

ALIBABA CLOUD

阿里云

Terraform  
Terraform

文档版本：20220713

 阿里云

## 法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

# 通用约定

格式	说明	样例
 危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 危险 重置操作将丢失用户配置数据。
 警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 警告 重启操作将导致业务中断，恢复业务时间约十分钟。
 注意	用于警示信息、补充说明等，是用户必须了解的内容。	 注意 权重设置为0，该服务器不会再接受新请求。
 说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击设置> 网络> 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击确定。
Courier字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <code>Instance_ID</code>
[ ] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

# 目录

1. 产品介绍	06
1.1. 什么是Terraform	06
1.2. 应用场景	06
2. 安装与配置	08
2.1. 在Cloud Shell中使用Terraform	08
2.2. 在本地安装和配置Terraform	10
3. 资源类型	12
3.1. 资源类型索引	12
4. 教程	48
4.1. 云服务器ECS	48
4.1.1. 创建一台ECS实例	48
4.1.2. 创建多台ECS实例	50
4.2. 负载均衡SLB	54
4.2.1. 通过Terraform管理负载均衡服务	54
4.2.2. 通过Terraform在专有网络中创建负载均衡实例	56
4.3. 云数据库RDS版	57
4.3.1. 创建一个云数据库实例	57
4.4. 专有网络VPC	59
4.4.1. 通过Terraform创建基于阿里云的自定义私有网络	59
4.5. 对象存储OSS	62
4.5.1. 通过Terraform管理Bucket	62
4.5.2. 五分钟入门阿里云Terraform OSS Backend	63
4.6. 访问控制RAM	67
4.6.1. 使用Terraform创建一个RAM用户	67
4.6.2. 使用Terraform创建角色并绑定自定义权限策略	69
4.6.3. 一键创建容器镜像仓库和授权RAM账号	72

---

4.7. 容器服务Kubernetes版 .....	75
4.7.1. 使用Terraform创建托管版Kubernetes .....	75
4.7.2. Terraform部署容器服务Kubernetes集群及WordPress应用 .....	82
4.8. 应用部署 .....	96
4.8.1. 部署Web集群 .....	96
4.8.2. 一键创建分布式集群并部署文件 .....	98
4.8.3. 使用Terraform一键部署OpenShift .....	104
5.参考 .....	108
5.1. Terraform常用命令 .....	108
5.2. 如何解决存量云资源的管理难题 .....	112
5.3. Alicloud Provider .....	115
5.4. Alibaba Cloud Module Registry .....	115
5.5. 如何贡献代码? .....	115
5.6. 支持的阿里云产品 .....	116
6.客户案例 .....	117
6.1. 小马智行基于Terraform的IaC实践 .....	117
6.2. 流利说基于Terraform的自动化实践 .....	126

# 1. 产品介绍

## 1.1. 什么是Terraform

Terraform是一种开源工具，用于安全高效地预览、配置和管理云基础架构和资源。

### 概览

HashiCorp Terraform 是一个IT基础架构自动化编排工具，可以用代码来管理维护 IT 资源。Terraform的命令行接口（CLI）提供一种简单机制，用于将配置文件部署到阿里云或其他任意支持的云上，并对其进行版本控制。它编写了描述云资源拓扑的配置文件中的基础结构，例如虚拟机、存储帐户和网络接口。

Terraform是一个高度可扩展的工具，通过 Provider 来支持新的基础架构。Terraform能够让您在阿里云上轻松使用 **简单模板语言** 来定义、预览和部署云基础结构。您可以使用Terraform来创建、修改、删除ECS、VPC、RDS、SLB等多种资源。

阿里云作为国内第一家与 Terraform 集成的云厂商，**terraform-provider-alicloud**目前已经提供了超过 163 个 Resource 和 113 个 Data Source，覆盖计算，存储，网络，负载均衡，CDN，容器服务，中间件，访问控制，数据库等超过35款产品，已经满足了大量大客户的自动化上云需求。

从 Terraform 0.12.2 版本开始，阿里云支持将对象存储服务 OSS 作为标准的**Remote State Backend**，开始提供远端存储 State 的能力，在提高 state 安全性的同时，提升多人协作效率。

为了给开发者提供“开箱即用”的使用体验，阿里云提供了丰富多样的**Modules**和**Examples**，覆盖计算，存储，网络，中间件，数据库等多个产品和使用场景，欢迎大家使用和贡献自己的Module。

### 优势

- 将基础结构部署到多个云

Terraform适用于多云方案，将类似的基础结构部署到阿里云、其他云提供商或者本地数据中心。开发人员能够使用相同的工具和相似的配置文件同时管理不同云提供商的资源。

- 自动化管理基础结构

Terraform能够创建配置文件的模板，以可重复、可预测的方式定义、预配和配置ECS资源，减少因人为因素导致的部署和管理错误。能够多次部署同一模板，创建相同的开发、测试和生产环境。

- 基础架构即代码（Infrastructure as Code）

可以用代码来管理维护资源。允许保存基础设施状态，从而使您能够跟踪对系统（基础设施即代码）中不同组件所做的更改，并与其他人共享这些配置。

- 降低开发成本

您通过按需创建开发和部署环境来降低成本。并且，您可以在系统更改之前进行评估。

## 1.2. 应用场景

Terraform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。

### 创建基础设施

您可以使用Terraform创建和管理ECS、VPC和SLB等基础资源。

创建多台ECS并挂载数据盘，请参见示例模板**ecs-instance**。

### 均衡负载业务流量

您可以将访问流量按照定义的转发规则分发到指定的后端服务器（ECS实例），提高应用系统对外的服务能力，消除单点故障。

部署负载均衡服务，请参见示例模板[ecs-slb](#)。

## 自动伸缩

根据您的业务需求和策略自动调整弹性计算资源，在业务需求增长时无缝增加ECS实例满足计算需要，在业务需求下降时自动减少ECS实例节约成本。

一键创建伸缩组，伸缩配置和伸缩规则相关操作，请参见示例模板[autoscaling](#)和[autoscaling-rule](#)。

## 集群管理

您可以使用Terraform快速创建专有网络的集群。

在阿里云中启动kubernetes集群，并且在集群中创建VPC、vSwitch和NAT网关等资源，请参见示例模板[kubernetes module](#)。

## 配置函数计算服务

阿里云函数计算是事件驱动的全托管计算服务。通过函数计算，您无需管理服务器等基础设施，只需编写代码并上传。借助于函数计算，您可以快速构建任何类型的应用和服务，无需管理和运维。

一键搭建函数服务运行环境，快速配置对象存储、内容分发网络、消息队列、HTTP服务、日志服务等多种触发器，请参见示例模板：[fc module](#)。

## 2. 安装与配置

### 2.1. 在Cloud Shell中使用Terraform

阿里云Cloud Shell是一款帮助您运维的免费产品，预装了Terraform的组件，并配置好身份凭证（credentials）。因此您可直接在Cloud Shell中运行Terraform的命令。

完成以下操作，在Cloud Shell中使用Terraform：

1. 打开浏览器，访问Cloud Shell的地址<https://shell.aliyun.com>。

更多Cloud Shell入口及使用请参见[使用云命令行](#)。

2. 登录Cloud Shell。
3. 编写Terraform模板。

您可以使用vim命令直接编写模板，如果开通了OSS存储，您可以直接将配置模板上传到为Cloud Shell创建的bucket中。

示例：



```
provider "alicloud" {}

resource "alicloud_vpc" "vpc" {
  name      = "tf_test_foo"
  cidr_block = "172.16.0.0/12"
}

resource "alicloud_vswitch" "vsw" {
  vpc_id          = alicloud_vpc.vpc.id
  cidr_block      = "172.16.0.0/21"
  availability_zone = "cn-hangzhou-b"
}

resource "alicloud_security_group" "default" {
  name = "default"
  vpc_id = alicloud_vpc.vpc.id
}

resource "alicloud_instance" "instance" {
  # 可用区
  availability_zone = "cn-hangzhou-b"
  # 绑定安全组
  security_groups = alicloud_security_group.default.*.id
  # 实例规格
  instance_type      = "ecs.n2.small"
  # 系统盘类型
  system_disk_category = "cloud_efficiency"
  # 系统镜像
  image_id            = "ubuntu_140405_64_40G_cloudinit_20161115.vhd"
  # 实例名称
  instance_name       = "test_foo"
  # 所在交换机
  vswitch_id          = alicloud_vswitch.vsw.id
  # 公网带宽
  internet_max_bandwidth_out = 10
}

resource "alicloud_security_group_rule" "allow_all_tcp" {
  type          = "ingress"
  ip_protocol   = "tcp"
  nic_type      = "intranet"
  policy        = "accept"
  port_range    = "1/65535"
  priority      = 1
  security_group_id = alicloud_security_group.default.id
  cidr_ip       = "0.0.0.0/0"
}
```

4. 执行`terraform init`命令初始化配置。

5. 执行`terraform plan`命令预览配置。

```
shell@alicloud:~$ 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:

+ alicloud_instance.master[0]
  id: <computed>
  availability_zone: "cn-beijing-f"
  host_name: "sample"
  image_id: "centos_7_04_64_20G_alibase_201701015.vhd"
  instance_charge_type: "PostPaid"
  instance_name: "instance_sample"
  instance_type: "ecs.t5-lc1m1.small"
  internet_charge_type: "PayByTraffic"
  internet_max_bandwidth_in: <computed>
  internet_max_bandwidth_out: "1"
  key_name: <computed>
  password: <sensitive>
  private_ip: <computed>
  public_ip: <computed>
  role_name: <computed>
  security_groups.#: <computed>
  spot_strategy: "NoSpot"
  status: <computed>
  subnet_id: <computed>
  system_disk_category: "cloud_efficiency"
  system_disk_size: "40"
  vswitch_id: "${alicloud_vswitch.vsw.id}"

+ alicloud_instance.master[1]
  id: <computed>
  availability_zone: "cn-beijing-f"
  host_name: "sample"
  image_id: "centos_7_04_64_20G_alibase_201701015.vhd"
  instance_charge_type: "PostPaid"
  instance_name: "instance_sample"
  instance_type: "ecs.t5-lc1m1.small"
  internet_charge_type: "PayByTraffic"
  internet_max_bandwidth_in: <computed>
```

6. 执行terraform apply创建ECS实例。

## 2.2. 在本地安装和配置Terraform

在使用Terraform的简单模板语言定义、预览和部署云基础结构前，您需要安装预配置Terraform。

### 操作步骤

1. 登录 [Terraform官网](#) 下载适用于您的操作系统的程序包。
2. 将程序包解压到 `/usr/local/bin`。

如果将可执行文件解压到其他目录，按照以下方法为其定义全局路径：

- o Linux：参见 [在Linux系统定义全局路径](#)。

- Windows: 参见 [在Windows系统定义全局路径](#)。
- Mac: 参见 [在Mac系统定义全局路径](#)。

### 3. 运行 `terraform` 验证路径配置。

命令运行后将显示可用的Terraform选项的列表，如下所示，表示安装完成。

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

### 4. 为提高权限管理的灵活性和安全性，建议您创建RAM用户，并为其授权。

- i. 登录 [RAM控制台](#)。
- ii. 创建名为 *Terraform* 的RAM用户，并为该用户创建AccessKey。具体步骤请参见[创建RAM用户](#)。
- iii. 为RAM用户授权。具体步骤请参见[为RAM用户授权](#)。

### 5. 创建环境变量，用于存放身份认证信息。

```
export ALICLOUD_ACCESS_KEY="LTAIUrZCw3*****"
export ALICLOUD_SECRET_KEY="zfwwWAMWIAiooj14GQ2*****"
export ALICLOUD_REGION="cn-beijing"
```

## 相关文档

- [简单模板语言](#)

# 3.资源类型

- 资源类型索引

## 3.1. 资源类型索引

本文为您提供Terraform所支持的资源类型索引，便于您进行查询。

类别	阿里云产品	资源类型
	云服务器ECS	<ul style="list-style-type: none"><li><code>alicloud_auto_provisioning_group</code>: 弹性供应组</li><li><code>alicloud_ecs_auto_snapshot_policy</code>: 自动快照策略</li><li><code>alicloud_ecs_auto_snapshot_policy_attachment</code>: 为云盘应用自动快照策略</li><li><code>alicloud_ecs_command</code>: 云助手命令</li><li><code>alicloud_ecs_dedicated_host</code>: 专有宿主机</li><li><code>alicloud_ecs_dedicated_host_cluster</code>: 专有宿主机集群</li><li><code>alicloud_ecs_deployment_set</code>: 部署集</li><li><code>alicloud_ecs_disk</code>: 云盘</li><li><code>alicloud_ecs_disk_attachment</code>: 挂载云盘到ECS实例</li><li><code>alicloud_ecs_hpc_cluster</code>: 高性能集群</li><li><code>alicloud_ecs_key_pair</code>: SSH 密钥对</li><li><code>alicloud_ecs_key_pair_attachment</code>: 绑定SSH密钥对到ECS实例</li><li><code>alicloud_ecs_launch_template</code>: 启动模板</li><li><code>alicloud_ecs_network_interface</code>: 弹性网卡</li><li><code>alicloud_ecs_network_interface_attachment</code>: 绑定弹性网卡到ECS实例</li><li><code>alicloud_ecs_snapshot</code>: 快照</li><li><code>alicloud_image</code>: 创建镜像</li></ul>

	<ul style="list-style-type: none"><li>alicloud_image_copy: 复制镜像</li><li>alicloud_image_export: 导出镜像</li><li>alicloud_image_import: 导入镜像</li><li>alicloud_image_share_permission: 共享镜像</li><li>alicloud_instance: ECS实例</li><li>alicloud_reserved_instance: 预留实例</li><li>alicloud_security_group: 安全组</li><li>alicloud_security_group_rule: 安全组规则</li><li>alicloud_ecs_session_manager_status: 会话管理</li><li>alicloud_ecs_prefix_list: 前缀列表</li><li>alicloud_ecs_storage_capacity_unit: 存储容量资源包</li><li>alicloud_ecs_image_component: 镜像组件</li><li>alicloud_ecs_snapshot_group: 快照组</li><li>alicloud_ecs_image_pipeline: 镜像模板</li><li>alicloud_ecs_network_interface_permission: 授权弹性网卡权限</li><li>alicloud_ecs_invocation: 触发一条云助手命令</li><li>alicloud_ecs_instance_set: 批量创建一组配置相同的实例资源</li></ul>
轻量应用服务器	<ul style="list-style-type: none"><li>alicloud_simple_application_server_instance: 实例</li><li>alicloud_simple_application_server_custom_image: 自定义镜像</li><li>alicloud_simple_application_server_firewall_rule: 防火墙规则</li><li>alicloud_simple_application_server_snapshot: 为指定的磁盘创建快照</li></ul>

弹性计算	弹性容器实例	<ul style="list-style-type: none"> <li>• <a href="#">alicloud_eci_container_group</a>: 容器组</li> <li>• <a href="#">alicloud_eci_image_cache</a>: 镜像缓存</li> <li>• <a href="#">alicloud_eci_virtual_node</a>: 虚拟节点</li> </ul>
	无影云桌面	<ul style="list-style-type: none"> <li>• <a href="#">alicloud_ecd_nas_file_system</a>: NAS文件系统</li> <li>• <a href="#">alicloud_ecd_network_package</a>: 互联网访问包</li> <li>• <a href="#">alicloud_ecd_policy_group</a>: 策略</li> <li>• <a href="#">alicloud_ecd_simple_office_site</a>: 工作区</li> <li>• <a href="#">alicloud_ecd_user</a>: 便捷用户</li> <li>• <a href="#">alicloud_ecd_desktop</a>: 云桌面</li> <li>• <a href="#">alicloud_ecd_image</a>: 自定义镜像</li> <li>• <a href="#">alicloud_ecd_command</a>: 在一台或多台云桌面中执行一段PowerShell或者Bat类型的脚本</li> <li>• <a href="#">alicloud_ecd_snapshot</a>: 为云桌面的磁盘创建快照</li> <li>• <a href="#">alicloud_ecd_bundle</a>: 创建自定义桌面模板</li> <li>• <a href="#">ecd_ram_directory</a>: RAM类型的目录</li> <li>• <a href="#">ecd_ad_connector_directory</a>: AD类型的目录</li> <li>• <a href="#">alicloud_ecd_custom_property</a>: 用户自定义属性</li> <li>• <a href="#">alicloud_ecd_ad_connector_of_fice_site</a>: AD工作区</li> </ul>
	弹性高性能计算E-HPC	<ul style="list-style-type: none"> <li>• <a href="#">alicloud_ehpc_job_template</a>: 作业模板</li> <li>• <a href="#">alicloud_ehpc_cluster</a>: 集群</li> </ul>

	弹性伸缩	<ul style="list-style-type: none"><li>• <code>alicloud_ess_alarm</code>: 报警任务</li><li>• <code>alicloud_ess_attachment</code>: 将一台或多台ECS实例关联到伸缩组中</li><li>• <code>alicloud_ess_notification</code>: 事件通知</li><li>• <code>alicloud_ess_scaling_configuration</code>: 伸缩配置</li><li>• <code>alicloud_ess_scaling_group</code>: 伸缩组</li><li>• <code>alicloud_ess_scaling_lifecycle_hook</code>: 生命周期挂钩</li><li>• <code>alicloud_ess_scaling_rule</code>: 伸缩规则</li><li>• <code>alicloud_ess_scalinggroup_vserver_groups</code>: 向伸缩组添加SLB虚拟服务器组</li><li>• <code>alicloud_ess_alb_server_group_attachment</code>: 向伸缩组添加ALB虚拟服务器组</li><li>• <code>alicloud_ess_scheduled_task</code>: 定时任务</li><li>• <code>alicloud_ess_eci_scaling_configuration</code>: ECI伸缩配置</li></ul>
	资源编排	<ul style="list-style-type: none"><li>• <code>alicloud_ros_change_set</code>: 更改集</li><li>• <code>alicloud_ros_stack</code>: 资源栈</li><li>• <code>alicloud_ros_stack_group</code>: 资源栈组</li><li>• <code>alicloud_ros_stack_instance</code>: 资源栈实例</li><li>• <code>alicloud_ros_template</code>: 自定义模板</li><li>• <code>alicloud_ros_template_scratch</code>: 资源场景</li></ul>

	运维编排服务	<ul style="list-style-type: none"><li>alicloud_oos_execution: 执行</li><li>alicloud_oos_template: 模板</li><li>alicloud_oos_application: 应用</li><li>alicloud_oos_application_group: 应用组</li><li>alicloud_oos_patch_baseline: 补丁基线</li><li>alicloud_oos_service_setting: 配置模板执行记录的投递</li><li>alicloud_oos_parameter: 普通参数</li><li>alicloud_oos_secret_parameter: 加密参数</li><li>alicloud_oos_state_configuration: 终态配置</li></ul>
	弹性加速计算实例EAIS	<ul style="list-style-type: none"><li>alicloud_eais_instance: 实例</li></ul>
	函数计算	<ul style="list-style-type: none"><li>alicloud_fc_alias: 别名</li><li>alicloud_fc_custom_domain: 自定义域名</li><li>alicloud_fc_function: 函数</li><li>alicloud_fc_function_async_invoke_config: 函数异步调用</li><li>alicloud_fc_service: 服务</li><li>alicloud_fc_trigger: 触发器</li></ul>



	Serverless应用引擎	<ul style="list-style-type: none"> <li>• <code>alicloud_sae_application</code>: 应用生命周期</li> <li>• <code>alicloud_sae_config_map</code>: 配置项管理</li> <li>• <code>alicloud_sae_ingress</code>: 路由规则</li> <li>• <code>alicloud_sae_namespace</code>: 命名空间</li> <li>• <code>alicloud_sae_application_scaling_rule</code>: 伸缩规则</li> <li>• <code>alicloud_sae_grey_tag_route</code>: 灰度规则</li> <li>• <code>alicloud_sae_load_balancer_internet</code>: 为应用绑定公网SLB</li> <li>• <code>alicloud_sae_load_balancer_internet</code>: 为应用绑定私网SLB</li> </ul>
	Serverless工作流	<ul style="list-style-type: none"> <li>• <code>alicloud_fnf_flow</code>: 流程</li> <li>• <code>alicloud_fnf_schedule</code>: 定时调度</li> <li>• <code>alicloud_fnf_execution</code>: 执行流程</li> </ul>
	弹性云手机	<ul style="list-style-type: none"> <li>• <code>alicloud_ecp_key_pair</code>: SSH 密钥对</li> <li>• <code>alicloud_ecp_instance</code>: 实例</li> </ul>
	对象存储 OSS	<ul style="list-style-type: none"> <li>• <code>alicloud_oss_bucket</code>: 存储空间</li> <li>• <code>alicloud_oss_bucket_object</code>: 存储对象</li> </ul>

	日志服务	<ul style="list-style-type: none"><li>• <code>alicloud_log_alert</code>: 告警</li><li>• <code>alicloud_log_audit</code>: 审计</li><li>• <code>alicloud_log_dashboard</code>: 仪表盘</li><li>• <code>alicloud_log_etl</code>: 数据加工</li><li>• <code>alicloud_log_machine_group</code>: 机器组</li><li>• <code>alicloud_log_oss_shipper</code>: 数据投递</li><li>• <code>alicloud_log_project</code>: 日志项目</li><li>• <code>alicloud_log_store</code>: 日志库</li><li>• <code>alicloud_log_store_index</code>: 日志库索引</li><li>• <code>alicloud_logtail_attachment</code>: 将日志库配置应用到机器组</li><li>• <code>alicloud_log_ingestion</code>: 日志导入</li><li>• <code>alicloud_logtail_config</code>: 日志库配置</li><li>• <code>alicloud_log_resource</code>: 资源数据</li><li>• <code>alicloud_log_resource_record</code>: 告警策略</li></ul>
--	------	---

存储	文件存储 NAS	<ul style="list-style-type: none"><li>alicloud_nas_access_group: 权限组</li><li>alicloud_nas_access_rule: 权限组规则</li><li>alicloud_nas_file_system: 文件系统</li><li>alicloud_nas_mount_target: 挂载点</li><li>alicloud_nas_snapshot: 为极速型NAS创建快照</li><li>alicloud_nas_fileset: 为CPFS文件系统创建文件集</li><li>alicloud_nas_auto_snapshot_policy: 自动快照策略</li><li>alicloud_nas_lifecycle_policy: 生命周期管理策略</li><li>alicloud_nas_data_flow: 数据流动</li><li>alicloud_nas_recycle_bin: 开启文件系统的回收站功能</li></ul>
	文件存储 HDFS	<ul style="list-style-type: none"><li>alicloud_dfs_access_group: 权限组</li><li>alicloud_dfs_access_rule: 权限组规则</li><li>alicloud_dfs_file_system: 文件系统</li><li>alicloud_dfs_mount_point: 挂载点</li></ul>
	数据库文件存储	<ul style="list-style-type: none"><li>alicloud_dbfs_instance: 实例</li><li>alicloud_dbfs_snapshot: 快照</li><li>alicloud_dbfs_instance_attachment: 关联ECS实例</li><li>alicloud_dbfs_service_linked_role: 为DBFS创建服务角色</li></ul>

	表格存储 Tablestore	<ul style="list-style-type: none"><li>alicloud_ots_instance: OTS 实例</li><li>alicloud_ots_instance_attachm ent: 绑定VPC到OTS实例</li><li>alicloud_ots_table: OTS 表格</li><li>alicloud_ots_tunnel: 通道服务</li></ul>
	智能媒体管理	<ul style="list-style-type: none"><li>alicloud_imm_project: 项目</li></ul>
	混合云备份 HBR	<ul style="list-style-type: none"><li>alicloud_hbr_ecs_backup_client : ECS 备份客户端</li><li>alicloud_hbr_ecs_backup_plan : ECS 备份计划</li><li>alicloud_hbr_nas_backup_plan : NAS 备份计划</li><li>alicloud_hbr_oss_backup_plan : OSS 备份计划</li><li>alicloud_hbr_restore_job: 恢复任务</li><li>alicloud_hbr_server_backup_pl an: ECS整机备份</li><li>alicloud_hbr_vault: 仓库</li><li>alicloud_hbr_replication_vault : 备份仓库</li><li>alicloud_hbr_ots_backup_plan : OTS 备份计划</li></ul>

	云存储网关	<ul style="list-style-type: none"> <li>alicloud_cloud_storage_gateway_gateway: 网关</li> <li>alicloud_cloud_storage_gateway_gateway_smb_user: SMB用户</li> <li>alicloud_cloud_storage_gateway_storage_bundle: 网关集群</li> <li>alicloud_cloud_storage_gateway_express_sync: 极速同步组</li> <li>alicloud_cloud_storage_gateway_express_sync_share_attachment: 将网关文件共享放置到极速同步组中</li> <li>alicloud_cloud_storage_gateway_gateway_block_volume: 网关块卷</li> <li>alicloud_cloud_storage_gateway_gateway_file_share: 网关文件共享</li> <li>alicloud_cloud_storage_gateway_gateway_logging: 网关日志</li> <li>alicloud_cloud_storage_gateway_gateway_cache_disk: 网关缓存盘</li> </ul>
	云原生关系型数据库 PolarDB	<ul style="list-style-type: none"> <li>alicloud_polardb_account: 数据库账号</li> <li>alicloud_polardb_account_privilege: 给数据库账号授权</li> <li>alicloud_polardb_backup_policy: 数据库备份策略</li> <li>alicloud_polardb_cluster: 集群</li> <li>alicloud_polardb_database: 数据库</li> <li>alicloud_polardb_endpoint: 自定义集群地址</li> <li>alicloud_polardb_endpoint_address: 集群的公网或经典网络连接地址</li> </ul>
	云原生分布式数据库 PolarDB-X 1.0	<ul style="list-style-type: none"> <li>alicloud_drds_instance: 实例</li> </ul>

	云数据库 RDS	<ul style="list-style-type: none"><li>alicloud_db_account_privilege: 给数据库账号授权</li><li>alicloud_db_backup_policy: 数据库备份策略</li><li>alicloud_db_connection: 数据库实例公网连接地址</li><li>alicloud_db_database: RDS 数据库</li><li>alicloud_db_instance: RDS 实例</li><li>alicloud_db_read_write_splitting_connection: 申请只读地址</li><li>alicloud_db_readonly_instance: 只读实例</li><li>alicloud_rds_account: 数据库账号</li><li>alicloud_rds_parameter_group: 参数模板</li><li>alicloud_rds_backup: 为实例创建备份</li><li>alicloud_rds_clone_db_instance: 克隆实例</li><li>alicloud_rds_upgrade_db_instance: 为RDS PostgreSQL实例升级大版本</li></ul>
	云数据库专属集群 MyBase	<ul style="list-style-type: none"><li>alicloud_cddc_dedicated_host_group: 专属集群</li><li>alicloud_cddc_dedicated_host: 专属主机</li><li>alicloud_cddc_dedicated_host_account: 专属主机账户</li></ul>
	云原生多模数据库 Lindorm	<ul style="list-style-type: none"><li>alicloud_lindorm_instance: Lindorm实例</li></ul>

## 数据库

云数据库 Redis/Memcache	<ul style="list-style-type: none"> <li>alicloud_kvstore_account: 账号</li> <li>alicloud_kvstore_audit_log_config: 设置Redis/Memcache实例的审计日志开关与保留时长</li> <li>alicloud_kvstore_backup_policy: 备份策略</li> <li>alicloud_kvstore_connection: 公网连接地址</li> <li>alicloud_kvstore_instance: Redis/Memcache 实例</li> </ul>
云数据库 MongoDB	<ul style="list-style-type: none"> <li>alicloud_mongodb_instance: 副本集实例</li> <li>alicloud_mongodb_sharding_instance: 分片集群实例</li> <li>alicloud_mongodb_audit_policy: 配置副本集实例和分片集群实例的审计日志</li> <li>alicloud_mongodb_account: 实例账户</li> <li>alicloud_mongodb_serverless_instance: Serverless版实例</li> <li>alicloud_mongodb_sharding_network_public_address: 为分片集群实例申请公网连接地址</li> <li>alicloud_mongodb_sharding_network_private_address: 为分片集群实例申请私网连接地址</li> </ul>
云数据库 HBase	<ul style="list-style-type: none"> <li>alicloud_hbase_instance: HBase实例</li> </ul>
云数据库 Cassandra	<ul style="list-style-type: none"> <li>alicloud_cassandra_backup_plan: 备份计划</li> <li>alicloud_cassandra_cluster: Cassandra实例</li> <li>alicloud_cassandra_data_center: 集群数据中心</li> </ul>
时序数据库 TSDB	<ul style="list-style-type: none"> <li>alicloud_tsdb_instance: TSDB实例</li> </ul>

图数据库 GDB	<ul style="list-style-type: none"> <li>alicloud_graph_database_db_instance: GDB 实例</li> </ul>
云原生数仓 AnalyticDB MySQL	<ul style="list-style-type: none"> <li>alicloud_adb_account: 账号</li> <li>alicloud_adb_backup_policy: 备份策略</li> <li>alicloud_adb_connection: 公网连接地址</li> <li>alicloud_adb_db_cluster: AnalyticDB MySQL版集群</li> </ul>
云原生数仓 AnalyticDB PostgreSQL	<ul style="list-style-type: none"> <li>alicloud_gpdb_account: 实例的初始账号</li> <li>alicloud_gpdb_connection: 公网连接地址</li> <li>alicloud_gpdb_elastic_instance: 存储弹性模式的AnalyticDB PostgreSQL版实例</li> <li>alicloud_gpdb_instance: AnalyticDB for PostgreSQL实例</li> </ul>
云数据库 ClickHouse	<ul style="list-style-type: none"> <li>alicloud_click_house_account: 账号</li> <li>alicloud_click_house_db_cluster: ClickHouse集群</li> <li>alicloud_click_house_backup_policy: 备份设置</li> </ul>
数据传输服务 DTS	<ul style="list-style-type: none"> <li>alicloud_dts_job_monitor_rule: 任务告警规则</li> <li>alicloud_dts_subscription_job: 订阅任务</li> <li>alicloud_dts_synchronization_instance: 异步任务实例</li> <li>alicloud_dts_synchronization_job: 异步任务</li> <li>alicloud_dts_consumer_channel: 订阅任务的消费组</li> <li>alicloud_dts_migration_job: 迁移任务</li> <li>alicloud_dts_migration_instance: 迁移实例</li> </ul>



	数据管理 DMS	<ul style="list-style-type: none"> <li>alicloud_dms_enterprise_instance: 企业数据库实例</li> <li>alicloud_dms_enterprise_user: 企业用户</li> </ul>
	数据库网关 DG	<ul style="list-style-type: none"> <li>alicloud_database_gateway_gateway: 网关</li> </ul>
	DDoS基础防护	<ul style="list-style-type: none"> <li>alicloud_ddos_basic_defense_threshold: 修改单个公网IP资产（云服务器ECS实例、负载均衡SLB实例、弹性公网IP实例）的DDoS防护清洗阈值</li> </ul>
	DDoS原生防护	<ul style="list-style-type: none"> <li>alicloud_ddosbgp_instance: DDoS高防实例</li> </ul>
	DDoS高防新BGP&国际	<ul style="list-style-type: none"> <li>alicloud_ddoscoo_domain_resource: 网站业务转发规则的域名</li> <li>alicloud_ddoscoo_instance: DDoS高防（新BGP）实例</li> <li>alicloud_ddoscoo_port: 端口</li> <li>alicloud_ddoscoo_scheduler_rule: 流量调度器调度规则</li> </ul>
	Web 应用防火墙	<ul style="list-style-type: none"> <li>alicloud_waf_certificate: 证书</li> <li>alicloud_waf_domain: 域名</li> <li>alicloud_waf_instance: 实例</li> <li>alicloud_waf_protection_module: 修改指定WAF防护功能模块（包括正则防护引擎、大数据深度学习引擎、CC安全防护、数据风控、主动防御等模块）中的防护模式</li> </ul>
	SSL证书服务	<ul style="list-style-type: none"> <li>alicloud_ssl_certificates_service_certificate: 证书</li> </ul>
	云安全中心	<ul style="list-style-type: none"> <li>alicloud_security_center_group: 服务器分组</li> <li>alicloud_security_center_service_linked_role: 为云安全中心服务创建服务角色</li> </ul>

	云防火墙	<ul style="list-style-type: none"><li>alicloud_cloud_firewall_control_policy: 访问控制策略</li><li>alicloud_cloud_firewall_control_policy_order: : 调整访问控制策略优先级</li><li>alicloud_cloud_firewall_instance: 实例</li></ul>

安全	堡垒机	<ul style="list-style-type: none"><li>alicloud_bastionhost_host: 主机</li><li>alicloud_bastionhost_host_account: 主机账号</li><li>alicloud_bastionhost_host_account_user_attachment: 为用户授权主机和主机账户</li><li>alicloud_bastionhost_host_account_user_group_attachment: 为用户组授权主机及主机账户</li><li>alicloud_bastionhost_host_attachment: 将主机加入指定主机组</li><li>alicloud_bastionhost_host_group: 主机组</li><li>alicloud_bastionhost_host_group_account_user_attachment: 为用户授权主机组和主机账号</li><li>alicloud_bastionhost_host_group_account_user_group_attachment: 为用户组授权主机组和主机账号</li><li>alicloud_bastionhost_instance: 堡垒机实例</li><li>alicloud_bastionhost_user: 用户</li><li>alicloud_bastionhost_user_attachment: 为用户组添加用户</li><li>alicloud_bastionhost_user_group: 用户组</li><li>alicloud_bastionhost_host_share_key: 主机共享密钥</li><li>alicloud_bastionhost_host_account_share_key_attachment: 为主机账户添加共享密钥</li></ul>
	操作审计	<ul style="list-style-type: none"><li>alicloud_actiontrail_history_delivery_job: 历史事件投递任务</li><li>alicloud_actiontrail_trail: 跟踪</li></ul>

		<ul style="list-style-type: none"><li>alicloud_ram_access_key: AccessKey</li><li>alicloud_ram_account_alias: 云账号别名</li><li>alicloud_ram_account_password_policy: RAM用户密码策略</li><li>alicloud_ram_group: 用户组</li><li>alicloud_ram_group_memberships: 将RAM用户添加到指定的用户组</li><li>alicloud_ram_group_policy_attachment: 为指定用户组添加权限</li><li>alicloud_ram_login_profile: 为一个RAM用户启用Web控制台登录</li><li>alicloud_ram_policy: 访问策略</li><li>alicloud_ram_role: 访问角色</li><li>alicloud_ram_role_attachment: 为ECS关联RAM角色</li><li>alicloud_ram_role_policy_attachment: 为指定角色添加权限</li><li>alicloud_ram_saml_provider: 角色SSO身份提供商</li><li>alicloud_ram_user: 用户</li><li>alicloud_ram_user_policy_attachment: 为指定用户添加权限</li><li>alicloud_ram_security_preference: 设置RAM用户的全局安全首选项</li></ul>
	访问控制	
	数据库审计	<ul style="list-style-type: none"><li>alicloud_yundun_dbaudit_instance: 实例</li></ul>
	数据安全中心	<ul style="list-style-type: none"><li>alicloud_sddp_config: 异常告警通用配置模块的参数</li><li>alicloud_sddp_instance: 实例</li><li>alicloud_sddp_rule: 自定义的敏感数据识别规则</li><li>alicloud_sddp_data_limit: 为资源授权</li></ul>

	密钥管理服务	<ul style="list-style-type: none"> <li>alicloud_kms_alias: 为主密钥 (CMK) 创建一个别名</li> <li>alicloud_kms_ciphertext: 生成一个密文</li> <li>alicloud_kms_key: 主密钥</li> <li>alicloud_kms_key_version: 主密钥版本</li> <li>alicloud_kms_secret: 凭据</li> </ul>
	金融级实人认证	<ul style="list-style-type: none"> <li>alicloud_cloudface_auth_face_config: 人脸配置</li> </ul>
大数据	MaxCompute	<ul style="list-style-type: none"> <li>alicloud_maxcompute_project: 项目</li> </ul>
	E-MapReduce	<ul style="list-style-type: none"> <li>alicloud_emr_cluster: 集群</li> </ul>
	阿里云Elasticsearch	<ul style="list-style-type: none"> <li>alicloud_elasticsearch: 实例</li> </ul>
	开放搜索	<ul style="list-style-type: none"> <li>alicloud_open_search_app_group: OpenSearch应用</li> </ul>
	DataWorks	<ul style="list-style-type: none"> <li>alicloud_data_works_folder: 文件夹</li> </ul>
	Quick BI	<ul style="list-style-type: none"> <li>alicloud_quick_bi_user: 组织用户</li> </ul>
人工智能	DataHub	<ul style="list-style-type: none"> <li>alicloud_datahub_project: 项目</li> <li>alicloud_datahub_subscription: 订阅</li> <li>alicloud_datahub_topic: Topic</li> </ul>
	工业大脑开放平台	<ul style="list-style-type: none"> <li>alicloud_brain_industrial_pid_loop: 回路</li> <li>alicloud_brain_industrial_pid_organization: 组织</li> <li>alicloud_brain_industrial_pid_project: 项目</li> </ul>

		<ul style="list-style-type: none"><li>• <code>alicloud_forward_entry</code>: DNAT 列表中添加DNAT条目</li><li>• <code>alicloud_havip</code>: 高可用虚拟IP</li><li>• <code>alicloud_havip_attachment</code>: 将HaVip绑定到专有网络ECS实例或弹性网卡上</li><li>• <code>alicloud_nat_gateway</code>: Nat网关</li><li>• <code>alicloud_network_acl</code>: 网络ACL</li><li>• <code>alicloud_network_acl_attachment</code>: 绑定网络ACL到交换</li><li>• <code>alicloud_network_acl_entries</code>: 网络ACL规则</li><li>• <code>alicloud_route_entry</code>: 自定义路由条目</li><li>• <code>alicloud_route_table</code>: 自定义路由表</li><li>• <code>alicloud_route_table_attachment</code>: 将创建的自定义路由表和同一VPC内的交换机绑定</li><li>• <code>alicloud_router_interface</code>: 路由器接口</li><li>• <code>alicloud_router_interface_connection</code>: 发起端路由器接口向接收端发起连接</li><li>• <code>alicloud_snat_entry</code>: 在SNAT列表中添加SNAT条目</li><li>• <code>alicloud_vpc</code>: 专有网络</li><li>• <code>alicloud_vpc_dhcp_options_set</code>: DHCP选项集</li><li>• <code>alicloud_vpc_dhcp_options_set_attachment</code>: 将DHCP选项集关联到VPC</li><li>• <code>alicloud_vpc_flow_log</code>: 流日志</li><li>• <code>alicloud_vpc_ipv6_egress_rule</code>: IPv6地址仅主动出规则</li><li>• <code>alicloud_vpc_ipv6_gateway</code>: IPv6网关</li><li>• <code>alicloud_vpc_ipv6_internet_bandwidth</code>: IPv6地址公网带宽</li><li>• <code>alicloud_vpc_nat_ip</code>: NAT IP地址</li><li>• <code>alicloud_vpc_nat_ip_cidr</code>: NAT IP地址段</li></ul>
	专有网络 VPC	

		<ul style="list-style-type: none"><li>alicloud_vpc_traffic_mirror_filter: 流量镜像筛选条件</li><li>alicloud_vpc_traffic_mirror_filter_egress_rule: 出方向规则</li><li>alicloud_vpc_traffic_mirror_filter_ingress_rule: 入方向规则</li><li>alicloud_vpc_traffic_mirror_session: 流量镜像会话</li><li>alicloud_vswitch: 虚拟交换机</li><li>alicloud_vpc_vbr_ha: 快速倒换组</li><li>alicloud_vpc_bgp_group: BGP组</li><li>alicloud_vpc_bgp_peer: BGP邻居</li><li>alicloud_vpc_bgp_network: 宣告BGP网段</li></ul>
	云解析 PrivateZone	<ul style="list-style-type: none"><li>alicloud_pvtz_user_vpc_authorization: 跨账号VPC授权</li><li>alicloud_pvtz_zone: Private Zone</li><li>alicloud_pvtz_zone_attachment: 绑定或者解绑zone与VPC列表两者之间的关系</li><li>alicloud_pvtz_zone_record: Private Zone的解析记录</li><li>alicloud_pvtz_rule: 解析器转发规则</li><li>alicloud_pvtz_rule_attachment: 绑定或者解绑Rule与VPC列表两者之间的关系</li><li>alicloud_pvtz_endpoint: 解析器终端节点</li></ul>

	负载均衡CLB	<ul style="list-style-type: none"><li>• <code>alicloud_slb_acl</code>: 访问控制策略组</li><li>• <code>alicloud_slb_backend_server</code>: 添加后端服务器</li><li>• <code>alicloud_slb_ca_certificate</code>: CA 证书</li><li>• <code>alicloud_slb_domain_extension</code>: 域名扩展</li><li>• <code>alicloud_slb_listener</code>: 监听</li><li>• <code>alicloud_slb_load_balancer</code>: 负载均衡实例</li><li>• <code>alicloud_slb_master_slave_server_group</code>: 主备服务器组</li><li>• <code>alicloud_slb_rule</code>: 为指定的 HTTP 或 HTTPS 监听添加转发规则</li><li>• <code>alicloud_slb_server_certificate</code>: 服务器证书</li><li>• <code>alicloud_slb_server_group</code>: 服务器组</li><li>• <code>alicloud_slb_tls_cipher_policy</code>: TLS 策略</li><li>• <code>alicloud_slb_acl_entry_attachment</code>: 在访问控制策略组中添加 IP 条目</li><li>• <code>alicloud_slb_server_group_server_attachment</code>: 为服务器组添加服务器</li></ul>
--	---------	---



	负载均衡ALB	<ul style="list-style-type: none"> <li>• <code>alicloud_alb_acl</code>: 访问控制策略组</li> <li>• <code>alicloud_alb_health_check_template</code>: 健康检查模板</li> <li>• <code>alicloud_alb_listener</code>: 监听</li> <li>• <code>alicloud_alb_load_balancer</code>: 负载均衡实例</li> <li>• <code>alicloud_alb_rule</code>: 转发规则</li> <li>• <code>alicloud_alb_security_policy</code>: 自定义安全策略</li> <li>• <code>alicloud_alb_server_group</code>: 服务器组</li> <li>• <code>alb_listener_additional_certificate_attachment</code>: 向监听添加额外证书</li> <li>• <code>alicloud_alb_listener_acl_attachment</code>: 为监听配置访问控制策略</li> <li>• <code>alicloud_alb_acl_entry_attachment</code>: 为访问控制添加Entry</li> </ul>
	弹性公网 IP	<ul style="list-style-type: none"> <li>• <code>alicloud_eip_address</code>: 弹性公网IP</li> <li>• <code>alicloud_eip_association</code>: 将弹性公网IP绑定到同地域的云产品实例上</li> </ul>
	共享带宽	<ul style="list-style-type: none"> <li>• <code>alicloud_common_bandwidth_package</code>: 共享带宽实例</li> <li>• <code>alicloud_common_bandwidth_package_attachment</code>: 添加EIP到共享带宽中</li> </ul>
		<ul style="list-style-type: none"> <li>• <code>alicloud_cen_bandwidth_limit</code>: 两个地域间的跨地域互通带宽</li> <li>• <code>alicloud_cen_bandwidth_package</code>: 带宽包实例</li> <li>• <code>alicloud_cen_bandwidth_package_attachment</code>: 将带宽包实例绑定至云企业网实例</li> <li>• <code>alicloud_cen_flowlog</code>: 流日志</li> </ul>

网络与CDN

云企业网

- `alicloud_cen_instance`: 云企业网实例
- `alicloud_cen_instance_attachm`  
`ent`: 将网络实例加载到云企业网实例中
- `alicloud_cen_instance_grant`: 为云企业网实例授权
- `alicloud_cen_private_zone`: 设置访问PrivateZone服务
- `alicloud_cen_route_entry`: 将加载到CEN中的VPC或VBR的路由条目发布到CEN中
- `alicloud_cen_route_map`: 路由策略
- `alicloud_cen_route_service`: 为本地网络添加访问云服务的配置
- `alicloud_cen_transit_router`: 企业版转发路由器实例
- `alicloud_cen_transit_router_pe`  
`er_attachment`: 为企业版转发路由器实例创建跨地域连接
- `alicloud_cen_transit_router_ro`  
`ute_entry`: 在企业版转发路由器的路由表中添加路由条目
- `alicloud_cen_transit_router_ro`  
`ute_table`: 为企业版转发路由器创建自定义路由表
- `alicloud_cen_transit_router_ro`  
`ute_table_association`: 关联转发关系
- `alicloud_cen_transit_router_ro`  
`ute_table_propagation`: 设置路由学习关系
- `alicloud_cen_transit_router_vbr`  
`_attachment`: 在企业版转发路由器下创建边界路由器VBR连接
- `alicloud_cen_transit_router_vp`  
`c_attachment`: 在企业版转发路由器下创建专有网络VPC连接
- `alicloud_cen_vbr_health_check`: 健康检查
- `alicloud_cen_traffic_marking_p`  
`olicy`: 流量标记策略

	<div>全球加速</div> <div><ul style="list-style-type: none"><li>alicloud_ga_accelerator: 全球加速实例</li><li>alicloud_ga_bandwidth_package: 带宽包</li><li>alicloud_ga_bandwidth_package_attachment: 将带宽包与全球加速实例绑定</li><li>alicloud_ga_endpoint_group: 终端节点组</li><li>alicloud_ga_forwarding_rule: 转发策略</li><li>alicloud_ga_ip_set: 加速地域</li><li>alicloud_ga_listener: 监听</li><li>alicloud_ga_acl: 访问控制策略组</li><li>alicloud_ga_acl_attachment: 为监听绑定访问控制策略组</li><li>alicloud_ga_additional_certificate: 为HTTPS监听绑定扩展证书</li><li>alicloud_ga_accelerator_spare_ip_attachment: 为加速实例绑定备用IP</li></ul></div>
--	---

	智能接入网关	<ul style="list-style-type: none"><li>• <code>alicloud_cloud_connect_network</code>: 云连接网实例</li><li>• <code>alicloud_cloud_connect_network_attachment</code>: 将智能接入网关绑定到指定的云连接网中</li><li>• <code>alicloud_cloud_connect_network_grant</code>: 将云连接网实例授权给跨账号云企业网实例</li><li>• <code>alicloud_sag_acl</code>: 访问控制</li><li>• <code>alicloud_sag_acl_rule</code>: 访问控制规则</li><li>• <code>alicloud_sag_client_user</code>: 用户</li><li>• <code>alicloud_sag_dnat_entry</code>: DNAT条目</li><li>• <code>alicloud_sag_qos</code>: QoS策略实例</li><li>• <code>alicloud_sag_qos_car</code>: QoS的限速规则</li><li>• <code>alicloud_sag_qos_policy</code>: QoS策略流分类规则</li><li>• <code>alicloud_sag_snat_entry</code>: SNAT条目</li><li>• <code>alicloud_smartag_flow_log</code>: 流日志</li></ul>
	高速通道	<ul style="list-style-type: none"><li>• <code>alicloud_express_connect_physical_connection</code>: 物理专线接入</li><li>• <code>alicloud_express_connect_virtual_border_router</code>: 边界路由器VBR实例</li></ul>
	CDN	<ul style="list-style-type: none"><li>• <code>alicloud_cdn_domain_config</code>: 域名配置</li><li>• <code>alicloud_cdn_domain_new</code>: 域名</li><li>• <code>alicloud_cdn_real_time_log_delivery</code>: 实时日志投递</li><li>• <code>alicloud_cdn_fc_trigger</code>: 函数计算触发器</li></ul>

	全站加速	<ul style="list-style-type: none"><li>• <code>alicloud_dcdn_domain</code>: 域名</li><li>• <code>alicloud_dcdn_domain_config</code>: 域名配置</li><li>• <code>alicloud_dcdn_ipa_domain</code>: IPA层应用加速域名</li></ul>
	SCDN	<ul style="list-style-type: none"><li>• <code>alicloud_scdn_domain</code>: 域名</li><li>• <code>alicloud_scdn_domain_config</code>: 域名配置</li></ul>
	边缘节点服务ENS	<ul style="list-style-type: none"><li>• <code>alicloud_ens_key_pair</code>: SSH密钥对</li></ul>
	VPN网关	<ul style="list-style-type: none"><li>• <code>alicloud_ssl_vpn_client_cert</code>: SSL-VPN客户端证书</li><li>• <code>alicloud_ssl_vpn_server</code>: SSL-VPN服务端</li><li>• <code>alicloud_vpn_connection</code>: IPsec连接</li><li>• <code>alicloud_vpn_customer_gateway</code>: 用户网关</li><li>• <code>alicloud_vpn_gateway</code>: VPN网关</li><li>• <code>alicloud_vpn_route_entry</code>: 为VPN网关实例创建目的路由</li><li>• <code>alicloud_vpn_ipsec_server</code>: Ipsec服务器</li><li>• <code>alicloud_vpn_pbr_route_entry</code>: 策略路由</li></ul>

	私网连接 PrivateLink	<ul style="list-style-type: none"><li>alicloud_privatelink_vpc_endpoint: 终端节点</li><li>alicloud_privatelink_vpc_endpoint_connection: 终端节点连接</li><li>alicloud_privatelink_vpc_endpoint_service: 终端节点服务</li><li>alicloud_privatelink_vpc_endpoint_service_resource: 为终端节点服务添加服务资源</li><li>alicloud_privatelink_vpc_endpoint_service_user: 添加服务白名单</li><li>alicloud_privatelink_vpc_endpoint_zone: 为终端节点添加可用区</li></ul>
	任播弹性公网IP	<ul style="list-style-type: none"><li>alicloud_eipanycast_anycast_eip_address: Anycast EIP实例</li><li>alicloud_eipanycast_anycast_eip_address_attachment: 将 Anycast EIP绑定到指定地域的云资源实例上</li></ul>
视频服务	视频点播	<ul style="list-style-type: none"><li>alicloud_vod_domain: 加速域名</li></ul>
	视图计算	<ul style="list-style-type: none"><li>alicloud_video_surveillance_system_group: 业务空间</li></ul>
	低代码音视频工厂	<ul style="list-style-type: none"><li>alicloud_imp_app_template: 应用模板</li></ul>

	容器服务 ACK	<ul style="list-style-type: none"><li>alicloud_cs_autoscaling_config : 自动伸缩配置</li><li>alicloud_cs_edge_kubernetes : Kubernetes边缘托管版集群</li><li>alicloud_cs_kubernetes: Kubernetes专有版集群</li><li>alicloud_cs_kubernetes_autoscaler</li><li>alicloud_cs_kubernetes_node_pool: 节点池</li><li>alicloud_cs_kubernetes_permissions: 全量更新RAM用户集群授权信息</li><li>alicloud_cs_managed_kubernetes: Kubernetes托管版集群</li><li>alicloud_cs_serverless_kubernetes: Serverless Kubernetes集群</li><li>alicloud_cs_kubernetes_addon : 为Kubernetes集群安装组件</li></ul>
	服务网格 ASM	<ul style="list-style-type: none"><li>alicloud_service_mesh_service_mesh: 服务网格实例</li><li>alicloud_service_mesh_user_permission: RBAC权限管理</li></ul>

容器与中间件	容器镜像服务 ACR	<ul style="list-style-type: none"><li>alicloud_cr_ee_instance: 容器企业版镜像实例</li><li>alicloud_cr_ee_namespace: 企业版镜像仓库命名空间</li><li>alicloud_cr_ee_repo: 企业版镜像仓库</li><li>alicloud_cr_ee_sync_rule: 企业版镜像仓库同步规则</li><li>alicloud_cr_endpoint_acl_policy: 为企业版实例访问入口（限公网）创建白名单策略</li><li>alicloud_cr_namespace: 个人版镜像仓库命名空间</li><li>alicloud_cr_repo: 个人版镜像仓库</li><li>alicloud_cr_chart_namespace: Chart命名空间</li><li>alicloud_cr_chain: 交付链</li><li>alicloud_cr_chart_repository: Chart仓库</li></ul>
	企业级分布式应用服务 EDAS	<ul style="list-style-type: none"><li>alicloud_edas_application: ECS应用</li><li>alicloud_edas_application_deployment: 部署应用</li><li>alicloud_edas_application_scale: 扩容应用</li><li>alicloud_edas_cluster: ECS集群</li><li>alicloud_edas_deploy_group: 应用分组</li><li>alicloud_edas_instance_cluster_attachment: 使用 ECS 自带的云助手安装 EDAS Agent（将 ECS 导入 EDAS）</li><li>alicloud_edas_k8s_application: K8s应用</li><li>alicloud_edas_k8s_cluster: K8S集群</li><li>alicloud_edas_slb_attachment: 绑定SLB到EDAS应用</li><li>alicloud_edas_namespace: 自定义命名空间</li></ul>



	微服务应用	<ul style="list-style-type: none"> <li>• <a href="#">alicloud_mse_cluster</a>: MSE注册&amp;配置中心集群</li> <li>• <a href="#">alicloud_mse_gateway</a>: MSE网关</li> <li>• <a href="#">alicloud_mse_engine_namespace</a>: 命名空间</li> </ul>
	消息队列RocketMQ版	<ul style="list-style-type: none"> <li>• <a href="#">alicloud_ons_group</a>: 客户端Group</li> <li>• <a href="#">alicloud_ons_instance</a>: 实例</li> <li>• <a href="#">alicloud_ons_topic</a>: Topic</li> <li>• <a href="#">alicloud_mse_znode</a>: Zookeeper数据节点</li> </ul>
	事件总线EventBridge	<ul style="list-style-type: none"> <li>• <a href="#">alicloud_event_bridge_event_bus</a>: 事件总线</li> <li>• <a href="#">alicloud_event_bridge_event_source</a>: 事件源</li> <li>• <a href="#">alicloud_event_bridge_rule</a>: 事件规则</li> <li>• <a href="#">event_bridge_service_linked_role</a>: 为事件总线EventBridge服务创建服务角色</li> </ul>
	消息队列Kafka版	<ul style="list-style-type: none"> <li>• <a href="#">alicloud_alikafka_consumer_group</a>: 消费组</li> <li>• <a href="#">alicloud_alikafka_instance</a>: 实例</li> <li>• <a href="#">alicloud_alikafka_sasl_acl</a>: SASL用户和客户端使用SDK收发消息权限</li> <li>• <a href="#">alicloud_alikafka_sasl_user</a>: SASL用户</li> <li>• <a href="#">alicloud_alikafka_topic</a>: 消息主题</li> <li>• <a href="#">alicloud_alikafka_instance_allowed_ip_attachment</a>: 为实例配置IP白名单</li> </ul>

	消息队列RabbitMQ版	<ul style="list-style-type: none"><li>alicloud_amqp_binding: 为Exchange绑定Queue或者Exchange</li><li>alicloud_amqp_exchange: Exchange</li><li>alicloud_amqp_instance: 实例</li><li>alicloud_amqp_queue: 消息队列</li><li>alicloud_amqp_virtual_host: 虚拟主机</li></ul>
	消息服务MNS	<ul style="list-style-type: none"><li>alicloud_mns_queue: 消息队列</li><li>alicloud_mns_topic: 消息主题</li><li>alicloud_mns_topic_subscription: 消息订阅</li></ul>
	分布式任务调度 SchedulerX	<ul style="list-style-type: none"><li>alicloud_schedulerx_namespace: 命名空间</li></ul>
	应用实时监控服务ARMS	<ul style="list-style-type: none"><li>alicloud_arms_alert_contact: 报警联系人</li><li>alicloud_arms_alert_contact_group: 报警联系组</li><li>alicloud_arms_dispatch_rule: 分派策略</li><li>alicloud_arms_prometheus_alert_rule: Prometheus 告警规则</li></ul>

	云监控	<ul style="list-style-type: none"><li>alicloud_cms_alarm: 阈值报警规则</li><li>alicloud_cms_alarm_contact: 报警联系人</li><li>alicloud_cms_alarm_contact_group: 报警联系组</li><li>alicloud_cms_dynamic_tag_group: 通过标签自动创建应用分组</li><li>alicloud_cms_group_metric_rule: 应用分组的报警规则</li><li>alicloud_cms_metric_rule_template: 报警模板</li><li>alicloud_cms_monitor_group: 应用分组</li><li>alicloud_cms_monitor_group_instances: 添加指定资源到指定应用分组</li><li>alicloud_cms_site_monitor: 站点监控任务</li><li>alicloud_cms_namespace: 指标仓库</li><li>alicloud_cms_sls_group: 为SLS日志的监控项创建Logstore组</li></ul>
	移动研发平台EMAS	<ul style="list-style-type: none"><li>alicloud_mhub_app: 应用</li><li>alicloud_mhub_product: 工作空间</li></ul>
	云效	<ul style="list-style-type: none"><li>alicloud_rdc_organization: 组织</li></ul>

开发与运维	资源管理	<ul style="list-style-type: none"><li>alicloud_resource_manager_account: 成员</li><li>alicloud_resource_manager_control_policy: 自定义管控策略</li><li>alicloud_resource_manager_control_policy_attachment: 绑定自定义管控策略</li><li>alicloud_resource_manager_folder: 资源夹名称</li><li>alicloud_resource_manager_handshake: 邀请</li><li>alicloud_resource_manager_policy: 权限策略</li><li>alicloud_resource_manager_policy_attachment: 授权对象（RAM用户、RAM用户组或RAM角色）添加权限策略</li><li>alicloud_resource_manager_policy_version: 权限策略版本</li><li>alicloud_resource_manager_resource_directory: 资源目录</li><li>alicloud_resource_manager_resource_group: 资源组</li><li>alicloud_resource_manager_resource_share: 共享单元</li><li>alicloud_resource_manager_role: 角色</li><li>alicloud_resource_manager_shared_resource: 为共享单元关联共享资源</li><li>alicloud_resource_manager_shared_target: 为共享单元关联资源使用者</li><li>alicloud_resource_manager_service_linked_role: 为资源管理创建服务角色</li></ul>
-------	------	---

	配置审计	<ul style="list-style-type: none"> <li>alicloud_config_aggregate_compliance_pack: 企业管理合规包</li> <li>alicloud_config_aggregate_config_rule: 企业管理规则</li> <li>alicloud_config_aggregator: 企业管理账号组</li> <li>alicloud_config_compliance_pack: 普通账号合规包</li> <li>alicloud_config_configuration_recorder: 配置记录</li> <li>alicloud_config_delivery_channel: 投递渠道（已过期）</li> <li>alicloud_config_rule: 规则</li> <li>alicloud_config_delivery: 普通账号投递渠道</li> <li>alicloud_config_aggregate_delivery: 指定账号组投递渠道</li> </ul>
	配额中心	<ul style="list-style-type: none"> <li>alicloud_quotas_quota_alarm: 配额告警</li> <li>alicloud_quotas_quota_application: 配额提升申请</li> </ul>
	云SSO	<ul style="list-style-type: none"> <li>alicloud_cloud_sso_access_assignment: 在RD账号上授权</li> <li>alicloud_cloud_sso_access_configuration: 访问配置</li> <li>alicloud_cloud_sso_directory: 目录</li> <li>alicloud_cloud_sso_group: 用户组</li> <li>alicloud_cloud_sso_scim_server_credential: SCIM密钥</li> <li>alicloud_cloud_sso_user: 用户</li> <li>alicloud_cloud_sso_user_attachment: 添加用户到用户组</li> <li>alicloud_cloud_sso_access_configuration_provisioning: 部署策略</li> </ul>
物联网	阿里云物联网平台	<ul style="list-style-type: none"> <li>alicloud_iot_device_group: 分组</li> </ul>

企业应用与服务	API 网关	<ul style="list-style-type: none"><li>alicloud_api_gateway_api: API</li><li>alicloud_api_gateway_app: 应用</li><li>alicloud_api_gateway_app_attachment: 给指定app添加多个API的访问权限</li><li>alicloud_api_gateway_group: API 分组</li><li>alicloud_api_gateway_vpc_access: 添加 VPC 授权</li></ul>
	邮件推送	<ul style="list-style-type: none"><li>alicloud_direct_mail_domain: 域名</li><li>alicloud_direct_mail_mail_address: 发信地址</li><li>alicloud_direct_mail_receivers: 收件人列表</li><li>alicloud_direct_mail_tag: 标签</li></ul>
	云市场	<ul style="list-style-type: none"><li>alicloud_market_order: 订单</li></ul>
	消息中心	<ul style="list-style-type: none"><li>alicloud_msc_sub_contact: 消息接收人</li><li>alicloud_msc_sub_subscription: 消息订阅</li><li>alicloud_msc_sub_webhook: 消息挂钩</li></ul>

域名与网站	云解析 DNS	<ul style="list-style-type: none"><li>• <code>alicloud_alidns_domain</code>: 域名</li><li>• <code>alicloud_alidns_domain_attachment</code>: 绑定付费版DNS域名到实例ID</li><li>• <code>alicloud_alidns_domain_group</code>: 域名分组</li><li>• <code>alicloud_alidns_instance</code>: 实</li><li>• <code>alicloud_alidns_record</code>: 解析记录</li><li>• <code>alicloud_alidns_custom_line</code>: 自定义线路</li><li>• <code>alicloud_alidns_gtm_instance</code>: 全局流量管理产品实例</li><li>• <code>alicloud_alidns_address_pool</code>: 地址池</li><li>• <code>alicloud_alidns_access_strategy</code>: 访问策略</li><li>• <code>alicloud_alidns_monitor_config</code>: 健康检查</li></ul>
-------	---------	--

## 4. 教程

### 4.1. 云服务器ECS

#### 4.1.1. 创建一台ECS实例

本文介绍如何使用Terraform创建一台ECS实例。

##### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

##### 操作步骤

##### 1. 创建专有网络和交换机。

- i. 创建`terraform.tf`文件，输入以下内容，并保存在当前的执行目录中。

```
resource "alicloud_vpc" "vpc" {  
  name      = "tf_test_foo"  
  cidr_block = "172.16.0.0/12"  
}  
  
resource "alicloud_vswitch" "vsw" {  
  vpc_id            = alicloud_vpc.vpc.id  
  cidr_block        = "172.16.0.0/21"  
  availability_zone = "cn-beijing-b"  
}
```

- ii. 运行 `terraform apply` 开始创建。

- iii. 运行 `terraform show` 查看已创建的专有网络和交换机。

您也可以登录VPC控制台查看专有网络和交换机的属性。

##### 2. 在上一步创建的专有网络中创建一个安全组，并添加一个允许任何地址访问的安全组规则。



- i. 在 `terraform.tf` 文件中增加以下内容。

```
resource "alicloud_security_group" "default" {
  name = "default"
  vpc_id = alicloud_vpc.vpc.id
}

resource "alicloud_security_group_rule" "allow_all_tcp" {
  type           = "ingress"
  ip_protocol    = "tcp"
  nic_type       = "intranet"
  policy         = "accept"
  port_range     = "1/65535"
  priority       = 1
  security_group_id = alicloud_security_group.default.id
  cidr_ip        = "0.0.0.0/0"
}
```

- ii. 运行 `terraform apply` 开始创建。
- iii. 运行 `terraform show` 查看已创建的安全组和安全组规则。

您也可以登录ECS控制台查看安全组和安全组规则。

### 3. 创建ECS实例。

- i. 在 `terraform.tf` 文件中增加以下内容。

```
resource "alicloud_instance" "instance" {
  # cn-beijing
  availability_zone = "cn-beijing-b"
  security_groups = alicloud_security_group.default.*.id
  # series III
  instance_type      = "ecs.n2.small"
  system_disk_category = "cloud_efficiency"
  image_id           = "ubuntu_18_04_64_20G_alibase_20190624.vhd"
  instance_name      = "test_foo"
  vswitch_id         = alicloud_vswitch.vsw.id
  internet_max_bandwidth_out = 10
  password           = "<replace_with_your_password>"
}
```

#### ? 说明

- 在上述示例中，指定了 `internet_max_bandwidth_out= 10`，因此会自动为实例分配一个公网IP。
- 详细的参数解释请参见 [阿里云参数说明](#)。

- ii. 运行 `terraform apply` 开始创建。
- iii. 运行 `terraform show` 查看已创建的ECS实例。
- iv. 运行 `ssh root@<publicip>`，并输入密码来访问ECS实例。

```
provider "alicloud" {}

resource "alicloud_vpc" "vpc" {
  name      = "tf_test_foo"
  cidr_block = "172.16.0.0/12"
}

resource "alicloud_vswitch" "vsw" {
  vpc_id      = alicloud_vpc.vpc.id
  cidr_block   = "172.16.0.0/21"
  availability_zone = "cn-beijing-b"
}

resource "alicloud_security_group" "default" {
  name = "default"
  vpc_id = alicloud_vpc.vpc.id
}

resource "alicloud_instance" "instance" {
  # cn-beijing
  availability_zone = "cn-beijing-b"
  security_groups = alicloud_security_group.default.*.id
  # series III
  instance_type      = "ecs.n2.small"
  system_disk_category = "cloud_efficiency"
  image_id            = "ubuntu_18_04_64_20G_alibase_20190624.vhd"
  instance_name       = "test_foo"
  vswitch_id          = alicloud_vswitch.vsw.id
  internet_max_bandwidth_out = 10
}

resource "alicloud_security_group_rule" "allow_all_tcp" {
  type          = "ingress"
  ip_protocol   = "tcp"
  nic_type      = "intranet"
  policy        = "accept"
  port_range    = "1/65535"
  priority      = 1
  security_group_id = alicloud_security_group.default.id
  cidr_ip        = "0.0.0.0/0"
}
```

## 4.1.2. 创建多台ECS实例

本文介绍如何使用Terraform模块批量创建多台ECS实例。

### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

### 操作步骤

1. 创建专有网络和交换机。

- i. 创建`terraform.tf`文件，输入以下内容，保存在当前的执行目录中。

```
resource "alicloud_vpc" "vpc" {
  name      = "tf_test_foo"
  cidr_block = "172.16.0.0/12"
}

resource "alicloud_vswitch" "vsw" {
  vpc_id      = alicloud_vpc.vpc.id
  cidr_block  = "172.16.0.0/21"
  availability_zone = "cn-beijing-b"
}
```

- ii. 运行 `terraform apply` 开始创建。
- iii. 运行 `terraform show` 查看已创建的专有网络和交换机。

您也可以登录VPC控制台查看专有网络和交换机的属性。

2. 在上一步创建的专有网络中创建一个安全组，并添加一个允许任何地址访问的安全组规则。

- i. 在`terraform.tf`文件中增加以下内容。

```
resource "alicloud_security_group" "default" {
  name = "default"
  vpc_id = alicloud_vpc.vpc.id
}

resource "alicloud_security_group_rule" "allow_all_tcp" {
  type          = "ingress"
  ip_protocol   = "tcp"
  nic_type      = "intranet"
  policy        = "accept"
  port_range    = "1/65535"
  priority      = 1
  security_group_id = alicloud_security_group.default.id
  cidr_ip       = "0.0.0.0/0"
}
```


- ii. 运行 `terraform apply` 开始创建。
  - iii. 运行 `terraform show` 查看已创建的安全组和安全组规则。

您也可以登录ECS控制台查看安全组和安全组规则。

3. 使用Module创建多台ECS实例。在本示例中，创建3台ECS实例。

i. 在 `terraform.tf` 文件中增加以下内容。

```
module "tf-instances" {
  source              = "alibaba/ecs-instance/alicloud"
  region              = "cn-beijing"
  number_of_instances = "3"
  vswitch_id          = alicloud_vswitch.vsw.id
  group_ids            = [alicloud_security_group.default.id]
  private_ips          = ["172.16.0.10", "172.16.0.11", "172.16.0.12"]
  image_ids            = ["ubuntu_18_04_64_20G_alibase_20190624.vhd"]
  instance_type        = "ecs.n2.small"
  internet_max_bandwidth_out = 10
  associate_public_ip_address = true
  instance_name        = "my_module_instances_"
  host_name             = "sample"
  internet_charge_type  = "PayByTraffic"
  password              = "User@123"
  system_disk_category = "cloud_ssd"
  data_disks = [
    {
      disk_category = "cloud_ssd"
      disk_name      = "my_module_disk"
      disk_size      = "50"
    }
  ]
}
```

 **说明** 在上述示例中，同时指定 `associate_public_ip_address = true` 和 `internet_max_bandwidth_out = 10`，因此会自动为实例分配一个公网IP。详细的参数解释请参见 [参数说明](#)。

- ii. 运行 `terraform apply` 开始创建。
- iii. 运行 `terraform show` 查看已创建的ECS实例。
- iv. 运行 `ssh root@<publicip>`，并输入密码来访问ECS实例。

```
provider "alicloud" {}

resource "alicloud_vpc" "vpc" {
  name      = "tf_test_foo"
  cidr_block = "172.16.0.0/12"
}

resource "alicloud_vswitch" "vsw" {
  vpc_id      = alicloud_vpc.vpc.id
  cidr_block   = "172.16.0.0/21"
  availability_zone = "cn-beijing-b"
}

resource "alicloud_security_group" "default" {
  name      = "default"
  vpc_id    = alicloud_vpc.vpc.id
}

resource "alicloud_security_group_rule" "allow_all_tcp" {
  type            = "ingress"
  ip_protocol     = "tcp"
  nic_type        = "intranet"
  policy          = "accept"
  port_range      = "1/65535"
  priority        = 1
  security_group_id = alicloud_security_group.default.id
  cidr_ip          = "0.0.0.0/0"
}

module "tf-instances" {
  source              = "alibaba/ecs-instance/alicloud"
  region              = "cn-beijing"
  number_of_instances = "3"
  vswitch_id          = alicloud_vswitch.vsw.id
  group_ids            = [alicloud_security_group.default.id]
  private_ips          = ["172.16.0.10", "172.16.0.11", "172.16.0.12"]
  image_ids            = ["ubuntu_18_04_64_20G_alibase_20190624.vhd"]
  instance_type        = "ecs.n2.small"
  internet_max_bandwidth_out = 10
  associate_public_ip_address = true
  instance_name        = "my_module_instances_"
  host_name             = "sample"
  internet_charge_type  = "PayByTraffic"
  password             = "User@123"
  system_disk_category = "cloud_ssd"
  data_disks = [
    {
      disk_category = "cloud_ssd"
      disk_name     = "my_module_disk"
      disk_size     = "50"
    }
  ]
}
```

## 相关文档

- [terraform-alicloud-ecs-instance](#)

## 4.2. 负载均衡SLB

### 4.2.1. 通过Terraform管理负载均衡服务

本文介绍如何使用Terraform创建一个负载均衡实例，并为其添加监听。

#### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。


#### 背景信息

本文以创建一个名为slb\_worder的预付费公网实例，并为它配置TCP、UDP和HTTP监听为例。

#### 操作步骤

##### 1. 创建一个负载均衡实例。

- i. 创建`terraform.tf`文件，输入以下内容，并保存在当前的执行目录中。

 说明 需要为每个Terraform项目创建一个独立的执行目录。在初始化配置之前至少需要有1个.tf文件。

```
resource "alicloud_slb" "instance" {
  name                = "slb_worder"
  internet_charge_type = "PayByTraffic"
  address_type        = "internet"
}
```


- ii. 运行`terraform apply`开始创建。出现类似下面的日志，说明创建成功。

```
alicloud_slb.instance: Creating...
alicloud_slb.instance: Creation complete after 7s [id=lb-bp1edqadye9xvpb3fvo7u]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

- iii. 运行`terraform show`查看已创建的负载均衡实例。

##### 2. 添加监听。以下示例添加了HTTP、UDP和HTTP三种协议的监听。

- i. 创建`listener.tf`文件，输入以下内容，并保存在当前的执行目录中。

 说明 在该目录下所有\*.tf 文件都会被terraform自动加载。因此，用户可以按照实际用途将配置信息写入不同的文件中。

```
resource "alicloud_slb_listener" "tcp" {
  load_balancer_id      = alicloud_slb.instance.id
  backend_port          = "22"
  frontend_port         = "22"
  protocol              = "tcp"
  bandwidth             = "10"
  health_check_type     = "tcp"
  persistence_timeout   = 3600
  healthy_threshold     = 8
  unhealthy_threshold   = 8
  health_check_timeout  = 8
  health_check_interval = 5
  health_check_http_code = "http_2xx"
  health_check_connect_port = 20
  health_check_uri      = "/console"
  established_timeout   = 600
}

resource "alicloud_slb_listener" "udp" {
  load_balancer_id      = alicloud_slb.instance.id
  backend_port          = 2001
  frontend_port         = 2001
  protocol              = "udp"
  bandwidth             = 10
  persistence_timeout   = 3600
  healthy_threshold     = 8
  unhealthy_threshold   = 8
  health_check_timeout  = 8
  health_check_interval = 4
  health_check_connect_port = 20
}

resource "alicloud_slb_listener" "http" {
  load_balancer_id      = alicloud_slb.instance.id
  backend_port          = 80
  frontend_port         = 80
  protocol              = "http"
  sticky_session        = "on"
  sticky_session_type   = "insert"
  cookie                = "testslblistenercookie"
  cookie_timeout        = 86400
  health_check          = "on"
  health_check_uri      = "/cons"
  health_check_connect_port = 20
  healthy_threshold     = 8
  unhealthy_threshold   = 8
  health_check_timeout  = 8
  health_check_interval = 5
  health_check_http_code = "http_2xx,http_3xx"
  bandwidth             = 10
  request_timeout       = 80
  idle_timeout          = 30
}
```

- ii. 运行`terraform apply`开始创建资源。出现类似下面的日志，说明创建成功。

```
alicloud_slb_listener.tcp: Creation complete after 5s (ID: lb-bp1mq8qmsxl96pw6lruja:22)
alicloud_slb_listener.http: Creation complete after 5s (ID: lb-bp1mq8qmsxl96pw6lruja:80)
alicloud_slb_listener.udp: Creation complete after 5s (ID: lb-bp1mq8qmsxl96pw6lruja:2001)
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

- iii. 运行`terraform show`查看已创建的监听。

## 相关文档

- [Terraform管理负载均衡服务代码示例](#)

## 4.2.2. 通过Terraform在专有网络中创建负载均衡实例

本文介绍如何使用Terraform在专有网络中创建负载均衡实例。

### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

### 操作步骤

1. 创建专有网络和交换机。本示例中在cn-hangzhou-b区域创建了一个专有网络和交换机。

- i. 创建`terraform.tf`文件，输入以下内容，并保存在当前的执行目录中。

```
resource "alicloud_vpc" "main" {
  name      = "alicloud"
  # 专有网络地址块
  cidr_block = "10.1.0.0/21"
}

resource "alicloud_vswitch" "main" {
  vpc_id      = alicloud_vpc.main.id
  # 交换机地址块
  cidr_block  = "10.1.0.0/24"
  # 可用区
  availability_zone = "cn-hangzhou-b"
  # 资源依赖
  depends_on = [alicloud_vpc.main]
}
```

- ii. 运行`terraform apply`开始创建资源。

- iii. 运行`terraform show`查看已创建的专有网络和交换机。

2. 创建负载均衡实例，并添加监听。本示例中创建了一个预付费的负载均衡实例，并添加了一个tcp监听。



i. 在`terraform.tf`文件中增加以下内容：

```
resource "alicloud_slb" "instance" {
  name           = "slb_test"
  vswitch_id     = alicloud_vswitch.main.id
  internet_charge_type = "PayByTraffic"
}

resource "alicloud_slb_listener" "listener" {
  load_balancer_id = alicloud_slb.instance.id
  backend_port     = "2111"
  frontend_port    = "21"
  protocol         = "tcp"
  bandwidth       = "5"
}
```

ii. 运行`terraform apply`开始创建。出现类似下面的日志，说明创建成功。

```
alicloud_slb.instance: Creating...
alicloud_slb.instance: Creation complete after 3s [id=lb-bp1li4zjp52xnzh2849hw]
alicloud_slb_listener.listener: Creating...
alicloud_slb_listener.listener: Creation complete after 1s [id=lb-bp1li4zjp52xnzh2849hw:tcp:21]
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

iii. 运行`terraform show`查看已经创建负载均衡实例。

## 相关文档

- [专有网络中创建负载均衡代码示例](#)

## 4.3. 云数据库RDS版

### 4.3.1. 创建一个云数据库实例

本文介绍如何使用Terraform创建一个云数据库实例。

#### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

#### 操作步骤

1. 定义专有网络、交换机、数据库实例及相关的数据库和账号等资源，`terraform.tf`文件内容如下。

 说明 需要为每个Terraform项目创建一个独立的执行目录。

```
resource "alicloud_vpc" "main" {
  name      = "alicloud"
  cidr_block = "10.1.0.0/21"
}
resource "alicloud_vswitch" "main" {
  vpc_id          = alicloud_vpc.main.id
  cidr_block      = "10.1.0.0/24"
  availability_zone = "cn-hangzhou-b"
  depends_on     = [alicloud_vpc.main]
}
resource "alicloud_db_instance" "instance" {
  engine          = "MySQL"
  engine_version  = "5.6"
  instance_type   = "rds.mysql.t1.small"
  instance_storage = "10"
  vswitch_id      = alicloud_vswitch.main.id
}
resource "alicloud_db_account" "account" {
  instance_id = alicloud_db_instance.instance.id
  name        = "tf_account"
  password    = "!Test@123456"
}
resource "alicloud_db_database" "db" {
  instance_id = alicloud_db_instance.instance.id
  name        = "tf_database"
}
resource "alicloud_db_account_privilege" "privilege" {
  instance_id = alicloud_db_instance.instance.id
  account_name = alicloud_db_account.account.name
  db_names    = [alicloud_db_database.db.name]
}
resource "alicloud_db_connection" "connection" {
  instance_id      = alicloud_db_instance.instance.id
  connection_prefix = "tf-example"
}
```

2. 运行**terraform apply**开始创建资源。出现类似下面的日志，说明创建成功。

```
alicloud_vpc.main: Creating...
alicloud_vpc.main: Creation complete after 6s [id=vpc-bplqdtiaztg1f9g5cqv3n]
alicloud_vswitch.main: Creating...
alicloud_vswitch.main: Creation complete after 5s [id=vsw-bpli5dkhraoplr0pga7oy]
alicloud_db_instance.instance: Creating...
alicloud_db_instance.instance: Still creating... [10s elapsed]
alicloud_db_instance.instance: Still creating... [20s elapsed]
alicloud_db_instance.instance: Still creating... [30s elapsed]
alicloud_db_instance.instance: Still creating... [40s elapsed]
alicloud_db_instance.instance: Still creating... [50s elapsed]
...
alicloud_db_instance.instance: Still creating... [5m0s elapsed]
alicloud_db_instance.instance: Creation complete after 5m9s [id=rm-bp1mwp0hs49bgoizk]
alicloud_db_connection.connection: Creating...
alicloud_db_database.db: Creating...
alicloud_db_account.account: Creating...
alicloud_db_database.db: Creation complete after 0s [id=rm-bp1mwp0hs49bgoizk:tf_databas
e]
alicloud_db_account.account: Creation complete after 2s [id=rm-bp1mwp0hs49bgoizk:tf_acc
ount]
alicloud_db_account_privilege.privilege: Creating...
alicloud_db_connection.connection: Still creating... [10s elapsed]
alicloud_db_account_privilege.privilege: Still creating... [10s elapsed]
alicloud_db_connection.connection: Still creating... [20s elapsed]
alicloud_db_account_privilege.privilege: Still creating... [20s elapsed]
alicloud_db_account_privilege.privilege: Creation complete after 22s [id=rm-bp1mwp0hs49
bgoizk:tf_account:ReadOnly]
alicloud_db_connection.connection: Creation complete after 28s [id=rm-bp1mwp0hs49bgoizk
:tf-example]
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
```

## 相关文档

- [Terraform创建云数据库代码示例](#)
- [alicloud\\_db\\_account](#)

## 4.4. 专有网络VPC

### 4.4.1. 通过Terraform创建基于阿里云的自定义私有网络

本文介绍如何使用Terraform创建专有网络，交换机以及NAT网关。

#### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

#### 操作步骤

## 1. 创建专有网络和交换机。

### i. 创建`terraform.tf`文件，定义专有网络和交换机资源，文件内容如下。

```
provider "alicloud" {  
  configuration_source = "terraform-provider-alicloud/examples/vpc"  
}  
  
resource "alicloud_vpc" "main" {  
  # VPC名称  
  vpc_name = "alicloud"  
  # VPC地址块  
  cidr_block = "10.1.0.0/21"  
}  
  
resource "alicloud_vswitch" "main" {  
  # VPC ID  
  vpc_id = alicloud_vpc.main.id  
  # 交换机地址块  
  cidr_block = "10.1.0.0/24"  
  # 可用区  
  availability_zone = "cn-hangzhou-b"  
  # 资源依赖, 会优先创建该依赖资源  
  depends_on = [alicloud_vpc.main]  
}
```

### ii. 运行`terraform apply`开始创建。出现类似下面的日志，说明创建成功。

```
alicloud_vpc.main: Creating...  
alicloud_vpc.main: Creation complete after 6s [id=vpc-bpluxxxxxxxxxxxxx]  
alicloud_vswitch.main: Creating...  
alicloud_vswitch.main: Creation complete after 6s [id=vsw-bp17yxxxxxxxxxxx]  
Apply complete! Resources: 2 added, 0 changed, 0 destroyed
```

### iii. 运行`terraform show`查看已创建的专有和专有网络。

## 2. 创建NAT网关。

i. 在 `terraform.tf` 文件中增加以下内容：

```
variable "name" {
  default = "natGatewayExampleName"
}

resource "alicloud_vpc" "enhanced" {
  vpc_name     = var.name
  cidr_block   = "10.0.0.0/8"
}

data "alicloud_enhanced_nat_available_zones" "enhanced" {}

resource "alicloud_vswitch" "enhanced" {
  vswitch_name = var.name
  zone_id      = data.alicloud_enhanced_nat_available_zones.enhanced.zones.0.zone_id
  cidr_block    = "10.10.0.0/20"
  vpc_id        = alicloud_vpc.enhanced.id
}

resource "alicloud_nat_gateway" "main" {
  vpc_id           = alicloud_vpc.enhanced.id
  nat_gateway_name = var.name
  payment_type     = "PayAsYouGo"
  vswitch_id       = alicloud_vswitch.enhanced.id
  nat_type         = "Enhanced"
}

resource "alicloud_eip" "foo" {}

resource "alicloud_eip_association" "foo" {
  allocation_id = alicloud_eip.foo.id
  instance_id   = alicloud_nat_gateway.main.id
}
```

ii. 运行 `terraform apply` 开始创建。出现下面的日志，说明创建成功。

```
alicloud_eip.foo: Creating...
alicloud_vpc.enhanced: Creating...
alicloud_vpc.enhanced: Creation complete after 6s [id=vpc-2zeqn3iafxxxxxxxxx]
alicloud_vswitch.enhanced: Creating...
alicloud_eip.foo: Creation complete after 8s [id=eip-2zew51gfgxxxxxxxxx]
alicloud_vswitch.enhanced: Creation complete after 6s [id=vsw-2zeesfxxxxxxxxx]
alicloud_nat_gateway.main: Creating...
alicloud_nat_gateway.main: Still creating... [10s elapsed]
alicloud_nat_gateway.main: Still creating... [20s elapsed]
alicloud_nat_gateway.main: Still creating... [30s elapsed]
alicloud_nat_gateway.main: Still creating... [40s elapsed]
alicloud_nat_gateway.main: Still creating... [50s elapsed]
alicloud_nat_gateway.main: Creation complete after 57s [id=ngw-2ze6yxxxxxxxxxxx]
alicloud_eip_association.foo: Creating...
alicloud_eip_association.foo: Still creating... [10s elapsed]
alicloud_eip_association.foo: Creation complete after 11s [id=eip-2zew5xxxxxxxx:ngw-2ze6ysuxxxxxxxxxx]
Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
```

iii. 运行 `terraform show` 查看已经创建的 NAT 网关。

## 相关文档

- [创建专有网络 Terraform 示例](#)
- [创建专有网络 Terraform Module](#)

# 4.5. 对象存储OSS

## 4.5.1. 通过Terraform管理Bucket

本文介绍了如何使用Terraform创建一个存储空间，并设置存储空间属性用来控制静态网站托管、日志、生命周期等。

### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey 管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

### 操作步骤

1. 创建一个Bucket。
  - i. 创建`terraform.tf`文件，输入以下内容，并保存在当前执行的目录中。

 **说明** 需要为每个Terraform项目创建一个独立的执行目录。


```
provider "alicloud" {  
  alias   = "bj-prod"  
  region = "cn-beijing"  
}  
  
resource "alicloud_oss_bucket" "bucket-new" {  
  provider = alicloud.bj-prod  
  bucket   = "bucket-20200310-1"  
  acl      = "public-read"  
}
```

- ii. 运行`terraform apply`开始创建。出现类似下面的日志，则表示创建成功。

```
alicloud_oss_bucket.bucket-new: Creating...  
alicloud_oss_bucket.bucket-new: Creation complete after 2s [id=bucket-20200310-1]  
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

2. 设置Bucket属性。

- i. 创建`resource.tf`文件，输入以下内容，并保存在当前执行的目录中。

 说明 该目录下所有\*.tf 文件都会被terraform自动加载。因此，用户可以按照实际用途将配置信息写入不同的文件中。

```
resource "alicloud_oss_bucket" "bucket-attr" {
  provider = alicloud.bj-prod
  bucket   = "bucket-20200310-2"
  # 静态网站的默认首页和404页面
  website {
    index_document = "index.html"
    error_document = "error.html"
  }
  # 访问日志的存储路径
  logging {
    target_bucket = alicloud_oss_bucket.bucket-new.id
    target_prefix = "log/"
  }
  # 文件生命周期规则
  lifecycle_rule {
    id       = "expirationByDays"
    prefix   = "path/expirationByDays"
    enabled  = true
    expiration {
      days = 365
    }
  }
  # 防盗链设置
  referer_config {
    allow_empty = true
    referers    = ["http://www.aliyun.com", "https://www.aliyun.com", "http://?.aliyun.com"]
  }
}
```

- ii. 运行`terraform apply`开始配置Bucket的属性。出现类似下面的日志，说明配置成功。

```
alicloud_oss_bucket.bucket-attr: Creating...
alicloud_oss_bucket.bucket-attr: Creation complete after 2s [id=bucket-20200310-2]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

## 相关文档

- [Terraform管理Bucket代码示例](#)
- [alicloud\\_oss\\_bucket](#)

## 4.5.2. 五分钟入门阿里云Terraform OSS Backend

本文将介绍Terraform的Backend机制及如何使用Terraform OSS Backend。

### Terraform State简介

Terraform State是用来存放基础设施资源及其属性和状态的机制。Terraform State从存储形态上分为两种：

- local：本地存储

资源状态存放在本地的一个state文件中，默认为执行目录下的名为`terraform.tfstate`文件。此方式也是Terraform默认的存储形式。

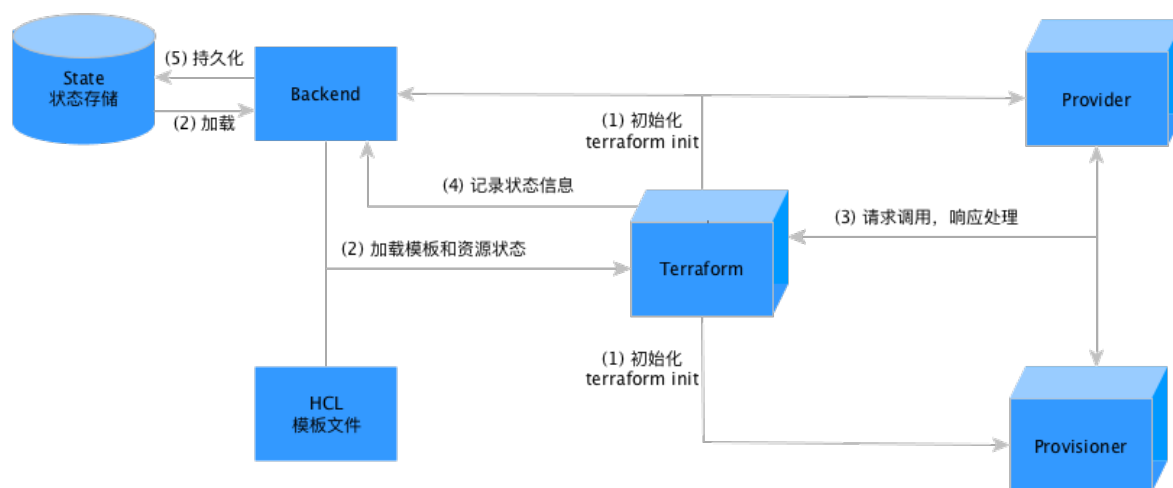
- remote：远端存储

资源状态存放在远端的一个服务中，例如阿里云的OSS Terraform Cloud，HashiCorp Consul等。远端存储带来的好处是实现了与资源定义模板管理的解耦，可以让State脱离本地磁盘而存储，在提升State安全性的同时，团队协作可以不再受制于Terraform的执行环境、执行目录和多人执行时间的限制，提升了管理的灵活性。

Terraform State的存储是由一个称之为Backend的组件决定的，local state使用的是local backend。除了local backend，其他所有的backend在使用之前都需要在模板中显式定义并通过`terraform init`来实现加载和配置。

## Terraform Backend简介

Backend是存储State的机制，它决定了State数据的加载逻辑和Terraform命令执行之后State的数据的更新过程。Terraform生命周期中与Backend的交互过程，如下图所示。



Terraform通过 `init` 命令完成Backend的加载和配置。在执行`plan`和`apply`命令，加载资源模板的同时，通过Backend加载State中的存量数据（如果有），命令结束后，将Provider或Provisioner响应中的数据通过Backend更新到State中，最终达到State数据与模板定义的一致。

从功能实现级别的角度，Backend可以分为两种：

- Standard

State管理的标准化实现，覆盖标准的功能点`State存储`和`State加锁`。这种Backend目前只适用于在本地系统上运行所有操作的场景，尽管也实现了对State的远程存储，但Backend的执行逻辑仍发生在本地并通过直接调用API请求来完成。

目前这种Backend总共有13种，阿里云的OSS Backend也属于此类。

- Enhanced

可以看作是Standard的加强版，即Backend的执行逻辑不仅可以发生在本地，还可以通过API或者CLI的方式发生在远端。目前这种Backend的种类有两种，一是local，另一个是只支持Terraform Cloud的remote。



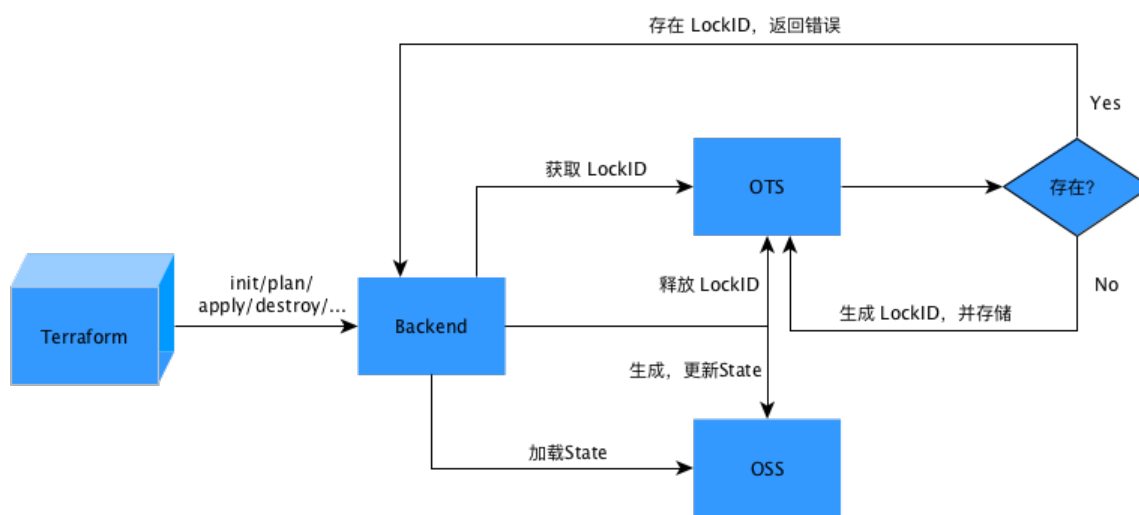
阿里云也提供了一个标准的Backend-oss，在Terraform 0.12.2中予以支持，并在Terraform 0.12.6 和0.12.8版本中对OSS进行了升级，支持profile设置，ecs\_role\_name，assume\_role等鉴权方式。

## 阿里云OSS Backend详解

OSS Backend是基于阿里云的表格存储服务（Tablestore）和对象存储服务（OSS）实现的Standard Backend，其中Tablestore用来存储运行过程中产生的“Locking”，保证State的正确性和完整性；OSS用来存储最终的State文件。接下来将详细介绍OSS Backend。

- 工作原理

OSS Backend的工作流程可以分为加锁、存储State、释放锁三步，下图是一个简单的工作流程图：



主要包含以下几个部分：

- 运行Terraform命令后，Backend首先会从Tablestore中获取LockID，如果已经存在，表明State被损坏或者有人正在操作，返回报错，否则，自动生成一个LockID并存储在Tablestore中。
- 如果是init命令，初次会生成一个新的state文件并存储在OSS的特定目录下，并释放LockID。
- 如果是plan、apply、destroy等涉及到修改State的命令，会在命令结束后将最新的数据同步更新到State文件中，并释放LockID。
- 如果是state、show等不涉及修改的操作，会直接读取State内容并返回。

- 模板定义

和Provider和Provisioner一样，Backend在使用时同样需要在模板中定义。Backend通过关键字backend来声明

如下代码声明了一个oss backend，其state存储在名为 *terraform-oss-backend-1024* 的bucket中，对应的文件为 *prod/terraform.tfstate*，并声明state文件为只读和加密；锁信息存储在一个名为 *terraform-oss-backend-1024* 的表格中，这个表格位于杭州的Tablestore实例 *tf-oss-backend* 中。

```
terraform {
  backend "oss" {
    profile           = "terraform"
    bucket            = "terraform-oss-backend-1024"
    key               = "prod/terraform.tfstate"
    tablestore_endpoint = "https://tf-oss-backend.cn-hangzhou.Tablestore.aliyuncs.com"
    tablestore_table   = "terraform-oss-backend-1024"
    acl               = "private"
    encrypt            = true
    ...
  }
}
```

对backend的定义包含如下几个部分：

- terraform为运行主体，定义了Backend的操作主体。Backend的逻辑实现是存放在Terraform仓库中的，服务于所有Provider和Provisioner，因此它的运行主体是terraform，而不是具体某个Provider。
  - oss 为Backend类型，用来标识一个特定的Backend。
  - 大括号里面的内容为参数配置，用来定义Backend属性，例如鉴权信息，OSS Bucket的名称，存放路径，Tablestore配置信息等。更多参数和含义可参考官方文档：<https://www.terraform.io/docs/backends/types/oss.html>
- 一键生成OSS Backend模板

为了更快捷的编写OSS Backend模板，阿里云提供了一个Terraform Module：[remote-backend](#)

```
module "remote_state" {
  source              = "terraform-alicloud-modules/remote-backend/alicloud"
  create_backend_bucket = true
  create_ots_lock_instance = true
  create_ots_lock_table   = true
  region                = "cn-hangzhou"
  state_name             = "prod/terraform.tfstate"
  encrypt_state          = true
}
```

运行完成后，会在当前目录下生成一个名为`terraform.tf`的配置文件，文件内容即为已经配置好的OSS Backend：

```
terraform {
  backend "oss" {
    bucket            = "terraform-remote-backend-94a22ee-0714-e8ef-8573-21df8b021f86"
    prefix            = "env:"
    key               = "new/terraform.tfstate"
    acl               = "private"
    region            = "cn-hangzhou"
    encrypt            = "true"
    tablestore_endpoint = "https://tf-oss-backend.cn-hangzhou.Tablestore.aliyuncs.com"
    tablestore_table   = "terraform_remote_backend_lock_table_38001042_0714_e8ef_8573_21df8b021f86"
  }
}
```

生成后的`terraform.tf`可以直接跟目标模板放在同一个目录下，以用于后续State的远端存储。

如果要把生成`terraform.tf`的state也存放在上述的OSS Backend中，只需再次运行`terraform init`命令，本地目录下的local state将会自动同步到OSS Backend中。

## 4.6. 访问控制RAM

### 4.6.1. 使用Terraform创建一个RAM用户

本文介绍如何使用Terraform创建一个RAM用户。

#### 方式一：通过Terraform Resource创建

##### 1. 创建RAM用户。

- i. 创建`terraform.tf`文件，输入以下内容，并保存在当前的执行目录中。

```
resource "alicloud_ram_user" "user" {
  name           = "user_test"
  display_name   = "TestAccount"
  mobile         = "86-1868888****"
  email          = "example@example.com"
  comments       = "yoyoyo"
  force          = true
}
```

- ii. 运行`terraform apply`开始创建。

- iii. 运行`terraform show`查看已创建的RAM用户，您也可以登录RAM控制台查看创建的用户。

##### 2. 指定控制台登录密码。

- i. 在`terraform.tf`文件中增加以下内容。

```
resource "alicloud_ram_login_profile" "profile" {
  user_name = alicloud_ram_user.user.name
  password  = "!Test@123456"
}
```

- ii. 运行`terraform apply`开始创建。

- iii. 运行`terraform show`查看账户密码安全策略，您也可以创建的新用户登录RAM控制台查看账户密码安全策略。

##### 3. 创建访问密钥AccessKey。

- i. 在`terraform.tf`文件中增加以下内容。

```
resource "alicloud_ram_access_key" "ak" {
  user_name = alicloud_ram_user.user.name
  secret_file = "accesskey.txt" # 保存AccessKey的文件名
}
```

- ii. 运行`terraform apply`开始创建。

- iii. 打开`accesskey.txt`查看创建的访问密钥。

##### 4. 创建RAM用户组。

- i. 在`terraform.tf`文件中增加以下内容。

```
resource "alicloud_ram_group" "group" {
  name      = "test_ram_group"
  force     = true
}
```

- ii. 运行`terraform apply`开始创建。
  - iii. 运行`terraform show`查看创建的RAM用户组，您也可以登录RAM控制台查看创建的RAM用户组。
5. 将用户加入用户组中。

- i. 在`terraform.tf`文件中增加以下内容。

```
resource "alicloud_ram_group_membership" "membership" {
  group_name = alicloud_ram_group.group.name
  user_names = [alicloud_ram_user.user.name]
}
```

- ii. 运行`terraform apply`开始创建。
- iii. 运行`terraform show`查看RAM用户组下的用户，您也可以登录RAM控制台查看RAM用户组下的用户。

完整代码如下：

```
provider "alicloud" {
}

resource "alicloud_ram_user" "user" {
  name          = "user_test"
  display_name  = "TestAccount"
  mobile        = "86-1868888****"
  email         = "example@example.com"
  comments      = "yoyoyo"
  force         = true
}

resource "alicloud_ram_login_profile" "profile" {
  user_name = alicloud_ram_user.user.name
  password  = "!!Test@123456"
}

resource "alicloud_ram_access_key" "ak" {
  user_name  = alicloud_ram_user.user.name
  secret_file = "accesskey.txt"
}

resource "alicloud_ram_group" "group" {
  name      = "test_ram_group"
  comments  = "this is a group comments."
  force     = true
}

resource "alicloud_ram_group_membership" "membership" {
  group_name = alicloud_ram_group.group.name
  user_names = [alicloud_ram_user.user.name]
}
```

## 方式二：通过Terraform Module一键创建

为了更快捷的创建RAM资源，阿里云提供了Terraform Module: `terraform-alicloud-ram`。简单使用示例如下：

```
module "ram_user" {
  // 引用module源地址
  source = "terraform-alicloud-modules/ram/alicloud"
  // RAM用户名
  name = "terraformtest1"
  // 是否创建控制台登录凭证
  create_ram_user_login_profile = true
  // 控制台登录密码
  password = "User@123"
  // 是否创建accesskey
  create_ram_access_key = true
  // 是否赋予管理员权限
  is_admin = true
}
```

当`create_ram_access_key`为`true`时，会在当前路径下生成文件`secret.txt`存放密钥信息；`is_admin`为`true`时，会自动为用户授予某些管理权限。

## 4.6.2. 使用Terraform创建角色并绑定自定义权限策略

本文介绍如何使用Terraform创建角色并绑定权限策略。

### 操作步骤

1. 创建RAM角色。
  - i. 创建`terraform.tf`文件，输入以下内容，并保存在当前的执行目录中。

```
resource "alicloud_ram_role" "role" {
  name = "testRole"
  document = <<EOF
  {
    "Statement": [
      {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": [
            "apigateway.aliyuncs.com",
            "ecs.aliyuncs.com"
          ]
        }
      }
    ],
    "Version": "1"
  }
  EOF
  description = "this is a role test."
  force       = true
}
```

- ii. 运行**terraform apply**开始创建。
  - iii. 运行**terraform show**查看创建的角色。
2. 创建自定义权限策略。

- i. 在**terraform.tf**文件中增加以下内容。

```
resource "alicloud_ram_policy" "policy" {
  name = "testPolicy"
  document = <<EOF
  {
    "Statement": [
      {
        "Action": [
          "oss:ListObjects",
          "oss:GetObject"
        ],
        "Effect": "Deny",
        "Resource": [
          "acs:oss:*:*:mybucket",
          "acs:oss:*:*:mybucket/*"
        ]
      }
    ],
    "Version": "1"
  }
EOF
  description = "this is a policy test"
  force       = true
}
```

- ii. 运行**terraform apply**开始创建。
  - iii. 运行**terraform show**查看创建的自定义权限策略。
3. 为角色绑定权限策略。

- i. 在**terraform.tf**文件中增加以下内容。

```
resource "alicloud_ram_role_policy_attachment" "attach" {
  policy_name = alicloud_ram_policy.policy.name
  role_name   = alicloud_ram_role.role.name
  policy_type = alicloud_ram_policy.policy.type
}
```

- ii. 运行**terraform apply**开始创建。
- iii. 运行**terraform show**查看角色拥有的自定义权限。

```
provider "alicloud" {
}

resource "alicloud_ram_role" "role" {
  name = "testRole"
  document = <<EOF
    {
      "Statement": [
        {
          "Action": "sts:AssumeRole",
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "apigateway.aliyuncs.com",
              "ecs.aliyuncs.com"
            ]
          }
        }
      ],
      "Version": "1"
    }
  EOF
  description = "this is a role test."
  force       = true
}

resource "alicloud_ram_policy" "policy" {
  name = "testPolicy"
  document = <<EOF
    {
      "Statement": [
        {
          "Action": [
            "oss:ListObjects",
            "oss:GetObject"
          ],
          "Effect": "Deny",
          "Resource": [
            "acs:oss:*:*:mybucket",
            "acs:oss:*:*:mybucket/*"
          ]
        }
      ],
      "Version": "1"
    }
  EOF
  description = "this is a policy test"
  force       = true
}

resource "alicloud_ram_role_policy_attachment" "attach" {
  policy_name = alicloud_ram_policy.policy.name
  role_name   = alicloud_ram_role.role.name
  policy_type = alicloud_ram_policy.policy.type
}
```

## 相关文档

- [权限策略语法和结构](#)
- [alicloud\\_ram\\_role](#)
- [alicloud\\_ram\\_role\\_policy\\_attachment](#)
- [alicloud\\_ram\\_policy](#)

## 4.6.3. 一键创建容器镜像仓库和授权RAM账号

本文介绍如何通过Terraform一键创建命名空间和容器镜像仓库并授权的RAM账号。

### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

### 背景信息

阿里云容器镜像服务（Container Registry）提供安全的应用镜像托管能力，精确的镜像安全扫描功能，稳定的镜像构建服务，便捷的镜像授权功能，方便用户进行镜像全生命周期管理。当我们的DevOps工具需要访问、使用在阿里云创建的容器镜像仓库时，就需要使用阿里云账号授权访问，我们使用Terraform Module（[cr](#)）可以一键创建具有访问目标仓库权限的RAM子账号，精确授权，规避安全风险。

### 操作步骤

1. 编写Terraform脚本代码。

i. 在`main.tf`文件中声明Module，文件内容如下：

```
provider "alicloud" {}

module "cr" {
  source = "roura356a/cr/alicloud"
  version = "1.3.0"
  # 命名空间名称
  namespace = "cr_repo_namespace"
  # 授权仓库列表
  repositories = ["one", "two", "three"]
}
```

ii. 在`outputs.tf`文件中定义输出参数，文件内容如下：



```
output "cr_namespace" {
  description = "The CR Namespace's ID"
  value       = module.cr.cr_namespace
}
output "cr_access_key" {
  description = "The CR Namespace's Access Key"
  value       = module.cr.cr_access_key
}
output "cr_user" {
  description = "The CR Namespace's User"
  value       = module.cr.cr_user
}
output "ram_user" {
  description = "The RAM User"
  value       = module.cr.ram_user
}
output "ram_console_username" {
  description = "Console login username"
  value       = module.cr.ram_console_username
}
output "cr_endpoint" {
  description = "Public endpoint of the registry"
  value       = module.cr.cr_endpoint
}
output "repository_ids" {
  description = "List of repository IDs created"
  value       = module.cr.repository_ids
}
output "disposable_password" {
  description = "Password to activate the console login profile, forces to reset it"
  value       = module.cr.disposable_password
}
output "access_key_status" {
  description = "Status of the created AccessKey"
  value       = module.cr.access_key_status
}
output "ram_policy_name" {
  description = "The RAM policy name"
  value       = module.cr.ram_policy_name
}
output "ram_policy_type" {
  description = "The RAM policy type"
  value       = module.cr.ram_policy_type
}
output "ram_policy_attachment" {
  description = "The RAM policy attachment ID"
  value       = module.cr.ram_policy_attachment
}
```

## 2. 运行terraform init初始化。

```
terraform init
```

命令输出结果类似如下：

```
Initializing modules...
Downloading roura356a/cr/alicloud 1.3.0 for cr...
- cr in .terraform\modules\cr\roura356a-terraform-alicloud-cr-c60a3d4
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "alicloud" (hashicorp/alicloud) 1.68.0...
- Downloading plugin for provider "random" (hashicorp/random) 2.2.1...
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.random: version = "~> 2.2"
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.
```

### 3. 运行terraform apply开始创建。

```
terraform apply
```

命令输出结果类似如下：

```

module.cr.data.alicloud_account.current: Refreshing state...
module.cr.data.alicloud_regions.current: 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:
...
Plan: 10 to add, 0 to change, 0 to destroy.
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
module.cr.random_string.cr_console_password: Creating...
...
Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
Outputs:
access_key_status = Active
cr_access_key = LTAI4FfqhU7csppPe*****
cr_endpoint = registry.cn-hangzhou.aliyuncs.com
cr_namespace = cr_repo_namespace
cr_user = cr_repo_namespace-cr-user
disposable_password = erlPQu*****
ram_console_username = cr_repo_namespace-cr-user@1231579085*****.onaliyun.com
ram_policy_attachment = user:cr_repo_namespace-cr-policy:Custom:cr_repo_namespace-cr-user
ram_policy_name = cr_repo_namespace-cr-policy
ram_policy_type = Custom
ram_user = cr_repo_namespace-cr-user
repository_ids = [
  "cr_repo_namespace/one",
  "cr_repo_namespace/two",
  "cr_repo_namespace/three",
]

```

同时，会在执行目录下生成文件`cr-cr_repo_namespace-ak.json`，该文件存储了创建的具有访问目标仓库权限的RAM子账号的密钥信息，文件内容如下：

```

{
  "AccessKeySecret": "qkxnlAkG6B50*****sneyCQDuurcW",
  "CreateDate": "2020-01-07T07:00:00Z",
  "Status": "Active",
  "AccessKeyId": "LTAI4Ff*****ppPeLRkJHES"
}

```

## 相关文档

- [terraform-alicloud-cr\\_example](#)

# 4.7. 容器服务Kubernetes版

## 4.7.1. 使用Terraform创建托管版Kubernetes

在容器服务控制台，我们为您提供了便捷使用的可视界面，一步一步引导式地创建该类型集群。但当您需要反复创建托管版集群、大批量创建集群，使用控制台操作就显得繁琐了，使用Terraform将会帮您解决这些问题。本文将介绍如何使用Terraform快速部署一个托管版的Kubernetes集群。

## 创建托管版 Kubernetes 集群

在阿里云托管版Kubernetes Terraform资源文档 [alicloud\\_cs\\_managed\\_kubernetes](#)中，可以看到该资源提供的参数列表。参数分为入参Argument和出参Attributes。入参列表内包含了必填参数以及可选参数，例如name和name\_prefix就是一对必填参数，但它们互斥，即不能同时填写。如果填了name，集群名就是name的值，如果填了name\_prefix，集群名会以name\_prefix开头自动生成一个。

1. 对照文档中的入参列表Argument Reference，先编写出一个集群的描述，代码如下：

```
provider "alicloud" {  
}  
  
# 默认资源名称  
variable "name" {  
  default = "my-first-kubernetes-demo"  
}  
  
# 日志服务项目名称  
variable "log_project_name" {  
  default = "my-first-kubernetes-sls-demo"  
}  
  
# 可用区  
data "alicloud_zones" default {  
  available_resource_creation = "VSwitch"  
}  
  
# 节点ECS实例配置  
data "alicloud_instance_types" "default" {  
  availability_zone = data.alicloud_zones.default.zones[0].id  
  cpu_core_count   = 2  
  memory_size      = 4  
  kubernetes_node_role = "Worker"  
}  
  
# 专有网络  
resource "alicloud_vpc" "default" {  
  name      = var.name  
  cidr_block = "10.1.0.0/21"  
}  
  
# 交换机  
resource "alicloud_vswitch" "default" {  
  name          = var.name  
  vpc_id        = alicloud_vpc.default.id  
  cidr_block    = "10.1.1.0/24"  
  availability_zone = data.alicloud_zones.default.zones[0].id  
}  
  
# 日志服务  
resource "alicloud_log_project" "log" {  
  name          = var.log_project_name  
  description = "created by terraform for managedkubernetes cluster"  
}  
  
# kubernetes托管版  
resource "alicloud_cs_managed_kubernetes" "default" {  
  # kubernetes集群名称的前缀。与name冲突。如果指定，terraform将使用它来构建唯一的集群名称。默认为“ Terraform-Creation”。  
}
```

```

name_prefix          = var.name
# 新的kubernetes集群将位于的区域。
availability_zone    = data.alicloud_zones.default.zones[0].id
# 新的kubernetes集群将位于的vswitch。指定一个或多个vswitch的ID。它必须在availability_zone指定的区域中
vswitch_ids          = [alicloud_vswitch.default.id]
# 是否在创建kubernetes集群时创建新的nat网关。默认为true。
new_nat_gateway      = true
# 节点的ECS实例类型。为单个AZ集群指定一种类型，为MultiAZ集群指定三种类型。您可以通过数据源instance_types获得可用的kubernetes主节点实例类型
worker_instance_types = [data.alicloud_instance_types.default.instance_types[0].id]
# kubernetes集群的总工作节点数。默认值为3。最大限制为50。
worker_number        = 2
# ssh登录集群节点的密码。您必须指定password或key_name kms_encrypted_password字段。
password             = "Yourpassword1234"
# pod网络的CIDR块。当cluster_network_type设置为flannel，你必须设定该参数。它不能与VPC CIDR相同，并且不能与VPC中的Kubernetes集群使用的CIDR相同，也不能在创建后进行修改。
pod_cidr             = "172.20.0.0/16"
# 服务网络的CIDR块。它不能与VPC CIDR相同，不能与VPC中的Kubernetes集群使用的CIDR相同，也不能在创建后进行修改。
service_cidr         = "172.21.0.0/20"
# 是否为kubernetes的节点安装云监控。
install_cloud_monitor = true
# 是否为API Server创建Internet负载均衡。默认为false。
slb_internet_enabled = true
# 节点的系统磁盘类别。其有效值为cloud_ssd和cloud_efficiency。默认为cloud_efficiency。
worker_disk_category = "cloud_efficiency"
# 节点的数据磁盘类别。其有效值为cloud_ssd和cloud_efficiency，如果未设置，将不会创建数据磁盘。
worker_data_disk_category = "cloud_ssd"
# 节点的数据磁盘大小。有效值范围[20~32768]，以GB为单位。当worker_data_disk_category被呈现，则默认为40。
worker_data_disk_size = 200
# 日志配置
log_config {
  # 收集日志的类型，目前仅支持SLS。
  type = "SLS"
  # 日志服务项目名称，集群日志将输出到该项目
  project = alicloud_log_project.log.name
}
}

```

2. 将以上的配置保存为一个`main.tf`描述文件，在该文件的当前目录下执行`terraform init`和`terraform apply`。

### i. 执行terraform init命令初始化。

```
$ terraform init
Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "alicloud" (1.26.0)...
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.
```

### ii. 执行terraform apply命令创建资源。

```
$ terraform apply
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:
# alicloud_cs_managed_kubernetes.default will be created
+ resource "alicloud_cs_managed_kubernetes" "default" {
  + availability_zone      = "cn-hangzhou-b"
  + force_update           = false
  + id                     = (known after apply)
  + install_cloud_monitor = true
  + name                   = (known after apply)
  + name_prefix            = "my-first-kubernetes-demo"
  + new_nat_gateway        = true
  + password               = (sensitive value)
  + pod_cidr               = "172.20.0.0/16"
  + security_group_id      = (known after apply)
  + service_cidr           = "172.21.0.0/20"
  + slb_internet_enabled   = true
  + vpc_id                 = (known after apply)
  + vswitch_ids            = (known after apply)
  + worker_data_disk_category = "cloud_ssd"
  + worker_data_disk_size   = 200
  + worker_disk_category    = "cloud_efficiency"
  + worker_disk_size        = 40
  + worker_instance_charge_type = "PostPaid"
  + worker_instance_types    = [
    + "ecs.n1.medium",
  ]
  + worker_nodes             = (known after apply)
  + worker_number            = 2
  + log_config {
    + project = "my-first-kubernetes-sls-demo"
    + type    = "SLS"
  }
}
# alicloud_log_project.log will be created
+ resource "alicloud_log_project" "log" {
  + description = "created by terraform for managedkubernetes cluster"
  + id          = (known after apply)
  + name        = "my-first-kubernetes-sls-demo"
}
```

```
# alicloud_vpc.default will be created
+ resource "alicloud_vpc" "default" {
  + cidr_block      = "10.1.0.0/21"
  + id              = (known after apply)
  + name            = "my-first-kubernetes-demo"
  + resource_group_id = (known after apply)
  + route_table_id  = (known after apply)
  + router_id        = (known after apply)
  + router_table_id  = (known after apply)
}
# alicloud_vswitch.default will be created
+ resource "alicloud_vswitch" "default" {
  + availability_zone = "cn-hangzhou-b"
  + cidr_block        = "10.1.1.0/24"
  + id                = (known after apply)
  + name              = "my-first-kubernetes-demo"
  + vpc_id            = (known after apply)
}
Plan: 4 to add, 0 to change, 0 to destroy.
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
Enter a value:
```

3. **terraform init** 命令会把我们用到的Provider插件下载好，**terraform apply**命令会根据我们的`main.tf`描述文件计算出需要执行的操作。上述日志中显示将会创建一个`alicloud_cs_managed_kubernetes.k8s`的资源，需要我们输入`yes`来确认创建。确认创建后，创建大约会耗时五分钟，**terraform**会输出类似下面的日志。

```
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
alicloud_vpc.default: Creating...
alicloud_log_project.log: Creating...
alicloud_log_project.log: Creation complete after 1s [id=my-first-kubernetes-sls-demo]
alicloud_vpc.default: Creation complete after 6s [id=vpc-bp1830x557ktabq*****]
alicloud_vswitch.default: Creating...
alicloud_vswitch.default: Creation complete after 5s [id=vsw-bp1vb35pc7bvc0e*****]
alicloud_cs_managed_kubernetes.default: Creating...
alicloud_cs_managed_kubernetes.default: Still creating... [10s elapsed]
alicloud_cs_managed_kubernetes.default: Still creating... [20s elapsed]
alicloud_cs_managed_kubernetes.default: Still creating... [30s elapsed]
alicloud_cs_managed_kubernetes.default: Still creating... [40s elapsed]
alicloud_cs_managed_kubernetes.default: Still creating... [50s elapsed]
.....
alicloud_cs_managed_kubernetes.k8s: Creation complete after 6m5s (ID: cc54df7d990a24ed18c1e0ebacd36418c)
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.
```

4. 当出现 `Apply complete! Resources: 4 added` 字样的时候，集群已经成功创建，此时我们也可以登录控制台在集群列表中查看此集群。



## 修改托管版Kubernetes集群

在Terraform Provider中，我们提供了一部分参数的修改能力，一般情况下，所有非Force New Resource（强制新建资源）的参数都可以被修改。

1. 下面我们修改部分参数，注释内容为更新的项目。

```
resource "alicloud_cs_managed_kubernetes" "default" {
  # 更换集群的名称为 test-managed-kubernetes-updated
  name = "test-managed-kubernetes-updated"
  availability_zone = data.alicloud_zones.default.zones[0].id
  vswitch_ids      = [alicloud_vswitch.default.id]
  new_nat_gateway  = true
  worker_instance_types = [data.alicloud_instance_types.default.instance_types[0].id]
  # 修改 worker_numbers 为 3，可以扩容一个 worker 节点
  worker_number = 3
  password      = "Yourpassword1234"
  pod_cidr      = "172.20.0.0/16"
  service_cidr  = "172.21.0.0/20"
  install_cloud_monitor = true
  slb_internet_enabled = true
  worker_disk_category = "cloud_efficiency"
  worker_data_disk_category = "cloud_ssd"
  worker_data_disk_size = 200
  log_config {
    type = "SLS"
    project = alicloud_log_project.log.name
  }
  # 导出集群的连接配置文件到 /tmp 目录
  kube_config = "/tmp/config"
  # 导出集群的证书相关文件到 /tmp 目录，下同
  client_cert = "/tmp/client-cert.pem"
  client_key = "/tmp/client-key.pem"
  cluster_ca_cert = "/tmp/cluster-ca-cert.pem"
}
```

2. 和创建集群一样，修改集群时使用的命令也是**terraform apply**。执行后我们得到以下日志输出，输入yes并回车，我们就可以把该集群的名称改为test-managed-kubernetes-updated，worker节点扩容至3节点，同时将导出证书和连接文件到本机的/tmp目录。

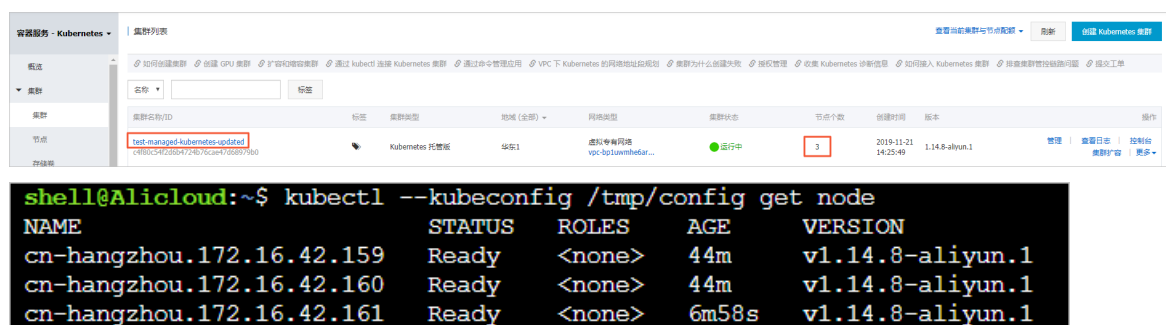


```

$ terraform apply
alicloud_cs_managed_kubernetes.k8s: Refreshing state... (ID: cc54df7d990a24ed18c1e0ebacd36418c)
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  ~ update in-place
Terraform will perform the following actions:
  ~ alicloud_cs_managed_kubernetes.k8s
    client_cert:      "" => "/tmp/client-cert.pem"
    client_key:       "" => "/tmp/client-key.pem"
    cluster_ca_cert:  "" => "/tmp/cluster-ca-cert.pem"
    kube_config:      "" => "/tmp/config"
    name:             "test-managed-kubernetes" => "test-managed-kubernetes-updated"
    worker_numbers.0: "2" => "3"
Plan: 0 to add, 1 to change, 0 to destroy.
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
alicloud_cs_managed_kubernetes.k8s: Modifying... (ID: cc54df7d990a24ed18c1e0ebacd36418c)
  client_cert:      "" => "/tmp/client-cert.pem"
  client_key:       "" => "/tmp/client-key.pem"
  cluster_ca_cert:  "" => "/tmp/cluster-ca-cert.pem"
  kube_config:      "" => "/tmp/config"
  name:             "test-managed-kubernetes" => "test-managed-kubernetes-updated"
  worker_numbers.0: "2" => "3"
alicloud_cs_managed_kubernetes.k8s: Still modifying... (ID: cc54df7d990a24ed18c1e0ebacd36418c, 10s elapsed)
alicloud_cs_managed_kubernetes.k8s: Still modifying... (ID: cc54df7d990a24ed18c1e0ebacd36418c, 20s elapsed)
alicloud_cs_managed_kubernetes.k8s: Still modifying... (ID: cc54df7d990a24ed18c1e0ebacd36418c, 30s elapsed)
.....
alicloud_cs_managed_kubernetes.k8s: Modifications complete after 4m4s (ID: cc54df7d990a24ed18c1e0ebacd36418c)
Apply complete! Resources: 0 added, 1 changed, 0 destroyed.

```

3. Terraform apply运行成功后，控制台中显示的集群信息已经表明现在集群已经变成了我们期望的状态。在本机上，我们也通过导出的连接文件，用kubectl连接到集群。



The screenshot shows the Alibaba Cloud console interface for managing Kubernetes clusters. The cluster 'test-managed-kubernetes-updated' is listed with a status of 'Running' and 3 nodes. Below the console, a terminal window shows the command 'kubectl get node' and its output, which lists three nodes in a 'Ready' state.

NAME	STATUS	ROLES	AGE	VERSION
cn-hangzhou.172.16.42.159	Ready	<none>	44m	v1.14.8-aliyun.1
cn-hangzhou.172.16.42.160	Ready	<none>	44m	v1.14.8-aliyun.1
cn-hangzhou.172.16.42.161	Ready	<none>	6m58s	v1.14.8-aliyun.1

## 相关文档

- 创建Kubernetes托管版集群

- [云生态下的基础架构资源管理利器Terraform](#)
- [阿里云 Terraform Provider 代码库](#)
- [阿里云 Terraform Provider 文档](#)
- [阿里云 Terraform Provider 文档 - 托管版 Kubernetes](#)
- [Kubernetes集群网络规划](#)

## 4.7.2. Terraform部署容器服务Kubernetes集群及WordPress应用

本教程介绍了如何通过Terraform在VPC环境下部署一个阿里云容器服务Kubernetes集群，并在该集群之上，部署一个WordPress样例应用。

### 前提条件

在使用本教程之前，请确保完成以下准备工作：

- 请确保您已开通阿里云容器服务，参见[容器服务Kubernetes版](#)。
- 使用Terraform部署容器服务Kubernetes集群及WordPress应用，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 安装Terraform，请参见[在本地安装和配置Terraform](#)。

### 步骤一：下载容器服务Kubernetes的Terraform模板

您可以从GitHub上下载创建Kubernetes集群的Terraform模板（[模板下载地址](#)），模板中包含以下文件：

- *main.tf*

Terraform主文件。定义了将要部署的资源。本模版加入了条件判断，可实现对已有网络资源的引用和多个Kubernetes集群的同时创建。该文件定义了以下资源：

- 可用地域

定义了资源将要被创建在哪个地域里。

```
provider "alicloud" {
  access_key = "${var.alicloud_access_key}"
  secret_key = "${var.alicloud_secret_key}"
  region = "${var.region}"
}

data "alicloud_zones" "default" {
  available_instance_type = data.alicloud_instance_types.default.instance_types[0].id
}
```

- 实例规格

```
data "alicloud_instance_types" "default" {
  cpu_core_count = var.cpu_core_count
  memory_size    = var.memory_size
}
```

### ◦ 专有网络

指定vpc\_id可使用已有VPC。

```
resource "alicloud_vpc" "vpc" {
  count      = var.vpc_id == "" ? 1 : 0
  cidr_block = var.vpc_cidr
  name      = var.vpc_name == "" ? var.example_name : var.vpc_name
}
```

### ◦ 交换机

指定vswitch\_ids可使用已有交换机。

```
resource "alicloud_vswitch" "vswitches" {
  count      = length(var.vswitch_ids) > 0 ? 0 : length(var.vswitch_cidrs)
  vpc_id     = var.vpc_id == "" ? join("", alicloud_vpc.vpc.*.id) : var.vpc_id
  cidr_block = element(var.vswitch_cidrs, count.index)
  availability_zone = data.alicloud_zones.default.zones[count.index % length(data.alicloud_zones.default.zones)][0].id
  name      = var.vswitch_name_prefix == "" ? format(
    "%s-%s",
    var.example_name,
    format(var.number_format, count.index + 1),
  ) : format(
    "%s-%s",
    var.vswitch_name_prefix,
    format(var.number_format, count.index + 1),
  )
}
```

### ◦ NAT网关

指定new\_nat\_gateway来决定是否要为模板中定义的 VPC 自动创建NAT网关，以保证Kubernetes集群成功创建。

```
resource "alicloud_nat_gateway" "default" {
  count = var.new_nat_gateway == "true" ? 1 : 0
  vpc_id = var.vpc_id == "" ? join("", alicloud_vpc.vpc.*.id) : var.vpc_id
  name   = var.example_name
}
```

### ◦ 弹性网卡

```
resource "alicloud_eip" "default" {
  count      = var.new_nat_gateway == "true" ? 1 : 0
  bandwidth = 10
}
```

### ◦ 绑定弹性网卡

```
resource "alicloud_eip_association" "default" {
  count      = var.new_nat_gateway == "true" ? 1 : 0
  allocation_id = alicloud_eip.default[0].id
  instance_id   = alicloud_nat_gateway.default[0].id
}
```

- 添加SNAT条目


在模板中定义的NAT网关下自动添加SNAT条目来保证Kubernetes集群成功创建。

```
resource "alicloud_snat_entry" "default" {
  count          = var.new_nat_gateway == "false" ? 0 : length(var.vswitch_ids) > 0 ? length(var.vswitch_ids) : length(var.vswitch_cidrs)
  snat_table_id = alicloud_nat_gateway.default[0].snat_table_ids
  source_vswitch_id = length(var.vswitch_ids) > 0 ? split(",", join(",", var.vswitch_ids))[count.index % length(split(",", join(",", var.vswitch_ids)))] : length(var.vswitch_cidrs) < 1 ? "" : split(",", join(",", alicloud_vswitch.vswitches.*.id))[count.index % length(split(",", join(",", alicloud_vswitch.vswitches.*.id)))]
  snat_ip        = alicloud_eip.default[0].ip_address
}
```

## ◦ 容器服务Kubernetes集群

改变k8s\_number的值可同时创建多个Kubernetes集群。

```
resource "alicloud_cs_kubernetes" "k8s" {
  count = var.k8s_number
  name = var.k8s_name_prefix == "" ? format(
    "%s-%s",
    var.example_name,
    format(var.number_format, count.index + 1),
  ) : format(
    "%s-%s",
    var.k8s_name_prefix,
    format(var.number_format, count.index + 1),
  )
  vswitch_ids = [length(var.vswitch_ids) > 0 ? split(",", join(",", var.vswitch_ids))[count.index % length(split(",", join(",", var.vswitch_ids)))] : length(var.vswitch_cidrs) < 1 ? "" : split(",", join(",", alicloud_vswitch.vswitches.*.id))[count.index % length(split(",", join(",", alicloud_vswitch.vswitches.*.id)))]
  new_nat_gateway = false
  master_instance_types = [var.master_instance_type == "" ? data.alicloud_instance_types.default.instance_types[0].id : var.master_instance_type]
  worker_instance_types = [var.worker_instance_type == "" ? data.alicloud_instance_types.default.instance_types[0].id : var.worker_instance_type]
  worker_numbers = [var.k8s_worker_number]
  master_disk_category = var.master_disk_category
  worker_disk_category = var.worker_disk_category
  master_disk_size = var.master_disk_size
  worker_disk_size = var.worker_disk_size
  password = var.ecs_password
  pod_cidr = var.k8s_pod_cidr
  service_cidr = var.k8s_service_cidr
  enable_ssh = true
  install_cloud_monitor = true
  depends_on = [alicloud_snat_entry.default]
}
```

 **说明** 指定 `kube_config = "~/.kube/config"` 可在Kubernetes集群创建完成后将Kube Config内容自动下载并存放在文件 `~/.kube/config`中。

## • outputs.tf

该文件定义了输出参数。作为执行的一部分而创建的资源会生成这些输出参数。和ROS模板指定的输出参数类似。例如，该模板将部署一个Kubernetes集群。以下输出参数将提供集群ID和其他资源参数。

```
// Output VPC
output "vpc_id" {
  description = "The ID of the VPC."
  value       = alicloud_cs_kubernetes.k8s[0].vpc_id
}
output "vswitch_ids" {
  description = "List ID of the VSwitches."
  value       = [alicloud_cs_kubernetes.k8s.*.vswitch_ids]
}
output "nat_gateway_id" {
  value = alicloud_cs_kubernetes.k8s[0].nat_gateway_id
}
// Output kubernetes resource
output "cluster_id" {
  description = "ID of the kubernetes cluster."
  value       = [alicloud_cs_kubernetes.k8s.*.id]
}
output "security_group_id" {
  description = "ID of the Security Group used to deploy kubernetes cluster."
  value       = alicloud_cs_kubernetes.k8s[0].security_group_id
}
output "worker_nodes" {
  description = "List worker nodes of cluster."
  value       = [alicloud_cs_kubernetes.k8s.*.worker_nodes]
}
output "master_nodes" {
  description = "List master nodes of cluster."
  value       = [alicloud_cs_kubernetes.k8s.*.master_nodes]
}
```

- *variables.tf*

该文件包含可传递到 *main.tf* 的变量，可帮助您自定义环境。

```
# common variables
variable "availability_zone" {
  description = "The available zone to launch ecs instance and other resources."
  default     = ""
}
variable "number_format" {
  description = "The number format used to output."
  default     = "%02d"
}
variable "example_name" {
  default = "tf-example-kubernetes"
}
# Instance types variables
variable "cpu_core_count" {
  description = "CPU core count is used to fetch instance types."
  default     = 2
}
variable "memory_size" {
  description = "Memory size used to fetch instance types."
  default     = 4
}
```

```
,
# VPC variables
variable "vpc_name" {
  description = "The vpc name used to create a new vpc when 'vpc_id' is not specified. De
fault to variable `example_name`"
  default     = ""
}
variable "vpc_id" {
  description = "A existing vpc id used to create several vswitches and other resources."
  default     = ""
}
variable "vpc_cidr" {
  description = "The cidr block used to launch a new vpc when 'vpc_id' is not specified."
  default     = "10.1.0.0/21"
}
# VSwitch variables
variable "vswitch_name_prefix" {
  description = "The vswitch name prefix used to create several new vswitches. Default to
variable `example_name`"
  default     = ""
}
variable "vswitch_ids" {
  description = "List of existing vswitch id."
  type        = list(string)
  default     = []
}
variable "vswitch_cidrs" {
  description = "List of cidr blocks used to create several new vswitches when 'vswitch_i
ds' is not specified."
  type        = list(string)
  default     = ["10.1.2.0/24"]
}
variable "new_nat_gateway" {
  description = "Whether to create a new nat gateway. In this template, a new nat gateway
will create a nat gateway, eip and server snat entries."
  default     = "true"
}
# Cluster nodes variables
variable "master_instance_type" {
  description = "The ecs instance type used to launch master nodes. Default from instance
types datasource."
  default     = ""
}
variable "worker_instance_type" {
  description = "The ecs instance type used to launch worker nodes. Default from instance
types datasource."
  default     = ""
}
variable "master_disk_category" {
  description = "The system disk category used to launch one or more master nodes."
  default     = "cloud_efficiency"
}
variable "worker_disk_category" {
  description = "The system disk category used to launch one or more worker nodes."
  default     = "cloud_efficiency"
```

```
}  
variable "master_disk_size" {  
  description = "The system disk size used to launch one or more master nodes."  
  default     = "40"  
}  
variable "worker_disk_size" {  
  description = "The system disk size used to launch one or more worker nodes."  
  default     = "40"  
}  
variable "ecs_password" {  
  description = "The password of instance."  
  default     = "Abc12345"  
}  
variable "k8s_number" {  
  description = "The number of kubernetes cluster."  
  default     = 1  
}  
variable "k8s_worker_number" {  
  description = "The number of worker nodes in each kubernetes cluster."  
  default     = 3  
}  
variable "k8s_name_prefix" {  
  description = "The name prefix used to create several kubernetes clusters. Default to variable `example_name`"  
  default     = ""  
}  
variable "k8s_pod_cidr" {  
  description = "The kubernetes pod cidr block. It cannot be equals to vpc's or vswitch's and cannot be in them."  
  default     = "172.20.0.0/16"  
}  
variable "k8s_service_cidr" {  
  description = "The kubernetes service cidr block. It cannot be equals to vpc's or vswitch's or pod's and cannot be in them."  
  default     = "172.21.0.0/20"  
}
```

## 步骤二：执行Kubernetes Terraform脚本

1. 在存放以上文件的路径，运行`terraform init`命令，初始化工作空间。



```
$ terraform init
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "alicloud" (hashicorp/alicloud) 1.62.0...
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.alicloud: version = "~> 1.62"
Terraform has been successfully initialized!
```

## 2. 运行terraform apply 命令，开始创建Kubernetes集群。

```
$ terraform apply
data.alicloud_instance_types.default: Refreshing state...
data.alicloud_zones.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:
...
Plan: 7 to add, 0 to change, 0 to destroy.
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
alicloud_vpc.vpc: Creating...
...
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
Outputs:
cluster_id = [
    c0f2e04c77e234*****
]
.....
vswitch_ids = [
    vsw-bp1c3hfcd6l80izqc3tbx
]
```

terraform apply命令执行完毕后，输出集群ID和其他参数。除此之外，将Kubernetes的Kube Config文件存放在了目录 `~/.kube` 下。

您现在可以在容器服务控制台查看通过terraform创建的Kubernetes集群，查看集群、节点、日志和容器等信息。

## 步骤三：下载WordPress的Terraform模板

在创建好Kubernetes并完成了Kube Config的下载后，接下来就可以在Kubernetes上部署WordPress。您可以从GitHub上下载创建WordPress的Terraform模板（[模板下载地址](#)，模板中定义了创建WordPress的相关资源和配置，帮助您完成在Kubernetes集群的快速搭建WordPress。更多Terraform Kubernetes的操作可参考Terraform官网的 [Kubernetes 文档介绍](#)。

模板中包含以下文件：

- *localvolumes.tf*

定义存储MySQL持久化数据的Persistent Volume。

```
resource "kubernetes_persistent_volume" "mysql" {
  metadata {
    name = "local-pv-mysql"
    labels {
      type = "local"
    }
  }
  spec {
    capacity {
      storage = "20Gi"
    }
    access_modes = ["ReadWriteOnce"]
    persistent_volume_source {
      host_path {
        path = "/tmp/data/pv-mysql"
      }
    }
  }
}
```

- *mysql.tf*

创建MySQL密码凭证Secret，并部署MySQL。

- secret

```
resource "kubernetes_secret" "mysql" {
  metadata {
    name = "mysql-pass"
  }
  data {
    password = "${var.mysql_password}"
  }
}
```

- Deployment

```
resource "kubernetes_service" "mysql" {
  metadata {
    name = "wordpress-mysql"
    labels {
      app = "wordpress"
    }
  }
  spec {
    port {
```

```
    port = 3306
  }
  selector {
    app = "wordpress"
    tier = "${kubernetes_replication_controller.mysql.spec.0.selector.tier}"
  }
  cluster_ip = "None"
}
}
resource "kubernetes_replication_controller" "mysql" {
  metadata {
    name = "wordpress-mysql"
    labels {
      app = "wordpress"
    }
  }
  spec {
    selector {
      app = "wordpress"
      tier = "mysql"
    }
    template {
      container {
        image = "mysql:${var.mysql_version}"
        name  = "mysql"
        env {
          name = "MYSQL_ROOT_PASSWORD"
          value_from {
            secret_key_ref {
              name = "${kubernetes_secret.mysql.metadata.0.name}"
              key  = "password"
            }
          }
        }
      }
      port {
        container_port = 3306
        name            = "mysql"
      }
      volume_mount {
        name      = "mysql-persistent-storage"
        mount_path = "/var/lib/mysql"
      }
    }
    volume {
      name = "mysql-persistent-storage"
      persistent_volume_claim {
        claim_name = "${kubernetes_persistent_volume_claim.mysql.metadata.0.name}"
      }
    }
  }
}
```

- *wordpress.tf*

部署 Wordpress。

```
resource "kubernetes_service" "wordpress" {
  metadata {
    name = "wordpress"
    labels {
      app = "wordpress"
    }
  }
  spec {
    port {
      port = 80
    }
    selector {
      app = "wordpress"
      tier = "${kubernetes_replication_controller.wordpress.spec.0.selector.tier}"
    }
    type = "LoadBalancer"
  }
}

resource "kubernetes_replication_controller" "wordpress" {
  metadata {
    name = "wordpress"
    labels {
      app = "wordpress"
    }
  }
  spec {
    selector {
      app = "wordpress"
      tier = "frontend"
    }
    template {
      container {
        image = "wordpress:${var.wordpress_version}-apache"
        name  = "wordpress"
        env {
          name = "WORDPRESS_DB_HOST"
          value = "wordpress-mysql"
        }
        env {
          name = "WORDPRESS_DB_PASSWORD"
          value_from {
            secret_key_ref {
              name = "${kubernetes_secret.mysql.metadata.0.name}"
              key  = "password"
            }
          }
        }
      }
      port {
        container_port = 80
        name            = "wordpress"
      }
    }
  }
}
```

```
    volume_mount {
      name = "wordpress-persistent-storage"
      mount_path = "/var/www/html"
    }
  }
  volume {
    name = "wordpress-persistent-storage"
    persistent_volume_claim {
      claim_name = "${kubernetes_persistent_volume_claim.wordpress.metadata.0.name}"
    }
  }
}
}
```

- *outputs.tf*

该文件定义了输出参数。输出Loadbalancer Public IP，借助该IP地址可直接访问部署好的WordPress应用。

```
output "slb_ip" {
  value = "${kubernetes_service.wordpress.load_balancer_ingress.0.ip}"
}
```

- *variables.tf*

该文件包含了部署MySQL和WordPress所依赖的参数。

```
variable "wordpress_version" {
  description = "The version of wordpress. Default to 4.7.3."
  default = "4.7.3"
}
variable "mysql_password" {
  description = "Please input mysql password."
}
variable "mysql_version" {
  description = "The version of mysql which wordpress used. Default to 5.6."
  default = "5.6"
}
```

## 步骤四：执行WordPress Terraform脚本

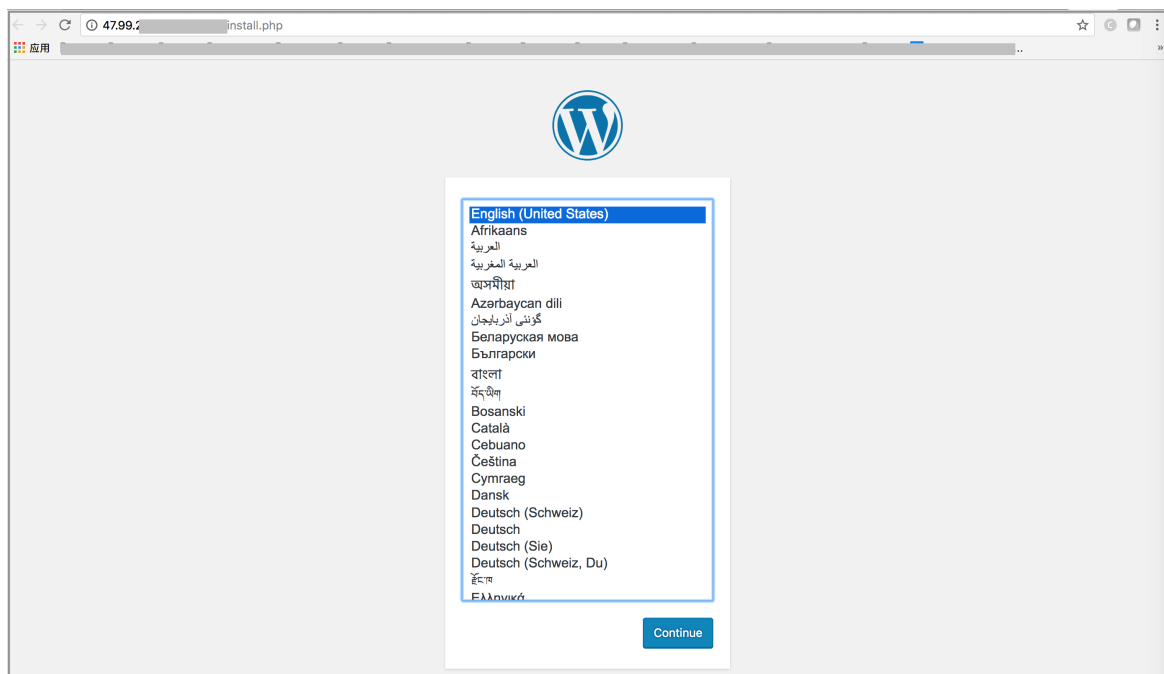
首先定位到您存放以上文件的目录，如 `/root/terraform/kubernetes-wordpress`。运行 **terraform apply** 命令，开始在创建好的Kubernetes集群上部署MySQL和WordPress应用。值得注意的是，由于变量 `mysql_password` 在变量文件中没有定义默认值，因此在执行命令时需要指定该参数值。

```
$ terraform apply -var 'mysql_password=Abc1234'
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:
...
Plan: 9 to add, 0 to change, 0 to destroy.
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
kubernetes_secret.mysql: Creating...
data.%:      "" => "1"
data.password: "<sensitive>" => "<sensitive>"
metadata.#:   "" => "1"
metadata.0.generation: "" => "<computed>"
metadata.0.name:      "" => "mysql-pass"
.....
Apply complete! Resources: 9 added, 0 changed, 0 destroyed.
Outputs:
slb_ip = 47.99.xx.xx
```

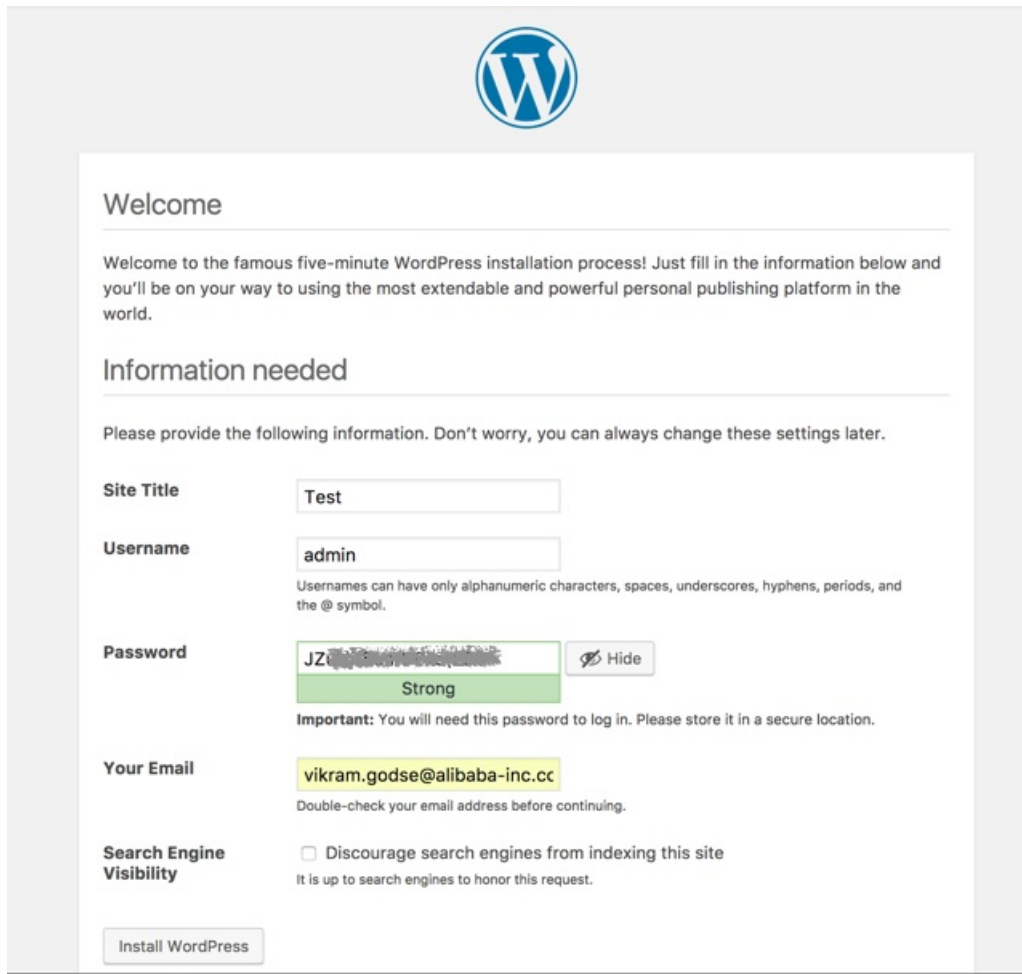
## 步骤五：访问WordPress

根据负载均衡Public IP，在浏览器中输入IP地址即可实现对部署好的WordPress直接访问：

1. 进入WordPress欢迎页面，选择语言，然后继续配置。



2. 输入站点名称以及管理员的用户名和密码。选择安装WordPress。

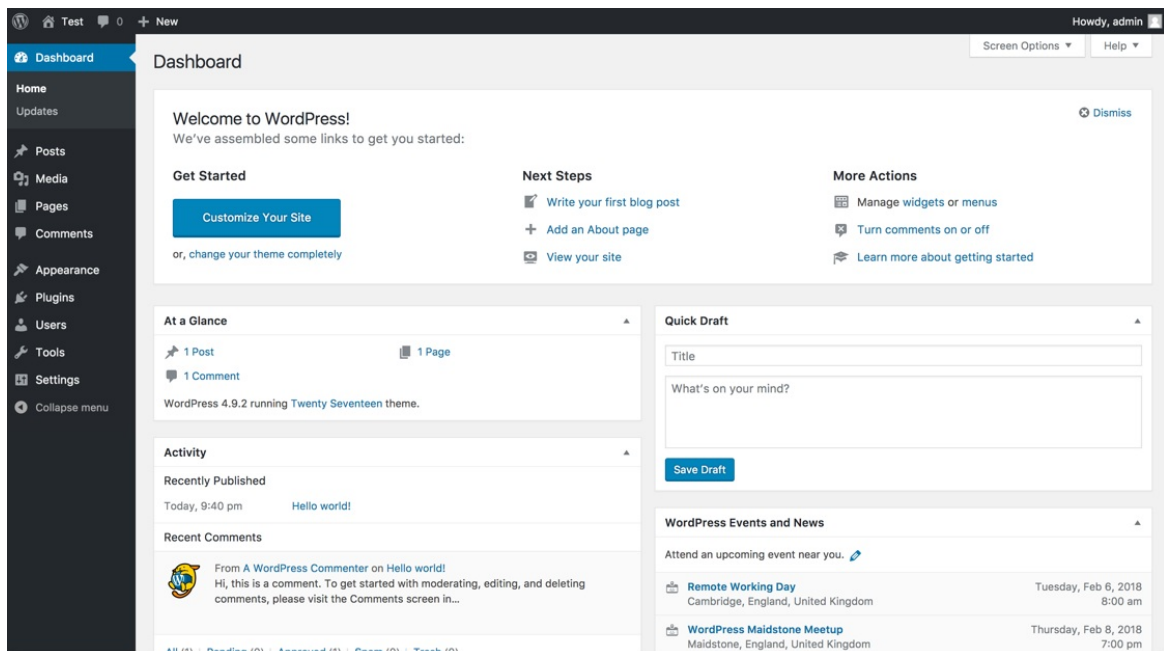


The image shows the WordPress installation 'Welcome' screen. At the top is the WordPress logo. Below it, a 'Welcome' heading is followed by a paragraph: 'Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.' This is followed by a section titled 'Information needed'. A message states: 'Please provide the following information. Don't worry, you can always change these settings later.' The form contains the following fields and options:

- Site Title:** A text box containing 'Test'.
- Username:** A text box containing 'admin'. Below it, a note says: 'Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.'
- Password:** A text box containing 'JZ...'. To its right is a 'Hide' button. Below the text box is a green bar with the word 'Strong'.
- Important:** A message below the password field: 'Important: You will need this password to log in. Please store it in a secure location.'
- Your Email:** A text box containing 'vikram.godse@alibaba-inc.cc'. Below it, a note says: 'Double-check your email address before continuing.'
- Search Engine Visibility:** A checkbox labeled 'Discourage search engines from indexing this site'. Below it, a note says: 'It is up to search engines to honor this request.'

At the bottom left of the form is a button labeled 'Install WordPress'.

3. WordPress安装完成后，单击 登录，输入管理员的用户名和密码，进入WordPress应用。



## 相关文档

- [Kubernetes Tutorials](#)

## 4.8. 应用部署

### 4.8.1. 部署Web集群

本文介绍如何使用Terraform部署Web集群。

#### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

#### 背景信息

在部署一个网站或者应用时，需要部署一系列的节点，SLB对各个节点分配请求，并根据访问数量或者资源使用的情况来自动伸缩。本文介绍将如何使用Terraform部署一个这样的Web集群。在本示例中，整个应用部署在一个可用区，并且只提供8080端口访问hello world网页。

#### 操作步骤

##### 1. 创建VPC网络和交换机。

- i. 创建`terraform.tf`文件，输入以下内容，并保存在当前的执行目录中。

```
resource "alicloud_vpc" "vpc" {
  name      = "tf_test_foo"
  cidr_block = "172.16.0.0/12"
}
resource "alicloud_vswitch" "vsw" {
  vpc_id      = alicloud_vpc.vpc.id
  cidr_block  = "172.16.0.0/21"
  availability_zone = "cn-beijing-b"
}
```

- ii. 运行 `terraform apply` 开始创建。
- iii. 运行 `terraform show` 查看已创建的VPC和VSwitch。

您也可以登录VPC控制台查看VPC和VSwitch的属性。

##### 2. 创建安全组，并将安全组作用于上一步创建的VPC中。



- i. 在 `terraform.tf` 文件中增加以下内容。

```
resource "alicloud_security_group" "default" {
  name = "default"
  vpc_id = alicloud_vpc.vpc.id
}

resource "alicloud_security_group_rule" "allow_all_tcp" {
  type           = "ingress"
  ip_protocol    = "tcp"
  nic_type       = "intranet"
  policy         = "accept"
  port_range     = "1/65535"
  priority       = 1
  security_group_id = alicloud_security_group.default.id
  cidr_ip        = "0.0.0.0/0"
}
```

- ii. 运行 `terraform apply` 开始创建。

- iii. 运行 `terraform show` 查看已创建的安全组和安全组规则。

3. 创建负载均衡实例，为其分配公网IP。在本示例中，为负载均衡实例配置了从前端80端口到后端8080端口的映射，并输出公网IP用于后续测试。

- i. 创建 `slb.tf` 文件，并增加以下内容。

```
resource "alicloud_slb" "slb" {
  name          = "test-slb-tf"
  vswitch_id    = alicloud_vswitch.vsw.id
  internet      = true
}

resource "alicloud_slb_listener" "http" {
  load_balancer_id = alicloud_slb.slb.id
  backend_port     = 8080
  frontend_port    = 80
  bandwidth       = 10
  protocol         = "http"
  sticky_session   = "on"
  sticky_session_type = "insert"
  cookie           = "testslblistenercookie"
  cookie_timeout   = 86400
  health_check     = "on"
  health_check_type = "http"
  health_check_connect_port = 8080
}

output "slb_public_ip" {
  value = alicloud_slb.slb.address
}
```

- ii. 运行 `terraform apply` 开始创建。

- iii. 运行 `terraform show` 查看已创建的负载均衡实例。

4. 创建弹性伸缩。

在本示例中，将创建以下资源：

- 伸缩组：在模版中指定伸缩最小为2，最大为10，并将伸缩组与新建的负载均衡实例绑定。由于伸缩

组的配置要求SLB必须有相应配置的监听器，因此模版中用depends\_on属性指定了部署顺序。

- o 伸缩组配置：在模版中指定ECS实例的具体配置。在初始化配置（user-data）中生成一个Hello World的网页，并在8080端口提供服务。为简化操作，本示例中会为ECS实例分配公网IP，并且设置 `force_delete=true` 用于后续删除环境。
- o 伸缩规则：定义具体的伸缩规则。
- i. 创建 `ess.tf` 文件，并增加以下内容。

```
resource "alicloud_ess_scaling_group" "scaling" {
  min_size = 2
  max_size = 10
  scaling_group_name = "tf-scaling"
  vswitch_ids = alicloud_vswitch.vsw.*.id
  loadbalancer_ids = alicloud_slb.slb.*.id
  removal_policies = ["OldestInstance", "NewestInstance"]
  depends_on = ["alicloud_slb_listener.http"]
}

resource "alicloud_ess_scaling_configuration" "config" {
  scaling_group_id = alicloud_ess_scaling_group.scaling.id
  image_id = "ubuntu_18_04_64_20G_alibase_20190624.vhd"
  instance_type = "ecs.c5.large"
  security_group_id = alicloud_security_group.default.id
  active = true
  enable = true
  user_data = "#!/bin/bash\nnecho \"Hello, World\" > index.html\nnohup busybox httpd\n-f -p 8080&"
  internet_max_bandwidth_in = 10
  internet_max_bandwidth_out = 10
  internet_charge_type = "PayByTraffic"
  force_delete = true
}

resource "alicloud_ess_scaling_rule" "rule" {
  scaling_group_id = alicloud_ess_scaling_group.scaling.id
  adjustment_type = "TotalCapacity"
  adjustment_value = 2
  cooldown = 60
}
```

- ii. 运行 `terraform apply` 开始创建。

创建成功后，会输出SLB的公网IP。

等待大约两分钟，弹性伸缩将自动创建ECS实例。

- iii. 输入命令 `curl http://<slb public ip>` 进行验证。

如果看到 `Hello, World`，表示成功通过负载均衡实例访问ECS实例提供的网页。

- 5. 运行 `terraform destroy` 删除测试环境。经确认后，整个部署的环境将被删除。

使用Terraform可以便捷地删除和重新部署一个环境。如果您想重新部署，运行 `terraform apply` 即可。

## 4.8.2. 一键创建分布式集群并部署文件

本教程介绍如何通过Terraform创建一个基于Master/Slave主从模式的分布式集群，并向集群中部署更新文件。

## 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

## 操作步骤

1. 编写Terraform脚本代码。本教程以Cloud Shell中运行terraform为例。

- i. 登录阿里云[Cloud Shell](#)。
- ii. 在`main.tf`文件中声明Module。

```
vim main.tf
```

按下 `i` 键进入vim的编辑模式，新增以下内容：

```
module "app-deployer" {
  source = "Starnop/app-deployer/alibabacloud"
  version = "0.0.1"
  region = var.region != "" ? var.region : null
  availability_zone = module.vpc.this_availability_zones[0]
  vpc_id = module.vpc.this_vpc_id
  app_name = "example"
  instance_settings = [
    {
      identifier = "master"
      description = "master node"
      hostnamePrefix = "master"
      ecs_password = "Example123"
      image_id = null
      image_owners = null
      image_name_regex = null
      instance_type = null
      cpu = 2
      memory = 4
      system_disk_size = 100
      data_disks = null
      max_bandwidth_out = 100
      security_groups = null
      vswitch_id = module.vpc.this_vswitch_ids[0]
      private_ip = null
      user_data = null
      temp_files = ["temp_files/master_template.file"]
      static_files = ["static_files/master_static.file"]
      entrypoint = "echo SUCCESS"
      size = 2
    },
    {
      identifier = "worker"
```

```
description      = "worker node"
hostnamePrefix   = "worker"
ecs_password     = "Example123"
image_id         = null
image_owners     = null
image_name_regex = null
instance_type    = null
cpu              = 4
memory           = 8
system_disk_size = 100
data_disks       = null
max_bandwidth_out = 100
security_groups  = null
vswitch_id       = module.vpc.this_vswitch_ids[0]
private_ip       = null
temp_files       = ["temp_files/worker_template.file"]
static_files     = ["static_files/worker_static.file"]
user_data        = null
entrypoint       = "echo SUCCESS"
size             = 3
}
]
}
module "vpc" {
  source = "alibaba/vpc/alicloud"
  region      = var.region != "" ? var.region : null
  vpc_name     = "my_vpc"
  vswitch_name = "my_vswitch"
  vswitch_cidrs = [
    "172.16.1.0/24",
  ]
}
```

按下 `Ecs` 键退出编辑模式，进入命令模式输入命令 `:wq`，保存并退出vim。

iii. 在文件 `variables.tf` 中定义常量参数。

```
vim variables.tf
```

按下 `i` 键进入vim的编辑模式，新增以下内容：

```
variable "region" {
  description = "The region ID used to launch this module resources. If not set, it
  will be sourced from followed by ALICLOUD_REGION environment variable and profile."
  default     = ""
}
```

按下 `Ecs` 键退出编辑模式，进入命令模式输入命令 `:wq`，保存并退出vim。

iv. 在文件 `outputs.tf` 中定义输出参数。

```
vim outputs.tf
```

按下 `i` 键进入vim的编辑模式，新增以下内容：

```
output "instances" {
  value = {
    for identifier, instance in module.app-deployer.this_instanceList : identifier
    => instance
  }
}
```

按下 `Ecs` 键退出编辑模式，进入命令模式输入命令 `:wq`，保存并退出vim。

v. 新增部署文件。

■ `static_files/master_static.file`

```
mkdir static_files && vim static_files/master_static.file
```

按下 `i` 键进入vim的编辑模式，新增以下内容：

```
master_static
```

按下 `Ecs` 键退出编辑模式，进入命令模式输入命令 `:wq`，保存并退出vim。

■ `static_files/worker_static.file`

```
mkdir static_files && vim static_files/worker_static.file
```

按下 `i` 键进入vim的编辑模式，新增以下内容：

```
worker_static
```

按下 `Ecs` 键退出编辑模式，进入命令模式输入命令 `:wq`，保存并退出vim。

■ `temp_files/master_template.file`

```
mkdir temp_files && vim temp_files/master_template.file
```

按下 `i` 键进入vim的编辑模式，新增以下内容：

```
Instances: "${Instances.master_0.private_ip}"
```

按下 `Ecs` 键退出编辑模式，进入命令模式输入命令 `:wq`，保存并退出vim。

■ `temp_files/worker_template.file`

```
mkdir temp_files && vim temp_files/worker_template.file
```

按下 `i` 键进入vim的编辑模式，新增以下内容：

```
Instances: "${Instances.master_0.public_ip}"
```

按下 `Ecs` 键退出编辑模式，进入命令模式输入命令 `:wq`，保存并退出vim。

2. 运行 `terraform init` 初始化。

```
terraform init
```

命令输出结果类似如下：

```
Initializing modules...
Downloading Starnop/app-deployer/alicloud 0.0.1 for app-deployer...
- app-deployer in .terraform/modules/app-deployer/Starnop-terraform-alicloud-app-deployer-8434639
Downloading alibaba/vpc/alicloud 1.4.2 for vpc...
- vpc in .terraform/modules/vpc/terraform-alicloud-modules-terraform-alicloud-vpc-7e25cee
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "null" (hashicorp/null) 2.1.2...
- Downloading plugin for provider "alicloud" (hashicorp/alicloud) 1.68.0...
- Downloading plugin for provider "local" (hashicorp/local) 1.4.0...
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.local: version = "~> 1.4"
* provider.null: version = "~> 2.1"
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.
```

### 3. 运行terraform apply开始创建。

```
terraform apply
```

命令输出结果类似如下：

```
module.app-deployer.data.alicloud_zones.zones_ds: Refreshing state...
module.vpc.data.alicloud_zones.default: Refreshing state...
module.vpc.data.alicloud_vpcs.this: Refreshing state...
module.vpc.data.alicloud_route_tables.this: Refreshing state...
module.app-deployer.data.alicloud_images.images["worker"]: Refreshing state...
module.app-deployer.data.alicloud_instance_types.ecs_type["master"]: Refreshing state..
.
module.app-deployer.data.alicloud_instance_types.ecs_type["worker"]: Refreshing state..
.
module.app-deployer.data.alicloud_images.images["master"]: Refreshing state...
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
Terraform will perform the following actions:
...
Plan: 23 to add, 0 to change, 0 to destroy.
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
module.vpc.alicloud_vpc.vpc[0]: Creating...
...
Apply complete! Resources: 23 added, 0 changed, 0 destroyed.
Outputs:
instances = {
  "master_0" = {
    "availability_zone" = "cn-chengdu-a"
    "credit_specification" = ""
    "data_disks" = []
    "deletion_protection" = false
    "description" = "OWNER: default\nmaster node"
    "dry_run" = false
    "host_name" = "master0"
  }
  "master_1" = {
    "availability_zone" = "cn-chengdu-a"
    "credit_specification" = ""
    "data_disks" = []
    "deletion_protection" = false
    "description" = "OWNER: default\nmaster node"
    "dry_run" = false
    "host_name" = "master1"
  }
  "worker_0" = {
    "availability_zone" = "cn-chengdu-a"
    "credit_specification" = ""
    "data_disks" = []
    "deletion_protection" = false
    "description" = "OWNER: default\nworker node"
    "dry_run" = false
    "host_name" = "worker0"
  }
  "worker_1" = {
    "availability_zone" = "cn-chengdu-a"
    "credit_specification" = ""
    "data_disks" = []
    "deletion_protection" = false
    "description" = "OWNER: default\nworker node"
    "dry_run" = false
    "host_name" = "worker1"
  }
  "worker_2" = {
    "availability_zone" = "cn-chengdu-a"
    "credit_specification" = ""
    "data_disks" = []
    "deletion_protection" = false
    "description" = "OWNER: default\nworker node"
```

```
"dry_run" = false
"host_name" = "worker2"
...
}
```

脚本执行完后，可登录服务器查看，部署的文件在服务器 `/tmp/example` 路径下。

## 相关文档

- [app-deployer\\_example](#)

## 4.8.3. 使用Terraform一键部署OpenShift

本教程介绍如何使用Terraform一键部署OpenShift平台。

### 前提条件

在开始之前，请您确保完成以下操作：

- 使用Terraform，您需要一个阿里云账号和访问密钥（AccessKey）。请在阿里云控制台中的[AccessKey管理页面](#)上创建和查看您的AccessKey。
- 已经安装并配置了Terraform，具体操作请参见[在本地安装和配置Terraform](#)和[在Cloud Shell中使用Terraform](#)。

### 背景信息

OpenShift是一个开源的容器云平台，底层基于容器资源编排系统Kubernetes和Docker引擎，企业可以基于此平台搭建内部PaaS（Platform as a Service，平台即服务）平台，贯穿CI/CD流程，提高企业IT效率，拥抱DevOps和敏捷开发。您可以使用Terraform Module（[OpenShift](#)）在阿里云上一键部署OpenShift平台。

### 操作步骤

1. 编写Terraform脚本代码。本教程以Cloud Shell中运行terraform为例。
  - i. 登录阿里云[Cloud Shell](#)。
  - ii. 生成当前用户的密钥对。

```
ssh-keygen
```



iii. 在`main.tf`中引用Module。

```
vim main.tf
```

按下`i`键进入vim的编辑模式，新增以下内容：

```
module "openshift" {
  source = "zzxwill/openshift/alibabacloud"
  version = "0.0.4"
  // 地域
  region = "cn-hangzhou"
  // 可用区
  availability_zone = "cn-hangzhou-i"
  // 请替换为您的本地公钥
  public_key = "ssh-rsa xxx"
}
```

其中`public_key`为本地公钥，可通过下面的命令查看。

```
cat ~/.ssh/id_rsa.pub
```

iv. 编写OKD安装脚本。

```
vim okd.sh
```

按下`i`键进入vim的编辑模式，新增以下内容：

```
#!/usr/bin/env bash
set -x
cd /opt/
wget https://github.com/openshift/origin/releases/download/v3.11.0/openshift-origin-
-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz
tar -zxvf openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz
cd openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit
export PATH=/opt/openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit:$PATH
yum install -y docker
service docker start
oc cluster up --skip-registry-check=true --public-hostname=$1
```

2. 运行`terraform init`初始化。

```
terraform init
```

命令输出结果类似如下：

```
Initializing modules...
Downloading zzxwill/openshift/alicloud 0.0.4 for openshift...
- openshift in .terraform/modules/openshift
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "null" (hashicorp/null) 2.1.2...
- Downloading plugin for provider "alicloud" (hashicorp/alicloud) 1.71.0...
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.alicloud: version = "~> 1.71"
* provider.null: version = "~> 2.1"
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.
```

### 3. 运行terraform apply开始创建。

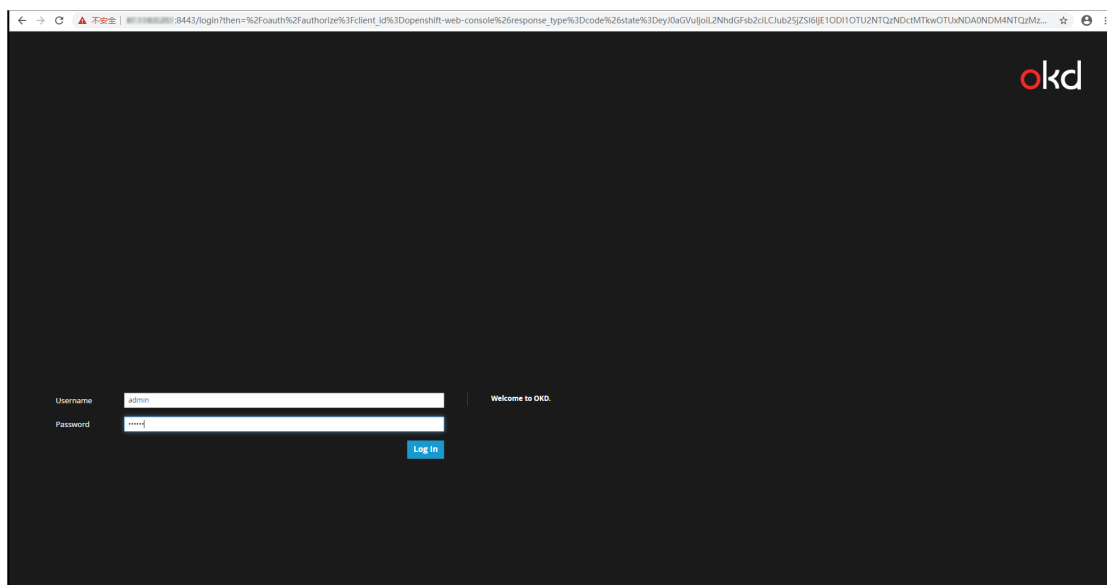
```
terraform apply
```

看到类似下面的输出结果，说明创建完成。

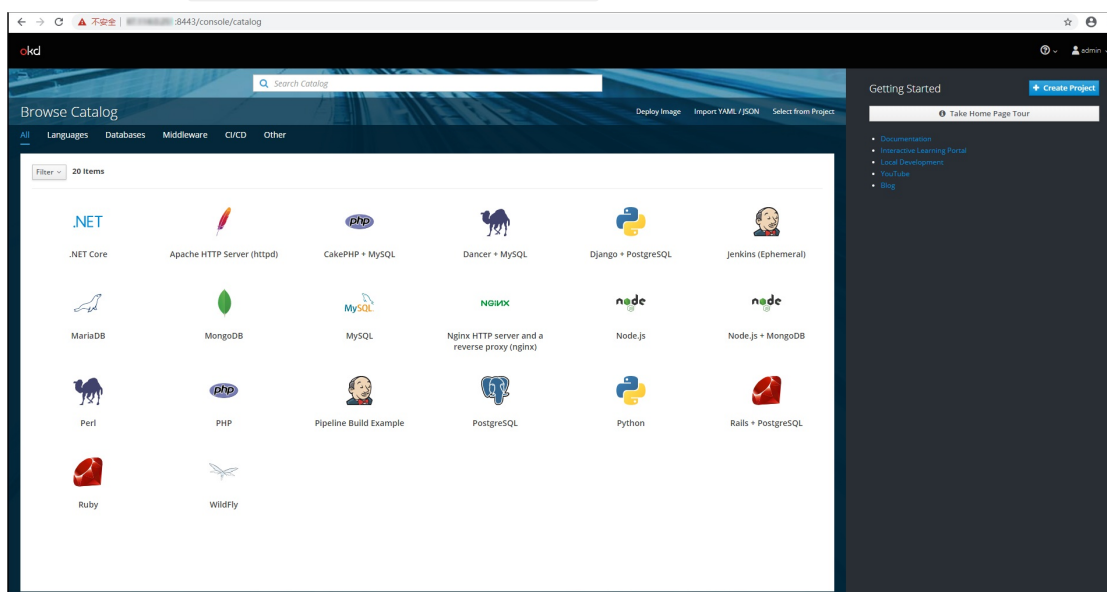
```
Apply complete! Resources: 11 added, 0 changed, 0 destroyed.
Outputs:
OpenShift = https://x.x.x.x:8443/console
Password = 123456
Username = admin
```

### 4. 登录OpenShift平台。

- i. 完成创建后在浏览器输入 `https://x.x.x.x:8443/console`（替换为自己的IP地址），就可以看到OpenShift登录页面。



- ii. 在登录界面输入 `Username=admin, Password=123456`，登录成功后如下图所示：



## 相关文档

- [openshift\\_module\\_sourcecode](#)

## 5. 参考

### 5.1. Terraform常用命令

Terraform是一个管理IT资源的客户端工具，您可以使用Terraform提供的命令来实现对所有资源的管理。本文将主要围绕资源管理和状态管理两个方面为您介绍涉及到的常用命令。

#### 资源管理常用命令

Terraform对资源的管理主要是对资源生命周期的管理，即通过命令实现对Terraform模板中所定义资源的创建，修改，查看和删除。

- **terraform plan**：资源的预览

plan命令用于对模板中所定义资源的预览，主要用于以下几个场景：

- 预览当前模板中定义的资源是否符合管理预期，和Markdown的预览功能类似。
- 如果当前模板已经存在对应的state文件，那么plan命令将会展示模板定义与state文件内容的diff结果，如果有变更，会将结果在下方显示出来。
- 对DataSource而言，执行plan命令，即可直接获取并输出所要查询的资源及其属性。

- **terraform apply**：资源的新建和变更

apply命令用于实际资源的新建和变更操作，为了安全起见，在命令运行过程中增加了人工交互的过程，即需要手动确认是否继续，当然也可以通过--auto-approve参数来跳过人工确认的过程。

apply命令适用于以下几种场景：

- 创建新的资源。
- 通过修改模板参数来修改资源的属性。
- 如果从当前模板中删除某个资源的定义，apply命令会将该资源彻底删除。可以理解为“资源的移除也是一种变更”。

- **terraform show**：资源的展示

show命令用于展示当前state中所有被管理的资源及其所有属性值。

- **terraform destroy**：资源的释放


destroy命令用于对资源的释放操作，为了安全起见，在命令执行过程中，也增加了人工交互的过程，如果想要跳过手动确认操作，可以通过--force参数来跳过。

terraform destroy默认会释放当前模板中定义的所有资源，如果只想释放其中某个特定的资源，可以通过参数 `-target=<资源类型>.<资源名称>` 来指定。

- **terraform import**：资源的导入

import命令用于将存量的云资源导入到terraform state中，进而加入到Terraform的管理体系中，适用的场景包含但不限于以下几种：

- 从来没有使用Terraform管控过任何资源，当前所有的存量云资源都是通过控制台，阿里云CLI，ROS或者直接调用API创建和管理的，现在想要切换为Terraform管理。
- 在不影响资源正常使用的前提下，重构资源模板中的资源定义。
- 阿里云的Provider进行了兼容性升级，新版Provider对原有模板中所定义的资源支持了更多的参数，需要把最新的参数同步进来。

 说明 有关import如何实现存量资源的管理，请参见[如何解决存量云资源的管理难题](#)。

- **terraform taint**：标记资源为被污染

taint命令用于把某个资源标记为被污染状态，当再次执行apply命令时，这个被污染的资源将会被先释放，然后再创建一个新的，相当于对这个特定资源做了先删除后新建的操作。

命令的详细格式为：`terraform taint <资源类型>.<资源名称>`，如：

```
$ terraform taint alicloud_vswitch.this
Resource instance alicloud_vswitch.this has been marked as tainted.
```

- **terraform untaint**：取消被污染标记

untaint命令是taint的逆向操作，用于取消被污染标记，使其恢复到正常的状态。命令的详细格式和taint类似为：`terraform untaint <资源类型>.<资源名称>`，如：

```
$ terraform untaint alicloud_vswitch.this
Resource instance alicloud_vswitch.this has been successfully untainted.
```

- **terraform output**：打印出参及其值

如果在模板中显示定义了output参数，那么这个output的值将在apply命令之后展示，但plan命令并不会展示，如果想随时随地快速查看output的值，可以直接运行命令 `terraform output`：

```
$ terraform output
vswitchId = vsw-gw8gl3lwz*****
```

## 状态管理常用命令

Terraform对资源状态的管理，实际上是对State文件中数据的管理。State文件保存了当前Terraform管理的所有资源及其属性，内容都是由Terraform自动存储的，为了保证数据的完整性，不建议手动修改State内容。对State数据的操作可以通过**terraform state**命令来完成。

- **terraform state list**：列出当前state中的所有资源

state list命令会按照 `<资源类型>.<资源名称>` 的格式列出当前state中存在的所有资源（包括datasource），例如：

```
$ terraform state list
data.alicloud_slbs.default
alicloud_vpc.default
alicloud_vswitch.this
```

- **terraform state show**：展示某一个资源的属性

state show命令按照Key-Value的格式展示出特定资源的所有属性及其值，命令的完整格式为 `terraform state show <资源类型>.<资源名称>`，例如：

```
$ terraform state show alicloud_vswitch.this
# alicloud_vswitch.this:
resource "alicloud_vswitch" "this" {
  availability_zone = "eu-central-1a"
  cidr_block       = "172.16.0.0/24"
  id               = "vsw-gw8gl3lwz*****"
  vpc_id           = "vpc-gw8calnzt*****"
}
```

- **terraform state pull**: 获取当前state内容并展示

**state pull**命令用于原样展示当前state文件数据，类似与Shell下的cat命令，例如：

```
$ terraform state pull
{
  "version": 4,
  "terraform_version": "0.12.8",
  "serial": 615,
  "lineage": "39aeeee2-b3bd-8130-c897-2cb8595cf8ec",
  "outputs": {
    ***
  },
  "resources": [
    {
      "mode": "data",
      "type": "alicloud_slbs",
      "name": "default",
      "provider": "provider.alicloud",
      ***
    },
    {
      "mode": "managed",
      "type": "alicloud_vpc",
      "name": "default",
      "provider": "provider.alicloud",
      ***
    }
  ]
}
```

- **terraform state rm**: 移除特定的资源

**state rm**命令用于将state中的某个资源移除，但是实际上并不会真正删除这个资源，命令格式为：`terraform state rm <资源类型>.<资源名称>`，例如：

```
$ terraform state rm alicloud_vswitch.this
Removed alicloud_vswitch.this
Successfully removed 1 resource instance(s).
```

移除后，如果模板内容不变并且再次执行**apply**命令，将会新增一个同样的资源。移除后的资源可以再次通过**import**命令再次加入。

- **terraform state mv**: 变更特定资源的存放地址

如果想调整某个资源所在的state文件，可以通过`state mv`命令来完成，类似于Shell下的`mv`命令，这个命令的使用有多种选项，可以通过命令`terraform state mv --help`来详细了解。本文只介绍最常用的一种：`terraform state mv --state=./terraform.tfstate --state-out=<target path>/terraform-t`  
`arget.tfstate <资源类型>.<资源名称A> <资源类型>.<资源名称B>`，如：

```
$ terraform state mv --state-out=./tf.tfstate alicloud_vswitch.this alicloud_vswitch.default
Move "alicloud_vswitch.this" to "alicloud_vswitch.default"
Successfully moved 1 object(s)
```

如上命令省略了默认的 `--state=./terraform.tfstate` 选项，命令最终的结果是将当前State中的vSwitch资源移动到了上层目录下名为 `tf.tfstate` 的State中，并且将VSwitch的资源名称由“this”改为了“default”。

- **terraform refresh**：刷新当前state

`refresh`命令可以用来刷新当前State的内容，即再次调用API并拉取最新的数据写入到state文件中。

## 其他常用命令

除了资源和状态的管理命令外，还有一些常用的应用模板，Provider等多种场景下的命令。

- **terraform init**：初始化加载模块

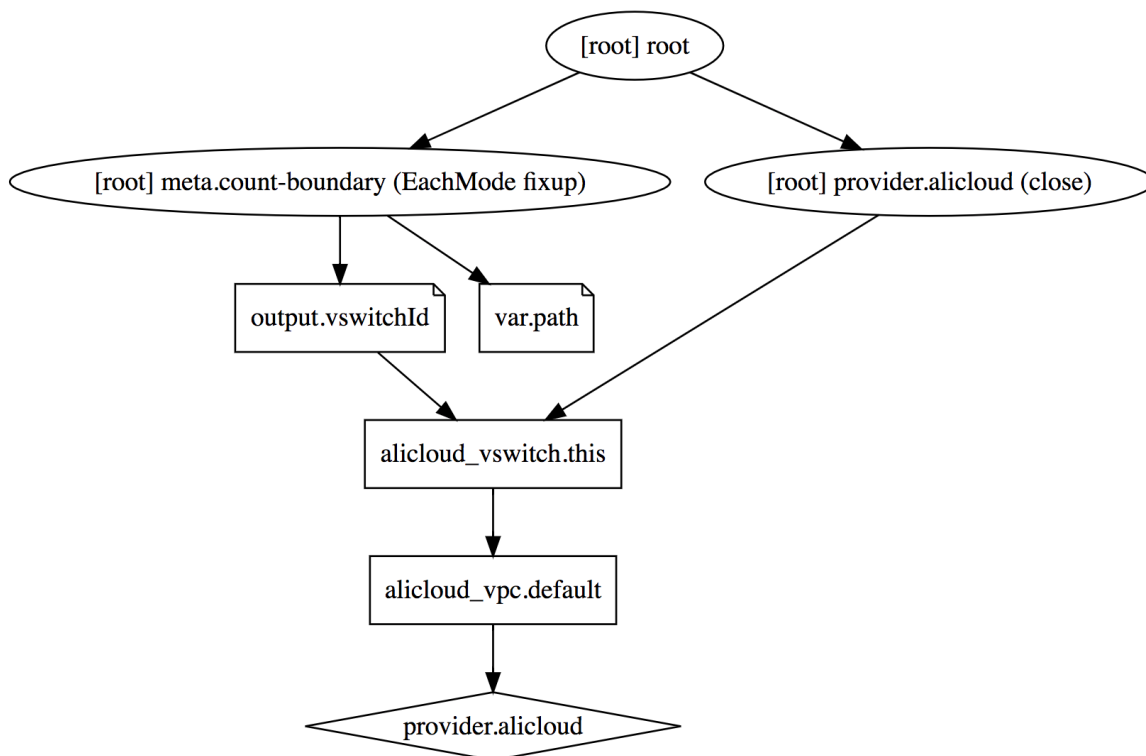
`init`用来初始化加载所需的模块，包括Provider，Provisioner，Module等。

- **terraform graph**：输出当前模板定义的资源关系图

每个模板定义的资源之间都存在不同程度的关系，如果想看资源关系图，可以使用命令`terraform graph`：

```
$ terraform graph
digraph {
    compound = "true"
    newrank = "true"
    subgraph "root" {
        "[root] alicloud_vpc.default" [label = "alicloud_vpc.default", shape = "box"]
        "[root] alicloud_vswitch.this" [label = "alicloud_vswitch.this", shape = "box"]
        *****
        "[root] output.vswitchId" -> "[root] alicloud_vswitch.this"
        "[root] provider.alicloud (close)" -> "[root] alicloud_vswitch.this"
        *****
        "[root] root" -> "[root] provider.alicloud (close)"
    }
}
```

该命令的结果还可以通过命令 `terraform graph | dot -Tsvg > graph.svg` 直接导出为一张图片（需要提前安装graphviz：`brew install graphviz`）：



- **terraform validate**：验证模板语法是否正确

Terraform模板的编写需要遵循其自身定义的一套简单的语法规则，编写完成后，如果想要检查模板是否存在语法错误或者在运行**plan**和**apply**命令的时候报错语法错误，可以通过执行命令**terraform validate**来检查和定位错误出现的详细位置和原因。

## 5.2. 如何解决存量云资源的管理难题

本文介绍如何使用Terraform解决存量云资源管理难题。

### 背景信息

本文内容适用于以下四种运维场景：

- 场景一：长期使用控制台、阿里云CLI、资源编排服务或者直接调用API创建和管理资源，初次使用Terraform的场景。
- 场景二：长期使用Terraform管理资源，如果通过控制台对单个云资源做属性变更，希望保持原有的资源状态（State）一致的场景。
- 场景三：所有资源都定义在一个模板中，想要对原有模板进行重构拆分，以降低随着资源不断增多而带来的模板和state的管理复杂度的场景。
- 场景四：想要将新版Provider中新增的参数同步到原文档中的场景。

Terraform基于资源模板定义不仅可以实现对新资源的创建，变更，删除等操作，还可以通过简单的命令将那些游离在Terraform管理体系之外的云资源进行导入和纳管，进而实现对所有云资源的统一管理。

### Terraform 导入存量资源

Terraform对资源的导入可以分为三个步骤：



### 1. 获取资源ID:

基于资源ID查询资源并获取其属性。

### 2. 模板声明所要导入的资源:

模板驱动，即使是要导入的资源，也需要在模板中进行声明。

### 3. 补齐资源模板定义:

导入成功后，需要根据资源属性补齐已经在模板中声明的资源定义。

### 1. 获取资源ID。

对资源ID的获取可以通过Web控制台，CLI，API等多种方式，最简单的方式是通过Terraform的DataSource，输入简单的查询条件，例如获取一个负载均衡实例：

```
data "alicloud_slbs" "default" {
  name_regex = "for-demo*"
}
output "slb_ids" {
  value = data.alicloud_slbs.default.ids
}
```

运行 **terraform apply** 命令即可展示所有符合条件的SLB的ID:

```
$ terraform apply
data.alicloud_slbs.default: Refreshing state...
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
Outputs:
slb_ids = [
  "lb-gw8vinrqxxxxxxxxxxxx",
  "lb-gw8axostxxxxxxxxxxxx",
]
```

### 2. 模板声明所要导入的资源。

和创建资源一样，在导入资源前，也需要在模板中进行资源声明，以便指定所要导入的资源在State中的存放路径。如下所示，声明一个负载均衡实例：

```
resource "alicloud_slb" "this" {}
```

简单的声明之后，无需定义具体的参数即可开始资源的导入操作。在Terraform中，导入一个资源的操作通过import命令来完成，完整的命令格式为 `terraform import <资源类型>.<资源标识> <资源ID>`，详细操作如下：

```
$ terraform import alicloud_slb.this lb-gw8vinrqxxxxxxxxxxxx
alicloud_slb.this: Importing from ID "lb-gw8vinrqxxxxxxxxxxxx"...
alicloud_slb.this: Import prepared!
  Prepared alicloud_slb for import
alicloud_slb.this: Refreshing state... [id=lb-gw8vinrqxxxxxxxxxxxx]
Import successful!
The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.
```

### 3. 补齐资源模板定义。

由于模板中没有完成对所导入资源的详细定义，因此，资源导入成功后，模板内容与State存储的内容存在差异，此时如果直接运行plan命令，将会看到一个update：

```
$ 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.alicloud_slbs.default: Refreshing state...
alicloud_slb.this: Refreshing state... [id=lb-gw8vinrqxxxxxxxxxxx]
-----
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  ~ update in-place
Terraform will perform the following actions:
  # alicloud_slb.this will be updated in-place
  ~ resource "alicloud_slb" "this" {
    address          = "47.254.181.122"
    ...
    ~ delete_protection = "on" -> "off"
    id                = "id=lb-gw8vinrqxxxxxxxxxxx"
    ...
    ~ name             = "for_demo-test" -> "tf-lb-20191108144235105700000001"
    ...
  }
Plan: 0 to add, 1 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.
```

为了保持资源模板与资源状态的一致，需要在模板中手动补齐缺失的参数定义，直到运行plan不会再有变更信息为止：

```
resource "alicloud_slb" "this" {
  delete_protection = "on"
  name               = "for_demo-test"
}
```

所要补齐的内容主要以那些引起更新的字段为主，补齐完成后运行terraform plan进行测试：

```
$ 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.alicloud_slbs.default: Refreshing state...
alicloud_slb.this: Refreshing state... [id=lb-gw8vinrqtqx1rolr94c96]
-----
No changes. Infrastructure is up-to-date.
This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

可以看到，此时已经没有任何需要变更的信息。至此完成了对一个资源的完整导入。

## Terraform移除存量资源

在实际操作场景中，经常会遇到资源的误导入，导入路径不符，想要调整资源路径，想要永久保留某资源等多种复杂情况。面对这些情况，整体的实现思路也是非常简单：先将导入后的资源从State中移除，然后重新导入。

资源的移除操作可以通过`state rm`命令来完成，完整的命令格式为 `terraform state rm <资源类型>.<资源标识>`，详细操作如下：

```
$ terraform state rm alicloud_slb.this
Removed alicloud_slb.this
Successfully removed 1 resource instance(s).
```

`state rm`命令只是将指定的资源从State文件中移除，并不会将其真正删除，这也正是为后续的导入操作做好了铺垫。

## 总结

对于以上四种场景我们可以使用以下解决方案：

- 场景一的解决方案：

通过`terraform import`命令来完成对存量资源的导入，进而使用Terraform统一管理。

- 场景二的解决方案：

在确定清楚参数属性的具体值之后，如果以模板参数值为准，那么只需要运行`apply`命令再变更回来即可；如果以控制台的值为准，那么只需要补充或修改模板参数值即可。

- 场景三的解决方案：

可以先通过`terraform state rm`命令将所有需要重组的资源移出State，等模板重构结束后，再使用`terraform import`将其导入即可。

- 场景四的解决方案：

和上一解决方案一样，通过“先移出再导入”调整即可。

Terraform的命令非常灵活和简单，基于模板和State一致性的原理，借助`terraform import`可以轻松地对存量资源的统一管理，不用再担心那些游离在Terraform管理体系之外资源无法管理的痛点，也无需惧怕某个资源从State中移除后无法继续管理的问题，所有的云资源都可以被Terraform统一管理起来。

## 5.3. Alicloud Provider

查看可用的阿里云资源，请参见[资源文档列表](#)。

## 5.4. Alibaba Cloud Module Registry

单击[这里](#)查看提供的module。

## 5.5. 如何贡献代码？

- 如果您在使用的过程中遇到任何问题，可直接通过[GitHub](#) submit issue或者pull request进行询问，我们将尽快予以解决。
- 欢迎有更多的开发者贡献代码，包括GoLang的源码及Terraform的模板。Fork此项目，然后提交pull request，我们review代码后，即可合并进主干。

## 5.6. 支持的阿里云产品

本文档已经弃用，请移步[资源类型索引](#)页面查看所有支持的阿里云产品和资源。

### 弹性计算

云服务器ECS

弹性伸缩

容器服务

函数计算

### 存储

对象存储 OSS

表格存储

### 网络

专有网络 VPC

负载均衡

弹性公网IP

VPN网关

云企业网

智能接入网关

### 数据库

云数据库 RDS

云数据库 MongoDB版

云数据库 Redis 版

### 管理与监控

访问控制

云监控

密钥管理服务

### CDN和域名

CDN

云解析 DNS

### 应用服务

日志服务

## 6. 客户案例

### 6.1. 小马智行基于Terraform的IaC实践

本文章由小马智行DevOps团队总结，阐述了如何使用Terraform完成云上业务100%自动化的选型、决策和落地过程。

#### 背景

小马智行（pony.ai）成立于2016年，在硅谷、广州、北京、上海、深圳设立研发中心，并获得中美多地自动驾驶测试、运营资质与牌照。凭借人工智能技术领域的最新突破，已与丰田、现代、一汽、广汽等车企建立合作。目前估值85亿美元。小马智行国内业务使用阿里云作为基础架构的重要部分，承载了包括小马智行自研的Data Labeling、Robotaxi、Robotruck等业务。这些业务使用了包括ECS、RDS、SLB、堡垒机、云安全中心在内的众多产品。如何管理好这些基础组件给DevOps团队带来了不小的挑战。从业务需求出发，我们定义了三个目标：

1. 「组件部署可评审」：保障小马智行整体运维活动从需求收集、架构设计、代码编写最终到部署不会出现任何偏差。同时也能够保证代码编写符合我们的要求。
2. 「组件部署版本化」：保障小马智行任何基础设施生产的迭代都有迹可循。当出现极端情况的时候，可以快速恢复到指定版本，避免影响到业务。
3. 「组件部署多环境一致」：不同环境的一致性则能够保证小马智行的基础设施部署不会因为环境部署差异导致故障。

#### 技术选型

从业内角度来看，我们看到主要有三种主流公有云组件部署和管理方案：

1. 云服务商的控制台管理能力
2. 使用管理系统(可能自研或者购买)调用公有云API做操作
3. 基础设施即代码（IaC）框架



IaC方案，在国际的主流社区已经成为基础设施自动化的既定标准，也是最广泛使用的多云管理框架。但是国内相对来说比较滞后，知道并使用的人还是比较少。在这方面，拥有海外背景的小马智行具备技术的先发优势。我们发现，结合Git等代码管理工具，可以很好的解决部署可追溯以和版本化的问题。所有运维团队部署的过程和变更细节都可以通过代码很好的管理起来。当需要回退的时候，可通过Git的分支对基础设施进行回滚。

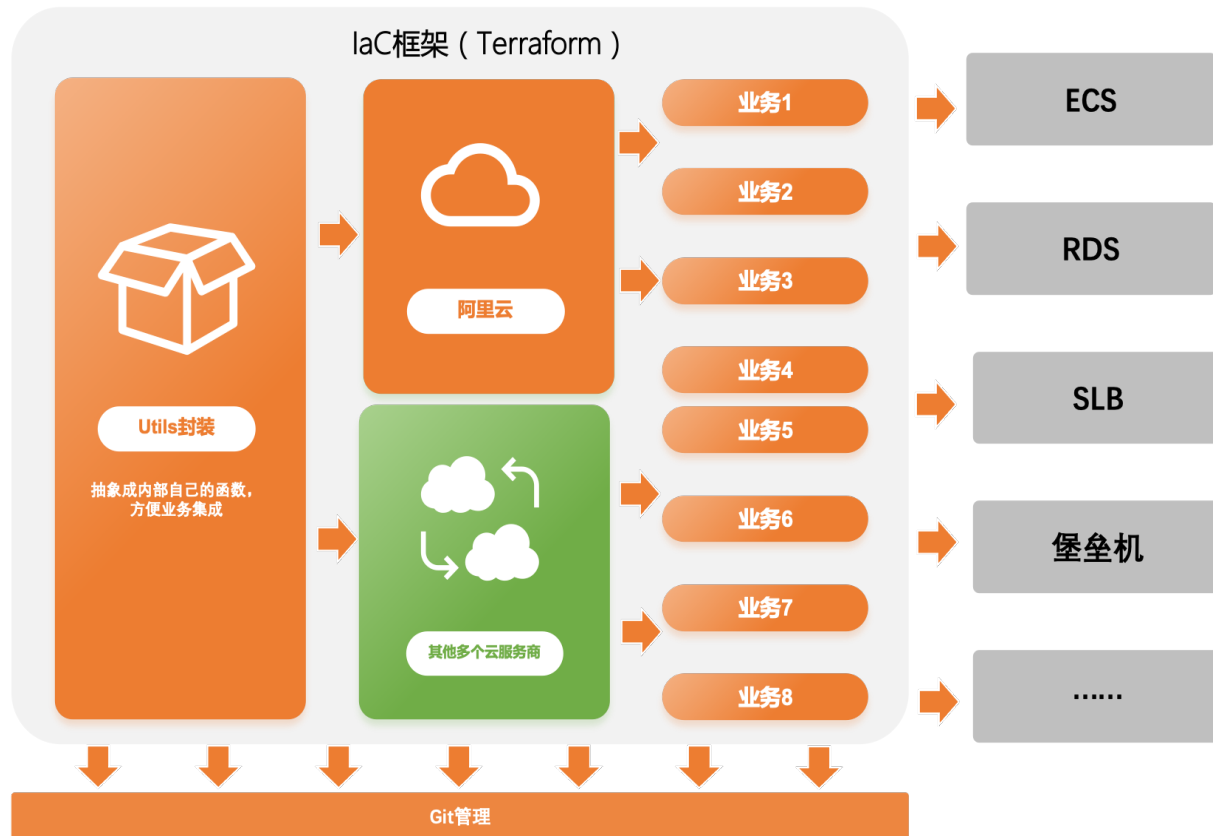
在社区生态方面，Terraform作为最优秀的、开源的IaC工具之一，已经被大量企业应用于生产环境。作为DevOps管理团队，我们则把主要精力投入到对应的基础架构逻辑代码的编写中去。

使用Terraform比调用各个云厂商API做二次开发不论是从开发量、复杂程度和运维难度都是极具优势的；帮助我们做到更好、更敏捷的部署。

最后，考虑到多云战略以及小马智行混合云的现状，结合Terraform的标准性、便捷性、易用性以及社区繁荣的特点，最终我们选择使用Terraform作为企业IaC落地的工具。

### 架构设计

小马智行团队选用以Terraform为技术核心的IaC解决方案，并通过如下架构图所示最终落地到业务生产中：



在Terraform配置文件格式上,技术团队综合考虑小马已经使用众多的json应用的现状，为了保持一致以及方便代码Review，没有选择直接使用HCL格式,而是选择了json格式。

在代码组织方面，小马智行选择了按业务维度来组织Terraform代码。比如：业务1使用了SLB、证书、ECS三种资源，那么在代码编写的时候会把三种资源都统一定义在一个Terraform文件上，类似如下：

```
{
  "output": {
    "ecs_instance_1-private-ip": {
      "value": "${alicloud_instance.ecs_instance_1.private_ip}"
    },
    "ecs_instance_2-private-ip": {
      "value": "${alicloud_instance.ecs_instance_2.private_ip}"
    },
    "ponyai_business_1-slb-address": {
      "value": "${alicloud_slb.ponyai_business_1-slb.address}"
    }
  },
  "provider": {
    "alicloud": {
      "region": "alicloud_region"
    }
  },
  "resource": {
```

```
"alicloud_instance": {
  "ecs_instance_1": {
    "availability_zone": "availability_zone_1",
    "data_disks": [
      {
        "category": "cloud_essd",
        "name": "data_volume",
        "size": "xx"
      }
    ],
    "host_name": "ecs_instance_1",
    "image_id": "image_id_1",
    "instance_name": "ecs_instance_1",
    "instance_type": "ecs_instance_type",
    "internet_charge_type": "PayByTraffic",
    "internet_max_bandwidth_out": 10,
    "key_name": "key_name_1",
    "security_groups": [
      "security_groups_1"
    ],
    "system_disk_category": "cloud_essd",
    "system_disk_size": "xx",
    "tags": {
      "host_name": "ecs_instance_1"
    },
    "vswitch_id": "vswitch_id_1"
  },
  "ecs_instance_2": {
    "availability_zone": "availability_zone_2",
    "data_disks": [
      {
        "category": "cloud_essd",
        "name": "data_volume",
        "size": "xx"
      }
    ],
    "host_name": "availability_zone_2",
    "image_id": "image_id_1",
    "instance_name": "availability_zone_2",
    "instance_type": "ecs_instance_type",
    "internet_charge_type": "PayByTraffic",
    "internet_max_bandwidth_out": 10,
    "key_name": "key_name_1",
    "security_groups": [
      "security_groups_1"
    ],
    "system_disk_category": "cloud_essd",
    "system_disk_size": "xx",
    "tags": {
      "host_name": "availability_zone_2"
    },
    "vswitch_id": "vswitch_id_2"
  }
},
```

```

    "alicloud_slb": {
      "slb-1": {
        "address_type": "internet",
        "internet_charge_type": "PayByTraffic",
        "name": "slb_name",
        "specification": "slb_specification"
      }
    },
    "alicloud_slb_listener": {
      "slb-listener-1": {
        "backend_port": "xx",
        "bandwidth": -1,
        "frontend_port": "xx",
        "health_check": "on",
        "health_check_connect_port": "xx",
        "health_check_domain": "domain_name",
        "health_check_type": "check_type",
        "health_check_uri": "uri_1",
        "load_balancer_id": "${alicloud_slb.slb-1.id}",
        "protocol": "protocol_1",
        "scheduler": "scheduler_1",
        "server_certificate_id": "${alicloud_slb_server_certificate.slb-certificate-1.id}",
        "server_group_id": "${alicloud_slb_server_group.slb-server-group-1.id}"
      }
    },
    "alicloud_slb_server_certificate": {
      "slb-certificate-1": {
        "alicloud_certificate_id": "xx",
        "alicloud_certificate_name": "xx",
        "name": "certificate_1"
      }
    },
    "alicloud_slb_server_group": {
      "slb-server-group-1": {
        "load_balancer_id": "${alicloud_slb.slb-1.id}",
        "name": "slb-server-group",
        "servers": {
          "port": "xx",
          "server_ids": [
            "${alicloud_instance.ecs_instance_1.id}",
            "${alicloud_instance.ecs_instance_2.id}"
          ]
        }
      }
    }
  },
  "terraform": {
    "backend": {
      "s3": {
        "bucket": "bucket_name",
        "dynamodb_table": "table",
        "key": "key_1",
        "profile": "profile_1",

```



```
        "region": "region_1"
      }
    },
    "required_providers": {
      "alicloud": {
        "source": "aliyun/alicloud",
        "version": "xx"
      }
    }
  }
}
```

### 业务挑战

在Terraform代码实际编写中，我们逐步发现，对一些资源的需求，我们更关心其中的一些参数和特性。比如：

- 阿里云的ECS，我们更多的是关心ECS创建时的instance\_type,instance\_name, availability\_zone等。
- 同一个业务在不同的部署环境仅仅只是一些资源的规格不太一样，比如业务1在正式的生产环境中的SLB使用的规格为slb.s2.medium，在测试环境中的规格为slb.s1.small，如果每次部署仅仅只是不同参数的相同组件时，代码上需要重新写一遍，无疑代码的可读性和重用性会非常差。

### 解决方案

考虑到Terraform使用的是json格式的文件，为了解决其代码重用和可读性问题，我们引入开源的jsonnet模板语言来生成Terraform使用的json文件，同时封装了丰富的Utils。比如：我们对于生成ECS，封装了如下的Function：

```
generateEcs(instance_name,
            availability_zone,
            vswitch_id,
            security_groups,
            instance_type,
            host_name,
            data_volume_size=null,
            system_disk_size=null,
            internet_charge_type="PayByTraffic",
            image_id="ubuntu_18_04_x64_20G_alibase_20200914.vhd",
            key_name="bootstrap-bot",
            system_disk_category="cloud_essd",
            internet_max_bandwidth_out=10,
            data_disk_category="cloud_essd"): {
instance_name: instance_name,
availability_zone: availability_zone,
vswitch_id: vswitch_id,
security_groups: security_groups,
instance_type: instance_type,
internet_charge_type: internet_charge_type,
image_id: image_id,
system_disk_category: system_disk_category,
[if system_disk_size != null then "system_disk_size"]:
  system_disk_size,
key_name: key_name,
internet_max_bandwidth_out: internet_max_bandwidth_out,
host_name: host_name,
data_disks: if data_volume_size != null then [
  {
    name: "data_volume",
    size: data_volume_size,
    category: data_disk_category,
  },
] else [],
tags: {
  host_name: host_name,
},
}
```

这样上层在使用的时候直接调用该Function就能生成对应的json部分。如下所示：

```
alicloud_instance: {
  [host_config.host_name]:
    ecsUtils.generateEcs(
      instance_name=host_config.host_name,
      availability_zone=host_config.az,
      security_groups=${.ecs_security_groups},
      host_name=host_config.host_name,
      instance_type=${.ecs_instance_type},
      vswitch_id=vpc_output["vswitch-public-" + host_config.az].value,
      data_volume_size=${.ecs_data_volume_size},
      system_disk_size=${.ecs_system_disk_size}
    )
  for host_config in host_configs
},
```

同时如果要调整的话直接调整对应的Utils函数即可,不需要每个基础架构组件去调整。

对于不同的环境(正式,预发布,测试)只有少量组件参数不同的情况,我们也仅仅先定义好一个基础的模板,然后不同的环境import之后对需要调整的参数赋予不同的值即可。

通过这样的方式,可以做到对于小马智行的某项业务在不同环境的部署,我们仅需要遵循如下的代码路径即可

备注: “generated/main.tf.json” 是文件 “main.tf.json.jsonnet” 用jsonnet工具生成出来的json文件,是Terraform最终执行的文件, “main.tf.json.jsonnet.output” 则是Terraform生效之后产生的一些字段和字段值输出):

```
├─ alicloud-region
│   ├── dev
│   │   ├── generated
│   │   │   └─ main.tf.json
│   │   ├── main.tf.json.jsonnet
│   │   └─ main.tf.json.jsonnet.output
│   ├── prod
│   │   ├── generated
│   │   │   └─ main.tf.json
│   │   ├── main.tf.json.jsonnet
│   │   └─ main.tf.json.jsonnet.output
│   └─ staging
│       ├── generated
│       │   └─ main.tf.json
│       ├── main.tf.json.jsonnet
│       └─ main.tf.json.jsonnet.output
└─ ponyai_business_1_base.libsonnet
```

以不同环境的SLB规格为例,在正式环境,测试环境中,我们仅仅需要把基础模板引入之后,调整不同的参数即可。比如正式环境我们用如下的代码:

```
local base = import "../..ponyai_business_1_base.libsonnet";
base {
  name: "ponyai_business_1_prod",
  environment: "prod",
  region: "alicloud_region",
  slb_specification: "slb.s2.medium"
}
```

测试环境我们则仅仅需要用如下的代码：

```
local base = import "../..ponyai_business_1_base.libsonnet";
base {
  name: "ponyai_business_1_dev",
  environment: "dev",
  region: "alicloud_region",
  slb_specification: "slb.s1.small"
}
```

这样可以实现较好的代码可读性和重用性。jsonnet也能够很方便的解决基础组件互相依赖导致在编写 Terraform 代码需要互相引用的问题。比如阿里云上创建 ECS，需要提供 VPC 的 ID，但是 VPC 一般是单独写在一个独立的 Terraform 文件中并单独生效，这个时候在代码层面创建 ECS 的 Terraform 代码则需要引用创建 VPC 生成出来的 vpc id。以小马智行为例子，我们用“main.tf.json”文件创建了一个 vpc，其 ID 按照我们的要求输出到了“main.tf.json.jsonnet.output”文件中：

```
├─ ali-cloud-region
│   └─ dev
│       └─ generated
│           └─ main.tf.json
│               └─ main.tf.json.jsonnet
│                   └─ main.tf.json.jsonnet.output
└─ prod
    └─ generated
        └─ main.tf.json
            └─ main.tf.json.jsonnet
                └─ main.tf.json.jsonnet.output
```

此时其“main.tf.json.jsonnet.output”文件输出如下：

```
{
  "vpc_id": {
    "sensitive": false,
    "type": "string",
    "value": "vpc_id_for_ponyai"
  },
  "vswitch-id": {
    "sensitive": false,
    "type": "string",
    "value": "vswitch_public_id_for_ponyai"
  }
}
```

我们在其他需要引用的地方用如下的语法就能很方便的进行引用，从而避免我们在代码库里面直接放入生成出来的值：

```
{
  "ali-cloud-region": {
    prod: import "./ali-cloud-region/prod/main.tf.json.jsonnet.output",
  }
}
```

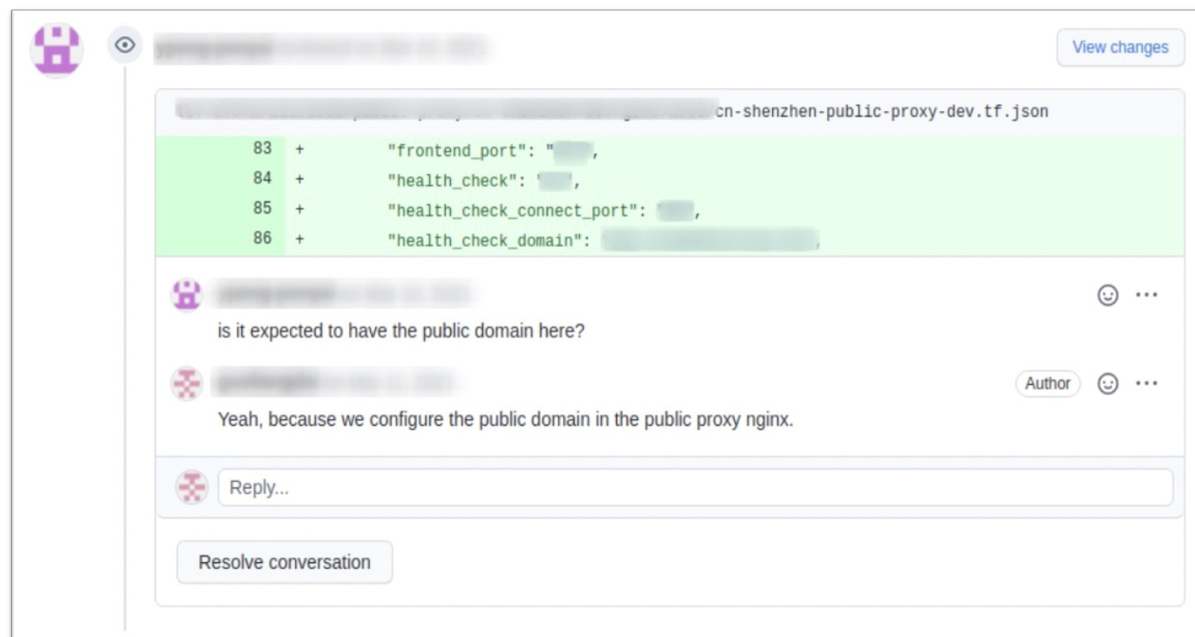
通过上述一层层的技术封装，小马智行DevOps团队即使用了Terraform的技术及生态能力，也解决了业务调用的复杂性问题。让运维效率整体的到很大提升。

### 业务收益

通过使用IaC的方式，我们对各个业务使用的各个阿里云的组件的参数都定义得非常明确，比如某业务使用了两台ECS（每一台的规格都是s6-c1m1.small且带有一块80G的系统盘和20G的数据盘）。结合Git管理，可以非常方便的进行Code Review；从而对整个部署过程进行把控。

如下图所示，我们团队在每次实际部署之前，都会通过Git进行代码Review整个PR，并在PR内进行讨论，最终确认好部署各个细节。并且，在未来的某一天，我们都有能力回溯到今天来看云基础设施发生了什么变更。

正是由于能够做到参数级别的Review，我们可以保证最终的部署和最初的设计不会出现偏差。如果在最终部署的时候发现和原先设计的时候会有偏差，我们也能在Terraform代码编写的时候发现并及时调整原先设计。同时我们在提交PR进行Review前，都要求进行自测，尽量避免出现Review多次，结果发现连运行都运行不了的情况出现。



回顾整个使用阿里云Terraform落地IaC的过程，我们总结了四个我们认为的主要业务收益：

- 「更快」：同样的基础设施生产不需要再去控制台重复低效操作，生产周期大大缩短。为企业的业务决策和市场机遇响应提供了强有力的基础支撑。
- 「更可控」：基础设施代码化，历史任何时间点都可以回溯。将组织从控制台操作升级至更优雅、可控、可信、可回溯的体系化管理阶段。
- 「更高效」：多团队、跨团队同尤其是跨国协同效率提高。解决了时差、工作模式带来的效率降低的问题。

- 「更安全」：大大减少了人为误操作导致的生产事故。针对不同的环境采取不同的审批链路，做到技术+人的双重结合。为小马智行的业务保驾护航。

#### 总结

- 管理模式升级

小马智行整体建设都是基于IaC的整体理念之下。目前企业内部已经将众多的阿里云组件抽象成内部自己的函数,包括ECS,VPC, OSS, PUBLIC DNS, RAM, SLS, USERS等。

- 业务模式升级

DevOps团队或者业务团队需要使用阿里云组件的时候都会使用这些封装好的函数去拉取对应的阿里云组件, DevOps团队则专心维护并迭代好这些函数即可。运维效率大大提升。

- 运维模式升级

小马智行目前内部20+业务都是按照IaC思路通过Terraform100%落地,我们能够非常清晰的看到各个业务使用的阿里云组件的迭代史;也能够很好的进行Review,及时拒绝不合理的部署,保证线上环境的干净、整洁、高可靠的同时兼顾优秀的可扩展性。

#### 作者介绍

小马智行 DevOps团队

## 6.2. 流利说基于Terraform的自动化实践

### 流利说简介

流利说® (NYSE: LAIX) 是卓越的科技驱动的教育公司,由王翌博士和胡哲人、林晖博士于2012年9月共同创立。作为智能教育的倡行者,公司拥有一支优秀的人工智能团队,其自主研发的人工智能英语老师,基于深度学习技术,能够为每一位用户提供个性化、自适应的学习课程。创立九年时间,流利说®连续推出了多样化、覆盖不同兴趣类别和细分人群的产品,包括“流利说®英语”、“流利说®阅读”等英语教育App以及“流利说®懂你英语®A+”、“流利说®发音”等英语学习产品。截至2021年6月,流利说®旗下产品累计注册用户超过2亿,用户覆盖175个国家和中国384个城市。同时,其搭建的巨型“中国人英语语音数据库”已累积实现记录超39亿分钟的对话和537亿句录音。

### 自动化的驱动力

面对瞬息万变的市场,业务团队希望快速响应市场的需求,打造更多有创意的产品。在这背后,要求Cloud Infra团队提供更敏捷的基础设施,提升用云的管理水平,主要体现在:

- 加速资源供应:支持业务团队以自服务的方式快速获取云资源;
- 降低总体成本:充分利用云的弹性能力,提升资源利用率以降低总体的成本;
- 提升运维和管理效率:提升运维的效率,减少重复性的人工操作。

### 流利说的自动化实践

我们的自动化实现不是一蹴而就的,而是逐步构建的过程,这一过程中我们总结出来的原则是:

- 1) 把日常重复性的工作进行自动化,让大家切身体会到自动化带来的价值,构建自动化的文化;
- 2) 以价值导向对自动化的优先级进行排序,释放自动化的业务价值。

我们从以下三个维度进行自动化的探索,可以说随着自动化的深入,我们的管理成熟度也越来越高:

- 部署自动化
- 管理自动化
- 治理自动化

## 部署自动化：资源供给的流程化和自动化

云资源的供给是所有工作开始的第一步，也是我们日常工作之一。在没有自动化之前，所有的操作都在控制台上完成，看似简单，但实际应用中会有许多的问题：

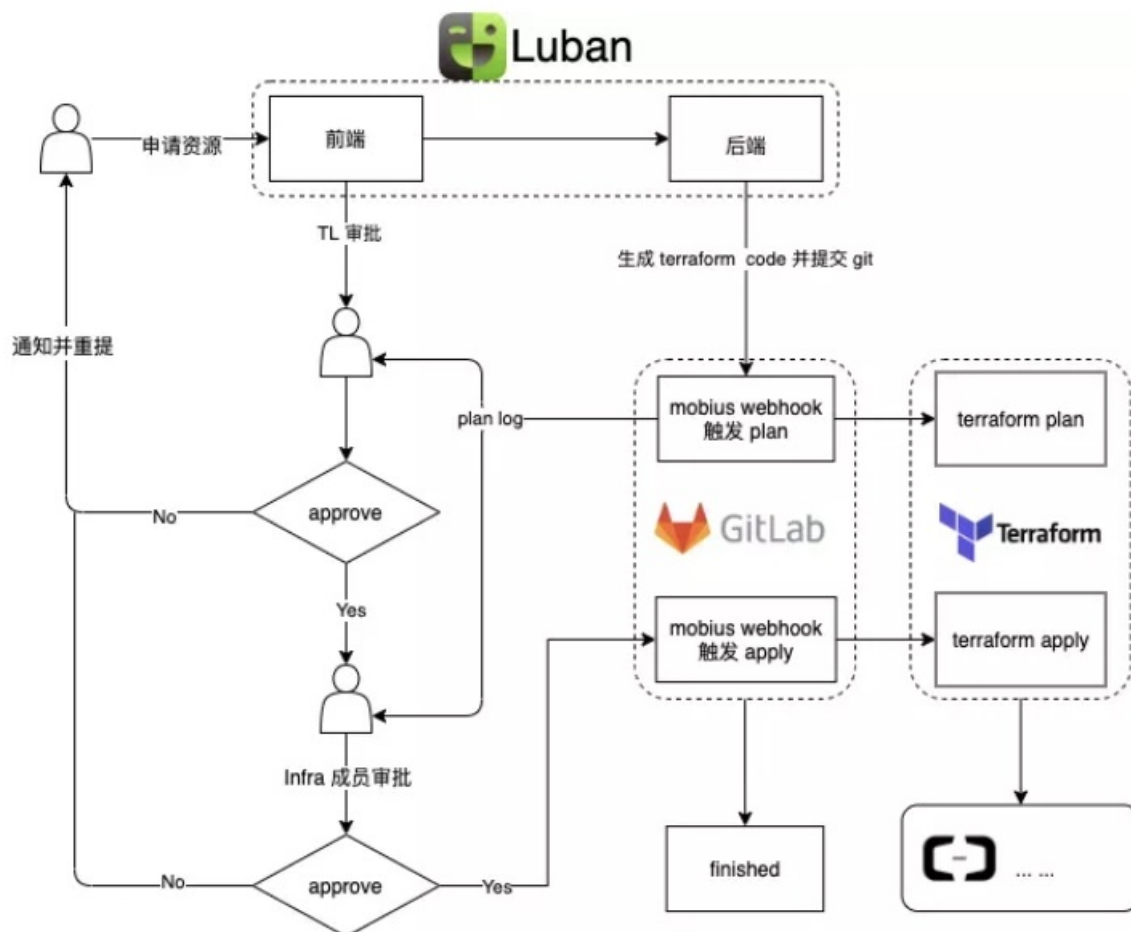
- 资源无法统一管理，没有统一的仓库去记录这些资源的归属与规格变更等信息，非常容易出现变动带来的混乱；
- 手工变更误操作影响线上服务正常运行，并难以回滚。人机交互过多，原来的便捷就变成了容易出错，最可怕的就是“点错了”，还忘了原来是什么样；
- 创建重复资源时，需要重复人工页面操作，耗时且无法标准化。

因此，我们基于Terraform、Luban（自研管理平台）、GitLab实现了完全的资源供给自动化，将供应效率从原来的小时级降低到分钟级，并且将运维支撑效率提升100%。

整体实现的架构如下：

- 通过Luban实现流程的自动化，基于Chatbox提升了效率和体验；
- 基于GitLab实现对IaC的资源配置库统一管理；
- 基于Terraform实现在阿里云的资源创建和变更操作。

以下我们将介绍详细的架构和实现方案。



### 1) 在 Luban 平台申请资源

我们给研发团队提供的各种窗口链接都放在了一个叫做 Luban 的平台上，申请人只需要在前端选择必要的参数，提交申请，对于申请人来说需要做的事情就结束了，接下来就是等待审批结果。例如，我想要申请一个阿里云 ECS 实例，如果直接写 Terraform，那大概要写一个如下的文件：

```
resource "alicloud_instance" "instance" {
  # cn-beijing
  availability_zone = "cn-beijing-b"
  security_groups  = alicloud_security_group.group.*.id

  # series III
  instance_type      = "ecs.n4.large"
  system_disk_category = "cloud_efficiency"
  system_disk_name   = "test_foo_system_disk_name"
  system_disk_description = "test_foo_system_disk_description"
  image_id           = "ubuntu_18_04_64_20G_alibase_20190624.vhd"
  instance_name      = "test_foo"
  vswitch_id         = alicloud_vswitch.vswitch.id
  internet_max_bandwidth_out = 10
  data_disks {
    name      = "disk2"
    size      = 20
    category  = "cloud_efficiency"
    description = "disk2"
    encrypted  = true
    kms_key_id = alicloud_kms_key.key.id
  }
}
```

看起来好像很容易看懂但有的地方又有点疑惑，接着就要去查 alicloud provider documentation 各个参数的意思然后改参数，对于一个没有写过 Terraform 的人来说还是比较麻烦的。所以我们做了一个前端申请页面后，只需要选择必定的参数，Luban 后端就会按既定的规则在相应目录下进行代码生成并触发 GitOps 流程，对于申请人来说，只需要等审批结果了。



Elastic Compute Service

实例申请

1 Basic Configurations

2 Confirm

3 Finish

应用名称:

申请人:

申请人邮箱:

主机名: dev-所选应用名称- 请自定义主机名

VPC子网类型:

Image类型:

实例规格: CPU/MEM/系统盘

磁盘容量GB: 40

付费类型: charge type of instance.

安全组列表: sg-default 必选

备注: 请备注

下一步

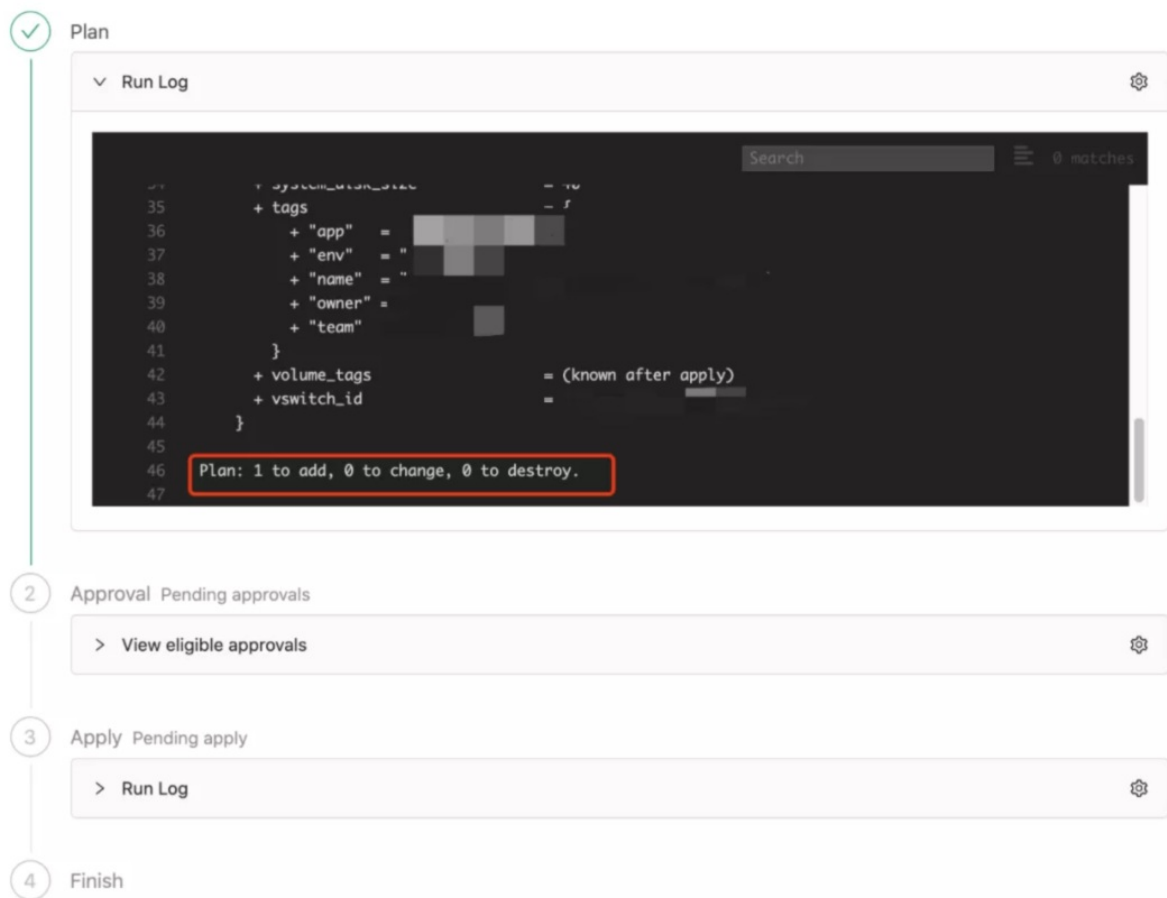
2) 运行Terraform Plan检查

这一步骤是后台自动执行的，触发 mobius webhook 进行 terraform plan。mobius 是 Luban 后端非常重要的一个引擎，它能够集成 GitLab 的 webhook, 处理 merge request 的 create/update/merge/cancel 事件，能够处理 Terraform 流程的 int/plan/apply, 并输出日志。

git 提交后，mobius 会自动进行 terraform plan 的 Pipeline。

¥3000	terraform-exa...	development	terraform-example	ECSInstance	Cancelled	test	2021-09-09 17:01:51	2021-09-08 14:26:48	Approve
-------	------------------	-------------	-------------------	-------------	-----------	------	---------------------	---------------------	---------

Tech Leader 通过前端的申请历史，可以看到对应资源申请的详细进展，查看 plan 结果是否符合预期，符合点击 approve 进入下个流程，由基础设施成员进行复审。



### 3) 提示值班人员进行资源变更审批

在Plan运行成功之后，自动进入到下一环节，Luban Bot模块提示Infra团队的值班人员进行资源变更审批。



### 4) 自动化对线上资源进行变更

Infra团队的值班人员在GitLab审批通过后，触发mobius webhook进行 terraform apply 即对线上做出变更，apply 成功后，mobius 自动进行代码 merge, 至此一个申请流程结束，而申请人也会在内部Chat上收到Luban Bot发出的资源申请成功的通知。

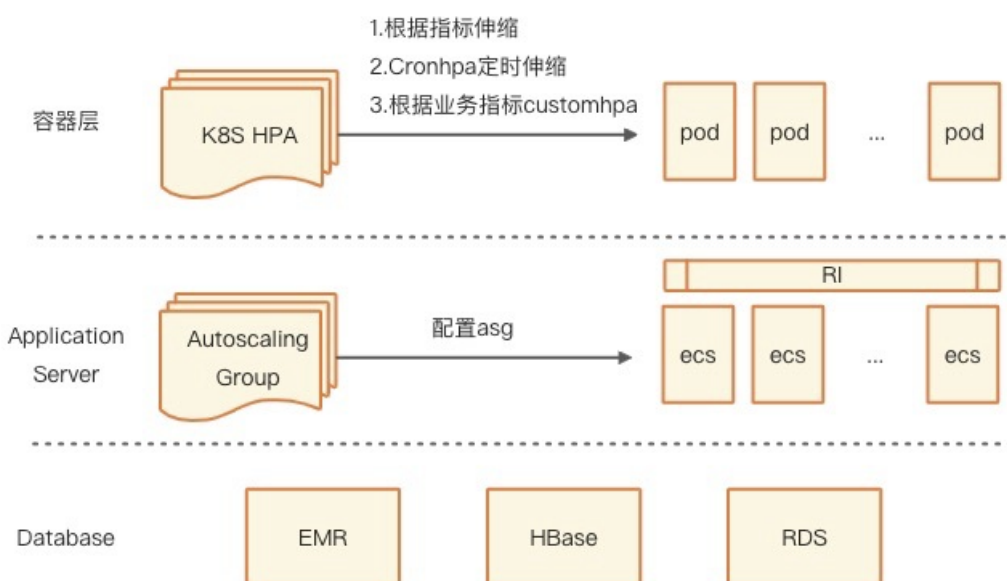
从以上四个步骤可以看出，资源的供应不仅是完全自动化的，而且非常的严谨，再也不需要人员专门去学习写 Terraform，重点就放在 review terraform plan 输出的变化是不是符合预期，并且通过ChatBot的触发性通知，提升了效率。

## 管理自动化：弹性伸缩自动化

我们的业务受到用户在一天内的使用习惯影响，在一天内不同时段呈现明显的波峰波谷；同时，定期的运营活动、市场促销活动也会带来用户访问的激增。在业务波峰时增加云资源、在业务波谷期释放资源，毫无疑问将可以帮助我们降低云上的成本。

如果只是根据业务活动计划手动调整资源，或者在负载高水位被动进行人工的配置，显然在对用户体验不是最佳的，也并不能动态实时响应降低成本。因此我们将开始着手研究如何实现弹性伸缩的自动化。

经过一段时间的摸索，我们做到了基于规则配置或业务指标监控根据业务的变化自动调节资源数量，实际应用效果来看，在相同的业务负载下，成本节约超过20%。



在技术上，我们从容器层、应用服务器层和数据库层各层都实现了自动化的弹性伸缩：

- 容器层：使用HPA进行pod伸缩，触发伸缩的事件有：监控指标、业务指标和定时任务；
- 服务器层：使用弹性伸缩组，基于服务器指标实现ECS的伸缩；
- 数据库层：使用云原生的数据库实现弹性，如EMR可定时或者根据cpu/mem指标弹出ECS。

## 治理自动化：成本管理自动化

对于流利说这样成长在云上的企业来说，成本管理是我们的关键任务。如何有效的管理云上开支、利用技术的手段杜绝资源浪费是我们解决的首要问题；而成本如何自动化的分摊到各个业务团队，是我们进行成本管理的基础。

### 1) 基于标签进行成本分摊

云上成本分摊其实没有想象中的那么复杂，云厂商在对应的云资源上面均有对应的 Tags 体系，K8S 的各种资源同样具有 Labels 体系，两个标签体系中其实是可以到很好的关联的；同时阿里云也有账单相关 API 可调用。

我们自研了Catalog系统，来进行资源与 App 的绑定，同时明确各个 App 的 Owner、Team，并且以此作为公司内部所有资源使用归属与统计的唯一数据源。至此我们已经具备了基于 Catalog 源数据管理来进行成本分摊的能力。

当资源有归属者转换，我们只需要修改 Catalog 系统即可，极致轻巧且高效。当然我们也有部分资源因为长久以往的积累，导致多个团队混用的情况发生。我们依靠大部分可以明确关系的资源均摊成本后，依靠此比例的准确性再来进行无法分摊的资源分摊，并与各个业务线达成共识。

2) 自动化的成本分析报告

对于公共支持团队，比如大数据、基础设施、业务中台等资源，我们通过在整体成本已经明确的业务线占比，再将公共支持团队成本均摊到各个业务线。同时我们也计算了研发环节对于整个成本在各个业务的占比情况，并将所有成本数据做好同比环比。

综上，基于 Catalog + Tags/Lables 进行明确资源关系，通过绝大部分精准的数据来进行分摊，解决公共资源的分摊难问题，最终每月有一份详细的自动化的成本分析报告给到各个业务线，同时也有相对简单的实时监控大盘。



此外，我们还基于 Prometheus 中各种资源的历史监控数据，统计了资源 CPU & Memory 利用率、存储资源 CPU & Memory & iops 利用率，做到了按每周为一个计算周期自动化发送，报表核心模板如下：

team	利用率小于30%占比	p80	p90	p99
cloud-infra	92.89%	44.10%	72.35%	100.00%
包含team下所有app的资源 数据来自catalog	1. 对多个指标做了合并处理 仅展示最大值 2. 利用率分布在0-30%的数据比例为92.89%，说明该team下绝大多数的资源利用率处于0-30%，处于相对空闲状态，可以考虑提示利用率	p80的利用率数据分布范围在0-44.1%，也就是说只有20%的数据可能大于44%利用率	p90的利用率数据分布范围在0-72.1%，也就是说只有10%的数据可能大于72%利用率，高峰期的业务已经有比较高的负载	p99的利用率数据分布范围0-100%，即高峰期的利用率已经达到了100%，处于比较繁忙状态

我们基于数据说话对低利用率资源进行合理的降配，至今没有出现任何反弹的情况、也没有出现任何线上生产事故。

总结

通过自动化的应用，对于我们Cloud Infra团队来说提高了工作的效率、提升我们的管理水平；同时对于业务也来带了显著的好处，包括更快的交付资源、更透明的成本开支。

作者介绍

---

流利说 技术部 Cloud Infra 团队