Terraform 介绍

by

Bin Liu

2017.4.28

Agenda

1. 什么是 Terraform
2. 使用 Terraform 部署 Docker 容器
3. Terraform 使用说明
4. 使用 Terraform 管理灵雀云服务

1. 什么是 Terraform

“ **Write, Plan, and Create Infrastructure as Code** ”

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

Infrastructure as `Code`

- 版本化
- 可测试
- 持续集成
- 自动化（测试、部署）
- 可重现
- 可审计
- 协作

Infrastructure as `Code`

- 配制管理
- 1993 CFEngine
- 2005 Puppet
- 2008 Infrastructure as Code
- 2009 Chef
- 2009 DevOps
- 2013 Ansible

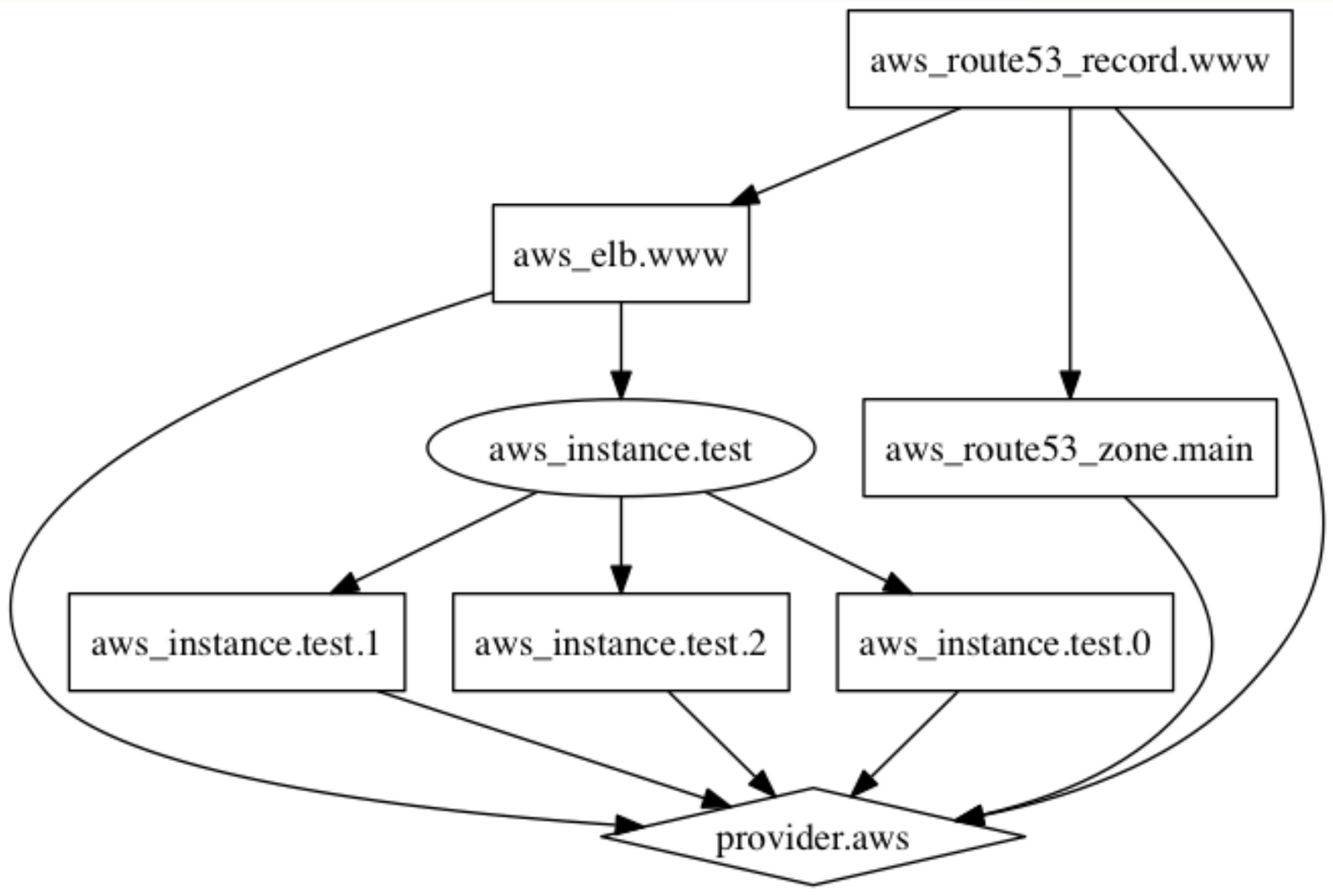
Terraform

```
provider "aws" {
  access_key = "ACCESS_KEY_HERE"
  secret_key = "SECRET_KEY_HERE"
  region     = "us-east-1"
}

resource "aws_instance" "example" {
  ami           = "ami-2757f631"
  instance_type = "t2.micro"
}

resource "aws_eip" "ip" {
  instance = "${aws_instance.example.id}"
}
```

Terraform

```
resource "dns_a_record_set" "www" {
  zone = "example.com."
  name = "www"
  addresses = [
    "192.168.0.1",
    "192.168.0.2",
    "192.168.0.3",
  ]
  ttl = 300
}
```

```
$ terraform graph | dot -Tpng > graph.png
```



Terraform

- 统一的工作流
- 跨云
- Instance/Service

概念

- Abstraction of resources
- 执行计划（Execution Plans）
- 资源图（Resource Graph）
- Provider（AWS、Docker）
- Bootstrap
- Provisioner
- HCL（配置文件语言）

2. 使用 Terraform 部署 Docker 容器

配置文件:

```
resource "docker_container" "redis" {
  image = "${docker_image.redis.name}"
  name  = "redis-server"
  env   =["A=B"]
}

resource "docker_image" "redis" {
  name = "redis:alpine"
}
```

操作命令

```
$ docker ps | grep redis
$ docker images | grep redis
$ terraform plan
$ terraform apply
$ terraform show
$ vi ...
$ terraform plan
$ terraform apply
$ terraform destroy -target=docker_container.redis
$ docker ps | grep redis
```

3. Terraform 使用说明

Variables

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

- Asked when run
- 命令行参数

```
$ terraform plan -var 'access_key=foo'
```

- varfile

```
$ terraform plan -var-file="secret.tfvars"
```

Data Types

- string/true/false/int
- List

```
cidrs = [ "10.0.0.0/16", "10.1.0.0/16" ]
```

- Map

```
default = {
  "us-east-1" = "ami-b374d5a5"
  "us-west-2" = "ami-4b32be2b"
}
```

变量引用

```
${var.foo}
${var.amis["us-east-1"]}
${var.subnets[idx]}
```

Output Variables

```
output "ip" {
  value = "${aws_eip.ip.public_ip}"
}
```

apply

```
$ terraform apply
...
Outputs:
  ip = 50.17.232.209
```

output

```
$ terraform output ip
50.17.232.209
```

Modules

> **Self-contained packages of configurations managed as a group**

Modules

```
module "consul" {
  source = "github.com/hashicorp/consul/terraform/aws"

  key_name = "AWS SSH KEY NAME"
  key_path = "PATH TO ABOVE PRIVATE KEY"
  servers  = "3"
}

output "consul_address" {
  value = "${module.consul.server_address}"
}
```

```
$ terraform get
$ terraform plan
```

Backends

- Storage
- Lock
- Team Collaboration
- Consul/S3/Terraform Enterprise

```
terraform {
  backend "consul" {
    address = "demo.consul.io"
    path    = "getting-started-xyz123"
    lock    = false
  }
}
```

Terraform Enterprise

```
terraform {
  backend "atlas" {
    name = "USERNAME/getting-started"
  }
}
```

```
$ git add example.tf
$ git commit -m "init commit"
$ terraform push
```

Interpolation Syntax

- 条件计算

```
resource "aws_instance" "web" {
  subnet = "${var.env == "production" ? var.prod_subnet : var.dev_subnet}"
}
```

Built-in Functions

- 基本类型（字符串、list等）
- 数学
- 编码（md5，base64，sha1）
- 文件
- cidr`xyz`（网络相关）

Provisioners

- SSH
- file copy
- chef

4. 使用 Terraform 管理灵雀云服务

```go
func Provider() *schema.Provider {
        return &schema.Provider{
                Schema: map[string]*schema.Schema{
                        "token": &schema.Schema{
                                Type:        schema.TypeString,
                                Optional:    true,
                                Default:     "",
                                Description: "The token used to authenticate t
                        },
                },

                // Provider factory
                ConfigureFunc: providerConfigure,

                ResourcesMap: map[string]*schema.Resource{
                        "alauda_service": resourceService(),
                },
        }
}
```

```
func resourceService() *schema.Resource {
        return &schema.Resource{
                Create: resourceServiceCreate,
                Read:   resourceServiceRead,
                Update: resourceServiceUpdate,
                Delete: resourceServiceDelete,

                Schema: map[string]*schema.Schema{
                        "service_name": &schema.Schema{
                                Type:     schema.TypeString,
                                Required: true,
                        },

                        "scaling_mode": &schema.Schema{
                                Type:         schema.TypeString,
                                Required:     true,
                                ValidateFunc: validateStringInWord([]string{"A
                        },
```

```
"instance_ports": &schema.Schema{
        Type:     schema.TypeList,
        Required: true,
        Elem: &schema.Resource{
                Schema: map[string]*schema.Schema{
                        "endpoint_type": &schema.Schema{
                                Type:         schema.TypeString,
                                Optional:     true,
                                Default:      "http-endpoint",
                                ValidateFunc:
                                        validateStringInWord(
                                    []string{"http-endpoint", "tcp-endpoint"}),
                        },
                        "service_port": &schema.Schema{
                                Type:         schema.TypeInt,
                                Optional:     true,
                                Default:      "80",
                                ValidateFunc: validateIntegerInRange(1, 65535)
                        },
                },
        }},
```

```
resource "alauda_service" "terraform-test" {
    target_num_instances = 1
    region_name = "testintcluster"
    service_name = "terraform-test"
    instance_size = "XXS"
    scaling_mode = "MANUAL"
    image_name = "index.alauda.cn/alauda/hello-world"
    network_mode = "BRIDGE"
    target_state = "STARTED"
    mipn_enabled = true
    space_name = "project4-dev-space"
    image_tag = "latest"
    instance_ports = [{
          "container_port" = 80
          "endpoint_type" = "http-endpoint"
          "protocol" = "tcp"
          "ipaddress" = "sharedip"
          "service_port" = 80
    }]
}
```

Install

```
cat <<EOF > ~/.terraformrc
providers {
  alauda = "/gopath/src/github.com/alauda/terraform-alauda/bin/terraform-alaud
}
EOF
```

Create

```
$ terraform apply -var-file=~/alauda.tfvars examples
```

Show

```
$ terraform show
alauda_service.terraform-test:
  id = a47978e8-19b6-4fb0-a49d-dfacebe6ea2c
  image_name = index.alauda.cn/alauda/hello-world
  instance_ports.# = 1
  instance_ports.0.container_port = 80
  instance_ports.0.endpoint_type = http-endpoint
  instance_ports.0.protocol = tcp
  instance_ports.0.service_port = 80
  instance_size = XXS
  network_mode = BRIDGE
  scaling_mode = MANUAL
  service_name = terraform-test
  target_num_instances = 1
  target_state = STARTED
  ... ...
```

Update

```
$ terraform apply -var-file=~/alauda.tfvars examples
```

Read（Sync）

```
$ terraform refresh -var-file=~/alauda.tfvars examples
```

Destroy

```
$ terraform destroy -var-file=~/alauda.tfvars \
-target=alauda_service.terraform-test examples
```

Start/Stop service

Failed!

```
target_state = "STOPPED"
```

Thanks