TERRAFORM DOCUMENTATION

Mithun Technologies, +91 99809 23226, devopstrainingblr@gmail.com

Mithun Technologies +91-9980923226	Terraform devopstrainingblr@gmail.com	Author	Mithun Technologies
		Web site	http://mithuntechnologies.com

Terraform

Introduction of Terraform

Terraform is an open-source, infrastructure as code software (IaC) tool, maintain the infrastructure change history using VCS like Git, created by HashiCorp and written in the Go programming language.

Infrastructure as code is the process of managing infrastructure in a file or files, rather than manually configuring resources in a user interface (UI).

Here resources are nothing but virtual machines, Elastic IP, Security Groups, Network Interfaces...

Terraform code is written in the HashiCorp Configuration Language (HCL) in files with the extension .tf

Terraform allows users to use HashiCorp Configuration Language (HCL) to create the files containing definitions of their desired resources on almost any provider (AWS, GCP, Azure, Digital Ocean, OpenStack, etc) and automates the creation of those resources at the time of apply.

Reference URL: https://www.terraform.io/docs/providers/index.html

Advantages of Terraform

Platform Agnostic
State Management
Operator Confidence

Difference between Terraform and Cloud Formation

Terraform	Cloud Formation
Terraform is developed by HashiCorp	Cloud Formation is Developed by AWS
It will work for many Cloud providers like AWS, Azure, GCP, Digital Ocean…	Cloud Formation will support only AWS
Terraform uses HashiCorp Configuration Language (HCL), a language built by HashiCorp. It is fully compatible with JSON.	AWS Cloud Formation utilizes either JSON or YAML. Cloud Formation has a limit of 51,000 bytes for the template body itself

Difference between Terraform and Ansible

Terraform	Ansible
Terraform is an Open Source Tool which is provided by HashiCorp.	Ansible is also an Open Source Tool.
Terraform is an Infrastructure as a Code, which means they are designed to provision the servers themselves.	Ansible is a Configuration Management Tool. Which means they are designed to install and manage software on existing servers.
Terraform is ideal for creating, managing, and improving infrastructure.	Ansible is ideal for software provisioning, application deployment, and configuration management.

Pre-Requisites

1) Any Cloud Provider (AWS, GCP, Azure, Digital Ocean, OpenStack, etc)
2) IAM User credentials (Secret Key and Access Key)

Add the following policies to IAM user if you are using AWS as cloud provider.

AmazonEC2FullAccess
AmazonS3FullAccess
AmazonDynamoDBFullAccess
AmazonRDSFullAccess
CloudWatchFullAccess
IAMFullAccess

Terraform Installation

Follow my blog for installation.

https://mithuntechnologies-devops.blogspot.com/2020/02/terraform-installation-linux-server.html

Follow my below YouTube channel video url.

https://youtu.be/kxOR-WrK4y8

Terraform First Example

Example: 1 - File Name: AwsEC2InstanceCreation.tf

vim AwsEC2InstanceCreation.tf

The first step in terraform script is typically a provider. Using provide section we can configure our desired cloud provider.

```
provider "aws"{
  region = "us-east-2"
  access_key = "AKIA4UQE3BUQ6GQ3BAFO"
  secret_key = "5LAzj2tYFxKf1NNvmvz0Z1USAoEzDAZHlc6R5wHF"
}
```

This tells Terraform that you are going to be using AWS as your provider and that you want to deploy your infrastructure into the us-east-2 region.

If you use,
export AWS_ACCESS_KEY_ID="AKIA4UQE3BUQ6GQ3BAEO"
export AWS_SECRET_ACCESS_KEY="5LAzj2tYFxKf1NNvmvz0Z1UASoEzDAZHlc6R5wHF"

No need to mention access_key and secret_key keys in provider section.

```
resource "aws_instance" "MSS" {
  ami = "ami-0a74bfeb190bd404f"
  instance_type = "t2.micro"
  key_name = "mithuntechnologies"
  security_groups = ["launch-wizard-19"]
  tags = {
     Name = "Terraform Server by Mithun Technologies"
  }
}
```

The general syntax for a Terraform resource is:

```
resource "<PROVIDER>_<TYPE>" "<NAME>" {
  [CONFIG ...]
}
```

Here PROVIDER is the name of a provider (e.g., aws), TYPE is the type of resource to create in that provider (e.g., instance),
NAME is an identifier you can use throughout the Terraform code to refer to this resource (e.g., my_instance),
and CONFIG consists of one or more arguments that are specific to that resource."

Terraform Commands

terraform init: The terraform init command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration

terraform fmt: The terraform fmt command is used to rewrite Terraform configuration files to a canonical format and style.

terraform validate: The terraform validate command validates whether a configuration is syntactically valid or not.

terraform plan: The terraform plan command is used to create an execution plan.
This command is a convenient way to check whether the execution plan for a set of changes matches your expectations
without making any changes to real resources or to the state.

terraform apply: terraform apply to actually create the infrastructure on AWS.
terraform apply -auto-approve:

terraform destroy: The terraform destroy command is used to destroy the Terraform-managed infrastructure.

terraform show:

terraform state list:

terraform graph:

Example: 2 - FileName: provider.tf

```
provider "aws"{
   region = "ap-south-1"
   access_key = "AKIA5KT2FR4HAAZW2BBC"
   secret_key = "TXrhgRBm5Zvxf+RcfTqpliVCSvuR9fuURY3tUsnY"
}
```

FileName: main.tf

```
resource "aws_instance" "AWSEC2Instance"{
   ami = "ami-0a9d27a9f4f5c0efc"
   instance_type = "t2.micro"
   security_groups = ["launch-wizard-3"]
   key_name = "devopsmss"
   tags = {
      Name = "RedHat Server by Terraform"
   }
}
```

Create Multiple Resources (By using "count" attribute)

```
count       = "5"
```

Using count argument, we can pass the number of resources we need.

Example: 3 - FileName: CountParameter.tf

```
provider "aws"{
    region = "ap-south-1"
    access_key = "AKIA5KT2FR4HAAZW2BBC"
    secret_key = "TXrhgRBm5Zvxf+RcfTqpliVCSvuR9fuURY3tUsnY"
}

resource "aws_instance" "AWSEC2Instance"{
    count = "4"
    ami = "ami-0a9d27a9f4f5c0efc"
    instance_type = "t2.micro"
    security_groups = ["launch-wizard-3"]
    key_name = "devopsmss"
    tags = {
        Name = "RedHat Server by Terraform"
    }
}
```

Variables

In real-time world, the project has more variables, it is difficult maintain the variables in terraform script, instead we will put the variables in a separate file called vars.tf as follows.

Example: 4

File Name: var.tf

```
variable "ami" {
  description = "Amazon Machine Image type.."
        default = "ami-04169656fea786776"
}

variable "instance_type" {
  description = "Instance type, weather t2.micro, t2.medium..."
        default = "t2.micro"
}

variable "instances" {
  description = "Total number of instances which we are going to create"
  default = 2
}

variable "instances" {
  description = "Total number of instances which we are going to create"
  default = 2
}

/*
variable "instances"{

}
```

//terraform apply -auto-approve -var instances="2"
*/

Here we are passing the variable value while running the script.

Now Terraform script looks like below.
File Name: main.tf


```
provider "aws" {
        region = "ap-south-1"
}

resource "aws_instance" "MT" {
        count = "${var.instances}"
        ami = "${var.ami}"
        instance_type = "${var.instance_type}"
        key_name = "mithuntechnologies"

   tags = {
     Name = "Terraform Server - MSS"
   }
}
```

Example: 5

File Name: vars.tf

```
variable "ami" {
   description = "Amazon Machine Image type.."
        default = "ami-04169656fea786776"
}

variable "instance_type" {
   description = "Instance type, weather t2.micro, t2.medium..."
        default = "t2.micro"
}

variable "instances" {
 description = "Total number of instances which we are going to create"
 default = 2
}
```

File Name: count.tf

```
provider "aws" {
 region = "ap-south-1"
 }

 resource "aws_instance" "AWSServer" {
```

```
count = "${var.instances}"
ami = "ami-052c08d70def0ac62"
instance_type = "t2.micro"
key_name = "devopsmssnovbatch"
security_groups = ["launch-wizard-7"]
tags = {
  Name = "Terraform Server - ${count.index}"
 }
}
```

Comments

The Terraform language supports three different syntaxes for comments:

a) # begins a single-line comment, ending at the end of the line.

b) // also begins a single-line comment, as an alternative to #.

c) /* and */ are start and end delimiters for a comment that might span over multiple lines.

AWS Credentials

The AWS provider offers a various method of providing credentials for authentication.
The following methods are supported.
a) Static credentials
b) Environment variables
c) Shared credentials file
d) EC2 Role

Static credentials

Static credentials can be provided by adding an access_key and secret_key in-line in the AWS provider block, as follows.

```
provider "aws" {
  region    = "ap-south-1"
  access_key = "AKIA4UQE3BUQ6GQ3BAEO"
  secret_key = "5LAzj2tYFxKf1NNvmvz0Z1UASoEzDAZHlc6R5wHF"
}
```

Environment variables

You can provide your credentials via the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY, environment variables, representing your AWS Access Key and AWS Secret Key, respectively.

```
export AWS_ACCESS_KEY_ID="accesskey"
export AWS_SECRET_ACCESS_KEY="secretkey"
export AWS_DEFAULT_REGION="ap-south-1"
```

Shared Credentials file

You can use an AWS credentials file to specify your credentials. The default location is $HOME/.aws/credentials on Linux and OS X, or "%USERPROFILE%\.aws\credentials" for Windows users. If we fail to detect credentials inline, or in the environment, terraform will check this location. You can optionally specify a different location in the configuration by providing the shared_credentials_file attribute, or in the environment with the AWS_SHARED_CREDENTIALS_FILE variable. This method also supports a profile configuration and matching AWS_PROFILE environment variable:

```
provider "aws" {
  region              = "ap-south-1"
  shared_credentials_file = "/home/mithun/.aws/creds"
  profile             = "customprofile"
}
```

Possible Errors:

Error 1

```
[mithun@mithuntechnologies iamuser]$ terraform apply

Error: error configuring Terraform AWS Provider: no valid credential sources for
 Terraform AWS Provider found.

Please see https://registry.terraform.io/providers/hashicorp/aws
for more information about providing credentials.

Error: NoCredentialProviders: no valid providers in chain. Deprecated.
        For verbose messaging see aws.Config.CredentialsChainVerboseErrors


[mithun@mithuntechnologies iamuser]$ ▌
```

Solution:
Provide Valid Access Key and Secret key.

Error 2

```
[mithun@mithuntechnologies terraformscripts]$ terraform apply

Error: error using credentials to get account ID: error calling sts:GetCallerIdentity: InvalidClientTokenId
: The security token included in the request is invalid.
        status code: 403, request id: 3f6f094d-8f8a-4ff5-bef0-4bb33130f903

  on instancecreation.tf line 1, in provider "aws":
  1: provider "aws" {


[mithun@mithuntechnologies terraformscripts]$ ▌
```

Not Valid credentials
Solution: Check the Access key and Secret Key

Error 3

```
aws_instance.AWSServers: Creating...
```

Error: Error launching source instance: UnauthorizedOperation: You are not authorized to perform this opera
tion. Encoded authorization failure message: vu3v1KzQvNc0KHh2yB5CQXh4JUOagsdwVuoVgIreOKtMyz6bXsbj2fuJeEh1VW
s3BkeyqT1J3Ai37a7dJolpvV77RXrK_bcF67ngHa1gAdkst2A_tv0TE6UhiMWttSAi4yMpRN_yleWrtHYNKMriWlyfRArvUDGJns-t0IXTq
3BghPVgXKyBiYUV-R-jQAaIaai-BybNUSTZyIz9qU31dbDGReRHLAw68Ccf_NLcSQRknC5mqlm3H2xtfr_9SFo5wEU9U1j69lNsqMJlELay
z80Vkqr3Rd2KRaon7xSsV35qd8VkmO_j6BChzQggujZFERZf9GqpmFpl23FXkkMMPfZxq4Y4RK3DnkgKNQP4nUPyVdzT7GDApVP3wU4GBFN
2LxTKGO9wma-W_6jkZxqGHRHcbFyr9zvRj041P_hRYxyfgmU86EVlSRemJnZNRtRY2EeQWbBPrqAN_vpHt2jNYgW3l5KQakC-3KbhX9S0uR
uVHrSy4OXb4emYUMI9GN3Aa1INvpB4n1OrddXSLgQuYGL3ZwKIKeWanAip71D3_I1B1buRZO0IPoi_Y1vgybDb-qScRxvrS-QAMVXskkS7e
qFB1VaW7LE6SCFcbneMjYE76BREAxL59zU7OBpmAu52AvbphvgqOhjoeM_OFKHFK0mAWzdEQqPMSvV62_jMl8Q
        status code: 403, request id: 8bdbbf40-836b-418a-b065-9e321a97f47b

```
  on instancecreation.tf line 7, in resource "aws_instance" "AWSServers":
   7: resource "aws_instance" "AWSServers" {


[mithun@mithuntechnologies terraformscripts]$ ▌
```

User doesn't have proper access.
Solution: Check the user permissions in AWS.

user_data

The user_data only runs at instance launch time.
 It will not execute after creation on instance.

Example: 6

FileName: var.tf

```
variable "ami"{
 description = "AMI Name..."
 default = "ami-052c08d70def0ac62"
}

variable "instance_type"{
 description = "Instance Type..."
 default = "t2.micro"
}

variable "instances" {
  description = "Total number of instances which we are going to create"
  default = 2
}
```

FileName: userdata.tf

```
provider "aws" {
 region = "ap-south-1"
 }

 resource "aws_instance" "AWSServer" {
  ami = "${var.ami}"
  instance_type = "${var.instance_type}"
  key_name = "devopsmssnovbatch"
  security_groups = ["launch-wizard-7"]
  user_data = "$(file("installApacheServer.sh"))"
  tags = {
    Name = "Terraform Server - MSS "
  }
 }
```

installApacheServer.sh

```
#!/bin/bash

sudo yum install httpd -y
sudo systemctl enable httpd
sudo systemctl start httpd
```

Security Groups Creation

A **security group** acts as a virtual firewall for your instance to control incoming and outgoing traffic. Inbound rules control the incoming traffic to your instance, and outbound rules control the outgoing traffic from your instance. ... **Security groups** are associated with network interfaces.

Example: 7

```
provider "aws" {
 region = "ap-south-1"
 }


resource "aws_security_group" "SecurityGroupsMSS" {
 name        = "SecurityGroupsMSS"
 description = "Allow TLS inbound traffic"
 vpc_id      = "vpc-27ec054c"

 ingress {
   description = "TLS from VPC"
   from_port   = 443
   to_port     = 443
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
 }
```

```
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "SecurityGroupsMSS"
  }
}
```

Identity and Access Management (IAM)

AWS Identity and Access Management (IAM) enables you to manage access to AWS services and resources securely. Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources. IAM is a feature of your AWS account offered at no additional charge.

Example: 8

FileName: iam.tf

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_iam_user" "IAMUSERS" {
  name = "mithuntechnologies"
}
```

Example: 9

FileName: vars.tf

```
variable "user_names" {
  description = "Create IAM users"
  type        = list(string)
  default     = ["mithuntechnologies"]
}
```

FileName: main.tf

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_iam_user" "IAMUsers" {
```

```
  count = length(var.user_names)
  name  = var.user_names[count.index]
}
```

Possible Error:

```
[mithun@mithuntechnologies iamuser]$ terraform apply

Error: Invalid resource name

  on main.tf line 5, in resource "aws_iam_user" "AWS IAM Users":
   5:  resource "aws_iam_user" "AWS IAM Users" {

A name must start with a letter or underscore and may contain only letters,
digits, underscores, and dashes.

[mithun@mithuntechnologies iamuser]$ █
```

Solution:
Give the Name without space like AWSIAMUser.

Possible Error:

```
[mithun@mithuntechnologies iamuser]$ terraform apply
aws_iam_user.IAMUsers[0]: Refreshing state... [id=mithunreddy]

Error: Error reading IAM User mithunreddy: AccessDenied: User: arn:aws:iam::73295
7639877:user/mithuntechnologies is not authorized to perform: iam:GetUser on reso
urce: user mithunreddy
        status code: 403, request id: fe1e1d87-9a32-4180-bc9d-a39dc7f9cab8


[mithun@mithuntechnologies iamuser]$ █
```

Solution:
Attach the **IAMFullAccess** to the user.

Simple Storage Service (S3)

Amazon S3 or Amazon Simple Storage Service is a service offered by Amazon Web Services (AWS) that provides object storage through a web service interface.

Example: 9

FileName: s3bucket.tf

```
provider "aws" {
  region     = "ap-southeast-1"
  access_key = "AKIA5CY7AN3CABU43L7A"
```

```
  secret_key = "crUSFGyfOo88q0Avt1dehAHRjEaVWt0nanlWYqBO"
 }

resource "aws_s3_bucket" "s3bucketmss" {
 bucket = "s3-bucket-mss"
 acl = "private"
 versioning {
  enabled = true
 }

lifecycle_rule {
  enabled = true

  transition {
   days = 10
   storage_class = "STANDARD_IA"
  }

  transition {
   days = 30
   storage_class = "GLACIER"
  }
 }

 tags = {
  Name = "S3 Bucket MSS by Terraform"
 }
}
```

Here:

bucket: name of the bucket, if we ommit that terraform will assign random bucket name
acl: Default to private (other options public-read and public-read-write)
versioning: Versioning automatically keeps up with different versions of the same object.

Life cycle has enabled here after 10 days move the objects to STANDARD_IA and after 30 days to GLACIER.

Resources

https://www.terraform.io/docs/providers/aws/d/ami.html
https://learn.hashicorp.com/terraform/getting-started/intro
https://www.terraform.io/docs/providers/aws/index.html
https://www.terraform.io/intro/index.html