Lab: Terraform State Default Local Backend

In order to correctly manage your infrastructure resources, Terraform stores and operates on the state of your managed infrastructure. Terraform uses this state on each execution to plan and make changes. This state must be stored and maintained on each execution so future operations can be performed correctly. The location and method of operation of Terraform's state is determined by the Terraform `backend`. By default Terraform uses a `local` backend, where state information is stored and acted upon locally within the working directory in a local file named `terraform.tfstate`.

- Task 1: Show current state (CLI Operation)
- Task 2: Show state file location
- Task 3: View/Update Terraform local backend configuration
- Task 4: Modify, Plan and Execute Changes
- Task 5: Show New State and State Backup

Task 1: Show Current State via CLI

On your workstation, navigate to the /workstation/terraform directory. To view the applied configuration utilize the `terraform show` command to view the resources created and find the IP address for your instance.

```
terraform show
```

> Note: If this command doesn't yield any information then you will need to redeploy your VPC infrastructure following the steps in Objective 1b via a `terraform apply`.

Alternatively you can run a `terraform state list` to list all of the items in Terraform's managed state.

```
terraform state list
```

```
data.aws_ami.ubuntu
data.aws_availability_zones.available
data.aws_region.current
aws_eip.nat_gateway_eip
aws_instance.ubuntu_server
aws_internet_gateway.internet_gateway
aws_key_pair.generated
aws_nat_gateway.nat_gateway
aws_route_table.private_route_table
```


```
aws_route_table.public_route_table
aws_route_table_association.private["private_subnet_1"]
aws_route_table_association.private["private_subnet_2"]
aws_route_table_association.private["private_subnet_3"]
aws_route_table_association.public["public_subnet_1"]
aws_route_table_association.public["public_subnet_2"]
aws_route_table_association.public["public_subnet_3"]
aws_security_group.ingress-ssh
aws_security_group.vpc-ping
aws_security_group.vpc-web
aws_subnet.private_subnets["private_subnet_1"]
aws_subnet.private_subnets["private_subnet_2"]
aws_subnet.private_subnets["private_subnet_3"]
aws_subnet.public_subnets["public_subnet_1"]
aws_subnet.public_subnets["public_subnet_2"]
aws_subnet.public_subnets["public_subnet_3"]
aws_vpc.vpc
local_file.private_key_pem
random_string.random
tls_private_key.generated
```

Task 2: Show state file location

The next logical question is “where is state stored”? By default, Terraform stores state locally in a JSON file on disk, but it also supports a number of state backends for storing “remote state”. If you are still learning how to use Terraform, we recommend using the default `local` backend, which requires no configuration.

Terraform’s `local` state is stored on disk as JSON, and that file must always be up to date before a person or process runs Terraform. If the state is out of sync, the wrong operation might occur, causing unexpected results.

By default terraform uses a `local` backend and saves its state file to a `terraform.tfstate` file located in the working directory. You can validate that your state file lives in the current directory by looking for the presence of a `terraform.tfstate` file. You may also see a backup that was created for this state file

```
|-- main.tf
|-- MyAWSKey.pem
|-- terraform.tf
|-- terraform.tfstate
|-- terraform.tfstate.backup
|-- terraform.tfstate.d
```


```
|-- |-- development
      |-- terraform.tfstate
      |-- terraform.tfstate.backup
|-- variables.tf
```

If you open and view the `terraform.tfstate` file you will notice that it is stored in a `json` format. As discussed in previous labs it is never a good idea to modify the contents of this file directly, but rather use the standard Terraform workflow and CLI state options available for advanced Terraform state management.

Task 3: View/Update Terraform local backend configuration

The terrafrom backend end configuration for a given working directory is specified in the Terraform configuration block. Our terraform configuaration block for this lab is located in the `terraform.tf` file.

```
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
    http = {
      source  = "hashicorp/http"
      version = "2.1.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.1.0"
    }
    local = {
      source  = "hashicorp/local"
      version = "2.1.0"
    }
    tls = {
      source  = "hashicorp/tls"
      version = "3.1.0"
    }
  }
}
```

By default there is no `backend` configuration block within our configuration. Because no `backend` was included in our configuration Terraform will use it's default backend - `local` This is why we see

the `terraform.tfstate` file in our working directory. If we want to be explicit about which backend Terraform should use it would cause no harm to add the following to our Terraform configuration block within the `terraform.tfstate` file

```
terraform {
  backend "local" {
    path = "terraform.tfstate"
  }
}
```

This is not required and generally not performed as it is the Terraform defalt backend that terraform uses.

> Note: Be sure not to copy just the `backend` block above and not the full `terraform block` You can validate the syntax is correct by issuing a `terraform validate`

Anytime a new piece of configuration is added to a Terraform configuration block the working directory must be re-initialized.

```
terraform init
```

Task 4: Modify, Plan and Execute Changes

During execution, Terraform will examine the state of the currently running infrastructure, determine what differences exist between the current state and the revised desired state, and indicate the necessary changes that must be applied. When approved to proceed, only the necessary changes will be applied, leaving existing, valid infrastructure untouched. Terraform can perform in-place updates after changes are made to the `main.tf` configuration file. Update your `main.tf` to include an second EC2 instance in the public subnet 2:

Append the following code to `main.tf`

```
# Terraform Resource Block - To Build EC2 instance in Public Subnet
resource "aws_instance" "web_server_2" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.public_subnets["public_subnet_2"].id
  tags = {
    Name = "Web EC2 Server"
  }
}
```


Save the configuration. Plan and apply the changes you just made and note the output differences for additions, deletions, and in-place changes.

Run a `terraform plan` to see the updates that Terraform needs to make.

```
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.web_server will be created
  + resource "aws_instance" "web_server_2" {
      + ami                                  = "ami-083654bd07b5da81d"
      + 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_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (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)
      + 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
      + subnet_id                            = "subnet-0ed3366fd5647c0e7"
      + tags                                 = {
          + "Name" = "Web EC2 Server"
```


```
    }
  + tags_all                                = {
      + "Name" = "Web EC2 Server"
    }
  + tenancy                                 = (known after apply)
  + user_data                               = (known after apply)
  + user_data_base64                        = (known after apply)
  + vpc_security_group_ids                  = (known after apply)

  + capacity_reservation_specification {
      + capacity_reservation_preference = (known after apply)

      + capacity_reservation_target {
          + capacity_reservation_id = (known after apply)
        }
    }

  + ebs_block_device {
      + delete_on_termination = (known after apply)
      + device_name           = (known after apply)
      + encrypted             = (known after apply)
      + iops                  = (known after apply)
      + kms_key_id            = (known after apply)
      + snapshot_id           = (known after apply)
      + tags                  = (known after apply)
      + throughput            = (known after apply)
      + volume_id             = (known after apply)
      + volume_size           = (known after apply)
      + volume_type           = (known after apply)
    }

  + enclave_options {
      + enabled = (known after apply)
    }

  + ephemeral_block_device {
      + device_name  = (known after apply)
      + no_device    = (known after apply)
      + virtual_name = (known after apply)
    }

  + metadata_options {
      + http_endpoint               = (known after apply)
      + http_put_response_hop_limit = (known after apply)
      + http_tokens                 = (known after apply)
    }

  + network_interface {
      + delete_on_termination = (known after apply)
```


```
          + device_index           = (known after apply)
          + network_interface_id   = (known after apply)
        }

      + root_block_device {
          + delete_on_termination = (known after apply)
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + tags                  = (known after apply)
          + throughput            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = (known after apply)
          + volume_type           = (known after apply)
        }
    }

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

Due to the stateful nature of Terraform, it only needs to add the one additional resource to our infrastructure build out.

Step 3.1.2

Run a `terraform apply` to execute the updates that Terraform needs to make.

```
terraform apply
```

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

When prompted to apply the changes, respond with `yes`.

```
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.web_server_2: Creating...
```


```
aws_instance.web_server_2: Still creating... [10s elapsed]
aws_instance.web_server_2: Still creating... [20s elapsed]
aws_instance.web_server_2: Still creating... [30s elapsed]
aws_instance.web_server_2: Creation complete after 33s [id=i-0331d9d6ff38214a3]

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

Task 5: Show New State and State Backup

To view the applied configuration utilize the `terraform state list` command to view the resources created. Look for the `aws_instance.web_server_2` which is now present within Terraform's managed state. You can then show the details of this resource by running a `terraform state show aws_instance.web_server_2`

```
terraform state list
terraform state show aws_instance.web_server_2
```

Terraform State example:

```
# aws_instance.web_server_2:
resource "aws_instance" "web_server" {
    ami                                  = "ami-036490d46656c4818"
    arn                                  = "arn:aws:ec2:us-east-1:508140242758:instance/
    associate_public_ip_address          = true
    availability_zone                    = "us-east-1b"
    cpu_core_count                       = 1
    cpu_threads_per_core                 = 1
    disable_api_termination              = false
    ebs_optimized                        = false
    get_password_data                    = false
    hibernation                          = false
    id                                   = "i-0d544e90777ca8c2f"
    instance_initiated_shutdown_behavior = "stop"
    instance_state                       = "running"
    instance_type                        = "t2.micro"
    ipv6_address_count                   = 0
    ipv6_addresses                       = []
    monitoring                           = false
    primary_network_interface_id         = "eni-0445ae3a8b38ae47a"
    private_dns                          = "ip-10-0-101-117.ec2.internal"
    private_ip                           = "10.0.101.117"
    public_ip                            = "18.234.248.120"
    secondary_private_ips                = []
    security_groups                      = []
    source_dest_check                    = true
    subnet_id                            = "subnet-0e3cbf2e577579360"
```


```
    tags                                   = {
        "Name" = "Ubuntu EC2 Server"
    }
    tags_all                               = {
        "Name" = "Ubuntu EC2 Server"
    }
    tenancy                                = "default"
    vpc_security_group_ids                 = [
        "sg-097b59a05720fb97c",
    ]

    capacity_reservation_specification {
        capacity_reservation_preference = "open"
    }

    credit_specification {
        cpu_credits = "standard"
    }

    enclave_options {
        enabled = false
    }

    metadata_options {
        http_endpoint               = "enabled"
        http_put_response_hop_limit = 1
        http_tokens                 = "optional"
    }

    root_block_device {
        delete_on_termination = true
        device_name           = "/dev/sda1"
        encrypted             = false
        iops                  = 100
        tags                  = {}
        throughput            = 0
        volume_id             = "vol-053758fb913734c4c"
        volume_size           = 8
        volume_type           = "gp2"
    }
}
```

Since the state of our configuration has changed, Terraform by default keeps a backup copy of the state in a `terraform.tfstate.backup` file.

```
|-- terraform.tfstate
|-- terraform.tfstate.backup
```