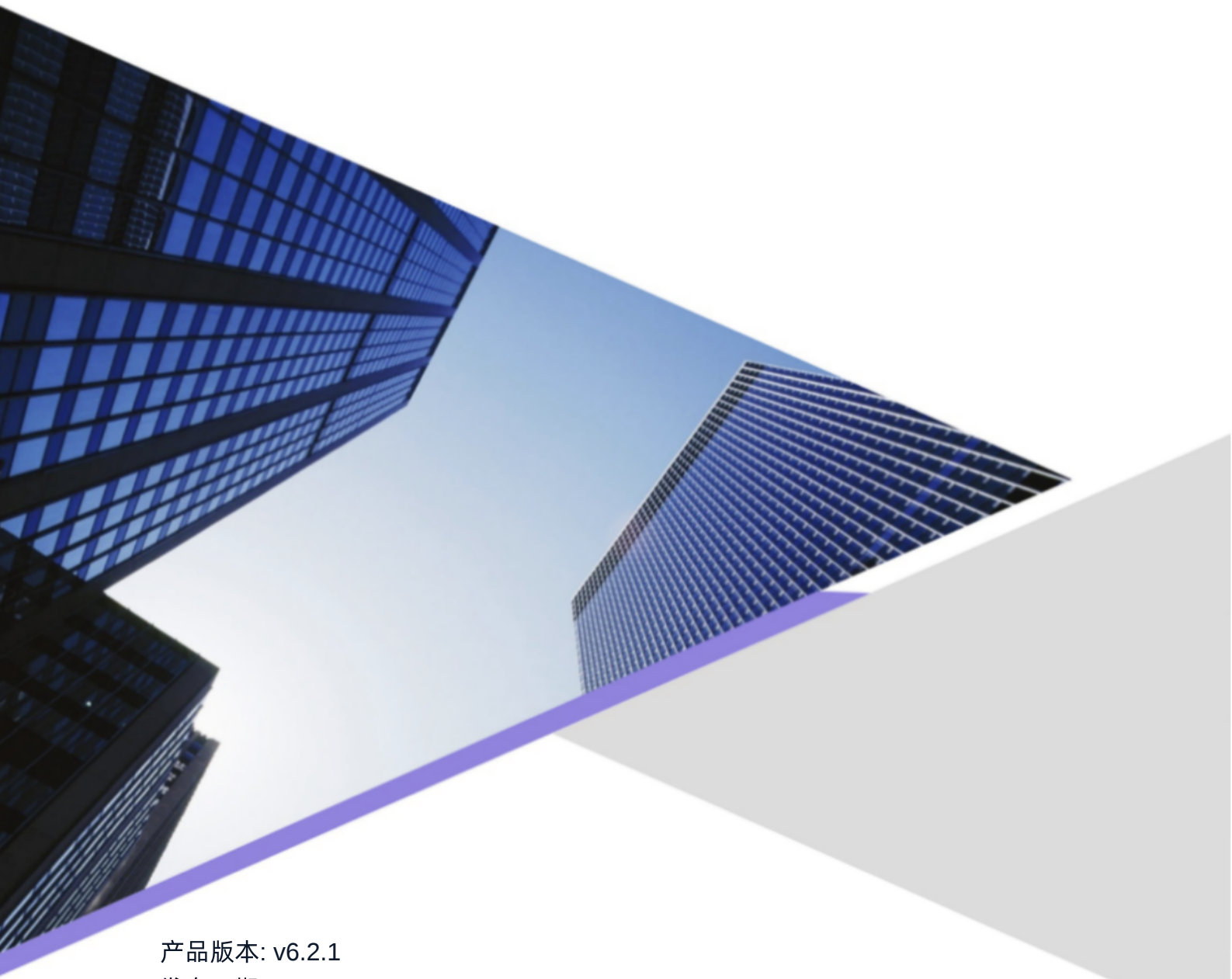
EasyStack
易捷行云

数字原生引擎 EOS

开发指南



产品版本: v6.2.1

发布日期: 2024-10-11

目录

1 开发指南

1.1 云产品

1.1.1 阅读指引

文档主题

本文档介绍了云产品的定义以及如何基于云开放平台开发云产品。

目标读者

本文档适合将要基于云开放平台进行云产品开发工作的相关人员，以及想要了解云产品及如何开发及使用的人员。

阅读建议

建议首先阅读**云产品介绍**章节，该章节中将总体介绍云产品的定义、符合规范的云产品由哪些部分组成。在对云产品有了整体了解后，可依次或按需阅读其余章节，了解开发规范细节。 阅读过程中若对某些专用名词产生疑惑，或许可以参考**术语**章节中的术语介绍。

约定

- 本规范均以名称为“mycloud-product”的云产品举例说明。

1.1.2 简介

什么是云产品

云产品是一组云端程序，在运行时为用户提供云服务，基于数字原生引擎，遵循云原生规范，以可进化为设计原则，具备所见即所得的能力，可以按需获取、使用、升级、卸载。 云产品开发规范完全兼容k8s云原生架构，开发者可以依据规范自主开发的提供云服务的产品。开发完成后云产品可通过OTA发布到云产品市场中，用户可以在云产品市场中浏览所有云产品，并付费订阅云产品。用户购买并安装云产品后，云产品服务目录将集成到云开放平台中，实现统一访问入口。

云产品运行环境

开放平台会创建云产品节点用于承载云产品运行，云产品节点可以是虚拟节点，也可以是物理节点。可供用户灵活选择，通过开放平台统一进行管理。

1.1.3 相关术语

- **OTA**：Over-the-Air Technology 空中下载技术。云产品开发完成后可以发布到易捷行云OTA服务器中，用户可以从OTA中获取需要的云产品。
- **Helm Chart**：Helm chart是用来封装Kubernetes原生应用程序的 YAML 文件，可以在部署应用的时候自定义应用程序的一些 metadata，便于应用程序的分发。
- **Service**：Service是Kubernetes中的一种抽象资源，它定义了一组Pods的逻辑集合和一个用于访问它们的策略。一个Service的目标Pod集合通常是由Label Selector来决定的。Service的类型有多种，一般采用ClusterIP的方式来提供各个内部服务和ingress调用。
- **Ingress**：Ingress用于暴露从集群外到集群内服务的HTTP或HTTPS路由。可以配置用于提供外部可访问的服务url、负载均衡流量、SSL终端和提供虚拟主机名配置。
- **独立云产品**：具有完整云产品生命周期，拥有独立namespace，资源完全隔离的云产品。
- **Foundation 云产品**：在openstack namespace下，共同组成平台基础平面（ECF）的服务。Foudantion云产品不应包含ESCL、EOS、Ceph等系统级别服务。
- **云产品ID**：云产品的唯一标识，与chart name和release name 保持一致，比如计算服务对应的云产品ID是nova.

1.1.4 Hello World

通过**云产品介绍**章节，相信您已经对云产品有了基本的了解。而在后续更加详细地了解云产品组成及开发细节前，您可以通过本章节快速感受一个云产品的制作与发布过程，我们将为您提供一些示例资源及操作指导，以便您从整体角度快速上手。

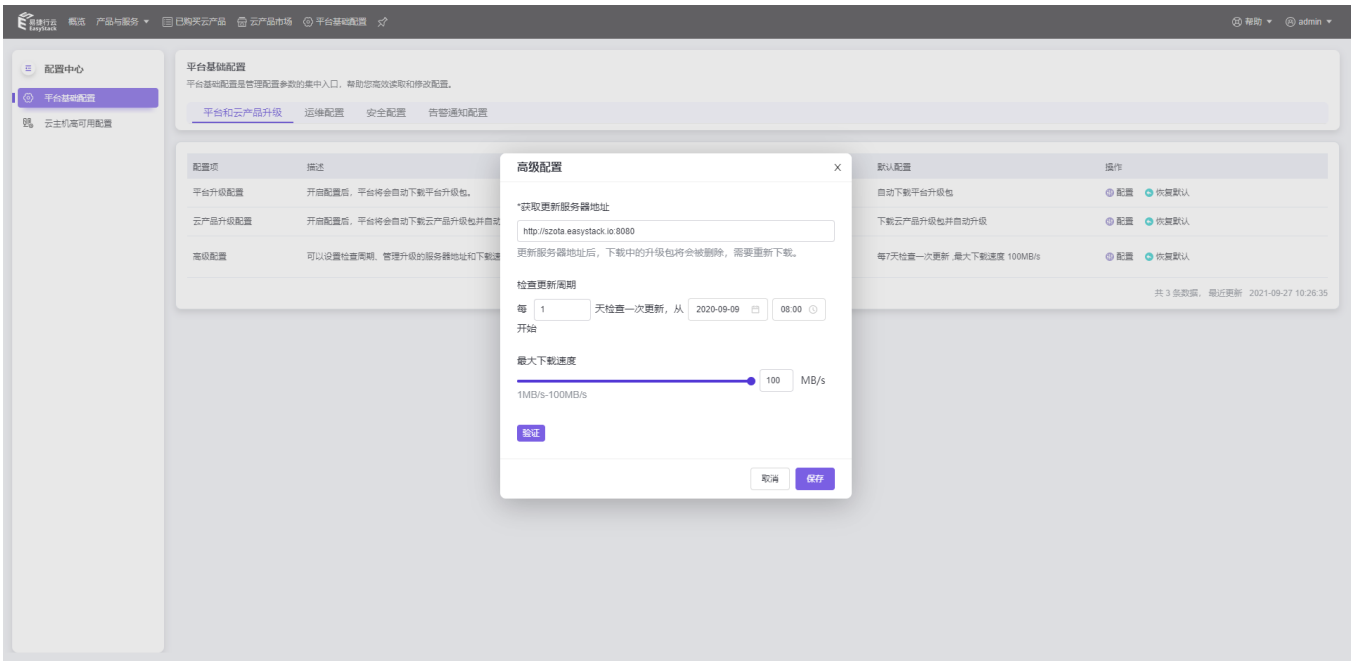
先决条件

已部署好ECP云开放平台

安装云产品

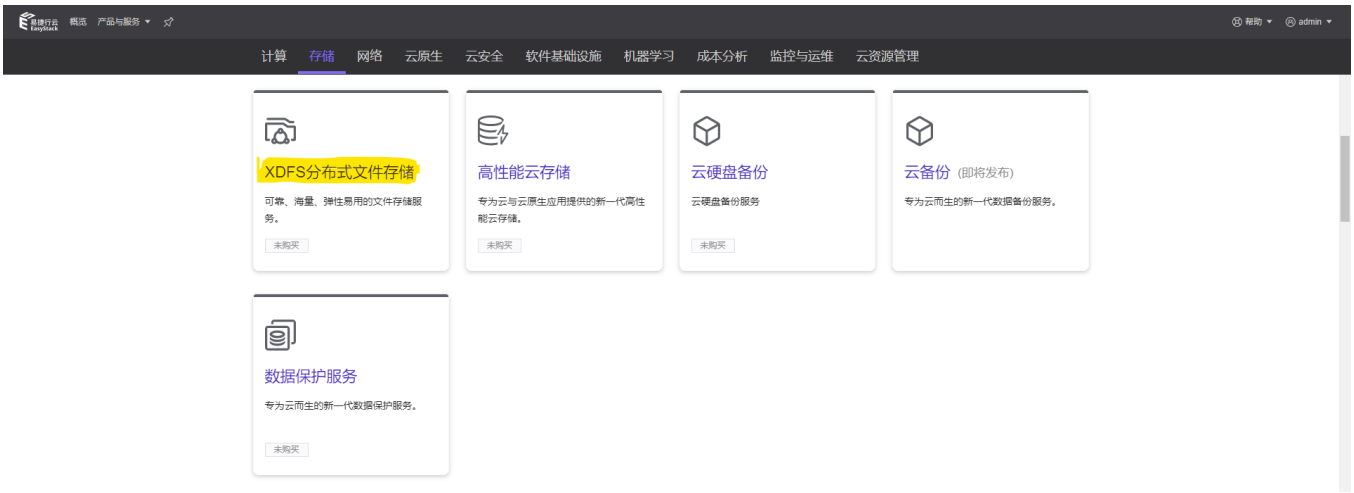
1. 配置对应的 ota server

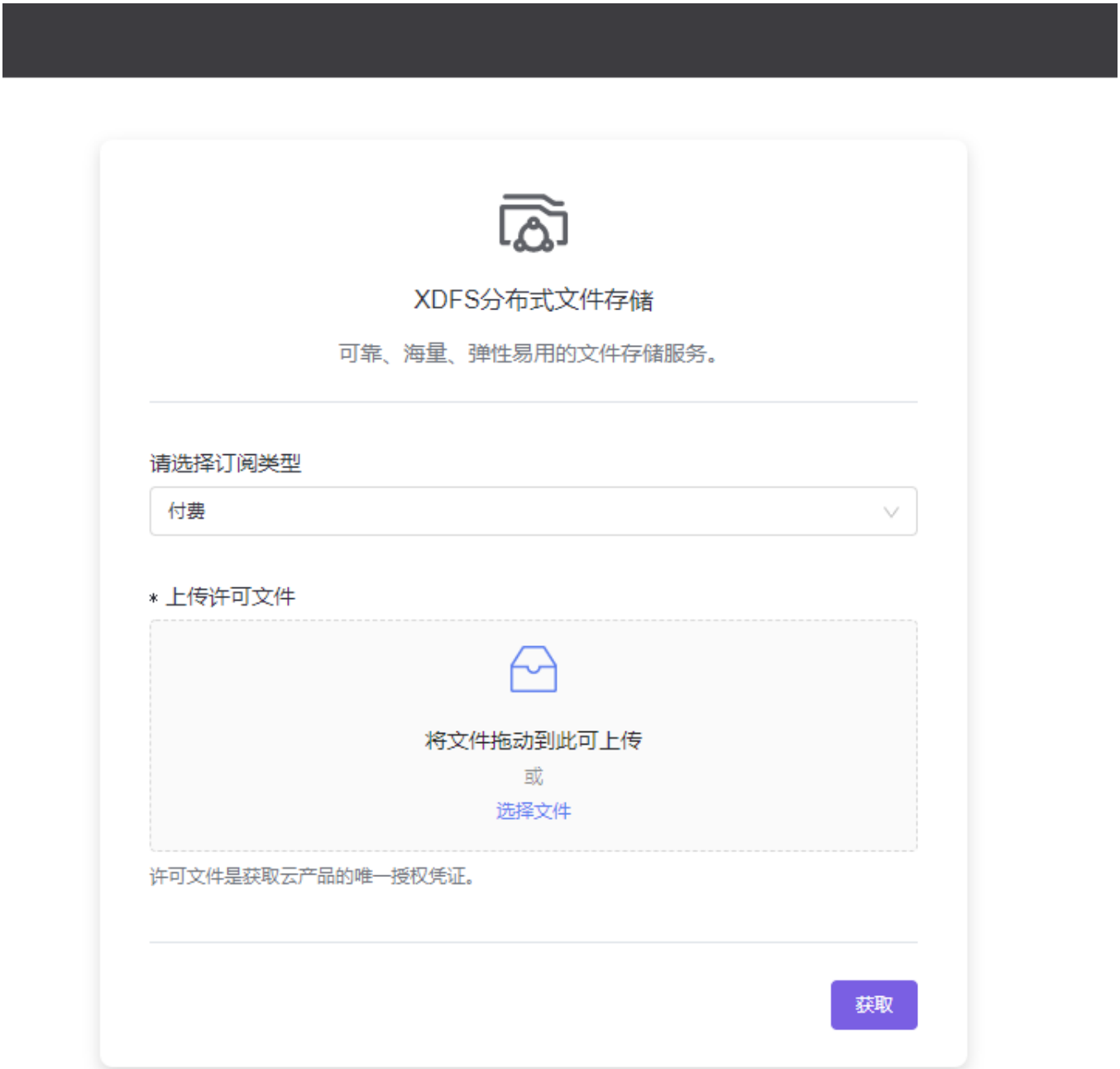
产品与服务》配置中心》平台基础配置》平台和云平台配置》高级配置

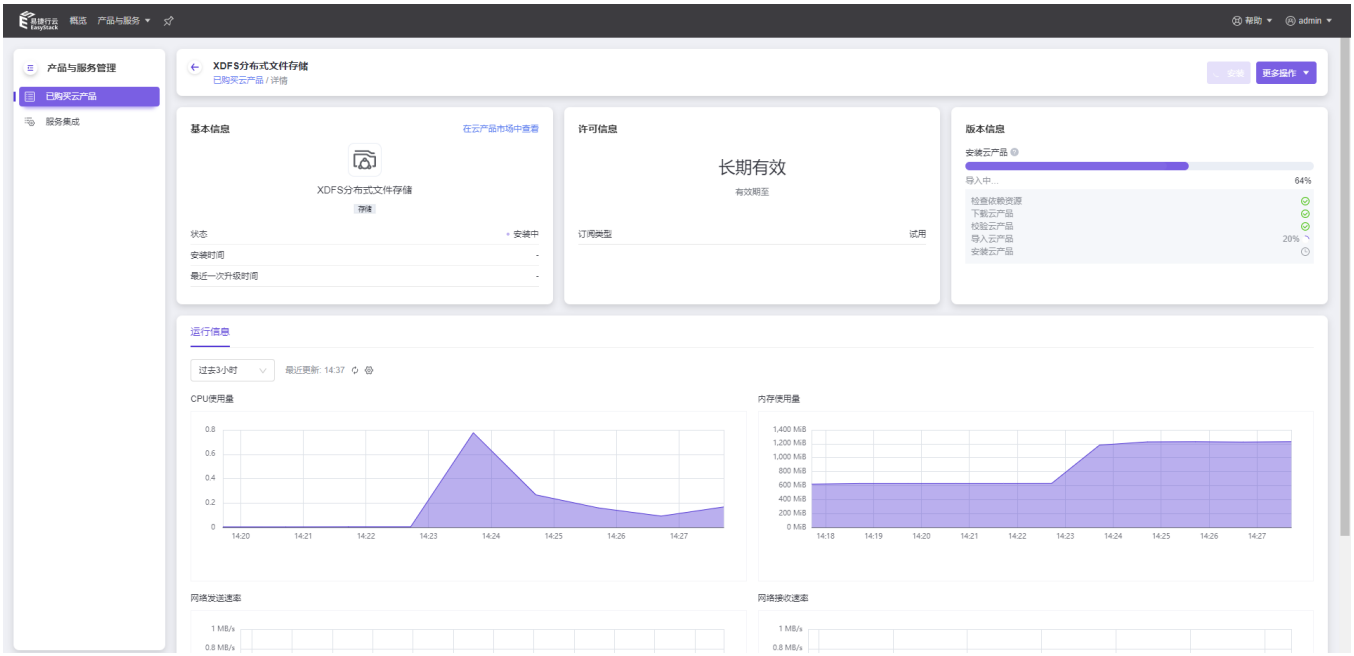


2. 在云产品市场中定位到发布的云产品，获取云产品授权，然后安装

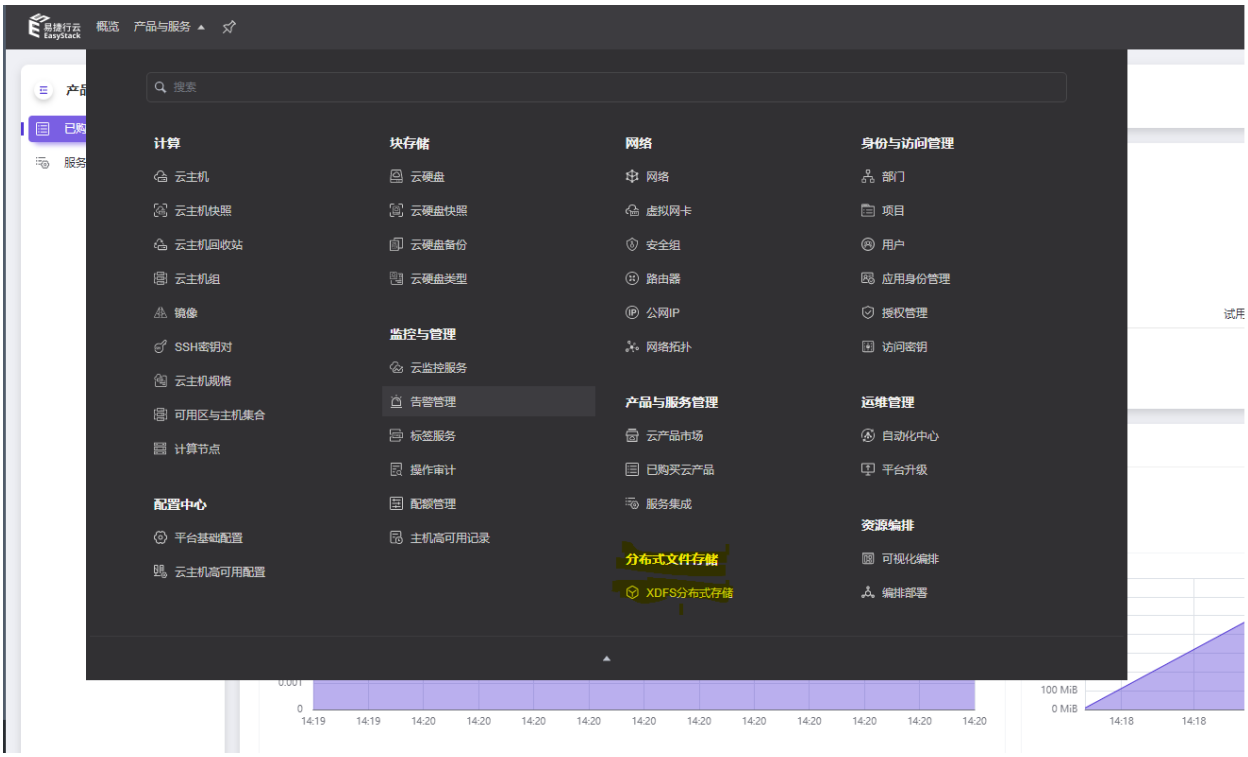
产品与服务》产品与服务管理》云产品市场




XDFS分布式文件存储
可靠、海量、弹性易用的文件存储服务。
请选择订阅类型
付费
* 上传许可文件
将文件拖动到此可上传
或
选择文件
许可文件是获取云产品的唯一授权凭证。
获取



3. 安装完成后，可以通过上导 *产品与服务* 直接访问云产品提供的服务



升级云产品

卸载云产品

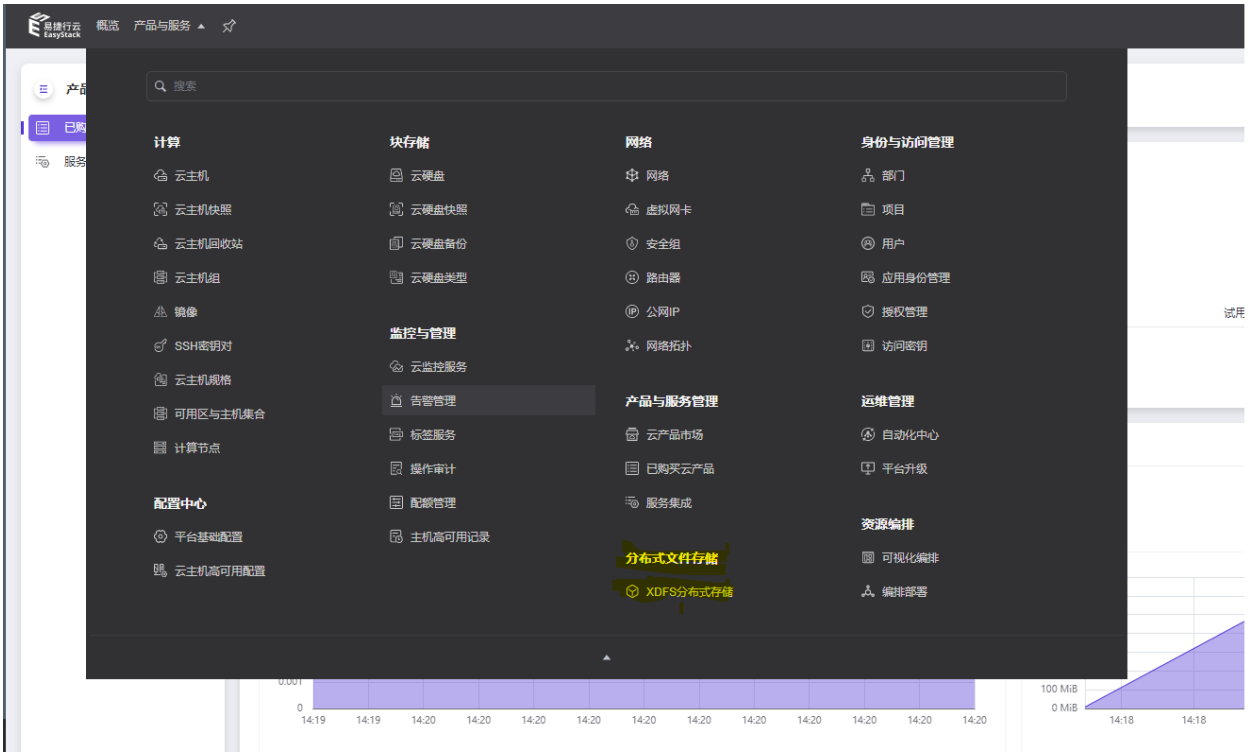
1.1.5 常见问题

Foundation 云产品和云产品有什么区别？

Foundation 云产品是指 V6 由 ECP 接管的之前由 ECAS 统一管理的服务，例如：nova、neutron、horizon 等。由于之前 ECAS 有很多安装业务逻辑，短期内很难完全拆分到每个云产品中，所以 V6 版本 Foundation 云产品的安装、配置还是由 ECAS 负责，云环境安装好之后会由 ECP 接管并负责云产品后续升级业务。因为我们称平台为 ECF Foundation，所以从 Foundation 拆分出来的云产品就被统称为 Foundation 云产品了，Foundation 云产品只是一个过渡状态，最终都会演进到以统一的云产品规范标准进行开发与发布。 我们新开发的云产品都是标准云产品，是无法新开发一个Foundation 云产品的。

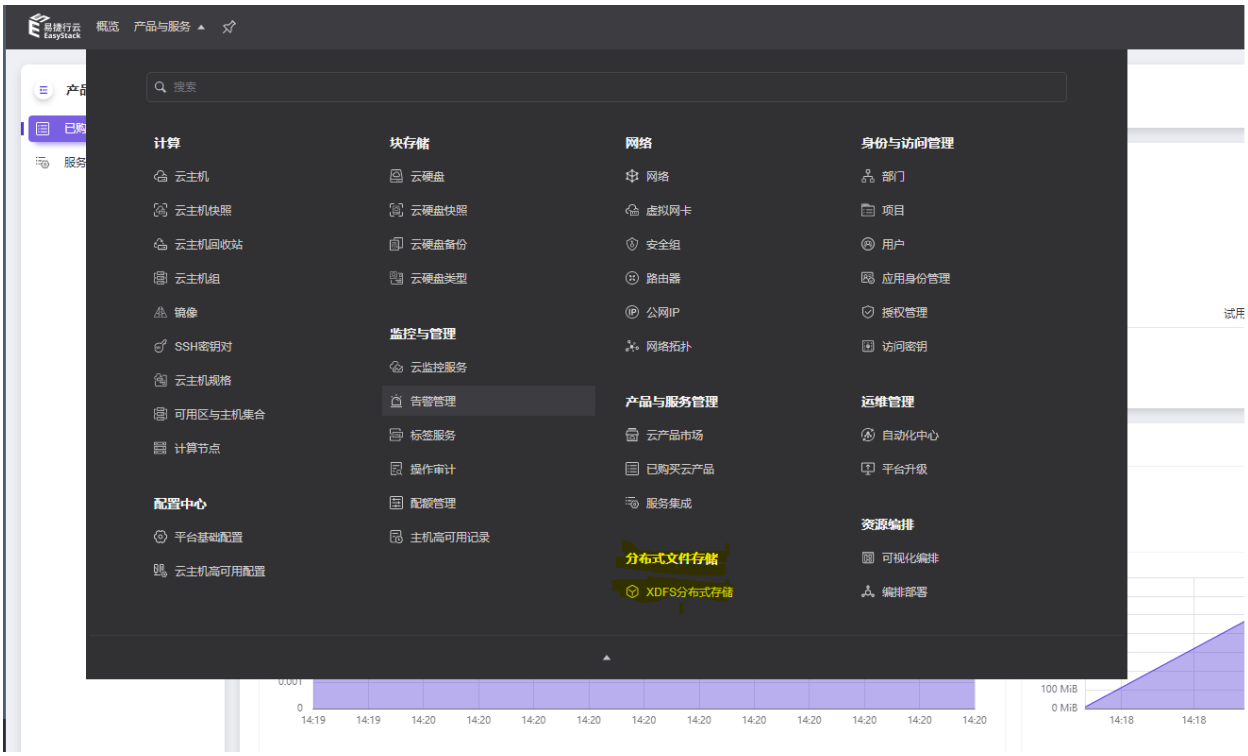
云产品依赖关系该如何定义？

云产品可以通过 _config_helpers.yaml 的 dependencies 属性声明对平台和其他云产品的版本依赖，例如：EKS云产品依赖平台最低版本为6.0.1，并且依赖容器镜像服务的最低版本为6.0.1，可以通过以下方式声明。



云产品间声明原则：如果缺少某个云产品无法正常运行，必须将其设置为依赖。ECP v6.0.2会开放Foundation云产品删除能力，所以如果依赖到Foundation云产品，也必须要设置为依赖。 新增云产品流程：新增云产品

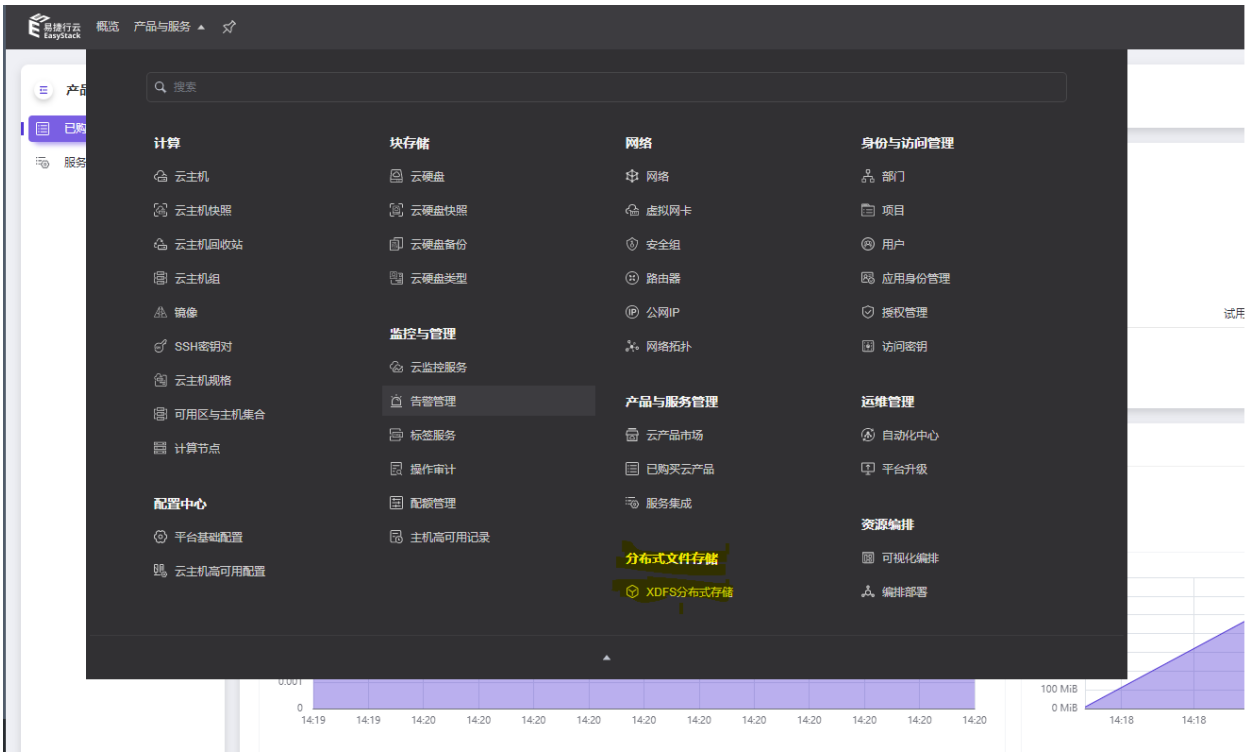
流程 ECP v6.0.2版本删除云产品的流程：此流程兼容v6.0.1云产品。



云产品如何使用 License 控制功能边界?

云产品 License 分为试用和正式两种，当两种许可同时存在时，正式 License 权重高于试用 License 。试用许可参数控制可以通过 _config_helpers.yaml 的 trial_values 属性进行声明，正式许可的参数控制需要在出

License时通过values属性声明。



ECP开放平台前端框架都包含什么？

前端框架包含 ecp-dashboard 和 组件库 两部分，前端框架跟随平台版本发布，例如：6.0.1 平台版本就会发布一个 6.0.1 版本的前端框架。所有云产品开发前端页镜像 build 都默认使用最新已发布版本前端框架。

计算、存储、网络等拆分为独立云产品的目的是什么？

目的是更彻底的实现平台与云产品分离，实现计算、存储、网络等云产品独立版本发布。 目前在做的拆分工作是V6.0.1 Foundation云产品拆分方案的正常推进。原方案设定先将horizon管理界面发布一个独立的云产品，完全和后端分离，计算、存储等云产品不带界面，只发布API能力。horizon云产品和计算、存储、网络等云产品独立发布与升级。 V6发布以后，每个云产品单独开发页面，逐渐替换horizon页面。

云产品界面上提跟多区域 iframe 方案有关系吗？

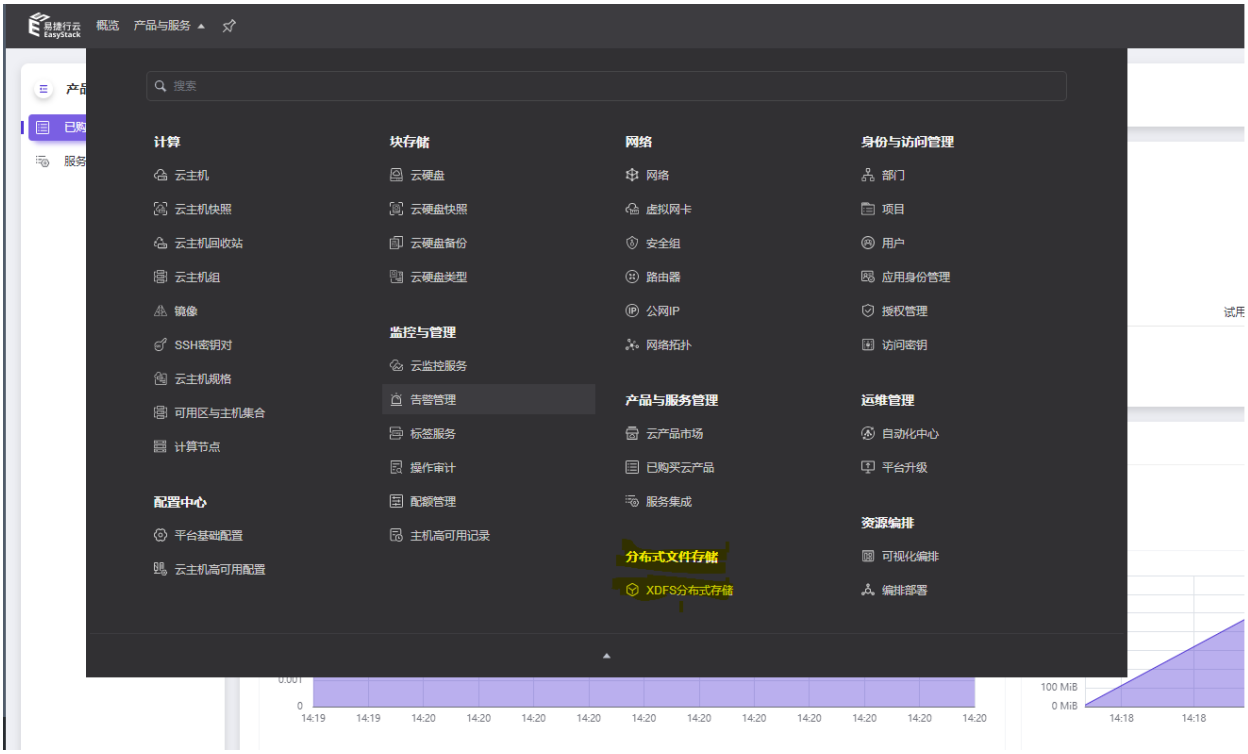
没有直接关系。多区域iframe方案是为了避免修改horizon而提出的兼容性方案，否则，在当时修改horizon会带来很大而且不可控的工作量，并且在V6发布以后将会通过计算、存储、网络云产品拆分逐渐取代horizon。

云产品如何支持多区域？

每个云产品需要发布独立的 SDK ，开放云产品 API 供外部使用。云产品支持多区域需要从 SDK 级别做支持，而不是业务API支持。例如：计算云产品支持多区域，需要支持根据不同区域认证信息封装 python-novaclient 以调用不同区域 nova 服务 API 接口。

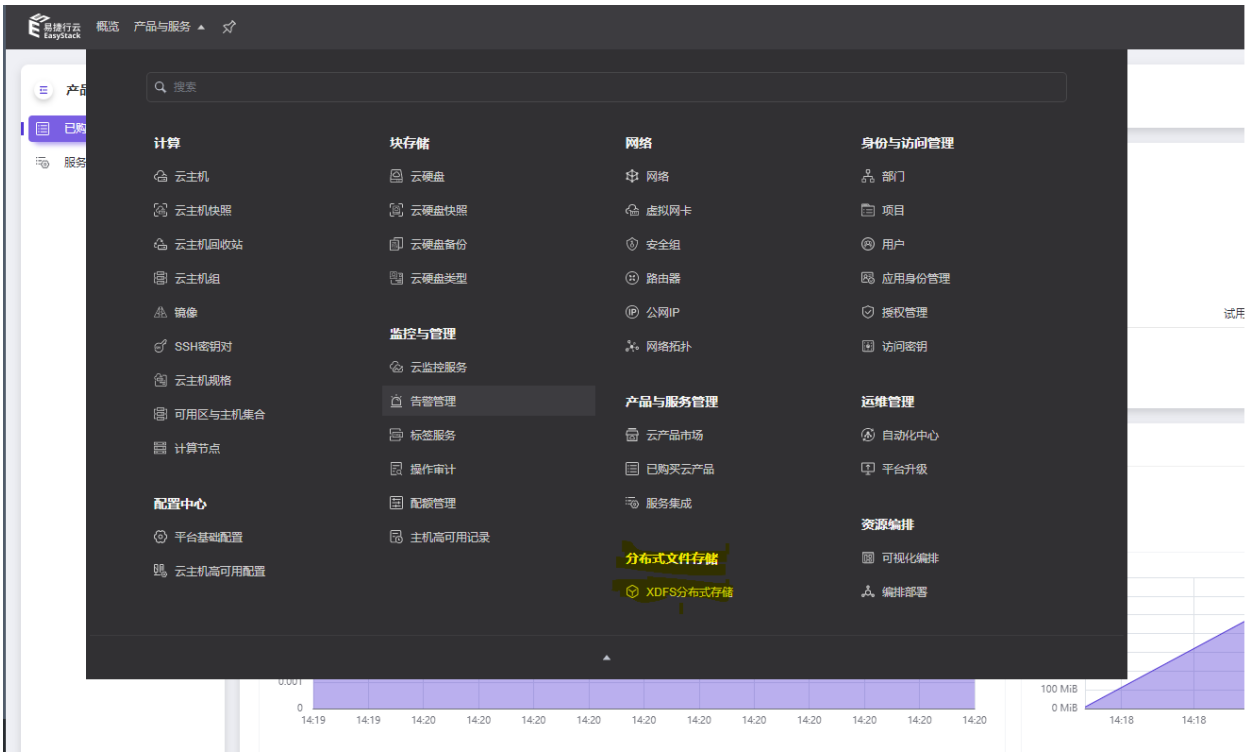
Matrix、SDK、多区域之间有什么关联关系?

统一版本发布时，随着业务层逐渐增多，一些公共逻辑需要能够在横向业务间复用，因此将产品业务逻辑统一下沉到 Matrix 项目，类似中台的概念。Matrix 项目定位是公共业务接口，Matrix 会对原生 SDK 进行二次封装，为业务层提供抽象接口，通过使用不同区域的认证信息实现后端服务多区域接口调用。



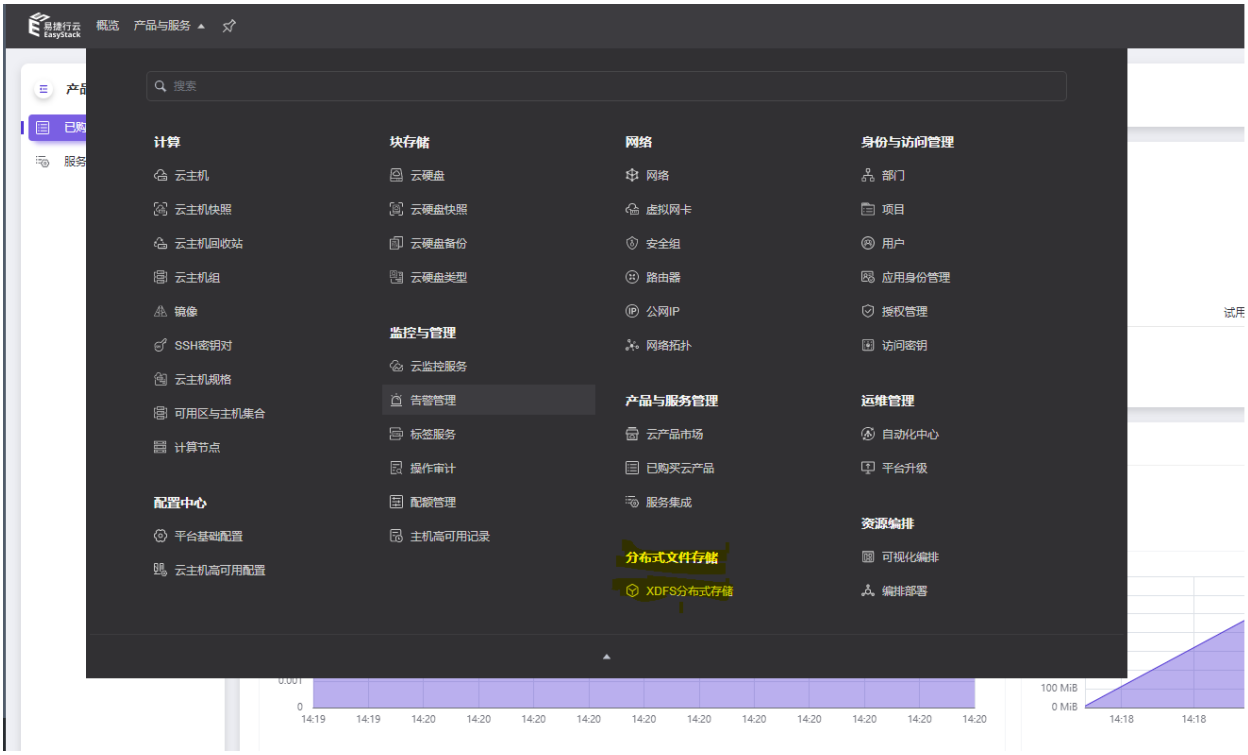
V6 平台与云产品分离之后，各云产品的 SDK 不再统一版本发布，所以每个云产品的 SDK 需要独立版本发布，不用再通过 Matrix 项目封装。这也要求云产品业务能够下沉到统一的 Open API，通过 SDK 对外暴露能

力。多区域功能的实现主要是能够使用不同区域认证信息封装 SDK，以调用不同区域后端服务的接口。



云产品导航如何支持打开新标签页？

如果云产品的某个页面需要支持打开新页面的方式跳转，可以在云产品的 ark 定义里 _navigation.yaml 中添加 url 和 layout 定义，url 和 src 的值相同，layout 设置为 outer。



云产品打包依赖 CMS 数据吗？

CMS 中保存了云产品的中英文产品营销素材，包括名称、描述、icon、详情信息等数据，云产品打包时会根据云产品唯一标识从 CMS 获取营销素材数据，如果找不到对应的数据会导致打包失败。所以在云产品立项之后，必需在 CMS 添加对应的云产品，并设置为 publish 状态。如果不需要在官网展示，可以将 Hidden 字段设置为 OFF。

云产品发布到 OTA 之后，控制台云产品列表页面为什么查询不到？

云产品打包成功，发布到 OTA 之后，在控制台查询不到有两种可能的原因： 1、云产品的架构与当前平台不一致，例如：OTA 中只有 x86 架构的云产品，但当前云平台是 arm 架构。 2、CMS 中没有将云产品加入到分类（Cloud Product Categories），导致 OTA http://{OTA_URL}/service_catalog/{x86|arm}/catalog.json 文件中没有对应的云产品索引，所以不会在控制台中展示。

1.1.6 基础

1.1.6.1 概述

云产品 Chart 包兼容 Helm Chart 的规范，并针对ECP平台进行加强，方便开发者使用云平台的各项能力，有 Helm Charts 的开发者可以快速上手，已有的 Helm Chart 包也可快速迁移到ECP平台。

一个典型的ECP云产品 Charts 整体目录结构组成如下：

```
— mycloud-product
├── Makefile                                #云产品Makefile
└── chart
    └── mycloud-product
        ├── Chart.yaml                                  #云产品基本信息
        ├── requirements.yaml                         #chart依赖，不存在时可忽略
        ├── values.yaml                         #云产品数据配置
        └── templates
            ├── _acl_helpers.yaml.tpl       #权限配置
            ├── _config_helpers.yaml.tpl    #云产品运行时及依赖
            ├──  deployment.yaml            #云产品Deployment
            ├──  service-api.yaml           #云产品Service
            ├──  ingress-api.yaml           #云产品ingress
            ├──  job-hooks.yaml             #云产品安装资源检查
            ├── _navigation_policy.yaml     #云产品左导航访问策略
            ├── _navigation.yaml            #云产品左导航
            └── _wizard.yaml                #使用向导
```

1.1.6.2 云产品信息

声明云产品基本信息

云产品 Charts 包中， `Chart.yaml` 文件在 Helm Chart 的规范基础上，需要定义云产品基本信息。

云产品标识

云产品唯一标识（cloud product id）用于唯一指定一个云产品，需要和chart包名一致，例如本规范的 `mycloud-product` ，chart名称必须是小写字母和数字。单词之间可以使用横杠分隔(-)，**不能用大写字母也不能用下划线，点(.)符号也不行。**

涉及参数如下：

```
name: mycloud-product
```

云产品分类

> ECP 6.0.2 变化：云产品分类通过OTA服务器指定，不再通过云产品声明

涉及参数如下：

```
keywords:
 - catalog:Cloud Native
```

当前支持的云产品分类包括：

- 计算（Computing）
- 存储（Storage）
- 网络（Network）
- 云原生（Cloud Native）
- 云安全（Cloud Security）
- 软件基础设施（Software Infrastructure）
- 机器学习（Machine learning）

- 成本分析（Cost Explorer）
- 监控与运维（Monitoring & Operation）
- 云资源管理（Cloud Resource Management）

云产品开发者

> ECP 6.0.2 变化：添加了 creator: 标识

涉及参数如下：

```
keywords:
    - creator:ems_service_chart
```

当前支持的云产品开发者包括：

- ems_service_chart：普通云产品
- foundation：基础云产品
- external：外部云产品

云产品名称

涉及参数如下：

```
keywords:
 - name_zh_CN:我的云产品
 - name_en:mycloud-product
```

云产品描述

涉及参数如下：

```
keywords:
- zh_CN:我的云产品描述
- en:description of mycloud-product
```

完整示例

```
apiVersion: v1
description: Helm mycloud-product
name: mycloud-product
version: 1.0
home: https://docs.openstack.org/mycloud-product
icon: https://www.openstack.org/mycloud-product.png
sources:
 - https://git.openstack.org/
 - https://git.openstack.org/mycloud-product

maintainers:
 - name: OpenStack-Helm Authors

keywords:
 - creator:ems_service_chart
 - zh_CN:我的云产品描述
 - en:description of mycloud-product
 - name_zh_CN:我的云产品
 - name_en:mycloud-product
 - catalog:Cloud Native
 - creator:ems_service_chart
```

注意事项：keywords下属的字段“:”之间没有空格。

1.1.6.3 云产品权限以及资源声明

云产品权限声明

> ECP 6.1.1：支持云产品声明安装的命名空间，新增云产品安装资源需求声明

云产品 charts 包中 `_acl_helpers.yaml.tpl` 文件用于定义云产品的所需权限、资源及安装的命名空间。

目前需要定义云产品的psp策略、cluster_role策略以及pvc策略 **psp_type**：云产品需要使用的psp类型，目前支持Unlimit和Limit **cluster_role**：云产品需要绑定到的cluster role，目前支持cluster-admin **pvc_enable**：云产品是否需要使用pvc，布尔值 **namespace**：指定云产品**安装**的命名空间，若无此声明时，则默认使用云产品标识作为云产品安装的命名空间。

示例

```
data:
  psp_type: Unlimited
  cluster_role: cluster-admin
  pvc_enable: true
  namespace: openstack
```

云产品资源需求

云产品默认会被调度到集群中的云产品节点中，安装或升级云产品前，会比对云产品节点资源是否满足云产品的需要，若不满足会触发云产品节点扩容。因此部署到云产品节点的云产品需要按规范声明资源需求。

注意：

1. 云产品 Chart.yaml 中 keywords 下声明了 `foundation` 、 `creator:foundation` 的云产品不会检查资源需求
2. 云产品 `_acl_helpers.yaml.tpl` 中指定了安装命名空间为 `namespace: openstack` 的云产品不会检查资源需求
3. 云产品需要安装到控制节点，同时没有按如上两种方式声明 keywords 或 namespace，需要参考_自行声明资源需求 _章节声明云产品资源需求为 0，避免检查资源。

4. 由于云产品节点的资源限制，云产品资源需求需要限制最大不超过 6 C 12 G

自动计算资源需求

在编写云产品 charts 时推荐在 values.yaml 统一声明云产品相关资源的资源需求到 `resources` key 下，构建云产品时会通过 values.yaml 中**所有 resources 字段**下的资源声明 `requests` 数值，自动计算出一个资源限制的值。

注意：如果 charts 中声明了资源，但是该资源有控制选项，实际部署过程中不一定部署，仍然可以可以保留该资源的需求声明，因为云产品的资源需求按最多需求资源来计算的

目前推荐有三档资源声明：

```
requests:
  cpu: "200m"
  memory: "512Mi"
```

```
requests:
  cpu: "500m"
  memory: "1024Mi"
```

```
requests:
  cpu: "1000m"
  memory: "2048Mi"
```

若云产品需要使用其它数值的自定义资源声明，建议 limits 的数值设置不超过 requests 的 1.5 倍。

示例

```
pod:
  resources:
    jobs:
      hook:
        requests:
          memory: "200Mi"
          cpu: "512m"
      db_init:
        requests:
```

```
          memory: "1024Mi"
          cpu: "500m"
    backend_api:
      requests:
        memory: "1024Mi"
        cpu: "500m"
    front_dashboard:
      requests:
        memory: "512Mi"
        cpu: "200m"
```

当云产品在 values.yaml 中按如上示例声明了资源需求后，构建云产品时会自动统计 values.yaml 下 `resources` 下除 **jobs **外各层级的 **requests **资源声明。构建完成后此云产品对应的资源需求为：cpu：700m memory：1536Mi

自行声明资源需求

建议云产品优先考虑在 values.yaml 中明确定义每项资源的资源需求，若自动计算的资源需求不能准确反应云产品实际所需的资源需求，或有其它特殊场景的考虑，如云产品需要安装到控制节点的非 openstack 命名空间下，无需检查资源；云产品可在 _acl_helpers.yaml.tpl 文件中自行声明所需的资源，构建完云产品后会优先此值作为云产品的资源需求，单位：max_cpu：m，max_memory： Mi。

示例

```
data:
  psp_type: Unlimited
  cluster_role: cluster-admin
  pvc_enable: true
  namespace: openstack
  resource_limit:
    max_cpu: 500
    max_memory: 1200
```

当云产品在 _acl_helpers.yaml.tpl 中按如上示例声明了资源需求后，构建完成后此云产品对应的资源需求为：cpu：500m memory：1200Mi

1.1.6.4 云产品配置及依赖

云产品 charts 包中 `_config_helpers.yaml.tpl` 文件用于定义云产品配置及其依赖。

定义云产品配置

> ECP 6.0.2：replicas_by_modality 支持通过 License 传参修改副本数 ECP 6.1.1：新增 safe_render 和 safe_render_multi模板函数

说明

`_config_helpers.yaml.tpl` 文件中 `data_template` 属性用于定义云产品的配置模板，该配置会在渲染后生成云产品配置传递给 `helm` 用于部署云产品，该模板使用mako template语法，并包含了如下的模板函数：

- `${cloud_product_label}` ：渲染为云产品节点label。
- `${label_enabled}` ：渲染为云产品节点label的value，默认为enabled。
- `${random_password('password_tag', 8)}` ：生成一个随机密码，，同一个password标识生成的随机密码在首次生成后，再次升级将不会改变。
  - param1：password标识
  - param2：password的位数
- `${replicas_by_modality}` ：平台设置的副本数，参考：通过 License 传参修改副本数。
- `${safe_render('product', '/path', 'default_value')}` ：获取某个云产品的配置。
  - param1：云产品标识
  - param2：配置路径
  - param3：获取配置失败时的默认配置
- `${safe_render_multi(['product1','product2'], '/path', 'default_value')}` ：获取某个云产品的配置。匹配规则为第一个配置存在且为已安装状态的云产品，优先级以param1的列表顺序为准。
  - param1：云产品标识列表
  - param2：配置路径
  - param3：获取配置失败时的默认配置

引用其他云产品配置

当云产品配置中需要用到其他云产品的配置时，可以通过 `${云产品名[配置项]}` 的方式进行引用。

例如，集群中已安装的了一个 mariaDB 的云产品，定义的配置如下：

```
data_template:
  endpoints:
        admin:
            password: 12sfeij113
```

mycloud-product 云产品需要使用 mariaDB 云产品的配置，获取 password 数据时，可通过如下方式进行获取：

```
data_template:
  endpoints:
        maria_db:
            password:
        ${mariadb['endpoints']['admin']['password']}
```

若无法确定是否存在 mariaDB 的配置中，是否存在 password 配置，可以通过如下方式设置一个默认值：

```
data_template:
  endpoints:
        maria_db:
            password:
                ${mariadb['endpoints'].get('admin', {}).get('password',
"12345")}
```

若 mariaDB 可能还没安装，在获取不到时需要使用一个默认值代替，可以使用 `safe_render` 模板函数

```
data_template:
  endpoints:
        maria_db:
            password:
                ${safe_render('mariadb', '/endpoints/admin/password',
'default')
```

引用平行依赖的云产品（依赖的云产品存在互斥云产品）时，可以使用safe_render_multi模板函数：

```
data_template:
  endpoints:
        maria_db:
            password:
                ${safe_render_multi(['neutron','proton'],
'/endpoints/admin/password', 'default')
```

使用mako语法实现条件判断

mako语法同时兼容平台6.0.2和6.1.1版本，可以使用条件判断，实现按照自定义优先级获取依赖的云产品配置。 如果云产品需要同时兼容6.0.2版本和6.1.1版本，推荐使用mako语法： password: |- % if neutron: ${neutron.get('endpoints',{}).get('oslo_db',{}).get('auth',{}).get('user',{}).get('password','default')} % elif proton: ${proton.get('endpoints',{}).get('oslo_db',{}).get('auth',{}).get('user',{}).get('password','default')} % else: ${random_password('va_oslo_db_user', 8)} % endif 使用时注意缩进，避免格式错误导致配置渲染失败。

示例

```
data_template:
  endpoints:
    maria_db:
      auth:
       admin:
        password:
          ${mariadb['endpoints']['oslo_db']['auth']['admin']['password']}
        user:
          password:
            ${random_password('user_mariadb', 8)}
        neutron:
        password:
        ${safe_render('neutron', '/endpoints/admin/password', 'default')
      password1:
        ${safe_render_multi(['neutron','proton'],
'/endpoints/admin/password', 'default')
      password2: |-
        % if neutron:
        ${neutron.get('endpoints',{}).get('oslo_db',{}).get('auth',
{}).get('user',{}).get('password','default')}
        % elif proton:
```

```
        ${proton.get('endpoints',{}).get('oslo_db',{}).get('auth',
{}).get('user',{}).get('password','default')}
        % else:
        default
        % endif
      password3: |-
        % if neutron:
        ${neutron.get('endpoints',{}).get('oslo_db',{}).get('auth',
{}).get('user',{}).get('password','default')}
        % else:
        ${random_password('va_oslo_db_user', 8)}
        % endif
  labels:
    node_selector_key:
      ${cloud_product_label}
    node_selector_value:
      ${label_enabled}
```

通过 License 传参修改副本数

说明

通过 ${replicas_by_modality} 来获取由平台设置的副本数，ECP 602 版本会从云产品 license 的 **values** 数据中获取指定的副本数。对应的 key 为default_product_replicas，获取不到时，默认值为3。

示例

license 定义为如下值时：

```
cloud_product:
- alcubierre:
    ttl: '2200-01-01'
    values:
        default_product_replicas: 3
company: EasyStack-certified-OS-in-OS-3bonds-v6
lic_type: cloud_product
project: EasyStack
serial: d93d10ad-2075-4327-a059-ebde38345a3b
```

_config_helpers.yaml.tpl 定义：

```
data_template:
  pod:
    replicas:
      devops_dashboard_api:
        ${replicas_by_modality}
      devops_dashboard_server:
        ${replicas_by_modality}
```

渲染后：

```
data_template:
  pod:
    replicas:
      devops_dashboard_api: 3
      devops_dashboard_server: 3
```

通过License主动更新云产品及平台软件配置

之前云产品license中的自定义值，需要云产品在 _config_helpers.yaml.tpl 的 data_template 字段下通过模板的方式显式的引用相关的值，才可通过更新 license 的方式动态更新，数据更新方式较为被动，现在增加通过 license 主动更新云产品配置的方案，云产品无需再通过模板声明的方式获取 license 数据，后续 license 中的数据将逐步禁止云产品直接读取。

说明

license详细配置方案如下：

1. 每个云产品 License 新增 config 字段，保存主动更新配置
2. License CRD 中新增 platform_software 字段，保存平台软件主动更新配置
3. License CRD 中 data 下新增 topology字段，保存平台软件和云产品的拓扑数据
4. 安装阶段将云产品或平台软件对应的 config 及拓扑数据 topology和 data_template 渲染后的数据进行 merge，使用此数据进行安装
5. 安装完成后，动态监控 license config，当出现变化时，将 license 中各云产品或平台软件 config 字段及拓扑数据和云产品的 cpconfig 的 data 数据进行 merge，merge 完成后计算 data hash，若 hash 不一致则触

发配置更新操作。 渲染配置模板时，会跳过license数据的读取，并置为空对象，以达到禁止读取 license 数据的能力。

配置优先级

目前云产品的配置数据源来自于 _config_helpers.yaml.tpl 的 data_template 字段、config crd，cpconfig 自定义配置 custom_config 字段，加入了 license 数据后，字段相同时预计的配置优先级从低到高如下：

1. data_template
2. config crd
3. 拓扑数据
4. license config 数据
5. custom_config 自定义配置

示例

values.yaml

```
# cloud_product_1 values.yaml
pod:
  replicas:
    api: 1
    dashboard: 1
    db: 1

# platform_software_1 values.yaml
pod:
  replicas:
    api: 1
    dashboard: 1
    db: 1
```

_config_helpers.yaml.tpl

```
data_template:
  pod:
    replicas:
```

```
      api: 2
      dashboard: 2
```

license config

```
apiVersion: servicecatalog.ecp.com/v1
data:
  cloud_product:
    cloud_product_1:
      ttl: -1
      values:
        api: 2
        dashboard: 2
      config:
        pod:
          replicas:
            dashboard: 3

  platform_software:
    platform_software_1:
      config:
        pod:
          replicas:
            dashboard: 3

  topology:
    cloud_product_1:
      pod:
        replicas:
          dashboard: 4
    platform_software_1:
      pod:
        replicas:
          dashboard: 4
```

最终渲染后的 values：

```
# cloud_product_1
pod:
  replicas:
```

```
    api: 2
    dashboard: 3
    db: 1

# platform_software_1
pod:
  replicas:
    api: 2
    dashboard: 3
    db: 1
```

云产品依赖

说明

`_config_helpers.yaml.tpl` 文件中涉及参数如下：

- `dependencies` 属性用于定义云产品的依赖
  - `arch` ：定义云产品依赖的系统架构，支持 x86 和 arm
  - `services` ：定义云产品依赖的服务
  - `parallel_services` ：定义云产品平行依赖的服务，其中声明的云产品只要有一个满足即达到依赖条件
  - `exclusion` ：定义云产品互斥依赖的服务，声明的云产品不可和此云产品同时安装
  - `platform_version` ：定义云产品依赖的最低平台版本
- `base_version` ：定义依赖的必升版本
- `trial_days` ：定义可试用天数（-1为免费，默认需要定义为180）
- `license_expired` ：定义license过期时需要upgrade的属性（例如应用中心在license过期时可能不会删除所有已安装应用，只会禁用应用商店功能），由云产品自行定义。

示例

```
dependencies:
  platform_version: 6.0.1
  arch:
    - x86
```

```
    - arm
  services:
    horizon: 6.0.1-alpha.0
    nova: 6.0.1-alpha.0
  exclusion:
    - nova
    - cinder
    parallel_services:
    - iam: 6.0.1
      schedule: 6.0.1
    - neturn: 6.0.1
      proton: 6.0.1

base_version: 6.2.1

trial_days: 180

license_expired:
  feature:
    disable: true
```

云产品必升版本

说明

`_config_helpers.yaml.tpl` 文件中 `base_version` 属性用于定义云产品升级到此版本时，依赖的必升版本，云产品必须先升级到 `base_version` 版本后，才能升级到此版本。

示例

`_config_helpers.yaml.tpl` 文件中存在如下定义时，必须先升级到 `6.0.3` 版本，才可继续升级到当前版本。

```
base_version: 6.0.3
```

1.1.6.5 云产品Deployment

高可用配置

云产品的 deployment 如果需要配置为高可用多副本模式，需要添加反亲和及滚动升级策略。目前，云平台的已提供了一套公共模板来配置反亲和及滚动升级，云产品可在添加了 helm-tookit 的依赖后直接使用。

前置条件

云产品的 `requirements.yaml` 中添加了 **helm-toolkit** 依赖：

```
dependencies:
  - name: helm-toolkit
    repository: http://charts.easystack.io:8090
    version: 5.0.1
```

云产品 **values.yaml **中定义了滚动升级策略

```
  lifecycle:
    upgrades:
      deployments:
        revision_history: 3
        pod_replacement_strategy: RollingUpdate
        rolling_update:
          max_unavailable: 1
          max_surge: 3
```

配置示例

在云产品的多副本 deployment 中添加如下配置：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-api
spec:
```

```
  replicas: 3
# 滚动升级策略
{{ tuple $envAll | include "helm-
toolkit.snippets.kubernetes_upgrades_deployment" | indent 2 }}
  template:
    metadata:
      labels:
# 生成 labels
{{ tuple $envAll "backend-api" "api" | include "helm-
toolkit.snippets.kubernetes_metadata_labels" | indent 8 }}
    spec:
      affinity:
# 反亲和策略
{{ tuple $envAll "backend-api" "api" | include "helm-
toolkit.snippets.kubernetes_pod_anti_affinity" | indent 8 }}
```

1.1.6.6 云产品使用持久化存储

前置条件

已在云产品权限声明一章中声明开启了持久化存储能力。

说明

云产品需要持久化数据存储能力时，ECP平台可提供基于Ceph的PVC存储能力。使用PVC时，需要在statefulset控制器中声明PVC，不建议在deployment等无状态资源中声明PVC，否则挂载了PVC存储的deployment可能会出现镜像更新后无法升级的问题。

示例

以下示例中，演示了 mycloud-product 通过 statefulset-dashboard-api.yaml 启动了一个 dashboard-api 的 pod，并声明了一个名为 data-pvc 的持久化存储，挂载到了backend-api 容器中。

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: dashboard-api
  labels:
    app: dashboard-api
spec:
  serviceName: dashboard-api
  replicas: 1
  selector:
    matchLabels:
      app: dashboard-api
  template:
    metadata:
      labels:
        app: dashboard-api
    spec:
      nodeSelector:
        {{ .Values.labels.node_selector_key }}: {{
.Values.labels.node_selector_value }}
```

```
      containers:
      - name: backend-api
        image: {{ .Values.images.tags.backend-api }}
        imagePullPolicy: {{ .Values.images.pull_policy }}
        ports:
        - name: http
          containerPort: {{ .Values.conf.container_port }}
        volumeMounts:
        - name: data-pvc
          mountPath: /var/lib/pgsql/data
          subPath: pgsql/data
        - name: data-pvc
          mountPath: /var/log/ambari-server/
          subPath: backend/log
  volumeClaimTemplates:
  - metadata:
      name: data-pvc
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: {{ .Values.pvc.storage }}
      storageClassName: {{ .Values.pvc.storageClassName }}
```

1.1.6.7 云产品访问策略

云产品部署到集群后，首先需要通过 service 和 ingress 定义云产品的路由，再通过 ECP 规范定义云产品的菜单数据和访问权限，在登录ECP后，符合权限的账户就可以通过上导访问到对应的云产品了。

云产品Service

说明

Service 是 k8s 中的抽象概念， 定义了一组 Pods 的逻辑集合和一个用于访问它们的策略。云产品 Chart 中可能存在多个Service来暴露不同的服务，可采用service-[功能].yaml 的命名方式来进行区分。

示例

以下示例中，mycloud-product 通过 service-product-api.yaml 定义了一个 my-cloud-product-api service 用于暴露后端 api 服务。

```
apiVersion: v1
kind: Service
metadata:
  name: my-cloud-product-api
spec:
  ports:
    - name: http-api
      port: 80
      protocol: TCP
      targetPort: {{ .Values.conf.container_port }}
  selector:
    app: mycloudproduct
  type: ClusterIP
```

云产品Ingress

说明

Ingress 是 k8s 定义 service 向外暴露的路由，可采用 `ingress-[功能].yaml` 的命名方式来进行区分。为了隔离各个云产品的路由，云产品的Ingress需要有一个统一的云产品路径前缀，如示例中的 `path: /my-cloud-product` ，访问此云产品的 api 时，都会携带 `/my-cloud-product` 的路由。

注意

只需要给 dashboard 和提供界面服务的 pod 添加 ingress.kubernetes.io/custom-http-errors: 404,500,xxx 的注释。API 服务不需要添加 custom http errors，否则异常信息会被 nginx 拦截，导致无法快速定位问题。

示例

以下示例中，mycloud-product 定义了两个ingress用于配置前端和后端的路由。通过 ingress-product-api.yaml 定义了一个 my-cloud-product-api ingress 用于配置后端 api 服务的路由。

ingress-product-dashboard.sh

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-cloud-product-dashboard
  annotations:
    kubernetes.io/ingress.class: "nginx"
    ingress.kubernetes.io/force-ssl-redirect: "true"
    ingress.kubernetes.io/custom-http-errors: 404,500
spec:
  rules:
  - http:
      paths:
      - path: /my-cloud-product/dashboard/
        backend:
          serviceName: my-cloud-product-dashboard
          servicePort: http-dashboard
```

ingress-product-api.sh

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-cloud-product-api
```

```
  annotations:
    kubernetes.io/ingress.class: "nginx"
    ingress.kubernetes.io/force-ssl-redirect: "true"
spec:
  rules:
  - http:
      paths:
      - path: /my-cloud-product/api/
        backend:
          serviceName: my-cloud-product-api
          servicePort: http-api
```

云产品自定义导航分类

说明

云产品的导航需要属于一个导航的分类，如果导航需要加入到一个已有的分类中，可跳过此章节，直接在云产品导航文件中声明已有的分类名。如果新增自定义导航的分类，则需要在 `templates/_category.yaml` 进行定义，涉及如下参数：

- 分类名（e.g.：mycloud-product）：云产品自定义分类名
- id（e.g.：mycloud-product）：需要和名称一致。
- dynamicNumber：动态排序保留字段，可以为任意数字。

示例

如下 `_category.yaml` 文件示例中，定义了如下两个导航：

- mycloud-product-a
  - id: mycloud-product-a
  - icon: mycloud-product-a
  - 中文名：云产品-a
  - 英文名：mycloud-product-a
- mycloud-product-b
  - id: mycloud-product-b

- icon: mycloud-product-b
- 中文名：云产品-b
- 英文名：mycloud-product-b

```yaml
mycloud-product-a:
  id: mycloud-product-a
  icon: mycloud-product-a
  name:
    en: mycloud-product-a
    zh-cn: 云产品-a
  navigations: []
  dynamicNumber: 1

mycloud-product-b:
  id: mycloud-product-b
  icon: mycloud-product-b
  name:
    en: mycloud-product-b
    zh-cn: 云产品-b
  navigations: []
  dynamicNumber: 1
```

云产品导航

说明

云产品 chart 包中， `templates/_navigation.yaml` 用于定义，涉及参数如下：

- `catagory` ： 默认导航分类
- `name` ： 云产品中英文翻译
- `nav` ： 左导航
  - `name` ： 导航中英文翻译
  - `icon` ： 导航 icon
  - `order` ： 导航顺序
  - `url` ： 导航 URL

  - `baseURI`：导航基础路由
  - `available`：导航可见性
  - `wizard`：是否有向导数据
  - `catagory`：导航分类（可选，若无此字段，则会添加到默认导航分类中）
- `sub_nav`：左导航二级菜单（可选，属性参见 `nav`）。
- `top_nav`：上导航数据。（可选，属性参见 `nav`，如果不设置，则将把 `nav` 的所有数据显示在上导航中，如果显示的过程中 `nav` 中有 `sub_nav` 的话，则会把 `sub_nav` 的第一个 url 取出用于显示）

示例

如下 _navigation.yaml 的示例中，此云产品向 **security 身份与访问管理** 分类下注册了两个导航：

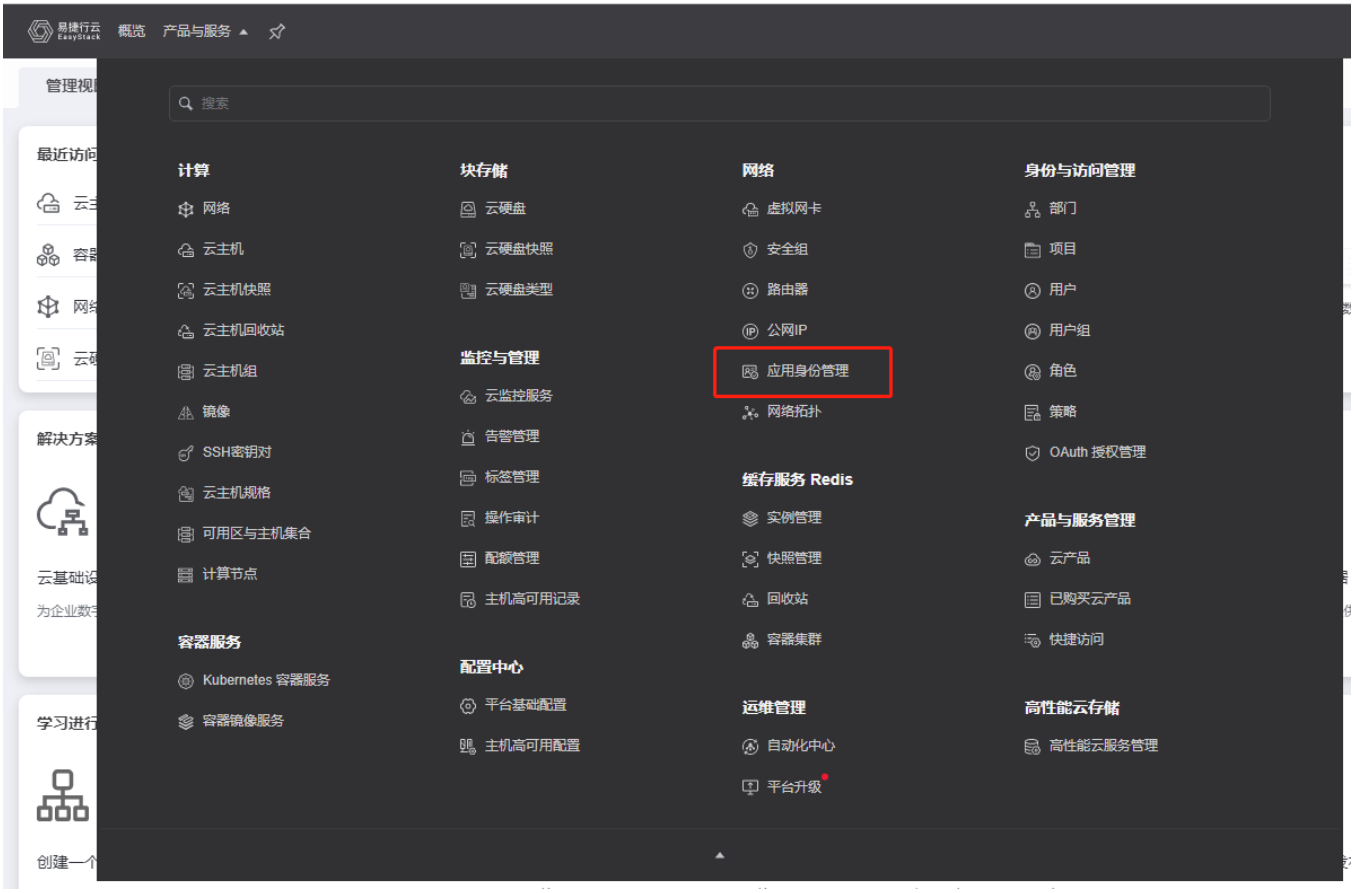
- 角色
- 策略

network 网络 分类下注册了一个导航：

- 应用身份管理

```yaml
category: security
name:
  en: Identity & Access Management
  zh-cn: 身份与访问管理
nav:
  role:
    proId: role
    name:
      zh-cn: 角色
      en: Role
    url: /iam/roles
    icon: role
    classify: identity
    baseURI: /iam/
    order: 4
    available: true
    wizard: false
  policy:
    proId: policy
```

```
    name:
      zh-cn: 策略
      en: Policy
    url: /iam/policies
    icon: policy
    classify: identity
    baseURI: /iam/
    order: 5
    available: true
    wizard: false
  applications:
    proId: applications
    category: network
    name:
      zh-cn: 应用身份管理
      en: Application Identity Management
    url: /iam/applications
    icon: application
    classify: identity
    baseURI: /iam/
    order: 6
    available: true
    wizard: false
```



导航访问策略

说明

云产品导航的访问策略在 `templates/_navigation_policy.yaml` 文件中进行定义，开发者可对不同角色的用户赋予不同的访问权限，例如某些高级管理页面设置仅云管理员可访问。设置访问策略的粒度包括 `top_nav` ， `nav` 、 `sub_nav` 字段下的所有url。如不填写，则默认所有角色均无权限访问此url，对所有类型账户都不可见。

当前仅支持使用四种默认角色：

- cloud_admin：云管理员，可管理云环境的所有资源。
- domian_admin：部门管理员，可管理所属部门下的所有资源。
- admin：项目管理员，可管理所属项目下的所有资源。
- member：普通用户，管理自己拥有的所有资源。

示例

```
/mycloud-product/mycloud-product-1:
  - cloud_admin
  - domain_admin
  - admin
  - member
/mycloud-product/mycloud-product-2:
  - member
/mycloud-product/mycloud-product-sub-feature:
  - member
```

云产品使用向导

说明

开发者可以在 `templates/_wizard.yaml` 文件中 `nav` 属性下对 cloud_admin，domian_admin，admin，member 定义使用向导，当用户安装或升级云产品后首次访问云产品时，将在界面展示账户对应权限的云产品使用向导，使用户能够掌握云产品新特性，快速上手。涉及参数如下：

- `order` ：新特性顺序
- `proId` ：云产品的上导航id
- `title` ：新特性中英文翻译
- `wizard` ：向导内容列表
  - `content` ：当前向导页标题中英文翻译
  - `pics` ：当前向导页中英文图片

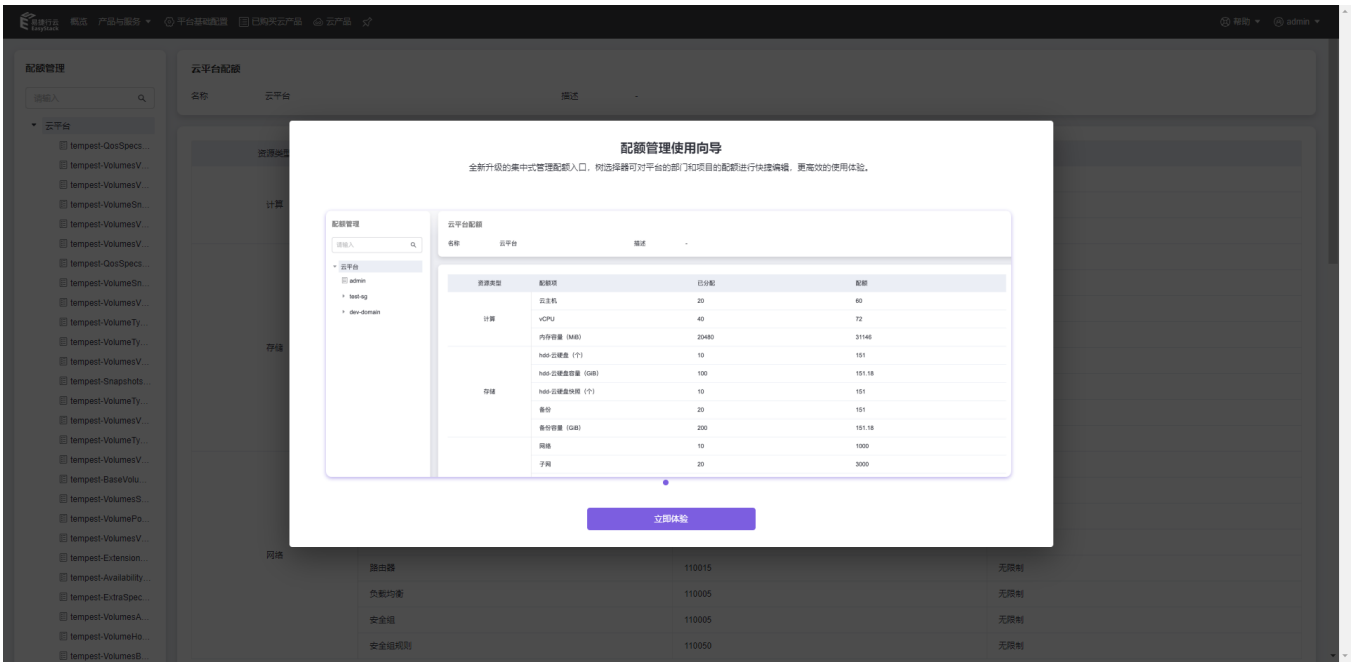
示例

```
nav:
  cloud_admin:
    mycloud_product_overview:
      order: 1
      proId: mycloud_product_overview
      title:
        en: mycloud-product
```

```
      zh-cn: 我的云产品
    wizard:
      - content:
          en: English description
          zh-cn: 中文功能描述
        pics:
          en: wizard/wizard_en.png
          zh-cn: wizard/wizard_zh_cn.png
      - content:
          en: 英文功能描述
          zh-cn: 中文功能描述。
        pics:
          en: wizard/admin_overview_en.png
          zh-cn: wizard/admin_overview_zh_cn.png

  mycloud_product_apps:
    order: 2
    proId: mycloud_product_apps
    title:
      en: mycloud_product_apps
      zh-cn: 云主机应用
    wizard:
      - content:
          en: 英文功能描述
          zh-cn: 中文功能描述。
        pics:
          en: wizard/admin_instance_1_en.png
          zh-cn: wizard/admin_instance_1_zh_cn.png
      - content:
          en: 英文功能描述
          zh-cn: 中文功能描述。
        pics:
          en: wizard/admin_instance_2_en.png
          zh-cn: wizard/admin_instance_2_zh_cn.png
```

展示效果：


配额管理使用向导
全新升级的集中式管理配额入口，例选择器可对平台的部门和项目的配额进行快捷编辑，更高效的使用体验。
立即体验

1.1.6.8 云产品安装完成状态检查

说明

云产品 chart 包中 `template/job-hooks.yaml` 文件用于定义云产品安装时的必要检查，云产品 chart 包开始部署后，会开始检查此 job 运行状态，当此 job 运行完成后，云产品会标记成已安装的状态。

注意：若不定义安装资源检查会导致云产品安装失败。且此任务最多运行时间为30分钟，超过会导致云产品安装超时而失败。

示例

使用通用模板

ECP 为云产品准备了一套通用的模板来定义此 job，使用此模板时，需要在云产品 chart 的 requirments.yaml 中定义依赖 `helm-toolkit` ，之后直接在 values.yaml 中填写固定格式的依赖检查即可，如下所示。

requirements.yaml：

```
dependencies:
  - name: helm-toolkit
    repository: http://charts.easystack.io:8090
    version: 5.0.1
```

job-hooks.yaml：

```
{{- $envAll := . }}
{{- include "helm-toolkit.snippets.job_hook" $envAll }}
```

values.yaml：

```
images:
  tags:
    hook: hub.easystack.io/production/escloud-linux-source-busybox:latest
    dep_check: hub.easystack.io/production/kubernetes-entrypoint:v0.2.1
  pullPolicy: "IfNotPresent"
  job_pull_policy: "Always"
```

```
dependencies:
  posthook:
    services:
    - service: dashboard_api
      endpoint: internal
    - service: dashboard_web
      endpoint: internal

endpoints:
  cluster_domain_suffix: cluster.local
  dashboard_api:
    hosts:
      default: dashboard-api
      public: dashboard-api
  dashboard_web:
    hosts:
      default: dashboard-web
      public: dashboard-web
```

部署后，job-hooks.yaml 会被渲染为类似如下的 job：

```
apiVersion: batch/v1
kind: Job
metadata:
  name: mycloud-product-ecpbackendcheck-1634803477
spec:
  backoffLimit: 128
  completions: 1
  parallelism: 1
  template:
    metadata:
      creationTimestamp: null
      labels:
        component: ecpbackendcheck
    spec:
      containers:
      - command:
        - echo
        - done
        image: hub.easystack.io/production/escloud-linux-source-
busybox:latest
```

```
      imagePullPolicy: IfNotPresent
      name: iam-ecpbackendcheck
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    initContainers:
    - command:
      - kubernetes-entrypoint
      env:
      - name: POD_NAME
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.name
      - name: NAMESPACE
        valueFrom:
          fieldRef:
            apiVersion: v1
            fieldPath: metadata.namespace
      - name: INTERFACE_NAME
        value: eth0
      - name: DEPENDENCY_SERVICE
        value: mycloud-product:dashboard-api,mycloud-product:dashboard-web
      - name: DEPENDENCY_JOBS
      - name: DEPENDENCY_DAEMONSET
      - name: DEPENDENCY_CONTAINER
      - name: COMMAND
        value: echo done
      image: hub.easystack.io/production/kubernetes-entrypoint:v0.2.1
      imagePullPolicy: IfNotPresent
      name: init
    nodeSelector:
      cloud-product: enabled
    restartPolicy: OnFailure
```

自定义检查Job

检查Job也可以自行定义，添加更多需要检查的内容，无需添加 helm-toolkit 的依赖，如下所示，使用集群中 kubernetes-entrypoint 镜像完成了对 mycloud-product namespace 下 dashboard-api 和 dashboard-web service 的就绪检查：

```
{{- $envAll := . -}}
{{- $releaseName := $envAll.Release.Name }}
{{- $releaseTime := $envAll.Release.Time.Seconds }}
{{- $check := "ecpbackendcheck" }}
---
apiVersion: batch/v1
kind: Job
metadata:
  name: {{ $releaseName }}-{{ $check }}-{{ $releaseTime }}
  annotations:
    ecp: "true"
spec:
  template:
    metadata:
      labels:
        release_group: {{ $envAll.Values.release_group | default
$envAll.Release.Name }}
        application: {{ $releaseName }}
        component: {{ $check }}
    spec:
      restartPolicy: OnFailure
      nodeSelector:
        {{ $envAll.Values.labels.node_selector_key }}: {{
$envAll.Values.labels.node_selector_value }}
      containers:
        - name: init
          image: {{ tuple .Values.images.tags "dep_check" . | include "helm-
toolkit.utils.update_image" }}
          imagePullPolicy: {{ .Values.images.pull_policy }}
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  apiVersion: v1
                  fieldPath: metadata.name
            - name: NAMESPACE
              valueFrom:
                fieldRef:
                  apiVersion: v1
                  fieldPath: metadata.namespace
            - name: INTERFACE_NAME
```

```
              value: eth0
            - name: DEPENDENCY_SERVICE
              value: mycloud-product:dashboard-api,mycloud-
product:dashboard-web
            - name: DEPENDENCY_JOBS
            - name: DEPENDENCY_DAEMONSET
            - name: DEPENDENCY_CONTAINER
            - name: COMMAND
              value: "echo done"
```

1.1.6.9 云产品Makefile

云产品项目根目录下的 Makefile 用于定义云产品的打包流程，当前 Makefile 采用统一的定义，可直接使用如下代码：

```
HELM := helm
TASK := build

CHARTS := $(filter-out $(EXCLUDES), $(patsubst chart/%/.,%,$(wildcard
chart/*/.)))

all: $(CHARTS)

$(CHARTS):
    @echo
    @echo "===== Processing [$@] chart ====="
    @make $(TASK)-$@

init-%:
    if [ -f chart/$*/Makefile ]; then make -C $*; fi
    if [ -f chart/$*/requirements.yaml ]; then [ "$*" = "helm-toolkit" ] ||
(mkdir -p chart/$*/charts && cp helm-toolkit*.tgz chart/$*/charts); fi


lint-%: init-%
    if [ -d chart/$* ]; then $(HELM) lint chart/$*; fi

build-%: lint-%
    if [ -d chart/$* ]; then $(HELM) package --save=false chart/$*; fi

clean:
    @echo "Removed _partials.tpl, and _globals.tpl files"
    rm -f chart/*/templates/_partials.tpl
    rm -f chart/*/templates/_globals.tpl
    rm -f chart/*tgz *tgz chart/*/charts/*tgz
    rm -f chart/*/requirements.lock


.PHONY: $(EXCLUDES) $(CHARTS)
```

1.1.6.10 云产品自定义hook脚本

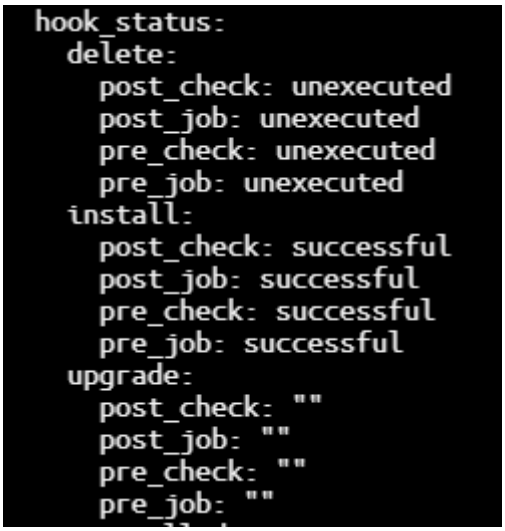
本页记录ECP平台hook的使用方法，用于指导现有云产品增加hook脚本以及foundation服务迁移为云产品时增加hook脚本。hook脚本在部署时独立运行， 不会侵入项目本身的hook操作。

hook 名词解析

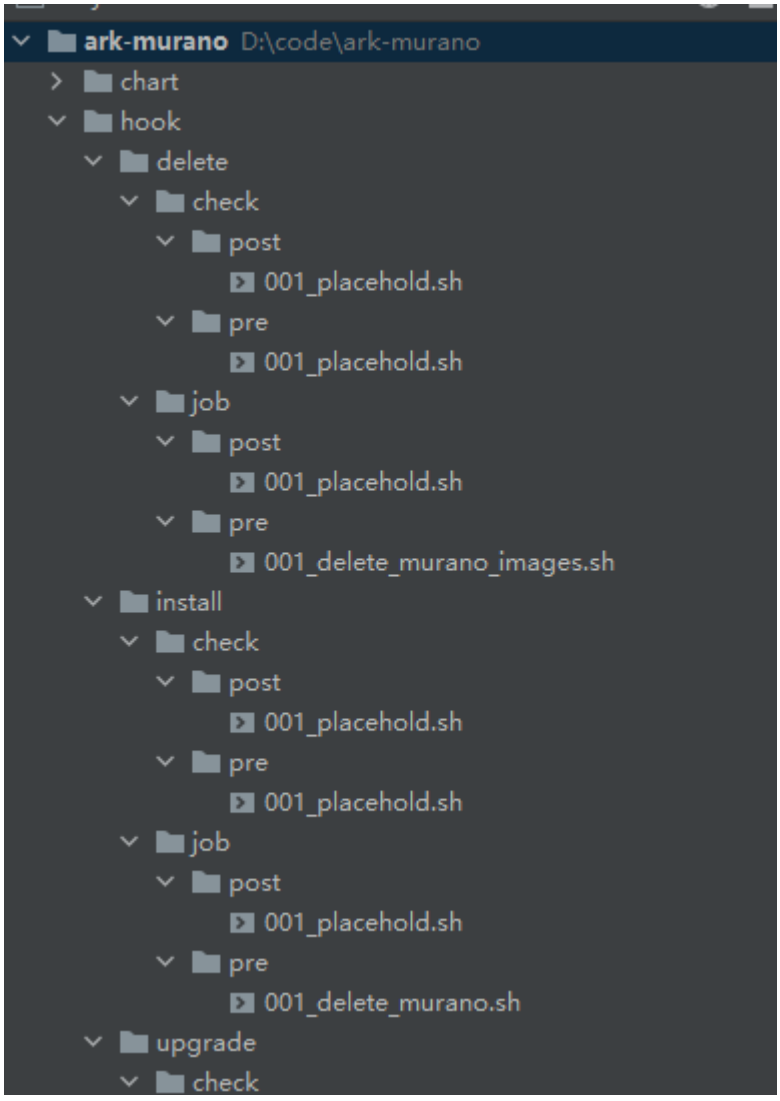
1、install: 安装。 2、upgrade: 升级。 3、delete: 删除。 4、check: 检查，执行具体操作前、后检查环境状态。 5、job: 工作， 执行具体的操作前、后做相关的调整操作。 6、post: 后置， 动作完成后执行。 7、pre: 前置， 动作完成前执行。

hook 定义方法

1、创建目录 在云产品的ark项目（如ark-murano）下创建 hook目录。 2、创建执行路径 在 hook下按需创建 install | upgrade | delete目录， 然后分别按需创建check| job, 在此层目录下按需创建post | pre目录， 完成三级目录， 对应着操作为 install 下的 check 下的 pre 时执行，以此类推。 3、创建脚本 在 pre 目录下， 以 001_abc.sh, 命名， 001， 002 代表该文件在此处执行顺序， abc 为该文件名称。里面为要执行的脚本， 该脚本文件会在busybox里执行。 4、验证 文件脚本经busybox 里验证后， 提交到云产品对应的ark项目，将云产品重新打包后到环境中执行安装、升级或者删除, 会在ems命名空间的云产品对应的cloudproduct cr中看到hook的相关状态信息。hook状态包括undefined、unexecuted、executing、successful、failed。

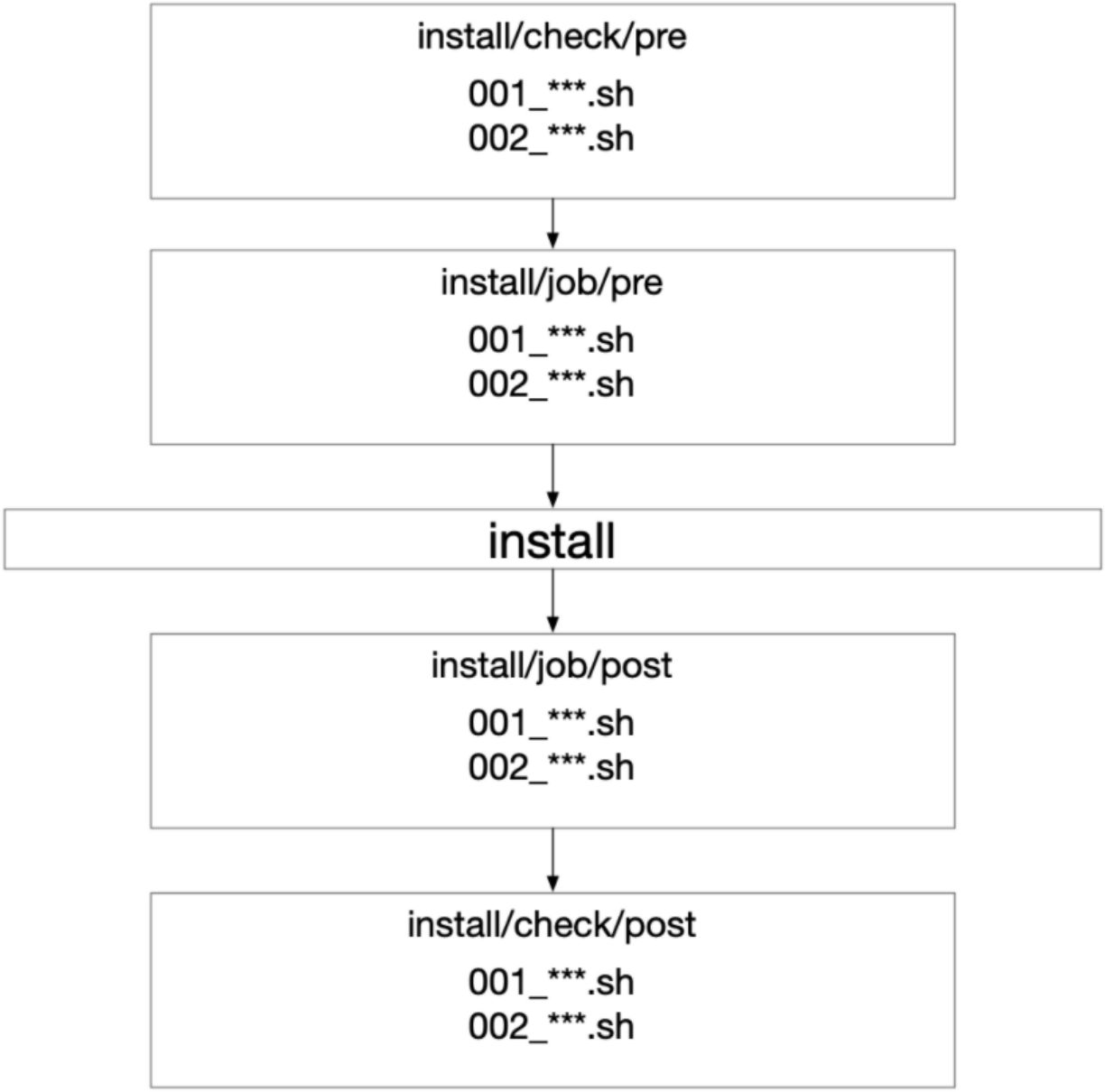


hook目录结构示例：



hook 执行流程

以安装流程为例：



升级过程和删除过程中hook的执行流程与安装流程基本相同。 ECP 6.0.2 版本开放了基础云产品删除能力，需要基础云产品基于平台hook规范定义了 delete/check/pre 前置删除检查脚本后才可进行删除操作。**foundation云产品必须基于平台hook规范定义delete/check/pre脚本才能删除, 非foundation云产品只能定义delete/check/pre脚本。**删除过程中的hook检查：

1. 前置检查
    1. 云产品基于平台hook规范定义了delete/check/pre脚本

   2. 是否存在依赖的云产品
2. 删除云产品CR
3. 触发operator完成删除操作
   1. 执行delete/check/pre删除前检查脚本
      1. 成功则继续后续删除流程
      2. 失败则记录错误信息, 云产品状态重置为已安装状态
   2. operator常规删除流程

删除前置检查定义

云产品可以通过定义 `delete/check/pre` 进行云产品的删除前置检查，检查不通过时，禁止删除云产品。由于很多基础云产品包中附带了一个默认的 delete/check/pre 脚本，不具备实际的检查能力，当云产品存在前置删除检查脚本时，同时会检查此目录下**任一**脚本中是否包含如下语句，当其中一个脚本存在此语句才可判断为一个有效的脚本： `**echo '{name} delete pre check pass'**` ** **（{name}为云产品名）

示例：

```
echo 'cinder delete pre check pass'
```

注意：ECP检查会完整匹配此段代码, '{name} delete pre check pass' 必须用单引号包裹.

自定义错误检查提示

安装、升级，删除**前置**检查脚本中，可通过注释定义此脚本执行失败时的提示信息，支持如下注释信息，当有多条同名注释时，只会取第一条注释，可以编写多个脚本来完成不同的检查。

- fail_zh_hans: 中文错误提示
- fail_en: 英文错误提示

当未定义错误提示信息时,提示信息为:

- 中文: 云产品删除失败
- 英文: Cloud product deletion failed

示例：

001_placehold.sh 此脚本执行失败时，中文界面下会弹出存在异常数据的提示信息。

```
#!/bin/bash

# fail_zh_hans: 存在异常数据
# fail_en: Has exception data

set -ex

function placehold() {
echo 'cinder delete pre check pass'
}

main() {
placehold
}

main
```

注意事项

- 在脚本中通过 kubectl 操作自定义资源相关的命令，必须使用自定义资源全称，原生资源不受影响 。

1.1.6.11 Job模板开发规范

Kubernetes Job 的某些特殊 YAML 属性是不允许更新的，因此需要规范云产品 Job 模板的开发规范。以保证云产品的 Job 可以正常升级。

Job 修改规范

首先，只要在云产品当前开发版本修改了 Job，包括 YAML 模板、脚本等内容，必须修改 Job 名称，规则是在现有名称后面加上 “-” + “云产品版本”。例如：Job 名称叫 “db-init”，云产品当前开发版本号为 6.2.1，在 Job 做了修改时，需要将 Job 的名称修改为 “db-init-6.2.1”。

Job 镜像使用规范

Job 的镜像必须使用固定版本，分为两种情况：

1. **依赖云产品自身组件的镜像。**需要将 values.yaml 中的 image tag 写为 5.0.1 标识（历史遗留问题，CD 会在打包时自动替换为云产品当前 build version），确保开发过程中每次构建都会跟当前 build version 保持一致，从而能够使用到云产品自身组件最新的镜像。
2. **依赖到外部镜像。**必须指定为确定的版本号，不能使用 latest。因为云产品是按需安装的，不同版本携带的 latest 镜像内容可能不一致，会导致 latest tag 相互覆盖。

已知问题

使用以上规范更新 Job，无法彻底解决云产品小版本升级问题。目前需要先删除环境中的 Job 之后才能正确执行升级。例如上面例子中的 Job 名称修改为 “db-init-6.2.1” 之后，只会在下一次升级时替换掉原有的 “db-init” Job，后续升级都不会重建 Job，会有以下两个异常场景：

- 如果这个 Job 的镜像版本是跟随 build version 的，或者再次修改了 Job 的 YAML 属性，会导致后续每次升级都失败，错误原因是 Job 某些 YAML 属性不允许更改；
- 如果这个 Job 的 YAML 模板不变，后续只更新了脚本内容，后续每次升级都不会生效。因为每次升级 Job 没有变化，不会被更新。

咨询热线：400-100-3070

北京易捷思达科技发展有限公司：
北京市海淀区西北旺东路10号院东区23号楼华胜天成科研大楼一层东侧120-123
南京分公司：
江苏省南京市雨花台区软件大道168号润和创智中心B栋一楼西101
上海office：
上海黄浦区西藏中路336号华旭大厦22楼2204
成都分公司：
成都市高新区天府五街168号德必天府五街WE602

邮箱：
contact@easystack.cn (业务咨询)
partners@easystack.cn(合作伙伴咨询)
marketing@easystack.cn (市场合作)
training@easystack.cn (培训咨询)
hr@easystack.cn (招聘咨询)