Lab: Benefits of State

Terraform State

In order to properly and correctly manage your infrastructure resources, Terraform stores the state of your managed infrastructure. Terraform uses this state on each execution to plan and make changes to your infrastructure. This state must be stored and maintained on each execution so future operations can perform correctly.

Benefits of State

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.

- Task 1: Show Current State
- Task 2: Update your Configuration
- Task 3: Plan and Execute Changes
- Task 4: Show New State

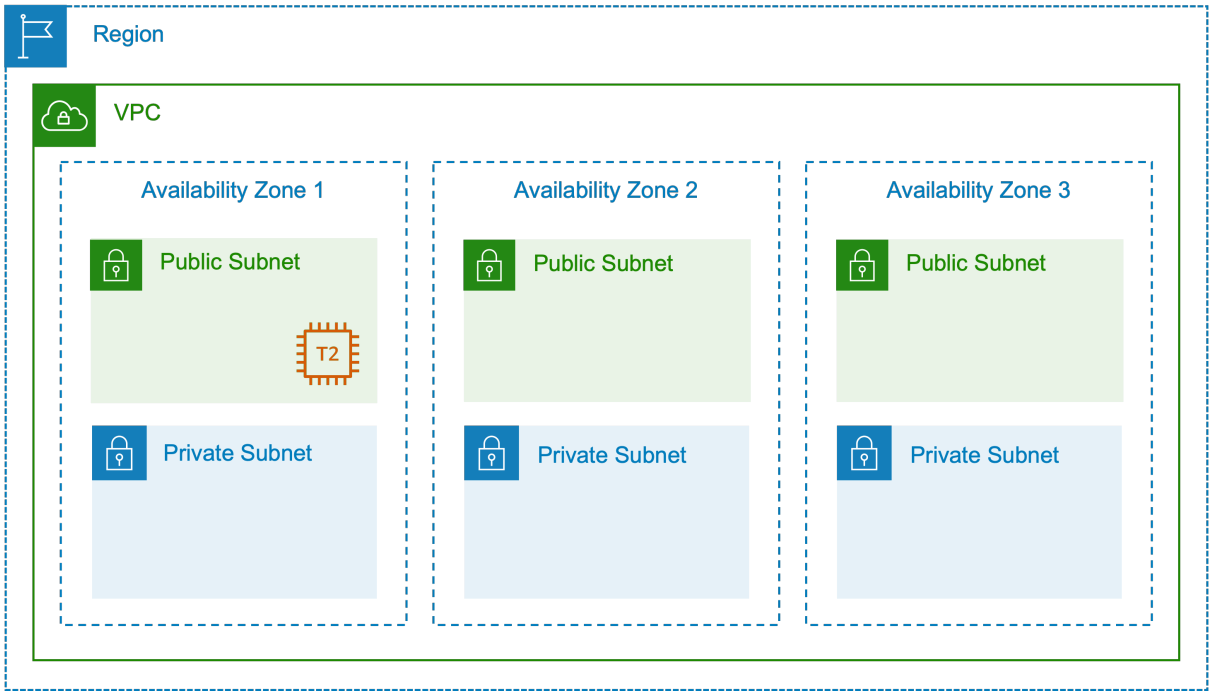

Figure 1: AWS Application Infrastructure Buildout

Task 1: Show Current State

In a previous lab, you wrote some Terraform configuration using the HashiCorp Configuration Language to create a new VPC in AWS. Now that that VPC exists we will build an EC2 instance in one of the public subnets. Since the VPC still exists the only change that Terraform needs to address is the addition of the EC2 instance.

Step 1.1.1

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.

> Note: If this command doesn't yield any information then you will need to redeploy your VPC infrastructure following the steps in Objective 1b. This directory should contain both a `main.tf` and `variables.tf` file.

```
terraform show
```

```
terraform show

# aws_eip.nat_gateway_eip:
resource "aws_eip" "nat_gateway_eip" {
    association_id       = "eipassoc-0f2986cc722254f2e"
    domain               = "vpc"
    id                   = "eipalloc-000eb0775a52a1e32"
    network_border_group = "us-east-1"
    network_interface    = "eni-0272f17827cbb823a"
    private_dns          = "ip-10-0-101-134.ec2.internal"
    private_ip           = "10.0.101.134"
    public_dns           = "ec2-54-205-175-114.compute-1.amazonaws.com"
    public_ip            = "54.205.175.114"
    public_ipv4_pool     = "amazon"
    tags                 = {
        "Name" = "demo_igw_eip"
    }
    tags_all             = {
        "Name" = "demo_igw_eip"
    }
    vpc                  = true
}

# aws_internet_gateway.internet_gateway:
resource "aws_internet_gateway" "internet_gateway" {
    arn      = "arn:aws:ec2:us-east-1:508140242758:internet-gateway/igw-0be99153cf7f3c6a
    id       = "igw-0be99153cf7f3c6ab"
    owner_id = "508140242758"
    tags     = {
        "Name" = "demo_igw"
    }
    tags_all = {
        "Name" = "demo_igw"
    }
    vpc_id   = "vpc-064a97911d85e16d4"
}

# aws_nat_gateway.nat_gateway:
resource "aws_nat_gateway" "nat_gateway" {
    allocation_id        = "eipalloc-000eb0775a52a1e32"
    connectivity_type    = "public"
    id                   = "nat-0705f0f6101212b3b"
    network_interface_id = "eni-0272f17827cbb823a"
    private_ip           = "10.0.101.134"
    public_ip            = "54.205.175.114"
```


```
...
```

Task 2: Update your Configuration to include EC2 instance

Terraform can perform in-place updates after changes are made to the `main.tf` configuration file. Update your `main.tf` to include an EC2 instance in the public subnet:

Append the following code to `main.tf`

```
# Terraform Data Block - To Lookup Latest Ubuntu 20.04 AMI Image
data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }

  owners = ["099720109477"]
}

# Terraform Resource Block - To Build EC2 instance in Public Subnet
resource "aws_instance" "web_server" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.public_subnets["public_subnet_1"].id
  tags = {
    Name = "Ubuntu EC2 Server"
  }
}
```

Save the configuration.

Task 3: Plan and Execute Changes

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

Step 3.1.1

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
following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.web_server will be created
  + resource "aws_instance" "web_server" {
      + ami                                  = "ami-036490d46656c4818"
      + 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"
 ...

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

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: Creating...
aws_instance.web_server: Still creating... [10s elapsed]
aws_instance.web_server: Still creating... [20s elapsed]
aws_instance.web_server: Still creating... [30s elapsed]
aws_instance.web_server: Still creating... [40s elapsed]
aws_instance.web_server: Still creating... [50s elapsed]
aws_instance.web_server: Creation complete after 59s [id=i-0d544e90777ca8c2f]

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

Task 3: Show New State

To view the applied configuration utilize the `terraform show` command to view the resources created. Look for the `aws_instance.web_server` which is now present within Terraform's managed state.

```
terraform show
```

Terraform State example:

```
# aws_instance.web_server:
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"
    }
}
```


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.web_server
aws_internet_gateway.internet_gateway
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_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
```