In this lab, we will work with remote terraform state files.
We will first start with the local state and then migrate it to remote state with an S3 backend.
The configuration directory we will work with is /root/terraform-projects/RemoteState

First, create a simple configuration to create a local_file resource within the directory called RemoteState. The resource block should be created inside the main.tf file. Follow the below specifications for provisioning this resource:
Resource Name: state
filename: /root/<variable local-state>
content: "This configuration uses <variable local-state> state"
Use the variable called local-state in the variables.tf file which is already created for you and make use of variable interpolation syntax ${..}.
Once the configuration is ready, run a terraform init, plan and apply.

Variable.tf
```
variable remote-state {
   type = string
   default = "remote"
}
variable local-state {
   type = string
   default = "local"
}
```

Main.tf
```
resource "local_file" "state" {
   filename    = "/root/${var.local-state}"
   content = "This configuration uses ${var.local-state} state"
}
```

Let's now move the state to a remote S3 backend. For this, we will make use of an S3 compatible storage called minio.
Minio provides an S3-compatible API and allows us to configure the s3 backend in the same way as the actual S3 service in AWS Cloud.
To explore minio and the S3 bucket that has been created, click on the Minio Browser tab on the top of the terminal window..
Use the following credentials to login:
Access Key: foofoo
Secret Key: barbarbar

Before we add the configuration for the s3 backend, let's first change the local file resource. Change the variable used to remote-state instead of local-state.
Once done, run terraform plan and apply.

In a TF file, replace the variable local-state to remote-state only and then run terraform plan (optional) and terraform apply.

Main.tf

```
resource "local_file" "state" {
   filename    = "/root/${var.remote-state}"
   content = "This configuration uses ${var.remote-state} state"
}
```

Great! Now, let us configure the remote backend with s3. Add a terraform block in a new file called terraform.tf with the following arguments:
bucket: remote-state
key: terraform.tfstate
region: us-east-1
Do not run terraform init yet! Since we are making use of minio we also have to add a couple of additional arguments to get this to work!
We will do that in the next step. When using the regular s3 service from AWS the above arguments should be sufficient to configure remote state.

Terraform.tf

```
terraform {
  backend "s3" {
    key = "terraform.tfstate"
    region = "us-east-1"
    bucket = "remote-state"

  }
}
```

To make the s3 backend with Minio to work, we have to add a few additional arguments. The terraform.tf file has been updated. Check it out.
Please note that these arguments are optional and not needed when working with the regular S3 service in AWS.

Terraform.tf

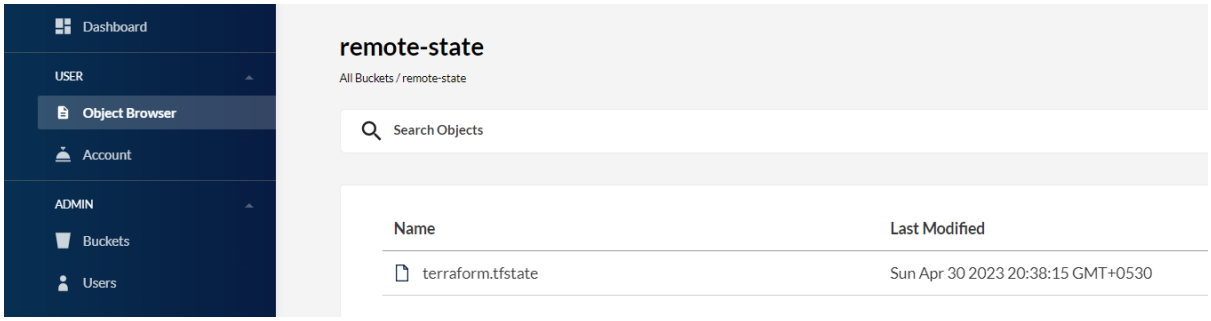
```
terraform {
  backend "s3" {
    key = "terraform.tfstate"
    region = "us-east-1"
    bucket = "remote-state"
    endpoint = "http://172.16.238.105:9000"
    force_path_style = true


    skip_credentials_validation = true

    skip_metadata_api_check = true
    skip_region_validation = true
  }
}
```

Run terraform init in our configuration directory now.

Once done you can proceed to delete the terraform.tfstate file from the local directory.

From the Minio Browser, you should now be able to see the state file uploaded to the bucket called remote-state.
Login credentials -
Access Key: foofoo
Secret Key: barbarbar

Run: terraform init and then delete the terraform.tfstate file by running rm terraform.tfstate command.



Which command would you use to show the attributes of the resource called classics stored in the terraform state?

Main.tf

```
resource "local_file" "top10" {
   filename = "/root/anime/top10.txt"
   content  = "1. Naruto\n2. DragonBallZ\n3. Death Note\nFullmetal Alchemist\nOne-Punch Man\n"

}
resource "local_file" "hall_of_fame" {
 filename = "/root/anime/hall-of-fame.txt"
 content = "1.Attack On Titan\n2. Naruto\n3. Bleach\n"

}
resource "local_file" "new_shows" {
 filename = "/root/anime/new_shows.txt"
 content = "1. Cannon Busters\n2. Last Hope\n3. Lost Song\n"

}
resource "local_file" "classics" {
 filename = "/root/anime/classic_shows.txt"
 content = "1. DragonBall\n"
}
```

Run terraform state show local_file.classics

```
iac-server $ terraform state show local_file.classics
# local_file.classics:
resource "local_file" "classics" {
    content              = <<~EOT
        1. DragonBall
    EOT
    content_base64sha256 = "6Ity8EEWB9hY2pJUjJQsdyBi7iDtrqnHg7E0VR9KS4A="
    content_base64sha512 = "lRKrxM4reT5okTZxIy6k/HdgLiXIJ+L1LIr2FUWcLldv
="
    content_md5          = "13d46e58bee23e8d0560d9cf3cef8966"
    content_sha1         = "69f539876d8db4e6873466ab5b4d56ebf32667b2"
    content_sha256       = "e88b72f0411607d858da92548c942c772062ee20edae
    content_sha512       = "9512abc4ce2b793e68913671232ea4fc77602e25c827
3b49f2dd897ac3a0beefd05d4f2dd32c71b41b97e"
    directory_permission = "0777"
    file_permission      = "0777"
    filename             = "/root/anime/classic_shows.txt"
    id                   = "69f539876d8db4e6873466ab5b4d56ebf32667b2"
}
```

We no longer wish to manage the file located at /root/anime/hall-of-fame.txt by Terraform. Remove the resource responsible for this file completely from the management of terraform.

Remove the resource block called hall_of_fame from the main.tf and also remove it from the state file by running terraform state rm local_file.hall_of_fame.

```
iac-server $ terraform state rm local_file.hall_of_fame
Removed local_file.hall_of_fame
Successfully removed 1 resource instance(s).
iac-server $
```

Main.tf

```
resource "local_file" "top10" {
   filename = "/root/anime/top10.txt"
   content  = "1. Naruto\n2. DragonBallZ\n3. Death Note\nFullmetal Alchemist\nOne-Punch Man\n"

}
resource "local_file" "new_shows" {
  filename = "/root/anime/new_shows.txt"
  content = "1. Cannon Busters\n2. Last Hope\n3. Lost Song\n"

}
resource "local_file" "classics" {
  filename = "/root/anime/classic_shows.txt"
  content = "1. DragonBall\n"

}
```

Within this configuration the terraform state commands are working (Try it!) but there is no terraform.tfstate file present!

What is the reason for this behavior? We are making use of a remote state in this configuration. Check out the remote s3 backend configuration in the terraform.tf file , we are using remote state .

What is the id of the random_pet resource called super_pet_2 in the state file?
Run terraform state show random_pet.super_pet_2 and inspect the id.

```
iac-server $ terraform state show random_pet.super_pet_2
# random_pet.super_pet_2:
resource "random_pet" "super_pet_2" {
    id        = "Wonder-funky-dane"
    length    = 2
    prefix    = "Wonder"
    separator = "-"
}
```

Rename the resource from super_pet_1 to ultra_pet.
Change the name in the main.tf file as well as the state.

Main.tf

```
resource "random_pet" "ultra_pet" {
   length = var.length1
   prefix = var.prefix1
}
resource "random_pet" "super_pet_2" {
   length = var.length2
   prefix = var.prefix2
}
```

Terraform.tf

```
terraform {
  backend "s3" {
    key = "terraform.tfstate"
    region = "us-east-1"
    bucket = "remote-state"
    endpoint = "http://172.16.238.105:9000"
    force_path_style = true


    skip_credentials_validation = true

    skip_metadata_api_check = true
    skip_region_validation = true
  }
}
```

Variables.tf

```
variable "length1" {
   default = "1"
}
variable "length2" {
   default = "2"
}
variable "prefix1" {
   default = "Super"
```

```
}
variable "prefix2" {
   default = "Wonder"
}
```

```
iac-server $ terraform state mv random_pet.super_pet_1 random_pet.ultra_pet
Move "random_pet.super_pet_1" to "random_pet.ultra_pet"
Successfully moved 1 object(s).
```

AWS ec2 and provisioners .

Navigate to the directory /root/terraform-projects/project-cerberus. We have an empty main.tf file in this directory.
Using this configuration file write a resource block to provision a simple EC2 instance with the following specifications:
Resource Name: cerberus
AMI: ami-06178cf087598769c, use variable named ami
region: eu-west-2, use variable named region
Instance Type: m5.large, use variable named instance_type
Once ready, run terraform init, plan and apply to provision this EC2 instance.

Main.tf

```
variable "ami" {
   default =   "ami-06178cf087598769c"
}
variable "instance_type" {
   default = "m5.large"
}
variable "region" {
   default = "eu-west-2"
}
resource "aws_instance" "cerberus" {
   ami =   var.ami
   instance_type   = var.instance_type
}
```

Provider.tf

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.15.0"
    }
  }
}
provider "aws" {
  region                  = var.region
  skip_credentials_validation = true
  skip_requesting_account_id  = true

  endpoints {
    ec2 = "http://aws:4566"
  }
}
```

Perfect! The instance has been created by terraform. To inspect the details of this instance, you can run terraform show command from the configuration directory.
This will print the resource attributes from the state file in a human-readable format.

```
iac-server $ terraform show
# aws_instance.cerberus:
resource "aws_instance" "cerberus" {
    ami                          = "ami-06178cf087598769c"
    arn                          = "arn:aws:ec2:eu-west-2::insta
    associate_public_ip_address  = true
    availability_zone            = "eu-west-2a"
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-343117b0df8eef6ec"
    instance_state               = "running"
    instance_type                = "m5.large"
    ipv6_address_count           = 0
    ipv6_addresses               = []
    key_name                     = "None"
    monitoring                   = false
    primary_network_interface_id = "eni-a5e81f62"
    private_dns                  = "ip-10-37-30-31.eu-west-2.cor
    private_ip                   = "10.37.30.31"
    public_dns                   = "ec2-54-214-58-220.eu-west-2.
    public_ip                    = "54.214.58.220"
    secondary_private_ips        = []
    security_groups              = []
    source_dest_check            = true
    subnet_id                    = "subnet-805b448f"
    tags_all                     = {}
    tenancy                      = "default"
    user_data_replace_on_change  = false
    vpc_security_group_ids       = []

    root_block_device {
        delete_on_termination = true
        device_name           = "/dev/sda1"
        encrypted             = false
```

The AMI ID we have used is an RHEL 8 image in the London region that only accepts SSH-Key based authentication. However, when we created the instance, we did not make use of a key!
Let's create a new key-pair!

A new SSH key pair has been created in the directory /root/terraform-projects/project-cerberus/.ssh.
The private key is called cerberus and the public key is called cerberus.pub
Using the public key, create a new key-pair in AWS with the following specifications:
Resource Name: cerberus-key
key_name: cerberus
Use the file functions to read the public key cerberus.pub
When ready, run a terrafom plan and apply to create this key pair.
If unsure, refer to the documentation to create a key-pair. Documentation tab is available at the top right.

Let us now configure the cerberus resource to make use of this key. Update the resource block to make use of the key called cerberus.

Once the configuration is updated, run a terraform plan and terraform apply. This will trigger the replacement of the instance with the new one having the key-pair created in our previous step.

Inside the resource block for the ec2 instance add an argument called key_name with the value cerberus.
Or use a variable with the same value.

Main.tf
```
variable "ami" {
  default = "ami-06178cf087598769c"
}

variable "instance_type" {
  default = "m5.large"
}

variable "region" {
  default = "eu-west-2"
}

resource "aws_instance" "cerberus" {
  ami           = var.ami
  instance_type = var.instance_type
  key_name  = "cerberus"
}
#You can also use variable for key_name
resource "aws_key_pair" "cerberus-key" {
    key_name = "cerberus"
    public_key = file(".ssh/cerberus.pub")
}
```

Let us now install nginx with EC2 instance. To do this, let's make use of the user_data argument.
Using the file function again or by making use of the heredoc syntax, use the script called install-nginx.sh as the value for the user_data argument.
Do not run terraform apply yet!

Main.tf
```
variable "ami" {
  default = "ami-06178cf087598769c"
}

variable "instance_type" {
  default = "m5.large"
}

variable "region" {
  default = "eu-west-2"
}

resource "aws_instance" "cerberus" {
  ami           = var.ami
  instance_type = var.instance_type
  key_name  = "cerberus"
  user_data = file("./install-nginx.sh")
}
```

```
#You can also use variable for key_name
resource "aws_key_pair" "cerberus-key" {
   key_name = "cerberus"
   public_key = file(".ssh/cerberus.pub")
}
```

Install-nginx.sh

```
#!/bin/bash
sudo yum update -y
sudo yum install nginx -y
sudo systemctl start nginx
```

In this case, an instance will be modified, but nginx will not be installed. It is due to the fact that User data scripts only run at first boot whereas the instance modification causes a reboot.
Let's apply the updated configuration in the next step!

provisioners should be added inside the resource block. Nested block inside resource block .
Which of the following provisioners does not need a connection block defined? local-exec provisioner does not need a connection block as it does not connect to a remote instance to run tasks.

What is the public IPv4 address that has been allocated to this EC2 instance?
Inspect the output of terraform show and lookup the value for public_ip.

We use the public IPv4 address to access this server. However, when this server is rebooted or recreated, this IP address would change.
To fix this, let's create an Elastic IP Address.
An Elastic IP address is a static IPv4 address which does not change over time.
Create an Elastic IP resource with the following specifications:
Resource Name: eip
vpc: true
instance: id of the EC2 instance created for resource cerberus (use a reference expression)
create a local-exec provisioner for the eip resource and use it to print the attribute called public_dns to a file /root/cerberus_public_dns.txt on the iac-server.
If unsure, refer to the documentation. Documentation tab is available at the top right.

Main.tf

```
variable "ami" {
  default = "ami-06178cf087598769c"
}

variable "instance_type" {
  default = "m5.large"
}

variable "region" {
  default = "eu-west-2"
}

resource "aws_instance" "cerberus" {
  ami           = var.ami
  instance_type = var.instance_type
  key_name  = "cerberus"
  user_data = file("./install-nginx.sh")
}
#You can also use variable for key_name
resource "aws_key_pair" "cerberus-key" {
   key_name = "cerberus"
```

```
  public_key = file(".ssh/cerberus.pub")
}
resource "aws_eip" "eip" {
  vpc = true
  instance   =  aws_instance.cerberus.id
  provisioner "local-exec" {
    command = "echo ${aws_eip.eip.public_dns} >> /root/cerberus_public_dns.txt"
  }
}
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
iac-server $ terraform show
# aws_eip.eip:
resource "aws_eip" "eip" {
    allocation_id     = "eipalloc-9d4846fe"
    association_id    = "eipassoc-8fbf1d0b"
    domain            = "vpc"
    id                = "eipalloc-9d4846fe"
    instance          = "i-13b96427391ab6c4f"
    network_interface = "eni-457dabc9"
    public_dns        = "ec2-127-186-244-199.eu-west-2.compute.amazonaws.com"
    public_ip         = "127.186.244.199"
    tags_all          = {}
    vpc               = true
}

# aws_instance.cerberus:
resource "aws_instance" "cerberus" {
    ami                          = "ami-06178cf087598769c"
    arn                          = "arn:aws:ec2:eu-west-2::instance/i-13b96427391ab6c4f"
    associate_public_ip_address  = true
    availability_zone            = "eu-west-2a"
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-13b96427391ab6c4f"
    instance_state               = "running"
    instance_type                = "m5.large"
    ipv6_address_count           = 0
    ipv6_addresses               = []
    key_name                     = "cerberus"
```

In the current configuration, which dependency is NOT true?
The Elastic IP resource called eip has a reference expression pointing to the AWS EC2 resource called cerberus. Hence the resource eip depends on cerberus and not the other way around.

Taint and debugging :

Which environment variable should be used to export the logs to a specific path?
Use the variable TF_LOG_PATH.

Can you export the debug logs from terraform just by setting TF_LOG_PATH environment variable and providing a path as the value to this variable?
You also need to export the variable TF_LOG and set it to one of the log levels.

We have a configuration directory called /root/terraform-projects/ProjectA. Enable logging with the log level set to ERROR and then export the logs the path /tmp/ProjectA.log.

Once the environment variables are set, run a terraform init and apply.
It's OK if this results in an error. Do not change any configuration files before you export the logs!

Which Log Level provides the most details when you run terraform commands?
TRACE provides the most verbose logs.

Main.tf
```
resource "aws_instance" "ProjectA" {
   ami = "ami-0c9bfc21ac5bf10eb"
   instance_type = "t2.large"
}
```

Provider.tf
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.15.0"
    }
  }
}

provider "aws" {
  region                  = "ca-central-1"
  skip_credentials_validation = true
  skip_requesting_account_id  = true

  endpoints {
    iam                   = "http://aws:4566"
  }
}
```

Main.tf
```
resource "aws_instance" "ProjectB" {
   ami = "ami-0c9bfc21ac5bf10eb"
   instance_type = "t2.large"
   tags = {
      Name = "projectb_webserver"
      Description = "Oversized Webserver"
   }

}
```

Provider.tf
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.15.0"
    }
  }
}

provider "aws" {
  region                  = "ca-central-1"
  skip_credentials_validation = true
```

```
  skip_requesting_account_id  = true

  endpoints {
    ec2                 = "http://aws:4566"
  }
}
```

```
iac-server $ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_instance.ProjectB: Refreshing state... [id=i-91349edaa12071662]

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_instance.ProjectB is tainted, so must be replaced
-/+ resource "aws_instance" "ProjectB" {
        ami                                  = "ami-0c9bfc21ac5bf10eb"
      ~ arn                                  = "arn:aws:ec2:ca-central-1::instance/i-9
      ~ associate_public_ip_address          = true -> (known after apply)
      ~ availability_zone                    = "ca-central-1a" -> (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      ~ disable_api_termination              = false -> (known after apply)
      ~ ebs_optimized                        = false -> (known after apply)
        get_password_data                    = false
      + host_id                              = (known after apply)
      ~ id                                   = "i-91349edaa12071662" -> (known after a
      + instance_initiated_shutdown_behavior = (known after apply)
      ~ instance_state                       = "running" -> (known after apply)
        instance_type                        = "t2.large"
      ~ ipv6_address_count                   = 0 -> (known after apply)
```

Untaint the resource called ProjectB so that the resource is not replaced any more.
The resource is currently tainted.

```
iac-server $ terraform untaint aws_instance.ProjectB
Resource instance aws_instance.ProjectB has been successfully untainted.
iac-server $
```

Terraform import :

Navigate to the directory /root/terraform-projects/project-jade. We have a few resources created using the configuration files.
Inspect them first.

Main.tf

```
resource "aws_instance" "ruby" {
  ami          = var.ami
```

```
  instance_type = var.instance_type
  for_each      = var.name
  key_name      = var.key_name
  tags = {
    Name = each.value
  }
}
output "instances" {
  value = aws_instance.ruby
}
```

Provider.tf

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "4.15.0"
    }
  }
}

provider "aws" {
  region                      = "us-east-1"
  skip_credentials_validation = true
  skip_requesting_account_id  = true

  endpoints {
    ec2 = "http://aws:4566"
  }
}
```

Variables.tf

```
variable "name" {
  type    = set(string)
  default = ["jade-webserver", "jade-lbr", "jade-app1", "jade-agent", "jade-app2"]

}
variable "ami" {
  default = "ami-0c9bfc21ac5bf10eb"
}
variable "instance_type" {
  default = "t2.nano"
}
variable "key_name" {
  default = "jade"

}
```