Tencent Cloud Infrastructure as Code

Terraform Guide

Product Documentation

Contents

Terraform Guide Overview

Last updated : 2022-01-21 09:58:08

Terraform Overview

Terraform is an open-source resource orchestration tool written in Go and running on the client. It is highly scalable based on the HashiCorp Plugin architecture. Currently, Tencent Cloud implements the TencentCloud Provider based on Terraform plugin to manage Tencent Cloud resources through Terraform. The schematic diagram is as follows:
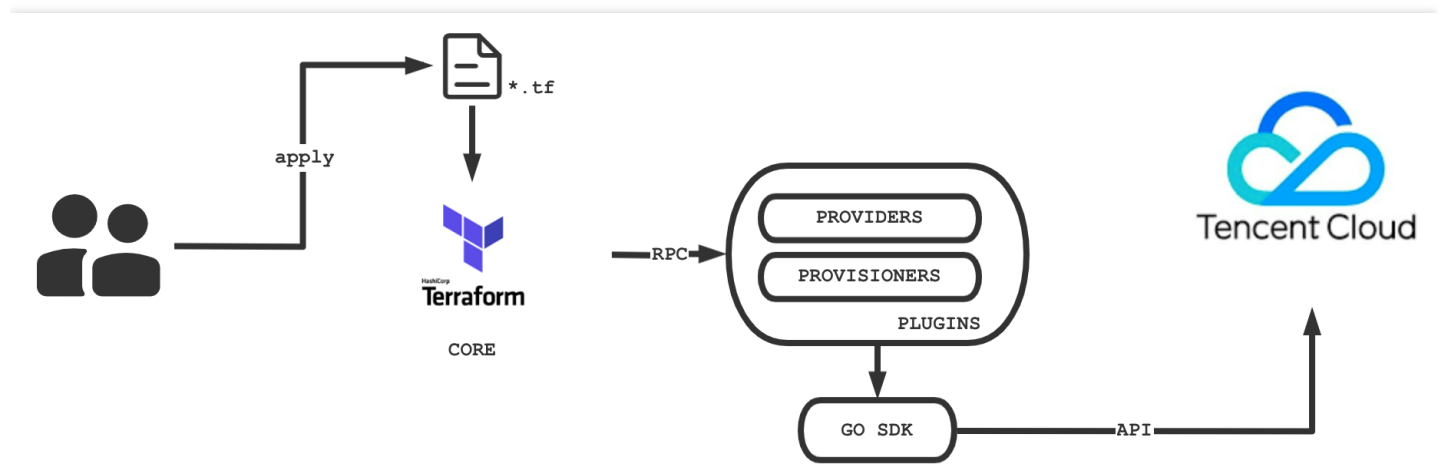
Based on tencentcloud-sdk-go, TencentCloud Provider offers more than 183 resources and 158 data sources across over 30 products, covering compute, storage, network, container service, load balancing, middleware, database, and cloud monitoring to meet your basic needs for cloudification.

For a quick start on Terraform, see TencentCloud Provider and Examples. In addition, certain resources have been and more resources will be supported on Terraform Module.

Terraform Strengths

Multi-Cloud orchestration

Terraform is suitable for multi-cloud solutions where you can deploy similar infrastructures in Tencent Cloud, other cloud providers, or local IDCs. You can manage resources from different cloud providers at the same time using the same tools and similar configuration files.

Infrastructure and code

You can use the high-level configuration syntax HCL to describe an infrastructure, so that it can be codified and versioned for sharing and reuse as shown in the following example:

```
resource "tencentcloud_mysql_instance" "mysql" {
mem_size = 16000
cpu = 4
volume_size = 50
charge_type = "PREPAID"
instance_name = "testAccMysql"
engine_version = "5.5"
root_password = "test1234"
availability_zone = var.availability_zone
internet_service = 1
intranet_port = 3360
prepaid_period = 1
tags = {
purpose = "for test"
}
parameters = {
max_connections = "1000"
}
count = 1
}
```

Execution plan

Terraform has a "planning" step to generate an execution plan, which shows the state of Terraform when `apply` is called. This allows you to avoid incidents when the infrastructure is manipulated on Terraform, as shown in the following example:

```
Terraform used the selected providers to generate the following execution plan. Resource actions
are indicated with the following symbols:
+ create
Terraform will perform the following actions:
# tencentcloud_ckafka_instance.foo will be created
+ resource "tencentcloud_ckafka_instance" "foo" {
+ band_width = (known after apply)
+ disk_size = 500
+ disk_type = "CLOUD_BASIC"
+ id = (known after apply)
+ instance_name = "tf-test"
+ kafka_version = "1.1.1"
+ msg_retention_time = 1300
+ partition = (known after apply)
```

```
+ period = 1
+ public_network = (known after apply)
+ renew_flag = 0
+ subnet_id = "subnet-dvzsb5ro"
+ vpc_id = "vpc-fvl16x63"
+ zone_id = 100006

+ config {
+ auto_create_topic_enable = true
+ default_num_partitions = 3
+ default_replication_factor = 3
}

+ dynamic_retention_config {
+ bottom_retention = (known after apply)
+ disk_quota_percentage = (known after apply)
+ enable = 1
+ step_forward_percentage = (known after apply)
}
}
Plan: 1 to add, 0 to change, 0 to destroy.
```

Resource topology

Terraform builds a resource map to create and modify non-dependent resources in parallel. This enhances the efficiency of infrastructure construction on Terraform and helps you gain better insights into infrastructure dependencies with the following command:

```
terraform graph | dot -Tsvg > graph.svg
```

A diagram of the generated resources is as shown below:
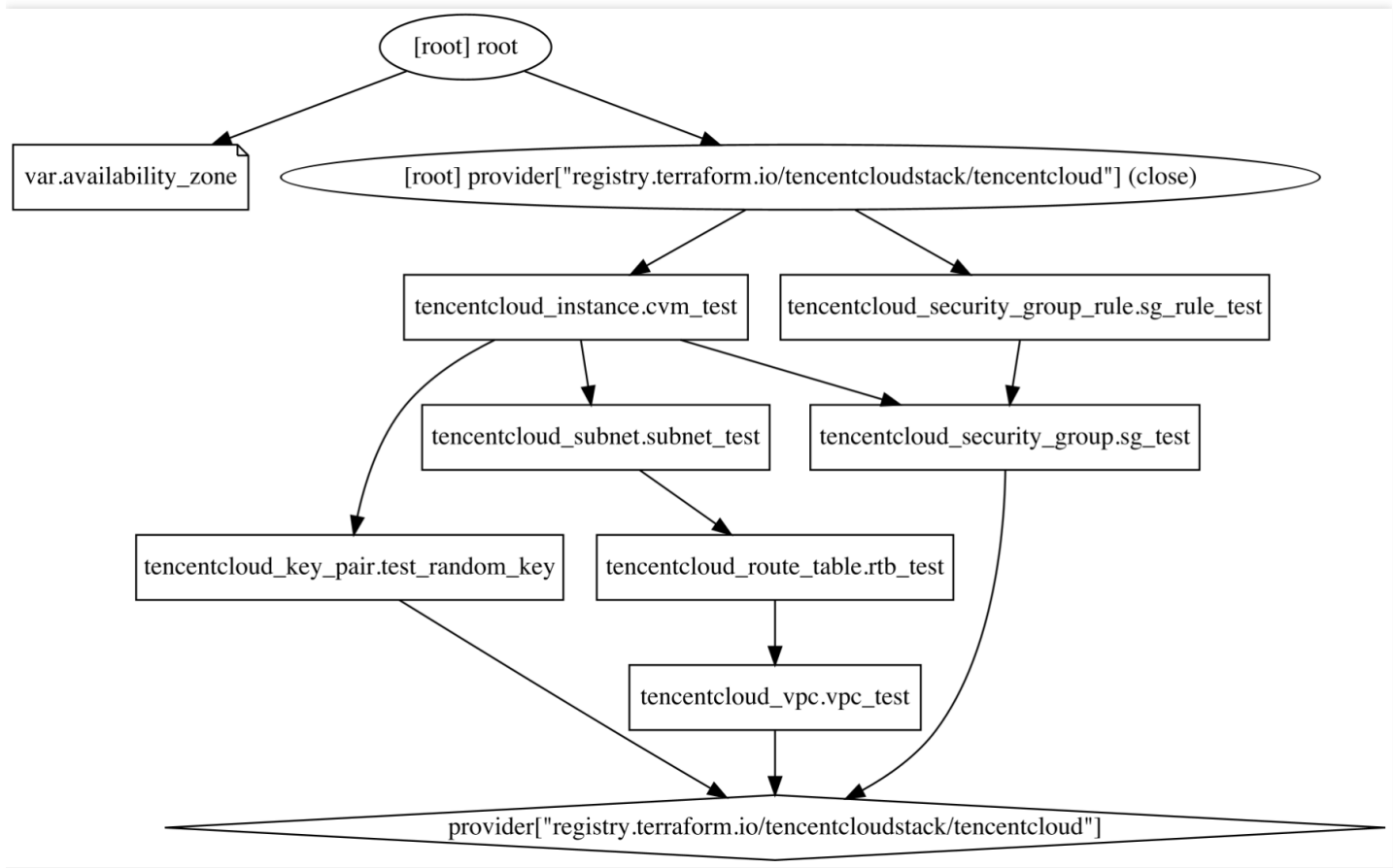
Auto change

You can apply complex change sets to your infrastructure with minimal manual intervention. With the execution plan and resource topology mentioned above, you can get an accurate picture of Terraform dynamics and avoid possible human errors.

Remote state management

Terraform introduces the concept of backend, a remote state storage mechanism. Currently, Tencent Cloud can manage your tfstate files through COS to avoid storing files locally and causing file losses. In addition, remote storage makes it possible for multiple users to manage Terraform resources concurrently.

Quick Start

Last updated : 2022-02-24 17:40:24

This document describes how to quickly create a Tencent Cloud VPC with Terraform.

Step 1. Install Terraform

1. Go to Terraform official website and use the command line to install Terraform directly or download the binary installation file.
2. Unzip the file and configure the global path.
   Skip this step if you use the command line.
   - Linux and macOS
   - Windows
   i. Run the following command to unzip the file. Replace 1.x.x with the actual version number of Terraform to be installed.

   ```
   unzip terraform_1.x.x_linux_amd64.zip
   ```

   ii. Run the following command to add the current directory to the `~/.profile` file.

   ```
   echo $"export PATH=¥$PATH:$(pwd)" >> ~/.bash_profile
   ```

   iii. Run the following command to make the global path configuration take effect.

   ```
   source ~/.bash_profile
   ```

3. Run the following command to check whether the installation is successful.

   ```
   terraform -version
   ```

If the following information is returned (the version number may be different), the installation is successful:

```
> Terraform v1.0.10
> on darwin_amd64
> Your version of Terraform is out of date! The latest version
> is 1.1.0. You can update by downloading from https://www.terraform.io/downloads.html
```

Step 2. Get credentials

Create and copy `SecretId` and `SecretKey` on the API Key Management page.

Step 3. Authenticate

You can authenticate in two ways:

- Authentication by Static Credential
- Authentication by Environment Variable

Create a `provider.tf` file in the user directory and enter the following content:
Replace `my-secret-id` and `my-secret-key` with `SecretId` and `SecretKey` obtained in the Get credentials step.

```
provider "tencentcloud" {
secret_id = "my-secret-id"
secret_key = "my-secret-key"
}
```

Step 4. Create a Tencent Cloud VPC with Terraform

1. Create a `provider.tf` file with the following content to specify the provider configuration information:

```
terraform {
required_providers {
tencentcloud = {
source = "tencentcloudstack/tencentcloud"
# Specify the version by `version`
# version = ">=1.60.18"
}
}
}
provider "tencentcloud" {
region = "ap-guangzhou"
# secret_id = "my-secret-id"
# secret_key = "my-secret-key"
}
```

2. Create a `main.tf` file with the following content to configure TencentCloud Provider and create a VPC. The file contains the following content:

```
resource "tencentcloud_vpc" "foo" {
name = "ci-temp-test-updated"
cidr_block = "10.0.0.0/16"
dns_servers = ["119.29.29.29", "8.8.8.8"]
is_multicast = false
tags = {
"test" = "test"
}
}
```

3. Run the following command to initialize the working directory and download the plugin.

```
terraform init
```

The following information is returned:

```
Initializing the backend...
Initializing provider plugins...
- Finding latest version of tencentcloudstack/tencentcloud...
- Installing tencentcloudstack/tencentcloud v1.60.18...
- Installed tencentcloudstack/tencentcloud v1.60.18 (signed by a HashiCorp partner, key ID 84F69E
1C1BECF459)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.
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.
```

4. Run the following command to upgrade the provider version.

```
terraform init -upgrade
```

The following information is returned:

```
Initializing the backend...
Initializing provider plugins...
- Finding tencentcloudstack/tencentcloud versions matching ">= 1.60.18"...
- Installing tencentcloudstack/tencentcloud v1.60.19...
- Installed tencentcloudstack/tencentcloud v1.60.19 (signed by a HashiCorp partner, key ID 84F69E
1C1BECF459)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.
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.
```

5. Run the following command to view the execution plan and display the details of the resource to be created.

```
terraform plan
```

The following information is returned:

```
Terraform used the selected providers to generate the following execution plan. Resource actions
are
indicated with the following symbols:
+ create
Terraform will perform the following actions:
# tencentcloud_vpc.foo will be created
+ resource "tencentcloud_vpc" "foo" {
+ cidr_block = "10.0.0.0/16"
+ create_time = (known after apply)
+ default_route_table_id = (known after apply)
+ dns_servers = [
+ "119.29.29.29",
+ "8.8.8.8",
]
```

```
+ id = (known after apply)
+ is_default = (known after apply)
+ is_multicast = false
+ name = "ci-temp-test-updated"
+ tags = {
+ "test" = "test"
}
}
Plan: 1 to add, 0 to change, 0 to destroy.
_____

_____

_____

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exac
tly these
actions if you run "terraform apply" now.
```

6. Run the following command to create the resource.

```
terraform apply
```

Enter `yes` as prompted to create the resource. The following information is returned:

```
Terraform used the selected providers to generate the following execution plan. Resource actions
are
indicated with the following symbols:
+ create
Terraform will perform the following actions:
# tencentcloud_vpc.foo will be created
+ resource "tencentcloud_vpc" "foo" {
+ cidr_block = "10.0.0.0/16"
+ create_time = (known after apply)
+ default_route_table_id = (known after apply)
+ dns_servers = [
+ "119.29.29.29",
+ "8.8.8.8",
]
+ id = (known after apply)
+ is_default = (known after apply)
+ is_multicast = false
+ name = "ci-temp-test-updated"
+ tags = {
+ "test" = "test"
}
}
Plan: 1 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
tencentcloud_vpc.foo: Creating...
tencentcloud_vpc.foo: Still creating... [10s elapsed]
tencentcloud_vpc.foo: Creation complete after 13s [id=vpc-07mx4yfd]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

After execution, you can view the created resource in the Tencent Cloud console.

7. (Optional) Update the resource.

If you change the resource configuration to the following information:

```
resource "tencentcloud_vpc" "foo" {
name = "ci-temp-test-updated2"
cidr_block = "10.0.0.0/16"
dns_servers = ["119.29.29.29", "8.8.8.8"]
is_multicast = false
tags = {
"test" = "test"
}
}
```

1. Run the `terraform plan` command to update the plan. The following information is returned:

```
tencentcloud_vpc.foo: Refreshing state... [id=vpc-jhmdf9q9]
Terraform used the selected providers to generate the following execution plan. Resource actio
ns are indicated with the following symbols:
~ update in-place
Terraform will perform the following actions:
# tencentcloud_vpc.foo will be updated in-place
~ resource "tencentcloud_vpc" "foo" {
id = "vpc-jhmdf9q9"
~ name = "ci-temp-test-updated" -> "ci-temp-test-updated2"
tags = {
"test" = "test"
}
# (6 unchanged attributes hidden)
}
Plan: 0 to add, 1 to change, 0 to destroy.
_____
_____
_____
____

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take e
```

```
xactly these actions if you run "terraform apply"
now.
```

2. Run the `terraform apply` command to create the resource with the updated data. The following information is returned:

```
tencentcloud_vpc.foo: Refreshing state... [id=vpc-jhmdf9q9]
Terraform used the selected providers to generate the following execution plan. Resource actio
ns are indicated with the following symbols:
~ update in-place
Terraform will perform the following actions:
# tencentcloud_vpc.foo will be updated in-place
~ resource "tencentcloud_vpc" "foo" {
id = "vpc-jhmdf9q9"
~ name = "ci-temp-test-updated" -> "ci-temp-test-updated2"
tags = {
"test" = "test"
}
# (6 unchanged attributes hidden)
}
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
tencentcloud_vpc.foo: Modifying... [id=vpc-jhmdf9q9]
tencentcloud_vpc.foo: Modifications complete after 1s [id=vpc-jhmdf9q9]
Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

3. You can run the following command to terminate the resource as needed.

```
terraform destroy
```

The following information is returned:

```
tencentcloud_vpc.foo: Refreshing state... [id=vpc-07mx4yfd]
Terraform used the selected providers to generate the following execution plan. Resource actions
are
indicated with the following symbols:
- destroy
Terraform will perform the following actions:
# tencentcloud_vpc.foo will be destroyed
- resource "tencentcloud_vpc" "foo" {
```

```
- cidr_block = "10.0.0.0/16" -> null
- create_time = "2021-12-15 16:20:32" -> null
- default_route_table_id = "rtb-4m1nmo0e" -> null
- dns_servers = [
- "119.29.29.29",
- "8.8.8.8",
] -> null
- id = "vpc-07mx4yfd" -> null
- is_default = false -> null
- is_multicast = false -> null
- name = "ci-temp-test-updated" -> null
- tags = {
- "test" = "test"
} -> null
}
Plan: 0 to add, 0 to change, 1 to destroy.
Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.
Enter a value: yes
tencentcloud_vpc.foo: Destroying... [id=vpc-07mx4yfd]
tencentcloud_vpc.foo: Destruction complete after 7s
Destroy complete! Resources: 1 destroyed.
```

User Guide
Syntax Guide
Terraform Style

Last updated : 2022-01-14 14:30:41

The Terraform language has customary style conventions, and we recommend you always follow them to ensure file and module consistency between different teams. In addition, automatic formatting tools also need to conform to such conventions, and you can use `terraform fmt` for formatting.

Code Constraint

- Indent two spaces for each nesting level.
- When multiple arguments with single-line values appear on consecutive lines at the same nesting level, align their equals signs:

```
ami           = "abc123"
instance_type = "t2.micro"
```

- When both arguments and blocks appear together inside a block body, place all of the arguments together at the top and then place nested blocks below them. Use one blank line to separate the arguments from the blocks.
- Use blank lines to separate logical groups of arguments within a block.
- For blocks that contain both arguments and "meta-arguments" (as defined by the Terraform language semantics), list meta-arguments first and separate them from other arguments with one blank line. Place meta-argument blocks last and separate them from other blocks with one blank line.

```
resource "tencentclould_instance" "example" {
count = 2 # meta-argument first
ami = "abc123"
instance_type = "t2.micro"
network_interface {
# ...
}
lifecycle { # meta-argument block last
create_before_destroy = true
```

```
    }
}
```

- Top-Level blocks should always be separated from one another by one blank line. Nested blocks should also be separated by blank lines, except when grouping together related blocks of the same type (like multiple `provisioner` blocks in a resource).
- Avoid separating multiple blocks of the same type with other blocks of a different type, unless the block types are defined by semantics to form a family.

Basic Syntax

Last updated : 2022-01-14 14:30:41

Basic Type

A basic type is simple and not composed of any other types. All basic types in Terraform are represented by the `type` keyword. Available basic types include:

- `string` : represents a Unicode character sequence of some text (such as "hello").
- `number` : represents a number, which can be an integer or a decimal.
- `bool` : represents a Boolean value, which can be `true` or `false` .

Below is an example:

```
id = 123
vpc_id = "123"
status = true
```

Composite Type

A composite type is a set of values.

Collection type

A collection contains a set of values of the same type:

- `list(...)` : is a sequence of values identified by consecutive integers starting from 0.
- `map(...)` : is a set of values, each of which is identified by a string label.
- `set(...)` : is a set of unique values.

Structure type

- `object(...)` : is a custom type that contains its own named attributes.
- `tuple(...)` : is a sequence of elements identified by consecutive integers starting from 0, where each element has its own type.

Special type

- `null` : an argument set to null is considered not entered. Terraform will automatically ignore the argument and use the default value.
- `any` : it is a very special type constraint in Terraform. It is not a type but simply a placeholder. Whenever a value is given a complex type constrained by `any` , Terraform will try to calculate the most accurate type to replace `any` .

Argument

Argument assignment means assigning a value to a specific argument, whose name can contain letters, digits, underscores, and hyphens and cannot begin with a digit, such as:

```
id = "123"
```

Block

A block is a container containing a set of arguments, such as:

```
resource "tencentcloud_instance" "foo" {
tags = {}
vpc_id = "vpc-5bt2ix8p"
}
```

Comment

Terraform supports the following three types of comments:

- `#` : a single-line comment followed by the comment content.
- `//` : a single-line comment followed by the comment content.
- `/*` and `*/` : multi-line comments, which should be across multiple lines.

Function

Last updated : 2022-01-14 14:30:41

Numeric Functions

Function Name	Feature	Example	Result
abs	Returns an absolute value	abs(-1024)	1024
ceil	Rounds up	ceil(5.1)	6
floor	Rounds down	floor(4.9)	4
log	Calculates a logarithm	log(16, 2)	4
pow	Calculates an exponential power	pow(3,2)	9
max	Returns the maximum value	max(12,54,3)	54
min	Returns the minimum value	min(12, 54, 3)	3

String Functions

Function Name	Feature	Example	Result
chomp	Removes newline characters at the end of a string	chomp("hello\n")	"hello"
format	Formats a string	format("Hello, %s!", "Ander")	"Hello, Ander!"
lower	Converts a string to lowercase letters	lower("HELLO")	"hello"
upper	Converts a string to uppercase letters	upper("hello")	"HELLO"

Function Name	Feature	Example	Result
join	Concatenates a string list by using a specified delimiter	join(", ", ["foo", "bar", "baz"])	"foo, bar, baz"
replace	Replaces specified characters in a string	replace("1 + 2 + 3", "+", "-")	"1 - 2 - 3"

For more information on functions, see Built-in Functions.

Expression

Last updated : 2022-01-14 14:30:41

Operators

Operators are used for arithmetic or logical operations.

Arithmetic operators

- **a + b**: returns the result of adding `a` and `b` together.
- **a - b**: returns the result of subtracting `b` from `a` .
- **a * b**: returns the result of multiplying `a` and `b` .
- **a / b**: returns the result of dividing `a` by `b` .
- **a % b**: returns the remainder of dividing `a` by `b` . This operator is generally useful only when used with whole numbers.
- **-a**: returns the result of multiplying `a` by -1.

Comparison operators

- **a == b**: returns `true` if `a` and `b` both have the same type and the same value, or `false` otherwise.
- **a != b**: is the opposite of `a == b` .
- **a < b**: returns `true` if `a` is less than `b` , or `false` otherwise.
- **a > b**: returns `true` if `a` is greater than `b` , or `false` otherwise.
- **a <= b**: returns `true` if `a` is less than or equal to `b` , or `false` otherwise.
- **a >= b**: returns `true` if `a` is greater than or equal to `b` , or `false` otherwise.

Logical operators

- **a || b**: returns `true` if either `a` or `b` is `true` , or `false` if both are `false` .
- **a && b**: returns `true` if both `a` and `b` are `true` , or `false` if either one is `false` .
- **!a**: returns `true` if `a` is `false` , or `false` if `a` is `true` .

Conditional Expression

A conditional expression uses the value of a Boolean expression to select one of two values. For example:

```
condition ? one_value : two_value
```

for Expression

A `for` expression can be used to traverse a set of collections and map one collection type to another. For example:

```
[for item in items : upper(item)]
```

Expanded Expression

An expanded expression is a concise expression similar to a `for` expression. For example:

```
[for o in var.list : o.id]
is equivalent to
var.list[*].id
```

Function Expression

Terraform supports the use of some built-in functions when expressions are calculated. An expression to call a function is similar to an operator. For example:

```
upper("123")
```
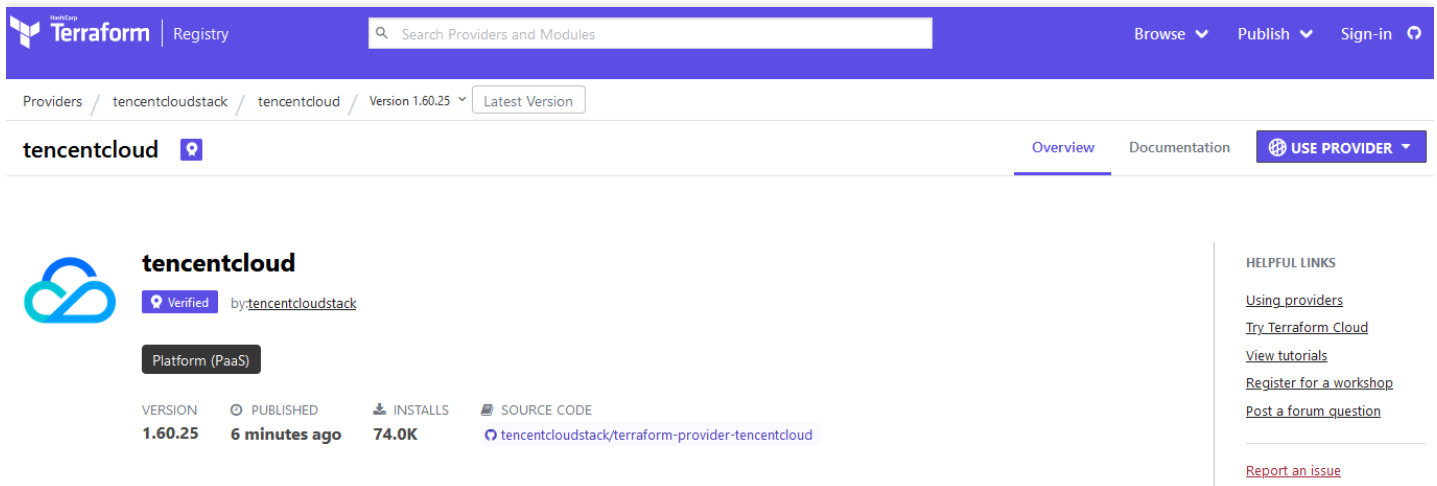
Configuration Guide
Provider

Last updated : 2022-01-14 14:30:41

Terraform relies on provider plugins to interact with cloud providers, SaaS providers, and other APIs. Terraform configurations must declare which providers they require so that Terraform can install and use them. Additionally, some providers require configuration before they can be used. This document describes how to configure provider plugins.

Searching for Provider

Go to the Providers page, search, and enter the TencentCloud Provider page to view the user guide as shown below:

Downloading Provider

Run the following command to download the latest plugin version from Terraform's official repository by default.

```
terraform init
```

If you need to use a legacy version, you can specify the version information with the `version` argument as shown below:

```
terraform {
required_providers {
tencentcloud = {
source = "tencentcloudstack/tencentcloud"
# Specify the version by `version`
version = "1.60.18"
}
}
}
```

Provider Declaration

```
provider "tencentcloud" {
region = "ap-guangzhou"
secret_id = "my-secret-id"
secret_key = "my-secret-key"
}
```

Variables

Last updated : 2022-01-14 14:30:41

Input Variable

- Input variables let you customize aspects of Terraform modules without altering the module's own source code. This allows you to share modules across different Terraform configurations, making your module composable and reusable.
- Input variables can be dynamically passed in; for example, you can pass in variables when creating or modifying the infrastructure, replace hard-coded access keys with variables when defining a provider in the code, and let users (infrastructure creators) decide the desired server size.
- You can understand a set of Terraform code as a function, so the input variables can be seen as function input parameters.

Defining input variable

Input variables are defined using a `variable` block. For example:

```
variable "image_id" {
type = string
}
variable "availability_zone_names" {
type = list(string)
default = ["us-west-1a"]
}
variable "docker_ports" {
type = list(object({
internal = number
external = number
protocol = string
}))
default = [
{
internal = 8300
external = 8300
protocol = "tcp"
}
]
}
```

The label after the `variable` keyword is a name for the variable, which must be unique among all variables in the same module. You can reference the variable value in the code through `var.<name>`.

A `variable` block can be declared with the following optional arguments:

- `default` : specifies the default value of the input variable.
- `type` : specifies that the input variable can only be assigned with a specific value.
- `description` : specifies the description of the input variable.
- `validation` : specifies the validation rules for the input variable.
- `sensitive` : limits Terraform UI output when the variable is used in configuration.
- `nullable` : specifies whether the input variable can be `null` or not.

Type

A type is defined in an input variable block by `type`.

- Basic types: `string` , `number` , `bool` .
- Complex types: `list(<type>)` , `set(<type>)` , `map(<type>)` , `object({<attr name="">` = `<type>, ... })` , `tuple([<type>, ...])` .

Description

You can briefly describe the purpose of each variable. For example:

```
variable "image_id" {
type = string
description = "The id of the machine image (AMI) to use for the server."
}
```

Custom validation rule

Prior to Terraform 0.13.0, only type constraints could be used to ensure that input arguments were of the correct type. Terraform 0.13.0 introduced custom validation rules for input variables. For example:

```
variable "image_id" {
type = string
description = "The id of the machine image (AMI) to use for the server."
validation {
condition = length(var.image_id) > 4 && substr(var.image_id, 0, 4) == "ami-"
error_message = "The image_id value must be a valid AMI id, starting with ¥"ami-¥"."
}
}
```

The `condition` argument is a bool argument. You can use an expression to determine whether the input variable is valid. When the `condition` is `true`, the input variable is valid; otherwise, it is not. A `condition` expression can reference only the currently defined variable by `var.<variable name="">` and must not produce errors. If the failure of an expression is the basis of the validation decision, use the `can` function to detect such errors. For example:

```
variable "image_id" {
type = string
description = "The id of the machine image (AMI) to use for the server."
validation {
# regex(...) fails if it cannot find a match
condition = can(regex("^ami-", var.image_id))
error_message = "The image_id value must be a valid AMI id, starting with ¥"ami-¥"."
}
}
```

In the above example, if the input `image_id` does not meet the requirements of the regex, the regex function call will throw an error, which will be captured by the `can` function to output `false`. If the condition expression outputs `false`, Terraform will return the error message defined in `error_message`, which should fully describe the reason for the failure of the input variable validation, along with the valid constraints on the input variable.

Using input variable

You can access an input variable with `var.<variable name="">` only within the module where the variable is declared. For example:

```
resource "tencentclould_instance" "example" {
instance_type = "t2.micro"
ami = var.image_id
}
```

Assigning value to input variable

Command line argument

To specify individual variables on the command line, you need to use the `-var` option when running the `terraform plan` and `terraform apply` commands. For example:

```
terraform apply -var="image_id=ami-abc123"
terraform apply -var='image_id_list=["ami-abc123","ami-def456"]' -var="instance_type=t2.micro"
terraform apply -var='image_id_map={"us-east-1":"ami-abc123","us-east-2":"ami-def456"}'
```

Argument file

When you set a large number of variables, we recommend you specify their values in the variable argument file (suffixed with `.tfvars` or `.tfvars.json`) and specify the file on the command line with `-var-file`. For example:

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

Argument definition files use the same basic syntax as Terraform language files but contain only variable name assignments. For example:

```
image_id = "ami-abc123"
availability_zone_names = [
"us-east-1a",
"us-west-1c",
]
```

Terraform automatically loads a number of variable definition files:

- Files named `terraform.tfvars` or `terraform.tfvars.json`.
- Files suffixed with `.auto.tfvars` or `.auto.tfvars.json`.
- `.json` files need to be defined with the JSON syntax. For example:

  ```
  {
  "image_id": "ami-abc123",
  "availability_zone_names": ["us-west-1a", "us-west-1c"]
  }
  ```

Environment variable

You can specify input variables by defining environment variables prefixed with `TF_VAR_`. For example:

```
export TF_VAR_image_id=ami-abc123
export TF_VAR_availability_zone_names='["us-west-1b","us-west-1d"]'
```

Input variable priority

- You may assign the same variable more than once when using multiple assignment methods at the same time. Terraform overwrites the old value with the new one and loads variable values in the following order:
  i. Environment variables
  ii. `terraform.tfvars` files (if any)

iii. `terraform.tfvars.json` files (if any)

iv. All `.auto.tfvars` or `.auto.tfvars.json` files in alphabetical order

v. Input variables passed in through the `-var` or `-var-file` command line argument, in the order defined in such argument

- If you have tried multiple assignment methods in vain, Terraform will try to use the default value. For variables without a default value defined, Terraform will ask you to input a value on an interactive UI. Some Terraform commands will report errors if they are executed with the `-input=false` argument that disables value passing on the interactive UI.

Output Variable

Output variables make information of the infrastructure available for the command line and other Terraform configurations. An output value is similar to a returned value in traditional programming languages.

Use cases

- Child modules can use output variables to pass their resource attributes to modules.
- Root modules can use output variables to print certain values in the CLI output after `terraform apply` is executed.
- When the remote state is used, other configurations can access the root module output through the `terraform_remote_state` data source.

Defining output variable

Output variables are declared using an `output` block. For example:

```
output "instance_ip_addr" {
value = tencentclould_instance.server.private_ip
}
```

Optional argument

- **description**

  Specifies descriptive information of the output value. For example:

  ```
  output "instance_ip_addr" {
  value = tencentclould_instance.server.private_ip
  description = "The private IP address of the main server instance."
  }
  ```

- **sensitive**

  Hides the value of the output variable on the CLI when the `terraform plan` and `terraform apply` commands are being executed.

- **depends_on**

  Terraform parses various data and resources defined by the code and their dependencies. For example, if the `image_id` argument used to create a virtual machine is queried by `data`, then the virtual machine instance depends on this image's `data`. Terraform creates `data` first and then the virtual machine `resource` after the query result is obtained.

  In general, the order for creating `data` and `resource` is automatically calculated by Terraform and does not need to be explicitly specified by the code writer. However, sometimes there are dependencies that cannot be derived through code analysis, in which case the dependencies can be explicitly declared in the code through `depends_on`. For example:

```
output "instance_ip_addr" {
value = tencentclould_instance.server.private_ip
description = "The private IP address of the main server instance."
depends_on = [
# Security group rule must be created before this IP address could
# actually be used, otherwise the services will be unreachable.
tencentclould_security_group_rule.local_access,
]
}
```

Local Variable

If you need to use a complex expression to calculate a value and use it repeatedly, you can assign the complex expression a local value and then reference it repeatedly. If you see the input variable as the function input and output value as the returned value of the function, the local value is equivalent to a local variable defined in the function.

Local variable definition

Local variables are declared using a `locals` block. For example:

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

Additionally, local variables include not only literal constants but also other variables of the module (variables, resource attributes, or other local values) in order to convert or combine them. For

example:

```
locals {
# Ids for multiple sets of EC2 instances, merged together
instance_ids = concat(tencentclould_instance.blue.*.id, tencentclould_instance.green.*.id)
}
locals {
# Common tags to be assigned to all resources
common_tags = {
Service = local.service_name
Owner = local.owner
}
}
```

Using local variable

Local variables are referenced through the `local.<name>` expression. For example:

```
resource "tencentclould_instance" "example" {
# ...
tags = local.common_tags
}
```

Data Sources

Last updated : 2022-01-14 14:30:41

Data sources allow Terraform to use information defined outside of Terraform, defined by another separate Terraform configuration, or modified by functions.

Using Data Source

A data source is accessed via a special kind of resource known as a data resource, declared using a `data` block as shown below:

```
data "tencentcloud_availability_zones" "my_favourite_zone" {
name = "ap-guangzhou-3"
}
```

Referencing Data Source

The syntax for referencing data from a data source is `data.<type>.<name>.<attribute>` as shown below:

```
resource "tencentcloud_subnet" "app" {
...
availability_zone = data.tencentcloud_availability_zones.default.zones.0.name
...
}
```

Resource

Last updated : 2022-01-14 14:30:41

Resources are the most important element in the Terraform language. A resource is defined using a `resource` block, which describes one or more infrastructure objects, such as VPC and VM.

Resource Syntax

A resource is defined using a `resource` block, which contains the `resource` keyword, resource type, resource name, and resource block body as shown below:

```
resource "tencentcloud_vpc" "foo" {
name = "ci-temp-test-updated"
cidr_block = "10.0.0.0/16"
dns_servers = ["119.29.29.29", "8.8.8.8"]
is_multicast = false
tags = {
"test" = "test"
}
}
```

Resource Reference

A resource attribute is referenced in the syntax format of `<resource type="">.<name>.<attribute>` as shown below:

```
tencentcloud_vpc.foo.resource # ci-temp-test-updated
```

Modules

Last updated : 2022-01-14 14:30:41

A module is a folder containing a set of Terraform code, which is an abstraction and encapsulation of multiple resources.

Calling Module

Modules are referenced in Terraform code using a `module` block, which contains the `module` keyword, `module` name, and `module` body (the part within the `{}`) as shown below:

```
module "servers" {
source = "./app-cluster"
servers = 5
}
```

A `module` can be called using the following arguments:

- `source` : specifies the source of the referenced module.
- `version` : specifies the version number of the referenced module.
- `meta-arguments` : is a feature supported since Terraform 0.13. Similar to `resource` and `data` , it can be used to manipulate the behaviors of `module` .

Argument Description

Source

The `source` argument tells Terraform where to find the source code for the desired child module. Terraform uses this during the module installation step of `terraform init` to download the source code to a directory on local disk so that it can be used by other Terraform commands.

The module can be installed from the following source types:

- Local paths
- Terraform Registry
- GitHub
- Bitbucket
- Generic Git, Mercurial repositories
- HTTP URLs

- S3 buckets
- GCS buckets
- Modules in Package Sub-directories

This document describes installation from local path, Terraform Registry, and GitHub.

Local path

Local paths can use child modules from the same project. Unlike other resources, local paths do not require the download of relevant code. For example:

```
module "consul" {
source = "./consul"
}
```

Terraform Registry

Terraform Registry is currently the module repository solution recommended by Terraform. It uses Terraform's custom protocol and supports version management and module use. Terraform Registry hosts and indexes a large number of public modules, allowing quick search for a variety of official and community-supplied quality modules.

Modules in Terraform Registry can be referenced with source addresses in the format of `<namespace>/<name>/<provider>`. You can get the exact source address in the module description. For example:

```
module "consul" {
source = "hashicorp/consul/xxx"
version = "0.1.0"
}
```

For modules hosted in other repositories, you can add `<hostname>/` to the source address header to specify the server name of the private repository. For example:

```
module "consul" {
source = "app.terraform.io/example-corp/k8s-cluster/azurerm"
version = "1.1.0"
}
```

GitHub

If Terraform reads a source argument value prefixed with `github.com`, it will automatically recognize it as a GitHub source. For example, you can clone a repository using the HTTPS protocol:

```
module "consul" {
source = "github.com/hashicorp/example"
}
```

To use the SSH protocol, use the following address:

```
module "consul" {
source = "git@github.com:hashicorp/example.git"
}
```

> Note
> The GitHub source is handled in the same way as generic Git repositories, as both of them get Git credentials and reference specific versions through the `ref` argument in the same way. If you want to access private repositories, you need to configure additional Git credentials.

Version

When a registry is used as a module source, you can use the `version` meta-argument to constrain the version of the module used. For example:

```
module "consul" {
source = "hashicorp/consul/xxx"
version = "0.0.5"
servers = 3
}
```

The format of the `version` meta-argument is in line with the provider version constraint. In this case, Terraform will use the latest version of the module instance that has been installed. If no compliant version is currently installed, the latest compliant version will be downloaded.

The `version` meta-argument can only be used with the registry to support public or private module repositories. Other types of module sources, such as local path, do not necessarily support versioning.

Backend

Last updated : 2022-01-14 14:30:41

Remote State Storage Mechanism

Storing state files locally only may cause the following issues:

- A `tfstate` file is stored locally in the current working directory by default. If computer damage causes file loss, all the resources corresponding to the `tfstate` file will become unmanageable, leading to a resource leak.
- A `tfstate` file cannot be shared among group members.

To facilitate the storage and sharing of state files, Terraform introduces the remote state storage mechanism called "backend", an abstract remote storage API. Similar to a provider, backend supports a variety of remote storage services as described in Available Backends. A Terraform backend has two modes:

- **Standard**: supports remote state storage and state lock.
- **Enhanced**: supports remote operations (such as `plan` and `apply` on a remote server) in addition to the standard features.

Notes

- After the backend configuration is updated, you need to run `terraform init` to verify and configure the backend.
- If no custom backend is configured, Terraform will use the local backend by default. For example, a `tfstate` file is stored in the local directory by default.
- Backend configuration is subject to the following restraints:
  - One configuration file provides only one backend block.
  - Backend blocks cannot reference named values (such as input variables, local variables, or data source attributes).

Using Backend

The definition of a `backend` block is nested in a top-level Terraform block. This document uses the Tencent Cloud Object Storage (COS) service as an example for configuration. For more information

on other storage modes, see [Available Backends](#).

```
terraform {
backend "cos" {
region = "ap-nanjing"
bucket = "tfstate-cos-1309190246"
prefix = "terraform/state"
}
}
```

If you have the `tfstate-cos-1309190246` bucket in COS, the Terraform state information will be written into the `terraform/state/terraform.tfstate` file as shown below:

MetaData

Last updated : 2022-01-14 14:30:41

`Metadata` is a built-in meta-argument supported by Terraform and can be used in provider, resource, data, and module blocks. It mainly includes:

- `depends_on` : explicitly declares dependencies.
- `count` : creates multiple resource instances.
- `for_each` : iterates a collection to create a corresponding resource instance for each element in the collection.
- `provider` : specifies a non-default provider instance.
- `lifecycle` : customizes the lifecycle behavior of a resource.
- `dynamic` : builds repeatable nested blocks.

depends_on

`depends_on` explicitly declares implicit dependencies between resources that cannot be automatically deducted by Terraform. This is useful only if there is a dependency but no data reference between resources. For example:

```
variable "availability_zone" {
default = "ap-guangzhou-6"
}
resource "tencentcloud_vpc" "vpc" {
name = "guagua_vpc_instance_test"
cidr_block = "10.0.0.0/16"
}
resource "tencentcloud_subnet" "subnet" {
depends_on = [tencentcloud_vpc.vpc]
availability_zone = var.availability_zone
name = "guagua_vpc_subnet_test"
vpc_id = tencentcloud_vpc.vpc.id
cidr_block = "10.0.20.0/28"
is_multicast = false
}
```

count

The `count` argument can be any natural number. Terraform creates `count` resource instances, each corresponding to a separate infrastructure object that is created, updated, or terminated separately during Terraform code execution. For example:

```
resource "tencentcloud_instance" "foo" {
availability_zone = var.availability_zone
instance_name = "terraform-testing"
image_id = "img-ix05e4px"
...
count = 3
tags = {
Name = "Server ${count.index}"
}
...
```

- **count.index**: represents the count subscript index (starting from 0) corresponding to the current object
- **Access to object with multiple resource instances**: `.`

for_each

`for_each` is a new feature introduced in Terraform 0.12.6. A `resource` block does not allow both `count` and `for_each` to be declared. The `for_each` argument can be a map or a set(string). Terraform creates a separate infrastructure resource object for each element in the collection; just like with `count`, each infrastructure resource object is created, modified, or terminated separately during Terraform code execution. For example:

- **map**

```
resource "tencentcloud_cfs_access_group" "foo" {
for_each = {
test1_access_group = "test1"
test2_access_group = "test2"
}
name = each.key
description = each.value
}
```

- **set(string)**

```
resource "tencentcloud_eip" "foo" {
for_each = toset(["awesome_gateway_ip1", "awesome_gateway_ip2"])
name = "awesome_gateway_ip"
}
```

provider

If multiple instances of the same type of provider are declared, you can specify a `provider` argument to select a provider instance to be used when creating a resource. If no `provider` argument is specified, Terraform will use the one corresponding to the first word in the resource type name by default. For example:

```
provider "tencentcloud" {
region = "ap-guangzhou"
# secret_id = "my-secret-id"
# secret_key = "my-secret-key"
}
provider "tencentcloud" {
alias = "tencentcloud-beijing"
region = "ap-beijing"
# secret_id = "my-secret-id"
# secret_key = "my-secret-key"
}
resource "tencentcloud_vpc" "foo" {
name = "ci-temp-test-updated"
cidr_block = "10.0.0.0/16"
dns_servers = ["119.29.29.29", "8.8.8.8"]
is_multicast = false
tags = {
"test" = "test"
}
provider = tencentcloud.tencentcloud-beijing
}
```

lifecycle

Each resource instance goes through creation, update, and termination, while the `lifecycle` block can specify a different behavior. Terraform supports the following types of `lifecycle` blocks:

Show All

create_before_destroy

展开&收起

By default, when Terraform needs to modify a resource that cannot be directly upgraded due to server-side API limitations, it will delete the existing resource object and replace it with one created using new configuration arguments. The `create_before_destroy` argument can modify this behavior so that Terraform creates a new object and terminates the old one only after it has been successfully replaced. For example:

```
lifecycle {
create_before_destroy = true
```

```
  }
```

Many infrastructure resources need to have a unique name or ID attribute, and this constraint applies to both the old and new objects when they coexist. Some resource types have special arguments that can add a random prefix to each object name to prevent conflicts, which is not adopted by Terraform by default. You need to resolve this type of constraint for each resource type before using `create_before_destroy`.

prevent_destroy

展开&收起

The `prevent_destroy` argument is a safety measure. As long as it is set to `true`, Terraform will refuse to run any change plan that might terminate the infrastructure resource. It prevents accidental deletion of a critical resource, such as erroneous execution of `terraform destroy` or accidental modification of an argument of a resource that makes Terraform decide to delete a resource instance and create a new one.

Declaring `prevent_destroy = true` inside a `resource` block will prevent `terraform destroy` from being executed. For example:

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

The `prevent_destroy` argument should be used with caution. Note that this measure does not prevent Terraform from deleting relevant resources after a `resource` block is deleted, as the corresponding `prevent_destroy = true` statement has also been deleted.

ignore_changes

展开&收起

By default, when Terraform detects any discrepancy between the configuration described by the code and a real infrastructure object, it will calculate a change plan to update the infrastructure object to match the state described by the code. In some very rare cases, the actual infrastructure object is modified by a process outside of Terraform, and Terraform will continually try to modify the object to bridge the discrepancy with the code. In such cases, you can instruct Terraform to ignore changes to certain attributes by setting `ignore_changes`. The value of `ignore_changes` defines a set of attribute names that need to be created according to the values defined by the code but do not need to be updated according to value changes. For example:

```
resource "tencentcloud_instance" "foo" {
...
lifecycle {
ignore_changes = [
# Ignore changes to tags, e.g. because a management agent
# updates these based on some ruleset managed elsewhere.
tags,
]
}
```

dynamic

In a top-level block such as `resource`, you usually can only perform one-to-one assignments in a form like `name = expression`. This assignment form is generally available, except when some resource types contain repeatable nested blocks. For example:

```
resource "tencentcloud_tcr_instance" "foo" {
name = "example"
instance_type = "basic"
open_public_operation = true
security_policy {
cidr_block = "10.0.0.1/24"
}
security_policy {
cidr_block = "192.168.1.1/24"
}
}
```

In this case, you can use the `dynamic` block to dynamically build repeatable nested blocks similar to `security_policy`. For example:

```
resource "tencentcloud_tcr_instance" "foo" {
name = "example"
instance_type = "basic"
open_public_operation = true
dynamic "security_policy" {
for_each = toset(["10.0.0.1/24", "192.168.1.1/24"])
content {
cidr_block = security_policy.value
}
}
}
```

`dynamic` can be used in `resource`, `data`, `provider`, and `provisioner` blocks. Similar to a `for` expression, it produces nested blocks that iterate a complex type of data and generate a

corresponding nested block for each element. In the above example:

- The label of `dynamic` , i.e. `"security_policy"` , determines the type of nested blocks to be generated.
- The `for_each` argument provides the complex type values to be iterated.
- The `iterator` argument (optional) sets the name of a temporary variable that represents the current iteration element. If `iterator` is not set, the temporary variable name will default to the label of the `dynamic` block, i.e. `security_policy` .
- The `labels` argument (optional) is an ordered list of block labels to generate a set of nested blocks in sequence. Temporary `iterator` variables can be used in expressions with the `labels` argument.
- The nested `content` block defines the body of the nested block to be generated. Temporary `iterator` variables can be used inside the `content` block.

`for_each` argument:

- As the `for_each` argument can be a collection or a structured type, you can use `for` or expanded expressions to convert the type of an existing collection.
- The value of `for_each` must be a non-empty map or set. If you need to declare a collection of resource instances based on nested data structures or combinations of elements in multiple data structures, you can use Terraform expressions and functions to generate appropriate values.

The `iterator` variable (setting in the above example) has the following attributes:

- key: if the iteration container is a map, then `key` is the key of the current element. If it is a list, then `key` is the subscript number of the current element in the list. In the case of a set produced by a `for_each` expression, `key` and `value` are equal, and `key` should not be used.
- value: value of the current element. A `dynamic` block can only generate nested block arguments within the current block definition. It is impossible to generate meta-arguments such as `lifecycle` and `provisioner` . Terraform must ensure the successful calculation of values for these meta-arguments.

CLI Command

Last updated : 2022-01-14 14:30:41

This document describes how to apply Terraform code and manage the infrastructure with the Terraform command-line interface (CLI).

Basic Features

Viewing command list

Terraform provides diversified command-line operations. You can enter `terraform` on the command line to see the complete list as shown below:

```
→  ~ terraform
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
```

You can use `-help` to view the detailed usage of specific subcommands. For example, you can run the `terraform validate -help` command to view the usage of the `validate` subcommand.

Switching working directory

The usual way to run Terraform is to first switch to the directory containing the .tf files for your root module (using the cd command), so that Terraform will automatically find those code files and argument files to be executed.

Global option -chdir

In some cases, particularly when wrapping Terraform in automation scripts, it can be convenient to run Terraform from a different directory than the root module directory. To allow that, Terraform supports a global option `-chdir=...` which you can include before the name of the subcommand you intend to run:

```
terraform -chdir=environments/production apply
```

The `-chdir` option instructs Terraform to change its working directory to the given directory before running the given subcommand. This means that any files that Terraform would normally read or write in the current working directory will be read or written in the given directory instead.
There are two exceptions where Terraform will use the original working directory even when you specify `-chdir` :

- Settings in the CLI Configuration are not for a specific subcommand and Terraform processes them before acting on the `-chdir` option.
- In case you need to use files from the original working directory as part of your configuration, a reference to `path.cwd` in the configuration will produce the original working directory instead of the overridden working directory. Use `path.root` to get the root module directory.

Auto-Completion

If `bash` or `zsh` is used, you can get the support for auto-completion with the following command.

```
terraform -install-autocomplete
```

You can run the following command to uninstall auto-completion.

```
terraform -uninstall-autocomplete
```

Basic Commands

Show All

terraform

展开&收起
The `terraform init` command is used to initialize a working directory containing Terraform configuration files. You should run this command first after writing Terraform code or cloning a Terraform project.

- **Usage**

  `terraform init [options]`

  This command initializes the current directory in a series of steps. It is always safe to run multiple times and will not delete configuration files or state information even if an error is reported.

- **General options**

  - `-input=true` : whether to ask for input in case no input variable value is obtained.
  - `-lock=false` : whether to lock a state file at runtime.
  - `-lock-timeout=¥` : timeout period for trying to get a state file lock. The default value is 0, indicating that an error will be reported as soon as the lock is found to have been held by another process.
  - `-no-color` : disables color codes in the command output.
  - `-upgrade` : whether to upgrade module code and plugins.

- **Source module copying**

  By default, the `terraform init` command assumes that the working directory already contains a configuration and will attempt to initialize that configuration. You can also run `terraform init` in an empty working directory with the `-from-module=MODULE-SOURCE` option, in which case the specified module will be copied into the current directory before any other initialization steps are run. This special mode of operation supports two use cases:

  - Checking out the specified code version and initializing the working directory for it if the version control system corresponds to the source.
  - Copying the sample code into a local directory to write new code accordingly if the module source points to an example project.

    For regular running operations, we recommend you check out the code from the version control system separately using the version control system's tool.

- **Backend initialization**

  During initialization, the root module code will be parsed to find the backend configuration, and the backend storage will be initialized with the given configuration settings.

  Re-running `init` with an already-initialized backend will update the working directory to use the new backend settings. The `init` command may prompt you for confirmation of state migration based on the changes. You can use the following options as needed:

  - `-force-copy` : skips the prompt to confirm the migration state directly.
  - `-reconfigure` : makes `init` ignore any existing configuration to prevent any state migration.
  - `-backend=false` : skips the backend configuration.

    Note that some initialization steps require an already initialized backend, and we recommend you use this option only after the backend has been initialized.

  - `-backend-config` : dynamically specifies the backend configuration.

- **Child module initialization**

  The `init` command will search for `module` blocks and get the module code with the `source`

argument. You can use the following options as needed:

- `-upgrade` : upgrades all modules to the latest code version. By default, re-running the `init` command after module installation will continue to install the modules added after the last `init` execution, but will not modify the already installed modules.
- `-get=false` : skips child module installation steps.

  Note that other initialization steps require a complete module tree, and we recommend you use this option only after the module has been successfully installed.

- **Plugin installation**

  The options are described as follows:

  - `-upgrade` : upgrades all previously installed plugins to the latest version in line with the version constraint. This option is invalid for manually installed plugins.
  - `-get-plugins=false` : skips plugin installation. Terraform will use plugins already installed in the current working directory or the plugin cache path. If these plugins are not sufficient to meet the requirements, `init` will fail.
  - `-plugin-dir=PATH` : skips plugin installation and loads plugins only from a specified directory. This option will skip plugins in the user plugin directory and all the current working directories. To restore the default behavior after this option is used, re-run `init` with the `-plugin-dir=""` options.
  - `-verify-plugins=false` : (not recommended) skips signature verification after plugin downloading (Terraform does not verify signatures of manually installed plugins). Official plugins are signed by HashiCorp and verified by Terraform.

terraform

展开&收起

The `terraform plan` command is used to create a change plan. Terraform will first run a refresh (this behavior can also be disabled explicitly):

- If changes are detected, you can decide which change to be executed in order to migrate the existing state to the desired state as described by the code. You can also use the optional `-out` option to save the change plan in a file for execution later using the `terraform apply` command.
- If no change is detected, you will be prompted that there is no change to be executed.

This command makes it easy to review all the details of a state migration without actually changing the existing resources and state files. For example, you can run `terraform plan` before committing the code to the version control system to confirm that the changes behave as expected.

- **Usage**

  `terraform plan [options]`

By default, the plan command does not require options. It runs a refresh using the code and state files in the current working directory.

- **General options**:
  - `-compact-warnings` : displays alarms only with a message summary if Terraform generates only alarm information but no error information.
  - `-destroy` : generates a plan to terminate all resources.
  - `-detailed-exitcode` : details the meanings of the code returned upon command exit:
    - 0 = successful empty plan (no change)
    - 1 = error
    - 2 = successful non-empty plan (with changes)
  - `-input=true` : whether to ask you to specify the input variable value in case no value is obtained.
  - `-lock=true` : similar to that of the `apply` command.
  - `-lock-timeout=0s` : similar to that of the `apply` command.
  - `-no-color` : disables color output.
  - `-out=path` : saves the change plan to a file in the specified path for execution using `terraform apply` .
  - `-parallelism=n` : limits the maximum parallelism of the Terraform traversing graph, with a default value of 10.
  - `-refresh=true` : executes a refresh before calculating changes.
  - `-state=path` : location of a state file, with a default value of `"terraform.tfstate"` . This option is invalid if remote backend is enabled.
  - `-target=resource` : address of the target resource. This option can be declared repeatedly to perform partial updates to the infrastructure.
  - `-var 'foo=bar'` : similar to that of the `apply` command.
  - `-var-file=foo` : similar to that of the `apply` command.
- **Partial update**

  Using the `-target` option allows Terraform to focus on a subset of resources, which can be marked with a resource address as described below:
  - If a resource can be located at a given address, only the resource will be marked. If the resource uses the `count` argument without a given access subscript, all instances of the resource will be marked.
  - If a module is located at a given address, all resources in the module and its embedded module resources will be marked.

  > Note

> Particular special scenarios, such as recovery from a previous error or bypassing certain Terraform design constraints, require the capability to mark partial resources and calculate update plans. The `-target` option is not recommended for routine operations, as it can cause undetectable configuration drift and make it impossible to deduce the current actual state from the code.

- **Security alarm**

  Change plan files saved (using the `-out` option) may contain sensitive information. We strongly recommend you encrypt the files by yourself for movement or save as Terraform does not encrypt them. Terraform is expected to launch new features to enhance the security of plan files.

terraform

展开&收起

As the most important command in Terraform, `terraform apply` is used to generate and (optionally) run an execution plan so that the infrastructure resource state matches the code description.

- **Usage**

  `terraform apply [options] [dir-or-plan]`

  By default, the `apply` command scans the current directory for code files and performs changes accordingly. Other code file directories can also be specified through options.

  When designing an automation pipeline, you can also explicitly divide the process into two steps: creating an execution plan and running the plan with the `apply` command. If no change plan file is explicitly specified, `terraform apply` will automatically create a change plan and ask you whether to approve the execution. If the created plan does not contain any changes, `terraform apply` will exit immediately without prompting you for input.

- **General options**

  - `-backup-path` : path to save a backup file, with a default value of the `-state-out` option plus the `".backup"` suffix. You can set it to "-" to disable backup (not recommended).
  - `-compact-warnings` : displays alarms only with a message summary if Terraform generates only alarm information but no error information.
  - `-lock=true` : whether to lock a state file first before execution.
  - `-lock-timeout=0s` : interval to attempt to get a state lock again.
  - `-input=true` : whether to prompt you for input in case no input variable value is obtained.
  - `-auto-approve` : skips interactive confirmation and runs changes directly.
  - `-no-color` : disables color output.

- `-parallelism=n` : limits the maximum parallelism of the Terraform traversing graph, with a default value of 10.
- `-refresh=true` : whether to query the current state of the recorded infrastructure object to refresh the state file first before specifying and running a change plan. This option is invalid if the command line specifies the change plan file to be executed.
- `-state=path` : path to save a state file, with a default value of `"terraform.tfstate"` . This option is invalid if a remote backend is used. It does not affect other commands; for example, the state file it sets will not be found when `init` is executed. If you want all commands to use the same location-specific state file, use a local backend.
- `-state-out=path` : path to write an updated state file, with a default value of `-state` . This option is invalid if a remote backend is used.
- `-target=resource` : specifies to update target resources by specifying resource addresses.
- `-var 'foo=bar'` : sets the value of a set of input variables. This option can be set repeatedly to pass in multiple input variable values.
- `-var-file=foo` : specifies an input variable file.

terraform

展开&收起

The `terraform destroy` command can be used to terminate and repossess all the Terraform-managed infrastructure resources.

Usage

`terraform destroy [options]`

Before Terraform-managed resources are terminated, you will be asked for confirmation on an interactive UI. This command can accept all options of the `apply` command, but cannot specify a plan file.

- If the `-auto-approve` option is `true` , resources will be terminated without your confirmation.
- If a resource is specified with the `-target` option, the resource will be terminated along with all the resources that depend on it.

> Note
> All the operations to be performed by `terraform destroy` can be previewed at any time by running the `terraform plan -destroy` command.

terraform

展开&收起

The `terraform graph` command is used to generate a visual graph of the infrastructure of the code description or the execution plan. Its output is in DOT format and can be converted to an image using GraphViz.

- **Usage**

  `terraform graph [options] [DIR]`

  This command generates a visual dependency graph of Terraform resources as described by the code lock in the DIR path (the current working directory is used if the default DIR option is used).

- **General options**
  - `-type` : specifies the type of diagram to be output. Terraform creates different graphs for different operations. The default type of the code file is `"plan"` , and that of the change plan file is `"apply"` .
  - `-draw-cycles` : highlights the rings in the graph with colored edges to help analyze ring errors in the code (Terraform prohibits ring dependencies).
  - `-type=plan` : generates different types of diagrams, including `plan` , `plan-destroy` , `apply` , `validate` , `input` , and `refresh` .

- **Image file creation**

  The `terraform graph` command outputs data in DOT format, which can be converted to an image file with the following command on GraphViz:

  ```
  terraform graph | dot -Tsvg > graph.svg
  ```

  If Graphviz is not installed, you can install it with the following command:
  - CentOS: `yum install graphviz`
  - Windows: `choco install graphviz`
  - macOS: `brew install graphviz`

    The output image is similar to the one as shown below:
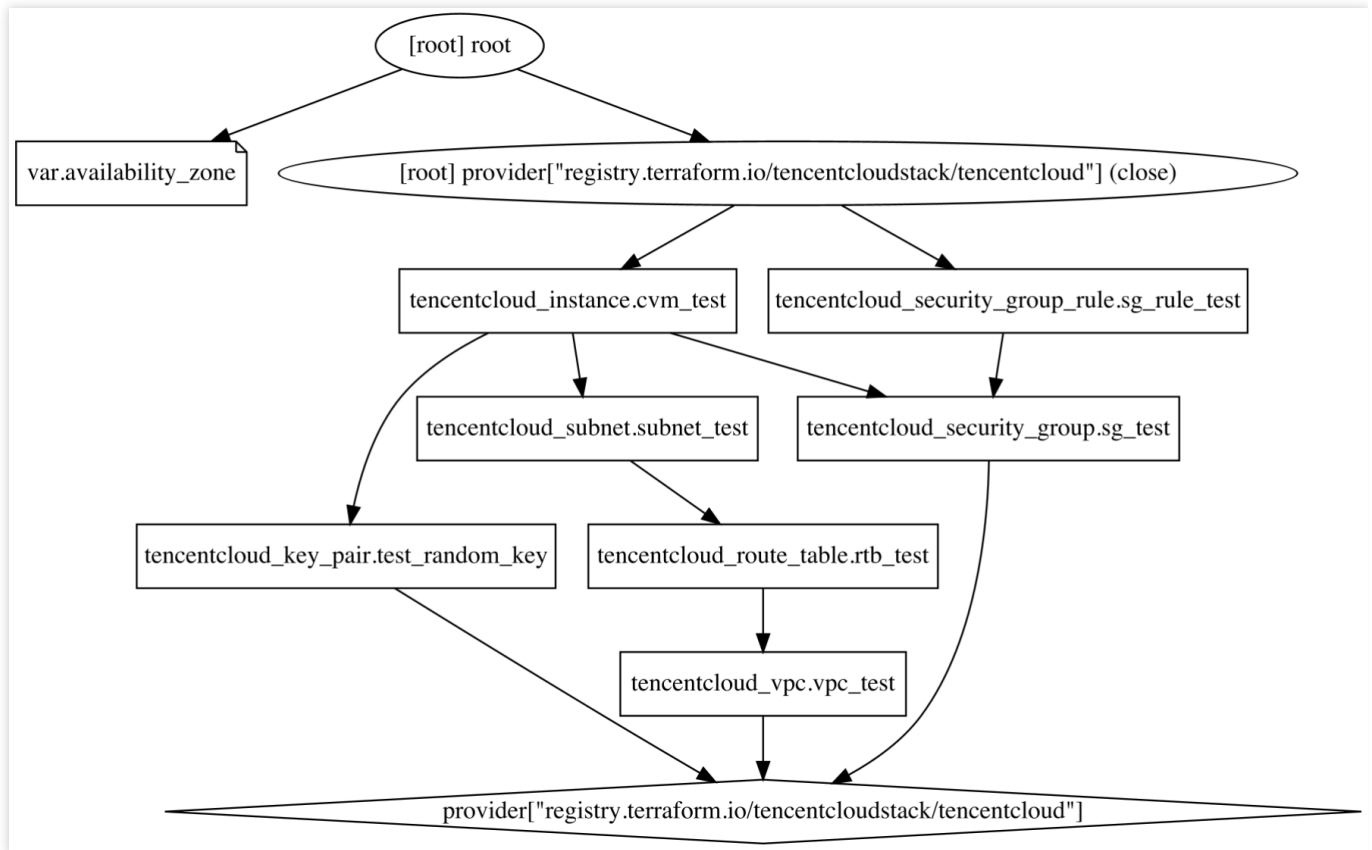
terraform

展开&收起

The `terraform show` command prints readable output from a state file or change plan file, which can be used to check the change plan to ensure that all operations are as expected, or to review the current state file.

> Note
> Machine-Readable JSON data can be output by adding the `-json` option, but all the data marked as `sensitive` will be output in plaintext.

- **Usage**

  `terraform show [options] [path]`

- **General options**
  - `path` : specifies a state file or change plan file. If no path is given, the state file corresponding to the current working directory will be used.
  - `-no-color` : similar to that of the `apply` command.

- `-json` : outputs in JSON format.
- **Output in JSON Format**

  The state information can be printed in JSON format using the `terraform show -json` command. If a change plan file is specified, `terraform show -json` will record the change plan, configuration, and current state in JSON format.

terraform

展开&收起

The `terraform import` command is used to import an existing resource object into Terraform.

If there exists a set of running infrastructure resources that are not built or managed using Terraform and corresponding Terraform code has been written for them, you can use `terraform import` to "import" the resource objects into the Terraform state file.

- **Usage**

  `terraform import [options] ADDRESS ID`

  `terraform import` finds the corresponding resource based on its resource ID (subject to the type of the imported resource object) and imports its information to the resource corresponding to `ADDRESS` in the state file. `ADDRESS` must be in the valid resource address format as described in the resource address. `terraform import` can import resources into not only root modules but also child modules.

  > Note
  > Each resource object in Terraform corresponds to only one actual infrastructure object. You need to avoid importing the same object to two and more addresses, which can lead to unpredictable behaviors in Terraform.

- **General options**
  - `-backup=path` : address of the generated state backup file, with a default value of the `-state-out` path plus the `".backup"` suffix. You can set it to "-" to disable backup (not recommended).
  - `-config=path` : path to the folder containing the Terraform code with the import target. The default path is the current working directory.
  - `-input=true` : whether to allow prompting for the input of provider configuration information.
  - `-lock=true` : whether to lock a state file with backend support.
  - `-lock-timeout=0s` : interval to attempt to get a state lock again.
  - `-no-color` : disables color output.

- `-parallelism=n` : limits the maximum parallelism of the Terraform traversing graph, with a default value of 10.
- `-state=path` : address of the state file to be read. The default address is the configured backend storage address or the `"terraform.tfstate"` file.
- `-state-out=path` : specifies the path to save a modified state file. By default, the source state file is overwritten. This option can be used to create a state file while avoiding corrupting the existing one.
- `-var 'foo=bar'` : sets the input variable value on the command line.
- `-var-file=foo` : similar to that of the `apply` command.

- **Provider configuration**

  Terraform will try to read the configuration information of the provider corresponding to the resource to be imported. If no relevant provider configuration is found, Terraform will ask you to input relevant access credentials. You can either input credentials or configure them through environment variables.

  The only restriction to read provider configuration in Terraform is that the configuration cannot rely on the input of "non-input variables", such as data sources.

  If you need to import Tencent Cloud resources, Terraform will use the `secret_id` and `secret_key` input variables to configure TencentCloud Provider. The configuration files are as follows:

  ```
  variable "secret_id" {}
  variable "secret_key" {}
  provider "tencentcloud" {
  secret_id = var.secret_id
  secret_key = var.secret_key
  }
  ```

  Upon the completion of configuration, you can import resources by running a command similar to the following:

  ```
  terraform import tencentcloud_instance.foo ins-2s6ewubw
  ```

Supported Resources

Last updated : 2022-01-14 14:49:15

The following resources are supported by Terraform:

Show All

API

展开&收起

- api_gateway_api
- api_gateway_api_key
- api_gateway_api_key_attachment
- api_gateway_custom_domain
- api_gateway_ip_strategy
- api_gateway_service
- api_gateway_service_release
- api_gateway_strategy_attachment
- api_gateway_usage_plan
- api_gateway_usage_plan_attachment

Anti-DDoS(Dayu)

展开&收起

- dayu_cc_http_policy
- dayu_cc_https_policy
- dayu_ddos_policy
- dayu_ddos_policy_attachment
- dayu_ddos_policy_case
- dayu_l4_rule
- dayu_l7_rule

Audit

展开&收起

audit

Auto

展开&收起

- as_attachment

- as_lifecycle_hook
- as_notification
- as_scaling_config
- as_scaling_group
- as_scaling_policy
- as_schedule

CVM

展开&收起

cdh_instance

Ckafka

展开&收起

- ckafka_acl
- ckafka_instance
- ckafka_topic
- ckafka_user

Cloud

展开&收起

- cam_group
- cam_group_membership
- cam_group_policy_attachment
- cam_policy
- cam_role
- cam_role_policy_attachment
- cam_saml_provider
- cam_user
- cam_user_policy_attachment

Cloud

展开&收起

- cbs_snapshot
- cbs_snapshot_policy
- cbs_snapshot_policy_attachment
- cbs_storage
- cbs_storage_attachment

Cloud

展开&收起

- ccn
- ccn_attachment
- ccn_bandwidth_limit

Cloud

展开&收起

- cfs_access_group
- cfs_access_rule
- cfs_file_system

Cloud

展开&收起

- alb_server_attachment
- clb_attachment
- clb_customized_config
- clb_instance
- clb_listener
- clb_listener_rule
- clb_log_set
- clb_log_topic
- clb_redirection
- clb_target_group
- clb_target_group_attachment
- clb_target_group_instance_attachment
- lb

Cloud

展开&收起

- cos_bucket
- cos_bucket_object
- cos_bucket_policy

Cloud

展开&收起

- eip

- eip_association
- image
- instance
- key_pair
- placement_group
- reserved_instance

Content

展开&收起

cdn_domain

CynosDB

展开&收起

- cynosdb_cluster
- cynosdb_readonly_instance

DNSPOD

展开&收起

dnspod_domain_instance

Direct

展开&收起

- dc_gateway
- dc_gateway_ccn_route

Direct

展开&收起

dcx

EMR

展开&收起

emr_cluster

Elasticsearch

展开&收起

elasticsearch_instance

Global

展开&收起

- gaap_certificate
- gaap_domain_error_page
- gaap_http_domain
- gaap_http_rule
- gaap_layer4_listener
- gaap_layer7_listener
- gaap_proxy
- gaap_realserver
- gaap_security_policy
- gaap_security_rule

KMS

展开&收起

- kms_external_key
- kms_key

MongoDB

展开&收起

- mongodb_instance
- mongodb_sharding_instance
- mongodb_standby_instance

Monitor

展开&收起

- monitor_alarm_policy
- monitor_binding_object
- monitor_binding_receiver
- monitor_policy_binding_object
- monitor_policy_group

MySQL

展开&收起

- mysql_account
- mysql_account_privilege
- mysql_backup_policy
- mysql_instance

- mysql_privilege
- mysql_readonly_instance

PostgreSQL

展开&收起

postgresql_instance

PrivateDNS

展开&收起

- private_dns_zone
- private_dns_record

Redis

展开&收起

- redis_backup_config
- redis_instance

SQLServer

展开&收起

- sqlserver_account
- sqlserver_account_db_attachment
- sqlserver_basic_instance
- sqlserver_db
- sqlserver_instance
- sqlserver_publish_subscribe
- sqlserver_readonly_instance

SSL

展开&收起

- ssl_certificate
- ssl_pay_certificate

SSM

展开&收起

- ssm_secret
- ssm_secret_version

Serverless

展开&收起

- scf_function
- scf_layer
- scf_namespace

TDMQ

展开&收起

- tdmq_instance
- tdmq_namespace
- tdmq_namespace_role_attachment
- tdmq_role
- tdmq_topic

TcaplusDB

展开&收起

- tcaplus_cluster
- tcaplus_idl
- tcaplus_table
- tcaplus_tablegroup

Tencent

展开&收起

- tcr_instance
- tcr_namespace
- tcr_repository
- tcr_token
- tcr_vpc_attachment

Tencent

展开&收起

- eks_cluster
- eks_container_instance
- kubernetes_addon_attachment
- kubernetes_as_scaling_group
- kubernetes_auth_attachment
- kubernetes_cluster

- kubernetes_cluster_attachment
- kubernetes_node_pool
- kubernetes_scale_worker

VPN

展开&收起

- vpn_connection
- vpn_customer_gateway
- vpn_gateway
- vpn_gateway_route

Video

展开&收起

- vod_adaptive_dynamic_streaming_template
- vod_image_sprite_template
- vod_procedure_template
- vod_snapshot_by_time_offset_template
- vod_sub_application
- vod_super_player_config

Virtual

展开&收起

- address_template
- address_template_group
- dnat
- eni
- eni_attachment
- ha_vip
- ha_vip_eip_attachment
- nat_gateway
- nat_gateway_snat
- protocol_template
- protocol_template_group
- route_entry
- route_table
- route_table_entry
- security_group

- security_group_lite_rule

- security_group_rule

- subnet

- vpc

- vpc_acl

- vpc_acl_attachment

OPS

Fixing Provider Update Failure

Last updated : 2022-01-14 14:30:42

Error Description

The following error message is returned when a provider is downloaded or updated:

```
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
Error installing provider "tencentcloud": openpgp: signature made by unknown entity.
Terraform analyses the configuration and state and automatically downloads
plugins for the providers used. However, when attempting to download this
plugin an unexpected error occured.
This may be caused if for some reason Terraform is unable to reach the
plugin repository. The repository may be unreachable if access is blocked
by a firewall.
If automatic installation is not possible or desirable in your environment,
you may alternatively manually install plugins by downloading a suitable
distribution package and placing the plugin's executable file in the
following directory:
terraform.d/plugins/darwin_amd64
```

Problem Locating

The update failed due to a lower Terraform version as illustrated at the official website.

Troubleshooting Procedure

Upgrade Terraform to 0.11.15 or later.

Managing Existing Resource

Last updated : 2022-01-14 14:30:42

Overview

This document describes how to use Terraform to manage resources created in the Tencent Cloud console.

Directions

To take over an existing resource (for example, TencentDB for PostgreSQL alarm policy in this document) on Terraform, you only need to reflect its state in both the source and state files of Terraform.

Getting resource ID

1. Log in to the CM console and select **Alarm Configuration** > **Alarm Policy** on the left sidebar.
2. Find and record the policy ID as shown below:
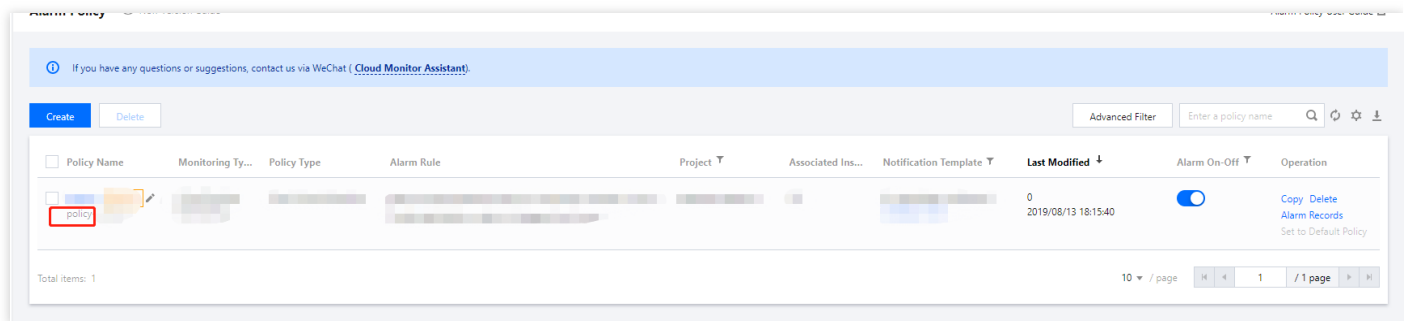
Installing Terraform

Install Terraform as instructed in Installing Terraform.

Importing resource file

1. Go to the Terraform working directory and run the following command to view the `main.tf` content.

   ```
   tencent-cloud cat main.tf
   ```

The returned result is as follows:

```
resource "tencentcloud_monitor_alarm_policy" "policy" {}
```

2. Run the following command in the directory where the file is located to complete the initialization.

```
terraform init --upgrade
```

The returned result is as follows:

```
Initializing the backend...
Initializing provider plugins...
- Finding latest version of tencentcloudstack/tencentcloud...
- Installing tencentcloudstack/tencentcloud v1.60.22...
- Installed tencentcloudstack/tencentcloud v1.60.22 (signed by a HashiCorp partner, key ID 84F69E
1C1BECF459)
Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html
Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.
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. After the initialization is completed, run the following command to import resources into the state file.

```
terraform import tencentcloud_monitor_alarm_policy.policy policy-vor9w72r
```

The following information is returned:

```
tencentcloud_monitor_alarm_policy.policy: Importing from ID "policy-vor9w72r"...
tencentcloud_monitor_alarm_policy.policy: Import prepared!
Prepared tencentcloud_monitor_alarm_policy for import
tencentcloud_monitor_alarm_policy.policy: Refreshing state... [id=policy-vor9w72r]
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.
```

4. Run the following command to view the state file.

```
cat terraform.tfstate
```

You can view the following resource information:

```
{
"version": 4,
"terraform_version": "1.1.0",
"serial": 1,
"lineage": "35791a73-d371-db51-5871-bfee13426217",
"outputs": {},
"resources": [
{
"mode": "managed",
"type": "tencentcloud_monitor_alarm_policy",
"name": "policy",
"provider": "provider[¥"registry.terraform.io/tencentcloudstack/tencentcloud¥"]",
"instances": [
{
"schema_version": 0,
"attributes": {
"conditions": [
{
"is_union_rule": 0,
"rules": [
{
"continue_period": 5,
"description": "cpu",
"filter": [],
"is_power_notice": 0,
"metric_name": "Cpu",
"notice_frequency": 86400,
"operator": "gt",
"period": 60,
"rule_type": "STATIC",
"unit": "%",
"value": "90"
}
]
}
],
"conditon_template_id": null,
"create_time": null,
"enable": 1,
"event_conditions": [
```

```
        {
          "continue_period": 0,
          "description": "HASwitch",
          "filter": [],
          "is_power_notice": 0,
          "metric_name": "ha_switch",
          "notice_frequency": 0,
          "operator": "",
          "period": 0,
          "rule_type": "",
          "unit": "",
          "value": ""
        }
      ],
      "id": "policy-vor9w72r",
      "monitor_type": "MT_QCE",
      "namespace": "POSTGRESQL",
      "notice_ids": [
        "notice-l9ziyxw6"
      ],
      "policy_name": "PgSql",
      "project_id": 0,
      "remark": "",
      "trigger_tasks": [],
      "update_time": null
    },
    "sensitive_attributes": [],
    "private": "eyJzY2hlbWFfdmVyc2lvbiI6IjAifQ=="
    }
  ]
  }
]
}
```

Updating source file

1. Run the following command to print the resource information.

```
terraform show
```

The following information is returned:

```
# tencentcloud_monitor_alarm_policy.policy:
resource "tencentcloud_monitor_alarm_policy" "policy" {
    enable = 1
    id = "policy-vor9w72r"
```

```
monitor_type = "MT_QCE"
namespace = "POSTGRESQL"
notice_ids = [
"notice-l9ziyxw6",
]
policy_name = "PgSql"
project_id = 0
conditions {
is_union_rule = 0
rules {
continue_period = 5
description = "cpu"
is_power_notice = 0
metric_name = "Cpu"
notice_frequency = 86400
operator = "gt"
period = 60
rule_type = "STATIC"
unit = "%"
value = "90"
}
}
event_conditions {
continue_period = 0
description = "HASwitch"
is_power_notice = 0
metric_name = "ha_switch"
notice_frequency = 0
period = 0
}
}
```

2. Copy the resource code to the Terraform source file `tencentcloud.tf`. You need to delete any options that cannot be set, such as ID.

Then, you will see the following `tencentcloud.tf` file:

```
provider tencentcloud {}
resource "tencentcloud_monitor_alarm_policy" "policy" {
enable = 1
# id = "policy-vor9w72r"
monitor_type = "MT_QCE"
namespace = "POSTGRESQL"
notice_ids = [
"notice-l9ziyxw6",
]
policy_name = "PgSql"
project_id = 0
```

```
conditions {
is_union_rule = 0
rules {
continue_period = 5
description = "cpu"
is_power_notice = 0
metric_name = "Cpu"
notice_frequency = 86400
operator = "gt"
period = 60
rule_type = "STATIC"
unit = "%"
value = "90"
}
}
event_conditions {
continue_period = 0
description = "HASwitch"
is_power_notice = 0
metric_name = "ha_switch"
notice_frequency = 0
period = 0
}
}
```

Verifying

Run the following command for a refresh using the code and state file in the current working directory.

```
terraform plan
```

The following information is returned, showing that the resource has been successfully taken over by Terraform.

```
tencentcloud_monitor_alarm_policy.policy: Refreshing state... [id=policy-vor9w72r]
No changes. Your infrastructure matches the configuration.
Terraform has compared your real infrastructure against your configuration and found no differenc
es, so no changes are needed.
```

You can delete the resource with the `destroy` command or modify it just like the code. After modifying the alarm threshold, run the following command for an update.

```
terraform plan
```

The following information is returned, showing that Terraform indicates that the alarm policy will be updated after value modification.

```
tencentcloud_monitor_alarm_policy.policy: Refreshing state... [id=policy-vor9w72r]
Terraform used the selected providers to generate the following execution plan. Resource actions
are indicated with the following symbols:
~ update in-place
Terraform will perform the following actions:
# tencentcloud_monitor_alarm_policy.policy will be updated in-place
~ resource "tencentcloud_monitor_alarm_policy" "policy" {
id = "policy-vor9w72r"
# (6 unchanged attributes hidden)
~ conditions {
# (1 unchanged attribute hidden)
~ rules {
~ value = "90" -> "99"
# (9 unchanged attributes hidden)
}
}
# (1 unchanged block hidden)
}
Plan: 0 to add, 1 to change, 0 to destroy.
```

Enabling Log Tracking

Last updated : 2022-01-14 14:30:42

Overview

This document describes how to enable local log tracking to get more detailed logs for self-check and assistance with ticket processing.

Directions

1. Before running `terraform apply` on the CLI, you can enable local log tracking with the following command:

```
export TF_LOG=DEBUG
export TF_LOG_PATH=./terraform.log
```

2. Run the following command:

```
terraform apply/destroy
```

After execution, you can see that the Terraform local folder generates a `terraform.log` file, which records the log output as defined by TencentCloud Provider.

Example

The following describes an execution error, along with the problem analysis and locating process.

In this example, a K8s cluster is created and an existing CVM instance is mounted to it as a node.

```
➜ terraform apply
2021/12/09 17:53:02 [WARN] Log levels other than TRACE are currently unreliable, and are supporte
d only for backward compatibility.
Use TF_LOG=TRACE to see Terraform's internal logs.
----
data.tencentcloud_instance_types.default: Refreshing state...
data.tencentcloud_cbs_storages.storages: Refreshing state...
```

```
data.tencentcloud_vpc_subnets.vpc2: Refreshing state...
data.tencentcloud_images.default: Refreshing state...
data.tencentcloud_vpc_subnets.vpc: 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:
# tencentcloud_kubernetes_cluster.managed_cluster will be created
+ resource "tencentcloud_kubernetes_cluster" "managed_cluster" {
+ certification_authority = (known after apply)
+ claim_expired_seconds = 300
+ cluster_as_enabled = false
+ cluster_cidr = "10.1.0.0/16"
+ cluster_deploy_type = "MANAGED_CLUSTER"
+ cluster_desc = "test cluster desc"
+ cluster_external_endpoint = (known after apply)
+ cluster_internet = false
+ cluster_intranet = false
+ cluster_ipvs = true
+ cluster_max_pod_num = 32
+ cluster_max_service_num = 32
+ cluster_name = "keep"
+ cluster_node_num = (known after apply)
+ cluster_os = "ubuntu16.04.1 LTSx86_64"
+ cluster_os_type = "GENERAL"
+ cluster_version = "1.10.5"
+ container_runtime = "docker"
+ deletion_protection = false
+ domain = (known after apply)
+ id = (known after apply)
+ ignore_cluster_cidr_conflict = false
+ is_non_static_ip_mode = false
+ kube_config = (known after apply)
+ network_type = "GR"
+ node_name_type = "lan-ip"
+ password = (known after apply)
+ pgw_endpoint = (known after apply)
+ security_policy = (known after apply)
+ user_name = (known after apply)
+ vpc_id = "vpc-h70b6b49"
+ worker_instances_list = (known after apply)
+ worker_config {
+ availability_zone = "ap-guangzhou-3"
+ count = 1
+ enhanced_monitor_service = false
+ enhanced_security_service = false
+ instance_charge_type = "POSTPAID_BY_HOUR"
+ instance_charge_type_prepaid_period = 1
```

```
+ instance_charge_type_prepaid_renew_flag = "NOTIFY_AND_MANUAL_RENEW"
+ instance_name = "sub machine of tke"
+ instance_type = "S1.SMALL1"
+ internet_charge_type = "TRAFFIC_POSTPAID_BY_HOUR"
+ internet_max_bandwidth_out = 100
+ password = (sensitive value)
+ public_ip_assigned = true
+ subnet_id = "subnet-1uwh63so"
+ system_disk_size = 60
+ system_disk_type = "CLOUD_SSD"
+ user_data = "dGVzdA=="
+ data_disk {
+ disk_size = 50
+ disk_type = "CLOUD_PREMIUM"
}
}
}
# tencentcloud_kubernetes_cluster_attachment.test_attach will be created
+ resource "tencentcloud_kubernetes_cluster_attachment" "test_attach" {
+ cluster_id = (known after apply)
+ hostname = "user"
+ id = (known after apply)
+ instance_id = "ins-lmnl6t1g"
+ labels = {
+ "test1" = "test1"
+ "test2" = "test2"
}
+ password = (sensitive value)
+ security_groups = (known after apply)
+ state = (known after apply)
+ worker_config {
+ docker_graph_path = "/var/lib/docker"
+ is_schedule = true
+ data_disk {
+ auto_format_and_mount = false
+ disk_size = 50
+ disk_type = "CLOUD_PREMIUM"
}
}
}
Plan: 2 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
tencentcloud_kubernetes_cluster.managed_cluster: Creating...
Error: [TencentCloudSDKError] Code=InternalError.CidrConflictWithOtherCluster, Message=DashboardE
rror,Code : -10013 , Msg : CIDR_CONFLICT_WITH_OTHER_CLUSTER[cidr 10.1.0.0/16 is conflict with clu
```

```
ster id: cls-1zc0kpyo], err : CheckCIDRWithVPCClusters failed,CIDR(10.1.0.0/16) conflict with clu
sterCIDR,ClusterID:cls-1zc0kpyo,clusterCIDR:10.1.0.0/16,err:CIDR1:10.1.0.0/16,firstIP:10.1.0.0,co
nflict with CIDR2:10.1.0.0/16, RequestId=d7dfb178-f081-480a-9bc3-89efc5fb1db5
on main.tf line 424, in resource "tencentcloud_kubernetes_cluster" "managed_cluster":
424: resource "tencentcloud_kubernetes_cluster" "managed_cluster" {
```

The CLI returns the following error:

```
[TencentCloudSDKError] Code=InternalError.CidrConflictWithOtherCluster, Message=DashboardError,Co
de : -10013 , Msg : CIDR_CONFLICT_WITH_OTHER_CLUSTER[cidr 10.1.0.0/16 is conflict with cluster i
d: cls-1zc0kpyo], err : CheckCIDRWithVPCClusters failed,CIDR(10.1.0.0/16) conflict with clusterCI
DR,ClusterID:cls-1zc0kpyo,clusterCIDR:10.1.0.0/16,err:CIDR1:10.1.0.0/16,firstIP:10.1.0.0,conflict
with CIDR2:10.1.0.0/16, RequestId=d7dfb178-f081-480a-9bc3-89efc5fb1db5
```

Problem analysis and locating:

1. Find `requestId: d7dfb178-f081-480a-9bc3-89efc5fb1db5` .

2. Open `terraform.log` , search for the `RequestId` , and find the following context:

```
2021-12-09T17:53:20.222+0800 [DEBUG] plugin.terraform-provider-tencentcloud.exe: 2021/02/25 1
7:53:20 [DEBUG] setting computed for "worker_instances_list" from ComputedKeys

_CONFLICT_WITH_OTHER_CLUSTER[cidr 10.1.0.0/16 is conflict with cluster id: cls-1zc0kpyo], err
 : CheckCIDRWithVPCClusters failed,CIDR(10.1.0.0/16) conflict with clusterCIDR,ClusterID:cls-1z
c0kpyo,clusterCIDR:10.1.0.0/16,err:CIDR1:10.1.0.0/16,firstIP:10.1.0.0,conflict with CIDR2:10.
1.0.0/16"},"RequestId":"d7dfb178-f081-480a-9bc3-89efc5fb1db5"}},cost 370.8109ms

6 is conflict with cluster id: cls-1zc0kpyo], err : CheckCIDRWithVPCClusters failed,CIDR(10.1.
0.0/16) conflict with clusterCIDR,ClusterID:cls-1zc0kpyo,clusterCIDR:10.1.0.0/16,err:CIDR1:10.
1.0.0/16,firstIP:10.1.0.0,conflict with CIDR2:10.1.0.0/16, RequestId=40d3ee5d-f723-4ef9-8f01-3
2d725464d51

2021-12-09T17:53:20.593+0800 [DEBUG] plugin.terraform-provider-tencentcloud.exe: 2021/12/09 1
7:53:20 common.go:79: [DEBUG] [ELAPSED] resource.tencentcloud_kubernetes_cluster.create elapse
d 371 ms
```

3. Log analysis shows that the problem occurred during the creation of the K8s cluster. Specifically, a conflict existed between the CIDR and another existing K8s cluster.

> Note
> If problem locating is difficult because the CLI prompt isn't clear enough or the error doesn't contain the `RequestId` , you can send the **TF project file, CLI error message, and its**

resulting terraform.log file by submitting a ticket for assistance.

Provider Co-Build
How It Works

Last updated : 2022-01-14 15:10:07

This document describes the directory structure and lifecycle of Terraform TencentCloud Provider.

Directory Structure

```
├──terraform-provider-tencentcloud Root directory
│  ├──main.go Program entry file
│  ├──AUTHORS Author information
│  ├──CHANGELOG.md Change log
│  ├──LICENSE License information
│  ├──debug.tf.example Example debugging configuration file
│  ├──examples Directory of example configuration files
│  │  ├──tencentcloud-eip Example EIP TF files
│  │  ├──tencentcloud-instance Example CVM TF files
│  │  ├──tencentcloud-nat Example NAT Gateway TF files
│  │  ├──tencentcloud-vpc Example VPC TF files
│  │  └── ... Directory of more examples
│  ├──tencentcloud Core provider directory
│  │  ├──basic_test.go Basic unit test
│  │  ├──config.go Public configuration file
│  │  ├──data_source_tc_availability_zones.go Availability zone query
│  │  ├──data_source_tc_availability_zones_test.go
│  │  ├──data_source_tc_nats.go NAT Gateway list query
│  │  ├──data_source_tc_nats_test.go
│  │  ├──data_source_tc_vpc.go VPC query
│  │  ├──data_source_tc_vpc_test.go
│  │  ├──... More data sources
│  │  ├──helper.go Some public functions
│  │  ├──provider.go Core provider file
│  │  ├──provider_test.go
│  │  ├──resource_tc_eip.go EIP resource manager
│  │  ├──resource_tc_eip_test.go
│  │  ├──resource_tc_instance.go CVM instance resource manager
│  │  ├──resource_tc_instance_test.go
│  │  ├──resource_tc_nat_gateway.go NAT Gateway resource manager
│  │  ├──resource_tc_nat_gateway_test.go
│  │  ├──resource_tc_vpc.go VPC Gateway resource manager
│  │  ├──resource_tc_vpc_test.go
│  │  ├──... More resource managers
```

```
|   |   ├──service_eip.go Encapsulated EIP-related service
|   |   ├──service_instance.go Encapsulated CVM-instance-related service
|   |   ├──service_vpc.go Encapsulated VPC-related service
|   |   ├──...
|   |   ├──validators.go Public argument validation function
|   ├──vendor Dependent third-party libraries
|   ├──website Web-Related files
|   |   ├──tencentcloud.erb Left sidebar file
|   |   ├──docs Source file directory of Markdown documents
|   |   |   ├──d Data-Related documents (data_source_*)
|   |   |   |   ├──availability_zones.html.md
|   |   |   |   ├──nats.html.markdown
|   |   |   |   ├──vpc.html.markdown
|   |   |   |   ├──...
|   |   |   ├──index.html.markdown
|   |   |   ├──r Resource-Related documents (resource_*)
|   |   |   |   ├──instance.html.markdown
|   |   |   |   ├──nat_gateway.html.markdown
|   |   |   |   ├──vpc.html.markdown
|   |   |   |   └──...
```

The structure is divided into five main parts:

- **main.go**: plugin entry.
- **examples**: example directory, which contains examples that can be used directly.
- **tencentcloud**: plugin directory to store service code, where:
  - `provider.go` : is the plugin root used to describe plugin attributes, such as the configured key, list of supported resources, and callback configuration.
  - `data_source_*.go` : defines some resources for read calls, mainly query APIs.
  - `resource_*.go` : defines some resources for write calls, including APIs for resource CRUD.
  - `service_*.go` : contains some public methods under broad resource categories.
- **vendor**: contains dependent third-party libraries.
- **website**: contains documents as importance as examples.

Provider Lifecycle

The Terraform execution process is as shown below:
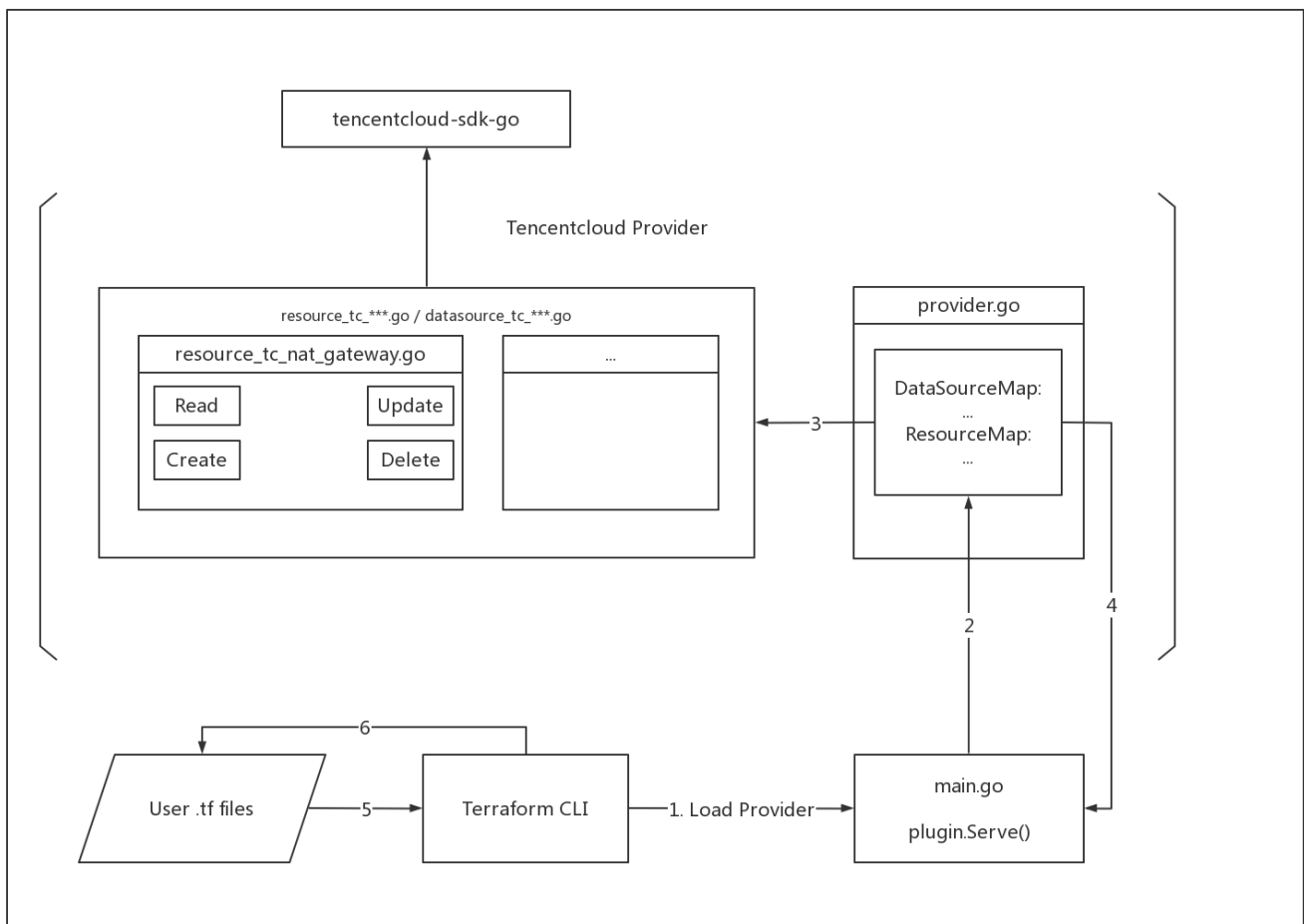
- `1 ~ 4` : looks for the provider and loads the tencentcloud plugin.
- `5` : reads your configuration file to get the resources you declared and their states.
- `6` : calls different functions (Create/Update/Delete/Read) based on the resource state.
  - **Create**
    Terraform determines adding a new resource configuration to a `.tf` file as `Create` .
  - **Update**
    Terraform determines modifying one or more arguments of a resource already created in a `.tf` file as `Update` .
  - **Delete**
    Terraform determines deleting the resource configuration already created in a `.tf` file or running the `terraform destroy` command as `Delete` .
  - **Read**
    `Read` is a resource query operation that checks whether a resource exists and updates the

resource attributes locally.

- **tencentcloud-sdk-go**

  `tencentcloud-sdk-go` is a TencentCloud API SDK for Go and used to call TencentCloud APIs for resource management.

Preparations

Last updated : 2022-01-14 14:30:42

Resource Constraint

This is an open-source project for both individual and team developers, and we welcome code contribution. Please observe the following rules for more efficient communication and development as well as a better user experience:

- Output product details, field lists, and corresponding APIs.
- Expose TencentCloud APIs for CRUD operations as required by products (at least the APIs for creation and deletion must be supported).
- Unique IDs or values such as names and SNs must be returned after resources are created.
- Input arguments must be able to be queried to ensure that the configuration and actual resource state are consistent.
- You must provide unit tests and ensure that they are passed.
- Single responsibility principle: do only one thing per change and avoid relying on or affecting other changes.

Code Development

You need to fork a copy of the code from the master repository to a subrepository. Develop the code as instructed in Development Notes and Development and Debugging, and ensure that the code can be executed after self-tests and unit tests are passed before committing it for push.
After the code is pushed, create a merge request to the master repository for code review.

Development Notes

Last updated : 2022-01-14 14:30:42

Official Repository

Official repository address: `https://github.com/terraform-providers/terraform-provider-tencentcloud` .

Development Steps

1. Prepare the Go development environment.
   The latest Go version is required, which is currently 1.14.x.

2. Register at GitHub.

3. Configure two-factor authentication to activate your GitHub account.

4. Fork an official repository.
   Fork the master branch of the official repository to your account.

5. Use `git clone` to clone the repository under your account to a local path.
   Run the following command to clone.

   ```
   git clone https://github.com/your-github-name/terraform-provider-tencentcloud
   ```

6. Complete the routine check.
   Run the following command to complete the check before committing.

   ```
   make hooks
   ```

7. Check out the branch.
   Check whether the branch format is `type/module-keyword` ; for example, `feat/tke-support-addon` indicates that the addon feature is added to the TKE module.

8. Modify the code.
   See Code Style for consistency with the existing style.

9. Modify the test case.

   If there are modifications, make sure that all the test cases are accurate and passed.

0. Implement automatic document generation.

   Run the following command to automate document generation. Your code must conform to certain rules as detailed in Terraform docs generator.

   ```
   make doc
   ```

1. Commit the code.

   Run the following command to commit the code. Make sure that the committed messages are as clear and standard as possible.

   ```
   git commit
   ```

2. Push the code.

   Run the following command to push the code to the repository under your account.

   ```
   git push
   ```

3. Commit the pull request.

   It is not allowed to merge the code directly to the official repository without PR + code review.

4. Notify others for code review.

   The code can be merged after approval by at least one member. You are not allowed to merge the code committed by yourself.

5. Release versions regularly.

   Plugins will be updated and features will take effect only after version release.

Code Style

Constrain the code style as instructed in Go Code Review Comments:

- Variable/Function names should follow the camel case convention, with the first letter in upper or lower case depending on access control.
- A single line of code should not contain more than 120 characters.

- Add a space between operators and operands, such as `num := a + b`, except when they are used as input or array subscripts.
- Import a package using its full but not relative path.
- Put the returned value of a function error at the end.
- Return an error or null as soon as possible when necessary. Do not put an `else` after the `if` statement returns a value.
- Return the error of a function call independently without judging other conditions.
- Do not use a magic variable twice, such as region ID `9` (for Singapore); instead, use a constant, such as `const AP_SINGAPORE = 9`.

> Note
> You can format code with the `make fmt` command. You must run `make hooks` containing a formatting step mentioned in the development steps.

Version/Tag Rule

Version names should follow the principles of Semver and be prefixed with `v`, such as `v1.2.3`.

CHANGELOG.md Rule

You must update `CHANGELOG.md` upon every modification according to the following versioning rules:

- **FEATURES**: when you add a data source or resource.
- **ENHANCEMENTS**: when you update a data source or resource.
- **BUG FIXES**: when you fix a bug.
- **DEPRECATED**: when you deprecate a data source, resource, or field.

According to the Semver master/sub/patch rules above, release a 1.X.0 subversion for FEATURES and DEPRECATED, and release a 1.0.X patch for ENHANCEMENTS and BUG FIXES.

Dynamic Input Limit

When users access Tencent Cloud services, you often need to limit the input arguments. For example, you need to limit the selectable database versions when a user purchases a TencentDB for

PostgreSQL instance. As the list of supported database versions is updated from time to time, without a timely update, the user may not be able to select the latest version, preventing normal use of the service. In this case, the list should not be specified directly in the code; instead, it should be dynamically pulled through an API.

This principle also applies to versions and memory specifications supported by TencentDB for PostgreSQL and TencentDB for Redis as well as programming languages and versions supported by SCF.

Development and Debugging

Last updated : 2022-01-14 14:30:42

This document describes how to perform basic Terraform development and debugging locally.

Step 1. Install Terraform

Install Terraform and configure global paths as instructed in Installing Terraform.

Step 2. Pull the provider

1. Go to terraform-provider-tencentcloud and fork the provider code to your personal repository.
2. Run the following commands in sequence to pull and set the upstream remote repository locally.

```
$ git clone https://github.com/{your username}/terraform-provider-tencentcloud # The code path
here is the personal code repository after the fork operation, which should be modified to the
actual path
```

```
$ cd terraform-provider-tencentcloud
```

```
$ git remote add upstream https://github.com/tencentcloudstack/terraform-provider-tencentcloud
```

After a successful pull, the following code structure can be viewed:

```
.
├──── .githooks/
├──── .github/
├──── examples/ # Sample code. In principle, ensure that users can directly copy and paste the o
utput code for use
├──── gendoc/ # Document generator
├──── scripts/
├──── tencentcloud/ # Product logic
├──── vendor/ # Local dependency cache
├──── website/ # Generated document directory
├──── .gitignore
├──── .go-version
├──── .golangci.yml
├──── .goreleaser.yml
├──── .travis.yml
```

```
├──── AUTHORS
├──── CHANGELOG.md
├──── GNUmakefile
├──── LICENSE
├──── README.md
├──── go.mod
├──── go.sum
├──── main.go
├──── staticcheck.conf
└──── tools.go
```

Step 3. Debug locally

1. Run the following command in the project's root directory to build the `terraform-provider-tencentcloud` binary file.

   ```
   go build
   ```

2. Create the `dev.tfrc` file with the following content and set `tencentcloudstack/tencentcloud` to point to the location of the binary file.

   ```
   provider_installation {
   # Use /home/developer/tmp/terraform-null as an overridden package directory
   # for the hashicorp/null provider. This disables the version and checksum
   # verifications for this provider and forces Terraform to look for the
   # null provider plugin in the given directory.
   dev_overrides {
   "tencentcloudstack/tencentcloud" = "path/to/your/provider/terraform-provider-tencentcloud"
   }
   }
   ```

3. Set the following environment variables.

   - Set the `TF_CLI_CONFIG_FILE` environment variable to point to the location of `dev.tfrc`.

     ```
     $ export TF_CLI_CONFIG_FILE=/Users/you/dev.tfrc
     ```

   - Set the `TF_LOG` environment variable to enable logging.

```
$ export TF_LOG=TRACE
```

○ Set your personal Tencent Cloud credentials, which can be obtained on the API Key Management page.

```
$ export TENCENTCLOUD_SECRET_ID=xxx
$ export TENCENTCLOUD_SECRET_KEY=xxx
```

4. At this point, the provider has been replaced locally. You can write your own `.tf` file and run commands such as `terraform plan/apply/destroy` for debugging.

Step 4. Perform unit testing

> Note
> - We strongly recommend you write your own unit test cases. You can view many `*_test.go` test cases under `tencentcloud/` .
> - A Terraform certified provider must have unit test cases.

1. The code of NAT Gateway is as follows:

```
package tencentcloud
import (
"encoding/json"
"fmt"
"log"
"testing"
"github.com/hashicorp/terraform/helper/resource"
"github.com/hashicorp/terraform/terraform"
"github.com/zqfan/tencentcloud-sdk-go/common"
vpc "github.com/zqfan/tencentcloud-sdk-go/services/vpc/unversioned"
)
func TestAccTencentCloudNatGateway_basic(t *testing.T) {
resource.Test(t, resource.TestCase{
PreCheck: func() { testAccPreCheck(t) },
Providers: testAccProviders,
// Configure the function for checking resource termination results
CheckDestroy: testAccCheckNatGatewayDestroy,
```

```go
// Configure test steps
Steps: []resource.TestStep{
{
// Configure the configuration content
Config: testAccNatGatewayConfig,
// Configure the validation function
Check: resource.ComposeTestCheckFunc(
// Verify resource IDs
testAccCheckTencentCloudDataSourceID("tencentcloud_nat_gateway.my_nat"),
// Verify resource attributes (a match indicates successful creation)
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "name", "terraform_test"),
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "max_concurrent", "3000000"),
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "bandwidth", "500"),
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "assigned_eip_set.#", "2"),
),
},
{
// Configure the configuration content
Config: testAccNatGatewayConfigUpdate,
Check: resource.ComposeTestCheckFunc(
testAccCheckTencentCloudDataSourceID("tencentcloud_nat_gateway.my_nat"),
// Verify the value of modified attributes (a match indicates successful modification)
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "name", "new_name"),
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "max_concurrent", "10000000"),
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "bandwidth", "1000"),
resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat", "assigned_eip_set.#", "2"),
),
},
},
})
}
// `testAccProviders` creates test resources based on Config before testing and terminates them all after testing
// This function is to check whether resources are terminated. The code logic is easy to understand, where the existence of a resource is queried by ID
func testAccCheckNatGatewayDestroy(s *terraform.State) error {
conn := testAccProvider.Meta().(*TencentCloudClient).vpcConn
// This uses the `s.RootModule().Resources` array
// The attributes of this array reflect the `terraform.tfstate` resource state file
for _, rs := range s.RootModule().Resources {
if rs.Type != "tencentcloud_nat_gateway" {
continue
}
descReq := vpc.NewDescribeNatGatewayRequest()
descReq.NatId = common.StringPtr(rs.Primary.ID)
descResp, err := conn.DescribeNatGateway(descReq)
b, _ := json.Marshal(descResp)
log.Printf("[DEBUG] conn.DescribeNatGateway response: %s", b)
```

```
if _, ok := err.(*common.APIError); ok {
return fmt.Errorf("conn.DescribeNatGateway error: %v", err)
} else if *descResp.TotalCount != 0 {
return fmt.Errorf("NAT Gateway still exists.")
}
}
return nil
}
// Basic usage configuration file, which is consistent with the debugged TF file
const testAccNatGatewayConfig = `
resource "tencentcloud_vpc" "main" {
name = "terraform test"
cidr_block = "10.6.0.0/16"
}
resource "tencentcloud_eip" "eip_dev_dnat" {
name = "terraform_test"
}
resource "tencentcloud_eip" "eip_test_dnat" {
name = "terraform_test"
}
resource "tencentcloud_nat_gateway" "my_nat" {
vpc_id = "${tencentcloud_vpc.main.id}"
name = "terraform_test"
max_concurrent = 3000000
bandwidth = 500
assigned_eip_set = [
"${tencentcloud_eip.eip_dev_dnat.public_ip}",
"${tencentcloud_eip.eip_test_dnat.public_ip}",
]
}
`
// Modify the usage configuration file to match the debugged TF file
const testAccNatGatewayConfigUpdate = `
resource "tencentcloud_vpc" "main" {
name = "terraform test"
cidr_block = "10.6.0.0/16"
}
resource "tencentcloud_eip" "eip_dev_dnat" {
name = "terraform_test"
}
resource "tencentcloud_eip" "eip_test_dnat" {
name = "terraform_test"
}
resource "tencentcloud_eip" "new_eip" {
name = "terraform_test"
}
resource "tencentcloud_nat_gateway" "my_nat" {
vpc_id = "${tencentcloud_vpc.main.id}"
```

```
name = "new_name"
max_concurrent = 10000000
bandwidth = 1000
assigned_eip_set = [
"${tencentcloud_eip.eip_dev_dnat.public_ip}",
"${tencentcloud_eip.new_eip.public_ip}",
]
}
`
```

2. Run the `TestAccTencentCloudNatGateway_basic` function to perform the unit test.

```
$ export TF_ACC=true
$ cd tencentcloud
$ go test -i; go test -test.run TestAccTencentCloudNatGateway_basic -v
```

This example shows that in addition to automatic compilation, the official `testAccProviders` has a more standardized testing process covering CRUD. You can write a more complex scenario for the same resource manager and then add it to steps or divide it into multiple test cases to make the testing more comprehensive.

Module Publish

Last updated : 2022-03-03 15:15:44

Overview

Modules are Terraform configurations that allow you to manage a group of resources and can provide better business abstraction and lower costs in some multi-resource scenarios. In addition, you can publish modules on GitHub to the [Terraform registry](). This document describes how to create and publish a Terraform TencentCloud module.

Creating a Public Module

Create a code repository on GitHub and name it in the format of `terraform-<provider>-<name>`, such as [terraform-tencentcloud-vpc]().

A basic module contains the following files:

```
.
├── main.tf # Write module resources
├── variables.tf # Declare module variables
├── outputs.tf # Declare module outputs
├── LICENCE # Declare license
└── README.md # Readme
```

You are advised to add the `examples` directory to store the examples for importing and using the module. For more information, [visit]().
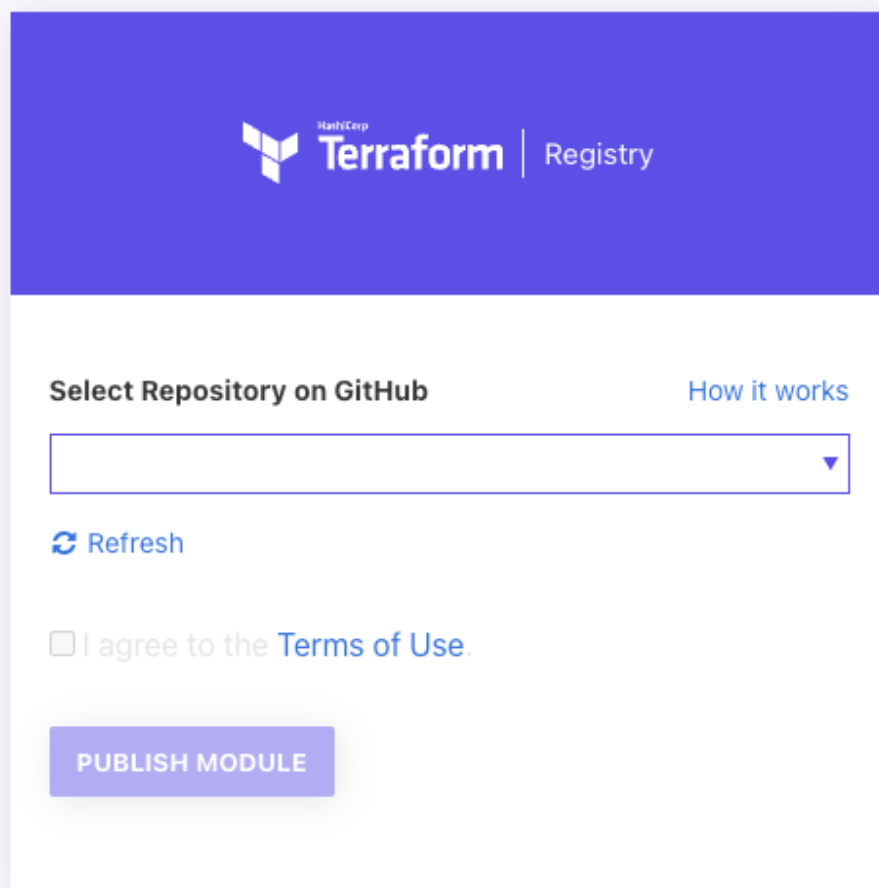
Publishing a Module

Directions

- Visit [registry.terraform.io]() and select **Publish** > **Module** in the top right.
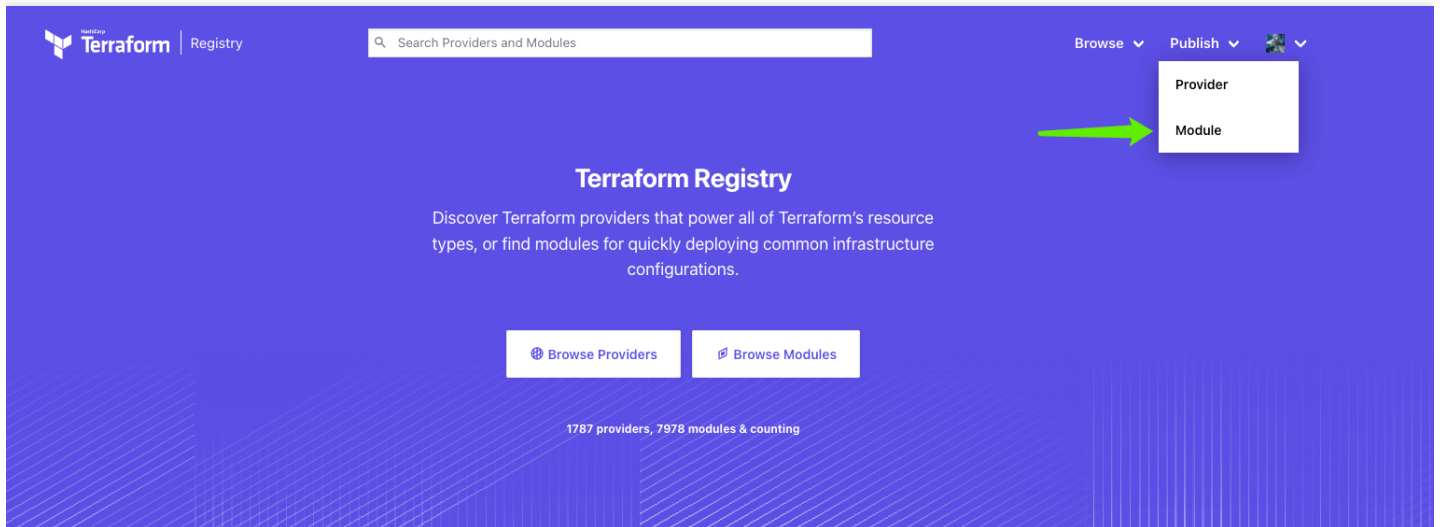
- Click the drop-down arrow for **Select Repository on GitHub** to view the module repositories you are authorized to manage. Then, select a repository and click **PUBLISH MODULE**.
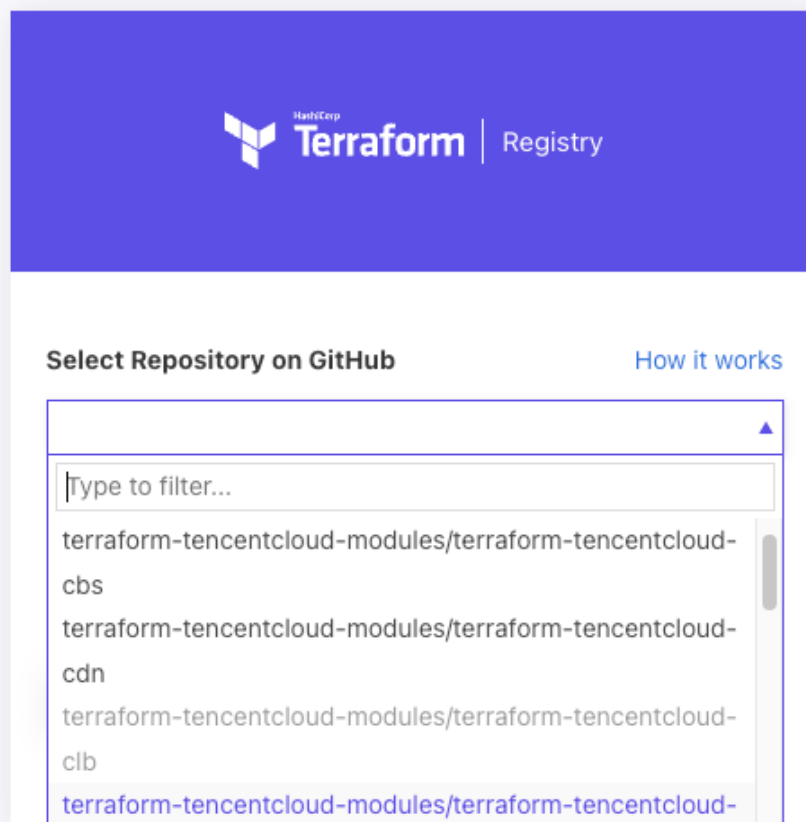
- Your repository will be automatically synchronized to the Terraform registry in a few minutes.

Note: You can also publish a module through a personal GitHub repository. The modules whose repositories are named in the format of `terraform-tencentcloud-<name>` will also be included in the tencentcloud modules.