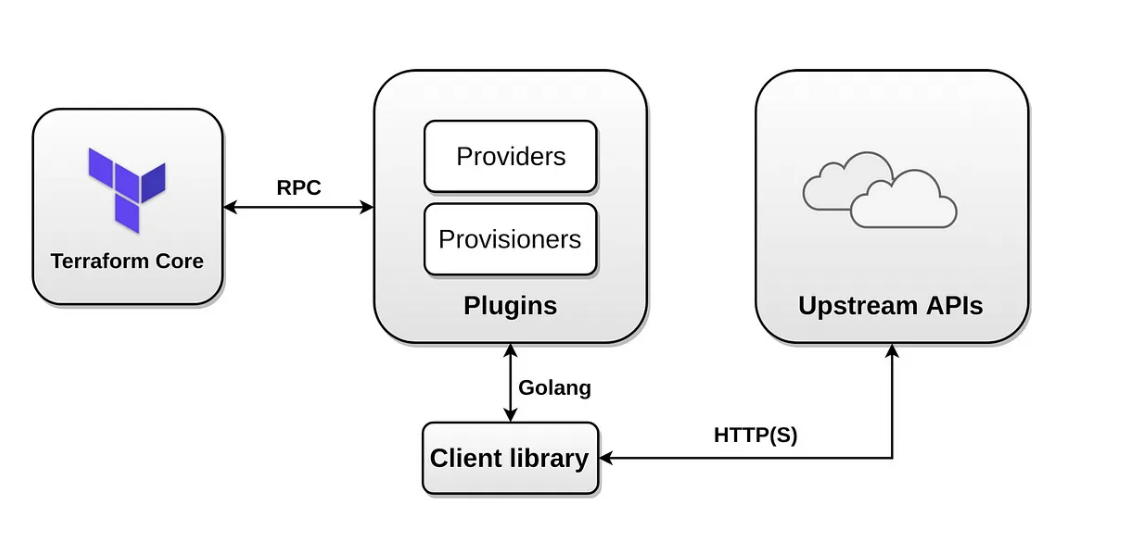
**What is Terraform**

Terraform is a well-known IaC tool in the DevOps sector, but I want to call it a Lifecycle-as-Code tool because, in addition to Infrastructure, you can manage the lifecycle of a multitude of applications like Grafana, Elasticsearch, Gitlab, Jenkins, etc. With the aid of Terraform and its providers, you can focus on writing codes and leave the deployment and management phase to the Terraform. Nowadays, the Terraform has become one of the most important parts of the DevOps lifecycle.

# Terraform Architecture:

Terraform is a plugin-based tool. So, it has a core application, terraform, and hundreds of plugins. The core application provides a unified layer to manage the IaC codes, and it’s responsible for installing required plugins, invoking them, managing state, etc. On the other hand, plugins are here to communicate with real infrastructure platforms and applications like AWS, GCP, Azure, Grafana, Jenkins, Gitlab, etc. Some of these plugins are maintained officially by the HashiCorp team, and some of them are maintained by third parties. Everyone can write and distribute his own plugin.



Terraform plugins are separated into two types: **Providers** and **Provisioners**. The Provider is a plugin which is responsible for connecting to the real infrastructure or application through the API and creating, modifying, and deleting objects and resources, and the Provisioner is a plugin which is responsible for connecting to a provisioned infrastructure and making some changes to it. For example, you can use the AWS PROVIDER to provision an EC2 instance and use remote-exec PROVISIONER to execute some commands on it after provisioning.

# Start it NOW, Do with Terraform:

I intend to explain the Terraform using the HCL language. So let’s get started. Look at the following code.

|  |  |
| --- | --- |
|  | resource "local\_file" "myfile" { |
|  | filename = "myfile.txt" |
|  | content = "Hello from Terraform." |
|  | } |

All terraform codes should be written in files with a**.tf**extension, usually main.tf for small and non-modular codes. Each terraform file may consist of various code blocks like resource, data, variable, output, etc. For example, in the above code, we have one resource block. All terraform blocks use the following coding convention.

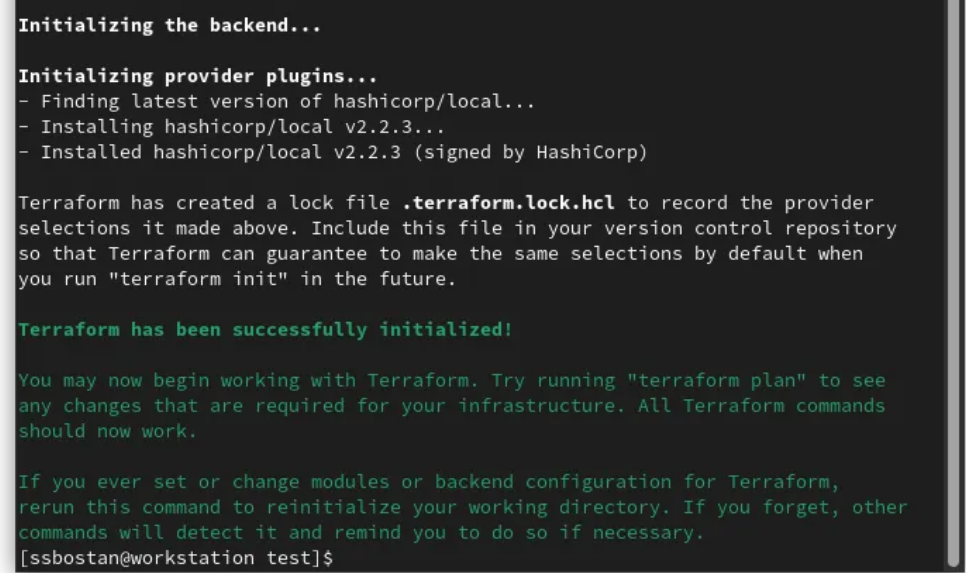
<BLOCK TYPE> [<BLOCK PARAMETER1> <BLOCK PARAMETER2> ...] {  
<ARGUMENT NAME1> = <ARGUMENT VALUE1>  
<ARGUMENT NAME2> = <ARGUMENT VALUE2>  
....  
<INNER BLOCK> {  
<ARGUMENT NAME1> = <ARGUMENT VALUE1>  
<ARGUMENT NAME2> = <ARGUMENT VALUE2>  
}  
}

Each block is identified by its type and may have zero or more parameters. For example, the terraform block has no parameters, the variable block has one parameter, and the resource block has two parameters. Each block may have zero or more arguments and inner blocks. For example, on the above code intro1.tf, we have one resource block has two parameters and two arguments.

|  |
| --- |
| Variable"myfile\_content" { |
|  | type = string |
|  | description = "Content of myfile.txt for test" |
|  | default = "Hello from Terraform." |
|  | } |
|  |  |
|  | resource "local\_file" "myfile" { |
|  | filename = "myfile.txt" |
|  | content = var.myfile\_content |
|  | } |
|  |  |
|  | output "myfile\_id" { |
|  | value = local\_file.myfile.id |
|  | } |

Save the above code into main.tf file and run the following commands.

terraform init

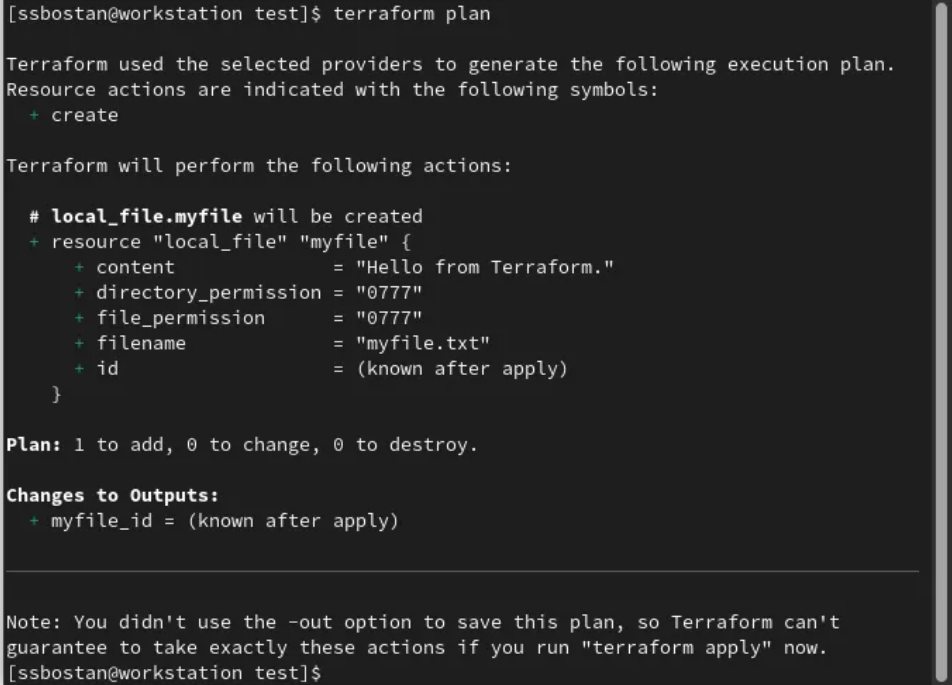


After running the **init** command, you can see the .terraform directory and .terraform.lock.hcl next to your main.tf file.



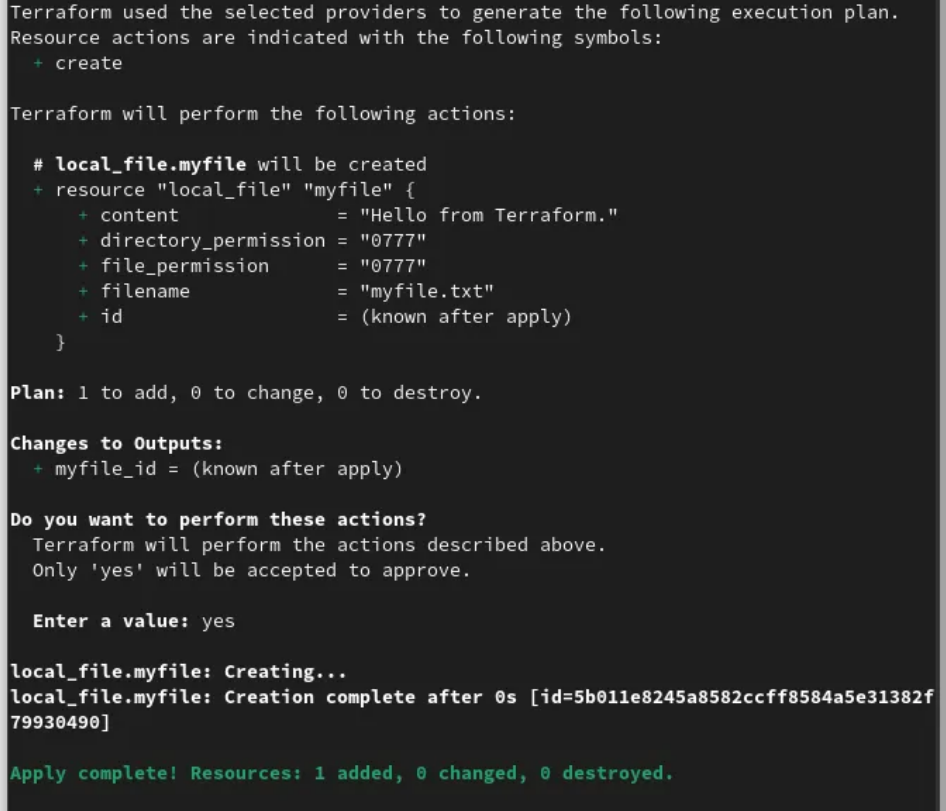
After initialization, run the plan command.

terraform plan

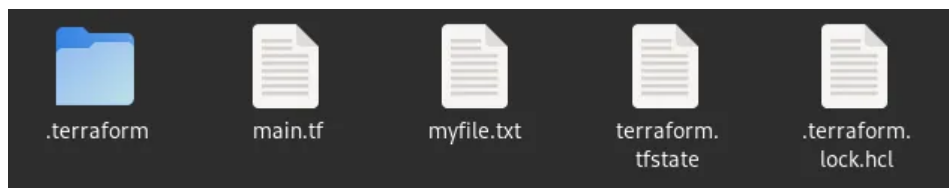


And then run the apply command.

terraform apply



After running the apply command, you can see myfile.txt as well as terraform.tfstate file next to previous files.



By running the terraform init command, the terraform initializes the current working directory by creating initial files, loading the remote state, downloading plugins, modules, etc. All used plugins in the current project will be downloaded to the .terraform directory and the .terraform.lock.hcl points to the version of used plugins as well as their signatures.

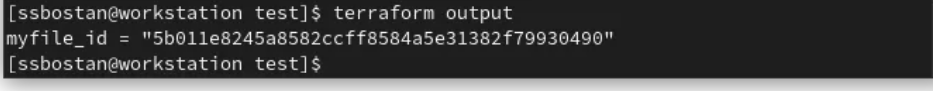
If you use a remote backend to store your Terraform state file, terraform.tfstate, the init command will load it during the initialization process.

By running the terraform plan command, you can check what terraform intends to do, create/destroy, etc. With this command, you can see what will happen next. The plan command does not change anything in the real infrastructure. You can apply your changes without running the plan step. It’s just for check before making an action.

And finally, by running the terraform apply command, the Terraform will make a real action, and the resource will be created/destroyed on the remote side, real infrastructure. By running this command, you will see what will be changed, just like the **plan** command, and after accepting it, the deployment process is started, and after that, Terraform makes the terraform.tfstate file, which is the important file of the Terraform ecosystem.

IaC code will create a file called myfile.txt with the specified content. The content is read from the defined variable, and after creating the file, the id of that will be shown as output.

terraform output



If we want to destroy the infrastructure or any previously deployed resources, we can use destroy command. With the destroy command, all Terraform deployed resources will be destroyed. Please note that if you destroy something, you can’t recover it with Terraform. If something is destroyed, it’s completely gone.

terraform destroy

