Terraform

Infrastructure as Code

Hugo Posca

Pagar.me

“ Terraforming (literally, "Earth-shaping") of a planet, moon, or other body is the hypothetical process of deliberately modifying its atmosphere, temperature, surface topography or ecology to be similar to the environment of Earth to make it habitable by Earth-like life. ”

What is Terraform?

“ Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. ”

Terraform

- Another wonderful tool created by HashiCorp ❤
- Started around may 2014

Features

- Manages the infrastructure, period.
  - It is not a configuration management tool
- Popular infrastructure providers
  - AWS
  - DigitalOcean
  - GCE
  - Azure
- Enable multiple providers

Features

- **Infrastructure as Code**
  - Configuration files (.tf)
- **Execution Plans**
  - What Terraform will do
- **Resource Graph**
  - Parallelizes the creation and modification of any non-dependent resources

How to Use It

- Download the appropriate package
- Unzip it
- Add `terraform` executable to your `$PATH`

```
$ terraform
Usage: terraform [--version] [--help] <command> [args]
...
```

Workflow

Step 1 - Create configuration 📝

```
$ vim main.tf
```

Step 2 - Check the execution plan ✅

```
$ terraform plan
```

Step 3 - Build the Infrastructure 🔨

```
$ terraform apply
```

Step X - Destroy the Infrastructure ⚠️

```
$ terraform plan --destroy
$ terraform destroy # It will ask you, unless --force
```

Examples

1 - A simple infrastructure

1.1 Build a simple infrastructure

```
resource "aws_instance" "example" {
    ami           = "ami-6edd3078"
    instance_type = "t2.micro"

    tags {
        Name = "Test Machine"
    }
}
```

- Uses the Terraform configuration format
- It is not JSON, but it accepts JSON.

1.2 Improve it

```
resource "aws_security_group" "ssh" {
  name = "allow_ssh"
  description = "Allow SSH connections"

  ingress {
      from_port = 22
      to_port = 22
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "example" {
  ...
  vpc_security_group_ids = ["${aws_security_group.ssh.id}"
}
```

1.3 Check that everything is as expected

```
$ terraform show
aws_instance.example:
  id = i-03be02b260814674a
  ami = ami-6edd3078
  instance_state = running
  instance_type = t2.micro
  private_dns = ip-172-31-46-252.ec2.internal
  private_ip = 172.31.46.252
  public_dns = ec2-54-174-18-13.compute-1.amazonaws.com
  public_ip = 54.174.18.13
  tags.% = 1
  tags.Name = Test Machine
  vpc_security_group_ids.# = 1
  vpc_security_group_ids.2555431445 = sg-e71f489b
```

```
aws_security_group.ssh:
  id = sg-e71f489b
  description = Allow SSH connections
  egress.# = 0
  ingress.# = 1
  ingress.2541437006.cidr_blocks.# = 1
  ingress.2541437006.cidr_blocks.0 = 0.0.0.0/0
  ingress.2541437006.from_port = 22
  ingress.2541437006.protocol = tcp
  ingress.2541437006.security_groups.# = 0
  ingress.2541437006.self = false
  ingress.2541437006.to_port = 22
  name = allow_ssh
  owner_id = 565223257425
  tags.% = 0
  vpc_id = vpc-72936d17
```

1.4 Destroy it

```
$ terraform plan --destroy
...

- aws_instance.example
- aws_security_group.ssh

Plan: 0 to add, 0 to change, 2 to destroy.
```

```
$ terraform destroy
Do you really want to destroy?
  Terraform will delete all your managed infrastructure.
  There is no undo. Only 'yes' will be accepted to confirm

  Enter a value: yes

aws_security_group.ssh: Refreshing state... (ID: sg-e71f48
aws_instance.example: Refreshing state... (ID: i-03be02b26
aws_instance.example: Destroying...
aws_instance.example: Still destroying... (10s elapsed)
aws_instance.example: Still destroying... (20s elapsed)
aws_instance.example: Still destroying... (30s elapsed)
aws_instance.example: Still destroying... (40s elapsed)
aws_instance.example: Still destroying... (50s elapsed)
aws_instance.example: Still destroying... (1m0s elapsed)
aws_instance.example: Destruction complete
aws_security_group.ssh: Destroying...
aws_security_group.ssh: Destruction complete

Destroy complete! Resources: 2 destroyed.
```

2.1 Isolating parts

> “ When invoking any command that loads the Terraform configuration, Terraform loads all configuration files within the directory specified in alphabetical order. ”

```
.
├── main.tf
└── security_group.tf
```

security_group.tf:

```
resource "aws_security_group" "ssh" {
  name = "allow_ssh"
  description = "Allow SSH connections"

  ingress {
  ...
  }
}
```

main .tf:

```
resource "aws_instance" "example" {
    ...
    vpc_security_group_ids = ["${aws_security_group.ssh.id}
}
```

3.1 Isolating components (A.K.A. Modules)

```
.
├── main.tf
└── security_group
    ├── main.tf
    └── outputs.tf
```

security_group/main.tf :

```
resource "aws_security_group" "ssh" {
  name = "allow_ssh"
  description = "Allow SSH connections"

  ingress {
      from_port = 22
      to_port = 22
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
  }
}
```

security_group/outputs.tf :

```
output "group_id" {
  value = "${aws_security_group.ssh.id}"
}
```

main. tf :

```
module "security_group" {
  source = "./security_group"
}

resource "aws_instance" "example" {
    ami           = "ami-6edd3078"
    instance_type = "t2.micro"
    vpc_security_group_ids = ["${module.security_group.grou

    tags {
        Name = "Test Machine"
    }
}
```

Before planning and applying:

```
$ terraform get
```

To destroy it use the `--target`

```
$ terraform plan --destroy --target module.security_group
$ terraform destroy --target module.security_group
```

3.2 Isolating components (A.K.A. Modules)

Modules are very useful as you can reuse them passing some variables:

```
.
├── main.tf
└── security_group
    ├── main.tf
    ├── outputs.tf
    └── variables.tf
```

security_group/variables.tf :

```
variable "sg_nametag" {
  default = "Security Group Tag"
}
```

security_group/main.tf :

```
resource "aws_security_group" "ssh" {
  ...

  tags {
        Name = "${var.sg_nametag}"
  }
}
```

main. tf :

```
module "security_group" {
  source     = "./security_group"
  sg_nametag = "A new tag"
}

resource "aws_instance" "example" {
  ...
}
```

```
$ terraform plan
...
+ module.security_group.aws_security_group.ssh
    description:                          "Allow SSH connec
    tags.%:                               "1"
    tags.Name:                            "A new tag"
```

Drawbacks

- The more you use the bigger it gets
- Code repetition

But don't worry, smarter people already fought with this:

- GeoEnginner
- TerraGrunt

They can provide:

- Extensible Validation
- Reusable Templates
- Validations
- Locking
- Enforced remote state management

And remember:

Are you too busy to improve?
No thanks!
We are too busy
Created by Håkan Forss @hakanforss http://hakanforss.wordpress.com

References

- Terraform
- Terraform - Docs
- Terraform - Introduction
- Terraform - Download
- GeoEnginner
- TerraGrunt
- Segmentio's Stack