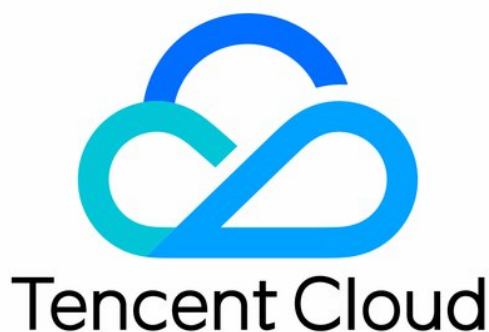


Tencent Infrastructure Automation for Terraform Provider Contributor Guide Product Documentation



Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Provider Contributor Guide

- Welcome

- Contribute

 - Writing Resource and Data Source

 - Bug Fix or Enhancement

 - Documentation Update

 - Tagging

 - Writing Test Case

 - Creating Pull Request

 - Submitting Changelog

 - Publishing Module

- Developer Reference

 - How It Works

 - Development and Debugging

 - Requirements and Suggestions

Provider Contributor Guide

Welcome

Last updated : 2024-12-02 16:01:33

[Terraform TencentCloud Provider](#) is maintained by the Tencent Cloud IaC team and welcomes contributions from developers.

Note:

This document is intended for Terraform TencentCloud Provider code developers.

Contribution

Contribute your code in the following steps. You can have a look at [How It Works](#) and [Requirements and Suggestions](#) to get a general picture of the provider.

1. Configuring the development environment

Before starting the development, you need to install Terraform and Go to pull the code repository, compile the program, and set the test as instructed in [Development and Debugging](#).

2. Writing the code

Write the code as instructed in the corresponding document based on the type of your code.

Contribution Type	Description
Resource and Data Sources	Add a resource or data source to Terraform TencentCloud Provider so as to manage Tencent Cloud product features or query the remote data of Tencent Cloud.
Bug Fix or Enhancement	Most requests are for feature enhancement or bug fix.
Tagging Support	Current resources require tag support and a unified tag service API.
Documentation Changes	Documentation updates and changes are included.

3. Writing the test

Cover all your contributed code in tests as instructed in [Writing Test Case](#).

4. Creating a pull request

When your contribution is ready, [create a pull request](#) in the repository of your provider.

5. Updating the changelog

An open-source project requires maintaining a user-friendly and readable `CHANGELOG.md` so that users can quickly know whether they will be affected by the release and assess the upgrade risk if applicable. For more information, see [Submitting Changelog](#).

Modules

Besides the source code, you can also submit modules to the provider as instructed in [Modules](#).

Contribute

Writing Resource and Data Source

Last updated : 2024-12-02 16:01:33

Resources and data sources are basic extensible datasets in Terraform that allow you to manage resources.

Writing Resources

A resource describes a class of resource entities allowing for CRUD operations, typically CVM instances, disks, databases, TKE clusters, and COS buckets.

This document takes the CVM instance with the following simple configuration as an example:

Name: `my-cvm`

AZ: Guangzhou Zone 4

Model: SA2.MEDIUM2

Image: TencentOS Server 3.2 (Final)

Network: `vpc-xxxxxxxx`

Subnet: `subnet-xxxxyyyy`

Billing mode: Pay-as-you-go

Whether a public IP is assigned: Yes

Public network bandwidth: 10 Mbps

You can use the high-level configuration syntax HCL for description. Below is the sample code:

```
resource "tencentcloud_instance" "cvm1" {
  instance_name           = "my-cvm"
  availability_zone       = "ap-guangzhou-4"
  instance_type           = "SA2.MEDIUM2"
  instance_charge_type    = "POSTPAID_BY_HOUR"
  image_id                = "img-9qrfy1xt"
  allocate_public_ip      = true
  internet_max_bandwidth_out = 10
}
```

When you run the `terraform apply` command for the first time, the declared resource will initiate the creation process. After the creation, if you modify the configuration in the code and run `terraform apply` again, the update process will be initiated. Running the `terraform destroy` command will initiate the termination process.

Writing resource code

To make the above HCL code effective, you need to register at TencentCloud Provider, find the

`Provider/ResourceMap` field in [tencentcloud/provider.go](https://tencentcloud.com/provider.go), add the `tencentcloud_instance` resource structure, and define the parameter schema based on HCL. Below is the sample code:

```
package tencentcloud

import (
    "github.com/hashicorp/terraform-plugin-sdk/helper/schema"
    "github.com/hashicorp/terraform-plugin-sdk/terraform"
)

func Provider() *schema.Provider {
    return &schema.Provider{
        ResourcesMap: map[string]*schema.Resource{
            "tencentcloud_xxx": { /* Other declared resources */ },
            "tencentcloud_yyy": { /* Other declared resources */ },
            "tencentcloud_instance": {
                Schema: map[string]*schema.Schema{
                    "instance_name": {
                        Optional: true, // Optional field
                        Type:      schema.TypeString, // Field type
                        Description: "Instance Name.",
                    },
                    "availability_zone": {
                        Required: true, // Required field
                        Type:      schema.TypeString,
                        ForceNew: true, // If this field is modified and the modification is su
                        Description: "Instance available zone.",
                    },
                    "instance_type": {
                        Optional: true,
                        Type:      schema.TypeString,
                        Description: "Instance Type.",
                    },
                    "instance_charge_type": {
                        Optional: true,
                        Type:      schema.TypeString,
                        Description: "Instance charge type.",
                    },
                    "image_id": {
                        Required: true,
                        Type:      schema.TypeString,
                        Description: "Instance OS image Id.",
                    },
                    "allocate_public_ip": {
                        Optional: true,
```

```

        Type:      schema.TypeBool, // Boolean type
        Description: "Specify whether to allocate public IP.",
    },
    "internet_max_bandwidth_out": {
        Optional: true,
        Type:      schema.TypeInt, // Integer type
        Description: "Specify maximum bandwidth.",
    },
},
},
},
}
}
}

```

After declaring the resource and parameter schema, you also need to define the CRUD logic triggered by `apply`, `plan`, or `destroy` in Terraform. The SDK for Terraform (v1) defines four CRUD methods:

`Create`, which is called if `terraform apply` is run for creation.

`Read`, which is called if `terraform plan / import` is run for remote state sync.

`Update`, which is called if `terraform apply` is run for update.

`Delete`, which is called if `terraform destroy` is run.

Add the four methods to the `schema.Resource` structure. Below is the sample code:

```

package tencentcloud

import (
    "github.com/hashicorp/terraform-plugin-sdk/helper/schema"
    "github.com/hashicorp/terraform-plugin-sdk/terraform"
)

func Provider() *schema.Provider {
    return &schema.Provider{
        ResourcesMap: map[string]*schema.Resource{
            "tencentcloud_xxx": { /* Other declared resources */ },
            "tencentcloud_yyy": { /* Other declared resources */ },
            "tencentcloud_instance": {
                Create: resourceTencentCloudInstanceCreate,
                Read:   resourceTencentCloudInstanceRead,
                Update: resourceTencentCloudInstanceUpdate,
                Delete: resourceTencentCloudInstanceDelete,
                Schema: map[string]*schema.Schema{
                    // The declaration written above (omitted)
                },
            },
        },
    },
}
}

```



```
}

func resourceTencentCloudInstanceCreate (d *schema.ResourceData, m interface{}) error
    instanceName := d.Get("instance_name").(string)
    instanceType := d.Get("instance_type").(string)
    // ...

    // Run `terraform apply` for resource creation
    instanceId := createInstance(&param{
        Name: &instanceName,
        Type: &instanceType,
        // ...
    })

    // Set a unique ID for the created resource
    d.SetId(instanceId)

    // Sync the remote state again after resource creation for consistency check
    return resourceTencentCloudInstanceRead(d, m)
}

func resourceTencentCloudInstanceRead (d *schema.ResourceData, m interface{}) error
    // Run `terraform plan` after resource creation/update

    instanceId := d.Id() // The ID set in `Create`
    instanceInfo := getInstance(instanceId)

    d.Set("instance_name", instanceInfo.Name)
    d.Set("instance_type", instanceInfo.Type)

    return nil
}

func resourceTencentCloudInstanceUpdate (d *schema.ResourceData, m interface{}) error
    // Run `terraform apply` to update existing resources

    updateParam := &param{}

    // Check whether fields are updated, and if not, update them in combinations
    if d.HasChange("instance_name") {
        name := d.Get("instance_name").(string)
        updateParam.Name = &name
    }

    if d.HasChange("instance_type") {
        insType := d.Get("instance_type").(string)
        updateParam.Type = &insType
    }
}
```

```
}
// ...
updateInstance(d.Id(), updateParam)

// Sync the remote state again after resource creation for consistency check
return resourceTencentCloudInstanceRead(d, m)
}

func resourceTencentCloudInstanceDelete (d *schema.ResourceData, m interface{}) error
// Run `terraform destroy`
destroyInstance(d.Id())

return nil
}
```

As you can see, the values of the HCL fields can be obtained by referencing the `d.Get` parameter. `d.Set` can write back (sync) these values. In addition, after each resource is created, you need to call `d.SetId` to set the unique resource index by which the specific resource instance can be located.

After the main logic structure is set up, the four functions can be called after Terraform runs `plan apply destroy`. Then, you can initiate TencentCloud API calls to manipulate resources.

CVM has the following CRUD APIs:

[RunInstances](#): Purchases an instance.

[DescribeInstances](#): Queries the list of instances.

[ModifyInstancesAttribute](#): Modifies an instance attribute.

[TerminateInstances](#): Returns an instance.

You can pass in the parameters obtained from HCL to the required parameters of the CVM API for call initiation.

Below is the sample code:

```
package main

import (
    cvm "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/cvm/v20170312"
)

func resourceTencentCloudInstanceCreate(d *schema.ResourceData, meta interface{}) error
// Read the values of the fields from HCL
instanceName      := d.Get("instance_name").(string)
instanceType      := d.Get("instance_type").(string)
imageId           := d.Get("image_id").(string)
instanceChargeType := d.Get("instance_charge_type").(string)
zone              := d.Get("availability_zone").(string)
allocatePublicIp  := d.Get("allocate_public_ip").(bool)
internetBandWith  := int64(d.Get("internet_max_bandwidth_out").(int))
```

```
client, _ := cvm.NewClient(credential, "ap-guangzhou")
request := cvm.NewRunInstancesRequest()
// Create the parameters required by CVM in combinations
request.InstanceName      = &instanceName
request.InstanceType      = &instanceType
request.ImageId           = &imageId
request.InstanceChargeType = &instanceChargeType
request.Placement        = &cvm.Placement {
    Zone: &zone,
}
request.InternetAccessible = &cvm.InternetAccessible {
    InternetMaxBandwidthOut: &internetBandWith,
    PublicIpAssigned:       &allocatePublicIp,
}

// Initiate the call
id, err := client.RunInstances(request)
if err != nil{
    return err
}

// Set the ID returned by the creation as the resource ID
d.SetId(id)

// Write back the state
return resourceTencentCloudInstanceRead(d, meta)
}
```

After implementing `Create`, `implement Read`, `Update`, and `Delete` :

```
package main

import (
    cvm "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/cvm/v20170312"
)

func resourceTencentCloudInstanceRead(d *schema.ResourceData, meta interface{}) error {
    // The ID set in `Create` or obtained during import
    id := d.Id()

    client, _ := cvm.NewClient(credential, "ap-guangzhou")
    request, _ := cvm.NewDescribeInstancesRequest()
    request.InstanceIds = []*string{&id}
    response, err := client.DescribeInstances(request)
    if err != nil{
        return err
    }
}
```

```
if len(response.Response.InstanceSet) == 0 {
    return fmt.Errorf("instance %s not exists.", id)
}
instance := response.Response.InstanceSet[0]

d.Set("instance_name", instance.InstanceName)
d.Set("instance_type", instance.InstanceType)
// `d.Set` writes back fields such as `image_id`, `instance_charge_type`, and `av

return nil
}

func resourceTencentCloudInstanceUpdate(d *schema.ResourceData, meta interface{}) error {
    id := d.Id()
    client, _ := cvm.NewClient(credential, "ap-guangzhou")
    request, _ := cvm.NewModifyInstancesAttributeRequest()
    request.InstanceIds = []*string{&id}

    // Check whether the fields are changed and add the parameters
    if d.HasChange("instance_name") {
        name := d.Get("instance_name").(string)
        request.InstanceName = &name
    }

    // Process other `HasChange` fields (omitted)

    _, err := client.ModifyInstanceAttribute(request)
    if err != nil {
        return err
    }

    return resourceTencentCloudInstanceRead(d, meta)
}

func resourceTencentCloudInstanceDelete(d *schema.ResourceData, meta interface{}) error {
    id := d.Id()

    client, _ := cvm.NewClient(credential, "ap-guangzhou")
    request, _ := cvm.NewTerminateInstancesRequest()
    request.InstanceIds = []*string{&id}
    _, err := client.TerminateInstances(request)
    if err != nil {
        return err
    }
    return nil
}
```

At this point, you have implemented the resource containing the name, field declaration, and CRUD logic in TencentCloud Provider.

Implementing the import logic

After implementing the resource CRUD successfully, you can implement the resource import logic. The command for importing a single resource is in the format of `terraform import <type>.<index> <id>`, such as:

```
terraform import tencentcloud_instance.cvm1 ins-abcd1234
```

As you can see, there is only the `ins-abcd1234` input for resource import. As indicated in the implemented `Read` method, when the ID is specified, you can use the ID to query the detailed configuration of a remote resource and automatically sync the configuration locally. Therefore, you only need to declare `Importer` in `schema.Resource` to indicate that the resource can be imported. Further sync will be taken over by `Read`:

```
package tencentcloud

import (
    "github.com/hashicorp/terraform-plugin-sdk/helper/schema"
    "github.com/hashicorp/terraform-plugin-sdk/terraform"
)

func Provider() *schema.Provider {
    return &schema.Provider{
        ResourcesMap: map[string]*schema.Resource{
            "tencentcloud_xxx": { /* Other declared resources */ },
            "tencentcloud_yyy": { /* Other declared resources */ },
            "tencentcloud_instance": {
                Create: resourceTencentCloudInstanceCreate,
                Read:   resourceTencentCloudInstanceRead,
                Update: resourceTencentCloudInstanceUpdate,
                Delete: resourceTencentCloudInstanceDelete,
                Schema: map[string]*schema.Schema{
                    // The declaration written above (omitted)
                },
                // In most cases, you only need to add `schema.ImportStatePassthrough`.
                Importer: &schema.ResourceImporter{
                    State: schema.ImportStatePassthrough,
                },
            },
        },
    }
}
```

Data Source

Data sources represent read-only entities for query only. No matter how many times they are called, existing resources will not be affected, such as CVM instance list, image list, AZ list, database list, parameter template, and audit log.

Theoretically, any data that can be found via APIs are data sources. As indicated above, the CVM image is displayed as `TencentOS Server 3.2 (Final)`, but the parameter passed in to the API is its ID `img-9qrfy1xt`. To get the ID based on the image name, you can use the data source to query and reference the image ID:

```
data "tencentcloud_images" "fav_os" {
  filters = {
    image_name: "TencentOS Server 3.2",
    image_type: "PUBLIC_IMAGE"
  }
}

resource "tencentcloud_instance" "cvm1" {
  image_id = data.tencentcloud_images.fav_os.images.0.image_id
}
```

As shown above, write `image_id` into the reference of `data.tencentcloud_images.fav_os`. Then, Terraform will read the data source information first and then calculate the value. Compared to a static method, a data source can effectively organize resources with dependencies.

Writing a data source

A data source is implemented in a similar way to a resource. You need to register at TencentCloud Provider, find the `Provider/DataSourceMap` field in the source code of [tencentcloud/provider.go](https://github.com/tencentcloud/tencentcloud-provider-go), add the `tencentcloud_images` structure (the names of all data sources must end with `s`), and define the parameter schema based on HCL:

```
package tencentcloud

import (
    "github.com/hashicorp/terraform-plugin-sdk/helper/schema"
    "github.com/hashicorp/terraform-plugin-sdk/terraform"
)

func Provider() *schema.Provider {
    return &schema.Provider{
        DataSourceMap: map[string]*schema.Resource{
            "tencentcloud_xxxs": { /* Other declared data sources */,
            "tencentcloud_yyys": { /* Other declared data sources */,
            "tencentcloud_images": {
                Schema: map[string]*schema.Schema{
```

```

"filters": {
  Optional: true,
  Type:      schema.TypeMap, // Specify the `Map` type
  Description: "Query filter",
},
// Saves the result in JSON locally
// TencentCloud Provider requires that every data source should have `result`
"result_output_file": {
  Optional: true,
  Type: schema.TypeString,
  Description: "Used for store as local file.",
},
"result": {
  Computed: true, // `Computed` indicates that the field will be updated
  Type:      schema.TypeList,
  Description: "",
  Elem: &schema.Resource{
    Schema: map[string]*schema.Schema{
      "id": {
        Type: schema.TypeString,
        Computed: true,
        Description: "Image Id.",
      },
      "name": {
        Type: schema.TypeString,
        Computed: true,
        Description: "Image name.",
      },
    },
  },
},
}
}

```

Compared to a resource, a data source requires implementing only the `Read` method:

```

package tencentcloud

import (
  "encoding/json"
  "github.com/hashicorp/terraform-plugin-sdk/helper/schema"
  "github.com/hashicorp/terraform-plugin-sdk/terraform"
)

```

```
const ImgNameFilterKey = "image-name"

func Provider() *schema.Provider {
    return &schema.Provider{
        DataSourceMap: map[string]*schema.Resource{
            "tencentcloud_xxxs": { /* Other declared data sources */ },
            "tencentcloud_yyys": { /* Other declared data sources */ },
            "tencentcloud_images": {
                Schema: map[string]*schema.Schema{ /* Omitted */},
                Read: func(d *schema.ResourceData, meta interface{}) {
                    // Declare the `Client` and `request`
                    client, _ := cvm.NewClient(credential, "ap-guangzhou")
                    request := cvm.NewDescribeImagesRequest()

                    // Get `Filters` of `Map` type
                    filters := d.Get("filters").(map[string]interface{})

                    // Check `filters["name"]` and add parameters
                    if name, ok := filters["name"].(string); ok {
                        request.Filters = append(request.Filters, &cvm.Filter{
                            Name: &ImgNameFilterKey,
                            Values: []*string{&name}
                        })
                    }

                    // Initiate the call
                    response, err := client.DescribeImages(request)
                    if err != nil{
                        return err
                    }

                    // Assemble and write the `result` field
                    imageSet := response.Response.ImageSet
                    result := make([]map[string]interface{}, 0, len(imageSet))
                    for i := range imageSet {
                        item := imageSet[i]
                        result = append(result, map[string]interface{}{
                            "id": *item.Id,
                            "name": *item.Name,
                        })
                    }
                    d.Set("result", result)

                    // If `result_output_file` has a definition, write the result into the sp
                    if v, ok := d.GetOk("result_output_file"); ok {
                        writeToFile(v.(string), result)
                    }
                }
            }
        }
    }
}
```



```
        }
        return nil
    }
},
},
}
```

At this point, you have written the `tencentcloud_images` and can use the data source to query specified OS images.

Writing an Acceptance Test

To ensure that your newly added resource or data source can work properly, you need to write an acceptance test as instructed in [Writing Test Case](#).

Bug Fix or Enhancement

Last updated : 2024-12-02 16:01:33

The provider requires continuous improvements and iteration. You can contribute your code to the repository in line with the minimal change principle, that is, implement only one enhancement or bug fix in each pull request to avoid large-scale or cross-feature code changes that can increase the test and review costs per time.

Checking the Code

Your merge request will trigger GitHub actions related to merge check, such as formatting, documentation generation, and acceptance test. When these checks are passed, we will review your code and give our feedback or directly merge it. Before the branch is pushed to the remote, you can execute these steps locally.

Setting the commit hook

Run `make hooks` in the directory of the local repository to install the dependencies of the formatting check and add `pre-commit.sh` to the commit hook.

Formatting the code

Run `make fmt` to format the `go` code and import order.

Syncing the documentation

Run `make doc`. Then, the `./gendoc` script will parse and sync any change you made to `./website`.

Writing an Acceptance Test

To ensure that your changes can work properly, you need to write acceptance test cases or add parameter assertions to existing cases to cover your changes. For detailed directions, see [Writing Test Case](#). Testing paid resources will initiate the actual billing process. Or you can write test cases and have them executed by us.

Documentation Update

Last updated : 2024-12-02 16:01:33

The source files of the TencentCloud Provider [documentation page](#) are in the `website` root directory of the project and are extracted and generated automatically from the code comments and schema description. This document describes the mechanism for generating the sidebar, sample code, and parameter description on the documentation page.

Documentation Page Generation

The documentation page consists of the following sections.

Directory

In the `tencentcloud/provider.go` file, the top comment starts with `Resources List` in the following format:

```
/*
Resources List

Product name
  Data Source
    tencentcloud_foos
    tencentcloud_bars
  Resource
    tencentcloud_baz

*/

package tencentcloud
```

The above text will be parsed into the `Product name -> DataSource / Resource -> tencentcloud_*` tree structure displayed on the left sidebar of the documentation page.

Sample

The topic page of the documentation consists of the overview, sample use, and parameter description. All samples are generated from the header comments in the `.go` file of each module. Below is a template:

```
/*
Provides a resource/datasource to create/query something.
```

```
~> **NOTE:** This is an optional TIPS, add it if needed.
```

Example Usage

Basic usage

```
hcl
resource "tencentcloud_foo" "foo" {
  name      = "baz"
}
```

Another usage

```
hcl
resource "tencentcloud_foo" "foo" {
  name      = "baz"
  another   = 12345
}
```

Import

This resource can be imported, e.g.

```
bash
$ terraform import tencentcloud_foo.foo foo_id

*/
package tencentcloud
// ...
```

First, the resource/data source use and instructions (if any) are presented, followed by the sample code block starting with `Example Usage`. Then, the import command is written if the type is `Resource` and the import method is provided.

Parameter description

The parameter description of each resource is parsed from the value of `Description` and closely follows the sample code:

```
map[string]*schema.Schema{
  "name": {
    Type: schema.TypeString,
    Description: "This is what will generated.",
  }
}
```

The script will read the `Schema` from the `Provider` to output the `Name - (Constraint, type)` introductory text item.

Documentation Update

Active update

The script generating the documentation is in `./gendoc`. If the code in the above location changes, run `cd` to enter `./gendoc` and run `go run .../.` to generate the documentation. Or you can run `make doc` in the project directory.

Commit for check

If you have configured the commit hook in the project, even if you don't run `gendoc`, the commit hook will automatically check for sync. If the documentation changes after the execution of the commit hook, you haven't synced the changed documentation to the index, and the commit operation will stop. Even if the hook is not configured, the merge check will do the job to ensure that your changes are accurately synced.

Tagging

Last updated : 2024-12-02 16:01:33

Tag is a cloud resource management tool that exists in the format of `key:values` . It can be associated with most of your Tencent Cloud resources, greatly simplifying resource categorization, search, and aggregation. In Terraform, a resource tag is defined through `Map` :

```
resource "tencentcloud_instance" "cvm" {
  tags = {
    key1: "val1",
    key2: "val2"
  }
}
```

Tag code implementation

A resource tag can be added via TencentCloud API in two ways. One is using the `CreateAPI` parameter:

```
rsType := "instance"

request := cvm.NewRunInstanceRequest()
request.TagSpecification = append(request.TagSpecification, &cvm.TagSpecification{
  ResourceType: &rsType,
  Tags: []*cvm.Tag{
    {
      Key: &key,
      Value: &value,
    },
  },
})
```

Currently, only certain resource creation APIs support passing in tags, with different data structures. To unify the management of the tag code, the second way is used, that is, calling the tag API after creation for association. The input parameter is the [six-segment resource description](#) in the following format:

```
qcs:<project_id, which is left empty here>:<Module>:<Region>:<Account/UIN>:<Resource>
```

For example, to modify the tag associated with a VPC instance, use the following input parameter:

```
qcs::vpc:ap-singapore:uin/:vpc/vpc-xxxxxxxx
```

In the code, just call the encapsulated `ModifyTags` :

```
package main

func ResourceTencentCloudVPCUpdate(d *schema.ResourceData, meta interface{}) {
    ctx := context.TODO()
    region := meta.(*TencentCloudClient).apiV3Conn.Region
    id := d.Id()

    resourceName := fmt.Sprintf("qcs::vpc:%s:uin/:vpc/%s", region, id)
    replaceTags, deleteTags := diffTags(oldTags.(map[string]interface{}), newTags.(ma

    if err := tagService.ModifyTags(ctx, resourceName, replaceTags, deleteTags); err
        return err
    }
}
```

Writing Test Case

Last updated : 2024-12-02 16:01:33

Complete test cases are necessary for a comprehensive program system. The SDK for Terraform integrates a test suite to verify written Terraform resources and data sources. This document describes how to write a test case for TencentCloud Provider.

Acceptance test

An acceptance test covers the entire lifecycle of a resource, including creation, query, update, import, and deletion. You need to write at least one test case for a resource. Running the acceptance test will **initiate an actual API call and affect Tencent Cloud resources**. Check your account costs before running the test, which can be written in the following steps:

Set environment variables and toggle on acceptance test execution:

```
export TF_ACC=true
```

Set the environment variables of the credential and specify an account to run the test:

```
export TENCENTCLOUD_SECRET_ID=xxxx
export TENCENTCLOUD_SECRET_KEY=yyyy
```

Set the log output level and directory for easier debugging:

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

For example, test the input parameters of a VPC resource to check whether the created and updated resource is as expected.

Create the `resource_tc_vpc_test.go` file in the `tencentcloud` directory, specify a VPC resource, and write its initial configuration and updated configuration:

```
// filename: tencentcloud/resource_tc_vpc_test.go
const testAccVpcConfig = `
resource "tencentcloud_vpc" "foo" {
  name          = "test-vpc"
  cidr_block    = "172.16.0.0/16"
}
`

const testAccVpcConfigUpdate = `
resource "tencentcloud_vpc" "foo" {
```



```
name      = "test-vpc__update"
cidr_block = "172.16.0.0/22"
is_multicast = true
}
```

Write the function of the case. A function name starting with `TestAccTencentCloud` indicates an acceptance test. The function name format is `TestAccTencentCloud${Module name}${Resource type}_${Subname}`, and the regex is `TestAccTencentCloud[a-zA-Z]+(Resource|DataSource)_[a-zA-Z]+`. Call the function to import the two configuration items and add the assertions in `resource.Test`:

```
package tencentcloud

import (
    "testing"

    "github.com/hashicorp/terraform-plugin-sdk/helper/resource"
    "github.com/hashicorp/terraform-plugin-sdk/terraform"
)

func TestAccTencentCloudVpcResource_Basic(t *testing.T) {
    resource.Test(t, resource.TestCase{
        Providers:     testAccProviders,
        Steps: []resource.TestStep{
            {
                // The initial configuration declared above
                Config: testAccVpcConfig,
                Check: resource.ComposeTestCheckFunc(
                    testAccCheckVpcExists("tencentcloud_vpc.foo"),
                    resource.TestCheckResourceAttr("tencentcloud_vpc.foo", "cidr_block", "172.16.0.0/22"),
                    resource.TestCheckResourceAttr("tencentcloud_vpc.foo", "name", "test-vpc"),
                ),
            },
            {
                // The updated configuration declared above
                Config: testAccVpcConfig,
                Check: resource.ComposeTestCheckFunc(
                    testAccCheckVpcExists("tencentcloud_vpc.foo"),
                    resource.TestCheckResourceAttr("tencentcloud_vpc.foo", "cidr_block", "172.16.0.0/22"),
                    resource.TestCheckResourceAttr("tencentcloud_vpc.foo", "name", "test-vpc__update"),
                    resource.TestCheckResourceAttr("tencentcloud_vpc.foo", "is_multicast", "true"),
                ),
            },
        },
    })
}
```

Then, run `go test -v -run TestAccTencentCloudVpcResource ./tencentcloud` in the root directory of the project to check the test result:

```
TestAccTencentCloudVpcResource_Basic
=== RUN    TestAccTencentCloudVpcResource_Basic
=== PAUSE TestAccTencentCloudVpcResource_Basic
=== CONT   TestAccTencentCloudVpcResource_Basic
--- PASS: TestAccTencentCloudVpcResource_Basic (26.30s)
PASS
ok      github.com/tencentcloudstack/terraform-provider-tencentcloud/tencentcloud
```

If log environment variables are set, the log details will be written into the `tencentcloud/terraform.log`.

Unit test

Compared to an acceptance test, a unit test is more refined and cheaper, which is suitable for checking whether logically complex code can be correctly executed.

For example, to use the `isExpectError(err)` function to check the TencentCloud API error code in the project in order to decide whether the program should retry (due to unstable client network connection, for example) or exit abnormally, write the following test case starting with `Test*`:

```
package tencentcloud

import (
    "testing"
    sdkErrors "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common/errors"
    "github.com/stretchr/testify/assert"
)

func TestIsExpectError(t *testing.T) {

    err := sdkErrors.NewTencentCloudSDKError("ClientError.NetworkError", "", "")

    // Expected
    expectedFull := []string{"ClientError.NetworkError"}
    expectedShort := []string{"ClientError"}
    assert.Equalf(t, isExpectError(err, expectedFull), true, "")
    assert.Equalf(t, isExpectError(err, expectedShort), true, "")

    // Unexpected
    unexpectedMatchHead := []string{"ClientError.HttpStatusCodeError"}
    unexpectedShort := []string{"SystemError"}
    assert.Equalf(t, isExpectError(err, unexpectedMatchHead), false, "")
    assert.Equalf(t, isExpectError(err, unexpectedShort), false, "")
}
```

```
}
```

Run `go test -v -run TestIsExpectError ./tencentcloud` in the root directory of the project to view the result:

```
=== RUN   TestIsExpectError
--- PASS: TestIsExpectError (0.00s)
PASS
ok      github.com/tencentcloudstack/terraform-provider-tencentcloud/tencentcloud
```

Cleaning up test resources

Terraform provides a mechanism called sweepers to clean up test resources, specifically, those not repossessed due to a test exception. Sweepers are a group of functions that can be filtered by parameter for selective execution. You need to declare and write the specific deletion logic. For example, to clean up VPC resources, add the `init` function to `tencentcloud/resource_tc_vpc_test.go` and register a sweeper named

```
tencentcloud_vpc :
```

```
func init() {
    resource.AddTestSweepers("tencentcloud_vpc", &resource.Sweeper{
        Name: "tencentcloud_vpc",
        F:    testSweepVpcInstance,
    })
}

// Pseudocode logic for implementing the cleanup of VPC resources starting with `te
func testSweepVpcInstance(region string) {
    vpcs := getAllVpc(region)
    for _, vpc in range vpcs {
        if vpc.name == "test" {
            deleteVpc(vpc.id)
        }
    }
}
```

To clean up VPC instances in a specified region (such as Guangzhou), run `go test -v ./tencentcloud -sweep=ap-guangzhou -sweep-run=tencentcloud_vpc`. Then, the SDK for Terraform will match the `tencentcloud_vpc` sweeper, pass in the region specified by `-sweep` to the function, and execute the function to clean up the resources in the region.

Creating Pull Request

Last updated : 2024-12-02 16:01:33

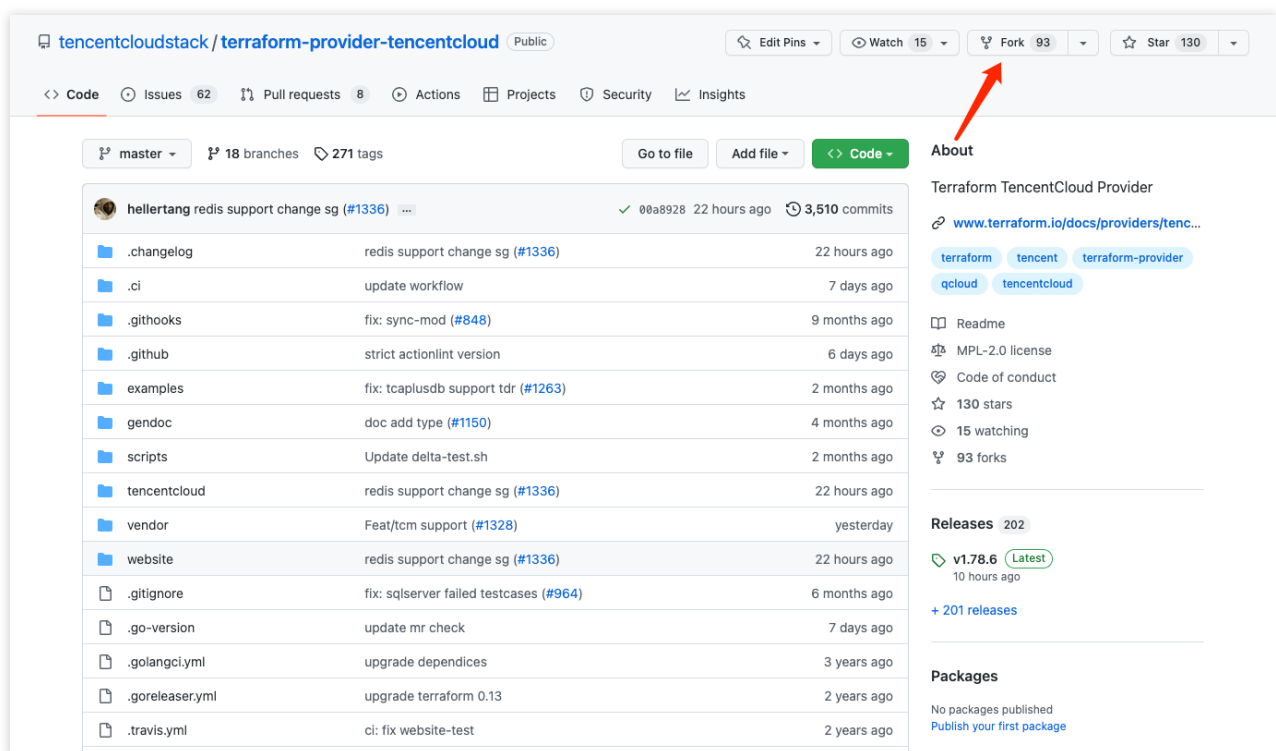
Any individual or team can contribute code to TencentCloud Provider and send a pull request to it in the following process:

Official repository

Visit the [official repository](#).

Forking code

You need to fork the main repository and make changes to the code in the newly created repository.



Branch naming convention

Generally, a branch needs to be named in the format of `type/scope-content` so that others can quickly locate the changed scope and content. Common prefixes are:

`fix/*` : Issue fix.

`feat/*` : Feature addition.

`doc/*` : Document change.

`style/*` : Format, spelling, or other code changes that do not affect the logic.

`chore/*` : Chore submission that is irrelevant to the code logic.

Summarize the changed module and content in the suffix, such as:

`fix/tke-auth-retry` : The authentication retrial issue is fixed for the TKE module.

`feat/new-free-ssl-resource` : An SSL resource is added.

`doc/cvm-field-misspell` : The spelling of a word is corrected in the CVM document.

Avoid names such as:

`john-test` : The name of a developer.

`fix/20221027` : The changed scope and content are not indicated.

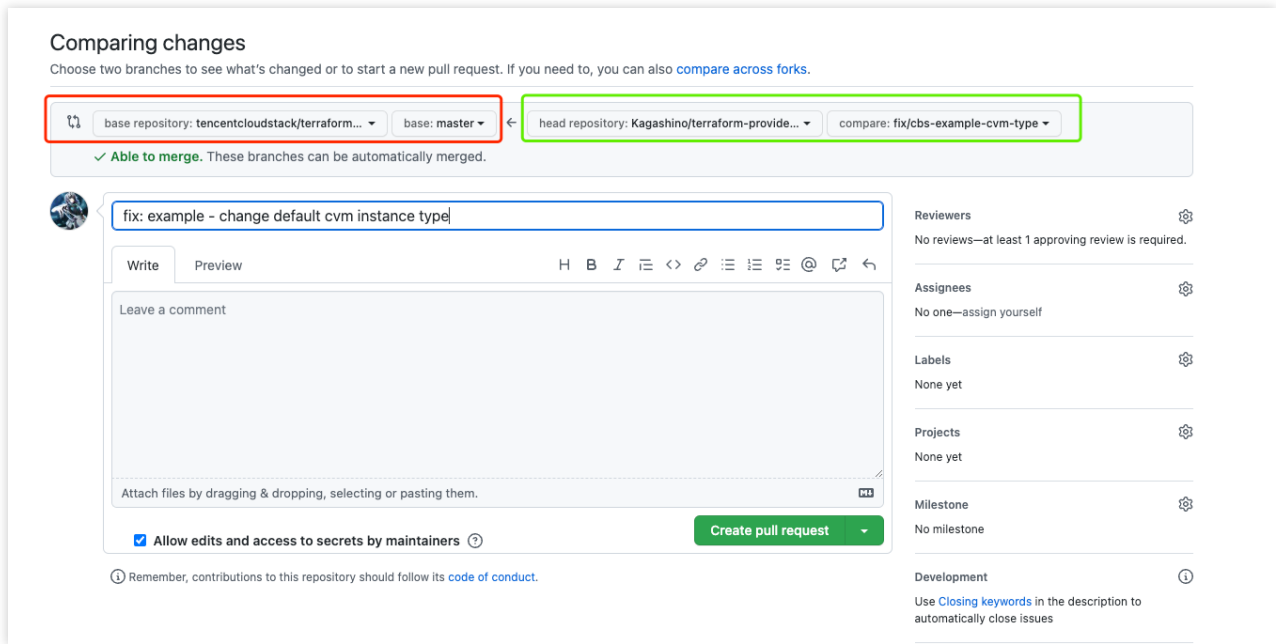
`fix/bug` : Improper content is contained.

Acceptance test

To ensure that your changes work properly, you need to write and execute an acceptance test for logic changes as instructed in [Writing Test Case](#).

Sending a pull request

After the change, create a merge request to the [main repository](#). Select the main repository in the red box and your repository in the green box as shown below:



Submitting the changelog inventory

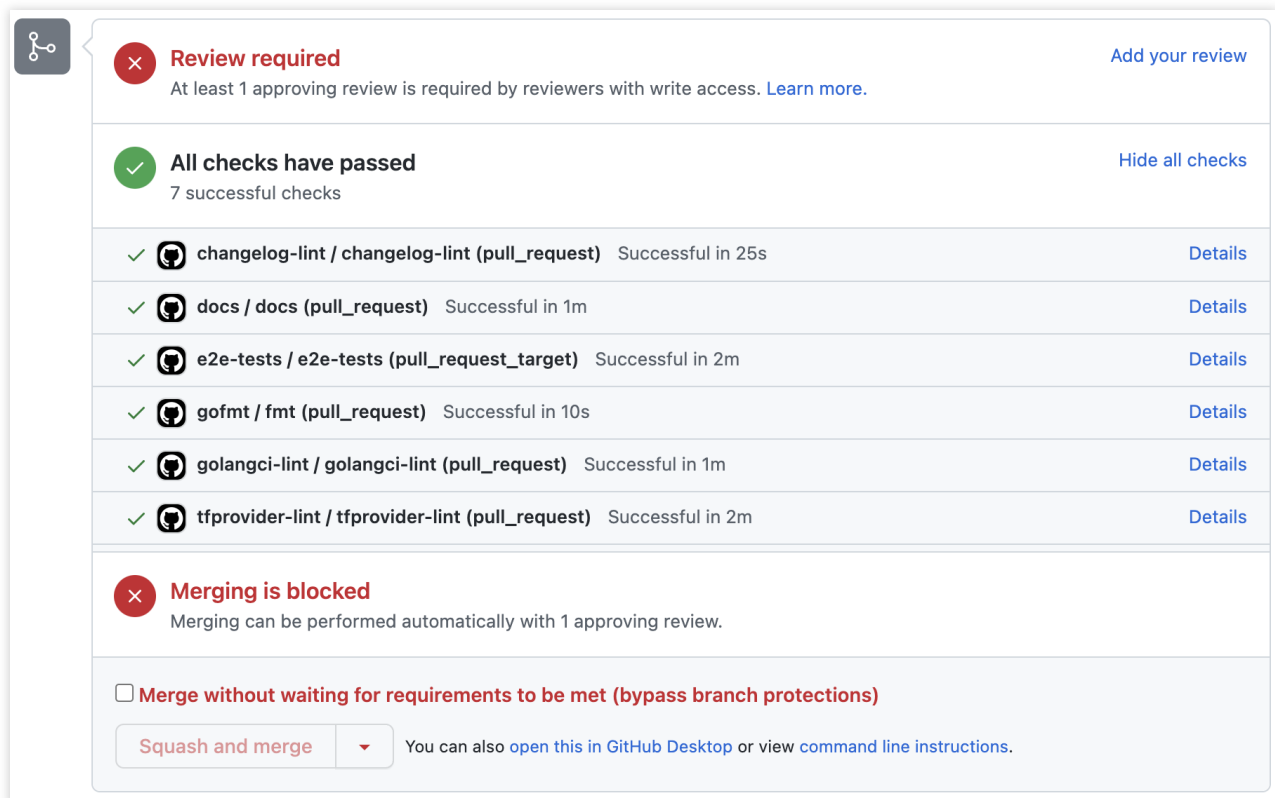
After sending the pull request, you need to submit a `.changelog/<Pull request number>.txt` file describing the request type, module, and change as follows:

```
resource/<module>: something has done
```

For more information, see [Submitting Changelog](#).

Pull request check

After the pull request is sent, basic merge checks will be performed by the action.



The screenshot displays a GitHub pull request interface. At the top, a red 'X' icon indicates 'Review required', with a sub-message: 'At least 1 approving review is required by reviewers with write access. [Learn more.](#)' and a link 'Add your review'. Below this, a green checkmark icon indicates 'All checks have passed', with a sub-message: '7 successful checks' and a link 'Hide all checks'. A list of seven checks follows, each with a green checkmark, a GitHub Actions icon, the check name, and its status: 'changelog-lint / changelog-lint (pull_request) Successful in 25s', 'docs / docs (pull_request) Successful in 1m', 'e2e-tests / e2e-tests (pull_request_target) Successful in 2m', 'gofmt / fmt (pull_request) Successful in 10s', 'golangci-lint / golangci-lint (pull_request) Successful in 1m', and 'tfprovider-lint / tfprovider-lint (pull_request) Successful in 2m'. Each check has a 'Details' link. At the bottom, a red 'X' icon indicates 'Merging is blocked', with a sub-message: 'Merging can be performed automatically with 1 approving review.' Below this is a checkbox labeled 'Merge without waiting for requirements to be met (bypass branch protections)'. A 'Squash and merge' button is visible, followed by a dropdown arrow and the text: 'You can also [open this in GitHub Desktop](#) or view [command line instructions](#).'

If your code requires an acceptance test, the code repository personnel will mark it with `run-check` to trigger execution of the test cases that can cover your changes.

Code merge

After the merge checks are passed and the repository personnel confirm that the branch can be merged, the branch will be merged into the main branch. Version release will be based on the merge. As this point, you have contributed your code.

Submitting Changelog

Last updated : 2024-12-02 16:01:33

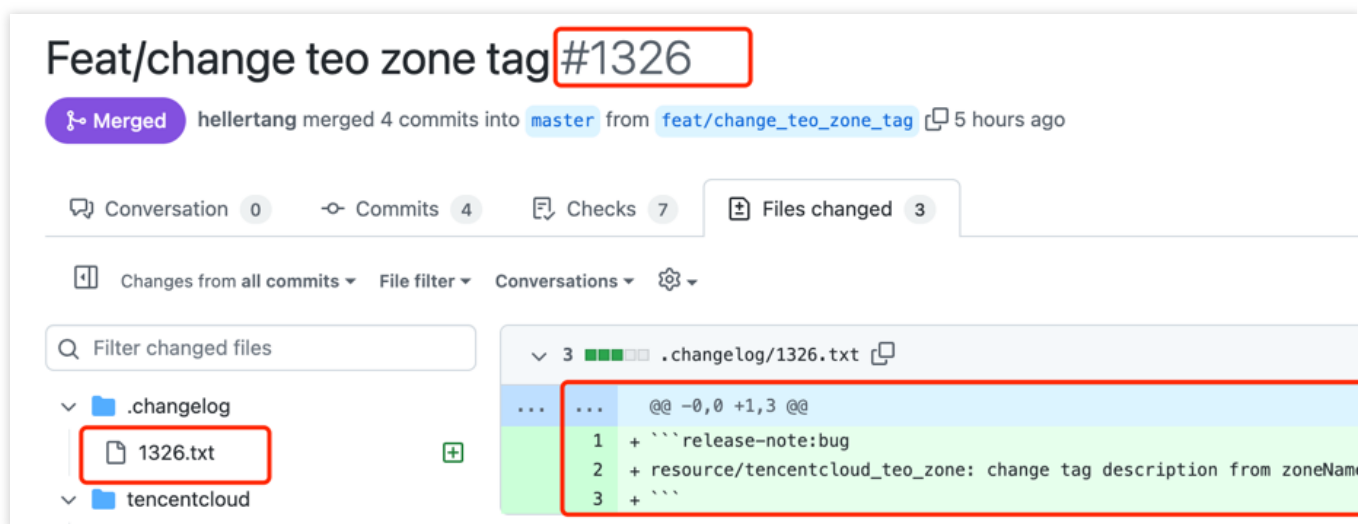
An open-source project requires maintaining a user-friendly and readable `CHANGELOG.md` so that users can quickly know whether they will be affected by the version and assess the upgrade risk if applicable.

You need to submit a `.txt` change file to the `.changelog` directory to describe the change when sending a pull request. The file will then be parsed via the [go-changelog](#) tool to generate a `CHANGELOG.md` file subject to the following update rules:

Changelog Format

Naming

The changelog involved in the current pull request should be submitted to the `.changelog` directory and named in the format of `{PR-NUMBER}.txt`. For example, if the pull request number is `1326`, the changelog should be named `.changelog/1326.txt`.



The screenshot shows a GitHub pull request titled "Feat/change teo zone tag #1326" which has been merged. The interface highlights the file `.changelog/1326.txt` in the file list and its content in the diff view. The content of the file is as follows:

```
@@ -0,0 +1,3 @@
1 + ```release-note:bug
2 + resource/tencentcloud_teo_zone: change tag description from zoneName
3 + ```
```

Format

The `changelog.txt` has the following format, where `{HEADER}` is the changelog type, and `{ENTRY}` is the changelog details.

```
```release-note:{HEADER}
{ENTRY}
```



```
```
```

If a pull request contains multiple changelog entries, you can add multiple blocks to the same changelog file as follows:

```
```release-note:bug
resource/tencentcloud_teo_zone: change tag description from zoneName to zoneId.
```

```release-note:enhancement
resource/tencentcloud_redis_instance: support update `security_groups`.
```
```

Changelog Rule Categorization

A changelog presents only changes to the code repository of a specific version affecting the operator. The following are general principles and samples of a change to be recorded in a changelog.

New resource

The changelog of a new resource should contain only the resource name and be named `release-note:new-resource`.

```
```release-note:new-resource
tencentcloud_postgresql_instance
```
```

New data source

The changelog of a new data source should contain only the data source name and be named `release-note:new-data-source`.

```
```release-note:new-data-source
tencentcloud_sqlserver_zone_config
```
```

Bug fix

The changelog of a new bug fix should be named `release-note:bug`, with a prefix indicating the corresponding resource or data source, a colon, and a summary.

Note :

Use the `provider` prefix for a provider-level fix.

```
```release-note:bug
resource/tencentcloud_cdn_domain: Fix `https_config` inconsistency after apply
```
```

Feature enhancement

The changelog of a new feature enhancement should be named `release-note:enhancement`, with a prefix indicating the corresponding resource or data source, a colon, and a summary.

Note :

Use the `provider` prefix for a provider-level feature enhancement.

```
```release-note:enhancement
resource/tencentcloud_cdn_domain: Support follow redirect and authentication
```
```

Feature disuse

The changelog of a feature disuse should be named `release-note:deprecation`, with a prefix indicating the corresponding resource or data source, a colon, and a summary.

Note :

Use the `provider` prefix for a provider-level feature disuse.

```
```release-note:deprecation
resource/tencentcloud_kubernetes_cluster: The `as_enabled` attribute is being depre
```
```

Situations Requiring No Changelog Submission

Resource and provider documentation updates

Test updates

Code refactoring

Publishing Module

Last updated : 2024-12-02 16:01:33

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 on the [Terraform registry](#). This document describes how to create and publish a Terraform TencentCloud module.

Directions

Creating a public module

Create a code repository on GitHub.

The name should be in the format of `terraform-<PROVIDER>-<NAME>` , such as [terraform-tencentcloud-vpc](#).

A basic module contains the following files:

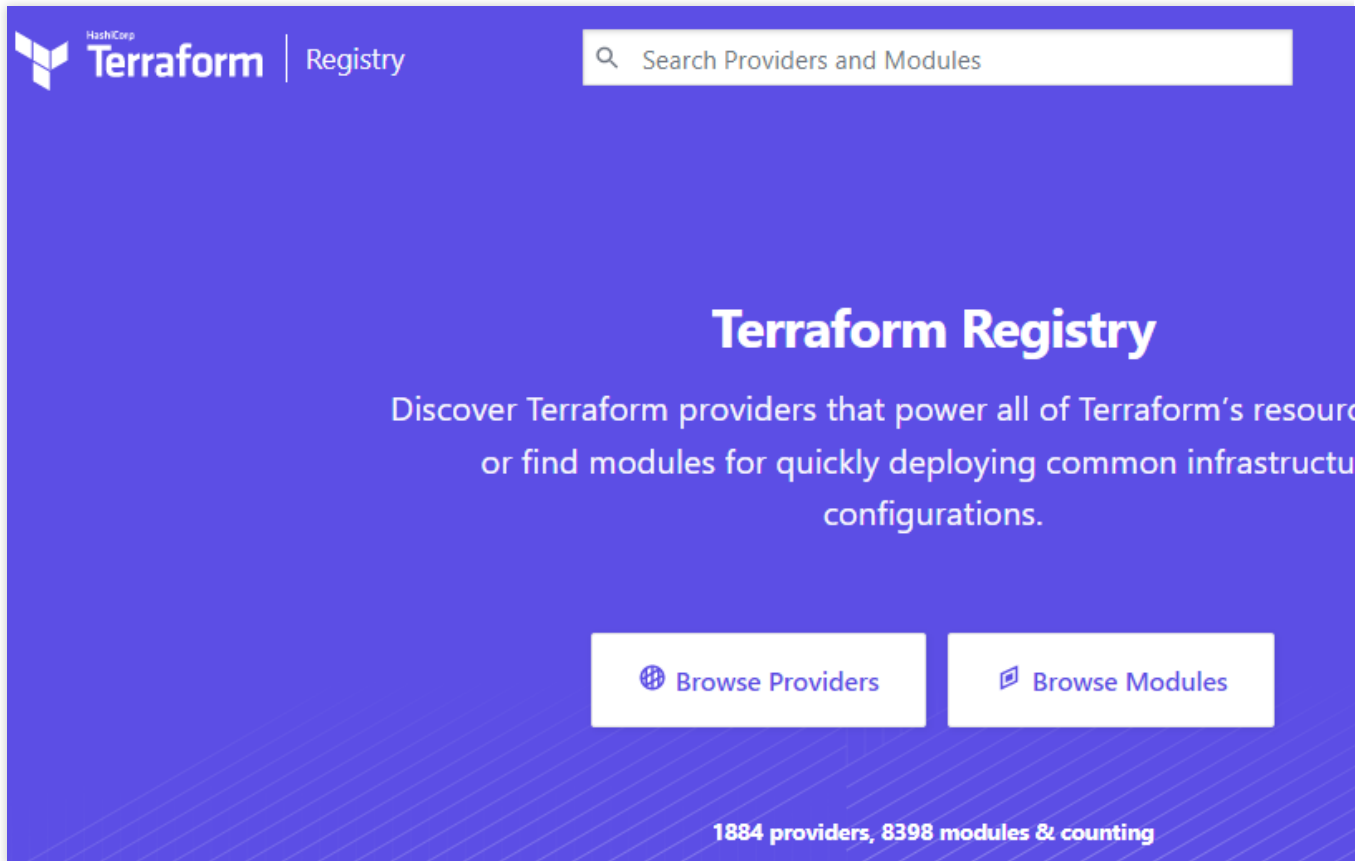
```
.
├── main.tf # Write module resources
├── variables.tf # Declare module variables
├── outputs.tf # Declare module outputs
├── LICENCE # Declare license
└── README.md # Readme
```

Note:

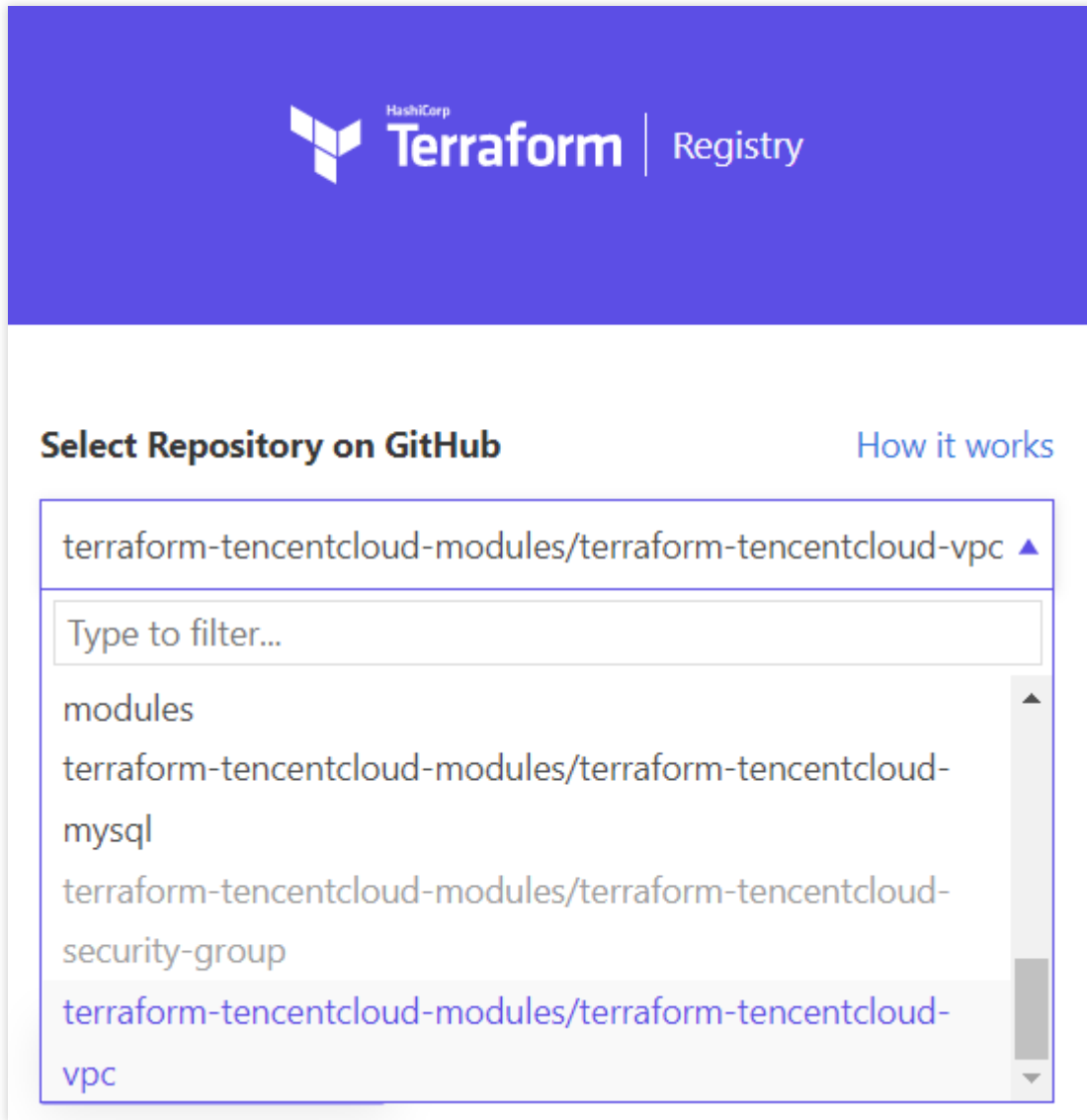
We recommend you add the `example` directory as instructed in [terraform-tencentcloud-vpc/examples](#) to store the examples for importing and using the module.

Publishing the module

1. Log in to [registry.terraform.io](#), select **Publish** in the top-right corner of the page, and click **Module** from the drop-down list as shown below:




2. Expand the **Select Repository on GitHub** drop-down list to view all the module repositories that you have management permissions for and select the target module as shown below:

**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.

3. Select **I agree to the Terms of Use.** and click **PUBLISH MODULE.**


4. The repository will be automatically synced to the Terraform registry in a few minutes as shown below:

 terraform-tencentcloud-modules / vpc

TencentCloud VPC Module for Terraform

 1.2K


tencent

 terraform-tencentcloud-modules / security-group

TencentCloud Security Group Module for Terraform

 1.1K

tencent

 terraform-tencentcloud-modules / clb ~400

tencent

(Optional) Adding repository merge check

If your module involves multi-person collaboration, you can use GitHub Actions to preliminarily check the code with merge requests.

Here, [terraform-tencentcloud-vpc](#) is used as an example. Create the `.github/workflow` directory in the root directory of the repository and create the `pull-request.yml` file. Below is the sample code:

```
name: MR_CHECK

on:
  pull_request:
    branches: [ master ]
  workflow_dispatch:
```

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: hashicorp/setup-terraform@v1
      - name: Module Files Checking
        run: |
          files=(
            LICENSE
            main.tf
            version.tf
            variables.tf
            outputs.tf
            README.md
          )

          test -d examples || echo "[WARN] Missing ./examples in modules directory,"

          for i in ${files[@]} ; do
            fileCount=$(find ./ -name $i | wc -l)
            if [[ $fileCount -gt 0 ]]; then
              echo "[INFO] File: $i exist."
            else
              echo "[ERROR] Missing $i, a recommend module should include these fil
              exit -1
            fi
          done
      - name: Terraform Validate
        run: |
          terraform init
          terraform validate

      - name: Terraform Format Check
        run: |
          terraform fmt -diff -check -recursive
```

The description is as follows:

Module Files Checking : Check whether the directory contains the file needed above.

Terraform Validate : Check the module parameters.

Terraform Format Check : Verify the Terraform code format in the module.

Developer Reference

How It Works

Last updated : 2024-12-02 16:01:33

This document describes the directory structure of Terraform TencentCloud Provider.

Directory structure

| | |
|---|--------------------------------|
| └─terraform-provider-tencentcloud | Root directory |
| └─main.go | Program entry file |
| └─AUTHORS | Author information |
| └─CHANGELOG.md | Changelog |
| └─LICENSE | License information |
| └─debug.tf.example | Example debugging configuratio |
| └─examples | Directory of example configura |
| └─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 servi | | |
| | | | | | └service_instance.go | Encapsulated CVM-instance-rela | | |
| | | | | | └service_vpc.go | Encapsulated VPC-related servi | | |
| | | | | | └... | | | |
| | | | | | └validators.go | Public argument validation fun | | |
| | | | | | └vendor | Dependent third-party librari | | |
| | | | | | └website | Web-Related files | | |
| | | | | | └tencentcloud.erb | Left sidebar file | | |
| | | | | | └docs | Source file directory of Markd | | |
| | | | | | | | └d | Data-Related documents (data_s |
| | | | | | | | └availability_zones.html.md | |
| | | | | | | | └nats.html.markdown | |
| | | | | | | | └vpc.html.markdown | |
| | | | | | | | └... | |
| | | | | | | | └index.html.markdown | |
| | | | | | | | └r | Resource-Related documents (re |
| | | | | | | | └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 storing service code, where:

`provider.go` : Plugin root describing 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.

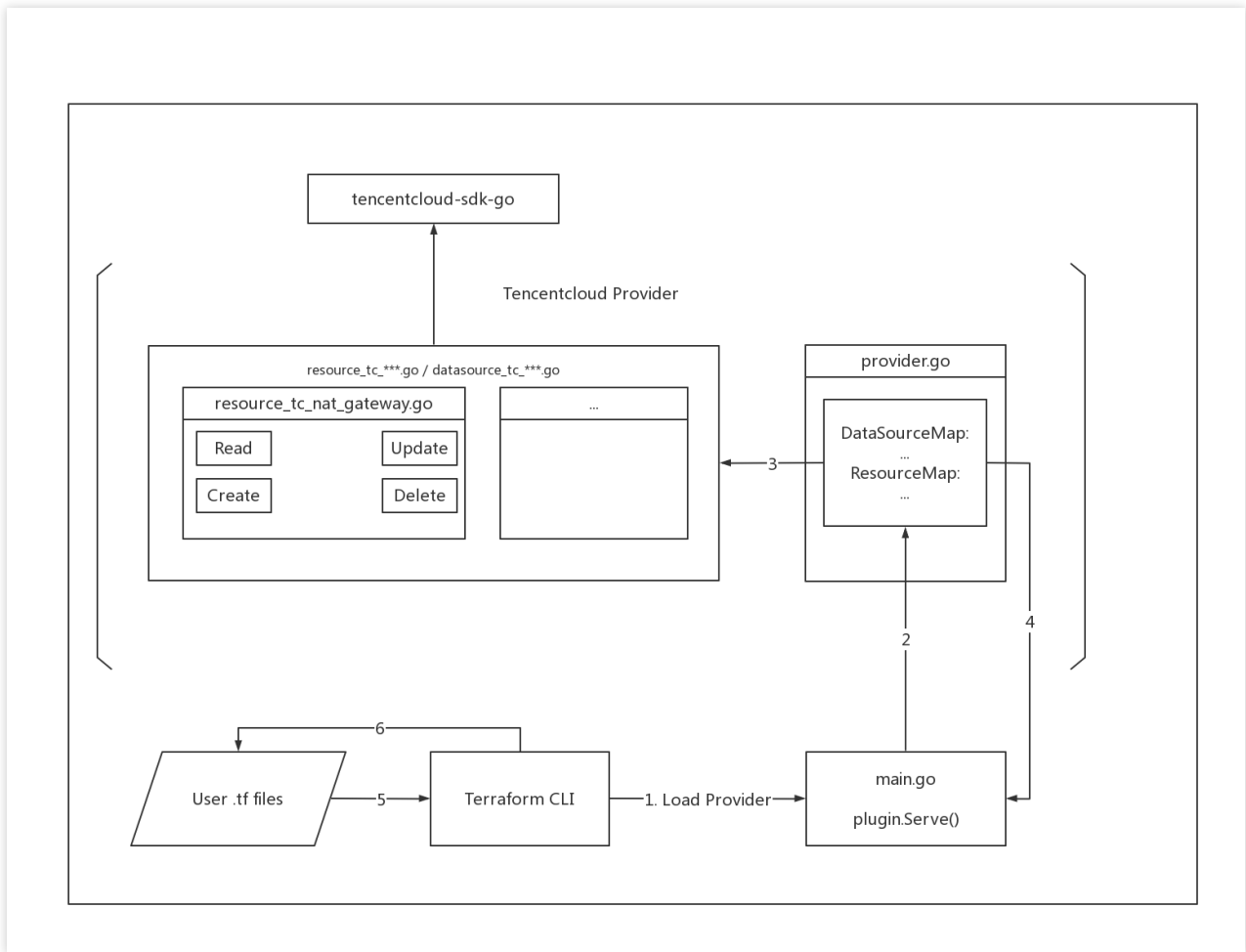
`service_*.go` : Contains some public methods under broad resource categories.

`vendor` : Contains dependent third-party libraries.

`website` : Contains documents as important as examples.

Provider lifecycle

The Terraform execution process is as shown below:



Note:

- 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 parameters 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.

Development and Debugging

Last updated : 2024-12-02 16:01:33

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 [Use Terraform in Local PC](#).

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

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

```
.
├── .githubhooks/
├── .github/
├── examples/ # Sample code. In principle, ensure that users can directly copy and
├── 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-t
  }
}
```

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,
        // 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.",
                    // Verify resource attributes (a match indicates successful cre
                    resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
                    resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
                    resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
                    resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
                ),
            },
        ),
    },
}
```

```
        // Configure the configuration content
        Config: testAccNatGatewayConfigUpdate,
        Check: resource.ComposeTestCheckFunc(
            testAccCheckTencentCloudDataSourceID("tencentcloud_nat_gateway.",
            // Verify the value of modified attributes (a match indicates s
            resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
            resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
            resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
            resource.TestCheckResourceAttr("tencentcloud_nat_gateway.my_nat
        ),
    },
},
})
}

// `testAccProviders` creates test resources based on Config before testing and ter
// This function is to check whether resources are terminated. The code logic is ea
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
    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.

Requirements and Suggestions

Last updated : 2024-12-02 16:01:33

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.

Using `ForceNew` with Caution

If the `ForceNew` field is set, **the resource will be terminated and recreated** if a change is found. If you need to set it, make sure that the local input is consistent with the remote state; otherwise, a `Diff` will be generated, leading to resource termination and recreation after creation.

Default Value of `Import`

Certain resources define default values:

```
map[string]*schema.Schema{
    "create_strategy": {
        Type: schema.TypeBool,
        Optional: true,
        Default: "foo",
        Description: "Available `foo`, `bar`.",
    },
}
```

If `create_strategy` is a write-only parameter which is passed in during creation and cannot be obtained after creation, and its value is left empty during import, `+diff` will be generated after comparison with the default value

`foo` . Therefore, you need to set it to the default value during import:

```
&schema.Resource{
  Importer: &schema.ResourceImporter{
    // `helper.ImportWithDefaultValue` is encapsulated in the provider and can be u
    State: helper.ImportWithDefaultValue (map[string]interface){}{
      "create_strategy": "foo",
    }},
},
```

Avoiding Constraining an Input Parameter

Taking a CVM instance as an example, the API documentation [Creating Instance](#) describes the input parameter for data disks as follows:

| Parameter | Required | Type | Description |
|-------------|----------|-------------------|--|
| DataDisks.N | No | Array of DataDisk | The configuration information of instance data disks. If this parameter is not specified, no data disks are purchased by default. Up to 21 data disks can be purchased, including up to one `LOCAL_BASIC` or `LOCAL_SSD` data disk and up to 20 `CLOUD_BASIC`, `CLOUD_PREMIUM`, or `CLOUD_SSD` data disks. |

The parameter indicates that up to 21 data disks are supported. The schema usually has the following constraint:

```
map[string]*schema.Schema{
  "data_disks": {
    Type: schema.TypeList,
    Optional: true,
    Description: "Data disk configuration",

    MaxItems: 21,
  },
}
```

The constraint is not recommended, as the parameter limit will be irregularly updated based on the product situation. If CVM supports more than 21 data disks after business expansion, Terraform needs to sync the update passively. Such verification delays may adversely affect developers and users. Instead of setting limits for such parameters, you can use `Description` . Abnormal inputs will be blocked by TencentCloud API, and Terraform does not require this additional layer of verification.

Nested Structure Design

TypeList block and TypeMap

TypeList items support basic types such as numbers and characters:

```
resource "foo" "bar" {
  strs = ["a", "b", "c"]
  nums = [1, 2, 3]
}
```

The complex Object and nesting are also supported:

```
resource "foo" "bar" {
  list_item {
    name = "l1" # foo.bar.list_item.0.name
  }
  list_item {
    name = "l2" # foo.bar.list_item.1.name
    sub_item {
      name = "l-2-1" # foo.bar.list_item.1.sub_item.0.name
    }
  }
}
```

Below is the sample code for declaration and acquisition:

```
var s = map[string]*schema.Schema{
  "list_item": {
    Type: schema.TypeList,
    Elem: &schema.Resource{
      Schema: map[string]*schema.Schema{
        "sub_item": {
          Type: schema.TypeList,
        },
      },
    },
  },
}

func create(d *schema.ResourceData) {
  listItems := d.Get("list_item").([]interface{})
  listItem1 := listItems[1].(map[string]interface{})
  listItem1Sub := listItem1["sub_item"].([]interface{})[0].(map[string]interface{})
}
```

`TypeMap` differs from `TypeList` in that it does not support nesting in the code schema and requires connecting parameters and braces ({} with `=` as follows:

```
resource "foo" "bar" {
  map_item = {
    key1: "1", # foo.bar.map_item.key1
    key2: "2" # foo.bar.map_item.key2
  }
}

var s = map[string]*schema.Schema{
  "map_item": {
    Type: schema.TypeMap,
    Optional: true,
    // `Elem: Map` does not support defining `Elem`. It is invalid in the SDK v1 an
  },
}

func create(d *schema.ResourceData) {
  mapItem := d.Get("map_item").(map[string]interface{})
  mapVal1 := mapItem["key1"].(string)
}
```

`TypeList` is recommended for implementing parameters of the nested type as it has more comprehensive constraints than `TypeMap`, which is more suitable for flat key-value pairs such as tags.

TypeSet

`TypeSet` is a special `TypeList`. It is used in the same way as `TypeList` but is unique and orderless.

```
resource "foo" "bar" {
  set_list = ["a", "b", "c", "c"] # It is ["a", "b", "c"] actually.
}

resource "foo" "bar_update" {
  set_list = ["c", "b", "a"] # It is equivalent to ["a", "b", "c"] and will not gen
}
```

The code schema can define the `Get` function that returns a unique index. The same index indicates the same element as follows:

```
var s = map[string]*schema.Schema{
  "set_list": {
    Type: schema.TypeList,
    Elem: &schema.Schema{Type: schema.TypeString},
    Get: func(v interface) { return getValueHash(v.(string)) },
  },
}
```

```
    },  
  }  
  
  func create(d *schema.ResourceData) {  
    setList := d.Get("set_list").(*schema.Set).List()  
    setListItemHead := listItems[0].(string)  
  }
```