**Basic Terraform Install and vSphere Vmware automation to deploy clusters.**

One of the great tools in automation and DevOps today is [Hashicorp’s Terraform](https://www.terraform.io/" \t "_blank). Terraform epitomizes the “infrastructure-as-code” design paradigm and is a great way to provide powerful automation in any environment ranging from on-premise infrastructure to public cloud environments. One of the many advantages of utilizing Terraform for automation is the simplicity with which you can it get up and running . Terraform provides a great way to easily getting started with basic infrastructure automation capabilities. There are a host of [providers](https://www.terraform.io/docs/providers/index.html) that are available on the terraform website. Let’s take a look at basic Terraform installation and VMware vSphere Automation to deploy clustered vms. We will use the latest vSphere provider ( 1.6 ) to enable this cluster deployment. In this article i will demonstrate on how to use Terraform and it’s resource definitions to deploy anti-colocated vm’s ( two vms to be deployed on a single esx cluster but will *never* be on the same host)

**Terraform — The basics**

While most of the information about Terraform can be obtained from the [website](http://terraform.io/), i will touch on some salient points about it.

Terraform is a declarative language that allows you to tell the system what you want the end infrastructure to look like, and Terraform will create or change it to fit that declared state. While this is a good thing ,you will find out that it has certain disadvantages as well ( lack of if-else , conditional paths, sub-resource constraints to name a few),but rest be assured that the terraform team is working to enrich this capability. Terraform has taken away the brunt of reading through reams of api documentation and giving the power to a system administrator / devops user the capability to define what they need in pure template ( aka text ) form. This template is built in Hashicorp Configuration Language or HCL with an extension of .tf that terraform will process as it runs through the deployment.

Terraform — Moving forward

The following concepts are key to defining a template.

1. Provider — Every platform or capability that you need to manage. Providers can be built for managing a platform ( AWS , Azure , IBM Softlayer ) or Network ( NSX , Palo Alto etc) . You can also build your own provider say to manage your DNS or Custom Storage Services . More on that later.
2. Resource — The type of object you want to create, manage, etc (servers, networks, dns , and load balancers.
3. Modules — These are reusable artifacts that be embedded in a template.
4. Variables — These are input /output values that the terraform template will process to execute the deploy . E.g vmname, cluster etc.

Once a template is defined, there are two actions to be run on them ,

* **Plan** — This action informs the user what “changes” Terraform will perform in your environment, and also is a **great** sanity check . The plan “verb” will tell you what the template will execute , so be sure you read through it before doing this in a production environment.
* **Apply** — Actually applies the creation or changes to an environment.

**Let’s get Rolling.**

Terraform install is a pretty straightforward process. Download it from [here](https://www.terraform.io/downloads.html). Please choose the right platform. I have this running on a mac. Make sure that the terraform binary is in your default path.

Run terraform init so that we can initialize the terraform engine and the providers

Text

Description automatically generated

Running Terraform init to kickstart the providers.

This screen shows that post init, the following providers were intialized.

Text

Description automatically generated

As you can see, i also have the vsphere provider at version 1.6. Terraform allows you to use / reference multiple providers for performing functions.

In my TF template i could either choose to use the exact version of this provider ,which you should in production since , TF will upgrade it to the latest.

Graphical user interface, text

Description automatically generated

Resource Definitions in the template.

A picture containing text

Description automatically generated

Create a virtual disk resource

Graphical user interface, text

Description automatically generated

Create a virtual resource — vm1

Text

Description automatically generated

Create a virtual resource — vm2

Text

Description automatically generated

Create an anti-affinity resource

Since we defined these resources in our terraform template , we can then run a quick terraform plan , which will help us check 1) Are the template definitions valid ? 2) Actions and Resources that terraform will create . Attaching the output and trimming it for brevity.

Text

Description automatically generated

Terraform Plan out ( trimmed for brevity).

As you can see here , the plan displays that 4 resources will be created (added) . The top resource is the “anti-affinity” rule for our deployment. Once we validated the plan , we can go ahead and do an apply which will initiate the execution of this plan.

Text

Description automatically generated

Terrform Apply Run output (trimmed for brevity)

Graphical user interface, text, application

Description automatically generated

VM + Disk Creation Status ..

When the vm’s are successfully created , tf will exit with a “Apply Complete” message. You can also login to vcenter and check if the vm’s have been created . We can also validate that the provider honored our request for anti-colocation and create rule in vmware .

Graphical user interface, text, application, email

Description automatically generated

Anti-Affinity Rule Created by Terraform

As you can see , all that we were required to built an anti-colocation cluster was to define a template and TF will do the rest.