# How to run terraform?

* Download the files terraform.zip from server.
* Extract it in that folder.
* For writing a script make file1 (main.tf) and write a code in it.
* For variables make file2 (variables.tf) and write a code in it.
* And making app registration that has to be given access according to its requirements.
* In file1 write the following code with appropriate id.

*provider* "azurerm" {

    version = "=2.0.0"

  features {}

    subscription\_id = "b616044f-0c6e-46e7-87e8-3ba295723db7"

    client\_id = "d75ff049-5a9d-40bc-a60c-1c5be2d1ad51"

    client\_secret = "5du\_7hT3\_72z-Tgse.hB797V84vV4JHE-v"

    tenant\_id = "d08718f4-1a78-4344-bf15-2a283cb16d36"

  }

* For writing resource Blocks:

*resource* "azurerm\_resource\_group" "resource\_gp" {

name = "hello"

location = "eastus2"

}

* The in command prompt write>> terraform init
* Then>> terraform plan
* Then>> terraform apply

That will deploy the resources in the specified id.

# WHAT IS TF STATE FILES?

* While the format of the state files are just JSON, direct file editing of the state is discouraged. Terraform provides the [terraform state](https://www.terraform.io/docs/commands/state/index.html) command to perform basic modifications of the state using the CLI.

# local-exec Provisioner:

The local-exec provisioner invokes a local executable after a resource is created. This invokes a process on the machine running Terraform, not on the resource. See the remote-exec provisioner to run commands on the resource.

Note that even though the resource will be fully created when the provisioner is run, there is no guarantee that it will be in an operable state - for example system services such as sshd may not be started yet on compute resources.

## [»](https://www.terraform.io/docs/provisioners/local-exec.html" \l "example-usage)Example usage

resource "aws\_instance" "web" {

provisioner "local-exec" {

command **=** "echo ${aws\_instance.web.private\_ip} >> private\_ips.txt"

}

}

## [»](https://www.terraform.io/docs/provisioners/local-exec.html" \l "argument-reference)Argument Reference

The following arguments are supported:

* [command](https://www.terraform.io/docs/provisioners/local-exec.html" \l "command) - (Required) This is the command to execute. It can be provided as a relative path to the current working directory or as an absolute path. It is evaluated in a shell, and can use environment variables or Terraform variables.

* [working\_dir](https://www.terraform.io/docs/provisioners/local-exec.html" \l "working_dir) - (Optional) If provided, specifies the working directory where command will be executed. It can be provided as as a relative path to the current working directory or as an absolute path. The directory must exist.

* [interpreter](https://www.terraform.io/docs/provisioners/local-exec.html" \l "interpreter) - (Optional) If provided, this is a list of interpreter arguments used to execute the command. The first argument is the interpreter itself. It can be provided as a relative path to the current working directory or as an absolute path. The remaining arguments are appended prior to the command. This allows building command lines of the form "/bin/bash", "-c", "echo foo". If interpreter is unspecified, sensible defaults will be chosen based on the system OS.

* [environment](https://www.terraform.io/docs/provisioners/local-exec.html" \l "environment) - (Optional) block of key value pairs representing the environment of the executed command. inherits the current process environment.

# Remote-exec Provisioner:

The remote-exec provisioner invokes a script on a remote resource after it is created. This can be used to run a configuration management tool, bootstrap into a cluster, etc. To invoke a local process, see the local-exec [provisioner](https://www.terraform.io/docs/provisioners/local-exec.html) instead. The remote-exec provisioner supports both ssh and winrm type [connections](https://www.terraform.io/docs/provisioners/connection.html).

## [»](https://www.terraform.io/docs/provisioners/remote-exec.html#example-usage)Example usage

resource "aws\_instance" "web" {

provisioner "remote-exec" {

inline **=** [

"puppet apply",

"consul join ${aws\_instance.web.private\_ip}",

]

}

}

## [»](https://www.terraform.io/docs/provisioners/remote-exec.html#argument-reference)Argument Reference

The following arguments are supported:

* [inline](https://www.terraform.io/docs/provisioners/remote-exec.html" \l "inline) - This is a list of command strings. They are executed in the order they are provided. This cannot be provided with script or scripts.

* [script](https://www.terraform.io/docs/provisioners/remote-exec.html" \l "script) - This is a path (relative or absolute) to a local script that will be copied to the remote resource and then executed. This cannot be provided with inline or scripts.

* [scripts](https://www.terraform.io/docs/provisioners/remote-exec.html" \l "scripts) - This is a list of paths (relative or absolute) to local scripts that will be copied to the remote resource and then executed. They are executed in the order they are provided. This cannot be provided with inline or script.

# TFVAR file>>

* These file are securing Terraform API keys.
* The method is to use a file called terraform.tfvars.
* The file terraform.tfvars should only live on your local computer and never checked into source control.
* It is very important that terraform.tfvars is placed in your .gitignore file.
* So we have a few changes to the files. I’ll start with aws.tf again. Here we have separately defined both access keys.

provider "aws" {

access\_key = "${var.aws\_access\_key}"

secret\_key = "${var.aws\_secret\_key}"

region = "${var.region}"

}

resource "aws\_instance" "web-server" {

ami = "ami-0c2aba6c"

instance\_type = "t2.micro"

tags {

Name = "terraformtraining.com"

}

}

* Here we define the variables, note the access and secret key are blank.

variable "aws\_access\_key" {}

variable "aws\_secret\_key" {}

variable "region" {

default = "us-west-2"

}

* And finally we have the terraform.tfvars file. aws\_access\_key = "ENTER-YOUR-ACCESS-KEY-HERE"
* aws\_secret\_key = "ENTER-YOUR-SECRET-KEY-HERE"
* So now Terraform picks up the credentials from the terraform.tfvars file and authenticates correctly.

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.)

EXaMPLE:

variable "image\_id" {

type **=** string

description **=** "The id of the machine image (AMI) to use for the server."

validation {

condition **=** length(var.**image\_id**) **>** 4 **&&** substr(var.**image\_id**, 0, 4) **==** "ami-"

error\_message **=** "The image\_id value must be a valid AMI id, starting with \"ami-\"."

}

}