

# Terraform，自动化配置与编排必备利器

## Terraform - Infrastructure as Code

### 什么是Terraform

Terraform 是一个安全、高效地部署、更改、版本化基础设施和应用程序的工具，可以用来管理多层次的资源。从上层的软件配置到底层的网络、系统配置都可以使用 Terraform 统一进行管理。

Terraform 用配置文件来描述一个应用。Terraform 会将配置文件与当前环境对比后，生成一个执行计划，这个计划会列出为了达到配置文件中定义的状态所需要执行的操作，然后执行计划以达到期望的状态。

Terraform 通过插件机制管理不同的资源提供者，以此来接入各种资源，如虚拟机，存储，网络和各种应用服务。

### Terraform的主要特性

#### Infrastructure as Code

将基础架构使用配置语法进行描述，这可以让数据中心的构建计划可以像其他代码一样进行版本化和追踪。

#### Execution Plans

Terraform 有一个规划步骤，它生成一个执行计划。执行计划显示当您调用应用程序时 Terraform 将执行的操作。使用这个功能可以保证操作基础设施时不发生意外

#### Resource Graph

Terraform 创建了一个所有资源的视图。这使得 Terraform 可以并行化没有依赖的创建与修改。

因此，Terraform 可以高效地构建基础架构。操作人员也能更加了解环境的结构。

#### Change Automation

Terraform 会自动的分析什么是需要修改的，从而避免了许多可能的人为错误。

## Terraform vs. Other Software

与 Terraform 类似的 Infrastructure as Code 工具大概有下面几种：

- Chef
- Puppet
- Ansible
- SaltStack
- CloudFormation

下面将从几个方面来说明 Terraform 与其他工具对比的优势。

### 配置管理工具与编排工具

Chef、Puppet、Ansible、SaltStack 都可以称为配置管理工具，这些工具的主要目标是在已经存在的机器上安装和管理软件。

而 Terraform 和 CloudFormation 可以称为编排工具，更侧重于数据中心以及相关服务的高级抽象。他们的工作重点是创建资源并且引导进行初始化。

而且在现在的环境下，大家使用容器等服务，镜像已经包括了软件的安装与配置。一旦你有了镜像，你需要的是一些服务器去运行它。

对于提供服务器这种需求，编排工具会比配置管理工具更适合做此类工作。

### 程式语言与声明式语言

Chef 和 Ansible 希望你一步步编写程序以达到最终所期望的状态。

Terraform、CloudFormation、SaltStack、Puppet 希望你声明最终想要的资源与资源的状态，工具本身会自动分析达到想要的状态需要进行怎样的操作。

在使用程式语言时，工具不会获取历史的状态，所以我们需要考虑的更多以达到与之前版本的兼容。

并且使用程式语言会导致代码库变得越来越庞大，不利于人们理解与代码的复用。不过声明式的语言的表达能力是较为欠缺的，例如我们需要基础设施的滚动升级时，声明式的语言是很难满足要求的。

为此 Terraform 提供了一些基础服务，例如输入变量，输出变量，在销毁之前创建等。

### 客户端服务器架构与客户端架构

Chef、Puppet、SaltStack 在默认情况下都使用了客户端服务器架构。客户端（可能是 Web UI 或 CLI 工具）是用来发出命令（例如“deploy X”）的东西。

这些命令到达一个服务器，它负责执行你的命令并存储系统的状态。要执行这些命令，服务器会与 agent 进行通信，agent 必须在要配置的每个服务器上运行，这有几个缺点：

- 您必须在每台服务器上安装并运行额外的软件。
- 为了配置管理，您必须部署额外的服务器（甚至是一组服务器以实现高可用性）。
- 由于客户端，服务器和代理都需要通过网络进行通信，因此您必须为其打开额外的端口，并配置相互验证。
- 这些配置会引入大量不同类型的故障，当收到错误警告时，必须要弄清楚是哪一部分出现了故障。

CloudFormation也是客户端/服务器架构，但AWS透明地处理所有的服务器细节，作为最终用户，您只需要考虑客户端代码。Ansible客户端则通过SSH直接连接到您的服务器。

Terraform使用云提供商API来配置基础架构，因此除了您已经使用云提供商之外，没有新的身份验证机制，并且不需要直接访问您的服务器。

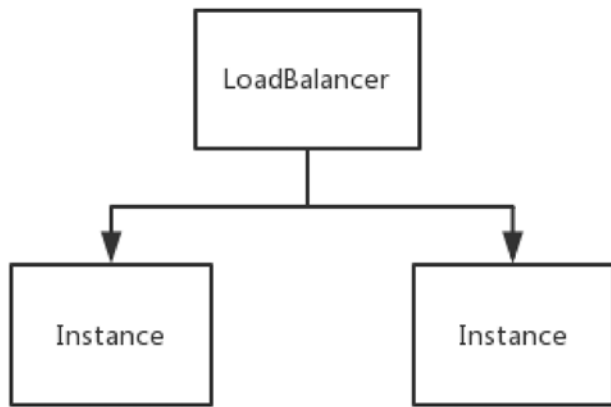
## 比较表格

	Chef	Puppet	Ansible	SaltStack	CloudFormation	Terraform
是否开源	开源	开源	开源	开源	闭源	开源
支持的云	All	All	All	All	AWS only	All
工具类型	配置管理	配置管理	配置管理	配置管理	编排工具	编排工具
语言类型	程式化	声明式	程式化	声明式	声明式	声明式
架构	客户端服务器	客户端服务器	客户端	客户端服务器	客户端	客户端

## Terraform的应用场景

### 应用场景1

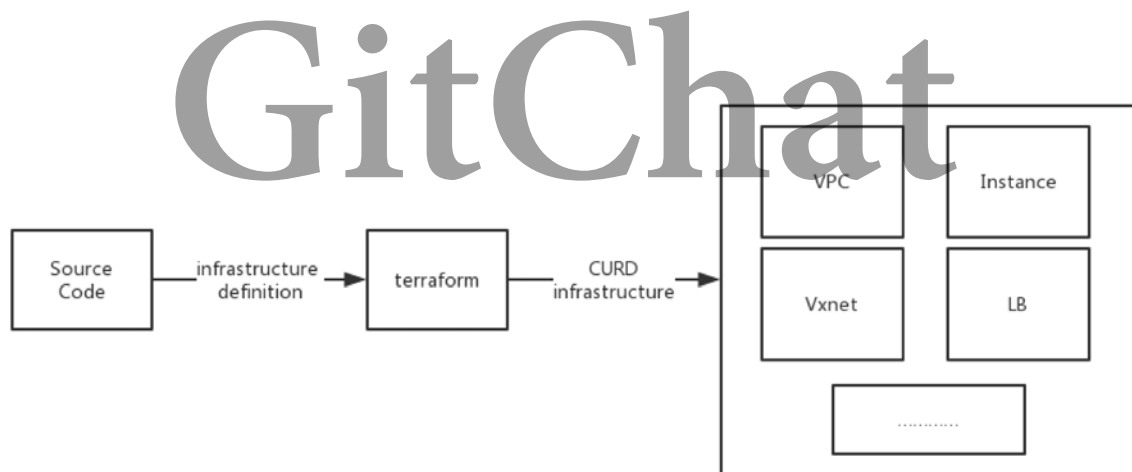
某应用使用了Loadbalancer进行Instance间的流量均衡处理，以增大吞吐率、扩大并发数、缩短延迟。



操作步骤：新建安全组、添加安全组规则、申请弹性公网IP、创建负载均衡器、创建 Instance、在负载均衡器中添加负载均衡器监听器、配置会话保持、添加健康检查、在负载均衡器监听器中添加后端.....

## 应用场景2

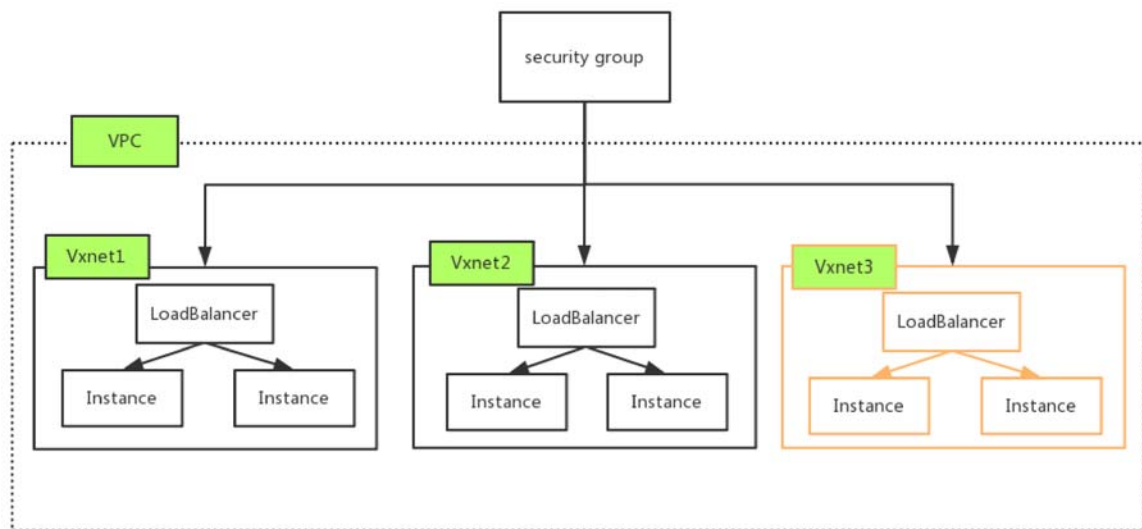
某应用需要隔离的网络环境，需要将应用搭建在VPC网络当中，架构如下：



操作步骤：创建安全组、配置安全组、申请弹性公网 IP、创建 VPC、创建 vxnet、创建负载均衡器、配置负载均衡器、创建 Instance 等配置操作。

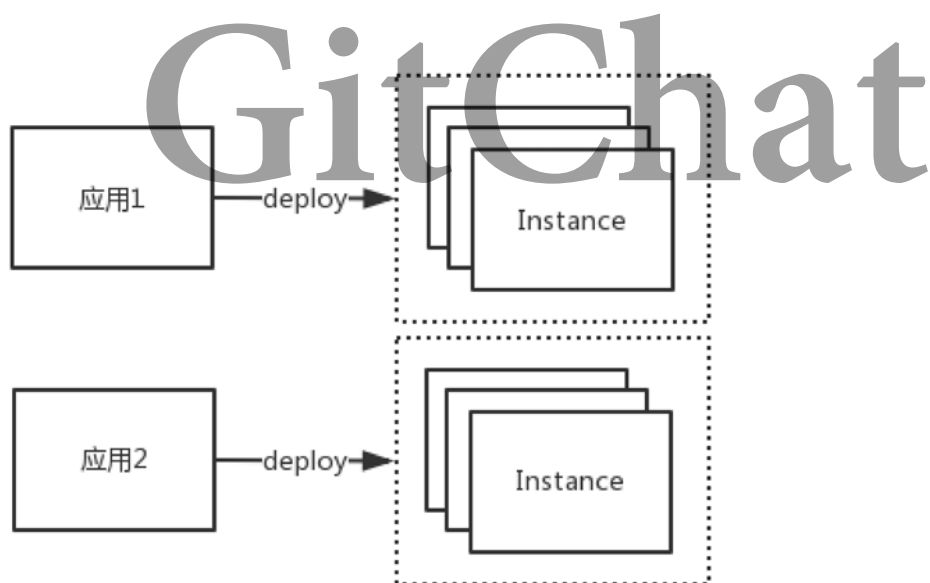
## 应用场景3

随着业务调整和业务量增大，需要增加更多的节点和集群以承载更多的请求，此时需要对已有的资源进行扩容：



## 应用场景4

随着应用的不断迭代，应用部署和发布的成本也在增加，如何实现应用的快速部署和发布：

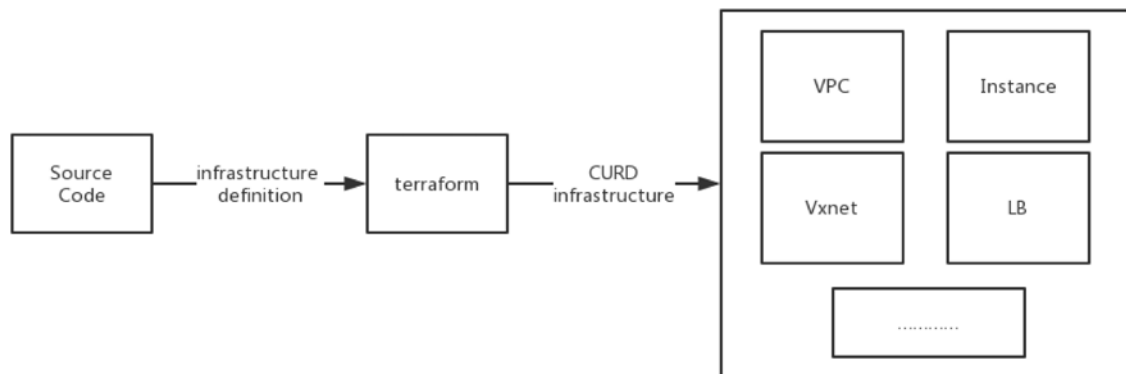


## 上述场景的共性

- 操作流程配置固定
- 手工操作效率低、时间长
- 手工操作可能导致错误
- 手工操作过程没有历史记录
- 手工操作过程不能审计
- 手工操作随着业务的复杂而不断增大

解决方法：使用 Terraform 自动化一切

针对场景一、二：创建基础设施



使用 Terraform，我们可以对基础设施进行编码，利用代码来进行资源的增删查改。

针对场景三、四：扩容和部署



修改资源代码以改变基础设施，并利用 provisioner 帮助运行部署脚本，完成扩容和部署。

## 使用 Terraform 的优点

快速和安全

利用代码进行创建资源的速度可以是人工点击的几倍甚至几十倍。

任何人都可以随时建立起一个环境

只要拥有基础设施代码的访问权限，就可以自己完成环境的创建，而不需要其他团队的帮助。

避免雪花服务器

人工配置环境可能会导致出现雪花服务器，而使用 Terraform 自动配置则不容易出现。

所有的环境使用类似的代码

使用 Terraform 你可以轻松的控制副本的数量，方便创建一个小的开发/测试环境，或者一个大的生产环境

像对待应用代码一样对待基础设施

方便不断的改进基础设施的代码。享受版本管理工具的历史查看、回滚、备份存储等功能。

在部署之前验证架构

对基础设施代码做静态分析，可以大大降低出错的可能性。

追踪基础设施的变化

如果在部署当中发生了错误，可以快速恢复到上一个版本，审计人员也可以通过历史记录进行审计。

快乐

这是一个非常重要但是经常被忽视的优点。手动部署代码和管理基础架构是重复和乏味的。开发人员和系统管理员对这种工作感到不满，因为它不涉及创造力，没有挑战，也没有认可。

除非你部署失败了，不然没有人会关注你。

这造成了一个压力和不愉快的环境。Terraform 提供了一个更好的选择，允许计算机做他们最擅长的（自动化执行）和开发人员来做他们最擅长的（编码）。

## Terraform-QingCloud使用

### 1.Terraform 及 terraform-provider-qingcloud安装

#### 安装 Terraform

我们首先安装 Terraform，我们需要进到 Terraform 的官网找到[适合的软件包](#)进行下载。

下载 Terraform 后，解压压缩包。压缩包中有一个名为 Terraform 的文件，我们只需要这个二进制文件就可以使用 Terraform 了。

最后一步是设置 Terraform 的 PATH。如何在 Linux 和 Mac 中设置 PATH 可以参考[这个页面](#)，如何在 Windows 当中设置 PATH 可以参考[这个页面](#)。

## 验证 Terraform 安装

在安装完 Terraform 之后，我们可以打开一个新的终端来验证 Terraform 安装成功了。

执行 terraform -v 可以看到类似下面的输出：

```
$ terraform -v
Terraform v0.11.1
```

## 安装 terraform-provider-qingcloud

terraform-provider-qingcloud 同样是以二进制文件进行发布，我们可以到 Github 上找到[适合的软件包](#)进行下载。

下载并解压以后会有一个二进制文件。

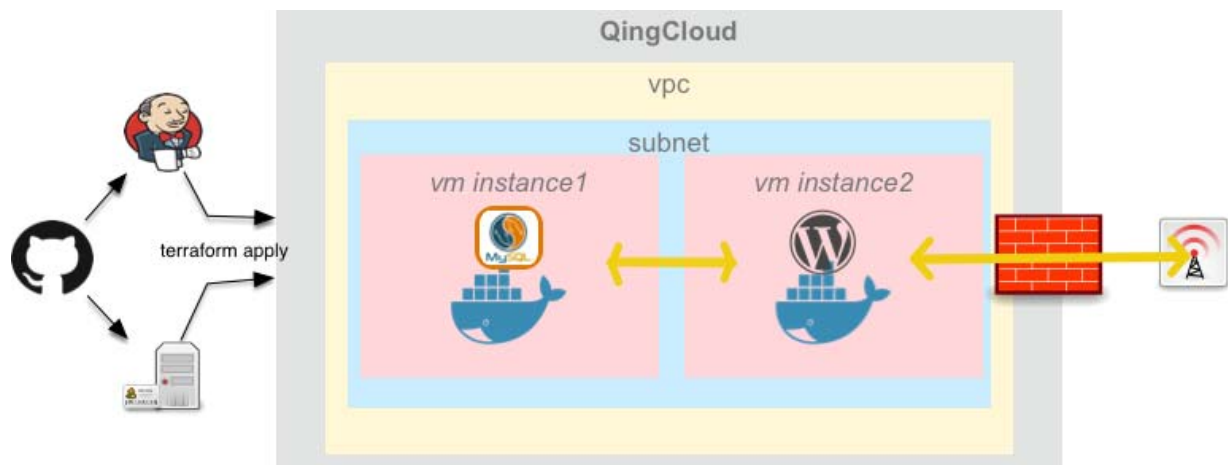
在 Linux 以及 Mac 当中我们需要将这个二进制文件放到 ~/.terraform.d/plugins/当中。在 Windows 当中把这个二进制文件放到用户的 "Application Data" 目录下的 terraform.d/plugins/当中。

## 2.Terraform 使用-以 wordpress 为例

下面我们以 wordpress 为例演示如何使用 Terraform 在青云中从零开始部署一整套环境。部署下图中所示的基础设施及软件资源。

[例子源码请单击这里查看。](#)

*注意：使用 terraform apply 会创建实际的资源，可能会产生一些费用。*





## 理解配置文件

Terraform 所有的配置文件以 tf 作为后缀名。在执行相关命令时，Terraform 会自动加载当前目录下的\*.tf文件。

Terraform 的配置文件是 HashiCorp 公司的 [HCL 语言](#)。

## Terraform init

与 git 类似，我们需要在 Terraform 项目的根目录运行 terraform init 去初始化项目。

在初始化项目的时候，Terraform 会解析目录下的\*.tf文件并加载相关的 provider插件。在 wordpress 文件夹下运行 terraform init 会看到类似下面的输出：

```
$ terraform init
Initializing modules...
- module.qingcloud
- module.wordpress
```

```
Initializing provider plugins...
```

```
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.null: 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.
```

## 验证 Terraform init

在 wordpress 文件夹下运行 terraform -v 会得到类似下面的输出：

```
$ terraform -v
Terraform v0.11.1
+ provider.null v1.0.0
+ provider.qingcloud (v1.1.0)
```

## 指定 Provider

在 ./provider.tf 文件我们指定了 Provider，QingCloud 的 Provider 需要 access\_key 与 secret\_key 进行调用 API，key 可以在 QingCloud Web 控制台进行申请。

zone 指定了资源会在哪个区中进行创建，默认为 Pek3a 区。

## 理解 Resource

HCL 语言是一种声明式语言，即在 \*.tf 文件中声明了我们所期望的资源状态。

我们在 ./modules/qingcloud/qingcloud.tf 文件当中指定了我们想要的资源以及他们的状态。

在定义的资源的时候我们可以在一个资源当中引用其他资源的字段，Terraform 会自动解析这些引用并且按顺序进行创建。

```
resource "qingcloud_security_group_rule" "ssh-wordpress-in" {
  security_group_id = "${qingcloud_security_group.foo.id}"
  //引用别名为foo的qingcloud_security_group的id
  protocol          = "tcp"
  priority           = 0
  action             = "accept"
  direction          = 0
  from_port          = 22
  to_port            = 22
}
```

在上面的这一小段代码当中，qingcloud\_security\_group\_rule 为资源的名称，需要 Provider 支持特定的资源。

ssh-wordpress-in 为资源的别名，是在这个项目当中唯一的。

上面我们创建了一个类型为 qingcloud\_security\_group\_rule 的资源，也就是一个防火墙规则资源。

在这个资源中我们指定了防火墙的ID，以及规则的协议、优先级、动作、方向以及端口范围。

注意：在创建资源 `qingcloud_keypair.foo` 的时候我们是上传用户的 `> ~/.ssh/id_rsa.pub` 文件，

这个公钥需要用户自己去生成，具体生成方法可以参考文档  
同样我们在使用 `provisioner` 安装软件环境时使用了 `~/.ssh/id_rsa` 作为私钥进行了远程连接，如果需要修改的话请自行修改配置文件。

在整个例子的源码当中，我们分别创建了下列资源：

- `qingcloud_eip.foo`：创建一个带宽为2的弹性公网IP
- `qingcloud_keypair.foo`：使用 `~/.ssh/id_rsa.pub` 的文件内容创建一个SSH key
- `qingcloud_security_group.foo`：创建一个名称为 `first_sg` 的防火墙
- `qingcloud_security_group_rule.http-in`：为防火墙添加一条接收80端口TCP请求的规则
- `qingcloud_security_group_rule.ssh-wordpress-in`：为防火墙添加一条接收22端口TCP请求的规则
- `qingcloud_security_group_rule.ssh-mysql-in`：为防火墙添加一条接收2222端口TCP请求的规则
- `qingcloud_vpc.foo`：创建一个vpc网络，并且绑定了防火墙与弹性公网IP，VPC的子网范围为 `192.168.0.0/16`
- `qingcloud_vxnet`：创建一个受管的vxnet，并且加入VPC当中，子网范围是 `192.168.0.0/24`
- `qingcloud_instance.wordpress`：创建一个实例，绑定了上面创建的SSH key，并且加入到了vxnet当中
- `qingcloud_instance.mysql`：创建一个实例，绑定了上面创建的SSH key，并且加入到了vxnet当中
- `qingcloud_vpc_static.http-portforward`：为VPC添加一条端口转发规则，将80端口的请求转发到instance的80端口当中
- `qingcloud_vpc_static.ssh-wordpress`：为VPC添加一条端口转发规则，将22端口的请求转发到 `qingcloud_instance.wordpress` 的22端口当中
- `qingcloud_vpc_static.ssh-mysql`：为VPC添加一条端口转发规则，将2222端口的请求转发到 `qingcloud_instance.mysql` 的2222端口当中

如果需要获取更多有关terraform基本操作的信息，可以参考官方文档的[Getting Started](#)部分。

获取更多资源的使用信息，可以查看terraform-qingcloud-provider的[文档](#)

## 利用Built-in Functions避免手工操作

在创建 `resource qingcloud_keypair` 的时候，我们使用了Terraform的内置函数去帮助获取文件 `~/.ssh/id_rsa.pub` 的内容。从而避免了我们手工复制粘贴

```
~/.ssh/id_rsa.pub的文件内容
resource "qingcloud_keypair" "foo" {
```

```
    public_key = "${file("~/ssh/id_rsa.pub")}"
}
```

Terraform 内置了许多函数帮助用户解决一些常见操作，如果需要获取更多的信息，请参考[官方文档](#)。

## 使用 Provisioners 进行环境配置

Provisioners 可以在资源创建/销毁时在本地/远程执行脚本。

Provisioners 通常用来引导一个资源，在销毁资源前完成清理工作，进行配置管理等。

Provisioners 拥有多种类型可以满足多种需求，如：文件传输（file），本地执行（local-exec），远程执行（remote-exec）等 Provisioners 可以添加在任何的 resource 当中：

```
resource "qingcloud_instance" "foo" {
  # ...

  provisioner "local-exec" {
    command = "echo ${self.private_ip} > file.txt"
  }
}
```

在 example 当中，我们使用了 null\_resource 和 provisioner 完成了 qingcloud\_instance 上安装 docker 并启动 wordpress 与 mysql。在 null\_resource.run\_docker\_wordpress 当中，我们指定了 depends\_on 参数，保证了在 mysql 已经启动成功后再启动 wordpress。

## 执行 Terraform plan 查看 Terraform 计划

Terraform plan 命令用于输出执行计划。除非明确禁用，Terraform 会调用 refresh 方法重新查询当前资源的状态。

完成状态刷新后，Terraform 会自动分析要进行的操作以达到配置文件中所需要的状态，并把分析的结果输出出来。在 vpc\_one\_instance 文件夹下执行 Terraform plan 会得到类似下面的结果：

```
$ 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.
```

```
-----
-----
```

```
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:

```
+ module.qingcloud.qingcloud_eip.foo
  id:                <computed>
  addr:              <computed>
  bandwidth:         "2"
  billing_mode:       "bandwidth"
  need_icp:           "0"
  resource.%.:        <computed>
  tag_names.#:        <computed>

+ module.qingcloud.qingcloud_instance.mysql
  id:                <computed>
  cpu:                "1"
  image_id:           "centos73x64"
  instance_class:     "0"
  keypair_ids.#:      <computed>
  managed_vxnet_id:   "${qingcloud_vxnet.foo.id}"
  memory:             "1024"
  private_ip:         <computed>
  public_ip:          <computed>
  security_group_id: <computed>
  tag_names.#:        <computed>
```

.....

Plan: 15 to add, 0 to change, 0 to destroy.

-----  
-----

Note: You didn't specify an "-out" parameter to save this plan, so Terraform can't guarantee that exactly these actions will be performed if "terraform apply" is subsequently run.

## 使用module进行代码的组织管理

Terraform中的模块是以组的形式管理不同的Terraform配置。模块用于在Terraform中创建可重用组件，以及用于基本代码组织。每一个module都可以定义自己的input与output，方便代码进行模块化组织。

在例子当中我们将配置文件分成了两个module进行处理：module qingcloud负责在qingcloud创建所需要的基础设施资源。module wordpress负责在创建好的虚机当中安装docker并且启动wordpress与mysql。

其中需要安装wordpress的机器信息是通过input传入进来的，而传入进来的input实际上是module qingcloud的output。通过input与output，我们将两个模块连接到了一起。在./module.tf当中，我们调用了两个module指定了两个module的参数传递关系。

## 使用 Terraform apply 提交资源创建及配置

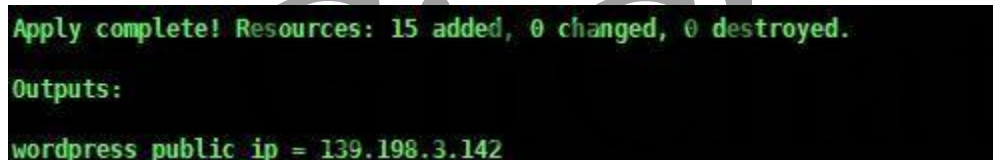
Terraform apply 命令用于应用所需的更改以达到所需的配置状态。为了更加方便的得到我们所关注的输出结果，可以使用 output 单独输出部分字段。如在 ./module.tf 当中，我们单独获取了 module.wordpress 的public\_ip：

```
output "wordpress_public_ip" {  
    value = "${module.wordpress.public_ip}"  
}
```

填写 provider.tf 中的 access\_key 与 secret\_key 后，我们使用 terraform apply 可以完成资源的创建与配置。

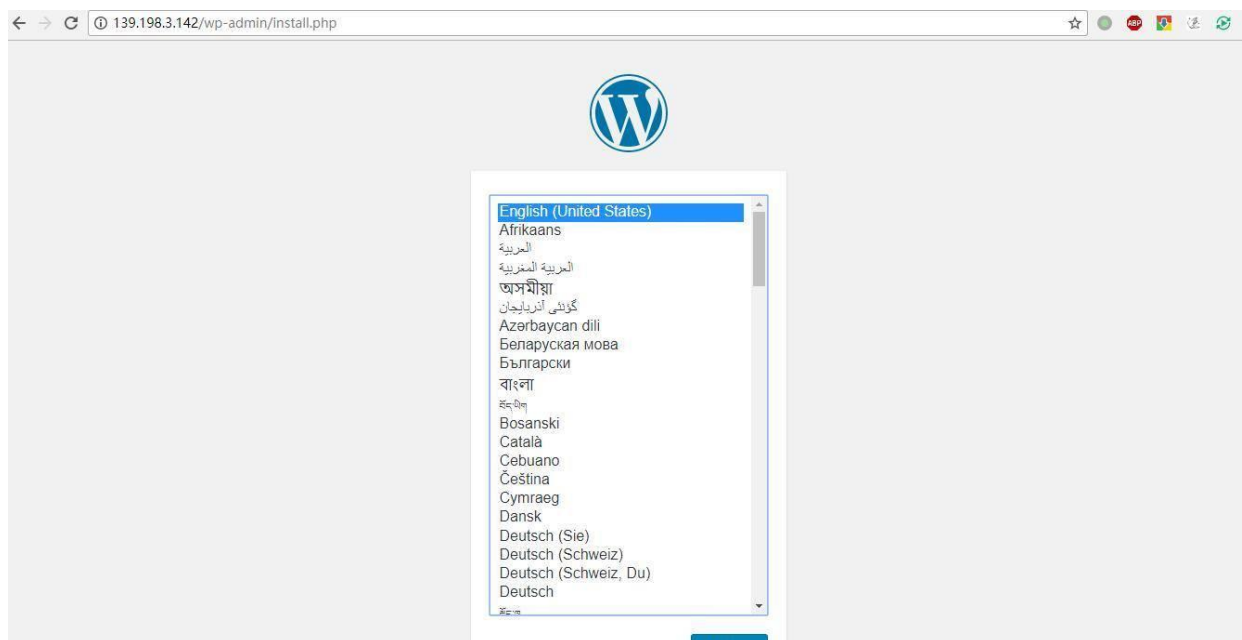
注意：在example中是根据文件来获取SSH key，在您机器上可能不存在此文件，请您自行创建SSH key。

我们会在输出的结尾获取到类似下图的输出：

A terminal window with a black background and green text. The text shows the result of a Terraform apply command: 'Apply complete! Resources: 15 added, 0 changed, 0 destroyed.' followed by 'Outputs:' and 'wordpress\_public\_ip = 139.198.3.142'.

```
Apply complete! Resources: 15 added, 0 changed, 0 destroyed.  
Outputs:  
wordpress_public_ip = 139.198.3.142
```

打开浏览器，输入output的IP，可以看到wordpress已经正常运行：



## 总结

## 多层应用的部署

一般来讲应用都是分为N层架构的，而我们的例子是一个非常典型的二层应用，分别是业务逻辑层的wordpress和数据层的mysql。

Terraform确保数据库层在Web服务器启动前可用。这得益于Terraform可以自动的去解析资源之间的关系，保证了有依赖关系的各层可以按顺序进行创建。

## 多云环境的部署

人们通常将基础架构分布在多个云中以提高容错性。通过仅使用单个区域或云提供商，容错受限于该提供商的可用性。进行多云部署可以更好地恢复地区或整个提供商的损失。

Terraform 是与云无关的，我们可以使用不同的 provider 实现多云环境的部署。并且可以将一个项目拆分为多个 module 实现代码的复用。

前面的例子当中，我们分为了两个 module，其中 module wordpress 是不依赖于云环境的 module，我们可以在不同云提供商中复用这个 module。

在同一个项目中同样可以使用多个提供者，甚至还能处理多个云当中的依赖关系。这可以帮助用户创建大型的云基础架构。

## 软件定义网络的配置

软件定义网络（SDN）在数据中心中越来越流行，它为用户提供了更多的控制权，同时也带来了人为管理负担。一般来讲SDN分为控制层与转发层。

Terraform可以编写SDN的配置文件，这些配置可以由Terraform自动调用控制层的接口生效。这些配置文件是可以进行版本化的。

例如，QingCloud VPC是一种非常典型的SDN，这种资源我们是可以利用Terraform进行管理，完成控制层的配置。

## 一次性测试环境

使用Terraform测试环境是可以被编码的，这些配置文件可以在QA、开发等团队中进行分享，可以极大减少开发测试团队重复准备环境的负担，减少人为错误，提高工作效率。

并且Terraform可以一键的进行资源的创建与删除，这可以帮助我们快速的创建测试环境，完成使用后可以及时的删除。

PS: QingCloud作为全球首家实现资源秒级响应并按秒计量的基础云服务商，使用 terraform-qingcloud可以让用户的成本最大限度的贴合实际的资源使用情况。\*

## Reference

[Terraform 官网](#)

[Terraform 青云官方 Github](#)

[Why we use Terraform](#)

# GitChat