Variables

What is a Variable?

Variables in Terraform are a great way to define centrally controlled reusable values. The information in Terraform variables is saved independently from the deployment plans, which makes the values easy to read and edit from single file.

*Two types are there, input variables and output variables.

*Types of input variables --> string, numbers, boolen, map, list

*How to pass variables in terraform --> env var, command line, file based.

Terraform variables can be defined within the infrastructure plan but are recommended to be stored in their own variables file. All files in your Terraform directory using the .tf file format will be automatically loaded during operations.

Create a variables file, for example, variables.tf and open the file for edit.

Let go to practicals.

Step1:- Create a resource file

mkdir -p variables/{demo1,demo2,demo3}

#vim resource.tf

```
[root@porali demo1]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = var.ami
  instance_type          = var.instance_type
  vpc_security_group_ids = ["sg-b844c2cd"]
  key_name               = "drhiju86"
  root_block_device {
    volume_size = 8
  }

  tags = {
    Name = "robo"
  }
}
[root@porali demo1]#
```

Step 1.1:-Create a variable file

#vim variable.tf

```
[root@porali demo1]# cat variable.tf
variable "ami" {}
variable "instance_type" {}
[root@porali demo1]#
```

*Important note: - curly brace {} is a block, we can give information.

Step 2:- Passing the input variable

terraform init

terraform plan

```
[root@porali demo1]# terraform plan
var.ami
  Enter a value: ami-00f8e2c955f7ffa9b

var.instance_type
  Enter a value: t2.micro
```

*Impartant note:- In this method, we need to give the input everytime, so it will be difficult to use in real time.

Step 2.2 Another method is exporting the value on enviroment.

export TF_VAR_ami=ami-00f8e2c955f7ffa9b

export TF_VAR_instance_type=t2.micro

```
[root@porali demo1]# export TF_VAR_ami=ami-00f8e2c955f7ffa9b
[root@porali demo1]# export TF_VAR_instance_type=t2.micro
[root@porali demo1]# env |grep TF_VAR
TF_VAR_instance_type=t2.micro
TF_VAR_ami=ami-00f8e2c955f7ffa9b
[root@porali demo1]#
```

terraform plan

```
[root@porali demo1]# terraform plan

Terraform used the selected providers to generate the following
the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                          = "ami-00f8e2c955f7ffa9b"
      + arn                          = (known after apply)
```

*Important note:- Now without asking variable, it has taken the info form env variables.

**End of plan output, you can see the –out, which denotes the option to save the terraform plan.

```
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if
you run "terraform apply" now.
[root@porali demo1]#
```

terraform plan -out=tfplan

```
Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
[root@porali demo1]#
```

Important note: - 1, we can apply by using tfplan, see able highlighted.

2, try to unset the env

unset TF_VAR_ami

unset TF_VAR_instance_type

env |grep TF_VAR

Step 2.3 Command line methods

terraform plan -out=tfplan -var "ami=ami-00f8e2c955f7ffa9b" -var "instance_type=t2.micro"

```
Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
```

#terraform apply “tfplan”

```
[root@porali demo1]# terraform apply "tfplan"
aws_instance.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Still creating... [40s elapsed]
aws_instance.web: Still creating... [50s elapsed]
aws_instance.web: Still creating... [1m0s elapsed]
aws_instance.web: Creation complete after 1m6s [id=i-0dafe70498fc6c111]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
[root@porali demo1]#
```

Important note :- now successfully created, let’s try to destroy the same and make sure to remove the –“out=tfplan”

terraform destroy -var "ami=ami-00f8e2c955f7ffa9b" -var "instance_type=t2.micro"

Step 2.4 file based method.

#vim terraform.tfvars

```
[root@porali demo1]# cat terraform.tfvars
ami = "ami-00f8e2c955f7ffa9b"
instance_type = "t2.micro"

[root@porali demo1]#
```

```
[root@porali demo1]# cat variable.tf
variable "ami" {}
variable "instance_type" {}
[root@porali demo1]#
```

*Important Note:-1, name and extension not supposed to changed.

2, Inside variable.tf, curly brace {} will take the info from terraform.tfvars.

3, var.ami and var.instance_type from resource.tf will fetch the info from variables.tf.

4, basically this is advisable for production and no need to edit main resource file, according to cloud providers, variable file can be used.

5, for variables.tf extension is not a matter, we can use without .tf by using this option -var-file=variable

terraform plan
terraform 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.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Still creating... [40s elapsed]
aws_instance.web: Still creating... [50s elapsed]
aws_instance.web: Still creating... [1m0s elapsed]
aws_instance.web: Creation complete after 1m5s [id=i-061285196f60b069d]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
[root@porali demo1]#
```

@@Types of input variables.

String --> sequence of Alphanumeric = ""

number --> values in number = 0-9

boolen --> true or false

map --> mapping key pair value, as like dictionary.

```
{
        "region1" = "us-east-1"
        "region2" = "us-east-2"
}
```

list --> list is nothing but array -- >["t3small","t3.micro","t3.medium"]

Step 3:- lets do practical with above types.

#vim variable.tf

```
[root@porali demo2]# cat variable.tf
variable "ami" {
  type = string
  description = "first ami"
  default = "ami-00f8e2c955f7ffa9b"
}

variable "instance_type" {
  type = list
  default = ["t2.micro","t3.micro","t3.medium"]
}

variable "region" {
  type = map
  default = {
    "region1" = "us-east-2"
    "region2" = "us-east-1"
  }
}

variable "volume_size" {
   type = number
   default = 8
}
[root@porali demo2]#
```

#cat resource.tf

```
[root@porali demo2]# cat resource.tf
resource "aws_instance" "web" {
  ami                    = var.ami
  instance_type          = var.instance_type[0]
  vpc_security_group_ids = ["sg-b844c2cd"]
  key_name               = "drhiju86"
  root_block_device {
    volume_size = var.volume_size
  }

  tags = {
    Name = "robo"
  }
}
```

#cat providers.tf

```
[root@porali demo2]# cat providers.tf
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.0"
    }
  }
}

#Configure tje AWS Provider
provider "aws" {
  region  = var.region.region1
  profile = "dev"
}
[root@porali demo2]#
```

*Note:- now the variables have been set on all the files.

terraform init

#terraform plan

#terraform apply

```
[root@porali demo2]# terraform apply

Terraform used the selected providers to generate the following e
the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                          = "ami-00f8e2c955f7ffa9b"
      + 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)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (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                     = "drhiju86"
      + outpost_arn                  = (known after apply)

      + root_block_device {
          + delete_on_termination = true
          + device_name           = (known after apply)
          + encrypted             = (known after apply)
          + iops                  = (known after apply)
          + kms_key_id            = (known after apply)
          + volume_id             = (known after apply)
          + volume_size           = 8
          + volume_type           = (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.web: Creating...
aws_instance.web: Still creating... [10s elapsed]
aws_instance.web: Still creating... [20s elapsed]
aws_instance.web: Still creating... [30s elapsed]
aws_instance.web: Creation complete after 35s [id=i-0aadeaf3d40a96586]

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

#terraform destroy

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

aws_instance.web: Destroying... [id=i-0aadeaf3d40a96586]
aws_instance.web: Still destroying... [id=i-0aadeaf3d40a96586, 10s elapsed]
aws_instance.web: Still destroying... [id=i-0aadeaf3d40a96586, 20s elapsed]
aws_instance.web: Still destroying... [id=i-0aadeaf3d40a96586, 30s elapsed]
aws_instance.web: Destruction complete after 31s

Destroy complete! Resources: 1 destroyed.
[root@porali demo2]#
```

Step 4:-lets add output variables and check the status.

*__Terraform output__ values allow you to export structured data about your resources. You can use this data to configure other parts of your infrastructure with automation tools, or as a data source for another Terraform workspace. Outputs are also necessary to share data from a child module to your root module.

cat output.tf

```
[root@porali demo2]# cat output.tf
output "instance_ips" {
  value = aws_instance.robo.*.public_ip
}
[root@porali demo2]#
```

#terraform plan

#terraform apply

```
Outputs:

instance_ips = [
  "3.143.254.13",
]
```

terraform output

```
[root@porali demo2]# terraform output
instance_ips = [
  "3.143.254.13",
]
[root@porali demo2]#
```