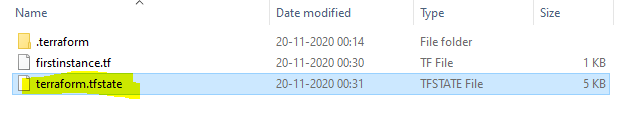
**Terraform—**Terraform is an open-source infrastructure as a code software tool created by HashiCorp It is used for building, changing, and versioning infrastructure safely and efficiently

**Terraform supports multiple cloud.**

**Terraform uses directory structure**

**Backend—**Placing state file to common location.(Lock when in use )

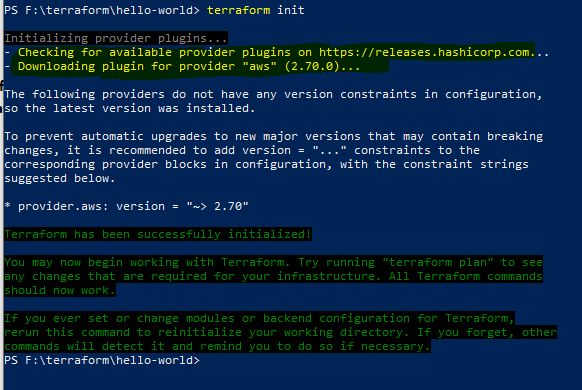


**Note :: We can do provisioning using shell scripting but they’re not idempotent.**

* Terraform can’t be used for physical machines
* For creating environment we use terraform, if we have environment already then we can use Ansible, puppet or chef for configuration.’

**Idempotent::** Executing the template multiple times gives us the same results.

1. **Terraform init** –it will look for the providers which we use and it’ll download plugins accordingly.(in **.Terraform** location)



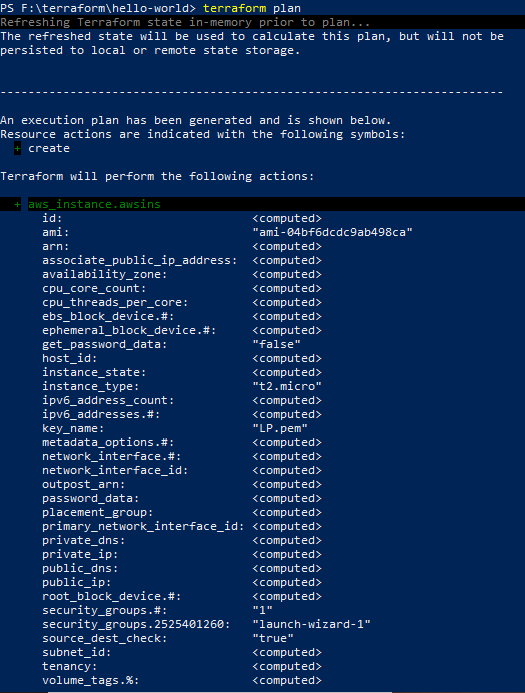
1. **validate - Validates the Terraform files**

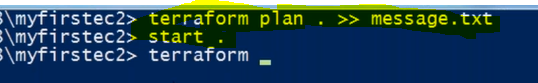


1. **plan - Generate and shows an execution plan**

**plan -out > file1.txt**

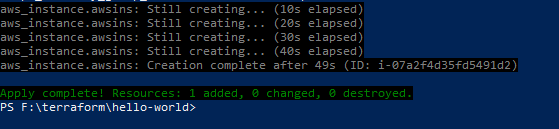




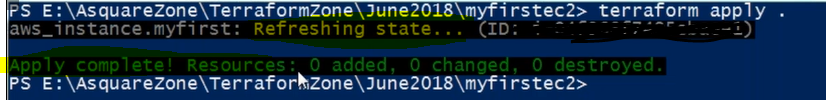


1. **apply - Builds or changes infrastructure**





If you try to apply again



**Provider**

**Resource**—What has to be created

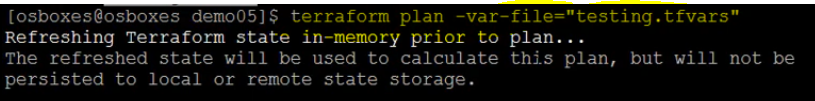
Provisioner

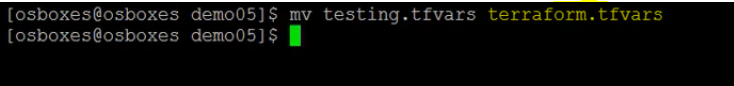
**Datasources**----Pull the data from provider

**Output--**

PS C:\WINDOWS\system32> terraform --help

Usage: terraform [--version] [--help] <command> [args]





**No need to mention –var while executing**

**Common commands:**

**apply - Builds or changes infrastructure**

console Interactive console for Terraform interpolations

**destroy - Destroy Terraform-managed infrastructure**

env Workspace management

**fmt Rewrites config files to canonical format**

get Download and install modules for the configuration

**graph - Create a visual graph of Terraform resources**

import Import existing infrastructure into Terraform

**init Initialize a Terraform working directory**

**output Read an output from a state file**

**plan Generate and show an execution plan**

**providers Prints a tree of the providers used in the configuration**

push Upload this Terraform module to Atlas to run

**refresh Update local state file against real resources**

**show Inspect Terraform state or plan**

taint Manually mark a resource for recreation

untaint Manually unmark a resource as tainted

**validate Validates the Terraform files**

version Prints the Terraform version

workspace Workspace management

All other commands:

debug Debug output management (experimental)

force-unlock Manually unlock the terraform state

state Advanced state management

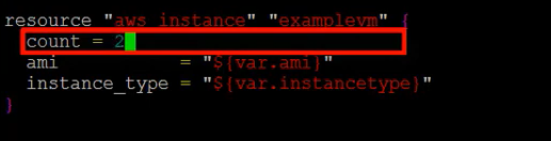
PS C:\WINDOWS\system32>

**Data sources** allow data to be fetched or computed for use elsewhere in Terraform

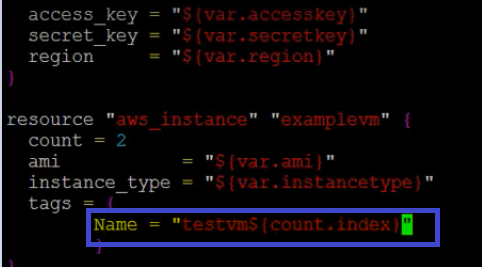
**Type.name.attriutes**

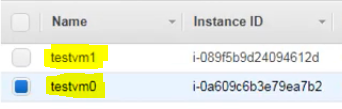
**Can we deploy multiple resources at a time in terraform and how?**

Yes ,,use **count** option and mention the number



How to use “**count.index**” ?



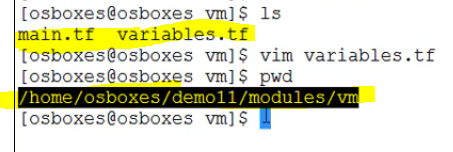


**Modules:**

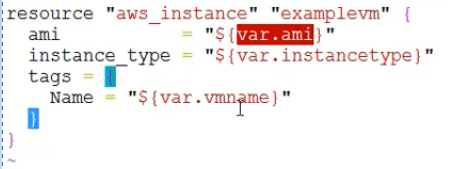
* combination of one or more resources , we can re-use them wherever we want
* A module is a container for multiple resources that are used together.

**Terraform module registry**

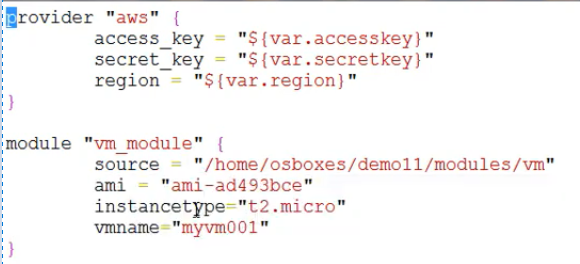
**How to write modules?**



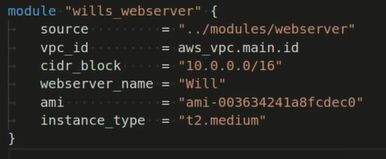
Module

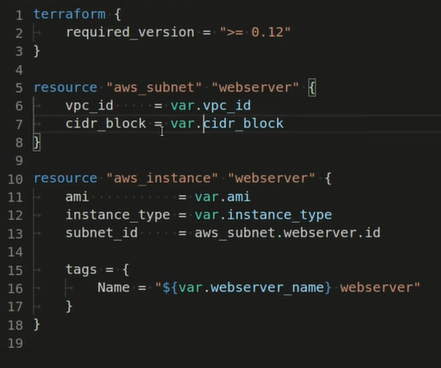


Main terraform file

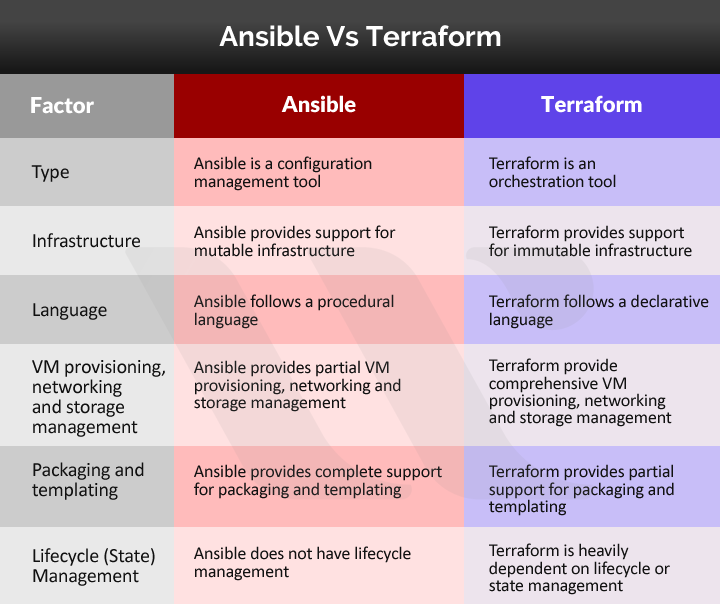


Or





**Diff b/w terraform and Ansible?**



Provisioning is the process of setting up IT infrastructure

Configurtion helps to configure the software and systems on the infrastructure

**what are the features of Terraform?**

1. **Graphing** – It gives us visual graph and the plan.
2. **Custom Syntax** - Its custom syntax is very friendly.
3. Maintains statefile for the previous execution
4. Supports every cloud providers.

**Use of Provisioners in Terraform?**

Provisioners are used for executing scripts or shell commands on a local or remote machine as part of resource creation/deletion.

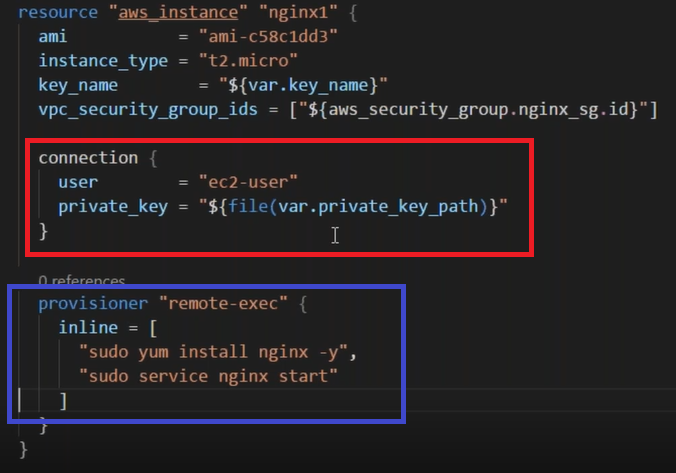
Ex ::

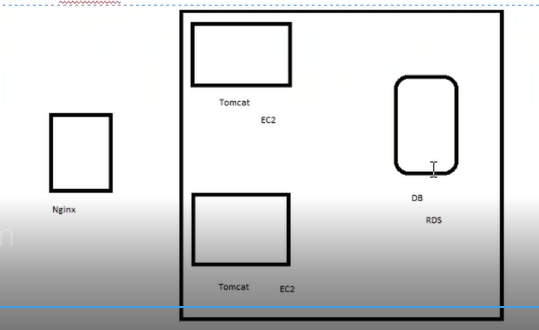
1. **local-exec**provisioner helps run a script on instance where we are running our terraform code, not on the resource we are creating.

**provisioner "local-exec" {  
command = "echo ${aws\_instance.testInstance.public\_ip} >> public\_ip.txt"  
}**  
}

1. remote-exec**provisioner helps invoke a script on the remote resource once it is created.**

provisioner "remote-exec" {  
inline = [  
"sudo yum -y install nginx",  
"sudo systemctl start nginx",  
]  
}





## How to lock statefile in terraform?

If the state file is stored remotely so that many people can access it, then you risk multiple people attempting to make changes to the same file at the exact same time. So we need to provide a mechanism that will “lock” the state if its currently in-use by another user. We can accomplish this by creating a dynamoDB table for terraform to use.

Create the dynamoDB table like this:

# example.tf# create a dynamodb table for locking the state file  
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"  
 }  
}

You will need to modify the Terraform S3 backend resource and add in the dynamoDB table:

# terraform.tfterraform {  
 backend “s3” {  
 encrypt = true  
 bucket = "terraform-remote-state-storage-s3"  
 dynamodb\_table = "terraform-state-lock-dynamo"  
 region = us-west-2  
 key = path/to/state/file  
 }  
}

**How to revover tfstatefile?**

Store the terraform statefile in a backend like S3 and enable versioning so to avoid unintended deletion ,in addition to that u can enable CRR(Cross region replication feature so that ,the objects in the buckets are replicated to other region)

**What is remote state in Terraform?**

https://github.com/ravsau/aws-labs/tree/master/terraform-aws/lesson4-s3-backend-state-file

## Store State Remotely in S3

If you are working on a team, then its best to store the terraform state file remotely so that many people can access it. In order to setup terraform to store state remotely you need two things: an s3 bucket to store the state file in and an terraform s3 backend resource.

You can create an s3 bucket in a terraform config like so:

# example.tfprovider "aws" {  
 region = "us-west-2"  
}# terraform state file setup  
# create an S3 bucket to store the state file in  
resource "aws\_s3\_bucket" "terraform-state-storage-s3" {  
 bucket = "terraform-remote-state-storage-s3"  
   
 versioning {  
 enabled = true  
 }  
   
 lifecycle {  
 prevent\_destroy = true  
 }  
   
 tags {  
 Name = "S3 Remote Terraform State Store"  
 }   
}

Then create the s3 backend resource like so:

# terraform.tfterraform {  
 backend “s3” {  
 encrypt = true  
 bucket = "terraform-remote-state-storage-s3"  
 region = us-west-2  
 key = path/to/state/file  
 }  
}