# Setting Up the Terraform Environment

Before you start writing the Terraform configuration file, it is necessary to install and configure a local development environment. This development environment will allow Terraform's configuration file to be written and validated as it is developed.

In the recipes in this chapter, we will learn how to download and install Terraform manually on a Windows machine.

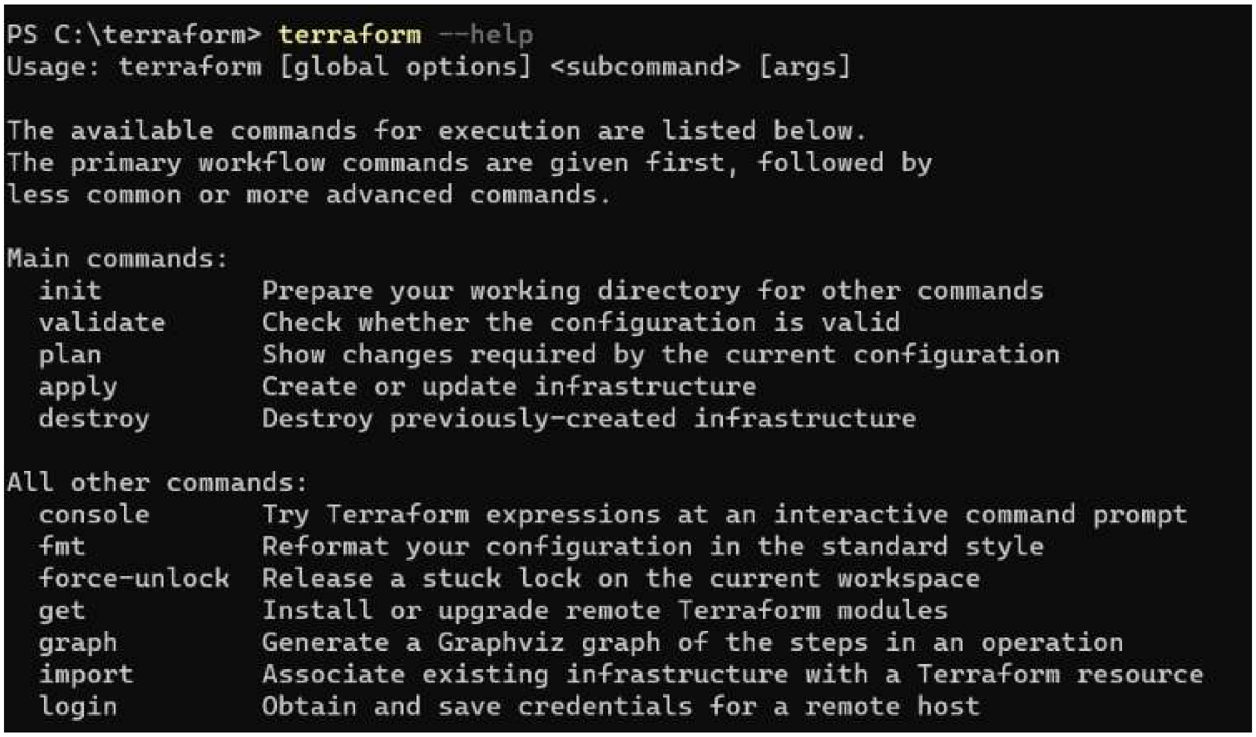
## Downloading and installing Terraform

1. Create a folder called `terraform`. We will use this to store the Terraform binary; for example, `C:/terraform`
2. Go to https://www.terraform.io/downloads.html
3. Click on windows 64-bit, which will download the Terraform ZIP package.
4. Unzip the downloaded ZIP file into the `terraform` folder that was created in step 1.
5. configure the Path environment variable by adding the path of the Terraform binary folder; for example, `C:/terraform`

After completing all these steps, we can check that Terraform is working properly by opening a command-line terminal and executing the following command:

```
terraform --help
```

The result of executing the preceding command is shown in the following screenshot:



```
PS C:\terraform> terraform --help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
  import        Associate existing infrastructure with a Terraform resource
  login         Obtain and save credentials for a remote host
```

By doing this, the list of Terraform commands will be displayed in the terminal.

# AWS EC2 Instance

Terraform code is written in a language called HCL in files with the extension `.tf`. It is a declarative language, so your goal is to describe the infrastructure you want, and Terraform will figure out how to create it. Terraform can create infrastructure across a wide variety of platforms using `providers`, including AWS, Azure, GCP and many others.

## Create a AWS EC2 instance

1. Create a file called `main.tf` and put the following code in it:

```
provider "aws" {
  region = "us-east-2"
}
```

This tells Terraform that you are going to be using the **AWS** provider and that you wish to deploy your infrastructure in the **us-east-2** region

2. Add the following code to `main.tf`, which uses the `aws_instance` resource to deploy an EC2 Instance

```
resource "aws_instance" "hello" {
  ami           = "ami-0d51ghh9cbfagedf68"
  instance_type = "t2.micro"
}
```

The general syntax for a Terraform resource is:

```
resource "<PROVIDER>_<TYPE>" "<NAME>" {
 [CONFIG …]
}
```

Where **PROVIDER** is the name of a provider (e.g., aws), **TYPE** is the type of resources to create in that provider (e.g., instance), **NAME** is an identifier you can use throughout the Terraform code to refer to this resource (e.g., hello), and **CONFIG** consists of one or more arguments that are specific to that resource (e.g., ami = "**ami-0d51ghh9cbfagedf68**").

3. In a terminal, go into the folder where you created `main.tf`, and run the `terraform init`

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws"
....
Terraform has been successfully initialized!
```

`terraform init` will tell Terraform to scan the code, figure out what providers you're using, and download the code for them. By default, the provider code will be downloaded into a `.terraform` folder.

4. Now that you have the provider code downloaded, run the `terraform plan` command:

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
(...)
```

```
+ aws_instance.example
    ami:                      "ami-4567803n"
    availability_zone:        "<computed>"
    ebs_block_device.#:       "<computed>"
    ephemeral_block_device.#: "<computed>"
    instance_state:           "<computed>"
    instance_type:            "t2.micro"
    key_name:                 "<computed>"
    network_interface_id:     "<computed>"
    placement_group:          "<computed>"
    private_dns:              "<computed>"
    private_ip:               "<computed>"
    public_dns:               "<computed>"
    public_ip:                "<computed>"
    root_block_device.#:      "<computed>"
    security_groups.#:        "<computed>"
    source_dest_check:        "true"
    subnet_id:                "<computed>"
    tenancy:                  "<computed>"
    vpc_security_group_ids.#: "<computed>"
Plan: 1 to add, 0 to change, 0 to destroy.
```

The plan command lets you see what Terraform will do before actually doing it. This is a great way to sanity check your changes before unleashing them onto the world. The output of the plan command is a little like the output of the diff command: resources with a `plus sign (+)` are going to be created, resources with a `minus sign (-)` are going to be deleted, and resources with a `tilde sign (~)` are going to be modified in-place. In the output above, you can see that Terraform is planning on creating a single EC2 Instance and nothing else, which is exactly what we want.

5. To actually create the instance, run the `terraform apply` command:

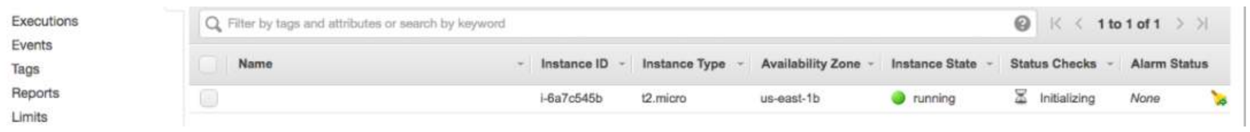
```
$ terraform apply
(...)
Terraform will perform the following actions:
  # aws_instance.example will be created
  + resource "aws_instance" "hello" {
      + ami                          = "ami-0c55b159cbfafe1f0"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)
      + key_name                     = (known after apply)
      (...)
  }
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
Enter a value:
```

You'll notice that the `terraform apply` command shows you the same plan output and asks you to confirm if you actually want to proceed with this plan. So while plan is available as a separate command, it's mainly useful for quick sanity checks and during code reviews, and most of the time you'll run apply directly and review the plan output it shows you.

Type in "yes" and hit enter to deploy the EC2 Instance:

```
Do you want to perform these actions?
 Terraform will perform the actions described above.
 Only 'yes' will be accepted to approve.
Enter a value: yes
aws_instance.example: Creating…
aws_instance.example: Still creating… [10s elapsed]
aws_instance.example: Still creating… [20s elapsed]
aws_instance.example: Creation complete after 26s
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Congrats, you've just deployed a server with Terraform! To verify this, you can login to the EC2 console, and you'll see something like this:



| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alarm Status |
|---|---|---|---|---|---|---|
| | i-6a7c545b | t2.micro | us-east-1b | running | Initializing | None |