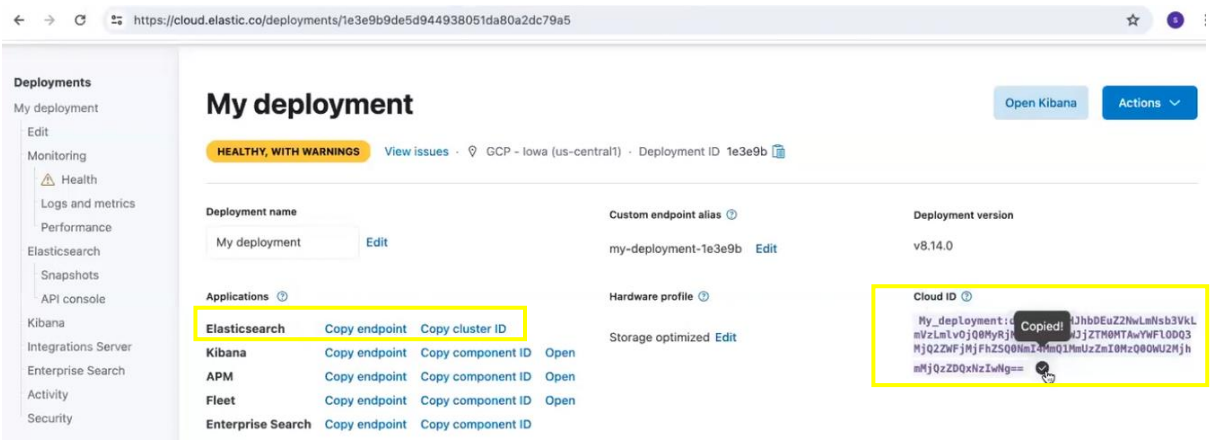
Now again come to Kubernetes server and open editor

```
ubuntu@kubernetesmaster:~$ vi filebeat-kubernetes.yaml
```

In the Env part this is without changing, remember

```yaml
env:
- name: ELASTICSEARCH_HOST
  value: elasticsearch
- name: ELASTICSEARCH_PORT
  value: "9200"
- name: ELASTICSEARCH_USERNAME
  value: elastic
- name: ELASTICSEARCH_PASSWORD
  value: changeme
- name: ELASTIC_CLOUD_ID
  value:
- name: ELASTIC_CLOUD_AUTH
  value:
- name: NODE_NAME
  valueFrom:
    fieldRef:
      fieldPath: spec.nodeName
```

Now copy the cloud id on right side



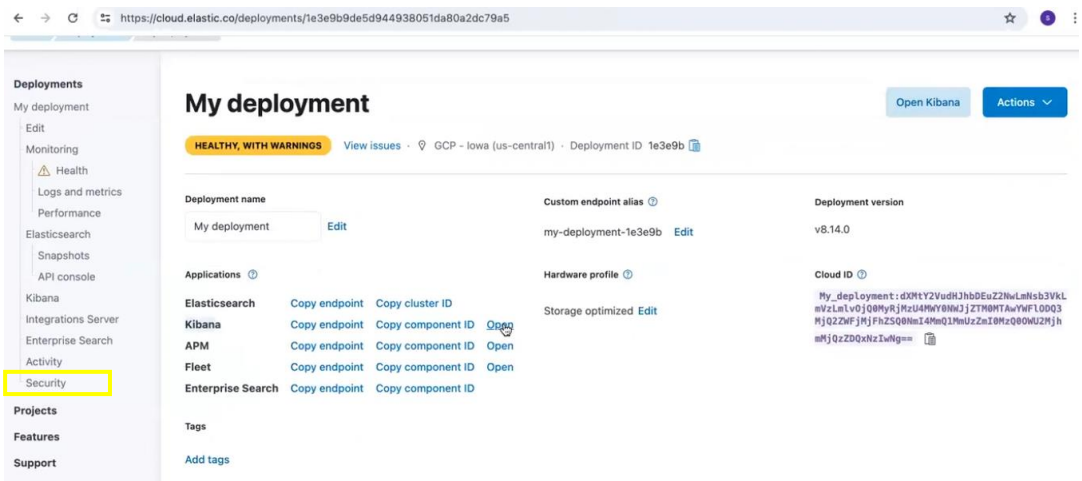
On the value paste copied endpoint, cloud id, paste deployment credentials on Elastic cloud Auth

```
        env:
        - name: ELASTICSEARCH_HOST
          value: https://my-deployment-1e3e9b.es.us-central1.gcp.cloud.es.
io
        - name: ELASTICSEARCH_PORT
          value: "9243"
        - name: ELASTICSEARCH_USERNAME
          value: elastic
        - name: ELASTICSEARCH_PASSWORD
          value: changeme
        - name: ELASTIC_CLOUD_ID
          value: "My_deployment:dXMtY2VudHJhbDEuZ2NwLmNsb3VkLmVzLmlvOjQ0My
RjMzU4MWY0NWJjZTM0MTAwYWFlODQ3MjQ2ZWFjMjFhZSQ0NmI4MmQ1MmUzZmI0MzQ0OWU2Mjhm
MjQzZDQxNzIwNg=="
        - name: ELASTIC_CLOUD_AUTH
          value: "elastic:PYu68zVwZygyLzjap4TZ4eaR"
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
```

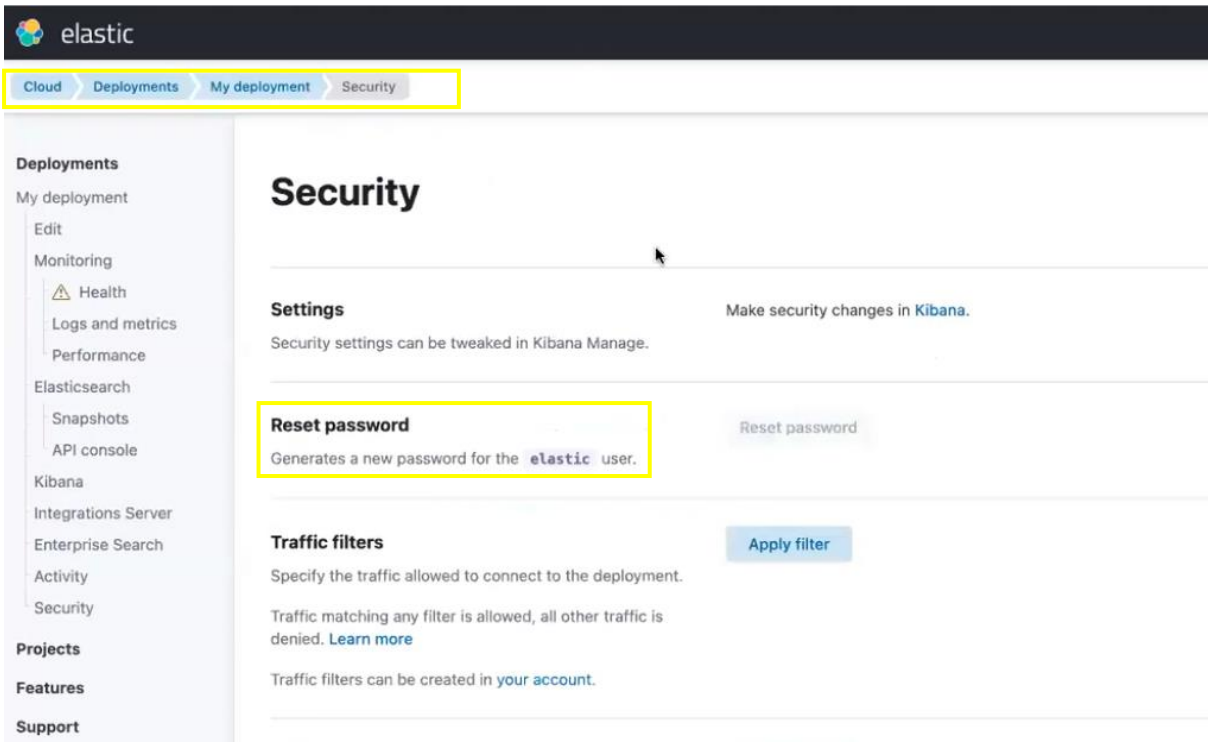
Save it. And Apply it

```
ubuntu@kubernetesmaster:~$ vi filebeat-kubernetes.yaml
ubuntu@kubernetesmaster:~$ kubectl apply -f filebeat-kubernetes.yaml
serviceaccount/filebeat created
clusterrole.rbac.authorization.k8s.io/filebeat created
role.rbac.authorization.k8s.io/filebeat created
role.rbac.authorization.k8s.io/filebeat-kubeadm-config created
clusterrolebinding.rbac.authorization.k8s.io/filebeat created
rolebinding.rbac.authorization.k8s.io/filebeat created
rolebinding.rbac.authorization.k8s.io/filebeat-kubeadm-config created
configmap/filebeat-config created
daemonset.apps/filebeat created
ubuntu@kubernetesmaster:~$ clea
```

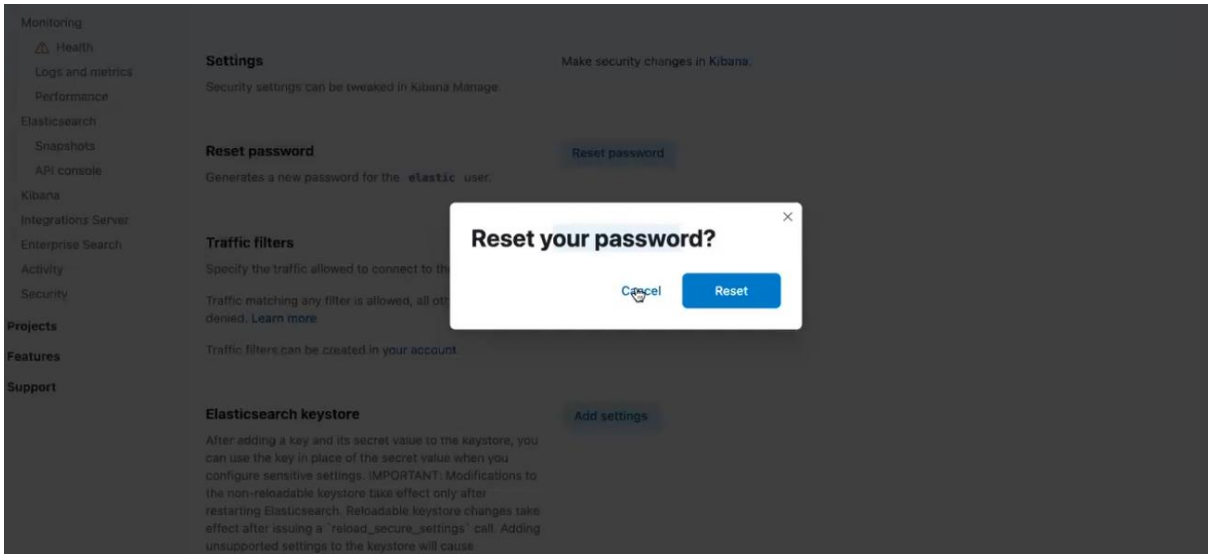
Click on Security



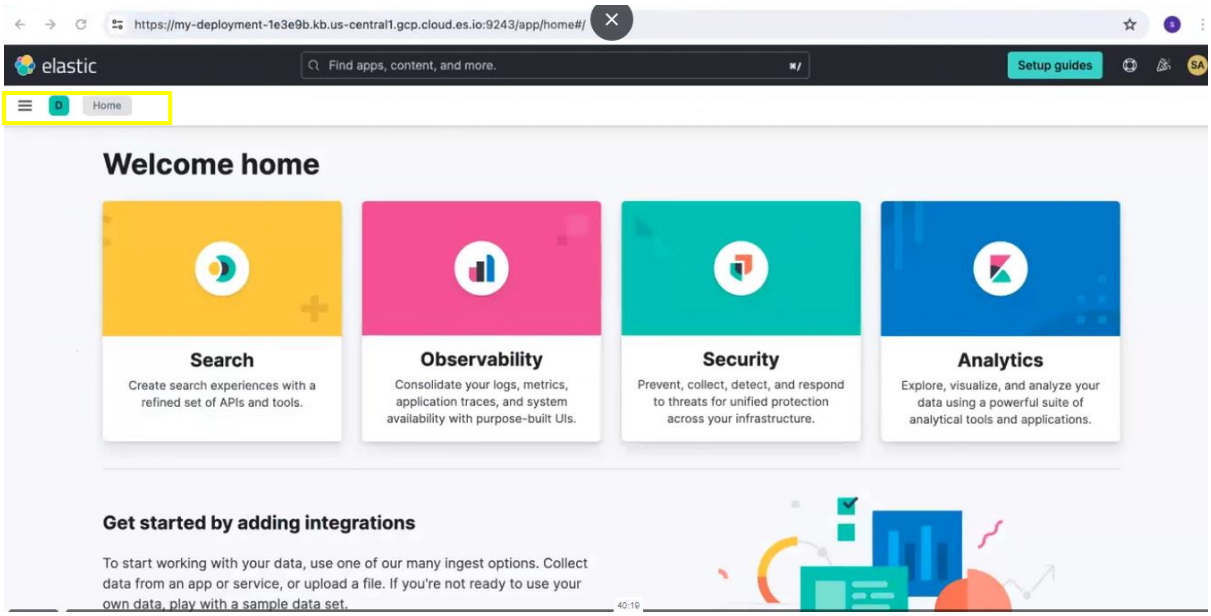
Here u can reset the password, if u want



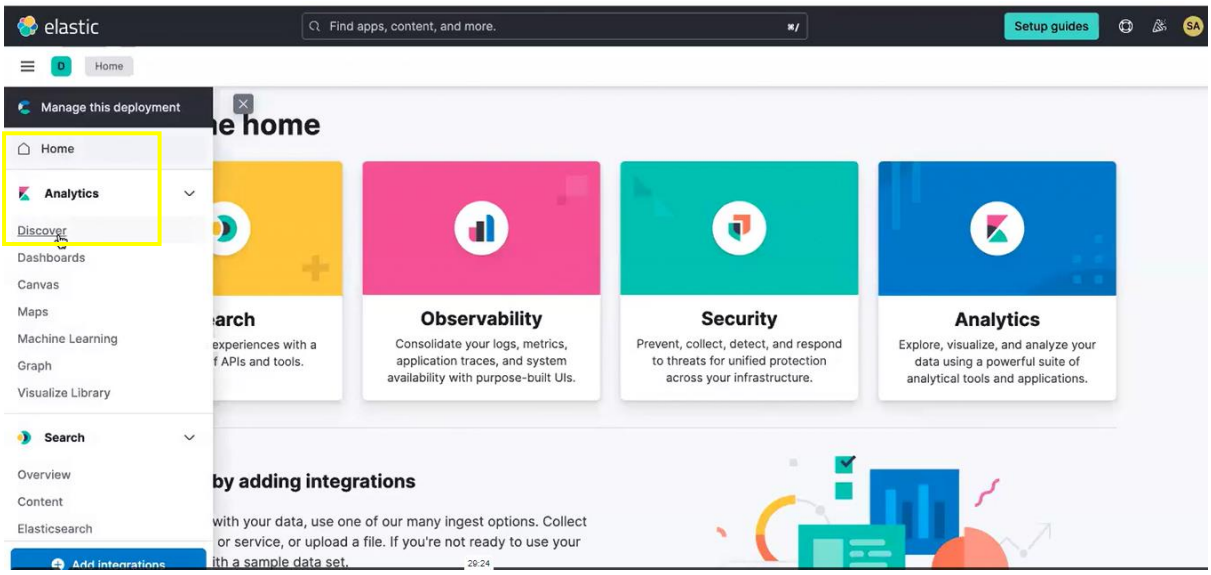
Just viewing



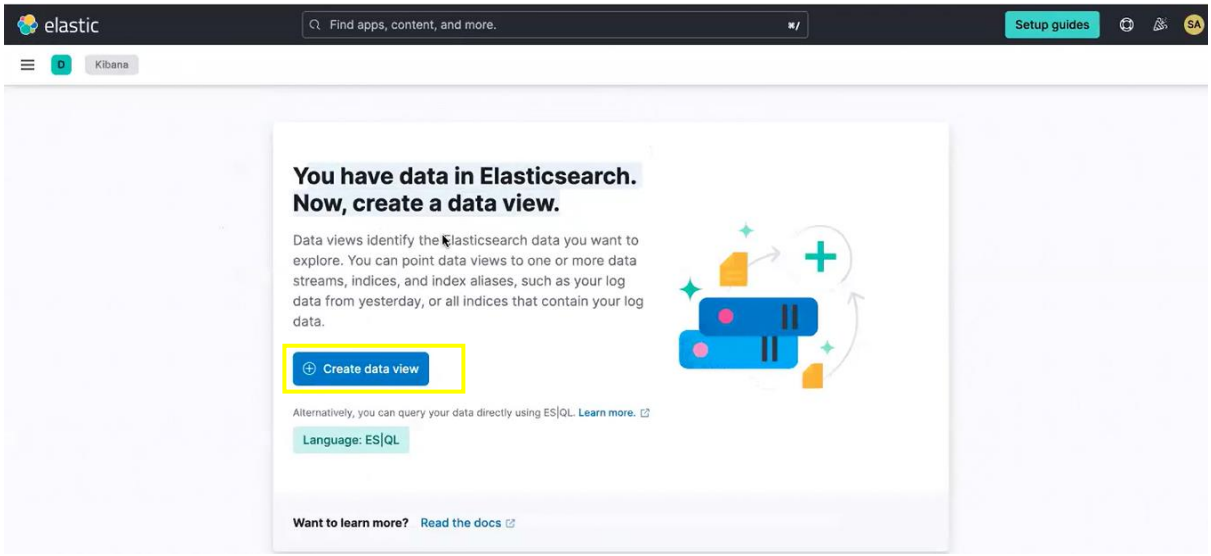
Now come to elastic search website click on 3 bar



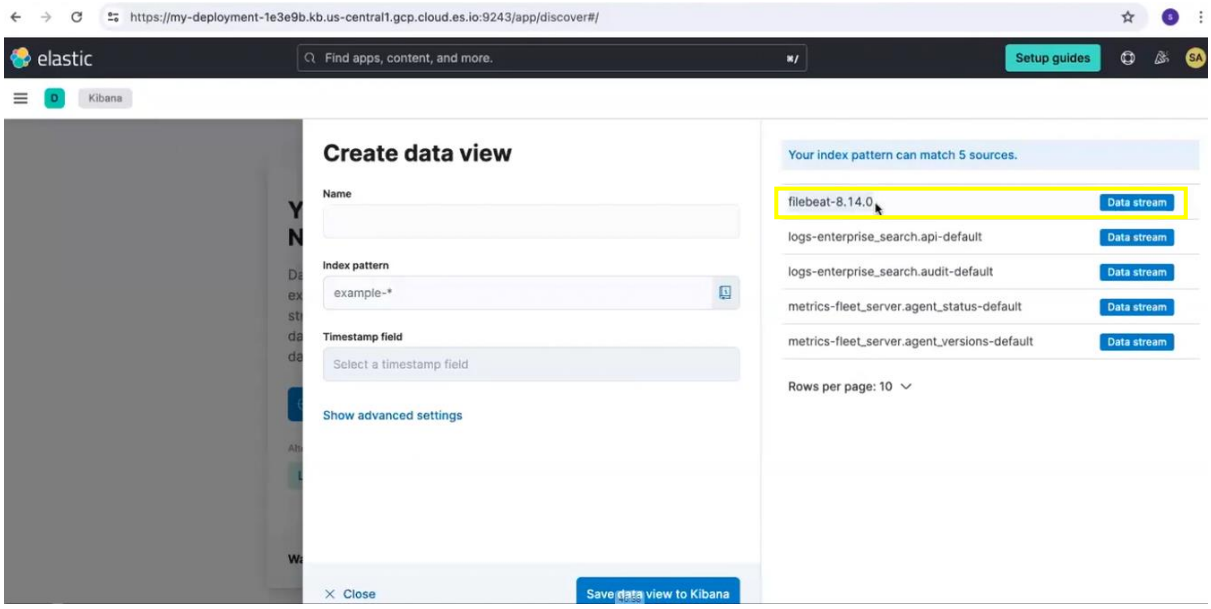
Click Discover or Analytics



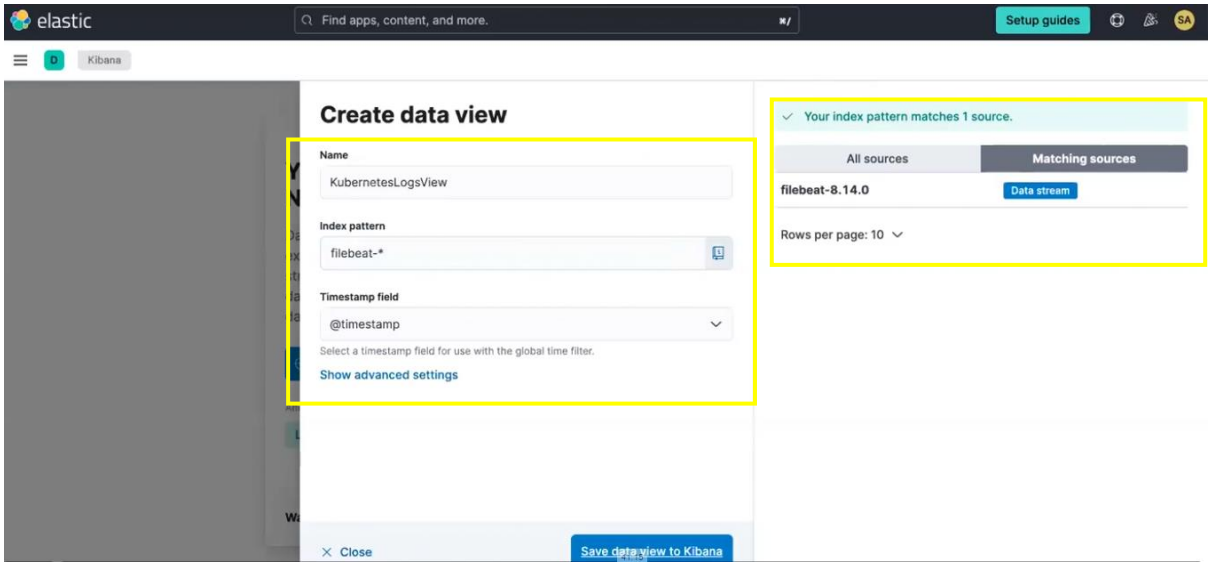
Now Click create data view



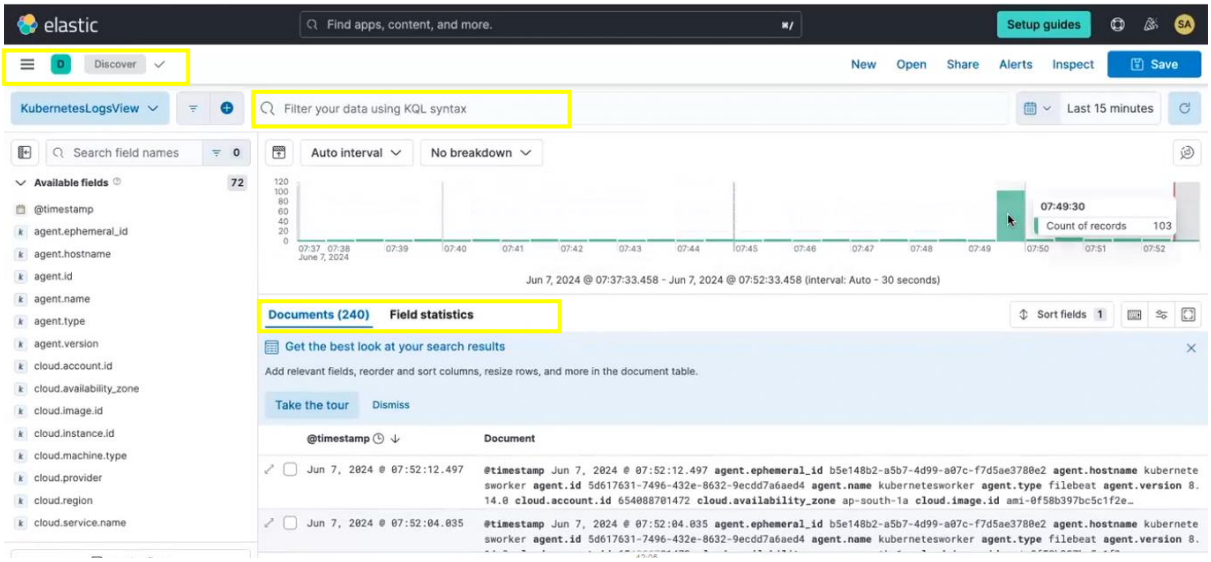
Now u will get the filebeat option on right side previous not so we gone through cloud elastic and my deployment copying endpoint and configuring on Kubernetes server and applying and now came to elastic search to visualizing logs through kibana it will asking credentials that we have created



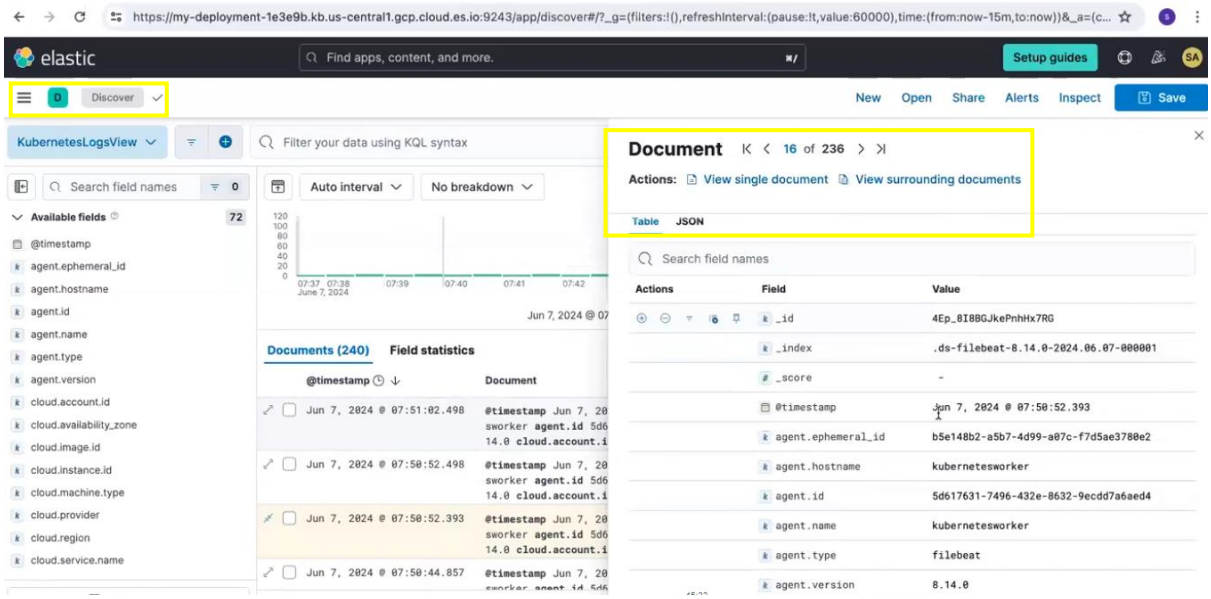
Type Name as ur wish and click data stream and follow it.



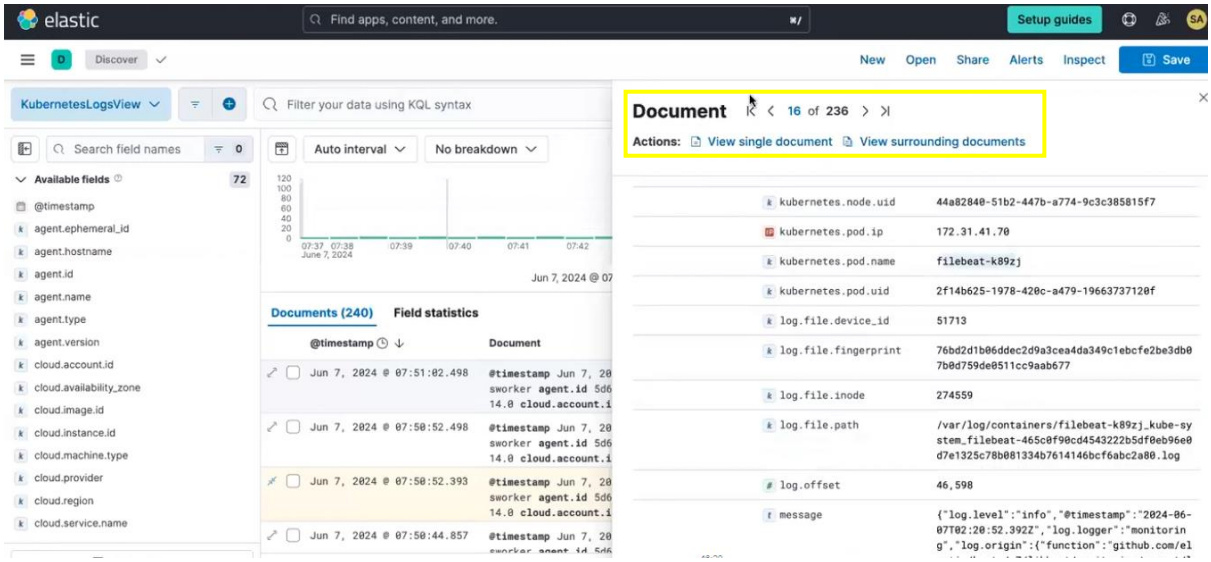
Now this is our Kubernetes logs dashboard in kibana



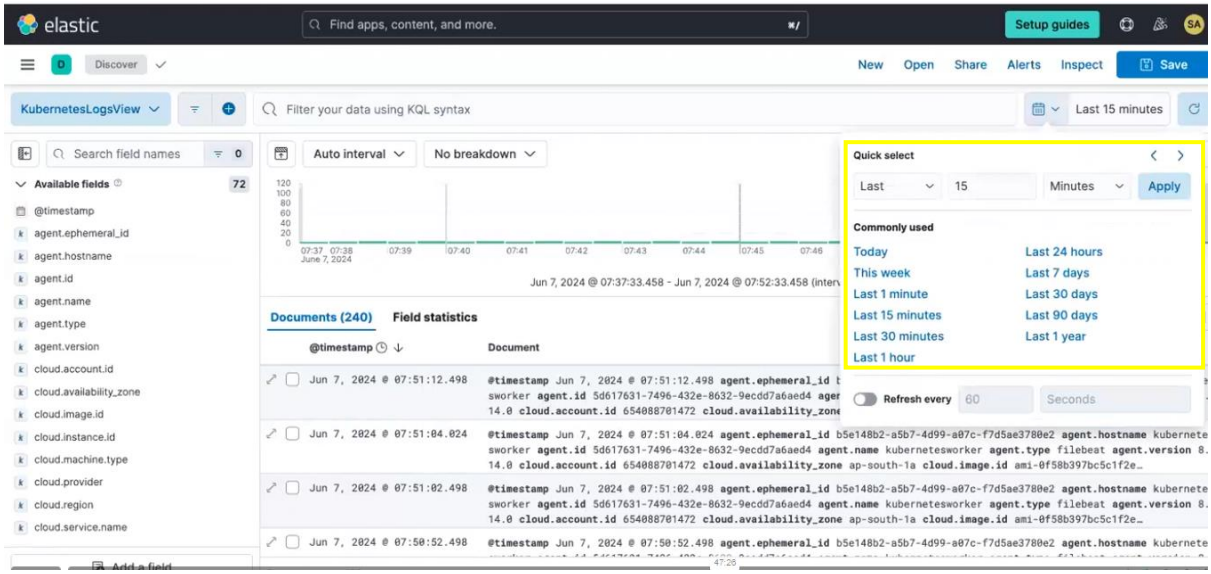
We can see the logs in table and json format



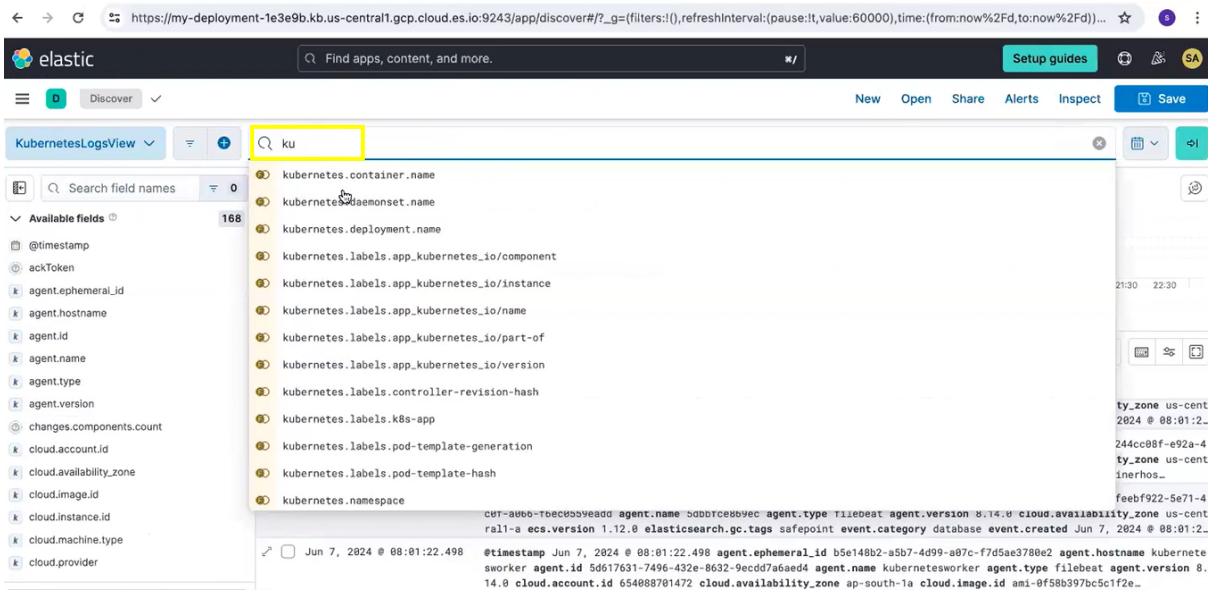
This is table format



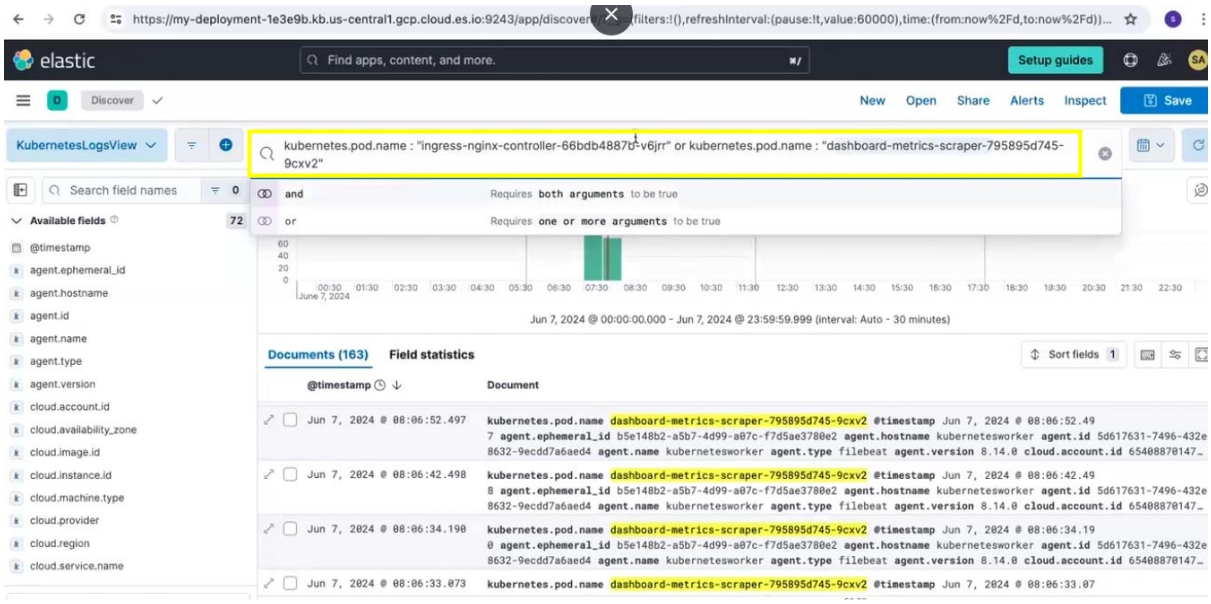
And we can filter according to our time frame



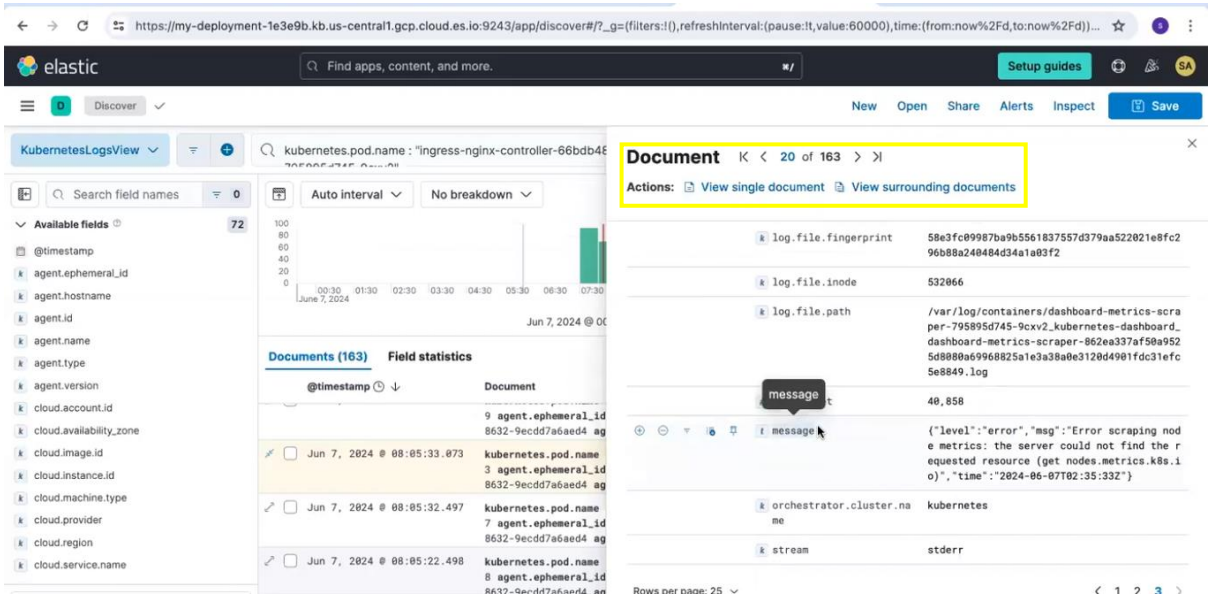
And we can search particular logs with keywords of log



Filtered with particular logs



Like explore it. To learn



ELK is a powerful combination of three open-source tools: Elasticsearch, Logstash, and Kibana.

Elasticsearch: Acts as a search and analytics engine, storing and indexing data for fast retrieval and analysis.

Logstash: Collects, processes, and forwards log data from various sources to Elasticsearch.

Kibana : Provides a user-friendly interface for visualizing and analyzing data stored in Elasticsearch.

Together, ELK helps organizations manage and analyze large volumes of log data, enabling them to gain insights into system performance, troubleshoot issues, and monitor trends.

Websites:

- [Elasticsearch](https://www.elastic.co/elasticsearch/):

Official website for Elasticsearch, providing information, documentation, and resources for the search and analytics engine.

- [Logstash](https://www.elastic.co/logstash/):

Official website for Logstash, offering details, documentation, and resources for the log collection and processing tool.

- [Kibana](https://www.elastic.co/kibana/):

Official website for Kibana, providing information, documentation, and resources for the visualization and analytics platform.

These websites offer resources for users to learn about and utilize each component of the ELK stack effectively.