第二部分是 云应用平台 建设

2.1 云应用

什么是 云应用 ？为什么要做 ？

2.1 云应用 - 背景

应用上云 是企业降本增效的大趋势

而云上应用软件的部署和管理却充满挑战

目前企业应用云化在世界范围内成为趋势 很多企业用户都在进行上云的转型

在云原生的趋势下，如何在云上部署/管理/运维业务软件与环境 是服务商和企业用户共同面临的挑战

2.1 云应用 - 背景

私有化部署的应用交付有那些痛点？

- **交付部署流程**的繁琐复杂
- **非标准交付**带来的高额成本
- **数据安全**的严格要求
- **服务商代运维操作**难以管控审计
- **多应用资源管理**的困难

私有化应用交付服务流程

现有服务商给客户 交付 [私有化部署应用] 时存在着很多痛点，首先基本都需要通过线下 1v1 进行

那么，每次将应用所需要的众多云产品进行散点式的创建/配置/管理本身就是个相当复杂的流程

交付非标准化也产生了高成本的人力消耗

同时，数据安全、服务商的代运维操作这些都无法得到统一的管控

此外，没办法按照[应用]来管理众多云资源也是后期运维时的一大痛点

这些痛点存在于服务商给客户交付应用软件时的各个环节，我们也接到业务的此类需求

如果能从工具层面围绕整体流程建立更便捷 更稳定 更安全的服务与信任关系，便能解决这些痛点

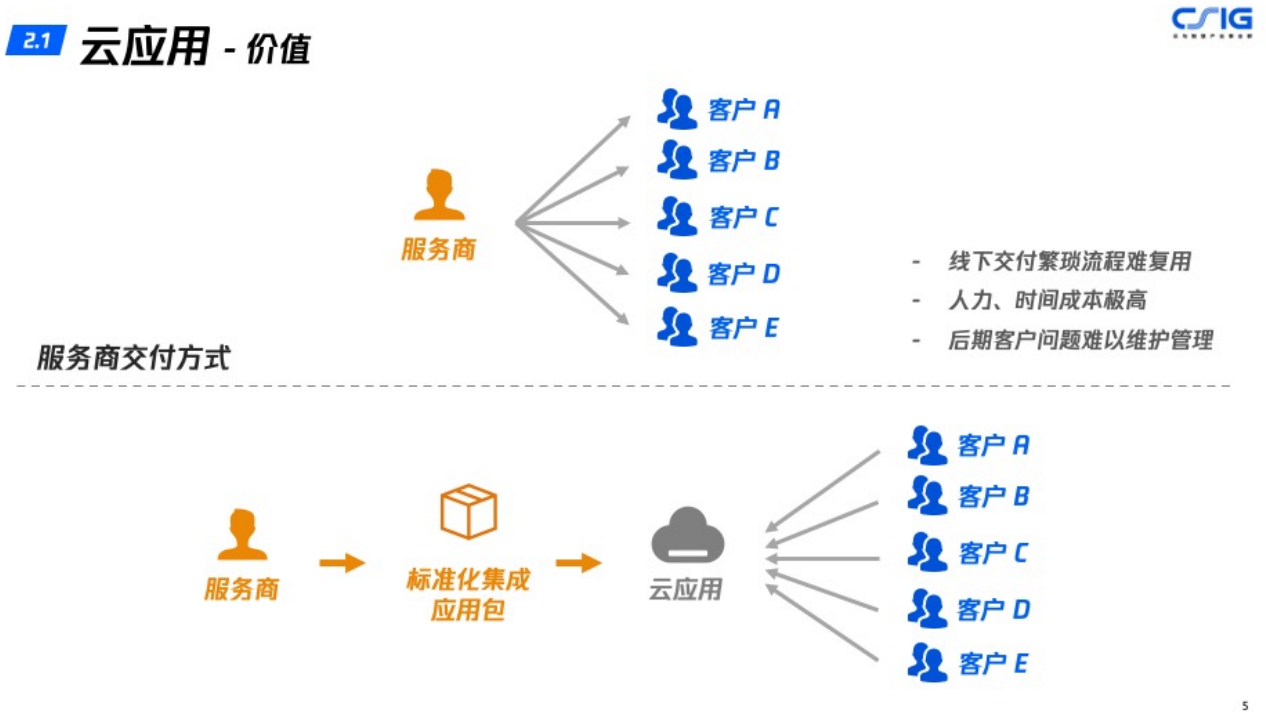
2.1 云应用 - 介绍

提供「软件」和「云资源」采买、部署、管理的一站式解决方案

云应用便是针对这些痛点推出的，*定位为「软件」和「云资源」采买、部署、管理的一站式解决方案*

服务商将腾讯云的资源与自己的应用软件 标准化集成 为应用包，上架至云应用平台

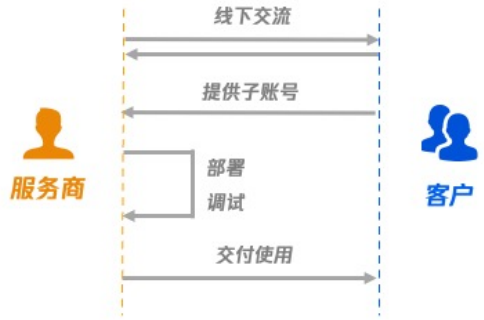
用户可在平台一键安装，并可以在平台便捷的管理软件及资源



通过云应用，服务商不需要再对每个客户都线下进行 1v1 的沟通/部署/测试/交付，大幅节省人力时间成本

同时通过应用的代运维管理，可以降低后期客户应用运维难度

2.1 云应用 - 价值



- 线下的繁琐流程
- 等待时间较长，可能周级
- 提供企业账号的安全风险

客户采购方式

对客户来说，采购流程也不需要进行复杂的线下沟通与等待，同时也无需把自身账号提供给服务商操作，降低安全风险

整个交付流程在分钟级就可完成

2.1 云应用 - 体验

部署复杂的
云上企业软件
就像 ..

Apple Developer
Apple
获取 App内购买

点击安装

执行安装
包括云资源及软件

可用

一键卸载
包括云资源及软件

在使用体验上，*部署一款复杂的云上企业软件，就像我们常用的应用商店*

点击安装 等待 然后就可以使用，无需关注安装流程

同样卸载软件，也是一键完成

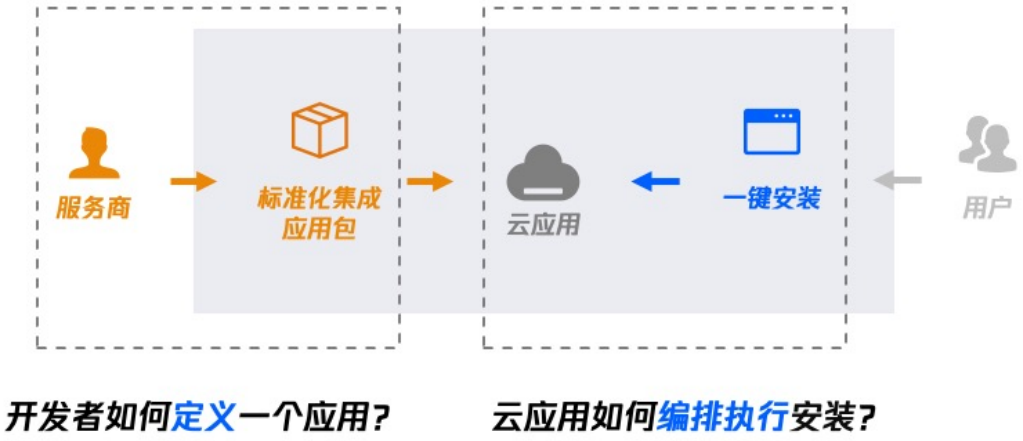
应用 部署安装

8

整个平台建设中最重要的问题就是要解决 应用 该如何部署安装

2:40

2.2 云应用 - 部署安装



我们把这个流程拆分成两方面看

首先我们要定义一套标准，来指导开发者需要定义哪些配置来实现应用的自动部署

然后云应用按照这些配置 进行编排执行 完成安装

2.2 云应用 - 分层

云应用当前支持的是 [使用容器镜像] 运行的应用

把这类应用软件的部署流程分层来看

首先我们需要创建应用依赖的 IaaS / PaaS 资源

然后配置容器集群，之后就能跑起应用服务

所以，我们需要解决 IaaS / PaaS 资源和 K8s 的配置如何定义

以及如何[按这些配置]编排创建

我们先来看定义的方案选择

在 IaaS / PaaS 层，

我们选择了业界共同认可的 Terraform

它提供了一套 模板语言来 定义 云资源、资源间的依赖关系等等。

--

K8s 配置上，也选用了业界通用的包管理器 Helm，腾讯云 TKE 也原生支持了使用 Helm Chart 初始化 容器集群

它同样提供了一套模板语言来定义 K8s 配置

同时，这两种模板语言都支持使用变量，可以在执行时传入变量动态设置配置

2.2 云应用

现在我们各层的配置定义方式有了，接下来就是如何执行安装

2.2 云应用

如何创建 IaaS / PaaS 资源?

- 自行实现 编排引擎，解析模板完成资源创建
  - Terraform 官方编排工具无法实现资源包组合售卖/折扣等能力
- 对每种 resource 使用 云 API 实现对应创建、销毁、查询等方法

```
resource "tencentcloud_vpc" "my_vpc" {
  name              = "app_vpc"
  availability_zone = var.availability_zone
  cidr_block        = var.vpc_cidr
}

resource "tencentcloud_subnet" "my_subnet" {
  name              = "app_subnet"
  vpc_id            = tencentcloud_vpc.my_vpc.id
  availability_zone = var.availability_zone
  cidr_block        = var.vpc_cidr
}

resource "tencentcloud_eks_cluster" "my_eks" {
  availability_zone = var.availability_zone
  cluster_name      = "app_eks"
  k8s_version       = "1.20.6"
  vpc_id            = tencentcloud_vpc.my_vpc.id
  subnet_ids        = [tencentcloud_subnet.my_subnet.id]
  service_subnet_id = tencentcloud_subnet.my_subnet.id
}
```

Terraform 定义示例

13

在云资源层，我们自行实现了编排引擎来解析模板进行资源创建

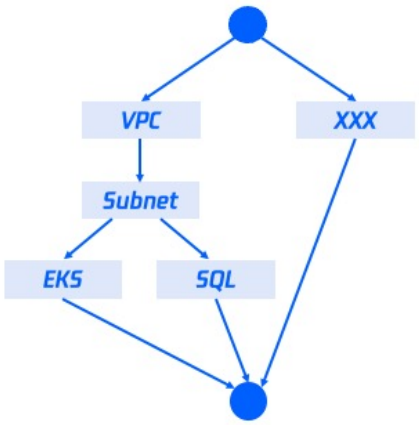
右图是使用 Terraform 模板定义的云资源

对模板中定义的每种 resource，编排引擎使用云 API 实现对应的创建、销毁、查询等一组方法

这里我们没有使用 TF 官方的工具

主要原因是我们想要支持***资源包组合售卖/折扣等能力，比如一个应用依赖的全部云资源组合优惠打包下单，这个官方工具做不到***

2.2 云应用



按 resource 层级顺序调用 云 API 执行创建配置

```
resource "tencentcloud_vpc" "my_vpc" {
  name              = "app_vpc"
  availability_zone = var.availability_zone
  cidr_block        = var.vpc_cidr
}

resource "tencentcloud_subnet" "my_subnet" {
  name              = "app_subnet"
  vpc_id            = tencentcloud_vpc.my_vpc.id
  availability_zone = var.availability_zone
  cidr_block        = var.vpc_cidr
}

resource "tencentcloud_eks_cluster" "my_eks" {
  availability_zone = var.availability_zone
  cluster_name      = "app_eks"
  k8s_version       = "1.20.6"
  vpc_id            = tencentcloud_vpc.my_vpc.id
  subnet_ids        = [tencentcloud_subnet.my_subnet.id]
  service_subnet_id = tencentcloud_subnet.my_subnet.id
}
```

依赖输入变量

依赖其他资源属性值

产生层级关系

14

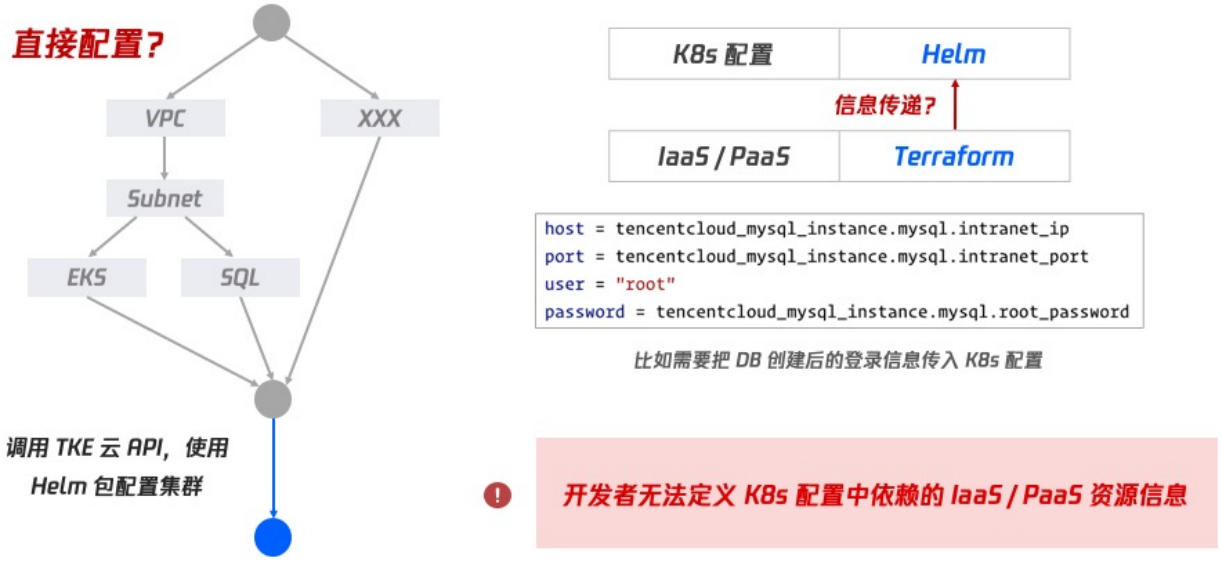
我们可以看到配置中，一个资源的创建参数是可以依赖其他资源的属性值的，比如（子网 x VPC） 这就形成了层级关系

一个合法的架构下，所有云资源的依赖关系应该形成一张有向无环图，我们只需要按照层级顺序，对每种资源调用云 API 执行创建，便可以完成整个架构的搭建

2.2 云应用

在云资源创建完成后，要怎么来用 Helm Chart 包初始化容器集群的配置

2.2 云应用



16

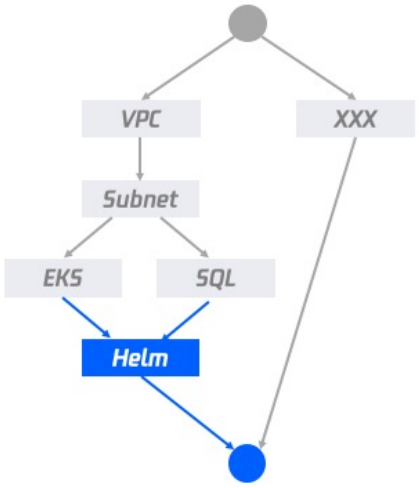
最简单的想法就是直接调用 云API 开搞，但是这样会有一个问题，

开发者没办法定义哪些云资源的信息需要传入 K8s 配置中，

比如创建了 mysql 后，我们在 Terraform 模板中可以拿到登录信息，同时需要把这些信息通过 Helm Chart 传入 K8s 配置中

那应该怎么办

2.2 云应用



- 把 Helm 也作为一种虚拟资源定义为 resource
- 将依赖的其他资源信息值传入配置参数中
- 在相应层级自动调用 云 API 执行配置

```
resource "cloudapp_helm_app" "app" {
  cluster_id = tencentcloud_eks_cluster.my_eks_cluster.id
  chart_url  = "https://xxx.com/charts/app.tgz"
  chart_username    = var.cloudapp_repo_username
  chart_password    = var.cloudapp_repo_password
  chart_values = {
    cloudappTargetSubnetID = var.app_target.subnet_id
    cloudappImageCredentials = {
      registry = var.cloudapp_repo_server
      username = var.cloudapp_repo_username
      password = var.cloudapp_repo_password
    }
    mysql = {
      host = tencentcloud_mysql_instance.mysql.intranet_ip
      port = tencentcloud_mysql_instance.mysql.intranet_port
      user = "root"
      password = tencentcloud_mysql_instance.mysql.root_password
    }
  }
}
```

17

因为 TKE 安装 Helm Chart 包同样是调用云 API 来执行，所以可以把 Helm 设计为一个虚拟的云资源，同样使用 resource 定义在 Terraform 模板中

这样就可以把其他资源的依赖信息作为创建参数填进来，

同样编排引擎会分析出它的层级，在对应资源都创建完成后，传入依赖值开始执行 Helm Chart 包安装

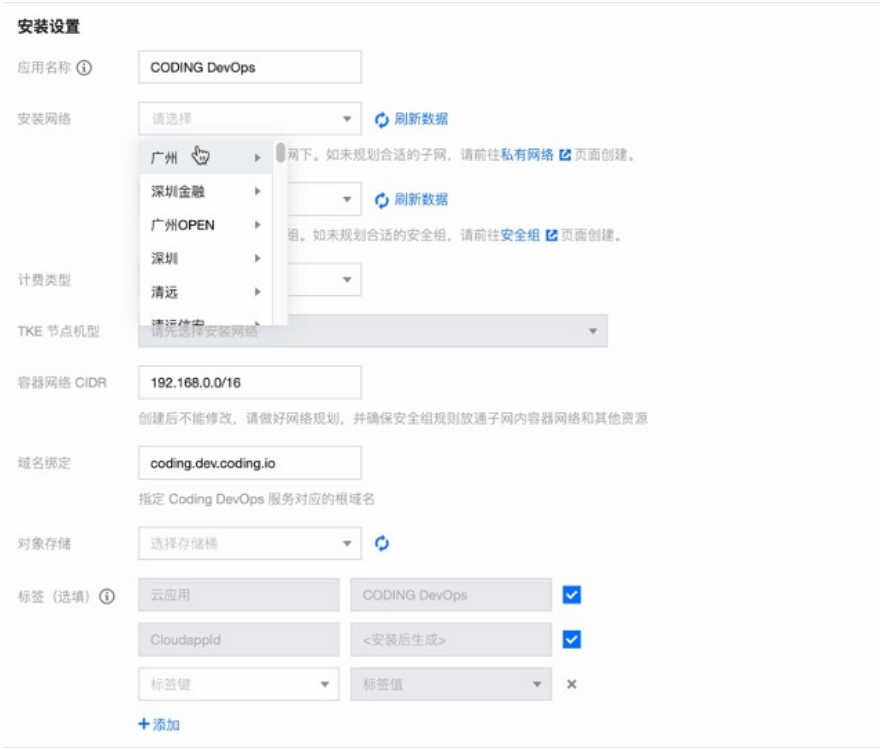
这样，我们整个部署安装流程都可以使用编排引擎驱动起来

同时，资源创建过程中，每一步的信息也都可以一层层向上传递

2.2 云应用

用户自定义安装配置？

- ***需要在企业管理员已经规划好的 VPC / 子网下安装应用***
- ***需要选择适合的资源配置或容量***
- ***需要选择已有的云资源***
- ***需要写入一些信息到应用中***
- ***...***



解决了部署安装后，在试用体验中，我们发现在应用安装前，用户还可能需要自定义一些安装配置

比如
企业的管理员已经规划好了 ***VPC / 子网，我们需要在指定的 VPC 下安装应用，而不能新建一个 VPC***

- ***需要选择适合的资源配置或容量***
- 等等

所以我们还需要支持让开发者来定义一些可变的配置项

并且在开始安装前让用户选择/填写这些信息

为此，我们设计了一套纯配置化表单引擎，只需要简单声明配置来定义表单，

即可实现表单变量和 Terraform 变量的双向绑定

同时也可以支持校验器、表单项联动等复杂的表单能力

2.2 云应用

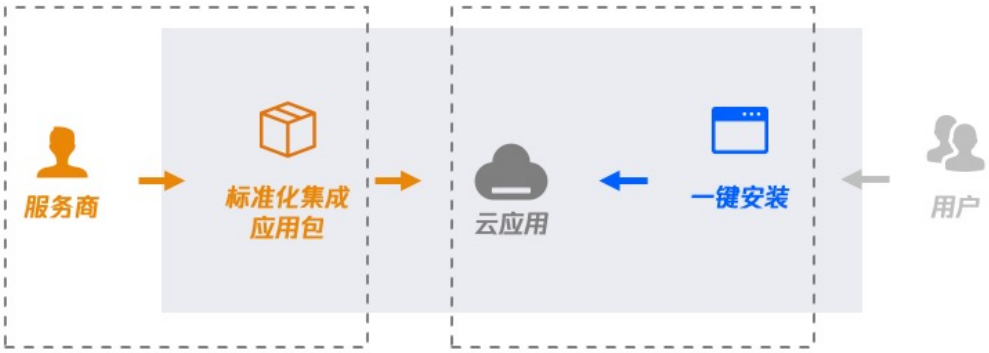
在前端页面渲染表单时，为了支持用户选择已有的 VPC/子网等等资源，

除了标准的表单组件外，云应用还为 [每类资源] 定制了 [用户已有实例] 的选择器

这个表单组件库也在随着 [云应用支持] 的 [云资源种类数] 不断扩充

在搭配上表单配置后，应用安装流程所需的数据，就可以从用户输入开始，流向安装的每个步骤中

2.2 云应用



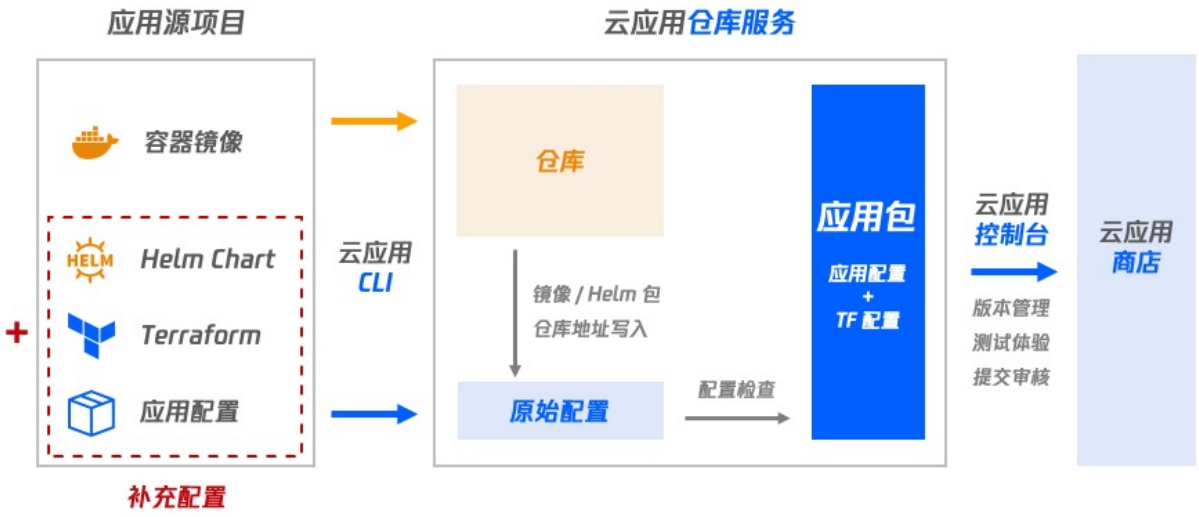
23

到此我们解决了应用如何部署安装

6：45

2.2 云应用

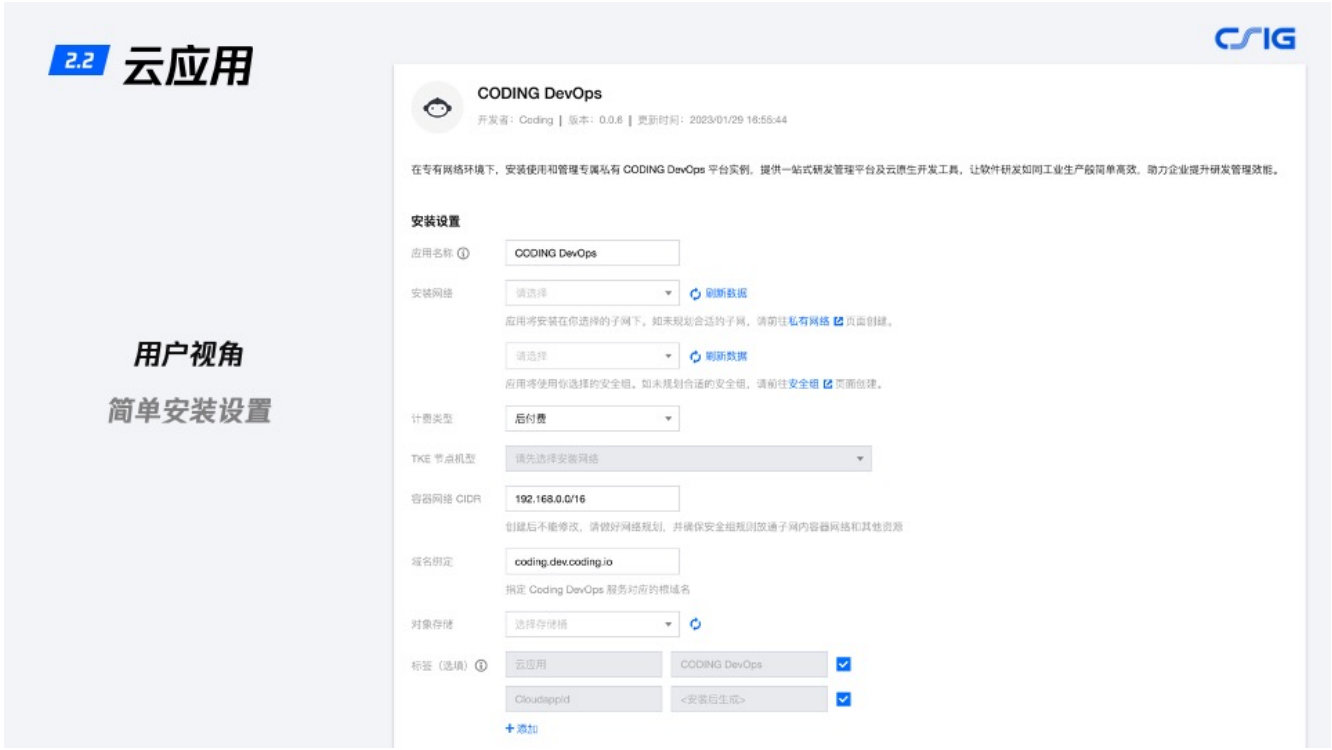
开发者视角



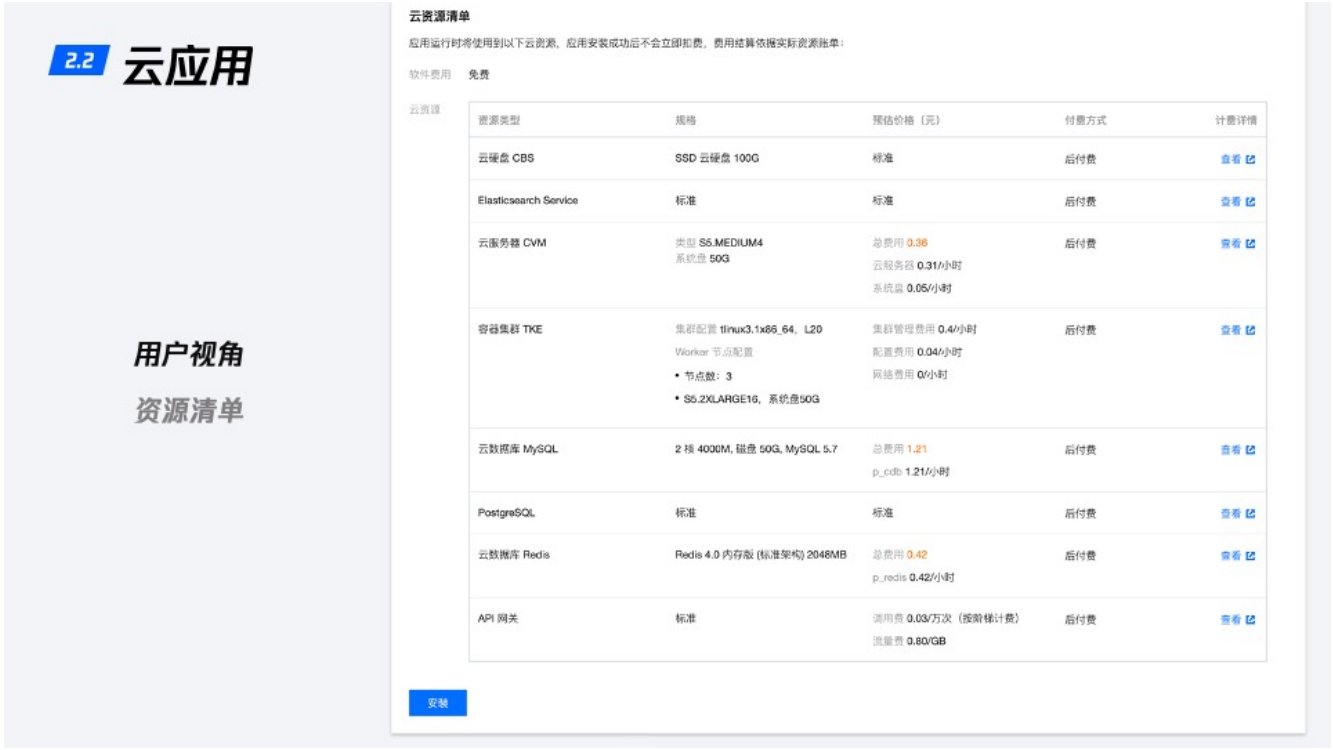
最终，对于开发者来说，只需要在原有项目中加入 Terraform / Helm Chart / 应用配置三类配置文件

同时使用云应用提供的命令行工具进行提交，此时应用产物会托管至云应用仓库，配置文件会打包成为一个应用包

之后开发者可在 云应用控制台 操作 进行测试体验 / 提交审核 /并上架商店



而在用户视角，安装一个应用只需要填写必要配置



然后云应用服务会根据这些配置，展示资源清单及价格，最后点击安装，等待即可使用

应用 运维

云应用第三个部分 应用运维

8:10

2.3 云应用

应用运维方案

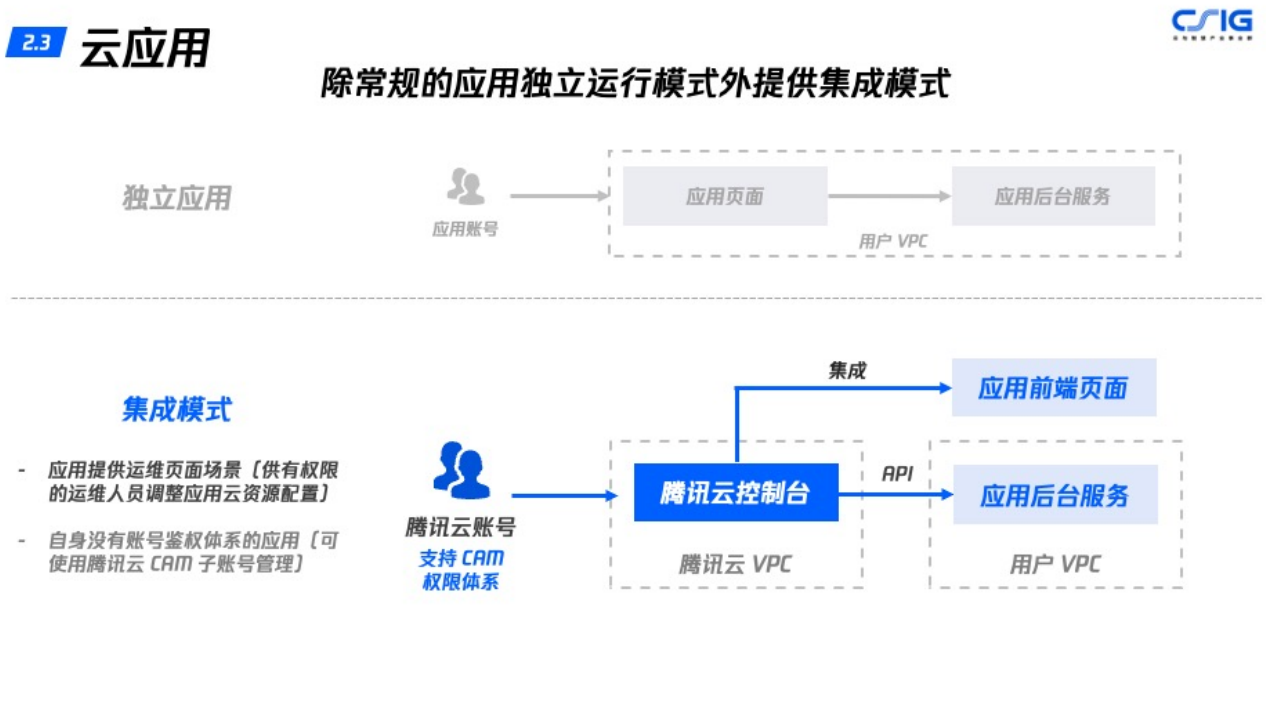
我们设计了一些方案来解决应用交付后 运维中 的一些困难

首先，我们建议应用将自身的一些运维操作封装为 API，比如整提缩容、变配操作等，

然后提供一个应用的运维页面，这样用户可以在不直接操作底层云资源的情况下，低风险的自行来进行一些运维操作

同时，这些 API 可以由用户授权给服务商调用，便可以安全的让服务商执行远程代运维，来解决一些问题，

关于完整的服务商代运维实现这里我们还在规划中。



在这套方案下，云应用除了提供常规的 [应用直接独立运行] 的模式外，还支持了集成模式

应用可以把前后端分离，把应用的前端页面托管至腾讯云控制台，这样便可以通过[腾讯云账号权限体系] 来管控 [当前应用] 的 [使用权限]

这个模式非常适合于应用提供运维页面给客户使用

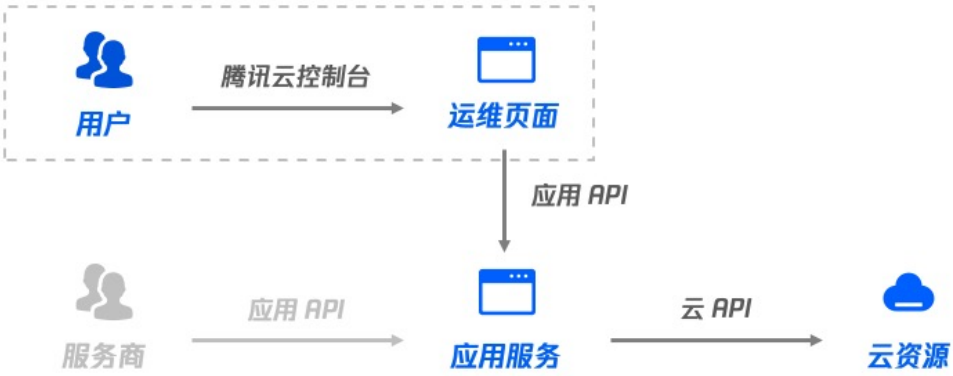
因为应用的所有云资源管理是在腾讯云账号体系下，所以同样应该使用这套权限体系，来管理的应用的运维操作权限

此外，如果一个应用没有自身实现账号鉴权体系，比如一些工具类的应用，也可以方便的使用集成模式接入云应用

9:30

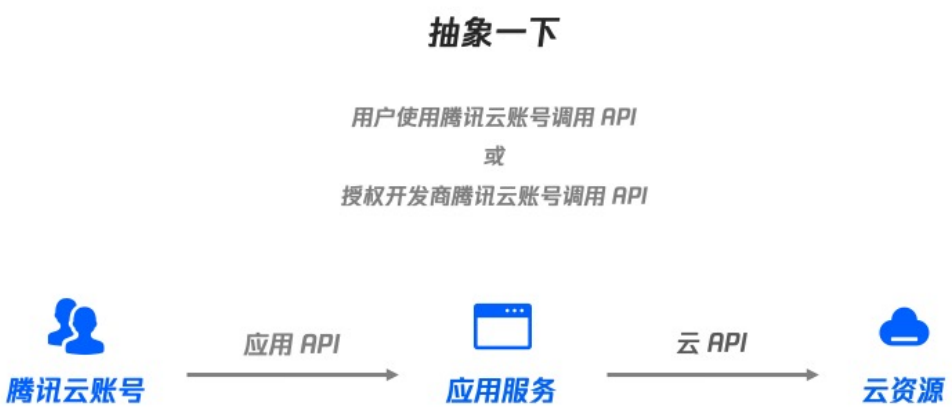
2.3 云应用

运维场景的调用链路



在集成模式下，用户进行运维操作应该是通过腾讯控制台，然后调用应用 API，此时应用的服务要去请求一些列的云 API 来完成底层云资源的变配

服务商进行代运维操作的链路也是类似的



我们来抽象一下，整条链路其实就是通过 腾讯云账号 来进行请求

那这条链路有什么问题？

10:15

2.3 云应用

应用怎么调用云 API？

应用如何获取用户密钥调用云 API？

我们先看链路后半段，应用要怎么调用云 API

其实也就是运行在用户 VPC 中的应用，怎么获取到用户的密钥来调用云 API

我们肯定不能让用户把密钥直接交给第三方的应用

2.3 云应用

应用内部调用云 API 方案

方案	接口代理（重网关）	密钥分发（轻网关）	服务角色（无网关）
描述	应用后端统一从公网通过云应用的网关代理调用云 API	应用后端通过云应用公网接口申请云 API 临时密钥调用云 API	应用启动时，预埋应用的服务角色信息到进程环境变量中，应用扮演角色获取临时密钥调用云 API
运行成本	高	一般	低
安全风险	中	高	低
调用复杂度	简单	复杂	复杂
权限专业性	非专业	非专业	专业（全交由 CAM）
稳定性压力	高（数据流量大）	一般	低（压力在云 API）
实现难度	低	低	高

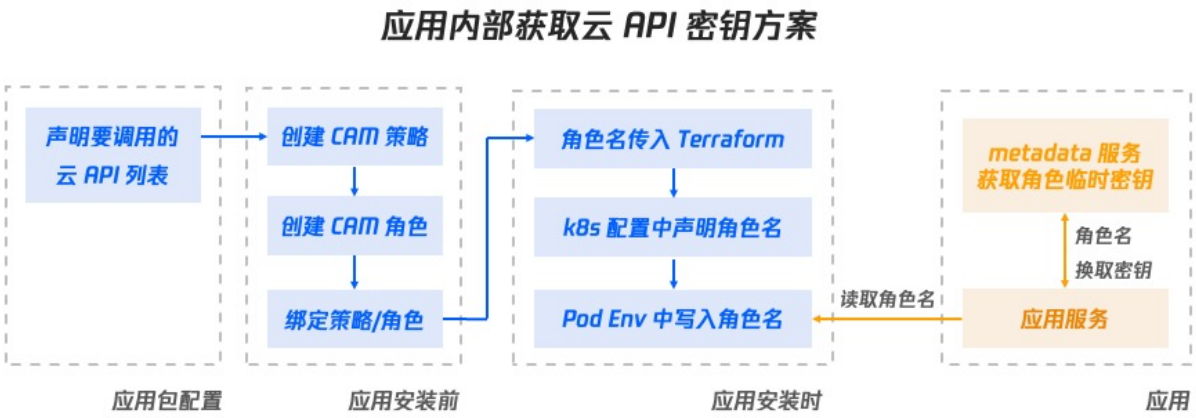
详见附录

为了解决这个问题，我们出了几种方案并进行了对比

因为涉及到密钥获取，我们主要考虑了安全风险及权限校验的专业性，选择了实现最复杂的方案，但也是安全风险最低稳定性最高的

它依赖了 TKE 的支持/CVM 元数据服务/以及 CAM 的角色能力

2.3 云应用



整体调用流程如图，时间关系先不展开

2.3 云应用

应用 API 怎么调用?

那前面的链路又要如何调用呢

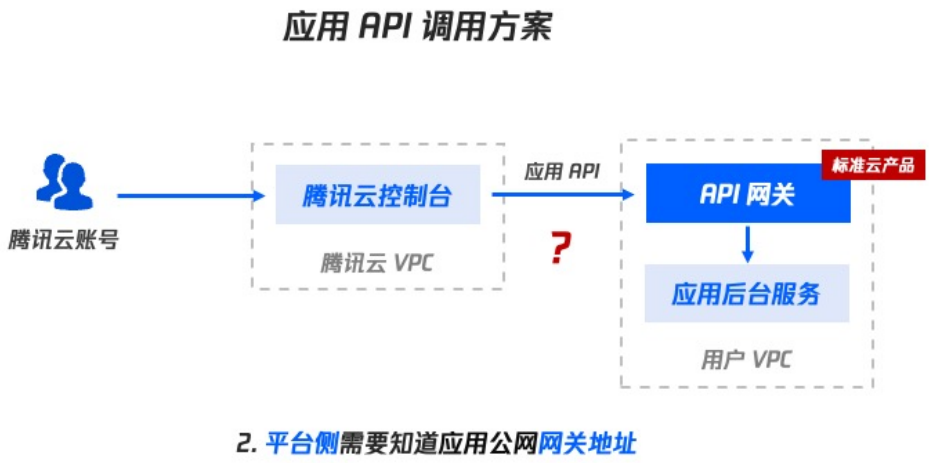
应用 API 调用方案

1. 应用 API 请求需要走公网

先从最简单的调用关系来看

首先用户使用的是腾讯云账号，访问的腾讯云控制台 Web 服务

那想要请求到 [用户VPC] 的 API，只能通过公网，所以应用要在 [用户VPC] 提供一个公网网关

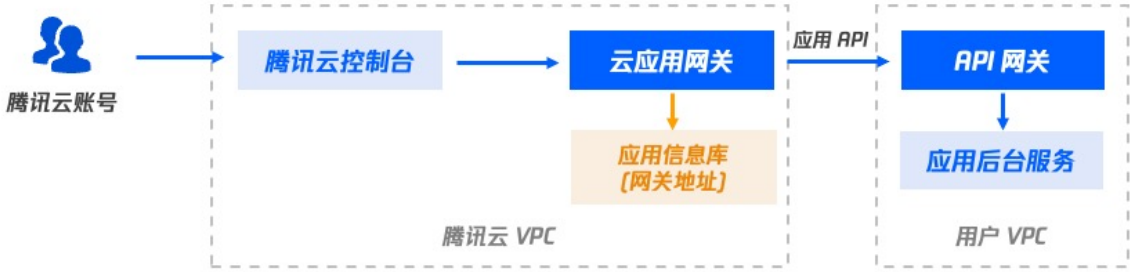


这里刚好有一个腾讯云标准的云产品 API 网关可以实现这个能力

但是想要调用到 API 网关的公网服务，我们还需要知道它的地址

2.3 云应用

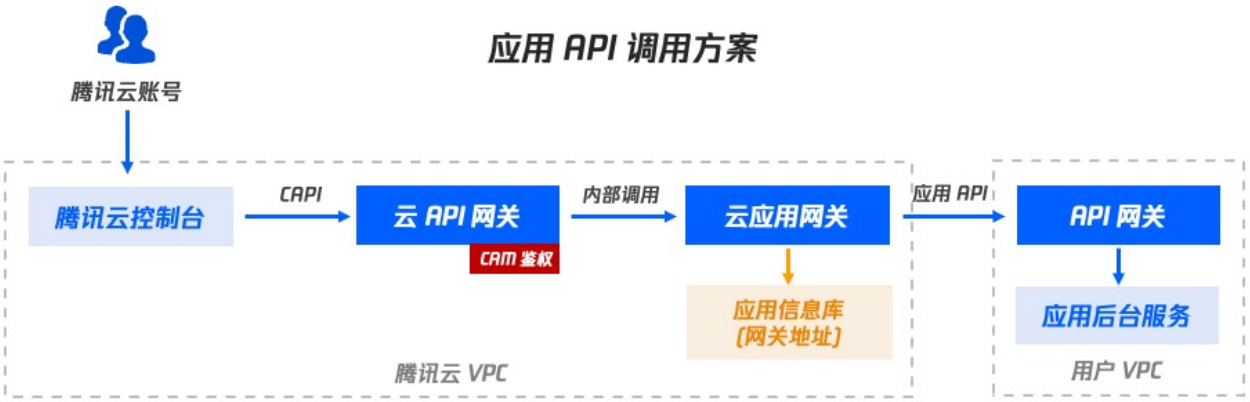
应用 API 调用方案



3. 云应用网关需要鉴权（确认用户有权发起调用）

所以我们还需要搭建一个云应用网关，它记录各个应用的 API 网关地址，在请求时查到对应地址并转发请求

同时 这个云应用网关需要限制只有拥有应用使用权限的人才能发起调用



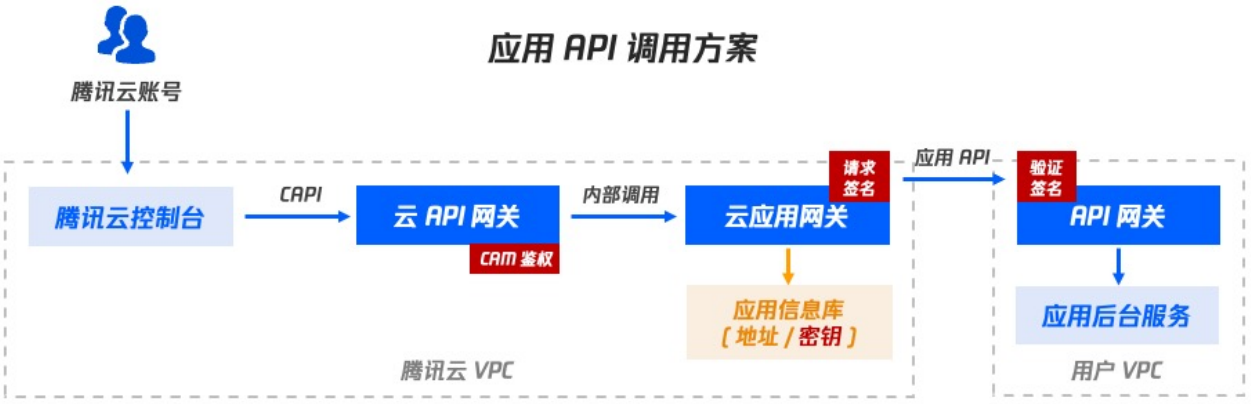
4. API 网关公网服务也需要鉴权

〔确认调用来自云应用网关〕

我们可以使用云 API 和 CAM 来确认用户身份，将用户身份信息由内网传递至云应用网关，

在对比信息库中的应用创建人信息后，便完成了应用使用权的校验

最后，应用的 API 网关这个公网服务也是需要鉴权的，它需要限制调用一定是来自云应用网关



API 网关怎么创建在用户 VPC？

地址和密钥怎么注册给云应用网关？

40

API 网关产品本身也提供了这个能力，我们在 API 网关生成密钥，也存入应用信息库，请求时拿到密钥进行签名，API 网关便可以验证身份

到此，整个调用链路就打通了，但是还有两个链路外的问题

API 网关怎么创建在用户 VPC？

地址和密钥怎么注册给云应用网关？

2.3 云应用

API 网关

- 标准云产品，可加入 Terraform 定义

- 编排引擎服务在应用创建完成后，获取 API 网关的公网地址及安全密钥注册写入云应用网关

```
# 声明 API 后端处理器，给定 TKE 集群中的 service id 即可。
resource "cloudapp_api_handler" "backend" {
    vpc_id = tencentcloud_vpc.vpc.id
    handler_type = "eks_service"
    eks_cluster_id = cloudapp_helm_chart.app.cluster_id
    service_name = "backend"
    handler_protocol = "http"
    handler_path = "/api/handler/:api_name"
    handler_method = "POST"
}
# 声明应用提供可调用的 API
resource "cloudapp_api" "GetTodoList" {
    handler_id = cloudapp_api_handler.backend.id
    api_name = "GetTodoList"
    api_desc = "获取待办事项列表"
}
```


因为 API 网关是腾讯云的标准云产品，所以我们同样在应用的 Terraform 配置中定义对应资源即可自动创建

在编排引擎服务完成安装后，可以获取到网关地址和密钥，注册写入云应用网关服务

云应用 成果

2.4 云应用 - 成果

当前已接入云应用平台的产品

线下人工交付	云应用交付
天/周级	分钟级

内部灰度测试中 ..

数款公司内产品正在调研接入

43

当前 云应用正在内网灰度测试，已经有 Coding 和用户资源迁移服务两款应用上线，并且有数款公司内的产品正在接入中

通过云应用 Coding 的交付时间从人工线下的 天级别 进化到了 分钟级

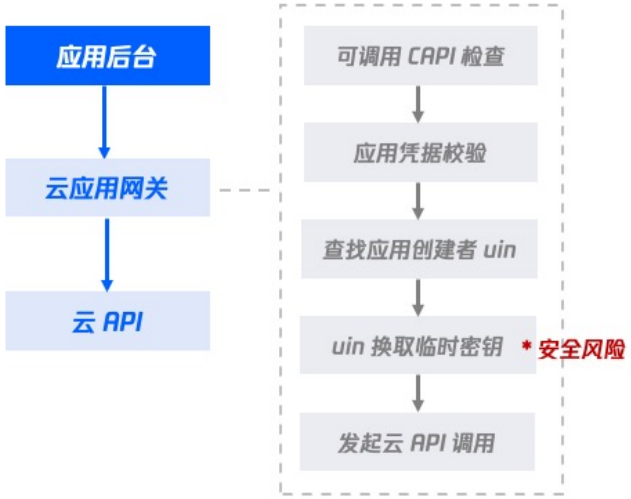
13:00

00

附录

0 「云应用」

接口代理（重网关）	
描述	应用后端统一从公网通过云应用的网关代理调用云 API
运行成本	高
安全风险	中
调用复杂度	简单
权限专业性	非专业
稳定性压力	高（数据流量大）
实现难度	低



首先是最容易想到的接口代理方案：

我们给每个应用安装时下发一个身份凭据，

应用持凭据通过公网调用云应用的一个公网服务，

服务验证凭据后查回安装者 UIN

使用账号内部接口用 UIN 申请一个临时密钥发起 CAPI 调用并返回结果

这个方案下所有请求数据流量都压在我们网关服务上，稳定性和运营成本会有很大问题

同时服务在经过我们自己实现的鉴权后换取了用户密钥，可能存在安全风险

0 「云应用」

密钥分发（轻网关）	
描述	应用后端通过云应用公网接口申请云 API 临时密钥调用云 API
运行成本	一般
安全风险	高
调用复杂度	复杂
权限专业性	非专业
稳定性压力	一般
实现难度	低

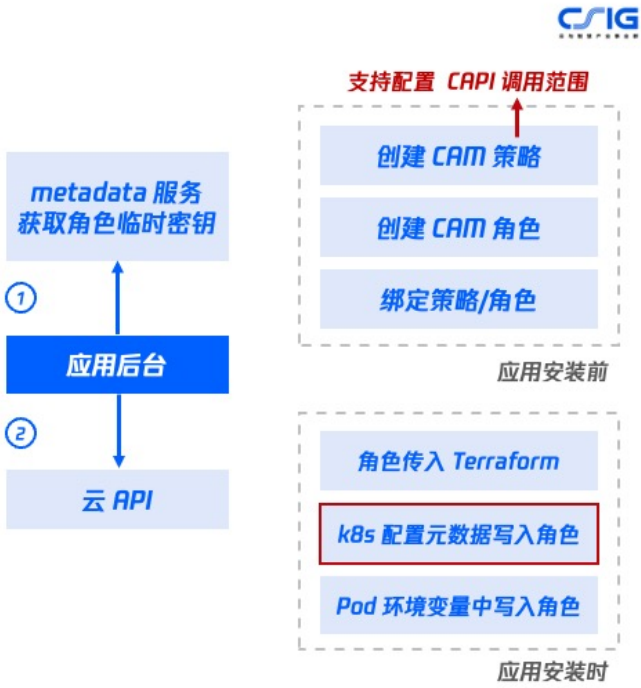
为了解决流量压力，我们可以不代理调用 云API 接口，只返回密钥

用户需要通过我们接口获取临时密钥后自行请求 云API

虽然服务压力减小了，但现在我们的接口变成了一个能换取到用户密钥的接口，安全风险变的更大了

0 「云应用」

服务角色（无网关）	
描述	应用启动时预埋应用服务角色信息到进程环境变量中，应用扮演角色获取临时密钥调用云 API
运行成本	低
安全风险	低
调用复杂度	复杂
权限专业性	专业（全交由 CAM）
稳定性压力	低（压力在云 API）
实现难度	高



因为前两个方案都不是特别理想，在进一步了解 CVM/TKE 和 CAM 的一些能力后，我们得到了最终的标准合规的方案

首先方案依赖于 CVM 在内网的 ***metadata 服务中提供的使用 CAM 角色换取临时密钥的能力***

同时 TKE 通过在集群配置的元数据中声明角色信息后可以在背后母机获得这个能力

所以我们只需要通过 CAM 创建调用策略和角色，并按要求配置后，就可以在内网原生获得获取临时密钥的能力

并且 CAM 策略可以直接约束 CAPI 调用范围，整个过程我们不需要实现任何服务，

只需要处理 安装前 安装时 的一些环节

0 「云应用」

资源编排中 resource 的实现

- 通过 *resource* 来支持某种 *云资源* 或 *资源间关联* 的创建、销毁、查询的方式

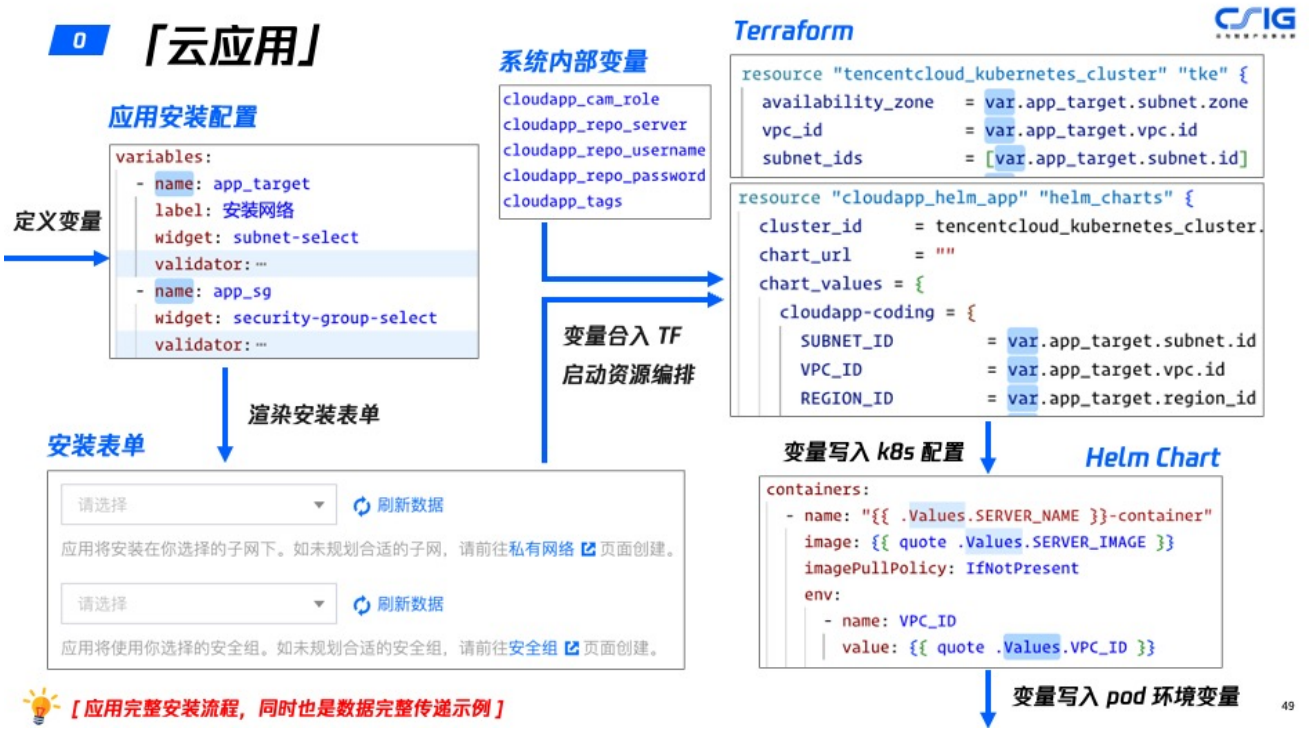
实现上，每个 resource 就是
通过 云 API 完成对应资源创建、销毁、查询的一组方法

```
export type ResourceType<S> = {
  /**
   * 创建资源，通过云产品的云 API 创建/走计费下单流程
   */
  create: API<S>;

  /**
   * 通过实例 ID 获取资源详情
   */
  read: API<S, ResourceData<S>['newState']['Attributes']>;

  /**
   * 销毁资源实例
   */
  delete: API<S>;

  /**
   * 对现有资源进行变配
   */
  update?: API<S>;

  /**
   * 导入现有资源
   */
  importer?: API<S>;
```

那么整个资源编排流程是怎么实现的

首先 我们要对每种云资源按照 Terraform 的规范来实现 *resource*

每个 resource 实际上 就是编写*一组方法*，通过调用云 API 完成一个云资源*创建、销毁、查询等*



解决了安装配置，就完成了应用的完整安装方案，同时这个方案下也打通了各个环节的数据传递

从最前置的用户输入变量算起，过程中的数据可一步步流通至应用 pod 环境中，最终可供应用服务读取