**Terraform**

What is Terraform?

Terraform is an IAC tool, used primarily by [DevOps teams](https://www.varonis.com/blog/aws-devops/?hsLang=en) to automate various infrastructure tasks. Terraform allows you to create your complete infrastructure in the form of code. Even if your servers come from different providers such as AWS or [Azure](https://www.varonis.com/blog/what-is-azure-devops/?hsLang=en), Terraform helps you build and manage these resources in parallel across providers.

**Installing Terraform on local machine:**

1. Create an user in AWS IAM to provide access to terraform for access key and secret to key to configure AWS CLI.
2. Install AWS CLI in our local machine by using below link.

msiexec.exe /i <https://awscli.amazonaws.com/AWSCLIV2.msi>

1. Check the version of AWS CLI

aws –version

1. Run aws configure to run the commands related to aws. To do that run the below commands
   1. aws configure
      1. enter your access key
      2. enter your secret key
      3. enter your region where you want to do the activities
      4. enter none in output format
2. In my case, I am using Visual studio code (VSD) for writing the terraform files, to do that, I have installed VSD. To create a new project in terraform, follow the below steps

Installation of terraform:

Go to the browser search for terraform select the required version and download the same.

Unzip the downloaded file and copy the file in winows32 folder in C drive

* 1. Create a project in terraform
     1. Select open folder
     2. Create a folder in local machine, where you want to save your terraform files.
     3. In the terraform terminal under the selected folder, create a file with .tf extension.
  2. Click on terminal
     1. Select new terminal
     2. You will see a new terminal at the bottom of the terminal.

Basic Commands of Terraform

## *Get Help:*

terraform -help = Get a list of available commands for execution with descriptions. Can be used with any other subcommand to get more information.

terraform fmt -help = Display help options for the fmt command.

## *Show Your Terraform Version:*

terraform version = To know the current version of your Terraform and notifies you if there is a newer version available for download.

## *Format Your Terraform Code:*

This should be the first command you run after creating your configuration files to ensure your code is formatted using the HCL standards. This makes it easier to follow and aids collaboration.

terraform fmt = Format your Terraform configuration files using the HCL language standard.

terraform fmt --recursive = Also formats files in subdirectories

terraform fmt --diff = Display differences between original configuration files and formatting changes.

terraform fmt --check = Useful in automation [CI/CD pipelines](https://spacelift.io/blog/terraform-in-ci-cd), the check flag can be used to ensure the configuration files are formatted correctly, if not the exit status will be non-zero. If files are formatted correctly, the exit status will be zero.

## *Initialize Your Directory:*

terraform init = In order to prepare the working directory for use with Terraform, the terraform init command performs Backend Initialization, Child Module Installation, and Plugin Installation.

terraform init -get-plugins=false = Initialize the working directory, do not download plugins.

terraform init -lock=false = Initialize the working directory, don’t hold a state lock during backend migration.

terraform init -input=false = Initialize the working directory, and disable interactive prompts.

terraform init -migrate-state = Reconfigure a backend, and attempt to migrate any existing state.

terraform init -verify-plugins=false = Initialize the working directory, do not verify plugins for Hashicorp signature

## *Download and Install Modules:*

Note this is usually not required as this is part of the terraform init command.

terraform get = Downloads and installs [modules](https://spacelift.io/blog/what-are-terraform-modules-and-how-do-they-work) needed for the configuration.

terraform get -update = Checks the versions of the already installed modules against the available modules and installs the newer versions if available.

## *Validate Your Terraform Code:*

terraform validate = Validates the configuration files in your directory and does not access any remote state or services. terraform init should be run before this command.

## *Plan Your Infrastructure:*

terraform plan = Plan will generate an execution plan, showing you what actions will be taken without actually performing the planned actions.

**terraform plan -out=<path> = Save the plan file to a given path. Can then be passed to the terraform apply command.**

terraform plan -destroy = Creates a plan to destroy all objects, rather than the usual actions.

## *Deploy Your Infrastructure:*

terraform apply = Creates or updates infrastructure depending on the configuration files. By default, a plan will be generated first and will need to be approved before it is applied.

terraform apply -auto-approve = Apply changes without having to interactively type ‘yes’ to the plan. Useful in automation CI/CD pipelines.

terraform apply <planfilename> = Provide the file generated using the terraform plan -out command. If provided, terraform will take the actions in the plan without any confirmation prompts.

terraform apply -lock=false = Do not hold a state lock during the Terraform apply operation. Use with caution if other engineers might run concurrent commands against the same workspace.

terraform apply -parallelism=<n> = Specify the number of operations run in parallel.

terraform apply -var="domainpassword=password123" = Pass in a variable value.

terraform apply -var-file="varfile.tfvars" = Pass in variables contained in a file.

terraform apply -target=”module.appgw.0" = Apply changes only to the targeted resource.

## *Destroy Your Infrastructure:*

terraform destroy = To destroy the infrastructure managed by Terraform.

terraform destroy -target=”module.appgw.0" = Destroy only the targeted resource.

terraform destroy -auto-approve = Destroys the infrastructure without having to interactively type ‘yes’ to the plan. Useful in automation CI/CD pipelines.

## *‘Taint’ or ‘Untaint’ Your Resources:*

Use the [taint command](https://spacelift.io/blog/terraform-taint) to mark a resource as not fully functional. It will be deleted and re-created.

terraform taint vm1.name = Taint a specified resource instance.

terraform untaint vm1.name = Untaint the already tainted resource instance.

## *Refresh the State File:*

terraform refresh = Modifies the state file with updated metadata containing information on the resources being managed in Terraform. Will not modify your infrastructure.

## *View Your State File:*

terraform show = Shows the state file in a human-readable format.

terraform show <path to statefile> = If you want to read a specific state file, you can provide the path to it. If no path is provided, the current state file is shown.

## *Manipulate Your State File:*

terraform state = One of the following subcommands must be used with this command in order to manipulate the state file.

terraform state list = Lists out all the resources that are tracked in the current state file.

terraform state mv = Moves an item in the state, for example, this is useful when you need to tell Terraform that an item has been renamed, e.g. terraform state mv vm1.oldname vm1.newname

terraform state pull > state.tfstate = Gets the current state and outputs it to a local file.

terraform state push = Update remote state from the local state file.

terraform state replace-provider hashicorp/azurerm customproviderregistry/azurerm = Replace a provider, useful when switching to using a custom provider registry.

terraform state rm = Remove the specified instance from the state file. Useful when a resource has been manually deleted outside of Terraform.

terraform state show <resourcename> = Show the specified resource in the state file.

## *Import Existing Infrastructure into Your Terraform State:*

terraform import vm1.name -i id123 = Import a VM with id123 into the configuration defined in the configuration files under vm1.name.

terraform import vm1.name -i id123 -allow-missing-config = Import and allow if the configuration block does not exist.

*Top of Form*

*Bottom of Form*

## *Get Provider Information:*

terraform providers = Displays a tree of providers used in the configuration files and their requirements.

## *Manage Your Workspaces:*

terraform workspace = One of the following subcommands must be used with the workspace command. Workspaces can be useful when an engineer wants to test a slightly different version of the code. It is not recommended to use Workspaces to isolate or separate the same infrastructure between different development stages, e.g. Dev / UAT / Production, or different internal teams.

terraform workspace show = Show the name of the current workspace.

terraform workspace list = List your workspaces.

terraform workspace select <workspace name> = Select a specified workspace.

terraform workspace new <workspace name> = Create a new workspace with a specified name.

terraform workspace delete <workspace name> = Delete a specified workspace.

## *View Your Outputs:*

terraform output = Lists all the [outputs](https://spacelift.io/blog/terraform-output) currently held in your state file. These are displayed by default at the end of a terraform apply, this command can be useful if you want to view them independently.

terraform output -state=<path to state file> = Lists the outputs held in the specified state file.

terraform output -json = Lists the outputs held in your state file in JSON format to make them machine-readable.

terraform output vm1\_public\_ip = List a specific output held in your state file.

## *Release a Lock on Your Workspace:*

## terraform force-unlock <lock\_id> — Remove the lock with the specified lock ID from your workspace. Useful when a lock has become ‘stuck’, usually after an incomplete Terraform run.

## *Log In and Out to a Remote Host (Terraform Cloud):*

terraform login = Grabs an API token for Terraform cloud (app.terraform.io) using your browser.

terraform login <hostname> = Log in to a specified host.

terraform logout = Removes the credentials that are stored locally after logging in, by default for Terraform Cloud (app.terraform.io).

terraform logout <hostname> = Removes the credentials that are stored locally after logging in for the specified hostname.

## *Produce a Dependency Diagram:*

*te*rraform graph = Produces a graph in DOT language showing the dependencies between objects in the state file. This can then be rendered by a program called Graphwiz (amongst others).

terraform graph -plan=tfplan = Produces a dependency graph using a specified plan file (generated using terraform plan -out=tfplan).

terraform graph -type=plan = Specifies the type of graph to output, either plan, plan-refresh-only, plan-destroy, or apply.

## Test Your Expressions

terraform console = Allows testing and exploration of expressions on the interactive console using the command line.

***Installation of Terraform in Ubuntu:***

Open browser and paste the below path and follow the instructions as below

https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

1. Select your OS as linux
2. Select ubuntu
3. Paste the commands in your ubuntu server

**sudo apt-get update && sudo apt-get install -y gnupg software-properties-common**

The above command will update your repository

**wget -O- https://apt.releases.hashicorp.com/gpg | \**

**gpg --dearmor | \**

**sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg**

The above command will install the GPG key

**gpg --no-default-keyring \**

**--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \**

**--fingerprint**

The above command will verify the keys fingerprint

**echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \**

**https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | \**

**sudo tee /etc/apt/sources.list.d/hashicorp.list**

The above command will download HashiCorp repository to your server

**sudo apt update**

The above command will update the apt repository

**sudo apt-get install terraform**

The above command will install the terraform into your server

***RESOURCE:***

Resource is an object that terraform manages. It could be a file on local host, virtual machines like instances on the cloud, or the services like S3, IAM, Lambda etc.

**How to create EC2 instance using terraform:**

*Below are the parameters highlighted in red colour to be considered while creating instances*

*You can check the below parameters in the given link for key and values.*

*https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#host\_id*

**resource "aws\_instance" "demo" {**

**+ ami = "ami-0590f3a1742b17914" (your ami id no)**

**+ arn = (known after apply)**

**+ associate\_public\_ip\_address = (known after apply) (it should be true or false)**

**+ availability\_zone = (known after apply) (provide your zone as per the required region)**

**+ cpu\_core\_count = (known after apply) (it will be like 16 core or 8 core or 2 core or 1 core) To check the cpu core in your server type “nproc”**

**+ cpu\_threads\_per\_core = (known after apply) (every single cpu has two cores, based on the cpu’s cores will be doubled)**

**+ disable\_api\_stop = (known after apply) (the value will be either true or false, When you given this value as true, you cant stop the instance, until it is change to false)**

**+ disable\_api\_termination = (known after apply) (the value will be either true or false, When you given this value as true, you cant terminate the instance, until it is change to false)**

**+ ebs\_optimized = (known after apply) (The value will be either true or false, ebs optimized are designed to deliver fast performance for workloads that process large data sets in memory.)**

**+ get\_password\_data = false (The value will be either true or false, check this once after making it to true) I think this is to generate key pair**

**+ host\_id = (known after apply) (it is nothing but instance id, It will be known after apply)**

**+ host\_resource\_group\_arn = (known after apply) (The value will be either true or false. A host resource group is a collection of Dedicated Hosts that you can manage as a single entity.** )

**+ iam\_instance\_profile = (known after apply) (need to check once)**

**+ id = (known after apply)**

**+ instance\_initiated\_shutdown\_behavior = (known after apply)**

**+ instance\_state = (known after apply) (it will be known after apply to see instance status is running/stopped/hibernation/terminated)**

**+ instance\_type = "t2.micro" (type of instance)**

**+ ipv6\_address\_count = (known after apply) (to provide the ipv6 addresses)**

**+ ipv6\_addresses = (known after apply) (mention the ipv6 address)**

**+ key\_name = "Tokyo" (provide key pair name)**

**+ monitoring = (known after apply) (If true, the launched EC2 instance will have detailed monitoring enabled)**

**+ outpost\_arn = (known after apply)**

**+ password\_data = (known after apply)**

**+ placement\_group = (known after apply) (Placement Group to start the instance in)**

**+ placement\_partition\_number = (known after apply)**

**+ primary\_network\_interface\_id = (known after apply)**

**+ private\_dns = (known after apply)**

**+ private\_ip = (known after apply) (It will be known after apply, Private IP address to associate with the instance in a VPC.)**

**+ public\_dns = (known after apply)**

**+ public\_ip = (known after apply) (It will be known after apply, Public IP address to associate with the instance in a VPC.)**

**+ secondary\_private\_ips = (known after apply)**

**+ security\_groups = (known after apply) (provide your security group name)**

**+ source\_dest\_check = true**

**+ subnet\_id = (known after apply) (Provide your subnet range, VPC Subnet ID to launch in)**

**+ tags = {**

**+ "Name" = "demo server"**

**}**

**+ tags\_all = {**

**+ "Name" = "demo server"**

**}**

**+ tenancy = (known after apply) (valid values are default, dedicated, host)**

**+ user\_data = (known after apply) (to provide scripts to run)**

**+ user\_data\_base64 = (known after apply)**

**+ user\_data\_replace\_on\_change = false (either true or false)**

**+ vpc\_security\_group\_ids = (known after apply) (provide your security group id)**

**+ capacity\_reservation\_specification {**

**+ capacity\_reservation\_preference = (known after apply)**

**+ capacity\_reservation\_target {**

**+ capacity\_reservation\_id = (known after apply)**

**+ capacity\_reservation\_resource\_group\_arn = (known after apply)**

**}**

**}**

**+ ebs\_block\_device {**

**+ delete\_on\_termination = (known after apply) (value either true or false, Whether the volume should be destroyed on instance termination)**

**+ device\_name = (known after apply) (name of the device to mount)**

**+ encrypted = (known after apply) (data to be encrypted or not, value will be true or false)**

**+ iops = (known after apply)** (**Amount of provisioned**[**IOPS**](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-io-characteristics.html)**. Only valid for volume\_type of io1, io2, gp3)**

**+ kms\_key\_id = (known after apply)**

**+ snapshot\_id = (known after apply)**

**+ tags = (known after apply) (Name of the volume)**

**+ throughput = (known after apply)**

**+ volume\_id = (known after apply) (Wil be known after apply, or else we can add existing volume with volume id)**

**+ volume\_size = (known after apply) Size of the volume in gibibytes (GiB))**

**+ volume\_type = (known after apply) (type of volume ie standard, gp2, gp3, io1, io2, sc1/st1, Default to gp2)**

**}**

**+ enclave\_options {**

**+ enabled = (known after apply)**

**}**

**+ ephemeral\_block\_device {**

**+ device\_name = (known after apply)**

**+ no\_device = (known after apply)**

**+ virtual\_name = (known after apply)**

**}**

**+ maintenance\_options {**

**+ auto\_recovery = (known after apply)**

**}**

**+ metadata\_options {**

**+ http\_endpoint = (known after apply)**

**+ http\_put\_response\_hop\_limit = (known after apply)**

**+ http\_tokens = (known after apply)**

**+ instance\_metadata\_tags = (known after apply)**

**}**

**+ network\_interface {**

**+ delete\_on\_termination = (known after apply)**

**+ device\_index = (known after apply)**

**+ network\_card\_index = (known after apply)**

**+ network\_interface\_id = (known after apply)**

**}**

**+ private\_dns\_name\_options {**

**+ enable\_resource\_name\_dns\_a\_record = (known after apply)**

**+ enable\_resource\_name\_dns\_aaaa\_record = (known after apply)**

**+ hostname\_type = (known after apply)**

**}**

**+ root\_block\_device {**

**+ delete\_on\_termination = (known after apply) (value either true or false, Whether the volume should be destroyed on instance termination)**

**+ device\_name = (known after apply) (name of the device to mount) + encrypted = (known after apply) (data to be encrypted or not, value will be true or false)**

**+ iops = (known after apply)** (**Amount of provisioned**[**IOPS**](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-io-characteristics.html)**. Only valid for volume\_type of io1, io2, gp3).**

**+ kms\_key\_id = (known after apply)**

**+ tags = (known after apply) (Name of the volume)**

**+ throughput = (known after apply)**

**+ volume\_id = (known after apply) (Will be known after apply, or else we can add existing volume with volume id)**

**+ volume\_size = (known after apply) (Size of the volume in gibibytes (GiB))**

**+ volume\_type = (known after apply) (type of volume ie standard, gp2, gp3, io1, io2, sc1/st1, Default to gp2)**

**}**

**}**

**How to use terraform input variables.**

Variables in [Terraform](https://upcloud.com/blog/upcloud-verified-terraform-provider/) 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 a single file.

variable “your variable name” {

description = “any meaning full name to your variable”

type = string, number, Boolean, list, set, map, etc

default = “variable value”

}

In the above variable snippet, we will change the variable name, description, type might be anything as per the above list and default value should be given as per the type.

For ex:

Type default value

string name of the string (t2.micro/t2.medium etc)

number 2/3/4 etc

bool true/false

**String variable**

**To create instance with t2.micro (string type) in aws console:**

resource “aws\_instance” “example {

ami = “ami id”

instance\_type = var.instance

tags = {

Name = “my instance”

}

}

Variable “instance” {

Description = “my instance”

Type = string

Default = “t2.micro”

}

**Boolean variable:**

**to create ec2 instance with public ip using Boolean variable**

resource “aws\_instance” “example” {

ami = “ami-id”

instance\_type = “t2.micro”

associate\_public\_ip\_address = var.public\_ip

tags = {

Name = “My-Instance”

}

}

Variable “public\_ip” {

Description = “assigning public ip to instance”

Type = bool

Default = true

}

**List variable:**

**To create 3 IAM users in aws console:**

resource “aws\_iam\_user” “example” {

count = length.var.user\_names

name = var.user\_names [count.index]

}

Variable “user\_names” {

Description = “IAM user creation”

Type = list(string)

Default = [“user1”, “user2”, “user3”]

}

**Map variable:**

to create ec2 instance with tags using map variable

resource “aws\_instance” “example” {

ami = “ami id”

instance\_type = “t2.micro”

tags = var.project\_environment

}

Variable “project\_environment” {

Description = “project name and environment”

Type = map(string)

Default = {

Project = “project-alpha”

Environment = “Dev”

}

}