# 资源编排（TIC）

## 产品文档

# 目录

资源编排（TIC） 3
- 产品简介 3
  - 产品概述 3
  - 应用场景 4
  - 云产品列表 6
- 操作指南 7
  - 产品简介 7
  - 资源栈管理 8
  - 新建资源栈 9
  - 模板管理 12
  - 平台设置 15
  - 资源类型 20
  - Provider使用文档 23
    - 具体功能项 23
    - CLI 使用方式 24
      - 安装二进制包 24
      - 认证和鉴权 25
      - 使用 Terraform 创建云平台资源（以 VPC 为例） 27
      - tf 代码示例 32
        - 私有网络 vpc 示例 32
        - 新建网络子网示例 33
        - 新建 cvm 虚拟机示例 34
        - 新建 cbs 磁盘示例 35
      - 附录 36
- 监控介绍
- 最佳实践 37
  - 云资源跨区复制 37
- 常见问题 40
  - 常见问题 40
- 运维管理指南 41
  - 产品架构 41
    - 核心功能组件 41
    - 产品架构图 42
    - 部署架构图 43
    - 业务流向图 44
    - 核心逻辑概述 45
  - 产品依赖 46
  - 故障处理 47
    - 故障影响范围 47
    - 故障恢复 48
      - 场景1：websocket服务无法连接 48
      - 场景2：创建资源栈时无法选择地域 50
      - 场景3：编排的时候报错（通用性问题） 51
  - 日常巡检 55
    - 日常巡检列表 55
    - 巡检处理 56
      - 巡检项1：检查TIC控制台 56
      - 巡检项2：检查COS状态 57
      - 巡检项3：检查terraform和插件的状态 58
      - 巡检项4：检查cgi pod的状态 60
  - 日常监控 61
    - Pod 自监控 61

# 产品简介

# 产品概述

资源编排 TIC 是云平台推出的 IaC 开放平台，融合开源技术，通过 IaC 的方式解决客户在云基础设施管理中面临的效率、成本和安全问题。TIC 是 Terraform 在云平台的应用。

# 使用场景

- 业务上线前的各类云资源准备
- 业务发展过程中的快速扩缩容
- 不同业务场景(开发测试生产)快速构建销毁
- 灾备恢复，快速创建与主站相同配置的资源

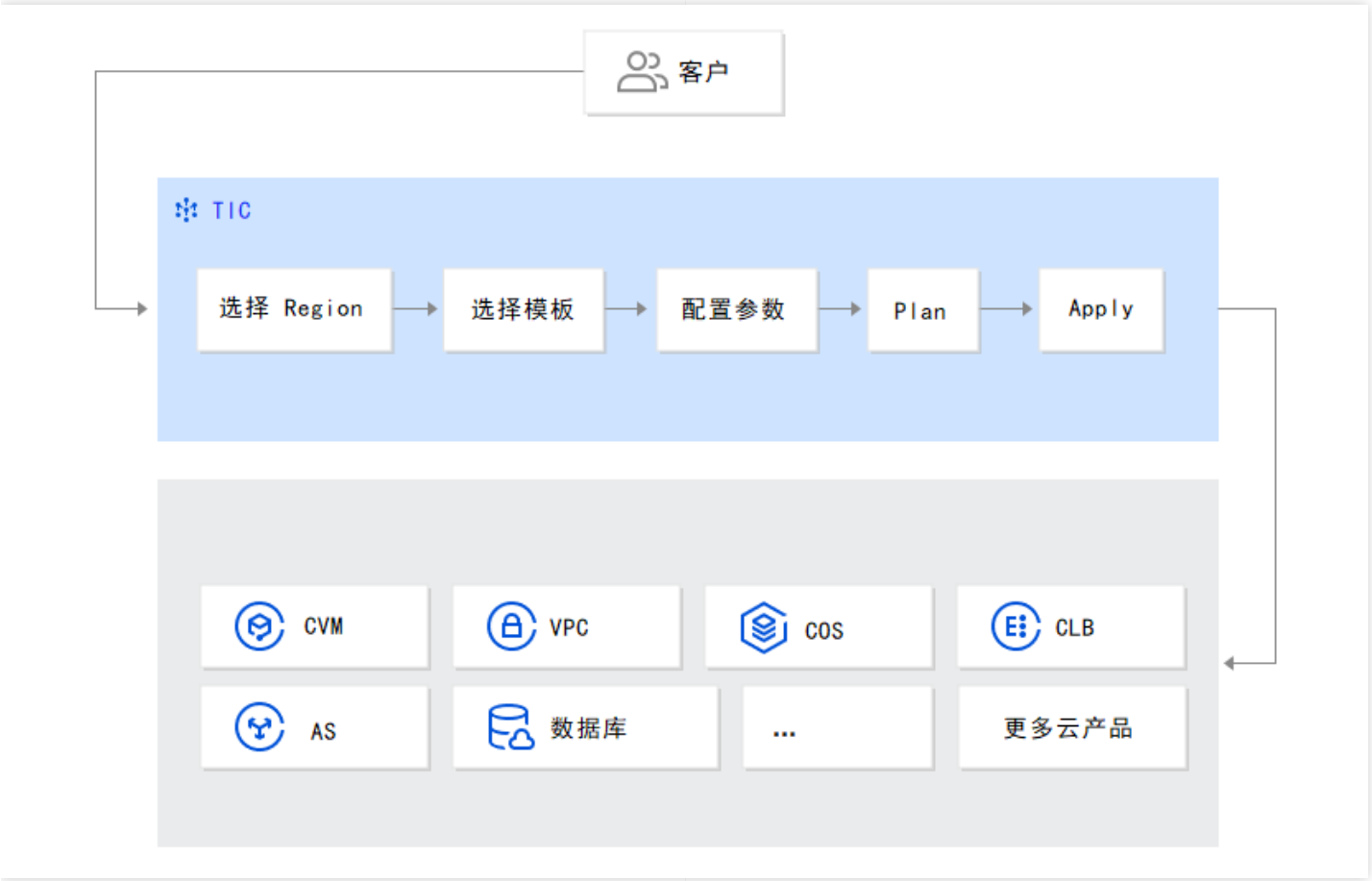
# 产品优势

- 开源兼容：TIC 全面支持 Terraform，兼容 HCL（Terraform）格式语法，已有Terraform模板可导入使用。
- 简单易用：支持可视化拖拽式编排，无需学习代码语法，即可快速上手。同时也提供了丰富的模板，进行简单地参数修改就可以迅速构建属于自己的基础设施。

# 应用场景

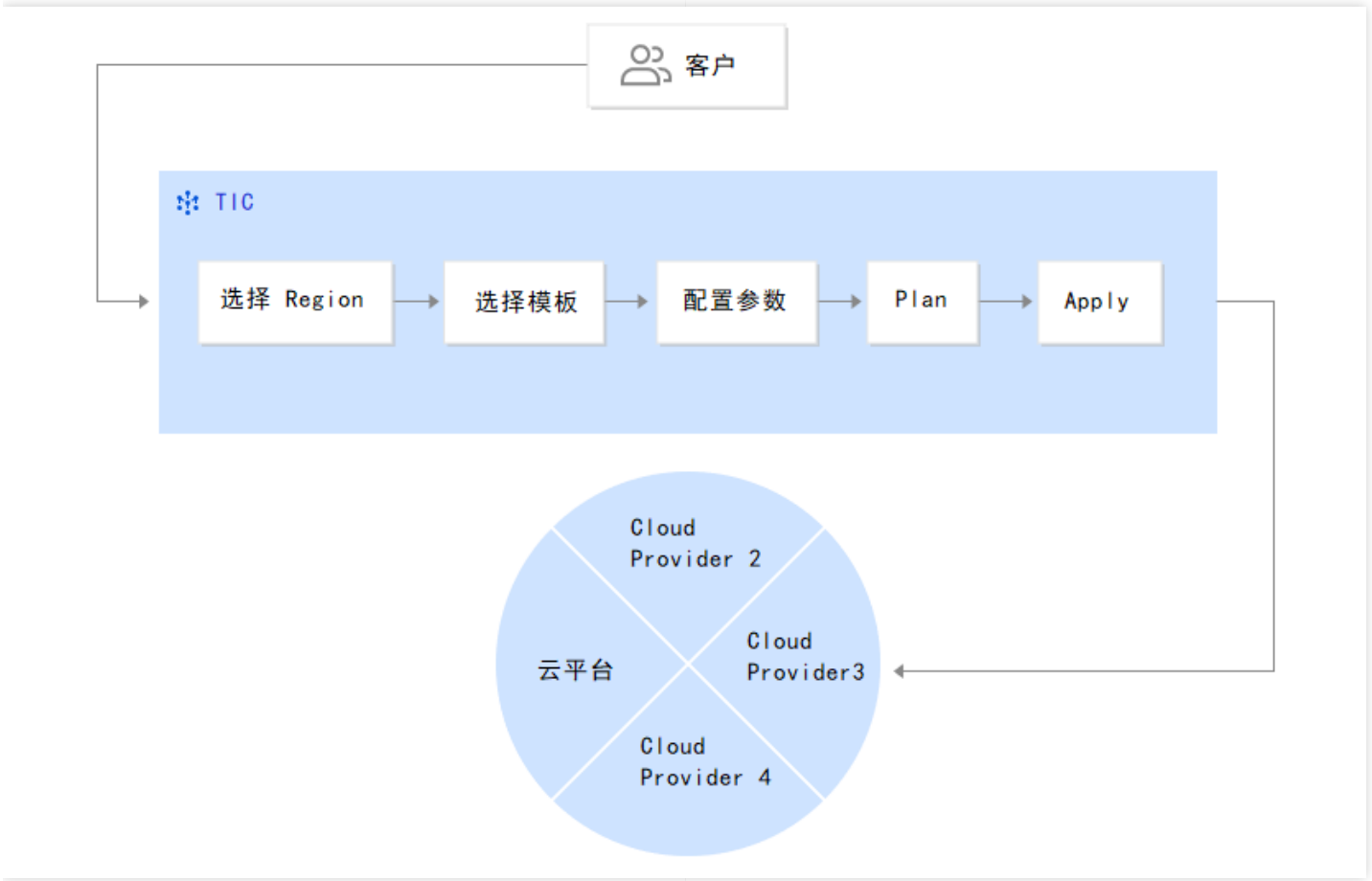
快速创建云平台基础架构

当您需要快速创建多个相同的基础架构需求时，例如在不同地域下创建相同的基础架构或者跨区灾备，TIC 将会是您的最佳选择。TIC 可以很好的避免重复性的控制台点击操作，也可以省去 API 使用情况下的学习成本，规避误操作风险。您只需要指定地域信息，同时选用已有的模板、修改必要的参数，提交即可创建对应的基础架构。




部署多云环境下的基础架构

当您需要构建多云环境下的基础架构时，实现多云间的资源管理、迁移，TIC 将会是您的最佳选择。TIC 会根据不同的云服务商资源调用相关 API，并基于资源的依赖关系去构建多云的基础架构。


客户
TIC
选择 Region
选择模板
配置参数
Plan
Apply
Cloud Provider 2
云平台
Cloud Provider3
Cloud Provider 4

# 云产品列表

支持的云产品列表：

| 分类 | 产品 |
| --- | --- |
| 计算 | CVM |
| | AS |
| | BMS |
| | TKE |
| 网络 | VPC |
| | EIP |
| | CLB |
| | VPCDNS |
| 存储 | CBS |
| | CFS |
| | COS |
| | CSP |
| 中间件 | Ckafka |
| | TDMQ |
| | TSF |
| 数据库 | TDSQL-MySQL |
| | TDSQL-PostgreSQL |
| | Redis® |

# 操作指南
# 产品简介

资源编排 TIC 是云平台推出的 IaC 开放平台，融合开源技术，通过 IaC 的方式解决客户在云基础设施管理中面临的效率、成本和安全问题。TIC 是 Terraform 在云平台的应用。

## 使用场景

- 业务上线前的各类云资源准备
- 业务发展过程中的快速扩缩容
- 不同业务场景(开发测试生产)快速构建销毁
- 灾备恢复，快速创建与主站相同配置的资源

## 产品优势

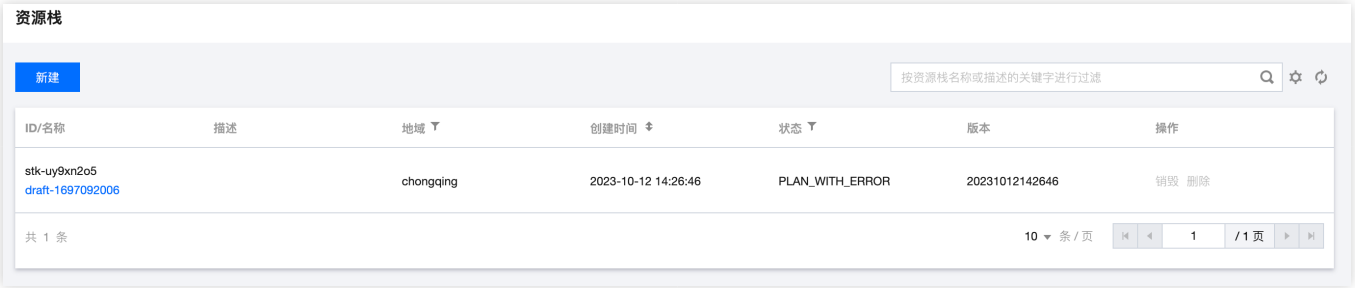
- 开源兼容：TIC 全面支持 Terraform，兼容 HCL（Terraform）格式语法，已有 Terraform 模板可导入使用。
- 简单易用：支持可视化拖拽式编排，无需学习代码语法，即可快速上手。同时也提供了丰富的模板，进行简单地参数修改就可以迅速构建属于自己的基础设施。

# 资源栈管理

本文为您介绍如何查看和管理资源栈。

## 操作步骤

1. 登录TIC控制台。
2. 在左侧菜单中，选择【资源编排】>【资源栈】，进入资源栈列表页面。该页面展示当前所有的资源栈列表、以及资源栈状态、地域、创建时间、版本等基本信息。




3. 支持资源栈销毁、删除。
4. 单击任一资源栈，可以查看资源栈详情，包括：
    - 属性：显示资源栈基本信息。
    - 版本：资源栈版本管理页面。可以查看和管理当前资源栈的历史版本，包括新建版本、版本导出、保存为模板等操作。
    - 资源：查看资源栈下的资源列表。
    - 事件：资源栈的状态变化、版本变化都会成为事件，展示资源栈所有的事件列表。

> 注意：
>
> 任一资源栈仅支持一个版本草稿，即在新建版本时如果未选择具体版本，若存在编辑态版本（VERSION_EDITING）则默认以该版本为基础进行创建，若无编辑态版本，则以当前运行版本为基础进行创建，否则以所选版本为基础进行创建。

# 新建资源栈

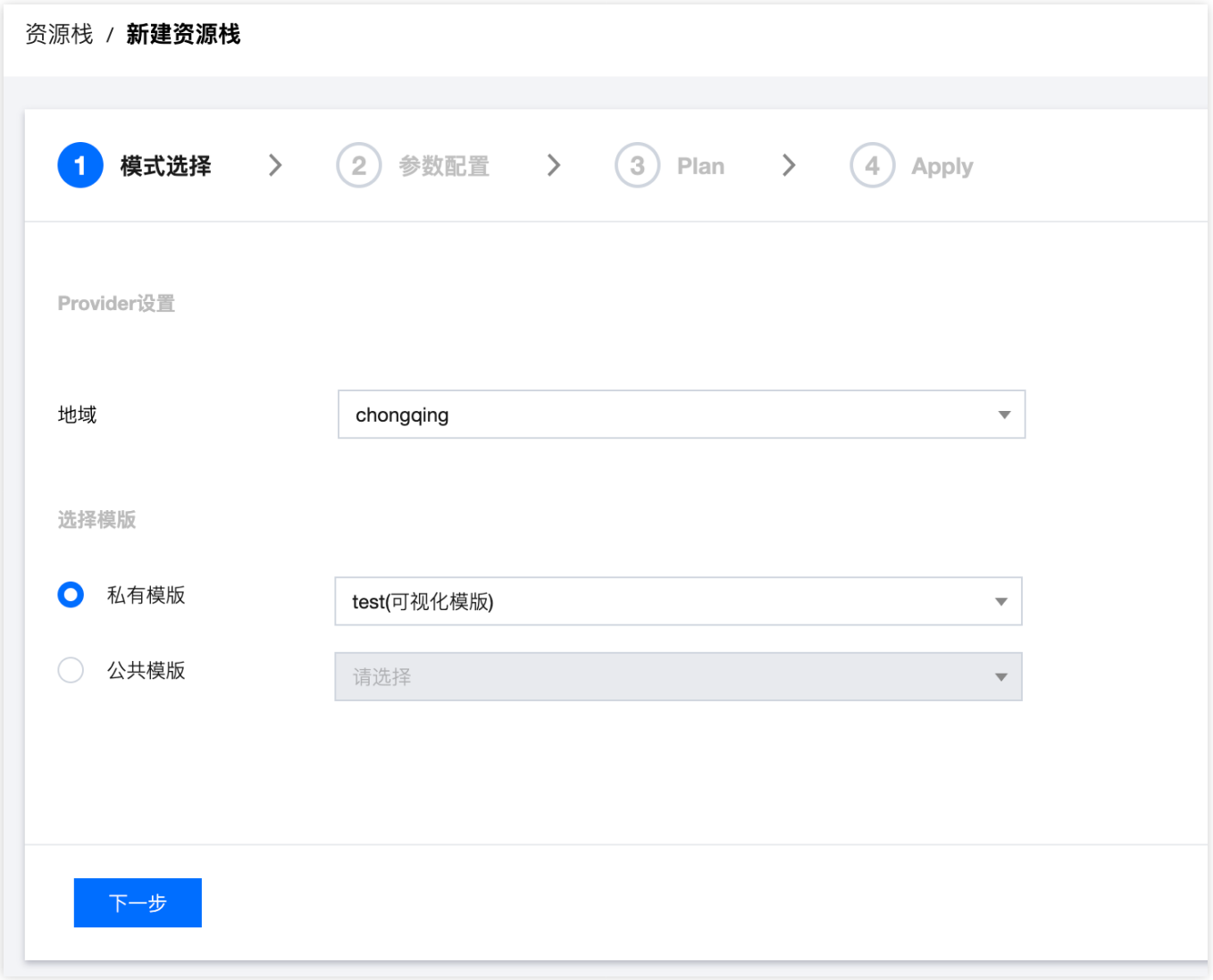
新用户首次使用 TIC 需要新建资源栈，或者根据业务需要创建新资源栈。

## 前提条件

新建资源栈前，您需要首先在平台设置中设置TIC授权。

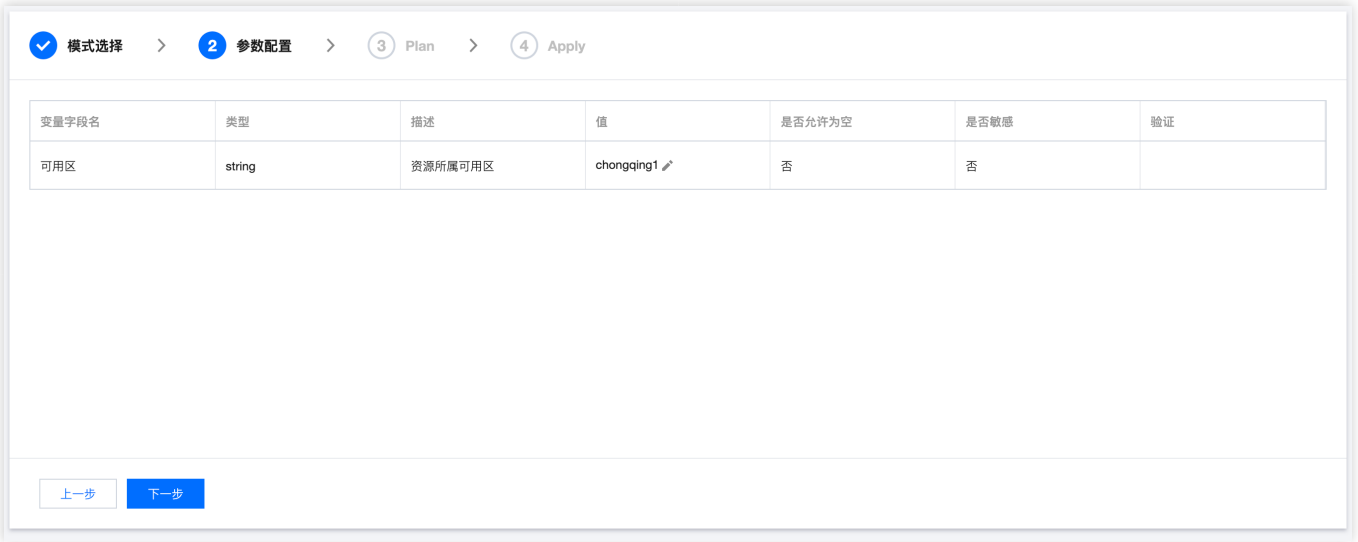
## 操作步骤

1. 登录TIC控制台。
2. 在左侧菜单中，选择【资源编排】>【资源栈】，进入资源栈页面。
3. 单击【新建】，进入新建资源栈页面，按以下步骤配置。
   步骤1：模式选择
4. 地域：选择资源栈的地域，即该资源栈下的所有的资源都属于该地域。
5. 指定新建资源栈的方式：
   - 私有模板：选择私有模板，了解更多模板详情，请参见 模板管理 。
   - 公共模板：选择公共模板，了解更多模板详情，请参见 模板管理 。
6. 在完成上述配置后，单击【下一步】，进入第二步。

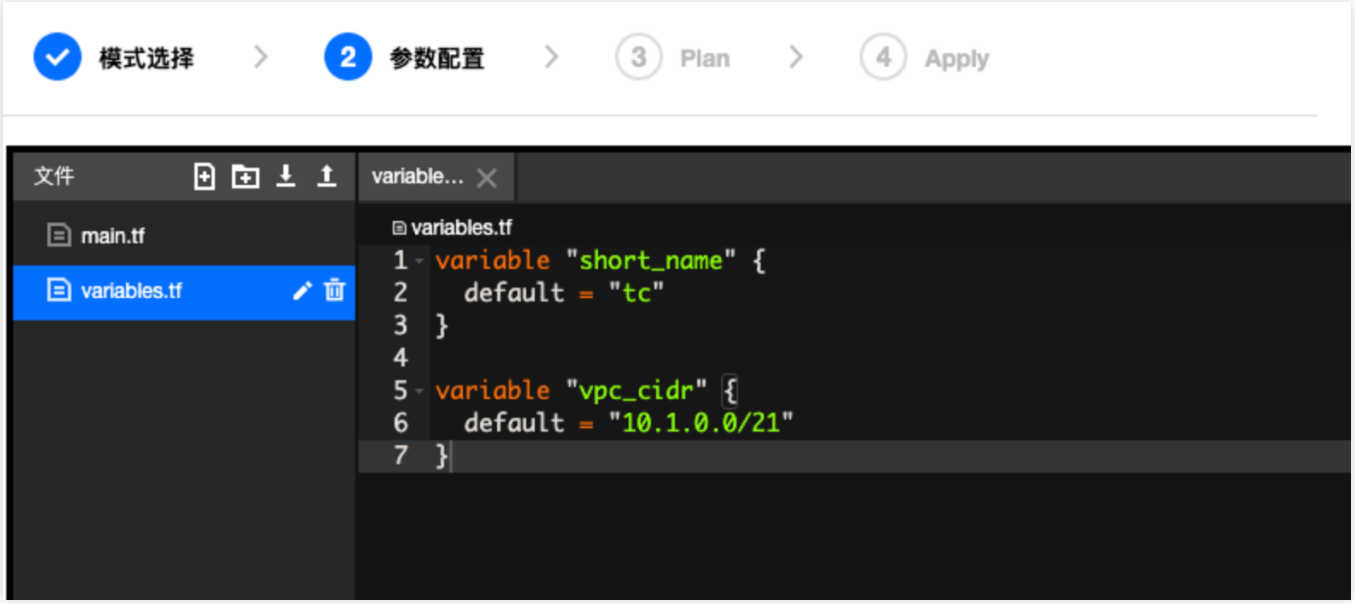



步骤2：参数配置

7. 如果选择的是可视化模板：在列表中设置每个入参的取值，系统将自动填充对应的代码。



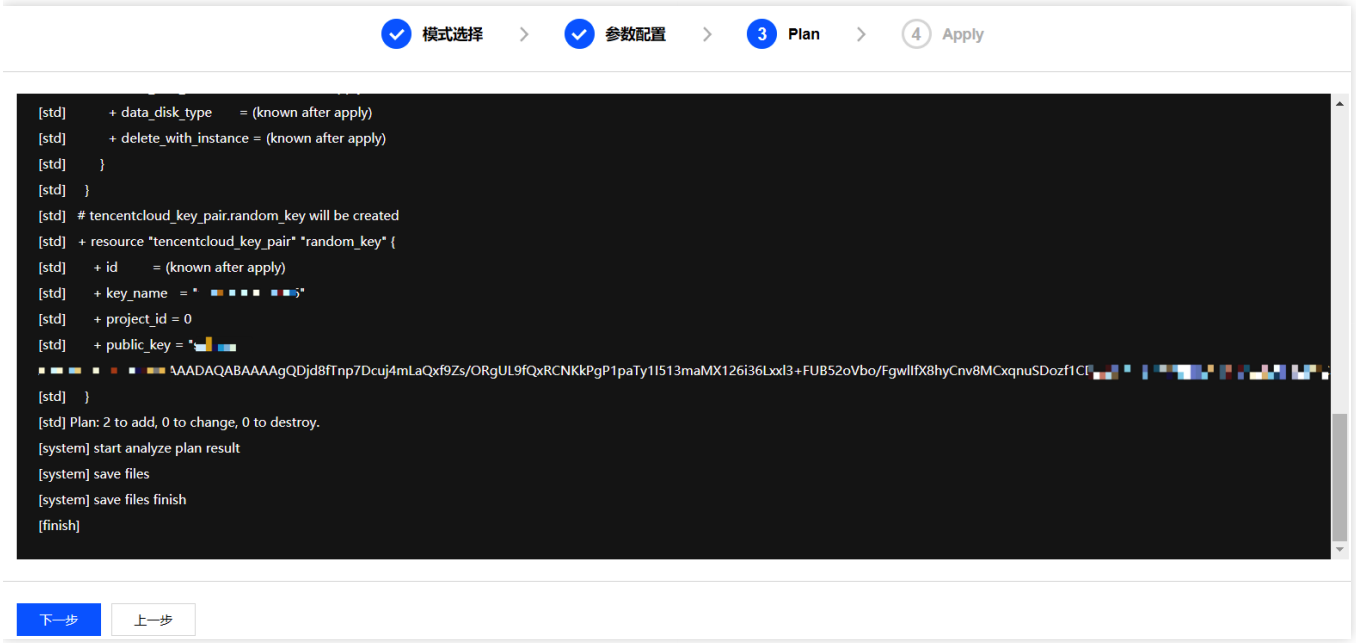

8. 如果选择的是代码模板：在变量设置文件中手动输入每个入参的取值。




9. 单击【下一步】进入 Plan 预览。

步骤3：Plan 操作

TIC 会进行语法检查、模拟相关创建，并给出 Plan 结果以供您参考是否与规划一致。单击【下一步】进入 Apply 资源栈创建步骤。




步骤4：Apply 操作

输入新建资源栈名称以及描述。

完成上述步骤后，单击【确认】，TIC 将正式提交创建请求，并跳转至新建资源栈事件列表。

# 模板管理

本文为您介绍如何管理私有模板和公共模板。

## 操作步骤

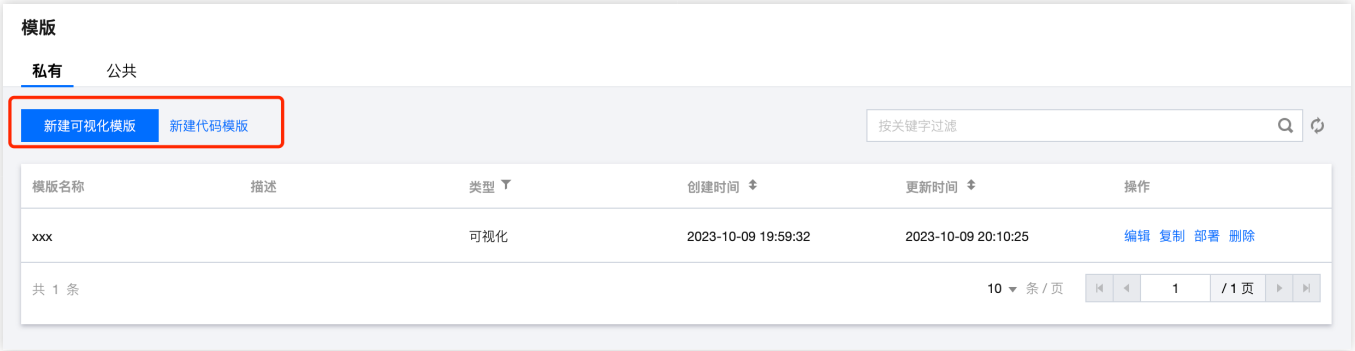
登录 TIC 控制台，进入“模板管理”页面。模板管理提供私有模板管理和公共模板管理：

## 私有模板

### 新建私有模板

TIC提供了两种方式新建模板：

1. 可视化编排，拖拽式生成模板。
2. 纯代码模式，通过代码编辑生成模板。




### 可视化模板

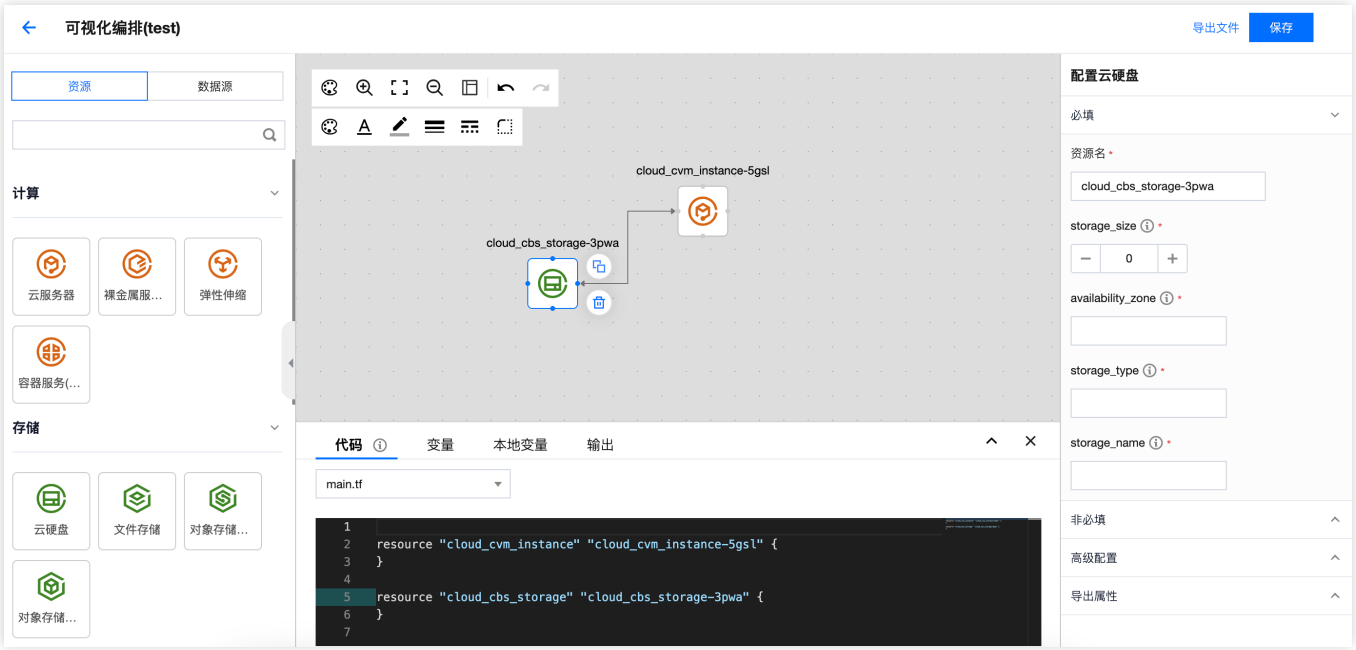
入口：模板管理-私有模板-新建可视化模板
配置项：

1. 模板名称、模板描述：均可自定义设置
2. 进入编排页面：
    i. 左侧：分产品呈现可用的资源、数据源列表，如创建CVM实例、创建云硬盘等。
    ii. 中间上方：编排区域，可拖拽左侧的资源/数据源到编排区域，并通过连线表达先后创建关系。整个编排区域可以自由灵活拖拽布局、连线、复制资源、删除资源，从而把整个资源架构拖拽呈现出来。
    iii. 中间下方：整体设置区域，包含自动生成的代码查看、变量管理、输出管理。
        a. 代码：根据编排区域拖拽的资源自动生成相关代码，无需人工写代码。
        b. 变量：在整个模板执行时需要输入的变量、描述、默认值设置。如：资源所属地域、可用区均可设置为变量。
        c. 本地变量：在编排的资源之间内部使用的变量，在模板执行时不感知。如：一个复杂的表达式、一段代码等，可在其他配置中多次引用。

d. 输出：模板执行完成后的输出的字段，如：CVM实例状态，可以在执行完模板后自动输出。

iv. 右侧：针对编排区域的单个资源进行具体配置项的设置。如创建CVM，设置实例名、镜像、实例类型等，也包括一些高级设置，如资源数量等。

3. 整体编排完可视化模板后，可以保存，也可以直接导出代码文件。
4. 也支持对模板的编辑、删除、复制、部署。

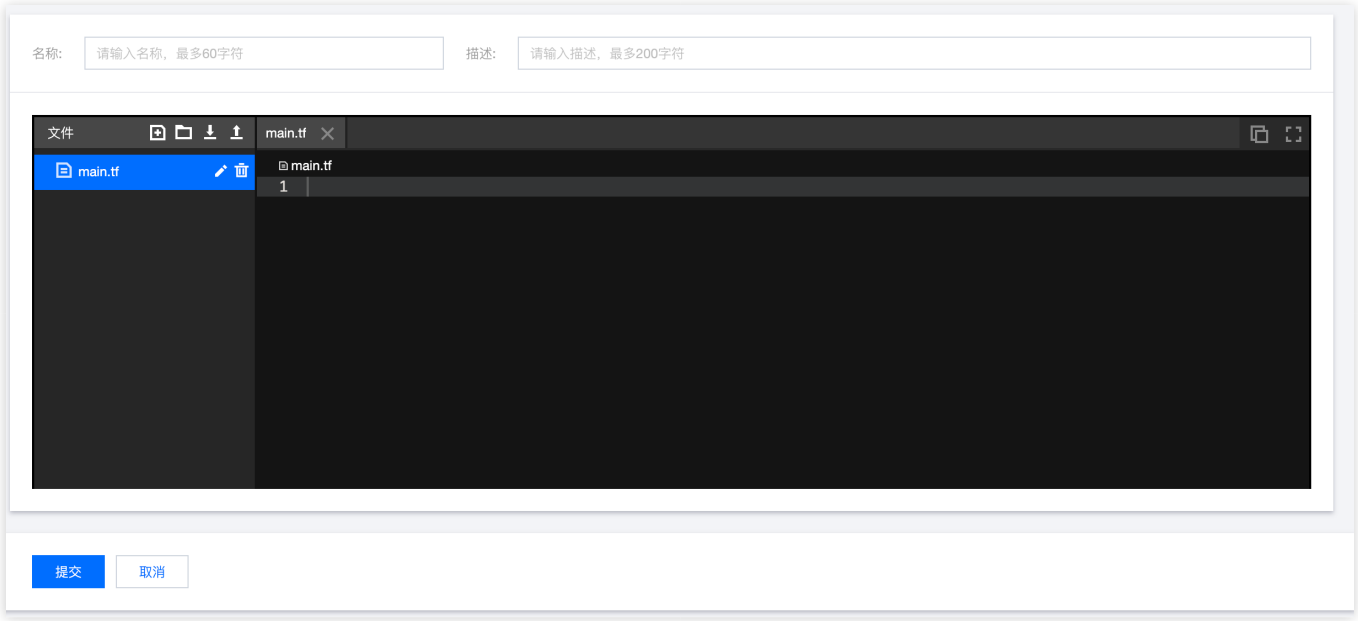



代码模板

入口：模板管理-私有模板-新建代码模板

配置项：

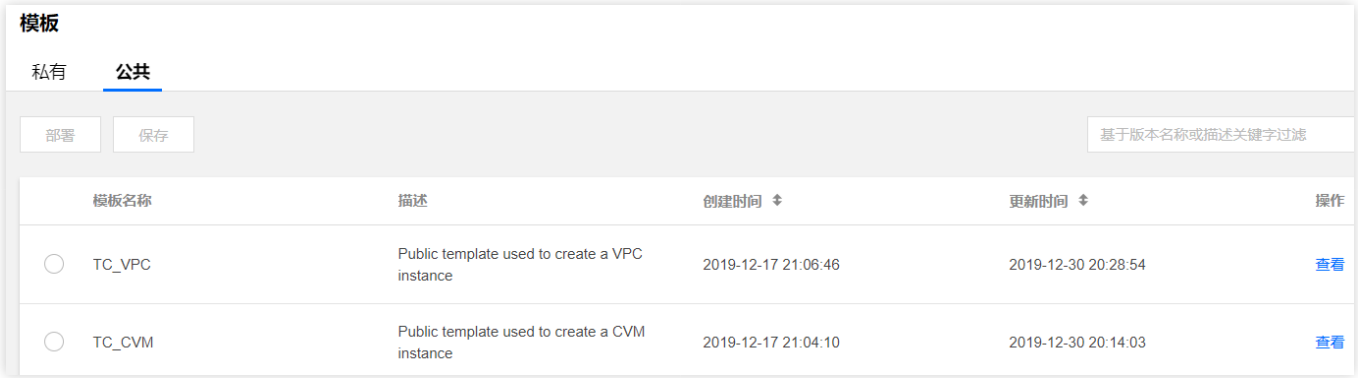
1. 模板名称、模板描述：均可自定义设置。
2. 自定义模板包含的.tf文件，针对每个文件进行代码编写。TIC 全面支持 Terraform，兼容 HCL（Terraform）格式语法。




# 公共模板

公共模板管理可进行查看、部署操作。

1. 单击【公共】，进入公共模板管理页面。
2. 选择任一公共模板，单击【部署】，TIC 控制台将会使用该模板创建新建资源栈。



# 平台设置

## 简介

首次使用资源编排 TIC 服务时，需要您显示授权 TIC 编排云账号下的云资源，否则 TIC 将无法执行云资源的编排操作。

资源编排 TIC 服务提供了 TIC 授权方式：

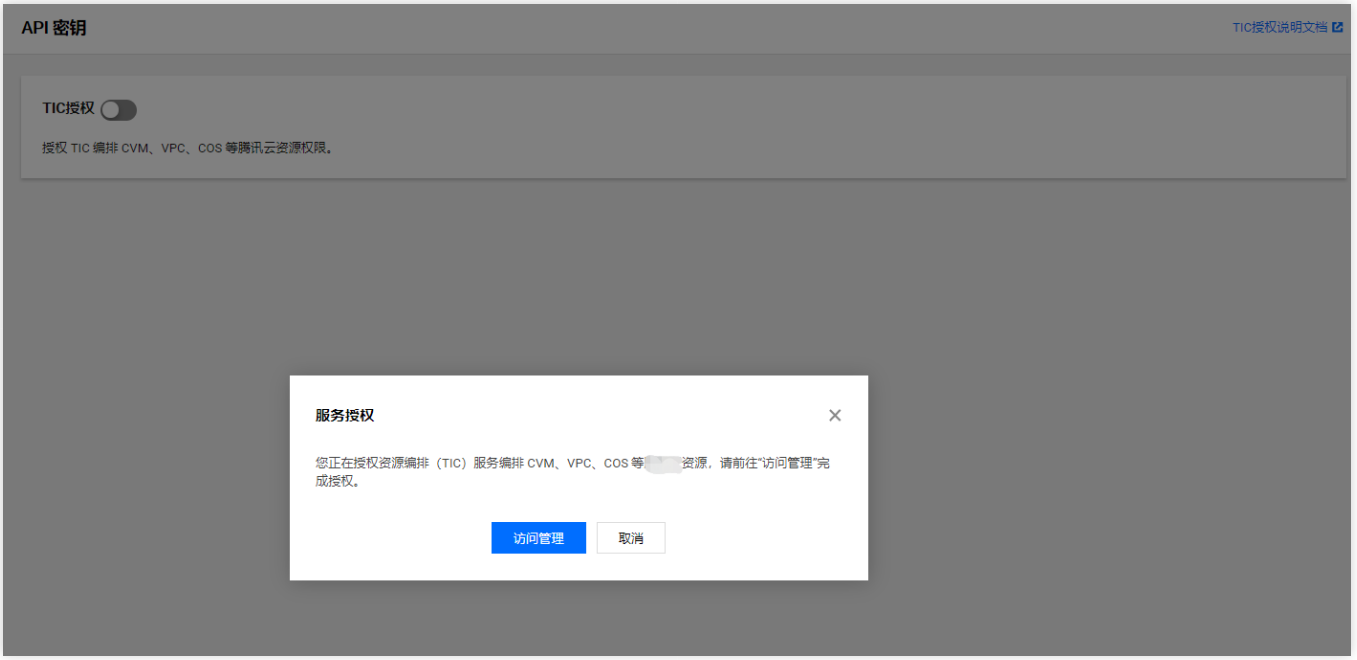
TIC 授权：基于访问管理 CAM 内置的服务相关角色授权机制，用户无需向 TIC 平台托管 API 密钥，即可实现通过 TIC 对云资源的编排，操作更高效，更安全，且满足审计合规要求。

## 操作步骤

### TIC 授权

开启 TIC 授权

1. 登录 TIC 控制台。
2. 在左侧菜单栏中，选择【平台设置】>【API 密钥】，进入 API 密钥管理页面。
3. 单击【TIC 授权】开关，提示跳转到访问管理 CAM 页面进行授权。



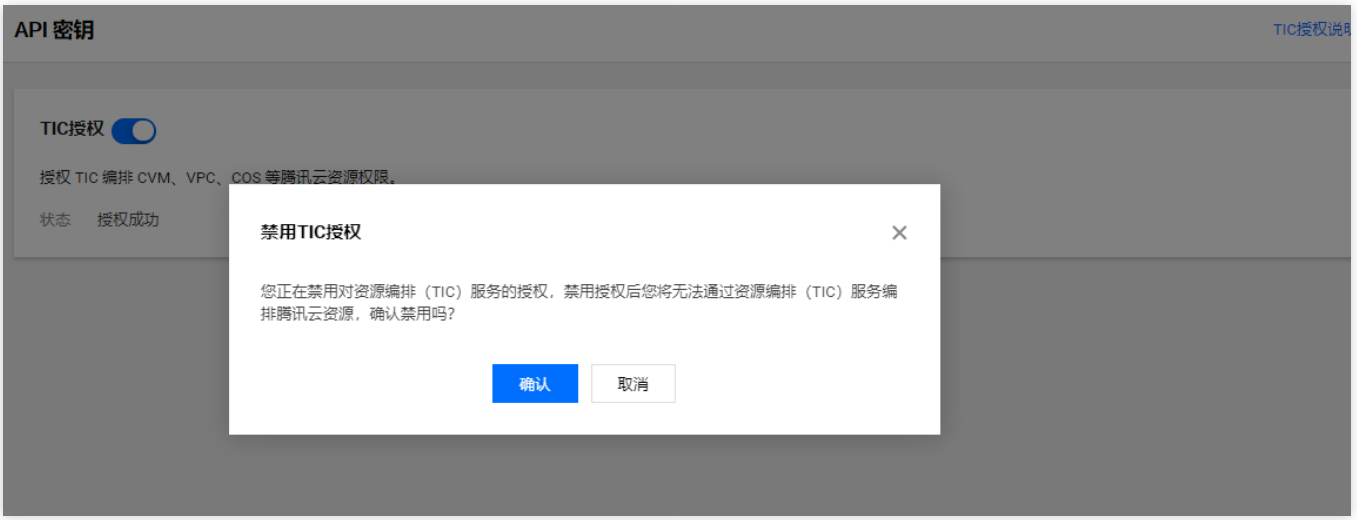

4. 跳转到访问管理 CAM 页面，单击【同意授权】，即可完成授权。

5. 授权成功，跳回到 TIC 页面，TIC 授权开启成功。

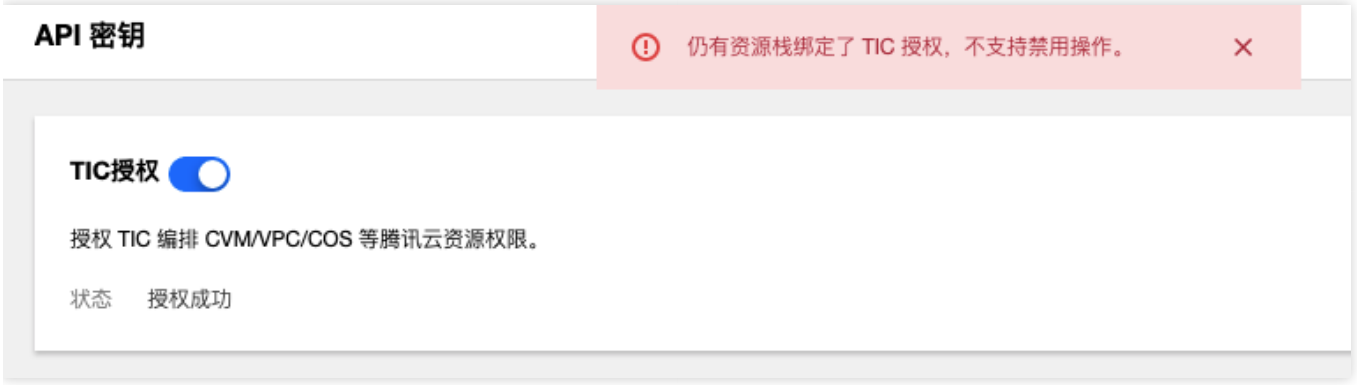
开启 TIC 授权后，您可以继续新建资源栈了解资源安全状态。

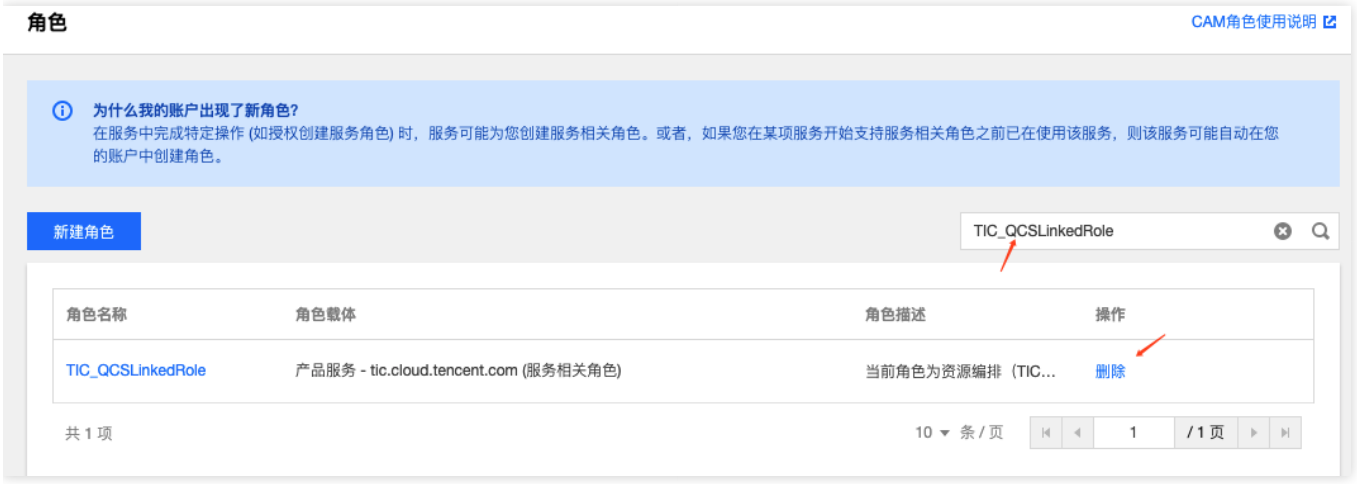
禁用 TIC 授权

1. 登录 TIC 控制台。
2. 在左侧菜单栏中，选择【平台设置】>【API 密钥】，进入 API 密钥页面。
3. 单击【TIC 授权】开关，禁用 TIC 授权。




4.若仍存在绑定了 TIC 授权的资源栈，将无法执行禁用 TIC 授权操作，需要您手动将资源栈删除后，才能禁用 TIC 授权：




> 注意：
>
> 禁用 TIC 授权，并不会删除访问管理 CAM 侧的服务相关角色配置，如果彻底删除 CAM 侧的服务相关角色授权配置，请跳转到 CAM 角色页面，搜索 TIC_QCSLinkedRole 角色名称，单击【删除】即可彻底删除 TIC 服务相关角色授权配置。

授权失败异常

若您在访问管理 CAM 角色中删除了 TIC_QCSLinkedRole 角色，且未关闭 TIC 授权，TIC API 密钥管理页面会提示【授权失败】信息。鼠标悬浮在 图标上，系统提示重新授权：

单击【重新授权】，将跳转到如下访问管理 CAM 页面，请进行重新授权：

授权成功后，授权状态恢复正常：

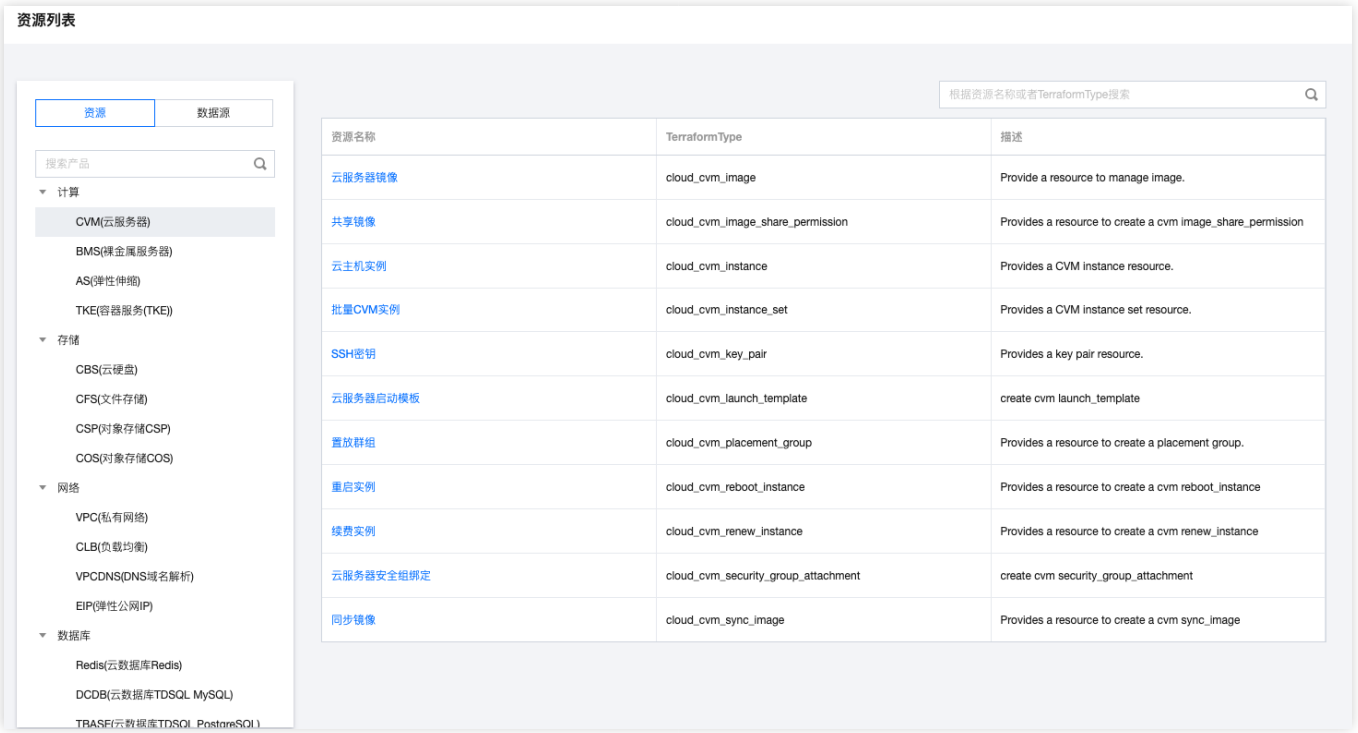
API 凭证
TIC授权
授权TIC编排CVM/VPC/COS等云资源权限。
状态
授权成功

# 资源类型

资源列表页面支持您查看资源的代码详情，方便直接复用代码进行模板构建。

# 操作步骤

1. 登录 TIC 控制台。
2. 在左侧菜单中，选择【资源列表】。
3. 左侧选择某个产品，右侧单击任一 资源名称，可以查看相关资源的参数信息、使用示例以及参考帮助，如下图所示：



**云主机实例** cloud_cvm_instance

参数 | 输出 | 代码示例

**基本类型**

| 参数 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- |
| disable_security_service | bool | 否 | 禁用安全服务 |
| password | string | 否 | 密码 |
| keep_image_login | bool | 否 | 保留镜像登录 |
| system_disk_type | string | 否 | 系统盘类型 |
| user_data | string | 否 | 用户数据，base64编码 |
| key_name | string | 否 | 密钥名称 |
| instance_count | int | 否 | 实例数量 |

**云主机实例** cloud_cvm_instance

参数 | 输出 | 代码示例

复制代码

```
data "cloud_cvm_images" "my_favorite_image" {
  image_type = ["PUBLIC_IMAGE"]
  os_name    = "CentOs"
}

data "cloud_cvm_instance_types" "my_favorite_instance_types" {
  filter {
    name   = "instance-family"
    values = ["HUZI345"]
  }

  cpu_core_count = 1
  memory_size    = 1
}

data "cloud_availability_zones" "my_favorite_zones" {
}

// Create VPC resource

resource "cloud_vpc" "app" {
  cidr_block = "10.0.0.0/16"
  name       = "awesome_app_vpc"
}

resource "cloud_vpc_subnet" "app" {
  vpc_id            = cloud_vpc.app.id
  availability_zone = data.cloud_availability_zones.my_favorite_zones.zones.0.name
  name              = "awesome_app_subnet"
  cidr_block        = "10.0.1.0/24"
}

// Create 2 CVM instances to host awesome_app
```

# Provider使用文档
# 具体功能项

## provider功能

| 序号 | 示例编号 | 功能 | 详细说明 |
| --- | --- | --- | --- |
| 1 | TTIC_F001_E001 | 私有网络 vpc 示例 | 新建私有网络 vpc 示例 |
| 2 | TTIC_F001_E002 | 新建网络子网示例 | 新建私有网络子网示例 |
| 3 | TTIC_F001_E003 | 新建 cvm 虚拟机示例 | 新建 cvm 虚拟机示例 |
| 4 | TTIC_F001_E004 | 新建 cbs 磁盘示例 | 新建 cbs 磁盘示例 |

# CLI 使用方式
# 安装二进制包

## 安装Terraform

下载安装包

前往 Terraform 官网，使用命令行直接安装 Terraform 或下载二进制安装文件。

## 解压并配置全局路径

Linux/MAC：export PATH=$PATH:${可执行文件所在目录}。

Windows：可执行文件所在目录添加到系统环境变量 PATH 中。

## 验证是否安装成功

执行以下命令，查看是否安装成功。

```
terraform  -version
```

返回信息如下所示（版本号可能存在差异），则表示安装成功。

```
Terraform v1.3.0
on darwin_amd64

Your version of Terraform is out of date! The latest version
is 1.3.2. You can update by downloading from https://www.terraform.io/downloads.html
```

# 认证和鉴权

# 凭证获取

在首次使用Terraform 之前，请前往云 API 密钥页面申请安全凭证 SecretId 和 SecretKey。若已有可使用的安全凭证，则跳过该步骤。

1. 登录访问管理控制台，在左侧导航栏，选择访问密钥 > API 密钥管理。
2. 在 API 密钥管理页面，单击新建密钥，即可以创建一对 SecretId/SecretKey。

# 环境变量鉴权

请将如下信息添加至环境变量配置：
YOUR_SECRET_ID 及 YOUR_SECRET_KEY 请替换为获取凭证 中的 SecretId 和 SecretKey。

| | |
|---|---|
| export TENCENTCLOUD_DOMAIN=api3.yfm18.tcepoc.fsphere.cn | 导入主域名 |
| export TENCENTCLOUD_PROTOCOL=HTTP | |
| export TENCENTCLOUD_REGION=city | 设置地域 |
| export TF_LOG=DEBUG | 开启 debug 模式 |
| export TF_LOG_PATH=./terraform.log | 设置 log 路径 |
| export TF_CLI_CONFIG_FILE=/Users/zzzzy.l/workspace/cvm/dev.tfrc | terraform 进程运行时读取的配置信息，配置内可以指定 provider.tf 里面定义的插件为本地文件 |
| export TENCENTCLOUD_SECRET_ID=YOUR_SECRET_ID | |
| export TENCENTCLOUD_SECRET_KEY=YOUR_SECRET_KEY | |

配置 dev.tfrc 文件，使用本地 provider，代码如下:

dev.tfrc

```
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.

 # 指定provider项目目录
 dev_overrides {
    "myorg/cloud" = "/Users/zzzzy.l/project/tce/terraform-provider-tencent"
 }
}
```

# 使用 Terraform 创建云平台资源（以 VPC 为例）

## 创建 provider.tf 文件

指定 provider 配置信息。文件内容如下：

```
terraform {
  required_providers {
    cloud = {
      source = "myorg/cloud"
    }
  }
}
```

## 创建 main.tf 文件，配置 Provider 并创建私有网络 VPC

文件内容如下：

```
main.tf

resource "cloud_vpc" "cloud_vpc-bppk" {
  cidr_block = "10.0.0.0/16"
  name       = "ci-temp-test-updated"
  tags = {
"test" = "test"
}
}
```

# 创建资源

执行以下命令，查看执行计划，显示将要创建的资源详情。

```
terraform plan
```

返回信息如下所示：

```
Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # cloud_vpc.cloud_vpc-bppk will be created
  + resource "cloud_vpc" "cloud_vpc-bppk" {
      + assistant_cidrs        = (known after apply)
      + cidr_block             = "10.0.0.0/16"
      + create_time            = (known after apply)
      + default_route_table_id = (known after apply)
      + dns_servers            = (known after apply)
      + docker_assistant_cidrs = (known after apply)
      + id                     = (known after apply)
      + is_default             = (known after apply)
      + is_multicast           = false
      + name                   = "ci-temp-test-updated"
      + tags                   = {
          + "test" = "test"
        }
      + vpc_id                 = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
```

执行以下命令，创建资源。

```
terrform apply
```

根据提示输入yes 创建资源，返回信息如下所示：

```
zzzzy.l@zzzzys-MacBook-Pro demo % terraform apply
╷
│ Warning: Provider development overrides are in effect
```

```
│
│ The following provider development overrides are set in the CLI
│ configuration:
│  - myorg/cloud in /Users/zzzzy.l/project/tce/terraform-provider-tencent
│
│ The behavior may therefore not match any released version of the provider and
│ applying changes may cause the state to become incompatible with published
│ releases.
╵

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # cloud_vpc.cloud_vpc-bppk will be created
  + resource "cloud_vpc" "cloud_vpc-bppk" {
      + assistant_cidrs        = (known after apply)
      + cidr_block             = "10.0.0.0/16"
      + create_time            = (known after apply)
      + default_route_table_id = (known after apply)
      + dns_servers            = (known after apply)
      + docker_assistant_cidrs = (known after apply)
      + id                     = (known after apply)
      + is_default             = (known after apply)
      + is_multicast           = false
      + name                   = "ci-temp-test-updated"
      + tags                   = {
          + "test" = "test"
        }
      + vpc_id                 = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

cloud_vpc.cloud_vpc-bppk: Creating...
cloud_vpc.cloud_vpc-bppk: Creation complete after 3s [id=vpc-4lpij1lx]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

执行完毕后，您可以在云平台控制台查看创建的资源。

# 销毁资源

执行以下命令，销毁资源。

```
terraform destroy
```

返回信息如下所示：

```
zzzzy.l@zzzzys-MacBook-Pro demo % terraform destroy
╷
│ Warning: Provider development overrides are in effect
│
│ The following provider development overrides are set in the CLI
│ configuration:
│  - myorg/cloud in /Users/zzzzy.l/project/tce/terraform-provider-tencent
│
│ The behavior may therefore not match any released version of the provider and
│ applying changes may cause the state to become incompatible with published
│ releases.
╵
cloud_vpc.cloud_vpc-bppk: Refreshing state... [id=vpc-4lpij1lx]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # cloud_vpc.cloud_vpc-bppk will be destroyed
  - resource "cloud_vpc" "cloud_vpc-bppk" {
      - assistant_cidrs       = [] -> null
      - cidr_block            = "10.0.0.0/16" -> null
      - create_time           = "2024-02-01 14:52:08" -> null
      - default_route_table_id = "rtb-0xbqrr6u" -> null
      - dns_servers           = [
          - "183.60.82.98",
          - "183.60.83.19",
        ] -> null
      - docker_assistant_cidrs = [] -> null
      - id                    = "vpc-4lpij1lx" -> null
      - is_default            = false -> null
      - is_multicast          = false -> null
```

```
    - name                 = "ci-temp-test-updated" -> null
    - tags                 = {
        - "test" = "test"
      } -> null
    - vpc_id               = "vpc-4lpij1lx" -> null
  }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

cloud_vpc.cloud_vpc-bppk: Destroying... [id=vpc-4lpij1lx]
cloud_vpc.cloud_vpc-bppk: Destruction complete after 3s

Destroy complete! Resources: 1 destroyed.
```

# tf 代码示例
# 私有网络 vpc 示例

| | |
|---|---|
| 示例编号 | TTIC_F001_E001 |
| 优先级 | P0 |
| 是否验收 | 是 |
| 示例名称 | 新建私有网络vpc示例 |
| 预置条件 | 上述环境变量配置完成 |
| 代码示例 | // Create VPC resource<br>resource "cloud_vpc" "app" {<br>cidr_block = "10.0.0.0/16"<br>name = "awesome_app_vpc"<br>} |

# 新建网络子网示例

| | |
|---|---|
| 示例编号 | TTIC_F001_E002 |
| 优先级 | P0 |
| 是否验收 | 是 |
| 示例名称 | 新建私有网络子网示例 |
| 预置条件 | 上述环境变量配置完成 |
| 代码示例 | data "cloud_availability_zones" "my_favorite_zones" {<br>}<br>resource "cloud_vpc_subnet" "app" {<br>vpc_id = cloud_vpc.app.id<br>availability_zone = data.cloud_availability_zones.my_favorite_zones.zones.0.name<br>name = "awesome_app_subnet"<br>cidr_block = "10.0.1.0/24"<br>} |

# 新建 cvm 虚拟机示例

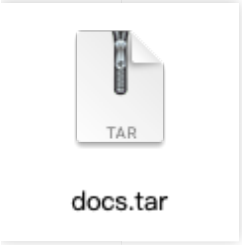
| | |
|---|---|
| 示例编号 | TTIC_F001_E003 |
| 优先级 | P0 |
| 是否验收 | 是 |
| 示例名称 | 新建cvm虚拟机示例 |
| 预置条件 | 上述环境变量配置完成 |
| 代码示例 | // Create 2 CVM instances to host awesome_app<br>resource "cloud_cvm_instance" "my_awesome_app" {<br>instance_name = "awesome_app"<br>availability_zone = data.cloud_availability_zones.my_favorite_zones.zones.0.name<br>image_id = data.cloud_cvm_images.my_favorite_image.images.0.image_id<br>instance_type =<br>data.cloud_cvm_instance_types.my_favorite_instance_types.instance_types.0.instance_type<br>system_disk_type = "CLOUD_PREMIUM"<br>system_disk_size = 50<br>hostname = "user"<br>project_id = 0<br>vpc_id = cloud_vpc.app.id<br>subnet_id = cloud_vpc_subnet.app.id<br>count = 2<br><br>data_disks {<br>data_disk_type = "CLOUD_PREMIUM"<br>data_disk_size = 50<br>}<br><br>tags = {<br>tagKey = "tagValue"<br>}<br>} |

# 新建 cbs 磁盘示例

| | |
|---|---|
| 示例编号 | TTIC_F001_E004 |
| 优先级 | P0 |
| 是否验收 | 是 |
| 示例名称 | 新建cbs磁盘示例 |
| 预置条件 | 上述环境变量配置完成 |
| 代码示例 | resource "cloud_cbs_storage" "storage" {<br>storage_name = "mystorage"<br>storage_type = "CLOUD_SSD"<br>storage_size = 100<br>availability_zone = "ap-city1-3"<br>project_id = 0<br>encrypt = false<br><br>tags = {<br>test = "tf"<br>}<br>} |

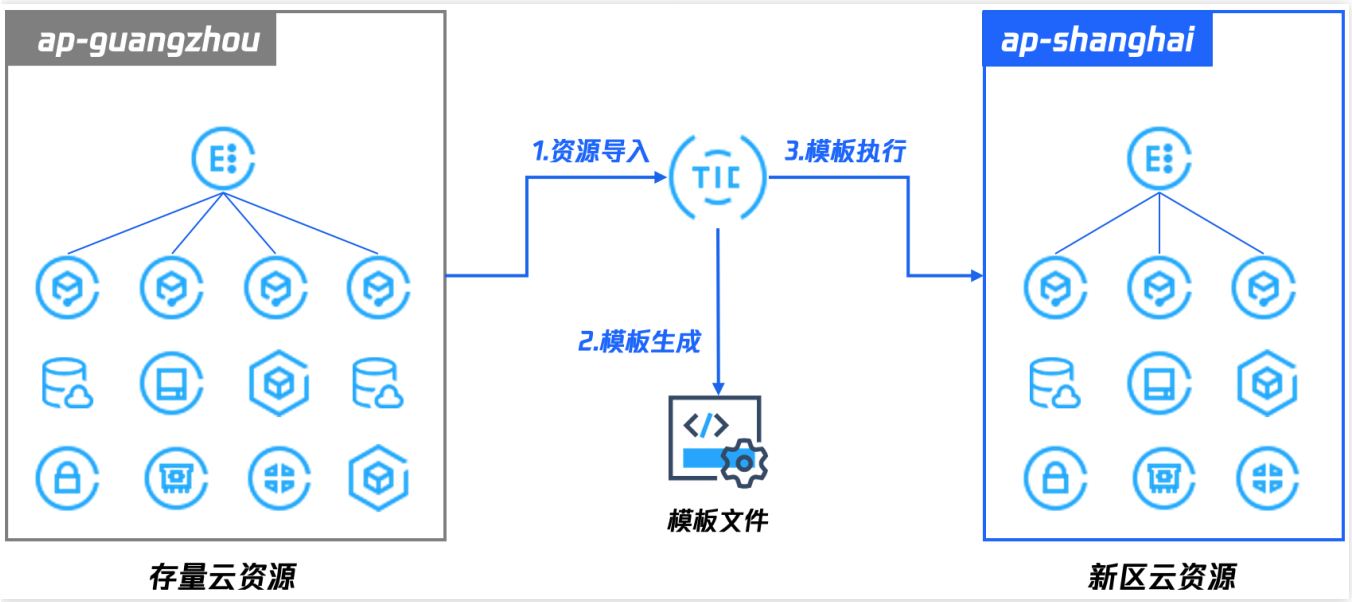
# 附录

使用 tar -xvf docs.tar 进行解压。



# 最佳实践
# 云资源跨区复制

通常情况下，如果将某个地域下云资源复制到新的地域，需要频繁操作控制台，来回确认资源参数细节，操作过程耗时费力，还容易出错。而基于资源编排 TIC 的【资源导入】能力，只需要三步，花费几分钟的时间，就可以快速完成云资源跨区域复制。如下为操作流程图：




步骤1：资源导入

1. 登录 资源编排 TIC 控制台。

2. 单击【新建资源栈】，选中当前资源所在的地域，例如城市2（ap-city2）。

3. 选择模板，选中【导入资源】，按标签导入资源。如果资源较多，强烈建议按标签方式来导入。若资源未设置标签，可以在控制台中为相关实例（例如云服务器、安全组、私有网络、子网、路由表、负载均衡实例）设置标签。

4. 单击【确定】，TIC 会自动选中【标签键】、【标签值】下的云资源实例。

5. 单击【下一步】，TIC 将根据上一步选中的资源列表，自动生成配置代码。

6. 单击【导入完成】，跳转到配置代码详情页。

从上述导出的代码可以看到，系统默认会导入 CLB 负载均衡器下的 listener 和 rule 配置。无需编辑配置代码，单击【预览】，执行导入资源预览操作。确认预览结果，对于资源导入场景，[std] No changes. Infrastructure is up-to-date. 表示不涉及现网资源变更，符合预期。

> 注意：
>
> 如果预览显示有资源变更，可能是因为手动修改了配置代码。需要再次强调的是，对于通过 TIC 导入资源功能，复制资源配置的场景，无需编辑系统自动生成的代码，如果未显示：[std] No changes. Infrastructure is up-to-date. 提示字段，请先联系 TIC 团队 进行确认后，再执行后续的操作，否则后续步骤会涉及到现网资源的配置变更。

7. 单击【发布】，同步更新现网资源配置，完成资源栈配置。

8. 确认资源栈创建成功后（【事件】页面显示 APPLY_COMPLETED 状态），请执行 步骤2：保存模板。
   步骤2：保存模板
   在步骤1中，我们通过资源导入自动生成了现网资源的配置代码，下一步我们需要将自动生成的配置代码，保存到模板，用于在新地域创建相关资源。

9. 单击步骤1创建的资源栈 ID，进入到【资源栈版本】页面，在其右侧操作一列中选择【更多】>【保存为模板】。

10. 输入“模板名称”和“描述”后，单击【确定】完成模板的创建。模板创建成功，将跳转到【我的模板】页面。

步骤3：通过模板创建资源

1. 进入资源编排控制台 我的模板 页面。

2. 找到在步骤2中创建的模板 import-template，在其右侧操作一列中单击【部署】，创建新的资源栈。

3. 选择新建资源所在的地域，例如城市1（ap-city1）。

4. 单击【下一步】，编辑 variables.tf、clb.tf 配置文件，将相关地域配置信息，调整成新地域简称或可用区简称，例如ap-city1。
   variables.tf 配置文件修改如下：

   clb.tf 配置文件修改如下：

除了地域调整以外，还需要针对导入冗余信息进行调整，例如云服务器实例绑定的系统盘 ID。

**（可选）**如果需要对实例配置做调整，也可以编辑对应的参数值进行调整。例如调整实例的内网 IP。

1. 单击【预览】，进行代码语法校验，并预览资源配置。预览结果为 [std] Plan: 11 to add, 0 to change, 0 to destroy.，则符合预期。
2. 单击【发布】，再次确认后完成资源栈创建，TIC 将根据资源栈的配置，新建相关云资源。
3. 在资源栈【事件】页面中，可以看资源创建状态（APPLY_IN_PROGRESS 表示资源创建中）。
4. 刷新事件状态，当状态为 APPLY_COMPLETED 时，资源创建成功。在资源栈【资源】页面中，可以看到在城市1（ap-city1）地域创建的资源列表。

经过了资源导入、模板保存、新建资源栈的三步操作后，我们完成了将城市2地域中的云资源（CLB 负载均衡器以及后端绑定的云服务器、私有网络、子网、安全组）拷贝到城市1地域的操作，实现了快速搭建游戏新区、备份中心的诉求。

# 常见问题
# 常见问题

如何获取 API 密钥？
您可以前往 API 密钥管理 获取 API 密钥。

什么是资源栈？
资源栈是云服务的有机集合，TIC 通过资源栈的概念来管理云上资源。

TIC 是否支持第三方代码库？
目前不支持，后续将支持 Github。

如何使用 TIC 创建第三方云资源？
目前 TIC 的自动化流程默认仅支持云平台，后续将陆续支持主流的云服务商。当然，您也可以在代码编辑页面增加新的 Provider、API 密钥和地域信息来创建第三方云资源。

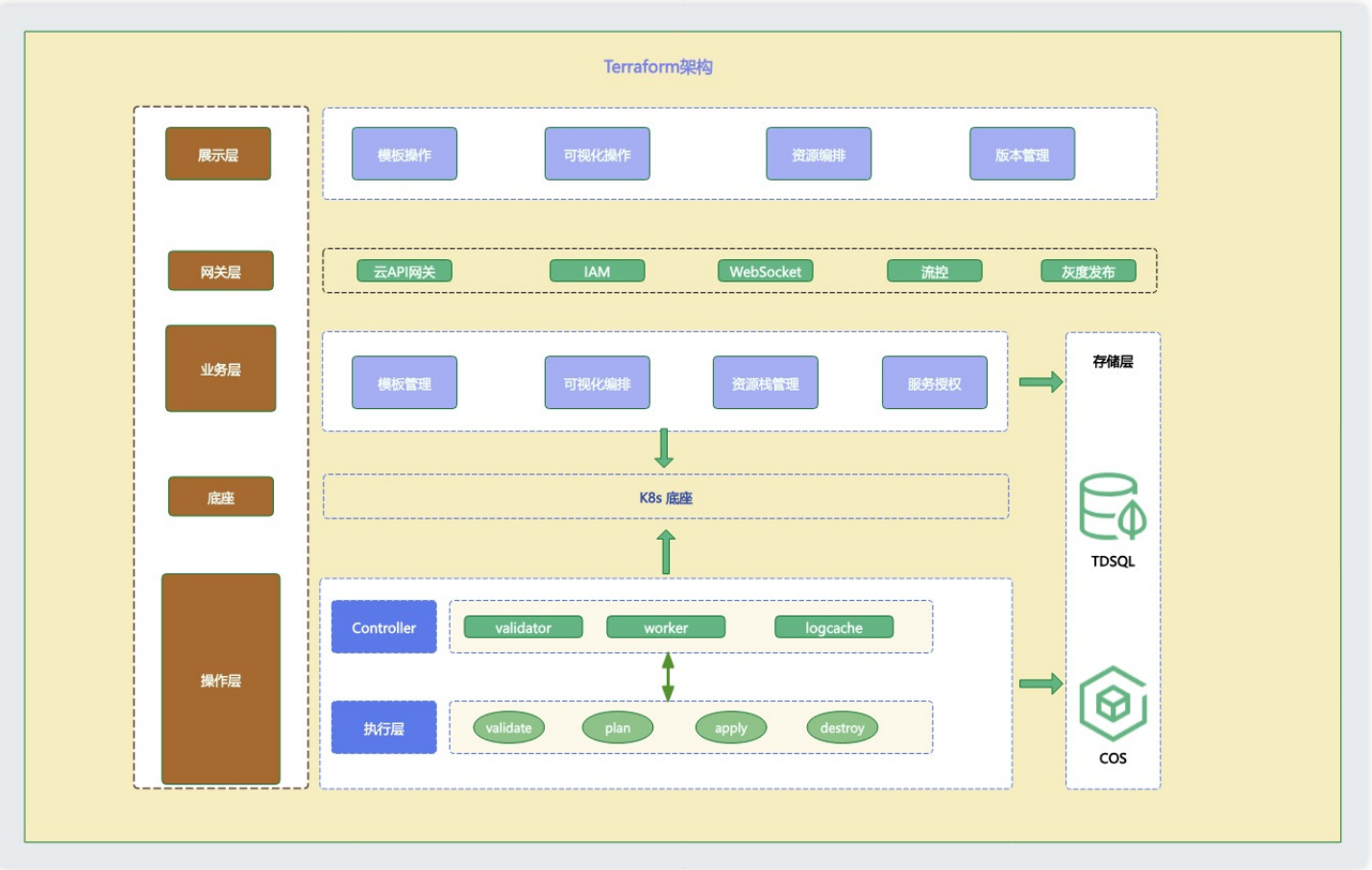
通过控制台修改 TIC 创建的云资源属性，TIC 是否能自动同步相关代码？
不能。TIC 目前无法实现与控制台资源属性的自动化同步，只能手动来修改对应资源在代码中的参数来实现。

# 运维管理指南
# 产品架构
# 核心功能组件

| 组件ID | 组件功能 |
| --- | --- |
| tcloud-tic-terraform-operator | 资源编排服务控制层，用于将terraform crd转换成job进行执行 |
| image-terraform-executor | terraform执行层镜像，负责推送执行层镜像，同时在configmap中记录镜像信息 |
| tcloud-tic-cgi-server | api交互层，负责与前端数据交互 |
| product-tic-preset | tic生产组件，负责将图标、模板等静态资源上传到cos存储桶 |

# 产品架构图

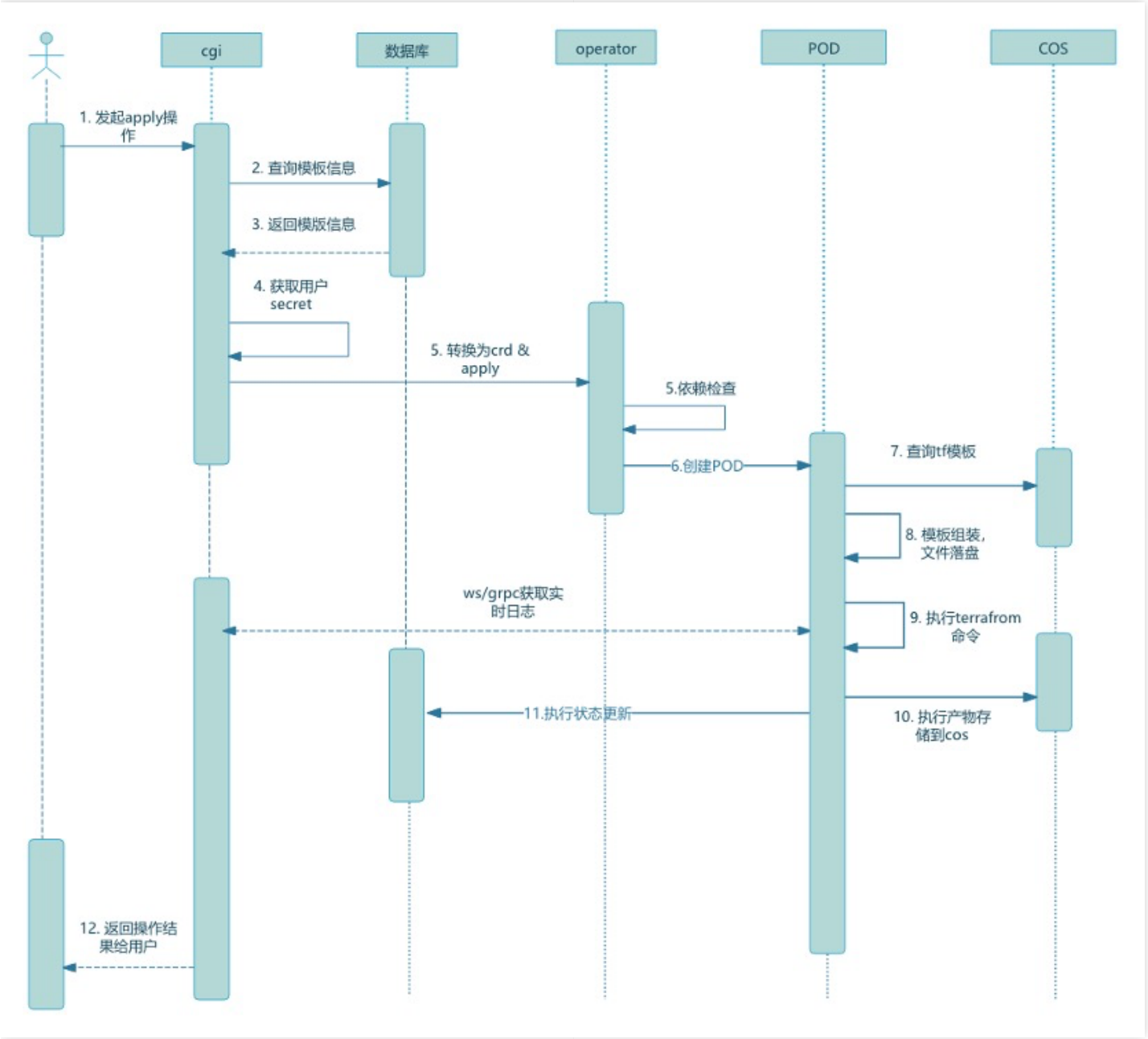
产品主要的业务逻辑图，程序架构图，能够快速一览其在TCE中的全局位置，上下游链路。



# 部署架构图

TIC产品为global级别服务，按照通用global级别应用架构部署即可。

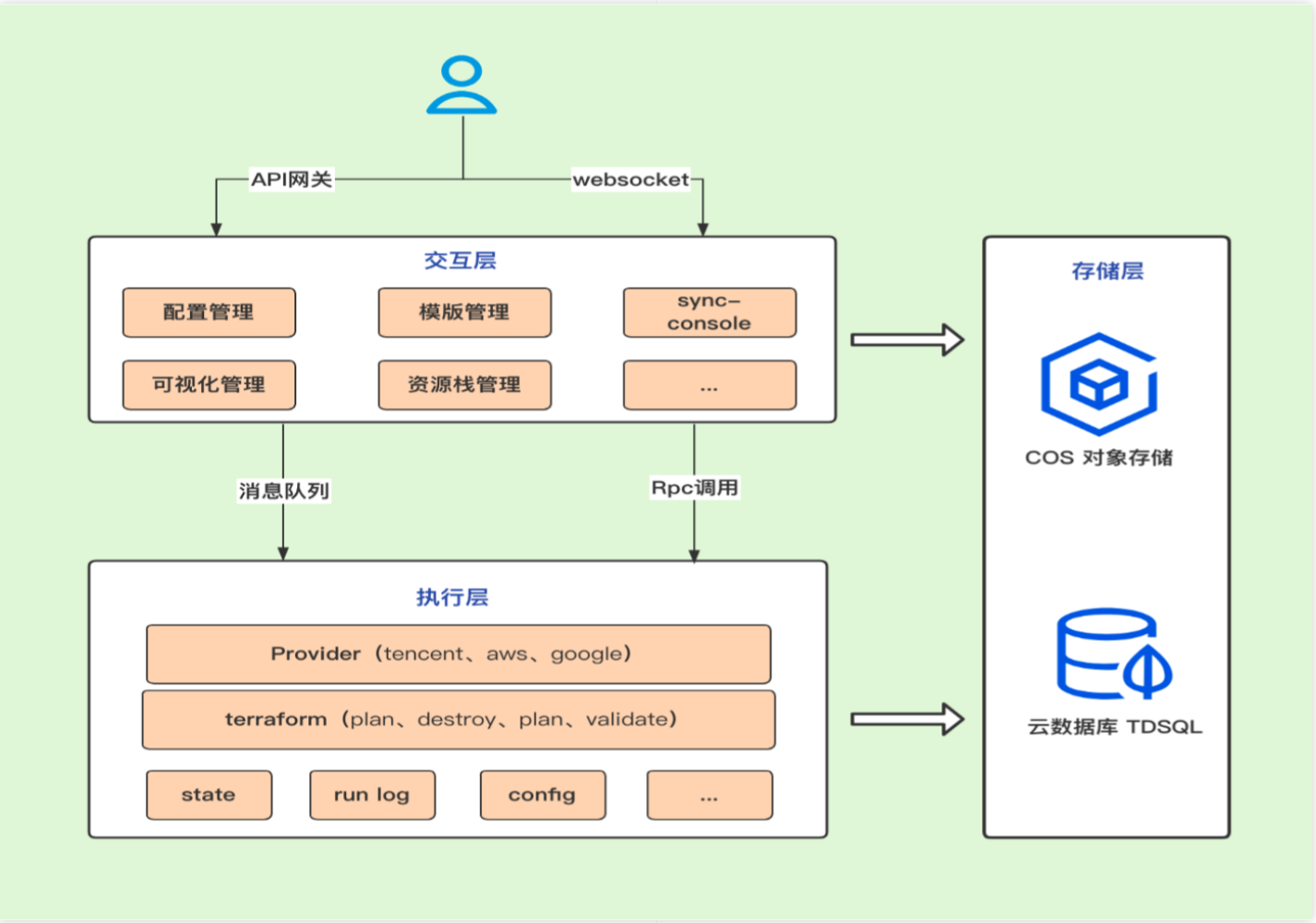
# 业务流向图




核心访问链路：用户通过前端创建模板，点击apply，cgi-server转换成对应crd，operator监听crd，创建pod执行具体任务，任务执行结果会持久化到cos和数据库，并通过websocket协议，同步显示到前端。

# 核心逻辑概述

TIC资源编排基于备受欢迎的基础设施即代码（Infrastructure as Code）工具 Terraform 插件扩展体系下，提供云资源部署、实施和管理的自动化能力。



# 产品依赖

| 依赖产品名称 | 依赖类型 | 依赖关系描述 | 影响程度 | 备注 |
| --- | --- | --- | --- | --- |
| 支撑tdsql 22001实例，iac-service库 | 数据库 | dbsql-tic-service，存储应用模板、版本等信息 | 高 | |
| 支撑csp iaconline存储桶 | 对象存储 | 静态图标模板需要上传到csp | 高 | |
| Tcenter基础平台location服务 | http接口 | 依赖DescribeLocationZone接口获取地域和已部署产品信息 | 高 | |

# 故障处理

# 故障影响范围

| 序号 | 场景 | 对云产品数据面影响 | 对平台控制面影响 |
| --- | --- | --- | --- |
| 1 | websocket服务无法连接 | 无影响 | 无影响 |
| 2 | 创建资源栈时无法选择地域 | 无影响 | 无影响 |
| 3 | 编排的时候报错 | 无影响 | 无影响 |

# 故障恢复
# 场景1：websocket服务无法连接

## 故障现象

执行terraform事件时，无法与服务器建立web socket连接，实时输出编排的控制台内容。

## 故障影响

高。

## 故障定位

websocket服务无法连接，可能的原因很多种，可能是网络问题（比如dns，clb配置问题），还有后端服务的问题。由于云API不支持websocket协议，前端的websocket的流量原封不动地转发到cgi-server pod中，因此需要检查websocket服务是否正常运行。

进入cgi-server pod, 查看websocket启动端口。

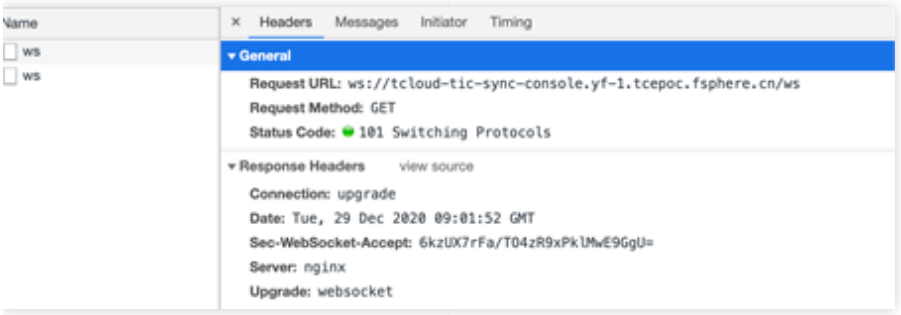
```
[root@tcloud-tic-cgi-server-65f4cb85b7-9cv8m cgi_server]# cat conf.yaml
Debug: true
CgiAddress: 0.0.0.0:80
Prometheus: 0.0.0.0:7100
ConsoleWebSockAddress: 0.0.0.0:7001


MysqlConnect:
 default: 595x4sn8hw:AES+V1+ccc332581d8746037ad9c0ed584e6709@(10.27.205.54:%!d(json.Number=22001))/iac-service?charset=utf8&timeout=5s

FileLogConfig: '{"filename":"/data/log/cgi_server.log","daily":true,"maxsize":4094967296}'
ESLogConfig: 'none'


#################
#qcloud sdk(腾讯云api使用的aksk)
#################
QCloudConfig:
  AccessKey:  AKID6HuwxN8hwArl9YfzF50b71GqXGUTXORN
  SecretKey: YVWwHMI8EhQSKw60dOhjyWnSSQo5YMpF
  Region: ap-shenzhen-region-jcctest-ops

#################
#临时密钥api使用的aksk
#################
TemporaryKeyConfig:
  AccessKey: AKIDgsyvS2q0YpD1hArtcFpJQBDHefx11XEc
  SecretKey: cFYcC5fkVeILJBtsuTEOp8U2WBkpIYPH
  Region: ap-shenzhen-region-jcctest-ops # todo: no such field in cam...
```

检查websocket端口服务是否正常运行。

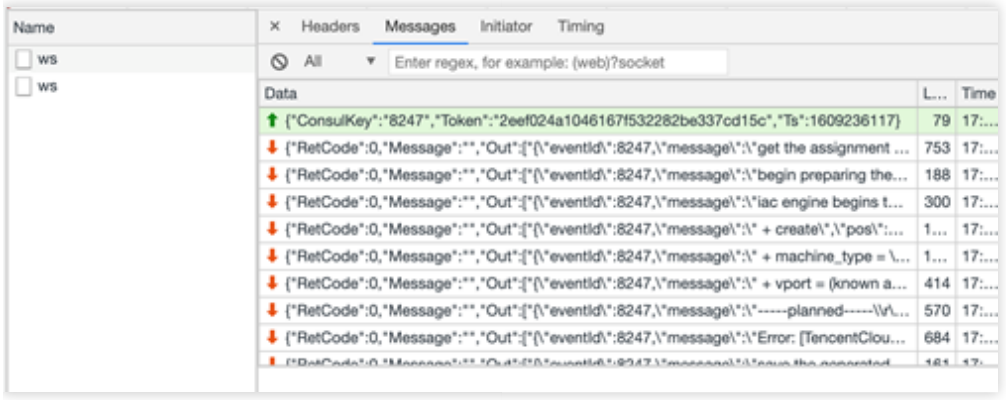
```
[root@tcloud-tic-cgi-server-65f4cb85b7-9cv8m cgi_server]# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp6       0      0 :::80                   :::*                    LISTEN      1/./cgi_server
tcp6       0      0 :::7001                 :::*                    LISTEN      1/./cgi_server
tcp6       0      0 :::7002                 :::*                    LISTEN      1/./cgi_server
tcp6       0      0 :::15535                :::*                    LISTEN      1/./cgi_server
[root@tcloud-tic-cgi-server-65f4cb85b7-9cv8m cgi_server]#
```

可以在浏览器上查看websocket的连接状态，比如说打开chrome的网络面板后，执行apply事件，我们可以看到

websocket的握手协议：




客户端发送了一个HTTP GET请求，服务返回101 Switching Protocols时即表示握手协议建立成功，已经从HTTP协议升级到了Websocket协议。然后点开Messages tab，就能够看到所有的websocket数据了，在定位问题的时候很有用：




## 规避措施（可选）

在tce环境初始化的时候，检查cgi-server的websocket服务是否正常启动，否则TIC将无法看到TIC实时编排日志信息。

# 场景2：创建资源栈时无法选择地域

## 故障现象

无法选择地域，导致没法进入下一步对资源进行编排。

## 故障影响

高。

## 故障定位

选择地域是tic自己的接口，所以问题应该定位到cgi的server，查看云api的返回发现no region found，于是我们查看一下数据库：

```
mysql> select * from region;
+-----------+-------------+---------------------+--------------+-----------+
| region    | region_name | english_region_name | provider     | last_time |
+-----------+-------------+---------------------+--------------+-----------+
| chongqing | chongqing   | chongqing           | tencentcloud |        -1 |
+-----------+-------------+---------------------+--------------+-----------+
1 row in set (0.00 sec)
```

如果数据库里没有对应的tce region信息，则需要填上，否则就无法选择区域了。

## 规避措施（可选）

在tce环境初始化的时候，将区域信息填入mysql的region表中，注意这个region一定是部署可用的，其他同一区域的云产品也应该用这个区域。

# 场景3：编排的时候报错（通用性问题）

## 故障现象

对资源执行编排apply的时候，可能是用户提供的hcl文件的问题导致的错误，也有可能是tic服务本身的问题，更有可能的是tic依赖的云API导致的问题。下面我们就这些错误一一说明一下该如何处理。

## 故障影响

高。

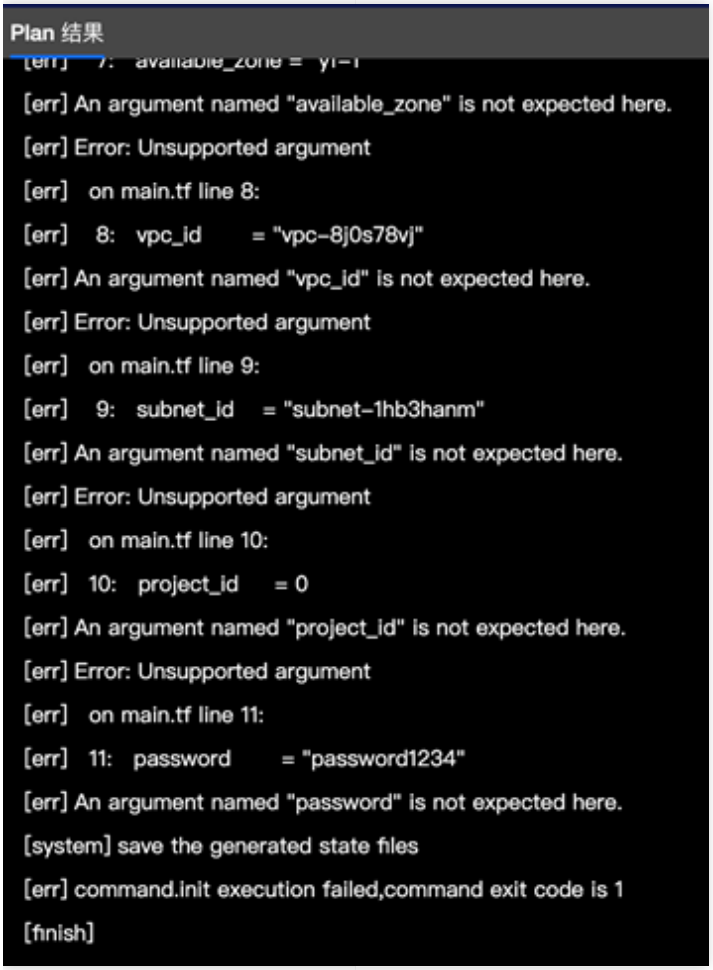
## 故障定位

如果用户提交的terraform模版文件本身语法有问题，那么在执行plan的时候就会报错。需要注意的是，语法没有问题不代表就可以编排，还需要在apply运行时才能知道。
比如说我们提交一个语法不合规的模版：

main.tf

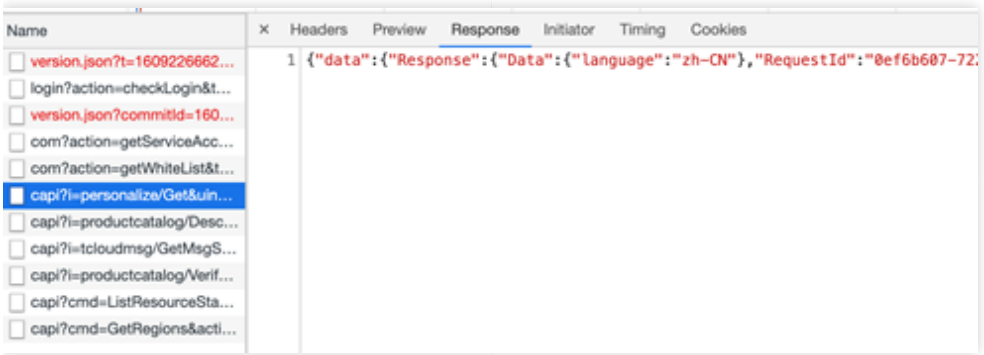
```
resource "tencentcloud_mongodb_instance" "mongodb"
  instance_name  = "mongodb-tce"
  memory         = 4
  volume         = 100
  engine_version = "MONGO_3_WT"
  machine_type   = "GIO"
  available_zone = "yf-1"
  vpc_id         = "vpc-8j0s78vj"
  subnet_id      = "subnet-1hb3hanm"
  project_id     = 0
  password       = "password1234"
}
```

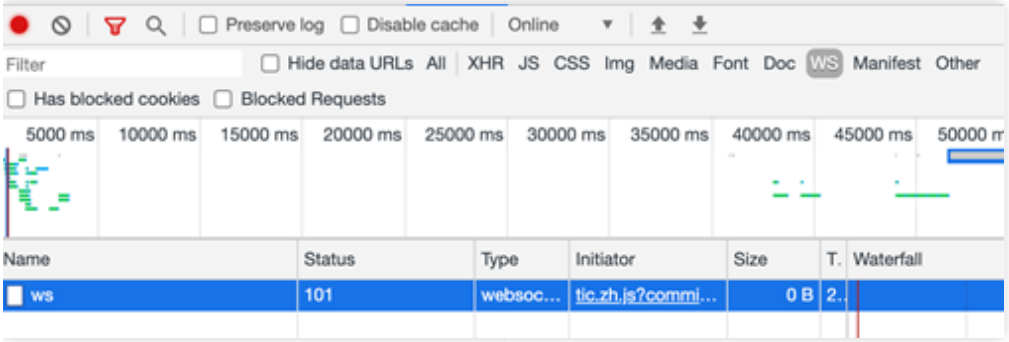
可以看到少了一个"}"符号，执行plan的时候tic就会报错，此时就需要用户修改hcl模版文件：




如果plan事件无法执行（apply，destroy事件同理），那么TIC的后台服务就出问题了，就需要去依次检查一下cgi网关和websocket服务是否正常，terraform operator、executor服务是否已启动，基础组件（数据库、csp等）是否运行正常，以及组件间依赖的tce配置是否正确。

cgi网关提供TIC的API服务，可以在浏览器F12查看网络面板得到请求响应的信息，还可以看到websocket的连接状态和双向数据传输。



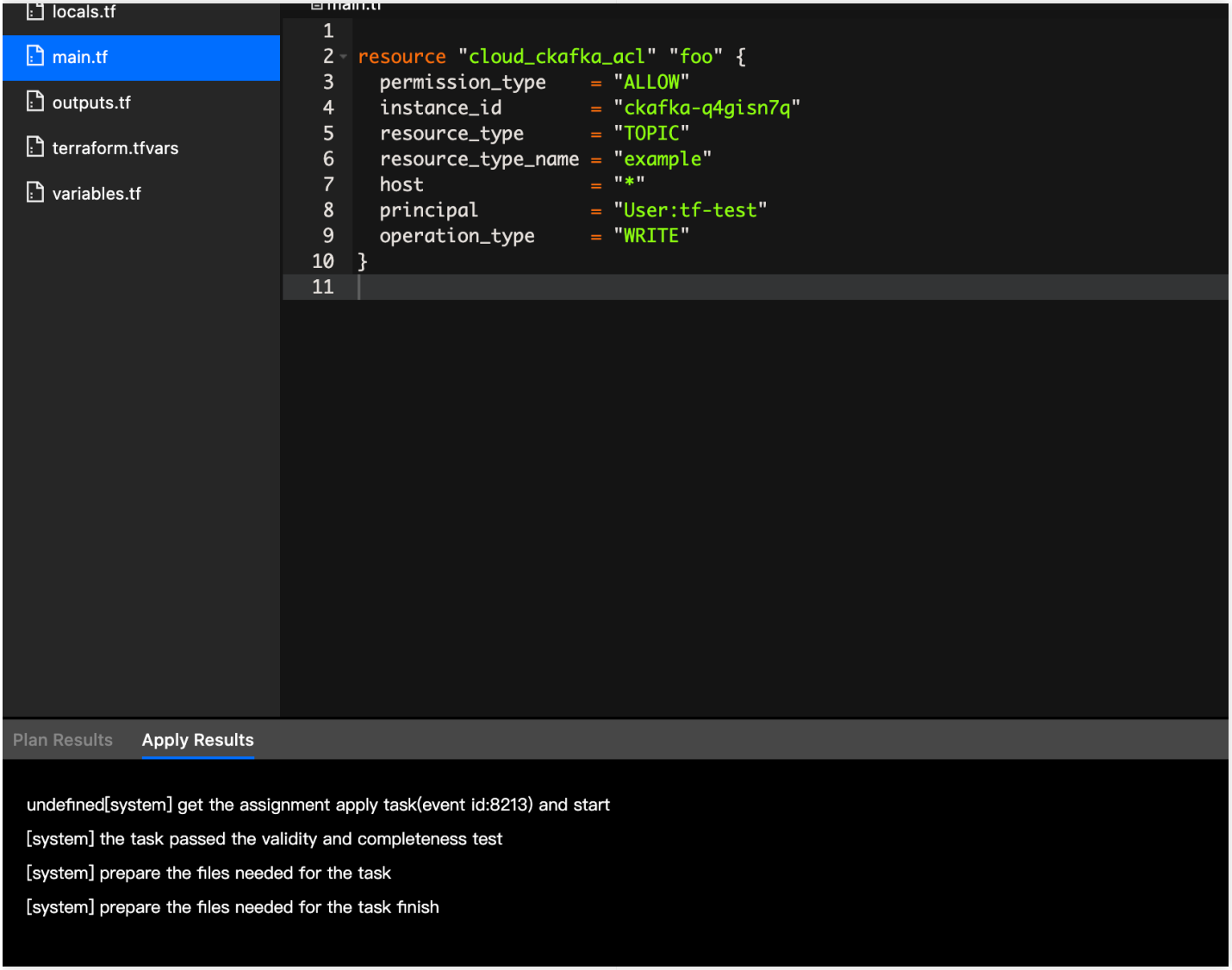



这里可以看看是否是客户端的请求问题，如果是的话，根据错误信息调整再试试（也有可能网络出了问题，比如网关服务，这时候只能去找管理员了）。如果不是的话，就是后台问题了，最容易想到的办法就是根据云API的requestid去服务器排查日志，但是我们需要先看看服务是否正常，首先登录服务器。

kubectl get pod -ntce |grep tic

这两个pod必须处于running状态，否则需要查看pod的事件排查没起来的原因然后重启pod。pod起不来的原因很有可能是依赖的基础组件出了问题，根据启动日志逐一排查。

更多的是依赖的云产品的API出了问题，根据前端报错的requestid，使用查询脚本定位到云产品具体报错原因。




[err] Error: [TCECloudSDKError] Code=InvalidParameter.CheckParamNotPass, Message=CreatePGInstanceHour, create master, but have no standby nodes, RequestId=19e23639-eedd-e7f5-4bcf-4e51ff1ca9b2

使用 bash searchlog.sh 19e23639-eedd-e7f5-4bcf-4e51ff1ca9b2 定位具体问题

# 日常巡检

## 日常巡检列表

| 编号 | 巡检项 | 巡检说明 |
| --- | --- | --- |
| 1 | cgi-server pod running | 对服务域名和端口进行curl，验证服务是否正常。 |
| 2 | cloud-tic-terraform-operator pod running | 检查operator是否正常运行。 |
| 3 | 服务组件可用性巡检 | 查看TIC各组件是否正常运行。 |
| 4 | 数据库巡检 | 检查iac-service数据库是否正常。 |

# 巡检处理
# 巡检项1：检查TIC控制台

登录租户端控制台，查看资源栈列表是否正常。

## 前提条件

已获取租户端的账号。

## 巡检步骤

步骤1.登录租户端控制台，查看资源栈列表。
步骤2.查看资源站详细信息。

## 巡检结果

正常：页面无报错，可以正常显示资源栈列表和模版管理列表。
错误：页面有报错，或无模版，资源栈数据。

## 异常处理

按照运维手册中故障处理章节进行排查与处理。

# 巡检项2：检查COS状态

## 前提条件

步骤1.已获取部署机器的IP和用户名及登录密码。
步骤2.已获取租户端的账号密码。

## 巡检步骤

步骤1.登录部署机器，查看COS的域名是否能正确解析并且端口是通的。
步骤2.登录租户端，进入对象存储产品，尝试给存储桶上传文件。

## 巡检结果

正常：DNS能够正确解析，COS成功上传文件。
错误：DNS无法正确解析或者COS上传文件失败。

## 异常处理

若COS的域名无法正常解析，检查一下DNS的配置是否是对的，查看一下/etc/resolv.conf文件。如果在租户端无法上传文件，请联系COS对应的运维人员。

# 巡检项3：检查terraform和插件的状态

## 前提条件

登录到k8s master节点。

## 巡检步骤

步骤1. kubectl get cm -ntce image-terraform-executor -oyaml ，执行命令发现执行层镜像正常显示。

```
[root@tcs-10-29-0-141 ~]# kubectl get cm -ntce image-terraform-executor -oyaml
apiVersion: v1
data:
  image: registry.tce.com/image-terraform-executor/image-terraform-executor:3.10.11-20241106-113125-8f9e69c.rhel.amd64
kind: ConfigMap
metadata:
  annotations:
    infra.tce.io/last-applied-definition: 602c440b6f434f60d2d503ddd9efa2cc
  creationTimestamp: "2024-08-30T06:36:40Z"
  labels:
    dawn.infra.tce.io/application-instance-name: image-terraform-executor
    dawn.infra.tce.io/application-instance-uuid: ad64ec392b13595fb4656352108a6dfb
    dawn.infra.tce.io/application-name: image-terraform-executor
    dawn.infra.tce.io/application-version: 3.10.11-20241106-113125-8f9e69c
    dawn.infra.tce.io/product-instance-id: tic-1
    dawn.infra.tce.io/region-id: "80000052"
    dawn.infra.tce.io/region-name: ap-shenzhen-region-jcctest-ops
    infra.tce.io/app-version: 3.10.11-20241106-113125-8f9e69c.rhel.amd64
    infra.tce.io/chart-name: image-terraform-executor
    infra.tce.io/managed-id: 66ea4052-8a3d-445d-ad40-a1d9e1a4e3fc
    infra.tce.io/oam-app: image-terraform-executor
    infra.tce.io/oam-comp: image-terraform-executor
    infra.tce.io/oam-product: tic
  name: image-terraform-executor
  namespace: tce
  ownerReferences:
  - apiVersion: infra.tce.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: Application
    name: image-terraform-executor
    uid: 6b853412-c2b1-4c16-a9a0-e33923e50634
  resourceVersion: "494183463"
  selfLink: /api/v1/namespaces/tce/configmaps/image-terraform-executor
  uid: ec0a3cda-e881-4fa1-80f7-5eef0b7ace99
```

步骤2.执行 docker run -it registry.tce.com/image-terraform-executor/image-terraform-executor:3.10.11-20241106-113125-8f9e69c.rhel.amd64 bash

步骤3.执行 tree . ，查看terraform、plan、apply、destroy、validate二进制文件是否正常。

```
root@tcs-10-29-0-141 ~]# docker run -it registry.tce.com/image-terraform-executor/image-terraform-executor:3.10.11-20241106-113125-8f9e69c.rhel.amd64 bash
root@5fc34e5a43f2 services]# ll
otal 63672
rwxr-xr-x 2 root root       10 Nov  6 11:36 conf
rwxr-xr-x 1 root root       36 Nov  6 11:37 executor
rwxr-xr-x 1 root root       31 Nov  6 11:36 registry.terraform.io
rwxrwxrwx 1 root root 65200128 Sep 26  2023 terraform
root@5fc34e5a43f2 services]# tree .

── conf
── executor
   ├── service_do_tf_apply
   ├── service_do_tf_destroy
   ├── service_do_tf_plan
   ├── service_do_tf_validate
   └── tf_op_service_switch.sh
── registry.terraform.io
   └── hashicorp
       └── cloud
           └── 1.18.5
               └── linux_amd64
                   └── terraform-provider-cloud
── terraform

 directories, 7 files
root@5fc34e5a43f2 services]#
```

## 巡检验证

Terraform镜像能够正常拉起，并且版本应该是v1.8.5。

## 异常处理

检查镜像物料包是否正常部署。

# 巡检项4：检查cgi pod的状态

## 前提条件

已获取tcs-master主机登录信息。

## 巡检步骤

步骤1.登录机器，执行命令 kubectl get pod -ntce |grep tic- ，查看服务是否是running状态：

">

```
tcloud-tic-cgi-server-6bc7d984f8-9v2j4                    1/1     Running            0          28m
tcloud-tic-cgi-server-6bc7d984f8-blsdm                    1/1     Running            0          28m
tcloud-tic-terraform-operator-5574f9f77f-88hzp            1/1     Running            51         8d
tcloud-tic-terraform-operator-5574f9f77f-tp6hp            1/1     Running            1          8d
```

## 巡检验证

正常：TIC相关的pod都处于running运行中状态。
异常：TIC相关的pod有非运行中的状态。

## 异常处理

查看异常pod的日志和事件信息，根据日志和事件信息进行处理。

## 运维经验

如果遇到异常pod，可以尝试重启pod。

# 日常监控
# Pod 自监控

当前 TIC 的两个镜像组件 tcloud-tic-cgi-server 和 tcloud-tic-terraform-operator 已经容器化部署到 Kubernetes 中，使用 liveness probe（存活探针）来确认何时重启容器，使用 readiness probe（就绪探针）来确定容器是否已经就绪接收外部流量提供服务。在 TCS 集群中存活探针和就绪探针的定义如下：

```
livenessProbe:
      exec:
        command:
        - bash
        - /tce/healthchk.sh
      failureThreshold: 3
      initialDelaySeconds: 30
      periodSeconds: 5
      successThreshold: 1
      timeoutSeconds: 5
    name: tcloud-tic-cgi-server
    ports:
    - containerPort: 80
      protocol: TCP
    readinessProbe:
      exec:
        command:
        - bash
        - /tce/healthchk.sh
      failureThreshold: 5
      initialDelaySeconds: 30
      periodSeconds: 5
      successThreshold: 1
      timeoutSeconds: 3
```

两个探针使用的都是脚本 /tce/healthchk.sh 进行检查，在不同镜像此脚本会根据镜像中的组件进行定义，如果组件运行正常则以0退出，否则以非0退出。借助存活探针和就绪探针 Kubernetes 集群保证业务请求的正常处理。