资源编排ROS

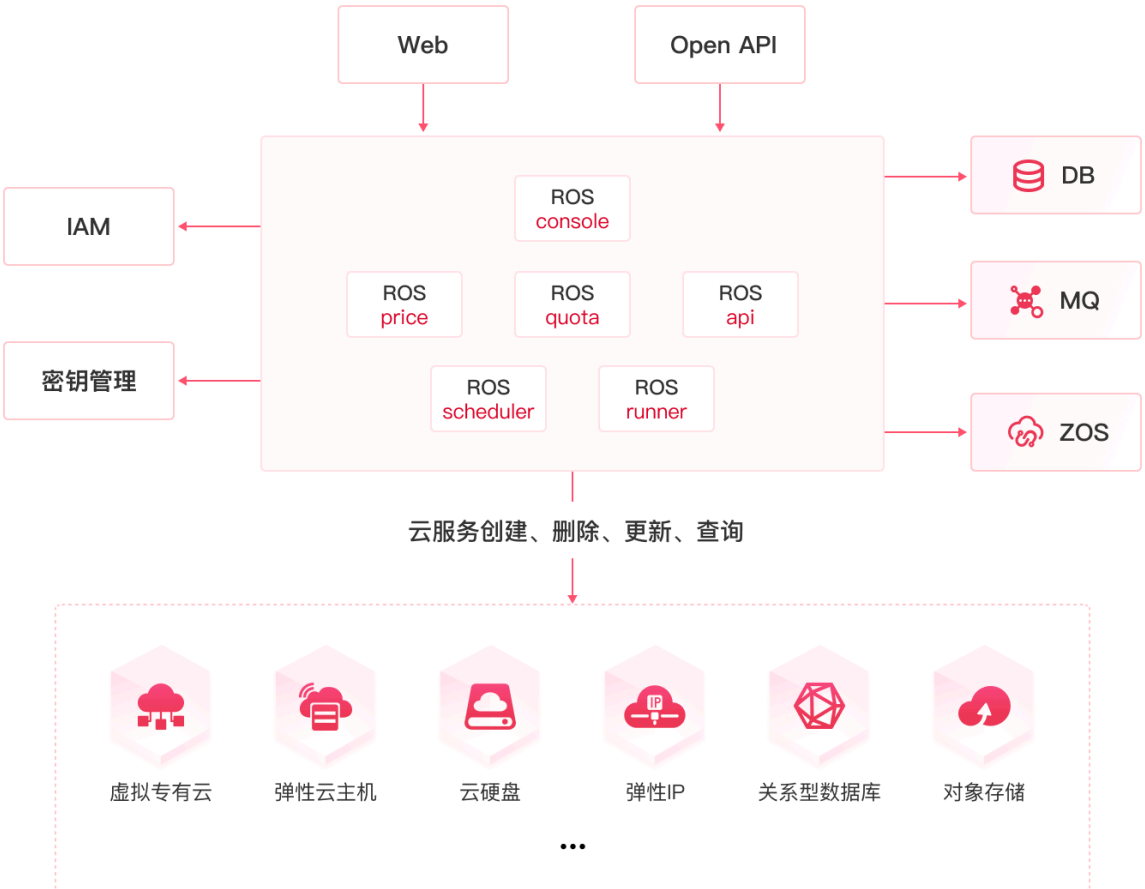
目录

目录

产品定义

资源编排ROS（ROS，Resource Orchestration Service）是基于Terraform（HCL + Provider）的新一代云资源全生命周期管理平台，您只需通过结构化模板集中定义各类云资源（例如弹性云主机、虚拟私有云、弹性IP等），即可自动、安全、高效地实现“一键创建与管理”云资源。借助 Terraform 的底层引擎能力，ROS 能够自动解析资源依赖关系、校验配置合规性，并通过预置的安全策略规避权限风险与配置冲突，最终实现从资源创建、关联配置到后期扩缩容、版本迭代的 “一键化” 操作。

资源编排ROS聚焦自动化批量创建云资源场景，全面践行“基础设施即代码”理念，帮助用户大幅降低人工操作失误率，同时支持资源的批量管理与自动化运维，让企业在复杂云架构下也能轻松实现资源的高效调度与精益化管理，助力用户高效、安全、一致性交付和运维天翼云服务。



为什么选择资源编排ROS

资源编排ROS可以帮助您高效、安全、轻松地管理一组资源。

强大的Terraform生态： 完全兼容Terraform，确保您的现有资源栈无缝迁移。

极简自动化部署： 通过模板化编排和自动化引擎，您只需“一次设计”即可“多次部署”。

安全合规： 内置评估审核与合规性校验，让变更可知、可控、可追溯。

成本优化： 模板可一键复用，变更前可预测资源与费用变更，助力企业降本增效。

友好交互： 企业级可视化交互和丰富API接口。

访问方式

天翼云提供如下方式进行ROS的配置和管理。

控制台：Web化的服务管理平台，即利用管理控制台来配置和管理ROS。

API：天翼云也支持用户通过API（Application Programming Interface）方式访问ROS，具体操作请参见ROS API参考。

功能特性

统一管理资源

通过统一的管理平面，可实现对分布在不同地域资源池中的各类云资源进行全局掌控，实现单一控制台完成状态监控与配置调整，打破资源孤岛，实现跨池资源的统一调度与协同管理，大幅简化复杂多云环境下的运维复杂度。

模板管理

提供灵活且强大的模板定义能力，支持通过声明式语法对云资源的类型、规格、配置参数及关联关系进行精确描述，同时允许用户自定义模板以适配个性化需求。

针对模板全生命周期，提供完善的版本控制机制，可追溯每一次模板修改记录，确保不同团队、不同环境使用的模板版本一致，避免因模板差异导致的资源配置偏差。

自动化管理资源生命周期

以资源栈为管理单元，将一组关联资源视为整体进行全生命周期的自动化管理。流程中嵌入预览执行计划、询价与配额校验环节，通过预览执行计划明确资源的变更，通过询价明确资源组合的成本预估，再校验对应资源的配额是否充足，可实现资源栈生命周期全程自动化：创建、更新、删除、回滚等操作。

智能编排资源、处理资源依赖

通过内置的依赖分析引擎，能够自动识别模板中各资源（如“云主机依赖于虚拟私有云”“数据库依赖于安全组”）的层级关系与关联规则，生成最优部署拓扑与执行顺序。同时，通过分布式锁与状态同步机制，确保多节点并行部署时的数据一致性。

可视化操作与监控

部署过程中，通过实时日志流展示每一步操作的执行情况与耗时，支持钻取查看单个资源的部署详情与配置参数；对于异常状态，通过醒目标识与错误提示快速定位问题点，让用户无需深入技术细节，即可全面掌握资源栈的实时动态，大幅降低运维门槛。

开放API与生态集成

提供丰富的API接口，轻松集成到现有DevOps流程或第三方工具。

产品介绍

产品优势

完全兼容 Terraform与Provider 生态

充分融入 Terraform 生态，兼容其 Provider、模板与状态管理能力，用户可直接复用广泛的社区资源，实现跨云一致的 IaC 编排体验，大幅降低学习成本并提升自动化效率。

极简自动化，实现高效部署

依托模板化编排框架与智能自动化引擎，ROS 将资源部署流程压缩至 “模板设计 - 参数配置 - 一键执行” 三个核心步骤。用户只需通过代码定义一次资源架构模板，即可在多环境中重复部署，避免重复劳动，自动化引擎会全程接管资源创建、依赖处理、配置校验等繁琐操作。这种 “一次设计，多次复用” 的模式，将传统数小时的部署流程缩短至分钟级，人力投入减少 80% 以上，让团队从机械操作中解放出来，聚焦核心业务创新。

安全治理，保障合规可控

ROS内置变更审核、模板合规性校验与资源风险扫描能力：操作前，通过模板合规性校验，平台将为您自动拦截不合规配置；操作后，平台将完整记录每一次变更的执行人、时间、内容及影响范围，形成可追溯的审计日志。

此外，部署前的变更评估功能会提前预判资源的变更内容，并提供规避建议，让每一次资源操作都处于可控范围内，满足金融、政务等行业的严苛合规要求，让每一次资源变更都可知、可控、可追溯，有效降低风险。

固化成熟架构，优化资源成本

支持将已验证的成熟架构固化为标准模板，新业务部署时只需调用模板即可一键复用成熟方案，既避免重复设计的低效，又减少架构设计失误风险。并结合资源变更预测与费用变更预测能力，从资源规划阶段即实现精细化成本管理，帮助企业在保障业务性能的前提下，平均降低云资源成本 20%-30%，从源头上助力企业降本增效。

应用场景

应用快速上云

场景说明

助力企业将复杂的多云/混合云应用架构，通过模板化方式实现一键自动化部署与全生命周期管理。

场景痛点

部署效率低下：传统手动创建与配置云资源流程繁琐，耗时冗长，严重阻碍业务上线速度。

准确性难保障：人工操作极易引入配置偏差，引发线上故障。

产品优势

企业级资源栈管理

提供清晰的项目视图，支持完全的环境隔离，实现对一套应用资源的整体性创建、升级、回滚和删除，管理粒度更符合企业运维习惯。

强大的模板能力与模版复用

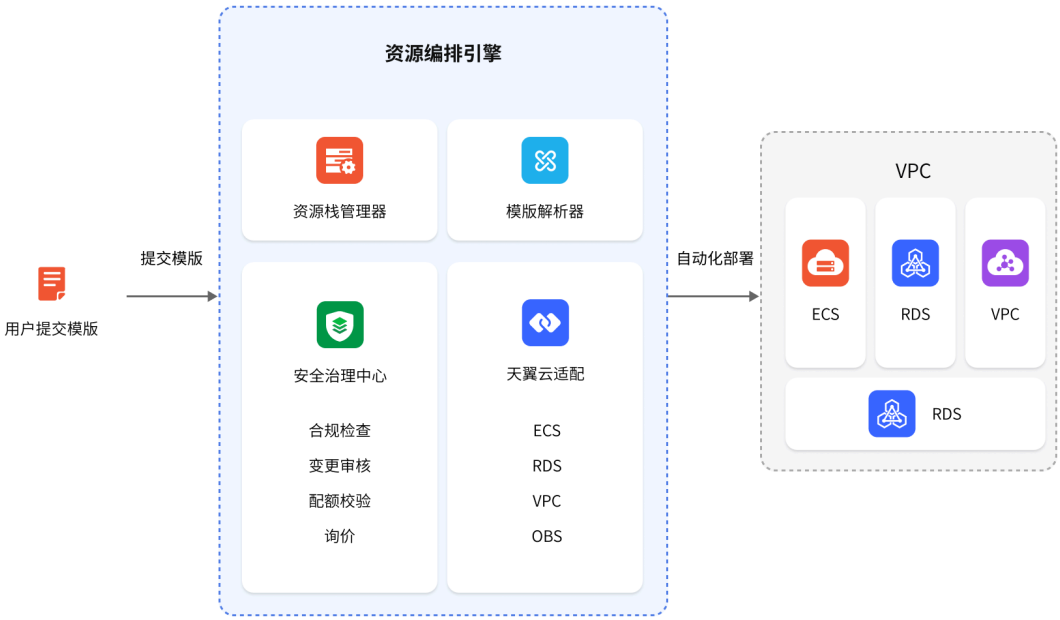
不仅支持原生terraform标准，还提供可视化编辑器和复用最佳实践模板，可直接复用或稍作修改后部署，极大降低模板编写门槛，加速上云进程。

内嵌式安全合规治理

内置变更审核流程、模板合规性校验与资源风险扫描能力，并在执行前提供价格预测，让每一次资源变更都可知、可控、可追溯，有效降低风险。

深度集成天翼云全栈服务

全面适配天翼云核心云服务，提供经过深度优化和验证的官方Provider，确保资源操作的稳定性与高性能。



环境快速复制

场景说明

应对业务弹性扩展与持续交付需求，通过模板一键式快速部署多套隔离且一致的应用环境。

场景痛点

环境搭建效率低下：业务扩展或新版本测试时，手动重复创建环境流程繁琐，无法快速响应需求。

环境一致性难以保证：人工操作易产生配置差异，导致开发、测试、预生产环境与生产环境不一致，引发部署风险。

产品优势

模板化一键复制

基于经过验证的模板，可在数分钟内快速复制出包括网络、计算、数据库在内的完整应用环境，效率远超人工搭建，实现对业务的快速响应。

高效资源调度与生命周期管理

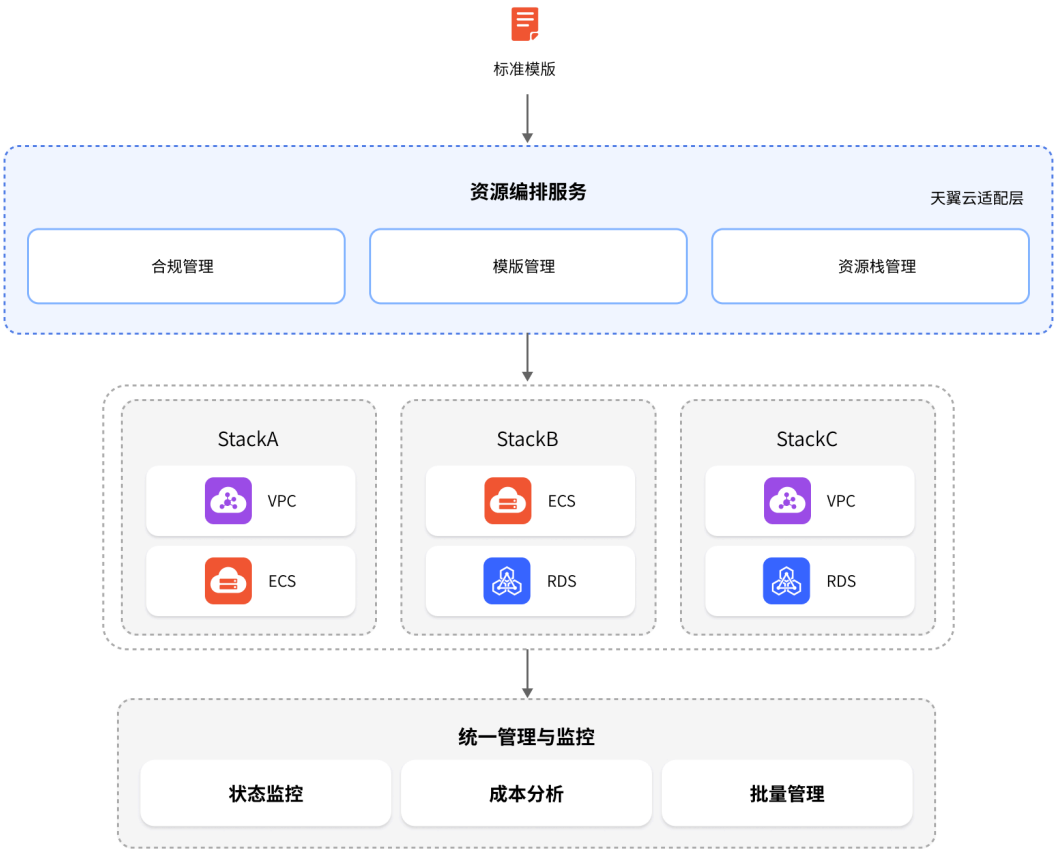
提供强大的资源栈组管理能力，可对多套环境进行集中监控、批量操作和统一运维，显著提升管理效率，避免环境混乱。

资源复用与成本优化

支持快速创建与销毁环境，轻松应对临时性测试需求。结合价格预测与资源分析功能，实现按需取用，有效避免资源闲置，节约成本。

内嵌安全合规管控

在快速复制环境的同时，自动继承模板中预设的安全与合规配置（网络策略、权限规则等），确保每套新环境都“部署即合规”，有效管控多环境安全风险。



基本概念

概念名称	描述
资源	指可被声明、创建、更新和管理的云上实体，如计算、网络、存储等基础组件，是模板编排和部署的最小管理单元。
模板	模板是一个HCL语法文本描述文件，支持tf、tf.json文件格式，用于描述您的云资源。资源编排根据模板完成各种云资源的创建。
资源栈	资源栈是云服务资源的集合。资源栈将模板描述的所有云服务资源作为一个整体来进行创建、删除、更新、查询等。
地域	地域（Region）是指物理的数据中心的地理区域。地域从地理位置和网络时延维度划分，同一个Region内共享弹性计算、块存储、对象存储、VPC网络、弹性公网IP、镜像等公共服务。资源栈隶属于一个地域，完成地域级别的资源创建、删除、更新、查询。

概念名称	描述
执行计划	执行计划提供对资源栈变化的预览。这个执行计划展示了当前模板与线上资源的对比变化，清晰地展示了资源编排对资源与属性将要执行的操作（如新增、修改、删除等）。用户可以预览这个计划，在确认符合预期后，再执行这个计划。资源编排就会完成模板定义资源的创建、变更等。
资源栈输出	指在模板部署完成后，由资源栈对外暴露的关键结果信息，如资源 ID、访问地址、配置信息等，便于其他系统或后续流程引用和集成。
资源栈事件	执行计划的执行，执行计划/资源栈的部署，销毁等执行过程中产生的关键过程记录，用于展示每一步的状态变化、操作动作及结果，方便排查问题和追踪执行流程。

与其它服务关系

在资源编排ROS中，资源栈是云服务资源的集合。因此ROS可以通过模板编排的方式，对其他云服务资源，整体地进行创建、删除、更新、查询操作。

同时ROS依托天翼云IAM服务完成用户的统一权限和管理，依托密钥管理，对用户模板中的加密参数进行加密存储，保证资源管理的安全性。

与其他服务关系

服务名称	交互功能	相关内容
IAM	在资源编排ROS中，资源栈是云服务资源的集合。因此ROS可以通过模板编排的方式，对其他云服务资源，整体地进行创建、删除、更新、查询操作。	统一身份认证
密钥管理	ROS中用户模板中加密参数，需要依托密钥管理服务进行加密存储，从而保证用户模板参数的安全性。	密钥管理服务
各类云服务	资源编排本身不直接提供云资源能力，而是通过调用各云服务的 OpenAPI 来完成资源的创建、更新、查询和删除等操作，将分散的服务能力以模板化、自动化方式组合起来，实现跨云产品的一致管理与统一交付。	各类云服务文档

使用限制

为确保资源编排产品可以正常为您服务，在使用之前请您务必仔细阅读以下配额限制。

资源编排ROS对用户资源数量限定了配额，如果您需要使用更多资源，请提交工单进行申请。

配额项	默认配额
单个天翼云账号允许创建的最大模板个数	100
单个模板最大版本个数	100
每个模板内容最大值	1MB
每个模板文件内容最大值	128KB
每个模板文件数量最大值	50
每个模板参数个数	100
模板参数默认值长度最大值	5KB
每个模板输出个数	100
资源栈单次执行输出内容最大值	1MB
单个天翼云账号允许创建的最大资源栈个数	100
单个资源栈事件保留	30天/10000条/单条最大2KB

计费说明

计费项及计费方式

计费项	计费说明	计费方式
资源编排服务	资源编排服务本身完全免费	无
资源编排创建的云资源	由资源编排创建的云资源遵循对应云产品的标准计费规则，与天翼云平台其他创建方式费用一致	按照各云产品实际计费方式进行收费，具体请参考对应产品的计费说明

费用预览功能

资源编排服务提供费用预览功能，在资源栈执行计划阶段会展示预计产生的费用详情，帮助您在资源创建前清晰了解相关成本，便于进行预算规划和管理。

说明

1. 资源编排服务为免费服务，不收取任何服务费用
2. 实际产生的费用为通过资源编排创建的云资源费用，收费标准与各云产品官方定价保持一致
3. 费用结算遵循天翼云平台统一的计费规则和结算流程
4. 建议在执行资源栈前仔细查看费用预估，确保符合预算预期
5. 如需了解具体云产品的详细计费信息，请参考天翼云官方产品文档或联系客服咨询。

创建一台包年包月云主机

操作场景

本示例以创建一台包年包月云主机为例，介绍如何使用模板创建一台包年包月云主机。

不同架构内设云资源不同，本文及供参考。

操作步骤

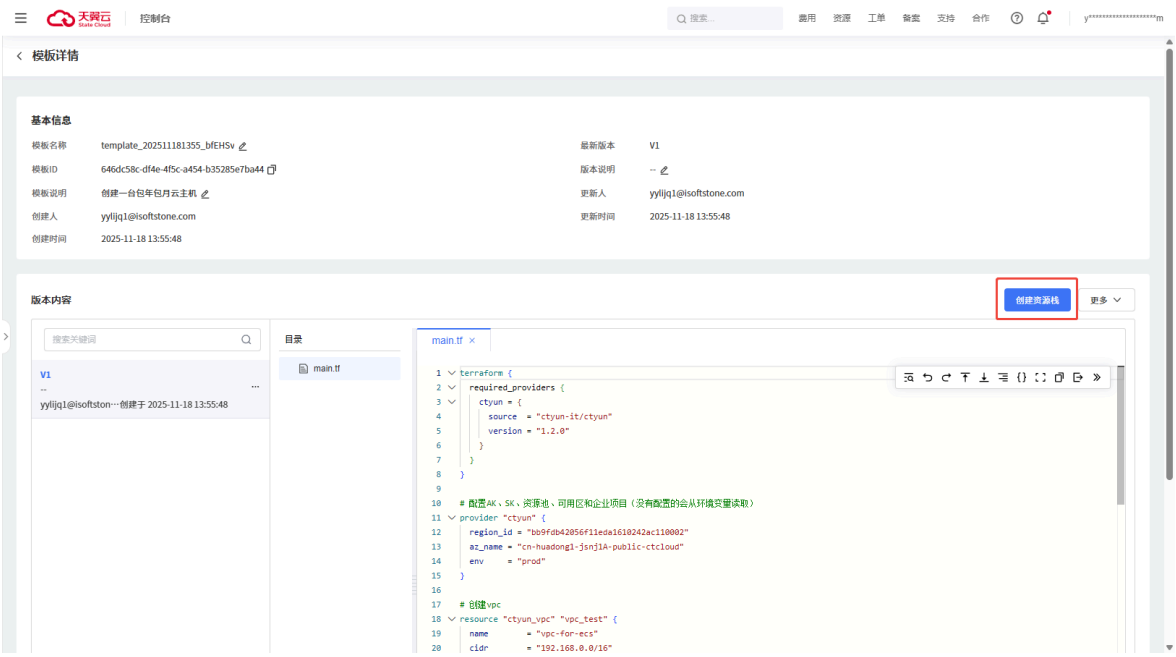
1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **模板管理**。
4. 在模板管理页面，单击**创建模板**。
5. 创建包年包月云主机的.tf模板示例如下，请参考文档创建模板完成模板配置。

```
terraform {
 required_providers {
  ctyun = {
   source  = "ctyun-it/ctyun"
   version = "1.2.0"
  }
 }
}

# 支持配置资源池、可用区和企业项目，若不配置资源池，则会使用页面选定的资源池
# ak/sk无需配置，会自动获取当前账号的ak/sk
provider "ctyun" {
 region_id = "bb9fdb42056f11eda1610242ac110002"
 az_name = "cn-huadong1-jsnj1A-public-ctcloud"
}

# 创建vpc
resource "ctyun_vpc" "vpc_test" {
 name      = "vpc-for-ecs"
 cidr      = "192.168.0.0/16"
 description = "terraform测试使用"
 enable_ipv6 = true
}

# 在vpc下创建子网
resource "ctyun_subnet" "subnet_test" {
 vpc_id    = ctyun_vpc.vpc_test.id
 name      = "subnet-for-ecs"
 cidr      = "192.168.1.0/24"
 description = "terraform测试使用"
```

```
  dns = [
    "114.114.114.114",
    "8.8.8.8"
  ]
  enable_ipv6 = true
}

# 查询可用镜像
data "ctyun_images" "image_test" {
  name      = "CentOS Linux 8.4"
  visibility = "public"
  page_no    = 1
  page_size  = 10
}

# 查询可用规格
data "ctyun_ecs_flavors" "ecs_flavor_test" {
  cpu   = 2
  ram   = 4
  arch  = "x86"
  series = "C"
  type  = "CPU_C7"
}

# 导入密钥对
resource "ctyun_keypair" "keypair_test" {
  name     = "keypair-for-ecs"
  public_key = "ssh-rsa AAAAB3Nz
aC1yc2EAAAADAQABAAABAQDAjUnAnTid4wmVtajSmElMtHO3OvOyY81ybfswbUu9Gt83DVVzDnwb3rcQW1us8SeKm/gRINkgd
RAgfXAmTKR7AorYtWWc/tzb6kcDpL2E8Qk+n6cyFAxX
NoX2vXBr4kC9wz1uwjGyxoSlpHLIpscfIOEf652gMlSyfODehAJHj3JPMr8pvtPIUqsZI3JOGTUzxaA2JVCOLxQegphYYf2Tx
1NaITXNVEj3A11hk1nrFoJMmvIwIUkLmRuQcxuNAdxeLB7GXXVjKpnKIJL4L64dyA9GWa3Gb7gCJyRaBc5UhK4hT57wmukCrl
}

resource "ctyun_ecs" "ecs_test" {
  instance_name   = "ecs-demo"
  display_name    = "ecs-demo"
  flavor_id      = data.ctyun_ecs_flavors.ecs_flavor_test.flavors[0].id
  image_id       = data.ctyun_images.image_test.images[0].id
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id        = ctyun_vpc.vpc_test.id
  cycle_type     = "month"
  cycle_count    = 1
  subnet_id      = ctyun_subnet.subnet_test.id
  key_pair_name   = ctyun_keypair.keypair_test.name
  is_destroy_instance = true # 表示退订后直接销毁
```

}

6. 在刚刚创建好的模板详情中，单击**创建资源栈**。



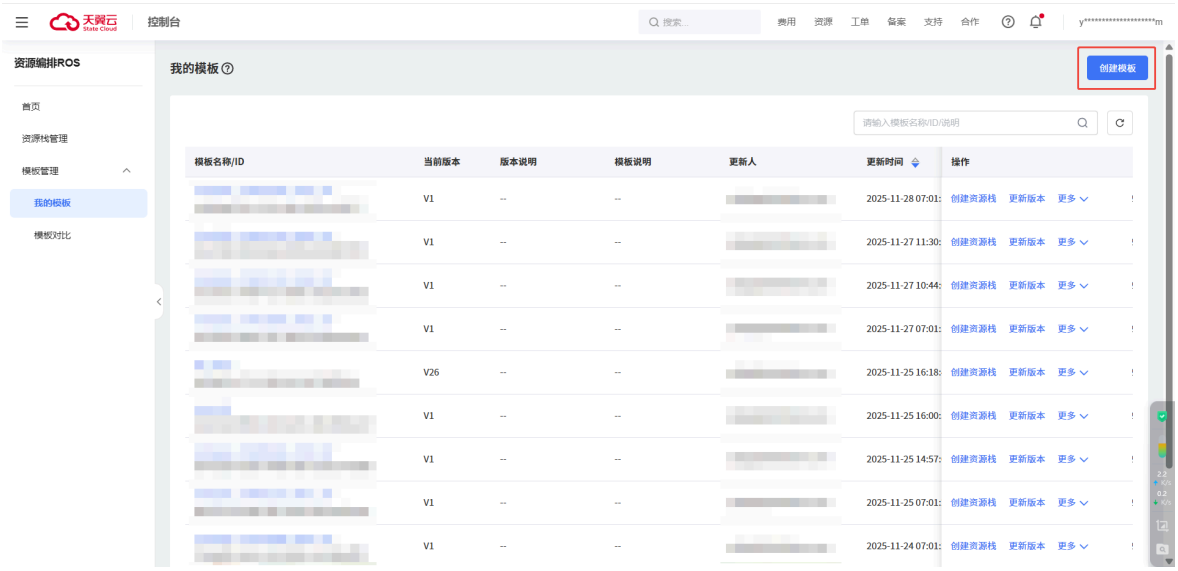
7. 参考创建资源栈，使用当前模板完成资源栈创建。
8. 参考部署资源栈，完成资源栈部署。
9. 部署完成后，您可前往对应资源控制台查看资源的创建结果。

创建模板

模板是创建和更新资源栈的脚本，资源栈的创建依赖于已有模板。创建资源栈前，请先创建一个模板。

操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **模板管理-我的模板**。
4. 在**我的模板**页面，单击**创建模板**。



5. 在**创建模板**页面，可以根据提示配置等基本参数，具体参数说明请参考步骤下面的表格。

参数	是否必填	参数说明	参考样例
模板名称	是	1. 自动填充默认值。 2. 最长可输入128字符。 3. 仅允许输入中英文、数字、下划线和中划线，且必须以字母或中文开头。	template_202509281539_xitYBv
模板说明	否	可简要说明模板用途，最长可输入255字符	批量创建5台云主机

参数	是否必填	参数说明	参考样例
版本说明	否	1. 创建模板将自动生成模板的第一个版本，之后每次更新都将自动为版本号+1。 2. 可在版本说明处简要说明当前版本特点，最长可输入255个字符。	批量创建5台云主机
模板内容	是	1. 新增tf文件，在文件中按照terraform语法定义各类云资源（例如弹性云主机、虚拟私有云、弹性IP等）的增删改查语句。 2. 您可通过添加多个文件、文件夹对您的模板进行结构化定义。	模板示例

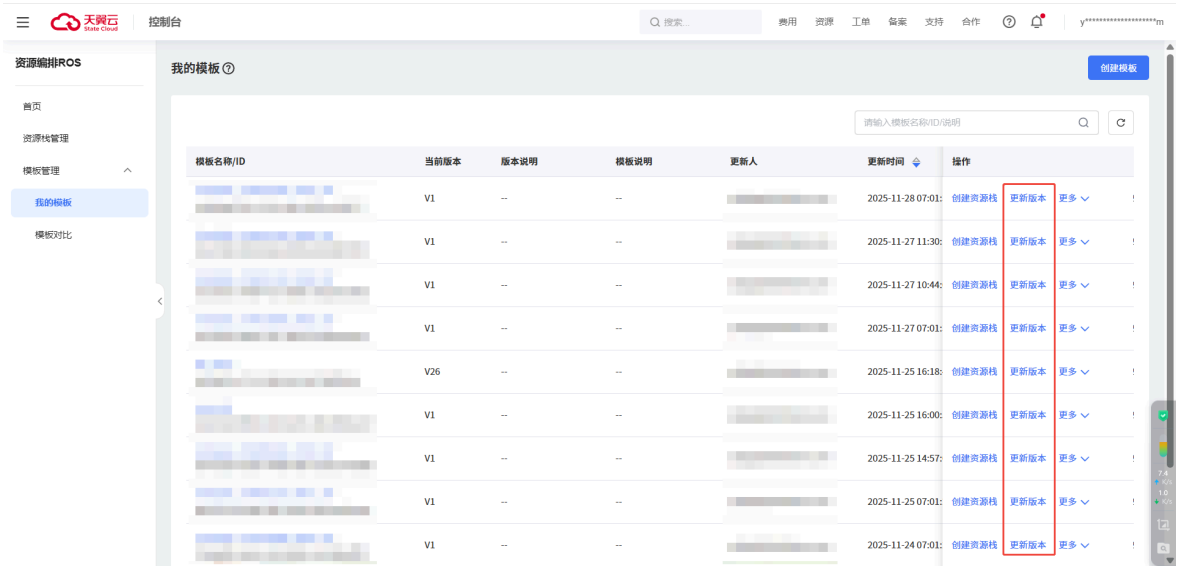
6. 单击**立即创建**完成创建。

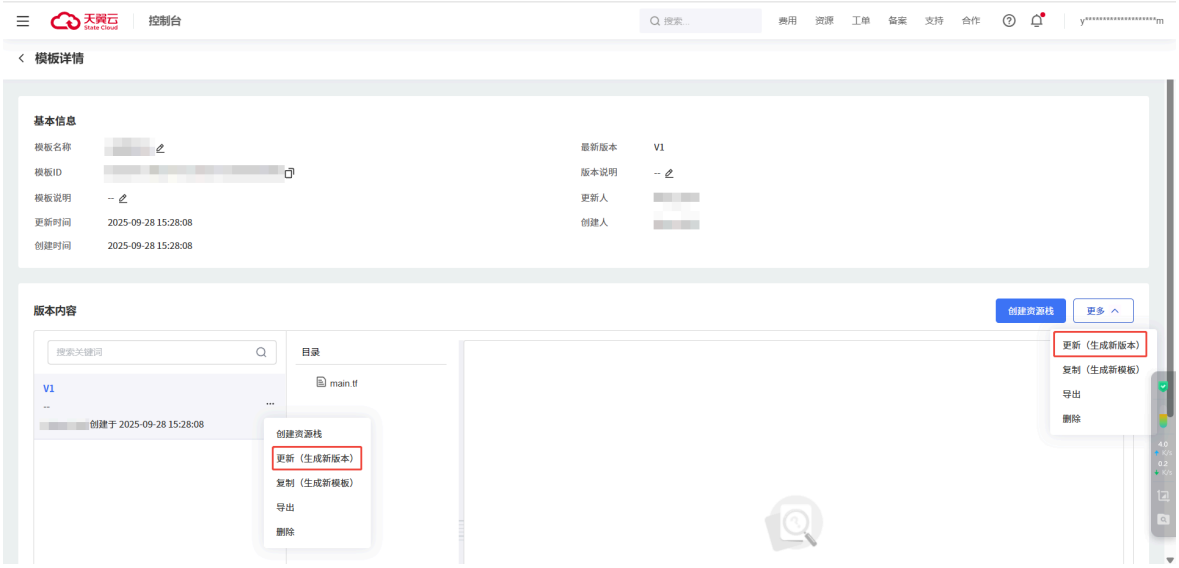
更新模板

更新模板将为模板创建一个新版本。

操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **模板管理-我的模板**。
4. 您可在模板列表中直接单击**更新模板**，此时将在模板的最新版本基础上更新。或者您可以单击模板，在**模板详情**中选择目标版本，单击**更新（创建新版本）**。



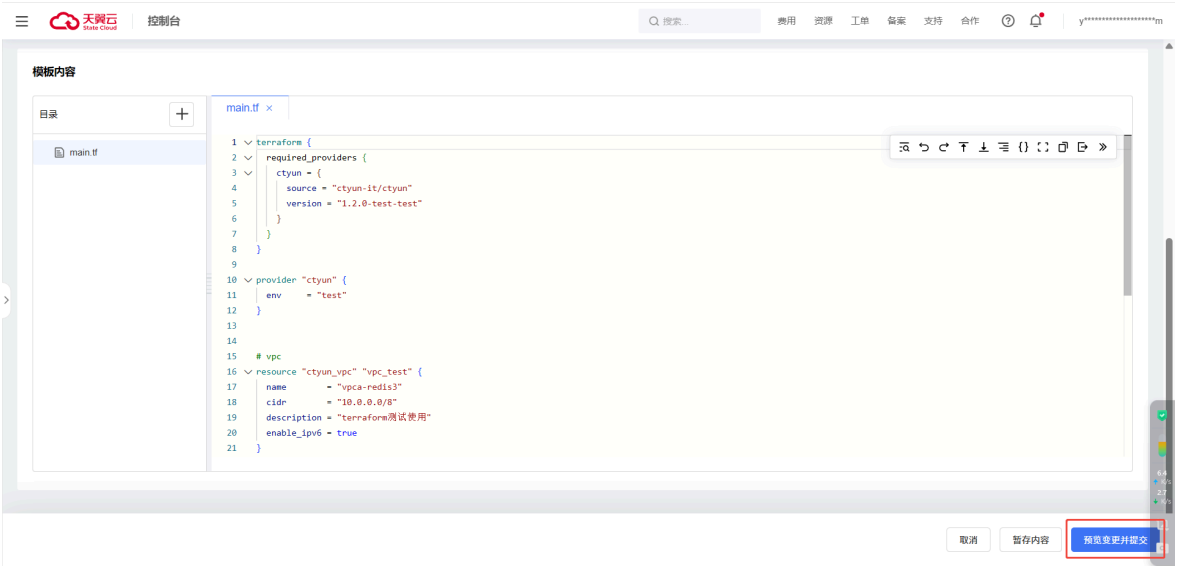


5. 在**更新模板**页面，可以根据提示配置等基本参数，具体参数说明请参考步骤下面的表格。

参数	是否必填	参数说明	参考样例
版本说明	否	1. 创建模板将自动生成模板的第一个版本，之后每次更新都将自动为版本号+1。 2. 可在版本说明处简要说明当前版本特点，最长可输入255个字符。	批量创建5台云主机

参数	是否必填	参数说明	参考样例
模板内容	是	1. 新增tf文件，在文件中按照terraform语法定义各类云资源（例如弹性云主机、虚拟私有云、弹性IP等）的增删改查语句。 2. 您可通过添加多个文件、文件夹对您的模板进行结构化定义。	模板示例

6. 完成更新后，点击**预览变更并提交**，进入预览页面



7. 如预览无误，您可单击**提交**完成更新。

查询模板

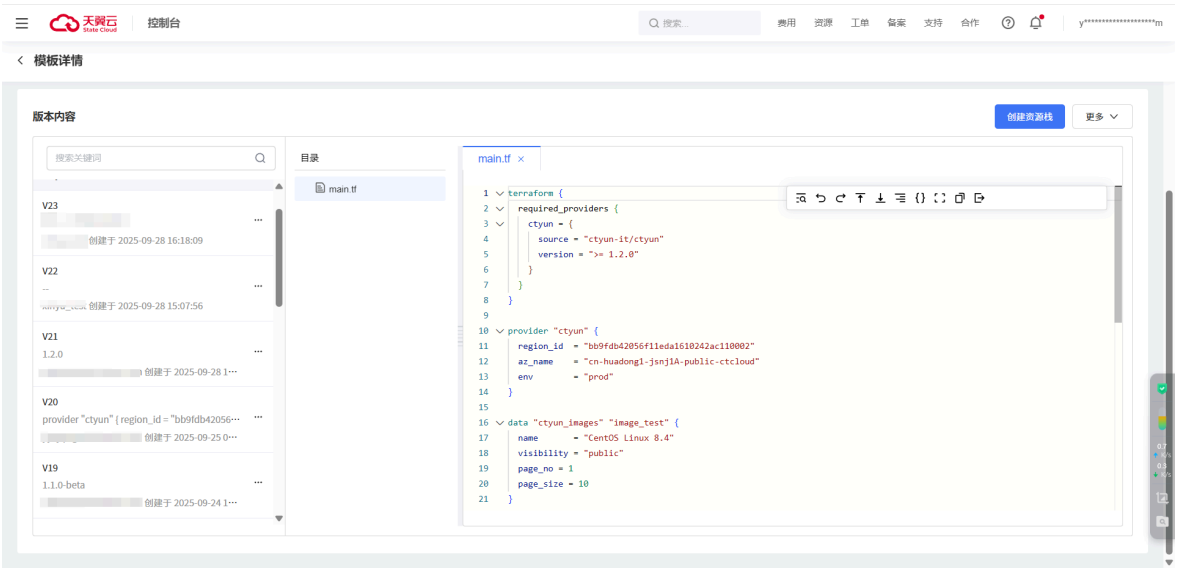
操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **模板管理-我的木板**。
4. 单击**模板**，进入**模板详情**。

模板详情

1. 基本信息：展示模板的基本信息如下

2. 版本列表及版本详情：您可在左侧版本列表中选择版本，即可在右侧查询版本内容



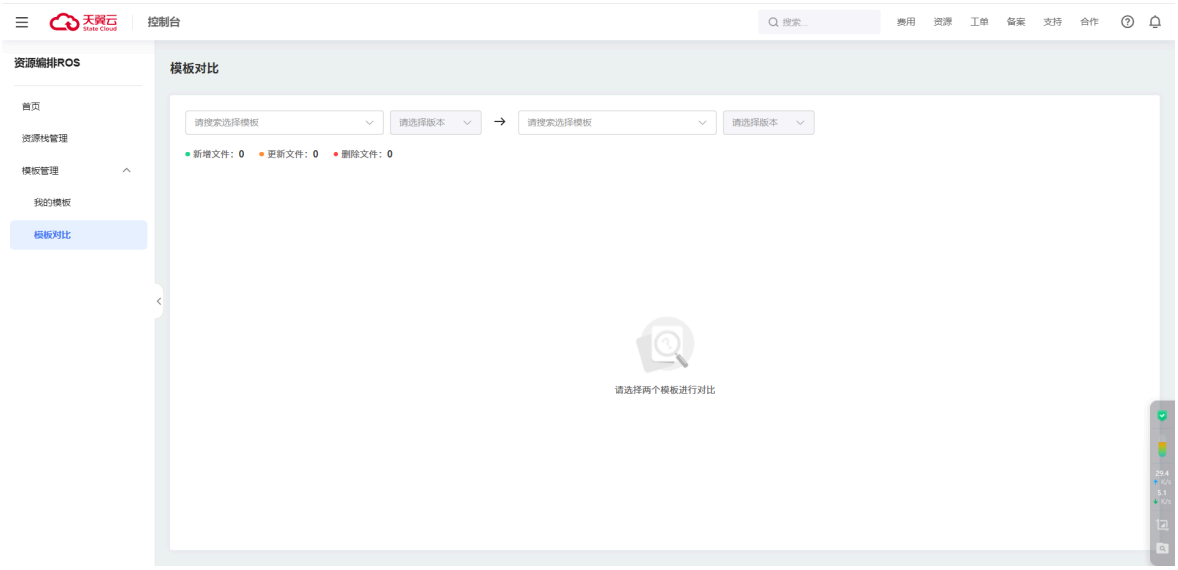
3. 版本操作

- 创建资源栈：您可以直接选择目标模板，单击创建资源栈，即可跳转新建，参考创建资源栈。
- 更新：更新模板将新建一个模板的新版本，参考更新版本。
- 复制：复制模板将基于当前版本创建一个新模板，参考创建模板。
- 版本对比：您可对比当前模板下不同版本之间的模板内容变更情况，参考模板对比。
- 导出：您可通过导出操作，下载模板文件。
- 删除。

模板对比

操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **模板管理-模板对比**。



4. 请选择两个模板和对应版本进行模板内容对比，您可在页面上看到文件数量变化和内容变化。

资源栈状态说明

资源栈和执行计划状态分为稳定状态和中间状态，中间状态是资源栈在到达稳定状态前暂时处于的状态，如果实例长时间处于中间状态，说明可能出现了异常。

资源栈状态

状态	状态类型	说明
待部署	稳定状态	创建资源栈成功后，资源栈中有“创建成功，待部署”的执行计划的状态，此时用户可单击部署，执行部署操作。
部署准备中	中间状态	开始执行部署操作和“部署中”之间的中间状态，准备好后状态将变为“部署中”。
部署中	中间状态	执行部署操作过程中的中间状态，部署完成后状态将变为“部署成功”或“部署失败”。 部署过程中，您可单击**取消部署**，取消部署成功后状态将变为“取消部署”。
部署成功	稳定状态	部署成功后的正常状态，用户可正常进行相关业务。

状态	状态类型	说明
部署失败	稳定状态	前置条件：用户没有配置“失败时回滚” 部署失败后的状态，此时资源栈中的资源可能与计划不一致。用户更新并部署资源栈后，将重新流转部署状态。 此时可以通过两种方式解决问题： • 方式一，请参考帮助文档常见问题进行自助修复； • 方式二，请提交工单，由技术人员协助解决问题。
部署失败，回滚准备中	中间状态	开始执行回滚操作和“回滚中”之间的中间状态，准备好后状态将变为“回滚中”。
部署失败，回滚中	中间状态	前置条件：用户配置了“失败时回滚” 执行回滚操作过程中的中间状态，回滚完成后状态将变为“回滚成功”或“回滚失败”。 回滚过程中，您可单击取消回滚，取消回滚成功后状态将变为“取消回滚”。
部署失败，回滚成功	稳定状态	前置条件：用户配置了“失败时回滚” 回滚成功后的正常状态，此时资源栈已回滚至上一次部署成功状态，用户可正常进行相关业务。
部署失败，回滚失败	稳定状态	前置条件：用户配置了“失败时回滚” 回滚失败后的状态，此时资源栈仍保持着本次回滚失败结果，此时资源栈中的资源可能与计划不一致。 此时可以通过两种方式解决问题： • 方式一，请参考帮助文档常见问题进行自助修复； • 方式二，请提交工单，由技术人员协助解决问题。
删除准备中	中间状态	开始执行删除操作和“删除中”之间的中间状态，准备好后状态将变为“删除中”。
删除中	中间状态	执行删除操作过程中的中间状态，删除完成后状态将变为“删除成功”或“删除失败”。 删除过程中，您可单击取消删除，取消删除成功后状态将变为“取消删除”。
删除失败	稳定状态	删除失败后的状态，此时资源栈仍保持着本次删除失败结果，此时资源栈中的资源可能与计划不一致。 此时可以通过两种方式解决问题： • 方式一，请参考帮助文档常见问题进行自助修复； • 方式二，请提交工单，由技术人员协助解决问题。
取消删除	稳定状态	用户执行取消删除后的状态。用户更新并部署资源栈后，将重新流转部署状态。
取消部署	稳定状态	用户执行取消部署后的状态。用户更新并部署资源栈后，将重新流转部署状态。
取消回滚	稳定状态	用户执行取消回滚后的状态。用户更新并部署资源栈后，将重新流转部署状态。

执行计划状态

状态	状态类型	说明
等待创建	中间状态	执行创建任务和“创建中”之间的瞬时状态，准备好后状态将变为“创建中”。
队列中	中间状态	执行创建任务和“创建中”之间的瞬时状态，准备好后状态将变为“创建中”。
创建中	中间状态	创建执行计划时的状态，创建完成后，状态将变为“创建成功，待部署”或“创建失败” 创建过程中，您可单击**取消创建**，取消创建成功后状态将变为“取消创建”。
创建失败	稳定状态	创建执行计划失败后的状态，此时仅是创建执行计划失败，对资源栈本身没有影响。 此时可以通过两种方式解决问题： • 方式一，请参考帮助文档常见问题进行自助修复； • 方式二，请提交工单，由技术人员协助解决问题。
创建成功，待部署	稳定状态	创建执行计划成功的状态，此时您可单击**部署**，执行部署操作，参考部署资源栈。
已触发部署	中间状态	执行部署操作和“部署中”之间的瞬时状态，准备好后状态将变为“部署中”。
部署中	中间状态	部署执行计划时的状态，部署完成后，状态将变为“部署失败”或“部署成功” 部署过程中，您可单击**取消部署**，取消部署成功后，转台将变为“取消部署”。
部署成功	稳定状态	部署成功后的正常状态，用户可正常进行相关业务。

状态	状态类型	说明
部署失败	稳定状态	部署失败后的状态，此时资源栈中的资源可能与计划不一致。用户更新并部署资源栈后，将重新流转部署状态。 此时可以通过两种方式解决问题： • 方式一，请参考帮助文档常见问题进行自助修复； • 方式二，请提交工单，由技术人员协助解决问题。
取消部署	稳定状态	用户执行取消部署后的状态。
取消创建	稳定状态	用户执行取消创建后的状态。

创建资源栈

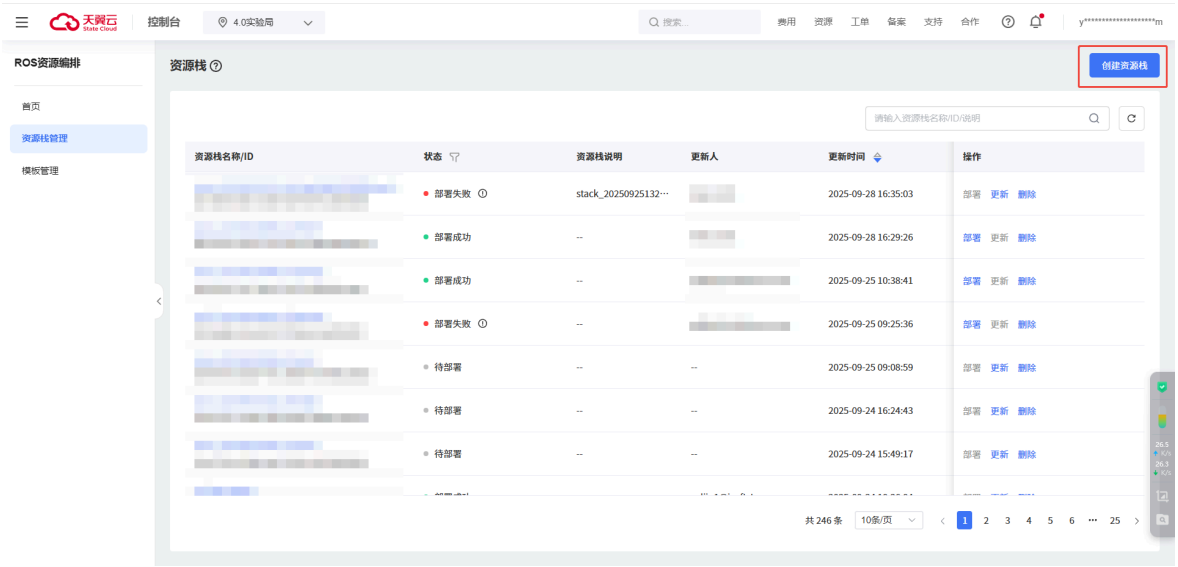
资源栈是一组由模板定义出来的云资源组，您可通过创建（更新）并部署资源栈，实现创建（更新）一组资源。

前置步骤

1. 已在**模板管理**中有已创建成功的模板。参考创建模板。

操作步骤

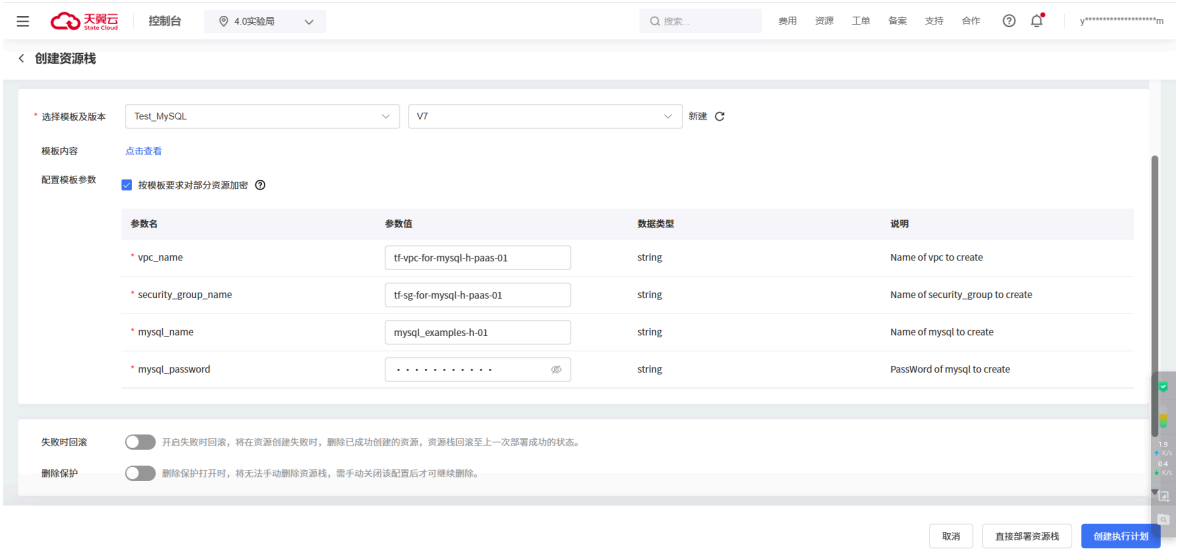
1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **资源栈管理**。
4. 在**资源栈管理**页面，单击**创建资源栈**。



5. 在**创建资源栈**页面，可以根据提示配置等基本参数，具体参数说明请参考步骤下面的表格。

参数	是否必填	参数说明	参考样例
资源栈名称	是	• 自动填充默认值。 • 最长可输入128字符。 • 仅允许输入中英文、数字、下划线和中划线，且必须以字母或中文开头。	stack_202509281651_MaWnRD
资源栈说明	否	可简要说明资源栈的组成/用途，最长可输入255字符	批量创建5台云主机
选择模板及版本	是	可选择一个模板及其版本，用来创建资源栈。如果没有可用的模板或版本，您可单击**新建**跳转创建。参考创建模板	模板示例

参数	是否必填	参数说明	参考样例
配置模板参数	否	• 如果模板中定义了自定义变量，您还需要完成变量配置，变量的配置需要符合模板中定义的校验规则。 • 如果您在模板中设置参数的“sensitive”值为“true”，ROS将使用天翼云云产品 密钥管理服务 对您的敏感参数进行加密。密钥管理服务会创建默认密钥“ros”，该默认密钥免费，请放心使用。	模板示例
失败时回滚	否	默认不开启。开启失败时回滚，将在资源创建失败时，删除已成功创建的资源，资源栈回滚至上一次部署成功的状态。	不开启
删除保护	否	默认不开启。删除保护打开时，将无法手动删除资源栈，需手动关闭该配置后才可继续删除。	不开启



6. 密钥管理服务开通校验。如果您在模板中设置参数的“sensitive”值为“true”，ROS将使用天翼云云产品 密钥管理服务 对您的敏感参数进行加密。如果您尚未开通过该服务，您将会收到提示如下。

- 如果您确认授权，请单击**确认授权，继续操作**，ROS将使用密钥管理服务为您设置了“sensitive”值为“true”的参数进行加密保存。
- 如果您不希望授权，请单击**取消加密，继续操作**，ROS将不再为您加密保存“sensitive”值为“true”的参数。

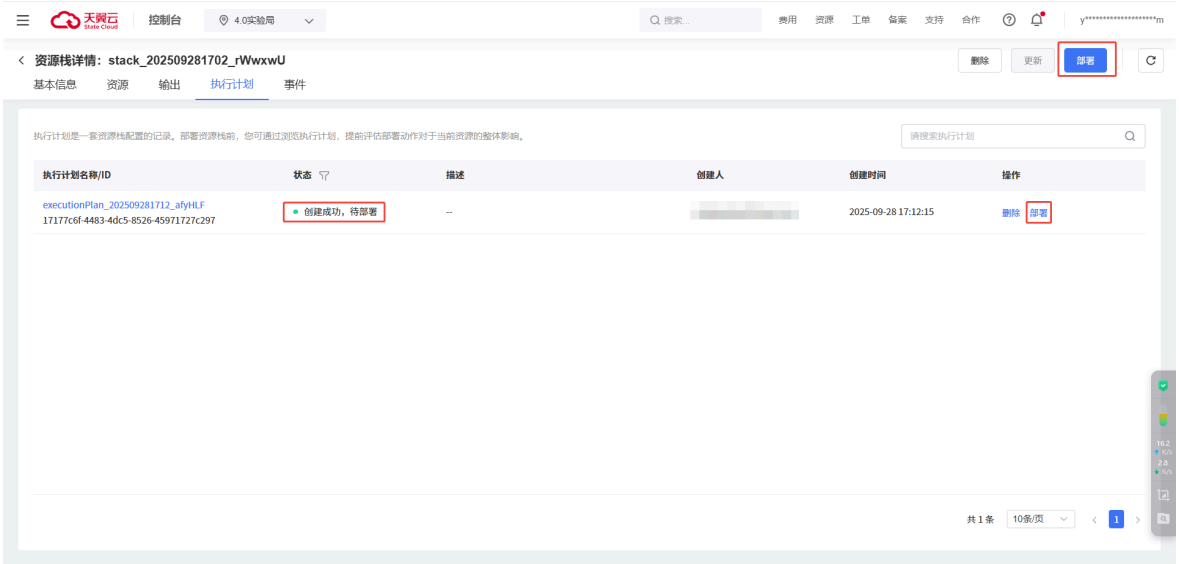
该默认密钥免费，请放心使用。

7. 提交表单时，您可选择单击**创建执行计划**，此时您还需要补充执行计划信息。

- 执行计划是一套资源栈配置的记录。部署资源栈前，您可通过浏览执行计划，提前评估部署动作对于当前资源的整体影响。

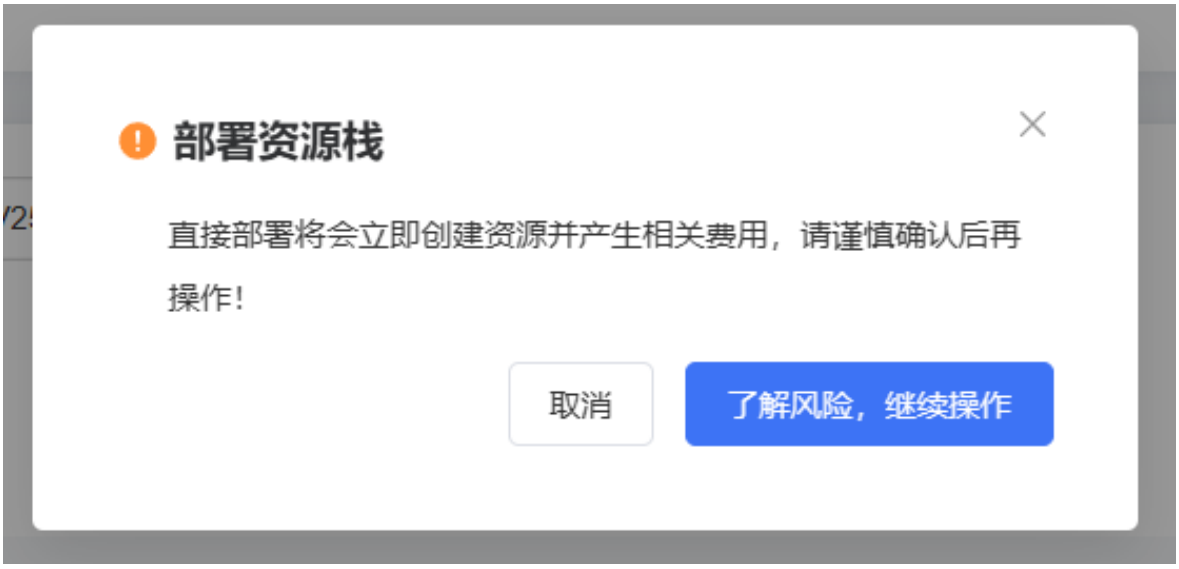
参数	是否必填	参数说明	参考样例
执行计划名称	是	• 自动填充默认值。 • 最长可输入128字符。 • 仅允许输入中英文、数字、下划线和中划线，且必须以字母或中文开头。	executionPlan_20250928 1713_sokRRg
执行计划说明	否	可简要说明资源栈的更新内容，最长可输入255字符	批量创建5台云主机

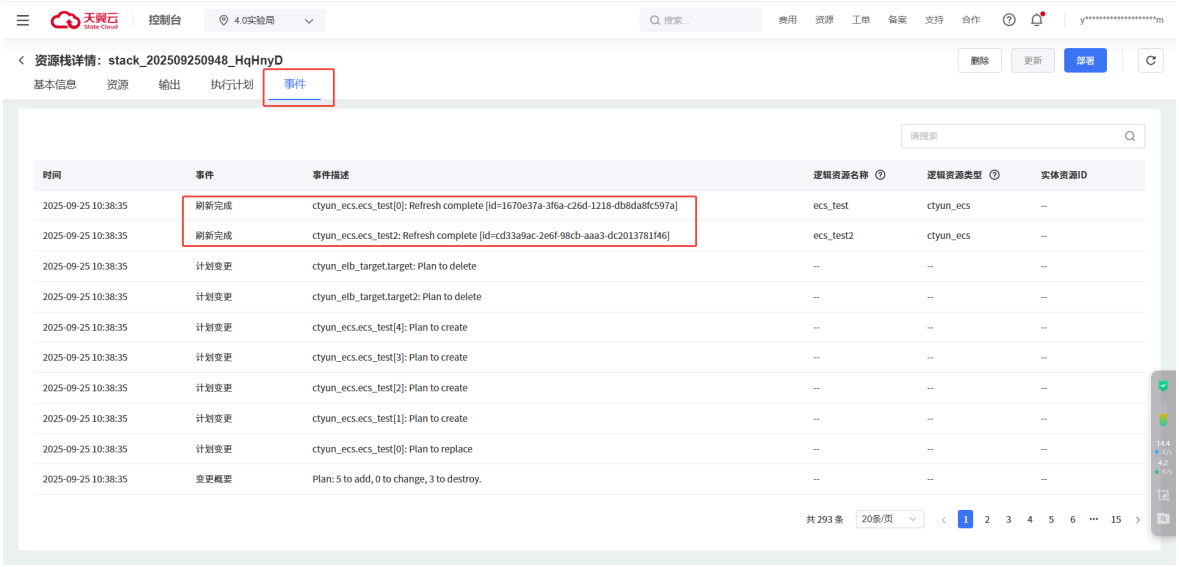
- 单击**确认**，平台将创建一个执行计划，待执行计划状态由“创建中”跳转为“创建成功，待部署”后，您可继续部署资源栈。部署资源栈前，您可在执行计划详情中，确认资源变更和价格变更内容后，再部署资源栈。



8. 提交表单时，您可选择单击**直接部署资源栈**，此时平台将直接部署资源栈，而不会创建执行计划，即不会记录您本次部署的变更内容。此时您可在**资源栈详情的事件页面**查看部署状态。部署资源栈请参考部署资源栈。

注意：直接部署将会立即创建资源并产生相关费用，该操作为危险操作，不建议执行，请谨慎确认后再操作。





更新资源栈

您可通过更新资源栈，更新多个资源的状态（升降配、更改配置、新增或删除资源等）。

限制条件

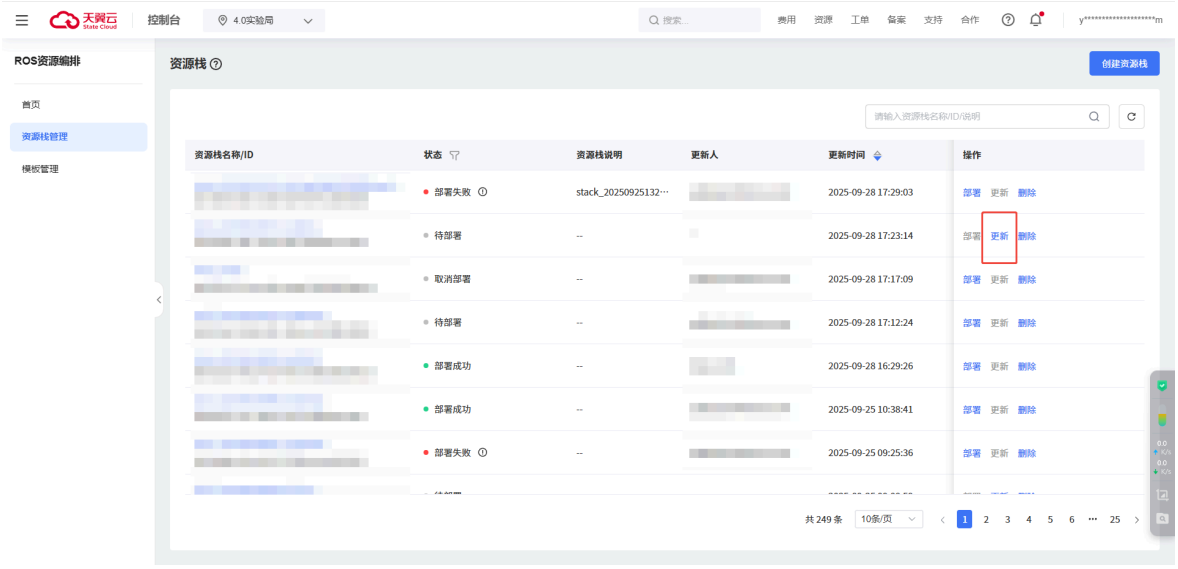
资源栈拥有可部署的执行计划时，不可更新资源栈，请先部署执行计划或删除待部署的执行计划后，再更新资源栈。

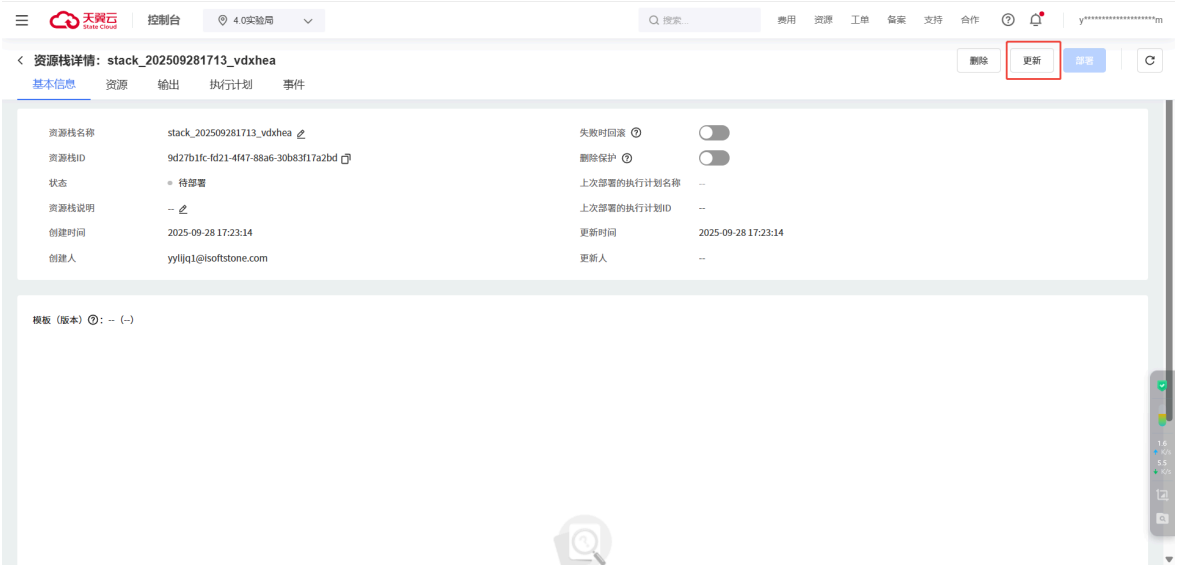
前置步骤

1. 已在**模板管理**中有已创建成功的模板。参考创建模板。

操作步骤

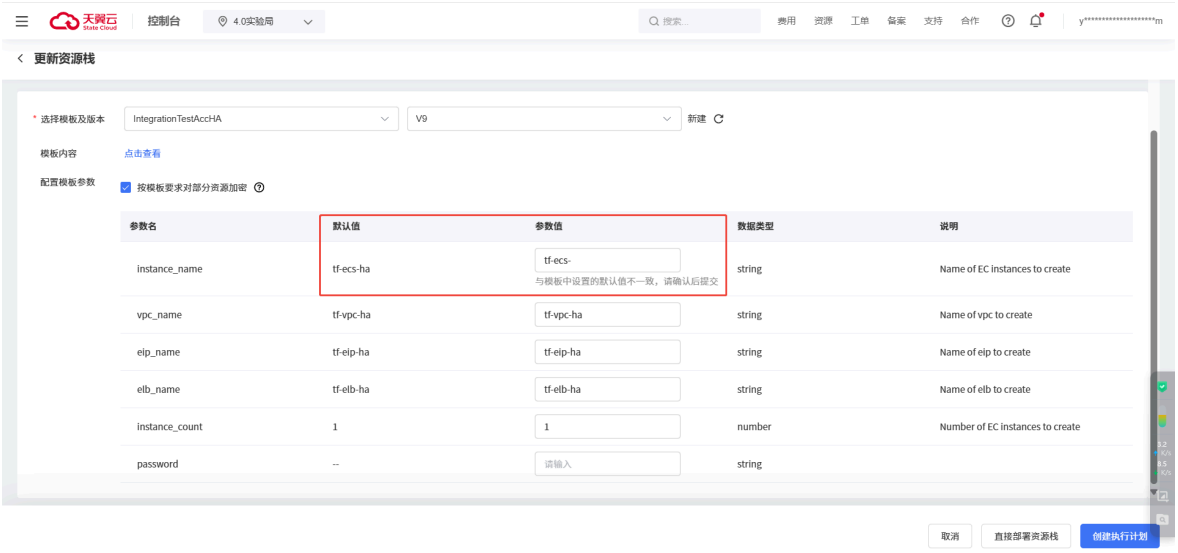
1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **资源栈管理**。
4. 在资源栈列表中直接单击**更新**，或在**资源栈详情**中单击**更新**。





5. 在**创建资源栈**页面，可以根据提示配置等基本参数，具体参数说明请参考步骤下面的表格。

参数	是否必填	参数说明	参考样例
选择模板及版本	是	可选择一个模板及其版本，用来创建资源栈。如果没有可用的模板或版本，您可单击**新建**跳转创建。参考创建模板	模板示例
配置模板参数	否	• 如果模板中定义了自定义变量，您还需要完成变量配置，变量的配置需要符合模板中定义的校验规则。 • 如果您在模板中设置参数的“sensitive”值为“true”，ROS将使用天翼云云产品 密钥管理服务 对您的敏感参数进行加密。密钥管理服务会创建默认密钥“ros”，该默认密钥免费，请放心使用。	模板示例



6. 密钥管理服务开通校验。如果您在模板中设置参数的“sensitive”值为“true”，ROS将使用天翼云云产品 密钥管理服务 对您的敏感参数进行加密。如果您尚未开通过该服务，您将会收到提示如下。

- 如果您确认授权，请单击**确认授权，继续操作**，ROS将使用密钥管理服务为您设置了“sensitive”值为“true”的参数进行加密保存。
- 如果您不希望授权，请单击**取消加密，继续操作**，ROS将不再为您加密保存“sensitive”值为“true”的参数。

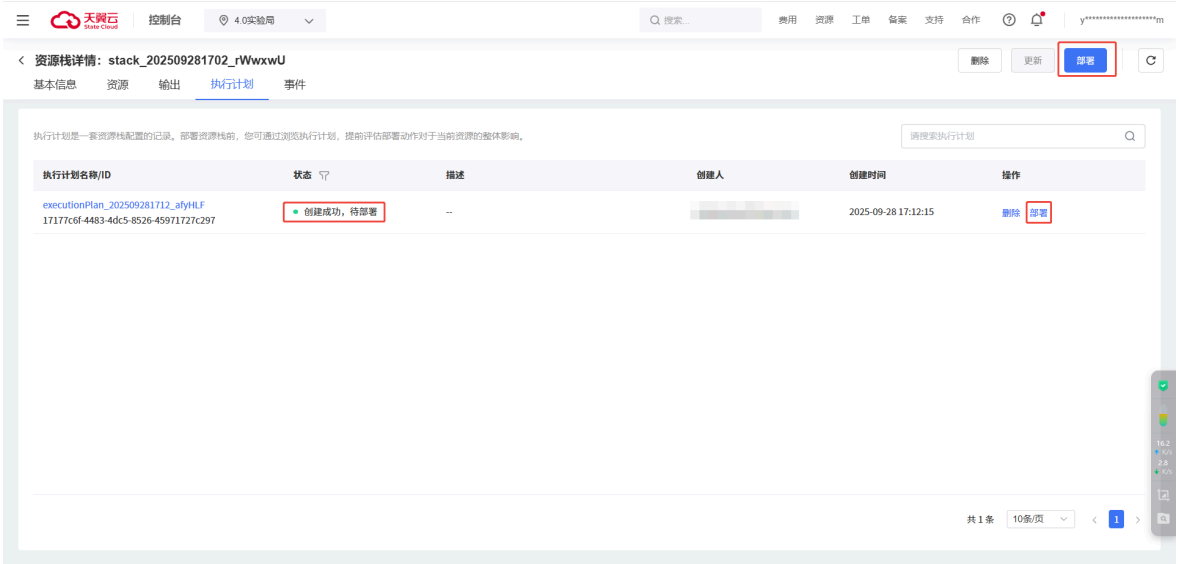
该默认密钥免费，请放心使用。

7. 提交表单时，您可选择单击**创建执行计划**，此时您还需要补充执行计划信息。

- 执行计划是一套资源栈配置的记录。部署资源栈前，您可通过浏览执行计划，提前评估部署动作对于当前资源的整体影响。

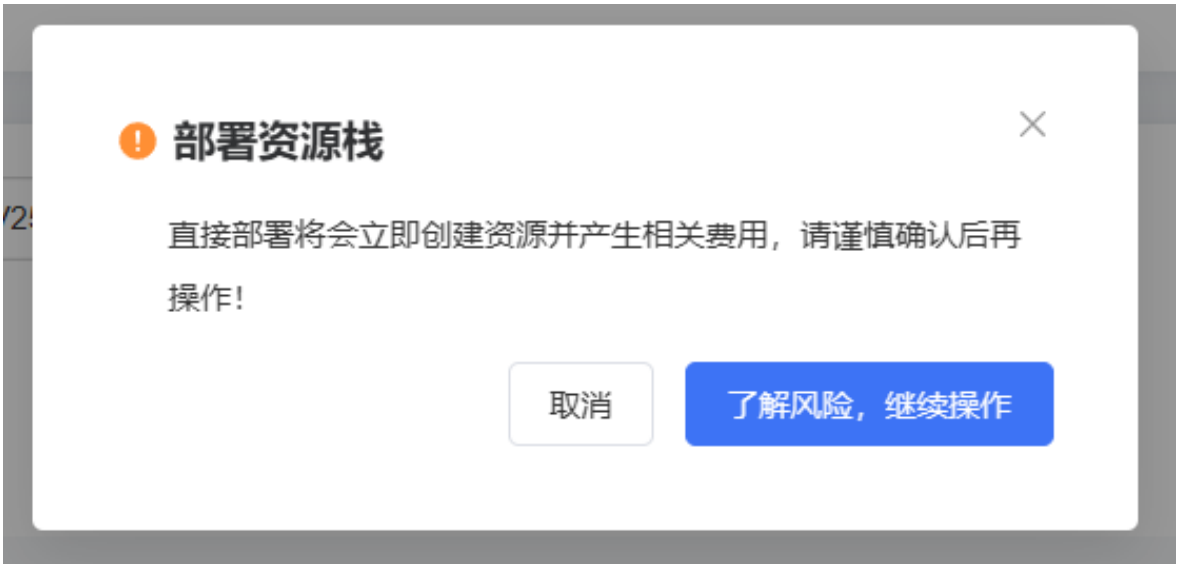
参数	是否必填	参数说明	参考样例
执行计划名称	是	• 自动填充默认值。 • 最长可输入128字符。 • 仅允许输入中英文、数字、下划线和中划线，且必须以字母或中文开头。	executionPlan_20250928 1713_sokRRg
执行计划说明	否	可简要说明资源栈的更新内容，最长可输入255字符	批量创建5台云主机

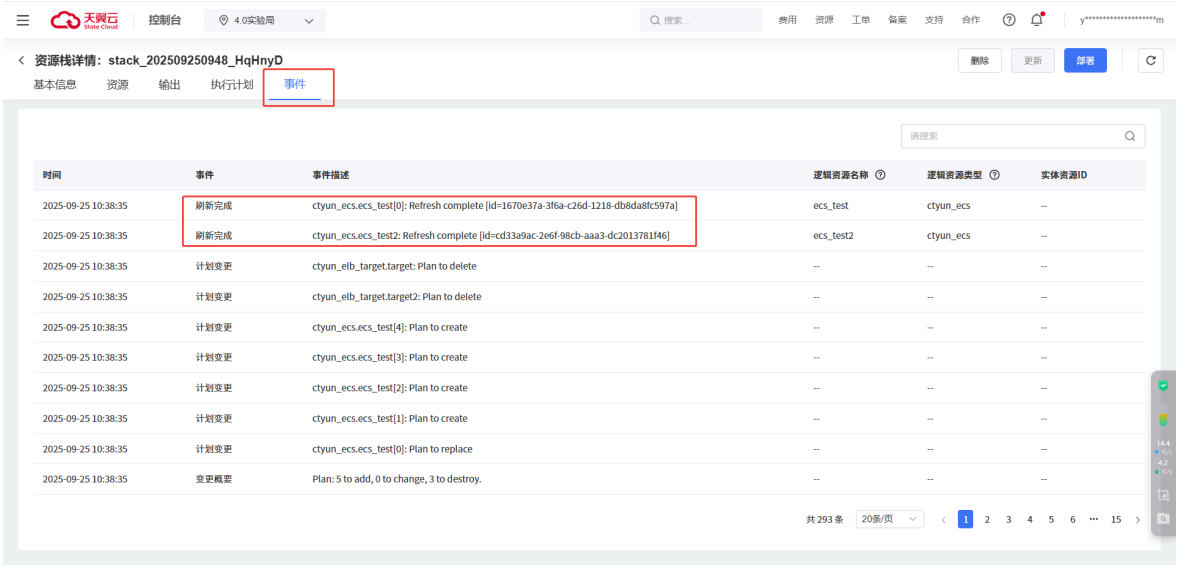
- 单击**确认**，平台将创建一个执行计划，待执行计划状态由“创建中”跳转为“创建成功，待部署”后，您可继续部署资源栈。部署资源栈前，您可在执行计划详情中，确认资源变更和价格变更内容后，再部署资源栈。



8. 提交表单时，您可选择单击**直接部署资源栈**，此时平台将直接部署资源栈，而不会创建执行计划，即不会记录您本次部署的变更内容。此时您可在**资源栈详情的事件页面**查看部署状态。部署资源栈请参考部署资源栈。

注意：直接部署将会立即创建资源并产生相关费用，该操作为危险操作，不建议执行，请谨慎确认后再操作。





查询资源栈

操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **资源栈管理**。
4. 单击资源栈，进入**资源栈详情**。

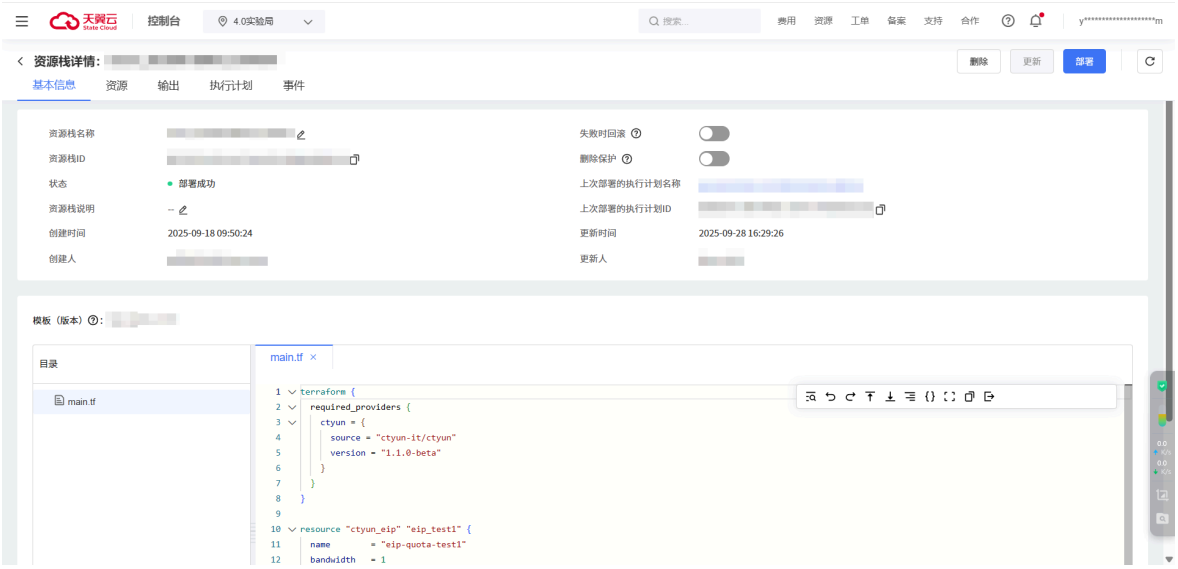
资源栈详情

基本信息

1. 展示资源栈基本信息，展示当前资源栈的模板及版本信息，展示最近一次部署成功的执行计划信息
2. 支持设置失败时回滚：

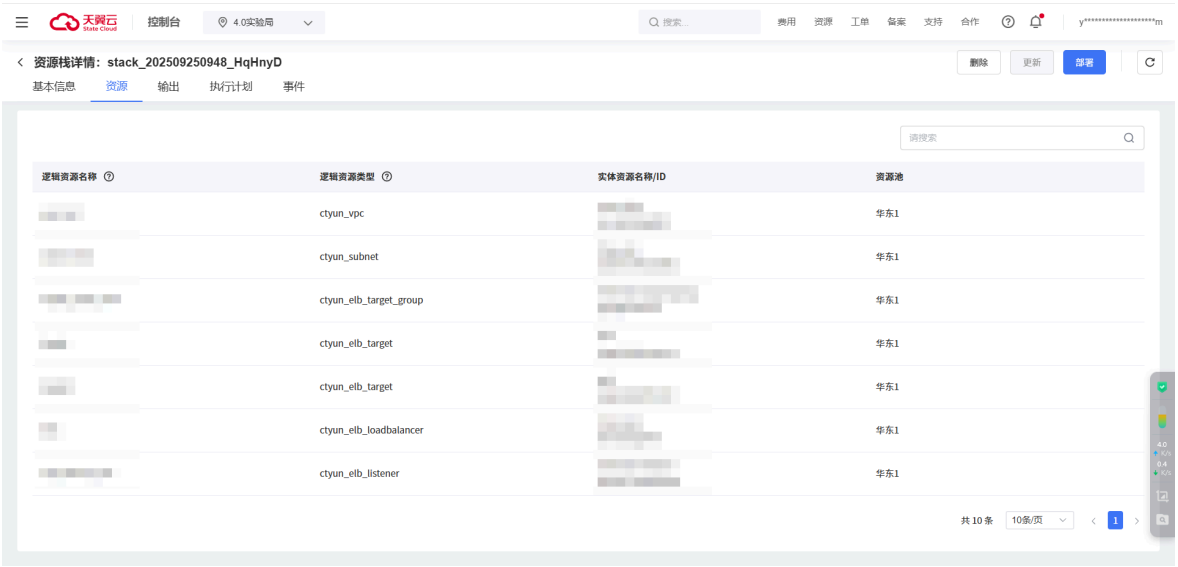
- 开启： 资源栈创建失败时，将删除已成功创建的资源，资源栈回滚至上一次创建成功的状态。
- 关闭： 资源栈创建失败时，将保留已成功创建的资源。

3. 支持设置删除保护：删除保护打开时，将无法手动删除资源栈，需手动关闭该配置后才可继续删除。



资源列表

展示最近一次部署成功的资源栈内所有资源。



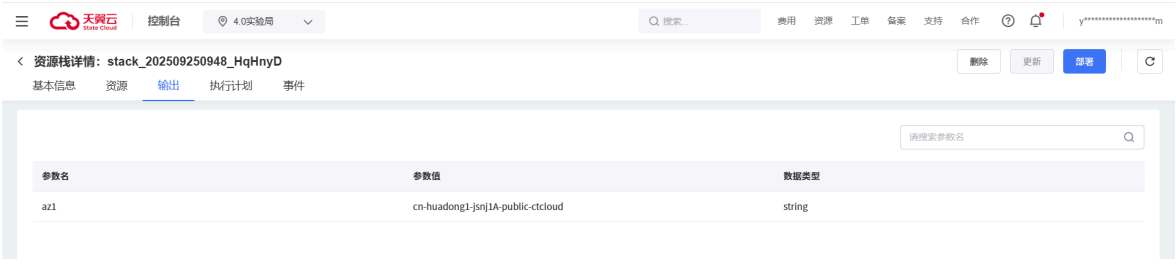
通过一个实际的模板示例，解释名词如下：

- 逻辑资源名称：指模板中定义的资源名称，即“vpc_logicName”
- 逻辑资源类型：指模板中定义的资源类型，即“ctyun_vpc”
- 实体资源名称/ID：指通过部署资源栈实际创建出来的资源，例如在虚拟私有云控制台上可查看到的vpc资源。

```
resource "ctyun_vpc" "vpc_logicName" {
  name        = var.vpc_name
  cidr        = "192.168.0.0/16"
  description = "terraform测试使用"
  enable_ipv6 = true
}
```

输出列表

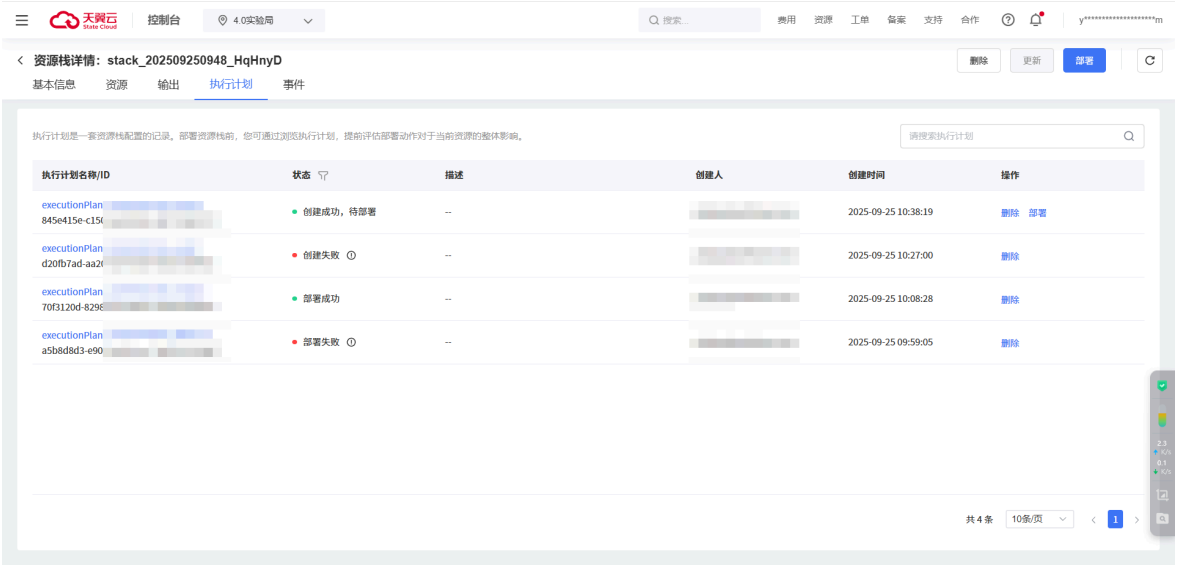
展示在模板部署完成后，由资源栈对外暴露的关键结果信息，对应模板中的output部分。



执行计划列表

展示资源栈所有执行计划的列表。

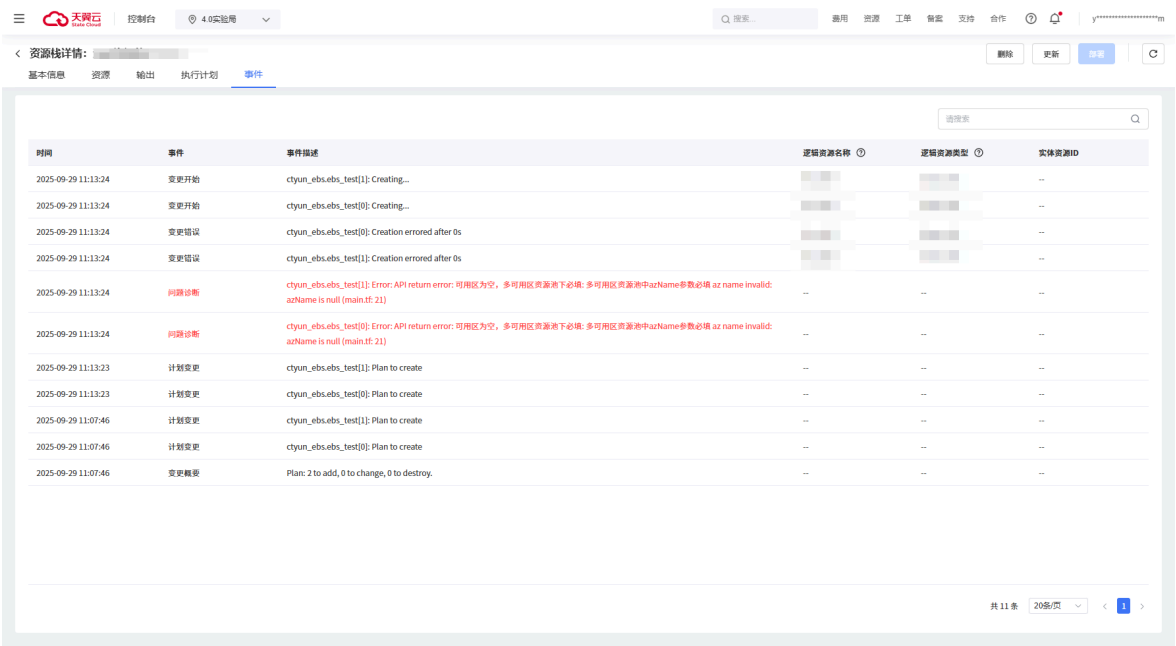
- 执行计划是一套资源栈配置的记录。部署资源栈前，您可通过浏览执行计划，提前评估部署动作对于当前资源的整体影响。
- 执行计划的部署动作是一次性操作，一旦部署过执行计划，将无法再次部署，仅支持删除
- 部署执行计划：执行计划在 创建成功，待部署 状态下，支持部署。此时单击部署，确认信息后可立刻部署执行计划。
  - 部署执行计划前，您可前往执行计划详情确认变更信息，如确认部署，您可单击部署执行，参考部署资源栈。
- 删除执行计划：删除执行计划仅删除执行计划记录，不影响资源栈的运行。您可放心删除执行计划。



事件列表

展示资源栈部署的事件信息。

- 事件列表每5秒自动刷新一次，同时页面也支持手动刷新，确保您可以看到实时部署信息
- 对于部署失败事件支持高亮提醒



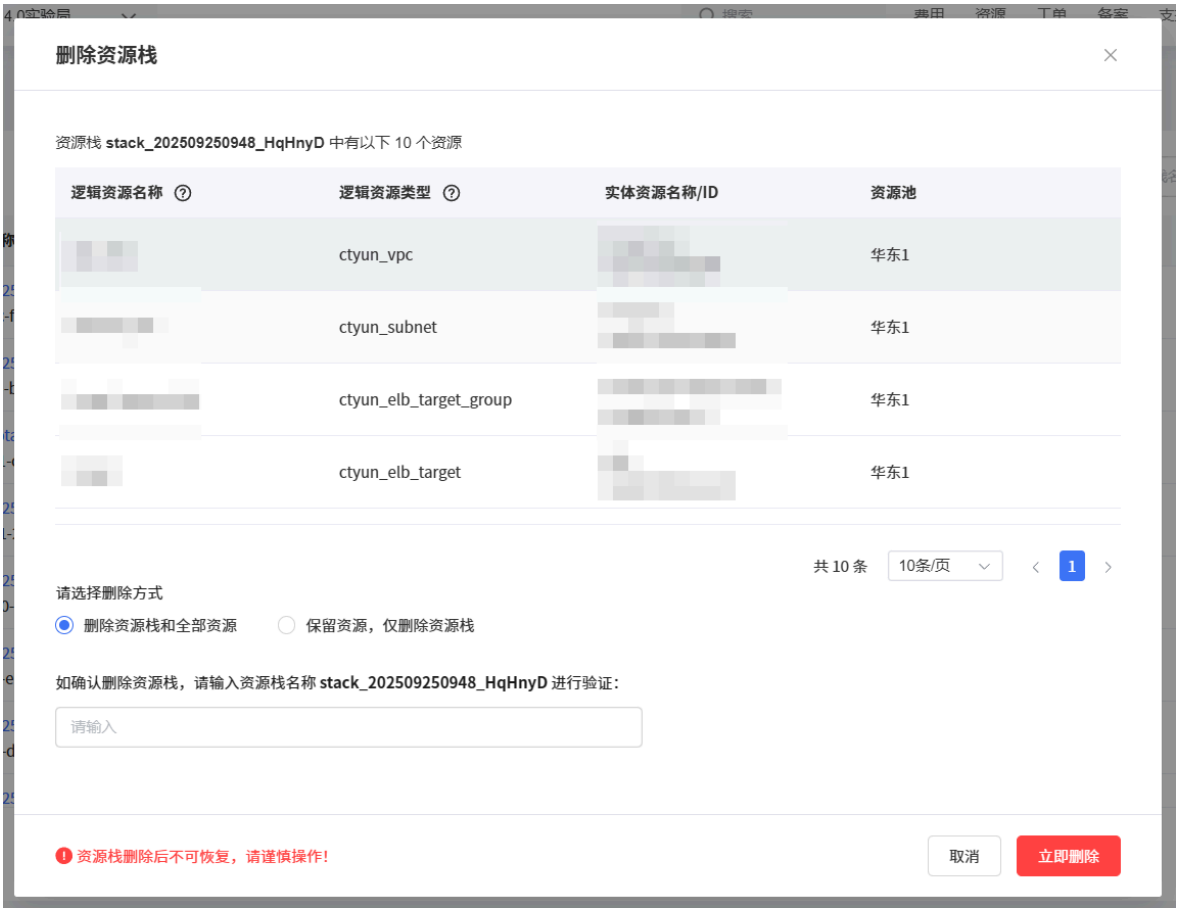
删除资源栈

限制条件

删除保护打开时，将无法手动删除资源栈，需手动关闭该配置后才可继续删除。

操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **资源栈管理**。
4. 在资源栈列表中直接单击**删除**，或在**资源栈详情**中单击**删除**。
5. 您可选择**删除资源栈及其全部资源**，或选择**保留资源，仅删除资源栈**。请在确认资源栈全部资源后再操作删除。



查询执行计划

执行计划是一套资源栈配置的记录。部署资源栈前，您可通过浏览执行计划，提前评估部署动作对于当前资源的整体影响。

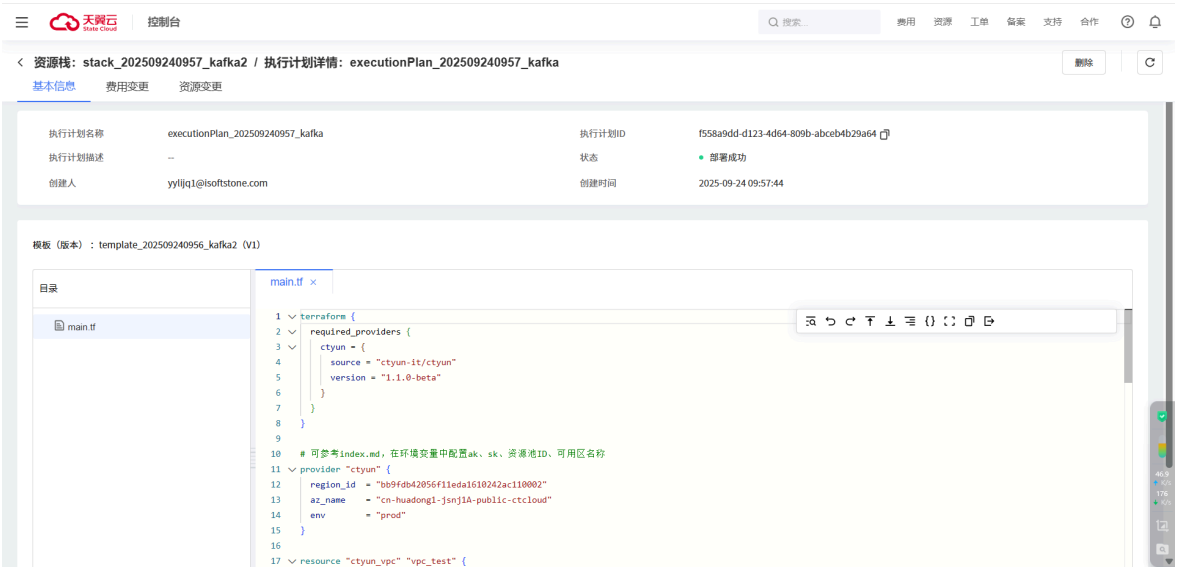
操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **资源栈管理**。
4. 单击资源栈，进入**资源栈详情**。
5. 在资源栈详情中的执行计划页面，单击执行计划，进入 **执行计划详情**。

执行计划详情

基本信息

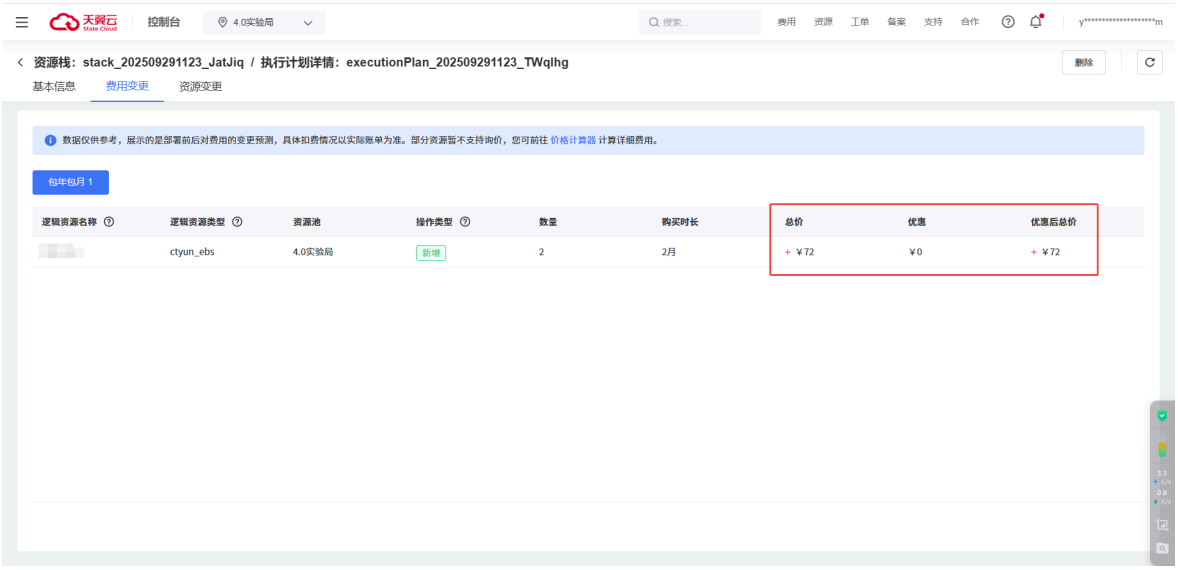
展示执行计划基本信息，展示当前执行计划的模板及版本信息。



费用变更

展示部署执行计划前后对费用的变更预测信息。

1. 数据仅供参考，具体扣费情况以实际账单为准。
2. 部分资源暂不支持询价，您可前往价格计算器计算详细费用。



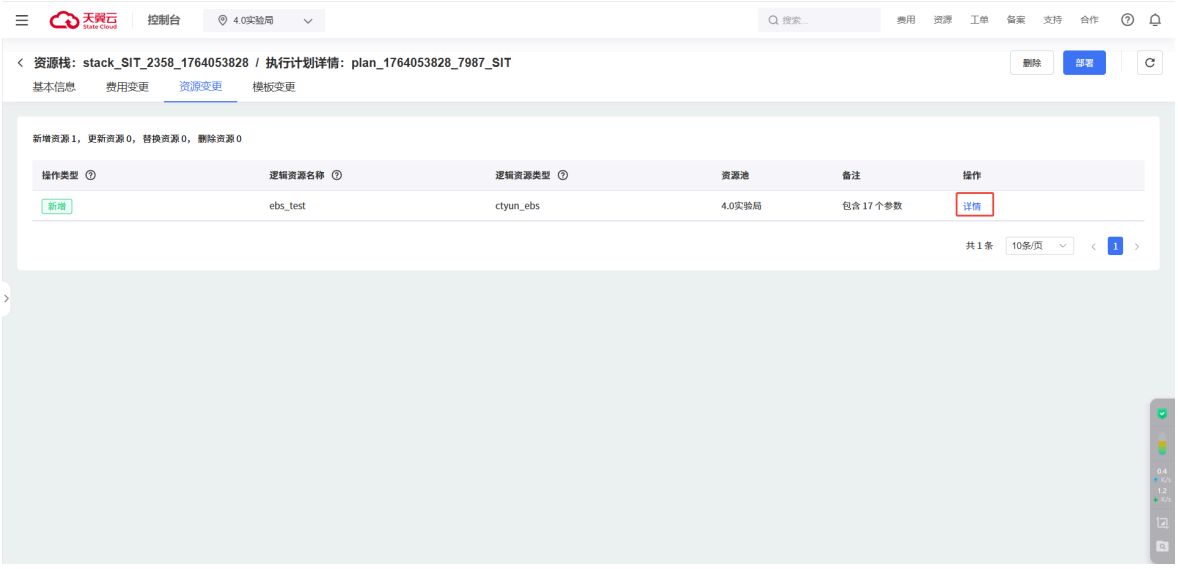
资源变更

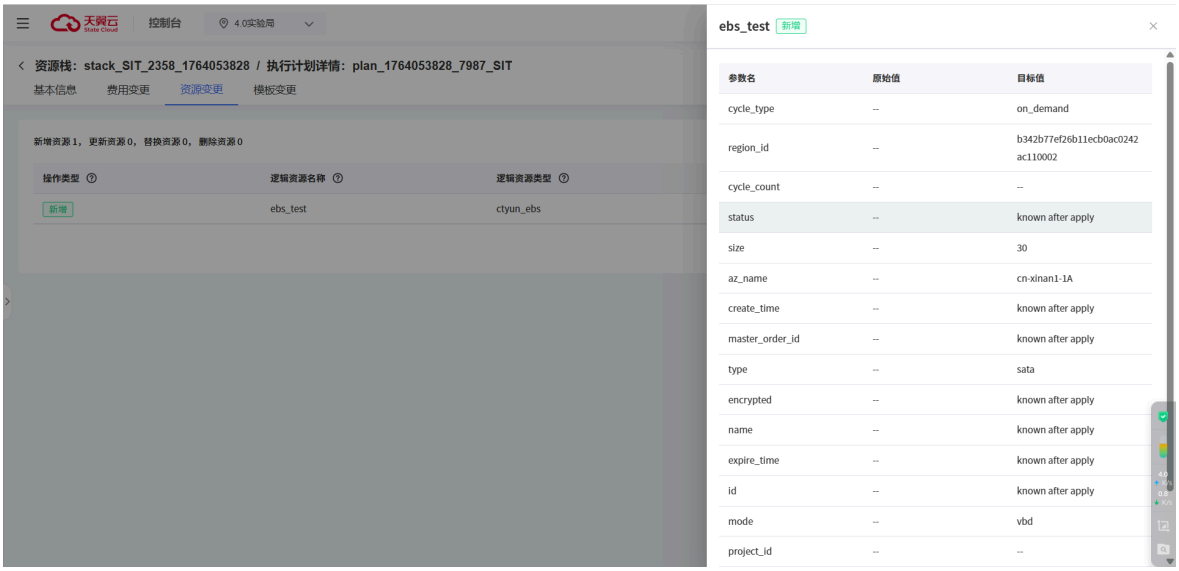
展示部署执行计划前后资源的变更情况。

变更类型包括：

- 新增：新增目标资源
- 更新：不销毁资源，直接更新配置
- 替换：销毁资源，并创建一个同类型的新资源
- 删除：删除目标资源

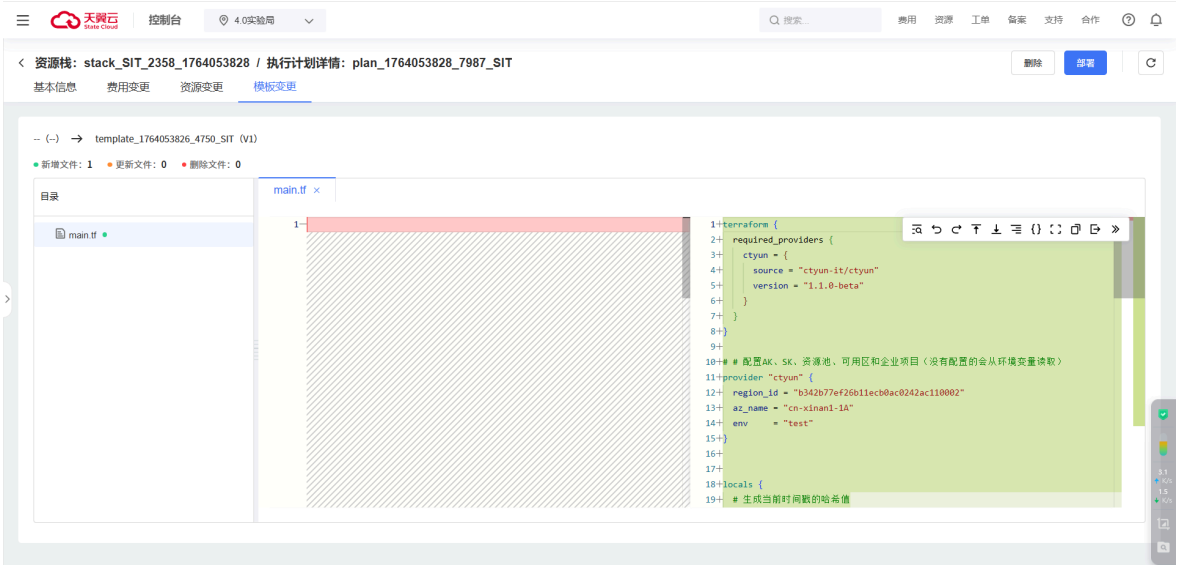
您可通过单击**详情**，查看变更参数列表。





模板变更

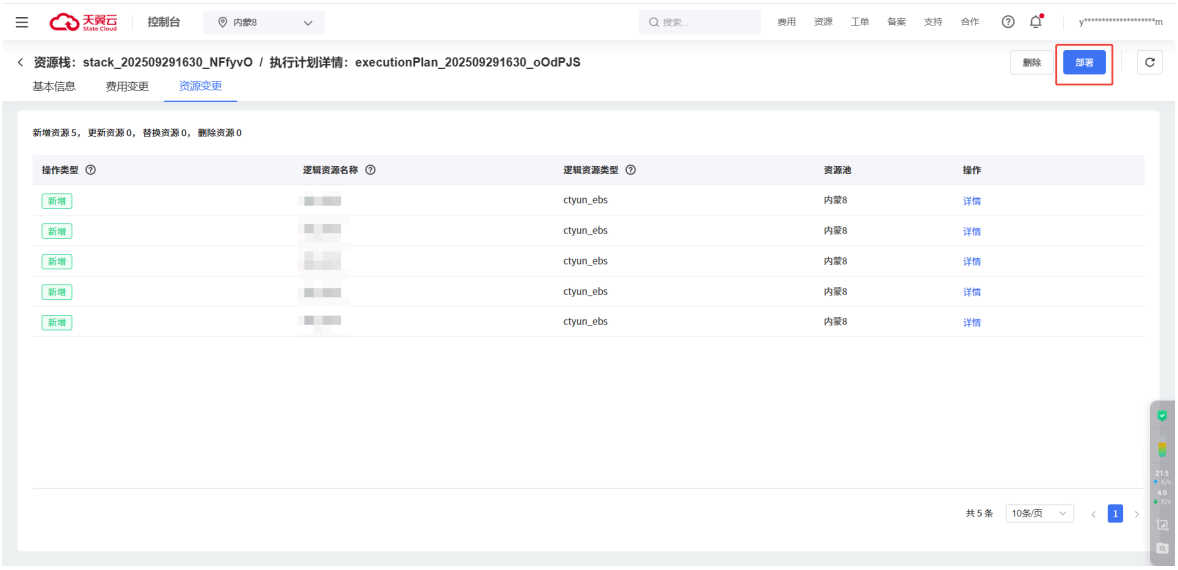
展示部署执行计划前后的模板变更情况，您可在页面上看到文件数量变化和模板内容变化。

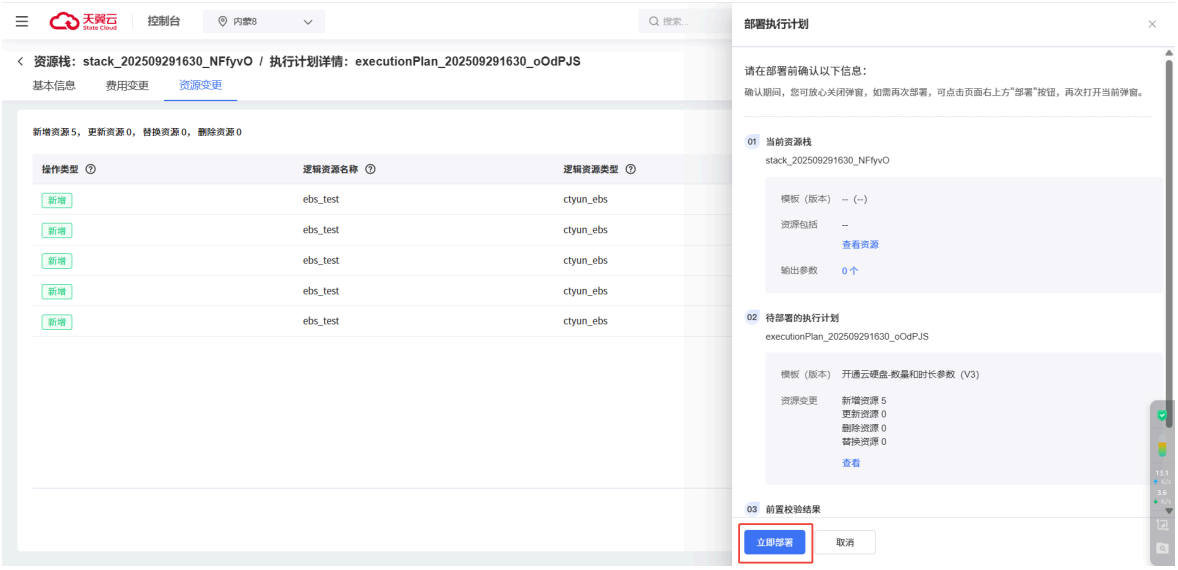


部署执行计划

执行计划在 **创建成功，待部署** 状态下，支持部署。此时单击**部署**，确认信息后可立刻部署执行计划。

部署执行计划前，您可前往执行计划详情确认变更信息，如确认部署，您可单击部署执行，参考部署资源栈。





删除执行计划

删除执行计划仅删除执行计划记录，不影响资源栈的运行。您可放心删除执行计划。

部署资源栈/执行计划

直接部署资源栈

创建 / 更新资源栈后，您可选择直接部署资源栈，此时平台将直不会创建执行计划，即不会记录您本次部署的变更内容。

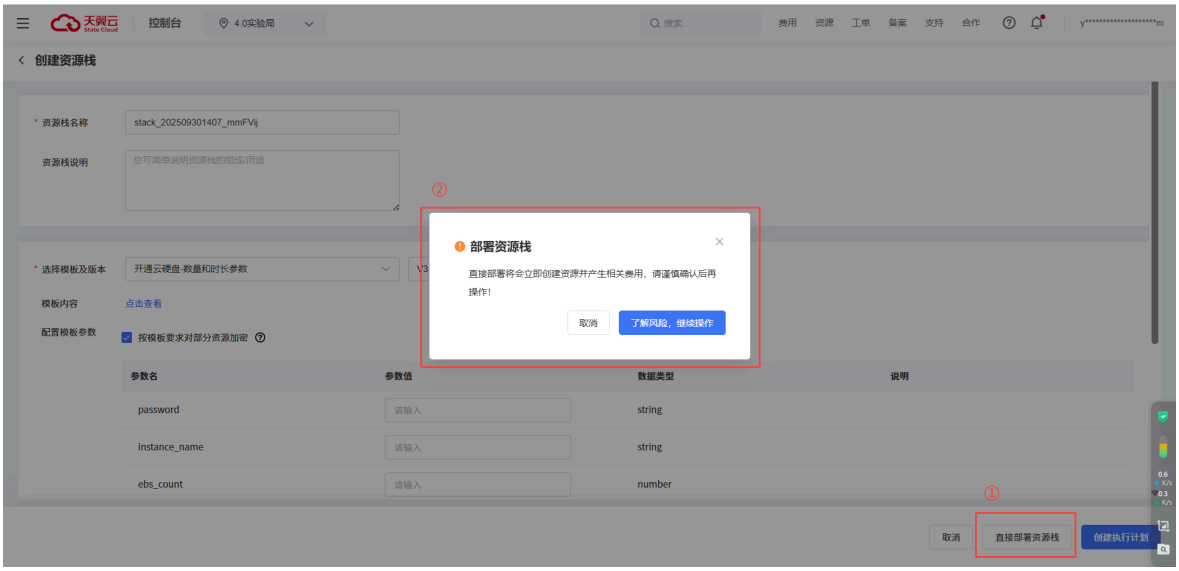
前置步骤

1. 已在**模板管理**中有已创建成功的模板。参考创建模板。

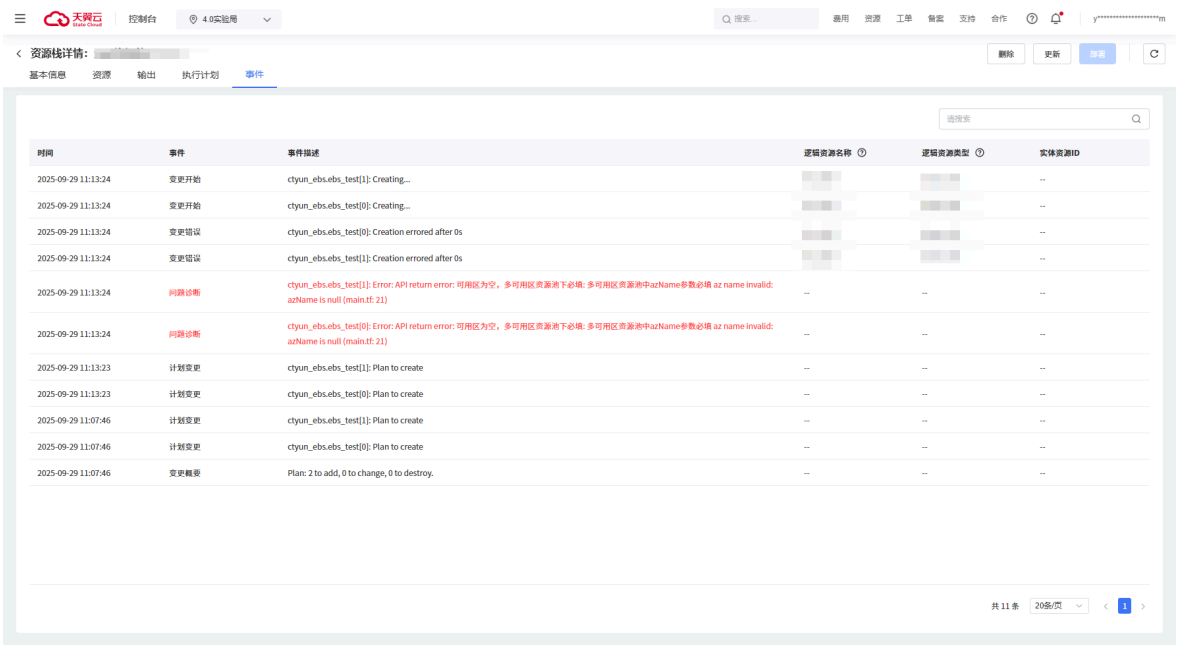
操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **资源栈管理**。
4. 在**资源栈管理**页面，单击**创建资源栈**。
   a. 如果您已有目标资源栈，您可在单击**更新**，进行更新资源栈。参考更新资源栈
5. 在**创建资源栈**页面，可以根据提示配置等基本参数，具体参数说明请参考创建资源栈。
6. 提交表单时，您可选择单击**直接部署资源栈**，此时平台将直接部署资源栈，而不会创建执行计划，即不会记录您本次部署的变更内容。

注意：直接部署将会立即创建资源并产生相关费用，该操作为危险操作，不建议执行，请谨慎确认后再操作。



7. 部署任务开始后，页面将自动跳转资源栈详情的事件页面，您可在事件列表中查看实时部署事件。



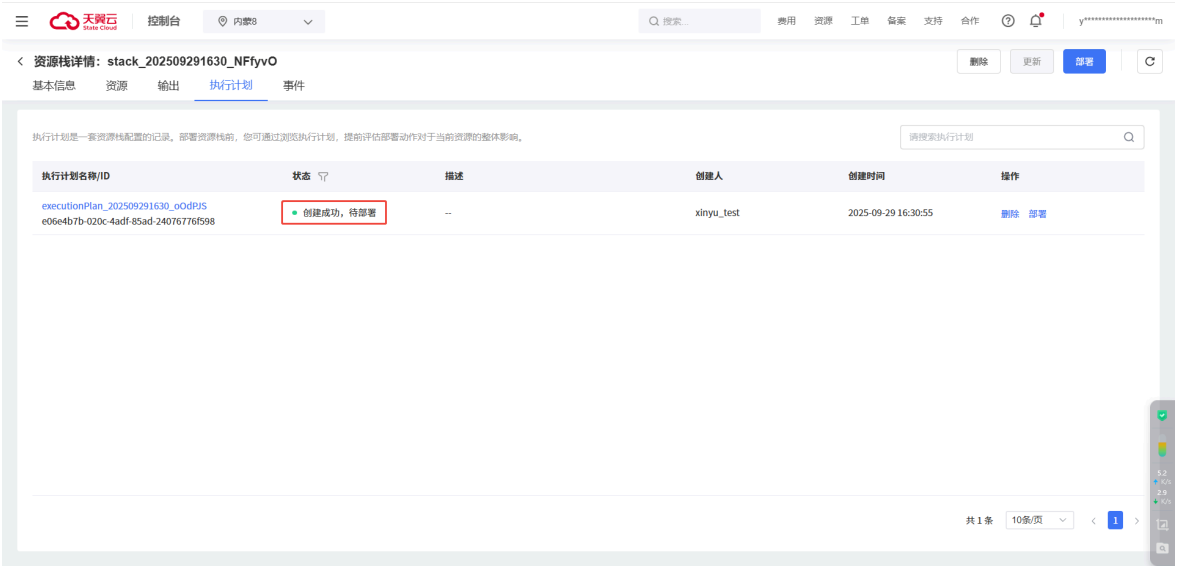
通过部署执行计划，实现部署资源栈

执行计划是一套资源栈配置的记录。部署资源栈前，您可通过浏览执行计划，提前评估部署动作对于当前资源的整体影响。

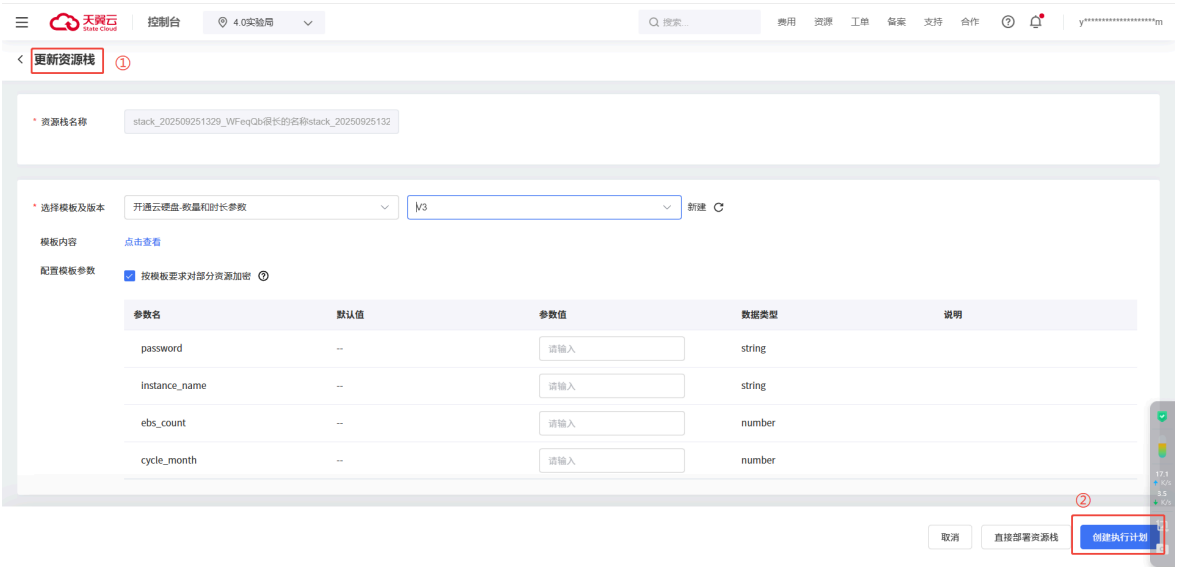
部署执行计划的过程，就是部署资源栈的过程，您可通过部署执行计划，实现部署资源栈。

前置条件

1. 已在资源栈管理中有已创建成功的资源栈，参考创建资源栈。
2. 资源栈中有状态为**创建成功，待部署**的执行计划。



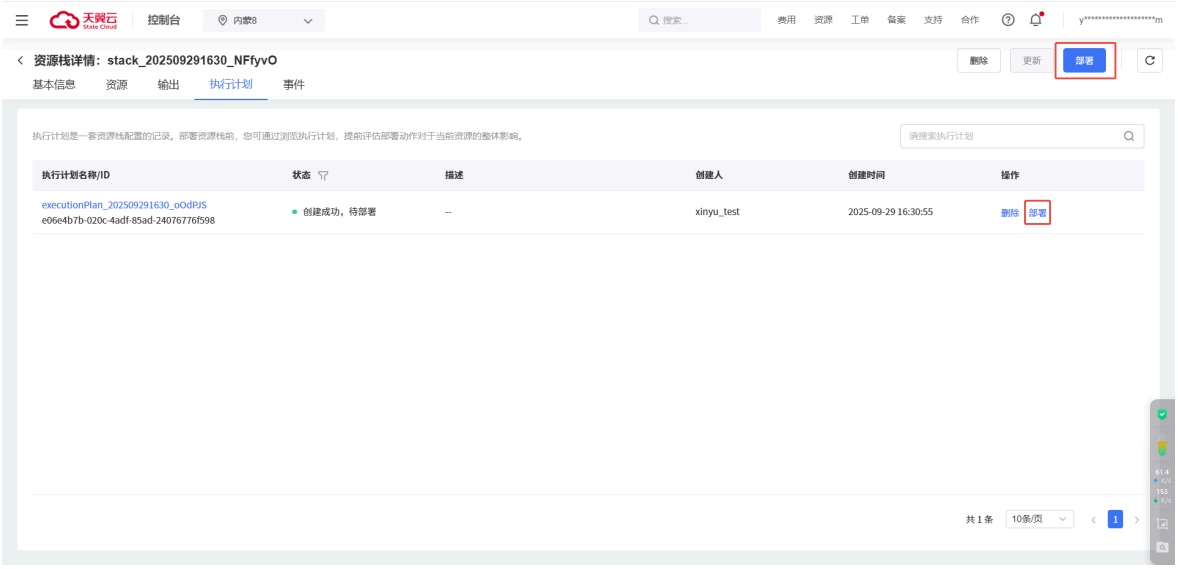
- 如果没有，请先更新资源栈，并选择创建执行计划，参考更新资源栈。

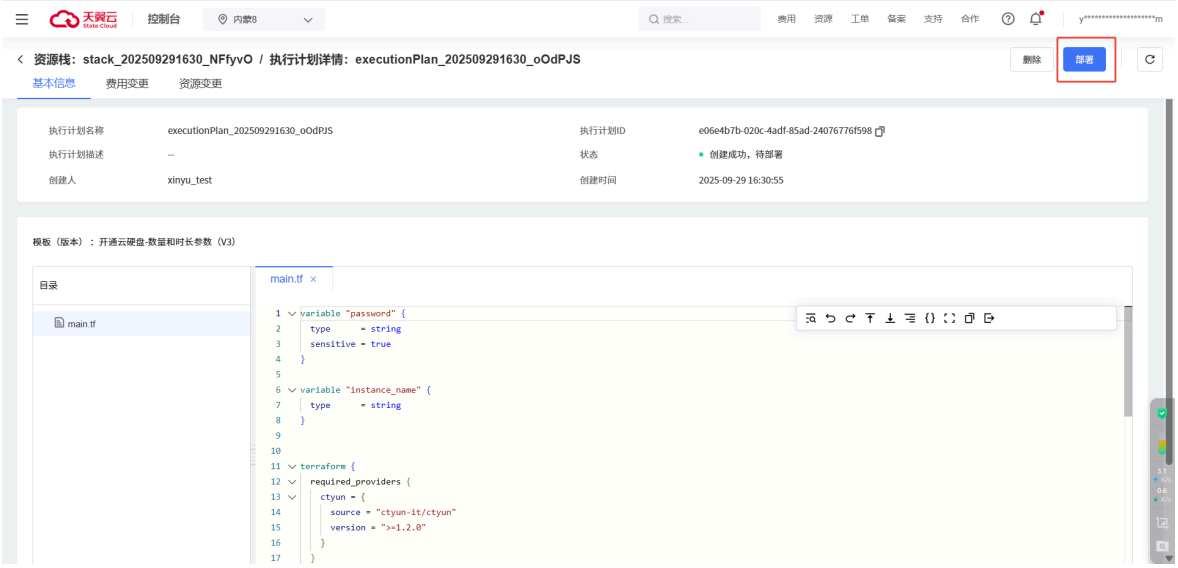


操作步骤

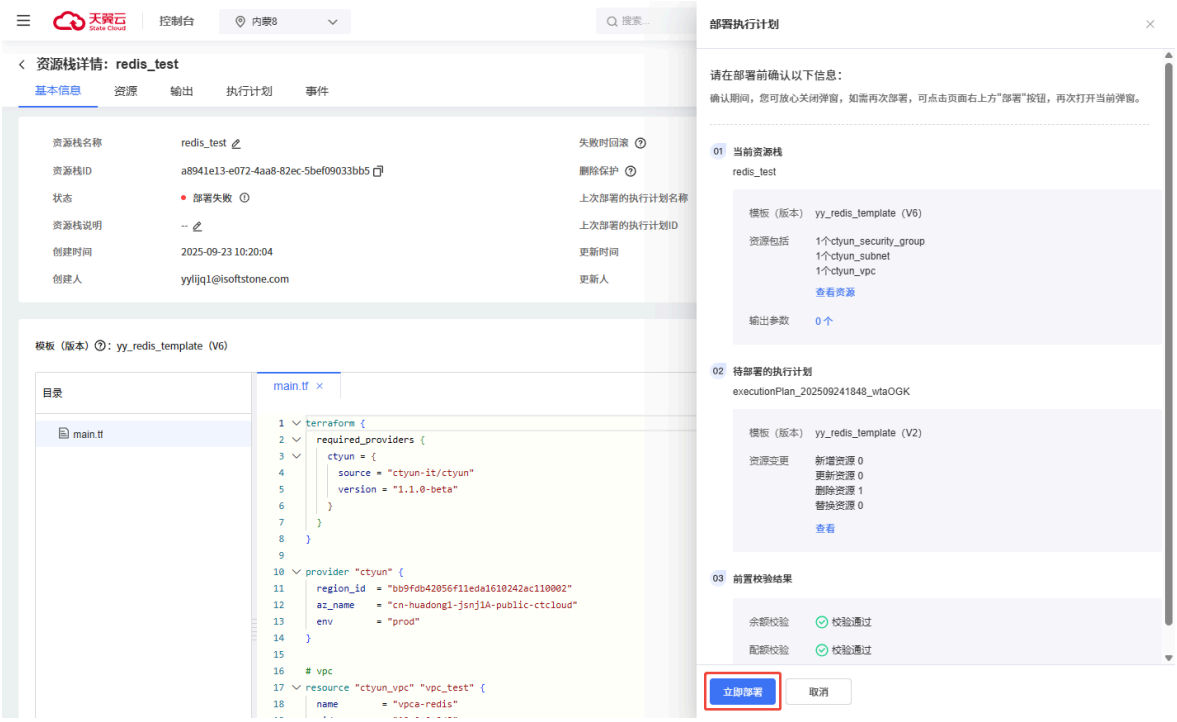
1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **资源栈管理**。

4. 在资源栈列表中直接单击**部署**，或在**资源栈详情**中单击**部署**，或在资源栈详情中的**执行计划列表**中直接单击**部署**，或在**执行计划详情**中单击**部署**。

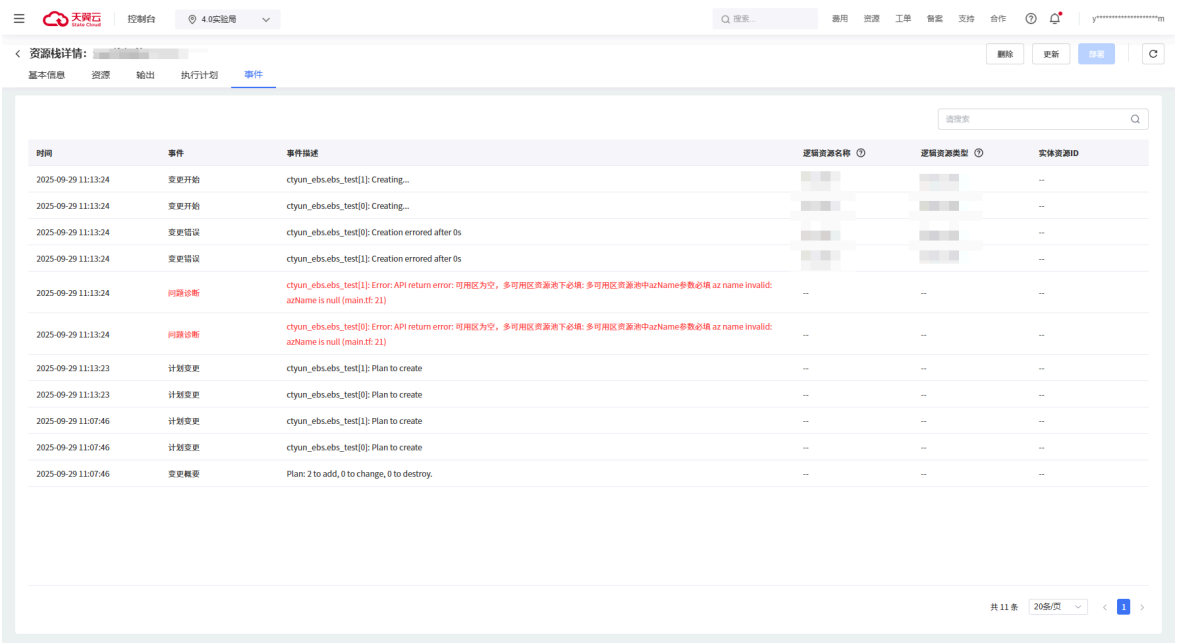




5. 您可在确认部署侧边栏中的信息后，单击**立即部署**。



6. 部署任务开始后，页面将自动跳转资源栈详情的事件页面，您可在事件列表中查看实时部署事件。



模板参考

概述

资源编排ROS 模板是用于创建、更新天翼云 资源编排ROS 资源栈的标准化配置脚本，作为资源部署的核心 “蓝图”，它采用 Terraform 的 HCL（HashiCorp Configuration Language）作为配置语法。

借助 HCL 的结构化语法特性，能够清晰定义资源栈所需的资源类型（如计算实例、存储服务、网络配置等）、具体参数（如实例规格、存储容量、网络地址等）及资源间依赖关系，确保资源部署逻辑具备可追溯性和可读性。

基于模板，用户无需手动逐一配置资源，可快速生成一致性的资源栈，同时支持后续对资源栈进行统一更新与管理，有效简化天翼云资源部署流程，提升操作效率与配置准确性，适配从简单到复杂场景的资源自动化管理需求。

版本与支持资源

支持的Provider版本

版本	Terraform Registry
1.2.0	https://registry.terraform.io/providers/ctyun-it/ctyun/1.2.0/docs

支持询价的资源

资源	模板名称定义
弹性云主机	ctyun_ecs
云硬盘	ctyun_ebs
物理机	ctyun_ebm
VPC终端节点	ctyun_vpce
弹性IP	ctyun_eip
共享带宽	ctyun_bandwidth
公网NAT网关	ctyun_nat
分布式缓存服务Redis版	ctyun_redis_instance
弹性负载均衡	ctyun_elb_loadbalancer
关系数据库MySQL版	ctyun_mysql_instance
关系数据库PostgreSQL版	ctyun_postgresql_instance
文档数据库服务	ctyun_mongodb_instance

语法指南

基本语法

资源编排ROS配置语言兼容HCL语法，具有配置简单、可读性强等特点，并且兼容JSON语法。本文主要介绍HCL语言的基本语法及常见函数。支持Terraform 1.0.0 - 1.5.7。

资源编排ROS配置语言主要由参数(Argument)、块(Block)、表达式(Experssion) 和函数(Functions) 组成。

参数 (Argument)

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

```
image_id = "965424eaa-533a-419d-9b40-ad091b52"
```

块 (Block)

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

```
resource "ctyun_ecs_backup_policy" "test" {
  name       = "test"
  ...
  adv_retention {
   adv_day = 3
  }
}
```

在使用块时必须先声明其对应的类型，上述样例中 resource 和 adv_retention 均为块类型：

- resource 为**顶层块类型**，adv_retention 为**嵌套块类型**。
- HCL语言支持的顶层块类型包括：provider、resource、data、variable、output、module、locals 等关键字。

块标签在块类型之后定义，且数量由块类型决定：

- 上述样例中 resource 块类型包含两个标签：ctyun_ecs_backup_policy 和 test。
- 嵌套的 adv_retention 类型没有块标签。
- 块主体定义在块最后，由 { 和 } 字符封装，在块主体内可嵌套其他类型以实现不同的层级结构。

参数类型

HCL支持以下参数类型，按类别分为基本类型、集合类型和特殊类型。

基本类型

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

HCL支持自动类型转换：

- 可自动将 number 和 bool 类型转换为 string 类型。
- 若一个字符串能表示为数字或布尔类型的值，也可进行反向转换。

- 基本类型参数可直接赋值，示例如下：

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

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

集合类型

- **map(...)**：映射类型，以键值对(key-value pair)组合的数据元素集合。其中 key 为 string 类型，对应的值可是 string、number、bool等类型，且所有元素的值必须为同一类型。
- **list(...)**：列表类型，同类型数据元素的集合，元素可为基本类型和块类型，列表索引从0开始。
- **set(...)**：集合类型，类似列表类型，但元素无辅助标识符或顺序，且具有唯一性。

映射类型（map）

使用 { } 封装，表示形式灵活，规则如下：

- 键值对可使用等号 = 或冒号 : 连接。
- 若 key 不以数字开头，可不加双引号。
- 多行映射中，键值对之间可用换行符或逗号分隔。

推荐使用等号连接键值对并用换行符分隔，各格式示例如下：

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

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

列表类型（list）与集合类型（set）

列表类型和集合类型表示方式相同：

- 元素为基本类型的列表/集合：使用 [ ] 封装。
- 元素为块类型的列表/集合：使用重复块的形式表示。

示例如下：

```
# 基本类型的列表（集合表示方式相同）
security_groups = ["default", "internal"]

# 块类型的列表（集合表示方式相同）
network {
  uuid = "55534eaa-533a-419d-9b40-ec427ea7195a"
}
network {
  uuid = "ad091b52-742f-469e-8f3c-fd81cadf0743"
}
```

特殊类型

- null：空类型。若将参数设置为 null，表示该参数未填写，HCL会自动忽略该参数并使用默认值。

  null 在条件表达式中较为常见，示例如下：

```
var.test=="" ? null : var.test
# 含义：当var.test的值为""时，忽略该参数；否则使用var.test的值
```

其他语法

注释规则

- 单行注释：以 # 或 // 开头。
- 多行注释：以 /* 开始、以 */ 结束，**不支持嵌套块注释**。

编码格式

Terraform配置文件使用UTF-8编码，标识符、注释和字符串均支持非ASCII字符.

多行字符串

以 <<EOF 开头，中间为字符串内容，最后以 EOF 结尾（ EOF 可替换为其他字符）。示例如下：

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

}

常见函数

HCL支持丰富的内置函数，用于处理字符串、数值计算、加密、类型转换等操作，您可以通过函数名称进行调用，其语法如下： <####>(<##1>, <##2> ...)

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

字符串函数

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

数值计算函数

函数名称	函数描述	样例	运行结果
abs	计算绝对值	abs(-12.4)	12.4
max	计算最大值	max(12, 54, 6) max([12, 54, 6]...)	54 54
min	计算最小值	min(12, 54, 6) min([12, 54, 6]...)	6 6
log	计算对数	log(16, 2)	4
power	计算x的y次幂	power(3, 2)	9

集合函数

函数名称	函数描述	样例	运行结果
element	通过下标从列表中检索对应元素值	element(["One", "Two", "Three"], 1)	Two

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

类型转化函数

函数名称	函数描述	样例	运行结果
toset	将列表类型转换为集合类型	toset(["One", "Two", "One"])	["One", "Two"]
tolist	将集合类型转换为列表类型	tolist(["One", "Two", "Three"])	["One", "Two", "Three"]
tonumber	将字符串类型转换为数字类型	tonumber("33")	33
tostring	将数字类型转换为字符串类型	tostring(33)	"33"

编码函数

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

哈希和加密函数

函数名称	函数描述	样例	运行结果
sha256	计算字符串的SHA256值（16进制）	sha256("Hello, cloud!")	0ad167d1e3ac8e9f4e4f7ba83e92d0e3838177e959858631c77

函数名称	函数描述	样例	运行结果
sha512	计算字符串的SHA512值（16进制）	sha512("Hello, cloud!")	6eb6ed9fc4edffaf90e742e7697f6cc7d8548e98aa4d5aa7498
base64sha256	计算字符串的SHA256值，并转换为base64编码	base64sha256("Hello, cloud!")	CtFn0e0sjp9OT3uoPpLQ44OBd+lZhYYxx3DKrtjMXjo=
base64sha512	计算字符串的SHA512值，并转换为base64编码	base64sha512("Hello, cloud!")	brbtn8Tt/6+Q50LnaX9sx9hUjpiqTVqnSYLlzfeDWehKOunyJjE7Lex2W/HqTIOSLb/kphY21YXaRP+9fpAPVg==
md5	计算MD5值	md5("hello world")	5eb63bbbe01eeed093cb22bb8f5acdc3

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

文件操作函数

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

样式约定

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

核心样式约定

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

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

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

5. 元参数与元参数块位置：

   a. 将元参数(meta-arguments)放在块主体的顶部，并使用空行与其它参数隔开；

   b. 将元参数块(meta-argument blocks)放在块主体的最后，并用空行与其他块隔开。

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

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

 bucket = "bucket_demo"
 acl    = "public-read"
 tags = {
  foo = "bar"
  env = "test"
 }

 lifecycle {
  create_before_destroy = true
 }
}
```

6. 顶层块分隔：顶层块之间使用空行将彼此隔开。

7. 嵌套块分类：建议将相同类型的嵌套块放在一起，不同类型的嵌套块使用空行隔开。

参考资料：https://developer.hashicorp.com/terraform/language

表达式

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

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

运算符

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

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

在表达式中使用多个运算符时，将按照以下优先级从高到低进行求解：

1. !、- (负数)
2. *、/、%
3. +、- (减法)

4. >、>=、<、<=
5. ==、!=
6. &&
7. ||

条件表达式

条件表达式采用布尔表达式的值进行二选一，其语法可表示为：

```
condition ? true_value : false_value
```

该语句含义：如果 condition 为 true，结果为 true_value，否则为 false_value。

注意：条件表达式的结果可以是任意类型，但 true_value 和 false_value 的类型必须保持一致。

条件表达式的常见用法是使用默认值替换无效值，示例如下：

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

该语句含义：如果 var.a 的值不为空，则返回 var.a 的值，否则返回默认值 "default-a"。

for表达式

for表达式用于遍历集合类型（map、list、set）中的每个元素，并对元素进行处理，最后将结果输出为一个新的集合类型。

for表达式的输出结果类型由所使用的括号类型决定：

- 使用 [和]：生成一个列表
- 使用 { 和 }：生成一个映射/对象

示例1：列表元素转换（输出列表）

假设列表 var.mylist 的值为 ["AA", "BBB", "CCCC"]，通过for表达式将每个字符串元素转换为小写，输出列表：

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

示例2：列表转映射（输出映射）

基于上述 var.mylist，通过for表达式生成映射，映射关系通过 => 确定（键为原列表元素，值为小写后的元素）：

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

示例3：映射键值转换

假设映射 var.mymap 的值为 {element1="aaa", element2="bbb", element3="ccc"}，通过for表达式将映射中每个值转换为大写：

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

示例4：带条件过滤的for表达式

通过 if 语句对元素进行过滤，仅处理满足条件的元素。例如，遍历列表 var.list，仅将长度 ≥3 的元素转换为大写：

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

参考资料：https://www.terraform.io/docs/configuration/expressions.html

配置指南

Provider

Terraform文件类型通常以 .tf 或 .tf.json 结尾，文件中由 provider，resource，data source 和 variable 组成。

每个 Provider 代表一个服务提供商，Terraform 依赖云服务商 Provider 进行云资源管理交互。Provider 通过关键字 provider 进行声明，Provider 的配置参数请参考 这里 。

执行 terraform init 命令时会下载使用的插件，默认将从 Terraform 官方仓库下载最新版本的插件。

对于 Terraform 0.13 之后的版本，可以使用 "required_providers" 指定 Provider的 registry 源和版本。

每个 Terraform 模块必须声明其所需的提供程序，以便 Terraform 能够安装和使用它们。提供程序要求在 required_providers 代码块中声明。

Provider 的声明要求包括 source 和 version 和 local name（下方举例 ctyun）：

```
terraform {
  required_version = ">= 0.10"
  required_providers {
    ctyun = {
      source = "ctyun-it/ctyun"
      version = "1.2.0"
    }
  }
}
```

required_version 用于指定哪个版本的Terraform CLI 运行配置，当前支持Terraform 1.0.0 - 1.5.7。

required_providers 声明块中包含一个或多个provider声明。每个声明指定一个 source 和 version。

required_providers 都支持以下参数：

关键字	描述	类型	是否必填
source	provider的全局源地址	string	是
version	可用版本号，版本约束规则可参考 这里	string	是

多Provider定义

在 required_providers 声明后，可以使用 provider 块进一步声明provider配置。

```
provider "ctyun" {
  region_id = "200000001852"
  az_name  = "cn-huabei2-tj1A-public-ctcloud"
}
```

上述代码块的主体（{和之间}）包含天翼云的provider的配置参数。在本例中 region_id 和 az_name 都是由 ctyun 的provider的定义的。除了provider中提供的参数外，还有两个由 Terraform 本身定义且适用于所有 provider 块的“元参数”：

alias ： 多provider配置字段

version：已弃用，被 required_providers 中 version 取代

利用alias参数可以支持云平台的多个region，用于提供多资源池，多场景高可用的需求。可以通过创建多个provider声明块实现。对于每个额外的非默认配置，请使用alias元参数提供额外的名称段。例如：

```
# 默认provider配置；以 `ctyun_` 开头的资源将使用
# 它作为默认值，并且可以将其引用为 `ctyun`。
provider "ctyun" {
  region_id = "200000002368"
  az_name  = "cn-xinan1-xn1A-public-ctcloud"
}

# 华北2的provider配置；资源可以将其引用为 `ctyun.huabei2`。
provider "ctyun" {
  alias   = "huabei2"
  region_id = "200000001852"
  az_name  = "cn-huabei2-tj1A-public-ctcloud"
}
```

上述示例中声明了西南1和华北2资源池的provider，并对华北2资源池增加了别名，在资源中使用元参数 provider 来选择非默认的 provider块，其格式为：<ctyun>.<alias>

```
# 创建vpc，指定vpc创建在华北2资源池
resource "ctyun_vpc" "vpc_test" {
  provider = ctyun.huabei2
```

```
  name     = "tf-vpc"
  cidr     = "192.168.0.0/16"
  description = "terraform测试使用"
  enable_ipv6 = true
}
```

当然，天翼云支持在某个资源声明中直接指定region和可用区（az）信息，相比 alias + provider 的方式，这种方式更加灵活简单。

```
provider "ctyun" {
  region_id = "200000002368"
  az_name  = "cn-xinan1-xn1A-public-ctcloud"
}

# 创建vpc，指定vpc创建在华北2资源池
resource "ctyun_vpc" "vpc_test" {
  region_id  = "200000001852"
  name     = "tf-vpc"
  cidr     = "192.168.0.0/16"
  description = "terraform测试使用"
  enable_ipv6 = true
}
```

Resource

Resource声明

Resource 是 Terraform 语言中最重要的组成部分，通过关键字 resource 进行声明。每个resource代码块描述了一个或多个云资源，如ctyun_ecs 表示ECS，ctyun_vpc表示VPC等。terraform会根据多个resource定义的业务逻辑进行隐性或手动显性的关联，例如：在创建ctyun_ecs之前一定需要先将ctyun_vpc创建完成，resource关联属性将在本页后面介绍。

```
# 创建vpc
resource "ctyun_vpc" "vpc_example" {
  ...
}
# 创建subnet
resource "ctyun_subnet" "subnet_example" {
  ...
}

# 创建ecs
resource "ctyun_ecs" "ecs_example" {
  subnet_id      = ctyun_subnet.subnet_example.id
  vpc_id        = ctyun_vpc.vpc_example.id
  ...
}
```

上述resource块声明三个云资源，分别为：

- ctyun_vpc
- ctyun_subnet
- ctyun_ecs

resource 声明块初次执行会触发创建(create)云资源动作，后续再执行会触发更新(update)或者替换(replace)操作。

访问Resource属性

ctyun_ecs 资源引用了ctyun_vpc 和 ctyun_subnet 资源，引用语法为：<####>.<####>.<####>，具体可以表示为：ctyun_vpc.vpc_example 和 ctyun_vpc.vpc_example.id。每个 resource 声明模块中需要填充的字段信息可参考 这里，除 provider 定义的参数外，Terraform 本身还定义了一些适用于所有资源类型的元参数，具体可参考 Meta-arguments。

Resource依赖关系

隐式排序

配置中的大多数资源没有任何特定的关系，并且 Terraform 可以并行对几个不相关的资源进行更改。但是，某些资源必须在其他特定资源之后进行处理；有时这是因为资源的工作方式，有时资源的配置只需要另一个资源生成的信息。大多数资源依赖关系都是自动处理的。Terraform 会分析代码块中的每个 resource，以查找对其他对象的引用，并在创建、更新或销毁资源时将这些引用视为隐式排序要求。由于大多数对其他资源具有行为依赖关系的资源也会引用这些资源的数据，因此通常无需手动指定资源之间的依赖关系。创建ctyun_ecs就是典型的隐式排序，Terraform 根据引用关系优先创建ctyun_vpc和ctyun_subnet资源，最后再创建ctyun_ecs。

显式排序

但是某些依赖项无法在配置中隐式识别，在这类较为罕见的情况下：可以使用 depends_on 元参数显式声明，指定依赖关系。示例如下：

```
# 创建postgresql数据库账户
resource "ctyun_postgresql_account" "account_test" {
  ...
}

# datasource查询postgresql账户列表
data "ctyun_postgresql_accounts" "accounts" {
  depends_on = [ctyun_postgresql_account.account_test]
  ...
}
```

上述示例中，利用 depends_on 声明ctyun_postgresql_accounts资源依赖ctyun_postgresql_account的执行完成。实际效果是再postgresql实例创建好数据库账户后，才会执行查询postgresql账户列表的datasource。

Data Source

Data Source定义

Data Source是通过一种特殊Resource，其作用是访问资源的各种数据，可以查询且不限于某种资源列表、数据库支持的字符集、云主机机器规格等等，Data Source使用 data 关键字声明：

```
data "ctyun_mysql_instances" "mysql_list" {
 page_no  = 1
 page_size = 10
 name  = "mysql_example"
}
```

Data Source 结果引用

上述实例查询MySQL数据库实例列表，筛选条件是名为"mysql_example"，页码为1，页面大小为10。查询成功后，引用MySQL实例格式为：data.<####>.<##>.<####>

```
data "ctyun_mysql_instances" "mysql_list" {
 page_no  = 1
 page_size = 10
 name  = "mysql_example"
}

resource "ctyun_eip" "eip_example" {
 ...
}

# 名为mysql_example的mysql实例绑定eip
resource "ctyun_mysql_association_eip" "association_eip" {
 eip_id = ctyun_eip.eip_example.id
 inst_id = data.ctyun_mysql_instances.mysql_list[0].id
}
```

Variable

本地变量

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

- 输入变量就像函数参数。
- 输出变量就像函数返回值。
- 本地变量就像函数的临时局部变量。

本地变量声明

一组相关的局部变量可以在一个locals 块中一起声明：

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

本地变量的赋值不仅限于文本常量，它们还可以引用模块中的其他值以进行转换或组合，包括变量、资源属性或其他本地值：

```
locals {
 security_group_ids = concat(ctyun_security_group.sg1.id, ctyun_security_group.sg2.id)
}
```

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

本地变量的引用

声明了局部值，您就可以在表达式中引用它 local.<NAME>。

```
resource "ctyun_ecs" "ecs_test" {
 instance_name = local.service_name
 ...
}
```

输入变量

输入变量声明

输入变量是一种无需更改模块自身的源代码，可以自定义名称和输入值。输入变量允许您在不同的 Terraform 配置之间共享模块，从而使您的模块可组合且可重用。

在配置的根模块中声明变量时，可以使用 Terraform CLI 输入或者通过环境变量设置它们的值。在子模块中声明变量时，调用模块应该在module代码块中传递值。

```
variable "image_id" {
 type = string
}
```

```
variable "password" {
 type     = string
 sensitive  = true
 nullable   = false
 description = "手动输入密码"
}
```

```
variable "dns" {
  type    = list(string)
  default = [
    "114.114.114.114",
    "8.8.8.8",
    "8.8.4.4"
  ]
}


variable "az_info" {
  type = list(object({
    availability_zone_name  = string
    availability_zone_count = number
    node_type          = string
  }))
  default = [
    {
      availability_zone_name  = "cn-gs-qyi2-1a-public-ctcloud"
      availability_zone_count = 1
      node_type          = "master"
    }
  ]
}
```

上述是 variable 的定义，关键字 variable 后的标签是变量名称，该名称在同一模块的所有变量中必须是唯一的。变量名称由用户自定义，但用户需要注意一下字段不可作为变量名称：source, version, providers, count, for_each, lifecycle, depends_on 和 locals。关于 {} 中定义的属性可参考**输入变量参数**的内容。

输入变量参数

Terraform CLI 为变量声明定义了以下可选参数

- default：配置 variable 的默认值，当配置默认值后，CLI交互时可不为变量赋值
- type：声明 variable 的值类型，如果未明确指定变量类型，则默认为 string。取值分为以下几种：string
  - number
  - bool
  - list(<TYPE>)
  - set(<TYPE>)
  - map(<TYPE>)
  - object(<ATTR NAME> = <TYPE>, ...)
  - tuple([<TYPE>, ...])
- description：变量的备注
- validation：输入值约束，可定义一个验证规则用于限制变量取值范围
- sensitive：将变量定义为敏感变量，定义后，Terraform UI不会将其打印出来。取值范围为true或false

- nullable：控制是否可以将值分配null给该变量，默认值为true，当 nullable 为时true，null 是变量的有效值，并且模块配置必须始终考虑变量值为 null 的可能性

输入变量的引用

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

```
variable "vpc_cidr" {
  type        = string
  description = "the CIDR of VPC"
}

resource "huaweicloud_vpc" "vpc_example" {
  name        = "tf-vpc"
  cidr        = var.vpc_cidr
  description = "terraform测试使用"
  enable_ipv6 = true
}
```

设置变量方式

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

命令行变量定义

要在命令行上指定单个变量，请 -var 在运行 terraform plan 和 terraform apply 命令时使用该选项：

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

您可以多次使用 -var 在单个命令中设置几个不同的变量。

变量定义 (.tfvars) 文件

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

terraform apply -var-file="examples.tfvars" 变量定义文件使用与 Terraform 语言文件相同的基本语法，但仅包含变量名称分配：

```
image_id = "ami-abc123"
dns      = [
  "114.114.114.114",
    "8.8.8.8",
    "8.8.4.4"
]
```

Terraform 还会自动加载一些变量定义文件：

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

对于以 .json 结尾的文件，需要使用 JSON对象表示：

```
{
  "vpc_name": "my_vpc"
  "az_info": [
     {
       availability_zone_name  = "cn-gs-qyi2-1a-public-ctcloud"
       availability_zone_count = 1
       node_type           = "master"
     }
  ]
}
```

环境变量定义变量

作为定义变量的备用方案，Terraform 在其自身进程的环境中搜索以 TF_VAR_ 声明变量的名称命名的环境变量。

在自动化运行 Terraform 时，或者使用相同变量连续运行一系列 Terraform 命令时非常有用。例如，在 bashUnix 系统的提示符下：

```
$ export TF_VAR_image_id=ami-abc123
$ terraform plan
...
```

变量定义优先级

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

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

输出变量

输出变量声明

输出变量可在命令行上提供用户需要的信息，并可公开其他 Terraform 配置使用的信息。输出变量类似于编程语言中的返回值，是一种对外公开部分信息的方式。

输出变量有多种用途：

- 子模块可以使用输出将其资源属性的子集公开给父模块。
- 根模块可以在运行 terraform apply/output 后 在CLI上打印特定值。

按照约定，输出变量通过“output”关键字进行声明：

```
output "az_name" {
 value = data.ctyun_zones.az.zones[0]
}
```

关键字 output 后面紧跟的标签是 output 名称，它必须是有效的标识符。在根模块中，此名称显示给用户；在子模块中，它可用于访问输出的值。

参数 value 采用一个赋值表达式，将结果返回给用户。在本例中，该表达式指的是将data source ctyun_zones中变量*az.zones[0]*输出。任何有效的表达式都可以作为输出值。

输出变量参数

output 块可以选择性地包含 description、sensitive 和 depends_on 参数，这些内容将在以下章节中进行描述。

- description：输出变量注释
- sensitive：限制是否在CLI 上输出
- depends_on：明确输出依赖关系

```
output "eip_addr" {
 value     = ctyun_eip.eip_example.address
 description = "EIP实例的公网地址"
}

output "db_password" {
 value     = ctyun_mysql_instance.mysql_examples.password
 description = "mysql实例的root密码"
 sensitive  = true
}


output "instance_ip_addr" {
 value     = ctyun_eip.eip_example.address
 description = "EIP实例的公网地址"
 depends_on = [
  ctyun_security_group_rule.sg_rule_example,
 ]
}
```

注意：标记为敏感项的输出变量在输出时会自动被隐藏，但其输出值仍然可以通过以下方式可见：

- 输出变量的值记录在 state 文件中，其值对所有能够访问state 文件的用户均可见.
- 子模块中敏感输出变量值被父模块调用，通过父模块的相关输出和资源引用后可以在CLI中显示。

Meta-arguments

depneds_on

在resouce文档中，我们提及了 depends_on。使用 depends_on 元参数来处理 Terraform 无法自动推断的隐藏资源或模块依赖关系。仅当某个资源或模块依赖于另一个资源的行为，但在其参数中不访问该资源的任何数据时，才需要显式指定依赖关系。depends_on 的表达式是依赖资源的地址列表。例如需要搭建一台ecs和mysql实例的小型服务端，mysql实例必须在ecs创建成功后才能部署。根据上述要求，具体实现如下：

```
resource "ctyun_ecs" "ecs_example" {
  ...
}

resource "ctyun_mysql_instance" "mysql_example" {
 depends_on = [ctyun_ecs.ecs_example]
 ...
}
```

count

默认情况下，resource 块会根据定义创建一个相应的云资源。但是，您希望需要同时创建多个类似的对象（例如相配置的ecs实例），无需为每个资源单独编写 resource 块。Terraform 有两种方法可以实现这一功能： count 和 for_each。如果 resource 块包含一个 count 值为整数的参数，Terraform 将创建那么多实例。

```
resource "ctyun_ecs" "ecs_example" {
  count      = 3
  instance_name = "tf-ecs-${count.index}"
  ...
}
```

上述示例可以创建 3 个 ctyun_ecs 资源。但按照要求，ecs 示例的名称是不允许相同的，可以利用 ${count.index} 对每个资源进行标记区分，这是一个从0开始计数的索引值，避免重复。除名称以外，还可以通过${count.index}访问数组，具体示例如下：

```
# 1. 创建2个vpc，名称分别为vpc_examples_0,和vpc_examples_1,并且我希望vpc的cidr分别不同
# 2. 定义一个variable，用于存储不同的cidr
variable "cidr_list" {
  type    = list(string)
  default = ["192.168.0.0/16", "172.16.0.0/16"]
}
# 3. 创建2个名称不同，cidr不同的vpc
resource "ctyun_vpc" "vpc_examples" {
  count = 2
  name  = "vpc_examples_${count.index}"
  cidr  = var.cidr_list[count.index]
  ...
}
```

count创建的资源引用方式本文也介绍下，为区分借助 count 创建的资源实例。实例的引用由索引标识，从0开始。具体引用方法如下：

- <RESOURCE TYPE>.<NAME>指的是引用资源块，借用上述vpc示例，想要引用vpc资源块可以表示为：ctyun_vpc.vpc_examples
- <RESOURCE TYPE>.<NAME>[INDEX]指的是引用具体单个实例，如果我先引用并输入第二个vpc实例的id，可以定义为：

```
output "vpc0_id" {
 value = ctyun_vpc.vpc_examples[1].id
}
```

count 和 for_each 这两个元参数无法同时在一个 resource 块中使用。所以关于 count 和 for_each 的抉择，可以根据如下准则去确定：

- 如果您的实例几乎相同，count 则是合适的。如果它们的某些参数不能直接简单的遍历获取到，则使用 for_each 更合理。

我们可以使用 for 语句解释 count 和 for_each 的区别：

```
# count = 10
for (int i = 0; i < 10; i++){
 resource块
}

# for_each
regions = ["华东1", "华北2", "华南2", "西南1", ...]
for (int i = 0; i < length(regions); i++){
 region = regions[i]
 resource 块
}
```

for_each

元参数 for_each 与 count 相似，可以同时创建多个配置相似的云资源。for_each 参数通过遍历一组集合，利用集合中每个元素为 resouce 块创建相应属性的实例，每个实例都会单独创建、更新或销毁。

for_each 利用字符串集合创建资源示例

```
resource "ctyun_mysql_account" "account_examples" {
 for_each = toset( ["Todd", "James", "Alice", "Dottie"] )
 name    = each.key
}
```

for_each 利用 map 创建资源示例

```
resource "ctyun_vpc" "vpc_examples" {
 for_each = {
  vpc_1 = "192.168.0.0/16"
  vpc_2 = "172.16.0.0/16"
 }
 name  = each.key
 cidr  = each.value
}
```

上述示例出现了 each.key 和 each.value 两个属性，当 for_each 遍历的是集合，而并非 map 时，each.value = each.key

for_each 使用限制：

- map的键（或一组字符串的所有值）必须是***已知值***
- for_each的 key 不能取 uuid、bcrypt 或 timestamp 这类函数的结果
- for_each 不能取敏感值进行遍历

for_each 的引用：

利用 for_each 创建资源实例组后，Terraform 会区分块本身和与其关联的多个资源。实例由提供给 的值中的映射键（或集合成员）标识 for_each。

- <RESOURCE TYPE>.<NAME>（例如 ctyun_vpc.vpc_examples）可以引用资源块。
- <RESOURCE TYPE>.<NAME>[KEY]（例如 ctyun_vpc.vpc_examples["vpc_1"]）可以引用个别实例

provider

利用 alias 参数可以支持云平台的多个region，用于提供多资源池，多场景高可用的需求。可以通过创建多个 provider 声明块实现。对于每个额外的非默认配置，请使用 alias 元参数提供额外的名称段。例如：

```
# 默认provider配置；以 `ctyun_` 开头的资源将使用
# 它作为默认值，并且可以将其引用为 `ctyun`。
provider "ctyun" {
 region_id = "200000002368"
 az_name  = "cn-xinan1-xn1A-public-ctcloud"
}

# 华北2的provider配置；资源可以将其引用为 `ctyun.huabei2`。
provider "ctyun" {
 alias  = "huabei2"
 region_id = "200000001852"
 az_name  = "cn-huabei2-tj1A-public-ctcloud"
}
```

上述示例中声明了西南1和华北2资源池的provider，并对华北2资源池增加了别名，在资源中使用元参数 provider 来选择非默认的 provider块，其格式为：<ctyun>.<alias>

```
# 创建vpc，指定vpc创建在华北2资源池
resource "ctyun_vpc" "vpc_test" {
 provider = ctyun.huabei2
 name     = "tf-vpc"
 cidr     = "192.168.0.0/16"
 description = "terraform测试使用"
 enable_ipv6 = true
}
```

lifecycle

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

create_before_destroy

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

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

prevent_destroy

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

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

ignore_changes

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

```
resource "ctyun_ecs" "ecs_examples" {
  ...
  lifecycle {
    ignore_changes = [
      instance_name,
    ]
  }
}
```

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

```
resource "ctyun_ecs" "ecs_examples" {
  ...
  lifecycle {
    ignore_changes = all
  }
```

```
}
```

replace_triggered_by

Terraform 1.2版本新增，当任何引用项发生更改时，替换资源。

```
resource "ctyun_ecs" "ecs_examples" {
 # ...
 lifecycle {
  replace_triggered_by = [
   ctyun_vpc.vpc_example.id
  ]
 }
}
```

模板约束与限制

核心约束与限制

1. 禁止使用的功能：不能使用Provisioners功能、Backend Configuration功能和Cloud功能。
2. required_providers限制:需要限制source，仅支持 "ctyun-it/ctyun"
3. Module Sources功能支持范围：可以使用Module Sources功能，但仅支持Local Modules。
4. 禁用系统的参数：部分系统参数禁止使用，具体清单如下
   - path.module
   - path.root
   - path.cwd
   - terraform.workspace
5. 禁止使用的函数：部分函数禁止使用，具体清单如下：
   - abspath
   - basename
   - dirname
   - file
   - filebase64
   - filebase64sha256
   - filebase64sha512
   - fileexists
   - fileset
   - filemd5
   - filesha1
   - filesha256
   - filesha512
   - pathexpand
   - templatefile

重要警告

不建议使用 nonsensitive 方法输出敏感信息。随意使用此方法可能会导致本该被隐藏的敏感信息被服务明文打印出来，从而造成敏感信息泄露。

若必须进行输出，建议优先考虑编码后再输出（例如：nonsensitive(sha256(var.sensitive_value))）。

模板示例

创建一台包年包月云主机

```
terraform {
 required_providers {
  ctyun = {
   source  = "ctyun-it/ctyun"
   version = "1.2.0"
  }
 }
}

# 支持配置资源池、可用区和企业项目，若不配置资源池，则会使用页面选定的资源池
# ak/sk无需配置，会自动获取当前账号的ak/sk
provider "ctyun" {
 region_id = "bb9fdb42056f11eda1610242ac110002"
 az_name = "cn-huadong1-jsnj1A-public-ctcloud"
 env    = "prod"
}

# 创建vpc
resource "ctyun_vpc" "vpc_test" {
 name     = "vpc-for-ecs"
 cidr     = "192.168.0.0/16"
 description = "terraform测试使用"
 enable_ipv6 = true
}

# 在vpc下创建子网
resource "ctyun_subnet" "subnet_test" {
 vpc_id    = ctyun_vpc.vpc_test.id
 name     = "subnet-for-ecs"
 cidr     = "192.168.1.0/24"
 description = "terraform测试使用"
 dns = [
  "114.114.114.114",
  "8.8.8.8"
 ]
 enable_ipv6 = true
```

```
}

# 查询可用镜像
data "ctyun_images" "image_test" {
  name       = "CentOS Linux 8.4"
  visibility = "public"
  page_no    = 1
  page_size  = 10
}

# 查询可用规格
data "ctyun_ecs_flavors" "ecs_flavor_test" {
  cpu    = 2
  ram    = 4
  arch   = "x86"
  series = "C"
  type   = "CPU_C7"
}

# 导入密钥对
resource "ctyun_keypair" "keypair_test" {
  name       = "keypair-for-ecs"
  public_key = "ssh-rsa AAAAB3Nz
aC1yc2EAAAADAQABAAABAQDAjUnAnTid4wmVtajSmElMtHO3OvOyY81ybfswbUu9Gt83DVVzDnwb3rcQW1us8SeKm/gRINkgd
RAgfXAmTKR7AorYtWWc/tzb6kcDpL2E8Qk+n6cyFAxX
NoX2vXBr4kC9wz1uwjGyxoSlpHLIpscfIOEf652gM1SyfODehAJHj3JPMr8pvtPIUqsZI3JOGTUzxaA2JVCOLxQegphYYf2Tx
1NaITXNVEj3A11hk1nrFoJMmvIwIUkLmRuQcxuNAdxeLB7GXXVjKpnKIJL4L64dyA9GWa3Gb7gCJyRaBc5UhK4hT57wmukCr
}

resource "ctyun_ecs" "ecs_test" {
  instance_name   = "ecs-demo"
  display_name    = "ecs-demo"
  flavor_id       = data.ctyun_ecs_flavors.ecs_flavor_test.flavors[0].id
  image_id        = data.ctyun_images.image_test.images[0].id
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id          = ctyun_vpc.vpc_test.id
  cycle_type      = "month"
  cycle_count     = 1
  subnet_id       = ctyun_subnet.subnet_test.id
  key_pair_name   = ctyun_keypair.keypair_test.name
  is_destroy_instance = true # 表示退订后直接销毁
}
```

配置弹性负载均衡

```
terraform {
```

```
 required_providers {
  ctyun = {
   source  = "ctyun-it/ctyun"
   version = "1.2.0"
  }
 }
}

# 支持配置资源池、可用区和企业项目，若不配置资源池，则会使用页面选定的资源池
# ak/sk无需配置，会自动获取当前账号的ak/sk
provider "ctyun" {
 region_id = "bb9fdb42056f11eda1610242ac110002"
 az_name = "cn-huadong1-jsnj1A-public-ctcloud"
}

# 创建vpc
resource "ctyun_vpc" "vpc_test" {
 name     = "tf-vpc-ha"
 cidr     = "192.168.0.0/16"
 description = "terraform测试使用"
}

# 创建子网
resource "ctyun_subnet" "subnet_test" {
 vpc_id    = ctyun_vpc.vpc_test.id
 name     = "tf-vpc-ha"
 cidr     = "192.168.1.0/24"
 description = "terraform测试使用"
 dns = [
  "114.114.114.114",
  "8.8.8.8"
 ]
}

# 创建弹性IP
resource "ctyun_eip" "eip_test" {
 name          = "tf-eip-ha"
 bandwidth       = 1
 cycle_type      = "on_demand"
 demand_billing_type = "bandwidth"
}

# 查询可用区
data "ctyun_zones" "test" {

}
```

```
locals {
  az1 = data.ctyun_zones.test.zones[0]
  az2 = data.ctyun_zones.test.zones[1]
}

# az1
data "ctyun_images" "image_test" {
  az_name   = local.az1
  name      = "CentOS Linux 8.4"
  visibility = "public"
  page_no   = 1
  page_size  = 10
}

data "ctyun_ecs_flavors" "ecs_flavor_test" {
  az_name = local.az1
  cpu    = 2
  ram    = 4
  arch   = "x86"
  series  = "C"
  type   = "CPU_C7"
}

resource "ctyun_ecs" "ecs_test" {
  instance_name   = "tf-ecs-ha-1"
  display_name    = "tf-ecs-ha-1"
  flavor_id      = data.ctyun_ecs_flavors.ecs_flavor_test.flavors[0].id
  image_id       = data.ctyun_images.image_test.images[0].id
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id        = ctyun_vpc.vpc_test.id
  password       = var.password
  az_name        = local.az1
  cycle_type      = "on_demand"
  subnet_id      = ctyun_subnet.subnet_test.id
}

# az2
data "ctyun_images" "image_test2" {
  az_name   = local.az2
  name      = "CentOS Linux 8.4"
  visibility = "public"
  page_no   = 1
  page_size  = 10
}

data "ctyun_ecs_flavors" "ecs_flavor_test2" {
```

```
  az_name = local.az2
  cpu    = 2
  ram    = 4
  arch   = "x86"
  series = "C"
  type   = "CPU_C7"
}

resource "ctyun_ecs" "ecs_test2" {
  instance_name   = "tf-ecs-ha-2"
  display_name    = "tf-ecs-ha-2"
  flavor_id       = data.ctyun_ecs_flavors.ecs_flavor_test2.flavors[0].id
  image_id        = data.ctyun_images.image_test2.images[0].id
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id          = ctyun_vpc.vpc_test.id
  password        = var.password
  az_name         = local.az2
  cycle_type      = "on_demand"
  subnet_id       = ctyun_subnet.subnet_test.id
}

# 创建弹性负载均衡
resource "ctyun_elb_loadbalancer" "test" {
  az_name     = local.az1
  subnet_id   = ctyun_subnet.subnet_test.id
  name        = "tf-elb-ha"
  sla_name    = "elb.s2.small"
  resource_type = "external"
  vpc_id      = ctyun_vpc.vpc_test.id
  eip_id      = ctyun_eip.eip_test.id
  cycle_type  = "on_demand"
}

# 创建监听器
resource "ctyun_elb_listener" "elb_listener_test" {
  loadbalancer_id   = ctyun_elb_loadbalancer.test.id
  name              = "tf-elb-listener-ha"
  protocol          = "TCP"
  protocol_port     = 456
  default_action_type = "forward"
  target_groups = [{ target_group_id = ctyun_elb_target_group.target_group_test.id }]
}

# 创建后端主机组
resource "ctyun_elb_target_group" "target_group_test" {
  name    = "tf-target-ha"
```

```
  vpc_id   = ctyun_vpc.vpc_test.id
  algorithm = "wrr"
}

resource "ctyun_elb_target" "target" {
  target_group_id = ctyun_elb_target_group.target_group_test.id
  instance_type = "VM"
  instance_id = ctyun_ecs.ecs_test.id
  protocol_port = 8000
}

resource "ctyun_elb_target" "target2" {
  target_group_id = ctyun_elb_target_group.target_group_test.id
  instance_type = "VM"
  instance_id = ctyun_ecs.ecs_test2.id
  protocol_port = 8000
}

# 需要输入密码
variable "password" {
  type    = string
  sensitive = true
}

output "eid_address" {
  value = ctyun_eip.eip_test.address
}
# 支持配置资源池、可用区和企业项目，若不配置资源池，则会使用页面选定的资源池
# ak/sk无需配置，会自动获取当前账号的ak/sk
provider "ctyun" {
  region_id = "bb9fdb42056f11eda1610242ac110002"
  az_name = "cn-huadong1-jsnj1A-public-ctcloud"
}

# 创建vpc
resource "ctyun_vpc" "vpc_test" {
  name     = "tf-vpc-ha"
  cidr     = "192.168.0.0/16"
  description = "terraform测试使用"
}

# 创建子网
resource "ctyun_subnet" "subnet_test" {
  vpc_id    = ctyun_vpc.vpc_test.id
  name     = "tf-vpc-ha"
  cidr     = "192.168.1.0/24"
  description = "terraform测试使用"
```

```
  dns = [
    "114.114.114.114",
    "8.8.8.8"
  ]
}

# 创建弹性IP
resource "ctyun_eip" "eip_test" {
  name          = "tf-eip-ha"
  bandwidth       = 1
  cycle_type      = "on_demand"
  demand_billing_type = "bandwidth"
}

# 查询可用区
data "ctyun_zones" "test" {

}

locals {
  az1 = data.ctyun_zones.test.zones[0]
  az2 = data.ctyun_zones.test.zones[1]
}

# az1
data "ctyun_images" "image_test" {
  az_name   = local.az1
  name      = "CentOS Linux 8.4"
  visibility = "public"
  page_no   = 1
  page_size  = 10
}

data "ctyun_ecs_flavors" "ecs_flavor_test" {
  az_name = local.az1
  cpu    = 2
  ram    = 4
  arch   = "x86"
  series  = "C"
  type   = "CPU_C7"
}

resource "ctyun_ecs" "ecs_test" {
  instance_name   = "tf-ecs-ha-1"
  display_name    = "tf-ecs-ha-1"
  flavor_id      = data.ctyun_ecs_flavors.ecs_flavor_test.flavors[0].id
  image_id       = data.ctyun_images.image_test.images[0].id
```

```
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id       = ctyun_vpc.vpc_test.id
  password      = var.password
  az_name       = local.az1
  cycle_type     = "on_demand"
  subnet_id     = ctyun_subnet.subnet_test.id
}

# az2
data "ctyun_images" "image_test2" {
  az_name   = local.az2
  name     = "CentOS Linux 8.4"
  visibility = "public"
  page_no   = 1
  page_size  = 10
}

data "ctyun_ecs_flavors" "ecs_flavor_test2" {
  az_name = local.az2
  cpu    = 2
  ram    = 4
  arch   = "x86"
  series  = "C"
  type   = "CPU_C7"
}

resource "ctyun_ecs" "ecs_test2" {
  instance_name   = "tf-ecs-ha-2"
  display_name    = "tf-ecs-ha-2"
  flavor_id     = data.ctyun_ecs_flavors.ecs_flavor_test2.flavors[0].id
  image_id      = data.ctyun_images.image_test2.images[0].id
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id       = ctyun_vpc.vpc_test.id
  password      = var.password
  az_name       = local.az2
  cycle_type     = "on_demand"
  subnet_id     = ctyun_subnet.subnet_test.id
}

# 创建弹性负载均衡
resource "ctyun_elb_loadbalancer" "test" {
  az_name     = local.az1
  subnet_id    = ctyun_subnet.subnet_test.id
  name       = "tf-elb-ha"
  sla_name    = "elb.s2.small"
```

```
  resource_type = "external"
  vpc_id      = ctyun_vpc.vpc_test.id
  eip_id      = ctyun_eip.eip_test.id
  cycle_type   = "on_demand"
}

# 创建监听器
resource "ctyun_elb_listener" "elb_listener_test" {
  loadbalancer_id   = ctyun_elb_loadbalancer.test.id
  name          = "tf-elb-listener-ha"
  protocol        = "TCP"
  protocol_port     = 456
  default_action_type = "forward"
  target_groups = [{ target_group_id = ctyun_elb_target_group.target_group_test.id }]
}

# 创建后端主机组
resource "ctyun_elb_target_group" "target_group_test" {
  name    = "tf-target-ha"
  vpc_id   = ctyun_vpc.vpc_test.id
  algorithm = "wrr"
}

resource "ctyun_elb_target" "target" {
  target_group_id = ctyun_elb_target_group.target_group_test.id
  instance_type = "VM"
  instance_id = ctyun_ecs.ecs_test.id
  protocol_port = 8000
}

resource "ctyun_elb_target" "target2" {
  target_group_id = ctyun_elb_target_group.target_group_test.id
  instance_type = "VM"
  instance_id = ctyun_ecs.ecs_test2.id
  protocol_port = 8000
}


# 需要输入密码
variable "password" {
  type    = string
  sensitive = true
}

output "eid_address" {
  value = ctyun_eip.eip_test.address
}
```

```
# 支持配置资源池、可用区和企业项目，若不配置资源池，则会使用页面选定的资源池
# ak/sk无需配置，会自动获取当前账号的ak/sk
provider "ctyun" {
 region_id = "bb9fdb42056f11eda1610242ac110002"
 az_name = "cn-huadong1-jsnj1A-public-ctcloud"
}

# 创建vpc
resource "ctyun_vpc" "vpc_test" {
 name     = "tf-vpc-ha"
 cidr     = "192.168.0.0/16"
 description = "terraform测试使用"
}

# 创建子网
resource "ctyun_subnet" "subnet_test" {
 vpc_id    = ctyun_vpc.vpc_test.id
 name     = "tf-vpc-ha"
 cidr     = "192.168.1.0/24"
 description = "terraform测试使用"
 dns = [
  "114.114.114.114",
  "8.8.8.8"
 ]
}

# 创建弹性IP
resource "ctyun_eip" "eip_test" {
 name          = "tf-eip-ha"
 bandwidth       = 1
 cycle_type      = "on_demand"
 demand_billing_type = "bandwidth"
}

# 查询可用区
data "ctyun_zones" "test" {

}

locals {
 az1 = data.ctyun_zones.test.zones[0]
 az2 = data.ctyun_zones.test.zones[1]
}

# az1
data "ctyun_images" "image_test" {
 az_name   = local.az1
```

```
  name     = "CentOS Linux 8.4"
  visibility = "public"
  page_no   = 1
  page_size  = 10
}

data "ctyun_ecs_flavors" "ecs_flavor_test" {
  az_name = local.az1
  cpu    = 2
  ram    = 4
  arch   = "x86"
  series  = "C"
  type   = "CPU_C7"
}

resource "ctyun_ecs" "ecs_test" {
  instance_name   = "tf-ecs-ha-1"
  display_name    = "tf-ecs-ha-1"
  flavor_id     = data.ctyun_ecs_flavors.ecs_flavor_test.flavors[0].id
  image_id      = data.ctyun_images.image_test.images[0].id
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id       = ctyun_vpc.vpc_test.id
  password      = var.password
  az_name       = local.az1
  cycle_type     = "on_demand"
  subnet_id     = ctyun_subnet.subnet_test.id
}

# az2
data "ctyun_images" "image_test2" {
  az_name   = local.az2
  name     = "CentOS Linux 8.4"
  visibility = "public"
  page_no   = 1
  page_size  = 10
}

data "ctyun_ecs_flavors" "ecs_flavor_test2" {
  az_name = local.az2
  cpu    = 2
  ram    = 4
  arch   = "x86"
  series  = "C"
  type   = "CPU_C7"
}
```

```
resource "ctyun_ecs" "ecs_test2" {
  instance_name   = "tf-ecs-ha-2"
  display_name    = "tf-ecs-ha-2"
  flavor_id       = data.ctyun_ecs_flavors.ecs_flavor_test2.flavors[0].id
  image_id        = data.ctyun_images.image_test2.images[0].id
  system_disk_type = "sata"
  system_disk_size = 40
  vpc_id          = ctyun_vpc.vpc_test.id
  password        = var.password
  az_name         = local.az2
  cycle_type      = "on_demand"
  subnet_id       = ctyun_subnet.subnet_test.id
}

# 创建弹性负载均衡
resource "ctyun_elb_loadbalancer" "test" {
  az_name      = local.az1
  subnet_id    = ctyun_subnet.subnet_test.id
  name         = "tf-elb-ha"
  sla_name     = "elb.s2.small"
  resource_type = "external"
  vpc_id       = ctyun_vpc.vpc_test.id
  eip_id       = ctyun_eip.eip_test.id
  cycle_type   = "on_demand"
}

# 创建监听器
resource "ctyun_elb_listener" "elb_listener_test" {
  loadbalancer_id   = ctyun_elb_loadbalancer.test.id
  name              = "tf-elb-listener-ha"
  protocol          = "TCP"
  protocol_port     = 456
  default_action_type = "forward"
  target_groups = [{ target_group_id = ctyun_elb_target_group.target_group_test.id }]
}

# 创建后端主机组
resource "ctyun_elb_target_group" "target_group_test" {
  name     = "tf-target-ha"
  vpc_id   = ctyun_vpc.vpc_test.id
  algorithm = "wrr"
}

resource "ctyun_elb_target" "target" {
  target_group_id = ctyun_elb_target_group.target_group_test.id
  instance_type = "VM"
  instance_id = ctyun_ecs.ecs_test.id
```

```
 protocol_port = 8000
}

resource "ctyun_elb_target" "target2" {
 target_group_id = ctyun_elb_target_group.target_group_test.id
 instance_type = "VM"
 instance_id = ctyun_ecs.ecs_test2.id
 protocol_port = 8000
}


# 需要输入密码
variable "password" {
 type    = string
 sensitive = true
}

output "eid_address" {
 value = ctyun_eip.eip_test.address
}
```

创建MySQL数据库

```
terraform {
 required_providers {
  ctyun = {
   source  = "ctyun-it/ctyun"
   version = "1.2.0"
  }
 }
}

# 支持配置资源池、可用区和企业项目，若不配置资源池，则会使用页面选定的资源池
# ak/sk无需配置，会自动获取当前账号的ak/sk
provider "ctyun" {
 region_id = "bb9fdb42056f11eda1610242ac110002"
 az_name = "cn-huadong1-jsnj1A-public-ctcloud"
}


# 创建vpc
resource "ctyun_vpc" "vpc_test" {
 name     = "vpc-for-mysql"
 cidr     = "192.168.0.0/16"
 description = "terraform测试使用"
}
```

```
# 在vpc下创建子网
resource "ctyun_subnet" "subnet_test" {
 vpc_id    = ctyun_vpc.vpc_test.id
 name      = "subnet-for-mysql"
 cidr      = "192.168.1.0/24"
 description = "terraform测试使用"
 dns = [
  "114.114.114.114",
  "8.8.8.8"
 ]
}


# 创建安全组
resource "ctyun_security_group" "security_group_test" {
 vpc_id    = ctyun_vpc.vpc_test.id
 name      = "security-group-for-mysql"
 description = "terraform测试使用"
}

# 查询规格
data "ctyun_ecs_flavors" "ecs_flavor_test" {
 cpu   = 4
 ram   = 8
 arch  = "x86"
 series = "C"
 type  = "CPU_C7"
}

# 创建数据库
resource "ctyun_mysql_instance" "mysql_test" {
 vpc_id          = ctyun_vpc.vpc_test.id
 subnet_id        = ctyun_subnet.subnet_test.id
 security_group_id   =ctyun_security_group.security_group_test.id
 name           = "mysql-test-10"
 storage_type       = "SSD"
 storage_space      = 200
 cycle_type        = "on_demand"
 prod_id          = "Single80"
 flavor_name       = data.ctyun_ecs_flavors.ecs_flavor_test.flavors[0].name
}
```

常见问题

基础概念类

资源编排ROS和 Terraform 有什么关系？

资源编排ROS基于 Terraform 引擎进行封装，用户可以通过控制台或 API 使用 Terraform 语法（HCL）来描述云上资源。服务负责资源创建、更新、销毁的生命周期管理，无需用户本地安装 Terraform。

资源编排如何收费

资源编排ROS本身不收取服务费。但是，通过本服务创建和管理的一切天翼云资源（如ECS、EIP、CCSE、Mysql等），都会按照各自产品的标准计费规则进行收费。

服务支持的 Terraform 版本是多少？

当前支持的Terraform版本号是1.5.7

使用资源编排ROS需要哪些权限？

用户需具备对应云资源的创建、查询、删除权限，以及对资源编排ROS本身的访问权限（例如：模板管理、资源栈管理）。

Terraform Provider 的 AK/SK 是如何管理的？

对于AK/SK：控制台会自动使用您当前登录的账号权限，生成委托ak/sk，以您的权限进行资源的开通，无需在模板中指定。

我的Terraform状态文件存储在哪里？

状态文件由资源编排ROS后端统一托管。您无需关心其存储位置和备份问题。这种机制保证了状态文件的安全性和多用户协作时的一致性。

如何将天翼云上已经存在的资源纳入到资源栈中进行管理？

当前版本暂时不支持，我们正在抓紧迭代中。

使用问题类

模板验证失败的是什么原因？

出于安全保证和的模版合规性的原因，我们内置了一些模版校验规则，具体规则请参考核心约束与限制

诊断资源栈部署失败的原因？

控制台会展示详细的报错原因和执行日志：

1. 部署失败状态会直接提示失败原因

2. 资源栈事件可以查看完整的执行日志

删除资源栈失败怎么办？

可能原因包括

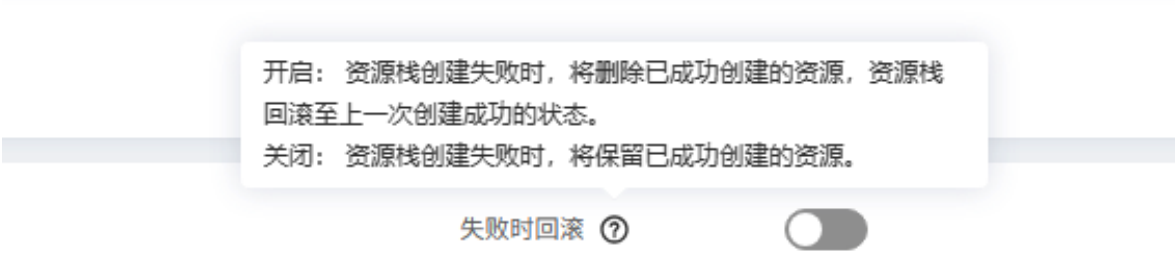
- 资源间存在依赖（如子网未解绑 EIP）；
- 用户或外部操作修改了资源属性；
- 部分资源已在控制台手动删除；
- 可查看失败原因和执行日志，手动处理后重试。

如何更新我现有的资源？

请参考以下流程：更新模版 -> 资源栈选择更新-> 填写参数-> 创建执行计划 -> 部署执行计划

部署失败能否回滚到上一次成功的栈状态？

可以开启资源栈的失败回滚功能，当部署失败的时候，会自动回滚到上次部署成功的状态。



删除资源栈时，会删除所有资源吗？

您可选择是否删除所有资源。如选择删除所有资源，执行删除操作时，系统会按照资源依赖关系的反向顺序，自动销毁资源栈内由模板创建的所有资源。此操作不可逆，请务必谨慎操作！ 在执行删除前，系统会要求您再次确认。

我可以通过天翼云控制台手动修改由资源栈创建的云服务器（ECS）吗？

强烈不建议这样做！ 手动修改会导致“配置漂移”，即资源的实际状态与资源栈中记录的状态不一致。当您下次通过资源栈更新或修改该资源时，系统可能会根据模板定义覆盖您的手动更改，导致非预期的结果。所有资源变更都应通过更新模板并重新更新资源栈来完成。

创建一个高可用架构

操作场景

本示例以创建一个弹性负载均衡为例，介绍如何使用模板创建一个高可用架构。

不同架构内设云资源不同，本文及供参考。

操作步骤

1. 登录控制中心。
2. 单击左侧导航栏“产品服务列表”，选择“迁移与管理 > 资源编排”，进入资源编排控制台。
3. 在左侧导航栏选择 **模板管理**。
4. 在模板管理页面，单击**创建模板**。
5. 创建高可用架构的.tf模板示例如下，请参考文档创建模板完成模板配置。

```
terraform {
 required_providers {
  ctyun = {
   source = "ctyun-it/ctyun"
   version = "1.2.0"
  }
 }
}

provider "ctyun" {
 region_id  = "bb9fdb42056f11eda1610242ac110002" //选定资源池
 az_name    = "cn-huadong1-jsnj1A-public-ctcloud" //选定可用区
}

variable "instance_name" { //定义变量：云主机名称
 type    = string
 default = "tf-ecs-ha"
 description = "Name of EC instances to create"
}

variable "vpc_name" {  //定义变量：虚拟私有云名称
 type    = string
 default = "tf-vpc-ha"
 description = "Name of vpc to create"
}

variable "eip_name" { //定义变量：弹性IP名称
 type    = string
 default = "tf-eip-ha"
 description = "Name of eip to create"
```

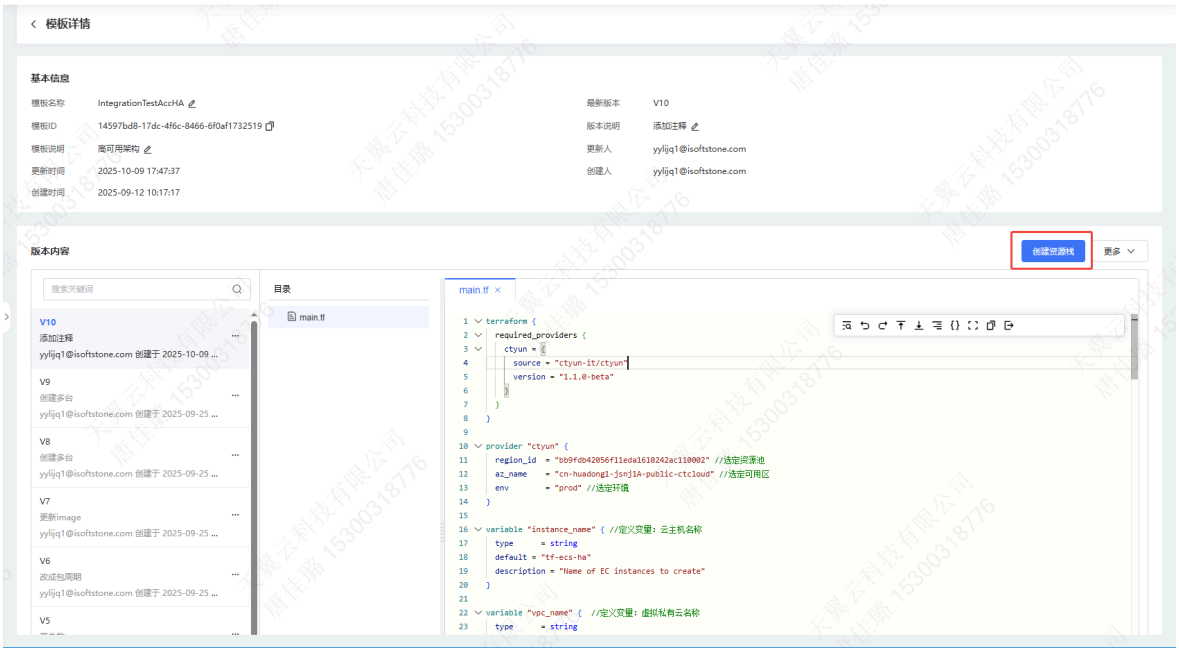
```
}

variable "elb_name" { //定义变量：弹性负载均衡名称
 type    = string
 default = "tf-elb-ha"
 description = "Name of elb to create"
}

variable "instance_count" { //定义变量：创建弹性云主机数量，默认为1个
 type   = number
 default = 1
 description = "Number of EC instances to create"
}

variable "password" { //定义变量：云主机密码
 type    = string
 sensitive = true
}

resource "ctyun_vpc" "vpc_test" { //定义vpc资源：vpc_test
 name     = var.vpc_name
 cidr     = "192.168.0.0/16"
 description = "terraform测试使用"
 enable_ipv6 = true
}

resource "ctyun_subnet" "subnet_test" { //定义子网资源：subnet_test
 vpc_id    = ctyun_vpc.vpc_test.id
 name     = "tf-vpc-ha"
 cidr     = "192.168.1.0/24"
 description = "terraform测试使用"
 dns = [
  "114.114.114.114",
  "8.8.8.8",
  "8.8.4.4"
 ]
 enable_ipv6 = true
}

resource "ctyun_eip" "eip_test" { //定义弹性IP资源：eip_test
 name          = var.eip_name
 bandwidth       = 1
 cycle_type      = "on_demand"
 demand_billing_type = "bandwidth"
}

data "ctyun_zones" "test" {
```

```
}

locals {
 az1 = data.ctyun_zones.test.zones[0]
 az2 = data.ctyun_zones.test.zones[1]
}

output "az1" {
 value = local.az1
}

data "ctyun_images" "image_test" { //确定云主机镜像
 az_name   = local.az1
 name      = "CTyunOS 22.06 64 位"
 visibility = "public"
 page_no   = 1
 page_size  = 10
}
data "ctyun_ecs_flavors" "ecs_flavor_test" { //确定云主机参数
 az_name = local.az1
 cpu    = 2
 ram    = 4
 arch   = "x86"
 series  = "C"
 type   = "CPU_C7"
}

resource "ctyun_ecs" "ecs_test" { //定义云主机资源：ecs_test
 count = var.instance_count

 instance_name     = "${var.instance_name}-${count.index + 1}"
 display_name      = "${var.instance_name}-${count.index + 1}"

 flavor_id      = data.ctyun_ecs_flavors.ecs_flavor_test.flavors[0].id
 image_id       = data.ctyun_images.image_test.images[0].id
 is_destroy_instance  = true
 system_disk_type = "sata"
 system_disk_size = 40
 vpc_id        = ctyun_vpc.vpc_test.id
 password       = var.password
 az_name       = local.az1
 cycle_type     = "year"
 cycle_count     = 1
 subnet_id      = ctyun_subnet.subnet_test.id
}
```

```
# az2
data "ctyun_images" "image_test2" {
 az_name   = local.az2
 name      = "CentOS Linux 8.4"
 visibility = "public"
 page_no   = 1
 page_size  = 10
}

data "ctyun_ecs_flavors" "ecs_flavor_test2" {
 az_name = local.az2
 cpu    = 2
 ram    = 4
 arch   = "x86"
 series  = "C"
 type   = "CPU_C7"
}

resource "ctyun_ecs" "ecs_test2" { //定义云主机资源：ecs_test2
 instance_name   = "${var.instance_name}-2"
 display_name    = "${var.instance_name}-2"
 flavor_id      = data.ctyun_ecs_flavors.ecs_flavor_test2.flavors[0].id
 image_id       = data.ctyun_images.image_test2.images[0].id
 is_destroy_instance  = true
 system_disk_type = "sata"
 system_disk_size = 40
 vpc_id        = ctyun_vpc.vpc_test.id
 password       = var.password
 az_name        = local.az2
 cycle_type     = "year"
 cycle_count    = 1
 subnet_id      = ctyun_subnet.subnet_test.id
}

resource "ctyun_elb_loadbalancer" "test" { //定义弹性负载均衡资源：test
 az_name     = local.az1
 subnet_id    = ctyun_subnet.subnet_test.id
 name        = var.elb_name
 sla_name     = "elb.s2.small"
 resource_type = "external"
 vpc_id      = ctyun_vpc.vpc_test.id
 eip_id      = ctyun_eip.eip_test.id
 cycle_type   = "on_demand"
}

resource "ctyun_elb_listener" "elb_listener_test" { //定义弹性负载均衡监听器资源：
elb_listener_test
```

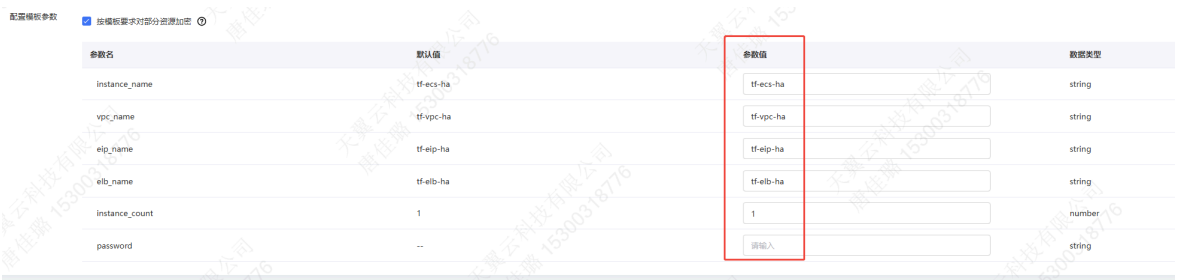
```
  loadbalancer_id    = ctyun_elb_loadbalancer.test.id
  name               = "${var.elb_name}-listerer"
  protocol           = "TCP"
  protocol_port      = 456
  default_action_type = "forward"
  target_groups = [{ target_group_id = ctyun_elb_target_group.target_group_test.id }]
}

resource "ctyun_elb_target_group" "target_group_test" { //定义负载均衡后端主机组
  name     = "${var.elb_name}-targer-group"
  vpc_id   = ctyun_vpc.vpc_test.id
  algorithm = "wrr"
}
```

6. 在刚刚创建好的模板详情中，单击**创建资源栈**。



7. 参考创建资源栈，使用当前模板完成资源栈创建。

需要注意的是，您需要在模板参数配置中，对模板中的变量进行赋值，变量值的校验规则您可以在模板中提前配置。



8. 参考部署资源栈，完成资源栈部署。
9. 部署完成后，您可前往对应资源控制台查看资源的创建结果。

API使用说明

OpenAPI门户提供了产品的描述、语法、参数说明及示例等内容。

关于用户如何使用天翼云资源编排ROS产品API的详细介绍，请参见使用API。您可以在OpenAPI门户可以了解到具体的API认证鉴权机制、接入终端节点、API概览、API详述、状态码接入点等内容，配合在线测试、sdk，API调试将更加方便。

相关协议

资源编排ROS产品服务协议