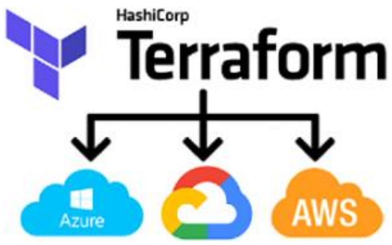
16 June 2023 23:11

What is Terraform?

Terraform is an open source “Infrastructure as Code” tool, created by HashiCorp.

A *declarative* coding tool, Terraform enables developers to use a high-level configuration language called HCL (HashiCorp Configuration Language) to describe the desired “end-state” cloud or on-premises infrastructure for running an application. It then generates a plan for reaching that end-state and executes the plan to provision the infrastructure.

Because Terraform uses a simple syntax, can provision infrastructure across multiple cloud and on-premises data centers, and can safely and efficiently re-provision infrastructure in response to configuration changes, it is currently one of the most popular infrastructure automation tools available. If your organization plans to deploy a hybrid cloud or multicloud environment, you’ll likely want or need to get to know Terraform.

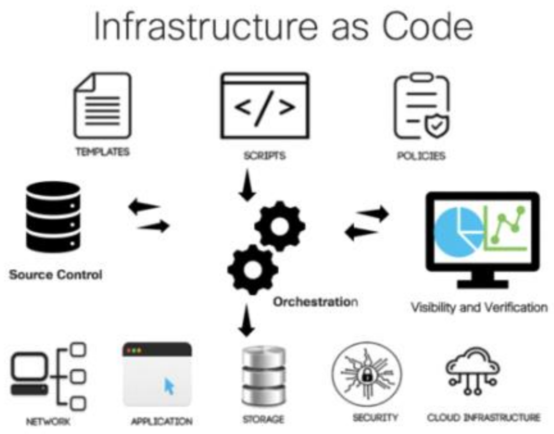


Introduction - Terraform by HashiCorp

Welcome to the intro guide to Terraform! This guide is the best place to start with Terraform. We cover what Terraform…

Why Infrastructure as Code (IaC)?

To better understand the advantages of Terraform, it helps to first understand the benefits of Infrastructure as Code (IaC). IaC allows developers to codify infrastructure in a way that makes provisioning automated, faster, and repeatable. It’s a key component of Agile and DevOps practices such as version control, continuous integration, and continuous deployment.



Infrastructure as code can help with the following:

- **Improve speed:** Automation is faster than manually navigating an interface when you need to deploy and/or connect resources.
- **Improve reliability:** If your infrastructure is large, it becomes easy to misconfigure a resource or provision services in the wrong order. With IaC, the resources are always provisioned and configured exactly as declared.
- **Prevent configuration drift:** Configuration drift occurs when the configuration that provisioned your environment no longer matches the actual environment. (See ‘Immutable infrastructure’ below.)
- **Support experimentation, testing, and optimization:** Because Infrastructure as Code makes provisioning new infrastructure so much faster and easier, you can make and test experimental changes without investing lots of time and resources; and if you like the results, you can quickly scale up the new infrastructure for production.

Why Terraform?

There are a few key reasons developers choose to use Terraform over other Infrastructure as Code tools:

- **Open source:** Terraform is backed by large communities of contributors who build plugins to the platform. Regardless of which cloud provider you use, it’s easy to find plugins, extensions, and professional support. This also means Terraform evolves quickly, with new benefits and improvements added consistently.

- **Platform agnostic:** Meaning you can use it with *any* cloud services provider. Most other IaC tools are designed to work with single cloud provider.
- **Immutable infrastructure:** Most Infrastructure as Code tools create *mutable* infrastructure, meaning the infrastructure can change to accommodate changes such as a middleware upgrade or new storage server. The danger with mutable infrastructure is *configuration drift* — as the changes pile up, the actual provisioning of different servers or other infrastructure elements ‘drifts’ further from the original configuration, making bugs or performance issues difficult to diagnose and correct. Terraform provisions *immutable infrastructure*, which means that with each change to the environment, the current configuration is replaced with a new one that accounts for the change, and the infrastructure is reprovisioned. Even better, previous configurations can be retained as versions to enable rollbacks if necessary or desired.

Install Terraform

Using the link below you can download the Terraform setup according to your OS.

terraform.io/downloads

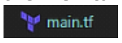
If you’re a windows user, you can follow this video tutorial to install Terraform successfully.

How to install Terraform in Windows 11?

After install you can check your terraform version using the `terraform –version` Command in the command prompt. If it shows 1.2.5 or above, congratulations, you installed Terraform successfully.

Create Terraform File

You can create a new terraform file like “YouFileName.tf”. If you use your favorite code editor for that you can see the terraform logo in front of the file name.


main.tf

Syntax

Let’s learn how to write Terraform code. Terraform consists of only a few basic elements.

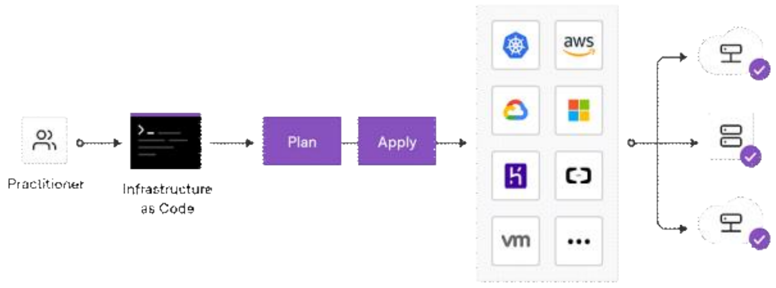
```
<BLOCK TYPE> “<BLOCK LABEL>” “<BLOCK LABEL>” {


# Block body


<IDENTIFIER> = <EXPRESSION> # Argument


}
```

There are several Block types like “resource”, “variable”, “output” and so on. You can use them according to your need. Block labels vary by cloud platforms. The other Block Label is used to name the resource. It can be defined by the user. Identifiers may vary by resource. The user can assign values to the identifiers. You can go through the following example.

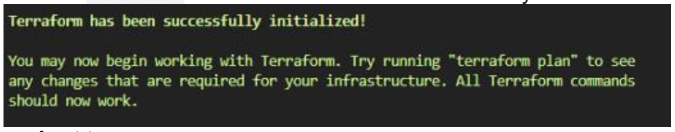
```
resource "azurerm_resource_group" "main" {  name     = "test-resource-group"  location = "centralus"  tags = {    environment = "dev"    source = "terraform"  }}
```

This is an example for configure a resource group for Azure.

RUN

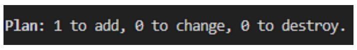


Now you can test your terraform code. First, we have to initialize our working directory containing Terraform configuration files. For that we can use the `terraform init` Command. After a few seconds you can see the “.terraform.lock.hcl” file with a success message.

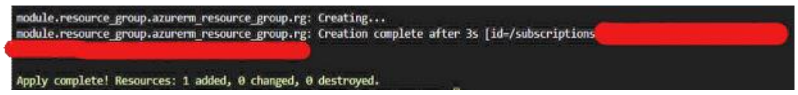

```
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

terraform init success message

Then you can hit the `terraform plan` Command to determine the desired state of all the resources it declares. If there are any errors you can see them in the terminal with line numbers. Else you can see how many resources that you added in the terminal.


```
Plan: 1 to add, 0 to change, 0 to destroy.
```

After all these configurations now you can deploy your first cloud resource in your favorite cloud platform using the `terraform apply` Command.

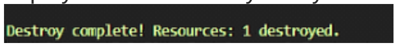


```
module.resource_group.azurerm_resource_group.rg: Creating...
module.resource_group.azurerm_resource_group.rg: Creation complete after 3s [id=/subscriptions[redacted]
[redacted]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Congratulations, you deploy your cloud resources in your favorite cloud platform using Terraform.

Destroy

If you don’t want to keep running your resources in the cloud platform, then you should hit the `terraform destroy`Command and delete all your deployed resources very easily.



```
Destroy complete! Resources: 1 destroyed.
```

In the next article, I will show how to deploy some resources to the Azure cloud platform with better folder structure and some good practices.