Labs of Terraform

Chapter 2 : Installing and Setting up Terraform

Lab 1 : Install terraform on linux

Step 1: Switch to /tmp directory.
 # cd /tmp/

Step 2: Download the terraform zip from the terraform website.
/tmp# wget https://releases.hashicorp.com/terraform/0.12.24/terraform_0.12.24_linux_amd64.zip

Step 3: unzip the downloaded file using the unzip command
/tmp$# unzip terraform_0.12.24_linux_amd64.zip

Step 4: Move the extracted binary to path location like */bin, /sbin etc.*
/tmp# sudo mv terraform /usr/local/bin

Step 5: Change the permission of the file to root user.
/tmp# sudo chown -R root:root /usr/local/bin/terraform

Step 6: Verify the installation
/tmp# cd
 # terraform version
Terraform v0.12.24

Congratulations on completing the lab.

Lab 2 : Setting/verifying up access and secret keys on the environment settings

Step 1: Create a credentical source file with following credentials access.

```
# cat awsrc
export AWS_ACCESS_KEY_ID=Your access_key
export AWS_SECRET_ACCESS_KEY=your secret key
```

Step 2: Verify the configuration setup in the environment setup.

```
# env | grep AWS
```

Lab 3: Deploy your first ec2 instance resource on aws

Step 1: Create the base directory to work with

```
# cd
# mkdir chapter1
# cd chapter1
```

Step 2: Create the template file. Output should match with following snippet.

```
# cat base.tf
provider "aws" {
  region = "ap-south-1"
}

resource "aws_instance"  "inst1" {
  ami = "ami-0470e33cd681b2476"
  instance_type = "t2.micro"
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Lab 4 : Deploy first aws ec2 with tag

Step 1: Modify the earlier created base file to match the following snippet

 # cat base.tf

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_instance"  "inst1" {
  ami = "ami-0470e33cd681b2476"
  instance_type = "t2.micro"
  tags = {
    Name = "My-ec2instance"
  }
}
```

Step2: Apply the changes using the terraform

 # terraform plan
 # terraform apply
 # terraform show

Chapter 3 : Terraform Variable

Lab 1 : Variable in the same file

Step 1: Create the directory to work on variable

```
# cd
# mkdir var-terra
# cd var-terra
```

Step2: Create the base.tf template file with variables defined in the same file. The content should match the following.

```
# cat newbase.tf

provider "aws" {
    region = "ap-south-1"
}

variable "ami" {
    default = "ami-0470e33cd681b2476"
}

variable "instance_type" {
    default = "t2.micro"
}

resource "aws_instance" "my-instance" {
    ami = "${var.ami}"
    instance_type = "${var.instance_type}"
    }
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Lab 2 : Define Variable in the seperate file

Step 1: Create a separate file for variable declaration. Content should match the following snnipet.

```
# cat vars.tf
variable "ami" {
    default = "ami-04169656fea786776"
}

variable "instance_type" {
    default = "t2.nano"
}
```

Step 1: Create the main template file and include the variable inside the same. Open the editor and put the content.

```
#vim newbase.tf
provider "aws" {
    region = "ap-south-1"
}

resource "aws_instance" "my-instance" {
    ami = "${var.ami}"
    instance_type = "${var.instance_type}"
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Chapter 4 : AWS Advance Infrastructure

Lab 1 : Deploy first aws ec2 with eip

Step 1: Create the new directory to work with.

```
# cd
# mkdir aws-advance
# cd aws-advance
```

Step 2 : Create the template with following content to deploy an EC2 instance with EIP.

```
# vim base.tf
```

```
provider "aws" {
region = "ap-south-1"
}

resource "aws_instance" "test-instance" {
    ami = "ami-0470e33cd681b2476"
    instance_type = "t2.micro"
}

resource "aws_eip" "test-ip" {
    instance = "${aws_instance.test-instance.id}"
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Lab 2: Deploy Basic VPC infrastructure.

Step 1: Create the new directory to work with.

```
# cd
# mkdir vpc-setup
# cd vpc-setup
```

Step 2: Create the template file for deploying vpc with following content

vim base.tf

```
provider "aws" {
region = "ap-south-1"
}
data "aws_availability_zones" "available" {}

resource "aws_vpc" "myVpc" {
  cidr_block = "10.20.0.0/16"
  enable_dns_hostnames = true
  tags {
    Name = "myVpc"
  }
}
resource "aws_subnet" "public_subnet" {
  count = "${length(data.aws_availability_zones.available.names)}"
  vpc_id = "${aws_vpc.myVpc.id}"
  cidr_block = "10.20.${10+count.index}.0/24"
  availability_zone = "$
{data.aws_availability_zones.available.names[count.index]}"
  map_public_ip_on_launch = true
  tags {
    Name = "PublicSubnet"
  }
}
resource "aws_subnet" "private_subnet" {
  count = "${length(data.aws_availability_zones.available.names)}"
  vpc_id = "${aws_vpc.myVpc.id}"
  cidr_block = "10.20.${20+count.index}.0/24"
  availability_zone= "$
{data.aws_availability_zones.available.names[count.index]}"
  map_public_ip_on_launch = false
  tags {
    Name = "PrivateSubnet"
  }
}
```

Lab 3: Deploy web server on ec2 instance on customized vpc.

Step 1: Create the directory to work with

 # cd
 # mkdir vpc-ec2
 # cd vpc-ec2

Step 2: Clone the following github repository to get the code.

 # git clone https://github.com/vsaini44/terraform-class.git
 # cd aws_advance/vpc
 # ls

Step 3: Check the content for the directory to verify all the files are present and available.

Step 4: Perform the terraform operation in sequence

 # terraform init
 # terraform validate
 # terraform plan
 # terraform apply
 # terraform show

Lab 4: Deploy RDS instance on customized vpc.

Step 1: Create the directory to work with

```
# cd
# mkdir vpc-rds
# cd vpc-rds
```

Step 2: Clone the following github repository to get the code.

```
# git clone https://github.com/vsaini44/terraform-class.git
# cd aws_advance/rds
# ls
```

Step 3: Check the content for the directory to verify all the files are present and available.

Step 4: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Lab 5: IAM user without any permission(policy)

Step 1: Create the directory to work with

```
# cd
# mkdir iam
# cd iam
```

Step 2: Create the template file with following configuration.

```
# vim base.tf
provider "aws" {
region = "ap-south-1"
}

resource "aws_iam_user" "iam1" {
name = "user1"
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Lab 6: IAM user with predefined permission(policy)

Step 1: Create the directory to work with

```
# cd
# mkdir iam
# cd iam
```

Step 2: Create the template file with following configuration.

```
# vim base.tf
provider "aws" {
region = "ap-south-1"
}

resource "aws_iam_user" "iam1" {
name = "user1"
}

resource "aws_iam_user_policy_attachment" "test-attach" {
user="${aws_iam_user.iam1.name}"
policy_arn="arn:aws:iam::aws:policy/AmazonEC2FullAccess"
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Lab 7: IAM Group, users and policy

Step 1: Create the directory to start the exercise

```
# cd
# mkdir iam-group
# cd iam-group
```

Step 2: Create the template with the following content

```
# vim base.tf
provider "aws" {
region = "ap-south-1"
}
resource "aws_iam_group_membership" "team" {
name = "tf-testing-group-membership"
users = [
"${aws_iam_user.user_one.name}",
"${aws_iam_user.user_two.name}",
]
group = "${aws_iam_group.group.name}"
}
resource "aws_iam_group" "group" {
name = "Developers"
}
resource "aws_iam_user" "user_one" {
name = "user1"
}
resource "aws_iam_user" "user_two" {
name = "user2"
}
resource "aws_iam_group_policy_attachment" "test-attach" {
group = "${aws_iam_group.group.name}"
policy_arn = "arn:aws:iam::aws:policy/AmazonEC2FullAccess"
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Lab 8: IAM Customized policy

Step 1: Create the directory to start the exercise

```
# cd
# mkdir iam-policy
# cd iam-policy
```

Step 2: Create the template with the following content

```
# vim base.tf
provider "aws" {
    region ="ap-south-1"
    }

resource "aws_iam_policy" "policy" {
    name= "test_policy"
    description = "My test policy"
    policy = <<EOF
        {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": "ec2:DescribeInstances",
                    "Resource": "*",
                    "Condition": {
                        "ForAllValues:StringEquals": {
                        "aws:RequestedRegion": "ap-south-1"
                        }
                    }
                }
            ]
        }
EOF
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Chapter 5 : Terraform States

Lab 1: Configuring remote terraform state

Step 1: Create the new directory to work into.

```
# cd
# mkdir terra-state
# cd terra-state
```

Step 2: Create the template file with the following content. The file will configure terraform to store the state file remotly on a S3 bucket. The same will be checked using by provisioning a basic ec2-instance.

```
# vim base.tf
provider "aws" {
  region = "ap-south-1"
}

terraform {
  backend "s3" {
    bucket = "terraform-mystate-vickydata"
    key    = "terraform.tfstate"
    region = "ap-south-1"
  }
}

resource "aws_instance"  "inst1" {
  ami = "ami-0470e33cd681b2476"
  instance_type = "t2.micro"
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Step 4: You can also go and check the s3 bucket from the management portal.

Lab 2: Configuring remote terraform state with locking

Step 1: Create the new directory to work into

```
# cd
# mkdir terralock
# cd terralock
```

Step 2: Create the template file to provision the dynamodb for locking the state file. The dynamodb resource getting created in the template file will be used for locking purpose of the state file, which will deny the concurrent runs on terraform plan.

```
# vim base.tf
provider "aws" {
  region = "ap-south-1"
}

resource "aws_dynamodb_table" "dynamodb-terraform-state-lock" {
  name = "terraform-state-lock-dynamo"
  hash_key = "LockID"
  read_capacity = 20
  write_capacity = 20
  attribute {
    name = "LockID"
    type = "S"
  }

  tags =  {
    Name = "DynamoDB Terraform State Lock Table"
      }
}
```

Step 3: Perform the terraform operation in sequence

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Step 4: Now that the dynamodb table is created, modify the template file to tell terraform to use s3 for remote state and dynamodb for state locking. Test the same by provisioning of ec2-instance and updating its parellely. Follow the steps your instructor is performing to test the setup.

```
# vim base.tf
provider "aws" {
  region = "ap-south-1"
}

terraform {
  backend "s3" {
    bucket="vickybultibucket"
    dynamodb_table = "terraform-state-lock-dynamo"
    key    ="terraform.tfstate”
    region="ap-south-1"
  }
}

resource "aws_instance"  "inst1" {
ami = "ami-0470e33cd681b2476"
  instance_type = "t2.micro"
}
```

Step 5: Apply the changes using terraform command.

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Chapter 6 : Terraform Modules

Lab 1: Basic terraform module lab

Step 1 : Create the new module directory to create the module

```
# cd
# mkdir “/tmp/module1”
# cd “/tmp/module1
```

Step 2 : Create the base.tf with simple instance launch template content

```
# vim base.tf
resource "aws_instance"  "inst1" {
ami = "ami-0470e33cd681b2476"
  instance_type = "t2.micro"
}
```

Step 3 : Now create another directory and call the module in the same

```
# cd
# mkdir mod-call
# cd mod-call
```

Step 4 : Create the base.tf to call the module in the same

```
# vim base.tf
module “vicky”
  source = “/tmp/module1”
}

provider “aws” {
  region = “ap-south-1”
}
```

Step 5 : Apply the changes using terraform command.

```
# terraform init
# terraform validate
# terraform plan
# terraform apply
# terraform show
```

Step 6 : After verification, delete the infrastructure

```
# terraform destroy
```

Lab 2: Module with variable input

Step 1 : Modify the Module template created in earlier lab with variable input

```
# cd “/tmp/module1”
# vim base.tf
resource "aws_instance"  "inst1" {
ami = "ami-0470e33cd681b2476"
  instance_type = "t2.micro"
  tags {
    Name = “${var.instance_name}”
    }
}

variable “instance_name” {}
```

Step 4 : Modify the base.tf in mod-call folder to pass the variable value in the program

```
# cd
# cd mod-call
# vim base.tf
module “vicky”
  source = “/tmp/module1”
  instance_name = “new-instance1”
}

provider “aws” {
  region = “ap-south-1”
}
```