**Terraform About**

Terraform is an infrastructure as code tool that lets you build, change, and version infrastructure safely and efficiently. This includes low-level components like compute instances, storage, and networking, as well as high-level components like DNS entries and SaaS features.

**What is Terraform?**

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

=========

**Terraform Cheat Sheet**

Terraform command accepts a variety of subcommands such as terraform init or terraform plan. Some of the widely used commands and subcommands I have listed below.

**Show version**

terraform –version Shows terraform version installed

**Initialize infrastructure**

terraform init Initialize a working directory

terraform init -input=true Ask for input if necessary

terraform init -lock=false Disable locking of state files during state-related operations

**Get**

terraform get downloads and update modules mentioned in the root module

terraform get -update=true modules already downloaded will be checked for updates and updated

**Provision infrastructure**

terraform plan Creates an execution plan (dry run)

terraform plan -out=path save generated plan output as a file

terraform plan -destroy Outputs a destroy plan

terraform apply Executes changes to the actual environment

terraform apply –auto-approve Apply changes without being prompted to enter ”yes”

terraform apply -refresh=true Update the state for each resource prior to planning and applying

terraform apply -input=false Ask for input for variables if not directly set

terraform apply -var ‘foo=bar’ Set a variable in the Terraform configuration, can be used multiple times

terraform apply -var-file=foo Specify a file that contains key/value pairs for variable values

terraform apply -target Only apply/deploy changes to the targeted resource

terraform destroy –auto-approve Destroy/cleanup without being prompted to enter ”yes”

terraform destroy -target Only destroy the targeted resource and its dependencies

**Terraform Workspaces**

terraform workspace new Create a new workspace and select it

terraform workspace select Select an existing workspace

terraform workspace list List the existing workspaces

terraform workspace show Show the name of the current workspace

terraform workspace delete Delete an empty workspace

**Format and validate Terraform Code**

terraform fmt Format code as per HCL canonical standard

terraform validate validate configuration files for syntax

**Inspect Infrastructure**

terraform graph creates a resource graph listing all resources in your configuration and their dependencies.

terraform output List all the outputs for the root module

terraform output instance\_public\_ip List only the specified output

terraform output -json List all the outputs in JSON format

terraform show provide human-readable output from a state or plan file

First, install the required dependencies using the following command:

apt-get install wget curl unzip software-properties-common gnupg2 -y

Next, download and add the HashiCorp signed gpg keys to your system:

curl -fsSL https://apt.releases.hashicorp.com/gpg | apt-key add -

Next, add the HashiCorp repository to the APT using the following command:

apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"

Next, update the repository using the command given below:

apt-get update -y

Finally, install the Terraform by running the following command:

apt-get install terraform -y

Once the Terraform has been installed, verify it using the following command:

terraform -v

You will get the Terraform version in the following output:

Terraform v1.1.2

on linux\_amd64

Install Azcli

Step 1: Get required packages for Azure CLI installation

sudo apt-get update

sudo apt-get install ca-certificates curl apt-transport-https lsb-release gnupg

Step 2: Download and install Microsoft signing key

curl -sL https://packages.microsoft.com/keys/microsoft.asc |

gpg --dearmor |

sudo tee /etc/apt/trusted.gpg.d/microsoft.gpg > /dev/null

Step 3: Add Azure CLI software repository

AZ\_REPO=$(lsb\_release -cs)

echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/ $AZ\_REPO main" |

sudo tee /etc/apt/sources.list.d/azure-cli.list

Step 4: Update repository information

sudo apt-get update

Step 5: Install the azure-cli package

sudo apt-get install azure-cli

Step 6: Verify the installation

az login

On successful installation above command should open Azure sign-in page in default browser.

===================================================

Test Case,

Create storage account using azcli command or create from portal.

az group create --location $location --name $resourceGroupName

az storage account create --name $storageAccountName --resource-group $resourceGroupName --location $location --sku Standard\_LRS

az storage container create --name state --account-name $storageAccountName

az storage account blob-service-properties update --account-name $storageAccountName --enable-change-feed --enable-versioning true

**Create AKS Cluster with windows node using terraform.**

Create File.

root@terrafromsenthil:~/akswin# tree

.

**├── app.yaml**

**├── main.tf**

**└── variables.tf**

Main.tf

vi main.tf

terraform {

required\_providers {

azurerm = {

source = "hashicorp/azurerm"

version = "3.12.0"

}

}

}

provider "azurerm" {

features {}

}

resource "azurerm\_resource\_group" "test-rs-santhil" {

name = "test-rs-santhil"

location = var.location

}

resource "azurerm\_virtual\_network" "vnet" {

name = "virtualNetwork1"

location = "eastus"

resource\_group\_name = "test-rs-santhil"

address\_space = ["10.1.0.0/16"]

subnet {

name = "subnet1"

address\_prefix = "10.1.1.0/24"

}

}

resource "random\_password" "winnode" {

length = 16

}

resource "azurerm\_kubernetes\_cluster" "aks" {

name = "aks1"

location = "eastus"

resource\_group\_name = "test-rs-santhil"

dns\_prefix = "aks1"

default\_node\_pool {

name = "lin"

node\_count = var.node\_count\_linux

vm\_size = "Standard\_D2\_v2"

vnet\_subnet\_id = element(tolist(azurerm\_virtual\_network.vnet.subnet),0).id

}

windows\_profile {

admin\_username = "azadmin"

admin\_password = random\_password.winnode.result

}

network\_profile {

network\_plugin = "azure"

}

identity {

type = "SystemAssigned"

}

}

resource "azurerm\_kubernetes\_cluster\_node\_pool" "win" {

name = "win"

kubernetes\_cluster\_id = azurerm\_kubernetes\_cluster.aks.id

vm\_size = "Standard\_DS2\_v2"

node\_count = var.node\_count\_windows

os\_type = "Windows"

}

#output "kube\_config" {

# value = azurerm\_kubernetes\_cluster.aks.kube\_config\_raw

#}

Create File

Vi variables.tf

variable "resource\_group" {

type = string

description = "test-rs-santhil"

default = "aksrg"

}

variable "location" {

type = string

description = "Location of the resources"

default = "eastus"

}

variable "node\_count\_linux" {

type = number

description = "Linux"

default = 1

}

variable "node\_count\_windows" {

type = number

description = "Windows"

default = 1

}

Create application yaml

Vi

app.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample

labels:

app: sample

spec:

replicas: 1

template:

metadata:

name: sample

labels:

app: sample

spec:

nodeSelector:

"kubernetes.io/os": windows

containers:

- name: sample

image: mcr.microsoft.com/dotnet/framework/samples:aspnetapp

resources:

limits:

cpu: 1

memory: 800M

ports:

- containerPort: 80

selector:

matchLabels:

app: sample

---

apiVersion: v1

kind: Service

metadata:

name: sample

spec:

type: LoadBalancer

ports:

- protocol: TCP

port: 80

selector:

app: sample

Terrafrom Command

**Provision infrastructure**

**terraform init -Initialize a working directory**

**terraform plan - Creates an execution plan (dry run)**

**terraform apply -Executes changes to the actual environment**

===========================

root@NCSUSNAUJPWS01:~/akswin# kubectl get node

NAME STATUS AGE VERSION

aks-lin-10005819-vmss000000 Ready 2h v1.26.6

akswin000000 Ready 1h v1.26.6

root@NCSUSNAUJPWS01:~/akswin#

root@NCSUSNAUJPWS01:~/akswin# kubectl get pod

NAME READY STATUS RESTARTS AGE

sample-65c7dcd9b7-pfsmk 1/1 Running 0 13m

root@NCSUSNAUJPWS01:~/akswin# kubectl get svc

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes 10.0.0.1 <none> 443/TCP 2h

sample 10.0.66.1 20.81.124.67 80:31805/TCP 13m

