文档中心

资源编排 Terraform

产品文档

目录

概览

产品概述

UCloud Terraform 是基于 Hashicorp 公司开源的 Terraform 实现的多云资源编排工具，用户可以通过编写 HCL(Hashicorp Configuration Language) 规格文件，实现对基础设施的自动化管理。

UCloud 目前已正式接入 Terraform 官方仓库，通过编写 Terraform 模板来管理 UCloud 资源，安全又高效，可以实现对基础架构的自动化部署、快速迁移等，配合 CLI、Ansible 等工具，可以进一步拓展 Terraform 的功能，实现基础架构可编程。

Write, Plan, and Create Infrastructure as Code.

产品优势

相对于 UCloud 其它资源管理方式(如控制台,程序调用 API,程序调用 SDK)来说,基于 Terraform 的资源编排系统拥有其不可替代的优势。

具有良好可读性的编排计划

UCloud Terraform 资源编排工具在执行真正的资源编排动作前,会生成一份可读性较好的编排计划,类似于 SQL 对数据库执行变更操作前生成的执行计划。

由于编排计划可以在执行真正的编排动作前生成,所以在执行编排动作前可以通过人工审查基础设施的重大变更,保障关键基础设施的安全性。

同时基于该特性,基础设施的管理者可以生成一份针对于资源定义文件的上一个版本的反向编排计划,从而实现基础设施的回滚。

基于 DAG 的资源依赖图管理

UCloud Terraform 资源编排工具将所有的资源构建为一张有向无环图(DAG),计算它们的依赖关系,并且并行地去创建和修改相互间没有依赖的那些资源。因此整个基础设施的构建过程是非常高效的,并且是严格有序的。

由此我们可以轻松构建和管理资源的拓扑,任何资源间的依赖都可以被明确地抽象和定义,编排工具可以帮助使用者完成资源间关系的实际构建工作。

统一的资源状态管理

UCloud Terraform 资源编排工具引入了面向资源的设计,更贴近于现代编程习惯。

资源编排工具将资源的状态描述为一个状态的集合,并支持若干种不同类型的状态存储。默认情况下,在 terraform cli 的执行目录下,会存储一个本地的资源状态文件,并在每次编排开始时,从远程同步状态到本地,比较该状态与用户定义的资源之间的差异,从而生成编排计划。

对于资源状态的统一管理,使得基于 Terraform 的资源编排系统可以保持基础设施的一致性。例如,使用 etcd 等远程强一致存储作为 Terraform 的后端状态存储,可以对资源状态的操作加锁,使得多个用户不会同时操作相同的基础设施实例。

安全且高效的重试策略

UCloud Terraform 资源编排工具实现了安全且高效的重试策略,由于工具内部存储了当前资源的状态,所以在资源变更时无法:

- 在自动化的环境中,比如 CI 执行环境下,对于偶发性质的问题,可以通过简单的重试策略来保障基础设施编排的可用性,大幅减少人工干预。
- 对于部分支持属性局部更新的产品,当编排失败重试时,仅更新失败的部分属性。

与其它工具的对比

配置管理工具（如 Chef，Puppet，Ansible 等）

配置管理工具用于安装和管理已存在的机器上的软件。资源编排工具不是一个配置管理工具，资源编排工具的主要关注点在于如何创建和初始化资源。

使用资源编排工具的 Provisioner 功能，可以与配置管理工具有机地结合在一起。资源编排工具主要关注对于数据中心资源更高层的抽象，尤其是资源之间的关联。与配置管理工具的功能并不重叠。一个比较好的实践是，由资源编排工具创建资源，配置管理工具配置机器，可以享受配置管理工具所带来的全部便利。

友商的资源编排系统，如 AWS CloudFormation，阿里 ROS

经过长时间的发展，Terraform 已经成为一个业内通用的资源编排工具，UCloud 兼容 Terraform 的协议，相比于友商的资源编排系统，适用性更强。而且近年来友商也陆续开始支持基于 Terraform 的资源编排系统，证明了业内对通用资源编排系统的强需求。

UCloud 资源编排工具集从最初就保持着对 Terraform 协议的兼容，使得围绕着 UCloud 公有云，集成其它服务商的卫星服务，进行多云编排的架构成为可能，比如将 UCloud EIP 与 DNSimple 集成在一起，轻松搭建一个云数据中心的公网流量入口。

另外一个不同点是，UCloud 的编排计划被抽象出来作为一等公民，可以在编排系统资源前充分审查变更，保证了关键基础设施的可靠性。

基于 API/SDK 自行研发

大多数企业是基于 API/SDK 手动编写一个简单脚本开始自动化运维之路的。这常常适用于最小化的概念验证与原型开发，随着规模的增长，这些方法往往容易出错且不容易修改。它们通常需要时间和资源来构建和维护，且目的在于满足短期需求，由于工具必须与任何新功能或基础架构保持同步更新，因此它成为基础架构发展速度的限制因素。

UCloud Terraform 资源编排工具基于 Hashicorp 公司开源的 Terraform 工具，使用简单且统一的 HCL(Hashicorp Configuration Language) 语法，几乎可以管理任何资源而无需学习新的工具。通过定义所需的所有资源，可以自动解决它们之间的依赖关系，以便基础设施管理员不需要记住和推理它们。消除构建工具的负担使运营商能够专注于他们的基础设施而不是工具。在多云环境下，Hashicorp Terraform 已经积累充足的案例，其中包括数十万行代码的基础设施构建实例，经过了充分的实践检验。

此外，Terraform 是一个开源工具。除了 HashiCorp 之外，Terraform 周围的社区还有助于扩展其功能，修复错误并记录新的用例。Terraform 有助于解决每个组织中存在的问题，并提供可采用的标准，以避免在组织之间和组织内重新发明轮子。它的开源性确保它将长期存在。

快速开始

快速链接

官方产品文档地址（本文档）

对于产品有一个大致的了解

官方参考文档地址

用于查询 UCloud Terraform Provider 的各种参数

开源仓库地址

欢迎对 UCloud Provider 贡献代码，成为 Contributor！

环境配置

安装 Terraform

- 打开 官方安装文档
- 按照文档安装 Terraform CLI 工具（可选）

配置默认用户

- 设置秘钥 UCLOUD_PUBLIC_KEY 和 UCLOUD_PRIVATE_ KEY 为全局环境变量（推荐），或在 HCL 文件中显式指定 public_key 和 private_key 参数。
- 如果是子账号，应配置项目ID UCLOUD_PROJECT_ID 为全局环境变量（推荐），或在 HCL 文件中显式指定 project_id 参数。

编写 HCL 文件

让我们以初始化一个主机为目标，首先创建一个干净的空文件夹作为工作区，并且换到该目录下，编写一个 HCL 规格文件(eg：main.tf)，如下：

```
# 配置 terraform 依赖（0.13 之后必须）
terraform {
required_providers {
ucloud = {
source = "ucloud/ucloud"
version = "~>1.23.0"
```

```
}
}
}

# 配置 UCloud terraform provider
provider "ucloud" {
# public_key = var.ucloud_public_key
# private_key = var.ucloud_private_key
# project_id = var.ucloud_project_id
region = "cn-bj2"
}

# 查询指定可用区中的主机镜像
data "ucloud_images" "default" {
availability_zone = "cn-bj2-05"
name_regex = "^CentOS 7.[1-2] 64"
image_type = "base"
}

# 创建一台 UCloud 云主机
resource "ucloud_instance" "example" {
availability_zone = "cn-bj2-05"
image_id = data.ucloud_images.default.images[0].id
instance_type = "n-standard-1"
root_password = "UCloud_2020"
name = "tf-instance-example"
}
```

除了自己编写 HCL 规格文件,还可以将 代码仓库 中的 example 拷贝进来,完成相应的部署。

初始化编排工具

在当前目录下执行 terraform init 命令,初始化工作区:

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

```
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.
```

查看编排计划（可选）

在当前目录下执行 terraform plan 命令，查看编排计划：

```
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.ucloud_images.default: 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:

+ ucloud_instance.example
id: <computed>
auto_renew: <computed>
availability_zone: "cn-bj2-05"
boot_disk_size: <computed>
boot_disk_type: "local_normal"
charge_type: "month"
cpu: <computed>
```

```
create_time: <computed>
data_disk_size: <computed>
data_disk_type: "local_normal"
disk_set.#: <computed>
duration: "1"
expire_time: <computed>
image_id: "uimage-eqdgkb"
instance_type: "n-standard-1"
ip_set.#: <computed>
memory: <computed>
name: "tf-instance-example"
remark: <computed>
root_password: <sensitive>
security_group: <computed>
status: <computed>
subnet_id: <computed>
tag: <computed>
vpc_id: <computed>


Plan: 1 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.
```

执行编排计划

在当前目录下执行 terraform apply 命令并确认，执行编排计划（创建一台云主机）：

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes
```

```
ucloud_instance.example: Creating...
auto_renew: "" => "<computed>"
availability_zone: "" => "cn-bj2-05"
boot_disk_size: "" => "<computed>"
boot_disk_type: "" => "local_normal"
charge_type: "" => "month"
cpu: "" => "<computed>"
create_time: "" => "<computed>"
data_disk_size: "" => "<computed>"
data_disk_type: "" => "local_normal"
disk_set.#: "" => "<computed>"
duration: "" => "1"
expire_time: "" => "<computed>"
image_id: "" => "uimage-eqdgkb"
instance_type: "" => "n-standard-1"
ip_set.#: "" => "<computed>"
memory: "" => "<computed>"
name: "" => "tf-instance-example"
remark: "" => "<computed>"
root_password: "<sensitive>" => "<sensitive>"
security_group: "" => "<computed>"
status: "" => "<computed>"
subnet_id: "" => "<computed>"
tag: "" => "<computed>"
vpc_id: "" => "<computed>"
ucloud_instance.example: Still creating... (10s elapsed)
ucloud_instance.example: Still creating... (20s elapsed)
ucloud_instance.example: Creation complete after 22s (ID: uhost-kayyin)

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

如果执行编排计划失败，有多种可能原因导致:

- 网络原因，则重试几次即可；
- 编排计划内容字段校验错误，则按照要求修改HCL文件（即 .tf 文件），可参考UCloud Terraform 官方文档;
- 资源不足，可切换地域或可用区；
- 其他原因，可查询常见问题，或联系技术支持。

查看编排资源状态（可选）

执行 terraform show 命令，查看编排资源状态（查看云主机状态）：

```
data.ucloud_images.default:
availability_zone = cn-bj2-05
image_type = base
name_regex = ^CentOS 7.[1-2] 64
images.0.id = uimage-eqdgkb
...
ucloud_instance.example:
id = uhost-kayyin
auto_renew = true
availability_zone = cn-bj2-05
boot_disk_size = 20
boot_disk_type = local_normal
charge_type = month
cpu = 1
create_time = 2018-12-19T16:55:21+08:00
data_disk_type = local_normal
disk_set.# = 1
disk_set.0.id = 3ff13c07-f368-4a08-8236-bde2a54bb36c
disk_set.0.is_boot = true
disk_set.0.size = 20
disk_set.0.type = local_normal
duration = 1
expire_time = 2019-01-19T16:55:21+08:00
image_id = uimage-eqdgkb
instance_type = n-standard-1
ip_set.# = 1
ip_set.0.internet_type = Private
ip_set.0.ip = 10.42.75.127
memory = 4096
name = tf-instance-example
remark =
root_password = UCloud_2020
security_group =
```

```
status = Running
subnet_id = subnet-hjohfy
tag = Default
vpc_id = uvnet-300sly
```

销毁编排资源（可选）

如需销毁当前编排资源，执行 terraform destroy 命令并确认，销毁编排资源（删除云主机）：

```
data.ucloud_images.default: Refreshing state...
ucloud_instance.example: Refreshing state... (ID: uhost-kayyin)

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

Terraform will perform the following actions:

- ucloud_instance.example


Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

ucloud_instance.example: Destroying... (ID: uhost-kayyin)
ucloud_instance.example: Destruction complete after 7s

Destroy complete! Resources: 1 destroyed.
```

UHost Instance 机型表

快杰型 O

- 简介：计算、存储与网络性能卓越的最新一代云主机。适合全面需求场景。
- CPU 平台支持：Intel Cascadelake, Amd Epyc2
- CPU 内存组合（支持配比1:1-1:8）
- 单位：CPU - 核数 Memory - GB
- 限制：
  - 目前仅在部分可用区支持，详情请见控制台
  - 仅支持 cloud_rssd 云盘做为系统盘
  - 基础镜像仅支持高内核版及更高
  - 范围：CPU: Intel 1-64, Amd 1-96; Memory: Intel 1-256, Amd 4-768

Category	High CPU（1:1）			Basic（1:2）			Standard（1:4）			High Memory（1:8）		
OutStanding (O)	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory
	o-highcpu-1	1	1	o-basic-1	1	2	o-standard-1	1	4	o-highmem-1	1	8
	o-highcpu-2	2	2	o-basic-2	2	4	o-standard-2	2	8	o-highmem-2	2	16
	o-highcpu-4	4	4	o-basic-4	4	8	o-standard-4	4	16	o-highmem-4	4	32
	o-highcpu-8	8	8	o-basic-8	8	16	o-standard-8	8	32	o-highmem-8	8	64
	o-highcpu-16	16	16	o-basic-16	16	32	o-standard-16	16	64	o-highmem-16	16	128
	o-highcpu-32	32	32	o-basic-32	32	64	o-standard-32	32	128	o-highmem-32	32	256
	o-highcpu-64	64	64	o-basic-64	64	128	o-standard-64	64	256	o-highmem-64	64	512
	o-highcpu-96	96	96	o-basic-96	96	192	o-standard-96	96	384	o-highmem-96	96	768

快杰型 S(OS)

- 简介: 快杰主机再次升级，支持Intel Cascadelake-Refresh CPU（主频3.0GHz），单核性能提高20%。
- CPU 平台支持: Intel CascadelakeR

- CPU 内存组合 (支持配比: 1:1-1:8)
- 单位: CPU-核数 Memory-GB
- 范围: CPU: Intel 1-64; Memory: Intel 1-512
- Limit:
  - 目前仅在部分可用区支持,详情请见控制台
  - 仅支持 cloud_rssd 云盘做为系统盘和数据盘
  - 基础镜像仅支持高内核版及更高
  - Must set boot_disk_type to cloud_rssd

Category	High CPU（1:1）			Basic（1:2）			Standard（1:4）			High Memory（1:8）		
OutStanding S(OS)	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory
	os-highcpu-1	1	1	os-basic-1	1	2	os-standard-1	1	4	os-highmem-1	1	8
	os-highcpu-2	2	2	os-basic-2	2	4	os-standard-2	2	8	os-highmem-2	2	16
	os-highcpu-4	4	4	os-basic-4	4	8	os-standard-4	4	16	os-highmem-4	4	32
	os-highcpu-8	8	8	os-basic-8	8	16	os-standard-8	8	32	os-highmem-8	8	64
	os-highcpu-16	16	16	os-basic-16	16	32	os-standard-16	16	64	os-highmem-16	16	128
	os-highcpu-32	32	32	os-basic-32	32	64	os-standard-32	32	128	os-highmem-32	32	256
	os-highcpu-64	64	64	os-basic-64	64	128	os-standard-64	64	256	os-highmem-64	64	512

通用型 N

- 简介:提供最灵活自由的CPU、内存、磁盘组合。适合计算、存储、网络等均衡的场景。
- CPU 平台支持:Intel IvyBridge/Haswell/Broadwell/Skylake
- CPU 内存组合(支持配比2:1-1:12)
- 单位:CPU - 核数 Memory - GB
- 范围:CPU: 1-32, Memory: 1-128

Category	High CPU（1:1）			Basic（1:2）			Standard（1:4）			High Memory（1:8）			Customized（2:1-1:12）		
	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory

Normal (N)	n-highcpu-1	1	1	n-basic-1	1	2	n-standard-1	1	4	n-highmem-1	1	8	n-customized-2-1	2	1
	n-highcpu-2	2	2	n-basic-2	2	4	n-standard-2	2	8	n-highmem-2	2	16	n-customized-2-14	2	14
	n-highcpu-4	4	4	n-basic-4	4	8	n-standard-4	4	16	n-highmem-4	4	32	n-customized-4-18	4	18
	n-highcpu-6	6	6	n-basic-6	6	12	n-standard-6	6	24	n-highmem-6	6	48	...	...	...
	n-highcpu-8	8	8	n-basic-8	8	16	n-standard-8	8	32	n-highmem-8	8	64	n-customized-4-48	4	48
	n-highcpu-10	10	10	n-basic-10	10	20	n-standard-10	10	40	n-highmem-10	10	80	...	...	...
	n-highcpu-12	12	12	n-basic-12	12	24	n-standard-12	12	48	n-highmem-12	12	96			
	n-highcpu-14	14	14	n-basic-14	14	28	n-standard-14	14	56	n-highmem-14	14	112			
	n-highcpu-16	16	16	n-basic-16	16	32	n-standard-16	16	64	n-highmem-16	16	128			
	n-highcpu-18	18	18	n-basic-18	18	36	n-standard-18	18	72						
	n-highcpu-20	20	20	n-basic-20	20	40	n-standard-20	20	80						
	n-highcpu-22	22	22	n-basic-22	22	44	n-standard-22	22	88						
	n-highcpu-24	24	24	n-basic-24	24	48	n-standard-24	24	96						
	n-highcpu-26	26	26	n-basic-26	26	52	n-standard-26	26	104						
	n-highcpu-28	28	28	n-basic-28	28	56	n-standard-28	28	112						
	n-highcpu-30	30	30	n-basic-30	30	60	n-standard-30	30	120						
	n-highcpu-32	32	32	n-basic-32	32	64	n-standard-32	32	128						

高主频型 (C)

- 简介: CPU主频≥3.0GHz的机型，适合计算类业务，如高频交易、渲染、人工智能等。
- CPU 平台支持: Intel Skylake
- CPU 内存组合 (support ratio: 1:1-1:8)
- 单位: CPU-kernel Memory-GB

- 范围:CPU: 1-32, Memory: 1-128
- 限制:
  - 目前仅在部分可用区支持,详情请见控制台

Category	High CPU（1:1）			Basic（1:2）			Standard（1:4）			High Memory（1:8）		
	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory	InstanceType	CPU	Memory
High Frequency (C)	c-highcpu-1	1	1	c-basic-1	1	2	c-standard-1	1	4	c-highmem-1	1	8
	c-highcpu-2	2	2	c-basic-2	2	4	c-standard-2	2	8	c-highmem-2	2	16
	c-highcpu-4	4	4	c-basic-4	4	8	c-standard-4	4	16	c-highmem-4	4	32
	c-highcpu-8	8	8	c-basic-8	8	16	c-standard-8	8	32	c-highmem-8	8	64
	c-highcpu-16	16	16	c-basic-16	16	32	c-standard-16	16	64	c-highmem-16	16	128
	c-highcpu-32	32	32	c-basic-32	32	64	c-standard-32	32	128			

DB Instance 机型表

高可用型 HA

MySQL/Percona DB

- 简介:高可用版提供双主热备架构,避免因宕机或硬件故障造成的数据库不可用,且高可用版使用InnoDB引擎对于主从同步及容灾更佳。
- 内存:支持 2, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256, 320 (单位 GB)

Category	Mysql		Percona	
	InstanceType	Memory	InstanceType	Memory
Mysql/Percona DB	mysql-ha-1	1	percona-ha-1	1
	mysql-ha-2	2	percona-ha-2	2
	mysql-ha-4	4	percona-ha-4	4
	mysql-ha-6	6	percona-ha-6	6
	mysql-ha-8	8	percona-ha-8	8
	mysql-ha-12	12	percona-ha-12	12
	mysql-ha-16	16	percona-ha-16	16
	mysql-ha-24	24	percona-ha-24	24
	mysql-ha-32	32	percona-ha-32	32
	mysql-ha-48	48	percona-ha-48	48
	mysql-ha-64	64	percona-ha-64	64
	mysql-ha-96	96	percona-ha-96	96
	mysql-ha-128	128	percona-ha-128	128
	mysql-ha-192	192	percona-ha-192	192
	mysql-ha-256	256	percona-ha-256	256
	mysql-ha-320	320	percona-ha-320	320

PostgreSQL DB

- 简介:UDB PostgreSQL 多道防线保障了数据的安全可靠,高可靠性硬件保障了存储的数据有保障;PostgreSQL 的高可用实例保证了有多个数据的冗余存储。
- 内存:支持 2, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128 (单位 GB)

Category	PostgreSQL	
	InstanceType	**Memory**
PostgreSQL DB	postgresql-ha-2	2
	postgresql-ha-4	4
	postgresql-ha-6	6
	postgresql-ha-8	8
	postgresql-ha-12	12
	postgresql-ha-16	16
	postgresql-ha-24	24
	postgresql-ha-32	32
	postgresql-ha-48	48
	postgresql-ha-64	64
	postgresql-ha-96	96
	postgresql-ha-128	128

高可用 NVMe SSD (public beta)

- 简介: 高可用性 NVMe SSD 版本使用双主热备结构,这是新一代超高性能云磁盘产品,适用于具有高容量和低延迟要求的业务场景。
- 内存: 支持 2, 4, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256, 320 (单位 GB)
- 限制:
  - 目前仅在部分可用区支持,详情请见控制台
  - 目前仅支持部分 engine 和 engine_version, 暂时支持的有 engine: mysql; engine_version: 5.6, 5.7; 详情请见控制台

Category	Mysql	
	InstanceType	**Memory**
	mysql-ha-nvme-2	2
	mysql-ha-nvme-4	4
	mysql-ha-nvme-8	8
	mysql-ha-nvme-12	12

Mysql DB	mysql-ha-nvme-16	16
	mysql-ha-nvme-24	24
	mysql-ha-nvme-32	32
	mysql-ha-nvme-48	48
	mysql-ha-nvme-64	64
	mysql-ha-nvme-96	96
	mysql-ha-nvme-128	128
	mysql-ha-nvme-192	192
	mysql-ha-nvme-256	256
	mysql-ha-nvme-320	320

Redis 与 Memcache 机型表

Redis

- 简介:UCloud 云内存 Redis 提供主备版 Redis 和分布式版 Redis 两种架构,基于高可靠双机热备架构及可平滑扩展的集群架构,满足高读写性能场景及弹性扩缩容的业务需求。
- 内存(单位 GB): 主备版支持 1, 2, 4, 6, 8, 12, 16, 24, 32, 40, 52, 64; 分布式版支持 16 到 1000 且必须被 4 整除。

Category	Active-Standby		Distributed	
	InstanceType	Memory	InstanceType	Memory
Redis	redis-master-1	1	redis-distributed-16	16
	redis-master-2	2	redis-distributed-20	20
	redis-master-4	4	redis-distributed-24	24
	redis-master-6	6	redis-distributed-28	28
	redis-master-8	8	...	...
	redis-master-12	12	redis-distributed-996	996
	redis-master-16	16	redis-distributed-1000	1000
	redis-master-24	24		
	redis-master-32	32		
	redis-master-40	40		
	redis-master-52	52		
	redis-master-64	64		

Memcache

- 简介:云内存 Memcache 是基于内存的缓存服务,支持海量小数据的高速访问。云内存 Memcache 可以极大缓解后端存储的压力,提高网站或应用的响应速度。云内存 Memcache 支持 Key-Value 的数据结构,兼容 Memcached 协议的客户端都可与云内存 Memcache 进行通信。
- 内存(单位 GB): 主备版支持 1, 2, 4, 8, 16, 24, 32。

Category	Active-Standby	
Memcache	InstanceType	Memory
	memcache-master-1	1
	memcache-master-2	2
	memcache-master-4	4
	memcache-master-8	8
	memcache-master-16	16
	memcache-master-24	24
	memcache-master-32	32

搭建一台 web 服务器

关键词:*UHost, EIP, UDisk*

摘要

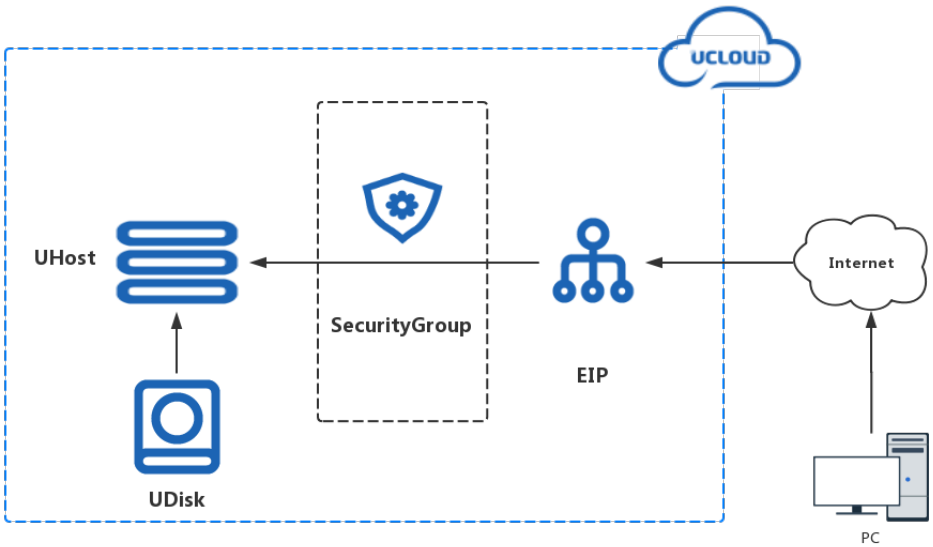
云主机是构建在云环境的弹性计算资源，是 UCloud 最为核心的服务。有些服务，如弹性 IP、镜像、云硬盘等必须与云主机结合后使用，另一些服务，如数据库、缓存、对象存储等可以和云主机结合共同构建 IT 环境。

此案例使用 Terraform 创建一台 web 服务器基础设施，通过创建一台云主机并在云主机上绑定云硬盘和外网弹性IP，同时使用外网防火墙来保护云主机的网络安全性。

使用 Terraform 来创建云主机可以享有由基础设施即代码 (IaC) 带来的便利。通过编写 HCL 文件，可以快速构建包含基础设施定义和它们之间关联的拓扑，并借助于代码版本管理工具，将基础设施的变更纳入版本控制中。

此案例需要一个可用的 UCloud 帐号，以及确保目标可用区有足够的权限和配额可以创建云主机，EIP 和 UDisk。可以在下方 **操作步骤**拷贝使用，或克隆 官方仓库 以获取完整的 案例演示代码.

拓扑图



操作步骤

定义资源

首先创建基础设施代码文件。

该样例中包含:

一个 variables.tf 文件，用于定义输入参数，代码详情如下：

```
variable "region" {
default = "cn-bj2"
}

variable "zone" {
default = "cn-bj2-05"
}

variable "instance_password" {
default = "ucloud_2020"
}
```

一个 main.tf 文件，用于建立一个从云资源到代码的映射，代码详情如下：

```
# 指定 UCloud Provider 和配置信息
provider "ucloud" {
region = var.region
}

# 查询默认可用区中的主机镜像
data "ucloud_images" "default" {
availability_zone = var.zone
name_regex = "^CentOS 7.[1-2] 64"
image_type = "base"
}

# 查询默认推荐 web 外网防火墙
data "ucloud_security_groups" "default" {
```

```
type = "recommend_web"
}

# 创建一台 web 服务器
resource "ucloud_instance" "web" {
availability_zone = var.zone
image_id = data.ucloud_images.default.images[0].id
instance_type = "n-basic-2"
root_password = var.instance_password
name = "tf-example-web-server"
tag = "tf-example"
boot_disk_type = "cloud_ssd"

# the default Web Security Group that UCloud recommend to users
security_group = data.ucloud_security_groups.default.security_groups[0].id

# create cloud data disk attached to instance
data_disks {
size = 20
type = "cloud_ssd"
}
delete_disks_with_instance = true
}

# 创建外网弹性 EIP
resource "ucloud_eip" "default" {
bandwidth = 2
charge_mode = "bandwidth"
name = "tf-example-web-server"
tag = "tf-example"
internet_type = "bgp"
}

# EIP 绑定到主机
resource "ucloud_eip_association" "default" {
resource_id = ucloud_instance.web.id
eip_id = ucloud_eip.default.id
```

```
}
```

生成执行计划

在当前目录下执行 terraform plan 命令，查看编排计划：

```
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.ucloud_images.default: Refreshing state...
data.ucloud_security_groups.default: 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:

# ucloud_eip.default will be created
+ resource "ucloud_eip" "default" {
+ bandwidth = 2
+ charge_mode = "bandwidth"
+ charge_type = (known after apply)
+ create_time = (known after apply)
+ expire_time = (known after apply)
+ id = (known after apply)
+ internet_type = "bgp"
+ ip_set = (known after apply)
+ name = "tf-example-web-server"
+ public_ip = (known after apply)
+ remark = (known after apply)
+ resource = (known after apply)
+ status = (known after apply)
+ tag = "tf-example"
```

```
}

# ucloud_eip_association.default will be created
+ resource "ucloud_eip_association" "default" {
+ eip_id = (known after apply)
+ id = (known after apply)
+ resource_id = (known after apply)
+ resource_type = (known after apply)
}

# ucloud_instance.web will be created
+ resource "ucloud_instance" "web" {
+ auto_renew = (known after apply)
+ availability_zone = "cn-bj2-05"
+ boot_disk_size = (known after apply)
+ boot_disk_type = "cloud_ssd"
+ charge_type = (known after apply)
+ cpu = (known after apply)
+ cpu_platform = (known after apply)
+ create_time = (known after apply)
+ data_disk_size = (known after apply)
+ data_disk_type = (known after apply)
+ delete_disks_with_instance = true
+ disk_set = (known after apply)
+ expire_time = (known after apply)
+ id = (known after apply)
+ image_id = "uimage-ohveag"
+ instance_type = "n-basic-2"
+ ip_set = (known after apply)
+ isolation_group = (known after apply)
+ memory = (known after apply)
+ name = "tf-example-web-server"
+ private_ip = (known after apply)
+ remark = (known after apply)
+ root_password = (sensitive value)
+ security_group = "firewall-h55aem"
+ status = (known after apply)
```

```
+ subnet_id = (known after apply)
+ tag = "tf-example"
+ vpc_id = (known after apply)

+ data_disks {
+ size = 20
+ type = "cloud_ssd"
}
}

Plan: 3 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.
```

可以看到即将创建一台云主机、一块云硬盘、一个弹性 EIP、一个主机和 EIP 之间的绑定关系，以及一个主机与云硬盘之间的挂载关系。

执行编排

执行 terraform apply 命令并确认，执行编排计划：

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes
```

可通过控制台确认资源已创建完成。

参考文献

详见 Instance docs 和 example

私有网络下批量部署多台云主机

关键词:*UHost, VPC, Subnet*

摘要

云主机是构建在云环境的弹性计算资源，是 UCloud 最为核心的服务。有些服务，如弹性 IP、镜像、云硬盘等必须与云主机结合后使用，另一些服务，如数据库、缓存、对象存储等可以和云主机结合共同构建 IT 环境。

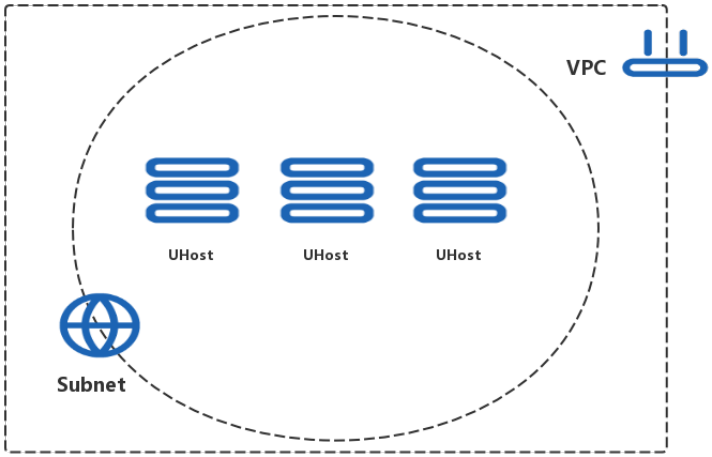
此案例使用 Terraform 并行批量创建多台云主机，并在每一台云主机上绑定 VPC, Subnet 用于网络隔离。

UCloud 是国内最早采用 SDN 技术的云计算服务商，VPC 基于 SDN 技术构建，是属于用户的、逻辑隔离的网络环境。在私有网络中，可以创建指定网段的 VPC，并在 VPC 中创建子网、自主管理云资源，同时可通过网络 ACL 实现安全防护。

使用 Terraform 来创建云主机除了享有由基础设施既代码 (IaC) 带来的便利外，还可以利用并行资源编排带来的性能提升，当基础设施十分庞大和复杂时，已定义的资源会自动被抽象为有向无环图 (DAG)，寻找尽可能的并行编排路径，以达到较优的编排性能。

此案例需要一个可用的 UCloud 帐号，以及确保目标可用区有足够的权限和配额可以创建云主机，VPC 和防火墙。可以在下方**操作步骤**中拷贝使用，或克隆 官方仓库 以获取完整的 案例演示代码.

拓扑图



操作步骤

定义资源

首先创建基础设施代码文件，可从 官方样例 中获取全部源码文件。

一个 variables.tf 文件，用于定义输入参数，代码详情如下：

```
variable "region" {
default = "cn-bj2"
}

variable "zone" {
default = "cn-bj2-05"
}

variable "instance_password" {
default = "ucloud_2020"
}

variable "instance_count" {
default = 3
}

variable "count_format" {
default = "%02d"
}
```

一个 main.tf 文件，用于建立一个从云资源到代码的映射，代码详情如下：

```
# 指定 UCloud Provider 和配置信息
provider "ucloud" {
region = var.region
}

# 查询默认可用区中的主机镜像
data "ucloud_images" "default" {
```

```
availability_zone = var.zone
name_regex = "^CentOS 7.[1-2] 64"
image_type = "base"
}

# 创建 VPC
resource "ucloud_vpc" "default" {
name = "tf-example-intranet-cluster"
tag = "tf-example"

# vpc network
cidr_blocks = ["192.168.0.0/16"]
}

# 创建 Subnet 到 VPC 下
resource "ucloud_subnet" "default" {
name = "tf-example-intranet-cluster"
tag = "tf-example"

# subnet's network must be contained by vpc network
# and a subnet must have least 8 ip addresses in it (netmask < 30).
cidr_block = "192.168.1.0/24"

vpc_id = ucloud_vpc.default.id
}

# 创建内网集群
resource "ucloud_instance" "intranet" {
count = "${var.instance_count}"

availability_zone = var.zone
image_id = data.ucloud_images.default.images[0].id
instance_type = "n-basic-2"
root_password = var.instance_password
boot_disk_type = "cloud_ssd"

# we will put all the instances into same vpc and subnet,
```

```
# so they can communicate with each other.
vpc_id = ucloud_vpc.default.id

subnet_id = ucloud_subnet.default.id

name = "tf-example-intranet-cluster-${format(var.count_format, count.index + 1)}"
tag = "tf-example"
}
```

生成执行计划

在当前目录下执行 terraform plan 命令，查看编排计划：

```
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.ucloud_zones.default: Refreshing state...
data.ucloud_images.default: 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:

+ ucloud_instance.intranet[0]
id: <computed>
auto_renew: <computed>
availability_zone: "cn-bj2-02"
boot_disk_size: <computed>
boot_disk_type: <computed>
charge_type: "month"
cpu: <computed>
create_time: <computed>
```

```
data_disk_size: <computed>
data_disk_type: <computed>
disk_set.#: <computed>
expire_time: <computed>
image_id: "uimage-f1chxn"
instance_type: "n-basic-2"
ip_set.#: <computed>
memory: <computed>
name: "tf-example-intranet-cluster-01"
private_ip: <computed>
remark: <computed>
root_password: <sensitive>
security_group: <computed>
status: <computed>
subnet_id: "${ucloud_subnet.default.id}"
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_instance.intranet[1]
id: <computed>
auto_renew: <computed>
availability_zone: "cn-bj2-02"
boot_disk_size: <computed>
boot_disk_type: <computed>
charge_type: "month"
cpu: <computed>
create_time: <computed>
data_disk_size: <computed>
data_disk_type: <computed>
disk_set.#: <computed>
expire_time: <computed>
image_id: "uimage-f1chxn"
instance_type: "n-basic-2"
ip_set.#: <computed>
memory: <computed>
name: "tf-example-intranet-cluster-02"
private_ip: <computed>
```

```
remark: <computed>
root_password: <sensitive>
security_group: <computed>
status: <computed>
subnet_id: "${ucloud_subnet.default.id}"
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_instance.intranet[2]
id: <computed>
auto_renew: <computed>
availability_zone: "cn-bj2-02"
boot_disk_size: <computed>
boot_disk_type: <computed>
charge_type: "month"
cpu: <computed>
create_time: <computed>
data_disk_size: <computed>
data_disk_type: <computed>
disk_set.#: <computed>
expire_time: <computed>
image_id: "uimage-f1chxn"
instance_type: "n-basic-2"
ip_set.#: <computed>
memory: <computed>
name: "tf-example-intranet-cluster-03"
private_ip: <computed>
remark: <computed>
root_password: <sensitive>
security_group: <computed>
status: <computed>
subnet_id: "${ucloud_subnet.default.id}"
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_subnet.default
id: <computed>
```

```
cidr_block: "192.168.1.0/24"
create_time: <computed>
name: "tf-example-intranet-cluster"
remark: <computed>
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_vpc.default
id: <computed>
cidr_blocks.#: "1"
cidr_blocks.3901788224: "192.168.0.0/16"
create_time: <computed>
name: "tf-example-intranet-cluster"
network_info.#: <computed>
remark: <computed>
tag: "tf-example"
update_time: <computed>


Plan: 5 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.
```

可以看到即将创建三台云主机、一个 VPC，一个 Subnet。

执行编排

执行 terraform apply 命令并确认，执行编排计划：

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes
```

可通过 控制台 确认资源已创建完成。

参考文献

详见 Instance docs 和 example

基于负载均衡器水平扩展的 Two-Tier 架构

关键词:*ULB, Two-Tier*

摘要

Two-Tier 架构是互联网最常见的应用部署方案,通常指由负载均衡器作为前端流量入口,通过一定的负载均衡策略,卸载流量至后端应用服务器的通用架构。

这种部署方案的特点有:

1. 应用服务器是无状态的,可以随业务的扩张尽可能地水平扩展。
2. 通过负载均衡器,可以完成用户无感知的,单一地域跨可用区的容灾切换。

ULB 是 UCloud 提供的 4/7 层负载均衡器,用于在多台云主机间实现应用程序流量的自动分配。可实现故障自动切换,提高业务可用性和资源利用率。

此案例使用 Terraform 创建如下资源:

- 创建一个 VPC 和 一个 Subnet,将接下来创建的所有资源都划进这个子网中
- 创建一台 ULB 并创建一个虚拟服务器实例用来对外提供 80 端口的监听服务,并为该 ULB 绑定一个公网弹性 IP 以供外网访问。
- 并行批量创建 N 台云主机,添加它们作为 ULB 的后端节点,配置从 ULB 到后端节点的转发规则和负载均衡策略。

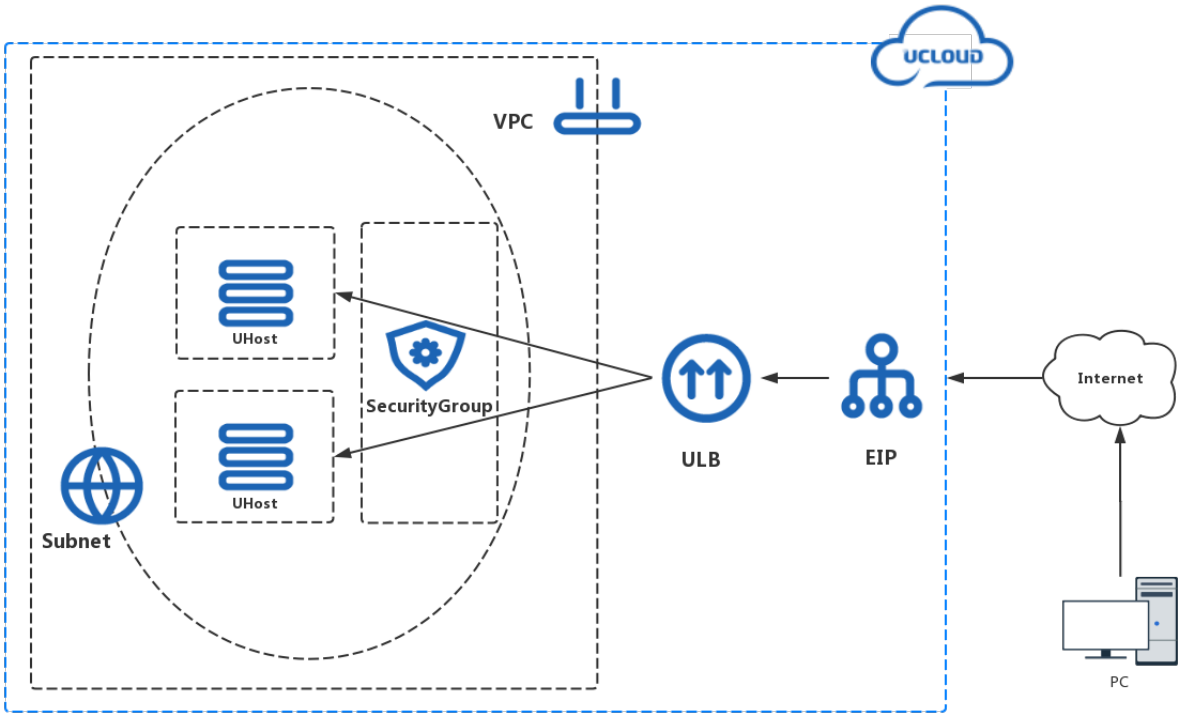
使用 Terraform 来创建云主机可以享有由基础设施既代码 (IaC) 带来的便利,定义复杂的,具有一定规模的基础设施架构,并且可以通过修改 HCL 文件,对已建成的基础设施进行任意方向的扩展。例如:

- 通过修改主机实例的定义,可以做到单一应用服务器的升降级(Scale-Up,Scale-Down)
- 通过修改主机实例的数量,可以做到 ULB 后端应用实例的规模伸缩(Scale-In,Scale-Out)

使得对基础设施的动态管理成为一个无需额外研发成本的自动化操作。

此案例需要一个可用的 UCloud 帐号,以及确保目标可用区有足够的权限和配额可以创建云主机,ULB,VPC,EIP 和防火墙。并从下方 **操作步骤** 中拷贝,或克隆 官方仓库 以获取完整的 案例演示代码。

拓扑图



操作步骤

定义资源

首先创建基础设施代码文件，可从 官方样例 中获取全部源码文件，此源码文件需要使用 terraform 0.12+。

该样例中包含:

一个 variables.tf 文件，用于定义输入参数，代码详情如下：

```
variable "region" {
default = "cn-bj2"
```

```
}

variable "zone" {
default = "cn-bj2-05"
}

variable "instance_password" {
default = "ucloud_2020"
}

variable "instance_count" {
default = 2
}

variable "count_format" {
default = "%02d"
}
```

一个 main.tf 文件，用于建立一个从云资源到代码的映射，代码详情如下：

```
# 指定 UCloud Provider 和配置信息
provider "ucloud" {
region = var.region
}

# 查询默认可用区中的主机镜像
data "ucloud_images" "default" {
availability_zone = var.zone
name_regex = "^CentOS 7.[1-2] 64"
image_type = "base"
}

# 创建外网防火墙
resource "ucloud_security_group" "default" {
name = "tf-example-two-tier"
tag = "tf-example"
```

```
# HTTP access from LAN
rules {
port_range = "80"
protocol = "tcp"
cidr_block = "0.0.0.0/0"
policy = "accept"
}

# HTTPS access from LAN
rules {
port_range = "443"
protocol = "tcp"
cidr_block = "0.0.0.0/0"
policy = "accept"
}
}

# 创建 VPC
resource "ucloud_vpc" "default" {
name = "tf-example-two-tier"
tag = "tf-example"

# vpc network
cidr_blocks = ["192.168.0.0/16"]
}

# 创建 Subnet 到 VPC 下
resource "ucloud_subnet" "default" {
name = "tf-example-two-tier"
tag = "tf-example"

# subnet's network must be contained by vpc network
# and a subnet must have least 8 ip addresses in it (netmask < 30).
cidr_block = "192.168.1.0/24"

vpc_id = ucloud_vpc.default.id
}
```

```
# 创建多台 web 服务器
resource "ucloud_instance" "web" {
availability_zone = var.zone
instance_type = "n-basic-2"
image_id = data.ucloud_images.default.images[0].id
root_password = var.instance_password
boot_disk_type = "cloud_ssd"

name = "tf-example-two-tier-${format(var.count_format, count.index + 1)}"
tag = "tf-example"

# we will put all the instances into same vpc and subnet,
# so they can communicate with each other.
vpc_id = ucloud_vpc.default.id

subnet_id = ucloud_subnet.default.id

# this security group allows HTTP and HTTPS access
security_group = ucloud_security_group.default.id

count = var.instance_count
}

# 创建负载均衡
resource "ucloud_lb" "default" {
name = "tf-example-two-tier"
tag = "tf-example"

# we will put all the instances into same vpc and subnet,
# so they can communicate with each other.
vpc_id = ucloud_vpc.default.id

subnet_id = ucloud_subnet.default.id
}

# 创建基于 http 协议的负载均衡监听器
```

```
resource "ucloud_lb_listener" "default" {
load_balancer_id = ucloud_lb.default.id
protocol = "http"
port = 80
}

# 挂载云主机到负载均衡监听器
resource "ucloud_lb_attachment" "default" {
load_balancer_id = ucloud_lb.default.id
listener_id = ucloud_lb_listener.default.id
resource_id = ucloud_instance.web[count.index].id
port = 80
count = var.instance_count
}

# 创建负载均衡监听器转发规则
resource "ucloud_lb_rule" "default" {
load_balancer_id = ucloud_lb.default.id
listener_id = ucloud_lb_listener.default.id
backend_ids = ucloud_lb_attachment.default.*.id
domain = "www.ucloud.cn"
}

# 创建外网弹性 EIP
resource "ucloud_eip" "default" {
bandwidth = 2
charge_mode = "bandwidth"
name = "tf-example-two-tier"
tag = "tf-example"
internet_type = "bgp"
}

# 绑定 EIP 到负载均衡
resource "ucloud_eip_association" "default" {
resource_id = ucloud_lb.default.id
eip_id = ucloud_eip.default.id
}
```

生成执行计划

在当前目录下执行 terraform plan 命令，查看编排计划：

```
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.ucloud_zones.default: Refreshing state...
data.ucloud_images.default: 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:

+ ucloud_eip.default
id: <computed>
bandwidth: "2"
charge_mode: "bandwidth"
charge_type: "month"
create_time: <computed>
expire_time: <computed>
internet_type: "bgp"
ip_set.#: <computed>
name: "tf-example-two-tier"
public_ip: <computed>
remark: <computed>
resource.%: <computed>
status: <computed>
tag: "tf-example"

+ ucloud_eip_association.default
id: <computed>
```

```
eip_id: "${ucloud_eip.default.id}"
resource_id: "${ucloud_lb.default.id}"
resource_type: <computed>

+ ucloud_instance.web[0]
id: <computed>
auto_renew: <computed>
availability_zone: "cn-bj2-05"
boot_disk_size: <computed>
boot_disk_type: <computed>
charge_type: "month"
cpu: <computed>
create_time: <computed>
data_disk_size: <computed>
data_disk_type: <computed>
disk_set.#: <computed>
expire_time: <computed>
image_id: "uimage-f1chxn"
instance_type: "n-basic-2"
ip_set.#: <computed>
memory: <computed>
name: "tf-example-two-tier-01"
private_ip: <computed>
remark: <computed>
root_password: <sensitive>
security_group: "${ucloud_security_group.default.id}"
status: <computed>
subnet_id: "${ucloud_subnet.default.id}"
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_instance.web[1]
id: <computed>
auto_renew: <computed>
availability_zone: "cn-bj2-05"
boot_disk_size: <computed>
boot_disk_type: <computed>
```

```
charge_type: "month"
cpu: <computed>
create_time: <computed>
data_disk_size: <computed>
data_disk_type: <computed>
disk_set.#: <computed>
expire_time: <computed>
image_id: "uimage-f1chxn"
instance_type: "n-basic-2"
ip_set.#: <computed>
memory: <computed>
name: "tf-example-two-tier-02"
private_ip: <computed>
remark: <computed>
root_password: <sensitive>
security_group: "${ucloud_security_group.default.id}"
status: <computed>
subnet_id: "${ucloud_subnet.default.id}"
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_lb.default
id: <computed>
create_time: <computed>
expire_time: <computed>
internal: <computed>
ip_set.#: <computed>
name: "tf-example-two-tier"
private_ip: <computed>
remark: <computed>
subnet_id: "${ucloud_subnet.default.id}"
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_lb_attachment.default[0]
id: <computed>
listener_id: "${ucloud_lb_listener.default.id}"
```

```
load_balancer_id: "${ucloud_lb.default.id}"
port: "80"
private_ip: <computed>
resource_id: "${ucloud_instance.web.*.id[count.index]}"
resource_type: <computed>
status: <computed>

+ ucloud_lb_attachment.default[1]
id: <computed>
listener_id: "${ucloud_lb_listener.default.id}"
load_balancer_id: "${ucloud_lb.default.id}"
port: "80"
private_ip: <computed>
resource_id: "${ucloud_instance.web.*.id[count.index]}"
resource_type: <computed>
status: <computed>

+ ucloud_lb_listener.default
id: <computed>
domain: <computed>
health_check_type: <computed>
idle_timeout: <computed>
listen_type: <computed>
load_balancer_id: "${ucloud_lb.default.id}"
method: "roundrobin"
name: <computed>
path: <computed>
persistence: <computed>
persistence_type: "none"
port: "80"
protocol: "http"
status: <computed>

+ ucloud_lb_rule.default
id: <computed>
backend_ids.#: <computed>
domain: "www.ucloud.cn"
```

```
listener_id: "${ucloud_lb_listener.default.id}"
load_balancer_id: "${ucloud_lb.default.id}"

+ ucloud_security_group.default
id: <computed>
create_time: <computed>
name: "tf-example-two-tier"
remark: <computed>
rules.#: "2"
rules.2733123863.cidr_block: "0.0.0.0/0"
rules.2733123863.policy: "accept"
rules.2733123863.port_range: "443"
rules.2733123863.priority: "high"
rules.2733123863.protocol: "tcp"
rules.845396726.cidr_block: "0.0.0.0/0"
rules.845396726.policy: "accept"
rules.845396726.port_range: "80"
rules.845396726.priority: "high"
rules.845396726.protocol: "tcp"
tag: "tf-example"

+ ucloud_subnet.default
id: <computed>
cidr_block: "192.168.1.0/24"
create_time: <computed>
name: "tf-example-two-tier"
remark: <computed>
tag: "tf-example"
vpc_id: "${ucloud_vpc.default.id}"

+ ucloud_vpc.default
id: <computed>
cidr_blocks.#: "1"
cidr_blocks.3901788224: "192.168.0.0/16"
create_time: <computed>
name: "tf-example-two-tier"
network_info.#: <computed>
```

```
remark: <computed>
tag: "tf-example"
update_time: <computed>


Plan: 12 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.
```

可以看到即将创建两台云主机、一个外网弹性 EIP、一个外网防火墙、一个负载均衡、一个负载均衡监听器、一个负载均衡监听器规则、一个 VPC、一个 Subnet、两台主机分别与负载均衡监听器的挂载关系、EIP 与负载均衡的挂载关系。

执行编排

执行 terraform apply 命令并确认，执行编排计划：

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes
```

可通过 控制台 确认资源已创建完成。

参考文献

详见 Two-Tier Example

常见问题

创建 1000 台，失败 500 台如何处理?

由于 Terraform 的状态存储持久化了当前资源的状态，所以在自动化的环境中，比如 CI 执行环境下，对于偶发性质的问题，可以通过简单的重试策略来保障基础设施编排的可用性，Terraform 会继续创建失败的 500 台，大幅减少人工干预。

API 有更新如何处理?

Terraform 的状态管理中，实现了基于 Schema 版本的状态迁移机制，可以对于破坏性的变更，保持向前兼容。

Terraform 的异地容灾的架构是否是指在灾备地域迅速拉起另一套基础设施?

不完全是，Terraform 负责管理基础设施从创建到销毁的整个生命周期，对于容灾切换可以借助外部的方式来实现，有几种可行的方式:

- 使用 Terraform 创建两套异地的 UCloud 基础设施，通过全局负载均衡做容灾切换
- 当监控指标达到异常触发条件时，用程序调用 Terraform 迅速拉起另一套基础设施

集群的扩容是否支持? 会影响到现有的资源么?

通过简单的增加 count 的值即可水平扩容相同配置的实例。不会影响到现有的资源。

没有 provider 的支持怎么用?

可以使用命令行与 Terraform 配合使用，Terraform 可以使用 Provisioner 执行任意脚本，比如通过 local-exec/remote-exec 脚本，将标准输出导出到文件中，再通过文件加载进来，纳入 Terraform 的管理，依然享有 Terraform 基础设施即代码的优势。

部分机房不通外网，如何转换成 Terraform 接入?

UCloud Terraform 已经支持 BaseUrl，可以调用内网 UCloud API。

创建 100 台主机需要多少时间?

Terraform 天生支持并行资源编排,理论上和创建 1 台主机所需的时间是一个数量级,需要的时间取决于最晚创建成功的那台主机所需的时间; 目前 Terraform 默认的最大并发数是 10,也就是说创建 100 台云主机相当于创建 10 台的时间; 另外可以通过在执行命令 terraform apply -parallelism=n 来设置最大并发数 n。

API GW 是否可以抗住 Terraform 高并发的调用场景?

目前 Terraform 默认的最大并发数是 10,不会由单个用户同时发起过多的并发连接,所以降低了业务间锁竞争的风险,可以支持更多资源同时编排。

是否所有的可用区都可以使用 Terraform?

由于 Terraform 基于 UCloud API。对于大部分 IaaS 产品,只要 API 支持,我们就支持。

用户如何升级版本及知晓升级? 用户可不可以不升级?

可以通过 Terraform 插件仓库,拿到 UCloud Provider 的最新版本,这个操作是由 terraform init 命令完成的。Terraform 支持在配置文件中使用版本锁定语法,指定一个具体的版本,也可以指定一个版本范围。

为了避免因误操作造成服务宕机，如何配置定义文件，保证特定资源不会被误删除?

可以通过配置特定资源的 lifecycle,设置 prevent_destroy = true 来指定特定资源禁止删除。