terraform init : install dependencies and packages

terraform plan : checking which one creating for that

terraform apply: after apply created resources and one more is every 10s its showing status

terraform destroy: first check the resources

/\* --> commented its assumes destroyed

terraform **tfstate file** its store the details of resources data

terraform refresh : its refresh the latest running data

## Desired state and current state

Desired state: EC2 = instance\_type = t2.micro

NOTE: After terraform refresh its update the current running data

Current state :” instance\_state” =”stopped”, && “instance\_type” = “t2.nano”

NOTE:

Desired state ==Current state -🡪 if any modification to current state after its reflecting the terraform apply its creating the Desired state Details

Note: main thing its check the .tf file data only for checking purpose

EX: its checking for security Group

earlier : default , After its manually change the new security group

Now : suneel-sg

Versions:

During terraform init,if version argument is not specified, the most recent provider will be downloaded during initialization.

For Production use, you should constrain the acceptable provider versions via configuration, to ensure that new versions with breaking changes will not be automatically installed.

Arguments for Specifying Provider

|  |  |
| --- | --- |
| >=1.0 | Greaterthan equal to the version |
| <=1.0 | Lessthan equal to the version |
| ~>2.0 | Any version in the 2.x range |
| >=2.10, <=2.30 | Any version between 2.10 and 2.30 |

Types of Providers:

HashiCorp Distributed Providers can be downloaded automatically during terraform init.

Terraform init cannot automatically download providers that are not distributed by hashicrop

|  |  |
| --- | --- |
| Operating System | user plugins directory |
| Windows | %APPDATA%\terraform.d\plugins |
| All other Systems | ~/.terraform.d/plugins |

Understanding Attributes:

Terraform has capability to output the attribute of a resource with the output values.

Ex: ec2\_public\_ip = 35.161.62.171

Bucket\_identifier = terraform-test-kplabs.s3.amazonaws.com

Attributes are important:

An outputted attributes can not only be used for the user reference but it can also act as a input to other resource being created via terraform

Lets understand this example.

After EIP is created,its IP address should automatically get whitelisted in the security group

Variables:

We can have a central source from which we can import the values from

Mutiple Approches to variables:

Variables in Terraform can be assigned values in multiple ways

Some of these include:

Environment Variables :

Command Line flags : Ex: terraform plan –var=”intsnacetype=t2.micro”

From a File : create one file like terraform.tfvars

And custom.tfvars also but its working in through CLI ex; terraform plan –var-file = “custom.tfvars”

Variable Defaults :

Datatypes of variables:

The type argument in a variable block allows you to restrict the type of value that will be accepted as the value for a variable

Variable “image\_id” {

Type = string

}

NOTE: If no type constraint is set then a value of any type is accepted.

Example:

Every employee in medium Corp is assigned a Identification number

Any resource that employee creates should be created with the name of the Identification number only

|  |  |
| --- | --- |
| Variables.tf | Terraform.tfvars |
| Variable “instance\_name” { } | Instance\_name=”john-123” |

Count Paramaters:

The count parameter on resources can simplifyconfigurations and let youscale resources by simply incrementing a number.

Lets assume, you needs to create two Ec2 instances. One of the common approach is to define two separate resource blocks for aws\_instance.

With count parameter,we can simply specify the count value and the resource can be scaled accordingly.

Count Index:

In resource blocks where count is set,an additional count object is variable in expressions, so you can modify the configuration of each instance.

This object has one attribute:

Count.index – The distict index number (starting with 0) corresponding the instance.

Condition Expression:

A conditional expression uses the value of a bool expression to select one of two values.

Syntax: condition ? true : false

If condition is true then the result is true. If condition is false then the result is false.

Local Values:

A local value assigns a name to an expression, allowing it to be used multiple times within a module without repeating it.

DataSources:

Data sources allow data to be feteched or computed for use elsewhere in terraform configuration.

--Defined under the data block.

Reads from a specific data source(aws\_ami)and exports results under (app\_ami).

Debugging Terraform:

Terraform has detailed logs which can be enabled by setting the TF\_LOG environment variable to any value.

You can set TF\_LOG to one of the log levels TRACE,DEBUG,INFO,WARN, or ERROR to change the verbosity of the logs.

Important Points:

TRACE is the most verbose and it is the default if TF\_LOG is set to something otherthan a log level name.

To persist logged output you can set TF\_LOG\_PATH in order to force the log to always be appendedto a specific file when logging is enabled.

Terraform Format:

Importance Readabulity:

Anyone who is into programming knows the importance of formatting the code for readability.

The terraform fmt command is used to rewrite Terraform configuration files to take care of the overall formatting.

Ex: terraform fmt

Terraform validate:

Terraform validate primarily checks whether a configuration is syntactically valid.

It can check various aspects including unsupported arguments ,undeclared variables and others.

Ex: terraform validate

Loaders & Semantics:

Terraform generally loads all the configuration files within the directory specified in alphabetical order.

The files loaded must end in either .tf or .tf.json to specify the format that is in use.

Dynamic Blocks:

In many of the use-cases, there are repetable nested blocks thata needs to be defined.

This can lead to a long code and it can be difficult to manage in a longer time.

-🡪 Dynamic Block allows us to dynamically construct repetable nested blocks which is supported inside resource,data,provider,and provisioner blocks.

Terraform Taint:

You have created a new resource via terraform.

Users have made a lot of manual changes(both infrastructureand inside the server)

Two ways to deal with this : Import the changes to terraform/Delete & Recreate the resource.

* The terraform taint command manually marks a terraform-managed resource as tainted,forcing it to be destroyed and recreated on the next apply.
* This command will not modify infrastructure , but does modify the state file in order to mark a resource as tainted.
* Once a resource is marked as tainted,the next plan will show that the resource will be destroyed and recreated and the next apply will implement this change.
* Note that tainting a resource for recreation may affect resources that depend on the newly tainted resource.

Spalat Expression:

Splat Expression allows us to get a list of all the attributes.

Graph:

The terraform Graph command is used to generate a visual representation of either a configuration or execution plan.

The output of terraform graph is in the DOT format ,which can easily be converted to an image.

Terraform plan file

The generated terraform plan can be saved to a specified path.

This plan can then be used with terraform apply to be certain that only the changes shown in this plan are applied.

Example: terraform plan –out=path