3 Terraform 基础知识

文档版本 01
发布日期 2023-07-03

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 Terraform 语法指南

1.1 基本语法

Terraform配置语言主要基于HCL语法，具有配置简单，可读性强等特点，并且兼容JSON语法。本文主要介绍Terraform配置语言的基本语法及常见函数。

Terraform配置语言主要由参数(Argument)，块(Block)，表达式(Experssion)和函数(Functions)组成。

参数 (Argument)

使用等号将一个值或表达式赋值给指定的参数名称，参数名称可以使用字母、数字、下划线(_)和连接符(-)表示，且首字母不能是数字，例如：

```
image_id = "ad091b52-742f-469e-8f3c-fd81cadf0743"
```

块 (Block)

块将多个参数聚合在一起，并支持嵌套。块由块类型、块标签和块主体构成，格式如下：

```
resource "huaweicloud_compute_instance" "myinstance" {
  name   = "myinstance"
  ......
  network {
    uuid = "55534eaa-533a-419d-9b40-ec427ea7195a"
  }
}
```

在使用块时必须先声明其对应的类型，样例中resource和network 均为块类型，其中resource为顶层块类型，network为嵌套块类型。Terraform支持的顶层块类型包括：provider，resource，data，variable，output，module，locals等关键字。

块标签在块类型之后定义，且数量由块类型决定，样例中resource块类型包含两个标签：huaweicloud_compute_instance和myinstance，嵌套的network类型没有块标

签。块主体定义在块最后，由 { 和 } 字符进行封装，在块主体内可以嵌套其他类型以实现不同的层级结构。

更多详细信息，请参见Terraform的 **配置语法**文档。

参数类型

Terraform支持以下参数类型：

基本类型

- **string**：字符串类型，由一个或多个Unicode字符组成，例如 "hello"。
- **number**：数字类型，可以表示整数和浮点数。
- **bool**：布尔类型，只能是 true 或 false。

Terraform能够根据参数类型自动将number和bool类型转换为string类型。如果一个字符串能够表示为一个数字或布尔类型的值，Terraform也可以进行反向转换。字符串、数字和布尔类型的参数可以直接赋值，例如：

```
disk_type = "SSD"
disk_size = 40
enable    = true

# 支持使用字符串表示数字和布尔类型
disk_size = "40"
enable    = "true"
```

集合类型

- **map(...)**：映射类型，以键值对(key-value pair) 的方式组合起来的数据元素集合，其中key为string类型，对应的值可以是string，number，bool等类型，且所有元素的值必须是同一类型。
- **list(...)**：列表类型，具有同类型的数据元素集合，元素可以是基本类型和块类型，列表索引从0开始。
- **set(...)**：集合类型，类似于列表类型，但是集合中的元素是没有任何辅助标识符或顺序，且元素具有唯一性。

映射类型使用 {} 封装，其表示形式非常灵活：键值对可以使用等号"="或冒号":"连接；如果key不以数字开头，可以不加双引号；对于多行映射，键值对之间可以使用换行符或者逗号进行分隔。推荐使用等号连接键值对并用换行符进行分隔，例如：

```
# 推荐格式
tags = {
  foo = "bar"
  key = "value"
}

# 其他格式
tags = {"foo" = "bar", "key" = "value"}
tags = {"foo" : "bar", "key" : "value"}
tags = {foo = "bar", key = "value"}
tags = {foo : "bar", key : "value"}
tags = {
  foo : "bar"
  key : "value"
}
```

列表类型和集合类型的表示方式相同，其中元素为基本类型的列表/集合使用 [] 封装，元素为块类型的列表/集合使用重复块的形式表示，例如：


```
# 基本类型的列表
security_groups = ["default", "internal"]

# 块类型的列表
network {
  uuid = "55534eaa-533a-419d-9b40-ec427ea7195a"
}
network {
  uuid = "ad091b52-742f-469e-8f3c-fd81cadf0743"
}
```

特殊类型

- **null：**空类型，如果将一个参数设置为null，表示这个参数未填写，Terraform会自动忽略该参数，并使用默认值。null在条件表达式中较为常见，如var.test=="" ? null : var.test，表示当var.test的值为""时，就将其忽略。

其他语法

- 单行注释以#或//开头；
- 多行注释以/*开始，以*/结束，不支持嵌套块注释。
- Terraform配置文件使用UTF-8编码，对于标识符、注释和字符串都支持非ASCII字符。
- 多行字符串在一行末尾以<<EOF开头，中间是字符串内容，最后以EOF结尾。EOF也可以替换为其他字符。例如：

```
resource "huaweicloud_obs_bucket" "web_bucket" {
 ...
 website {
   ...
   routing_rules = <<EOF
[{
   "Condition": {
       "KeyPrefixEquals": "docs/"
   },
   "Redirect": {
       "ReplaceKeyPrefixWith": "documents/"
   }
}]
EOF
 }
}
```

1.2 样式约定

样式约定

Terraform约定了一些惯用的风格样式，以确保不同团队编写的文件和模块的风格一致性。建议用户遵循这些约定，推荐的样式约定如下：

- 对于每个嵌套级别，缩进两个空格。
- 当多个单行的参数在同一嵌套级别连续出现时，建议将等号对齐。

```
name            = "myinstance"
security_groups = ["default", "internal"]
```

- 使用空行分隔块中的逻辑参数组。
- 当块主体同时包含参数和块时，建议将所有参数放在顶部，嵌套块放在参数的下方并使用空行隔开。

- 将元参数(meta-arguments) 放在块主体的顶部，并使用空行与其它参数隔开；将元参数块(meta-argument blocks) 放在块主体的最后，并用空行与其他块隔开。

```
resource "huaweicloud_obs_bucket" "demo" {
  count = 1

  bucket = "bucket_demo"
  acl    = "public-read"

  tags = {
    foo = "bar"
    env = "test"
  }

  lifecycle {
    create_before_destroy = true
  }
}
```

- 顶层块之间使用空行将彼此隔开。
- 建议将相同类型的嵌套块放在一起，不同类型的嵌套块使用空行隔开。

参考资料

https://www.terraform.io/docs/configuration/style.html

1.3 表达式

表达式用于引用或计算配置中的值，最简单的表达式是文字表达式，如 "hello world" 或5。Terraform支持多种表达式，包括运算符、条件表达式以及丰富的内置函数。

通过 "terraform console" 命令可以打开一个交互式的控制台，我们可以使用该控制台进行表达式及内置函数的体验和测试。

运算符

运算符是执行特定的数学或逻辑操作的服务，Terraform支持以下类型的运算符：

- 算术运算符：操作数和结果都为数字类型，包括：+，-（减法），*，/，%，-（负数）。
- 关系运算符：操作数为任意类型，结果为布尔值，包括：==，！=。
- 比较运算符：操作数为数字类型，结果为布尔值，包括：>，>=，<，<=。
- 逻辑运算符：操作数和结果都为布尔类型，包括：||，&&，！。

在表达式中使用多个运算符时，将按照以下优先级进行求解：

1. !, - (负数)
2. *, /, %
3. +, - (减法)
4. >, >=, <, <=
5. ==, !=
6. &&
7. ||

条件表达式

条件表达式采用布尔表达式的值进行二选一，其语法可以表示为：

```
condition ? true_value : false_value
```

该语句表示：如果condition为true，结果为true_value，否则为false_value。条件表达式的结果可以是任意类型，但true_value和false_value的类型必须保持一致。条件表达式的常见用法是使用默认值替换无效值，如下：

```
var.a != "" ? var.a : "default-a"
```

该语句表示：如果var.a的值不为空，则返回var.a的值，否则返回一个默认值。

for 表达式

for表达式用于遍历集合类型 (map、list、set) 中的每个元素，并对元素进行处理，最后将结果输出为一个新的集合类型。for表达式的输出结果取决于所使用的括号类型：

- 使用 '[' 和 ']' 将生成一个列表
- 使用 '{' 和 '}' 将生成一个映射/对象

假设列表 mylist 的值为 ["AA", "BBB", "CCCC"]，我们可以使用for表达式对 mylist 中的每个字符串元素转换为小写，并输出一个列表：

```
> [for str in var.mylist : lower(str)]
[
  "aa",
  "bbb",
  "cccc",
]
```

我们也可以将结果输出为一个映射，映射关系通过 "=>" 确定：

```
> {for str in var.mylist : str => lower(str)}
{
  "AA" = "aa"
  "BBB" = "bbb"
  "CCCC" = "cccc"
}
```

映射类型也可以通过for表达式转换进行处理，假设 mymap 的值为 {element1="aaa", element2="bbb", element3="ccc"}，我们可以将映射中的每个键值转换为大写：

```
> {for key, value in var.mymap : key => upper(value)}
{
  "element1 = "AAA"
  "element2 = "BBB"
  "element3 = "CCC"
}
```

此外，for表达式还可以使用if语句对元素进行过滤，假设 mylist 的值为 ["aa", "bbb", "cccc"]，我们可以将所有满足if条件的元素转为大写：

```
> [for str in var.mylist : upper(str) if length(str) >= 3]
[
  "BBB",
  "CCCC",
]
```

参考资料

https://www.terraform.io/docs/configuration/expressions.html

1.4 常见函数

Terraform支持丰富的内置函数，用于处理字符串、数值计算、加密，类型转换等操作，我们可以通过函数名称进行调用，其语法如下：

```
<函数名称>(<参数1>, <参数2> ...)
```

本文主要对Terraform中常见的函数进行总结并通过样例说明其用法。您可以通过Terraform 官方文档查看完整的函数支持列表。

字符串函数

表 1-1 字符串函数

函数名称	函数描述	样例	运行结果
format	字符串格式化	format("Hello, %s!", "cloud")	Hello, cloud!
lower	将字符串中的字母转换为小写	lower("HELLO")	hello
upper	将字符串中的字母转换为大写	upper("hello")	HELLO
join	使用自定义字符将列表拼接成字符串	join(", ", ["One", "Two", "Three"])	One, Two, Three
split	根据分隔符拆分字符串	split(", ", "One, Two, Three")	["One", "Two", "Three"]
substr	通过偏移量和长度从给定的字符串中提取一个子串	substr("hello world!", 1, 4)	ello
replace	把字符串中的str1替换成str2	replace("hello, cloud!", "h", "H")	Hello, cloud!

数值计算函数

表 1-2 数值计算函数

函数名称	函数描述	样例	运行结果
abs	计算绝对值	abs(-12.4)	12.4
max	计算最大值	max(12, 54, 6) max([12, 54, 6]...)	54 54
min	计算最小值	min(12, 54, 6) min([12, 54, 6]...)	6 6

函数名称	函数描述	样例	运行结果
log	计算对数	log(16, 2)	4
power	计算x的y次幂	power(3, 2)	9

集合函数

表 1-3 集合函数

函数名称	函数描述	样例	运行结果
element	通过下标从列表中检索对应元素值	element(["One", "Two", "Three"], 1)	Two
index	返回给定值在列表中的索引，如果该值不存在将报错。	index(["a", "b", "c"], "b")	1
lookup	使用给定的键从映射表中检索对应的值。如果给定的键不存在，则返回默认值。	lookup({IT="A", CT="B"}, "IT", "G") lookup({IT="A", CT="B"}, "IE", "G")	A G
flatten	展开列表中的嵌套元素	flatten([["a", "b"], [], ["c"]])	["a", "b", "c"]
keys	返回map中的所有key	keys({a=1, b=2, c=3})	["a", "b", "c"]
length	获取列表、映射或是字符串的长度	length(["One", "Two", "Three"]) length({IT="A", CT="B"}) length("Hello, cloud!")	3 2 13

类型转化函数

表 1-4 类型转化函数

函数名称	函数描述	样例	运行结果
toset	将列表类型转换为集合类型	toset(["One", "Two", "One"])	["One", "Two"]
tolist	将集合类型转换为列表类型	tolist(["One", "Two", "Three"])	["One", "Two", "Three"]

函数名称	函数描述	样例	运行结果
tonumber	将字符串类型转换为数字类型	tonumber("33")	33
tostring	将数字类型转换为字符串类型	tostring(33)	"33"

编码函数

表 1-5 编码函数

函数名称	函数描述	样例	运行结果
base64encode	将UTF-8字符串转换为base64编码	base64encode("Hello, cloud!")	SGVsbG8sIGNsb3VkIQ==
base64decode	将base64编码解码为UTF-8字符串(结果非UTF-8格式会报错)	base64decode("SGVsbG8sIGNsb3VkIQ==")	Hello, cloud!
base64gzip	将UTF-8字符串压缩并转换为base64编码	base64gzip("Hello, cloud!")	H4sIAAAAAAAA//JIzcnJ11FIzskvTVEEAAAA//8BAAD//wbrhYUNAAAA

哈希和加密函数

表 1-6 哈希和加密函数

函数名称	函数描述	样例	运行结果
sha256	计算字符串的SHA256值（16进制）	sha256("Hello, cloud!")	0ad167d1e3ac8e9f4e4f7ba83e92d0e3838177e959858631c770caaed8cc5e3a
sha512	计算字符串的SHA512值（16进制）	sha512("Hello, cloud!")	6eb6ed9fc4edffaf90e742e7697f6cc7d8548e98aa4d5aa74982e5cdf78359e84a3ae9f226313b2dec765bf1ea4c83922dbfe4a61636d585da44ffbd7e900f56
base64sha256	计算字符串的SHA256值，并转换为base64编码	base64sha256("Hello, cloud!")	CtFn0eOsjp9OT3uoPpLQ44OBd+lZhYYxx3DKrtjMXjo=

函数名称	函数描述	样例	运行结果
base64sha512	计算字符串的SHA512值，并转换为base64编码	base64sha512("Hello, cloud!")	brbtn8Tt/6+Q50LnaX9sx9hUjpiqTVqnSYLlzfeDWehKOunyJjE7Lex2W/HqTIOSLb/kphY21YXaRP+9fpAPVg==
md5	计算MD5值	md5("hello world")	5eb63bbbe01eeed093cb22bb8f5acdc3

说明

base64sha512("Hello, cloud!")不等于base64encode(sha512("Hello, cloud!"))，因为sha512计算的十六进制值结果在Terraform中是Unicode编码格式，并没指定UTF-8实现。

文件操作函数

表 1-7 文件操作函数

函数名称	函数描述	样例	运行结果
abspath	计算文件的绝对路径	abspath("./hello.txt")	/home/demo/test/terraform/hello.txt
dirname	计算字符串中包含的路径	dirname("foo/bar/baz.txt")	foo/bar
basename	计算字符串中的文件名	basename("foo/bar/baz.txt")	baz.txt
file	读取文件并返回文件内容	file("./hello.txt")	Hello, cloud!
filebase64	读取文件并返回文件内容的base64编码	filebase64("./hello.txt")	SGVsbG8sIGNsb3VkIQ==

2 Terraform 配置指南

2.1 Provider

Terraform的配置文件以 ".tf" 结尾，主要由**provider**，**resource**，**data source**和**变量**组成。

每个 Provider 代表一个服务提供商，Terraform 通过插件机制与Provider进行交互。Provider通过关键字 "provider" 进行声明，Provider的配置参数请参考**这里**。

执行 terraform init 命令时会下载使用的插件，默认将从Terraform官方仓库下载最新版本的插件。对于Terraform 0.13之后的版本，可以使用 "required_providers" 指定Provider的 registry 源和版本。

```
terraform {
  required_providers {
    huaweicloud = {
      source = "huaweicloud/huaweicloud"
      version = ">= 1.20.0"
    }
  }
  required_version = ">= 0.13"
}
```

在Terraform中，我们可以使用 provider块创建多个配置，其中一个 provider块为默认配置，其它块使用 "alias" 标识为非默认配置。在资源中使用元参数 provider 可以选择非默认的 provider块。例如我们需要在不同的地区管理资源，首先需要声明多个provider块：


```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

provider "huaweicloud" {
  alias  = "guangzhou"
  region = "cn-south-1"
  ...
}
```

示例中我们声明了北京和广州的华为云provider，并对广州地区的provider增加了别名。我们在资源中使用元参数 provider 来选择非默认的 provider块，其格式为：<provider名称>.<别名>。

```
resource "huaweicloud_vpc" "example" {
  provider = huaweicloud.guangzhou
  name     = "terraform_vpc"
  cidr     = "192.168.0.0/16"
}
```

华为云Provider 支持在Resource中指定region参数，可以在不同的地区创建资源。相比 alias + provider 的方式，这种方式更加灵活简单。

```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

resource "huaweicloud_vpc" "example" {
  region = "cn-south-1"
  name   = "terraform_vpc"
  cidr   = "192.168.0.0/16"
}
```

2.2 Resource

Resource 是Teraform中最重要的元素，通过关键字 "resource" 进行声明。Provider中支持的云服务都有一个或多个资源与之对应，如huaweicloud_compute_instance表示ECS，huaweicloud_vpc表示VPC等。资源之间的关系可以通过关系型资源进行关联，如我们可以使用 huaweicloud_compute_eip_associate 给ECS分配EIP。

```
resource "huaweicloud_compute_instance" "myinstance" {
  ...
}

resource "huaweicloud_vpc_eip" "myeip" {
  ...
}

resource "huaweicloud_compute_eip_associate" "associated" {
  public_ip   = huaweicloud_vpc_eip.myeip.address
  instance_id = huaweicloud_compute_instance.myinstance.id
}
```

资源引用

我们可以通过表达式引用资源的属性，格式为：<资源类型>.<名称>.<属性>。假设我们已经创建了名称为 myinstance 的 huaweicloud_compute_instance 资源，举例如下：

```
# 实例ID
> huaweicloud_compute_instance.myinstance.id
```



```
55534eaa-533a-419d-9b40-ec427ea7195a

# 实例安全组
> huaweicloud_compute_instance.myinstance.security_groups
["default", "internet"]

# 实例第一个网卡的IP地址
> huaweicloud_compute_instance.myinstance.network[0].fixed_ip_v4
192.168.0.245

# 实例所有网卡的IP地址
huaweicloud_compute_instance.myinstance.network[*].fixed_ip_v4
["192.168.0.24", "192.168.10.24"]

# 标签key的值
> huaweicloud_compute_instance.myinstance.tags["key"]
value
```

2.3 Data Source

Data Source 可以认为是特殊的Resource，通过关键字 "data" 进行声明。Data Source 用于查询已经存在资源的属性和信息，例如我们可以通过 "Ubuntu 18.04 server 64bit" 的镜像名称查询得到对应镜像的ID及其他属性：

```
data "huaweicloud_images_image" "myimage" {
  name = "Ubuntu 18.04 server 64bit"
}
```

查询到镜像之后，我们可以引用该镜像的属性供其他资源使用，引用格式为：data.<数据类型>.<名称>.<属性>

```
resource "huaweicloud_compute_instance""flexibleengine_compute_instance_v2" "demo" {
  name      = "ecs-demo"
  image_id = data.huaweicloud_images_image.myimage.id
  ...
}
```

2.4 变量

2.4.1 输入变量

输入变量可以理解为模块的参数，通过关键字 "variable" 进行声明。通过定义输入变量，我们可以无需变更模块的源代码就能灵活修改配置。输入变量的值可以使用默认值，CLI 选项，环境变量等方式来设置。

定义输入变量

按照约定，输入变量通常在名为 variables.tf 的文件中定义。输入变量通过关键字 "variable" 进行声明：

```
variable "iamge_id" {
  type        = string
  description = "image id of Ubuntu 1804"
}

variable "availability_zone_name" {
  type    = string
  default = "cn-north-1a"
}
```


variable 关键字后的标签是输入变量的名称，该名称在同一模块中的所有变量之间必须唯一。变量的名称可以是除保留关键字以外的任何有效的标识符。保留关键字包括：

```
source  version  providers  count  for_each  lifecycle  depends_on  locals
```

variable块中主要包括如下参数：

- type：指定变量的类型，默认为 string。
- description ： 指定变量的描述信息，用于描述变量的用途。
- default：指定变量的默认值，存在默认值的变量可视为可选变量。
- validation块：指定变量的自定义验证规则。

如果未明确指定变量类型，则默认为 string。建议开发者显式指定变量类型，这样可以方便地提醒用户如何使用该模块，并允许Terraform在使用错误的类型后返回有用的错误信息。Terraform 输入变量支持的类型有：

- 基本类型：string，number，bool
- 复合类型：list(<TYPE>)，set(<TYPE>)，map(<TYPE>)

复合类型的变量定义如下：

```
variable "availability_zone_names" {
  type    = list(string)
  default = ["cn-north-1a"]
}

variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  default = [{
    internal = 8300
    external = 8300
    protocol = "tcp"
  }]
}
```

自定义验证规则

我们可以使用 validation嵌套块为输入变量指定自定义验证规则，该特性在 Terraform 0.13.0之后的版本支持，例如：

```
variable "iam_user_password" {
   type        = string
   description = "The password for iam user to log in."

   validation {
     condition     = length(var.iam_user_password)>=8
     error_message = "The password is too short."
   }
}
```

其中，condition 参数是一个布尔表达式，我们可以使用 can 函数来检测表达式是否会产生错误，例如：

```
variable "iam_user_name" {
   type        = string
   description = "This name is used for iam user to log in."
```



```
  validation {
    # regex(...) 如果匹配失败将返回错误
    condition     = can(regex("([a-zA-Z])", var.iam_user_name))
    error_message = "Incorrect user name. Please check whether it contains upper and lower case letters."
  }
}
```

如果condition 的结果为false，Terraform 将产生一条错误消息，其内容为 error_message 所定义的字符串。error_message 应该至少是一个完整的句子，以大写字母开头，以 "." 或者 "?" 结尾。

引用输入变量

输入变量可以通过 var.<变量名称> 的形式访问，且只能在声明该变量的模块内访问：

```
# variables.tf
variable "vpc_cidr" {
  type        = string
  description = "the CIDR of VPC"
}

# main.tf
resource "huaweicloud_vpc" "vpc_example" {
    name = "my_vpc"
    cidr = var.vpc_cidr
}
```

设置变量

通过如下方式可以设置输入变量：

- 通过命令行中 -var 选项指定
- 通过变量定义文件 (.tfvars)，在命令行中指定或自动加载
- 设置环境变量

在 "terraform plan" 和 "terraform apply" 命令行中，通过 -var 选项可以指定变量。其中，-var 选项可以多次使用，例如：

```
terraform apply -var='vpc_name=my_vpc'
terraform apply -var='vpc_name=my_vpc' -var='vpc_cidr=192.168.0.0/16'
terraform apply -var='availability_zone_names=["cn-north-1a", "cn-north-1c"]'
```

注意：变量名称和等号之间不能有空格。

变量定义 (.tfvars) 文件

如果配置中使用了很多变量，建议使用变量定义文件来设置这些变量，然后通过 -var-file 选项指定该文件：

```
terraform apply -var-file="testing.tfvars"
```

变量定义文件的扩展名为 ".tfvars"，变量定义文件的语法与配置文件的语法相同，但仅用于指定变量名称：

```
vpc_name = "my_vpc"
vpc_cidr = "192.168.0.0/16"
availability_zone_names = [
    "cn-north-1a","cn-north-1c",
]
```

Terraform 还会自动加载特殊命名的变量定义文件：

- 文件名为 terraform.tfvars 或 terraform.tfvars.json 的文件
- 文件名称以 .auto.tfvars 或 .auto.tfvars.json 结尾的文件

对于以 .json 结尾的文件，需要使用 JSON对象表示：

```
{
   "vpc_name": "my_vpc"
   "availability_zone_names": ["cn-north-1a", "cn-north-1c"]
}
```

环境变量

我们可以通过设置以 "TF_VAR_" 为前缀的环境变量来指定置输入变量。这对以自动化的方式运行或连续运行使用相同变量的 Terraform 命令很有帮助。

```
$ export TF_VAR_vpc_name=my_vpc
$ export TF_VAR_availability_zone_names='["cn-narth-1a", "cn-north-1c"]'
$ terraform plan
...
```

变量定义优先级

我们可以自由组合使用上述设置变量的方式。对于复合类型的变量，为了提高可读性并避免转义带来的问题，建议使用变量定义文件来设置。如果我们为同一个变量分配了多个值，Terraform 将使用最后一个值进行覆盖。Terraform 根据以下顺序加载变量(根据顺序，后面的源优于前面的源)：

1. 环境变量
2. terraform.tfvars 或 terraform.tfvars.json 文件
3. *.auto.tfvars 或 *.auto.tfvars.json 文件
4. 命令行中的 -var 和 -var-file 选项

注：不能在单个源中为同一个变量分配多个值。

有关变量的更多信息，请参见Terraform的 **输入变量**文档。

2.4.2 输出变量

输出变量可以理解为模块的返回值，通过关键字 "output" 进行声明。输出变量是一种对外公开某些信息的方法，既可以在根模块中运行 terraform apply/output 命令输出特定的值，又可以在子模块中将资源的属性值提供给父模块。

声明输出变量

按照约定，输出变量通常在名为 variables.tf 的文件中定义。输出变量通过“output”关键字进行声明：

```
output "ecs_address" {
  value       = huaweicloud_compute_instance.myinstance.network[0].fixed_ip_v4
  description = "The private IP address of my ECS"
}
```

output 关键字后的标签为输出变量的名称，该名称必须是有效的标识符。output块中主要包括以下参数：

- value：必选项，输出变量的值，任何有效的表达式都可作为输出使用。

- description：输出变量的描述信息，用于描述输出变量的用途。

```
output "vpc_id" {
  value       = huaweicloud_vpc.myvpc.id
  description = "Check out the VPC ID"
}

$ terraform output
vpc_id = df507d37-bce2-4750-8873-f62abb3b085c
```

- sensitive：将输出变量标记为敏感项，在 CLI 中将隐藏输出变量值的显示。

```
output "vpc_id" {
  value       = huaweicloud_vpc.myvpc.id
  description = "Check out the VPC ID"
  sensitive   = true
}

$ terraform output
vpc_id = <sensitive>
```

  注意：标记为敏感项的输出变量在输出时会自动被隐藏，但其输出值仍然可以通过以下方式可见：
  - 输出变量的值记录在 state 文件中，其值对所有能够访问state 文件的用户均可见。
  - 子模块中敏感输出变量值被父模块调用，通过父模块的相关输出和资源引用后可以在CLI中显示。

- depends_on：指定输出变量的依赖关系。由于输出变量只是导出数据的一种手段，因此通常不需要设置与其他资源、数据的依赖关系。

2.4.3 本地变量

本地变量可以理解为模块中的临时变量，其作用范围在所声明的模块内，通过关键字 "local" 进行声明。本地变量适用于配置中有重复定义相同值或表达式的场景，可以减少代码冗余，并且易于修改。同时过度使用本地变量会导致变量的实际值被隐藏，代码晦涩，不利于维护，因此建议合理使用本地变量。

声明本地变量

本地变量通过“locals”关键字进行声明：

```
locals {
  service_name = "forum"
  owner        = "Community"
}
```

本地变量的表达式不仅限于字符和数值常量，还可以使用输入变量、资源属性和其他本地变量的引用和表达式结果：

```
locals {
  dns_list = concat(huaweicloud_vpc_subnet.subnet_1.dns_list, huaweicloud_vpc_subnet.subnet_2.dns_list)
}

locals {
  common_tags = {
    Service = local.service_name
    Owner   = local.owner
  }
}
```

引用本地变量

在声明本地变量后，可以通过 local.<变量名称> 对其进行引用。


```
resource "huaweicloud_obs_bucket" "bucket_demo" {
  ...
  tags = local.common_tags
}
```

2.5 Metadata

2.5.1 Metadata 说明

Metadata是Terraform支持的内置元参数，可以在 provider，resource，data块中使用。本章节主要介绍 resource块支持的元参数，主要包括：

- depends_on：用于指定资源的依赖项
- count：用于创建多个相同配置的资源
- for_each：用于根据映射、字符串集合创建多个资源
- provider：用于选择非默认的 provider
- lifecycle：用于定制资源的生命周期

2.5.2 depends_on

在同一个 Terraform 配置文件中可以包含多个资源。通过在资源中引用其他资源的属性值，Terraform可以自动推断出资源的依赖关系。然而，某些资源的依赖关系对于Terraform是不可见的，这就需要使用 depends_on 来创建显式依赖。我们可以使用 depends_on 来更改资源的创建顺序或执行顺序，使其在所依赖资源之后处理。depends_on 的表达式是依赖资源的地址列表。例如我们在远程操作一台ECS服务器之前，需要为其绑定EIP或配置NAT规则。

```
resource "huaweicloud_compute_instance" "myinstance" {
  ...
}

resource "huaweicloud_vpc_eip" "myeip" {
  ...
}

resource "huaweicloud_compute_eip_associate" "associated" {
  public_ip   = huaweicloud_vpc_eip.myeip.address
  instance_id = huaweicloud_compute_instance.myinstance.id
}

resource "null_resource" "provision" {
  depends_on = [huaweicloud_compute_eip_associate.associated]

  provisioner "remote-exec" {
    connection {
      # 通过公网地址访问 ECS
      host = huaweicloud_vpc_eip.myeip.address
      ...
    }
    inline = [
      ...
    ]
  }
}
```

2.5.3 count

默认情况下，Terraform的 resource块只配置一个资源。当我们需要创建多个相同的资源时，如果配置多个独立的 resource块就显得很冗余，且不利于维护。我们可以使用

count 或 for_each 参数在同一个 resource块中管理多个相同的资源。在同一个 resource块中不能同时使用count 和 for_each 参数。示例如下：

```
resource "huaweicloud_evs_volume" "volumes" {
  count = 3

  size              = 20
  volume_type       = "SSD"
  availability_zone = "cn-north-4a"
}
```

我们通过如上配置创建了3个相同的云硬盘（EVS）。在很多情况下，Provider 要求创建资源的某些参数具有唯一性，这时我们可以使用 "count.index" 属性来进行区分，这是一个从0开始计数的索引值。

```
resource "huaweicloud_vpc" "vpcs" {
  count = 2
  name  = "myvpc_${count.index}"
  cidr  = "192.168.0.0/16"
}
```

我们通过如上配置创建了两个VPC，名字分别为 myvpc_0 和 myvpc_1，它们具有相同的CIDR值。如果进一步修改CIDR值，我们可以声明一个string列表用于存储不同VPC的CIDR值，然后通过 count.index 去访问列表元素。

```
variable "name_list" {
  type    = list(string)
  default = ["vpc_demo1", "vpc_demo2"]
}
variable "cidr_list" {
  type    = list(string)
  default = ["192.168.0.0/16", "172.16.0.0/16"]
}

resource "huaweicloud_vpc" "vpcs" {
  count = 2
  name  = var.name_list[count.index]
  cidr  = var.cidr_list[count.index]
}
```

使用 count 创建的资源需要通过索引值进行访问，格式为：<资源类型>.<名称>[索引值]

```
# 访问第一个VPC
> huaweicloud_vpc.vpcs[0]

# 访问第一个VPC的ID
> huaweicloud_vpc.vpcs[0].id

# 访问所有VPC的ID
> huaweicloud_vpc.vpcs[*].id
```

2.5.4 for_each

for_each 在功能上与 count 相似，for_each 使用键值对或字符串集合的形式快速地将值填入到对应的属性中，不仅可以优化脚本结构也有利于理解多实例间的关系。

在使用映射类型表达时，我们可以使用 "each.key" 和 "each.value" 来访问映射的键和值。以创建VPC为例，通过 for_each 中的键值对，我们可以灵活配置VPC的名称和CIDR。

```
resource "huaweicloud_vpc" "vpcs" {
  for_each = {
    vpc_demo1 = "192.168.0.0/16"
    vpc_demo2 = "172.16.0.0/16"
}
```



```
  name = each.key
  cidr = each.value
}
```

在使用字符串集合类型表达时，"each.key" 等同于 "each.value"，我们一般使用each.key表示，另外，可以通过 toset() 函数将定义的 list 类型进行转化：

```
resource "huaweicloud_networking_secgroup" "mysecgroup" {
  for_each = toset(["secgroup_demo1", "secgroup_demo2"])
  name     = each.key
}

# 通过变量表示 for _each
variable "secgroup_name" {
  type = set(string)
}
resource "huaweicloud_networking_secgroup" "mysecgroup" {
  for_each = var.secgroup_name
  name     = each.key
}
```

使用 for_each 创建的资源需要通过键名进行访问，格式为：<资源类型>.<名称>[键名]

```
# 访问 vpc_demo1
> huaweicloud_vpc.vpcs["vpc_demo1"]

# 访问 vpc_demo1 的ID
> huaweicloud_vpc.vpcs["vpc_demo1"].id
```

由于 count 和 for_each 都可用于创建多个资源，建议参考以下规则进行选择：

1、如果资源实例的参数完全或者大部分一致，建议使用count；

2、如果资源的某些参数需要使用不同的值并且这些值不能由整数派生，建议使用for_each；

2.5.5 provider

在Terraform中，我们可以使用 provider块创建多个配置，其中一个 provider块为默认配置，其它块使用 "alias" 标识为非默认配置。在资源中使用元参数 provider 可以选择非默认的 provider块。例如我们需要在不同的地区管理资源，首先需要声明多个provider块：

```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

provider "huaweicloud" {
  alias  = "guangzhou"
  region = "cn-south-1"
  ...
}
```

示例中我们声明了北京和广州的华为云provider，并对广州地区的provider增加了别名。我们在资源中使用元参数 provider 来选择非默认的 provider块，其格式为：<provider名称>.<别名>。

```
resource "huaweicloud_networking_secgroup" "mysecgroup" {
  # 使用非默认 provider块名，对应非默认provider块的别名(alias)
  provider = huaweicloud.guangzhou
  ...
}
```


华为云Provider 支持在Resource中指定region参数，可以在不同的地区创建资源。相比 alias + provider 的方式，这种方式更加灵活简单。

```
provider "huaweicloud" {
  region = "cn-north-1"
  ...
}

resource "huaweicloud_vpc" "example" {
  region = "cn-south-1"
  name   = "terraform_vpc"
  cidr   = "192.168.0.0/16"
}
```

2.5.6 lifecycle

每个资源实例都具有创建 、更新和销毁三个阶段，在一个资源实例的生命周期过程中都会经历其中的2至3个阶段。通过元参数 lifecycle 可以对资源实例的生命周期过程进行改变，lifecycle 支持以下参数：

- **create_before_destroy**

  默认情况下，当我们需要改变资源中不支持更新的参数时，Terraform会先销毁已有实例，再使用新配置的参数创建新的对象进行替换。当我们将 create_before_destroy 参数设置为 true 时，Terraform将先创建新的实例，再销毁之前的实例。这个参数可以适用于保持业务连续的场景，由于新旧实例会同时存在，需要提前确认资源实例是否有唯一的名称要求或其他约束。

  ```
  lifecycle {
    create_before_destroy = true
  }
  ```

- **prevent_destroy**

  当我们将 prevent_destroy 参数设置为true时，Terraform将会阻止对此资源的删除操作并返回错误。这个元参数可以作为一种防止因意外操作而重新创建成本较高实例的安全措施，例如数据库实例。如果要删除此资源，需要将这个配置删除后再执行 destroy 操作。

  ```
  lifecycle {
    prevent_destroy = true
  }
  ```

- **ignore_changes**

  默认情况下，Terraform plan/apply 操作将检测云上资源的属性和本地资源块中的差异，如果不一致将会调用更新或者重建操作来匹配配置。我们可以用 ignore_changes 来忽略某些参数不进行更新或重建。ignore_changes 的值可以是属性的相对地址列表，对于 Map 和 List 类型，可以使用索引表示法引用，如 tags["Name"]，list[0] 等。

  ```
  resource "huaweicloud_rds_instance" "myinstance" {
    ...
    lifecycle {
      ignore_changes = [
        name,
      ]
    }
  }
  ```

  此时，Terraform 将会忽略对 name 参数的修改。除了列表之外，我们也可以使用关键字 all 忽略所有属性的更新。

  ```
  resource "huaweicloud_rds_instance" "myinstance" {
    ...
    lifecycle {
      ignore_changes = all
  ```



```
  }
}
```

2.6 Backend

默认情况下，通过Terraform完成资源的创建和修改后，会将资源的状态和属性信息会保存在当前目录下的 terraform.tfstate 文件中。这个 state 文件可以看作是Terraform存储资源属性的映射，当执行 "terraform show" 命令时，Terraform直接读取这个 state 文件，无需再去调用云平台的API查询。

Terraform后续的 'plan' 和 'apply' 操作，都是基于当前的模块配置和 state 文件进行比较。如果 state 文件被损坏或者被删除，Terraform会认为其管理的资源也发生了变更和移除。此时再执行 'apply' 命令将会按照模块的定义变更或者重建资源，直到模块对资源的定义与 state 中的映射保持一致。如果实际的资源依然存在于云平台中，这将会导致资源的重复创建或者创建失败。

在团队协作中，如果想维护同一套资源，需要将资源的配置文件和 state 文件一起拷贝，这无形中增加了代码维护的成本。为了解决这个问题，Terraform提供了远端存储的能力 ，即将 state 文件存放在远端的一个服务中，并支持锁定，实现代码与 state 的管理分离，提升了管理的灵活性。

Backend 是实现远端存储的机制。通过在 terraform块中声明 backend嵌套块，并指定不同的**Backend 类型**标签，可以将 state 文件存储在不同的远端服务中，如AWS S3，HashiCorp Consul，etcd等。

在华为云中使用 S3 Backend

由于华为云的OBS (对象存储服务) 兼容AWS S3接口，且S3 Backend支持自定义endpoint，我们可以利用S3 Backend将 state 文件存储在华为云OBS桶中。操作步骤如下：

步骤1 在terraform块中配置backend

```
terraform {
  backend "s3" {
    bucket   = "backendbucket"
    key      = "myproject/terraform.tfstate"
    region   = "cn-north-1"
    endpoint = "obs.cn-north-1.myhuaweicloud.com"

    skip_region_validation      = true
    skip_metadata_api_check     = true
    skip_credentials_validation = true
  }
}
```

参数说明：

- bucket：桶名称
- key：对象名称；
- region：OBS桶所属地区；
- endpoint：OBS为每个区域提供的终端节点，各区域的终端节点详情参照华为云**地区和终端节点**；
- skip*：调用AWS S3 API时跳过部分参数的校验和检查；

步骤2 定义AK/SK环境变量

OBS服务通过AK/SK对请求进行认证，用户可以在IAM服务中获取AK和SK，获取方法请参见**华为AK/SK获取访问密钥**。然后将AK/SK导出为环境变量：

```
$ export AWS_ACCESS_KEY_ID="******"
$ export AWS_SECRET_ACCESS_KEY="******"
```

步骤3 执行 terraform init 命令

配置完Backend后，需要执行terraform init命令对Backend进行初始化，首次执行时会有以下提示：

```
$ terraform init

Initializing the backend...
Backend configuration changed!

Terraform has detected that the configuration specified for the backend
has changed. Terraform will now check for existing state in the backends.

Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.

Initializing provider plugins...
......
Terraform has been successfully initialized!
```

如果初始化失败，则会有以下提示：

```
Error:No valid credential sources found for AWS Provider.
Please see https://terraform.io/docs/providers/aws/index.html for more information on
providing credentials for the AWS Provider
```

此时应检查Backend的配置参数和环境变量是否正确。如果当前目录下已经存在"terraform.tfstate"文件，则会提示是否将这个"state"文件同步至远端。

```
$ terraform init

Initializing the backend...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "s3" backend to the
  newly configured "s3" backend. No existing state was found in the newly
  configured "s3" backend. Do you want to copy this state to the new "s3"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value:
```

输入"yes"后继续执行远程同步。

----结束

2.7 Modules

一个Terraform配置文件中通常包含多个Resource，Data Source以及变量，为了简化配置和降低维护复杂度，Terraform提供了Modules功能。Module相当于一个Terraform模块，是对多个资源的封装及抽象。

调用模块

在配置文件中声明 module块从可以调用module，其语法如下：

```
module "child_module" {
  source = "./child"
  ...
}
```


一个 module块包含关键字、module名称和块主体三个部分，块主体中需要指定source以及module中定义的输入变量等参数。在添加 module块之后，需要通过 "terraform init" 命令将模块代码的副本保存至工作目录下。在修改或删除 module块后，也必须重新运行 "terraform init" 命令以更新相应的配置。

source 是 module 中的必选参数，用于指向包含模块配置文件的源路径。Terraform支持使用以下源路径：

- **本地路径**

  本地路径必须以 "./" 或 "../" 开头，如 "./child"，"../parent"等。

- **GitHub**

  Terraform将以下路径解析为GitHub仓库：

  ```
  # 使用Https协议克隆
  module "myvpc" {
    source = "github.com/terraform-huaweicloud-modules/terraform-huaweicloud-vpc.git"
    ...
  }

  # 使用SSH协议克隆
  module "myvpc" {
    source = "git@github.com:terraform-huaweicloud-modules/terraform-huaweicloud-vpc.git"
    ...
  }
  ```

  默认情况下，Terraform将克隆仓库的master分支。如选择其他分支或版本，需要在路径中指定ref参数，例如：

  ```
  module "myvpc" {
    source = "github.com/terraform-huaweicloud-modules/terraform-huaweicloud-vpc.git?ref=v1.0.0"
    ...
  }
  ```

- **通用Git仓库**

  Terraform将以 "git::" 开头的路径解析为通用Git仓库，支持Https和SSH协议克隆仓库，支持指定分支或版本。

  ```
  module "myvpc" {
    source = "git::https://example.com/vpc.git"
  }

  module "mystorage" {
    source = "git::ssh://username@example.com/storage.git"
  }

  module "myvpc" {
    source = "git::https://example.com/vpc.git?ref=v1.2.0"
  }
  ```

- **OBS/S3桶**

  Terraform将以 "s3::" 开头的路径解析为OBS/S3存储桶，例如：

  ```
  module "myvpc" {
    source = "s3::https://mybucket.obs.cn-north-1.myhuaweicloud.com/myproject/vpc-example.zip
  }
  ```

  示例中，使用了mybucket桶中的myproject/vpc-example.zip对象作为module的源路径。使用OBS/S3桶之前，需要进行认证，将AK/SK导出为环境变量：

  ```
  $ export AWS_ACCESS_KEY_ID="******"
  $ export AWS_SECRET_ACCESS_KEY="******"
  ```

- **HTTP URL**

  当源路径为HTTP或HTTPs的URL时，Terraform将向给定的URL发送GET请求并下载对应的文件。我们也可以通过HTTP URL的形式来访问OBS桶中的对象，将对象的访问策略设置为公共读，然后使用对应的endpoint链接即可：

```
module "myvpc" {
  source = "https://mybucket.obs.cn-north-1.myhuaweicloud.com/myproject/vpc-example.zip"
}
```

访问模块的输出变量

由于模块是对资源的封装和抽象，我们不能直接访问模块中定义的资源属性，只能访问模块中定义的输出变量，格式为 "module.<MODULE NAME>.<OUTPUT NAME>"。我们采用本地路径源的方式进行说明，首先在工作目录的 "./modules/network" 路径下定义模块，创建一个VPC，并输出VPC ID，内容如下：

```
variable "mycidr" {
  type    = string
  default = "192.168.0.0/16"
}

resource "huaweicloud_vpc" "vpc" {
  name = "vpc_demo"
  cidr = var.mycidr
}

output "vpc_id" {
  value = huaweicloud_vpc.vpc.id
}
```

在工作目录中调用该模块，并在VPC下新增一个子网，此时 vpc_id 的值需要用 "module.network.vpc_id”来表示。

```
module "network" {
  source = "./modules/network"
}

resource "huaweicloud_vpc_subnet" "subnet" {
  name       = "subnet_new"
  cidr       = "192.168.12.0/24"
  gateway_ip = "192.168.12.1"
  vpc_id     = module.network.vpc_id
  dns_list   = ["100.125.1.250","100.125.129.250"]
}
```

通过Module对输出值的使用隐藏了资源实现的细节，父模块无需关心子模块的具体实现，也不会改变子模块的模块和结构。此方式不仅避免了外部错误对资源的影响，同时也降低了模块之间的耦合度。

2.8 Terraform 命令行

2.8.1 命令行说明

Terraform 支持丰富的命令行操作，本文档仅介绍一些常用的命令， 您可以通过 "terraform" 命令查看完整的命令列表。对于特定的子命令，可以通过 -h / --help 选项获取完整的用法。

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

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.
...
```

```
$
terraform init -h
Usage: terraform init [options] [DIR]

  Initialize a new or existing Terraform working directory by creating
  initial files, loading any remote state, downloading modules, etc.

  This is the first command that should be run for any new or existing
  Terraform configuration per machine. This sets up all the local data
  necessary to run Terraform that is typically not committed to version
  control.
...
```

2.8.2 基本命令

terraform init

terraform init 命令是我们执行的第一条命令，主要用于初始化工作目录，完成Provider、Backend、Modules等模块的加载。

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of huaweicloud/huaweicloud...
- Installing huaweicloud/huaweicloud v1.20.0...
- Installed huaweicloud/huaweicloud v1.20.0 (self-signed, key ID 55E52798EF815D18)

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgardes to new major versions that may contain breaking
changes, we recommend adding version constraints in a required_providers block
in your configuration, with the constraint strings suggested below.

* huaweicloud/huaweicloud: version = "~> 1.20.0"

Terraform has been successfully initialized!
```

在同一个工作目录下，terraform init 命令可以重复执行。我们可以使用 "-upgrade=true" 选项对使用的Provider和Modules进行更新。

```
$ terraform init -upgrade=true
```

terraform plan

terraform plan 命令用于创建执行前的计划，是 terraform apply 执行前的一个预览方式，可以检查当前的变更是否符合预期。terraform plan 命令将检测云上资源的属性和状态文件是否存在差异，如果不一致，Terraform 会将差异结果显示在命令下方：

```
$ 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 state storage.
...
Plan: 1 to add, 0 to change, 1 to destroy.
...
```

如果 Terraform 未检测到资源或根模块的更改，则 terraform plan 会输出如下提示：

```
$ terraform plan
...
No changes. Infrastructure is up-to-date.
```



```
This means that Terraform did not detect ant differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

默认情况下，terraform plan 命令首先会从远端更新资源的属性。在管理资源较多的情况下，该操作会耗时较长，我们可以使用 "-refresh=false" 选项来禁止更新。

terraform apply

terraform apply 命令用于执行资源的创建或变更。在操作执行前会进行一次人机交互，用于对资源创建和变更的确认。我们也可以使用 "-auto-approve" 选项跳过人机交互直接执行。

```
$ terraform apply
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
Terraform will perform the following action:
...
Plan: 1 to add, 0 to change, 1 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:
```

terraform apply 的执行结果会保存在状态文件 (terraform.tfstate) 中，并且会显示定义的输出变量值。

```
Apply complete! Resources: 1 to add, 0 to change, 1 to destroy.

Outputs:

vpc_id = df507d37-bce2-4750-8873-f62abb3b085c
```

terraform destroy

terraform destroy 命令用于销毁目标资源，并且会在操作执行前进行一次人机交互，用于对资源销毁确认。我们也可以使用 "-auto-approve" 选项跳过人机交互直接销毁资源。

```
$ terraform destroy -auto-approve
...
Destroy complete! Resources: 3 destroyed.
```

terraform destroy 命令默认会释放当前模块中的所有资源，我们可以通过 -target 选项销毁特定的资源，格式为：-target=<资源类型>.<资源名称>：

```
$ terraform destroy -target=huaweicloud_vpc_subnet.subnet_1
An execution plan has been generated and is show below.
Resource actions are indicated with the following symbols:
  - destroy
Terraform will perform the following actions:
  # huaweicloud_vpc_subnet.subnet_1 will be destroyed
  - resource "huaweicloud_vpc_subnet" "subnet_1" {
    ...
  }
Plan: 0 to add, 0 to change, 1 to destroy.
...
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.
  Enter a value:
```


terraform refresh

terraform refresh 命令用于刷新当前 state 文件的配置，该命令会调用远端 API 获取最新数据并将结果写入 state 文件中。

terraform show

terraform show 命令用于展示当前 state 文件中所有被管理的资源及其属性值。

terraform output

terraform output 命令用于显示当前配置中的输出变量。根据配置中定义的输出变量，执行 terrafrom output 命令后会按照变量定义逐一进行输出，规则为 <输出变量名> = <输出变量值>。输出变量的用法请参考**2.4.2 输出变量**。

```
$ terraform output
vpc_id = df507d37-bce2-4750-8873-f62abb3b085c
```

2.8.3 state 管理命令

Terraform 在完成资源的创建和修改后，会将资源的状态和属性信息会保存在 terraform.tfstate 文件中。我们可以使用 "terraform state" 相关命令对 state 进行管理。

terraform state list

该命令列出当前state文件中配置的所有资源，输出格式为：<资源类型>.<资源名称>

```
$ terraform state list
data.huaweicloud_identity_role.role_example
huaweicloud_identity_group_membership.membership_example
huaweicloud_identity_group.group_example
huaweicloud_identity_user.user_example
random_password.password
```

terraform state show

该命令可以查看某个资源的所有属性值，命令格式为：terraform state show <资源类型>.<资源名称>

```
$ terraform state show huaweicloud_identity_user.user_example
# huaweicloud_identity_user.user_example:
resource "huaweicloud_identity_user" "user_example" {
  domain_id = "0970d7b7d400f*************a03560"
  enabled   = true
  id        = "0aa800908f00f35b1f92c0038250d042"
  name      = "user_example"
  password  = (sensitive value)
  region    = "cn-north-1"
}
```

terraform state rm

该命令用于在state文件中移除指定的资源，而非真正从华为云中将资源删除，命令格式为：terraform state rm <资源类型>.<资源名称>

```
$ terraform state rm huaweicloud_identity_group_membership.membership_example
Removed huaweicloud_identity_group_membership.membership_example
Successfully removed 1 resource instance(s).
```


terraform state mv

该命令用于变更指定资源的存放地址，主要用法有：

- 变更资源名称

  命令格式为：terraform state mv <资源类型>.<资源名称1> <资源类型>.<资源名称2>

  ```
  $ terraform state mv huaweicloud_identity_group_membership.membership_example huaweicloud_identity_group_membership.membership_1
  Move "huaweicloud_identity_group_membership.membership_example" to "huaweicloud_identity_group_membership.membership_1"
  Successfully moved 1 object(s).
  ```

- 变更资源路径

  用于将指定资源从当前 state 文件中移动至另一个指定 state 文件中，同时可以变更资源名称，命令格式为:

  terraform state mv -state-out=state文件相对/绝对路径 <资源类型>.<资源名称1> <资源类型>.<资源名称2>

  ```
  $ terraform state mv -state-out=../vpc_basic/terraform.tfstate huaweicloud_identity_user.user_example huaweicloud_identity_user.user_1
  Move "huaweicloud_identity_user.user_example " to "huaweicloud_identity_user.user_1"
  Successfully moved 1 object(s).
  ```

2.8.4 其他命令

terraform validate

该命令用于快速检查配置文件中的语法错误，无需通过 plan/apply 命令便可定位错误的详细位置和原因。

- 检验正确

  ```
  $ terraform validate
  The configuration is valid.
  ```

- 检验错误

  ```
  $ terraform validate
  Error: Missing newline after argument
    on main.tf line 34, in data "huaweicloud_identity_role" "auth_admin":
    34:     name = "system_all"]
  An argument definition must end with a newline.
  ```

terraform fmt

该命令用于将当前目录及其子目录下的所有的 .tf 文件进行格式化，使其代码风格统一。

terraform graph

该命令用于根据配置文件或执行计划输出资源的可视化依赖关系，命令的输出为 DOT 格式数据。

```
$ terraform graph
digraph {
  compound = "true"
  newrank = "true"
  subgraph "root" {
    "[root] huaweicloud_vpc.vpc_1 (expand)" [label = "huaweicloud_vpc.vpc_1", shape = "box"]
    "[root] huaweicloud_vpc_subnet.subnet_1 (expand)" [label = "huaweicloud_vpc_subnet.subnet_1", shape = "box"]
    "[root] provider[\"registry.terraform.io/huaweicloud/huaweicloud\"]" [label =
```



```
"provider[\"registry.terraform.io/huaweicloud/huaweicloud\"]", shape = "diamond"]
    "[root] huaweicloud_vpc.vpc_1 (expand)" -> "[root] provider[\"registry.terraform.io/huaweicloud/
huaweicloud\"]"
    "[root] huaweicloud_vpc_subnet.subnet_1 (expand)" -> "[root] huaweicloud_vpc.vpc_1 (expand)"
    "[root] meta.count-boundary (EachMode fixup)" -> "[root] huaweicloud_vpc_subnet.subnet_1 (expand)"
    "[root] provider[\"registry.terraform.io/huaweicloud/huaweicloud\"] (close)" -> "[root]
huaweicloud_vpc_subnet.subnet_1 (expand)"
    "[root] root" -> "[root] meta.count-boundary (EachMode fixup)"
    "[root] root" -> "[root] provider[\"registry.terraform.io/huaweicloud/huaweicloud\"] (close)"
  }
}
```

我们通过 dot 命令将结果转换为可视化的图表：

```
$ apt instal graphviz
$ terraform graph | dot -Tsvg > graph.svg
```

terraform import

该命令用于将存量的资源导入到 state文件中，命令格式为：terraform import <资源类型>.<资源名称> <资源ID>

```
$ terraform import huaweicloud_nat_dnat_rule.my_dnat_rule 130fcccf-a587-414f-b51f-e3b2dfa06b92
huaweicloud_nat_dnat_rule.my_dnat_rule: Importing from ID "130fcccf-a587-414f-b51f-e3b2dfa06b92"...
  Prepared huaweicloud_nat_dnat_rule for import
huaweicloud_nat_dnat_rule.my_dnat_rule: Refreshing state... [id=130fcccf-a587-414f-b51f-e3b2dfa06b92]
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.
```

terraform taint

该命令用于手动地将资源标记为污染状态 (tainted)。一旦资源被标记为“tainted”，当再次执行 terraform apply 命令时，Terraform 会将该资源被销毁并重新创建。命令格式为：terraform taint <资源类型>.<资源名称>

```
$ terraform taint huaweicloud_vpc_subnet.subnet_1
Resource instance huaweicloud_vpc_subnet.subnet_1 has been marked as tainted.
```

标记后使用 terraform show 命令可以看到相应资源状态已改变：

```
$ terraform show
...
# huaweicloud_vpc_subnet.subnet_1: (tainted)
resource "huaweicloud_vpc_subnet" "subnet_1" {
  ...
}
```

terraform untaint

该命令用于手动地取消资源的污染状态，使其恢复到正常状态，是 taint 的逆操作。在某些异常情况下，Terraform 会将资源标记为污染状态，我们可以手工处理后，使用 terraform untaint 命令取消被污染标记，这样可以避免资源的重新创建。命令格式为：terraform untaint <资源类型>.<资源名称>

```
$ terraform untaint huaweicloud_vpc_subnet.subnet_1
Resource instance huaweicloud_vpc_subnet.subnet_1 has been successfully untainted.
```

terraform console

该命令可以打开一个交互式的控制台，我们可以使用该控制台进行表达式及内置函数的体验和测试。


```
$ terraform console
> length("Hello, cloud!")
13
```

