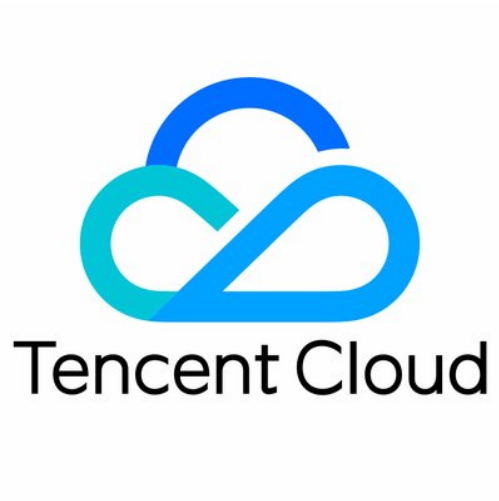
Tencent Infrastructure Automation for Terraform Best Practices

Product Documentation



Contents

Best Practices
Deploying Cloud Native Service

Last updated：2023-04-20 17:44:42

This document describes how to use Terraform to create a Tencent Cloud TKE general cluster and use the Kubernetes provider for Terraform to deploy a simple Nginx application.

Prerequisites

- Terraform is on v0.14.0 or later.
- Register at Tencent Cloud.
- Get the credentials. Create and copy `SecretId` and `SecretKey` on the Manage API Key page.
- Authorize TKE as prompted in the TKE console.

Creating TKE Resources

Create an empty directory, for example, `tf-tke-example`. Then, declare Tencent Cloud resources in the following steps.

Configuring the classic network

Create the `network.tf` file and configure the VPC, subnet, and security group as follows:

```
# Networks
variable "vpc_name" {
default = "example-vpc"
}

variable "subnet_name" {
default = "example-subnet"
}

variable "security_group_name" {
default = "example-security-group"
}
```

```
variable "network_cidr" {
default = "10.0.0.0/16"
}

variable "security_ingress_rules" {
default = [
"ACCEPT#10.0.0.0/16#ALL#ALL",
"ACCEPT#172.16.0.0/22#ALL#ALL",
"DROP#0.0.0.0/0#ALL#ALL"
]
}

resource "tencentcloud_vpc" "vpc" {
cidr_block = var.network_cidr
name = var.vpc_name
tags = var.tags
}

resource "tencentcloud_subnet" "subnet" {
availability_zone = var.available_zone
cidr_block = var.network_cidr
name = var.subnet_name
vpc_id = tencentcloud_vpc.vpc.id
tags = var.tags
}

resource "tencentcloud_security_group" "sg" {
name = var.security_group_name
description = "example security groups for kubernetes networks"
tags = var.tags
}

resource "tencentcloud_security_group_lite_rule" "sg_rules" {
security_group_id = tencentcloud_security_group.sg.id
ingress = var.security_ingress_rules
egress = [
"ACCEPT#0.0.0.0/0#ALL#ALL"
]
}
```

Configuring the cluster

Create the `cluster.tf` file and configure the TKE cluster as follows:

```
# TKE
variable "cluster_name" {
default = "example-cluster"
}

variable "cluster_version" {
default = "1.22.5"
}

variable "cluster_cidr" {
default = "172.16.0.0/22"
}

variable "cluster_os" {
default = "tlinux2.2(tkernel3)x86_64"
}

variable "cluster_public_access" {
default = true
}

variable "cluster_private_access" {
default = true
}

variable "worker_count" {
default = 1
}

variable "worker_instance_type" {
default = "S5.MEDIUM2"
}

variable "available_zone" {
default = "ap-guangzhou-3"
}

variable "tags" {
default = {
terraform = "example"
}
}

resource "random_password" "worker_pwd" {
length = 12
min_numeric = 1
```

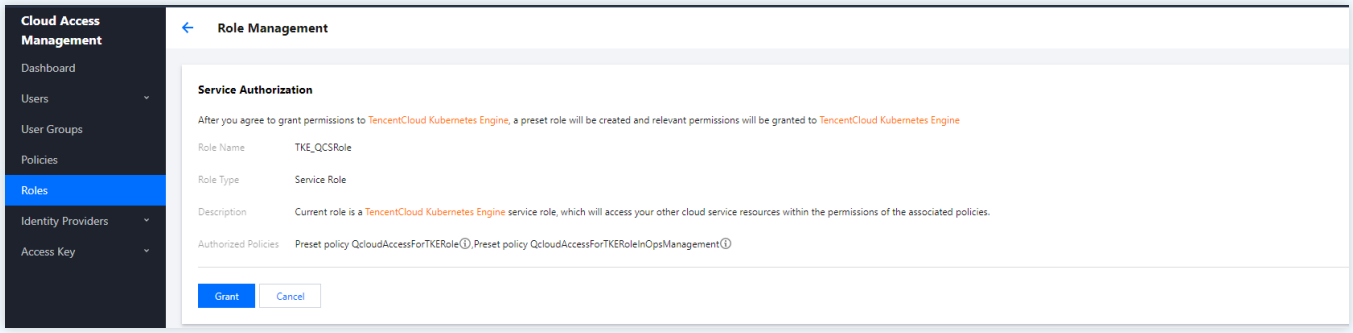
```
min_special = 1
min_upper = 1
override_special = "!#$%&*()-_=+[]{}<>:?"
}

resource "tencentcloud_kubernetes_cluster" "cluster" {
cluster_name = var.cluster_name
cluster_version = var.cluster_version
cluster_cidr = var.cluster_cidr
cluster_os = var.cluster_os
cluster_internet = var.cluster_public_access
cluster_internet_security_group = var.cluster_public_access ? tencentcloud_securi
ty_group.sg.id : null
cluster_intranet = var.cluster_private_access
cluster_intranet_subnet_id = var.cluster_private_access ? tencentcloud_subnet.sub
net.id : null
vpc_id = tencentcloud_vpc.vpc.id

worker_config {
availability_zone = var.available_zone
count = var.worker_count
instance_type = var.worker_instance_type
subnet_id = tencentcloud_subnet.subnet.id
security_group_ids = [tencentcloud_security_group.sg.id]
password = random_password.worker_pwd.result
}

tags = var.tags
}
```

(Optional) Configuring the CAM role

TKE requires the access to other resources. You need to create the `TKE_QCSRole` role and the `TF_QcloudAccessForTKERole` and `TF_QcloudAccessForTKERoleInOpsManagement` preset policies.

Note：

You don't need to create the file if you have completed the authorization in the console as shown below:



Create the `cam.tf` file, configure the CAM role, and associate the policy as follows:

```
resource "tencentcloud_cam_role" "TKE_QCSRole" {
name = "TKE_QCSRole"
document = <<EOF
{
"statement": [
{
"action":"name/sts:AssumeRole",
"effect":"allow",
"principal":{
"service":"ccs.qcloud.com"
}
}
],
"version":"2.0"
}
EOF
description = "The TKE service role."
}

data "tencentcloud_cam_policies" "ops_mgr" {
name = "QcloudAccessForTKERoleInOpsManagement"
}

data "tencentcloud_cam_policies" "qca" {
name = "QcloudAccessForTKERole"
}

locals {
ops_policy_id = data.tencentcloud_cam_policies.ops_mgr.policy_list.0.policy_id
qca_policy_id = data.tencentcloud_cam_policies.qca.policy_list.0.policy_id
}
```

```
resource "tencentcloud_cam_role_policy_attachment" "QCS_OpsMgr" {
role_id = lookup(tencentcloud_cam_role.TKE_QCSRole, "id")
policy_id = local.ops_policy_id
}

resource "tencentcloud_cam_role_policy_attachment" "QCS_QCA" {
role_id = lookup(tencentcloud_cam_role.TKE_QCSRole, "id")
policy_id = local.qca_policy_id
}
```

(Optional) Encapsulating into a module

You can encapsulate these .tf files into a module so that you can focus less on the internal implementation. Or you can directly refer to the existing module terraform-tencentcloud-tke. You can submit issues and pull requests.

Configuring Kubernetes

The above configurations are enough to create a basic TKE managed cluster. The following describes how to use Kubernetes and TKE to deploy a simple Nginx application.

Configuring the K8s provider

Get the public network address, CA certificate, and user credentials of the cluster from the above .tf files.
Taking the above module as an example, enter the server and credentials of the cluster in the Kubernetes provider as follows:

```
terraform {
required_providers {
kubernetes = {
source = "hashicorp/kubernetes"
version = ">= 2.0.0"
}
tencentcloud = {
source = "tencentcloudstack/tencentcloud"
version = ">=1.77.7"
}
}
}

provider "tencentcloud" {
region = "ap-hongkong"
}

module "tencentcloud_tke" {
```

```
source = "github.com/terraform-tencentcloud-modules/terraform-tencentcloud-tke"
available_zone = "ap-hongkong-3" # Available zone must belongs to the region.
}

provider "kubernetes" {
host = module.tencentcloud_tke.cluster_endpoint
cluster_ca_certificate = module.tencentcloud_tke.cluster_ca_certificate
client_key = base64decode(module.tencentcloud_tke.client_key)
client_certificate = base64decode(module.tencentcloud_tke.client_certificate)
}
```

Configuring the public network access of the security group

We recommend you not open all public IP ranges. By default, the security group opens only `10.0.0.0/16` and `172.16.0.0/22`. To test the public network access of the cluster, add rules to open the target IP range.

Modify the above `module` block by passing in the specified rule as follows:

```
module "tencentcloud_tke" {
source = "../../"
available_zone = var.available_zone # Available zone must belongs to the region.
create_cam_strategy = false
security_ingress_rules = [
"ACCEPT#10.0.0.0/16#ALL#ALL",
"ACCEPT#172.16.0.0/22#ALL#ALL",
"ACCEPT#(Your IP address without the parentheses `()`)#ALL#ALL",
"DROP#0.0.0.0/0#ALL#ALL"
]
}
```

Configuring resources

In Terraform, declare the namespace, Deployment, and Service via HCL instead of YAML as follows:

```
resource "kubernetes_namespace" "test" {
metadata {
name = "nginx"
}
}

resource "kubernetes_deployment" "test" {
metadata {
name = "nginx"
namespace = kubernetes_namespace.test.metadata.0.name
}
```

```
spec {
replicas = 2
selector {
match_labels = {
app = "MyTestApp"
}
}
template {
metadata {
labels = {
app = "MyTestApp"
}
}
spec {
container {
image = "nginx"
name = "nginx-container"
port {
container_port = 80
}
}
}
}
}
}

resource "kubernetes_service" "test" {
metadata {
name = "nginx"
namespace = kubernetes_namespace.test.metadata.0.name
}
spec {
selector = {
app = kubernetes_deployment.test.spec.0.template.0.metadata.0.labels.app
}
type = "NodePort"
port {
node_port = 30201
port = 80
target_port = 80
}
}
}
```

Configuring an Ingress

The following describes how to configure an Ingress to associate a CLB instance to implement public network access. First, create a CLB instance as follows:

```
locals {
lb_vpc = module.tencentcloud_tke.vpc_id
lb_sg = module.tencentcloud_tke.security_group_id
}

resource "tencentcloud_clb_instance" "ingress-lb" {
address_ip_version = "ipv4"
clb_name = "example-lb"
internet_bandwidth_max_out = 1
internet_charge_type = "BANDWIDTH_POSTPAID_BY_HOUR"
load_balancer_pass_to_target = true
network_type = "OPEN"
security_groups = [local.lb_sg]
vpc_id = local.lb_vpc
}
```

Configure the Ingress and specify the ID of the created CLB instance as follows:

```
resource "kubernetes_ingress_v1" "test" {
metadata {
name = "test-ingress"
namespace = "nginx"
annotations = {
"ingress.cloud.tencent.com/direct-access" = "false"
"kubernetes.io/ingress.class" = "qcloud"
"kubernetes.io/ingress.existLbId" = tencentcloud_clb_instance.ingress-lb.id
"kubernetes.io/ingress.extensiveParameters" = "{\"AddressIPVersion\": \"IPV4\"}"
"kubernetes.io/ingress.http-rules" = "[{\"path\":\"/\",\"backend\":{\"serviceName
\":\"nginx\",\"servicePort\":\"80\"}}]"
"kubernetes.io/ingress.https-rules" = "null"
"kubernetes.io/ingress.qcloud-loadbalance-id" = tencentcloud_clb_instance.ingress
-lb.id
"kubernetes.io/ingress.rule-mix" = "false"
}
}
spec {
rule {
http {
path {
backend {
service {
name = kubernetes_service.test.metadata.0.name
port {
```

```
number = 80
}
}
}
path = "/"
}
}
}
}
}
```

To get the CLB IP, create the `output` variable as follows:

```
output "load_balancer_ip" {
value = kubernetes_ingress_v1.test.status.0.load_balancer.0.ingress.0.ip
}
```

Executing Creation

Run the following commands in sequence after writing all the .tf files:

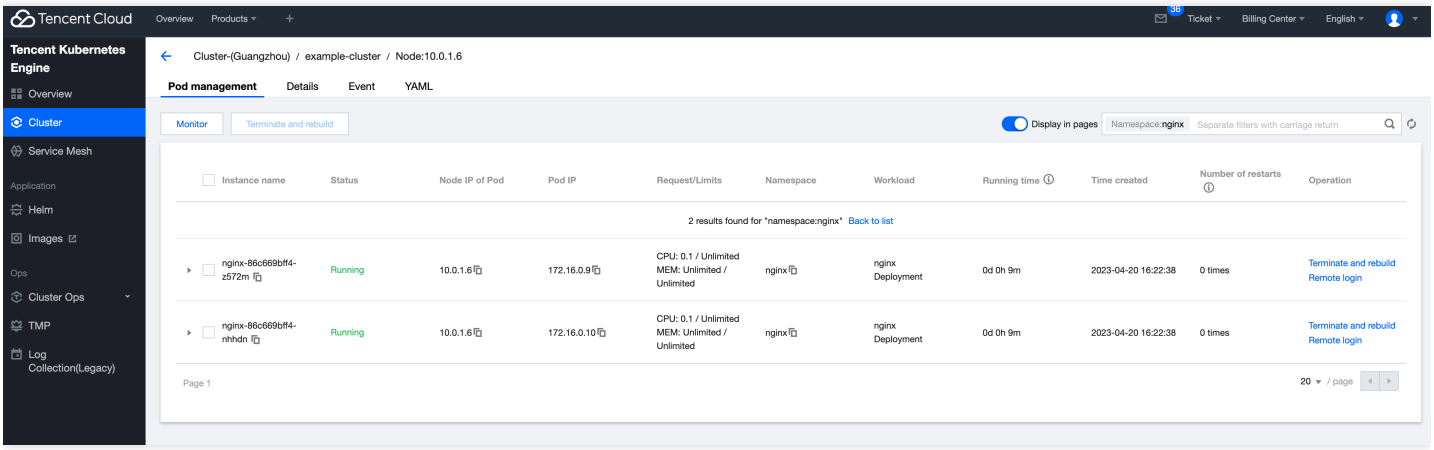
```
$ terraform init
$ terraform plan
$ terraform apply
```

After the successful creation, the console will output the above `output` information:

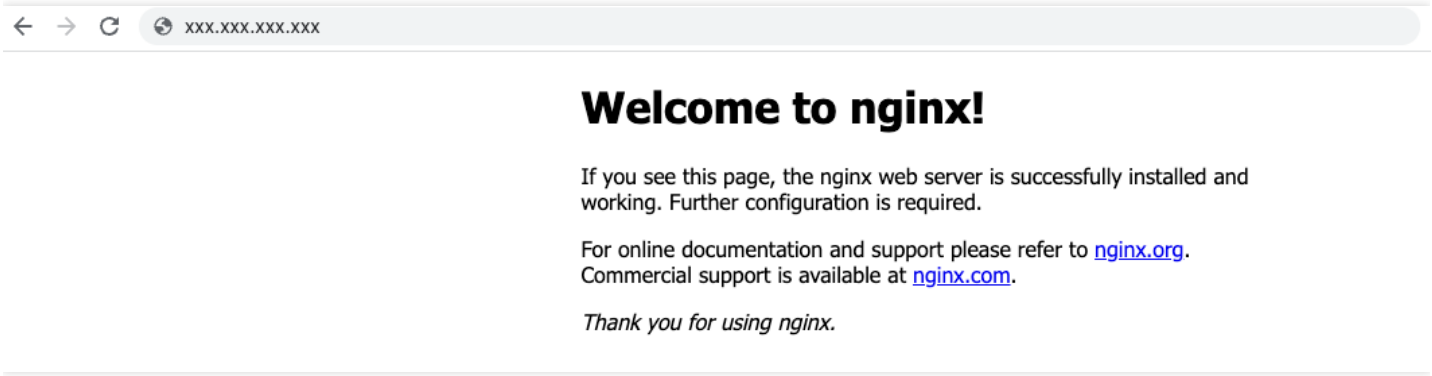
```
Apply complete! Resources: 16 added, 0 changed, 0 destroyed.

Outputs:

load_balancer_ip = "xxx.xxx.xxx.xxx"
```

Verifying Deployment

Log in to the Tencent Cloud console and go to TKE > **example-cluster**. You can see that Nginx Pods are **Running**:



Access the address of `load_balancer_ip`. If the page displays "Welcome to nginx!", the application is deployed successfully.



Cross-Region Replication of Resources

Last updated：2023-05-29 16:08:53

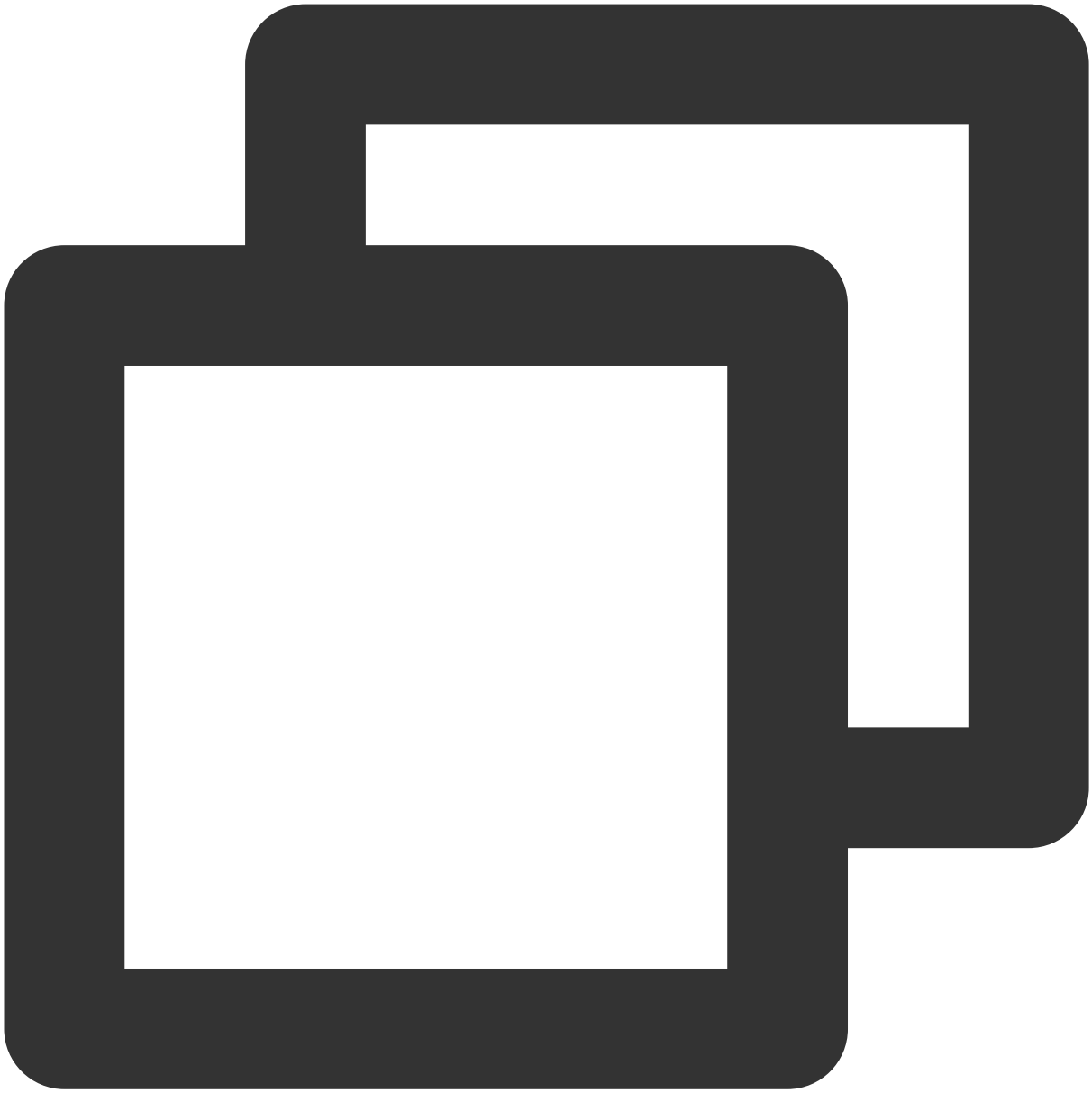
This document describes how to import an existing resource to Terraform and create a resource through file copying for cross-region replication.

Importing an Existing Resource to Terraform

Most users new to Terraform may want to import existing cloud resources to Terraform for management. Terraform allows for importing a single resource. To import multiple resources and instances, open-source tools are required. The following describes the import methods in the two scenarios.

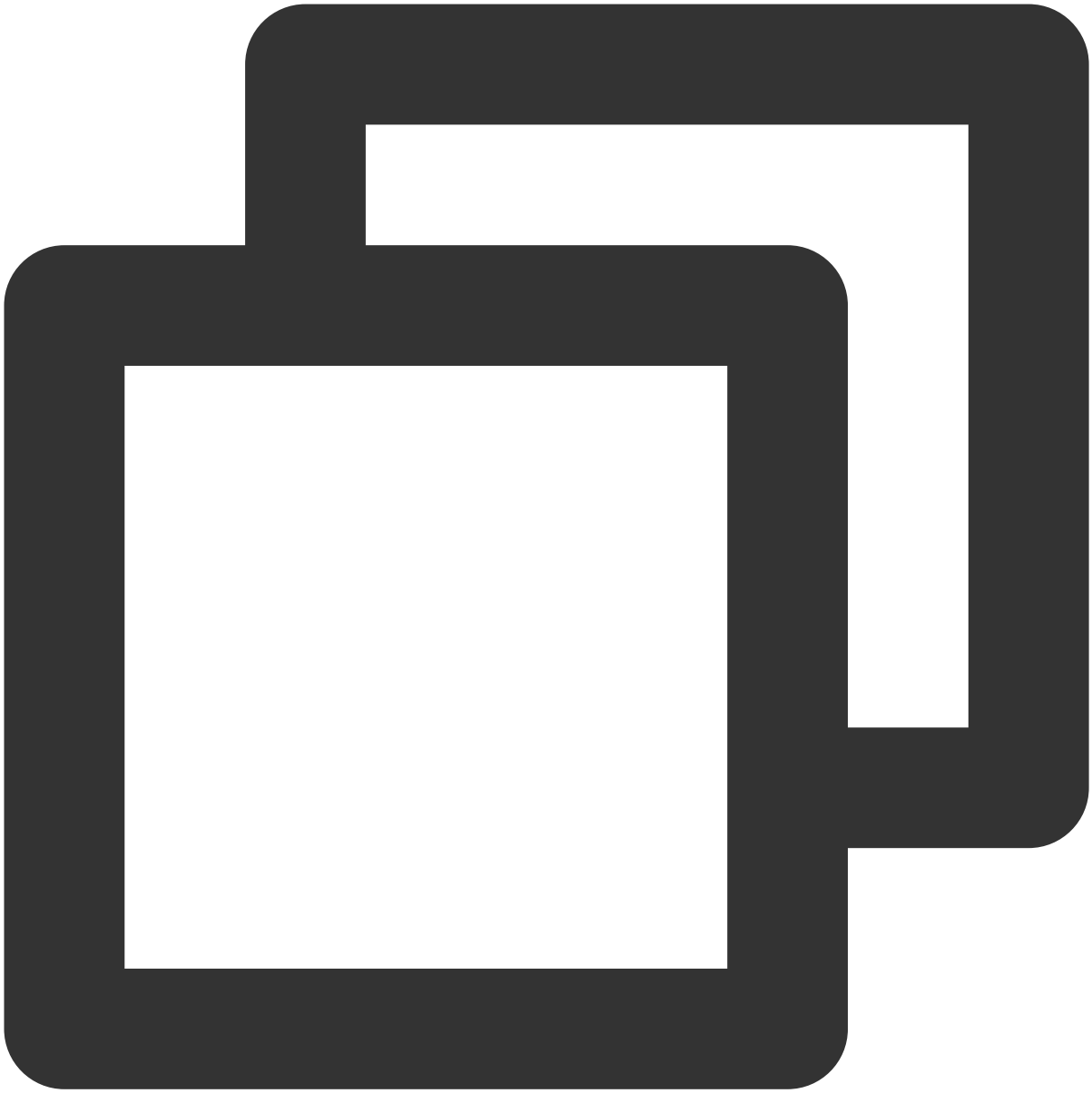
Importing a single resource

You can run the `Import` command in Terraform to import a single resource in the format of `terraform import [Resource type].[Name] [Input parameter]`. The name is custom, and the input parameter is a required string for resource query (which is an ID in most cases and a name or multi-field combination for certain resources). Taking the CVM instance as an example, the import command indicated in Import is as follows:



```
$ terraform import tencentcloud_instance.ins ins-jvu2hiw2 -allow-missing-config
```

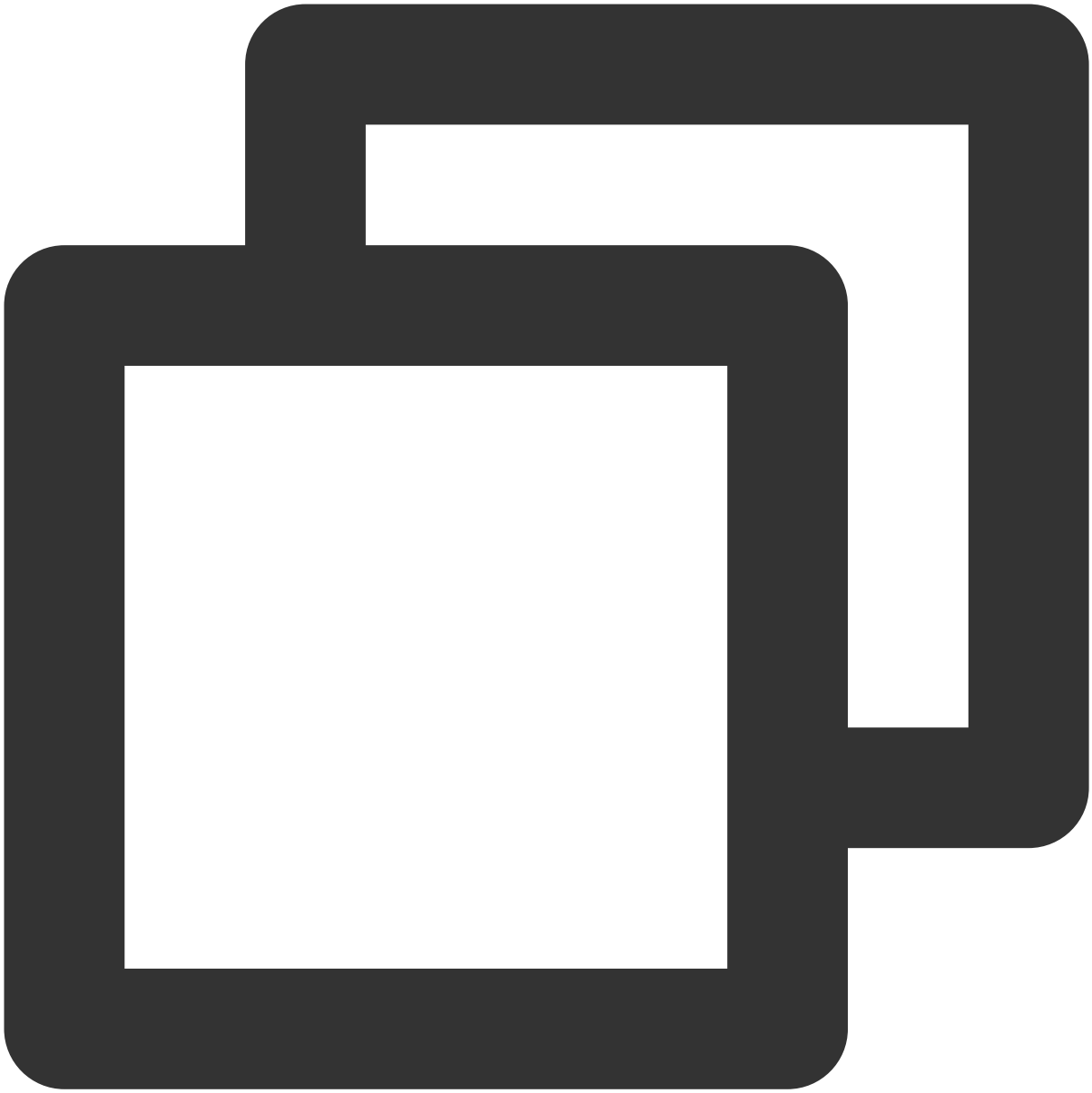
Here, `-allow-missing-config` indicates not to require a pre-declared block locally; otherwise, you need to pre-write a `resource [Resource type].[Name] {}` block in the file. After the import, fields **will not** be written into the .tf file, and you need to run `terraform show` to view the code of the imported resource:



```
# tencentcloud_instance.ins:
resource "tencentcloud_instance" "ins" {
    allocate_public_ip         = true
    availability_zone          = "ap-guangzhou-3"
    create_time                = "2022-01-01T01:11:11Z"
    id                         = "ins-xxxxxxxx"
    image_id                   = "img-xxxxxxxx"
    instance_charge_type       = "POSTPAID_BY_HOUR"
    instance_name              = "xxxxxxxx"
    instance_status            = "RUNNING"
    instance_type              = "S3.MEDIUM2"
```

```
    internet_charge_type       = "TRAFFIC_POSTPAID_BY_HOUR"
    internet_max_bandwidth_out = 1
    key_name                   = "skey-xxxxxxxx"
    private_ip                 = "10.0.1.1"
    project_id                 = 0
    public_ip                  = "1.1.1.1"
    running_flag               = true
    security_groups            = [
        "sg-xxxxxxxx",
    ]
    subnet_id                  = "subnet-xxxxxxxx"
    system_disk_id             = "disk-xxxxxxxx"
    system_disk_size           = 50
    system_disk_type           = "CLOUD_PREMIUM"
    tags                       = {}
    vpc_id                     = "vpc-xxxxxxxx"
}
```

Write the code into your .tf file and remove read-only fields (`id` , `create_time` , and `public_ip` as indicated in Attributes Reference) before the import.



```
resource "tencentcloud_instance" "ins" {
    allocate_public_ip         = true
    availability_zone          = "ap-guangzhou-3"
#    create_time                = "2022-01-01T01:11:11Z"
#    id                         = "ins-xxxxxxxx"
    image_id                   = "img-xxxxxxxx"
    instance_charge_type       = "POSTPAID_BY_HOUR"
    instance_name              = "xxxxxxxx"
#    instance_status            = "RUNNING"
    instance_type              = "S3.MEDIUM2"
    internet_charge_type       = "TRAFFIC_POSTPAID_BY_HOUR"
```

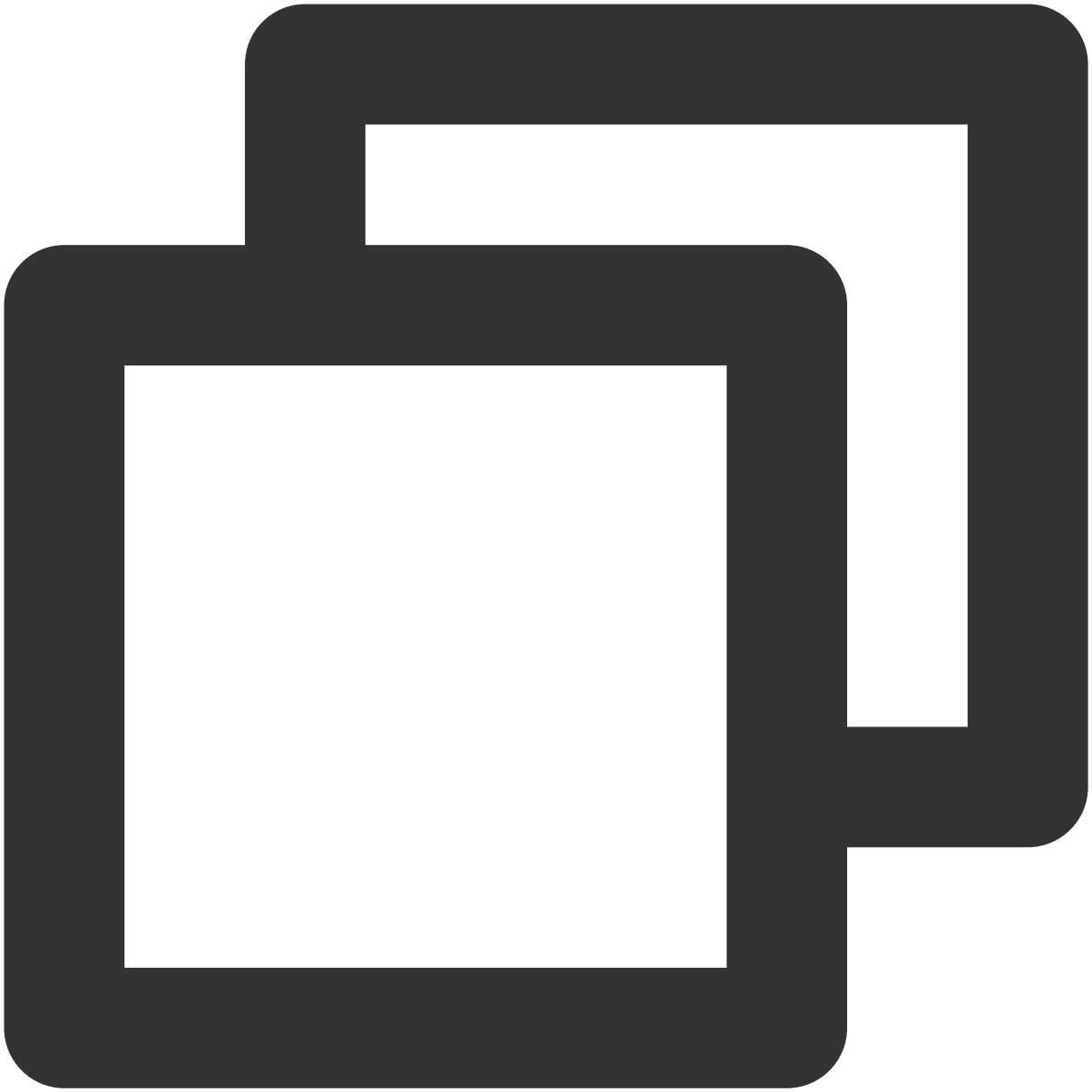
```
    internet_max_bandwidth_out = 1
    key_name                   = "skey-xxxxxxxx"
    private_ip                 = "10.0.1.1"
    project_id                 = 0
#    public_ip                  = "1.1.1.1"
    running_flag               = true
    security_groups            = [
        "sg-xxxxxxxx",
    ]
    subnet_id                  = "subnet-xxxxxxxx"
    system_disk_id             = "disk-xxxxxxxx"
    system_disk_size           = 50
    system_disk_type           = "CLOUD_PREMIUM"
    tags                       = {}
    vpc_id                     = "vpc-xxxxxxxx"
}
```

Get the `Import` command and read-only fields of each resource here. If they are not found, the resource cannot be imported currently.

Batch importing resources via Terraformer

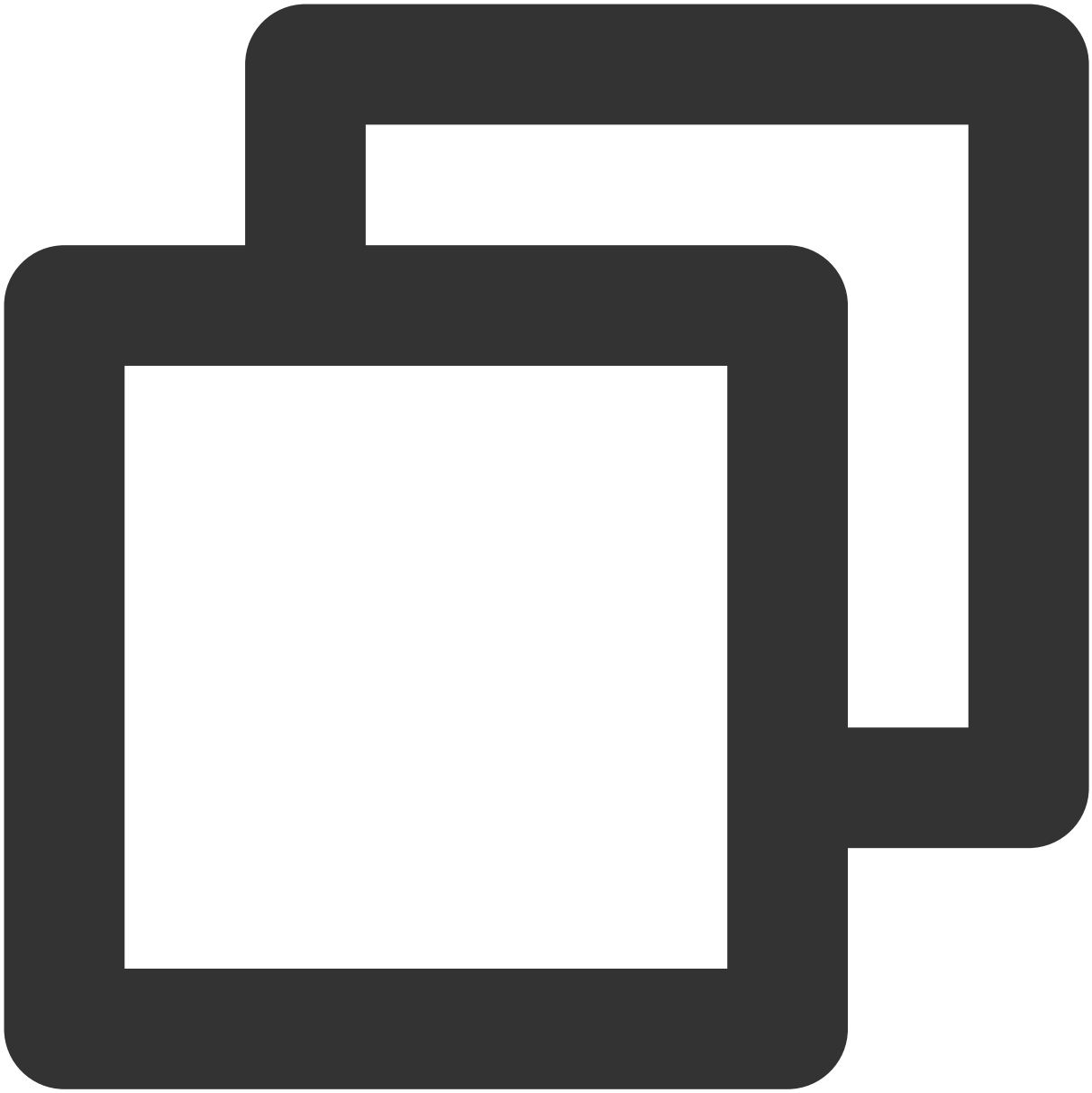
The above method applies only when the number of resources is small and will become very troublesome for batch import. In this case, you can use Terraformer, a command line tool of Google Cloud Platform that can tag and import most cloud resources under your account as .tf files.

1. Install.



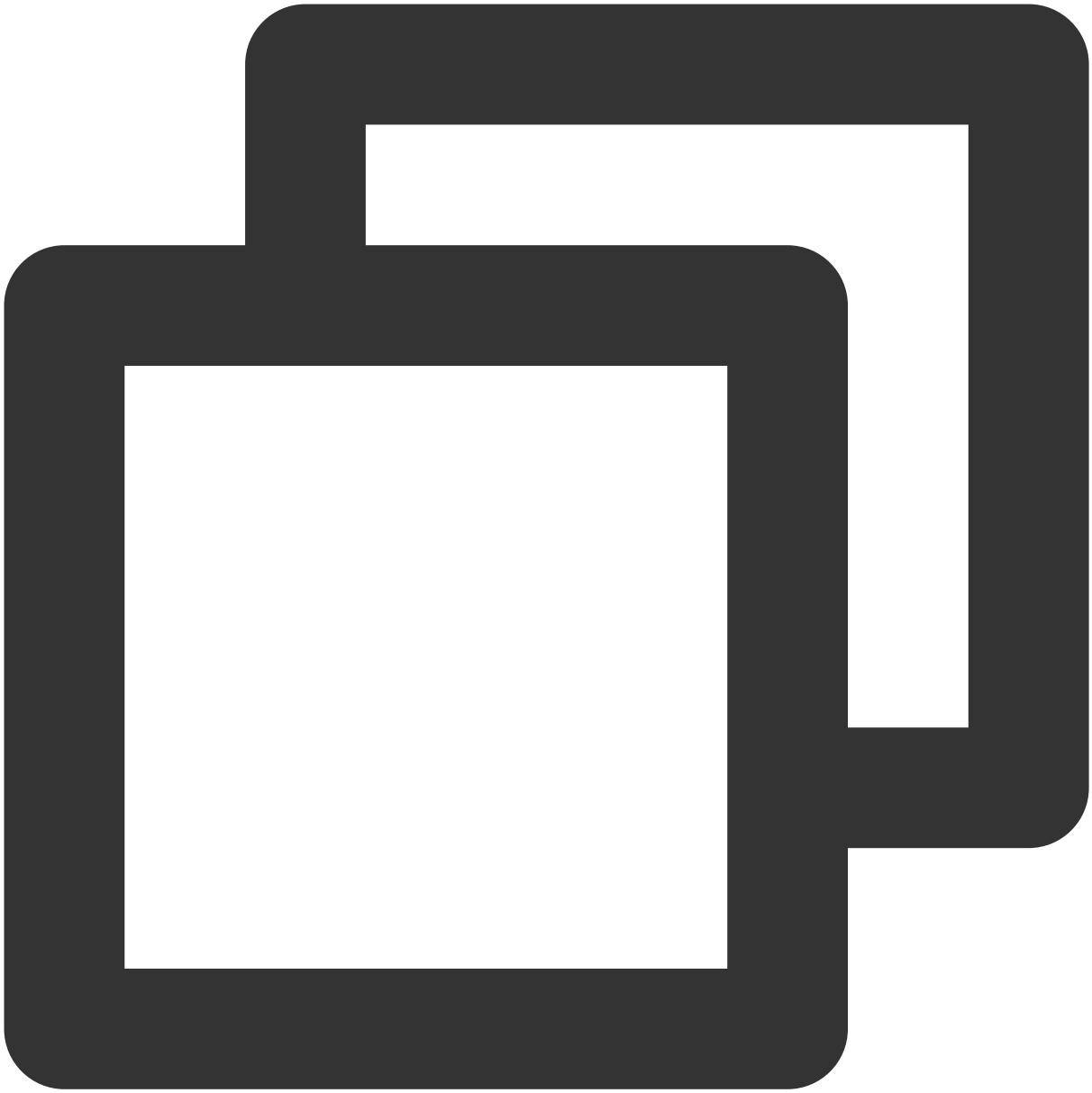
```
$ brew install terraformer
```

2. Run the import command. To import all CVM and VPC resources in Guangzhou region, the command format will be as follows:



```
terraformer import tencentcloud --resources="vpc,cvm" --regions=ap-guangzhou
```

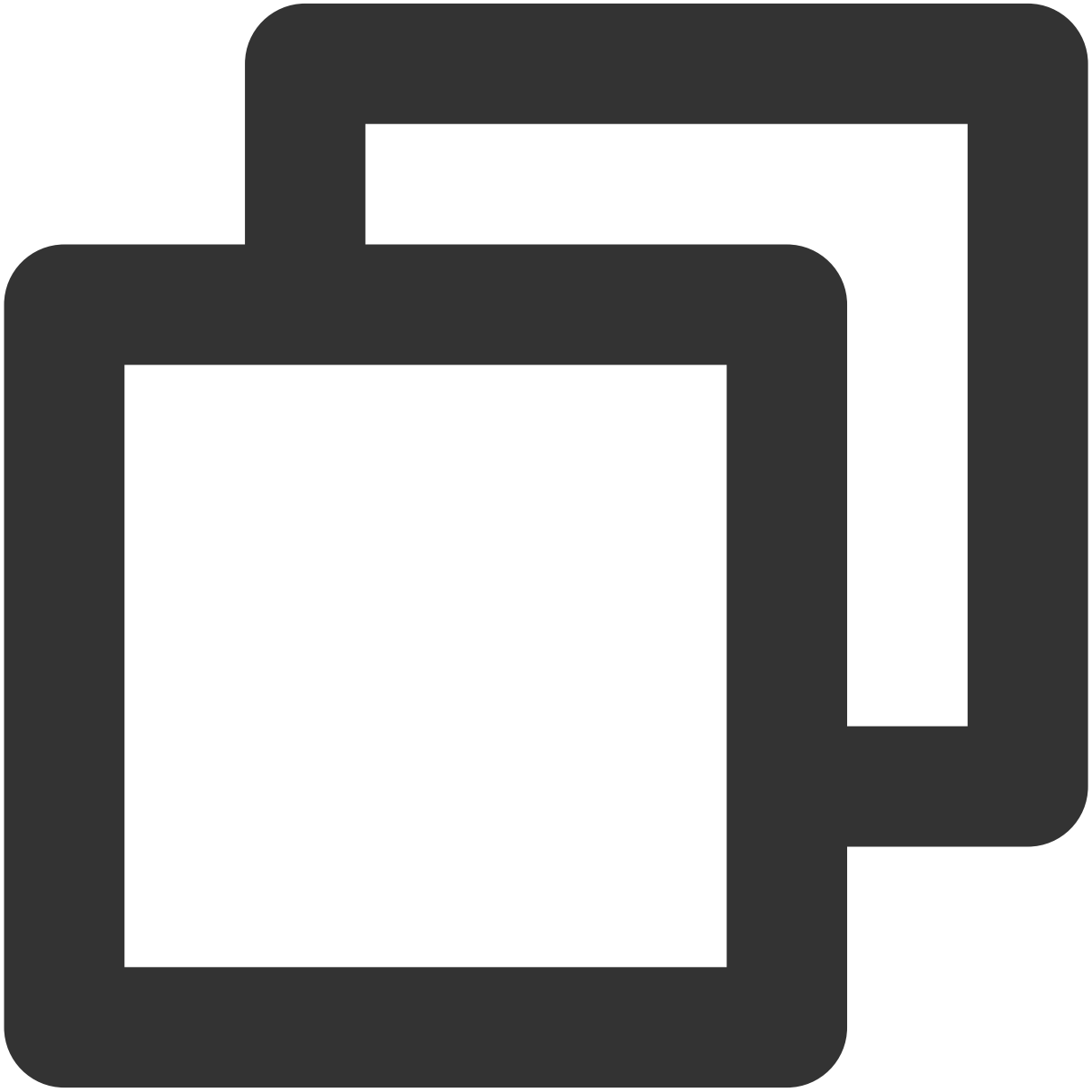
After the command is run, Terraform will write the imported resource files into the `./generated` directory by default as shown below:



```
.
└── tencentcloud
    ├── cvm
    │   └── ap-guangzhou
    │       ├── instance.tf
    │       ├── key_pair.tf
    │       ├── outputs.tf
    │       ├── provider.tf
    │       ├── terraform.tfstate
    │       └── variables.tf
    └── vpc
```

```
        └── ap-guangzhou
            ├── outputs.tf
            ├── provider.tf
            ├── terraform.tfstate
            └── vpc.tf
```

3. **Change the source**: TencentCloud Provider is maintained by Tencent Cloud but not Terraform. Therefore, you need to add the `source` field to the generated `provider.tf` file, with the value of `tencentcloudstack/tencentcloud`.



```
provider "tencentcloud" {
```

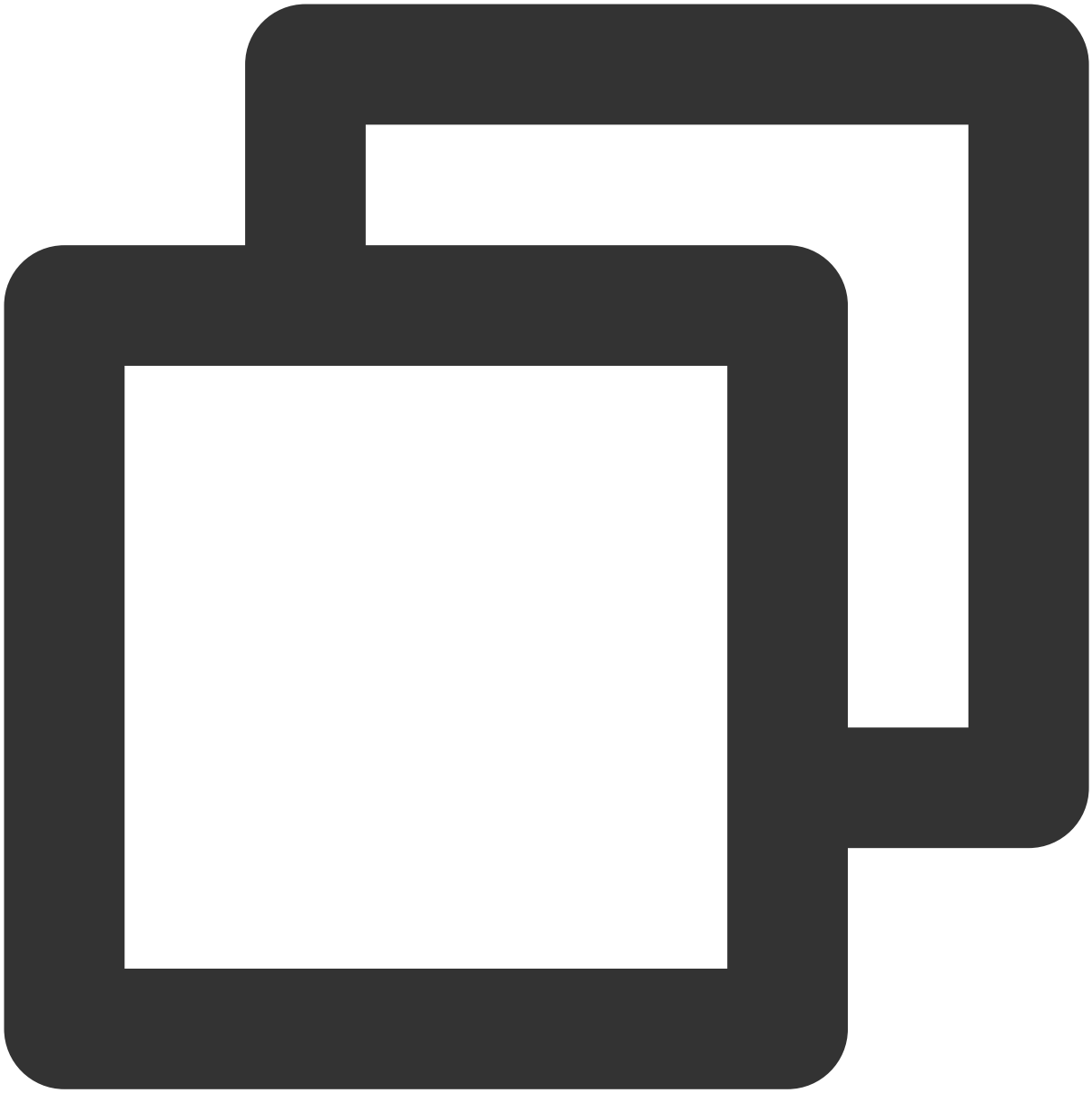
```
  version = "~> 1.77.11"
}

terraform {
  required_providers {
    tencentcloud = {
      source  = "tencentcloudstack/tencentcloud" # Add `source` to specify the name
      version = "~> 1.77.11"
    }
  }
}
```

Not all Tencent Cloud resources can be imported via Terraformer. For more information, see terraformer/providers/tencentcloud.

Cross-Region Replication

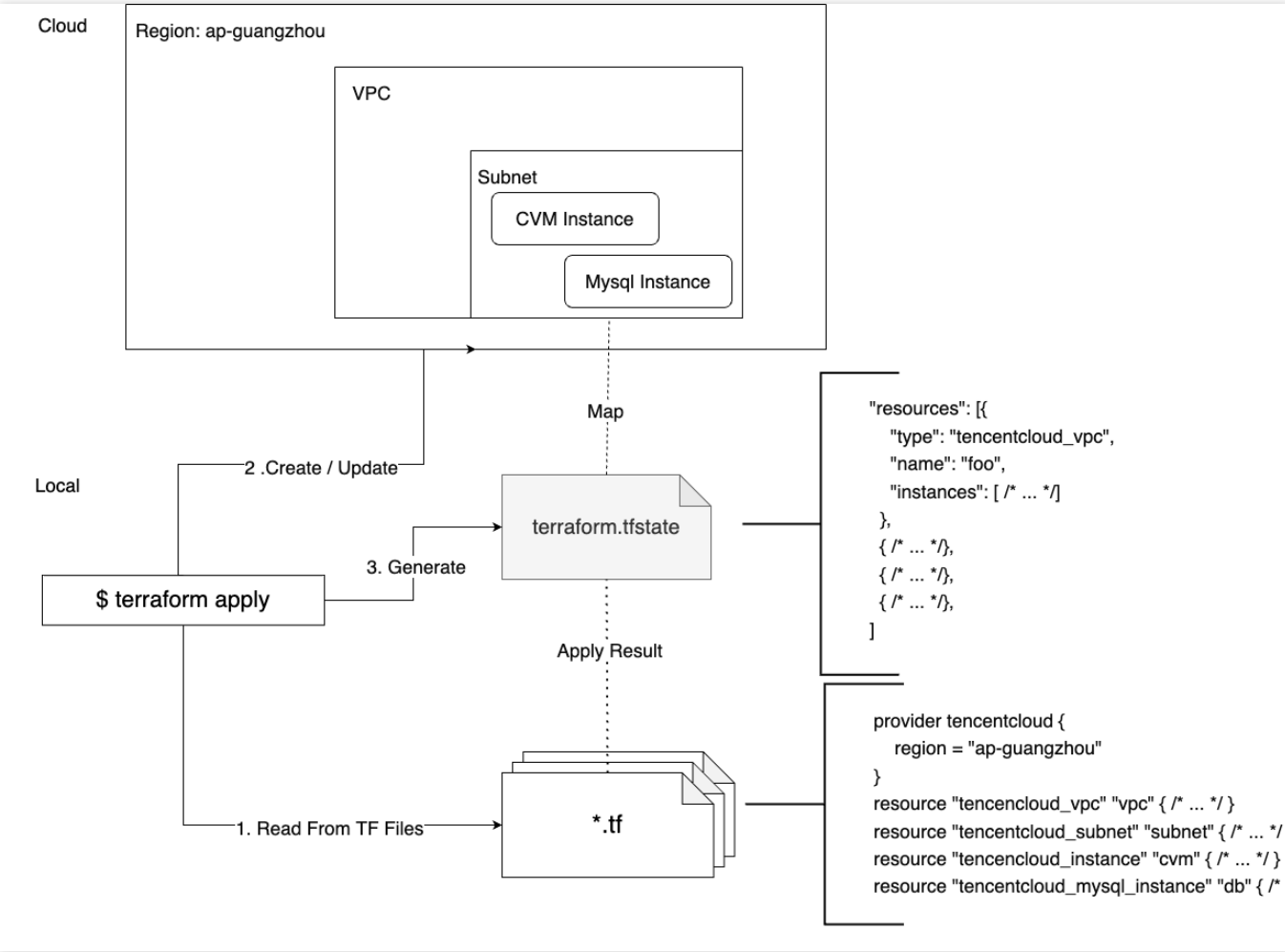
How it works

Below is a simple Terraform working directory structure:



```
.
├── .terraform
│   └── providers # Referenced providers
├── .terraform.lock.hcl # Provider lock version
├── main.tf             # tf. file
├── vars.tf             # tf. file
├── outputs.tf          # tf. file
└── terraform.tfstate   # State file
```

Local working directories are mapped to Tencent Cloud resources as shown below:

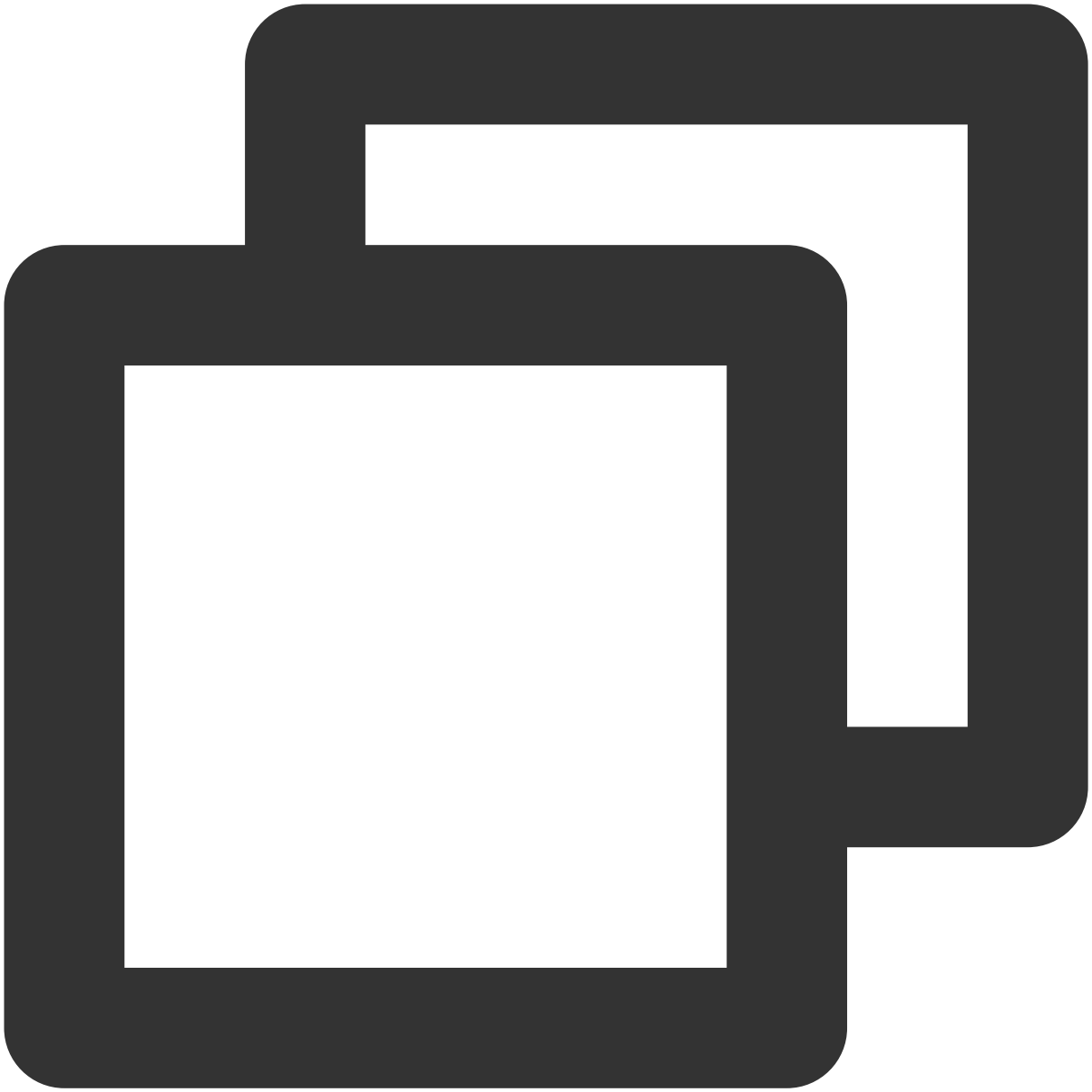


After you run the `terraform apply` command and complete the deployment, the `terraform.tfstate` file will be generated. It is a JSON file stored locally by default or configured in a remote bucket (you need to configure the backend) to describe the mapping between resources declared by Terraform and real cloud resources. If `terraform.tfstate` does not exist in the local directory or backend, or no cloud resource data is written into it, Terraform will consider that no resources are deployed and run `apply` for resource creation.

Sample: Cross-region replication in TKE Serverless clusters

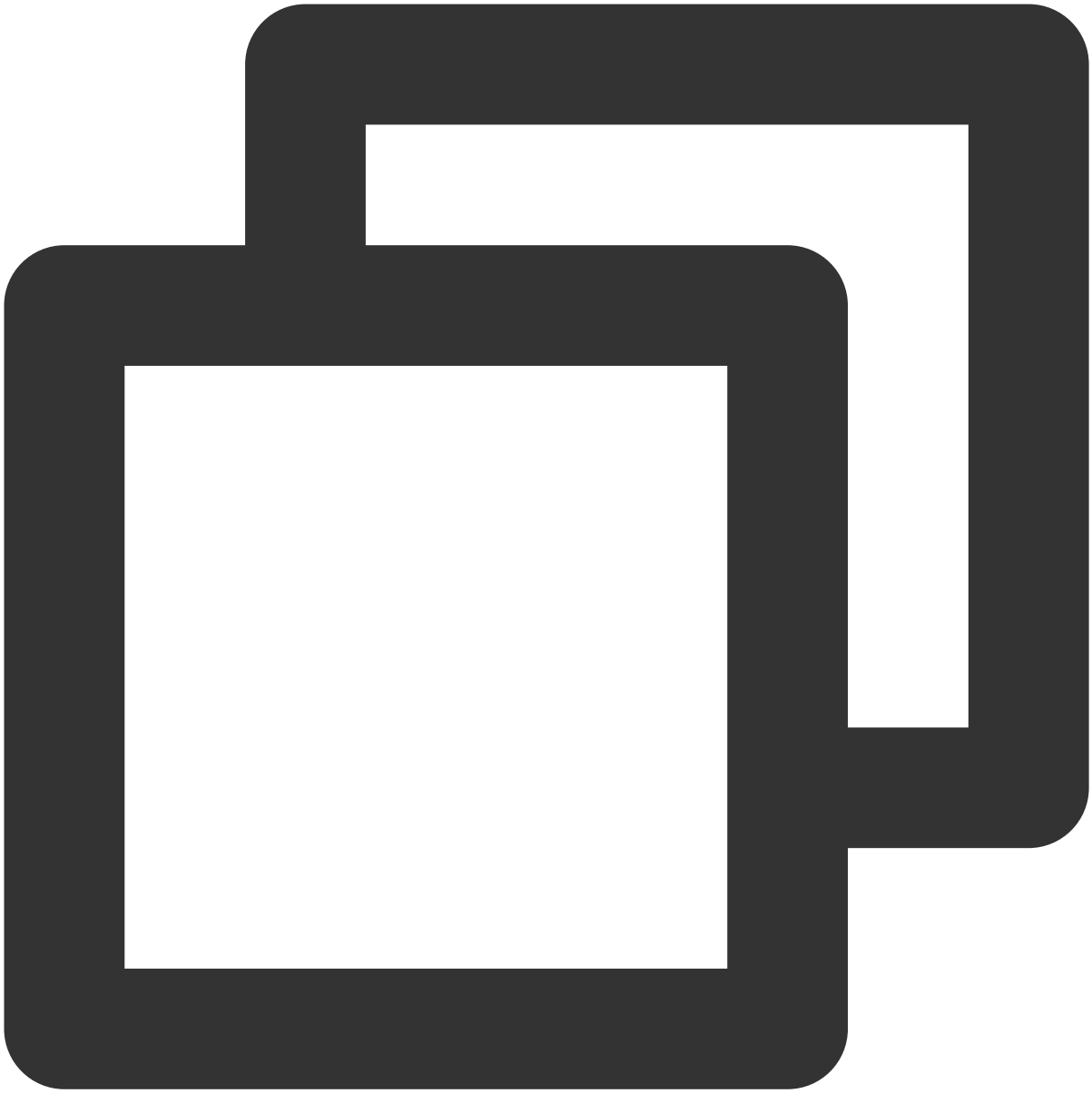
Resource declaration without the `tfstate` mapping is regarded as creation. Therefore, you can copy a file, modify the region, and run `apply` for cross-region replication of resources.

Below is the sample directory of the application deployed in Guangzhou region via Terraform based on the serverless cluster service:



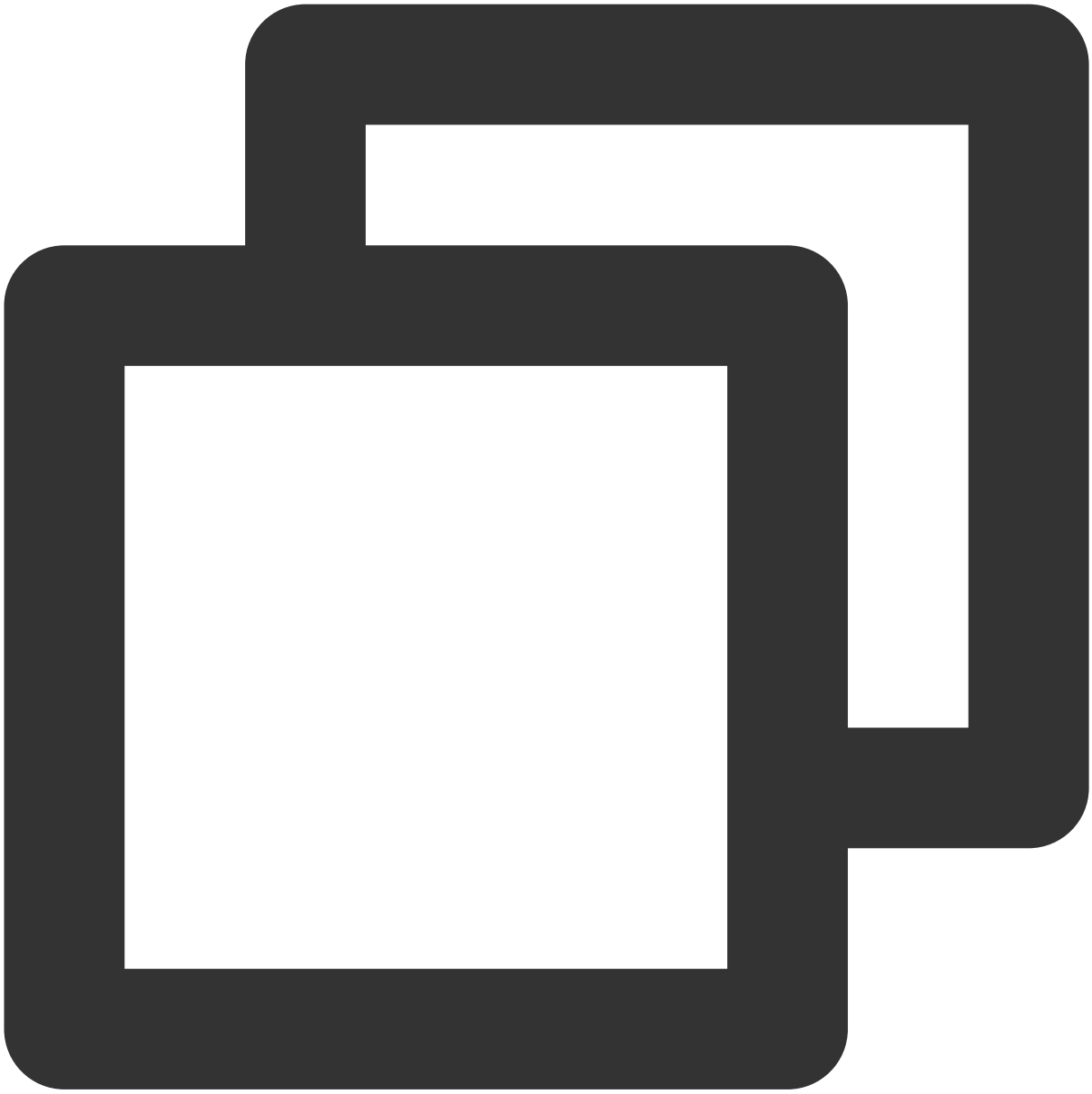
```
eks-app-guangzhou
├── crds.tf
├── infra.tf
├── main.tf
├── terraform.log
└── terraform.tfstate
```

Here, `main.tf` specifies the meta information of Terraform and the provider as follows:



```
terraform {
  required_providers {
    tencentcloud = {
      source = "tencentcloudstack/tencentcloud"
    }
  }
}

provider "tencentcloud" {
  region = "ap-guangzhou"
}
```

`infra.tf` specifies the TKE serverless cluster and required resources: VPC, subnet, security group, TKE serverless cluster, and CLB instance as follows:

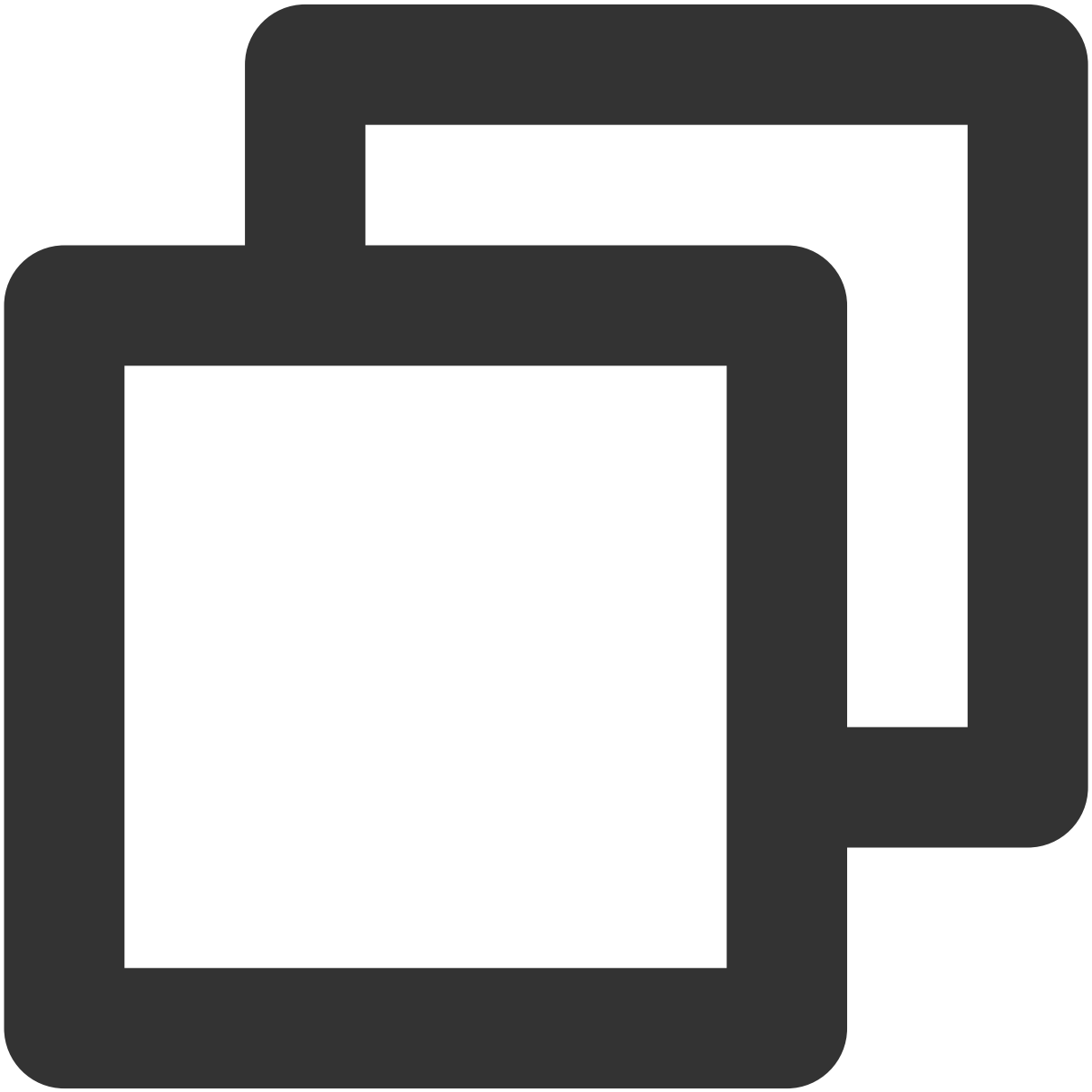


```
# The IP address for the test on the external accessibility of the service
variable "accept_ip" {
   description = "Use EnvVar: $TF_VAR_accept_ip instead"
}

resource "tencentcloud_vpc" "vpc" {
  name       = "eks-vpc"
```

```
  cidr_block = "10.2.0.0/16"
}

resource "tencentcloud_subnet" "sub" {
  vpc_id            = tencentcloud_vpc.vpc.id
  name              = "eks-subnet"
  cidr_block        = "10.2.0.0/20"
  availability_zone = "ap-guangzhou-3"
}

resource "tencentcloud_security_group" "sg" {
  name = "eks-sg"
}

resource "tencentcloud_security_group_lite_rule" "sgr" {
  security_group_id = tencentcloud_security_group.sg.id
  ingress = [
    "ACCEPT#10.2.0.0/16#ALL#ALL",
    "ACCEPT#${var.accept_ip}#ALL#ALL"
  ]
}

resource "tencentcloud_eks_cluster" "foo" {
  cluster_name = "tf-test-eks"
  k8s_version = "1.20.6"
  vpc_id = tencentcloud_vpc.vpc.id
  subnet_ids = [
    tencentcloud_subnet.sub.id,
  ]
  cluster_desc = "test eks cluster created by terraform"
  service_subnet_id = tencentcloud_subnet.sub.id
  enable_vpc_core_dns = true
  need_delete_cbs = true
  public_lb {
    enabled = true
    security_policies = [var.accept_ip]
  }
  internal_lb {
    enabled = true
    subnet_id = tencentcloud_subnet.sub.id
  }
}

resource "tencentcloud_clb_instance" "ingress-lb" {
  address_ip_version           = "ipv4"
  clb_name                     = "example-lb"
  internet_bandwidth_max_out   = 1
```

```
  internet_charge_type          = "BANDWIDTH_POSTPAID_BY_HOUR"
  load_balancer_pass_to_target = true
  network_type                  = "OPEN"
  security_groups               = [tencentcloud_security_group.sg.id]
  vpc_id                        = tencentcloud_vpc.vpc.id
}
```

`crds.tf` specifies the CRD of the TKE Serverless cluster as follows:



```
locals {
```

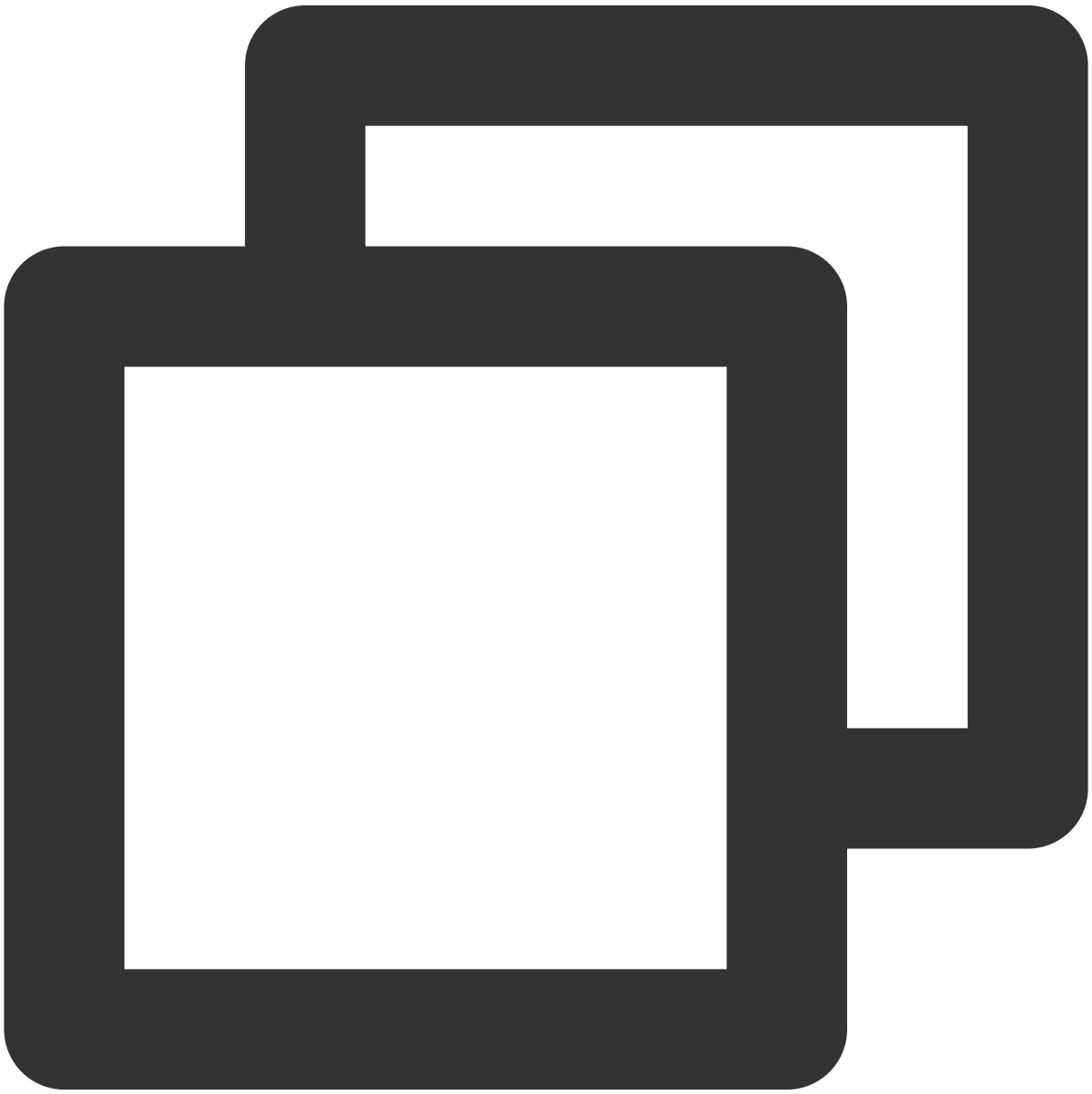
```
  kubeconfig = yamldecode(tencentcloud_eks_cluster.foo.kube_config)
}

provider "kubernetes" {
  host                   = local.kubeconfig.clusters[0].cluster.server
  cluster_ca_certificate = base64decode(local.kubeconfig.clusters[0].cluster["certi
  client_key             = base64decode(local.kubeconfig.users[0].user["client-key-
  client_certificate     = base64decode(local.kubeconfig.users[0].user["client-cert
}

resource "kubernetes_namespace" "test" {
  metadata {
    name = "nginx"
  }
}

resource "kubernetes_deployment" "test" {
  metadata {
    name      = "nginx"
    namespace = kubernetes_namespace.test.metadata.0.name
  }
  spec {
    replicas = 2
    selector {
      match_labels = {
        app = "MyTestApp"
      }
    }
    template {
      metadata {
        labels = {
          app = "MyTestApp"
        }
      }
      spec {
        container {
          image = "nginx"
          name  = "nginx-container"
          port {
            container_port = 80
          }
        }
      }
    }
  }
}
```

```
resource "kubernetes_service" "test" {
  metadata {
    name      = "nginx"
    namespace = kubernetes_namespace.test.metadata.0.name
  }
  spec {
    selector = {
      app = kubernetes_deployment.test.spec.0.template.0.metadata.0.labels.app
    }
    type = "NodePort"
    port {
      node_port   = 30201
      port        = 80
      target_port = 80
    }
  }
}

resource "kubernetes_ingress_v1" "test" {
  metadata {
    name      = "test-ingress"
    namespace = "nginx"
    annotations = {
      "ingress.cloud.tencent.com/direct-access"     = "false"
      "kubernetes.io/ingress.class"                 = "qcloud"
      "kubernetes.io/ingress.existLbId"             = tencentcloud_clb_instance.ing
      "kubernetes.io/ingress.extensiveParameters"   = "{\\"AddressIPVersion\\": \\"
      "kubernetes.io/ingress.http-rules"            = "[{\\"path\\":\\"/\\",\\"back
      "kubernetes.io/ingress.https-rules"           = "null"
      "kubernetes.io/ingress.qcloud-loadbalance-id" = tencentcloud_clb_instance.ing
      "kubernetes.io/ingress.rule-mix"              = "false"
    }
    #    selfLink = "/apis/networking.k8s.io/v1/namespaces/nginx/ingresses/test-ing
  }
  spec {
    rule {
      http {
        path {
          backend {
            service {
              name = kubernetes_service.test.metadata.0.name
              port {
                number = 80
              }
            }
          }
          path = "/"
```

```
            }
          }
        }
      }
    }
```

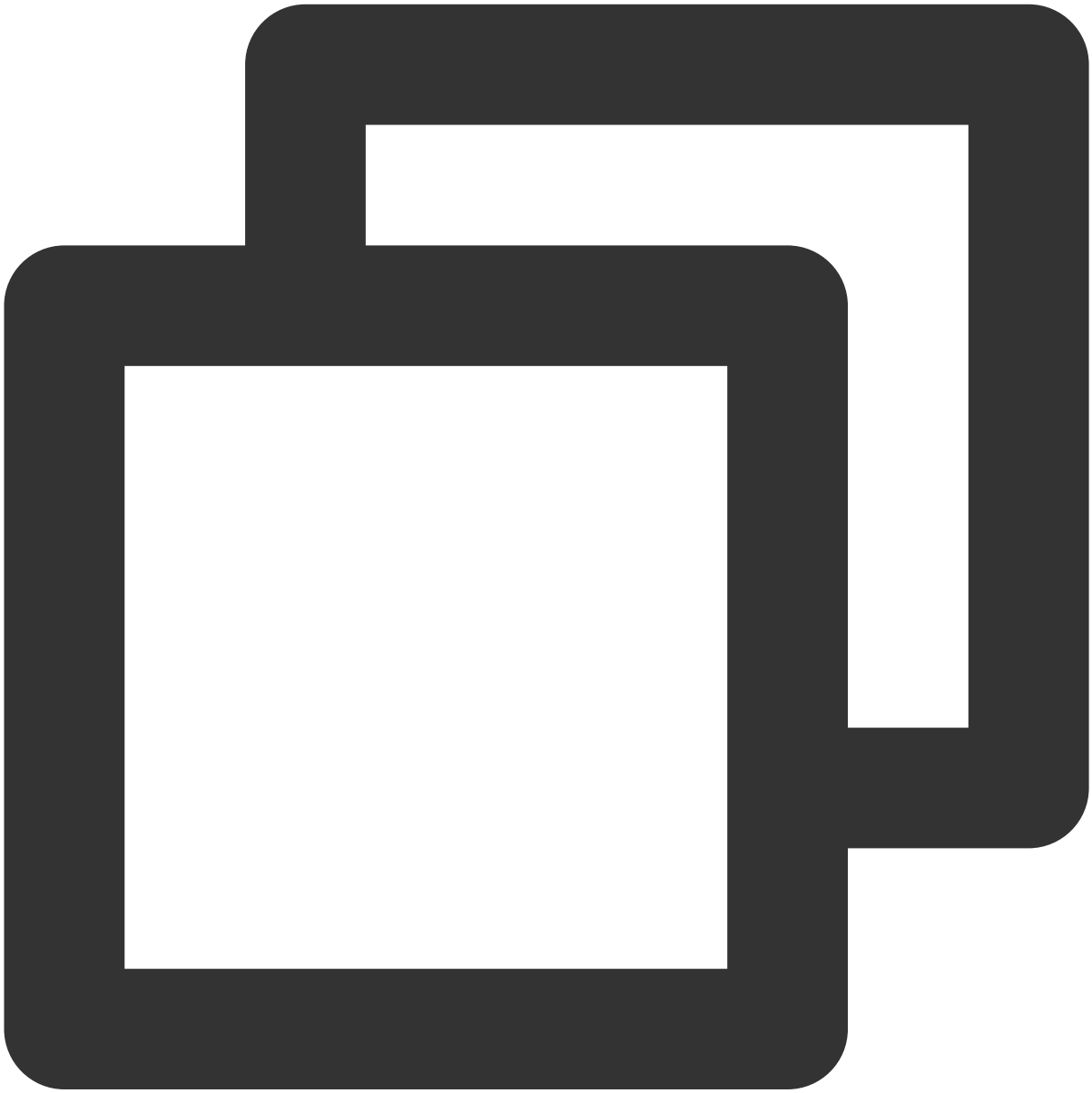
Copy these resources to another region (such as Singapore) in the following steps:

1. Copy all the .tf files to a new directory, for example, `eks-app-singapore` , and remove the original directory's reference to `tfstate` :



```
$ mkdir ../eks-app-singapore
$ cp *.tf ../eks-app-singapore
$ cd ../eks-app-singapore
```

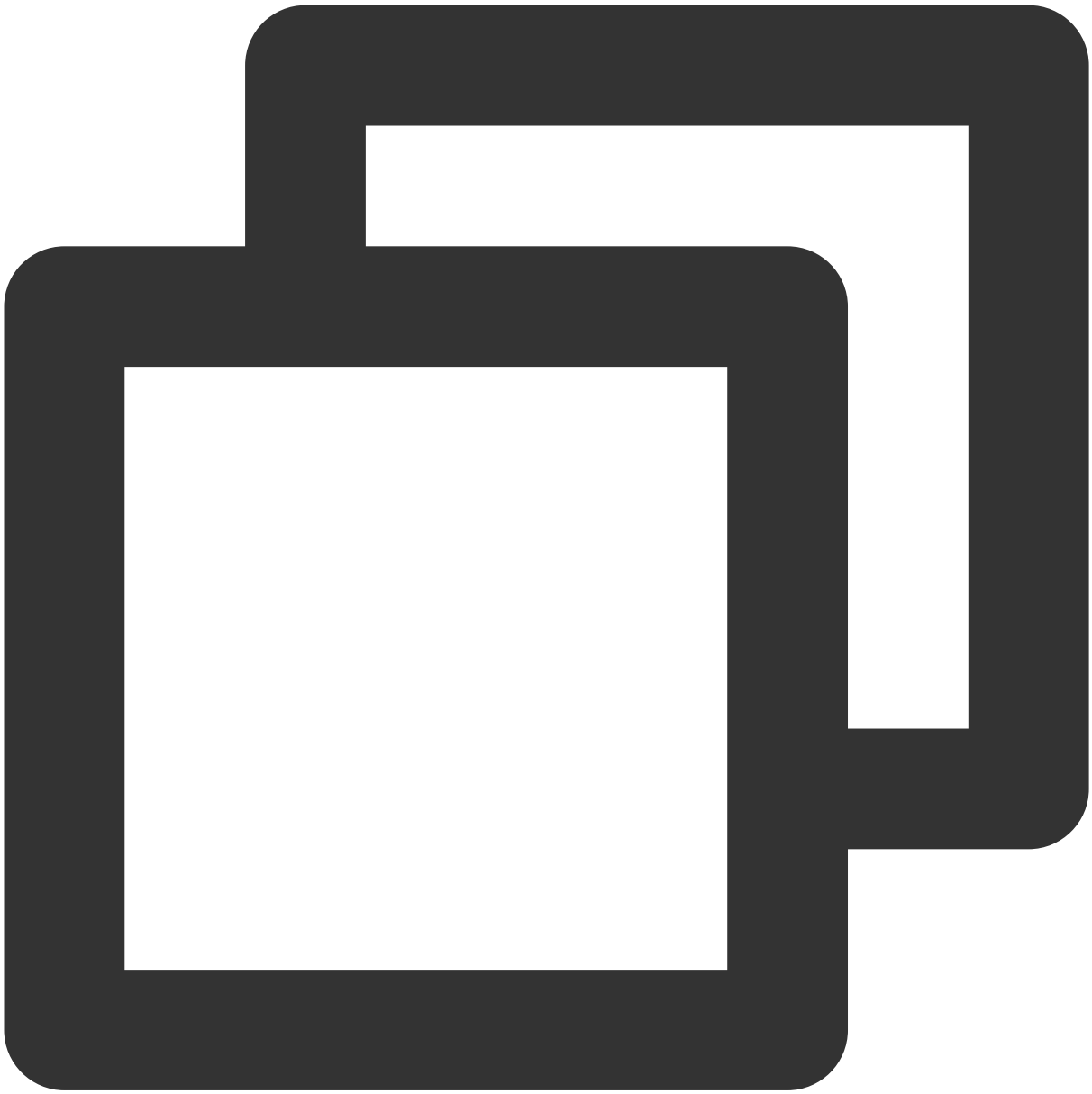
2. Modify the region of the TencentCloud Provider as follows:



```
provider "tencentcloud" {
  # - replace
  # region = "ap-guangzhou"
  # + to
  region = "ap-singapore"
```

```
}
```

3. Run `terraform init` and `terraform plan` in the `eks-app-singapore` directory. As there is no `tfstate` file, `terraform plan` will prompt that resources will be created:



```
Plan: 11 to add, 0 to change, 0 to destroy.
─────────────────────────────────────────────

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

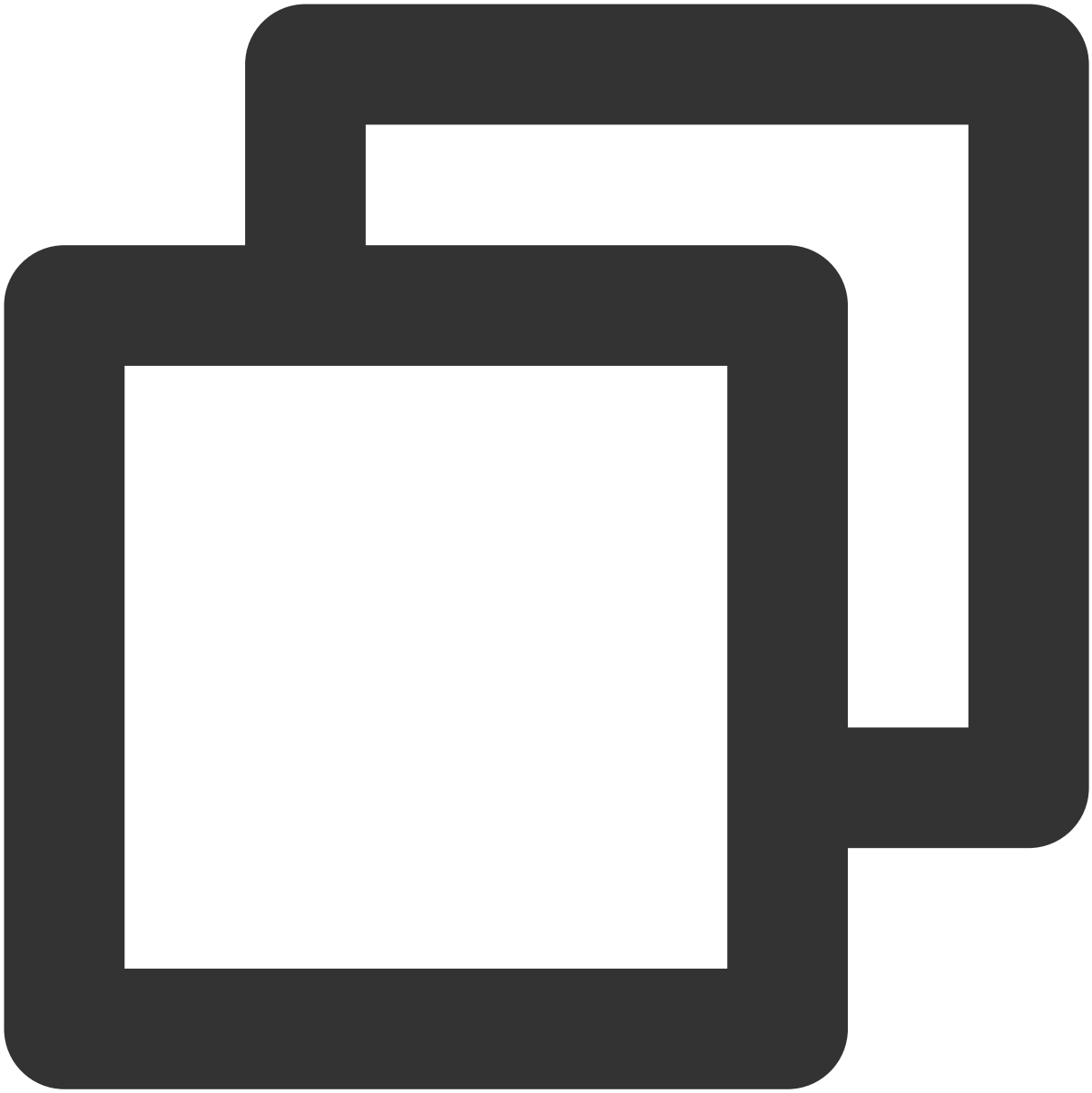
4. After confirming that everything is OK, run `terraform apply` to configure management of cloud resources in the new directory in the new region.

Limitations

Products vary greatly by business form and logic, which makes cross-region replication troublesome. Main limitations are described as follows.

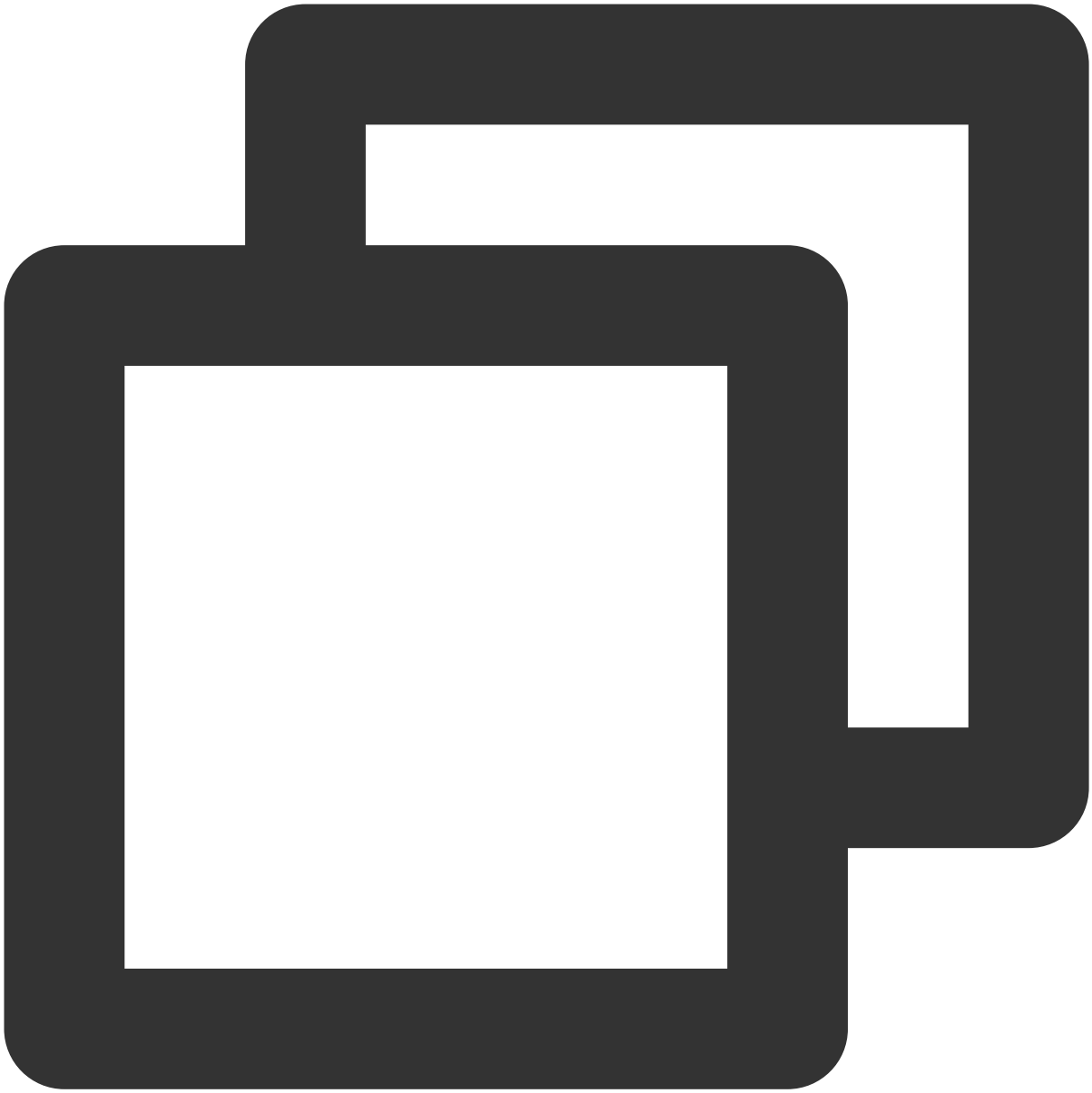
Instance specification and inventory limits

The specification and inventory limits of different Tencent Cloud resources, such as CVM, CBS, and TencentDB instances, vary greatly by AZ, which means that the specification of the current instance may be sold out or unavailable in another AZ. We recommend you use dynamic instance types, that is, query the `datasource` of each resource to query available instance specifications, instead of hard-coding the data in a file, for example: Purchase a 2-core 2 GB MEM CVM instance in Shanghai Zone 4 as follows:



```
resource "tencentcloud_instance" "cvm" {
  name              = "my-instance"
  availability_zone = "ap-shanghai-4"
  image_id          = "local.cvm_img_id"
  instance_type     = "S5.MEDIUM2"
}
```

Switch to Guangzhou region and dynamically get the information via `datasource` as follows:



```
provider "tencentcloud" {
   region = "ap-guangzhou"
}

# Query the AZs in Guangzhou region where CVM instances are available
data "tencentcloud_availability_zones_by_product" "cvm" {
  product = "cvm"
}

# Query CVM images starting with `Tencent`
data "tencentcloud_images" "img" {
```

```
  image_name_regex = "Tencent"
}

# Query the 2-core 2 GB MEM instance types in the specified AZ
data "tencentcloud_instance_types" "types" {
  availability_zone = data.tencentcloud_availability_zones_by_product.cvm.zones.0.n
  cpu_core_count = 2
  memory_size = 2
}

locals {
   # Select the first result in the AZ list
  cvm_zone = data.tencentcloud_availability_zones_by_product.cvm.zones.0.name
   # Select the first result in the image list
  cvm_img_id = data.tencentcloud_images.img.images.0.image_id
   # Select the first result in the instance type list
  cvm_type = data.tencentcloud_instance_types.types.instance_types.0.instance_type
}

resource "tencentcloud_instance" "cvm" {
  name              = "my-instance"
  availability_zone = local.cvm_zone
  image_id          = local.cvm_img_id
  instance_type     = local.cvm_type
}
```

Resource quantity limit

Certain resources (such as TKE clusters, VPCs, and COS buckets) are subject to quantity limits in each region. Before replication, check whether the existing quota is sufficient in the target region. To increase the quota, submit a ticket for application.

Resources that do not need replication

Certain resources are not region-specific, such as CAM users/roles and policies, SSL certificates, and SSH keys. You need to filter them out during replication to avoid recreation.