Webserver deployment on

"Amazon Web Services" using IAC tool "Terraform"

Raghavendra Angara

Department of Dev-Ops Engineering

NexiiLabs

1. Abstract

The purpose of this technical paper is to provide a solution for deployment of webserver application facing public internet which process the user's request by automating the entire infrastructure deployment. The process of creating, deploying and configuring the webserver on to one of the popular cloud platforms, Amazon Web Services using Dev-ops tool, Terraform. This technical paper will take you to a step-by-step procedure on how to automate and deploy webserver application. This paper will cover all the technical aspects which helps to achieve webserver running on desired network along with the security.

2. Introduction

Terraform is a simple and very powerful orchestration tool developed to provision the infrastructure for the popular cloud platform which includes AWS, Google Cloud and Microsoft Azure. Terraform can also be used as a configuration management tool but is best suited for orchestration (provisioning).

AWS, Amazon Web Services, the popular cloud platform today supports huge enterprise application deployments. AWS provides secure cloud platform which offers compute, database, content delivery, storage and many more functionalities to serve small, medium and large scale enterprise.

This paper will deep dive the reader for automating the deployment process using Terraform in AWS.

3. Work Done

3.1 Problem statement:

To Deploy an infrastructure solutions like a webserver, we require a considerable efforts of AWS architects for configuring VPCs, subnets, routing tables, creating instances etc., these efforts are not easily repeatable and cannot modify easily in one shot.

Also there will be me many configuration changes in the infrastructure after the deployments, any change in the configuration may lead to the confusion and there is no way to store/track the state changes between old and new deployments. For example a change in the number of instances, deletion/creation of snapshot, Instances name, instances type changes etc., are not tracked and need lot of manual efforts to store the changes and restore the configuration if required.

3.2 Proposed solution:

Terraform, an "Dev-Ops" IAC (Infrastructure as a code) tool address these issues and helps the DEV-OPS teams to automate their infrastructure provisioning to third party cloud platforms. Terraform uses it CORE executable binary and make RPC to communicate with terraform plugins and provision the infrastructure configuration deployments.

Terraform is divided in to two parts, Terraform CORE and Terraform Plugins. The Terraform CORE is a main executable binary which invokes the plugins. A terraform Plugin is an implementation of provider services, which has all the modules that are developed to provision infrastructure in a particular cloud provider Platform.

This paper will give a comprehensible view to automate a simple webserver deployment in a Virtual Private network of a VPC.

Note: This paper gives only a brief and simple automated template of terraform which does not cover modules and other functions

3.3 AWS configuration:

This section covers the various resources that are required to deploy a webserver in Amazon Web Services using "Terraform". This complete configuration section is built using terraform templates and executed by terraform.

- Create Virtual Private network (VPC) with CIDR range
- Create a Public subnet with any of the CIDR block range, where the webserver will be deployed to face Internet users and handles their requests. The webserver will be assigned with a public IP as it is facing the internet users
- Create an "internet gateway" which connects the above created VPC to the internet.
- A routing table will be created to allow the EC2 Instances (where the webserver is deployed) to the outside WAN network.
- A Front End server, virtual instance is created to serves as a webserver which deployed in public subnet of a VPC.

The configuration is being deployed in any of the selected region in the Amazon Web Services.

AWS Resources:

Virtual Private Network

Public subnet

Route tables

Internet gateways

EC2 instance

Security Group

Note: Assuming the reader has basic understanding on Amazon Web services

3.4 Terraform Configuration:

The terraform works with Terraform CORE, an executable binary by "Hasicorp" which uses the RPC calls to Terraform plugins to provision the infrastructure on Cloud platforms.

Terraform is a declarative method of automating the infrastructure configuration completely and executes the code for further provisioning. Terraform provides a strong, powerful and secure functions to save or restore the configuration after the changes have being made. Terraform's state file ".tfstate" will maintain the current state and can be backed up using ".tfstate.backup" to retrieve the old state.

Terraform, a tool that allows the DEV-OPS team to easily change any of the configuration in the cloud using the same terraform template and can be saved in the ".tfstate" file.

Terraform template:

A terraform template is supported with two formats ".tf" and ".tf.json". This paper will present the template with ".tf" format. The user will present all his/her infrastructure requirements which to be provisioned. The terraform CORE reads the template files and download the necessary modules/provider-modules for specified providers and execute to provision the infrastructure.

Terraform Execution:

Once the user capture all his requirements in template with ".tf" or ".tf.json" formats, the user will initialize the terraform configuration nothing but all the templates that ends with ".tf" or "tf.json". The command "terraform init" will read all the templates and loads all the necessary modules and provider plugins modules to deal with particular provider in order to provision the infrastructure.

Once the necessary modules and provider plugins are loaded, the "terraform plan" will give an overview of a planned configuration plan.

Lastly, the command "terraform apply" will execute the planned configuration and provision the infrastructure on specified cloud platform.

Here are some of the useful commands that are helpful to execute terraform template:

Terraform init, will initializes and load all necessary plugins with respect to the provider. It also load modules/functions.

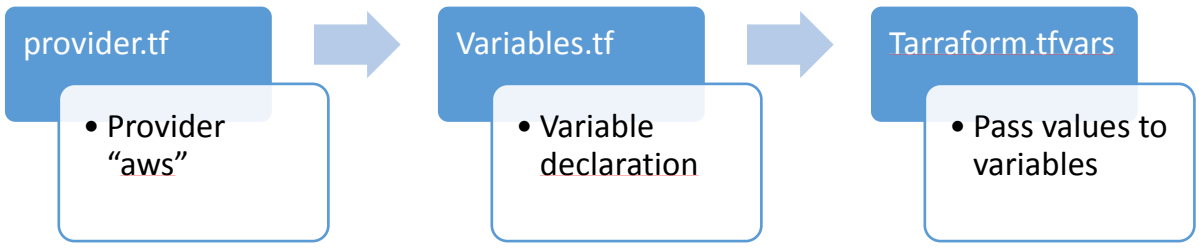
Terraform plan, helps to pre-check the infrastructure configuration which is about to provision.

Terraform apply, will apply the infrastructure configuration to a cloud platform based on the template prepared.

Terraform state show, will helps to understand the current configuration which is deployed using terraform template.

3.5 Infrastructure provisioning with Terraform:

This section will dive into the step-by-step procedure of creating a terraform template for deploying a webserver in Amazon Web Services



- Terraform Provider Configuration:

Configure the provider by passing Access Key, Secret Key and region to provide authentication with the AWS Cloud. When a terraform is initialized using "terraform init", the required cloud provider Plugins are installed first time. The terraform CORE execution binary will use the REST APIs and provision the infrastructure.

```
provider "aws"{
access_key = "${var.access_key}"
secret_key = "${var.secret_key}"
region = "${var.region}"
}
```

- Terraform Variable declaration:

Create a file "variables.tf" and declare the input variables to pass across the "main.tf" files.

```
variable "access_key" {}
variable "secret_key" {}
variable "region" {}
variable "vpc_cidr" {}
variable "public_cidr" {}
variable "vpc_tag" {}
variable "pub_subnet_tag" {}
variable "igw_tag" {}
variable "ingressfromport-http" {}
variable "ingresstoport-http" {}
variable "ingressfromport-https" {}
variable "ingresstoport-https" {}
variable "ingressfromport-ssh" {}
variable "ingresstoport-ssh" {}
variable "ingressfromport-rdp" {}
variable "ingresstoport-rdp" {}
variable "egressfromport" {}
variable "egresstoport" {}
variable "amiid" {}
variable "instance_type" {}
variable "key_pair" {}
variable "instance_tags" {}
```

- Terraform.tfvars declaration:

The values that are declared here are passed to the variable.tf files and from there the terraform core will be executed.

```
access_key = "xxxxxxxxxxxxxxxxxxxxxxxx"
secret_key = "xxxxxxxxxxxxxxxxxxxxxxxx"
region = "us-west-1"
vpc_cidr = "12.0.0.0/16"
public_cidr = "12.0.0.1/24"
vpc_tag = "terraform-vpc"
pub_subnet_tag = "publicsubnet-terraform"
igw_tag = "igw-terraform"
ingressfromport-http = "80"
ingresstoport-http = "80"
ingressfromport-https = "443"
ingresstoport-https = "443"
ingressfromport-ssh = "22"
ingresstoport-ssh = "22"
ingressfromport-rdp = "3389"
ingresstoport-rdp = "3389"
egressfromport = "0"
egresstoport = "0"
amiid = "ami-e0ba5c83"
instance_type = "t2.micro"
key_pair = "terraform"
instance_tags = "Webserver_instance"
```

- VPC Configuration:

```
resource "aws_vpc" "main" {
cidr_block = "${var.vpc_cidr}"
tags = {
Name = "${var.vpc_tag}"
}
}
```

- Public Subnet:

```
resource "aws_subnet" "main" {
vpc_id = "${aws_vpc.main.id}"
cidr_block = "${var.public_cidr}"
tags = {
Name  = "${var.pub_subnet_tag}"
}
}
#AWS Internet Gateway
resource "aws_internet_gateway" "gw" {
vpc_id = "${aws_vpc.main.id}"
tags = {
Name = "${var.igw_tag}"
}
}

# AWS Public Route - Main Route table
resource "aws_route" "main" {
route_table_id = "${aws_vpc.main.main_route_table_id}"
destination_cidr_block = "0.0.0.0/0"
gateway_id = "${aws_internet_gateway.gw.id}"
}

resource "aws_route_table_association" "b" {
    subnet_id = "${aws_subnet.main.id}"
    route_table_id = "${aws_vpc.main.main_route_table_id}"
}
```

- Security Group for Webserver:

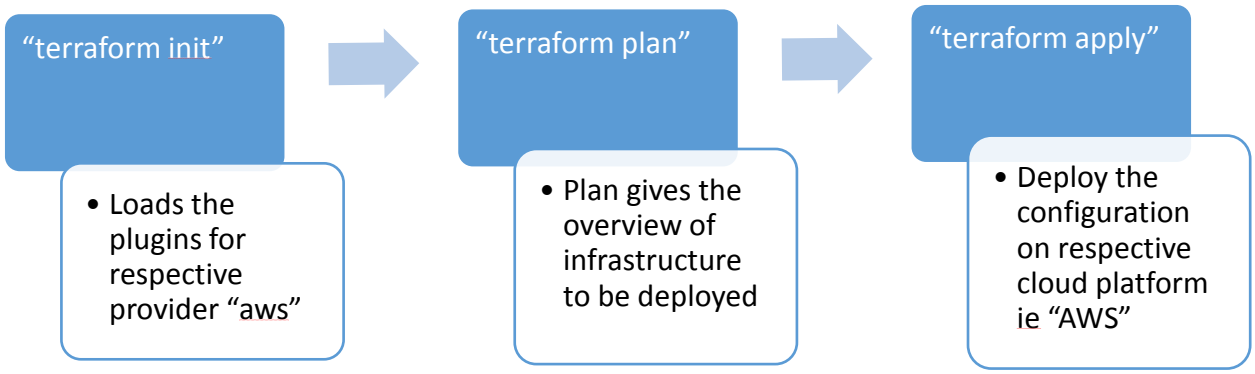
```
# AWS Security Group -Webserver
resource "aws_security_group" "Webserver-SG" {
name = "Webserver-SG"
description = "Allow inbound traffic for the Webserver"
vpc_id = "${aws_vpc.main.id}"

  ingress {
    from_port   = "${var.ingressfromport-http}"
    to_port     = "${var.ingresstoport-http}"
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = "${var.ingressfromport-https}"
    to_port     = "${var.ingresstoport-https}"
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = "${var.ingressfromport-ssh}"
    to_port     = "${var.ingresstoport-ssh}"
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = "${var.ingressfromport-rdp}"
    to_port     = "${var.ingresstoport-rdp}"
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = "${var.egressfromport}"
    to_port     = "${var.egresstoport}"
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  tags {
    Name = "WebserverSG"
  }
}
```

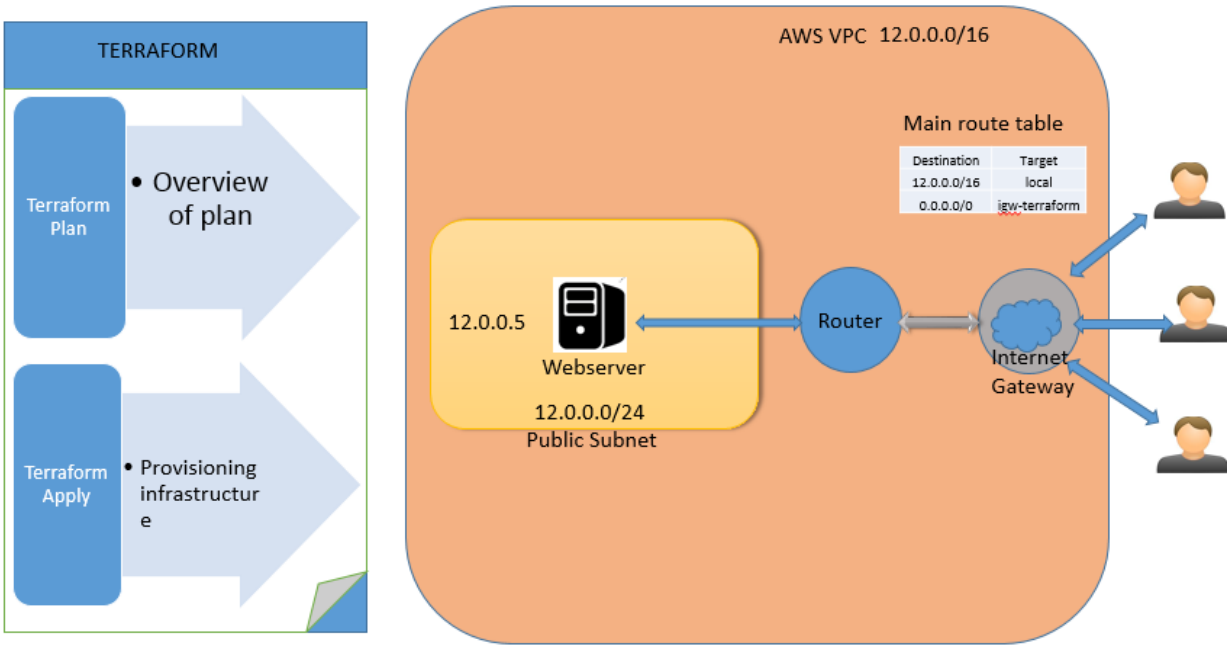
- Webserver:

```
# AWS Webserver
resource "aws_instance" "webserver" {

  ami = "${var.amiid}"
  user_data = <<-EOF
  #!/bin/bash -xe
  sudo yum update -y
  sudo yum install httpd -y
  sudo /etc/init.d/httpd start
  echo "<html><body><h1>Awesome</h1></body></html>" > /var/www/html/index.html
  EOF
  count = 1
  instance_type = "t1.micro"
  key_name = "${var.key_pair}"
  subnet_id = "${aws_subnet.main.id}"
  associate_public_ip_address = true
  security_groups = ["${aws_security_group.Webserver-SG.id}"]
  tags {
  Name = "${var.instance_tags}"
  }
}
```

3.6 Deploy the infrastructure:

Once the template creation is done, use “terraform init” to load the required provider plugins and continue to check the infrastructure to be built using “terraform plan”. Once the configuration is verified, use “terraform apply” to deploy the infrastructure





“terraform init”

```
D:\terraform_0.11.7_windows_amd64\templates\terraform_webserver>terraform init

Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "aws" (1.28.0)...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 1.28"
* provider.template: version = "~> 1.0"

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.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

“terraform plan”

```
D:\terraform_0.11.7_windows_amd64\templates\terraform_webserver>terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

data.template_file.user_data: Refreshing state...
data.aws_ami_ids.ids1: Refreshing state...

------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + aws_instance.webserver
      id:                               <computed>
      ami:                              "ami-0e86606d"
      associate_public_ip_address:      "true"
      availability_zone:                <computed>
      ebs_block_device.#:               <computed>
      ephemeral_block_device.#:         <computed>
      get_password_data:                "false"
      instance_state:                   <computed>
      instance_type:                    "t1.micro"
      ipv6_address_count:               <computed>
      ipv6_addresses.#:                 <computed>
      key_name:                         "terraform"
      network_interface.#:              <computed>
      network_interface_id:             <computed>
      password_data:                    <computed>
      placement_group:                  <computed>
      primary_network_interface_id:     <computed>
      private_dns:                      <computed>
      private_ip:                       <computed>
      public_dns:                       <computed>
      public_ip:                        <computed>
      root_block_device.#:              <computed>
      security_groups.#:                <computed>
      source_dest_check:                "true"
      subnet_id:                        "${aws_subnet.main.id}"
      tags.%:                           "1"
      tags.Name:                        "Webserver_instance"
      tenancy:                          <computed>
      user_data:                        "cc20c04cf0d94e85384bcfe9b3e645606aa53c07"
      volume_tags.%:                    <computed>
      vpc_security_group_ids.#:         <computed>
```

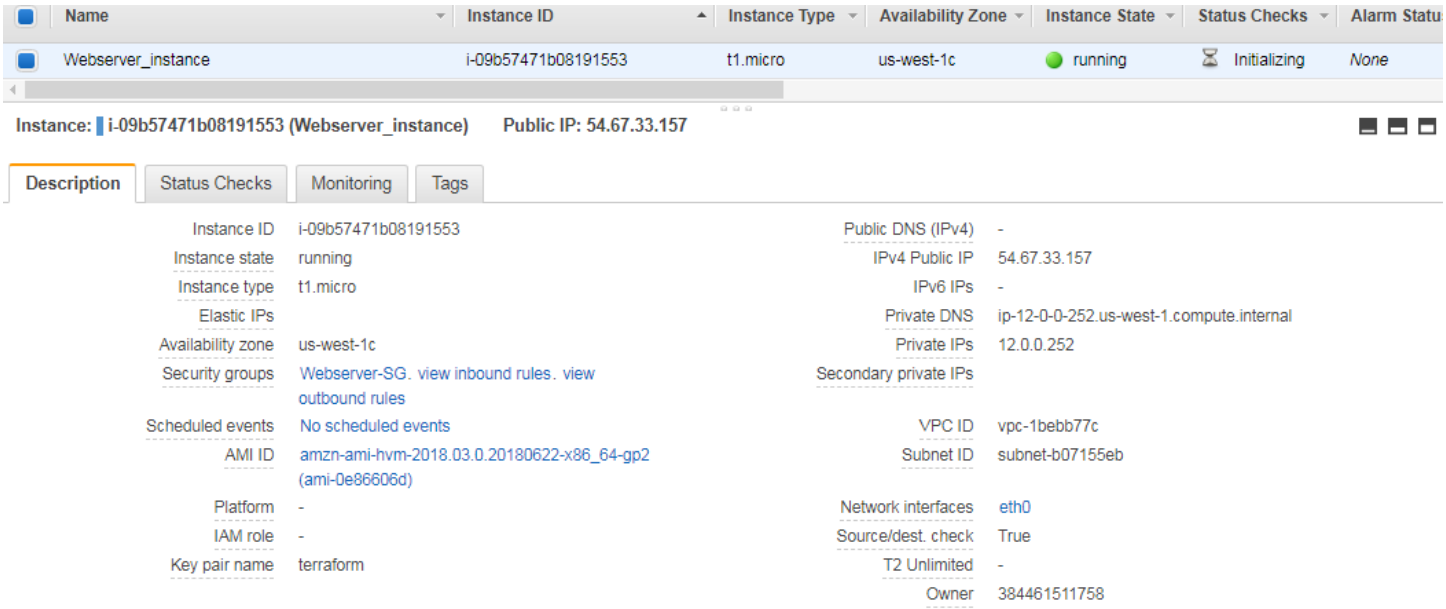
Note; The above output is only showing the webserver instance installation as the output of this command is too large.

“terraform apply”

```
D:\terraform_0.11.7_windows_amd64\templates\terraform_webserver>terraform apply
data.template_file.user_data: Refreshing state...
data.aws_ami_ids.ids1: Refreshing state...

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + aws_instance.webserver
      id:                               <computed>
      ami:                              "ami-0e86606d"
      associate_public_ip_address:      "true"
      availability_zone:                <computed>
      ebs_block_device.#:               <computed>
      ephemeral_block_device.#:         <computed>
      get_password_data:                "false"
      instance_state:                   <computed>
      instance_type:                    "t1.micro"
      ipv6_address_count:               <computed>
      ipv6_addresses.#:                 <computed>
      key_name:                         "terraform"
      network_interface.#:              <computed>
      network_interface_id:             <computed>
      password_data:                    <computed>
      placement_group:                  <computed>
      primary_network_interface_id:     <computed>
      private_dns:                      <computed>
      private_ip:                       <computed>
      public_dns:                       <computed>
      public_ip:                        <computed>
      root_block_device.#:              <computed>
      security_groups.#:                <computed>
      source_dest_check:                "true"
      subnet_id:                        "${aws_subnet.main.id}"
      tags.%:                           "1"
      tags.Name:                        "Webserver_instance"
      tenancy:                          <computed>
      user_data:                        "cc20c04cf0d94e85384bcfe9b3e645606aa53c07"
      volume_tags.%:                    <computed>
      vpc_security_group_ids.#:         <computed>

  + aws_internet_gateway.gw
      id:                               <computed>
      tags.%:                           "1"
      tags.Name:                        "igw-terraform"
      vpc_id:                           "${aws_vpc.main.id}"
```

Note; The above output is only showing the webserver instance installation as the output of this command is too large.

AWS console output:



Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Statu
Webserver_instance	i-09b57471b08191553	t1.micro	us-west-1c	running	Initializing	None

Instance: i-09b57471b08191553 (Webserver_instance) Public IP: 54.67.33.157

Description | Status Checks | Monitoring | Tags

Instance ID	i-09b57471b08191553	Public DNS (IPv4)	-
Instance state	running	IPv4 Public IP	54.67.33.157
Instance type	t1.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-12-0-0-252.us-west-1.compute.internal
Availability zone	us-west-1c	Private IPs	12.0.0.252
Security groups	Webserver-SG. view inbound rules. view outbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-1bebb77c
AMI ID	amzn-ami-hvm-2018.03.0.20180622-x86_64-gp2 (ami-0e86606d)	Subnet ID	subnet-b07155eb
Platform	-	Network interfaces	eth0
IAM role	-	Source/dest. check	True
Key pair name	terraform	T2 Unlimited	-
		Owner	384461511758

4. Conclusion:

Using an IAC tool, Terraform, infrastructure provisioning is made easy, reliable and reusable. The same template can be used to deploy a webservers in different regions in a couple of minutes without user errors and rework. The same template can also be used to modify the infrastructure based on the business requirements and these changes can be saved in a state file.

5. References

Terraform:

https://www.terraform.io/intro/index.html

https://www.terraform.io/intro/getting-started/build.html

AWS:

https://aws.amazon.com/vpc