Day 2 - Iac – Terraform

We will start from creating the main.tf, so, as I told you your terraform file. Can you can give any name for your terraform file? Make sure you give main.tf, where this main.tf, main.tf it's like heart of your terraform, where only in the main.tf, you give all your resource details what all resources you want to create in the infrastructure. You place it inside the main.tf, and you create one more file called variables.tf,

so this variables.tf, namely, used to separate the dynamically changing value from your main.tf, file so that you don't get have to go and edit your main.tf, in case you want to change the value of the instance type or ami later in future. If you want to change the ami with a different version or the different ami you want to do it, you don't have to open up, or you don't have to touch your main.tf, because this is this is a crucial file way. It has got all your details or all the details about your resource creation.

what I will do the same file. I will just create a variables and later we will cut and create a new file. So this is my variables. File variables.tf, so inside my variables.tf, file what all I have to give. So I have to 1st create a variables block, variable. What variable? I'm going to create variable region. Okay, this is going to be your variable name. You can have descriptions. So this description is optional, even if you don't give it. Not a problem. But it's always good to give the description type.

```
provider "aws" {
        region = var.region
}

resource "aws_instance" "demo" {
        ami = var.ami_id
        instance_type = var.instance_type

        tags = {
                Name = "myterrad-instance"
}
}


variables.tf

variable "region" {
        description = "The AWS Region to deploy ec2"
        type = "string"
        default = "ap-south-1"
}

variable "ami_id" {
        description = "The AWS AMI to deploy ec2"
        type = "string"
        default = "ami-0c2af51e265bd5e0e"
}
variable "instance_type" {
        description = "The AWS Instance_type to deploy ec2"
        type = "string"
        default = "t2.micro"
-- INSERT --
```

Main.tf file

Variable.tf

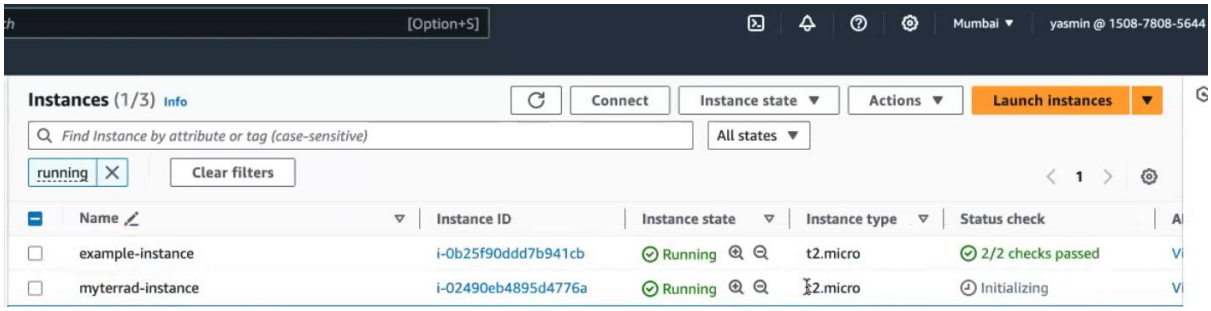
In main.tf , give variable names [region ,ami_id , instance_type]

Duplicate the window ,Variable.tf file written in that

Then give all terraform commands one by one.

```
ubuntu@ip-172-31-33-105:~/test$ vi main.tf
ubuntu@ip-172-31-33-105:~/test$ ls
main.tf  variables.tf
ubuntu@ip-172-31-33-105:~/test$ terraform init
```

```
variable "region" {
        description = "The AWS Region to deploy ec2"
        type = string
        default = "ap-south-1"
}

variable "ami_id" {
        description = "The AWS AMI to deploy ec2"
        type = string
        default = "ami-0c2af51e265bd5e0e"
}
variable "instance_type" {
        description = "The AWS Instance_type to deploy ec2"
        type = string
        default = "t2.micro"
}
```



this is another file. So based on the stages or based on the environment, you would have to give different values to your infrastructure. So for the so if I'm like creating for **my dev environment**, assume that I'm creating it for my dev environment

```
ubuntu@ip-172-31-33-105:~/test$ vi terraform-dev.tfvars
```

```
instance_type = "t2.micro"
region = "us-east-2"
ami_id = "ami-0649bea3443ede307"
~
```

Use diff region so give ami id. click launch instance get ami id

ami-0649bea3443ede307 (64-bit (x86), uefi-preferred) / ami-0e8ca3db2504c5f6d (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term suppo
is optimized for AWS and designed to provide a secure, stable and high-performance execution environment t
develop and run your cloud applications.

Architecture	Boot mode	AMI ID	
64-bit (x86)	uefi-preferred	ami-0649bea3443ede307	Verified provider

my Qa environment:

```
ubuntu@ip-172-31-33-105:~/test$ vi terraform-qa.tfvars
```

```
instance_type = "t2.medium"
region = "us-west-1"
ami_id = "ami-0fb83b36371e7dab5"
```

Prod.tfvars environment:

```
ubuntu@ip-172-31-33-105:~/test$ vi terraform-prod.tfvars
```

```
instance_type = "t2.medium"
region = "us-west-1"
```

So now I've given the values. So what I will do now I have given the values here we will again go to the variables.tf So now, instead of passing the values here, I will not. I will remove the default value, because I don't want the default value here, because, anyhow, we are for the different environment. We are passing it through.

```
variable "region" {
        description = "The AWS Region to deploy ec2"
        type = string
}

variable "ami_id" {
        description = "The AWS AMI to deploy ec2"
        type = string

}
variable "instance_type" {
        description = "The AWS Instance_type to deploy ec2"
        type = string

}
```

Give all terraform command:

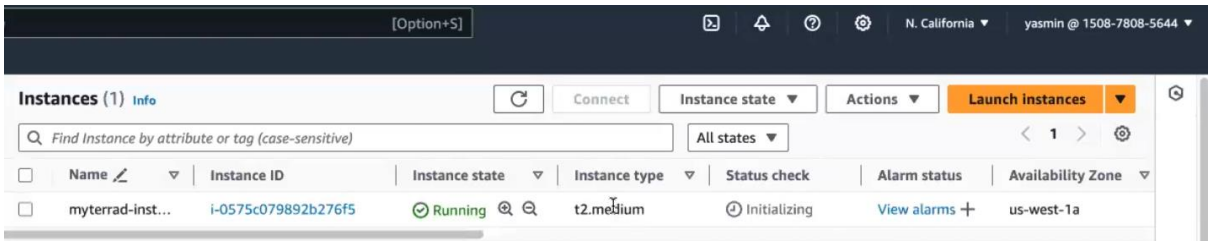
In this we mention the file name **[dev.tfvars]**

```
ubuntu@ip-172-31-33-105:~/test$ terraform init --var-file="terraform-dev.tfvars"
Initializing the backend...
ubuntu@ip-172-31-33-105:~/test$ terraform plan --var-file="terraform-dev.tfvars"
aws_instance.demo: Refreshing state... [id=i-02490eb4895d4776a]
```

```
ubuntu@ip-172-31-33-105:~/test$ terraform apply --var-file="terraform-dev.tfvars"
```

Prod.tfvars:

```
ubuntu@ip-172-31-33-105:~/test$ terraform plan --var-file="terraform-prod.tfvars"
ubuntu@ip-172-31-33-105:~/test$ terraform init --var-file="terraform-prod.tfvars"
ubuntu@ip-172-31-33-105:~/test$ terraform apply --var-file="terraform-prod.tfvars"
```



Qa.tfvars:

Init, plan , apply give also for qa.tfvars file

```
ubuntu@ip-172-31-33-105:~/test$ ls
main.tf  terraform-dev.tfvars  terraform-prod.tfvars  terraform-qa.tfvars  terraform.tfstate  terraform.tfstate.backup  variables.tf
ubuntu@ip-172-31-33-105:~/test$ cat main.tf
provider "aws" {
        region = var.region
}

resource "aws_instance" "demo" {
        ami = var.ami_id
        instance_type = var.instance_type

        tags = {
                Name = "myterrad-instance"
}
}
```

in this variables.tf, file, you're referring to all the values that you pass inside the main.pf, file, you give the variable name, but the value, instead of assigning it in the same file, you're assigning it in a different file because we have multiple values. You are not sticking with one value for your infra, you have multiple values based on the environment.

How to save your state file In the remote infrastructure:

Okay, so now **your state file is located in your local mission**, but in the real time we cannot afford to have our state file in our local mission. **We have to put it in our private (i.e)** put it in our remote resources. Okay, so this state file is very important when it comes to terraform. **So you have to save your state file In the remote infrastructure**. So remote means In our case we are creating our infra and the aws, we can put this state file in into your S3 bucket and you can lock it with your dynamodb table. So dynamodb table what it does, it will help you to log the S3 bucket access.

Backend.tf:

```
ubuntu@ip-172-31-33-105:~/test$ vi backend.tf
```

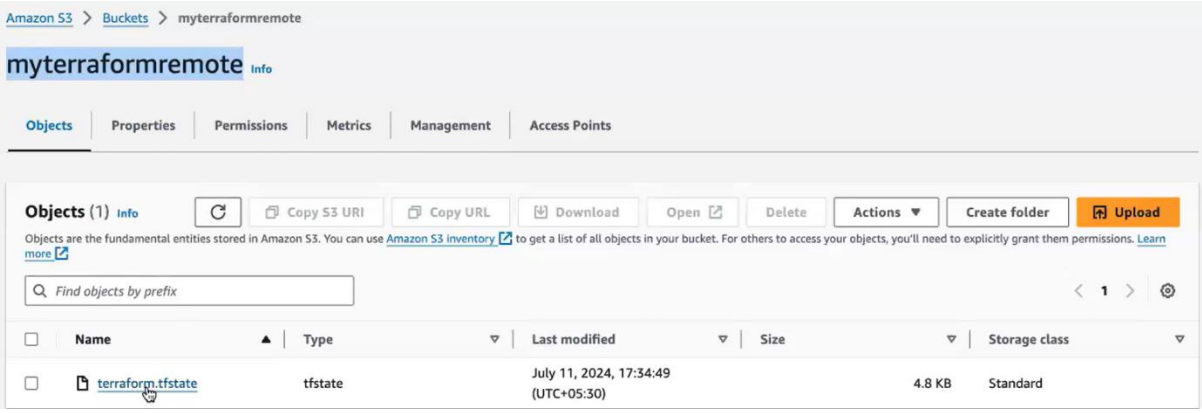
The key is nothing but the name of the State file.tf. so your terraform file will be saved in the S3 bucket with this name.

```
terraform {
        backend "s3" {
          bucket = "myterraformremote"
          key = "terraform.tfstate"
          region = "ap-south-1"

}
}
```

```
ubuntu@ip-172-31-33-105:~/test$ terraform init
Initializing the backend...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "local" backend to the
  newly configured "s3" backend. No existing state was found in the newly
  configured "s3" backend. Do you want to copy this state to the new "s3"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes
```

```
ubuntu@ip-172-31-33-105:~/test$ terraform apply --var-file="terraform-prod.tfvars"
```

```
ubuntu@ip-172-31-33-105:~/test$ terraform init --var-file="terraform-prod.tfvars"
Initializing the backend...
```



Terraform state file stored in this bucket. So only a certain number can modify or edit this state file. They can. They will have permission to modify this state file.

Output.tf :

```
ubuntu@ip-172-31-33-105:~/test$ vi output.tf
```

```
output "instance_id" {
        description = " The ID of EC2 instance"
        value = aws_instance.demo.id
}

output "public_ip" {
        description = "The public ip"
        value = aws_instance.demo.public_ip
}
```

That demo [block name] name already gave in main.tf

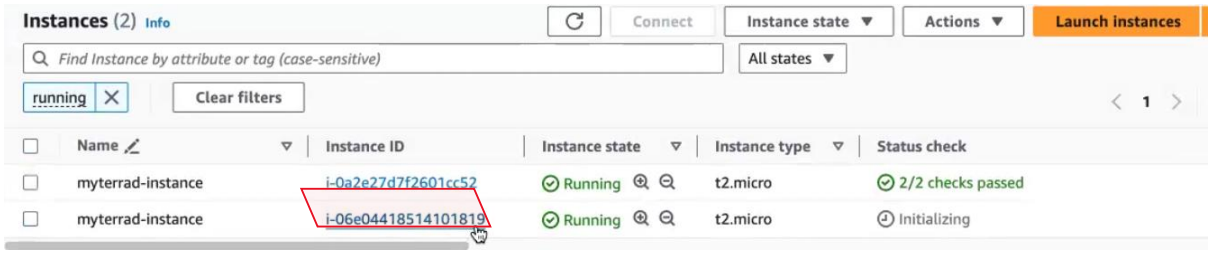
```
provider "aws" {
        region = var.region
}

resource "aws_instance" "demo" {
        ami = var.ami_id
        instance_type = var.instance_type

        tags = {
                Name = "myterrad-instance"
}
}
```

```
ubuntu@ip-172-31-33-105:~/test$ terraform init --var-file="terraform-dev.tfvars"
```

```
ubuntu@ip-172-31-33-105:~/test$ terraform plan --var-file="terraform-dev.tfvars"
```

```
ubuntu@ip-172-31-33-105:~/test$ terraform apply --var-file="terraform-dev.tfvars"
```

```
Changes to Outputs:
  + instance_id = (known after apply)
  + public_ip   = (known after apply)

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.demo: Creating...
aws_instance.demo: Still creating... [10s elapsed]
aws_instance.demo: Still creating... [20s elapsed]
aws_instance.demo: Still creating... [30s elapsed]
aws_instance.demo: Creation complete after 35s [id=i-06e04418514101819]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

instance_id = "i-06e04418514101819"
public_ip = "18.219.40.1"
ubuntu@ip-172-31-33-105:~/test$
```



```
ubuntu@ip-172-31-33-105:~/test$ ls
backend.tf  output.tf             terraform-prod.tfvars  terraform.tfstate         variables.tf
main.tf     terraform-dev.tfvars  terraform-qa.tfvars    terraform.tfstate.backup
ubuntu@ip-172-31-33-105:~/test$ ls -lrt
total 36
-rw-rw-r-- 1 ubuntu ubuntu  182 Jul 11 11:38 main.tf
-rw-rw-r-- 1 ubuntu ubuntu   48 Jul 11 11:46 terraform-qa.tfvars
-rw-rw-r-- 1 ubuntu ubuntu   81 Jul 11 11:53 terraform-dev.tfvars
-rw-rw-r-- 1 ubuntu ubuntu   82 Jul 11 11:56 terraform-prod.tfvars
-rw-rw-r-- 1 ubuntu ubuntu  316 Jul 11 11:59 variables.tf
-rw-rw-r-- 1 ubuntu ubuntu  133 Jul 11 12:04 backend.tf
-rw-rw-r-- 1 ubuntu ubuntu 4867 Jul 11 12:04 terraform.tfstate.backup
-rw-rw-r-- 1 ubuntu ubuntu    0 Jul 11 12:04 terraform.tfstate
-rw-rw-r-- 1 ubuntu ubuntu  188 Jul 11 12:12 output.tf
ubuntu@ip-172-31-33-105:~/test$
```

even though I have removed my state file. I have everything in my remote okay, my state file is in my S 3 bucket.

```
ubuntu@ip-172-31-33-105:~/test$ rm -rf *.tfstate
ubuntu@ip-172-31-33-105:~/test$ ls -lrt
total 36
-rw-rw-r-- 1 ubuntu ubuntu  182 Jul 11 11:38 main.tf
-rw-rw-r-- 1 ubuntu ubuntu   48 Jul 11 11:46 terraform-qa.tfvars
-rw-rw-r-- 1 ubuntu ubuntu   81 Jul 11 11:53 terraform-dev.tfvars
-rw-rw-r-- 1 ubuntu ubuntu   82 Jul 11 11:56 terraform-prod.tfvars
-rw-rw-r-- 1 ubuntu ubuntu  316 Jul 11 11:59 variables.tf
-rw-rw-r-- 1 ubuntu ubuntu  133 Jul 11 12:04 backend.tf
-rw-rw-r-- 1 ubuntu ubuntu 4867 Jul 11 12:04 terraform.tfstate.backup
-rw-rw-r-- 1 ubuntu ubuntu  188 Jul 11 12:12 output.tf
ubuntu@ip-172-31-33-105:~/test$ rm -rf *.tfstate.backup
ubuntu@ip-172-31-33-105:~/test$ ls -lrt
total 28
-rw-rw-r-- 1 ubuntu ubuntu 182 Jul 11 11:38 main.tf
-rw-rw-r-- 1 ubuntu ubuntu  48 Jul 11 11:46 terraform-qa.tfvars
-rw-rw-r-- 1 ubuntu ubuntu  81 Jul 11 11:53 terraform-dev.tfvars
-rw-rw-r-- 1 ubuntu ubuntu  82 Jul 11 11:56 terraform-prod.tfvars
-rw-rw-r-- 1 ubuntu ubuntu 316 Jul 11 11:59 variables.tf
-rw-rw-r-- 1 ubuntu ubuntu 133 Jul 11 12:04 backend.tf
-rw-rw-r-- 1 ubuntu ubuntu 188 Jul 11 12:12 output.tf
```

I give terraform plan, or I do the updating here in my terraform file. It will 1st go and refer my S 3 bucket state file, and then it will start creating the environment. Okay, so now you can see, I create a main.tf, so inside the main.tf. Here, you give all your resource block.

how to launch a Vpc Via terraform:

we will see how to create, how to launch a Vpc Via terraform. So always in the real time we do it from the Vpc. Only we start by launching our Vpc, and then only we start creating the resources inside the Vpc.

I've created a folder, and inside this folder we are going to write to the main.tf file.

```
ubuntu@ip-172-31-33-105:~$ mkdir vpc
ubuntu@ip-172-31-33-105:~$ cd vpc/
ubuntu@ip-172-31-33-105:~/vpc$ vi main.tf
```

So I don't want default Vpc, since I already have my default Vpc. I'm going to create a custom vpc.

[Docs overview | hashicorp/aws | Terraform | Terraform Registry]

1. Vpc
2. Subnets

aws_network_interface_attachment

aws_network_interface_sg_attachment

aws_route

aws_route_table

aws_route_table_association

aws_security_group

aws_security_group_rule

aws_subnet

```
resource "aws_subnet" "main" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "Main"
  }
}
```

In the above link that ,copy the syntax.

```
provider "aws" {
  region = var.aws_region
}


resource "aws_vpc" "myvpc" {
  cidr_block       = "10.0.0.0/16"


  tags = {
    Name = "MyVPC"
  }
}

resource "aws_subnet" "public_subnet" {
  vpc_id     = aws_vpc.myvpc.id
  cidr_block = var.public_subnet_cidr_block
  availability_zone = "ap-south-1a"
  tags = {
    Name = "PublicSubnet"
  }
}
```

Day 2 - Iac - Terraform (DO14WD-E

For this refer this class

//creating a VPC

```
resource "aws_vpc" "rtp03-vpc" {
   cidr_block = "10.1.0.0/16"
   tags = {
     Name = "rpt03-vpc"
   }

}
```

// Creating a Subnet

```
resource "aws_subnet" "rtp03-public_subent_01" {
   vpc_id = "${aws_vpc.rtp03-vpc.id}"
   cidr_block = "10.1.1.0/24"
   map_public_ip_on_launch = "true"
```

```
    availability_zone = "us-east-2a"
    tags = {
      Name = "rtp03-public_subent_01"
    }

}
```

//**Creating a Internet Gateway**

```
resource "aws_internet_gateway" "rtp03-igw" {
    vpc_id = "${aws_vpc.rtp03-vpc.id}"
    tags = {
      Name = "rtp03-igw"
    }
}
```

// **Create a route table**

```
resource "aws_route_table" "rtp03-public-rt" {
    vpc_id = "${aws_vpc.rtp03-vpc.id}"
    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = "${aws_internet_gateway.rtp03-igw.id}"
    }
    tags = {
      Name = "rtp03-public-rt"
    }
}
```

So I'm installing in the public subnet. I'm creating an Ec 2 in that Ec 2, I'm launching my user data. I'm launching my nginix application. So let me save this.

```
# Associate Public Subnet with Public Route Table
resource "aws_route_table_association" "public_subnet_association" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_route_table.id
}

# Elastic IP for NAT Gateway
resource "aws_eip" "my_eip" {
}

# NAT Gateway for Private Subnet
resource "aws_nat_gateway" "my_nat_gateway" {
  allocation_id = aws_eip.my_eip.id
  subnet_id     = aws_subnet.public_subnet.id

  tags = {
    Name = "MyNATGateway"
  }
}
resource "aws_instance" "my_ec2_instance" {
  ami           = "ami-0c2af51e265bd5e0e"
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.public_subnet.id
  user_data = <<EOF
              #!/bin/bash
              sudo apt-get update
              sudo apt-get install -y nginx
              sudo systemctl enable nginx
              sudo systemctl start nginx
-- INSERT --
```

```
ubuntu@ip-172-31-33-105:~/vpc$ vi variables.tf
```

```
variable "public_subnet_cidr_block" {
        description = "CIDR block for public subnet"
        default = "10.0.1.0/24"
}
variable "aws_region" {
        description = "AWS Region"
        default = "ap-south-1"
}
```

Give all terraform cmds

subnet, Internet gateway, public subnet, then the Internet gateway from a public subnet route table. then routable association, add gateway. So it is creating one by one.

Go & check in instance and vpc.everything is created

Class docu link:

https://github.com/yasminjeelani/sample-terrafom

https://docs.google.com/document/d/1sF167WT3fcjvUnOT23bZsKjQ2pOeSzLB8BeHmlAlnws/edit