

Provider plugins are released separately from Terraform itself.

They have a different set of version numbers.



Version 1



Version 2

PROVIDER VERSIONING :

▀ Different Version Parameters :

▀ version = "2.7"

▀ version = ">= 2.8"

▀ version = "~> 2.x"

▀ version = "<= 2.8"

▀ version = ">=2.10,<=2.30"

LAB 7: PROVIDER VERSIONING ..

Output from a run

Terraform provides output for every run and same can be used to list the resources details which are created using help of Terraform:

```
output "myawssserver-ip" {  
  value = [aws_instance.myawssserver.public_ip]  
}
```

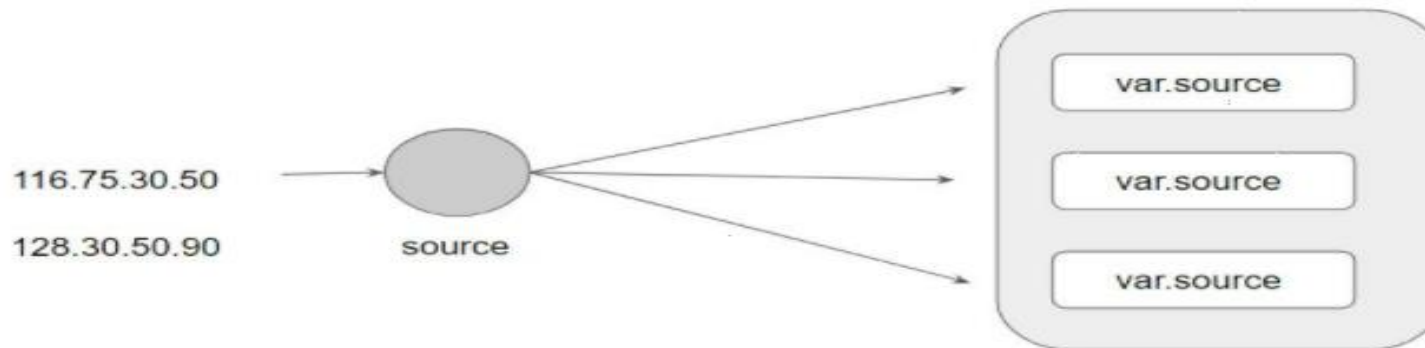
LAB 8 : EC2 instance with output value ..

Variables

Repeated static values can create more work in the future.



Terraform Variables allows us to centrally define the values that can be used in multiple terraform configuration blocks.



“ ■ *Lab 9: Variables* ..

Variables

- Variables can be of different types, based on terraform versions:

- **Strings**

```
variable "project" {  
  type = string }
```

- **Numbers**

```
variable "web_instance_count" {  
  type  = number  
  default = 1 }
```

- **Lists**

```
variable "cidrs" { default = ["10.0.0.0/16"] }
```

- **Maps**

```
variable "machine_types" {  
  type  = map  
  default = {  
    dev = "f1-micro"  
    test = "n1-highcpu-32"  
    prod = "n1-highcpu-32"  
  }  
}
```


“ ■ *LAB 10: Different Approaches for Variable Assignment* ..

Variables

- Variables can be assigned via different ways:
 - Via UI (if no default value is set in variable.tf)

*Via variable.tf default value

*Via .tfvars file (terraform.tfvars or custom.tfvars)

- Via command line flags:

```
terraform plan -var="instancetype=t2.small"
```

```
terraform plan -var-file="custom.tfvars"
```

Variables Definition Precedence

Terraform loads variables in the following order, with later sources taking precedence over earlier ones:

- Environment variables
- The terraform.tfvars file, if present.
- The terraform.tfvars.json file, if present.
- Any *.auto.tfvars or *.auto.tfvars.json files, processed in lexical order of their filenames.
- Any -var and -var-file options on the command line, in the order they are provided. (This includes variables set by a Terraform Cloud workspace.)

Lab 11: Variables

- 1) Declare AMI as variable and use same in your aws_instance resource
- 2) Define the value of AMI(with AMIID) inside terraform.tfvars file
- 3) terraform plan
- 4) Rename terraform.tfvars with abc.tfvars
- 5) Run terraform plan again
- 6) Run terraform plan with -var-file=abc.tfvars and see the outputs
- 7) Run terraform plan with -var ami="AMI_ID"

Executions

```
[root@main-tf app1]# terraform plan -var-file="abc.tfvars"
```

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.myawssserver will be created
+ resource "aws_instance" "myawssserver" {
+   ami           = "ami-064ff912f78e3e561"
```

LAB 12: Fetching data from map and list in variable ..



Working with change

Changes are of two types:

- Up-date In-place
- Disruptive

So always be careful with what you are adding/modifying

Update in-place

Update in-place will ensure your existing resources intact and modify the existing resources only. Here also based on what configuration is required to be changed, server may or may-not shutdown.

Making an update in your infra so that your resource state does not get affected.

(1 change)

“ ■ *LAB 13 :UPDATE IN PLACE ..*

Update - Disruptive

Disruptive updates require a resource to be deleted and recreated.

For example, modifying the image type for an instance will require instance to be deleted and re-created.

Making an update in your infra so that your resource state does not get affected.

(1 change)

update Disruptive:

Making an update in your infra so that old resource gets terminated/destroyed and a new resource gets created/deployed


(1 add, 1 destroy)

“ ■ LAB 14 :UPDATE DISRUPTIVE ..

LAB 15 : Changes outside of terraform

Changes which occurred outside of terraform are unwanted changes and if anything which is modified outside of terraform is detected, same will be marked in state files and will be corrected at next apply.

- Create a terraform code creating a server (code on next page)
- Run terraform show command to check current required state of infrastructure.
- Modify the tag through management console .
- Run terraform plan to check the behavior of terraform against the changes
- Check the terraform show command to view state file
- Check terraform refresh command to update the state frontend
- Run terraform apply to revert the changes
- Check the terraform refresh/show command as well as console again to validate the reversion of changes.



```
[root@main-tf app1]# cat fetch.tf
provider "aws" {
  region    = "us-east-2"
}

resource "aws_instance" "myec2" {
  ami = "ami-064ff912f78e3e561"
  instance_type = "t2.micro"
  tags = {
    Name = "raman-server"
  }
}
```