**TERRAFORM IN MICROSOFT AZURE**

**Terraform:**

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions.

Configuration files describe to Terraform the components needed to run a single application or your entire datacentre. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, terraform can determine what changed and create incremental execution plans which can be applied.

The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

[https://www.terraform.io/intro/index.html#what-is-terraform](https://www.terraform.io/intro/index.html#what-is-terraform-)

**The key features of Terraform are:**

* Infrastructure as Code
* Resource Graph
* Change Automation
* Execution Plans

To implement terraform in azure we need to write a terraform configuration file which uses Hashicorp configuration language (HCL). The azure provisioner for terraform is known as Azurerm.

The prerequisites for using terraform are

* An azure account
* A system with terraform 0.12.6 or later installed

**Note:** Azure Cloud Shell is the most convenient platform to use terraform. Everything you need is preinstalled, and Cloud Shell uses the credentials of the signed-in Azure user. Most of the procedures assume that you are using Cloud Shell, either from the Azure Portal or standalone

**Installing terraform:**

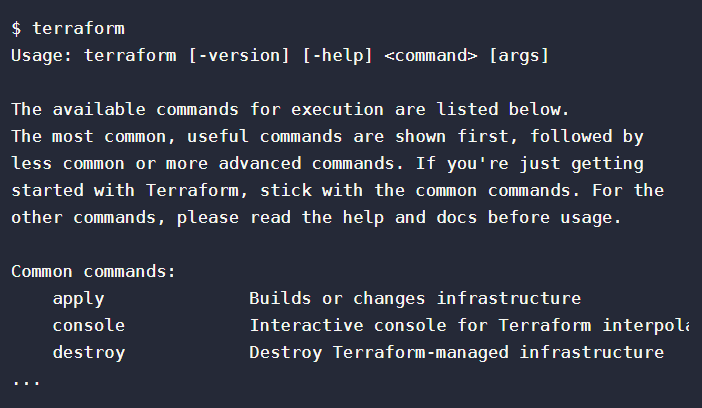
If u choose not to use cloud shell, then a terraform must be installed in the target system.

To install terraform on any supported system:

* Find the appropriate [Terraform distribution package](https://www.terraform.io/downloads.html) for your system and download it. Terraform is distributed as a single .zip file.
* After downloading Terraform, unzip the package to a directory of your choosing. Terraform runs as a single binary named terraform. Any other files in the package can be safely removed and Terraform will still function.
* Optional but highly recommended: modify the path to include the directory that contains the Terraform binary.
* [How to set the $PATH on Linux and Mac](https://stackoverflow.com/questions/14637979/how-to-permanently-set-path-on-linux)
* [How to set the PATH on Windows](https://stackoverflow.com/questions/1618280/where-can-i-set-path-to-make-exe-on-windows)

An alternative to modifying the path is to move the Terraform executable to a directory that is normally included in the path by default, for example: sudo mv terraform /user/local/bin.

To check if the terraform installed in the system type terraform in your terminal to see



Note: If you get an error that terraform could not be found, your PATH environment variable was not set up properly. Please go back and ensure that your PATH variable contains the directory where the Terraform binary was installed.

**Create configuration file:**

The set of files used to describe infrastructure in Terraform is known as a Terraform **configuration**.

Terraform configurations are made up of one or more files in a directory. By default, this directory will also contain provider binaries, plan file, and state files once Terraform has run the configuration.

Terraform uses a declarative model for defining infrastructure: You write a configuration that declares your desired state and then leave it up to Terraform and the Azure provider to determine how to create and configure Azure to match the desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.

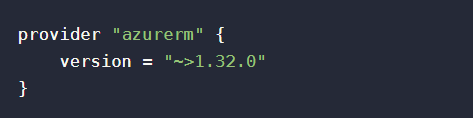
So, the components of this configuration file in HCL(.tf) file includes:

**Providers:** Terraform can configure resources across multiple clouds. For example, a single configuration can span both Azure and AWS. In such cases, there needs to be a way for Terraform to know how to manage each cloud. This is where cloud providers come in. Each cloud provider can have a providerblock present in the configuration.

provider "azurerm" {

version = "~>1.32.0"

}



**Note:** Here in version ~> pessimistic operator which tells >= 1.32.0 and <1.33.0.

**Authentication:** Authentication is configured in the provider block. The following example shows a configuration for authentication to Azure using a managed identity.

provider "azurerm" {

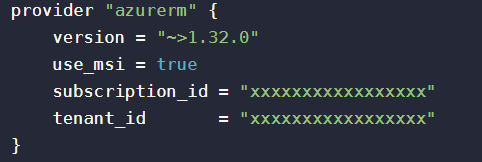
version = "~>1.32.0"

use\_msi = true

subscription\_id = "xxxxxxxxxxxxxxxxx"

tenant\_id = "xxxxxxxxxxxxxxxxx"

}



**Resources:**

A resource block defines the desired state for a given resource within the infrastructure. A resource can be a physical component such as a network interface, or it can be a logical resource such as an application.

A resource block has two string parameters before opening the block: the resource type (first parameter) and the resource name (second parameter). The combination of the type and name must be unique in the configuration.

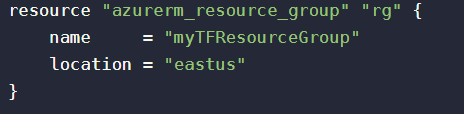
The following example uses a resource block to provision a new Azure resource group. A resource group is a fundamental object in the Azure Resource Manager (ARM) deployment model, and it is required to create, modify, or destroy infrastructure in ARM. A configuration can create a new resource group or use an existing group, and often does both.

resource "azurerm\_resource\_group" "rg" {

name = "myTFResourceGroup"

location = "eastus"

}



**Note:** In this example, the resource type is [azurerm\_resource\_group](https://www.terraform.io/docs/providers/azurerm/r/resource_group.html) and the name is "rg". The resource name is used to refer to the object created in the resource block throughout the configuration. It is not the same as the name of the resource group in Azure.

**Complete configuration: (.tf)**

*# Configure the Azure provider*

provider "azurerm" {

version = "~>1.32.0"

use\_msi = true

subscription\_id = "xxxxxxxxxxxxxxxxx"

tenant\_id = "xxxxxxxxxxxxxxxxx"

}

*# Create a new resource group*

resource "azurerm\_resource\_group" "rg" {

name = "myTFResourceGroup"

location = "eastus"

}

**Building the infrastructure in cloud using this configuration file:**

There are three steps to build infrastructure with Terraform:

1. Initialize the Terraform configuration directory using terraform init
2. Create an execution plan using terraform plan
3. Create or modify infrastructure using terraform apply

**INIT:**

Use command “terraform init” to run in the configuration directory. Init will create a hidden directory .terraform and download plugins as needed by the configuration. Init also configures the backend for Terraform state, an important detail that we'll come back to later.

